# Towards automatic generation of congestion control algorithms by coevolving the environment

Teruto Endo[1], Hirotake Abe[1] and Mizuki Oka[1]

[1]University of Tsukuba, Tsukuba, Ibaraki 305-8577 Japan
enteru_2020@websci.cs.tsukuba.ac.jp

**Abstract**

We discuss the applicability of a co-evolutionary algorithm of environments and agents for the automatic generation of network congestion control algorithms. To co-evolve the network simulation as an environment and the network congestion control algorithm as an agent, we investigated methods for controlling the difficulty of an environment and for generating and optimizing congestion control algorithms, which are necessary for co-evolution. First, we examined the possibility of controlling the difficulty level by varying the amount of cross-traffic generation and the rate of packet loss, which causes throughput degradation. Next, we tested the feasibility of using grammatical evolution for the automatic generation and optimization of congestion control algorithms. The results of these experiments demonstrated that the difficulty of the environment can be controlled by varying the amount of cross-traffic generation and the rate of packet loss. Additionally, it was confirmed that grammatical evolution can be used to optimize the network congestion control algorithm for environments with different parameter settings. In particular, the network congestion control algorithm that we obtained in an environment with high packet loss rates worked robustly in other environments. We show that a co-evolutionary algorithm of environments and agents can be used for the network congestion control algorithm.

## Introduction

Recently, a reinforcement learning algorithm called paired open-ended trailblazer (POET), which co-evolves agents and environments, has garnered a significant amount of research attention (Wang et al., 2019). POET starts learning through an agent paired with an environment, that are generated by environmental mutation. The agent evolves and optimizes itself for the environment it is paired. The evaluation function of the environment is made neither too difficult nor too easy for the agents. Agents with high adaptability are transferred to other environments where they evolve into new pairs.

Thus, by using an algorithm such as POET, which co-evolves agents and environments, we can automatically generate a wide variety of environments and acquire different strategies for agents. An example of the application of POET is the study of the co-evolution of game levels and player characters(Dharna et al., 2020). However, there are still few examples of real-world applications of POET-like algorithms. In this study, we examine network congestion control as a potential application of the co-evolutionary algorithm of agents and environments.

Network congestion is the temporary concentration of a large amount of data on an Internet connection. Generally on the Internet, users behave in their own self-interest to maximize their communication, and several users constantly compete for the communication bandwidth of shared resources. Hence, a large amount of data temporarily flows across the Internet, causing network congestion. Congestion causes packet loss and delays, which can slow down the transmission speed to the service and cause connection problems. As congestion can cause disruptions in the use of the Internet, it is necessary to take measures for its prevention. To prevent congestion, network congestion control algorithms are used to adjust the data transmission rate.

Congestion control algorithms decrease the amount of packet transmission when congestion is detected and increase the amount of transmission otherwise. Because it is not possible to directly observe Internet connection conditions, a congestion control algorithm uses the information received by each terminal during communication to verify the occurrence of congestion.

Network congestion control algorithms can be broadly classified into loss-based and delay-based methods, depending on the method used to determine congestion(Al-Saadi et al., 2019). Loss-based methods determine whether congestion occurs based on packet loss and decide whether to increase or decrease the amount of transmission. In contrast, delay-based methods determine whether congestion occurs based on the round-trip time of packets and then decide whether to increase or decrease the transmission volume.

Research on network congestion control algorithms has been ongoing for more than 40 years. However, a definitive congestion control algorithm is yet to be developed. This is due to the evolving Internet usage and connection patterns. Additionally, existing network congestion control algorithms cannot adapt to changes in the network environ-

ment (bandwidth, delay, packet loss, network topology, etc.) because network congestion control algorithms are built using a rule-based approach. To overcome the limitations of rule-based algorithms, a reinforcement learning-based algorithm has recently been proposed(Jay et al., 2019; Li et al., 2019). Attempts to use learning-algorithms and network simulation environments in algorithms such as POET, which co-evolve agents with environments, have the potential to study congestion control algorithms more effectively.

Specifically, the co-evolution of the environment and the agent may be used to discover new and efficient congestion control algorithms. The conventional development of network congestion control algorithms and the setting up of network simulation environments have been based on the experience of researchers and the findings of their observations of the Internet. Therefore, it cannot be denied that there is a possibility of researcher bias in the algorithm development process. Additionally, there is a limit to the number of algorithms that can be constructed heuristically by researchers. Thus, it is believed that more effective network congestion control algorithms that are yet to be discovered. Therefore, it is expected that algorithms, such as POET, can be used to automatically discover congestion control algorithms that have not been discovered owing to bias or other constraints.

