Simulation Based Strategy for Nonlinear Optimal Control: Application to a Microbial Cell Reactor

Niket S. Kaisare   Jong Min Lee   Jay H. Lee*

niket.kaisare@che.gatech.edu   jongmin.lee@che.gatech.edu   jay.lee@che.gatech.edu

Department of Chemical Engineering

Georgia Institute of Technology

778 Atlantic Drive, Atlanta, GA 30332-0100

Phone: 404-385-2148

Fax: 404-894-2866

February 22, 2002

Submitted to: International Journal of Robust and Nonlinear Control

*The author to whom correspondence must be addressed

Abstract

Optimal control of systems with complex nonlinear behaviour such as steady state multiplicity results in a nonlinear optimization problem that needs to be solved online at each sample time. We present an approach based on simulation, function approximation and evolutionary improvement aimed towards simplifying online optimization. Closed loop data from a suboptimal control law, such as MPC based on successive linearization, is used to obtain an approximation of the '*cost-to-go*' function, which is subsequently improved through iterations of the Bellman equation. Using this offline-computed cost approximation, an infinite horizon problem is converted to an equivalent single stage problem — substantially reducing the computational burden. This approach is tested on continuous culture of microbes growing on a nutrient medium containing two substrates that exhibits steady state multiplicity. Extrapolation of the cost-to-go function approximator can lead to deterioration of online performance. Some remedies to prevent such problems caused by extrapolation are proposed.

Keywords: Optimal Control, Continuous Bioreactor, Multiple Steady States, Neuro-Dynamic Programming

1 Introduction

In optimal control, one is often faced with the task of solving nonlinear optimization problems — either off-line or more commonly on-line. An example is the popular method of Model Predictive Control (MPC) [1, 2, 3], which requires a nonlinear dynamic optimization problem cast over a prediction window to be solved at each sample time when applied on a nonlinear process model. Such problems also arise in batch processes where operating recipes that minimize the batch time or maximize the product quantity and quality are desired. Optimization problems involving nonlinear dynamic models are intrinsically hard problems and it is difficult to assure the attainment of quick, reliable solutions, which are needed in most of the practical applications. Difficulties exist even in off-line optimization, when the problem involves a model of high dimension and a large time window, thus yielding a large set of optimization variables and constraints. In practice, these problems are often solved in a highly approximate sense (by using a linear approximation of the model, for example) or are avoided by adopting heuristic policies instead.

One approach for solving dynamic optimization is Dynamic Programming (DP). Here, the aim is to find the optimal *'cost-to-go'* function, which can be used to parameterize the solution with respect to system state – either as a continuous function or as a lookup table – thereby simplifying the task of obtaining on-line solutions. However, the approach is largely considered impractical as analytical solution of resulting dynamic program is seldom possible and numerical solution suffers from the 'curse of dimensionality' [4].

Neuro-Dynamic Programming (NDP) was proposed as a way to alleviate the curse of dimensionality [5]. It 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. In value iteration approach of NDP, the initial approximate cost-to-go function is further improved by an iteration procedure based on the so called Bellman equation. In this context, the simulation's role is two-fold. First, by simulating the process under a reasonably chosen suboptimal policy and all possible operating parameters / disturbances, it provides a set of data points that define the relevant or "working" region in the state space. Second, the simulation provides the cost-to-go value under the suboptimal policy for each state visited, with which iteration of the Bellman equation can be initiated.

The NDP approach has received significant attention for its successes in several applications such as elevator dispatch problem and a program that plays Backgammon at the world championship level. Recently, we used NDP to develop a nonlinear model predictive controller for a benchmark *Van de Vusse reaction* system [6]. The NDP approach was used to improve the performance provided by an initial suboptimal policy through iterative 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 online optimization to reduce a 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.

In this paper, we seek to generalize this approach and apply it to a more complex problem involving a continuous bioreactor. We consider the growth of bacterial cells (such as *Klebsiella oxytoca*) on multiple substitutable nutrients. The cybernetic modeling framework developed by Ramkrishna and coworkers (see [7], [8] and references therein) is used to model the system. This non-linear system is characterized by multiple steady states and long-lived transient behavior, which demands that a nonlinear optimization cast over a large time window be solved at each sample time. In a previous conference presentation [9], we had applied a MPC method based on successive linearization of the nonlinear model [10] to this system. Long prediction and control horizons were shown to be necessary to control the reactor at desired steady state because of certain peculiar dynamics of the bioreactor, *e.g.*, quickly settling to an almost stable behavior that lasts for a long period followed by a sharp drift to another steady state, which is "triggered" by a change in the cells' metabolic states [1].

