The Relationship Between (Un)Fractured Problems and Division of Input Space

Danilo Vasconcellos Vargas
Kyushu University
Fukuoka, Japan
vargas@cig.ees.kyushu-u.ac.jp

Hirotaka Takano
Kyushu University
Fukuoka, Japan
takano@cig.ees.kyushu-u.ac.jp

Junichi Murata
Kyushu University
Fukuoka, Japan
murata@cig.ees.kyushu-u.ac.jp

ABSTRACT

Problems can be categorized as fractured or unfractured ones. A different set of characteristics are needed for learning algorithms to solve each of these two types of problems. However, the exact characteristics needed to solve each type are unclear. This article shows that the division of the input space is one of these characteristics. In other words, a study is presented showing that while fractured problems benefit from a finer division of the input space, unfractured problems benefit from a coarser division of input space. Many open questions still remains. And the article discusses two conjectures which can be used to solve fractured problems more easily.

Categories and Subject Descriptors

I.2.6 [**Artificial Intelligence**]: Learning

Keywords

Fractured Problems, Machine Learning, Reinforcement Learning, Self Organizing Classifiers, Novelty Organizing Classifiers

1. INTRODUCTION

Learning algorithms should impact how problems are solved in many disciplines in the following years. In principle, anything can be learned by a learning algorithm. They are specially useful to solve complex problems which are either too hard to code or too dynamic. But they can still provide better as well as novel solutions to old hardcoded problems, because they can experiment and discover solutions provided that an environment with a fitness function and a simulation (or trial and error environment) are given.

However, state of the art learning algorithms need most of the time many adjustments in their parameters to work on different environments, requiring a specialist to enable its application. To make things worse, there are often some problems which do not work well no matter what parameters are chosen. For example, NeuroEvolution of Augmenting Topologies (NEAT) was shown to behave poorly on fractured problems [19]. The design of algorithms which can learn and adapt to widely different problems is a difficult challenge. But as important as the development of new algorithms is the study of which characteristics are necessary to solve a given class of problems.

There are many characteristics that help a method to solve fractured problems. Basically, the addition of local refinement was shown to improve the quality of solutions for the NEAT algorithm. In other words, by either adding local processing nodes (such as the use of radial basis function neural networks) [17] or adding constraints on the search focusing on topologies that do local refinement (this can be achieved by using an add cascade node mutation and freezing the past hidden nodes similar to the original cascade correlation algorithm) [18], NEAT was able to improve the performance on some fractured problems.

The division of the input space characteristic can theoretically add local refinement to the model. Not only that, it also facilitates the solution of the problem by a divide and conquer process. Thus, here we present a study of (un)fractured problems by varying the amount of division of the input space with the type of problem. A relationship is discovered between (un)fractured problems and the input space division. For fractured problems a finer division of the input space means a better quality of result, while the inverse is true for unfractured problems. This hints at why NeuroEvolution (NE) methods suffer with fractured problems while Learning Classifier Systems (LCS) do not. Naturally, there is a trade-off between LCS and NE. And we are not in any way arguing that an approach is better than the other.

Many fractured problems, although challenging and exponentially difficult with the increase of dimensions, still have patterns that once understood enable the solution of the same problem with an arbitrary dimension easily. Moreover, sometimes memorizing the input-output map is easier and less error prone than trying to represent it. These open questions and possible solutions are pointed out and discussed.

For the study the Novelty Organizing Team of Classifiers (NOTC) algorithm is used [36, 37]. Therefore, a brief overview of the method as well as the definition of fractured problems are provided.

GECCO'15 Companion, July 11–15, 2015, Madrid, Spain

ISBN 978-1-4503-3488-4/15/07…$15.00

DOI: http://dx.doi.org/10.1145/2739482.2768447

2. EVOLUTIONARY LEARNING SYSTEMS

Evolutionary learning systems have mainly three lines of research: Genetic Programming (GP) [20], LCS [12, 13] and NE [10]. These three lines of research are in no way mutually exclusive, with many algorithms using more than one line of research. Here we give a brief review of LCS and NE.