However, to apply environment-agent co-evolutionary algorithms to network congestion control, it is necessary to investigate methods to control the degree of difficulty in the environment and automatically generate and optimize the network congestion control algorithm. In this study, we address these two challenges and experimentally demonstrate that it is possible to use network congestion control algorithms as an application of environment-agent co-evolutionary algorithms. This work contributes to the study of the open-ended evolution category "Interesting new kinds of entities and interactions"(Packard et al., 2019) in terms of generating new network congestion control algorithms and network simulation environments through coevolution.

## Environments and Agents in Network Congestion Control

The environments and agents in our target application are network simulation and congestion control algorithms, respectively. In this study, we used a network simulator called ns3gym(Gawłowicz and Zubow, 2019). ns3gym is an environment that enables a network simulator called ns-3 (network simulator 3)[1] to be used with a reinforcement learning toolkit called OpenAI Gym (Brockman et al., 2016). ns3 is an open-source network simulator developed for research purposes that allows users to write network simulation scenarios and various parameter settings in C++ to perform simulations.

The evolution of algorithms requires the generation of programs that contain conditional branches. Therefore, we use grammatical evolution (O'Neill and Ryan, 2001), which guarantees that the generated programs are grammatically correct. Examples of applications of grammatical evolution are the construction of algorithms used for solving cutting and packing problems(Burke et al., 2012) and optimizing swarm behavior(Neupane and Goodrich, 2019). Additionally, besides programs and mathematical expressions, anything that has a grammatical structure can be evolved using grammatical evolution. For example, grammatical evolution is used in level generation for two-dimensional (2D) side-scrolling action games(Shaker et al., 2012) or for optimizing the structure of neural networks(Assunção et al., 2017). In this study, we used PonyGE2(Fenton et al., 2017) to perform grammatical evolution.

### Control the difficulty of the environment

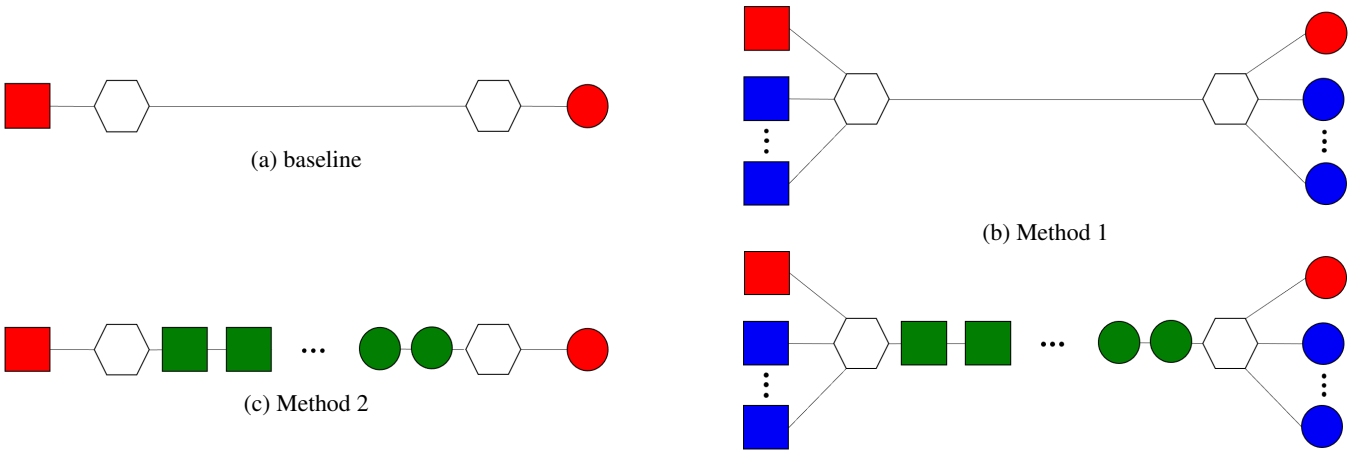
For the environment and the agent to co-evolve, the difficulty of the environment must be controllable in gradual steps. Therefore, we have to consider such parameters in the network simulator, where we use the throughput of the environment to define the difficulty level, with higher throughput corresponding to a lower difficulty level and vice-versa. We consider two types of parameters to control the throughput: the amount of cross-traffic generated and the packet loss rate.