We seek to use this approach not only to reduce the on-line computational demand but also to improve the controller performance, through the use of cost-to-go approximator. A neural network is chosen as an approximator to obtain cost-to-go as a function of system states. While a properly trained neural network has good interpolation capabilities, one may not use it to extrapolate over regions of state space not covered during its training. Extrapolation by neural network is shown to result in deteriorated performance of the

[1]The microbes display different nutrient uptake patterns in the two steady states, indicating distinct internal cellular mechanisms

controller. In turn, we discuss possible remedies, either by confining the control calculations to the visited regions of the state space or by expanding coverage through additional simulations.

2 Preliminaries

2.1 Open Loop Optimal Control Problem

A general dynamic optimization problem commonly found in optimal control literature is as follows:

$$\min_{u_0,\ldots,u_{p-1}} \sum_{i=0}^{p-1} \phi(x_i,u_i) + \bar{\phi}(x_p) \tag{1}$$

with

$$\begin{aligned} \text{Path Constraint:} \quad & g_i(x_i,u_i) \geq 0, 0 \leq i \leq p-1 \\ \text{Terminal Constraint:} \quad & \bar{g}(x_p) \geq 0 \\ \text{Model Constraint:} \quad & \dot{x} = f(x,u) \end{aligned}$$

for a given initial state x_0 and a piecewise constant input $u(\tau) = u_i \; i \cdot h \leq \tau < (i+1) \cdot h$. In the above, h is the sampling time and x_i represents the value of x at the i^{th} sample time (*i.e.*, $x(t)$ at $t = h \cdot i$). ϕ is the single stage cost function and $\bar{\phi}$ is the terminal state cost function. Such a problem may be solved in the context of finding an open-loop input trajectory off-line for a fixed finite-time process (*e.g.*, a batch process). For a continuous system, the problem may be solved on-line at each sample time in order to find the optimal input adjustment for the given state – as in 'Receding Horizon Control' [1, 2, 3]. In the latter case, it is shown to be advantageous to solve an infinite horizon problem (in which p is set to infinity). Obviously, the computational load for solving the optimization increases horizon size, thereby limiting the usefulness of such an approach in practice, especially when non-linear system models are involved and the use of a large optimization window is required to obtain a satisfactory result.

A slight variation to the above is a free-end problem, which can be expressed as follows:

$$\min_{u_0,\ldots,u_{p-1},p} \sum_{i=0}^{p-1} p \cdot h + \sum_{i=0}^{p-1} \phi(x_i,u_i) + \bar{\phi}(x_p) \tag{2}$$

The same set of constraints are enforced here. The objective is to minimize some combination of the total span, and the stagewise and terminal costs subject to given path and terminal constraints.

2.2 Dynamic Programming and Bellman Equation

Dynamic Programming (DP) is an elegant way to solve the previously introduced optimization problem. It involves stagewise calculation of the *cost-to-go* function to arrive at the solution, not just for a specific x_0 but for general x_0. For (1), the cost-to-go at each stage is defined as

$$J_i = \min_{u_{p-i},\dots,u_{p-1}} \sum_{j=p-i}^{p-1} \phi(x_j, u_j) + \bar{\phi}(x_p) \tag{3}$$

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

$$J_i(x) = \min_u \phi(x, u) + J_{i-1}(F_h(x, u)), \tag{4}$$

where $F_h(x, u)$ denotes the resulting state after integrating the differential equation for one sample interval with the starting state of x and constant input of u. The above is sequentially solved from $i = 1$ through $i = p$ with the initialization of $J_0 = \bar{\phi}(x)$. Of course, the pertinent terminal / path constraints need to be imposed at each stage.

The cost-to-go function, once obtained, represents a convenient vehicle to obtain the optimal solution for a general state x_0. For example, in the Receding Horizon Control problem where one has to calculate u_0 for a given x_0 (the given state at a current time), one can solve the following equivalent one-stage problem with the terminal cost given by the cost-to-go function J_{p-1}:

$$\min_{u_0} \phi(x_0, u_0) + J_{p-1}(F_h(x_0, u_0)) \tag{5}$$

By continuing the cost-to-go iteration of (4) until convergence within the above procedure, we can see that the infinite horizon cost-to-go function J_∞ satisfying the following *'Bellman Equation'* can be obtained.

$$J_\infty(x) = \min_u \{\phi(x, u) + J_\infty(F_h(x, u))\} \tag{6}$$

In very few cases can we solve the stagewise optimization analytically to obtain a closed-form expression for the cost-to-go function. The conventional numerical approach to the problem involves gridding the state space, calculating and storing the cost-to-go for each grid point as one marches backward from the last stage to the first. 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 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.

