12th Int. Workshop on Generalized Nets
Burgas, 17 June 2012, 17–26

Generalized net model of neuro-dynamic programming algorithm

Tatiana Ilkova[1], Olympia Roeva[1], Mitko Petrov[1] and Juris Vanags[2]

[1] Bioinformatics and Mathematical Modelling Department
Institute of Biophysics and Biomedical Engineering, Bulgarian Academy of Sciences
e-mails: {tanja, olympia, mpetrov}@biomed.bas.bg

[2] Laboratory of Bioengineering
Latvian State Institute of Wood Chemistry
Riga, Latvia
e-mail: btc@edi.lv

Abstract: The apparatus of generalized nets (GN) is applied here to describe the algorithm of *neuro-dynamic programming* (*NDP*). *NDP* was an alternative to alleviate the "*curse of dimensionality*" of the *Dynamic programming* (*DP*). The traditional approach for solving the Bellman's equation involves gridding of the state space, solving the optimization for each grid point, as well as performing the stagewise optimization until convergence is reached. The comprehensive sampling of state space can be avoided by identifying the relevant regions of the state space though simulation under judiciously chosen suboptimal policies, which is presented using *NDP* methods. The proposed method is particularly simple to implement and can be applied for *on-line* optimization
Keywords: Generalized nets, Neuro-dynamic algorithm.
AMS Classification: 68Q85, 90C39.

1 Introduction

The theory of the Generalized nets (GN) [2–4] proved to be quite successful when applied to the description of the functioning of expert systems, machine learning and different technological processes. Up to now GN are successfully applied as a tool for modeling in several areas – economics, transport, medicine, computer technologies etc., [1–5, 14].

In this paper the apparatus of GN is used to describe the algorithm of *Neuro-dynamic programming* (*NDP*).

NDP is proposed as an alternative to alleviate the "*curse of dimensionality*" of the *Dynamic programming* (*DP*). The term *NDP* expresses the reliance of the methods, described in

this article with respect to both the *DP* and the neural network concepts [7]. The term reinforcement learning is also used in the artificial intelligence community where the methods originated from. Using common artificial intelligence terms, the methods help the systems "*learn how to make good decisions by observing their own behavior and use built-in mechanisms for improving their actions through a reinforcement mechanism*" [7].

The key idea is to use a scoring function to select decisions in complex dynamic systems, arising from a broad variety of applications for engineering design, operations research, resource allocation, finance, etc. This is much similar to a computer chess, where positions are evaluated by means of a scoring function and the move that leads to the position with the best score is chosen. *NDP* provides a class of systematic methods for computing the appropriate scoring functions using approximation schemes and simulation/evaluation of the system's performance, [8].

In more mathematical meaning "*observing their own behavior*" relates to simulation and "*improving their actions through a reinforcement mechanism*" relates to the iterative schemes for improving the quality of approximation of the optimal cost function, the Q-factors or the optimal policy. There has been a gradual realization that the reinforcement learning techniques can be fruitfully motivated and interpreted in terms of classical *DP* concepts such as the value and policy iteration [6, 16].

Some applications of *NDP* (*reinforcement learning application*) are: theoretical, finances, inventory management, semiconductor manufacturing, telecommunications, games and strategy, artificial intelligence and *NDP* applications, Bayesian networks, internet, optimal stopping, autonomous vehicles, monitor planning, military, industrial applications and other as combinatorial optimization; maintenance and repair, dynamic channel allocation, job-shop scheduling, etc. [7, 8].

NDP is a relatively new class of the dynamic programming methods for control and sequential decision making under uncertainty. These methods have the potential of dealing with some problems that were thought to be intractable for a long time due to either a large state space or the lack of an accurate model. They combine ideas from the fields of neural networks, artificial intelligence, cognitive science, simulation, and approximation theory.

The aim of this study is to describe the method and algorithm of *NDP* with a GN model as a premise for it qualitative learning.

