A CBR-ANN hybrid for dynamic environments

Sondre Steinsland Hegdal[1] and Anders Kofod-Petersen[1,2]

[1] Department of Computer Science, Norwegian University of Science and Technology, NO-7491 Trondheim, Norway,
https://www.ntnu.edu/idi/
sondresh@stud.ntnu.no

[2] The Alexandra Institute,
Njalsgade 76, 3. sal, 2300 København S, Denmark,
https://alexandra.dk/uk
anders@alexandra.dk

Abstract. This paper proposes a Case-based Reasoning (CBR) and Artificial Neural Network (ANN) hybrid solution for dynamic problems. In this solution, a CBR system chooses between several expert neural networks for a given case/problem. These neural networks are Recurrent Neural Networks, which are trained using Deep Q-Learning (DQN). The system was tested on the game Mega Man 2 for the NES, and is compared to how a single recurrent neural network performed. The results collected outperforms the basic ANN that it was compared against, and provides a good base for future research on the model.

Keywords: Neural Network · Case-based reasoning · dynamic environment · CBR-ANN hybrid · Reinforcement learning · Game · Artificial Intelligence · Q-learning

CEUR-WS.org/Vol-2567/paper2.pdf

1 Introduction

The suggested method in this paper is a CBR-ANN hybrid that is supposed to work under dynamic, continuous problems. It aims at learning different behaviors for different scenarios, without running the risk of forgetting how to act in a certain way later on during the training section. To test this, the game Mega Man 2 is used, as this is both dynamic and continuous as well as having the possibility of having many, natural cases to take from during testing. The CBR section of the system will decide which neural network a given case should use as its solution, while the ANN chosen will decide what behavior is appropriate for the current scenario. The active neural network will control the behavior continuously until a new case is discovered, at which point the CBR takes over again. The problem that it tries to solve can be seen in the video by Seth Bling [2], with his NEAT approach for Super Mario World.

In order to evaluate the proposed method, the following research question was made:

RQ: How will a CBR system choosing between several expert Artificial Neural Networks perform compared to a single Artificial Neural Network in a dynamic, continuous system.

Copyright © 2019 for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

To answer this, the CBR-ANN hybrid system is compared against how a single ANN did on the same levels in Mega Man 2. The evaluation criteria is the accumulated rewards from the reward function that is used for training the system with Q-learning. The behavior is also compared. This should answer the question sufficiently, and the differences should become bigger the more cases are added.

While answering this question, the paper goes through the motivation for the method as well as lists some previous CBR-ANN hybrids in section 2. Then the model is explained in section 3, and the experiment in section 4. The results are then explained in 5, with discussion around them, as well as future work in 6. Finally there is a conclusion in section 7.

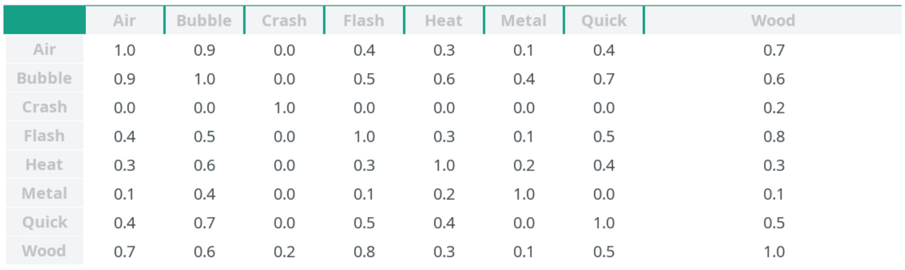
2 Background

The method proposed was inspired by the work by Seth Bling [2]. His NEAT approach to Super Mario World could do very well in single levels, but had trouble both when trying other levels, as well as forgetting previous good behavior when learning how to cope with new, but similar scenarios in the same levels. Because of the single neural network solution having trouble with different cases, a CBR-system was thought of that could be on top of several neural networks for the different situations, possibly making several neural networks together able to solve these problems that the single network could not.

