OPTIMIZATION OF A FED-BATCH BIOREACTOR USING SIMULATION-BASED APPROACH

Catalina Valencia Peroni [*] **Jay H. Lee** [**,1]
Niket S. Kaisare [**]

[*] *Universitat Rovira I Virgili, Tarragona, Catalunya, Spain*
[**] *School of Chemical Engineering, Georgia Institute of Technology, Atlanta, GA 30332, U.S.A.*

Abstract: We use simulation-based approach to find the optimal feeding strategy for cloned invertase expression in *Saccharomyces cerevisiae* in a fed-batch bioreactor. The optimal strategy maximizes the productivity and minimizes the fermentation time. This procedure is motivated from Neuro Dynamic Programming (NDP) literature, wherein the optimal solution is parameterized in the form of a cost-to-go or profit-to-go functions. The proposed approach uses simulations from a heuristic feeding policy as a starting point to generate the profit-to-go vs state data. An artificial neural network is used to obtain profit-to-go as a function of system state. Iterations of Bellman equation are used to improve the profit function. The profit-to-go function thus obtained, is then implemented in an online controller, which essentially converts infinite horizon problem into an equivalent one-step-ahead problem.

Keywords: Fed-batch Reactor, Optimal Control, Neuro-Dynamic Programming

1. INTRODUCTION

A vast majority of industrially important fermentors are operated in fed-batch mode, which involves addition of substrates continuously to an otherwise batch reactor. Fed-batch reactors are especially useful when growth or metabolite production follows substrate or product inhibition kinetics. In such cases, substrates need to be added in a controlled manner in order to maximize the productivity with respect to the desired product. For example, if product formation is inhibited under excess substrate conditions, low substrate concentrations are desired for high product yields. At the same time, higher substrate levels are required to prevent starvation of cells and to maintain high growth rates. Thus, there exists an optimum feed profile that maximizes the productivity of the process.

The determination of optimal feed profile is a challenging problem, as the bioreactors follow complex nonlinear dynamics. Resulting optimal control problem is therefore a nonlinear programming (NLP) problem accompanied by various input and state constraints. The optimization is often non-convex and the global optimum difficult to achieve (Banga *et al.*, 1997).

Many authors have used Pontryagin's maximum principle to solve the optimal control problem. However, this approach may be very difficult for such problems; so several alternative solution techniques have been proposed. Cuthrell and Biegler (1989) solved the optimal control problem of a fed-batch penicillin reactor using successive quadratic programming (SQP) based on orthogonal collocation on finite elements. Luus (1993) used iterative dynamic programming (IDP) to find optimal feed profile for the same reactor; while Banga *et al.* (1997) presented a fast stochastic dynamic optimization method — called in-

[1] The author to whom correspondence should be addressed. Email: jay.lee@che.gatech.edu

tegrated controlled random search for dynamic systems, ICRS/DS — for this reactor. Recently, Bonvin *et al.* (2002) provided a good review of dynamic optimization methods for batch processes, and also presented algorithms that achieve nearly optimal performance in the presence of uncertainties.

In this paper, optimal control of a fed-batch fermentor for cloned invertase expression in *Saccharomyces cerevisiae* is considered. The expression of the enzyme is repressed at high glucose concentration. Hence, a fed-batch reactor is ideal for this process (Patkar and Seo, 1992). Patkar *et al.* (1993) proposed a model for this fermentation process, involving a set of four coupled ODEs. They used first-order conjugate gradient method in order to find the optimal feed rate profile for this system. Later, Chaudhuri and Modak (1998) used a neural network model into the generalized reduced gradient method for the same optimization problem.

The main disadvantage of the above methods is that the fermentation ending time should be fixed a priori in both cases. In order to find the optimal fermentation ending time, several different fermentation ending times should be guessed and, for each one of them, one productivity optimization problem should be solved. This is extremely computationally demanding. Methods such as control parameterization may be used for free-end-time problems to obtain open loop optimal policies. Another drawback of these methods stems from the fact that they are open loop optimal, which means that each time an initial condition changes, a new and different optimization problem should be solved. Besides, the resulting fixed policies do not take into account the possible process disturbances.