2 Neuro-dynamic programing

In the systems, decisions are made in stages. The outcome of each decision is not fully predictable but can be anticipated to some extent before the next decision is made. Each decision results in some immediate cost, but it also affects the context in which the future decisions are to be made and thus it affects the cost incurred in future stages. *DP* provides a mathematical formalization of the tradeoff between the immediate and future costs. Generally, in *DP* formulations there is a discrete-time dynamic system whose state evolves according to given transition probabilities that depend on the decision/control **u**.

DP is an elegant way to solve many optimization problems. It involves a stagewise calculation of the *cost-to-go function* to arrive at the solution not just for a specific initial state,

but for a general initial state. Once obtained the *cost-to-go function*, represents a convenient means to obtain the solution for a general state. In very few cases, the stagewise optimization to obtain analytically a closed-form expression for the *cost-to-go function* has been solved. The conventional approach to the problem involves gridding the state space, calculating and storing the *cost-to-go* for each grid points as one march backward from the first stage to the last. For an infinite horizon problem the number of iterations required for convergence can be very large. Such an approach is seldom practically feasible due to the exponential growth of the computation with respect to the state dimension. Unfortunately, from the very beginning it is apparent that an increase of the dimensionality of the problem causes an exponential increase in the time required to find a solution. This is referred to as the "*curse of dimensionality*", which must be removed so that this approach can find a widespread use.

NDP aims to develop a methodological foundation for combining dynamic programming, compact representations, and simulation to provide the basis for a rational approach to complex stochastic decision problems [7, 11].

Two fundamental *DP* algorithms, policy iteration and value iteration, are the starting points for the *NDP* methodology. The most straightforward adaptation of the policy iteration method operates as follows: we start with a given policy (a rule for choosing a decision u at each possible state *i*), and we approximately evaluate the cost of that policy (as a function of the current state) by least-squares-fitting a scoring function to the results of many simulated system trajectories using that policy. A new policy is then defined by minimization in Bellman's equation where the optimal cost is replaced by the calculated scoring function and the process is repeated. This type of algorithm typically generates a sequence of policies that eventually oscillates in a neighboorhood of an optimal policy. The resulting deviation from optimality depends on a variety of factors, principal among which is the ability of the scoring function architecture to accurately approximate the cost functions of the various policies.

NDP uses simulated process data received under suboptimal policies to fit an approximate *cost-to-go function* – generally by fitting artificial network. With the value iteration approach *NDP* the initial approximate *cost-to-go function* in the future was improved by an iteration procedure based on Bellman's equation. In this way the simulation role has two points. First, by simulation the process under a reasonably chosen suboptimal policy and all possible operating parameters it provides set data points that define the relevant "*working*" region in the state space. Second, the simulation provides the *cost-to-go value* under the suboptimal policy for each state visited, which iteration of the Bellman's equation can be initialed with [11].

A general dynamic optimization problem can be defined as follows:

$$\max_{\mathbf{u}_0,\ldots,\mathbf{u}_{k-1}} \sum_{i=1}^{k-1} f\ (\mathbf{W}_i, \mathbf{u}_i) \tag{1}$$

where **W** is a vector of the variables that describe process, **u** is a vector of control variables, and *k* is the current stage.

The objective is to maximize the combination of the total span and the stagewise, together with the terminal costs subject and the terminal constrains.

DP includes a stagewise calculation of the *cost-to-go function* to reach the solution for the general initial state. The *cost-to-go* (Eq. (1)) at each stage is defined by:

$$B_i\left(W(t_i),t_i\right)=\max_{u_{\min}\le u_k\le u_{\max}}\Delta t\sum_{k=1}^{N-1}f_k\left(\mathbf{W}_k,\mathbf{u}_k\right) \tag{2}$$

where *B* is Bellman's function.

Then the calculation of the *cost-to-go function* at each stage can be done as:

$$B_i\left(W(t_i),t_i\right)=\max_{\mathbf{u}_{\min}\le\mathbf{u}_k\le\mathbf{u}_{\max}}\left\{f_i\left(W(t_i),\mathbf{u}_i\right)+B\left(W(t_{i+1}),t_{i+1}\right)\right\} \tag{3}$$

Once obtained the *cost-to-go function*, represents a convenient vehicle to obtain the optimal solution for the general stage.

