32nd European Modeling & Simulation Symposium
17th International Multidisciplinary Modeling & Simulation Multiconference

ISSN 2724-0029 ISBN 978-88-85741-44-7 © 2020 The Authors.
DOI: 10.46354/i3m.2020.emss.015

Attaining global optimized solution by applying Q-learning

Ghulam Muhammad Ali[1,*], Asif Mansoor[1], Shuai Liu[1], Jacek Olearczyk[1], Ahmed Bouferguene[2] and Mohamed Al-Hussein[1]

[1]Department of Civil and Environmental Engineering, University of Alberta, 9105 116 Street NW, Edmonton, T6H 2W2, Canada
[2]Campus Saint-Jean, University of Alberta, Edmonton, T6C 4G9, Canada

[*]Corresponding author. Email address: gmali@ualberta.ca

Abstract

The assembly of modular construction projects depends primarily on the use of heavy capacity cranes. The increase in usage of heavy cranes is the impetus for the optimization of resources utilized for the crane work. One of the major cost drivers is the design optimization process of such resources. Even for an initial mechanical/structural design prior to a prototype, the design phase with finite element analysis (FEA) is time-consuming and extensive work. The greedy algorithm could be an answer to these problems to accomplish the optimization in short period of time. In some of the complex cases, the greedy algorithm can confine to the local optimum and thus, overlooking the global optimum. To avoid this, a model-free reinforcement learning (RL) algorithm, Q-learning, is employed in this study to evaluate its suitability for use in the design optimization process with the use of FEA. A hypothetical structural support design problem is used to formulate a framework for comparing the greedy algorithm and Q-learning. The findings show that not only can Q-learning overcome the local optimum confinement of the greedy algorithm, but it in fact can surpass the greedy algorithm along the progressive iteration, by refining policy and reward.

Keywords: Greedy algorithm; reinforcement learning; Q-learning; finite element analysis; structural design

1. Introduction

The modern heavy construction industry is gradually adopting modularization as its primary design and project delivery paradigm. The most noteworthy change is the shifting of the most substantial portion of work from onsite construction to offsite construction. To further reduce the amount of onsite construction work required, modules are designed and fabricated to include an increasing number of functionalities and packages (e.g., plumbing, electrical, finishing, etc.). Therefore, maximizing the functionality of a module leads to an increase in the weight of the module from tens to hundreds of tons. The larger, heavier modules are also having an impact on resource utilization for crane work like crane mats, rigging gears, assisting cranes, and trailers. Given this new weight constraint, heavy cranes have evolved to be able to lift heavier modules. However, this high crane capacity is synonymous with cranes that have heavier weights and increased structural complexity. Heavier cranes require better crane ground support for safe crane operation. To overcome this challenge, the crane industry in Canada uses 2–3 layers of matting for crane work. The layering of timber mats is motivated by the concept of extra safety for crane stability on site. A study by Occupational Safety and Health Administration, USA, (OSHA) reported that, from 2000 to 2009 in the United

States, approximately 50 deaths were caused by "Crane Tipped Over", which is directly associated with poor ground support (Zhao, 2011). Recently, 937 construction workers lost their lives at work, of which 505 (54%) of those deaths were linked with mobile crane work (Kan, Zhang, Fang, Anumba & Messner, 2018). To address this problem, the design of crane mats needs to be revised to provide a better optimized crane support solution, which can be accomplished with the application of finite element analysis (FEA). The traditional way is to do the design calculations, and after getting the design ready, it is examined using FEA for any known or unknown risks associated with the design. These models are used by engineers and designers as the basis for discussion during which weaknesses are identified and improvements are proposed. This is an extensive amount of work, which can be shortened using the greedy algorithm to achieve the optimized design solution (Cormen, Leiserson, Rivest & Stein, 2009). But one of the drawbacks of the greedy algorithm is that the agent, after reaching the local optimum, gets confined to the local optimum (Bang-Jensen, Gutin & Yeo, 2004; Gutin, Yeo & Zverovich, 2002). To overcome this situation, the agent needs to explore the area beyond the local optimum by increasing the number of layers or steps for further exploration (future steps). The greedy agent must probe the layers down the heuristic tree for the minimum or maximum point to proceed further, similar to A* algorithm, without storing any data (Doran & Michie, 1966). To avoid this scenario, reinforcement learning (RL) is used in the present study to accomplish the global optimal solution. A hypothetical structural support model is used for purposes of comparing the greedy algorithm and model-free RL algorithm, Q-learning (Sutton & Burto, 2018; Watkins, 1989; Watkins & Dayan, 1992). The study results show that Q-learning can easily overcome the local optimum to obtain the global optimal point or final state. Not only that, the cumulative time duration for FEA to reach the final state also decreases as the number of iterations increases. The results indicate that the policy and reward value function get refined by the RL agent by performing the act repeatedly. The number of actions taken by the agent is also minimized during the iterative process.

