Toward automatic generation of diverse congestion control algorithms through co-evolution with simulation environments

Teruto Endo, Hirotake Abe, Mizuki Oka

University of Tsukuba, Tsukuba, Ibaraki 305-8573 Japan

mizuki@cs.tsukuba.ac.jp

Abstract

Congestion control algorithms are used to help prevent congestion from occurring on the Internet. However, a definitive congestion control algorithm has yet to be developed. There are three reasons for this: First, the environment and usage of the Internet continue to evolve over time. Second, it is not clear what congestion control algorithms will be required as the environment evolves. Third, there is a limit to the number of the congestion control algorithms that can be developed by researchers. This paper proposes a method for automatically generating diverse congestion control algorithms and optimizing them in various environments by co-evolving network simulations as environments and congestion control algorithms as agents. In experiments conducted using co-evolution, although the algorithms generated were not on par with conventional practical congestion control algorithms, the intent of the procedures in the algorithms was interpretable from a human perspective. Furthermore, our results verify that it is possible to automatically discover a suitable environment for the evolution of a congestion control algorithm.

Introduction

Congestion is the excessive temporal flow of data over a network. On the Internet, many users constantly compete for communication bandwidth. This causes congestion, which may result in reduced communication speed and connection problems for the service. To prevent Internet congestion, congestion control algorithms are used to adjust the data transmission rate.

A congestion control algorithm determines whether congestion is occurring and accordingly decides whether to increase or decrease the amount of data sent. The two main types of congestion control algorithms are loss-based and delay-based algorithms (Al-Saadi et al., 2019). Loss-based algorithms determine whether congestion is occurring based on the packet loss, whereas delay-based algorithms make the determination based on the round-trip time of packets.

Research on congestion control algorithms has been ongoing for over 40 years; however, no definitive congestion control algorithm has yet been developed. This is because Internet usage and connection patterns change over time,

and it is unclear what algorithms are necessary when the Internet structure or bandwidth drastically changes. Existing congestion control algorithms are built using a rule-based approach, which makes it difficult for them to respond to changes in the environment (Li et al., 2019). In addition, the number of the congestion control algorithms that can be developed by researchers and the number of environments in which they can be tested are limited.

One possible approach to address these issues is to use program synthesis to automatically build algorithms without human intervention (Gulwani et al., 2017). In this regard, an interesting study was conducted by Lones (2020), who used genetic programming to automatically generate an optimization algorithm. The study highlights the possibility that only a limited portion of the design space of optimization algorithms has been explored because of biases in human thinking. In fact, by using genetic programming, they succeeded in generating an algorithm that has both superior performance and different characteristics from existing human-made optimization algorithms. A similar challenge exists for congestion control algorithms.

In machine learning, which has made remarkable progress as represented by deep learning, a method called AutoML-Zero has been proposed to build algorithms from scratch using evolutionary methods (Real et al., 2020). Although AutoML-Zero has not yet discovered any new valuable algorithms, it has rediscovered useful machine learning methods, such as backpropagation. This result, which automatically builds on existing methods from scratch, suggests the possibility of automatically discovering useful undiscovered algorithms in the future.

This attempt to automatically generate algorithms using evolutionary methods without human intervention has the potential to devise algorithms that have not yet been discovered. This indicates that there may be useful algorithms existing outside the space already heuristically explored by humans. Adding diversity to the solution, rather than simply searching for a single optimal solution, will lead to the discovery of a global optimal solution. Recently, an algorithm called the quality–diversity algorithm, which attempts

to improve both the quality of the solution (the fitness of the solution) and the diversity of the solution (the types of solutions), has been studied (Pugh et al., 2016). The results show that improving the diversity of solutions rather than searching for a single optimal value can lead to the discovery of better solutions.

The paired open-ended trailblazer (POET) (Wang et al., 2019), which co-evolves the agent and environment, has been attracting attention as a further development of the quality–diversity algorithm. POET learns by pairing an agent with an environment generated by mutation, and optimizes the agent for each paired environment. In addition, high-performance agents are transferred to other environments. In this manner, a new environment is automatically generated at the appropriate time, and agents are optimized in the new environment as well to discover solutions (agents) to unknown problems (environments). By co-evolving agents and environments, we can obtain useful environments and agents that cannot be obtained without co-evolution. However, most applications of the quality–diversity algorithm are in games and robotics. Therefore, further development can be expected by demonstrating that it can be applied to other fields.

