Novelty-Organizing Team of Classifiers in Noisy and Dynamic Environments

Conference Paper · May 2015

DOI: 10.1109/CEC.2015.7257254

3 authors, including:

Danilo Vasconcellos Vargas
Kyushu University
30 PUBLICATIONS **130** CITATIONS
SEE PROFILE

Hirotaka Takano
Gifu University
54 PUBLICATIONS **111** CITATIONS
SEE PROFILE

Novelty-Organizing Team of Classifiers in Noisy and Dynamic Environments

Danilo Vasconcellos Vargas
Graduate School of Information Science and Electrical Engineering
Kyushu University
Fukuoka, Japan
Email: vargas@cig.ees.kyushu-u.ac.jp

Hirotaka Takano
Faculty of Information Science and Electrical Engineering
Kyushu University
Fukuoka, Japan
Email: takano@cig.ees.kyushu-u.ac.jp

Junichi Murata
Faculty of Information Science and Electrical Engineering
Kyushu University
Fukuoka, Japan
Email: murata@cig.ees.kyushu-u.ac.jp

***Abstract*—In the real world, the environment is constantly changing with the input variables under the effect of noise. However, few algorithms were shown to be able to work under those circumstances. Here, Novelty-Organizing Team of Classifiers (NOTC) is applied to the continuous action mountain car as well as two variations of it: a noisy mountain car and an unstable weather mountain car. These problems take respectively noise and change of problem dynamics into account. Moreover, NOTC is compared with NeuroEvolution of Augmenting Topologies (NEAT) in these problems, revealing a trade-off between the approaches. While NOTC achieves the best performance in all of the problems, NEAT needs less trials to converge. It is demonstrated that NOTC achieves better performance because of its division of the input space (creating easier problems). Unfortunately, this division of input space also requires a bit of time to bootstrap.**

I. Introduction

Everything in the real world is naturally dynamic and noisy. Although most of the systems employ some type of preprocessing to treat these type of problems, sometimes the preprocessing systems may face some unexpected new dynamics which they were not prepared for or the noise may suddenly change in type and amplitude. Actually, learning algorithms seems the most natural solution to these problems, since they were developed right from the beginning with the idea of adaptation. That is, all of them are in principle capable of learning new dynamics or noise variations of the problem on the fly.

In this paper, NOTC and NEAT are tested on problems that have noise and need a certain degree of adaptability. On one hand, NEAT is a promising direct encoding topology evolving neuroevolution algorithm with a complexification philosophy (the chromosome starts simple and gets complex over time) [1]. On the other hand, NOTC is a learning classifier system based algorithm with a divide and conquer philosophy, it currently evolves a fixed topology neural network with a direct encoding genome. NOTC has the following distinct features:

- Novelty Map Population - Experiments show that this type of population allows for better adaptation at the same time that it is not sensitive to noise;
- Dual (team-individual) Fitness - The dual fitness presents a way to join Michigan and Pittsburgh approaches, leveraging the benefits from both points of view;
- Hall of Fame - With the Hall of Fame it is possible to keep the best combination of individuals. This is important to join Michigan and Pittsburgh approaches.

NOTC was first proposed and superficially described in [2], with applications only to pole balancing and a discrete version of mountain car. Here we explain NOTC more deeply and apply it to continuous action mountain car, noisy mountain car and unstable weather mountain car (a problem requiring some level of adaptability). Moreover, a comparison between NOTC and NEAT is done in all of the problems, revealing a trade-off between the two approaches.

It is verified that NOTC achieves the best performance in all of the problems because of its ability to divide the input space creating smaller easier problems. NEAT, on the other hand, needs less trials to converge because it does not need to divide the input space as well as it only has one problem to solve.

II. Related Work

Allow us to divide the literature in two lines of thought:

- Divide and Conquer Approach - division of the problem into easier ones and the use of simple computational models to solve those easier problems;
- Complexification Approach - start with simple solutions, while increasing the complexity of solutions over time.

On one hand, Learning Classifier Systems (LCS) falls within the divide and conquer approach, where a set of agents with condition-action-prediction rules cooperate and compete in an evolutionary system to solve the problem at hand [3], [4]. The condition coded by each agent automatically divides the input space, creating smaller problems, therefore the coded solutions does not need to be complex. On the other hand, many evolutionary algorithms using variable length genomes fall within the complexification approach. For example, algorithms evolving both the topology and parameters of neural networks complexify the network over time [5].

