Transferring Evolved Reservoir Features in Reinforcement Learning Tasks

Kyriakos C. Chatzidimitriou[1,2], Ioannis Partalas[3], Pericles A. Mitkas[1,2], and Ioannis Vlahavas[3]

[1] Dept. of Electrical & Computer Engineering, Aristotle University of Thessaloniki, Greece
[2] Informatics and Telematics Institute, Centre for Research and Technology Hellas
`kyrcha@issel.ee.auth.gr, mitkas@eng.auth.gr`
[3] Dept. of Informatics, Aristotle University of Thessaloniki, Greece
`{partalas,vlahavas}@csd.auth.gr`

Abstract. The major goal of transfer learning is to transfer knowledge acquired on a source task in order to facilitate learning on another, different, but usually related, target task. In this paper, we are using neuroevolution to evolve echo state networks on the source task and transfer the best performing reservoirs to be used as initial population on the target task. The idea is that any non-linear, temporal features, represented by the neurons of the reservoir and evolved on the source task, along with reservoir properties, will be a good starting point for a stochastic search on the target task. In a step towards full autonomy and by taking advantage of the random and fully connected nature of echo state networks, we examine a transfer method that renders any inter-task mappings of states and actions unnecessary. We tested our approach and that of inter-task mappings in two RL testbeds: the mountain car and the server job scheduling domains. Under various setups the results we obtained in both cases are promising.

1 Introduction

Reinforcement learning (RL) [9] deals with the problem of how an agent, situated in an environment and interacting with it, can learn a policy, a mapping of states to actions, in order to maximize the total amount of reward it receives over time, by inferring on the immediate feedback returned in the form of scalar rewards as a consequence of its actions. RL has enjoyed increased popularity due to its ability to deal with complex and limited feedback problems, while it is believed to be an appropriate paradigm for creating fully autonomous agents in the future [8]. Despite the suitability to solve such problems, RL algorithms often require a considerable amount of training time especially for complex problems. A solution to speed up the learning procedure is through *transfer learning* (TL).

TL refers to the process of using knowledge that has been acquired in a previous learned task, the *source task*, in order to enhance the learning procedure in a new and more complex task, the *target task*. The tasks can belong to either the same domain, for example two mazes with different structure, or to different domains, for example checkers and chess board games. The more similar those two tasks are, the easier it is to transfer knowledge between them. TL is to play a crucial role in the development of fully autonomous agents since it is believed that it would be a core component in lifelong learning agents that persist over time [12].

For agents to perform well under a RL regime in real world problems, there is the need for *function approximators* (FAs) to model the policy and be able to generalize well to unseen environment states. Fully autonomous agents will need FAs that will adapt to the environment at hand with little, if any, human intervention [8]. Following this trend, we selected the *echo state network* (ESN) to be our FA of choice. Its recurrent neural network nature makes it appropriate for use as FA in agents dealing with sequential decision making problems, because it enables temporal computations and can process non-linear and non-Markovian state signals (Section 2).

In order to augment the capabilities of the adaptive FA approach, in this work, we tested methods of transferring reservoir topologies (or alternatively reservoir features). These topologies were adapted in a source RL task to be used as templates for the initial population of the neuroevolution procedure on a target RL task. Besides testing network transfer under the standard way of using mappings between the source and target, state variables and actions, we evaluated a transfer method agnostic of any mappings, taking advantage of the random, fully connected nature of ESNs. This procedure is performed at the microscopic level, that is in the level of the topology and the weights of the reservoir (Section 3).

Our methodology is evaluated empirically on two RL test-beds: a) the *mountain car* problem from the area of control and b) the *server job scheduling* problem from the area of autonomic computing (Section 4). The results of the experiments from both testbeds (Section 5) are promising under several different metrics with respect to the base search approach. We discuss related work in the area and how our approach differs from it (Section 6), closing with the conclusions of our research and plans for future work (Section 7).

2 Background

2.1 Echo State Networks

The idea behind *reservoir computing* (RC) and in particular ESNs [4] is that a random *recurrent neural network* (RNN), created under certain algebraic constraints, could be driven by an input signal to create a rich set of dynamics in its reservoir of neurons, forming non-linear response signals. These signals, along with the input signals, could be combined to form the so-called *read-out function*, a linear combination of features, $y = \mathbf{w}^T \cdot \phi(\mathbf{x})$, which constitutes the prediction of the desired output signal, given that the weights, w, are trained accordingly.