Therefore, this study examines congestion control algorithms as a new application of the quality–diversity algorithm and agent–environment co-evolution. As no attempt has been made to automatically generate congestion control algorithms using evolutionary algorithms, it is hoped that the application of the quality–diversity algorithm and POET to the automatic generation of congestion control algorithms will lead to the discovery of new and useful congestion control algorithms. Specifically, we aim to automatically construct a variety of congestion control algorithms and simulation environments without human intervention.

Agent and Environment

In this study, the congestion control algorithms are the agents, and the network simulations are the environments. This section describes how the congestion control algorithms are generated and how the environments are controlled.

Controlling the environments

To co-evolve environments and agents, it is necessary to control the difficulty of the environments. In this study, the throughput is used as an indicator of the difficulty of the environment. It is necessary to design parameters such that the higher the throughput, the lower the difficulty; conversely, the lower the throughput, the higher the difficulty. Therefore, we use two types of parameters to control the difficulty level: the amount of cross-traffic generated and the packet loss rate.

Cross-traffic is traffic that interferes with the main communication. For example, it communicates on a communi-

cation path that overlaps with the communication path between the main server and clients. When the communication paths overlap, it puts pressure on the bandwidth. In such a case, if the communication data are not properly controlled, congestion will occur and throughput will decrease. Therefore, the amount of cross-traffic generated is varied to control the difficulty of the environment.

There are two ways to control the amount of cross-traffic generated. The first is to vary the amount of data communication between the server and client that generates the cross-traffic, and the second is to vary the number of nodes that generate the cross-traffic.

Figure 1 is a conceptual diagram of a network topology in which cross-traffic can occur. There are three types of nodes: servers, clients, and relay nodes. Relay nodes only relay data between main traffic and cross-traffic, and do not generate any traffic. The red nodes indicate nodes that generate main traffic, and the blue nodes indicate nodes that generate cross-traffic. The server that measures the throughput is called the main server. The client that sends data to the main server is called the main client.

When the second parameter, packet loss, occurs, data must be retransmitted and received. Consequently, it takes time to retransmit packets, resulting in a decrease in throughput. Packet loss can be caused by disconnection in the case of wired networks or radio interference in the case of wireless networks. This is particularly likely to occur in wireless applications. Therefore, packet loss rate is considered an effective parameter for controlling the difficulty of the environment. Packet loss during an experiment occurs at all nodes in the network topology.

Figure 1: A network topology in which cross-traffic can occur.

We use ns3gym (Gawłowicz and Zubow, 2019) for network simulation, which is our environment. ns3gym is a framework that enables ns3 [1], a network simulator often used in network research, to be used in OpenAI Gym (Brockman et al., 2016), which is a library that provides an environment for reinforcement learning.

Basic congestion control algorithms

To automatically generate a congestion control algorithm, it is necessary to determine the base of the generated algorithm. We use the additive increase and multiplicative

---

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

decrease (AIMD) algorithm, a relatively simple but widely used classical congestion control algorithm, as a reference to create the base algorithm (Peterson and Davie, 2011).

The AIMD algorithm can be implemented in a relatively simple manner. Therefore, instead of using complex congestion control algorithms, this research also uses simple basic arithmetic operations and conditional branching, such as AIMD, as the basis for the design of the algorithms.

For AIMD, the congestion window size is determined by Equation 1.

$$x(t+1) = \begin{cases} x(t) + a & \text{(no congestion detected)} \\ x(t) \times b & \text{(congestion detected),} \end{cases} \quad (1)$$

where $x(t)$ represents the congestion window size at time t. AIMD is a very simple method that increases the congestion window size by addition and decreases it by multiplication. If congestion is not detected, the congestion window size is increased via parameter a $(a > 0)$. If congestion is detected, the congestion window size is decreased via parameter b $(0 < b < 1)$.