2. Literature review

The greedy algorithm is probably one of the most widely used meta-heuristic optimization methods used to identify an optimal location for problems that are known to be time consuming when tackled with deterministic methods. However, in some complex situations, the greedy algorithm goes to a quarantine state after landing at a local optimal location. This can be surpassed if the user is aware of the number of layers (future steps) down the greedy heuristic tree to explore before taking a decision (Bang-Jensen et al, 2004; Cormen et al, 2009).

RL is a branch of machine learning and like any other machine learning paradigm, the concept of RL is based on the core methodology of learning in nature. The interaction of infant with the environment creates a wealth of information portraying cause and effect, which leads to the achievement of goals. Such interactions are a major source of knowledge to decide what to do next (Hilgard, Marquis & Kimble, 1961). Taking the same behavior and approach into consideration, RL can be considered for the optimization problems to initiate an action or to develop a strategy to achieve the goal, with the addition of reward or punishment as the guideline for action selection. RL is slightly different from supervised learning due to the non-availability of a training dataset, much like an infant who grasps knowledge from the environment in absence of a training dataset. The structure of RL is composed of four parts: policy, a reward signal, a value function, and a model. A policy is the way a RL agent is to behave at a given time. The value function defines the amount of reward and punishment the RL agent gets, and the model (optional) mimics the behavior of the environment (Sutton & Barto., 2018). The basic RL concept is shown in Figure 1. Addition of model can expediate the process of learning and convergence of policy and value function.

Q-learning is one of the RL algorithms with a model-free approach that can optimize the stochastic states and rewards (Lillicrap et al, 2015; Sutton & Barto, 2018). Q-learning is based on a finite Markov decision process (Bellman, 1957). Q-learning was first introduced in 1989 (Watkins, 1989). The learning process follows a similar pattern as that of temporal difference (Sutton, 1988). The RL agent learns from the current state reward or punishment and takes a decision based on the expected future state and updates the current state accordingly. The equation of Q-learning is as shown in Eq. (1):

$$Q(s_t, a_t) = Q(s, a_t)(1-\alpha) + \alpha\{R_t + \gamma\, maxQ_a(s_{t+1}, a)\} \qquad (1)$$

where $Q(s_t, a_t)$ is the value of Q at the state s_t after expediating the action a_t, α is the learning rate, γ the discount factor, R_t the reward received by the agent at state s_t after taking the action a_t based on the value function for reward, a the next action to reach s_{t+1} and $maxQ_a$ is the value of Q of the next state (from a set of possible immediate future states) with maximum Q value. This way, the value of Q for each state is refined and updated with each episode. The policy of the RL agent is to maximize the reward and to minimize the steps (number of actions) in between. The action taken by the agent depends on the future maximum Q value, not the reward it will get at that state, so it is known as an off-policy RL algorithm (Sutton & Barto, 2018; Watkins, 1989; Watkins & Dayan, 1992).

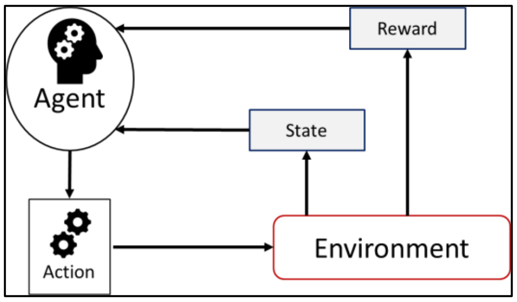

Figure 1. Basics of reinforcement learning (RL)

Looking at the Q-learning algorithm, there are two factors involved: learning rate α and the discount factor γ. Both can be any value between 0 and 1. The learning rate determines how much the newly obtained information overrides the previous collected information. If the value of the learning rate is 0, the agent will learn nothing. In contrast to that, the learning rate of 1 drives the agent to consider only the most recent information. The discount factor defines the relevance of future reward. A value of 0 for the discount factor makes the future value null and void and the RL agent becomes "shortsighted" (Sutton & Barto, 2018). On the other hand, a discount factor value of 1 will urge the agent to grab future reward more strongly, making it "farsighted" (Sutton & Barto, 2018). A lot of research work has been done to compare the results with various learning and discount factor combinations. Most researchers have used a learning factor of 0.1 and a discount factor of 0.9 (Sutton & Barto, 2018).

The same application of RL is used in this study to train an agent to reach the global optimized location by trial and error. As the agent proceeds with the iterations, the value of Q is updated continuously. The Q-table is updated for states involved in each episode (i.e., each iteration) accordingly. The Q-learning will motivate the value function for maximization and the policy to the shortest path, decreasing the time required to reach the global optimization solution. A learning factor of 0.1 and a discount factor of 0.9 are employed in this simulation.

