Hardy-Ramanujan Journal 43 (2020), 91-98 submitted 20/12/2020, accepted 10/03/2021, revised 10/03/2021

A Reinforcement Learning Based Algorithm to Find a Triangular Graham Partition

Byungchan Kim

In memory of Ramanujan on occasion of the 100th anniversary of his death

Abstract. We introduce an algorithm to find a partition of n of the form

$$n = T_{k_1} + T_{k_2} + \cdots + T_{k_m} \quad \text{and} \quad 1 = \frac{1}{k_1} + \frac{1}{k_2} + \cdots + \frac{1}{k_m},$$

where $T_k = \frac{k(k+1)}{2}$ is the k-th triangular number. The algorithm is based on a reinforcement learning algorithm (TD3) and a genetic algorithm.

Keywords. Graham partition, triangular number, reinforcement learning, genetic algorithm

2010 Mathematics Subject Classification. 11P81 and 05A17

1. Introduction

Three striking integer partition congruences

$$p(5n+4) \equiv 0 \pmod 5, \ p(7n+5) \equiv 0 \pmod 7, \ p(11n+6) \equiv 0 \pmod{11},$$

and the asymptotic formula on the number of integer partitions

$$p(n) \sim \frac{1}{4n\sqrt{3}} e^{\pi\sqrt{\frac{2n}{3}}}$$

are probably two most well-known results among Ramanujan's thousands of thousands results, where $p(n)$ is the number of integer partitions of n [An98]. Ramanujan got an inspiration by examining the values of $p(n)$ up to $n = 200$, which P.A. MacMahon calculated.

The goal of this note is to introduce an algorithm to find a partition of a special form and to give a conjecture regarding the existence of such partitions. Using the algorithm, we make a table of such partitions. We hope this table help researchers get insights as Ramanujan found surprising patterns from MacMahon's table on partition function values. We say that $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_\ell)$ is an integer partition of a positive integer n if $\sum_{i=1}^{\ell} \lambda_i = n$, where a different order of λ_i's is considered the same partition. Here, we call λ_i a part of the partition λ. As far as the author knows, Graham [Gr63] first studied the partition with a specific condition on the reciprocal sum of parts. He showed that if $n \geq 78$, then there is a partition of n into distinct parts such that the reciprocal sum of the parts is 1. The authors in [KKLLP] call a partition λ a *Graham partition* if the reciprocal sum of its parts is 1. For example, 78 has a Graham partition with distinct parts:

$$78 = 2 + 6 + 8 + 10 + 12 + 40 \quad \text{and} \quad 1 = \frac{1}{2} + \frac{1}{6} + \frac{1}{8} + \frac{1}{10} + \frac{1}{12} + \frac{1}{40}.$$

The author was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science and ICT (NRF-2019R1F1A1043415).

In this note, we consider how to find a partition of n into triangular numbers $T_k = \frac{k(k+1)}{2}$ satisfying that

$$n = T_{k_1} + T_{k_2} + \cdots + T_{k_m} \quad \text{and} \quad 1 = \frac{1}{k_1} + \frac{1}{k_2} + \cdots + \frac{1}{k_m}.$$

We call a partition $(T_{k_1}, T_{k_2}, \ldots, T_{k_m})$ a triangular Graham partition if it satisfies the above condition. For example, since

$$18 = 6 + 6 + 6 \quad \text{and} \quad 1 = \frac{1}{3} + \frac{1}{3} + \frac{1}{3},$$

$(6, 6, 6)$ is a triangular Graham partition of 18. Graham conjectured that there is a triangular Graham partition of n for sufficiently large n. Actually, Graham [Gr63] made much stronger conjectures in which triangular numbers can be replaced by any polygonal numbers. For the square numbers case, Alekseyev [Al19] showed that if $n \geq 8543$, then there is a partition of n into distinct squares such that

$$n = k_1^2 + k_2^2 + \cdots + k_m^2 \quad \text{and} \quad 1 = \frac{1}{k_1} + \frac{1}{k_2} + \cdots + \frac{1}{k_m}.$$

For example, there is such a partition of 49 as

$$49 = 2^2 + 3^2 + 6^2 \quad \text{and} \quad 1 = \frac{1}{2} + \frac{1}{3} + \frac{1}{6}.$$