Many algorithms using the AIMD scheme have been proposed, but one of the most popular is NewReno (Gurtov et al., 2012). NewReno increases the congestion window size linearly if no congestion is detected and halves the congestion window size if congestion is detected. The formula for updating the congestion window size for NewReno is given by Equation 2.

$$x(t+1) = \begin{cases} x(t) + 1 & \text{(no congestion detected)} \\ \frac{x(t)}{2} & \text{(congestion detected)} \end{cases} \quad (2)$$

Automatic generation of congestion control algorithms

We use grammatical evolution (O'Neill and Ryan, 2001) to generate the congestion control algorithms. Grammatical evolution allows for a relatively free design of the generated code and functions, and because the generated code is grammatically guaranteed, it can be naturally applied to implementations that require conditional branching, such as congestion control algorithms. To run the grammatical evolution, we use PonyGE2 (Fenton et al., 2017), a Python 3 implementation.

The fitness is defined by the average throughput, as shown in Equation 3.

$$Fitness = \frac{1}{N} \sum_t throughput_t, \quad (3)$$

where $throughput_t$ represents the throughput measured at time t and N represents the number of times the throughput is measured. The throughput is measured using the main traffic server at regular intervals. The server records the amount of data received at regular intervals and calculates it using Equation 4.

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

where B_t indicates the number of bytes received by the server at time t and $interval$ indicates the interval in seconds at which the server records the amount of data received. It is multiplied by eight to convert the byte unit to bit unit. By maximizing the fitness, it is possible to develop the congestion control algorithm with a higher throughput.

Figure 2 illustrates the grammar used to generate the congestion control algorithm. The grammar is defined such that it can generate an algorithm that can be implemented using four arithmetic operations and conditional branching, such as AIMD. The "obs" in the grammar indicates the information observed during the simulation. The "OPEN," "DISORDER," "RECOVER," and "LOSS" in lines 1 through 7 of the grammar represent the following four congestion states:

- OPEN: Normal state

- DISORDER: State in which a duplicate ACK is received

- RECOVER: Duplicate ACK is received three times (stronger suspicion of congestion occurring)

- LOSS: State in which ACK timeout is detected (transmission is lost owing to congestion)

"OPEN" indicates that congestion is not present, "DISORDER" and "RECOVER" indicate that congestion may be present, and "LOSS" indicates that communication has been lost owing to congestion. In addition, ACK is a packet that indicates to the data sender that the data were received correctly. The grammar shown in Figure 2 defines four methods for updating the congestion window size (new_cwnd) for the four different congestion states. These update methods are defined by the observational information of the environment, the four arithmetic operators, and conditional branching. The larger the value of the congestion window size, the more data that can be transmitted. To improve the fitness, we need to evolve the algorithm to adjust the congestion window size to prevent congestion and communicate more data.

Co-evolution of simulation environments and algorithms

Our aim is to automatically generate congestion control algorithms in a variety of simulation environments through co-evolution. We use POET to co-evolve the simulation environments and congestion control algorithms. POET is characterized by the fact that it generates new environments that are appropriate for the evolution of individuals. In addition to the automatic generation of new algorithms, POET is expected to automatically generate environments that are optimal for evolution, and discover environments that will generate interesting individuals.