By continuing the *cost-to-go iteration* of (Eq. (2)) until convergence within the procedure it can be seen that the infinite horizon *cost-to-go function* B_∞, satisfying the following "Bellman's equation" can be obtained:

$$B_\infty\left(W\right)=\max_{\mathbf{u}}\left\{f\left(W,\mathbf{u}\right)+B\left(W,\mathbf{u}\right)\right\} \tag{4}$$

Unfortunately, in very few cases the problem can be solved through the stagewise optimization in order to analytically obtain a closed-form expression for the *cost-to-go problem*. The conventional numerical approach to the problem involves gridding the state space, calculating and storing the *cost-to-go* for each grid points as one marches backward from the first (or last) stage to the last (first). For an invite horizon problem the number of iterations required for convergence can be very large. Such an approach is seldom practically feasible due to the exponential growth of the computation with respect to the state dimension.

The traditional approach for solving the Bellman's equation involves gridding of the state space, solving the optimization (Eq. (1)) for each grid point and performing the stagewise optimization until convergence is achieved. The comprehensive sampling of the state space can be avoided by identifying the relevant regions of the state space by simulation under judiciously chosen suboptimal policies [9–11, 13, 15, 18].

The policy improvement theorem states that a new policy that is greedy (a greedy policy is one whose current cost is the least) with respect to the *cost-to-go function* of the original policy is as good as or better than the original policy, so the new policy can be defined as follows:

$$\mathbf{u}(W)=\arg\max_{\mathbf{u}}\ f\ \ (\mathbf{W},\mathbf{u})+B(\mathbf{W},\mathbf{u}) \tag{5}$$

where $\arg G(u, x, i) \in R^{m+n+r}$ is an improvement over the original policy and $\mathbf{u} \in R^m$, $\mathbf{W} \in R^n$ and $i \in R^r$.

When the new policy is as good as the original policy the above equation becomes the same as Bellman's equation (Eq. (3)).

The relevant regions of the state space are identified by simulation of *NDP* control and the initial suboptimal *cost-to-go function* is calculated from the simulation data. In this survey a functional approximant is used to interpolate between this data. The improvement is obtained through the iteration of the Bellman's equation. When the iteration converges, this off-line computed *cost-to-go function* can be used for an on-line optimal control calculation [19].

NDP uses neural network approximations for the approximation of *cost-to-go function*. The *cost-to-go function* was not used to generate an explicit control law; instead, it was used in an *on-line* optimization to reduce the large (or infinite) horizon problem to a relatively short

horizon problem. The method was found to be robust to approximation errors. Both deterministic (step changes in kinetic parameters) and stochastic problems (random variations in kinetic parameters and feed composition) were explored [11, 12, 17].

The following notations are used for description of the algorithm:

- B – Bellman's equation;
- $\tilde{B}(x)$ – approximated Bellman's equation corresponding to state W;
- $()^i$ – iteration index for cost iteration loop;
- k – discrete time.
- $\tilde{B}(k) \equiv \tilde{B}(W(k))$ and $f(k) = f\left(W(k), \mathbf{u}(k)\right)$.

The *NDP* algorithm flowchart is shown in Figure 1. The general simulation-approximation scheme involves computation of the converged *cost-to-go approximation* off-line. The architecture of the scheme can be seen in Figure 1. Step 1, Step 2, Step 3 and Step 4 represent the "Simulation part", while Step 5 and Step 6 – the "Cost Approximation Part".