The literature however does not have a lot of examples on using CBR and ANNs together in this way. The most common ways to combine these two methods in the literature was to either use ANNs as part of the retrieval process (see e.g. [7,9,4]) or to alter the solution to a problem [5]. These could be possible extensions to the proposed method, but not much more. Using neural networks as the reuse phase, like in the proposed method, seems rather rare, but not unexplored. The ones that did, reported good results, and outperformed the methods they were tested against [13,12,17,3]. None of these uses the same approach to using several networks though, and none use Q-learning as the revise phase. The approaches already explored are more for prediction/classification of a single or cluster of instance/case, before moving on to the next one, instead of the continuous approach done by the proposed system, where it has an active neural network that makes all decisions, until a new case appears and the CBR-section of the system changes the active neural network. Considering the positive results of previous work done on CBR-ANN hybrids the proposed method should do well, even if there is little to no research on this exact hybrid system previously.

3 Model

The model suggested here uses a rather simple version of a CBR system, following the traditional CBR cycle [1]. The task of the CBR-part of the system is to chose the best neural network to play a case/level, which is trained for each new level/case it encounters, using Deep Q-learning. During training, it uses the most similar case it has already seen as the base, or makes a random neural network if it has not seen any cases before. It uses a lookup table for the comparison of the levels (see Fig. 1), which is used for case retrieval. This section of the system is used at each new case encountered, which happens at set intervals of predicted actions in the experiments in this paper. After retrieving the most similar case it has previously seen, it reuses the neural network of this case for the new problem. During the training process it also has the revise step in the form of the Q-learning algorithm, to alter the neural network weights to better fit the current case. Lastly it has the retain step of the cycle active while it is training, and saves every new case it encounters in this mode. When it is in pure inference mode (while testing), it does not use the revise and retain steps.



	Air	Bubble	Crash	Flash	Heat	Metal	Quick	Wood
Air	1.0	0.9	0.0	0.4	0.3	0.1	0.4	0.7
Bubble	0.9	1.0	0.0	0.5	0.6	0.4	0.7	0.6
Crash	0.0	0.0	1.0	0.0	0.0	0.0	0.0	0.2
Flash	0.4	0.5	0.0	1.0	0.3	0.1	0.5	0.8
Heat	0.3	0.6	0.0	0.3	1.0	0.2	0.4	0.3
Metal	0.1	0.4	0.0	0.1	0.2	1.0	0.0	0.1
Quick	0.4	0.7	0.0	0.5	0.4	0.0	1.0	0.5
Wood	0.7	0.6	0.2	0.8	0.3	0.1	0.5	1.0

Fig. 1. The similarity measure used for the CBR part of the system.

The neural network architecture used is a recurrent neural network (RNN), which has backwards pointing weights into other nodes, so that it can remember previous experiences [10]. There are many ways to this, both comparetively simple and rather advanced, following is one of the simpler ways to make an RNN. These values from the backwards pointing weights into the recurrent nodes are then used as an extra input at the next time frame during the forward pass of the network, as well as being able to point to another recurrent node, making it remember several steps back in time. When these recurrent nodes point towards "regular" nodes, it behaves almost like a bias node, but one which has an updated value after each timestep, instead of a constant value. An example of a simple neural network can be seen in Figure 2. This is a rather strong architecture when it comes to sequential tasks, which playing a game is, and can also be seen with the LSTM architecture that is often used for other sequential tasks, like natural language processing [6]. All networks used in this paper has the same structure and parameters. The networks use 5 recurrent levels, meaning it remembers the last 5 inputs it has seen. They are places between the input and the first hidden layer. 242 input nodes were used, representing what the player sees on the screen in addition to the player's and the boss' remaining health points. There are 3 hidden layers of 1000 nodes each. It then ends in 14 input nodes, representing all the possible actions the agent can take.

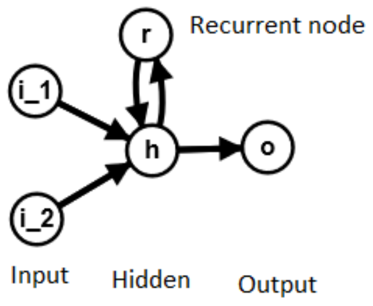


Fig. 2. A simple recurrent neural network.