In the following, there is a brief review of the LCS's and Neuroevolution's literature's related with this article. Here,

978-1-4799-7492-4/15/$31.00

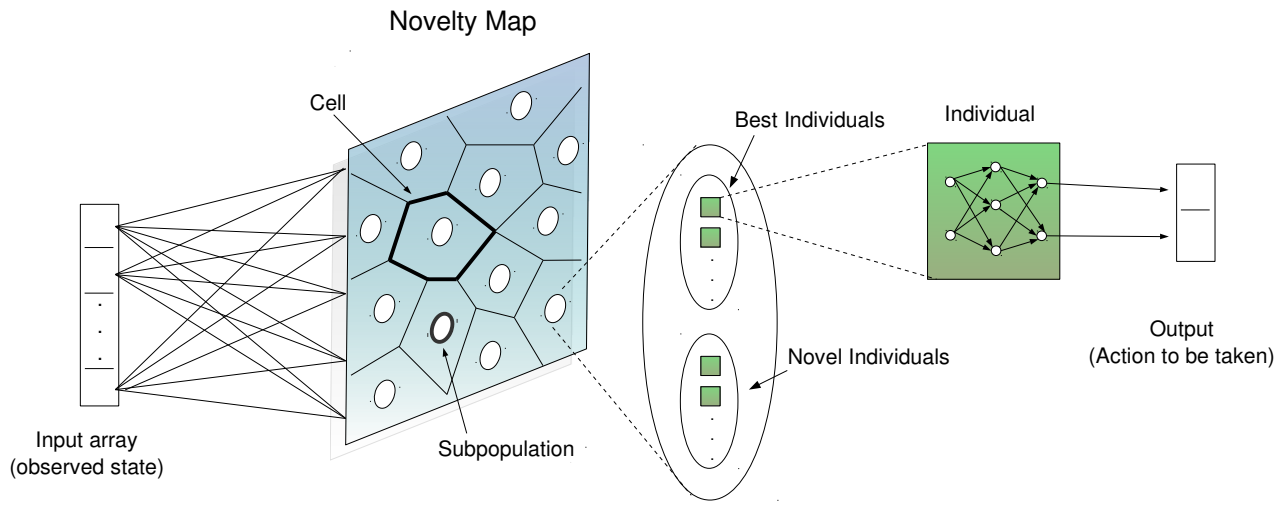

Fig. 1. NOTC's structure is illustrated. Notice that this is just an example, the size of the Novelty Map, number of inputs and so on were not taken into consideration.

we will restrain the review to only continuous action LCS algorithms and some of most salient Neuroevolution methods. For a complete review of both LCS's and Neuroevolution's literature please refer to [6], [7] and [5], respectively.

LCSs with continuous actions were applied to function approximation first with the XCSF algorithm [8]–[10], later with variants using fuzzy logic [11]–[13], neural-based LCS algorithms [12], [14] and genetic programming-based algorithms [15]. Regarding reinforcement learning problems, LCSs with discrete actions were used to solve complex mazes [16], cart-pole balancing [17], [18] and the two-actions mountain car [19] problems. Continuous action LCSs were applied to control robotic arms [20], [21], navigation problems [22], [23], complex mazes [24], [25] and dynamical mazes [26]. NOTC, specifically, was applied to pole balancing and a discrete action of mountain car in [2]. NOTC related algorithm without the concepts of team and dual fitness, Novelty-Organizing Classifiers (NOC), was applied to continuous mazes and classification problems [27].

Neuroevolution, where both the structure and parameters are evolved, is a relatively new research area. Therefore, there are fewer algorithms. To cite some: GNARL [28], NEAT [1], EANT [29] and EPNet [30].

III. NOTC'S STRUCTURE

In divide and conquer approaches, an algorithm has usually two distinct procedures, one for breaking the problem (divide procedure) and another to build the solution for the problem pieces (conquer procedure). NOTC uses novelty map as the divide procedure and multilayer perceptron as the conquer procedure.

The details of NOTC's structure is shown in Figure 1. Basically, its components are:

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

A. Subpopulation

The subpopulation is a set of individuals. It is divided in two groups (best and novel). The best and novel groups are useful to increase the diversity of the population and attain a good balance between exploration and exploitation. In one hand, individuals inside the best group are the best individuals, in the sense that they were already tested before (survived the last generation). On the other hand, individuals from the novel group were recently created (created in the last generation).

B. Individuals

Individuals can be any computational model. The individuals used in this paper are feedforward neural networks with a single hidden layer containing a fixed number of neurons.

Regarding the activation function, the hyperbolic tangent was used in the neurons from the hidden layer and the identity function was used in both input and output layer's neurons. The bias is absent in the input layer. Naturally, the chromosome encodes the weights for each connection as well as the bias.

C. Novelty Map and Novelty Map population

Before describing a Novelty Map population, it is necessary to detail solely the Novelty Map.

1) Novelty Map: Novelty measures, when used as fitness functions, allow algorithms to keep track of what they have already seen, i.e., they provide the stepping stones to reach the objective [31], [32]. Moreover, novelty can also be used to divide the space into points of interest, where each point is substantially different from each other.