The reservoir consists of a layer of K input units, connected to N reservoir units through a $N \times K$ weighted connection matrix W^{in}. The connection matrix of the reservoir, W, is a $N \times N$ matrix. Optionally a backprojection matrix W^{back} could be present, with dimensions $N \times L$, where L is the number of output units, connecting the outputs back to the reservoir neurons. The weights from input units (linear features) and reservoir units (non-linear features) to the output are collected into a $L \times (K + N)$ matrix, W^{out}. For this work, the reservoir units use $f(x) = tanh(x)$ as an activation function, while the output units use either $g(x) = tanh(x)$ or the identity function, $g(x) = x$.

Best practices for generating ESNs, that is procedures for generating the random connection matrices W^{in}, W and W^{back}, can be found in [4]. Briefly, these are: (i) W should be sparse, (ii) the mean value of weights should be around zero, (iii) N should be large enough to introduce more features for better prediction performance, (iv) the spectral radius, ρ, of W should be less than 1 to practically (and not theoretically) ensure that the network will be able to function as an ESN. Finally, a weak uniform white noise term can be added to the features for stability reasons.

In this work, we consider discrete time models and ESNs without backprojection connections. As a first step, we scale and shift the input signal, $\mathbf{u} \in \mathbb{R}^K$, depending on whether we want the network to work in the linear or the non-linear part of the sigmoid function. The reservoir feature vector, $\mathbf{x} \in \mathbb{R}^N$, is given by Equation 1:

$$\mathbf{x}(t+1) = \mathbf{f}(\mathbf{W}^{in}\mathbf{u}(t+1) + \mathbf{W}\mathbf{x}(t) + \mathbf{v}(t+1)) \tag{1}$$

where $\mathbf{f}$ is the element-wise application of the reservoir activation function and $\mathbf{v}$ is a uniform white noise vector. The output, $\mathbf{y} \in \mathbb{R}^L$, is then given by Equation 2:

$$\mathbf{y}(t+1) = \mathbf{g}(\mathbf{W}^{out}[\mathbf{u}(t+1)|\mathbf{x}(t+1)]) \tag{2}$$

with $\mathbf{g}$, the element-wise application of the output activation function and $|$, the aggregation of vectors.

For RL tasks with K continuous states and L discrete actions, we can use an ESN to model a Q-value function, where each network output unit l, can be mapped to an action $a_l, l = 1 \dots L$, with the network output value y_l denoting the long-term discounted value, $Q(\mathbf{s}, a_l)$ of performing action a_l, when the agent is at state $\mathbf{s}$. Given $g(x) = x$, this Q-value can be represented by an ESN as:

$$y_l = Q(\mathbf{s}, a_l) = \sum_{i=1}^{K} w_{li}^{out} s_i + \sum_{i=K+1}^{K+N} w_{li}^{out} x_{i-K}, l = 1, \dots, L \tag{3}$$

while actions can be chosen under the ϵ-greedy policy [9].

Linear Gradient Descent (GD) SARSA TD-learning can be used to adapt weights [9, 10], where the update equations take the form of:

$$\delta = r + \gamma Q(\mathbf{s}, a') - Q(\mathbf{s}, a_l) \tag{4}$$

$$\mathbf{w}_l^{out\prime} = \mathbf{w}_l^{out} + \alpha\delta[\mathbf{s}|\mathbf{x}] \tag{5}$$

with a' the next action to be selected and α the learning rate.

2.2 NeuroEvolution of Augmented Reservoirs

NeuroEvolution of Augmented Topologies (NEAT) [7] is a topology and weight evolution of artificial neural networks algorithm, constructed on four principles that made it a reference algorithm in the area of NE. First of all, the network, i.e. the phenotype, is encoded as a linear genome (genotype), making it memory efficient with respect to algorithms that work with full weight connection matrices. Secondly, using the notion of *historical markings*, newly created connections are annotated with innovation numbers. NEAT during crossover aligns parent genomes by matching the innovation numbers and performs crossover on these matching genes (connections). The third principle is to protect innovation through *speciation*, by clustering organisms into species in order for them to have time to optimize by competing only in their own niche. Last but not least, NEAT starts with minimal networks, that is networks with no hidden units, in order (a) to initially start with a minimal search space and (b) to justify every complexification made in terms of fitness. NEAT complexifies networks through the application of structural mutations, by adding nodes and connections, and further adapts the networks through weight mutation by perturbing or restarting weight values. The above successful ideas could be used in other NE settings in the form of a meta-search evolutionary procedure. In our case, we follow these ideas to achieve an efficient search in the space of ESNs.