3. Methodology

The methodology is divided into three main parts, the hypothetical structural support design, the greedy algorithm and its limitation, and the development of the Q-learning algorithm.

3.1. Hypothetical structural support

A hypothetical design is formulated for the purposes of comparing RL with the greedy algorithm in the context of design optimization. A hypothetical steel plate of 0.25 m thick with the dimensions of 112 m × 112 m is supported by four supports (as shown in Figure 2). There are a couple of weights, forces, and moments acting randomly and simultaneously on the plate as shown in Figure 2 & Figure 3.

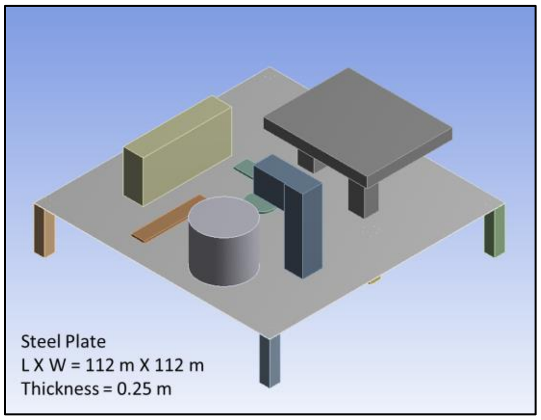

Figure 2. Basic configuration of steel plate with random objects

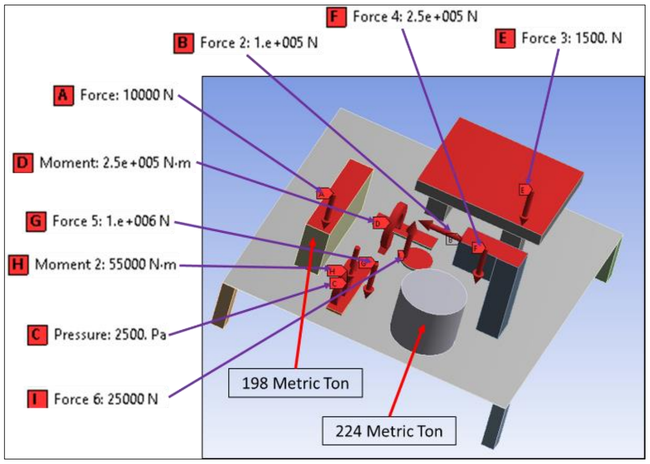

Figure 3. Steel plate with randomly acting forces and moments

The four supports are permanent at the corners of the plate. There is another adjustable support, which is meant to be placed anywhere under the plate (as shown in Figure 4). There are 10^4 possible locations for the adjustable support. The objective is to find the location using an FEA platform, so that the deflection in the steel plate is the minimum. ANSYS (19.2) is used as the FEA platform for the simulation.

It is important to mention that this study only consists of the comparison between greedy algorithm and RL for design purposes. One of the applications in future could be the design of a crane mat. The current design problem is similar to crane mat design in that there are many forces (weights), such as the crane superstructure, crane boom, payload, counterweights, and crane undercarriage, acting on a crane mat at different locations. For a crane mat, the placement of a supporting element, like H-beam, I-Beam or plates, requires optimization given the variations in load and crane radius. The research methodology described herein, is intended to be used later with slight modifications for the design optimization of a crane mat for use under crawler crane tracks, with varying loads and crane radiuses. The main purpose of crane mats is to distribute the load acting on them with a minimum amount of crane mat deflection. Therefore, the objective of crane mat design and the current design problem is to find the location of the support (H-beam, I beam or plate in case of crane mat) to achieve minimum deflection. Later, this objective can be expanded by adding cost and fabrication constraints. The surface area of the plate is assumed to be 112 m × 112 m, instead of traditional mat dimensions (3.6 m × 2.4 m), to observe and differentiate the variations in greater detail. For crane mat design optimization, the steel plate size will be reduced to match the traditional crane mat dimensions.

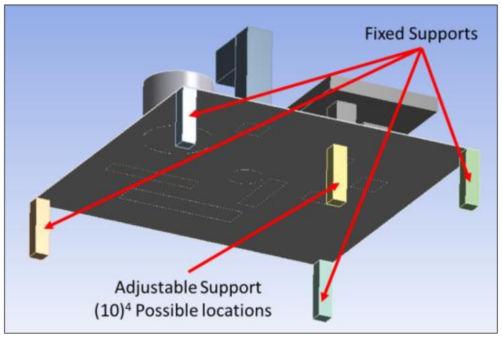

Figure 4. Steel plate with adjustable support

The overall process is shown in Figure 5. The RL agent is not aware of the model, it can only select the action in the form of location of the adjustable support and based on the action, it receives the reward and new state to move on. ANSYS takes approximately 30–40 seconds on average to simulate each location of the adjustable support and delivers the deflection of the plate as the outcome of each state. The deflection in the plate determines whether the action was favorable or not, which in return, making use of the value function, provides a scalar value of reward or punishment to the agent.