Dynamic Programming (DP) is an optimization method that can be used to overcome these limitation. It was introduced by Bellman and coworkers (Bellman, 1957) as a feasible approach to solve the dynamic optimization that results from an optimal control problem. Here, the aim is to find the optimal *'cost-to-go'* function, which can be used to parameterize the optimal solution as a function of the system state. This method is promising as presents a feasible approach to solve any optimal control problem. However, it suffers from *'curse of dimensionality'*, which refers to exponential increase in computational cost with increase in state dimension.

Recently, Neuro-Dynamic Programming (NDP) was proposed as a way to alleviate the curse of dimensionality (Bertsekas and Tsitsiklis, 1996). NDP uses simulated process data obtained under suboptimal policies to fit an approximate cost-to-go function – usually by fitting artificial neural networks, hence the name. The initial approximate cost-to-go function is further improved by an iteration procedure based on the so called Bellman equation. Closely related to NDP are the methods in AI literature, collectively classified as Reinforcement Learning (RL, Sutton and Bartow (1998)). We applied this approach to online control of a continuous bioreactor (Kaisare *et al.*, 2002) as well as that of a benchmark *Van der Vusse* reactor (Lee and Lee, 2001). Simulation-based NDP method will be applied for optimal control of the fed-batch *Saccharomyces cerevisiae* fermentor. We have considered a deterministic optimization problem, i.e. the model is known and full state feedback is assumed. The method can be expanded to stochastic case. Moreover, simulation-based NDP methods do not require the explicit model to be known—methods like Q-learning (Watkins and Dayan, 1992) developed in RL community are model-free stochastic learning techniques.

2. STATEMENT OF THE OPTIMAL CONTROL PROBLEM

Our objective is to find an optimal feed profile μ that adapts itself when initial conditions change or when disturbances occur. The optimal policy is a function of the system state, represented as

$$\mu(x) = \arg\max_{u,t_f} \{productivity - \lambda \cdot t_f\} \quad (1)$$

subject to relevant input and state constraints and following system dynamics. λ is a positive constant that penalizes invertase fermentation time.

In general, the performance index is mathematically represented as the sum of stage-wise costs incurred (or rewards obtained) until the end of horizon.

$$J(x_k) = \sum_{i=k}^{t_f} \phi(x_i, u_i) + \bar{\phi}_t(x_{t_f}) \quad (2)$$

subject to

$$\text{Path Constraint:} \quad g_i(x_i, u_i) \geq 0$$
$$\text{Model Constraint:} \quad \dot{x} = f(x, u)$$

Here ϕ is the stage-wise cost/reward and $\bar{\phi}_p$ is the terminal cost/reward. The path constraints include all input and state constraints. The system dynamics appear as model constraints. For discrete-time system, the model $f(x, u)$ can be integrated for one time step. Equivalently, model constraint becomes $x_{k+1} = f_h(x_k, u_k)$.

The above problem is an infinite-horizon control problem, as we wish to solve free terminal time optimal control (i.e. t_f is not fixed).

3. NEURO-DYNAMIC PROGRAMMING

We begin this section with the description of Dynamic Programming (DP), which is an elegant way to solve the previously introduced optimization problem. In this method, the process is modeled as a chain of consecutive transitions from one state to another. The way each transition is made depends on the decision variable, which has an associated cost or reward. The objective of dynamic programming is to maximize the total profit or minimize the total cost, obtained from the transitions needed to reach the final desired process state from initial process state.

DP involves stage-wise calculation of the *profit-to-go* function [2] to arrive at the solution, not just for a specific x_0 but for general x_0. Thus, the objective function is split into two parts — one stage reward obtained in going from x_k to x_{k+1}, and total future rewards as a function of x_{k+1}. The latter is parameterized as profit-to-go function, which represents the "desirability" of state x_{k+1}. Using the profit-to-go function, the multi-stage optimization problem is cast into an equivalent single-stage optimization that is solved online.

$$\max_{u_k} \{\phi(x_k, u_k) + J^*(x_{k+1})\} \tag{3}$$

where the optimal profit-to-go function is at each stage is defined as

$$J^* = \max_u \left\{ \sum_{i=k+1}^{t_f} \phi(x_i, u_i) + \bar{\phi}_t(x_{t_f}) \right\} \tag{4}$$

The crucial step in DP is calculation of the profit-to-go function $J^*(x)$. This involves stage-wise evaluation of rewards obtained in all possible transitions from each state in the state space. Without going into further details, we mention here that this approach is seldom practically feasible due to the exponential growth of the computation with respect to state dimension. This is referred to as the 'curse of dimensionality', which must be removed in order for this approach to find a widespread use.