LCSs solve problems with a divide and conquer approach. They are usually composed of agents with condition-action-prediction rules that cooperate and compete in an evolutionary system [12, 13]. In each agent, the condition part of the rule divides the input space, creating smaller problems, therefore the action portion of the rule faces a simplified problem that is only a part of the total. The LCS's literature is reviewed in [31, 22].

LCSs with continuous actions were applied to function approximation using the classic XCSF algorithm [39, 7, 29]. Some LCS variations such as fuzzy variations [32, 6, 9] neural-based LCS [5, 6] and genetic programming-based algorithms [15] were also applied to function approximation. XCS with memory also solved sequence labeling problems [24].

Some reinforcement learning problems were solved by discrete action LCSs such as complex mazes [21], cart-pole balancing [30, 3] and the two-actions mountain car [23] problems. Recently, continuous action reinforcement learning problems were also solved by LCSs such as robotic arm control problems [26, 8], navigation problems [4, 14], complex mazes [35, 34] and dynamical mazes [33]. The algorithm used in this article, NOTC, was applied to pole balancing and discrete action mountain car in [36] as well as noisy and dynamic continuous action problems [37].

Neuroevolution consists of algorithms that evolve both the structure and parameters of artificial neural networks. Therefore genetic operators either modify the structure (e.g., adding /deleting neurons, adding/removing connections) or vary the weights of connections. Some of the classic NE algorithms are: GNARL [1], NEAT [27], EANT [16] and EPNet [40]. Naturally, many extensions of these algorithms exist, such as the indirect encoding with the use of CPPN evolution called HyperNEAT [11], an improvement of EANT called EANT2 [25], etc.

3. (UN)FRACTURED PROBLEMS

In unfractured problems the correct outputs (actions in the reinforcement learning context) are similar to other nearby correct outputs (actions) (see Figure 1). When the problem is continuous the correct actions vary smoothly, for discrete problems the actions vary infrequently. Fractured problems are the opposite, with correct actions that vary strongly and frequently.

4. NOTC'S STRUCTURE

Here we will describe NOTC succintly, for an extensive description and explanation please refer to [37]. NOTC uses novelty map to break the problem and a multilayer perceptron to solve the pieces. Basically, its components are:

- A Novelty Map population;
- Subpopulations divided in two groups (best and novel);
- Individuals.

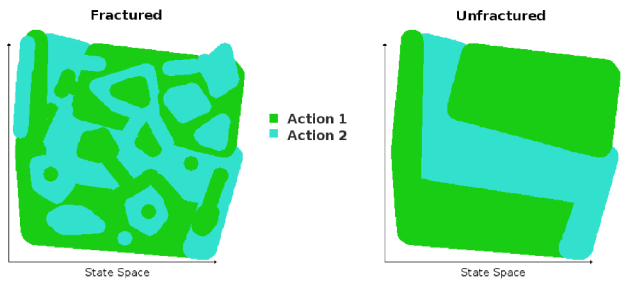

Figure 1: Example of fractured and unfractured problems.

4.1 Subpopulation

The subpopulation is a set of individuals divided in two groups called best and novel groups. Individuals of the best group are the ones that survived the last generation. On the other hand, individuals from the novel group were created in the last generation.

4.2 Individuals

Individuals are solutions to a problem piece, since they act on only a portion of the problem. Any computational model can be used for individuals. In this paper, feedforward neural networks with a single hidden layer containing a fixed number of neurons are used.

For the activation function, the hyperbolic tangent is used in the hidden layer neurons whereas the identity function is used in both input and output layer's neurons. The bias is absent in the input layer. As usual, the chromosome is a real number array that encodes the weights for each connection as well as the bias.

