Learning with Delayed Synaptic Plasticity

Anil Yaman
Eindhoven University of Technology
Eindhoven, the Netherlands
a.yaman@tue.nl

Giovanni Iacca
University of Trento
Trento, Italy
giovanni.iacca@unitn.it

Decebal Constantin Mocanu
Eindhoven University of Technology
Eindhoven, the Netherlands
d.c.mocanu@tue.nl

George Fletcher
Eindhoven University of Technology
Eindhoven, the Netherlands
g.h.l.fletcher@tue.nl

Mykola Pechenizkiy
Eindhoven University of Technology
Eindhoven, the Netherlands
m.pechenizkiy@tue.nl

ABSTRACT

The plasticity property of biological neural networks allows them to perform learning and optimize their behavior by changing their configuration. Inspired by biology, plasticity can be modeled in artificial neural networks by using Hebbian learning rules, i.e. rules that update synapses based on the neuron activations and reinforcement signals. However, the distal reward problem arises when the reinforcement signals are not available immediately after each network output to associate the neuron activations that contributed to receiving the reinforcement signal. In this work, we extend Hebbian plasticity rules to allow learning in distal reward cases. We propose the use of neuron activation traces (NATs) to provide additional data storage in each synapse to keep track of the activation of the neurons. Delayed reinforcement signals are provided after each episode relative to the networks' performance during the previous episode. We employ genetic algorithms to evolve delayed synaptic plasticity (DSP) rules and perform synaptic updates based on NATs and delayed reinforcement signals. We compare DSP with an analogous hill climbing algorithm that does not incorporate domain knowledge introduced with the NATs, and show that the synaptic updates performed by the DSP rules demonstrate more effective training performance relative to the HC algorithm.

CCS CONCEPTS

• **Theory of computation → Evolutionary algorithms**; **Bio-inspired optimization**;

KEYWORDS

Evolving plastic artificial neural networks, Hebbian learning, delayed plasticity, distal reward problem

1 INTRODUCTION

The plasticity property of biological neural networks enables the networks to change their configuration (i.e. *topology* and/or *connection weights*) and learn to perform certain tasks during their lifetime. The learning process involves searching through the possible configuration space of the networks until a configuration that achieves a satisfactory performance is reached. Modelling plasticity, or rather *evolving* it, has been a long-standing goal in Neuro-evolution (NE), a research field that aims to design artificial neural networks (ANNs) using evolutionary computing approaches [6, 12, 25].

Neuro-evolutionary methods can be divided roughly in *direct* and *indirect* encoding approaches [6]. In direct encoding, the topology and/or connection weights of the ANNs are directly represented within the genotype of the individuals. However, the number of possible network configurations (i.e. connectivity) increases exponentially depending on the number of neurons in a network. Therefore, it may be challenging to scale direct encoding approaches to large networks [14, 29]. In indirect encoding on the other hand, this drawback is overcome by encoding in the genotype rather than the network parameters *per se*, the rules to construct and/or optimize the ANNs, usually during their lifetime [12, 22]. Indirect encoding has also the additional advantage of being biologically plausible, as evidence has showed that biological neural networks undergo changes throughout their lifetime without the need to change the genes involved in the expression of the neural networks.

Among the indirect encoding approaches, evolving plastic artificial neural networks (EPANNs) [2, 22] implement plasticity by modifying the networks' configuration based on some plasticity rules which are activated throughout the networks' lifetime. These are encoded within the genotype of a population of individuals, in order to optimize the learning procedure by means of evolutionary computing approaches. One possible way of modelling plasticity rules in EPANNs is by means of *Hebbian learning*, a biologically plausible mechanism hypothesized by Hebb [9] to model synaptic plasticity [13]. According to this model, synaptic updates are performed based on local neuron activations. Previous works on EPANNs have implemented this concept by using evolutionary computing to optimize the coefficients of first-principle equations that describe the synaptic updates between pre- and post-synaptic neurons [2, 7, 16]. Others have used machine learning models (i.e.

ANNs) that can take the activations of pre- and post-synaptic neurons as inputs to compute the synaptic update as output [17, 18].

Reinforcement signals (and/or the output of other neurons, as in neuromodulation schemes [4, 20]) can be used as modulatory signals to guide the learning process by signaling how and when the plasticity rules are used. To allow learning, these signals are usually required right after each output of the network, to help associating the activation patterns of neurons with the output. If the reinforcement signals are available only after a certain period of time, but not immediately after each action step, it may not be possible to directly associate the behavior of the network to the activation patterns of the neurons. Thus, the *distal reward problem* [10, 23] arises, which cannot be addressed using Hebbian learning in its basic form, but requires a modified learning model to take into account the neuron activations over a certain period of time.

In this work, we propose a modified Hebbian learning mechanism, which we refer to as *delayed synaptic plasticity (DSP)*, for enabling plasticity in ANNs in tasks with distal rewards. We introduce the use of *neuron activation traces (NATs)*, i.e. additional data storage in each synapse that keep track of the average pre- and post-synaptic neuron activations. We use discrete DSP rules to perform synaptic updates based on the NATs and a reinforcement signal provided after a certain number of activation steps, that we refer to as an *episode*. The reinforcement signals are based on the performance of the agent relative to the previous episode (i.e. if the agent performs better/worse relative to the previous episode, a positive/negative reinforcement signal is provided). We further introduce competition in incoming synapses of neurons to stabilize delayed synaptic updates and encourage self-organization in the connectivity. As such, the proposed DSP scheme is a distributed and self-organized form of learning which does not require global information of the problem.

We employ *genetic algorithms (GA)* to evolve DSP rules to perform synaptic changes on *recurrent neural networks (RNNs)* that have to learn to navigate in a triple T-maze to reach a goal location. To test the robustness of the evolved DSP rules, we evaluate them for multiple trials with various goal positions. We then note how the process of training RNNs for a task using DSP can be seen as analogous to optimizing them using a single-solution metaheuristic, except for the fact that in contrast to general-purpose metaheuristics, the DSP rules take into account the (domain-specific) knowledge of the local neuron interactions for updating the synaptic weights. Therefore, to assess the effect of the domain knowledge introduced with DSP, we compare our results with a classic *hill climbing (HC)* algorithm. Our results show that DSP is highly effective in speeding up the optimization of RNNs relative to HC in terms of number of function evaluations needed to converge. On the other hand, the NATs data structure introduces an additional computational complexity.