Table I describes the algorithm. Basically, Novelty Map is a table with the most novel individuals according to a novelty metric. When a new input is presented to the map, a competition takes place where the cell with the closest weight array wins. This winner cell is activated and can be used in

many ways depending on the application (the novelty map population presents one way of using it). Afterwards, the table 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. This way the table is always kept up to date.

TABLE I. NOVELTY MAP ALGORITHM

Parameters:
1) Max_n: maximum size of the map;
2) Novelty metric;

Set the size of the map n to zero
Infinite Loop:
1) When an input is presented to the novelty map do:
2) If the size of the map n is smaller than Max_n
   a) Insert the input in the map
   b) Increment the size of the novelty map

   else
   a) Evaluate the input's novelty with the novelty metric
   b) If the input's novelty is higher than the lowest novelty from the samples inside the map
      i) Insert the input and remove the sample with the lowest novelty from the map
3) Return the weight array of the cell which is closest to the input

This table may share some similarities with the self-organizing map [33] or even the neural gas [34], but some important differences must be highlighted:

- Independence on Input Frequency - Both neural gas and self-organizing map (SOM) are sensitive to the frequency of the input, forgetting previous experience if the input starts to concentrate on a small portion of the spectrum. The novelty map does not have this disadvantage, always retaining even the most rare occurrences if they are novel enough.
- Cell's Efficiency - By ignoring the input frequency, fewer cells can be used to map the input space.

The novelty metric used in this article is the uniqueness. Let S be a set of arrays. The uniqueness is defined for an array a_i in relation to the other arrays in S with the following equation:

$$U = S \setminus \{a_i\} \tag{1}$$

$$uniqueness = min_{a_k \in U}(dist(a_i, a_k)). \tag{2}$$

In other words, uniqueness of an array is the smallest distance to the respective array for any array present in the set, excluding the array itself. This novelty metric was chosen because of its simplicity and quality though any other novelty measure could have been used instead.

2) Novelty Map Population: The Novelty Map population is very similar to the SOM population [24], [25]. The only difference is the exchange of the SOM dynamics to the Novelty Map one. As before, in addition to the cell's original weight array, subpopulations (see Section III-A) are present in all cells of the Novelty Map. The original dynamics of the Novelty Map happens when a new input is presented, i.e., the cell which is closest to the input wins and the Novelty Map is updated. Moreover, the winner cell and its subpopulation is used for some algorithm specific procedure. For example, in reinforcement learning problems, the winner cell's subpopulation have one of its individuals selected to act on the environment.

IV. NOTC'S BEHAVIOR

Before going into the details of NOTC's behavior, it is necessary to explain two concepts (team and hall of fame concepts).

A. Team

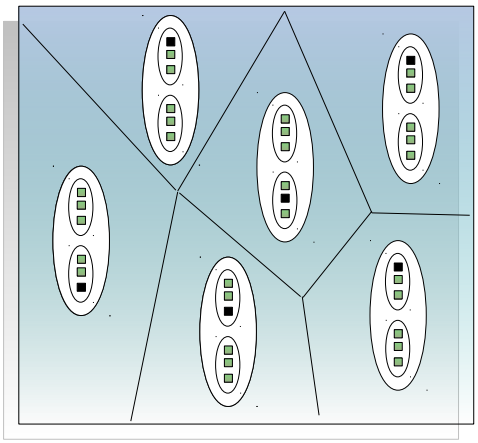

Fig. 2. Novelty Map Population with the black individuals (squares) forming a team. Once an individual from a certain cell is chosen to act in a trial, every time that cell is activated the same individual will be chosen to act. Suppose that all black squares are individuals which already acted in this trial. Therefore, they compose a team that is going to stay fixed until the end of the trial.

A reward is an evaluation of the last action and all of the actions that helped arrive in that last action. One way of thinking about the problem is to have a set of individuals that are activated depending on the state, and let them receive the reward directly after its action, as well as a percentage of the fitness from the individual that acts in the next step. Another way of thinking is to give the accumulated reward to all of the individuals that acted. In fact, these two are the main reasoning behind Michigan and Pittsburgh LCS approaches with their pros and cons associated. However they can be joined together if the team concept is used.

A team is a set of individuals each from a different cell in the Novelty Map Population (see Figure 2). When the cell is activated for the first time in the respective trial (i.e., the period from the start until the end of a run which is also called episode) an individual is chosen randomly. Afterwards, the same previous individual is chosen every time this cell is activated in the same trial. The team concepts is a consequence from this dynamic. When a cell was not activated in a given trial, no individual is selected to be part of the team and therefore a don't care symbol is stored instead.