NeuroEvolution of Augmented Reservoirs (NEAR) [2] utilizes NEAT as a meta-search algorithm and adapts its four principles to the ESN model of neural networks. The structure of the evolutionary search algorithm is exactly the same as in NEAT with adaptations being made mainly with respect to gene representation, crossover with historical markings, clustering, including some additional evolutionary operators related to ESNs. An important difference from NEAT is that both evolution and learning are used in order to adapt networks to the problem at hand. NEAR, to its advantage, incorporates TD learning in order to make a local gradient descent search on the output matrix, W^{out}, of the ESN and to locate good solutions that reside nearby instead of performing just evolutionary search. In this work we use NEAR with the Lamarckian type of evolution, where learned weights, W^{out}, are transferred from generation to generation instead of being reset to zero before each generation.

3 Transfer of Reservoir Topologies

The idea behind our work is that certain parts of high performing ESNs in the source task could be reused as templates of the networks that make up the initial population in the target task. These certain parts should be ESN properties that distinguish one network from the other. In our case we have selected:

– the reservoir, denoted by matrix W, and along with it, the number N of neurons in the reservoir, the graph of the topology and the connection weights,
– the density D of the reservoir, and
– the spectral radius ρ, a factor used to dampen signals in the reservoir.

Our goal is to *alleviate completely* the problem of mappings in state and action variables between tasks by just transferring the reservoir connection matrix W and its particularities, leaving the other matrices W^{in} and W^{out} to be handled by the NEAR method. In particular, like in the standard way of generating ESN, W^{in} is randomly initialized and later adapted through NEAR, while W^{out} is also initialized randomly and under an evolutionary weight mutation operator that uses perturbation, appropriate weights are derived. This makes the transfer agnostic of any state or action mappings, since these are randomly created through the matrices W^{in} and W^{out}, only to be later adapted through NEAR to the problem at hand.

We have targeted our approach on transferring evolved reservoir repositories using the following methodologies. The prime symbol is used to denote properties of the target task ESN.

1. **Reservoir-Transfer**: The mapping agnostic method, that transfers the reservoir matrix $W' = W$ and the spectral radius $\rho' = \rho$ and randomly initializes W^{out} and W^{in} matrices, as discussed in the paragraph above (Figure 1).
2. **Mapping**: Use mappings that relate state and action variables from the source to the target task in the same way as that presented in [15] for the NEAT algorithm. The matrix W is transferred as is. The state and action mappings indicate which connection weight from matrices W^{in} and W^{out} of the final network of the source task, will be set to which position in matrices W'^{in} and W'^{out} of the initial networks of the target task. Such mappings are provided by a domain expert (Figure 2). This specific setup was chosen in order to test the mapping agnostic approach against a reference inter-task mapping methodology using neural networks.
3. **Mapping+Doubling**: Use mappings to account for W^{in} and W^{out} weights, but also increase the reservoir neurons (i.e. the hidden neurons) in order to account for the increased task complexity. The increase is directly proportional to the number of state and action variables growth, from the source to the target task. We have used doubling because in our testbeds we have a doubling in the number of state and action variables. Thus, we have created the new reservoir matrix W' to contain the matrix W in its upper left ($1 \leq i' \leq N$, $1 \leq j' \leq N$) and lower right ($N+1 \leq i' \leq N'$, $N+1 \leq j' \leq N'$) blocks, with the rest of the matrix elements set to 0 (Figure 3). This method was chosen in order to survey whether, in the presence of a higher dimensional target task, the computational units of the network need to be augmented as well. In fact, we wanted to experimentally test if the two reservoirs could handle the same number of variables each as in the source task and let the neuroevolution algorithm grow connections between them.