The rest of the paper is organized as follows: in Section 2, we discuss Hebbian learning and the distal reward problem; in Section 3, we introduce our proposed approach for DSP and provide a detailed description of the evolutionary approach we used to optimize DSP; in Section 4, we present our experimental setup; in Section 5, we provide a comparison analysis of our proposed approach and the baseline HC algorithm; finally, in Section 6, we discuss our conclusions.

2 BACKGROUND

ANNs are computational models inspired by biological neural networks [3]. They consists of a number of *artificial neurons* arranged in a certain connectivity pattern. Adopting the terminology from biology, a directional connection between two neurons is referred to as a *synapse*, and the antecedent and subsequent neurons relative to a synapse are called *pre-synaptic* and *post-synaptic neurons*. The activation of a post-synaptic neuron a_i can be computed by using the following formula:

$$a_i = \psi\Big(\sum_j w_{i,j} \cdot a_j \Big) \tag{1}$$

where a_j is the activation of the j-th pre-synaptic neuron, $w_{i,j}$ is the synaptic efficiency between i-th and j-th neurons, and $\psi(\cdot)$ is an activation function (i.e. a sigmoid function [3]). The pre-synaptic neuron a_0 is usually set a constant value of 1.0 and referred to as the *bias*. Let us introduce also the equivalent matrix notation, that will be used in the following sections: considering that all the activations of pre- and post-synaptic neurons in layers k and l can be represented, respectively, as a column vectors A_k and A_l, then the activation of post-synaptic neurons can be calculated as: $A_l = \psi(W_{lk} \cdot A_k)$, where W_{lk} is a matrix that encodes all the synaptic weights.

2.1 Hebbian Learning

Hebbian learning is a biologically plausible learning model that performs synaptic updates based on local neuron activations [13]. In its general form, a synaptic weight $w_{i,j}$ at time t is updated using the *plain* Hebbian rule, i.e.:

$$w_{i,j}(t + 1) = w_{i,j}(t) + m(t) \cdot \Delta w_{i,j}(t) \tag{2}$$

$$\Delta w_{i,j}(t) = f\big(a_i(t), a_j(t)\big) = \eta \cdot a_i(t) \cdot a_j(t) \tag{3}$$

where η is the learning rate and $m(t)$ is the modulatory signal.

Using the plain Hebbian rule, the synaptic efficiency between neurons i and j, $w_{i,j}$ is strengthened/weakened when the sign of their activations are positively/negatively correlated, and does not change when there is no correlation. The modulatory signal $m(t)$ can alter the sign of Hebbian learning implementing *anti-Hebbian* learning if the modulatory signal is negative [1, 21].

The plain Hebbian rule may cause indefinite increase/decrease in the synaptic weights because strengthened/weakened synaptic weights increase/decrease neuron correlations, which in turn cause synaptic weights to be further strengthened/weakened. Several versions of the rule were suggested to improve its stability [27].

2.2 Evolving Plastic Artificial Neural Networks

Plasticity in EPANNs makes them capable of learning during their lifetime by changing their configuration [2, 15, 22]. Hebbian learning rules are often employed to model plasticity in EPANNs. On the other hand, the evolutionary aspect of EPANNs typically involve the use of evolutionary computing approaches to find a near-optimum Hebbian learning rules to perform synaptic updates.

In the literature, several authors suggested evolving the type and parameters of Hebbian learning rules using evolutionary computing. For instance, Floreano and Urzelai evolved four Hebbian learning rules and their parameters in an unsupervised setting where synaptic changes are performed periodically during the networks'

lifetime [7]. Niv *et al.,* [16] evolved the parameters of a Hebbian rule that defines a complex relation between the pre- and post-synaptic neuron activations on a reinforcement learning setting. Others suggested evolving complex machine learning models (i.e. ANNs) to replace Hebbian rules to perform synaptic changes [11, 19]. Orchard and Wang [17] compared evolved plasticity based on a parameterized Hebbian learning rule with evolved synaptic updates based on ANNs. However, they also included the initial weights of the synapses into the genotype of the individuals. Risi and Stanley [18] used *compositional pattern producing networks* (CPPN) [24] to perform synaptic updates based on the location of the connections. Tonelli and Mouret [26] investigated the learning capabilities of plastic ANNs evolved using different encoding schemes.

2.3 Distal Reward Problem

When reinforcement signals are available after a certain period of time, it may not be possible to associate the neuron activations that contributed to receiving the reinforcement signals. This is referred to as the *distal reward problem*, and has been studied in the context of time dependent plasticity [8, 10, 23].

From a biological viewpoint, *synaptic eligibility traces (SETs)* have been suggested as a plausible mechanism to trace the activations of neurons over a certain period of time [8] by means of chemicals present in the synapses. According to this mechanism, co-activations of neurons may trigger an increase in the SETs, which then decay over time. Therefore, their level can indicate a recent co-activation of neurons when a reinforcement signal is received, and as such be used for distal rewards.

3 METHODS

In this work, we focus on a navigation in triple T-maze environment (see Section 4) that requires memory capabilities. Thus, we use a RNN model illustrated in Figure 4b.

Our RNN model consist of input, hidden and output layers connected with four sets of connections. All the neuron activations are set to zero at the initial time step $t = 0$. Then the activation of the neurons in the hidden and output layers at each discrete time step $t + 1$, respectively $A_h(t + 1)$ and $A_o(t + 1)$, are updated as:

$$A_h(t + 1) = \psi\Big(W_{hi} \cdot A_i(t + 1) + \alpha_h \cdot W_h \cdot A_h(t)$$
$$+ \alpha_o \cdot W_{ho} \cdot A_o(t)\Big) \quad (4)$$

$$A_o(t + 1) = \psi\Big(W_{oh} \cdot A_h(t + 1)\Big) \quad (5)$$

where:
1) W_{hi} and W_{oh} are feed-forward connections between input-hidden and hidden-output layers, W_h is the recurrent connection of the hidden layer, and W_{ho} is the feedback connection from the output layer to hidden layer. The recurrent and feedback connections provide inputs of the activations of the hidden and output neurons from the previous time step. We do not allow self-recurrent connections in W_h (the diagonal elements of W_h equal to zero).
2) α_h and α_o are coefficient used to scale the recurrent and feedback inputs from the hidden and output layers respectively.
3) $\psi(\cdot)$ is a binary activation function defined as:

$$\psi(x) = \begin{cases} 1, & \text{if } x > 0; \\ 0, & \text{otherwise.} \end{cases} \quad (6)$$
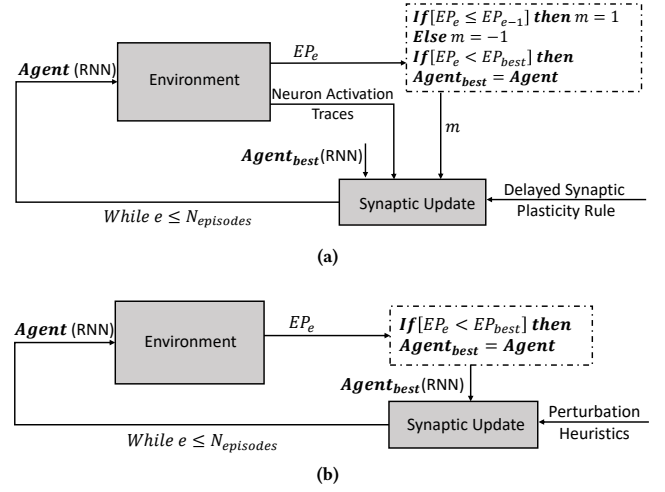

Figure 1: (a) The learning process by using the delayed synaptic plasticity, and (b) the learning process by optimizing the parameters of the RNNs using the hill climbing algorithm.

3.1 Learning

The learning process of an RNN is a search process that aims to find an optimal configuration of the network (i.e., its synaptic weights) that can map the sensory inputs to proper actions in order to achieve a given task. In this work, we use independently the proposed DSP rules, and the HC algorithm [3], to perform the learning the RNNs for the specific tripe T-maze navigation task, and compare their results. Both of these approaches perform synaptic updates based on the progress of the performance of an individual agent in consecutive episodes; however, the DSP rules incorporate the knowledge of the local neuron interactions, while HC uses a certain random perturbation heuristic that does not incorporate any knowledge. Here, we assume that the progress of the performance of an agent can be measured relative to its performance in a previous episode. We refer to the measure of the performance of an agent in a single episode as "episodic performance" (EP). We should note that we formalize our task as a minimization problem. Therefore, in our experiment an agent with a lower EP is better.

The illustration of the optimization processes of the RNNs using the DSP rules and the HC algorithm are given in Figures 1a and 1b respectively. Both algorithms run for a pre-defined number of episodes ($N_{episodes}$), starting from an RNN with randomly initialized synaptic weights, and record the best encountered RNN throughout the optimization process.

In the DSP-based algorithm illustrated in Figure 1a, the synaptic updates are performed after each episode —thus, the synaptic weights of the RNN during an episode are fixed— using DSP rules which take the RNN, NATs, and a modulatory signal as inputs. The NATs provide the average interactions of post- and pre-synaptic neurons during an episode. The structure of the NATs alongside with the DSP rules are explained in Section 3.2 in detail. The modulatory signal is used as a reinforcement which depends on the performance of the agent in the current episode relative to its

performance during the previous episode. If the current episodic performance EP_e is lower than the previous episodic performance EP_{e-1}, then the modulatory signal m is set to 1 (reward), otherwise m is set to −1 (punishment).

The HC algorithm that is illustrated in Figure 1b performs instead synaptic updates after each episode using a perturbation heuristic that does not assume any knowledge of the neuron activations. We use a Gaussian perturbation with a zero mean and unitary standard deviation and scale it with a parameter σ to perturb all the synaptic weights of the RNN of the best agent. The synaptic update procedure generates a new candidate **Agent** that is tested in the environment for the next episode and replaced with the best RNN if it performs better. Conventionally, in standard HC the measure of the performance of an agent in an episode would be called "fitness". However, in this work we refer to it as EP, to make it analogous to the algorithm that uses DSP rules.

3.2 Evolving Delayed Synaptic Plasticity

We propose delayed synaptic plasticity to allow synaptic changes based on the progress of the performance of an agent relative to its performance during the previous episode, i.e.:

$$\Delta w_{i,j}(e) = DSP(NAT_{i,j}, m, \theta) \tag{7}$$

$$w'_{i,j}(e) = w_{i,j}(e) + \eta \cdot \Delta w_{i,j}(e) \tag{8}$$

where the synaptic change $\Delta w_{i,j}(e)$ between neurons i and j after an episode e is computed based on their NATs, the modulatory signal m (which can either be 1 or −1, see Section 3.1), and a threshold θ that is used to convert NATs into binary vectors. The resulting DSP synaptic changes can be of three types (*decreased*, *stable*, or *increased*), i.e. at each time step $DSP(NAT_{i,j}, m, \theta) \in \{-1, 0, 1\}$. In Equation (8), the synaptic change is scaled with a learning rate η.

Subsequent to the synaptic updates of all synapses, the synaptic weights for the next episode $w_{i,j}(e + 1)$ are computed as:

$$w_{i,j}(e + 1) = \frac{w'_{i,j}(e)}{||\mathbf{w}'_i(e)||_2} \tag{9}$$

where $\mathbf{w}'_i(e)$ is a row vector encoding all incoming synaptic weights of a post-synaptic neuron i and $|| \cdot ||_2$ is the Euclidean norm. Here, the synaptic weights are scaled to have a unitary length, thus preventing an indefinite increase/decrease, and allowing self-organized competition between synapses [5]. Alternatively, a decay mechanism and/or saturation limits for weights can be introduced to prevent an indefinite increase/decrease of the weights [21].
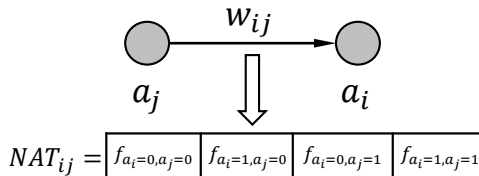

Figure 2: The neuron activation trace $NAT_{i,j}$ of the pre- and post-synaptic neuron activations a_j and a_i.

The NAT data structure is illustrated in Figure 2. It keeps track of the frequencies of the pre- and post-synaptic neuron activation

Table 1: A delayed synaptic plasticity rule visualized in a tabular form. Depending on a certain threshold θ for converting the binary forms of specific NATs, the DSP rule provides the synaptic changes $x_1, x_2, \ldots, x_{32}$ for all possible combinations of binary NATs and m states.