B. Hall of Fame

Without a place to store the best teams, this information would be lost and good combinations of individuals would be forgotten. To prevent this, the hall of fame is created. Hall of fame is the set of teams that received the highest accumulated reward. Naturally, the accumulated reward is the sum of the rewards received by each individual in the given trial. In this article, the size of the hall of fame is fixed to half the number of best individuals in a cell.

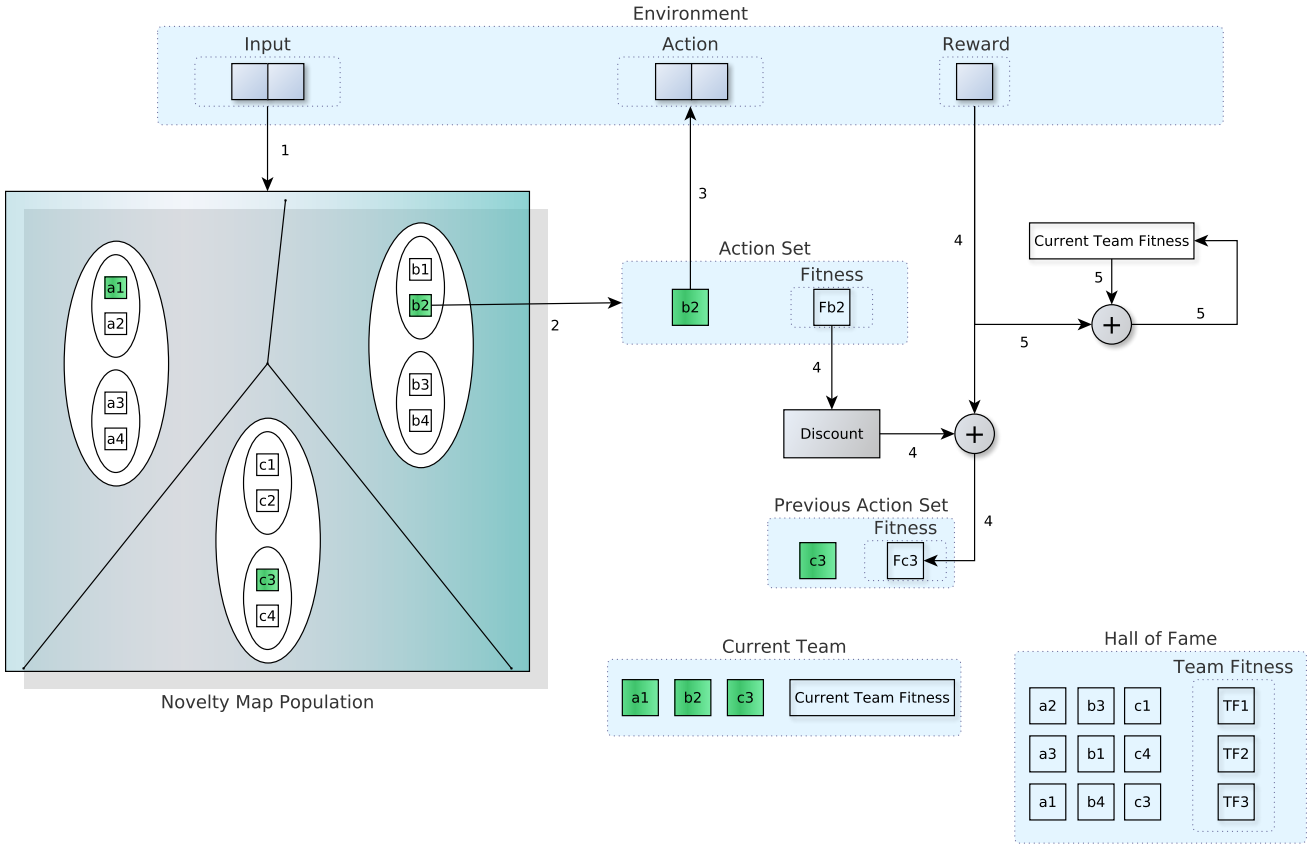

Fig. 3. NOTC's behavior.

C. Behavior

Figure 3 shows the NOTC's behavior. This behavior is triggered when an input is received and happens throughout the trials. The number showed in the arrows inside the figure correspond 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 (how one individual is chosen to act is explained in Section IV-A).
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 [35]:

$$F = F + \eta(\hat{F} - F), \tag{3}$$

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:

$$\hat{F}(c, cell)_{t-1} = R_{t-1} + \gamma \max_{c' \in cell'} \{F(c', cell')\}, \tag{4}$$

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

There is though a single exception to how NOTC behaves. 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 an 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.

D. Evolution

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

$$evolution\_trigger = S_{size} * \iota, \tag{5}$$

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. Sometimes an individual is included multiple times, because it is part of both the fittest individuals as well as part of the hall of fame. 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.