Cross-traffic, which is the first parameter, interferes with the main communication. For example, in the case of a server and a client that are in the main communication, the traffic that arises because of the sending of data to the nodes on the communication path between the server and client is cross-traffic. Thus, when communication paths overlap, additional pressure is placed on the bandwidth. In such cases, if the communication data are not well controlled, congestion will occur and the throughput will be reduced. Therefore, it is possible to control the difficulty of the environment by varying the amount of cross-traffic generated.

Figure1 shows a conceptual diagram of the network topology in which the cross-traffic is generated. There are three types of nodes: servers, clients, and relay nodes. The relay node only relays data between the main traffic and the cross-traffic and does not generate traffic. In the figure, the red node indicates the node that generates the main traffic, and blue and green nodes indicate nodes that generate cross-traffic. The server at which the throughput is measured is called the main server. The client that sends data to the main server is called the main client.

Figure1-(a) shows a baseline topology of the network, which comprises only one pair of servers that transmits the main traffic, client nodes, and two relay nodes. Figures 1-(b) and (c) show two methods of setting up a topology to generate cross-traffic. In the former method (method 1), a given number of server-client pairs is connected directly to both ends of the relay nodes(shown in blue in the figure).

[1] https://www.nsnam.org/



(a) baseline

(b) Method 1

(c) Method 2

(d) Method 1 and method 2 are used together.

Figure 1: Network topology in which the cross-traffic is generated. Squares indicate servers, circles indicate clients, and hexagons indicate relay nodes. Red indicates nodes that generate main traffic, blue indicates nodes that directly connect to relay nodes at both ends and generate cross-traffic, and green indicates nodes that connect relay nodes and generate cross-traffic. (a) shows the baseline, and (b) illustrates method 1 in which server-node pairs are added directly to both ends of the relay nodes. (c) illustrates method 2 in which server-node pairs are added between the relay nodes, and (d) illustrates the scenario when method 1 and method 2 are used together.

In the later method (method 2), a given number of server-client pairs is connected between the relay nodes (shown in green in the figure). Methods 1 and 2 can be used together, as shown in Figure1-(d).

A network topology with cross-traffic was automatically created, based on the number of server-client pairs, using our network generator. The network topology information generated by the network generator was loaded into the network simulator, and subsequently, simulations were performed.

When packet loss, which is the second parameter, occurs, it is necessary to resend and receive data. Consequently, throughput decreases as re-transmitting the packets takes time. The packet loss can be attributed to disconnection in the case of wired networks and radio interference in the case of wireless networks. This is especially likely to occur in wireless applications. Therefore, the packet loss rate is considered an effective parameter for controlling the difficulty of the environment. Packet loss during the experiment causes packet loss at all nodes in the network topology generated by the network generator.

## Classical congestion control algorithms

To design an automatically generated network congestion control algorithm, we refer to a relatively simple but widely used classical network congestion control algorithm called the additive increase and multiplicative decrease (AIMD) algorithm(Peterson and Davie, 2011). The transmission rate is determined using Equation 1.

$$x(t+1) = \begin{cases} x(t) + a & \text{(congestion is not detected)} \\ x(t) \times b & \text{(congestion is detected)} \end{cases} \tag{1}$$

$x(t)$ denotes the transmission rate at time $t$. When congestion is not detected, the transmission rate increases by $a$ $(a > 0)$; when congestion is detected, the transmission rate decreases by $b$ $(0 < b < 1)$. The transmission rate is determined by the determines the number of bytes that can be sent out at any time, called the congestion window size.

While a number of AIMD-based algorithms have been proposed, one of the most popular algorithms that use the AIMD method is NewReno(Gurtov et al., 2012). The equation for updating the transmission rate of NewReno is as follows:

$$x(t+1) = \begin{cases} x(t) + \frac{1}{x(t)} & \text{(congestion is not detected)} \\ \frac{x(t)}{2} & \text{(congestion is detected)} \end{cases} \tag{2}$$

NewReno involves a linear increase in the transmission rate when congestion is not detected, and the congestion window size is halved when congestion is detected.