The reinforcement learning approach taken is Deep Q-learning, which has been used for learning Atari games previously [11], with some modifications to fit this problem. DQN differentiates itself from regular Q-learning by having the neural network both do the prediction in the current timestep, and use it to calculate the predicted best value in the next step, instead of having a set function that can calculate the expected best value in the next timestep. The result is then used to change the weights of the neural network, using the predicted reward for the current time step up against the actual reward it got added with the predicted reward for the next timestep. The weights are updated using the backpropagation algorithm for deep neural networks [10]. This has to be done because the system does not have perfect understanding of the environment, and therefore can not calculate the possible rewards at every timestep, without doing the action and getting the reward. This approach does have a problem of overestimating the predicted value, and therefore eventually only do one action. To combat the overestimation problem, this paper uses a Double DQN approach, this means there is a target network used while training that predicts the next state, instead of using the same network that also predicted the current state [15]. This network is periodically updated to become the same as the network doing the action predictions, but much less frequently than the more simple approach.

The agent is rewarded for:

– Movement horizontally. 10 points per unit moved right, and -10 for each unit moved left. In addition it gets extra 10 points for each position it improves its best position.
– No movement gives -250 points, and eventually kills the player. An extra chance of random actions is added if there is no movement for a while.
– A death gives -510 points.
– Hurting the boss gives it 1 point for each health point the boss loses.
– Killing the boss gives 1020 points.

In the end, the reward is divided by 210 to keep the values slightly smaller, to avoid overflows. These values were the best ones found while testing parameters.

The agent(s) seemed to get better results when having big punishments for undesirable behavior, and only get a small amount of points for the desired behavior. This because otherwise it would find some place it could go back and forth to get a lot of points, but at the same time not progress in the level. The goal is to maximize the amount of points given by the reward function per level.

The agent(s) learn by collecting the input, input in the next timestep, output and reward at every timestep into a list. This is then used to calculate the data needed to do backpropagation and update the weights of the active neural network, by comparing what the network thinks the reward is for a given action, with what it actually got. The CBR-part of the system learns by saving the neural network it adapts to every new case it sees.

4 Experiment

The testing was done on the 1989 game Mega Man 2 for the NES. The agents played on normal difficulty on the American version of the game. This game, and other classic Mega Man game, was determined to be a fitting game for a CBR system to be tested on, because of how naturally you can make different cases with the level select screen.

There are 8 levels, where each one gives you a unique powerup for beating it, and it lets you choose any of them at any time, as long as you did not already beat it. The two figures below depict the level select screen of Mega Man 2, and how this works in the game. Fig. 3 shows the screen with no levels beaten, Fig. 4 shows it with Bubble man defeated.

There are 8! possible scenarios/cases including both the levels and the possible powerups that could be used, however only the 8 levels with no powerups are used in this experiment. The levels share the same goal of going to the right and kill the boss of the level at the end. The structure of the levels are unique for all of them outside of these general guidelines, including the boss fights at the end, which builds into it fitting well with a CBR system, and the goal is potentially rather simple for a Neural Network to learn.

This paper does not use the entire case-base mentioned above. The powerups are not used in this test, which then gives eight different cases used. Five of them are used in training, while all the levels are used for the testing stage. This was done to have it simpler in the early stages of research, as this is a rather untested method, especially in a continuous environment like this game is. The game also provided checkpoints at certain places, this experiment however made the agent play the entire level over again if it died, ignoring the checkpoints entirely.

For this experiment, the five levels *Quick man*, *Wood man*, *Metal man*, *Flash man* and *Air man* was used for training. The three levels *Heat man*, *Bubble man* and *Crash man* was then added during the testing stages. For the training part, the agents both started on *Wood man's stage*, but otherwise did training on the levels in random order. For testing, the levels were ordered the same for each agent, which is the same order as the levels were listed, but with *Heat man* and *Air man* switching places. *Wood man* was chosen as the starting point because