This is the only progress toward Graham's conjecture during the last few decades. In both proofs for Graham's result [Gr63] and for Alekseyev's result [Al19], the key idea is using the induction via recurrence relations among partitions. For example, if $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_\ell)$ is a Graham partition of n, then $(2, 2\lambda_1, 2\lambda_2, \ldots, 2\lambda_\ell)$ is also a Graham partition of $2n + 2$. If the parts are squares, then one can find a recurrence from the fact that four times of a square is again a square. For triangular numbers (and other polygonal numbers beside squares), this trick does not work anymore. Moreover, since the number of partitions of n into triangular numbers grows quickly, an exhaustive search for triangular Graham partitions of an integer n eventually becomes infeasible.

A common approach in the theory of partitions is finding a generating function. Let $G_t(n)$ be the number of triangular Graham partitions of n. Then, we see that

$$\begin{aligned}\sum_{n \geq 0} G_t(n) q^n &= [z] \prod_{n=1}^{\infty} \frac{1}{1 - z^{1/n} q^{n(n+1)/2}} \\ &= q + q^6 + q^{18} + q^{23} + q^{30} + q^{40} + q^{47} + q^{54} + q^{66} + q^{75} + \cdots,\end{aligned}$$

where $[z]f(z, q)$ means the coefficient of z for the power series $f(z, q) \in \mathbb{Z}[[z, q]]$. This generating function is useful for first few hundreds of n, but it is not effective to find $G_t(n)$ for a sufficiently large n.

Here we take an alternative approach to find a triangular Graham partition, which is based on reinforcement learning (RL). RL is a branch of machine learning (see Sutton and Barto's excellent guide [SuBa18] for details), which has recently achieved many remarkable successes, beating Go world champions (Silver et. al. [Si16]), human level video game plays (Mnih et. al. [Mn15]), playing a team play game (OpenAI [OpenAI]) to name few. Our algorithm is inspired by AlphaGo[1][Si16] in the following sense. To train a policy network, we use data from an exhaustive search like that AlphaGo learns at first from human experts' moves. Beside pre-generated data, the policy network learns from experiences like that AlphaGo learns a better policy via self-plays. Most importantly, the trained policy itself is not sufficient to find a desired partition, thus we employ another selection method, a variant of the genetic algorithm. This is like that AlphaGo uses a Monte Carlo Tree Search to find a better move in the game of Go. While there are at most 361 moves available in the game of Go, when we find a partition of a given integer n, the number of possible parts depends on n. At least

[1]Not AlphaGo Zero[Si17], which do not use human generated data.

theoretically, there is no upper bound on n, which means that the number of possible actions (choosing an appropriate part) could be very large. Therefore, we employ a continuous control algorithm, twin delayed deep deterministic policy gradient (TD3) algorithm [FHM18], which achieved the state-of-arts performances in many simulated robot control tasks.

Using this algorithm, we try to find a triangular Graham partition of n from $n = 500$ to $n = 1004$.[2] Interested readers can find the list of triangular Graham partitions at the below.

https://github.com/math-bkim/tri_GP_finder/blob/master/Tri_GP_Table.csv

These examples support the following effective version of Graham's conjecture.

Conjecture 1. *For all integers $n \geq 645$, $G_t(n) > 0$.*

Remark 1.1. *Using an exhaustive search, we check that there is no triangular Graham partitions of n if $n \in \{500, 503, 518, 529, 570, 589, 644\}$. During the search, we have used the single agent and the same hyper-parameters (see Section 2). Our algorithm fails to find the following triangular Graham partitions.*

$$\begin{aligned}
565 &= 210 + 210 + 55 + 55 + 15 + 10 + 10,\\
653 &= 300 + 55 + 55 + 55 + 55 + 55 + 36 + 21 + 21,\\
685 &= 120 + 78 + 78 + 78 + 78 + 78 + 78 + 55 + 21 + 21,\\
685 &= 210 + 210 + 210 + 15 + 15 + 15 + 10,\\
719 &= 231 + 120 + 120 + 78 + 78 + 28 + 28 + 21 + 15,\\
774 &= 78 + 78 + 78 + 78 + 78 + 78 + 78 + 78 + 78 + 36 + 36,\\
774 &= 210 + 210 + 210 + 78 + 21 + 15 + 15 + 15,\\
774 &= 465 + 120 + 120 + 21 + 21 + 21 + 6.
\end{aligned}$$

It seems that our algorithm is struggling if a desired partition has many repeated parts.