We close this section by noting that the 'Bellman Equation' for the free-end problem of 2 can be written as

$$J(x) = \min_u \{h + \phi(x,u) + J(F_h(x,u))\} \tag{7}$$

2.3 A Simulation Based Alternative to Obtain Cost-to-go Approximation

The traditional approach to solving the Bellman equation involves gridding of the state space, solving the optimization (4) for each grid point, and performing the stagewise optimization until convergence. 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 a new policy that is greedy[2] with respect to the cost-to-go function of the original policy is as good as or better than the original policy; i.e. the new policy defined by $u(x) = \arg\min_u(\phi(x,u) + J^i(F_h(x,u))$ is an improvement over the original policy [11]. Indeed, when the new policy is as good as the original policy, the above equation becomes the same as Bellman optimality equation (6). 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) [5], Reinforcement Learning (RL) [12], Temporal Difference [13] and such.

In this paper, the basic idea from NDP and RL literature is used to improve the performance of a successive linearization based Nonlinear Model Predictive Control (NMPC) method [10] applied to a bioreactor. Relevant regions of the state space are identified through *simulations* of the NMPC control law, and initial suboptimal cost-to-go function is calculated from the simulation data. A *functional approximator* is used to interpolate between these data points. *Evolutionary improvement* is obtained through iterations of the

[2]A greedy policy is one whose current cost is the least.

Bellman equation (8). When the iterations converge, this offline-computed cost-to-go approximation is then used for online optimal control calculation for the reactor.

In the remainder of the paper, we refer to our proposed algorithm as *simulation-approximation-evolution* (S-A-E in short) scheme as this term captures the essence of the algorithm much better than NDP or RL.

2.4 The Algorithm

Following notations are used in this section and rest of the algorithm. J represents cost-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 - approximation - evolution scheme involves computation of the converged cost-to-go approximation offline. The architecture of the scheme is illustrated in figure 1. The following steps describe the general procedure for the infinite horizon cost-to-go approximation. Steps 1 and 2 represent the "Simulation Part", and 3 and 4 represent the "Cost Approximation Part" in the figure.

1. Perform simulations of the process with chosen suboptimal policies under all representative operating conditions.

2. Using the simulation data, calculate the ∞-horizon cost-to-go for each state visited during the simulation. For example, each closed loop simulation yields us data $x(0), x(1), \ldots, x(N)$, where N is sufficiently large for the system to reach equilibrium. For each of these points, one-stage cost $\phi(k)$ is computed (see equation (14) in section 4.1). Cost-to-go is the sum of single stage costs from the next point to the end of horizon — $J(k) = \sum_{i=k+1}^{N} \phi(i)$.

3. Fit a neural network or other function approximator to the data to approximate the cost-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 iteration) until convergence:

- With the current cost-to-go approximation $\tilde{J}^i(x)$, calculate $J^{i+1}(k)$ for the given sample points of x by solving

$$J^{i+1} = \min_u \phi(x,u) + \tilde{J}^i(F_h(x,u)) \tag{8}$$

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 ($\min_u \phi + \tilde{J}^i$) are used to increase the coverage or the number of data points in certain region of state space.

Assuming that one starts with a fairly good approximation of the cost-to-go (which would result from using a good suboptimal policy), the cost iteration should converge fairly fast — faster than the conventional stagewise cost-to-go calculation.

3 Microbial Cell Reactor With Multiple Steady States

Continuous bioreactors often display steady state multiplicity, and significant delayed responses to changes in the environment due to the tendency of living cells to switch metabolic states in response to environmental pressures. Steady state multiplicity is a condition in which a system displays two or more distinct states and output conditions for the same set of input conditions. Product formation is associated with specific metabolic states that can be achieved only by carefully controlling the cells' environment. A striking example of such steady state multiplicity was observed in hybridoma cell cultures [14, 15] for producing monoclonal antibodies. Analysis of models of microbial cell cultures indicate coexistence of multiple stable steady states in a certain range of operation [16, 17]. This work focuses on the control of a continuous stirred tank containing bacterial cells such as *Klebsiella oxytoca* growing on a mixture of two substitutable nutrients (such as glucose and arabinose). Specifically, switching between multiple steady states to drive the reactor to the preferred steady state is considered.