V. Experiments' Settings

In the following, experiments comparing NEAT with NOTC will be conducted in several variations of the mountain car problem. All results are averaged over 30 runs and only the best result among 100 trials is plotted.

The NEAT code used is the 1.2.1 version of the NEAT C++ software package [36]. Notice that although with different problem's settings (the initial position was randomized), NEAT was previously applied to a discrete action Mountain Car [37]. Therefore, both the settings used in that paper and the settings present in the original package were evaluated. In the end, the original package settings had better results, therefore the settings used for NEAT is the one provided with the software package (i.e. the same settings that was previously used in a double pole balancing task with success). The parameters for NEAT are written in Table II.

For NOTC, the settings are similar to the ones used in [2].

VI. Experiment 1 - Continuous Action Mountain Car

The mountain car problem [38] is shown in Figure 4. It is defined by the following equation:

$$\begin{aligned} &pos \in (-1.2, 0.6) \\ &v \in (-0.07, 0.07) \\ &a \in (-1, 1) \\ &v_{t+1} = v_t + (a_t) * 0.001 + \cos(3 * pos_t) * (-0.0025) \\ &pos_{t+1} = pos_t + v_{t+1}, \end{aligned} \tag{6}$$

where pos is the position of the car, a is the car's action and v is the velocity of the car. The starting velocity and position

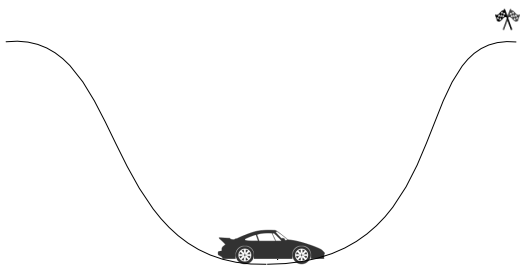

Fig. 4. Mountain car problem. The car's objective is to reach the flags uphill, although its acceleration is not enough to climb the mountain.

are respectively 0.0 and -0.5. If $v < 0$ and $pos \leq -1.2$, the velocity is set to zero. When the car reaches $pos \geq 0.6$ the trial is terminated and the algorithm receives 0 as reward. In all other positions the algorithm receive -1 as a reward. Moreover, if the algorithm's steps exceeds 10^3 the trial is terminated and the common reward of -1 is returned to the algorithm.

NOTC takes more trials to converge, but surpasses NEAT in performance (see Figure 5). The reasoning behind the better performance for NOTC lies in its ability of dividing the input space, creating smaller problem pieces that are easier to solve. In fact, the importance of dividing the input space is verified by comparing with a NOTC with only two cells in the Novelty Map (see Figure 6). This NOTC is called two cells NOTC. To make a fair comparison, the two cells NOTC has the same total number of individuals as the NOTC (i.e., both has the same initial diversity). Therefore, the number of best and novel individuals in the two cells NOTC was increased to 50.

For all experiments described in this paper, similar results were observed for their discrete action (-1,0 and 1) versions. Although the difference in the final performance between NEAT and NOTC was smaller.

VII. Experiment 2 - Continuous Action Mountain Car with Noise

In the real world, sensors are always affected by noise. To reflect this, a Gaussian noise is added to both the mountain car position and velocity every time they are read by the agent. The added Gaussian noise is respectively $\mu = 0$, $\sigma = 0.06$ and $\mu = 0$, $\sigma = 0.009$ for the position and velocity of the car, where μ stands for the mean and σ is the standard deviation.

Figure 7 shows the results. This result is very similar to the result from experiment 1, therefore the same observations made in experiment 1 can be made here.

VIII. Experiment 3 - Continuous Action Mountain Car with Unstable Weather

When driving a car, it is common for the weather to change from clear weather to rainy weather. In that moment, for safety purposes, the maximum velocity decreases. To reflect this, every 10000 trials the maximum velocity changes from the original to the $(-0.04, 0.04)$ range and vice-versa. The motivation behind this problem is to verify the capability of an algorithm to adapt to changes in the environment.