In all the above cases the basic properties of an ESN, N, D and ρ, are transferred implicitly in the first two cases (through matrix W) and explicitly in the case of ρ (by setting it initially in all target task genomes).

We have focused our attention on source-target task pairings that diverge from each other due to an increase in the dimensional complexity of the problem,

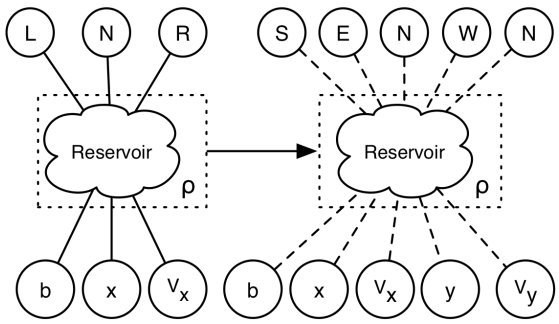

Fig. 1. In this setting only the reservoir, implicitly including the N and the D properties, is transferred along with spectral radius ρ. W^{in} and W^{out} matrices are randomly initialized and adapted through the NEAR process. The figure is an example of reservoir transfer in the mountain car domain discussed in the next section.

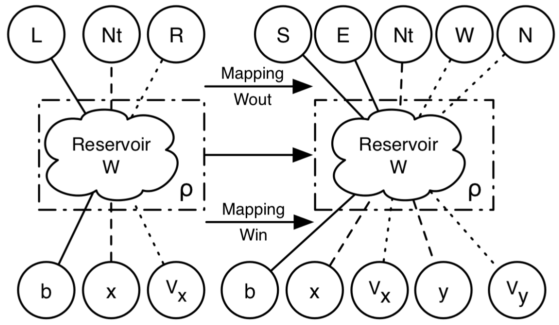

Fig. 2. Besides reservoir transfer, weights found in the source task for W^{in} and W^{out} matrices are transferred to the target task using inter-task mappings as described in [15].

but belong to the same domain. For example, situating the agent from a two dimensional (2D) problem to a three dimensional (3D) version or increasing the number of things it has to control, leading to an increase in the number of sensors (state variables) and actuators (actions). The main objectives of transfer learning are: (a) increased *asymptotic performance* of the transfer enabled agent over the basic one and (b) improvement in adaptation time to reach pre-specified thresholds of performance, a metric known as `time-to-threshold`.

4 Domains

4.1 Mountain Car

For the mountain car (MC) domain we use the version by [6]. In the standard 2D task an underpowered car must be driven up to a hill. The state of the

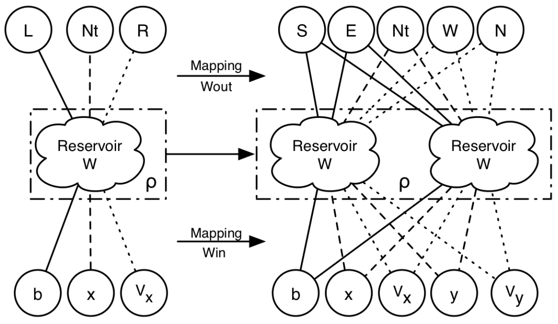

Fig. 3. Going a step beyond, and exploiting the doubling with respect to the number of state and action variables, the reservoir is doubled along with inter-task mappings.

environment is described by two continuous variables: horizontal position $x \in [-1.2, 0.6]$ and velocity $v_x \in [-0.007, 0.007]$. The actions are {Neutral, Left and Right} which modify the velocity by $-0.001, 0$ and 0.001, respectively. At each time step, an amount of $-0.0025 * \cos 3x$ is added to the velocity, in order to represent the gravity.

In our case, each episode starts with the car at a random position in the valley, having a random speed, and ends when x becomes greater than 0.5. At each time step, the agent selects among the three available actions and receives a reward of -1. The objective is to move the car to the goal state as fast as possible.

The 3D MC extends the 2D task by adding an extra spatial dimension. The 3D task was originally proposed in [14]. The state is composed by four continuous variables: the coordinates in space x, and $y \in [-1.2, 0.6]$, as well as the velocities v_x and $v_y \in [-0.07.0.07]$. The available actions are {Neutral, West, East, South, North}.