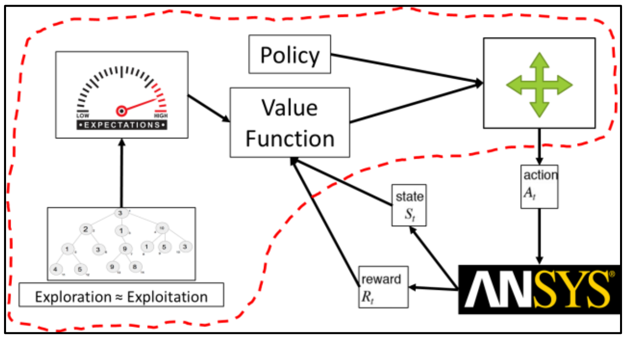

Figure 5. Basics model of RL

The first approach was to explore all the possible locations and the respective deflection. The topography of the deflection with respect to each state is shown in Figure 6. The latitudinal movement is stated as X-move (100 steps), and the longitudinal movement is stated as Y-move (100 steps). There are 10,000 combinations of X-move and Y-move, as the position of the fifth support under the plate, to minimize the deflection. The optimal location is also indicated in Figure 6. The total time taken by FEA to explore all the states was 101.94 hours, which is equal to 4.25 days in total.

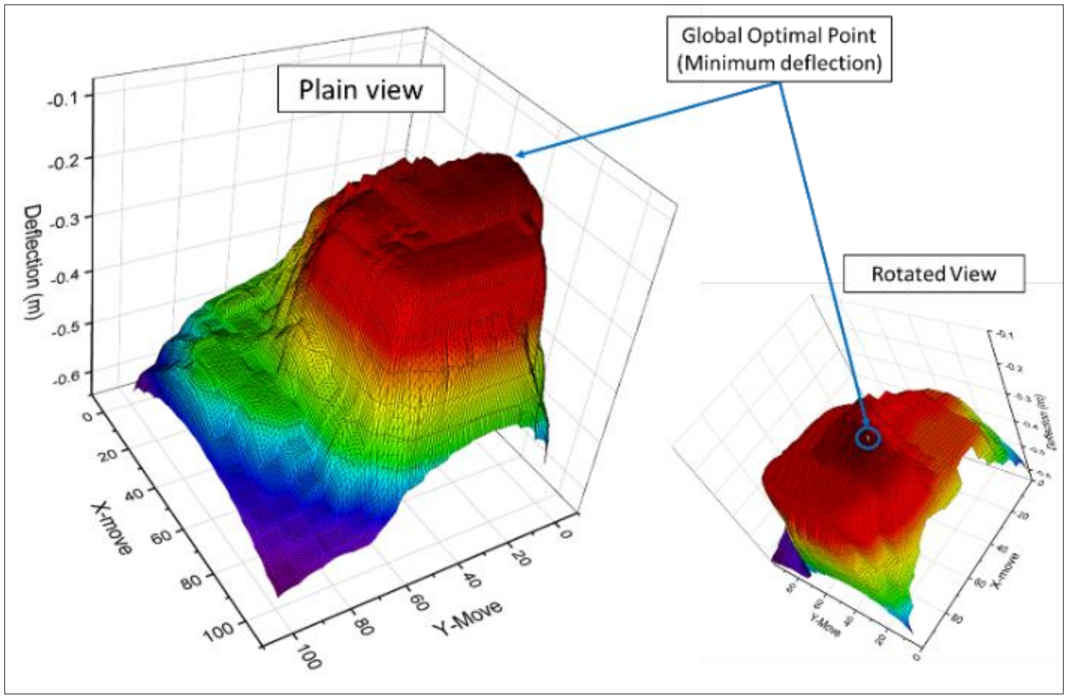

Figure 6. All the possible states with deflection

3.2. Greedy algorithm approach and its limitations

For both RL and greedy algorithm, for current contribution, it is assumed that the final deflection is known, but not the final location of the support. The easiest way to reach the optimal location starting from any edge is to use the greedy algorithm; however, the problem is that the greedy algorithm can confine to local optimal location and the local optimum location can become a sink for the greedy agent. This is the same as in the case of this problem. There are many local optimal locations. One of them is shown in Figure 7.

