ISSN 2450-0003

Filip Wójcik
Wroclaw University of Economics and Business
e-mail: filip.wojcik@ue.wroc.pl
ORCID: 0000-0001-5938-7260

UTILIZATION OF DEEP REINFORCEMENT LEARNING FOR DISCRETE RESOURCE ALLOCATION PROBLEM IN PROJECT MANAGEMENT – A SIMULATION EXPERIMENT

WYKORZYSTANIE UCZENIA ZE WZMOCNIENIEM W PROBLEMACH DYSKRETNEJ ALOKACJI ZASOBÓW W ZARZĄDZANIU PROJEKTAMI – EKSPERYMENT SYMULACYJNY

DOI: 10.15611/ie.2022.1.05
JEL Classification: C44, C45, C61

Quote as: Wójcik, F. (2022). Utilization of deep reinforcement learning for discrete resource allocation problem in project management – a simulation experiment. *Business Informatics*, (1).

Abstract: This paper tests the applicability of deep reinforcement learning (DRL) algorithms to simulated problems of constrained discrete and online resource allocation in project management. DRL is an extensively researched method in various domains, although no similar case study was found when writing this paper. The hypothesis was that a carefully tuned RL agent could outperform an optimisation-based solution. The RL agents: VPG, AC, and PPO, were compared against a classic constrained optimisation algorithm in trials: "easy"/"moderate"/"hard" (70/50/30% average project success rate). Each trial consisted of 500 independent, stochastic simulations. The significance of the differences was checked using a Welch ANOVA on significance level alpha = 0.01, followed by post hoc comparisons for false-discovery control. The experiment revealed that the PPO agent performed significantly better in moderate and hard simulations than the optimisation approach and other RL methods.

Keywords: reinforcement learning (RL), operations research, management, optimisation.

Streszczenie: W artykule zbadano stosowalność metod głębokiego uczenia ze wzmocnieniem (DRL) do symulowanych problemów dyskretnej alokacji ograniczonych zasobów w zarządzaniu projektami. DRL jest obecnie szeroko badaną dziedziną, jednak w chwili przeprowadzania niniejszych badań nie natrafiono na zbliżone studium przypadku. Hipoteza badawcza

zakładała, że prawidłowo skonstruowany agent RL będzie w stanie uzyskać lepsze wyniki niż klasyczne podejście wykorzystujące optymalizację. Dokonano porównania agentów RL: VPG, AC i PPO z algorytmem optymalizacji w trzech symulacjach: „łatwej"/„średniej"/„trudnej" (70/50/30% średnich szans na sukces projektu). Każda symulacja obejmowała 500 niezależnych, stochastycznych eksperymentów. Istotność różnic porównano testem ANOVA Welcha na poziomie istotności $\alpha = 0.01$, z następującymi po nim porównaniami *post hoc* z kontrolą poziomu błędu. Eksperymenty wykazały, że agent PPO uzyskał w najtrudniejszych symulacjach znacznie lepsze wyniki niż metoda optymalizacji i inne algorytmy RL.

Słowa kluczowe: uczenie ze wzmocnieniem, badania operacyjne, zarządzanie, optymalizacja.

1. Introduction

According to selected definitions, project management is a set of actions, including the "application of knowledge, skills, tools, and techniques to project activities to meet project requirements. Project management refers to guiding project work to deliver the intended outcomes" (Institute, 2021; Manikantan and Gurusamy, 2016). Work on each project is usually divided into several phases: conceptualisation, definition, planning, execution, and termination (Schwindt, 2010). Optimal planning, including estimation, scheduling, and constrained allocation, has been addressed in many research studies because any project activities are subject to precedence, resource, and time constraints on limited resources (Selaru, 2012). The need for better precision and speed in decision-making has generated significant demand for automated systems, which have slowly started to replace older solutions (Gupta, Modgil, Bhattacharyya, and Bose, 2022). Numerous recommendations for the so-called data-driven companies have been created, including the need to have systems for predictive modeling, forecasting, optimisation, and planning (Anderson, 2015; Sharda, Delen, and Turban, 2020). In particular, in fields like Operations Research (OR) which focuses on improving day-to-day company decisions (Gupta et al., 2022), fusion with machine learning and big data fields have become more critical as the volumes and velocity of data grow every year (Bhimani and Willcocks, 2014; Duan, Edwards, and Dwivedi, 2019).

This paper aimed to assess the applicability of a rapidly growing family of machine learning algorithms, called Reinforcement Learning (RL), to the problem of constrained resource allocation in project management under a strong environment uncertainty. The research hypothesis in the study is that a carefully chosen RL-based agent can outperform a classic constrained-optimisation approach in a simulated environment.

The paper is structured as follows: Sections 1.1 and 1.2 give an overview of the existing research on resource allocation problems and the applicability of RL techniques. Section 1.3 presents some theoretical background necessary to understand RL mechanics. Part 2 describes the experimental setup: Sections 2.1

and 2.2 present the simulator design and possible actions, while 2.3 and2.4 give more detail on the objective function and experiment variants. Section 2.5 focuses on the algorithms selected for comparative study. Part 3 presents a detailed breakdown of the results and their analysis.

1.1. Previous work on optimisation techniques for resource allocation

Optimisation has been a research subject for over a century and laid a foundation for fields such as operations research (Ackoff, 1956). It is one of the critical tools, as it helps to make decisions that allow managers to choose the most promising options out of the available alternatives, often including the long-term horizon (Schwindt, 2010). A review and meta-analysis of numerous publications from the late 1990s up to the present reveals an increasing interest in the design of optimisation systems with meta-heuristics, evolutionary algorithms, and parallel computing, that operate under strong uncertainty (Chiang and Lin, 2020; Farhang Moghaddam, 2019).

The existing solutions to such problems included traditional linear programming (mostly mixed-integer constraint optimisation) (Islam, 2011; Kane and Tissier, 2012), entropy minimisation (Ye, Shi, Li, and Shi, 2014), or combinatorial multi-armed bandits (Zuo and Joe-Wong, 2021). The applicability of swarm or evolutionary algorithms, constraint satisfaction, and linear optimisation was assessed in key areas such as human resource allocation (Chiang and Lin, 2020).