## Evolving congestion control algorithms

Grammatical evolution is a method that employs generation rules written in the Backus-Naur notation (BNF) as substitution rules from the genotype to phenotype. This method improves the disadvantage of genetic programming; the functions and programs generated using genetic programming are not guaranteed to be grammatically correct.

```
1  <algorithm> ::= if state == OPEN:
2                      <code1>
3                  elif state == DISORDER:
4                      <code2>
5                  elif state == RECOVER:
6                      <code2>
7                  elif state == LOSS:
8                      <code2>
9                  else:
10                     <code2>
11 <code1> ::= new_cwnd = <update>
12           | if <condition>:
13                 new_cwnd = <update>
14             else:
15                 new_cwnd = <update>
16 <code2> ::= new_cwnd = <update>
17             new_ssthresh = <update>
18           | if <condition>:
19                 new_cwnd = <update>
20                 new_ssthresh = <update>
21             else:
22                 new_cwnd = <update>
23                 new_ssthresh = <update>
24 <condition> ::= obs[<obs_index>]<comp_op>obs
     [<obs_index>]
25 <update> ::= <update><arith_op><update>
26           | obs[<obs_index>]
27           | <num>
28 <obs_index> ::=
     0|1|2|3|4|5|6|7|8|9|10|11|13|
29 <comp_op> ::= <|>|<=|>=|==|!=
30 <arith_op> ::= +|-|*|/|%
31 <num> ::= 1|2|3|4|5|6|7|8|9
```

Figure 2: Grammar for generating network congestion control algorithms using grammatical evolution

Considering we need a measure to evaluate the performance of each agent (algorithm) when it is evolving, we also used an average throughput as a fitness measure to evaluate the performance of the congestion control algorithm using the equation 3.

$$Fitness = \frac{1}{N}\sum_{t} throughput_t \tag{3}$$

Here, $throughput_t$ is the throughput measured at time $t$ and $N$ is the number of times the throughput is measured. The throughput was measured by a single server, and the average throughput was calculated from the throughput measured at regular time intervals during the simulation. The throughput is defined as the amount of data received by the server per second, using the equation 4.

$$throughput_t = \frac{(B_t - B_{t-interval}) \times 8}{interval}\,[bps] \tag{4}$$

Here, $B_t$ denotes the bytes of data received by the server by time $t$ and $interval$ indicates the interval in seconds at which the total amount of data received by the server is measured. The reason for considering 8 as the product is to convert bytes to bits. By maximizing the fitness in equation 3, we can evolve network congestion control algorithms to obtain a higher throughput.

Next, Figure 2 shows the grammar for generating network congestion control algorithms. We defined the grammar in such a way that the AIMD algorithm can be generated. In the grammar, "obs" indicates the observation information of the simulation, and "OPEN," "DISORDER," "RECOVER," and "LOSS" in lines 1 through 7 of the grammar indicate the following four congestion states, respectively.

- OPEN: Normal state
- DISORDER: State in which a duplicate ACK is received（congestion may be occurring）
- RECOVER: Duplicate ACK received three times（stronger suspicion of congestion occurring）
- LOSS: State in which ACK timeout is detected（transmission is lost due to congestion）

"OPEN" indicates no congestion, "DISORDER" and "RECOVER" indicate that congestion may be occurring, and "LOSS" indicates that transmission is lost due to congestion. The grammar defines a method for updating the congestion window size (new_cwnd) and the slow start threshold (new_ssthresh) for these four congestion states. The update method updates only four arithmetic operations or uses conditional branching by if statements and four arithmetic operations.

An example of the code generation process for the "OPEN" state using this grammar is shown below.

1. Generate code starting from a non-terminal symbol <algorithm> in the grammar.

```
if state == OPEN:
    <code1>
```

2. Generate code for non-terminal symbol <code1>. The code to be generated is determined from the value of the gene because there are two codes that can be generated for <code1>.

```
if state == OPEN:
    new_cwnd = <update>
```

3. Generate code for non-terminals <update> as in 2.

```
if state == OPEN:
    new_cwnd = <update> <arith_op> <update>
```

4. The process for determining the code generated from the non-terminal symbols based on the genes is repeated. For example, the following code is generated:

```
if state == OPEN:
    new_cwnd = obs[5] + obs[6]
```

The network congestion control algorithm generated in the example updates the congestion window size to the sum of the two observed values $obs[5]$ and $obs[6]$ when the congestion state is " OPEN." The larger the value of the congestion window size, the more data can be sent, and the higher the throughput, the more the algorithm evolves to allow the congestion window size to be increased while adjusting the congestion window size to prevent the occurrence of congestion.

## Experiments

In the experiments conducted in the present study, we examined the following three points.