4.3 Novelty Map and Novelty Map population

The Novelty Map population is a Novelty Map where every cell is a subpopulation.

4.3.1 Novelty Map

In short, Novelty Map is a set of cells, where the weight array of each cell corresponds to the most novel individuals according to a given novelty metric. Similarly to the Self Organizing Map, when a new input is presented to the map, a competition takes place where the cell with the closest weight array wins. The winner cell can be used in many ways depending on the application (the novelty map population presents one way of using it). Afterwards, the set of cells is updated by substituting the weight array of the least novel cell (according to the novelty measure) with the input array if and only if the input array has higher novelty. The novelty metric used in this article is the uniqueness, where the novelty of an array is equal to the smallest distance from it to any of the other arrays.

4.3.2 Novelty Map Population

The Novelty Map Population is basically a Novelty Map where each cell stores a population of individuals, i.e., in addition to the cell's original weight array, subpopulations (see Section 4.1) are present in all cells of the Novelty Map. Moreover, when a new input is presented to the Novelty Map, the winner cell's subpopulation select one of its individuals to act on the environment.

5. NOTC'S BEHAVIOR

NOTC's behavior is based on two concepts:

- **Team** - A team is a set of individuals that act until the end of the episode. This concept enables the use of common fitness functions for Pittsburgh-style LCS in Michigan-style LCSs.
- **Hall of Fame** - Hall of Fame is a collection of the best teams (e.g., teams that received the highest accumulated rewards in reinforcement learning problems) evaluated so far.

5.1 Behavior

When an input is received, the behavior descripted in Figure 2 is executed. The number shown in the arrows inside the figure corresponds to a given step. In the following these steps will be explained in detail:

1. Novelty Map Population receives the input. Its cells compete for the input with the winning cell having one of its individuals chosen to act. When a given cell is first activated during an episode, an individual is chosen randomly to act. Afterwards, the same individual is chosen everytime this cell activates during this episode (notice that the constant use of the same individuals in every cell implements the concept of team in this algorithm and that the team changes every episode).
2. The chosen individual and its fitness compose the action set.
3. The chosen individual's neural network is activated, outputting the action to be performed.
4. The individual that composed the previous action set has its fitness updated. The fitness update is done using the Widrow-Hoff rule [38]:

$$F = F + \eta(\hat{F} - F), \tag{1}$$

where η is the learning rate, F is the current fitness and $\hat{F}$ is a new fitness estimate. The fitness estimate of cell $cell$ and individual c which were activated at time $t-1$ is given by the following equation:

$$\hat{F}(c, cell)_{t-1} = R_{t-1} + \gamma \max_{c' \in cell'} \{F(c', cell')\}, \tag{2}$$

where R is the reward received, γ is the discount-factor and $\max_{c' \in cell'} \{F(c', cell')\}$ is the maximum fitness of individual c' inside the activated cell $cell'$ at the current cycle t.

5. The current team fitness accumulates the rewards received until the end of the trial.

NOTC's behavior has a single exception to the description above. After the evolution, the first trials are reserved for the teams in the Hall of Fame. Therefore, each of the teams in the Hall of Fame have a trial where it must act and have its fitness updated. This is important, otherwise a lucky team may stay for quite a long time as well as influence the evolution negatively. Notice that a team enters in the Hall of Fame if its fitness is better than the worst team in the Hall of Fame. In this case, the worst team in the Hall of Fame is replaced.

5.2 Evolution

When $evolution\_trigger$ number of trials happened, the evolution is triggered. The following equation defines the $evolution\_trigger$:

$$evolution\_trigger = S_{size} * \iota, \tag{3}$$

where S_{size} is the subpopulation size (best plus novel individuals) and ι is a parameter.