As a fan of Ramanujan, we should mention that our algorithm can find a triangular Graham partition of 1729:

$$1729 = 1176 + 210 + 136 + 78 + 36 + 36 + 21 + 21 + 15,$$
$$1 = \frac{1}{48} + \frac{1}{20} + \frac{1}{16} + \frac{1}{12} + \frac{1}{8} + \frac{1}{8} + \frac{1}{6} + \frac{1}{6} + \frac{1}{5}.$$

The rest of the note is organized as follows. In Section 2, we introduce basic concepts of reinforcement learning and sketch briefly how our algorithm works. In Section 3, we remark characteristics of our algorithm and propose mathematical questions derived from our search.

2. Algorithm

In this section, we explain the structure of the policy network, the training method, and a variant of the genetic algorithm.

2.A. Reinforcement Learning

RL is a branch of machine learning in which the agent learns how to act to achieve a task goal while interacting with the environment. Mathematically, this learning process is modelled by a Markov Decision Process (MDP) consisting of the set of states, $\mathcal{S}$, the set of actions, $\mathcal{A}$, the reward signal $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, the transition probability $p(s_{t+1} = s' | s_t = s, a_t = a)$, and the discount factor $\gamma \in [0, 1]$. Roughly speaking, the goal of the agent is to find an optimal policy $\pi : \mathcal{S} \to \mathcal{A}$ to maximize the expected return $\mathbb{E}[\sum_{i=1}^{\infty} \gamma^{i-1} r(s_i, a_i) | a_i = \pi(s_i)]$. There have been extensive studies how to find an optimal policy and here we use a policy gradient method as our state-action space is quite huge.

[2]1004 is pronounced same with "angel" in Korean. We hope readers get intuition from an angel like Ramanujan dreamed the goddess Namakkal.

2.B. Policy network

To limit the number of possible actions reasonably, we set the state as two-dimensional (w_1, w_2), where $w_1 \in \mathbb{Q}$ represents the distance toward 0 and w_2 represents the sum of reciprocals. If we want to find a triangular Graham partition of n, we set the initial state $(n/100, 0)$. At the state (w_1, w_2), the policy selects a triangular number less than or equal to $100w_1$. If the policy selects T_k, then the new state is $(w_1 - T_k/100, w_2 + 1/k)$. The policy repeats this step until $w_1 < \frac{3}{100}$ or $w_2 \geq 1$. For example, if we want the policy finds a triangular Graham partition of 18, then the first state is $(\frac{18}{100}, 0)$. Our policy selects a triangular number which is most likely to being a part of a triangular Graham partition according to the policy's evaluation. Say, the policy selects 6. Then, the next state becomes $(\frac{18-6}{100}, 0 + \frac{1}{3})$. Once the policy receives a new state, then it selects another triangular number. If it selects 6 again, then the next state is $(\frac{18-6-6}{100}, 0 + \frac{1}{3} + \frac{1}{3})$. If the policy selects a triangular number 6 at this state, then the policy succeeds to find a Graham partition. Otherwise, it goes to a new state.

Now we explain how the policy network chooses a triangular number. To make the output of the network is in the same range regardless of the size of s_1, we choose tanh as the output activation function, so the image of π is between -1 and 1. We partition the interval $[-1, 1]$ uniformly into the number of triangular numbers between 3 and $100w_1$. If the value of tanh is in the i-th interval, then the policy chooses $i + 1$-th triangular number. This is because we do not want to select the first triangular number $T_1 = 1$, which never appears in a triangular Graham partition of $n > 1$.

To reflect the above considerations, we construct the policy network as follows. The input dimension is 2 and it has two hidden layers with 300 neurones with rectified linear unit activation functions, and the output layer is 1-dimensional with the hyperbolic tangent activation function.