TABLE II. PARAMETERS FOR NEAT

Parameter	Value	Parameter	Value
trait_param_mut_prob	0.5	trait_mutation_power	1.0
linktrait_mut_sig	1.0	nodetrait_mut_sig	0.5
weigh_mut_power	2.5	recur_prob	0.00
disjoint_coeff	1.0	excess_coeff	1.0
mutdiff_coeff	0.4	compat_thresh	3.0
age_significance	1.0	survival_thresh	0.20
mutate_only_prob	0.25	mutate_random_trait_prob	0.1
mutate_link_trait_prob	0.1	mutate_node_trait_prob	0.1
mutate_link_weights_prob	0.9	mutate_toggle_enable_prob	0.00
mutate_gene_reenable_prob	0.000	mutate_add_node_prob	0.03
mutate_add_link_prob	0.05	interspecies_mate_rate	0.001
mate_multipoint_prob	0.6	mate_multipoint_avg_prob	0.4
mate_singlepoint_prob	0.0	mate_only_prob	0.2
recur_only_prob	0.0	pop_size	100
dropoff_age	15	newlink_tries	20
print_every	5	babies_stolen	0
num_runs	1		

TABLE III. 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	10
	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

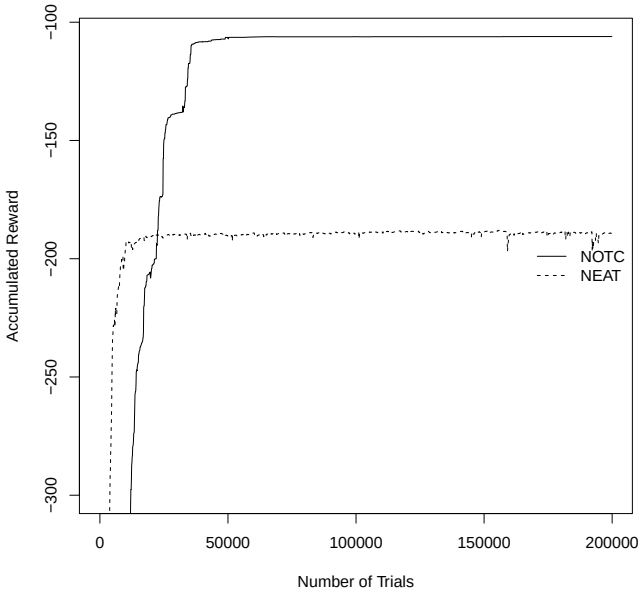

Fig. 5. Comparison of NOTC and NEAT in the continuous action mountain car problem.

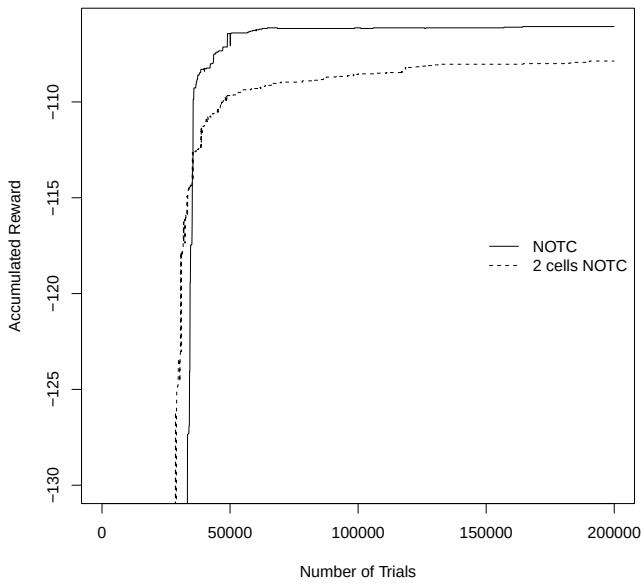

Fig. 6. Comparison of the original NOTC with a NOTC using a Novelty Map with only two cells in the continuous action mountain car problem.

The results shown in Figure 8 demonstrate that NOTC achieves a better performance in both problems (reduced velocity or not) when compared with NEAT. Moreover, NOTC has less variation of performance when the problem changes, which reveals that the solution found can easily change between both problems. NEAT, on the other hand, has a very abrupt curve when the problem changes.

IX. NOVELTY MAP ANALYSIS

Previous sections showed the behavior of the algorithm as a whole, but what can we say about the Novelty Map? How fast does it self-organizes? Is it always changing?

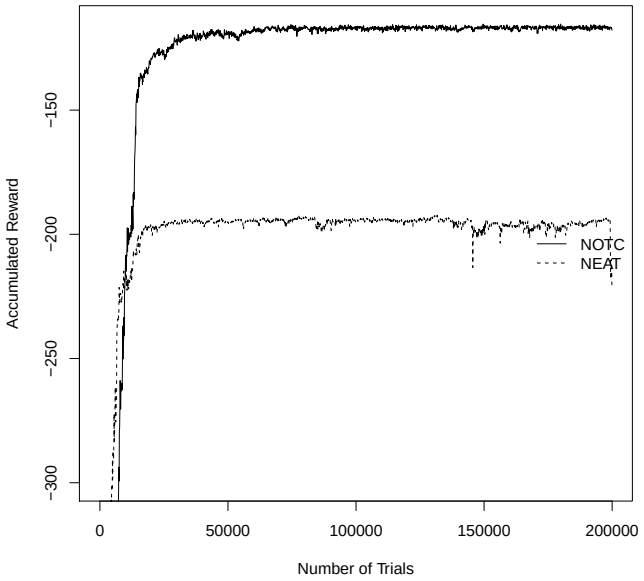

Fig. 7. Results of NOTC and NEAT when run on the noisy mountain car problem.