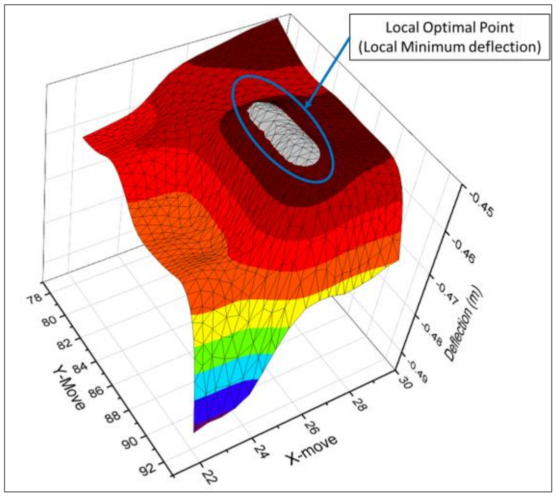

Figure 7. Local optimization location

The greedy algorithm agent can only transcend this local optimization point by probing the heuristic tree further below (future actions). It is not easy to foretell the number of layers of the heuristic tree to be explored. It can be a trial and error method to search the level where the greedy algorithm can find the minimum to maximum point to proceed further to the global optimization point. For this problem, for some of the local optimization locations, the greedy agent goes to several steps down in the heuristic tree to find the path towards the minimum deflection value (global optimum).

3.3. Development of Q-learning algorithm

Due to the limitations of the greedy algorithm, RL is explored. Q-learning is used as the RL agent to search the minimum deflection. RL is a trade-off between exploration and exploitation. If the maximum states are not explored, the agent will promote for exploration. As the percentage of explored states increases, the RL agent switches from exploration to exploitation for the refining of policy and reward value function. The exploration for the RL agent is defined in Eq. (2).

$$P_t = \begin{cases} 1 - \dfrac{\sum_{j=1}^{m} s_{xj}}{\sum_{i=1}^{n} s_i} & if \ \dfrac{\sum_{j=1}^{m} s_{xj}}{\sum_{i=1}^{n} s_i} > 0.1 \\ 0.1 & if \ \dfrac{\sum_{j=1}^{m} s_{xj}}{\sum_{i=1}^{n} s_i} \le 0.1 \end{cases} \qquad (2)$$

where P_t is the probability for exploration at current state, s_t, $\sum_{j=1}^{m} s_{xj}$ is the sum of the states explored after reaching s_t throughout all the episodes completed, and $\sum_{i=1}^{n} s_i$ is the total states $(i, j = 1,2,3, \dots, 10{,}000)$. As the states explored increase, the probability of exploration decreases, and agent tends to move toward exploitation instead of exploration.

One of the major features of RL is the value function for reward. There are two ways to define the reward function, one is sparse reward function and the other one is the shaping reward function. The sparse function provides a large quantity of a scalar reward value after reaching the final state. On the other hand, the shaping reward function provides a fraction of the final reward on each state, and increases the intensity of the reward as the agent moves closer to the final state (Gullapalli & Barto, 1992). The shaping reward function is used in the present study to expediate the learning of the agent. The shaping reward function is formulated in Eq. (3).

$$R_t = \begin{cases} -\left(\dfrac{D_t - D_f}{D_{max} - D_f}\right)^{sf} & if \ (D_t - D_f) > 0 \\ 1000 & if \ (D_t - D_f) = 0 \end{cases} \qquad (3)$$

where D_t is the deflection of plate at a state s_t, D_{max} the maximum deflection, D_f is the global optimized deflection, and sf is the shaping factor. The shaping factor can be of any real number. Results with various shaping factors are explored in this research. As the RL agent get closer to the minimum deflection D_f, the reward it gets increases. To speed up the learning and convergence (policy and value function), sparse reward is also used when the agent reaches the global optimal location, when $(D_t - D_f) = 0$. This sparse reward will trickle down the effect in the form of updated Q value.

The Q-learning approach is formulated like the greedy approach. The agent starts randomly from any edge (400 options) and proceeds towards the final state (global optimal location).

4. Results and discussion

For the greedy algorithm, in some of the cases, the greedy agent needed to overcome the local optimum location by exploring 20 steps further (in the future). If the minimum deflection is known, it becomes easy for the greedy agent to explore the future steps until it comes out of the sink and proceeds towards the global optimum location. Over 1,000 episodes, 318 times the greedy agent looked 20 steps ahead to overcome the local optimum sink, as shown in Figure 8.