An alternative method, which uses simulations to overcome the curse of dimensionality, involves computation of an approximation of the profit-to-go function. Exhaustive sampling of state space can be avoided by identifying relevant regions of the space through simulation under judiciously chosen suboptimal policies. The *policy improvement theorem* states that the new policy defined by $\mu(x) = \arg\max_u \left\{\phi(x,u) + J^i(f_h(x,u))\right\}$ is an improvement over the original policy (Howard, 1960). Indeed, when the new policy is as good as the original policy, the above equation becomes the same as Bellman optimality equation (5).

$$J_\infty(x) = \max_u \{\phi(x,u) + J_\infty(f_h(x,u))\} \tag{5}$$

Equivalently, for a free-end-time batch optimization problem, the above equation can be formulated as

$$J_\infty(x) = \max \left[\bar{\phi}_t(x), \max_u \{\phi(x,u) + J_\infty(f_h(x,u))\}\right] \tag{6}$$

where the modification accounts for termination of batch at a state x. In other words, if $\bar{\phi}_t(x)$ is greater than the second term, the batch should be terminated.

Use of the Bellman equation to obtain iterative improvement of cost-to-go approximator forms the crux of various methods like Neuro-Dynamic Programming (NDP) (Bertsekas and Tsitsiklis, 1996), Reinforcement Learning (RL) (Sutton and Bartow, 1998), Temporal Difference (Tsitsiklis and Roy, 1997) and such.

In this paper, the basic idea from NDP and RL literature is used to obtain optimal performance of a fed-batch bioreactor. Relevant regions of the state space are identified through *simulations* of several heuristic policies, and initial suboptimal profit-to-go function is calculated from the simulation data. A *functional approximator* is used to interpolate between these data points. Neural network is the chosen function approximator (hence the name 'Neuro' in NDP). *Evolutionary improvement* is obtained through iterations of the Bellman equation (7). When the iterations converge, this offline-computed profit-to-go approximation is then used for online optimal control calculation for the reactor.

3.1 *Algorithm*

A detailed algorithm was presented in our previous work (Kaisare *et al.*, 2002), which is reproduced in this section. Following notations are used in this section and rest of the paper. J represents profit-to-go values. A function approximation relating J to corresponding state x is denoted as $\tilde{J}(x)$. Superscript $()^i$ represents iteration index for cost iteration loop and k is discrete time. Finally, $\tilde{J}(k) \equiv \tilde{J}(x(k))$ and $\phi(k) \equiv \phi(x(k), u(k))$.

The simulation-based approach involves computation of the converged profit-to-go approximation offline. The following steps describe the general procedure for the infinite horizon profit-to-go approximation.

[2] It is customary in DP to use cost-to-go for minimization problem. We use profit-to-go as we solve maximization problem. Both cost and profit are exactly equivalent

(1) Perform simulations of the process with chosen suboptimal policies under all representative operating conditions.
(2) Using the simulation data, calculate the ∞-horizon profit-to-go for each state visited during the simulation. For example, each closed loop simulation yields us data $x(0), x(1), \ldots, x(t_f)$, where t_f is the termination time for the specific suboptimal policy. For each of these points, compute the profit-to-go value.
(3) Fit a neural network or other function approximator to the data to approximate the profit-to-go function — denoted as $\tilde{J}^0(x)$ — as a smooth function of the states.
(4) To improve the approximation, perform the following iteration (referred to as the cost or value iteration) until convergence:
    - With the current profit-to-go approximation $\tilde{J}^i(x)$, calculate $J^{i+1}(k)$ for the given sample points of x by solving

$$J^{i+1} = max\left[\bar{\phi}_t(x_k) \max_u \{\phi(x_k, u_k) + \tilde{J}^i(f_h(x_k, u_k))\}\right] \quad (7)$$

      which is based on the Bellman Equation.
    - Fit an improved cost-to-go approximator $\tilde{J}^{i+1}$ to the x vs. $J^{i+1}(x)$ data.
(5) **Policy Update** may sometimes be necessary to increase the coverage of the state space. In this case, more suboptimal simulations with the updated policy ($\max_u \phi + \tilde{J}^i$) are used to increase the coverage or the number of data points in certain region of state space.