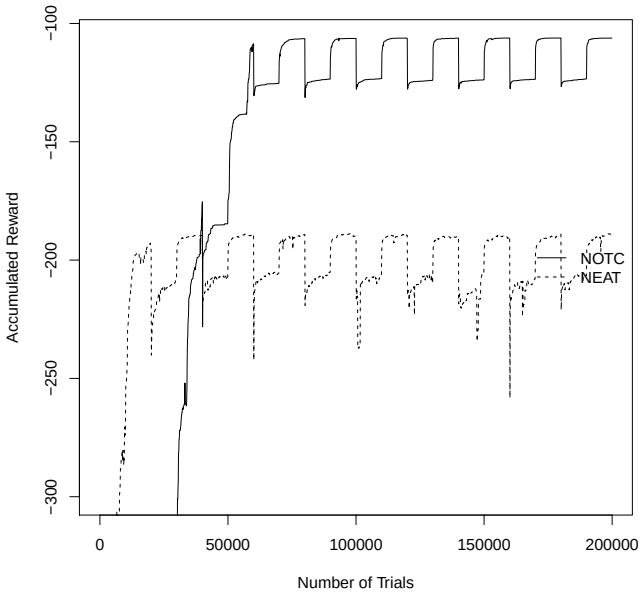

Fig. 8. NOTC and NEAT are compared in the unstable weather mountain car.

These questions can be answered by observing the number of times the value of cells are modified (updated) inside the Novelty Map (see Figure 9). The number of updates decreases with the number of trials faster than exponentially. Moreover, once the updates stop, the probability of another update appearing is very low. In other words, the division of the input space is fixed after some time. This fact allows for the evolution to focus on each of the smaller problems created. The additional time required for the Novelty Map to stop updating also explains partially the reason why NOTC needs more trials than NEAT. Naturally, this time also depends on the agent's exploration of the problem, the faster an agent explores the environment, the faster Novelty Map stops updating.

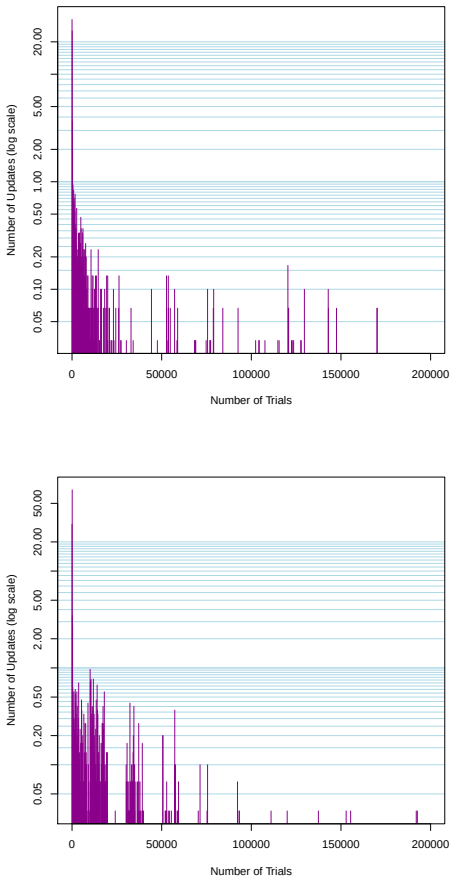

Fig. 9. Number of updates (changes in the value of cells) in the Novelty Map for the noisy mountain car (top) and unstable weather mountain car (below). The number of updates is in log scale.

X. Conclusion

In this article, NOTC was described in detail. Moreover, NOTC and NEAT were compared in a continuous action mountain car and two variations of it, one with noise and the other with the problem dynamics changing throughout the experiments. The experiments revealed a trade-off between the algorithms. NOTC achieved better performance in all of the problems, although NEAT needed less trials to converge. That is, despite the fact that NEAT evolves both the topology and parameters of the neural network, allowing for more robust and complex models, it was surpassed in performance by NOTC using a divide and conquer approach with a fixed topology neural network. The division of the input space was shown to be the reason why NOTC has a better performance. NOTC, however, takes more time to converge due to the additional time involved in learning the division of the problem.

Thus, the verified trade-off should be to some extent present when comparing a divide and conquer with a com-

plexification approach. Having said that, it should be noticed that both algorithms can improve, alleviating the trade-off. In special, the divide and conquer strategy was very far from its full potential, since the division was done over a space where the problem is non-separable (i.e. the input space for the mountain car was not enough to define a state space).