3.1 Description of the Modeling Scheme

This system was originally studied for a batch reactor by Kompala et al.[7]. They applied the cybernetic modeling framework for modeling the diauxic behavior of the system. The model utilized data obtained from growth of the bacterium on single substrates and did not require *a priori* specification of the order in which the substrates are consumed. This model does not account for maintenance phenomenon occurring at low dilution rates; it simulates continuous bioreactors at dilution rates close to the maximum growth rates. The five states of the system correspond to substrate concentrations, biomass concentration and concentration of the two key enzymes within the cells. The model consists of five ODEs:

$$\frac{ds_1}{dt} = D[s_{1f} - s_1] - Y_1[r_1 v_1]c \tag{9}$$

$$\frac{ds_2}{dt} = D[s_{2f} - s_2] - Y_2[r_2 v_2]c \tag{10}$$

$$\frac{de_1}{dt} = r_{e1}u_1 + r_{e1}^* - \beta_1 e_1 - r_g e_1 \tag{11}$$

$$\frac{de_2}{dt} = r_{e2}u_2 + r_{e2}^* - \beta_2 e_2 - r_g e_2 \tag{12}$$

$$\frac{dc}{dt} = r_g c - Dc \tag{13}$$

where the rates are given by

$$\begin{aligned} r_i &= r_i^{max}\frac{s_i}{K_i + s_i}\left(\frac{e_i}{e_i^{max}}\right) \\ r_{ei} &= \alpha_i \frac{s_i}{K_{ei} + s_i} \\ r_g &= r_1 v_1 + r_2 v_2 \end{aligned}$$

As seen in equations (9) and (10), the Monod type kinetics are modified by cybernetic regulation variables of the second type v_i, that modify enzyme activity. Similarly, the cybernetic regulation variables u_i in equations (11) and (12) modify the rates of enzyme synthesis. Mathematically, the cybernetic regulation functions are related to the reaction rates as follows:

$$u_i = \frac{r_i}{r_1 + r_2} \qquad v_i = \frac{r_i}{\max(r_1, r_2)}$$

Numerical bifurcation analysis of the above-mentioned cybernetic model of bacterial growth on substitutable substrates revealed the existence of two stable steady states in a certain range of operating parameters [16], which arise due to cells' ability to switch their physiological states under nutritional pressures. In fact, five steady states are predicted by the bifurcation analysis, two of which are stable. Figure 2 shows the steady state bifurcation diagram for a bacterium *Klebsiella Oxytoca* growing on mixed feed of glucose and arabinose, in a continuous stirred reactor at dilution rate of $0.8 hour^{-1}$. The steady state which results in high biomass yield is the desired state. Values of the state variables for the two different steady states are shown in Table 2. One can observe that the "working steady state" is close to turning point bifurcation. Thus, relatively small changes in dilution rate and/or substrate feed concentrations could cause the reactor to drift to the other steady state.

3.2 Problem Statement

Due to the proximity of the working region to the turning point bifurcation, step disturbances in s_{2f} can drive the reactor to the undesirable steady state. The problem of switching of steady states also assumes significance during start up of the reactor. Typically steady state switching is a difficult problem due to hard nonlinearity associated with the switch. The control objective is, therefore, to drive the reactor from the low biomass steady state to the desirable high biomass yield state. It may be viewed as a step change in the setpoint at time $t = 0$ from the low biomass to the high biomass yield steady state. The performance of the controller is evaluated under step disturbances of various magnitude in parameter s_{2f}.

4 Implementation and Discussions

4.1 Suboptimal control law: Nonlinear MPC

The successive linearization based NMPC algorithm [10] was used as the initial suboptimal control law. This method linearizes the nonlinear model at each current state and input values to compute a linear prediction equation. The control is computed by solving a QP, with the hessian and the gradient computed at each sample time based on the new linear approximation. Detailed description is excluded for brevity, steps in implementation are given below.