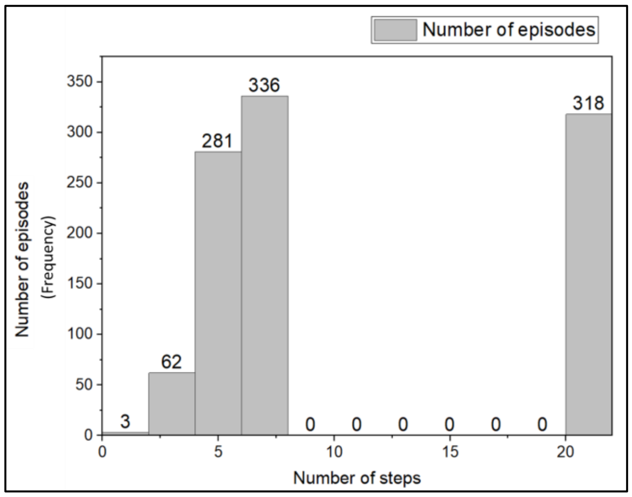

Figure 8. Frequency of number of steps over 1,000 episodes

Q-learning is free of such drawbacks; however, the main problem with Q-learning is that it requires state exploration at the start and refines the Q-learning table to proceed with exploitation. With each episode, the Q value for the states involved in the episode are updated towards reward and policy refinement. Due to its exploration behavior at the start, the RL agent requires more time to reach the global optimal location. The greedy agent takes approximately 6.1 hours (average over 1,000 episodes) to reach the global optimal location, whereas, the RL agent needed approximately 30 hours to reach the final state in first episode. That was due to the exploration, instead of exploitation. As the RL agent moves from exploration to exploitation, the RL agent outperforms the greedy algorithm. Figure 9 shows how quickly the RL agent overcomes the greedy algorithm in searching and reaching the final state. The RL agent learns the path, refines it, and improves with each episode.

The outcome corresponding to various shaping factors is also shown for comparison purposes in Figure 9. The results show that the RL agent with a shaping factor of 0.5 was slow in finding the final state. The agent became efficient with a shaping factor of 1.5, 2, 2.5, or 3. A shaping factor above 1 was effective in ramping up the RL agent's learning process in this case.

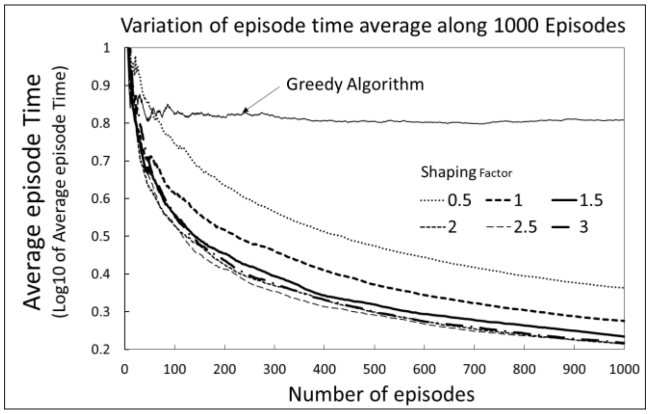

Figure 9. Average episode time with various shaping factors

Another important aspect to observe was the number of actions taken by the RL agent to reach the final state (as shown in Figure 10). The RL agent with a shaping factor of 1.5, 2, and 2.5 initiated fewer actions per episode to reach the final state.

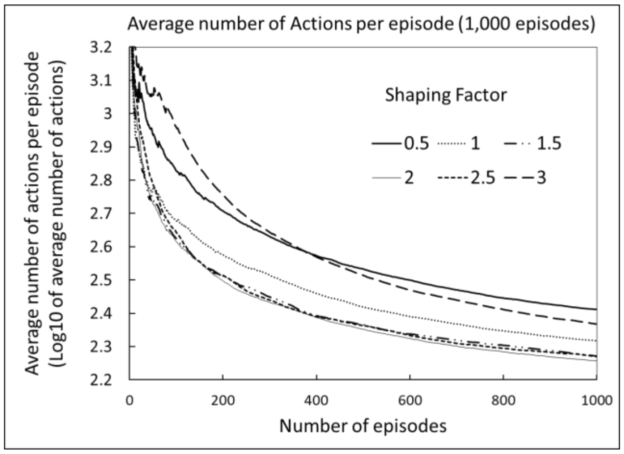

Figure 10. Average number of actions per episode

Based on the data obtained for the case example used in the present study, the question arises as to which shaping factor was most effective for the RL agent to learn quickly, diminish the time required by shortening the path towards the final state, and to maximize the reward over episode. To that end, the product of average time and average actions per episode can be utilized to define the selection criteria. Figure 11 shows that the value is minimum for a shaping factor of 2. This means that for the current case study, a shaping factor of 2 maximizes the learning process.

Moreover, an additional sensitivity analysis could include the variation of states explored over 100 episodes, which can also provide the ranking for these shaping factors. Figure 12 shows that the RL agent with a shaping factor of 2 completed 100 episodes with just 89% states explored.