Along with the version discussed above, one more MC version was tested, which we will call the non-Markovian (NM) one, since the speed variable or variables in the 3D case, are kept from the agent, and only its position or positions in the 3D case are given, making the task more challenging. Each episode lasts 2500 time steps in all four MC versions after which another episode begins.

In this work we use the mountain car software[4] that is based on version 3.0 of the RL-Glue library[5] [11].

4.2 Server Job Scheduling

Server job scheduling (SJS) [17] is a domain that belongs to the realm of autonomic computing. It was previously used for performing transfer learning with the NEAT algorithm in [15]. Certain types of jobs are waiting in the job queue

[4] Available at http://library.rl-community.org/
[5] Available at http://glue.rl-community.org/

of a server to be processed as new jobs arrive in the queue. Each job type has a utility function that changes over time and represents the user anticipation over having the job scheduled. Certain users want quicker response from the server, while others are not as eager. The goal of the scheduler is to pick a job (action) based on the status of the queue (state), receiving as utility the value of the function of the scheduled job type at the timestep of execution (immediate reward). The performance is calculated as the sum of utilities (long term reward) when the queue empties. Each task begins with the scheduler finding 100 jobs in the queue, while at each time step a new job is added to the queue for the first 100 ticks. Each episode lasts 200 timesteps. For the state variable and action setup we used the modeling found in [17]. Job types 1 and 3 were used for the source tasks and all four jobs for the target task. The state variable mapping was to match the inputs concerned with job type 1 with the inputs of job type 2 and the inputs about job type 3 with inputs of job type 4.

5 Experiments

Each one of the experiments was conducted 10 times. The population was initialized with 100 individual networks and was evolved over 50 generations. In each generation, every individual was allowed to learn over 100 episodes with randomly initializing the starting state variables both in the MC and the SJS domains. In each generation, the champion network was evaluated for 1000 additional episodes with random restarts and its performance was recorded as the average fitness over those episodes. In the presence of transfer learning, before starting the evolution of the networks in the target task, the initial population was initialized as discussed in Section 3 with the seed network being the champion network produced by a neuroevolution procedure using the NEAR methodology in the source task.

Table 1 presents the results of the four approaches for all testbeds. The results illustrate the asymptotic performance of the agent for each task, i.e. the total reward received over time under the policy produced by the champion ESN over the 100 randomly initialized episodes. Our first observation is that, as the testbed difficulty increases, the performance gains of transferring reservoir features is even more evident.

The SJS domain can be considered a much more difficult domain than the MC, at least in terms of the number of state variables and actions the agent has to handle. In this specific testbed, as far as the average asymptotic performance is concerned, all transfer methods outperform evolution from scratch with a difference that is statistically significant at the 95% level under t-test. Moreover a smaller variation in the asymptotic behavior of the algorithms is observed under the transfer regimes. This is of particular value, when we are dealing with autonomous agents, which have to build such mechanisms without human supervision.

Last but not least, even though the mapping method has better performance overall, it does not statistically dominate over the mapping agnostic method.

Table 1. The average and standard deviation of the generalization performance for all the domains and algorithms under examination.

Domain	Mean	Sd
MC-M-Scratch	-100.04	3.51
MC-M-Reservoir	-96.47	4.48
MC-M-Mapping	-97.67	4.40
MC-M-Map+Double	-97.95	4.32
MC-NM-Scratch	-293.52	47.86
MC-NM-Reservoir	-288.52	62.31
MC-NM-Mapping	-261.92	59.30
MC-NM-Map+Double	-279.71	33.67
SJS-Scratch	-5243.08	82.20
SJS-Reservoir	-5187.02	31.24
SJS-Mapping	-5176.42	16.74
SJS-Map+Double	-5187.17	30.46

In fact, this could indicate that the reservoir along with its properties captures crucial information of the domain in this inter-task transfer. This kind of information could be specific sub-graphs existing in the reservoir, which together with specific weight values in the connections, calculate temporal features needed to produce efficient policies faster than when starting from scratch.