The evolution procedure consists of the following steps:

1. For each cell, the first half of the best individuals is filled by the individuals present in the Hall of Fame teams and the second half with the fittest individuals according to their individual fitness. When a don't care symbol is present in the Hall of Fame team, a random individual from the cell is used.
2. The remaining individuals are removed, resulting in an empty group of novel individuals.
3. New novel individuals are created by using the differential evolution genetic operator (DE operator) or indexing with a chance of 50% each. Therefore, for each novel individual a new individual is created with either one of the following:
   - Indexing - A random individual from the population is copied;
   - DE operator - Consider that the number of best and novel individuals are the same. The DE operator takes as base vector the best individual with the same index as the current novel individual to be created, in this way all best individuals will be used as base vectors of at least one novel individual. To build the DE's mutant vector, three random individuals from the entire population (i.e., any individual from any subpopulation) are selected. The resulting trial vector is stored as the new novel individual.

6. EXPERIMENTS

First the settings will be described, followed by the tests's results and discussions.

6.1 Settings

The parameters of NOTC are similar to the ones used in [36, 37], see Table 1 for details. All results are averaged over 30 runs and only the best result among 100 trials is plotted.

7. EXPERIMENT 1 - MULTIPLEXER

Multiplexer is a challenging fractured problem where the agent must learn to select the correct data bit for the output by looking at the value provided in the address bits. Here we use a very challenging problem setting with three bits of address and eight bits of data. Notice that NEAT reported less than 75% correct answers in an easier version of this problem with six bits of data and three bits of address [19].

Figure 3 shows surprising results from applying NOTC on the multiplexer problem. The more the algorithm divides the input space the better the results, i.e., the higher the number of cells from the Novelty Map the better the results. What justifies the need for more divisions of the input? And why does having a higher number of cells make it faster?