NAT^θ				m	Δw
00	**01**	**10**	**11**		
0	0	0	0	−1	x_1
0	0	0	0	1	x_2
...	...	...	...	...	...
1	1	1	1	1	x_{32}

states throughout an episode. We use four-dimensional vectors for each synapse —since there are four possible states for the activations of pre- and post-synaptic neurons— to store the number of times the pre- and post-synaptic neurons were in one of the following states: 00, 01, 10, and 11, where the first and second bits represent the pre- and post-synaptic neuron activations, and 0 and 1 represent nonactive and active states of neurons respectively. At the beginning of an episode, all NATs are initialized as zeros; and the end of an episode, they are divided by the number of total activations to convert them into frequencies.

The NATs are used in their binary forms to discretize the search space. Each frequency in a NAT is converted to either 0 or 1 based on a threshold θ (1 for the frequencies more than θ, and 0 for the ones that are below θ). Thus, a DSP rule, as illustrated in Table 1, is a combined form of 32 synaptic change decisions provided for all possible states of the binary forms of the NATs and the signal m.

We use genetic algorithms (GAs) to evolve three possible synaptic changes $\Delta w = \{-1, 0, 1\}$ (*decrease, stable, increase*) for each possible states of the NATs and m. In total, there are 32 possible states that can take three possible values. Thus, there is a total of 3^{32} number of possible DSP rules. In addition to the discrete part, we optimize four continuous parameters ($\eta, \theta, \alpha_h, \alpha_o$) by including them into the genotype of the individuals. We used suitable evolutionary operators separately for discrete and continuous variables. The details of the GA are provided in Section 4.3.

4 EXPERIMENTAL SETUP

In the following sections, we provide the details of our experimental setup including the test environment, the architecture of the agent, and the details of the GA.

4.1 Triple T-Maze Environment

A visual illustration of the triple T-maze environment is given in Figure 3. The triple T-maze environment consists of 29×29 cells. Each cell can be occupied by one of five possibilities: *empty, wall, goal, pit, agent*, color-coded in white, black, blue, green, red respectively. The starting position of the agent is illustrated in red. There are eight final positions, illustrated in blue or green. Among eight final positions, one of them is assigned as the goal position (in blue), and the rest as pits (in green). Starting from the initial position, an agent navigates the environment for a total of 100 action steps. To solve the task, it is required to find a set of
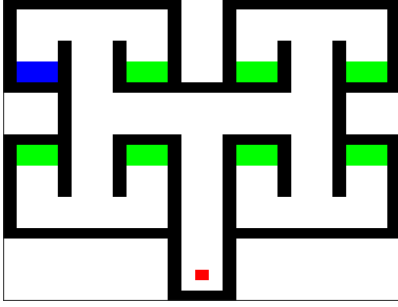
Figure 3: Triple T-maze environment. The walls, starting position of the agent, goal and pits are color-coded using black, red, blue and green.
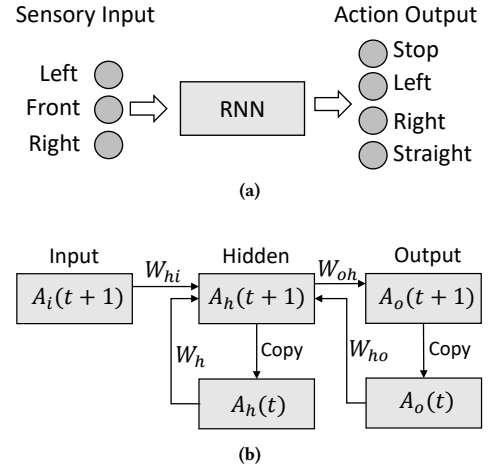

Figure 4: (a) The sensory inputs and action outputs of the recurrent neural networks and (b) the architecture of the networks that are used to control the agents.

connection weights of the recurrent neural network that can allow the agent to achieve the goal from the starting position.

4.2 Agent Architecture

An illustration of the architecture of the agents used for the triple T-maze environment is provided in Figure 4. An agent has a certain orientation in the environment, facing one of the four possible directions along the x and y axes of the environment (i.e. north, south, west, east), and can only move one cell at a time horizontally or vertically. It can take sensory inputs from the nearest cells from its left, front and right, depending on its orientation. We let the agent sense only an empty cell, or a wall (i.e., the agent is not aware of the presence of the goal or pits), so the sensory inputs are encoded using one bit, representing an empty cell with 0 and a wall with 1. Thus, there are three bits as input, that the agent uses to decide one of four possible actions: *stop*, *left*, *right*, and *straight*. In case of stop, the agent stays in the same cell with the same orientation for an action step; in cases of left and right, the agent changes its orientation accordingly and proceeds one cell straight; in case of straight, the agent keeps its original orientation and proceeds one cell straight. If the cell the agent wants to move in is occupied with wall, then the agent stays in its original position. We use RNNs to control the agents since our task requires memory capabilities, and may not be solved by a static architecture such as a feed-forward ANN. The recurrent and feedback connections between the hidden-hidden and output-hidden layers help agents to keep track of the sequences of actions they perform throughout an episode. The RNN networks used in all experiments consist of 20 hidden neurons unless otherwise is specified. Therefore, the networks consist of $(3 + 1) \times 20 = 80$ input to hidden connections, $20 \times 19 = 380$ hidden to hidden connections (except self), $(20 + 1) \times 4 = 84$ hidden to output connections, and $4 \times 20 = 80$ output to hidden connections, in total of 624 connections between all layers (+1 refers to the bias).

4.3 Genetic Algorithm

We use a standard GA to evolve DSP rules with the exception of a custom-designed mutation operator. The genotype of the DSP rules consists of 32 discrete (Δw for all possible states of the NATs

and m, see Section 3.2) and four continuous values ($\eta \in [0, 1], \theta \in [0, 1], \alpha_h \in [0, 1], \alpha_o \in [0, 1]$). Therefore, we encode a population of individual genotypes represented by 36-dimensional discrete/real-valued vectors.

The pseudo-code for the evaluation function is provided in Algorithm 1. To find the DSP rules that can robustly learn the triple T-maze navigation task independently of the goal position and starting from a random RNN state, each evaluation called during the evolutionary process consists of testing each rule for five trials for each of the eight possible goal positions (for total of 40 independent trials). The final positions (one goal in blue, seven pits in green) of the triple T-maze environment are shown in Figure 3. We switch the position of the goal with one of the pits to have eight distinct goal positions in total. For all distinct goal positions, the rest of the final positions are assigned as pits. Due to the computational complexity, the maximum number of episodes per trial is set to 100.