2.C. Training Method

To train the policy network, we use the TD3 algorithm. To this end, we need to define the reward signal. Let (w_1, w_2) be the input state and (w_1^*, w_2^*) be the output state. Recall that if $\pi(w_1, w_2) = T_k$, then $w_1^* = w_1 - T_k/100$ and $w_2^* = w_2 + \frac{1}{k}$. The reward signal r is defined by

$$r(w_1^*, w_2^*) = \frac{0.5}{1 + |w_1^*|} + \frac{0.5}{1 + |1 - w_2^*|} + 0.5\chi(w_1^* = 0) + 2\chi(w_2^* = 1),$$

where $\chi(T) = 1$ if the statement T is true, 0 otherwise.

Let D be the set of triangular Graham partitions of n up to $n = 450$, which we can find by an exhaustive search. During the training, we set the initial state either a random portion of an element in D with the probability 0.1 or $(n/100, 0)$ for a random integer n between $n = 100$ and $n = 700$ with the probability 0.9. Other hyper-parameters for the TD3 algorithm are as follows.

- the number of training episodes : 2×10^5
- the size of the replay buffer : 10^5 (steps)
- the size of a mini batch : 100
- training starts from the time step : 10^3
- γ (a discount factor) : 0.99
- τ (a soft update rate for the target network) : 0.01
- the exploration noise factor : 0.3
- the policy noise factor : 0.2
- the noice clip factor : 0.5
- the policy update frequency : 3
- the actor learning rate : 5×10^{-4}
- the critic learning rate : 1×10^{-3}

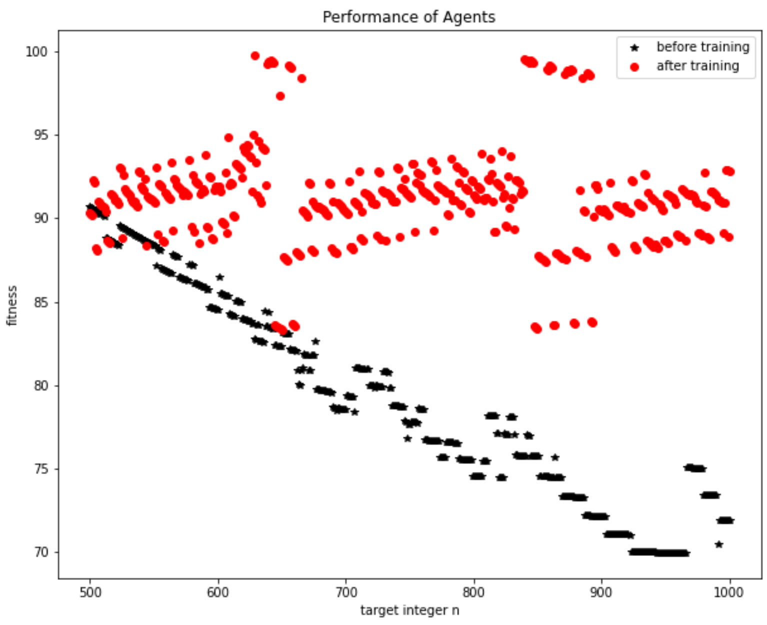

Figure 1: Fitness of partitions generated by the policy.

We use PyTorch for the training process, and we use a code based on the TD3 paper author's code [FHM].

Since there are too many partitions of n into triangular numbers, It is hard for the agent to get an experience which succeeds to find a triangular Graham partitions. For example, there are $3,977,459,451$ partitions of 590 into triangular numbers. Among these partitions, there is only one triangular Graham partition. Therefore, the training number 2×10^6 is very small compared to the whole search space. Thus, it is not surprising that the performance of the policy network is far from satisfactory. Figure 1 shows the performance of the policy network for target integers n from $n = 500$ to $n = 1000$.

In Figure 1, the fitness of the partition $\lambda = (T_{k_1}, T_{k_2}, \dots, T_{k_m})$ is defined by

$$\text{fitness}(\lambda) = \frac{50n}{n + |n - \sum_{i=1}^{m} T_{k_i}|} + \frac{50}{1 + \left|1 - \sum_{i=1}^{m} \frac{1}{k_i}\right|}, \tag{2.1}$$

where n is a desired weight. If the partition λ is a triangular Graham partition of n, then fitness(λ) is 100. We observe that the fitness becomes larger after the training even for un-trained cases, i.e. $n > 700$ cases. This shows that the agent has obtained some generality. However, the fitness are still far from the perfect. To overcome this issue, we use a genetic algorithm to improve the fitness of the partition generated by the policy network.