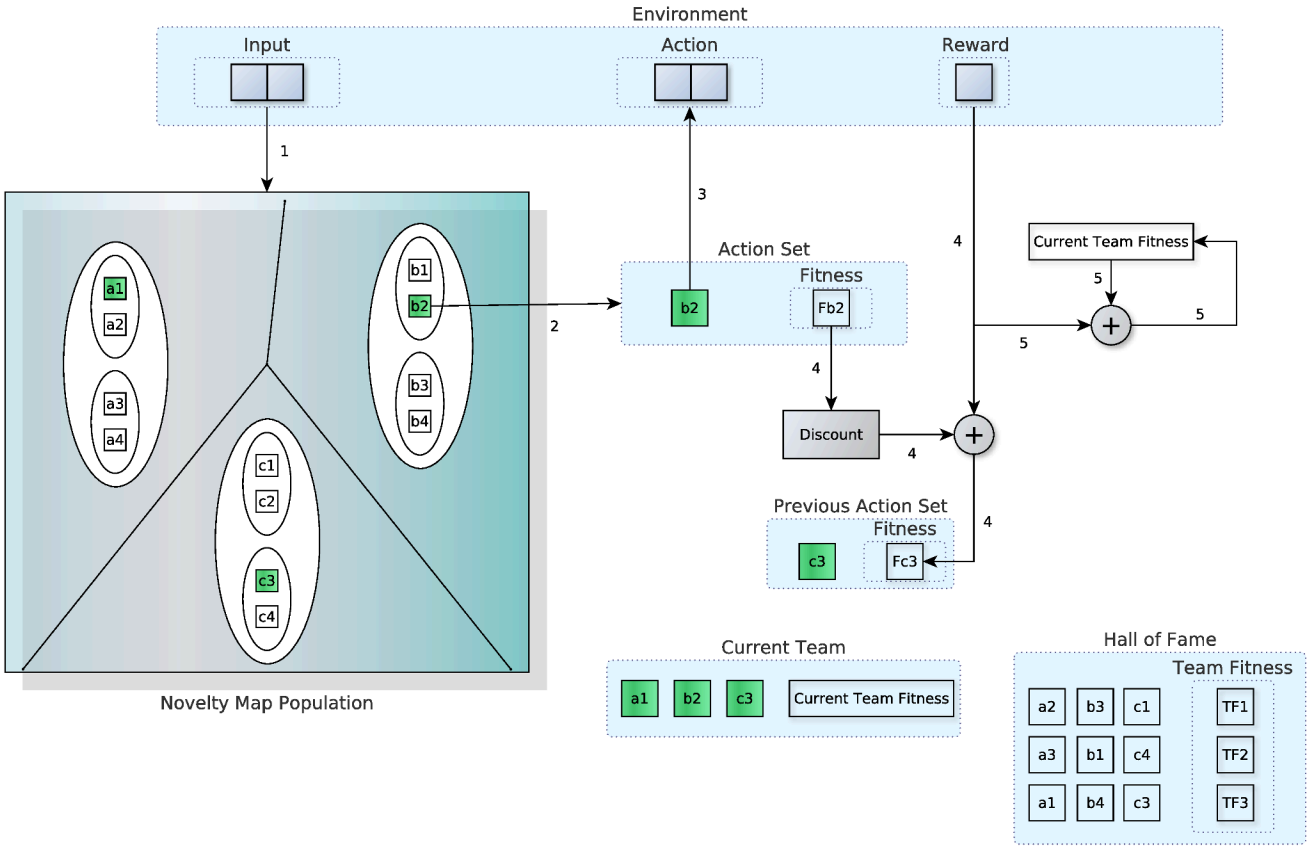

Figure 2: NOTC's behavior.

First, recall that neural networks learn by dividing the input space into linearly separated response regions [2]. The maximum number of regions that they can divide is related with the size and depth of the network. Therefore, when facing problems that need a huge ammount of divisions, neural networks with fixed size need the help of previous input space divisions. This explains the first question. Naturally, just increasing the size and depth of neural networks is not enough to enable them to solve such problems. Deep and/or big neural networks are harder to evolve, because the search space becomes too large and deceptive. Experiments not shown here reveal even worse results for evolving fixed sized big and/or deep neural networks.

Second, since all portions of the input space are always visited in this problem, it is not difficult to learn in all those regions. Moreover, the problems are simplified further with a fine division of the input space, turning the resulting smaller problems faster to learn.

8. EXPERIMENT 2 - CONTINUOUS ACTION MOUNTAIN CAR

The mountain car problem is a difficult unfractured problem where the agent can not climb the montain directly to the destination. It must instead learn to go up and down the mountain to have enough kinetic energy to climb it. It is defined by the following equations [28]:

$$
\begin{aligned}
&pos \in (-1.2, 0.6)\\
&v \in (-0.07, 0.07)\\
&a \in (-1, 1)\\
&v_{t+1} = v_t + (a_t) * 0.001 + \cos(3 * pos_t) * (-0.0025)\\
&pos_{t+1} = pos_t + v_{t+1},
\end{aligned}
\tag{4}
$$

Results on the mountain car shown in Figure 4 reveals that less cells in the Novelty Map is better. In other words, the less divisions in the input space the better, contrasting with the previous results on the multiplexer problem.This happens because mountain car is an unfractured problem and therefore dividing the input space into small pieces do not confer any benefits. On the contrary, the division may cause an abrupt change of action caused by a change in the acting agent. In this regard, it turns difficult to keep a smooth transition between divided regions of the input space, causing complications to appear on unfractured problems.

Another difference from this problem to the multiplexer one is that the difference in performance present in the three lines decreases with time. Therefore, the difference spotted is not a limit in the maximum attainable performance but rather a difference in the time required for convergence.