Once the value iteration described above converges, the resulting profit-to-go function can be used for online control. At each time, state estimates are obtained (for deterministic problem, we have the state itself). Single-stage optimization problem, as shown in Eq. (3), is solved using the converged profit-to-go function $\tilde{J}_a$ in place of the optimal J^*.

4. INVERTASE PRODUCTION OPTIMIZATION

The fermentation process considered here consists of production of invertase in *Saccharomyces cerevisiae* utilizing glucose for growth. Patkar and Seo (1992) reported the fermentation kinetics of invertase production in fed-batch cultures; as well as experimental data for cell density (c, expressed as optical density OD), glucose concentration (s gm/L) and specific invertase activity (i) obtained with various glucose feeding strategies in 1.2 liter bioreactor. The productivity of the reactor at any time is given by

$$productivity = icV \quad (8)$$

Thus the optimization problem as defined in Eq. (1) becomes

$$\max_{u,t_f} \left\{ icV|_{t_f} - \lambda t_f \right\} \quad (9)$$

with constraints $u \in [0, 0.2722]$ and $V \leq 1.2$. The model equations are summarized in appendix A.

4.1 *Obtaining profit-to-go function*

The first step in simulation-based NDP optimization is to obtain a suboptimal profit-to-go function. This was done through simulations of a set of suboptimal heuristic policies. The heuristics were selected so that the 4-dimensional state space was sufficiently covered. The policies involved maintaining the reactor at initial volume $V(0)$ for certain amount of time t_i and then increasing the feed rate until end of fermentation time or until constraints are met (if V_{max} is reached before t_f, feed rate is reduced to $u = 0$ and reactor is operated in batch mode until $t = t_f$). Mathematically, the feed rate profiles followed were

$$u(t; t_i, b) = \begin{cases} 0 & \text{if } t < t_i \\ 0.02(1 + b(t - t_i)^{2.2}) & \text{if } t \geq t_i \end{cases}$$

We used four values of $b = [0.05, 0.07, 0.1, 0.13]$ and nine values of $t_i = [1, 2, \ldots, 9]$ to generate 36 different heuristic policies. Each of these policies were implemented for three different initial values of $V(0) = [0.4, 0.6, 0.8]$.

For each of the heuristic policies, the optimal ending time was determined — as the time that yielded the maximum profit value calculated according to Eq. (A.5). The productivity thus calculated gives the terminal reward function $\bar{\phi}_t = icV|_{t_f}$. Stage-wise reward is given by $\phi(x_k) = \lambda \Delta t_k$. Thus, for each of the states corresponding to a specific heuristic policy, the profit-to-go function is calculated as

$$J(x_k) = icV|_{t_f} - \lambda \cdot (t_f - t_k)$$

where $t_k = k.\Delta t$ is the "current" time for x_k.

We obtained a total of 9328 states and corresponding profit-to-go values through simulation of the heuristic policies. Next, a function approximator was used to correlate the profit-to-go as a function of system state. A backpropagation neural network was used that had 4 input nodes (corresponding to 4 state variables), 1 output node (profit-to-go) and two hidden layers with 17 and 5 nodes respectively. We denote this approximation as $\tilde{J}^0$.

Policy	Profit	$icV\|_{t_f}$	t_f
Patkar (1993)	3.70	7.30	12
Chaudhuri (1998)	3.50	7.10	12
Best heuristic	3.72	7.23	11.7
NDP	3.80	7.25	11.5

Table 1. Optimal control results for $V(0) = 0.6$.

Given this initial approximation of profit function, Bellman equation was used iteratively to improve the optimality of the profit-to-go approximation. For each of the 9328 data points, following equation was solved

$$\tilde{J}^{i+1} = \max \Big[icV|_{x_k} \max_{u_k} \Big\{ -\lambda \Delta t_k + \tilde{J}^i(x_{k+1}) \Big\} \Big]$$

Here, superscript i represents the iteration index. Δt represent time steps, which were taken to be constant and equal to 0.1 hours. Value iterations were performed as elaborated in section 3.1. Three iterations were required for the profit function to converge with 1-norm less than 0.2. The best neural network structures for each of the iterations were 4-17-5-1, 4-17-5-1 and 4-13-5-1 respectively.