The EP of an agent in an episode is computed as follows:

$$EP = \begin{cases} steps(Agent), & \text{if the goal is reached;} \\ N_{steps} + d(XY(Agent), XY(g)), & \text{Otherwise.} \end{cases}$$

(10)

For each episode, the agent is given 100 action steps (N_{steps}) to reach the goal. We aim to minimize the number of action steps that the agents take to reach the goal (in case they do reach it); otherwise, we aim to minimize their distance to the goal. Thus, if the agent reaches the goal, the EP is the number of action steps the agent took to reach to the goal ($steps(Agent)$). Otherwise, the EP is $N_{steps} + d(XY(Agent), XY(g))$, that is the maximum number of total action steps, plus the distance between the final agent's position and the goal's position. The distance between the agent and the goal is found by the A* algorithm [30], which finds the closest path (distance) taking into account the obstacles. Additionally, the EP of an agent is increased by 5 every time the agent steps into a pit (recalling that the EP is minimized).

Finally, the fitness value (the lower the better) of a DSP rule is obtained by averaging EP_{best} found in each trial (i.e. for 5 trials and

---

Algorithm 1 Evaluation of an individual DSP rule

1: **procedure** Evaluate(x, $trials$, $N_{episodes}$)
2:     $fitness \leftarrow 0$
3:     **for each** $t \in trials$ **do**
4:         **for each** $g \in goalPositions$ **do**   ▷ N_{goals} positions
5:             $EP_0 \leftarrow inf$, $EP_{best} \leftarrow inf$, $e \leftarrow 1$
6:             $RNN \leftarrow initializeRandom$
7:             **while** $e \leq N_{episodes}$ **do**
8:                 $EP_e \leftarrow testNetwork(RNN)$
9:                 $m \leftarrow -1$
10:                 **if** $EP_e \leq EP_{e-1}$ **then**
11:                     $m \leftarrow 1$
12:                 **end if**
13:                 $EP_{best} \leftarrow min(EP_e, EP_{best})$
14:                 $RNN \leftarrow synapticUpdate(RNN, x, m)$
15:                 $e \leftarrow e + 1$
16:             **end while**
17:             $fitness \leftarrow fitness + EP_{best}$
18:         **end for**
19:     **end for**
20:     **return** $fitness / (trials \cdot N_{goals})$
21: **end procedure**

---

8 distinct goal positions, in total of 40 trials). We should note that the proposed design of the EP and fitness is chosen to introduce a gradient into the evaluation process of agents. For instance, the selection process is likely to prefer agents that on average reach the goal with a smaller number of action steps, and with the least number of interactions with pits.

To limit the runtime of the algorithm, we use a small population size, 14 individuals in total. We employ a *roulette wheel selection* operator with an elite number of four, which copies the four individuals with the highest fitness values to the next generation without any change. The rest of the individuals are selected with a probability proportional to their fitness values. We use *1-point crossover* operator with a probability of 0.5. We designed a custom *mutation* operator which re-samples each discrete dimension of the genotype with a probability of 0.15, and performs a Gaussian perturbation with zero mean and 0.1 standard deviation for the continuous parameters. We run the evolutionary process for 300 generations and finally select at the end of the evolutionary process the DSP rule that achieved the best fitness value.

4.4 Hill Climbing

We use the HC algorithm as a baseline to compare the results of the DSP. The HC algorithm, visualized in Figure 1b, is a single-solution local search algorithm [3]. It is analogous to the DSP, except the fact that it is used as a direct encoding optimization approach where domain knowledge (i.e. knowledge of the neuron activations) is not introduced into the optimization process. Therefore, it provides a suitable baseline to assess the effectiveness of the domain-based heuristic used in the DSP.

A trial of the HC algorithm aims to find an optimum set of RNN weights that allows the agent to reach a given goal position, starting from the starting position. All the connection weights of the RNN
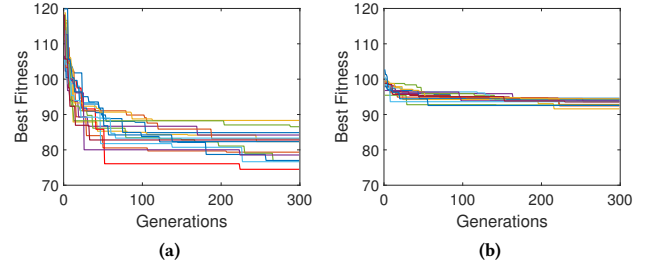

(a)   (b)

Figure 5: The change of the best fitness during 15 independent evolutionary processes for optimizing (a) the DSP rules and (b) the HC parameters. The y-axes of figures are scaled between 70–120 to allow a better visual comparison.

(644 in total) are directly encoded into a single real-valued vector that represents an ***Agent*** (candidate solution). At the beginning of the algorithm, each variable of the ***Agent*** is randomly initialized in the range $[-1, 1]$ with uniform probability. The ***Agent*** is then evaluated and assigned as ***Agent***$_{best}$, with fitness equal to EP_{best}. After the evaluation, ***Agent***$_{best}$ is perturbed using a *perturbation heuristic* to generate a new ***Agent*** as follows:

$$Agent = Agent_{best} + \sigma \cdot \mathcal{N}(0, 1) \qquad (11)$$

where $\mathcal{N}$ is a random vector whose length is the same as that of ***Agent***$_{best}$, and each dimension is independently sampled from a Gaussian distribution with zero mean and unitary standard deviation, scaled by the parameter σ.

The ***Agent*** is evaluated and if its EP_e is smaller than EP_{best}, ***Agent***$_{best}$ is replaced by the new ***Agent***. The perturbation and evaluation processes are performed iteratively for 100 episodes (evaluations). The overall performance of the HC algorithm is then obtained by averaging the EP_{best} from 40 trials (consisting of 5 trials for 8 distinct goal positions), that is the same evaluation procedure used to evaluate DSP rules.