2.D. Genetic Algorithm

The Genetic Algorithm is a widely used global optimization algorithm inspired by the natural selection in the evolution. A population of chromes evolves via cross-overs and mutations and the chromes fit well in the environment get more chances to spread springs. In our case, the chromes are partitions generated by the policy network. As generations span, we desire one of these springs to become a triangular Graham partition. We use the policy network not only to generate the initial population

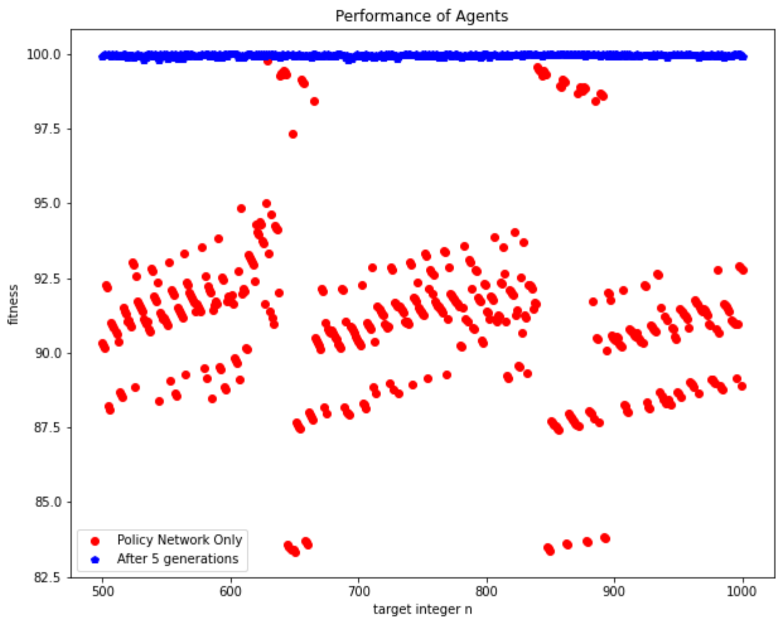

Figure 2: Fitness of partitions generated by the policy and the genetic algorithm after 5 generations.

of chromes, but also to cross-over two chromes (partitions) and to mutate the chromes (partitions). For a cross-over step, we select a length of parts from one of its parent, and use the policy network to get a new partition from the selected parts. The mutation step works similarly. We select a random portion of the partition and get a new partition using the policy network. Since we use the deterministic policy, we use an exploration rate. To encourage explorations, we increases the exploration rate and the mutation rate if we do not get a better partition for a given period of generations. Interested readers can find the code and running examples at the below.[3]

`https://github.com/math-bkim/tri_GP_finder/`

By incorporating a genetic algorithm, the predicted partition is more like a Graham partition. In Figure 2, we see that the resulting fitness after 5 generations are near 100. Since we require that the fitness is exactly 100, it usually takes 20 to 200 generations to get a desired partition. By the nature of a genetic algorithm, our algorithm is not deterministic and thus the required number of generations until we find a Graham partition varies for each trial.

3. Concluding remark

One nice feature of our algorithm is that we can set a list of parts which we desire to be contained in a Graham partition. If we give some parts, then the algorithm tries to find a Graham partition containing them. For the triangular partition case, there are very few Graham partitions of small weights, thus it is unlikely that there is a Graham partition with desired parts. However, for the ordinary partition case, there are relatively much more Graham partitions, so there is a higher chance

[3]In the actual application of a genetic algorithm, we put more weights on the condition for the reciprocal sum as it is harder to achieve. We use the same policy network and hyper-parameters during the search: the initial population 500 and the initial mutation rate 0.1.

to succeed. For example, the below[4] is a Graham partition of 1729 with that 1200 and 300 are given as initial parts.