1. Whether or not the difficulty of the environment can be controlled when the amount of generated cross-traffic generated and the rate of packet loss are varied
2. Whether it is possible to optimize the congestion control algorithm using grammatical evolution
3. The applicability of congestion control to environment-agent co-evolutionary algorithms

Throughout the experiments conducted in this study, the parameters of the network simulation were set as: bandwidth of 10 Mbps, delay of 45 ms, and simulation time of 10 s. The average throughput was calculated from the throughput measured every 0.1 s. During the simulation, we set the main client to continue sending data to the main server for as long as possible. In addition, the client generating the cross-traffic was set to send packets to the paired server at a transmission rate of 450 kpbs.

### Controllability of the level of difficulty in environment

First, we varied the amount of cross-traffic generation to determine whether it was possible to control the difficulty of the environment. Specifically, we examined whether the average throughput could be reduced by increasing the amount of cross-traffic generated. Using methods 1 and 2, we increased the number of server-client pairs to be added, thereby increasing the amount of cross-traffic generated. The number of server-client pairs to be added at both ends of the relay nodes (method 1) was varied from 1 to 2 and finally to 3 (topology ID: m1_p1, m1_p2, and m1_p3, respectively). We also varied the number of server-client pairs to be added between relay nodes (method 2) from 1 to 2 and finally to 3 (topology ID: m2_p1, m2_p2, and m2_p3, respectively). Then, we measured the average throughput in these environments. For comparison, we also measured the average throughput at the baseline of the experiment (topology ID: base, where the number of nodes for methods 1 and 2 is zero).

The simulation results are shown in Figure 3. The average throughput of the baseline was higher than that of all the cross-traffic environments. Moreover, the average throughput decreased as the number of pairs increased for both methods 1 and 2. From this result, we can infer that the difficulty of the environment can be controlled by varying the amount of cross-traffic generated.

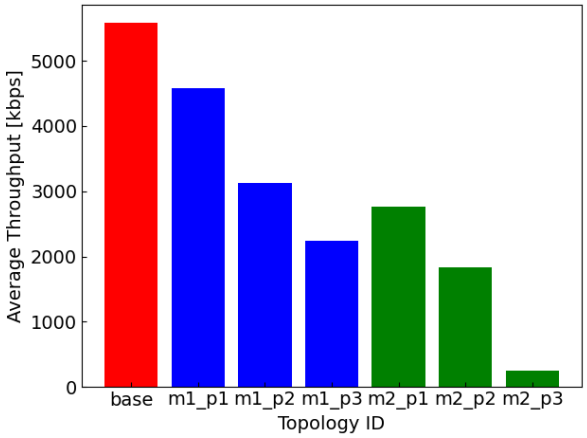



Figure 3: Changes in throughput in a cross-traffic-generated environment. The topology ID on the horizontal axis indicates the node placement method $m$(= 1 or 2) and the number of server-client node pairs $p$(= 1, 2, or 3).

### Evolvability of congestion control algorithms

Next, we tested whether a packet-loss-generating environment was effective for evolving a network congestion control algorithm using grammatical evolution. A baseline environment with no cross-traffic was used as the experimental environment. Network congestion control algorithms were evolved using grammatical evolution in environments with and without packet loss, and the changes in fitness between the two were compared. The packet loss rates were set to 0.0, 0.001, 0.01, and 0.1. A packet loss incidence of 0.0 corresponds to an environment without packet loss. The parameter settings for grammatical evolution were as follows: 50 for the number of individuals, 200 for the gene length of each individual, 20 for the number of mutations, and 900 for the number of generations. In addition, the gene values for each individual were initialized to include the update equation for the slow-start congestion window of NewReno. By using the update equation of the NewReno algorithm, we increased the probability of generating an algorithm with minimum performance.

The results are shown in Figure 4. When the fitness between the environments with and without packet loss was compared, the fitness was observed to be better in the envi-

ronment without packet loss (packet loss rate = 0). This result can be attributed to the fact that the environment without packet loss is more likely to produce throughput. Furthermore, considering the changes in fitness in the environment without packet loss, the fitness became a constant value after the initial generation. In contrast, in an environment with packet loss, the fitness gradually improved after the initial generation. This can be attributed to the fact that the occurrence of packet loss provides an environment with the level of difficulty necessary for the evolution of the algorithm. Considering the case of a packet loss ratio of 0.01, the algorithm for updating the congestion window of the first generation became a simple additive formula as follows (the following shows the updating method only in the "OPEN" state).