The performance of the HC algorithm may depend on the parameter of the perturbation heuristic. Thus, to provide a fair comparison with DSP, we optimize the parameter $\sigma \in [0, 1]$ with respect to the hyper-parameters of the RNN model $\alpha_h \in [0, 1]$ and $\alpha_o \in [0, 1]$ by using a GA with the same settings used to optimize the continuous part of the DSP rules (see Section 4.3). We refer these three parameters as HC parameters.

5 RESULTS

Figures 5a and 5b show the change of the best fitness value during 15 independent evolutionary optimization processes of the DSP rules and the HC parameters respectively. We run all the experiments on a single-core Intel Xeon E5 3.5GHz computer; therefore, we fix the number of generations to 300, to keep the runtime of the algorithm reasonably limited. In each generation, the best fitness value obtained by the DSP rules and the HC algorithm with evolved parameters are shown. A complete list of the evolved DSP rules can be found on an extended version of this paper online[1].

---

[1] An extended version of the paper with complete list of evolved DSP rules can be found here: https://arxiv.org/abs/1903.09393.
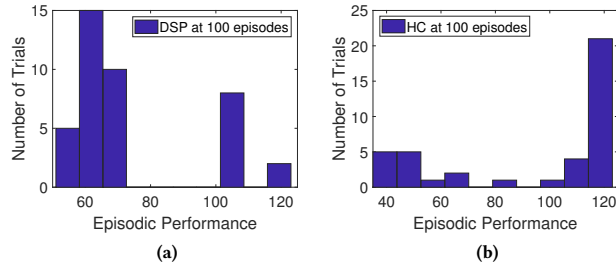
Figure 6: The distribution of the episodic performance of 40 trials using the best performing evolved DSP rule (a) and HC parameters (b) trained for 100 episodes.

The initial DSP rules obtain an average fitness of 113.79, with a standard deviation of 4.30; at the end of the evolutionary processes, they achieve an average fitness of 81.39, with a standard deviation of 4.02. On the other hand, the initial HC parameters obtain an average fitness value of 98.71, with a standard deviation of 1.64; and at the end of the evolutionary process, they achieve an average fitness of 93.50, with a standard deviation of 0.87. We used the Wilcoxon test to statistically assess the significance of the results produced by the evolutionary processes [28]. The null-hypothesis that the mean of the results produced by two processes are the same is rejected if the p-value is smaller than $\alpha = 0.05$. In our case, the results of the evolutionary processes of DSP rules are statistically different (better than the HC results) with a p-value of 3.3×10^{-6}.

The distribution of the episodic performance of the best evolved DSP rule and HC parameters are given in Figure 6. The trials with an episodic performance smaller than 100 indicate that the goal is achieved in that trial. Thus, 75% of the 40 trials reached the goal when the agents are trained with the DSP rules. On the other hand, only 35% of the trials reached the goal when the agents are trained using the HC. A Wilcoxon rank-sum test (calculated on the EPs) shows that the DSP rule is better with a p-value of 0.03. The results show that the training process with 100 episodes does not seem to be sufficient for the HC algorithm to provide results as good as the results provided by the DSP rules. Moreover, it may be possible to improve the success percentage of achieving the goal using the DSP rule by increasing the number of episodes. To test this, we separately tested the DSP rules and the HC with evolved parameters provided by the multiple runs of the GA on the same task settings with 10000 episodes. The results are given in Figure 7 where we show the change of the fitness values (average of 40 trials) w.r.t. the episode number during the training of the DSP rules and the HC.

The best DSP rule achieved fitness of 54.27 and 44.10, and the HC with the best parameters achieved fitness of 69.35 and 42.5 in 1000 and 10000 episodes respectively. The results indicate that the DSP rules converge at a better fitness value faster than the HC. However, when the number of episodes is increased (for 10000), the HC achieves slightly (a fitness value difference of 1.6) better performance than the best DSP rule. The p-values for the Wilcoxon tests for the results at episodes 1000 and 10000 are 0.02 and 0.3; thus, at 1000 episodes DSP rule is significantly better than HC, whereas at 10000 there is no significant difference between their results.
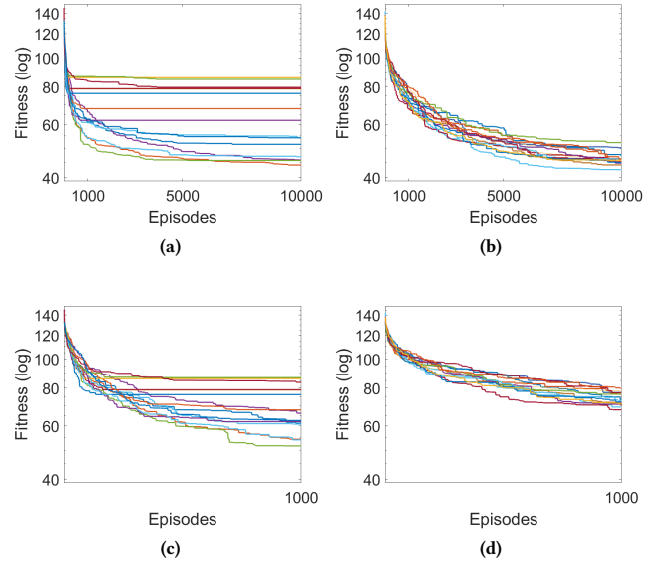

Figure 7: (a) The fitness values of the best evolved DSP rules, and (b) the fitness values of the best evolved HC parameters, both independently tested for 40 trials with 10000 episodes. Figures 7c and 7d are magnified views between 1–1000 episodes of (a) and (b) respectively.
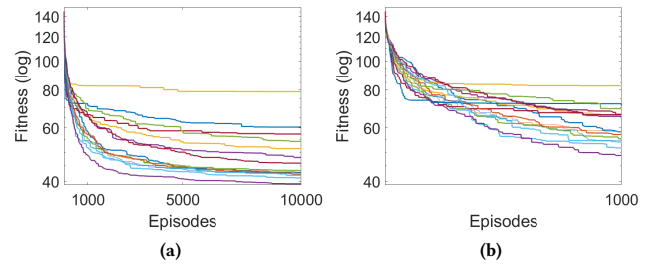

Figure 8: The fitness values of the best evolved delayed plasticity rules tested for 40 trials for 10000 episodes, with iterative re-sampling every 100 episodes. Figure 8b is a magnified view of the same results between episodes 1–1000.