The steps that describe the general procedure of *NDP* algorithm are:

1. Performing of simulations of the process with chosen suboptimal policies under all representative operating conditions. Starting with a given policy (a rule for choosing a decision *u* at each possible state *i*), and approximately evaluate the cost of that policy (as a function of the current state) by least-squares-fitting a scoring function to the results of the many simulated system trajectories using that policy.
2. Calculation of the ∞-horizon *cost-to-go* for each state visited during the simulation, using the simulation data. The solution of *one-stage-ahead cost* plus *cost-to-go problem* results in the improving of the cost values. *Cost-to-go* is the sum of the single state cost from the next point to the end of the horizon: $B(k) = \sum_{i=k+1}^{N}$.
3. The deviation, which is a result of the optimality, depends on a variety of factors, among which the principal one is the ability of the architecture $\tilde{B}^i(W)$ to accurately approximate the cost functions of the various policies.
4. A new policy is then defined by minimizing Bellman's equation where the optimal cost is replaced by the calculated scoring function and the process repeats. This type of algorithm typically generates a sequence of policies that eventually oscillate in a neighborhood of an optimal policy.
5. Fitting a neural network function approximant to the data to approximate the *cost-to-go function* as a smooth function of the states.
6. As described above the improved costs are again fitted to a neural network, to obtain subsequent iterations $\tilde{B}^1(k)$, $\tilde{B}^2(k)$, and so on, until convergence is achieved.
7. Policy update may sometimes be necessary to increase the coverage of the state space. In this case more suboptimal simulations with the updated policy are used to increase the coverage or the number of the data points in certain region of the state space.

Assuming that the optimization procedure starts with a pretty good approximation, the difference between the estimated value of the Bellman's function $\tilde{B}^{i+1}(x)$ in stage i and predicted value, based on the simulation results should converge fast enough. If they are accurate, the difference would be zero. These differences are used to adjustment the vector of the control variables. In this way, a control sequence is generated, as the decision begins to fluctuate in the vicinity of the optimal control.

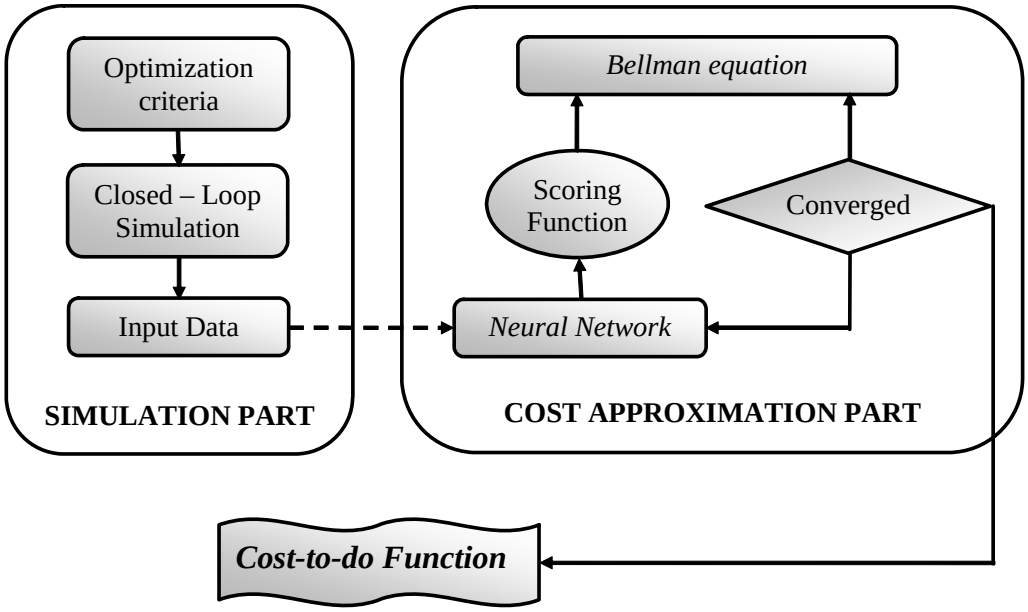

Figure 1: NDP algorithm block-scheme

Improvement of the *cost-to-go* is obtained through the iterations of the Bellman's equation (Eq. (13)). This method is known as a value iteration. The solution of the *one-stage-ahead* cost plus *cost-to-go problem*, results in the improvement of the cost values. The improved prices were again fitted to the neural network described above to obtain subsequent iterations $\tilde{B}^1(k)$, $\tilde{B}^2(k)$ and so on, until they have converged. Cost is said to be "*converged*" if the sum of the absolute error is less than 5% of the maximum cost.