Simulations using the converged profit-to-go approximator resulted in visits to regions of state space previously unvisited by heuristic controller. Hence, policy update was performed by including these unvisited states, and value iteration of Bellman equation was performed again. It took just one iteration for the profit function to converge. The neural network structure was found to be 4-15-5-1, and this $\tilde{J}^4$ was used for control.

5. RESULTS

The fourth trained neural network was implemented into online optimal control. The reactor was started with $V(0) = 0.6$. This was the case solved by Patkar *et al.* (1993) and Chaudhuri and Modak (1998). The results are shown in Table 1. State space plots for online controller performance are shown in Fig. 1. It can be seen from the figure that the optimal policy results from interpolation in the state space and not in the policy space.

The controller was then tested with different initial volume $V(0) = 0.5$. The results are shown in the second column in Table 2. This condition was not "seen" before by the function approximator. Next, we tested the controller in presence of unknown disturbance: abrupt cell death occurs at 9 h resulting in a 50% decrease in cell concentration. The NDP method still gives most optimal performance over the other methods. The best heuristic reported above is the heuristic that gave maximum profit value. It should be noted that

Policy	Profit $V_0 = 0.5$	Profit Cell death
Patkar *et al.* (1993)	3.74	0.62
Best heuristic	3.58	1.88
NDP	4.06	1.97

Table 2. Control results for a different V_0 and unknown disturbance cases

we found different heuristics to be the best for different $V(0)$ values. Thus, the optimal controller does not select any particular heuristic policy; instead it "patches" solution of various heuristics to come with a different, optimal policy.

6. CONCLUSIONS

A simulation-based Neuro-Dynamic Programming (NDP) strategy was applied to obtain optimal feeding profile for different initial conditions for invertase production in a fed-batch bioreactor. Simulation from suboptimal heuristic laws is used to identify relevant regions of the state space and to initialize the profit-to-go function approximation. The profit-to-go approximator is then improved by performing iterations of Bellman equation over only the relevant regions of the state space. The profit-to-go function thus obtained was then used for online optimal control. This method gives nearly optimal performance for different initial conditions without requiring to recompute the profit-to-go function. This method therefore has a promise in controlling fed-batch reactors in presence of disturbances.

7. REFERENCES

Banga, J. R., A. A. Alonso and R. P. Singh (1997). Stochastic dynamic optimization of batch and semicontinuous bioprocesses. *Biotechnology Progress* **13**, 326–335.

Bellman, R. E. (1957). *Dynamic Programming*. Princeton University Press. New Jersey.

Bertsekas, D. P. and J. N. Tsitsiklis (1996). *Neuro-Dynamic Programming*. Athena Scientific. Belmont, MA.

Bonvin, D., B. Srinivasan and D. Ruppen (2002). Dynamic optimization in the batch chemical industry. In: *Chemical Process Control–VI*. pp. 255–273.

Chaudhuri, B. and J. M. Modak (1998). Optimization of fed-batch bioreactor using neural network model. *Bioprocess Engineering* **19**, 71–79.

Cuthrell, J. E. and L. T. Biegler (1989). Simultaneous optimization and solution methods for batch reactor control profiles. *Computers and Chemical Engineering* **13**, 49–62.

Howard, R. (1960). *Dynamic programming and markov processes*. MIT Press. Cambridge Massachussets.