1.2. Previous work on reinforcement learning applications

Over the years, multiple approaches to utilise reinforcement learning for resource allocation have been made. Some of them were based on the fundamental principles of Markov Decision Processes (MDP) and their use in Supply Chain Management (Giannoccaro and Pontrandolfo, 2002); others directly followed early RL formulations such as Q-learning for Business Process Management (Huang, van der Aalst, Lu, and Duan, 2011). Numerous publications describe attempts to apply different RL tools for constrained task scheduling and packing problems (Jędrzejowicz and Ratajczak-Ropel, 2013; Mao, Alizadeh, Menache, and Kandula, 2016) and logistics (Yan et al., 2021; Yuan, Li, and Ji, 2021).

The RL solutions for production planning and dynamic worker scheduling have been reviewed extensively (Koulinas, Xanthopoulos, Kiatipis, and Koulouriotis, 2018; Shyalika, Silva, and Karunananda, 2020), which highlight the benefits and drawbacks of different algorithm families, assessed with criteria such as convergence speed and sampling efficiency. Production planning differs from the problems presented in this study, but the initial choice of the RL algorithms was based on previous research in this area and the existing recommendations (Shyalika et al., 2020; Yu, Zhang, Jiang, Yang, and Shang, 2021).

RL is widely used in technical constraint resource allocation problems, e.g. server load balancing, channel allocation in telecom, and network management

(Li et al., 2018; Xu et al., 2021; Ye, Li, and Juang, 2019). Most of these approaches utilise different *Q*-learning variants.

To date, no publications directly focused on RL in project management sequential resource allocation with simulation experiments have been found.

1.3. Reinforcement learning overview

Reinforcement learning (RL) is one of the subfields of machine learning, focused on the autonomous agent interacting with the environment. An agent receives rewards by randomly acting in the environment, gradually improving performance (Schulman, 2016). The goal is to learn the actions that maximise the expected cumulative reward over time (Mousavi, Schukat, and Howley, 2018). Typically, reinforcement learning problems are modelled using MDP, formalised as follows (Arulkumaran, Deisenroth, Brundage, and Bharath, 2017):

1. $\mathcal{S} = \{s_0, s_1, \ldots s_m\}$ is a set of states that describe an environment.
2. $\mathcal{A}$ is a set of actions that can be performed in the environment.
3. $\mathcal{T}(s_{t+1} \mid s_t, a_t)$ – transition dynamics that describes the consequences of executing action a_t state s_t. This part can be probabilistic, lading to different s_{t+1} values.
4. $\mathcal{R}(s_t, a_t, s_{t+1})$ is a set of rewards obtained in transition.
5. A sequence of states, actions and rewards during the episode is called a trajectory *T* rollout.
6. All rewards accumulated during the trajectory rollout are called returns, denoted as *R*. The agent attempts to remember the consequences of past actions and utilises a discounting future reward by factor $\gamma \in [0,1]$. The return is formalised as follows

$$R = \sum_{t=0}^{T-1} \gamma^t r_{t+1} \ . \tag{1}$$

The goal of any agent is to define a policy π that represents the agent's behaviour during interaction. It is a strategy function that maps encountered states s_t into actions, and is denoted as (Sutton, Bach, and Barto, 2018):

$$\pi : \mathcal{S} \to p(\mathcal{A} = a \mid \mathcal{S}). \tag{2}$$

An agent should learn a policy that achieves a maximum return from all the states

$$\pi^* = \arg\max_{\pi} \mathbb{E}[R \mid \pi] \ . \tag{3}$$

During the learning process, the agents use some additional functions that help them find the best way to construct the policy. Policy action value (Mnih et al., 2016; Sutton et al., 2018)

$$Q^{\pi}(s,a) = \mathbb{E}^{\pi}\left[R_t \mid S_t = s, a\right] = \mathbb{E}^{\pi}\left[\sum_{k=0}^{T} \gamma^k r_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S} \tag{4}$$

describes the expected return from executing action while being in state and is often called a "q-function" or "q-value". It can be expressed in a recursive form, known as the Bellman equation which laid the foundation for more advanced RL algorithms (Arulkumaran et al., 2017; Bellman, 1954)

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s_{t+1}}\left[r_{t+1} + \gamma Q^{\pi}\left(s_{t+1}, \pi(s_{t+1})\right)\right]. \tag{5}$$

This equation can be interpreted as the relationship between the current and future Q-values. The current value consists in the actual reward and discounted future Q-values, each one being a reward obtained from following policy π (as described in equation (2)), starting from current state s_t.

The value function of state s_t under policy π describes the expected returns when following π from state s_t

$$V^{\pi}(s) = \mathbb{E}\left[R \mid s_t = s\right]. \tag{6}$$

In practice, there are multiple paradigms and approaches for learning such mappings. Two prominent families of algorithms include on-policy and off-policy learning. In the on-policy setting, an agent executing the algorithm is restricted to following a policy learnt so far; it can improve in a later phase. An agent can gain experience using functions other than the current policy in the off-policy approach (Sutton et al., 2018). Typical examples of these approaches are SARSA and Q-learning algorithms, where the former is an on-policy, and the latter is an off--policy. SARSA is formalised as follows (see Sutton et al., 2018)

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha\left[r_t + \gamma Q^{\pi}(s_{t+1}, a_{t+1}) - Q^{\pi}(s_t, a_t)\right], \tag{7}$$

where α is a learning speed parameter. This function corrects the current estimation of the q-value for a given state-action pair, after calculating and receiving future values, when following current policy. In contrast, off-policy Q-learning update utilises maximal next time-step q-value, not necessarily the one from policy π

$$Q^{\pi}(s_t, a_t) \leftarrow Q^{\pi}(s_t, a_t) + \alpha\left[r_t + \gamma \max_a Q(s_{t+1}, a) - Q^{\pi}(s_t, a_t)\right]. \tag{8}$$

In a deep learning RL variant (DRL), both functions are approximated with a set of (at least one) neural networks parameters θ, which is formalized as $Q^{\pi}(s, a, \theta) \approx Q^{\pi}(s, a)$ and $V^{\pi}(s, \theta) \approx V^{\pi}(s)$ (Mnih et al., 2016; Sutton et al., 2018). One of the significant improvements over the previous methods was the introduction of the so-called 'policy gradient'. It optimises the policy by calculating the gradient with respect to its parameters. Formally (Schulman, 2016):

$$\nabla_\theta \mathbb{E}_{\mathcal{T}}\left[R(\mathcal{T})\right] = \mathbb{E}_{\mathcal{T}}\left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi(a_t \mid s_t, \theta) R(\mathcal{T})\right], \tag{9}$$

where is a trajectory rollout under policy parameters θ and R(T) is the total reward of the trajectory.

