

## Action-Dependent Heuristic Dynamic Programming for Neurooptimization

[Nikita A. Visnevski](#), and [Danil V. Prokhorov](#)

[Applied Computational Intelligence Laboratory](#),

[Dept. of Electrical Engineering](#), Box 43102,

[TexasTech University](#), Lubbock, TX, 79409

e-mail: [visnev@image2.ee.ttu.edu](mailto:visnev@image2.ee.ttu.edu)

### **Abstract.**

We propose a new approach to solving optimization problems based on an adaptive critic design called Action-Dependent Heuristic Dynamic Programming (ADHDP) [1]. We consider the well-known Traveling Salesman Problem (TSP) as a helpful benchmark to highlight main principles of our approach. TSP formulation and many methods to solve it including the classical sequential nearest city algorithm and Hopfield neural network are given in [2,3].

Our alternative approach is based on using two recurrent neural networks connected as in ADHDP. The first network, called Critic, outputs estimates of the cost-to-go in the Bellman equation of dynamic programming; whereas the second network, called Action, provides minimization of those estimates. For the ten-city TSP considered, the cost-to-go is a cost of a portion of a tour left for a salesman to travel from a current city. Action sequentially produces codes of cities a salesman should visit in order to minimize the cost-to-go. To assist Action in always outputting correct codes of cities, a constraint function is developed, and this function, together with Critic, forms a combined Critic employed to train Action. We describe recurrent network training procedure and the overall ADHDP design, followed by the development of the constraint function and Action's training. In a conclusion, we discuss simulation results and provide directions for further research.

### **Recurrent Network Training algorithm.**

A procedure called Real Time Recurrent Learning with Node Decoupled Extended Kalman Filter training algorithm is considered in [4,5]. We can obtain all derivatives of the state variables of a recurrent neural network with respect to network's inputs and weights simultaneously with forward propagation through the network. Although real-time learning is not necessary for the TSP, we decided to use this training procedure because it allows to eliminate both backpropagation routines and dual network constructions. Derivatives of the state variables with respect to network's inputs (2) and weights (3) can be written in the following form convenient for direct implementation:

$$y_{-in_j}^t(t) = \omega_{j,1}^t + \sum_{k=1}^{n_j-1} \omega_{j,k+1}^t \cdot y_k^{t-1}(t) + \sum_{k=1}^{n_j} \omega_{j,k+1+n_{j-1}}^t \cdot y_j^t(t-1), \text{ where: } y_j^t(t) = f(y_{-in_j}^t(t)) \quad (1)$$

$$\frac{\bar{\partial} y_j^t(t)}{\bar{\partial} \omega_{j,b}^0(t)} = f'(y_{-in_j}^t(t)) \cdot \left[ (1 - \delta_{i,t}) \cdot \sum_{k=1}^{n_j-1} \omega_{j,k+1}^t \cdot \frac{\bar{\partial} y_k^{t-1}(t)}{\bar{\partial} \omega_{j,b}^0(t)} + \delta_{i,t} \cdot \omega_{j,b}^t \right] \quad (2)$$

$$\frac{\bar{\partial} y_j^t(t)}{\bar{\partial} \omega_{j,b}^t} = f'(y_{-in_j}^t(t)) \cdot \left\{ \gamma \cdot \sum_{k=1}^{n_j} \omega_{j,k+1+n_{j-1}}^t \cdot \frac{\bar{\partial} y_k^t(t-1)}{\bar{\partial} \omega_{j,b}^t} + (1 - \delta_{i,t}) \cdot \sum_{k=1}^{n_j-1} \omega_{j,k+1}^t \cdot \frac{\bar{\partial} y_k^{t-1}(t)}{\bar{\partial} \omega_{j,b}^t} + \right.$$

$$\delta_{i,l} \cdot \delta_{g,j} \cdot \left[ \sum_{k=1}^{n_{l-1}} \delta_{h,k+l} \cdot y_k^{l-1}(t) + \sum_{k=1}^{n_l} \delta_{h,k+l+n_{l-1}} \cdot y_k^l(t-1) + \delta_{h,1} \right], \quad l \leq i \quad (3)$$

$$\frac{\bar{z}_j^l(t)}{\partial \omega_{g,h}^l} = f'(z_{-in_j^l}(t)) \cdot \left\{ \gamma \sum_{k=1}^{n_{l-1}} v_{j,k+l+n_{l-1}}^l \cdot \frac{\bar{z}_j^l(t-1)}{\partial \omega_{g,h}^l} + (1 - \delta_{l,1}) \cdot \sum_{k=1}^{n_{l-1}} v_{j,k+l}^l \cdot \frac{\bar{z}_j^{l-1}(t)}{\partial \omega_{g,h}^l} + \delta_{l,1} \cdot \sum_{k=1}^{n_l} v_{j,k+1}^l \cdot \frac{\bar{z}_k^1(t)}{\partial \omega_{g,h}^l} \right\} \quad (4)$$

where:

$y_j^i(t)$ ,  $z_j^i(t)$  are state variables of Action and Critic, respectively: output of the neuron  $j$  in the layer  $i$ ;

$y_g^0(t)$  is the  $g$ -th network's input;

$\omega_{j,k}^i$ ,  $v_{j,k}^i$  are  $k$ -th weights of the neuron  $j$  in the layer  $i$  of Action and Critic, if  $k = 1$  - the bias case;

$\delta_{i,j}$  is the Kronecker delta;  $\delta_{i,j} = 1$  if  $i=j$  and 0 otherwise;

$f'(y_{-in_j^l})$  is the derivative of the neuron's  $y_j^l(t)$  activation function with respect to  $y_{-in_j^l}(t)$ ;

$y_{-in_j^l}(t)$  is neuron's  $y_j^l(t)$  activation (1);

$L$  is the number of layers in the Action;

$N_i$ ,  $M_i$  is the number of neurons in the layer  $i$  of Action and Critic;

$\bar{\mathcal{D}}(\bullet)$  is a dynamic derivative (see ,e.g. [5,6]);

$\gamma$  is a discount factor,  $0 < \gamma < 1$ .

The activation function may be chosen as a bipolar sigmoidal function in the form of hyperbolic tangent.

### **ADHDP design for TSP.**

The basic configuration of ADHDP is shown on the Fig. 1. Critic outputting estimates  $z^0$  of the cost-to-go is connected to Action which produces a code of a city to be visited next. A code of the current city serves as the input to Action. Both Action and Critic are recurrent networks of the multilayer perceptron's type. Dynamic derivatives of the Critic's states, including its outputs, with respect to the Action's weights are obtained by (4) Dynamic derivatives of the Critic's output with respect to the Critic's inputs may be obtained similarly through (2).

Action generates codes of the cities. We adopt the following notation:

$S = \{1, \dots, N_{City}\}$  - set of decimal city's numbers;  $Bip\{n\}$ ,  $n \in S$ , gives a bipolar code with the efficiency  $M_{Code}$  as a bipolar representation of the decimal city's number. Here  $M_{Code}$  is defined below:

$$M_{Code} = \left\lceil \log_2(N_{City}) \right\rceil \quad (5)$$

We can also define a reverse mapping of the Action output vector,  $Y^\perp(t) = \{y_k^\perp(t)\}$ ,  $k = \overline{1, M_{Code}}$ , into  $S^c$ :

$Dec\{Y^\perp(t)\} = n$ :  $n \in S^c = S \cup \{N_{City} + 1, \dots, 2^{M_{Code}}\}$  - the whole set of decimal numbers which can be mapped completely into a bipolar code of the efficiency  $M_{Code}$ .

Other ways of encoding the cities can also be considered. The simplest form of encoding - the use of just one output neuron with graded response - fails due to excessive accuracy demands during training. It is our believe, however, that, unless some structural changes are incorporated into Action, its training, or Critic, it is hardly possible to avoid using a certain constraint function as addition to the Critic.

In ADHDP, Critic could be trained by applying the following rule:

$$z^0 = \psi \cdot z^0(t+1) + U(t, t+1), \quad (6)$$

where:

$z^0(t)$ ,  $z^0(t+1)$  are two successive estimates of the cost-to-go,  $0 < \psi < 1$ ,

$U(t, t+1)$  is the local cost, or the cost of the travel from the city  $Y^\perp(t)$  to the city  $Y^\perp(t+1)$ .

Alternatively, one could train Critic to explicitly predict the following sums:

$$z^0(t) = \sum_{k=t}^{End\_tour} U(k, k+1), \quad t = \overline{1, N_{City}} \quad (7)$$

Using (7), we trained Critic on a variety of valid tours with the total cost from 3.5 to 5.5 (the minimum total cost is known to be 2.71 for the ten-city TSP [1]). We exploited (3) with replacement of  $y$  and  $\omega$  by  $z$  and  $v$ , respectively, and Node Decoupled Extended Kalman Filter training algorithm [5].