9. DISCUSSION

The division of the input space was important to improve

Table 1: Parameters for Novelty-Organizing Team of Classifiers

	Parameter	Value
Differential Evolution	CR	0.2
	F	random $\in [0.0, 2.0]$
Novelty Map	Number of Cells	variable
	Novelty Metric	Uniqueness
Novelty-Organizing Team of Classifiers	Widrow-hoff coefficient	0.1
	Number of best individuals	10
	Number of novel individuals	10
	ι	10
	Discount factor	0.99
	Initial fitness for novel individuals	-1
	Initial fitness for best individuals	0
	Number of hidden nodes	10
	Hall Of Fame's size	5

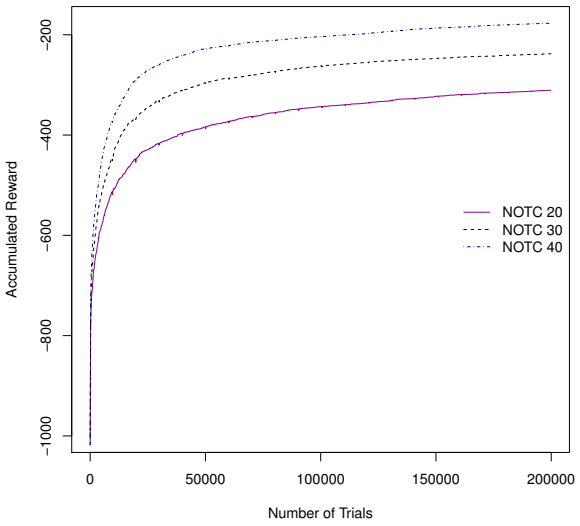

Figure 3: NOTC applied to a multiplexer problem with three bits of address and eight bits of data. NOTC 20, 30 and 40 have respectively 20, 30 and 40 cells in the Novelty Map. The reward is zero for correct answers and -1 for incorrect ones, with all the 2048 possibilities being evaluated for every episode.

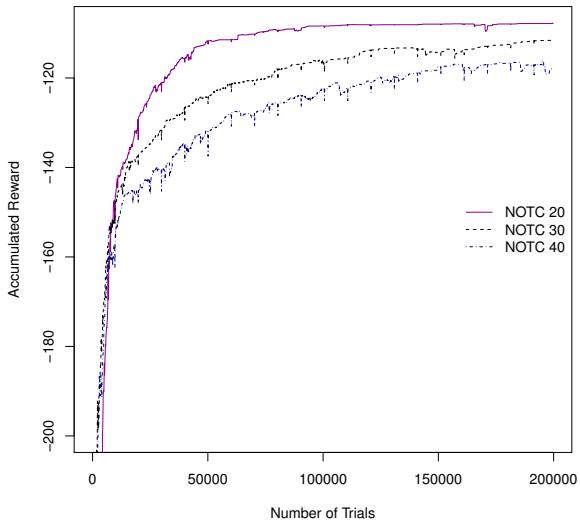

Figure 4: NOTC's performance on the mountain car problem. NOTC 20, 30 and 40 have respectively 20, 30 and 40 cells in the Novelty Map.

the quality of the solutions. With the Novelty Map, it is not hard to automatically adapt the division of the problem. The simple addition of cells to the Novelty Map create a finer division while the removal of cells make for a coarser one. Notice that Novelty Map can have cells added or removed on the fly without any problems. Moreover, the Novelty Maps differing in number of cells can be compared, offering a guidance to the best number of cells for a given problem.

Maybe another way around this problem would be to improve computational models of individuals. As mentioned before, merely increasing the complexity of the individuals does not help, since the search space becomes wider and harder to search. Therefore, improved learning algorithms specific to the computational model used may be required. However, even if they do solve with similar quality, the exponentially increasing difficulty of the multiplexer with the increase of dimensions might remain still.

Fractured problems are not just difficult problems, they also bring interesting questions that challenges not only algorithms but our way of thinking about them. We call attention for two points.

First point, some of the fractured problems, like the multiplexer, possess a basic pattern that once understood makes the problem trivial. This basic pattern also enables the solution of higher dimension versions of the same problem very easily.

Second point, for some of these fractured problems it is easier to memorize exhaustively the input-output map than to represent it into a computational model. Therefore, it raises the question of if a mixture of complementary computational models should be used.