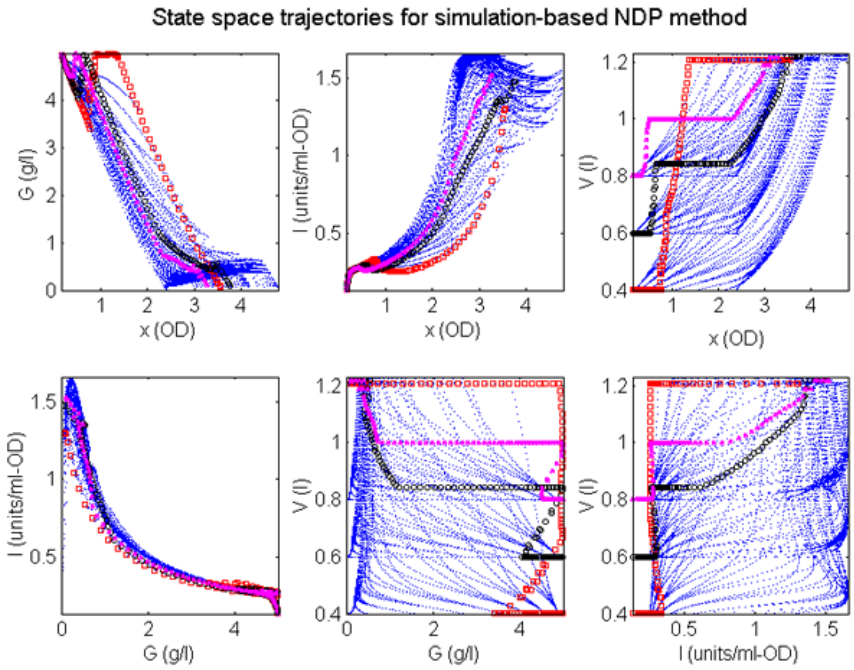

Fig. 1. State-space plot showing online performance for the three cases $V_0 = 0.4(\square)$, $V_0 = 0.6(\circ)$, and $V_0 = 0.8(\times)$. Dots represent the original data from heuristic feeding policies.

Kaisare, N. S., J. M. Lee and J. H. Lee (2002). Simultion based strategy for nonlinear optimal control: Application to a microbial cell reactor. Int. J. of Robust and Nonlinear Control, accepted for publication.

Lee, J. M. and J. H. Lee (2001). Neuro-dynamic programming method for mpc. In: *DYCOPS VI*. pp. 157–162.

Luus, R. (1993). Optimization of fed-batch fermentors by iterative dynamic programming. *Biotechnology and Bioengineering* **41**, 599–602.

Patkar, A. and J. H. Seo (1992). Fermentation kinetics of recombinant yeast in batch and fed-batch cultures. *Biotechnology and Bioengineering* **40**, 103–109.

Patkar, A., J. H. Seo and H. C. Lim (1993). Modeling and optimization of cloned invertase expression in *saccharomyces cerevisiae*. *Biotechnology and Bioengineering* **41**, 1066–1074.

Sutton, R. S. and A. G. Bartow (1998). *Reinforcement learning: an introduction*. MIT Press. Cambridge Massachussets.

Tsitsiklis, J. N. and B. Van Roy (1997). An analysis of temporal-difference learning with function approximation. *IEEE Transactions on Automatic Control* **42**, 674–690.

Watkins, C. J. C. H. and P. Dayan (1992). Q-learning. *Machine Learning* **8**, 279–292.

Appendix A. FED-BATCH REACTOR MODEL

Mass balance equations

$$(c\dot{V}) = (R_r Y_{cr} + R_f Y_{cf})\, cV \tag{A.1}$$

$$(s\dot{V}) = us_f - R_t cV \tag{A.2}$$

$$(i\dot{cV}) = (\pi - k_d i)\, cV \tag{A.3}$$

$$\dot{V} = u \tag{A.4}$$

where u is the feed rate, $c(OD)$, $s(g/L)$ and $i(units/OD/l)$ represent concentrations of cell, substrate and invertase respectively. The rate expressions are

$$R_r = \frac{0.55s}{0.05+s} \qquad R_t = \max\left\{\frac{1.25s}{0.95+s}, R_r\right\}$$

$$\pi = \frac{6.25s}{0.1+s+2s^2} \qquad R_f = R_t - R_r$$

$Y_{cr} = 0.6$, $Y_{cf} = 0.15$, $k_d = 1.85$.

Operating conditions: $c(0) = 0.15$, $s(0) = 5$, $i(0) = 0.1$, $s_f = 10$, $V(0) = \{0.4, 0.5, 0.6, 0.7, 0.8$.

Objective function:

$$\max_u \left\{ icV|_{t_f} - \lambda t_f \right\} \tag{A.5}$$

subject to constraints $0 \le u \le 0.2722$, $V \le 1.2$. Here, t_f is fermentation time, $\lambda = 0.3$ and sampling interval is $\Delta t = 0.1$.