The implementations of this general idea include approaches such as "Vanilla" Policy Gradient (VPG), which calculates a policy gradient after each episode termination, effectively performing a Monte Carlo update (Schulman, 2016). Due to its instability and sampling inefficiency, such an approach was further extended by combining a policy learning and value function learning simultaneously – called the actor-critic method. "Actor" (parametrised by θ) learns policy π, which is verified by the "critic" – a value function estimator parameterised by ϕ (Mnih et al., 2016; Schulman, 2016; Sutton et al., 1999). In that case, a learned value-function is treated as a baseline in the policy gradient, which serves the normalisation and correction purposes (Arulkumaran et al., 2017). Stability and variance improvement was achieved with the so-called "advantage estimations", where the returns per each time step are replaced by other calculations, emphasising the differences between actions and the default state-value (Mnih et al., 2016; Schulman, Moritz, Levine, Jordan, and Abbeel, 2016). In that context, an advantage can be defined as

$$A^\pi(s_t, a_t) := Q^\pi(s_t, a_t) - V^\pi(s_t). \tag{10}$$

Several variants of such calculation exist, including Generalised Advantage Estimation (GAE), utilising an exponentially-weighted moving average of subsequent time-step advantages (Schulman et al., 2016)

$$\delta_t^V = V(s_t) - \left[r_t - \gamma V(s_{t+1})\right] \tag{11}$$

and

$$\hat{A}_t^{GAE(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V, \tag{12}$$

where λ is an additional smoothing parameter, used along with discounting factor γ.

Such calculations can be used optionally in the VPG algorithm and its later extensions, e.g. Asynchronous Advantage Actor-Critic (A3C) (Mnih et al., 2016), Advantage Actor-Critic (A2C) (Wu et al., 2017) and Proximal Policy Optimisation (PPO) (Schulman, Wolski, Dhariwal, Radford, and Klimov, 2017). The latter algorithm further improves the policy stability by introducing a clipped surrogate objective function, limiting the amount of changes to the policy on each iteration (Schulman et al., 2017).

2. Experiments

2.1. Simulator overview and observation space

In order to assess the applicability of RL in resource allocation for project management, a dedicated simulator was designed. It can be treated as an analogy to a constrained human resource allocation task for a company in subsequent months or any other venture that requires subsequent allocation of limited resources.

It is composed of multiple discrete time steps, and in each timestep, two projects are available. Each project has a different resource allocation requirement, associated probability (chance) of success, and payout per resource allocated; it can be formalised as follows:

1. $b_i \in \mathbb{N}$ – is the balance of simulated "company" at time step i, the total amount of money available. The starting balance is indicated as b_0.
2. $res_i \in \mathbb{N}$ – is the number of resources available to the simulated "company," at step i. The starting resources are indicated as res_0.
3. $C \in \mathbb{R}$ – is the upkeep cost for each idle resource unit. If the unit is not allocated to any project, a "company" will have to pay the given amount of money for upkeep.
4. $C^+ \in \mathbb{R}$ – is the resource increase cost, incurred when the agent wants to increase the number of resources available. It replicates the real-world market (e.g. the recruitment process for new employees).
5. $N \in \mathbb{N}$ – number of discrete time steps, equal to the number of decision points.
6. $p_i^1 \in [0,1], p_i^2 \in [0,1]$ – for each -th timestep, is the probability of success for projects one and two.
7. $pay_i^1 \in \mathbb{R}, pay_i^2 \in \mathbb{R}$ – for each i-th timestep, is the reward in each project (payout) per resource allocated.
8. $d_i^1 \in \mathbb{N}, d_i^2 \in \mathbb{N}$ – for each i-th timestep, is the maximum demand for resources in projects one and two.

Therefore, the observation space in each timestep (s_i) is a vector composed of the following elements

$$s_i = \left\{ d_i^1, pay_i^1, p_i^1, d_i^2, pay_i^2, p_i^2, res_i, b_i \right\}. \tag{13}$$

The vectors with such a structure are used as input in subsequent timesteps for all agents in the simulations.

2.2. Action space

After receiving a vector indicated by equation (13), the agent must pick an action. In a discrete control setting, it can choose one action out of the following:

1. a_1 – try to allocate demand for project 1, resulting in $alloc_i^1 = \min(d_i^1, res_i)$.
2. a_2 – try to allocate demand for project 2, resulting in $alloc_i^2 = \min(d_i^2, res_i)$.

3. a_{12} – try to allocate half of the demand for project 1 and a half for project 2, resulting in $alloc_i^1 = \min\left(\frac{d_i^1}{2}, res_i\right)$ and $alloc_i^2 = \min\left(\frac{d_i^2}{2}, res_i\right)$.

4. a_0 – keep all resources idle (incurring upkeep cost), resulting in $alloc_i^0 = res_i$.

5. $a_{-x} : x \in \{0.1, 0.25, 0.5\}$ – reduce the number of resources by respectively 10/25/50%, while keeping the remaining idle (incurring upkeep cost), resulting in $alloc_i^{-x} = \min(0, (1-x) res_i)$.

6. $a_{+x} : x \in \{0.1, 0.25, 0.5\}$ – increase the number of resources by respectively 10/25/50%, while keeping the remaining idle (incurring upkeep cost), resulting in $alloc_i^{+x} = x \times res_i$).