2 The generalized net model

The GN model of *NDP* algorithm is presented in Figure 2.

In the transition Z_1 input places are l_1, l_2 and l_3. The initial token characteristics are as follows:

- in place l_1 – "*optimization criterion*";
- in place l_2 – "*discrete mathematical model with initial model conditions*";
- in place l_3 – "*initial vector of control variables*".

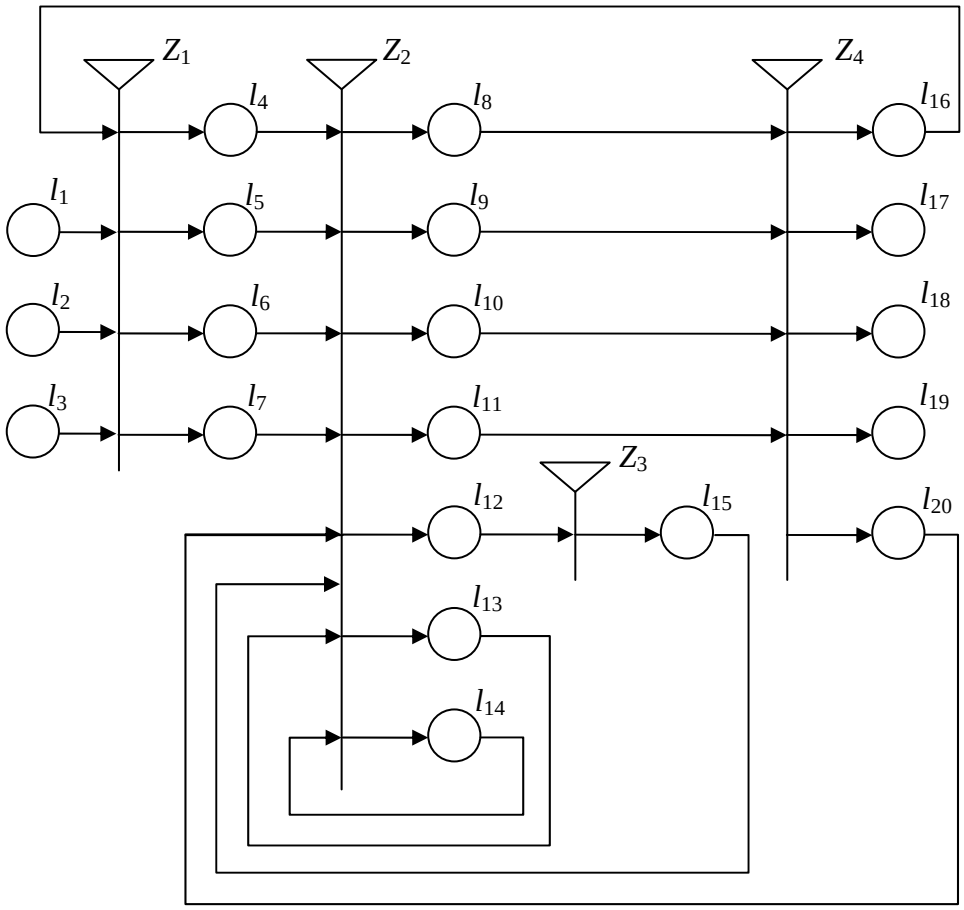

Figure 2: Generalized net model of NDP algorithm

The transition Z_1 has the following form:

$$Z_1 = \langle \{l_1, l_2, l_3, l_{16}\}, \{l_4, l_5, l_6, l_7\}, r_1, \wedge (l_1, l_2, l_3, l_{16}) \rangle$$

where:

$r_1 =$

	l_4	l_5	l_6	l_7
l_1	*false*	*true*	*true*	*false*
l_2	*true*	*false*	*true*	*false*
l_3	*true*	*true*	*true*	*false*
l_{16}	*true*	*false*	*true*	*true*

The transition Z_1 presents the first stage of Bellman's equations evaluations and the transition Z_2 presents the second stage, where the Bellman's equations are calculated best on previous one and current Bellman's equations evaluations.