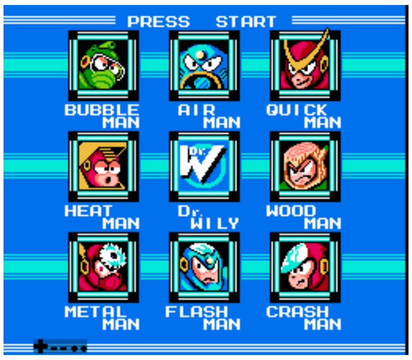


Fig. 3. No beaten levels

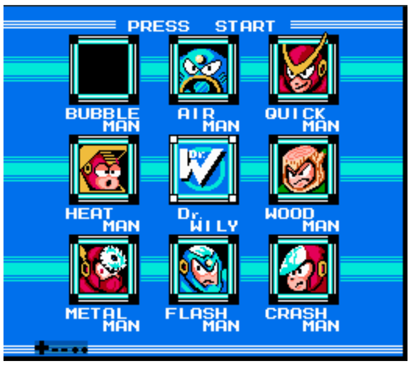


Fig. 4. Bubble man beaten

it is the most basic/simple of the levels, both considering the layout of the level and in its use of gimmicks compared to the rest. The other levels were chosen to have one of each "Level type", as they are rather unique, but still have a similar level in the test cases. The exception is *Crash man*, which is so different from the rest that the reward function does improve performance of the agents no matter how long they train.

After the training on the levels were completed, two tests were done. The first test with only the agents themselves choosing what action to take, and the other with 0.1 chance of a random action being taken instead. The full list of parameters, as well as all the code used for the training and testing can be found online[3].

5 Results

The complete training and testing used in the experiment is available online[4].

Fig. 5 shows the data that each agent accumulated during training. As can be seen here, they both follow the same patterns and end up at almost the same spot. Which levels were the same is also not hard to spot in these as the patterns are almost the same for each level in the two different agents. The baseline ANN does get slightly more points in total compared to the CBR-ANN hybrid during training though.

The two figures (Fig. 6 and Fig. 7) depict the accumulated rewards of the two agents during testing. Fig. 6 shows the data with no random actions, whilst Fig. 7 shows the data with 0.1 chance of random actions.

Unlike the training data there is a big difference in the two agents when testing. The patterns still seem similar for the data, but the CBR-ANN hybrid collects almost twice as many points with the 0.1 chance of random actions. It also gets more points with no random actions, but the difference isn't as big.

[3] https://github.com/imaltont/MasterCode/tree/paper-code
[4] https://www.youtube.com/playlist?list=PL76UkDpbLSScNASm93c5WTFeVmjT6aB1T

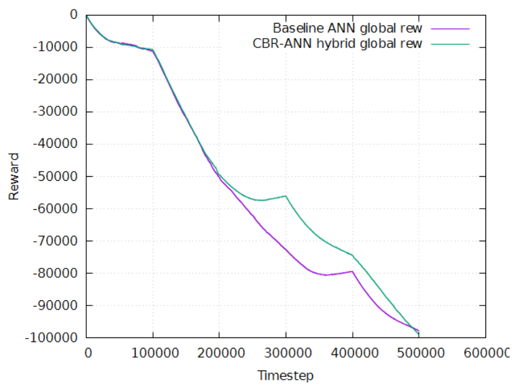


Fig. 5. Data collected from the agents while training.

The reason can be seen in the previously linked videos. Both agents try to just hold right and jump for the most part, but they didn't seem to learn that you need to release jump sometimes to actually get a jump from it.

The performance with no random actions was therefore not very different in the two agents, with the exception of *Quick man's stage* with no random actions, as the baseline ANN still did the same behaviour as it did in all the other levels, while the CBR-ANN hybrid still remembered what it did during training, and did the right/left pattern needed to get slightly further.