If the number of resources changes (due to action selection), a modified number of resources will become the starting value for the next time step. Therefore:

$$res_{i+1} = \begin{cases} res_i, & \text{if } a_1, a_2, a_{12} \text{ or } a_0 \\ res_i + alloc_i^{+x}, & \text{if } a_{+x} \\ alloc_i^{-x}, & \text{if } a_{-x} \end{cases}. \tag{14}$$

2.3. Simulation objective and rewards

After the allocation is selected, the simulator will check if the selected project(s) succeeded; each project is an independent Bernoulli trial with a probability of success p_i^1, p_i^2. The agent will receive a reward, depending on the allocation choice and if the project j succeeded in time step i (indicated as $\mathbb{I}_i^j$).

The agent will receive a reward depending on the allocation choice and success of the project

$$r_i = \begin{cases} alloc_i^j \times pay_i^j \times \mathbb{I}_i^j : j \in \{1,2\}, & \text{if } a_1, a_2, a_{12} \text{ or } a_0 \\ res_i \times C, & \text{if } a_0 \\ \left(res_i \times C\right) + \left(alloc_i^{+x} \times C^+\right), & \text{if } a_{+x} \\ alloc_i^{-x} \times C, & \text{if } a_{-x} \end{cases}. \tag{15}$$

The accumulated rewards change the running balance of the agent. The goal is to maximise the rewards accumulated during the whole episode. The simulation terminates in one of three situations: when the balance is less or equal to zero, or when the agent runs out of resources, or after a predefined number of time steps.

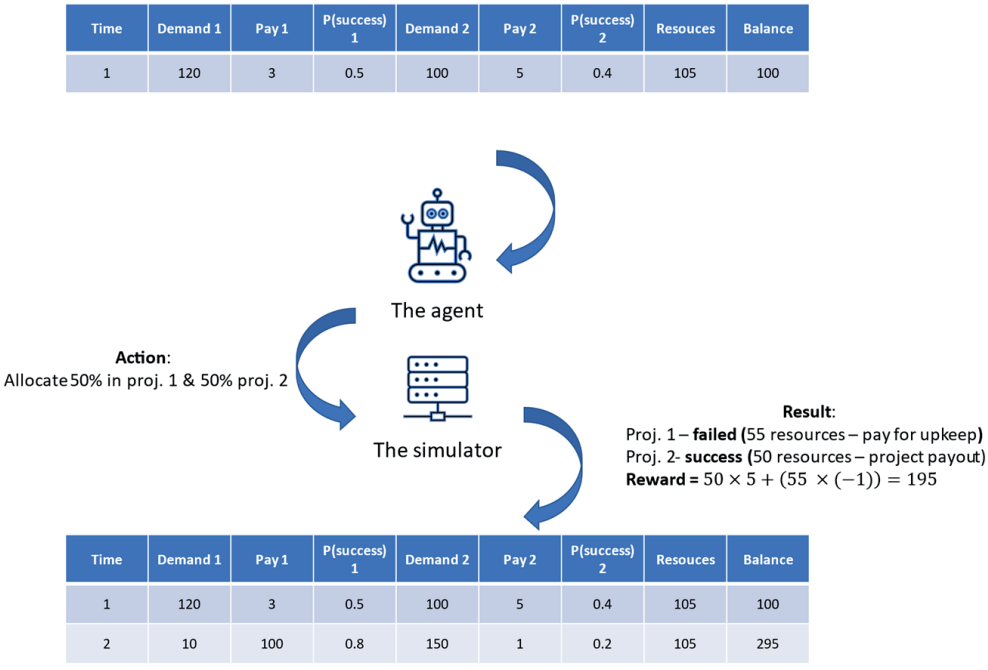

Fig. 1. Example of agent interaction

Source: own work.

2.4. Simulation variants

In order to judge how different agents behave in different conditions, three scenarios were used in the simulation. The distinctive parameter for these environments was the probability of project success capped to a min/max between 0.1.

1. In an "easy" environment, the probability of success was a random number drawn from a truncated normal distribution with $\mu_1 = 0.7$, $\sigma_1 = 0.2$. The mean chance for a project to succeed in this setting is 70%. With an initial resource pool and starting balance, it should be easy for an agent to successfully generate large incomes during simulation without risking "bankruptcy" or running out of resources too early.
2. In a "moderate" environment, the probability of success was a random number drawn from a truncated normal distribution with $\mu_2 = 0.5$, $\sigma_2 = 0.2$. On average, only 50% of the projects succeed. With an initial balance and resource pool, the agent should choose projects wisely, potentially splitting allocation or investing in new resources, to avoid bankruptcy or inability to operate.
3. In a "hard" environment, the probability of success was a random number drawn from a truncated normal distribution with $\mu_3 = 0.3$, $\sigma_3 = 0.2$. Only 30% of projects succeed in this setting. It will be tough for an agent to generate any income and avoid failure even in the early stages of the simulation.

The success probabilities mentioned before, are presented in Figure 2.

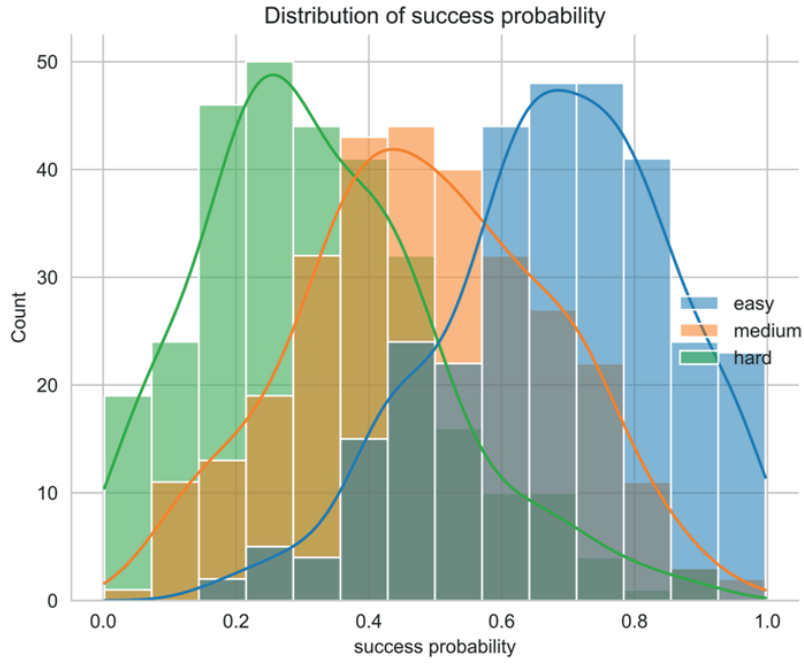

Fig. 2. Success probabilities across environments

Source: own work.