The RL agent needs to learn how to decrease the average amount of time for each episode, which is the outcome of the policy of the RL agent to maximize the cumulative reward along each episode. The RL agent refines the policy and value function after each episode. The best example can be seen in this case, where the policy maximizes the reward by minimizing the path towards the final state along the number of episodes.

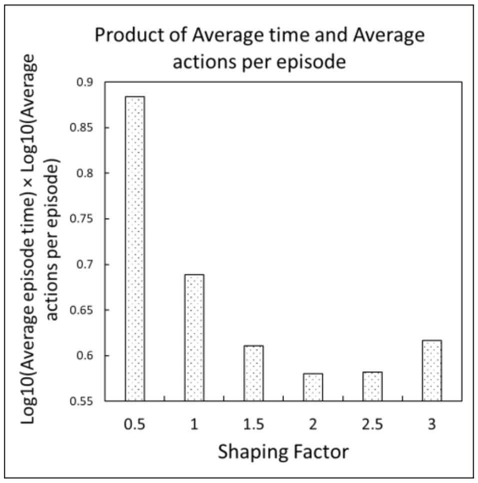

Figure 11. Product of average time and average actions per episode

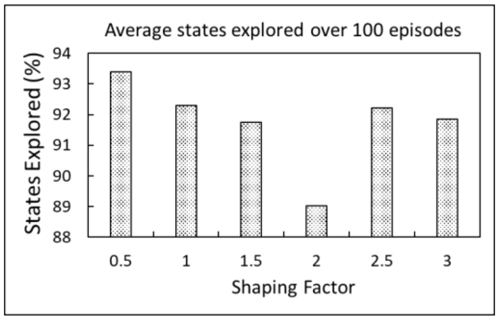

Figure 12. Average states explored over 100 episodes

Figure 13 shows the trendlines for episode time and reward over 100 episodes for the RL agent (with a shaping factor of 2). The trendlines converge towards the optimum for episode time and each episode's cumulative reward (value function).

The RL agent cannot surpass the greedy algorithm approach for locating the global optimal point on the first try (first episode). Nevertheless, the greedy agent cannot overcome the local optimal location, until it knows to search the steps further away from the local optimal location. The greedy agent becomes stable after overcoming the local optimal location. However, for the RL agent, there is no dilemma of local optimal location. It is important to mention that there are 400 starting points for greedy algorithm and RL agent. The greedy algorithm, starting from stochastic starting point for each iteration (after overcoming the local optimal location), takes a uniform cumulative average time to find the global optimal location. However, for RL agent, the cumulative average time decreases along the progression of episodes. After exploring the states, the RL agent refines itself with each episode and reaches the optimal location quicker compared to the greedy algorithm.

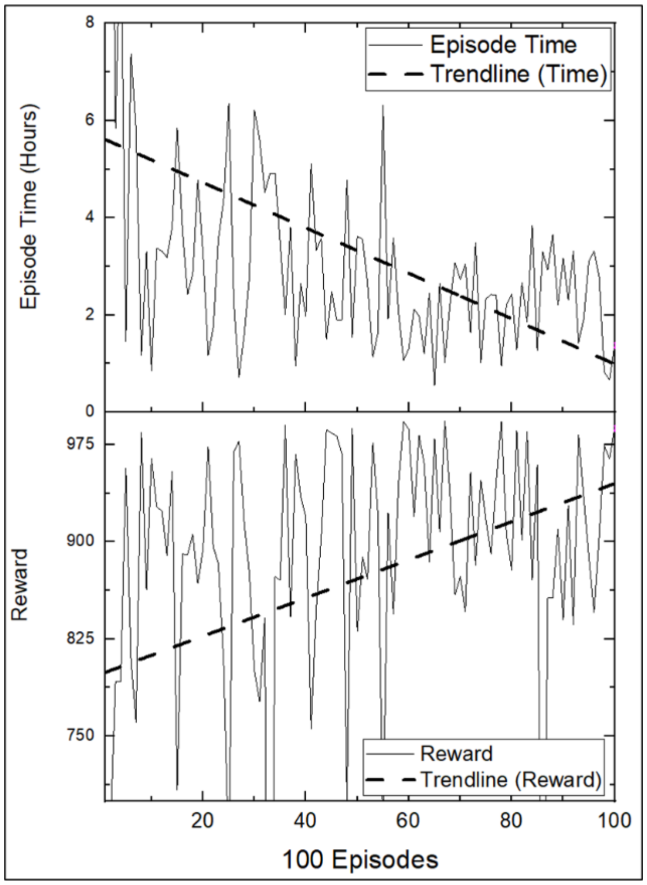

Figure 13. Convergence of policy and value function along 100 episodes (shaping factor 2)

5. Conclusions

The application of RL in the construction industry is new and has the potential to refine a design in a short time. In general, researchers are exploring RL to minimize the amount of time required for machine design. One example is the use of RL for the design of a rocket engine using ANSYS platform. The traditional procedure consists of trial and error to fine tune the parameters for the rocket engine design. To minimize this development time, the researcher used RL with the integration of FEA. A basic fluid dynamic problem was developed with diverse parameters as input and output. The input values were manipulated to obtain the desired output values, following the norms of RL (Mehr, 2019).