We observe that some of the DSP rules seem to get stuck at a local optimum after around 100 episodes, which may be due to the fact that they were optimized for 100 episodes. Thus, to reduce this effect we used an iterative re-sampling approach to randomly reset all the weights of the networks (from the initial domain) at every 100 episodes without resetting the best found fitness value. Thus, in the visualization, the fitness value does not appear worse than the best fitness due to each re-initialization. The training process generated by the iterative re-sampling based DSP rules is provided in Figure 8. The best DSP rule achieves fitness values of 48.72 and 39.32 at 1000 and 10000 episodes. A perfect agent (an average of the distances of starting and 8 goal positions found by the A* algorithm)
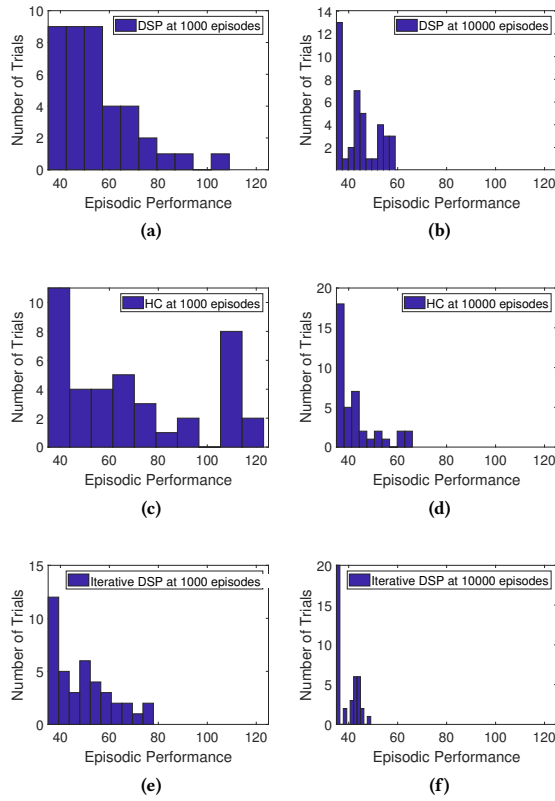
Figure 9: The distribution of the episodic performance of 40 trials using the best DSP rule, HC, and DSP rule with iterative re-sampling approach at episodes 1000 and 10000 are given in (a) (b), (c) (d), and (e) (f) respectively.

achieves a fitness value of 38.5. For completeness, we also performed additional experiments for the HC using the iterative re-sampling approach. However, we observed in this case that their results were not better than the standard HC with the tested settings.

The distributions of the episodic performance of the best performing DSP rule, HC with best parameters and DSP with iterative approach at 1000 and 10000 episodes are given in Figure 9. We fixed the minimum and maximum values of the x-axes of all figures to 35–125 for a better visual comparison. At 1000 episodes, all of the agents in 40 trials reach the goal with the smallest number of steps when the DSP rule with iterative re-sampling is used. On the other hand, 39 and 30 of the agents in 40 trials reach the goal when we use the DSP rule and HC respectively. At 10000 episodes, all of the agents in 40 trials reach the goal when the best heuristic from each approach is used. However, in the case of the DSP rule with iterative re-sampling, the agents reach the goal with the smallest number of steps on average. Figure 10 shows the comparison of the best evolved DSP[2], iterative DSP, HC and iterative HC heuristics with best evolved parameters. Iterative HC performs worse than

---

[2] We have recorded the performance of the agents on the triple T-maze task after training with the best evolved DSP rule for 100, 1000 and 10000 episodes. A video of the experiment is available online at: https://youtu.be/J0WYMrAMSdU.
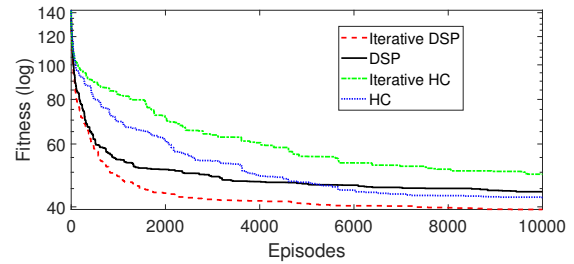

Figure 10: The fitness of the best evolved DSP (standard and iterative re-sampling variants) rule and the HC with best evolved parameters (standard and iterative re-sampling variants) tested on 10000 episodes.

the HC. On the other hand, the HC was able to outperform the DSP at around 5000 generations, while it could not perform better than the iterative DSP.

6 CONCLUSIONS

When the reinforcement signals are available after a certain period of time, it may not be possible to associate the activations of the neurons during this period with the reinforcement signals, and perform synaptic updates using Hebbian learning. In this work, we proposed the NATs, i.e. additional data storage in each synapse to keep track of neuron activations. We used DSP rules that take into account the NATs and delayed reinforcement signals to perform synaptic updates. We used relative reinforcement signals that were provided after an episode based on the relative performance of the agent in a previous episode. Since the NATs introduce knowledge of neuron activations into the DSP rules, we compared their results to an analogous HC algorithm that performs random synaptic updates without any knowledge of the network activations.

We observed that the DSP rules were highly efficient at training networks with a smaller number of episodes compared to the HC, as they converge quickly at a better fitness than the HC. When they were tested on a larger number of episodes on the other hand, they seemed to be outperformed slightly by HC. We hypothesized that this could be due to the fact that the DSP rules were optimized for a relatively small number of episodes (100). When we tried a iterative re-sampling approach, the DSP rules provided the best results. On the other hand, the DSP rules introduce an additional complexity that requires storing/updating four parameters per synapse during the network computation, and looking-up the update rule based on the activation patterns for each synaptic update.

We aim to apply the DSP on different ANN network models (i.e. continuous neuron activations) with various sizes, and test it on tasks with various complexity. It would also be interesting to investigate the adaptation capabilities of the networks with DSP when the environmental conditions change.

7 ACKNOWLEDGEMENTS

REFERENCES

[1] Thomas H Brown, Edward W Kairiss, and Claude L Keenan. 1990. Hebbian synapses: biophysical mechanisms and algorithms. *Annual review of neuroscience* 13, 1 (1990), 475–511.

[2] Oliver J. Coleman and Alan D. Blair. 2012. Evolving Plastic Neural Networks for Online Learning: Review and Future Directions. In *AI 2012: Advances in Artificial Intelligence*, Michael Thielscher and Dongmo Zhang (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 326–337.

[3] Leandro Nunes De Castro. 2006. *Fundamentals of natural computing: basic concepts, algorithms, and applications*. CRC Press, Boca Raton, Florida.