The upkeep costs were kept constant across all variants. The payouts for each project were random numbers generated from a truncated normal distribution with $\mu_{pay} = 1.0$, $\sigma_{pay} = 0.5$ capped to a min/max of between −0.5/ +1.5. Each environment consisted of 300 timesteps before termination.

2.5. Agents in the study

Five different types of agents were tested.

1. **Random agent** – treated as a benchmark, selects random actions at every timestep.

2. **Optimisation agent** – performs classic constraint optimisation on each step. It checks the expected reward for each action based on the probability of success. Therefore it seeks an action that maximises the following

$$\max_{a_i \in \{a_o, a_1, a_{12}, a_{+x}, a_{-x}\}} b_i + \mathbb{E}\left[r_i \mid p_i^1, p_i^2 \right], \tag{16}$$

where r_i is a reward as indicated in equation (15). The predictions (decisions) of this agent are deterministic.

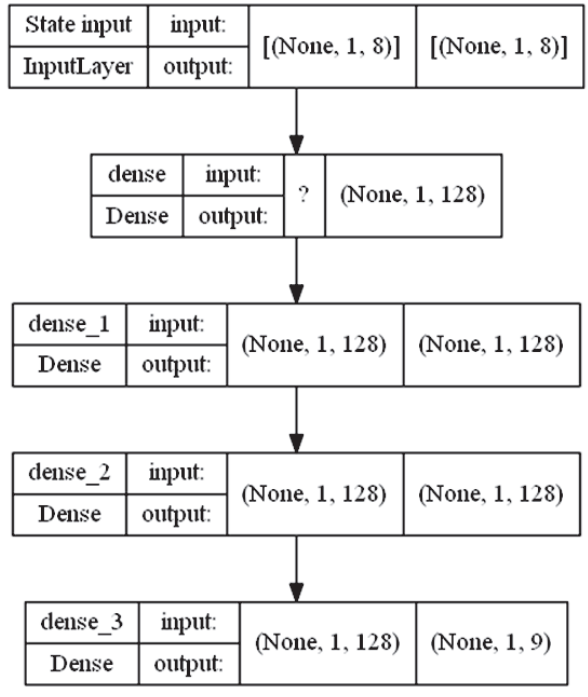

Fig. 3. Policy net architecture

Source: own work.

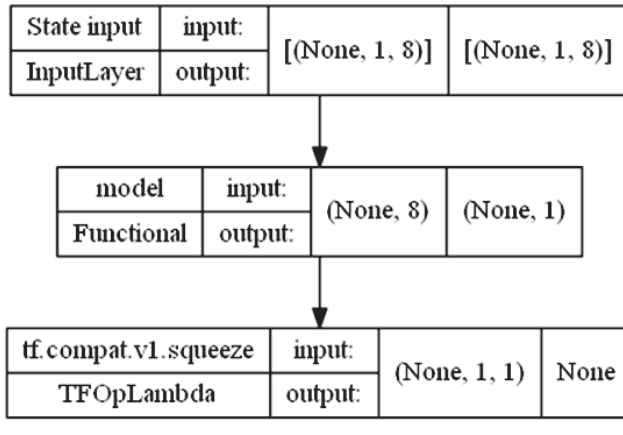

Fig. 4. Value net architecture

Source: own work.

3. **Reinforcement learning, policy gradient agents** (based on the algorithms described in Section 1.2):
   a. **The VPG agent**, composed of "Actor" and "Critic" neural networks. The agent was improved with a GAE calculation for the full Monte Carlo episode for better performance and variance stability.
   b. **The A2C agent** with *N*-step GAE advantage estimation. During the tuning of the parameters, the optimal number of steps was set to 25.
   c. **The PPO agent** with *N*-step GAE advantage estimation and surrogate objective clipping function was set to 0.2.

The Actor and Critic neural networks for all the aforementioned agents have the same architecture and are presented in the figures 3 and 4.

The inner network layers utilised a hyperbolic tangent activation function due to its scaling properties. The outer layer of a critic network utilised a linear activation function, while the actor-network performed a softmax classification to one of the available actions.

2.6. Experimental design

The testing procedure remained the same in all the environments. It consisted of the following steps:

1. Check the score of each agent on 500 independent, random simulation iterations before the training or fitting procedure.
2. Fit the agent (if applicable) on 300 independent, random simulation iterations.
3. Check the final scores on 500 independent, random simulation iterations. Store the results.
4. Perform statistical tests on gathered data – scores after the training.
5. Assumptions:
   a. Each test score is independent from the others and identically distributed – each model is trained on 500 random episodes. The simulation environment is reset every time.
   b. Each set of results per model was checked for normality using the Shapiro Wilk test; because of skewness, the tests did not confirm that data comes from a normal distribution.
   c. Equality of variance across model scores was checked using the Levene test. In each case the null hypothesis was rejected, so population variances cannot be considered equal.
6. According to the research (Arcuri and Briand, 2014; Colas, Sigaud, and Oudeyer, 2019), the Welch *T*-test with *p*-value correction, followed by *post-hoc* pairwise testing, is considered the most robust in comparing RL algorithms results violating equality of variance and normality assumptions. Considering the large number of independent samples ($N = 500$), the Welch *T*-test on a stricter significance level $\alpha = 0.01$ can be considered as the primary testing procedure (Arcuri and Briand, 2014). It yielded the lowest false positive errors compared to the non-parametric Mann-Whitney or ranked *t*-tests (Colas et al., 2019). Therefore, the following methods were utilised for each simulation setup:

a. Perform a single Welch ANOVA on a significance level $\alpha = 0.01$, to test if there is a significant difference in the scores between all agents.
b. Perform a series of post-hoc tests between pairs of agents to judge which one in the pair performs better. The method of choice for pairwise comparison was the Games-Howell test, as it can be combined with the Welch ANOVA in a non-equal variances setup (Bagherzadeh, Kahani, and Briand, 2021; Games and Howell, 1976).