To use POET, we need to determine the minimum and maximum values of the fitness that will be used as a criterion to decide whether or not to perform mutation of the environment. It is necessary to set a criterion value that is

```
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: A grammar for generating congestion control algorithms in Grammatical Evolution

neither too easy nor too difficult for the agent. However, it is difficult to determine criterion values in advance. Therefore, we set 10% of the bandwidth in the experiment as the minimum value and 80% as the maximum value. In addition, the search space for combinations of algorithms that can be generated by grammatical evolution and environments that can be generated by network simulation is huge. To reduce the search space, we first evolve a congestion control algorithm using a quality–diversity algorithm and use it in the initial population set of POET.

For the quality–diversity algorithm, we use CMA-ME (Fontaine et al., 2020), coupled with an improvement emitter to improve the fitness of the individuals. Thus, individuals are generated by CMA-ME, and the genetic information of the generated individuals is replaced with code using the grammar defined by BNF. In the quality–diversity algorithm, it is necessary to define a behavior descriptor (BD) that represents the characteristics of the solution. We use the mean and standard deviation of the congestion window size as the BDs. The mean value of the congestion window

size is an indicator of the trend in the size of the congestion window set by the algorithm. The standard deviation of the congestion window size is an indicator of the degree of variation in the congestion window size. For example, if the mean value of the congestion window size is high and the standard deviation is low, the algorithm is considered to be a congestion control algorithm with the characteristic of selecting a consistently large congestion window size. We used the pyribs (Tjanaka et al., 2021) library to run CMA-ME.

Experimental Evaluation

We conducted two experiments to evaluate the feasibility of the proposed method.

First, we evolved congestion control algorithms using the quality–diversity algorithm for use as the initial population set for co-evolution. We analyzed the generated set of algorithms, the characteristics of the algorithms with a high degree of fitness, and the algorithms that are frequently generated. Next, we examined whether useful simulation environments and congestion control algorithms could be generated automatically through co-evolution using POET.

Automatic generation experiments using the Quality–Diversity algorithm

We automatically generated congestion control algorithms by combining CMA-ME and grammatical evolution. The simulation environment was configured as follows: a dumbbell-shaped network topology with two nodes at each end, as shown in Figure 1; a packet loss rate of 0.0; a bandwidth of 10 Mbps; a delay of 45 s; and a cross-traffic data transmission rate of 6 Mbps. The simulation time was set to 30 s.

The mean (hereinafter referred to as BD1) and standard deviation (denoted BD2) of the congestion window size were used as the BD for CMA-ME. Because BD needs to be discretized in CMA-ME, the minimum and maximum values of BD1 were set to 0 and 3×10^5, the maximum and minimum values of BD2 were set to 0 and 1.5×10^5, and BD1 and BD2 were divided into 50 equally spaced parts. In the experiment, 48 individuals were generated in each iteration of CMA-ME, and 5600 iterations were performed.

Analysis of the relationship between behavior descriptor and fitness

We analyzed the relationship between the BDs and fitness to determine which features of the algorithm adapted to the environment. A heat map showing the relationship between the BDs and fitness is given in Figure 3. The horizontal axis shows the mean value of the congestion window size, which is the first BD, and the vertical axis shows the standard deviation of the congestion window size, which is the second BD. The fitness is expressed by the brightness of the color,
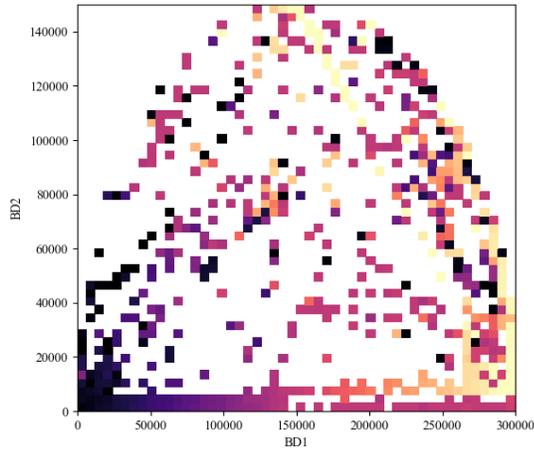
Figure 3: Heat map showing the relationship between BD and fitness (horizontal axis (BD1): mean value of the congestion window size; vertical axis (BD2): standard deviation of the congestion window size).
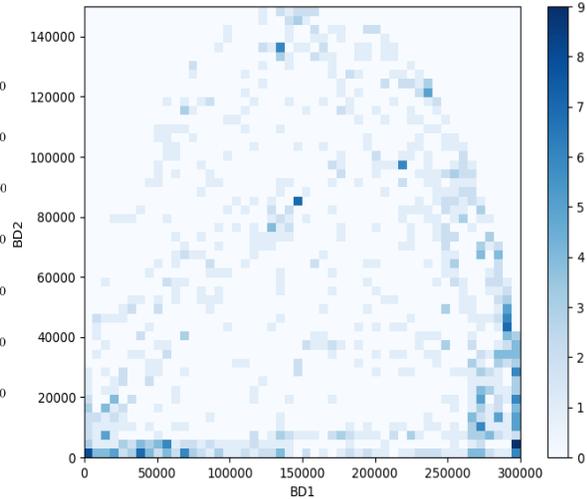

Figure 4: Heat map showing the relationship between BDs and the number of updates for each cell (horizontal axis (BD1): mean value of the congestion window size; vertical axis (BD2): standard deviation of the congestion window size).

with lighter colors indicating higher adaptation and darker colors indicating lower adaptation.