[4] Peter Dürr, Claudio Mattiussi, Andrea Soltoggio, and Dario Floreano. 2008. Evolvability of neuromodulated learning for robots. In *Learning and Adaptive Behaviors for Robotic Systems, 2008. LAB-RS'08. ECSIS Symposium on*. IEEE, New York, 41–46.

[5] Sami El-Boustani, Jacque PK Ip, Vincent Breton-Provencher, Graham W Knott, Hiroyuki Okuno, Haruhiko Bito, and Mriganka Sur. 2018. Locally coordinated synaptic plasticity of visual cortex neurons in vivo. *Science* 360, 6395 (2018), 1349–1354.

[6] Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary Intelligence* 1, 1 (2008), 47–62.

[7] Dario Floreano and Joseba Urzelai. 2000. Evolutionary robots with on-line self-organization and behavioral fitness. *Neural Networks* 13, 4-5 (2000), 431–443.

[8] Wulfram Gerstner, Marco Lehmann, Vasiliki Liakoni, Dane Corneil, and Johanni Brea. 2018. Eligibility Traces and Plasticity on Behavioral Time Scales: Experimental Support of NeoHebbian Three-Factor Learning Rules. *Frontiers in Neural Circuits* 12 (2018), 53.

[9] Donald Olding Hebb. 1949. *The organization of behavior: A neuropsychological theory*. Wiley & Sons, New York.

[10] Eugene M. Izhikevich. 2007. Solving the Distal Reward Problem through Linkage of STDP and Dopamine Signaling. *Cerebral Cortex* 17, 10 (2007), 2443–2452.

[11] Eduardo Izquierdo-Torres and Inman Harvey. 2007. Hebbian Learning using Fixed Weight Evolved Dynamical 'Neural' Networks. In *Artificial Life, 2007. ALIFE'07. IEEE Symposium on*. IEEE, New York, 394–401.

[12] Taras Kowaliw, Nicolas Bredeche, Sylvain Chevallier, and René Doursat. 2014. Artificial neurogenesis: An introduction and selective review. In *Growing Adaptive Machines*. Springer, Berlin, Heidelberg, 1–60.

[13] Eduard Kuriscak, Petr Marsalek, Julius Stroffek, and Peter G Toth. 2015. Biological context of Hebb learning in artificial neural networks, a review. *Neurocomputing* 152 (2015), 27–35.

[14] Decebal Constantin Mocanu, Elena Mocanu, Peter Stone, Phuong H Nguyen, Madeleine Gibescu, and Antonio Liotta. 2018. Scalable training of artificial neural networks with adaptive sparse connectivity inspired by network science. *Nature Communications* 9, 1 (2018), 2383.

[15] Jean-Baptiste Mouret and Paul Tonelli. 2014. Artificial evolution of plastic neural networks: a few key concepts. In *Growing Adaptive Machines*. Springer, Berlin, Heidelberg, 251–261.

[16] Yael Niv, Daphna Joel, Isaac Meilijson, and Eytan Ruppin. 2002. Evolution of Reinforcement Learning in Uncertain Environments: A Simple Explanation for Complex Foraging Behaviors. *Adaptive Behavior* 10, 1 (2002), 5–24.

[17] J. Orchard and L. Wang. 2016. The evolution of a generalized neural learning rule. In *2016 International Joint Conference on Neural Networks (IJCNN)*. IEEE, New York, 4688–4694.

[18] Sebastian Risi and Kenneth O. Stanley. 2010. Indirectly encoding neural plasticity as a pattern of local rules. In *International Conference on Simulation of Adaptive Behavior*. Springer, Berlin, Heidelberg, 533–543.

[19] Thomas Philip Runarsson and Magnus Thor Jonsson. 2000. Evolution and design of distributed learning rules. In *Combinations of Evolutionary Computation and Neural Networks, 2000 IEEE Symposium on*. IEEE, New York, 59–63.

[20] Andrea Soltoggio, John A Bullinaria, Claudio Mattiussi, Peter Dürr, and Dario Floreano. 2008. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *International conference on Artificial Life (Alife XI)*. MIT Press, Cambridge, MA, 569–576.

[21] Andrea Soltoggio and Kenneth O. Stanley. 2012. From modulated Hebbian plasticity to simple behavior learning through noise and weight saturation. *Neural Networks* 34 (2012), 28–41.

[22] Andrea Soltoggio, Kenneth O. Stanley, and Sebastian Risi. 2018. Born to learn: The inspiration, progress, and future of evolved plastic artificial neural networks. *Neural Networks* 108 (2018), 48–67.

[23] Andrea Soltoggio and Jochen J. Steil. 2013. Solving the distal reward problem with rare correlations. *Neural computation* 25, 4 (2013), 940–978.

[24] Kenneth O. Stanley. 2007. Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines* 8, 2 (2007), 131–162.

[25] Kenneth O Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. 2019. Designing neural networks through neuroevolution. *Nature Machine Intelligence* 1, 1 (2019), 24–35.

[26] Paul Tonelli and Jean-Baptiste Mouret. 2013. On the relationships between generative encodings, regularity, and learning abilities when evolving plastic artificial neural networks. *PloS one* 8, 11 (2013), e79138.

[27] Zlatko Vasilkoski, Heather Ames, Ben Chandler, Anatoli Gorchetchnikov, Jasmin Léveillé, Gennady Livitz, Ennio Mingolla, and Massimiliano Versace. 2011. Review of stability properties of neural plasticity rules for implementation on memristive neuromorphic hardware. In *International Joint Conference on Neural Networks (IJCNN)*. IEEE, New York, 2563–2569.

[28] Frank Wilcoxon. 1945. Individual comparisons by ranking methods. *Biometrics bulletin* 1, 6 (1945), 80–83.

[29] Anil Yaman, Decebal Constantin Mocanu, Giovanni Iacca, George Fletcher, and Mykola Pechenizkiy. 2018. Limited evaluation cooperative co-evolutionary differential evolution for large-scale neuroevolution. In *Genetic and Evolutionary Computation Conference, 15-19 July 2018, Kyoto, Japan*. ACM, New York, NY, USA, 569–576.

[30] Wei Zeng and Richard L Church. 2009. Finding shortest paths on real road networks: the case for A. *International journal of geographical information science* 23, 4 (2009), 531–543.