Following in those same footsteps, the authors of the present study believe that RL can be used to minimize the amount of time required for the extensive trial and error and fine tuning of parameters involved in the designing of a crane ground support mat. The parameters used for the multi-objective optimization problem will be the crane ground bearing pressure, soil composition, soil capacity, design parameters and mat configuration. The mat configuration and design parameters consist of placement of beam, type of beam, plate thickness, etc. It is important to note that other parameters (transportation, fabrication constraint, lifting constraints, etc.) can be added later based on the available information.

In addition to crane mat design, the intention is to develop a novel generic norm for the use of RL in machine design. Machine design is a complex problem, where many factors impact the design. Due to this complexity, the procedure for machine design takes a relatively long time from concept to the prototype. The application of FEA in machine design was a great help, but at the same time, the processing time for FEA increases exponentially ("curse of dimensionality") with the addition of more design parameters. The back and forth from manual design to FEA eats up a large portion of the design engineer's time. This time could be saved by developing an algorithm that can mimic the behavior of a design engineer.

Besides machine design, resource allocation is another optimization problem that requires a lot of an engineer's time to finalize. RL can be helpful to shorten this time. One of the applications could be the optimization of crane mat layout on the construction site. There could be multiple combinations for a set of mats. The use of RL can refine the mat layout with maximum area covered and minimum mat usage. This could be a classic example of RL usage for optimization problems.

References

Bang-Jensen, J., Gutin, G., & Yeo, A. (2004). When the greedy algorithm fails. *Discrete Optimization*, 1(2), 121–127. https://doi.org/10.1016/j.disopt.2004.03.007

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). Cambridge Mass: MIT Press

Doran, J., & Michie, D. (1966). Experiments with the Graph Traverser Program. *Proceedings of The Royal Society A: Mathematical, Physical and Engineering Sciences, 294*, 235–259. 10.1098/rspa.1966.0205

Gullapalli, V., & Barto, A. G. (1992). Shaping as a method for accelerating reinforcement learning. *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, 554–559. 10.1109/ISIC.1992.225046

Gutin, G., Yeo, A., & Zverovich, A. (2002). Traveling Salesman Should not be Greedy: Domination Analysis of Greedy-Type Heuristics for the TSP. *Discrete Applied Mathematics*, 117, 81–86. 10.1016/S0166-218X(01)00195-0

Hilgard, E. R., Marquis, D. G., & Kimble, G. A. (1961). *Hilgard and Marquis' Conditioning and Learning*. East Norwalk, CT, US: Appleton-Century-Crofts, Inc.

Kan, C., Zhang, P., Fang, Y., Anumba, C., & Messner, J. (2018). A Taxonomic Analysis of Mobile Crane Fatalities for CPS-based Simulation. *17th International Conference on Computing in Civil and Building Engineering*. Retrieved from http://programme.exordo.com/icccbe2018/delegates/presentation/253/

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., & Wierstra, D. (2015). Continuous control with deep reinforcement learning. *ArXiv Preprint ArXiv:1509.02971*. Retrieved from https://arxiv.org/abs/1509.02971

Mehr, E. (2019, April 9). *Using Reinforcement Learning to Design a Better Rocket Engine*. Retrieved from https://blog.insightdatascience.com/using-reinforcement-learning-to-design-a-better-rocket-engine-4dfd1770497a

Richard Bellman. (1957). A Markovian Decision Process. *Journal of Mathematics and Mechanics*, 6(4), 679–684. Retrieved from www.jstor.org/stable/24900506

Sutton, R. S. (1988). Learning to Predict by the Method of Temporal Differences. *Machine Learning*, 3, 9–44. 10.1007/BF00115009

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement Learning: An Introduction* (2nd ed.). Cambridge, Mass: MIT press. Retrieved from http://incompleteideas.net/book/RLbook2020.pdf

Watkins, C. (1989). *Learning from Delayed Rewards* [Doctoral dissertation, King's College, Cambridge, UK]. University of Cambridge. http://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf

Watkins, C., & Dayan, P. (1992). Technical Note: Q-Learning. *Machine Learning*, 8, 279–292. 10.1007/BF00992698

Zhao, Q. (2011). *Cause Analysis of U.S. Crane-related Accidents* [Master Dissertation, University of Florida, USA]. UF Theses & Dissertations. http://ufdc.ufl.edu/UFE0042972/00001