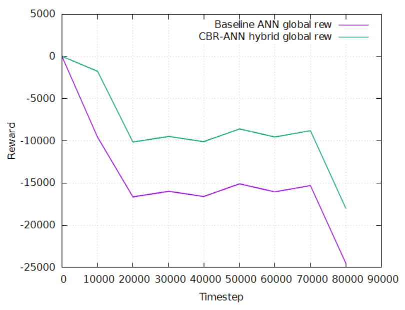


Fig. 6. No random actions

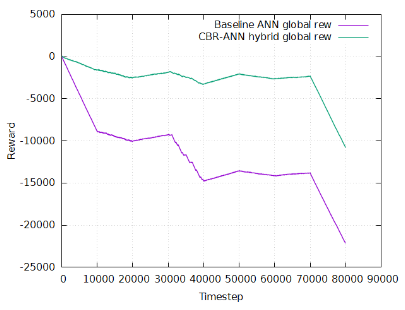


Fig. 7. 0.1 chance of random actions

With the 0.1 chance of random actions there were bigger differences between the two. When the agents got to have random actions intercept the holding jump and right action that it got stuck in in the beginning of the stages, the CBR-ANN agent showed a lot more advanced behaviour later in the levels compared to the baseline ANN. With different inputs from just the beginning, the CBR-ANN agent could still remember the less seen instances where it might have to e.g. do big jumps (*Metal man*) and press down (*Wood man*) to advance further. Even with this though, neither agent made it really far into the levels with the given time to train.

6 Discussion and future work

6.1 Results

The results for the CBR-ANN hybrid is rather disappointing, as it did not get far into the levels, and it was still in negative points in total over all the levels, though it did gain points in some of them.

However, when compared to the baseline ANN it starts getting interesting. They had the same pattern in all the levels during training, suggesting that they managed to learn the same patterns for the different levels. During testing however, the CBR-ANN hybrid outperformed the baseline ANN by a lot, simply by not forgetting the behaviour patterns of the previous levels.

The difference becomes even more visible when the chance for a random action is added, as it slightly worked against what seemed to be either not enough training to learn that it needed to alternate between pressing the jump button and releasing it to jump, or a case of overestimation. The random actions worked for both the agents, but the baseline ANN did not remember the more advanced movement later in the levels.

Their performance on the unseen cases was however very similar. Without the random chance, the main difference was the *Quick man's stage*, while the others got stuck at the same spot for both the CBR-ANN hybrid and the baseline ANN agent.

As for the research question presented in section 1, the CBR-ANN hybrid seems to be performing well compared to an established method like a single neural network.

It performs the same for unseen cases in this experiment, and outperforms the single ANN in most cases that both have already seen. The results are however very preliminary, and needs more research to be able to state that it is better where applicable with confidence.

Some ways to make both the results of this experiment stronger, as well as possible extensions to the dataset and the model will be discussed in the following subsection.

6.2 Future work

Both the model and the problem, as well as just having more data would be beneficial for this, therefore this subsection will discuss some ways that research could be continued on the CBR-ANN hybrid.

The simplest way to extend it could be to have more runs to compare with, instead of just a single play through, as well as comparing to more methods than just against an ANN. Tuning parameters more, especially the training time and amount of training examples seen, could prove to improve the results. Other parameters for the model itself could also be tuned, e.g. the number of hidden nodes or the number of recurrent nodes. Another rather simple way to improve the system could be to have a smarter random chance, so that it gets to experience being stuck more often, and just gradually add more random actions if it can't find a way out.

Extending the number of cases should better demonstrate the differences between this system and other methods it could be compared to. Two ways to possibly do this could be to have each level divided into several cases instead of just one case per level, as well as starting to use the powerups. Using the powerups should be the easiest way to extend the dataset, as you can still keep the similarity measure relatively simple. This because you can keep the same type of input, just add one for which levels has been beaten beforehand rather than just which level is currently being played. It does however require some changes to make the agent able to change which powerup it is using, which is not possible with the current setup. With the dataset becoming bigger, it could also be beneficial to use some form of CBR-maintenance [8]. In this case, grouping cases together could also be beneficial, having one solution answer to a group of cases rather thanjust a single case, if possible.

A problem that could arise when using a bigger amount of cases than this is that some of them could potentially learn something that is useful in the other cases. The different solutions to cases however does not communicate in any way between each other, and might therefore the other cases will not learn this, even if another case can do this. It did not become an issue in this rather small experiment. Another challenge for future research could be to make the system change when to be in the CBR section of the system and when it should be selecting actions for the ANN part. Some possible extensions could be to train up a neural network on when it should switch between the two systems, or some other image recognition system to make the decision. This could however extend the training period by a lot, if you need to train this up either as the system is learning, or beforehand. It could potentially allow for more advanced behaviors compared to the more static approach used in this paper.