ACKNOWLEDGMENT

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

REFERENCES

[1] K. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[2] 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*. IEEE, 2014, pp. 1785–1792.

[3] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," *ACM SIGART Bulletin*, no. 63, pp. 49–49, 1977.

[4] J. H. Holmes, P. L. Lanzi, W. Stolzmann, and S. W. Wilson, "Learning classifier systems: New models, successful applications," *Information Processing Letters*, vol. 82, no. 1, pp. 23–30, 2002.

[5] D. Floreano, P. Dürr, and C. Mattiussi, "Neuroevolution: from architectures to learning," *Evolutionary Intelligence*, vol. 1, no. 1, pp. 47–62, 2008.

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

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

[8] S. W. Wilson, "Classifiers that approximate functions," *Natural Computing*, vol. 1, no. 2-3, pp. 211–234, 2002.

[9] M. Butz, P. Lanzi, and S. Wilson, "Function approximation with XCS: Hyperellipsoidal conditions, recursive least squares, and compaction," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 3, pp. 355–376, 2008.

[10] 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*, vol. 7, no. 11, 2007, pp. 1861–1869.

[11] 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*, vol. 991, 1991, pp. 223–230.

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

[13] J. Casillas, B. Carse, and L. Bull, "Fuzzy-XCS: A michigan genetic fuzzy system," *Fuzzy Systems, IEEE Transactions on*, vol. 15, no. 4, pp. 536–550, 2007.

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

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

[16] 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*. ACM, 2005, pp. 1859–1866.

[17] K. Twardowski, "Credit Assignment for Pole Balancing with Learning Classifier Systems," pp. 238–245.

[18] A. Bonarini, "Evolutionary learning of fuzzy rules: competition and cooperation," in *Fuzzy Modelling*. Springer, 1996, pp. 265–283.

[19] 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*. ACM, 2006, pp. 1497–1504.

[20] P. Stalph and M. Butz, "Learning local linear jacobians for flexible and adaptive robot arm control," *Genetic programming and evolvable machines*, vol. 13, no. 2, pp. 137–157, 2012.

[21] 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*. ACM, 2008, pp. 1357–1364.

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

[23] 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*. ACM, 2009, pp. 1219–1226.

[24] D. V. Vargas, H. Takano, and J. Murata, "Self organizing classifiers: first steps in structured evolutionary machine learning," *Evolutionary Intelligence*, vol. 6, no. 2, pp. 57–72, 2013.

[25] ——, "Self organizing classifiers and niched fitness," in *Proceedings of the fifteenth annual conference on Genetic and evolutionary computation conference*. ACM, 2013, pp. 1109–1116.

[26] ——, "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*. IEEE, 2013, pp. 1–2.

[27] ——, "Novelty-organizing classifiers applied to classification and reinforcement learning: towards flexible algorithms," in *Genetic and Evolutionary Computation Conference, GECCO '14, Vancouver, BC, Canada, July 12-16, 2014, Companion Material Proceedings*, 2014, pp. 81–82.

[28] P. J. Angeline, G. M. Saunders, and J. B. Pollack, "An evolutionary algorithm that constructs recurrent neural networks," *Neural Networks, IEEE Transactions on*, vol. 5, no. 1, pp. 54–65, 1994.

[29] 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.

[30] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *Neural Networks, IEEE Transactions on*, vol. 8, no. 3, pp. 694–713, 1997.

[31] E. Reehuis, M. Olhofer, M. Emmerich, B. Sendhoff, and T. Bäck, "Novelty and interestingness measures for design-space exploration," in *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*. ACM, 2013, pp. 1541–1548.

[32] J. Lehman and K. O. Stanley, "Abandoning objectives: Evolution through the search for novelty alone," *Evolutionary computation*, vol. 19, no. 2, pp. 189–223, 2011.

[33] T. Kohonen, *Self-organizing maps*. Springer, 2001, vol. 30.

[34] B. Fritzke *et al.*, "A growing neural gas network learns topologies," *Advances in neural information processing systems*, vol. 7, pp. 625–632, 1995.

[35] B. Widrow and M. E. Hoff, "Adaptive Switching Circuits," in *1960 IRE WESCON Convention Record, Part 4*. New York: IRE, 1960, pp. 96–104. [Online]. Available: http://isl-www.stanford.edu/~widrow/papers/c1960adaptiveswitching.pdf

[36] K. Stanley, I. Karpov, B. Erkin, and D. Thomas, "NEAT C++," http://nn.cs.utexas.edu/keyword?neat-c, 2001–2011.

[37] S. Whiteson and P. Stone, "Evolutionary function approximation for reinforcement learning," *The Journal of Machine Learning Research*, vol. 7, pp. 877–917, 2006.

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