Listing 1: first generation with packet loss rate 0.01

```
new_cwnd = obs[5] + obs[6]
```

By the 300th generation, the algorithm for updating the congestion window evolved into a more complex update formula that included conditional statements:

Listing 2: 300th generation with packet loss rate 0.01

```
if obs[8] != obs[6]:
    new_cwnd = obs[8] / obs[5] + obs[6] % 6
else:
    new_cwnd = obs[4]
```

Next, we show a part of the two algorithms evolved for the 300th generation with packet loss rates of 0.001 and 0.1.

Listing 3: 300th generation with packet loss rate 0.001

```
new_cwnd = obs[8] * obs[7]
```

Listing 4: 300th generation with packet loss rate 0.1

```
if obs[5] != obs[6]:
    new_cwnd = obs[7]
else:
    new_cwnd = obs[4] / 1
```

For a packet loss rate of 0.001, the result of multiplying the two observed values was used as the congestion window size. In contrast, the packet loss rate of 0.1 used the observed value after conditional branching as the congestion window size. These network congestion control algorithms have evolved to possess different characteristics for each environment. However, it is difficult to clearly explain the difference between the fitness values of the generated algorithms. Generating an algorithm that is rational and human-understandable is a challenge to be addressed in future.

## Applicability to co-evolution of environment and agent

Finally, we tested the applicability of the network congestion control algorithms to environment-agent co-evolutionary algorithms. In this experiment, we evaluated the fitness of the three network congestion control algorithms described in Section "Evolvability of congestion control algorithm". That is, the 300th generation of the network congestion control algorithms with packet loss rates evolving at loss rates of 0.001, 0.01, and 0.1.

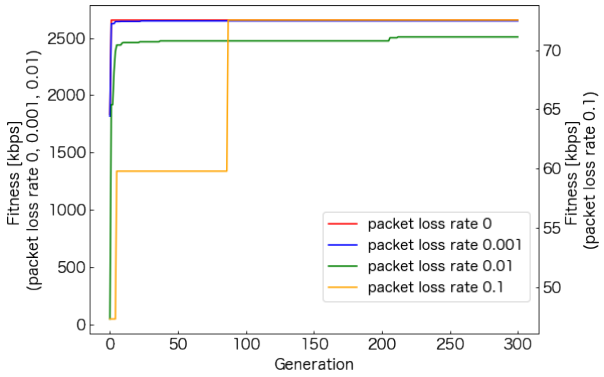



Figure 4: Changes in best fitness of each generation in the no packet loss (loss rate = 0.0) and packet loss (loss rate = 0.001, 0.01, 0.1) environments

We evaluated these algorithms in environments with and without cross-traffic. In environments without cross-traffic, NewReno and an algorithm that always uses a constant value as the congestion window size (the upper limit of the congestion window size in our experiments) were used as baselines. In environments with cross-traffic, we verified the efficiency of these algorithms (evolved in environments without cross-traffic) in the cross-traffic environment. For the network without cross-traffic, we used baseline topology. For the generation of network topology with cross-traffic, we used both methods 1 and 2. Specifically, we used a network topology where the number of server-client pairs for method 1 was one, and that of method 2 was three. For each network, the simulation was conducted for packet loss rates of 0.0, 0.1, 0.05, 0.01, 0.005, 0.001, and 0.0005. For each parameters, we conducted 25 trials by making staring time of the main flow slightly varied in 0.01 second step.

Figure 5 shows the results[2]. In comparing the fitness of the algorithms evolved with a packet loss rate of 0.001 and 0.01 without cross-traffic and with cross-traffic, the fitness of the algorithms without cross-traffic is higher for all packet loss rates. Hence, we can say that these algorithms have been optimized for the environment without cross-traffic, which is the environment at the time of evolution.

Similarly, in regard to the best algorithms under packet loss rate 0.001 and 0.01 in an environment without cross-traffic, we found that the algorithms with the best fitness

[2]Some results with cross-traffic and packet loss rate of 0.05 and 0.1 are excluded from the figure because network simulations could not be completed because of irregular stops in ns3.