In place l_4 the token obtains no new characteristic. In place l_5 the token obtains a characteristic "*optimization criterion value based on data in place* l_2". In place l_6 the token obtains a characteristic "*calculated Bellman's equations*". In place l_7 the token obtains a characteristic "*calculated control value*".

The transition Z_2 has the following form:

$$Z_2 = \langle \{l_4, l_5, l_6, l_7, l_{13}, l_{14}, l_{15}, l_{20}\}, \{l_8, l_9, l_{10}, l_{11}, l_{12}, l_{13}, l_{14}\}, r_2, \wedge(\wedge(l_4, l_5, l_6, l_7), \vee(\wedge(l_{13}, l_{14}), l_{20}))\rangle$$

where:

$r_2 =$

	l_8	l_9	l_{10}	l_{11}	l_{12}	l_{13}	l_{14}
l_4	*false*	*true*	*false*	*true*	*false*	*true*	*false*
l_5	*false*	*true*	*true*	*false*	*true*	*false*	*false*
l_6	*false*	*false*	*true*	*false*	*false*	*false*	*true*
l_7	*true*	*false*	*false*	*true*	*false*	*false*	*false*
l_{13}	W_1	W_1	W_1	W_1	*false*	*false*	*false*
l_{14}	*false*	*false*	W_1	*false*	W_2	*false*	*false*
l_{15}	*false*	*false*	*true*	*true*	*false*	*false*	*true*
l_{20}	*false*	*false*	*false*	W_1	*false*	*false*	*true*

- W_1 = "*the next iteration is performed*";
- W_2 = "*there is a need of a neural network Bellman's function approximator*".

The tokens obtain the following new characteristics:

- in place l_8 – "*current system performance*";
- in place l_9 – "*current optimization criterion value*";
- in place l_{10} – "*current Bellman's functions*";
- in place l_{11} – "*current control value* ";
- in place l_{12} – "*value of Bellman's functions for a neural network approximation*";
- in place l_{13} – "*stored system performance*";
- in place l_{14} – "*stored calculated Bellman's functions*".

The transition Z_3 presents the fitting of the Bellman's function to a neural network. The transition has the following form:

$$Z_3 = \langle \{l_{12}\}, \{l_{15}\}, r_3, \wedge(l_{12}) \rangle$$

where:

$r_3 =$

	l_{15}
l_{12}	*true*

The token obtains the new characteristic in place l_{15} "*approximated Bellman's function*".

The transition Z_4 has the following form:

$$Z_4 = \langle \{l_8, l_9, l_{10}, l_{11}\}, \{l_{16}, l_{17}, l_{18}, l_{19}, l_{20}\}, r_4, \wedge(l_8, l_9, l_{10}, l_{11})\rangle$$

where:

$r_4 =$

	l_{16}	l_{17}	l_{18}	l_{19}	l_{20}
l_8	*true*	W_3	*false*	*false*	*false*
l_9	*false*	*false*	W_3	*false*	*false*
l_{10}	*true*	*false*	*false*	W_3	*true*
l_{11}	*true*	*false*	*false*	W_3	*false*

where W_3 = "*NDP algorithm end*".

After the last transition the tokens obtain the following new characteristics:

- in place l_{16} – "*control value*";
- in place l_{17} – "*system performance at the end of the NDP algorithm*";
- in place l_{18} – "*optimization criterion value at the end of the NDP algorithm*";
- in place l_{19} – "*control value at the end of the NDP algorithm*";
- in place l_{20} – "*current Bellman's functions*".

3 Conclusions

A technique based on *NDP* approach has been illustrated as an object for description with the apparatus of GN.

The technique from suboptimal heuristic laws of *NDP* has been applied to identify appropriate regions and to initialize the gain-to-go approximation. The gain-to-go approximation was after that improved by performing iterations of Bellman's equation only over the appropriate regions. This technique provides an approximately optimal control operation for various initial conditions without requiring recalculation of the gain-to-go function.

Realization of such an optimal control approach combined with advanced control techniques (artificial neural networks with classical optimization method) in practice can lead to value and elaboration time reduction in the laboratory fed-batch bioprocesses, and not only that, but also to an elaboration time reduction technique for optimal control.