In both testbeds, the reservoir transfer methods win most of the races towards the performance threshold targets than the baseline version. This can be seen in the Figures 4, 5 and 6. The reservoir transfer method wins the race 7 times, followed by the mapping transfer method with 6 times and the mapping with reservoir doubling with 1 times, indicating a dominance of the transfer methods versus evolving a network from scratch with respect to this criterion. We also, note that in the Markovian variation of the MC domain, the agnostic method is the best algorithm as it preserves a slightly better performances against its rivals.

6 Related Work

This section presents related work in TL and contrasts it with the proposed approach. For a broader review of TL in reinforcement learning domains the reader can refer to a comprehensive survey of the field [12].

The transfer of neural networks has been studied in [15] and [1]. More specifically, Taylor et al. [15] proposed a method, named *Transfer via inter-task mappings for Policy Search Reinforcement Learning* (TVITM-PS), that initializes the weights of the target task, using the learned weights of the source task, by utilizing mapping functions. In particular, in [15], they proposed a TL method for policy search algorithms where the policies are represented as neural networks. The internal structure of the source networks along with the weights are copied to the target task using predefined or learned mapping functions. We have

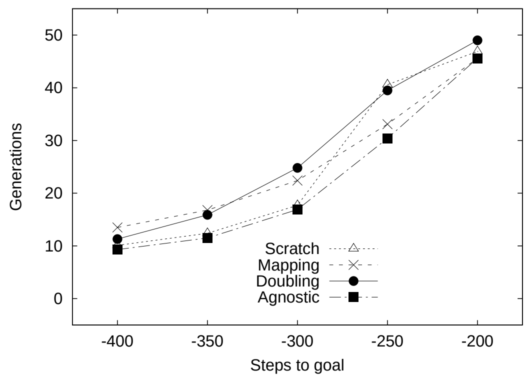

Fig. 4. The number of generations to reach pre-specified performance thresholds in the Markovian MC problem.

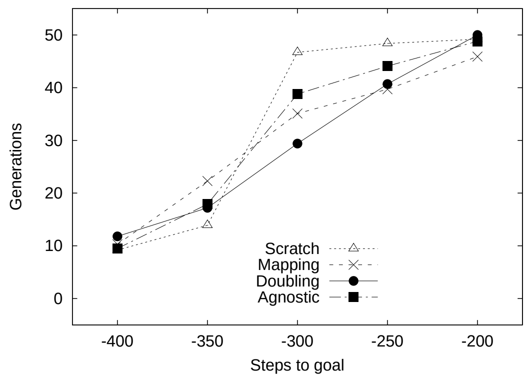

Fig. 5. The number of generations to reach pre-specified performance thresholds in the non-Markovian MC testbed.

evaluated this method with respect to ESNs and the NEAR algorithm, which is the first time to our knowledge. Such a test has value since ESNs have inherently more connections than NEAT derived networks. Additionally, we have tested a method that tries to take advantage of the fully, random, connectivity of ESNs and does not require the use of mappings, with promising results.

Finally, in [1] Bahceci and Miikkulainen introduce a method that transfers pattern-based heuristic in games. A population of evolved neural networks (which represent the patterns) to a target task as the starting population. In contrast our method allows different action and state spaces between the source and target tasks, and also is agnostic of any mappings.

Several other approaches have been proposed in the past for transfer learning. More specifically, [3] describes an algorithm which reuses policies from previously learned source tasks, while methods that use rules extracted from experience pre-

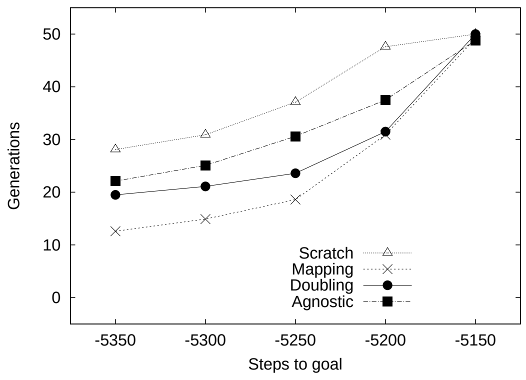

Fig. 6. The number of generations to reach pre-specified utility thresholds for the SJS domain.

viously gained in a source task have been proposed in [5, 16]. Both our testbeds have been used before in TL research. The 2D to 3D MC source-target pair has been used in [13], while the SJS in [15].