10. CONCLUSIONS

To understand further the learning characteristics necessary to solve different classes of problems, here, a study over (un)fractured problems was conducted. We discussed the relationship between (un)fractured problems and the division of the input space. From the experiments. the following conclusions were drawn:

- Fractured problems benefit from a finer division of the input space, because classifiers such as neural networks need a higher number of response regions in these problems. Notice that merely increasing the number of response regions of a neural network by increasing its size or depth does not help, since the search space becomes too large and deceptive.

- Unfractured problems benefit from a coarser division of input space, because too much breaks in the input space cause responses to vary abruptly from one agent to another, slowing the learning rate.

This article also incentives further work into the remaining open questions regarding fractured problems. Complementary computational models or pattern discovery strategies are the two ideas proposed to face the difficulties of these types of problems.

11. ACKNOWLEDGMENTS

This work was supported in part by JSPS KAKENHI Grant Number 24560499.

12. REFERENCES

[1] P. J. Angeline, G. M. Saunders, and J. B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54–65, 1994.

[2] M. Anthony and P. L. Bartlett. *Neural network learning: Theoretical foundations*. cambridge university press, 2009.

[3] A. Bonarini. Evolutionary learning of fuzzy rules: competition and cooperation. In *Fuzzy Modelling*, pages 265–283. Springer, 1996.

[4] A. Bonarini, C. Bonacina, and M. Matteucci. Fuzzy and crisp representations of real-valued input for learning classifier systems. *Learning Classifier Systems*, pages 107–124, 2000.

[5] L. Bull. On using constructivism in neural classifier systems. *Parallel problem solving from nature-PPSN VII*, pages 558–567, 2002.

[6] L. Bull and T. O'Hara. Accuracy-based neuro and neuro-fuzzy classifier systems. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 905–911. Morgan Kaufmann Publishers Inc., 2002.

[7] M. Butz, P. Lanzi, and S. Wilson. Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction. *Evolutionary Computation, IEEE Transactions on*, 12(3):355–376, 2008.

[8] M. V. Butz and O. Herbort. Context-dependent predictions and cognitive arm control with xcsf. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1357–1364. ACM, 2008.

[9] J. Casillas, B. Carse, and L. Bull. Fuzzy-XCS: A michigan genetic fuzzy system. *Fuzzy Systems, IEEE Transactions on*, 15(4):536–550, 2007.

[10] D. Floreano, P. Dürr, and C. Mattiussi. Neuroevolution: from architectures to learning. *Evolutionary Intelligence*, 1(1):47–62, 2008.

[11] J. Gauci and K. Stanley. Generating large-scale neural networks through discovering geometric regularities. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 997–1004. ACM, 2007.

[12] J. H. Holland and J. S. Reitman. Cognitive systems based on adaptive algorithms. *ACM SIGART Bulletin*, (63):49–49, 1977.

[13] J. H. Holmes, P. L. Lanzi, W. Stolzmann, and S. W. Wilson. Learning classifier systems: New models, successful applications. *Information Processing Letters*, 82(1):23–30, 2002.

[14] G. Howard, L. Bull, and P. Lanzi. Towards continuous actions in continuous space and time using self-adaptive constructivism in neural XCSF. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, pages 1219–1226. ACM, 2009.

[15] M. Iqbal, W. N. Browne, and M. Zhang. Xcsr with computed continuous action. In *AI 2012: Advances in Artificial Intelligence*, pages 350–361. Springer, 2012.

[16] Y. Kassahun and G. Sommer. Efficient reinforcement learning through evolutionary acquisition of neural topologies. In *In 13th European Symposium on Artificial Neural Networks (ESANN)*. Citeseer, 2005.

[17] N. Kohl and R. Miikkulainen. Evolving neural networks for fractured domains. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 1405–1412. ACM, 2008.

[18] N. Kohl and R. Miikkulainen. Evolving neural networks for strategic decision-making problems. 2009.