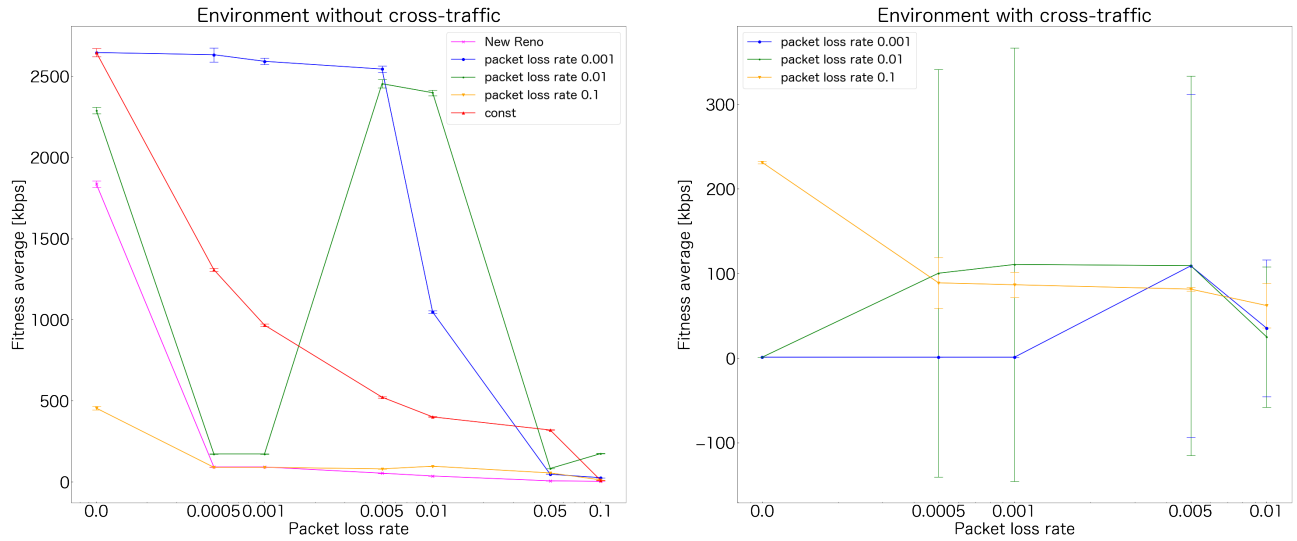



Figure 5: Relationship between packet loss rate and fitness of each algorithm: Average fitness are calculated from the results of 25 trials. Error bars show standard deviations. The Figure-(left) shows results in the environment without cross-traffic. This shows the results of comparing baseline algorithms(NewReno, const) with algorithms evolved in an environment with packet loss rates of 0.001, 0.01, and 0.1. The Figure-right shows results in the environment with cross-traffic. This shows the performance of the algorithms evolved in an environment without cross-traffic in an environment with cross-traffic. In some environments with cross-traffic and a packet loss rate of 0.1 and 0.05, a network simulation could not be completed because of an irregular stop of ns3. Those faulty results are excluded.

were those that evolved in the same packet loss rate environment as that used for evolution. Hence, we can say that the algorithm has evolved such that it is optimized for each packet loss rate environment.

However, a very simple algorithm with a constant value as the congestion window size (const of 5) had the same level of fitness as the other algorithms. This result indicates that the simulation environment may not be able to reflect the characteristics of the Internet. This suggests that the algorithm evolved in the experiment is adapting to a special environment that is different from the Internet. Therefore, the future challenge is to generate an algorithm that performs better than existing algorithms such as NewReno in an environment that reflects the characteristics of the Internet.

In contrast, in an environment with cross-traffic, the average and standard deviation of fitness shows that the algorithm evolved with a packet loss rate of 0.1 has a stable fitness for all the evaluated environments. This result suggests that the algorithm evolved in a difficult environment can be an effective algorithm for other environments. Interestingly, the algorithm that evolved in a more favorable environment without cross-traffic worked well in a relatively challenging environment with cross-traffic. This may be because a packet loss ratio of 0.1 created a rather harsh environment even in the absence of cross-traffic.

This result suggests the feasibility of our approach regarding the network congestion control algorithm using a co-evolutionary algorithm of environments and agents such as POET.

## Conclusion

We examined the applicability of an environment-agent co-evolutionary algorithm to network congestion control algorithms. We found that the amount of cross-traffic generation and the rate of packet loss are useful for controlling the difficulty of the environment, which is necessary when using an environment-agent co-evolutionary algorithm. In addition, grammatical evolution was found to be useful for optimizing the congestion control algorithm, which corresponds to the optimization part of the agent. Finally, we show that the co-evolutionary algorithm of the environment and agent can be used for the congestion control algorithm. This result is a contribution to open-ended evolution research in that it shows that a co-evolutionary framework such as POET can be adapted to network congestion control algorithms (agents) and network simulations (environments).