### **Combined Critic design and Action Network training.**

The design above has an important limitation which stems from the necessity to encode the city numbers. What if the Action network would output a code of an unexisting city or visit one city twice? It is clearly

infeasible to train Critic to anticipate all possible code sequences. Thus, we need to develop the constraint function that would make Action avoid producing invalid code sequences. This function could work in a linear combination with the Critic's output. Let us define a set  $\mathcal{R} = \{n_i\}$ :  $n_i \in \mathcal{S}$  of all cities already visited during the current tour. The set  $\mathcal{R}$  is a set of the variable cardinality  $|\mathcal{R}|$ . We can construct the constraint function  $\tilde{J}(Y^1(t))$  in a variety of ways. As an example we consider two possible forms:

$$\tilde{J}(Y^1(t)) = \frac{1}{2} \cdot \left\{ \xi_1 \cdot \sum_{k=1}^{d_{\text{Code}}} \left( \phi^k - [y_k^1(t)]^2 \right)^2 + \xi_2 \cdot \prod_{k=1}^{n_{\text{City}} - |\mathcal{R}|} \left[ \frac{1}{2} \cdot \sum (\bar{\lambda}_k - y_k^1(t))^2 \right] \right\} \quad (8)$$

$$\tilde{J}(Y^1(t)) = \sum_{i=1}^{n_{\text{City}} - |\mathcal{R}|} \left\{ 1 - \exp \left[ -\frac{1}{2 \cdot \sigma^2} \cdot \sum_{k=1}^{d_{\text{Code}}} (y_k^1(t) - \bar{\lambda}_k)^2 \right] \right\} \quad (9)$$

where:

$\phi$ ,  $\sigma$  is a desired amplitude of the Action's output and a deviation of the Gaussian PDF, respectively;

$\bar{\Lambda}^1 = \{\bar{\lambda}_k\}$ :  $\text{Dec}(\bar{\Lambda}^1) \in \bar{\mathcal{R}}$ ,  $k = \overline{1, M_{\text{Code}}}$ ,  $i = \overline{1, |\bar{\mathcal{R}}|}$ , and  $\bar{\mathcal{R}}$  is a complement of the set  $\mathcal{R}$  defined on  $\mathcal{S}$ ;

$\xi_1$ ,  $\xi_2$  are normalization coefficients  $\in [0, 1]$ .

Examples of these functions in the 2D state space are shown on the Figs. 2 and 3.

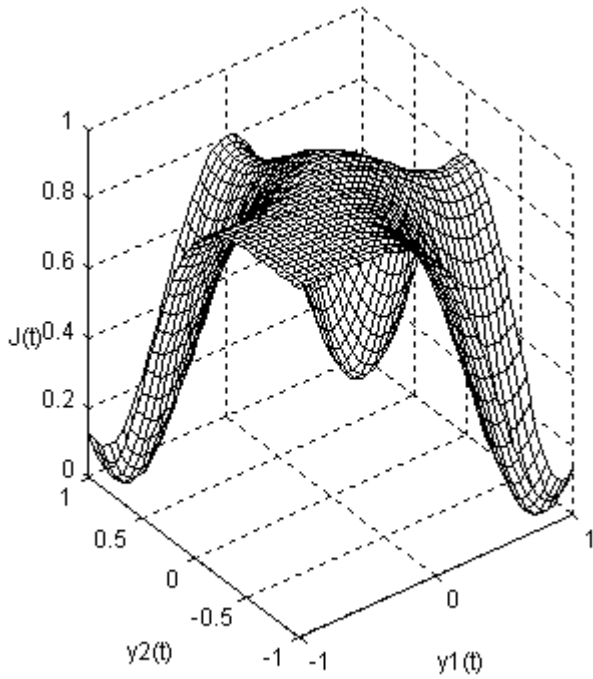
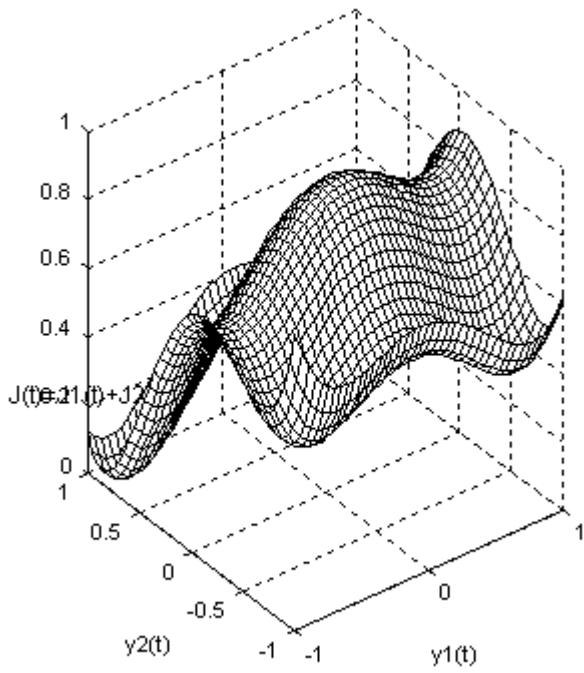


Fig. 2.  $\tilde{J}(Y^1(t))$  in the NonGaussian form (8). Fig. 3.  $\tilde{J}(Y^1(t))$  in the Gaussian form (9).