7 Conclusions and Future Work

In this paper we presented a method in the field of transfer learning and more specifically a method of transferring reservoir topologies and weights from a source task in order to facilitate adaptation of ESNs in a more complex target task. The main contribution of this paper is a method that renders any inter-task mappings of states and actions unnecessary. We believe that this is a step towards the fully automation of the transfer learning procedure.

We have tested the approach in three different problems from two different domains. Results indicated that as the difficulty of the task increases in terms of complexity in the states and actions, the gains from transferring knowledge are more evident. We hypothesize that as problems become increasingly difficult, the population initialization of NE methods will play an even more crucial role. The mapping approach is the method that outperformed any other, since it is difficult to beat expert knowledge with randomly assigned weights. On the other hand, transferring just reservoir and its properties seems to be a viable solution when the problem's dimensionality increases and mappings are becoming obscure. The NEAR methodology as is, is capable of finding solutions quite rapidly in the above problems, so the statistically significant improvement in the asymptotic performance on the server job scheduling task, is a promising result for further gains through reservoir transfer in much more demanding environments.

We plan to investigate the reservoir transfer approaches in more difficult testbeds like the keepaway domain. Additionally, we plan to work on cross-domain reservoir transfer, where the source and the target tasks belong to different domains. Finally, another issue that deserves further research is the sensitivity of the transfer learning method discussed in this paper with the presence of noise for both the source and the target tasks.

References

1. Bahceci, E., Miikkulainen, R.: Transfer of evolved pattern-based heuristics in games. In: IEEE Symposium on Computational Intelligence and Games (2008)
2. Chatzidimitriou, K.C., Mitkas, P.A.: A neat way for evolving echo state networks. In: European Conference on Artificial Intelligence. pp. 909–914 (2010)
3. Fernández, F., Veloso, M.: Probabilistic policy reuse in a reinforcement learning agent. In: 5th international joint conference on Autonomous agents and multiagent systems. pp. 720–727 (2006)
4. Jaeger, H.: Tutorial on training recurrent neural networks, covering BPTT, RTRL, EKF and the "echo state network" approach. Tech. Rep. GMD Report 159, German National Research Center for Information Technology (2002)
5. Madden, M.G., Howley, T.: Transfer of experience between reinforcement learning environments with progressive difficulty. Artif. Intell. Rev. 21(3-4), 375–398 (2004)
6. Singh, S.P., Sutton, R.S.: Reinforcement learning with replacing eligibility traces. Machine Learning 22(1-3), 123–158 (1996)
7. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2), 99–127 (2002)
8. Stone, P.: Learning and multiagent reasoning for autonomous agents. In: Proceedings of the 20th International Joint Conference on Artificial Intelligence. pp. 13–30 (January 2007), http://www.ijcai-07.org/
9. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
10. Szita, I., Gyenes, V., Lőrincz, A.: Reinforcement learning with echo state networks. In: Artificial Neural Networks ICANN 2006. Lecture Notes in Computer Science, vol. 4131/2006, pp. 830–839. Springer Berlin / Heidelberg (2006)
11. Tanner, B., White, A.: Rl-glue: Language-independent software for reinforcement-learning experiments. Journal of Machine Learning Research 10, 2133–2136 (2010)
12. Taylor, M., Stone, P.: Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research 10, 1633–1685 (2009)
13. Taylor, M.E., Jong, N.K., Stone, P.: Transferring instances for model-based reinforcement learning. In: Machine Learning and Knowledge Discovery in Databases. vol. 5212, pp. 488–505 (September 2008)
14. Taylor, M.E., Kuhlmann, G., Stone, P.: Autonomous transfer for reinforcement learning. In: AAMAS '08: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems. pp. 283–290 (2008)
15. Taylor, M.E., Whiteson, S., Stone, P.: Transfer via inter-task mappings in policy search reinforcement learning. In: 6th international joint conference on Autonomous agents and multiagent systems. pp. 1–8 (2007)
16. Torrey, L., Shavlik, J., Walker, T., Maclin, R.: Skill acquisition via transfer learning and advice taking. In: 17 th European Conference on Machine Learning. pp. 425–436 (2005)
17. Whiteson, S., Stone, P.: Evolutionary function approximation for reinforcement learning. Journal of Machine Learning Research 7, 877–917 (May 2006)