The GN model presented here confirms the apparatus of GN as a very appropriate tool for the modeling and description of complex algorithms and processes.

Acknowledgements

This work is partially supported by the National Science Fund of Bulgaria under Grant DID-02-29/2009 "Modeling Processes with Fixed Development Rules (ModProFix)".

References

[1] Atanassov, K. *Generalized Nets and Systems Theory*, "Prof. M. Drinov" Academic Publishing House, Sofia, 1997.

[2] Atanassov, K. *Generalized Nets*, World Scientific, Singapore, 1991.

[3] Atanassov, K. *On Generalized Nets Theory*, "Prof. Marin Drinov" Academic Publishing House, Sofia, 2007.

[4] Atanassov, K., H. Aladjov, *Generalized Nets in Artificial Intelligence, Vol. 2: Generalized Nets and Machine Learning*, "Prof. Marin Drinov" Academic Publishing House, Sofia, 2000.

[5] Aladjov, H., K. Atanassov, A generalized net for genetic algorithms learning, *Proc. of the XXX Spring Conf. of the Union of Bulgarian Mathematicians*, Borovets, Bulgaria, April 8–11 2001, 242–248.

[6] Roeva, O., T. Pencheva, Generalized net model of *Brevibacterium flavul* 22LD fermentation process, *Int. J. Bioautomation*, Vol. 2, 2005, 17–23.

[7] Bertsekas, D., J. Tsitsiklis, *Neuro-dynamic Programming* (1st ed.), Athena Scientific, Belmont, MA, 1996.

[8] Driessens, K., S. Dzeroski, Integrating guidance intro relational reinforcement learning, *Mach. Learn.*, Vol. 57, 2004, 217–304.

[9] Barto, A. G., S. J. Bradtke, S. P. Singh, Real-time learning and control using asynchronous dynamic programming, *Technical Report (TR-91-57)*, Amherst, MA, 1991.

[10] Sutton, R. S. Learning to predict by the methods of temporal Differences, *Mach. Learn.*, Vol. 3, 1988, 9–44.

[11] Kaisare, N. S., J. M. Lee, J. H. Lee, Simulation based strategy for nonlinear optimal control: Application to a microbial cell reactor, *Internat. J. Robust Nonlinear Control*, Vol. 13, 2003, 347–363.

[12] Soni, A. *A Multi-scale Approach to Fed-batch Bioreactor Control*, University of Pittsburgh Press, PA, 2002.

[13] Vlachos, D. G., A. B. Mhadeshwar, N. S. Kaisare, Hierarchical multiscale model-based design of experiments, catalysts and reactors for fuel processing, *Computer & Chemical Engineering*, Vol. 30, 2006, 1712–1724.

[14] Lee, J. M., N. S. Kaisare, J. H. Lee, Choice of approximator and design of penalty function for an approximate dynamic programming based control approach, *J. Process. Contr.*, Vol. 16, 2006, 135–156.

[15] Ilkova, T. Using of dynamic and rollout Neuro-dynamic programming for static and dynamic optimization of a fed-batch fermentation process, *Int. J. Bioautomation*, Vol. 3, 2005, 29–35.

[16] Ilkova, T., St. Tzonkov, Optimal control of a fed-batch Fermentation process by Neuro-dynamic programming, *Int. J. Bioautomation*, Vol. 1, 2004, 57–66.

[17] Xiong, Zh., J. Zhang, Neural network model-based on-line re-optimisation control of fed-batch processes using a modified iterative dynamic programming algorithm, *Chem. Eng. Process.*, Vol. 44, 2005, 477–484.

[18] Lee, J. M., J. H. Lee. An approximate dynamic programming approach based approach to dual adaptive control, *J. Process. Contr.*, Vol. 19, 2009, 859–864.

[19] Tosukhowong, T., J. H. Lee, Approximate dynamic programming based optimal control applied to an integrated plant with a reactor and a distillation column with recycle, *AIChE J.*, Vol. 55, 2009, 919–930.