Considering the regions with high adaptability, we find that algorithms with high adaptability are distributed on the lower righthand side of the heat map and the upper center of the heat map. The lower righthand side of the heat map shows the area where the congestion window size tends to be always large. The upper center of the heat map is a region where the value of the congestion window size frequently switches between the upper limit of the congestion window size and zero. This indicates that in the environment, setting the congestion window size to a large value results in high throughput.

Analysis of the number of updates for each cell in the archive

Next, we visualized the relationship between the number of updates of each cell in the archive and BDs and analyzed the characteristics of the individuals that were likely to be generated. A heat map showing the relationship between the number of updates for each cell and the BDs is shown in Figure 4. The horizontal and vertical axes are the same as those shown in Figure 3. The number of updates is represented by the brightness of the color, with brighter colors indicating fewer updates and darker colors indicating more updates.

The regions with the highest number of updates are identified in the lower right and lower left of the heat map. These are the areas where algorithms that barely change the size of the congestion window are distributed. In other words, the

most frequent updates were for simple algorithms that did not significantly change the congestion window size. This result indicates that relatively simple algorithms tend to be generated.

Analysis of generated algorithms

We investigated how algorithms with a high fitness value control congestion window size. First, we visualized the time variation of the congestion window size using the generated algorithms. Among the generated algorithms, the top three individuals with the highest fitness were Algorithms 1, 2, and 3, and the time variation of their congestion window size is shown in Figures 5a, 5b, and 5c, respectively.

Note that the fitness of all of these algorithms was 7,928 kbps. Algorithm 1 (Figure 5a) sets the congestion window size alternately between the upper limit and zero within a very short time interval. Algorithm 2 (Figure 5b) sets the upper limit and zero for the congestion window size to longer time intervals than those in Algorithm 1. Algorithm 3 (Figure 5c) varies the congestion window size between 250,000 and 300,000, while alternating between the upper limit and zero.

We verified in detail the control method of Algorithm 3 (Fig. 5c) from the generated code because it had a more complicated control method than the other two algorithms. Part of the code for Algorithm 3 is shown in Figure 6. There are two noteworthy points. The first is the setting of the congestion window size when a duplicate ACK is received in Line 2. Here, the congestion window size was set to the current slow-start threshold (obs[4] in the code). The second is

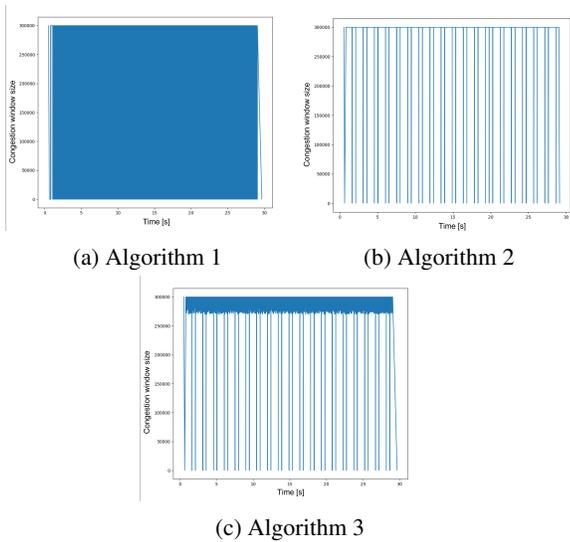(a) Algorithm 1   (b) Algorithm 2

(c) Algorithm 3

Figure 5: Time variation of congestion window size for the three algorithms with the highest fitness value.

the setting of the slow-start threshold when a duplicate ACK is received three times in Line 6. Here, the new slow-start threshold was set to a value that was manipulated from the current slow-start threshold. Such processing has also been observed in existing algorithms. From these findings, it is clear that part of the operation of the congestion control algorithm in Algorithm 3 is processed in a human-explainable manner. Therefore, we decided to include Algorithm 3 in the initial population set of the co-evolution.

Co-evolution of simulation environments and congestion control algorithms

We used POET to co-evolve a network simulation environment and a congestion control algorithm to verify whether a useful simulation environment and congestion control algorithm can be generated automatically. To increase the efficiency of evolution, we initialized half of the individuals using Algorithm 3 (Figure 5c) and the other half randomly.

A dumbbell network topology was used for the network simulation environment, similar to the experiments with the quality–diversity algorithm. We used the number of dumbbell node pairs, packet loss rate, and cross-traffic data transmission rate as three parameters to control the environment. The possible values to be selected for each parameter are as follows: 2, 3, 4, or 5 number of node pairs, packet loss rates at 0.0, 0.001, 0.01, or 0.1, and data transmission rates 6, 7, 8, or 9 Mbps. We set the number of dumbbell node pairs in the initial environment to two, the packet loss ratio to 0.0, and the cross-traffic data transmission rate to 6 Mbps. As parameters common to all environments, we set the bandwidth to 10 Mbps and delay to 45 ms. Environmental mutations were performed every 30 iterations and individual transfers every 10 iterations, for a final total of 303 iterations.