3. Results and discussion

3.1. Results overview

The table 1 presents the mean, median and standard deviation of the results achieved by each agent in the simulations. The numbers in brackets indicate the standard deviation for the scores. The maximal score for a given setup is given in bold. All the scores were rounded up to the third decimal place.

Table 1. Scores of each agent in each environment

Setup/Agent	Random	Optimization	VPG	AC	PPO
Easy	−290.89 −270.731 (181.360)	**5400.996** **5423.375** (619.269)	4681.178 4680.816 (437.656)	4685.794 4690.119 (425.214)	5221.025 5367.668 (653.343)
Medium	−266.279 −171.603 (187.549)	1696.866 2050.942 (1081.587)	1649.711 1721.975 (653.539)	1524.030 1640.094 (668.603)	**3145.002** **3471.829** (1426.586)
Hard	−268.207 −171 (185.856)	−115.573 −113.268 (12.073)	−73.364 −72.109 (9.932)	137.743 −107.305 (531.494)	**529.445** **532.573** (134.387)

Numbers represent mean/median/(std) of scores.

Source: own work.

The boxplot below presents a graphical distribution of each model's scorings in each environment. The bounds of the box represent the 25th and 75th percentile, while the line in the middle – the median. The Upper/lower whiskers are the lowest/highest values that are within 2.5 standard deviations from the mean. The more spread or the longer the box is – the more stretched the scorings distribution.

Looking at the table and plots, it is clear that the PPO algorithm performed best in the most challenging environments, i.e. "moderate" and "hard". As described in the section "experimental design", a series of statistical tests were performed to judge the significance of differences between the models: Welch ANOVA on significance

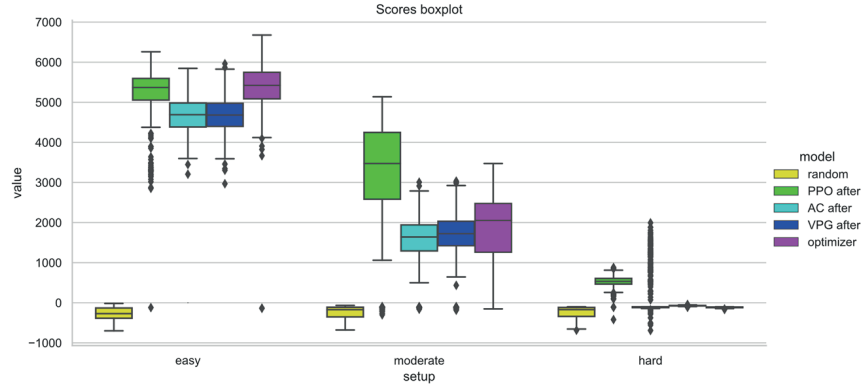

Fig. 5. Distribution of scores per each model

Source: own work.

level $\alpha = 0.01$ to evaluate the overall differences, and a series of pairwise comparisons using the Games-Howell method, followed by a p-value correction for controlling the false discovery level; the results are presented below. In all the tables, the top row represents the Welch ANOVA test with degrees of freedom 1 (treatment levels/groups), degrees of freedom 2 (no. of observations – no. of groups), the test statistic, and p-value. The subsequent rows represent pairwise model comparisons, with the score's difference, its standard deviation, the test statistic, and Hedge's effect size (*/**/***– small/medium/large) (Ferguson, 2009; Sullivan and Feinn, 2012). The insignificant comparisons (with $pval \geq 0.01$) were greyed out. In each pair, the model with better results was highlighted in bold and underlined.

Table 2. Pairwise comparisons of agents on the "easy" environment

Model A	Model B	Score diff	Diff se	Statistic	P-val	Eff. size
AC	**PPO**	−535.23	34.86	−15.35	<<0.001	−0.97***
AC	VPG	4.62	27.29	0.17	0.9	0.01*
AC	**optimiser**	−715.2	33.59	−21.29	<<0.001	−1.35***
AC	random	4976.68	20.67	240.73	<<0.001	15.21***
PPO	VPG	539.85	35.17	15.35	<<0.001	0.97***
PPO	**optimiser**	−179.97	40.26	−4.47	<<0.001	−0.28**
PPO	random	5511.91	30.32	181.77	<<0.001	11.49***
VPG	**optimiser**	−719.82	33.91	−21.23	<<0.001	−1.34***
VPG	random	4972.07	21.19	234.68	<<0.001	14.83***
optimiser	random	5691.89	28.86	197.24	<<0.001	12.47***

In the table: conducted statistical test is Welch ANOVA with degrees of freedom equal to 4 and 1151.14 (for groups 1 and 2 respectively), F-statistic equal to 34759.39, and overall test p-value << 0.001

Source: own work.

Table 3. Pairwise comparisons of agents on the "moderate" environment.

Model A	Model B	Score diff	Diff se	Statistic	P-val	Eff. size
AC	**PPO**	−1620.97	70.46	−23.01	<<0.001	−1.45***
AC	VPG	−125.68	41.81	−3.01	0.02	−0.19
AC	optimiser	−172.84	56.87	−3.04	0.02	−0.19
AC	random	1790.31	31.05	57.65	<<0.001	3.64***
PPO	VPG	1495.29	70.17	21.31	<<0.001	1.35***
PPO	optimiser	1448.14	80.06	18.09	<<0.001	1.14***
PPO	random	3411.28	64.35	53.01	<<0.001	3.35***
VPG	optimiser	−47.16	56.51	−0.83	0.9	−0.05
VPG	random	1915.99	30.41	63.01	<<0.001	3.98***
optimiser	random	1963.14	49.09	39.99	<<0.001	2.53***

In the table: conducted statistical test is Welch ANOVA with degrees of freedom equal to 4 and 1080.87 (for groups 1 and 2 respectively), F-statistic equal to 2581.87, and overall test p-value << 0.001.

Source: own work.

Table 4. Pairwise comparisons of agents on the "hard" environment.