$$\begin{aligned}1729 &= 1200 + 300 + 90 + 54 + 30 + 27 + 16 + 6 + 3 + 3\\ 1 &= \frac{1}{1200} + \frac{1}{300} + \frac{1}{90} + \frac{1}{54} + \frac{1}{30} + \frac{1}{27} + \frac{1}{16} + \frac{1}{6} + \frac{1}{3} + \frac{1}{3}.\end{aligned}$$

We also remark that we may find a different partition if we equip with a different policy.[5] For example, the below are triangular Graham partitions of 1729 from other policy networks:

$$\begin{aligned}1729 &= 300 + 300 + 300 + 171 + 136 + 136 + 120 + 78 + 78 + 55 + 45 + 10,\\ 1729 &= 1176 + 136 + 105 + 78 + 78 + 36 + 36 + 28 + 28 + 28.\end{aligned}$$

Though our algorithm has its own interests and is able to find a desired partition for a wide range of integers, it is surely far from the best effective algorithm. In particular, the algorithm could be much slower than an exhaustive search for small integers. We hope one can find a better algorithm whether it is machine learning based or not.

It would be very interesting if one can find a one-variable generating function for $G_t(n)$ and if one can use the one variable generating function to determine how fast $G_t(n)$ grows. In fact, we do not know the one-variable generating function for the number of Graham partitions of n, say $G(n)$,

$$\begin{aligned}\sum_{n\geq 0} G(n)q^n &= [z]\prod_{n=1}^{\infty}\frac{1}{1-z^{1/n}q^n}\\ &= q + q^4 + q^9 + q^{10} + q^{11} + q^{16} + q^{17} + q^{18} + q^{20} + \cdots .\end{aligned}$$

By employing the inductive method we explain in the introduction, it is not hard to see that $G(n) > 0$ for all integers $n > 23$. However, it is still desired to find a better expression for the generating function and to investigate how fast $G(n)$ grows.

Acknowledgement. The author appreciates the anonymous referee for his or her detailed comments on an earlier version of the paper.

References

[Al19] M.A. Alekseyev, *On partitions into squares of distinct integers whose reciprocals sum to 1*, The mathematics of various entertaining subjects. Vol. 3, 213–221, Princeton Univ. Press, Princeton, NJ, 2019.

[An98] G.E. Andrews,*The theory of partitions*, Cambridge Mathematical Library, Cambridge University Press, Cambridge, 1998.

[FHM18] Scott Fujimoto, Herke Hoof, David Meger , *Addressing Function Approximation Error in Actor-Critic Methods*, Proceedings of the 35th International Conference on Machine Learning, PMLR 80 (2018), 1587–1596.

[FHM] Scott Fujimoto, Herke Hoof, David Meger , *Author's code for Addressing Function Approximation Error in Actor-Critic Methods*, `https://github.com/sfujim/TD3`.

[Gr63] R. L. Graham, *A theorem on partitions*, Journal of the Australian Mathematical Society, **3** (1963), 435–441.

[KKLLP] B. Kim, J.-Y. Kim, C.-G. Lee, S.-J. Lee, and P.-S. Park, *On the existence of Graham partitions with congruence conditions*, preprint.

[Mn15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Rusu, J. Veness, M. Bellemare, A. Graves, M. Riedmiller, A. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, *Human-level control through deep reinforcement learning*, Nature **518** (2015), 529–533.

[4]Here we use another policy network trained separately as we do not need to choose a triangular number

[5]Since a genetic algorithm is not deterministic, we may find a different partition even if we use the same policy and hyper-parameters.

[OpenAI] OpenAI: Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, Susan Zhang, *Dota 2 with Large Scale Deep Reinforcement Learning*, `arXiv:1912.06680`.

[Si16] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, *Mastering the Game of Go with Deep Neural Networks and Tree Search*, Nature **529** (2016), 484–489.

[Si17] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui, L. Sifre, G. van den Driessche, T. Graepel, and D. Hassabis, *Mastering the game of Go without human knowledge*, Nature **550** (2017), 354–359.

[SuBa18] R. Sutton and A. Barto, *Reinforcement Learning: An Introduction*, The MIT Press, Second edition, 2018.

Byungchan Kim
School of Liberal Arts
Seoul National University of Science and Technology
232 Gongneung-ro, Nowon-gu, Seoul, 01811
Republic of Korea
e-mail: bkim4@seoultech.ac.kr