```
1  elif state == DISORDER:
2      new_cWnd = obs[4]
3      new_ssThresh = 5
4  elif state == RECOVER:
5      new_cWnd = obs[8]
6      new_ssThresh = 5 * obs[4]
```

Figure 6: Part of the code for Algorithm 3.

Results of co-evolution experiments

Fourteen pairs of network simulator environments and congestion control algorithms were generated. Among the environments, we found an environment that appeared to have prompted the evolution of the congestion control algorithm. The environment was generated during the 239th POET iteration and had the following parameters: packet loss rate, 0.001; dumbbell node pair count, 4; cross-traffic data transmission rate, 8 Mbps (this environment is referred to as Environment A). A graph showing the highest fitness of the algorithm for each generation is represented by the blue line in Figure 7. It can be observed that the fitness gradually increased with each generation. However, except for the pair of Environment A, no increase in fitness was observed and the fitness was constant from the initial generation. These results suggest that an environment with a packet loss rate of 0.001, dumbbell node pair count of 4, and cross-traffic data transmission rate of 8 Mbps is conducive to the evolution of congestion control algorithms.

However, we had to ascertain whether the set of algorithms in environment A changed fitness in other environments. Therefore, we evaluated the algorithms generated in Environment A in other environments to verify whether Environment A promoted evolution. The evaluation environment was not the 10 Mbps bandwidth used up to this point, but a bandwidth of 1 Gbps, assuming the current Internet environment. The other environmental parameters were identical to those of Environment A. Owing to the time constraints of the experiment, we evaluated the algorithms for the generations (1, 2, 11, 12, 57, 58, 61, and 62) in which there were changes in fitness in the 10 Mbps environment. Another purpose of the evaluation in a 1 Gbps bandwidth environment was to verify that the generated algorithms have high fitness in a modern environment.

Looking at the change in the highest value of the fitness of the algorithm generated in environment A at 1 Gbps, as shown by the orange line in Figure 7, there was no change in the fitness observed in environment A in the 1 Gbps environment. In other words, the differences in congestion control algorithms as captured in Environment A were not captured in the 1 Gbps environment. An environment in which differences in algorithms cannot be evaluated is inappropriate for algorithm evolution. This suggests that environment A is suitable for the evolution of paired algorithms.
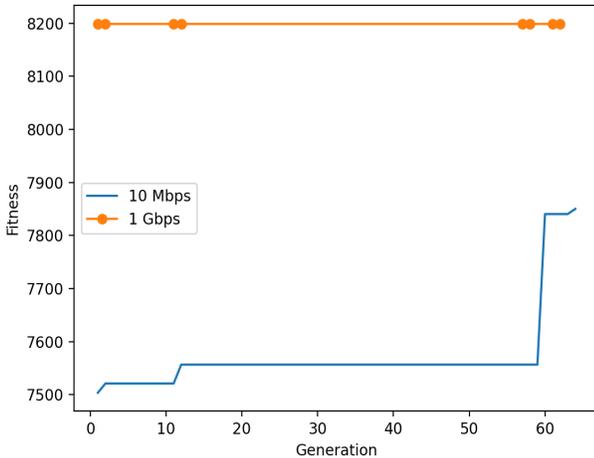
Figure 7: Change in maximum fitness for each generation in environments with bandwidths of 10 Mbps (blue line) and 1 Gbps (orange line). The environmental parameter settings were as follows: packet loss ratio of 0.001, 4 pairs of dumbbell nodes, and cross-traffic data transfer rate of 8 Mbps. (Owing to experimental time constraints, only the generation of algorithms that showed a change in fitness at 10 Mbps was evaluated at 1 Gbps.)

However, no congestion control algorithm was generated in Environment A, which performed interesting processing. The generated congestion control algorithm is a congestion control algorithm with very simple control, similar to the algorithm generated using CMA-ME. For example, algorithms that alternately select an upper limit and zero for the congestion window size (Figure5a, 5b) or algorithms that only set a small congestion window size. However, from Figure7, it is possible that further evolution would produce a congestion control algorithm that performs interesting processing, because the fitness increase was observed even after the last 60 generations.

Discussion

In this study, we were not able to automatically generate a useful congestion control algorithm that is as good as or better than existing algorithms. There are two possible challenges in this approach.

The first is the need to re-examine grammar. In the current grammar, using the quality–diversity algorithm and POET, many of the algorithms generated were very simple. As can be seen in Figure 4, which shows the number of updates for each cell in the BD space in the quality–diversity algorithm, many congestion control algorithms were generated that set very small or large values for the congestion window size. From this, it can be said that the current grammar easily generates simple congestion control algorithms. By improving