Model A	Model B	Score diff	Diff se	Statistic	P-val	Eff. size
AC	**PPO**	−391.7	24.52	−15.98	<<0.001	−1.01***
AC	VPG	211.11	23.77	8.88	<<0.001	0.56**
AC	optimiser	253.32	23.78	10.65	<<0.001	0.67**
AC	random	405.95	25.18	16.12	<<0.001	1.02***
PPO	VPG	602.81	6.03	100.03	<<0.001	6.32***
PPO	optimiser	645.02	6.03	106.89	<<0.001	6.76***
PPO	random	797.65	10.26	77.77	<<0.001	4.91***
VPG	optimiser	42.21	0.7	60.37	<<0.001	3.82***
VPG	random	194.84	8.32	23.41	<<0.001	1.48***
optimiser	random	152.63	8.33	18.33	<<0.001	1.16***

In the table: conducted statistical test is Welch ANOVA with degrees of freedom equal to 4 and 1136.26 (for groups 1 and 2 respectively), F-statistic equal to 3694.87, and overall test p-value << 0.001.

Source: own work.

In the "easy" environment, where the chance for success for each project is relatively high, the agent utilising classic optimisation procedures performed the best, followed by the PPO and AC agents, which proved to be slightly less effective. This difference can be attributed to the stochastic nature of the RL algorithms and their inherent instability – as machine learning procedures, their results may vary. When the chance of success is very high, the sequential problem reduces to subsequent single-step, greedy optimisation tasks, for which classic tools proved to be the best choice.

In both the "moderate" and "hard" environments, the PPO agent significantly outperformed both the classic optimisation method and the other RL algorithms. These differences' size of effect and absolute value are large, measured in thousands for the "moderate" environment. This effect can be attributed to the fact that the RL algorithms possess the ability to plan in the long-term horizon due to reward discounting and advantage estimation, which is not possible for more straightforward, greedy optimisation tools.

In conclusion, the study shows that deterministic optimisation remains the best choice for stable, well-defined environments. Advanced RL methods, such as PPO, are best suited for challenging, stochastic environments connected with uncertainty.

3.2. Discussion

The simulation presented in this study is far from being a perfect model of a real resource allocation scenario.

The first improvement and future study direction could be to transform it into a continuous control problem, where an agent can allocate an exact number of resources or their proportion to projects instead of discrete chunks. Problem simplification to discrete actions instead of numerical ones is standard for many RL algorithms. Such an approach can be problematic when the dimensionality of the action vector is large (Dulac-Arnold et al., 2015; Lillicrap et al., 2016; Smart and Kaelbling, 2000). Therefore it will be a natural extension to transform the simulator presented in this study to operate in numerical action-space, and properly adjust the agents' policies.

Other improvements could include resource-locking for long-term projects, nondeterministic project shutdowns, and sudden resource outages to make the simulation more realistic.

4. Conclusion

The simulation experiment performed in this study was supposed to replicate sequential resource allocation processes similar to these in various project management tasks. The classic resource optimisation algorithm was compared to RL algorithms on different difficulty levels to judge the performance of such methods. Strict statistical experimentation proved that the PPO algorithm performed best out of all the tested methods in a more challenging setup connected with uncertainty.

The implementation of a simulator presented in this paper can be treated as a starting point for more complicated designs that include resource locking, changing market conditions, and continuous control.

References

Ackoff, R. L. (1956). The development of operations research as a science. *Operations Research*, *4*(3). https://doi.org/10.1287/opre.4.3.265

Anderson, C. (2015). *Creating a data-driven organization*. O'Reilly Media.

Arcuri, A., and Briand, L. (2014). A Hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering. *Software Testing Verification and Reliability*, *24*(3). https://doi.org/10.1002/stvr.1486

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. (2017). Deep reinforcement learning: A brief survey. *IEEE Signal Processing Magazine*, *34*(6). https://doi.org/10.1109/MSP.2017.2743240

Bagherzadeh, M., Kahani, N., and Briand, L. (2021). Reinforcement learning for test case prioritization. *IEEE Transactions on Software Engineering*. https://doi.org/10.1109/TSE.2021.3070549

Bellman, R. (1954). The theory of dynamic programming. *Bulletin of the American Mathematical Society*, *60*(6). https://doi.org/10.1090/S0002-9904-1954-09848-8

Bhimani, A., and Willcocks, L. (2014). Digitisation, Big Data and the transformation of accounting information. *Accounting and Business Research*, *44*(4). https://doi.org/10.1080/00014788.2014.910051

Chiang, H. Y., and Lin, B. M. T. (2020). A decision model for human resource allocation in project management of software development. *IEEE Access*, *8*. https://doi.org/10.1109/ACCESS.2020.2975829

Colas, C., Sigaud, O., and Oudeyer, P. Y. (2019). A Hitchhiker's guide to statistical comparisons of reinforcement learning algorithms. *RML@ICLR 2019 Workshop – Reproducibility in Machine Learning*.

Duan, Y., Edwards, J. S., and Dwivedi, Y. K. (2019). Artificial intelligence for decision making in the era of Big Data – evolution, challenges and research agenda. *International Journal of Information Management*, *48*. https://doi.org/10.1016/j.ijinfomgt.2019.01.021

Dulac-Arnold, G., Evans, R., van Hasselt, H., Sunehag, P., Lillicrap, T., Hunt, J., … and Coppin, B. (2015). Deep reinforcement learning in large discrete action spaces. *ArXiv Preprint ArXiv:1512.07679*.

Farhang Moghaddam, B. (2019). Mapping optimization techniques in project management. *Journal of Project Management*. https://doi.org/10.5267/j.jpm.2019.3.003

Ferguson, C. J. (2009). An effect size primer: A guide for clinicians and researchers. *Professional Psychology: Research and Practice*, *40*(5), 532.