Cities with codes (-1,-1) and (1,-1) are already City with the code (-1,-1) is already visited

visited or do not exist in the problem. or does not exist in the problem.

To produce optimal sequences we need to augment a process of minimizing the constraint function by the adaptation through the trained Critic. We call this augmentation a combined Critic, and it is shown included in ADHDP design on the Fig. 4. The output of the combined Critic and the output's derivatives with respect to Action weights can be written in the form (10):

$$J(Y^1(t)) = \xi_3 \cdot \tilde{J}(Y^1(t)) + \xi_4 \cdot Z^{\text{Cost}}(t), \text{ and } \frac{\partial J(Y^1(t))}{\partial \omega_{g,h}^j} = \xi_3 \cdot \sum_{k=1}^{N_{\text{Cities}}} \frac{\partial \tilde{J}(Y^1(t))}{\partial \phi_k^1(t)} \cdot \frac{\partial \phi_k^1(t)}{\partial \omega_{g,h}^j} + \xi_4 \cdot \frac{\partial Z^{\text{Cost}}(t)}{\partial \omega_{g,h}^j} \quad (10)$$

where:  $\xi_3, \xi_4$  are normalization coefficients,  $\in [0,1]$ . We utilize  $\xi_4$  as a parameter mainly growing with a number of produced tours. The combined Critic (10) requires a target cost-to-go function to train Action by the Extended Kalman Filter procedure. We used the linear target function  $z_d(t) = 2 - 0.2 \cdot t, t = \overline{1, N_{\text{City}}}$ . One can avoid using this somewhat artificial target function by exploiting the necessary condition for an extremum of (10) in the  $Y^1(t)$  space [7].

## Conclusion.

In comparison to Hopfield neural networks, where the number of network's outputs is a square of the total number of cities, our approach is a much more parsimonious with the number of network's outputs scaling as (5). Our experiments demonstrate that although the constraint function  $\tilde{J}(Y^1(t))$  helps the Action network not to output an invalid code, it does not completely eliminate occurrence of staying at the same city for several time steps. This would not be a problem if Critic could anticipate such cases. However, Critic used in this study was trained only on examples of valid tours. While currently working on improvements of this approach, we are investigating alternatives. For instance, it appears to be promising to use a stochastic learning automaton as Action and left Critic operate on probabilities of transitions from one city to another, rather than on codes of the cities. Another alternative is to exploit more advanced adaptive critic designs [1,8]. Active research are currently under way to investigate these directions for solving optimization problems with adaptive critic designs.

## References.

- [1] P. Werbos, "Approximate dynamic programming for real-time control and neural modeling." *Handbook of Intelligent Control*, D. White and D. Sofge, eds., VNR, N.Y., 1992.
- [2] Fausett L. *Fundamentals of Neural Networks: Architectures, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1994

- [3] Lawler E. L., Lenstra J. K., Rinnoy Kann A. H. G., Shmoys D. B. *The Traveling Salesman Problem*. John Wiley & Sons, 1984
- [4] Haykin S. *Neural network, A Comprehensive Foundation*. Maxwell Macmillon International 1994
- [5] Puskorius G. V., Feldkamp L. A. "Neurocontrol of Nonlinear Dynamical systems with Kalman Filter Trained Recurrent Networks". *IEEE Transactions on Neural Networks*, vol. 5, no. 2, March 1994, 279-97
- [6] Werbos P. J. "Backpropagation through time: what it does and how to do it." *Proceedings of the IEEE*, vol. 78, no. 10, October 1990,1550-60
- [7] Yuan, F., Feldkamp, L., Puskorius, G., and L. Davis. A Simple Solution to the Bioreactor Benchmark Problem by Application of Q-learning. In *Proceedings of the World Congress on Neural Networks (WCNN-95)*, Washington, DC, July 1995.
- [8] Prokhorov, D., and D. Wunsch. Advanced Adaptive Critic Designs, submitted to *the World Congress on Neural Networks (WCNN-96)*.