the grammar, we believe that it will be possible to generate useful congestion control algorithms with a higher probability. One way to improve the grammar is to make the patterns of programs that can be generated more limited than they are now.

Another issue is that the fitness used in this study may be insufficient for evaluating the performance of the algorithm. It is clear from the results that Algorithms 1, 2, and 3 each perform different congestion control. However, the average throughput, which indicates the fitness of these three algorithms, was 7,928 kbps for all three algorithms. Given that the bandwidth of the experimental environment was 10 Mbps, this means that all algorithms performed very well. In other words, it is unlikely that the fitness could be improved further. This is undesirable from the perspective of the objective of this study, which is the automatic generation of new congestion control algorithms. This is because the current index shows high fitness for simple algorithms, which may have a negative impact on the evolution of more complex algorithms, such as existing congestion control algorithms. We believe that it is possible to facilitate the generation of algorithms with the same complexity as existing congestion control algorithms by defining adaptations that can differentiate Algorithm 3, which is a relatively complex control compared to Algorithms 1 and 2.

Conclusion

Using the quality–diversity algorithm, we generated an algorithm that, while not as good as existing congestion control algorithms, can be interpreted by a human observer as to the intent of the process. This provides a foundation for the automatic generation of new and useful congestion control algorithms.

We also showed that by co-evolving the congestion control algorithm and the network simulation environment, it is possible to automatically discover useful environments for evolving the congestion control algorithm. This result indicates that a single environment alone is insufficient to promote evolution and suggests the effectiveness of the agent–environment co-evolution method.

In future work, we will examine the definition of fitness and improve the grammar used to generate congestion control algorithms. Then, we will perform automatic generation of congestion control algorithms to verify whether useful algorithms can be generated. Moreover, one of the advantages of environment generation in co-evolution is the possibility of automatically discovering environments that can yield useful insights. One environmental control parameter that may provide useful insights is network topology. The dumbbell network topology has been used frequently for many years in congestion control research because it is a simplified representation of a real-world situation where congestion is likely to occur. However, the dumbbell network topology is designed by humans, and there is no guarantee that there is

no network topology that can provide useful knowledge that has not yet been discovered. Therefore, we believe that the use of various automatically generated network topologies as environments may provide new insights.

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.

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

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.

Fontaine, M. C., Togelius, J., Nikolaidis, S., and Hoover, A. K. (2020). Covariance matrix adaptation for the rapid illumination of behavior space. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*, GECCO '20, page 94–102, New York, NY, USA. Association for Computing Machinery.

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.

Gulwani, S., Polozov, O., and Singh, R. (2017). Program synthesis. *Foundations and Trends® in Programming Languages*, 4(1-2):1–119.

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

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.

Lones, M. A. (2020). Optimising optimisers with push gp. *Genetic Programming. EuroGP 2020. Lecture Notes in Computer Science*.

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

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.

Pugh, J. K., Soros, L. B., and Stanley, K. O. (2016). Quality diversity: A new frontier for evolutionary computation. *Frontiers in Robotics and AI*, 3:40.

Real, E., Liang, C., So, D. R., and Le, Q. V. (2020). Automl-zero: Evolving machine learning algorithms from scratch. *arXiv preprint arXiv:2003.03384*.

Tjanaka, B., Fontaine, M. C., Zhang, Y., Sommerer, S., Dennler, N., and Nikolaidis, S. (2021). pyribs: A barebones python library for quality diversity optimization. https://github.com/icaros-usc/pyribs.

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*, GECCO '19, page 142–151, New York, NY, USA. Association for Computing Machinery.