Games, P. A., and Howell, J. F. (1976). Pairwise multiple comparison procedures with unequal N's and/or variances: A Monte Carlo Study. *Journal of Educational Statistics*, *1*(2). https://doi.org/10.3102/10769986001002113

Giannoccaro, I., and Pontrandolfo, P. (2002). Inventory management in supply chains: A reinforcement learning approach. *International Journal of Production Economics*, *78*(2). https://doi.org/10.1016/S0925-5273(00)00156-0

Gupta, S., Modgil, S., Bhattacharyya, S., and Bose, I. (2022). Artificial intelligence for decision support systems in the field of operations research: review and future scope of research. *Annals of Operations Research*, *308*(1-2). https://doi.org/10.1007/s10479-020-03856-6

Huang, Z., van der Aalst, W. M. P., Lu, X., and Duan, H. (2011). Reinforcement learning based resource allocation in business process management. *Data and Knowledge Engineering*, *70*(1). https://doi.org/10.1016/j.datak.2010.09.002

Institute, P. M. (2021). *Guide to the Project Management Body of Knowledge (PMBOK Guide) and the Standard for Project Management*. Project Management Institute.

Islam, M. N. (2011). Crashing project time with least cost: A linear programming approach. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1012525

Jędrzejowicz, P., and Ratajczak-Ropel, E. (2013). Reinforcement learning strategy for A-Team solving the resource-constrained project scheduling problem. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8083 LNAI*, 457-466. https://doi.org/10.1007/978-3-642-40495-5_46

Kane, H., and Tissier, A. (2012). A resource allocation model for multi-project management. *9th International Conference on Modelling, Optimization & SIMulation*.

Koulinas, G., Xanthopoulos, A., Kiatipis, A., and Koulouriotis, D. (2018). A summary of using reinforcement learning strategies for treating project and production management problems. *2018 13th International Conference on Digital Information Management, ICDIM 2018*, 33-38. https://doi.org/10.1109/ICDIM.2018.8847099

Li, R., Zhao, Z., Sun, Q., Chih-Lin, I., Yang, C., Chen, X., Zhao, M., and Zhang, H. (2018). Deep reinforcement learning for resource management in network slicing. *IEEE Access*, *6*, 74429-74441. https://doi.org/10.1109/ACCESS.2018.2881964

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *4th International Conference on Learning Representations, ICLR 2016 – Conference Track Proceedings*.

Manikantan, P., and Gurusamy, S. (2016). Impact of knowledge sharing on project management in the IT industry – An empirical investigation. *SUMEDHA Journal of Management*, *5*(2).

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. *HotNets 2016 – Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. https://doi.org/10.1145/3005745.3005750

Mnih, V., Badia, A. P., Mirza, L., Graves, A., Harley, T., Lillicrap, T. P., Silver, D., and Kavukcuoglu, K. (2016). Asynchronous methods for deep reinforcement learning. *33rd International Conference on Machine Learning, ICML 2016*, *4*.

Mousavi, S. S., Schukat, M., and Howley, E. (2018). Deep reinforcement learning: An overview. In *Lecture notes in networks and systems* (Vol. 16). https://doi.org/10.1007/978-3-319-56991-8_32

Schulman, J. (2016). *Optimizing expectations: From deep reinforcement learning to stochastic computation graphs*.

Schulman, J., Moritz, P., Levine, S., Jordan, M. I., and Abbeel, P. (2016). High-dimensional continuous control using generalized advantage estimation. *4th International Conference on Learning Representations, ICLR 2016 – Conference Track Proceedings*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *ArXiv Preprint ArXiv:1707.06347*.

Schwindt, Ch. (2010). *Resource allocation in project management*. Springer.

Selaru, C. (2012). Resource allocation in project management. *International Journal of Economic Practices and Theories*, *2*(4), 274-282.

Sharda, R., Delen, D., and Turban, E. (2020). *Analytics, data science, & artificial intelligence: Systems for decision support*. Pearson Education Limited.

Shyalika, C., Silva, T., and Karunananda, A. (2020). Reinforcement learning in dynamic task scheduling: A review. *SN Computer Science 2020 1:6*, *1*(6), 1-17. https://doi.org/10.1007/S42979-020-00326-5

Smart, W. D., and Kaelbling, L. P. (2000). Practical reinforcement learning in continuous spaces. *Proceedings of the Seventeenth International Conference on Machine Learning*.

Sullivan, G. M., and Feinn, R. (2012). Using effect size – or why the P value is not enough. *Journal of Graduate Medical Education*, *4*(3), 279-282.

Sutton, R., Bach, F., and Barto, A. (2018). *Reinforcement learning : An introduction.* MIT Press Ltd.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in Neural Information Processing Systems*, *12*.

Wu, Y., Mansimov, E., Grosse, R. B., Liao, S., and Ba, J. (2017). Scalable trust-region method for deep reinforcement learning using Kronecker-factored approximation. *Advances in Neural Information Processing Systems*, *30*.

Xu, Y., Zhao, Z., Cheng, P., Chen, Z., Ding, M., Vucetic, B., & Li, Y. (2021). Constrained Reinforcement Learning for Resource Allocation in Network Slicing. *IEEE Communications Letters*, *25*(5), 1554-1558. https://doi.org/10.1109/LCOMM.2021.3053612

Yan, Y., Chow, A. H. F., Ho, C. P., Kuo, Y.-H., Wu, Q., and Ying, C. (2021). Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3935816

Ye, H., Li, G. Y., and Juang, B.-H. F. (2019). Deep reinforcement learning based resource allocation for V2V communications. *IEEE Transactions on Vehicular Technology*, *68*(4), 3163. https://doi.org/10.1109/TVT.2019.2897134

Ye, K., Shi, X., Li, H., and Shi, N. (2014). Resource allocation problem in port project portfolio management. *Proceedings – 2014 7th International Joint Conference on Computational Sciences and Optimization, CSO 2014*, 159-162. https://doi.org/10.1109/CSO.2014.36

Yu, L., Zhang, C., Jiang, J., Yang, H., and Shang, H. (2021). Reinforcement learning approach for resource allocation in humanitarian logistics. *Expert Systems with Applications*, *173*. https://doi.org/10.1016/j.eswa.2021.114663

Yuan, Y., Li, H., and Ji, L. (2021). Application of deep reinforcement learning algorithm in uncertain logistics transportation scheduling. *Computational Intelligence and Neuroscience*. https://doi.org/10.1155/2021/5672227

Zuo, J., and Joe-Wong, C. (2021). Combinatorial multi-armed bandits for resource allocation. *2021 55th Annual Conference on Information Sciences and Systems, CISS 2021*. https://doi.org/10.1109/CISS50987.2021.9400228