[19] N. Kohl and R. Miikkulainen. An integrated neuroevolutionary approach to reactive control and high-level strategy. *Evolutionary Computation, IEEE Transactions on*, 16(4):472–488, 2012.

[20] J. R. Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[21] P. Lanzi, D. Loiacono, S. Wilson, and D. Goldberg. XCS with computed prediction in multistep environments. In *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pages 1859–1866. ACM, 2005.

[22] P. Lanzi and R. Riolo. A roadmap to the last decade of learning classifier system research (from 1989 to 1999). *Learning Classifier Systems*, pages 33–61, 2000.

[23] P. L. Lanzi, D. Loiacono, S. W. Wilson, and D. E. Goldberg. Classifier prediction based on tile coding. In *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, pages 1497–1504. ACM, 2006.

[24] M. Nakata, T. Kovacs, and K. Takadama. Xcs-sl: a

rule-based genetic learning system for sequence labeling. *Evolutionary Intelligence*, pages 1–16, 2015.

[25] N. T. Siebel and G. Sommer. Evolutionary reinforcement learning of artificial neural networks. *International Journal of Hybrid Intelligent Systems*, 4(3):171–183, 2007.

[26] P. Stalph and M. Butz. Learning local linear jacobians for flexible and adaptive robot arm control. *Genetic programming and evolvable machines*, 13(2):137–157, 2012.

[27] K. Stanley and R. Miikkulainen. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127, 2002.

[28] R. S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems 8*, 1996.

[29] H. Tran, C. Sanza, Y. Duthen, and T. Nguyen. XCSF with computed continuous action. In *Genetic And Evolutionary Computation Conference: Proceedings of the 9 th annual conference on Genetic and evolutionary computation*, volume 7, pages 1861–1869, 2007.

[30] K. Twardowski. Credit Assignment for Pole Balancing with Learning Classifier Systems. pages 238–245.

[31] R. Urbanowicz and J. Moore. Learning classifier systems: a complete introduction, review, and roadmap. *Journal of Artificial Evolution and Applications*, 2009:1, 2009.

[32] M. Valenzuela-Rendón. The fuzzy classifier system: A classifier system for continuously varying variables. In *Proceedings of the Fourth International Conference on Genetic Algorithms pp346-353, Morgan Kaufmann I*, volume 991, pages 223–230, 1991.

[33] D. V. Vargas, H. Takano, and J. Murata. Continuous adaptive reinforcement learning with the evolution of self organizing classifiers. In *Development and Learning and Epigenetic Robotics (ICDL), 2013 IEEE Third Joint International Conference on*, pages 1–2. IEEE, 2013.

[34] D. V. Vargas, H. Takano, and J. Murata. Self organizing classifiers and niched fitness. In *Proceedings of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 1109–1116. ACM, 2013.

[35] D. V. Vargas, H. Takano, and J. Murata. Self organizing classifiers: first steps in structured evolutionary machine learning. *Evolutionary Intelligence*, 6(2):57–72, 2013.

[36] D. V. Vargas, H. Takano, and J. Murata. Novelty-organizing team of classifiers-a team-individual multi-objective approach to reinforcement learning. In *SICE Annual Conference (SICE), 2014 Proceedings of the*, pages 1785–1792. IEEE, 2014.

[37] D. V. Vargas, H. Takano, and J. Murata. Novelty-organizing team of classifiers in noisy and dynamic environments. In *Evolutionary Computation, 2015. CEC 2015. IEEE Congress on*. IEEE, 2015.

[38] B. Widrow and M. E. Hoff. Adaptive Switching Circuits. In *1960 IRE WESCON Convention Record, Part 4*, pages 96–104, New York, 1960. IRE.

[39] S. W. Wilson. Classifiers that approximate functions. *Natural Computing*, 1(2-3):211–234, 2002.

[40] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *Neural Networks, IEEE Transactions on*, 8(3):694–713, 1997.