In the future, we would like to extend our work to use frameworks such as POET to automatically co-evolve both the environment and the algorithm. Eventually, we aim to automatically generate algorithms and environments that will surpass those already discovered by humans through co-evolution. The usefulness of the co-evolution of an environment and agent also lies in the fact that the environment is generated according to the progress of the agent's optimization. In the past, when network congestion control algorithms were evaluated, the parameters of the environment

were adjusted by a researcher. Hence, the more the number of parameters used, the more the level of difficulty in making appropriate adjustments. Therefore, there was a limit to the number of environments that could be used for evaluation. However, by using the co-evolutionary algorithm of environments and agents, we could search for parameter settings with an appropriate level of difficulty for each algorithm. This is expected to lead to the automatic discovery of new parameter settings for environments that are useful for verification. Simultaneously, by automatically generating and optimizing algorithms in the various environments generated, new and useful network congestion control algorithms can be discovered.

In addition, we believe that it is possible to generate a wider variety of network congestion control algorithms by improving the grammar used in grammatical evolution. In the grammar used in the present study, the method of updating the congestion window is defined inside a predefined conditional branch. Therefore, there is a limit to the number of congestion control algorithms that can be generated. Removing these restrictions allows for the generation of many more network congestion control algorithms. However, there are two problems that arise when removing these constraint: first, the guarantee that the generated algorithms will work correctly is reduced; second, finding the optimal solution may take a very long time. Hence, to improve the grammar and enable generation of a wide variety of network congestion control algorithms, it is necessary to investigate methods to solve these problems. However, if this problem can be solved, new network congestion control algorithms can be discovered.

## References


Al-Saadi, R., Armitage, G., But, J., and Branch, P. (2019). A survey of delay-based and hybrid tcp congestion control algorithms. *IEEE Communications Surveys Tutorials*, 21(4):3609–3638.

Assunção, F., Lourenço, N., Machado, P., and Ribeiro, B. (2017). Automatic generation of neural networks with structured grammatical evolution. In *2017 IEEE Congress on Evolutionary Computation (CEC)*, pages 1557–1564.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym. *CoRR*.

Burke, E. K., Hyde, M. R., and Kendall, G. (2012). Grammatical evolution of local search heuristics. *IEEE Transactions on Evolutionary Computation*, 16(3):406–417.

Dharna, A., Togelius, J., and Soros, L. B. (2020). Co-generation of game levels and game-playing agents. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 16(1):203–209.

Fenton, M., McDermott, J., Fagan, D., Forstenlechner, S., Hemberg, E., and O'Neill, M. (2017). Ponyge2: Grammatical evolution in python. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pages 1194–1201.

Gawłowicz, P. and Zubow, A. (2019). Ns-3 meets openai gym: The playground for machine learning in networking research. In *ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWiM)*, pages 113—120.

Gurtov, A., Henderson, T., Floyd, S., and Nishida, Y. (2012). The NewReno Modification to TCP's Fast Recovery Algorithm. RFC 6582.

Jay, N., Rotman, N., Godfrey, B., Schapira, M., and Tamar, A. (2019). A deep reinforcement learning perspective on internet congestion control. In *Proceedings of the 36th International Conference on Machine Learning*, pages 3050–3059.

Li, W., Zhou, F., Chowdhury, K. R., and Meleis, W. (2019). Qtcp: Adaptive congestion control with reinforcement learning. *IEEE Transactions on Network Science and Engineering*, 6(3):445–458.

Neupane, A. and Goodrich, M. (2019). Learning swarm behaviors using grammatical evolution and behavior trees. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 513–520. International Joint Conferences on Artificial Intelligence Organization.

O'Neill, M. and Ryan, C. (2001). Grammatical evolution. *IEEE Transactions on Evolutionary Computation*, 5(4):349–358.

Packard, N., Bedau, M. A., Channon, A., Ikegami, T., Rasmussen, S., Stanley, K. O., and Taylor, T. (2019). An Overview of Open-Ended Evolution: Editorial Introduction to the Open-Ended Evolution II Special Issue. *Artificial Life*, 25(2):93–103.

Peterson, L. L. and Davie, B. S. (2011). *Computer Networks, Fifth Edition: A Systems Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 5th edition.

Shaker, N., Nicolau, M., Yannakakis, G. N., Togelius, J., and O'Neill, M. (2012). Evolving levels for super mario bros using grammatical evolution. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 304–311.

Wang, R., Lehman, J., Clune, J., and Stanley, K. O. (2019). Poet: Open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 142–151.