Improvements could also be made by changing the model with new methods/strategies. One possibility could be to change the similarity measure for the CBR section of the system to something like an ANN, which has shown good results previously [7,9]. Another possibility to improve the CBR section with bigger changes could be to abuse the CBR properties even more, with e.g. being able to have different reward functions for the Q-learning part or different ANN topologies, depending on the properties of the case. This could potentially make it vastly superior to a model that uses only a single reward function/topology, or one that forgets what was good in the previous cases it has seen because of the updated reward function. Another possible change could be to pre-train neural network(s) to be used as the base networks for some cases, through supervised learning. The neural networks and the Q-learning could also see some changes, independent of the CBR part of the system. Adding convolutional layers to the neural networks is one such possibility, that could improve performance [11]. Another change that should be beneficial to the performance is to adapt the dueling Q-learning approach rather than the double deep Q-learning approach used in this paper, which should fight overestimation better [16]. A more radical change could be to abolish the Q-learning approach entirely and instead use another approach to learning instead. One such possibility is to use an evolutionary algorithm, like NEAT, which was used in the work that inspired the proposed method in the paper [2]. Lastly a way to combat the agent forgetting previous experiences would be beneficial [14], both to the system itself and for creating stronger evidence by improving the methods it is compared against.

7 Conclusion

The experiment yields good results for the CBR-ANN compared to the baseline ANN, but it still needs more work and research to be able to compete properly. The system can still be improved by a lot with both changing parameters and problems and with bigger changes to the model itself. The evidence shown is not very strong, but it is promising for future research on the model.

References

1. Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI communications **7**(1), 39–59 (1994)
2. Bling, S.: Mari/o - machine learning for video games. https://www.youtube.com/watch?v=qv6UVOQ0F44 (2015), [Online; accessed 25-February-2019]
3. De Paz, J.F., Bajo, J., González, A., Rodríguez, S., Corchado, J.M.: Combining case-based reasoning systems and support vector regression to evaluate the atmosphere–ocean interaction. Knowledge and information systems **30**(1), 155–177 (2012)
4. Guo, Y., Hu, J., Peng, Y.: Research on cbr system based on data mining. Applied Soft Computing **11**(8), 5006–5014 (2011)
5. Henriet, J., Leni, P.E., Laurent, R., Roxin, A., Chebel-Morello, B., Salomon, M., Farah, J., Broggio, D., Franck, D., Makovicka, L.: Adapting numerical representations of lung contours using case-based reasoning and artificial neural networks. In: International Conference on Case-Based Reasoning. pp. 137–151. Springer (2012)
6. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation **9**(8), 1735–1780 (1997)
7. Liu, Y.c., et al.: Hybridization of cbr and numeric soft computing techniques for mining of scarce construction databases. Automation in Construction **15**(1), 33–46 (2006)
8. Lu, N., Lu, J., Zhang, G., De Mantaras, R.L.: A concept drift-tolerant case-base editing technique. Artificial Intelligence **230**, 108–133 (2016)
9. Ma, F., He, Y., Li, S., Chen, Y., Liang, S.: Research on case retrieval of case-based reasoning of motorcycle intelligent design. In: The Sixth International Symposium on Neural Networks (ISNN 2009). pp. 759–768. Springer (2009)
10. Mitchell, T.M.: Machine Learning. WCB/McGraw-Hill (1997)

11. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
12. Pinzón, C., De Paz, J.F., Bajo, J., Herrero, Á., Corchado, E.: Aiida-sql: an adaptive intelligent intrusion detector agent for detecting sql injection attacks. In: Hybrid Intelligent Systems (HIS), 2010 10th International Conference on. pp. 73–78. IEEE (2010)
13. Pinzón, C., de Paz, Y., Cano, R., Rubio, M.P.: An attack detection mechanism based on a distributed hierarchical multi-agent architecture for protecting databases. In: 7th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS 2009). pp. 246–255. Springer (2009)
14. Titsias, M.K., Schwarz, J., Matthews, A.G.d.G., Pascanu, R., Teh, Y.W.: Functional regularisation for continual learning using gaussian processes. arXiv preprint arXiv:1901.11356 (2019)
15. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double q-learning. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
16. Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N.: Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581 (2015)
17. Zehraoui, F., Kanawati, R., Salotti, S.: Casep2: Hybrid case-based reasoning system for sequence processing. In: European Conference on Case-Based Reasoning. pp. 449–463. Springer (2004)