- At each sample time, the model equations (9) – (13) are linearized using the state estimate of $x(k)$ denoted as $x(k|k)$.
- Model predictions under constant input conditions are obtained within the prediction horizon by integrating the differential equations (9) – (13).
- Using the model predictions, a linear approximation is obtained for considering the effects of further input adjustments. Using these model predictions and the linear approximation, the optimization problem is formulated as a QP (quadratic program). The control horizon and prediction horizons were chosen to be $m = 10$ and $p = 50$ respectively. The one-stage-cost was chosen as follows, with $Q = 100$ and $R = 250$:

$$\phi(x(k), u(k)) = \{Q[x_5(k+1) - c_{SP}]\}^2 + \{R\Delta u(k)\}^2 \tag{14}$$

  where $x_5(k+1)$ denotes biomass concentration and c_{SP} is the set point. $x(k+1)$ is related to $x(k)$ and $u(k)$ through integration of the model equations for one sample interval.
- The future control moves $u(k+i|k),\ i = 1 \ldots m$ are discarded and only $u(k|k)$ is implemented at the current time.
- At the next time step, an observer such as extended Kalman filter may be used to obtain a state estimate based on fresh measurements. Using this estimate, the procedure is repeated all over again in a receding horizon manner. For a deterministic case, state estimation is not required because $x(k|k) = x(k)$.

The closed loop simulation with the NMPC algorithm under a parameter conditions $s_{2f} = 0.146$ and $c_{SP} = 0.055$ is shown in Figures 3, 5 as thick line. At time $t = 0$, the system was at the low biomass steady state, when a step change in the set point to the high biomass steady state was applied. The constraints on the dilution rate were chosen to be $u_{\min} = 0.6$, $u_{\max} = 1.0$ and $\Delta u_{\max} = 0.05$, keeping in mind that the model is not valid for low dilution rates and to avoid washout condition that occurs at high dilution.

4.2 Obtaining optimal cost-to-go function approximator

4.2.1 Simulations using suboptimal controller

The suboptimal NMPC controller described above, was used to obtain closed loop simulation data for the proposed strategy. It was implemented for four values of $s_{2f} = \begin{bmatrix} 0.14 & 0.145 & 0.15 & 0.155 \end{bmatrix}$, to cover the possible range of variations. For each of the parameter values, the reactor was started at three different $x(0)$ values around the low biomass yield steady state. We obtained 100 data points for each run. Thus a total of 1200 data points were obtained. The infinite horizon cost-to-go values were computed for all the 1200 points. Note that the calculated cost-to-go value is approximate infinite horizon cost, as described in section 2.4.

4.2.2 Cost approximation

States were augmented with the parameter s_{2f} (see Table 1). A functional approximation relating cost-to-go with augmented state was obtained by using neural network — a multi-layer perceptron with five hidden nodes, six input and one output node. The neural network showed a good fit with mean square error of 10^{-3} after training for 1000 epochs. This is the zeroth iteration, denoted as $\tilde{J}^0(x)$.

4.2.3 Improvement through Bellman iterations

Improvement to the cost-to-go function is obtained through iterations of the Bellman equation (8). This method, known as cost iteration (or value iteration), is described in section 2.4. The solution of the one-stage-ahead cost plus cost-to-go problem, results in improvements in the cost values. The improved costs were again fitted to a neural network, as described above, to obtain subsequent iterations $\tilde{J}^1(x)$, $\tilde{J}^2(x)$, and so on $\ldots$, until convergence. Cost was said to be "converged" if the sum of absolute error was less than 5% of the maximum cost. The cost converged in 4 iterations for our system.

4.3 Online implementation

The converged cost-to-go function from above was used online in solving the one stage ahead problem. The control move was calculated as in 15 and implemented online in a receding horizon manner.

$$u(k) = \arg\min_{u(k)} \left\{ \phi\left(x(k), u(k)\right) + \tilde{J}^4\left(f_h(x(k), u(k))\right) \right\} \tag{15}$$

The results are shown as broken line in Figure 3 and a numerical comparison is shown in Table 3. The method was tested for various s_{2f} values. Representative results for a single s_{2f} value of 0.146 are shown. First two rows in the table represent the online performance of the two approaches, viz. NMPC and the proposed S-A-E scheme. In the table, the last two columns show the comparison between the two schemes; the first four columns represent the control algorithm, the number of data points used in obtaining cost-to-go function, the number of cost-iterations and the number of hidden nodes in the neural network approximation of converged cost function.

Clearly, the new scheme performs worse than the original NMPC scheme. An overshoot is observed and the total cost is also increased. However, there is a dramatic reduction in computational time — from almost half an hour to under 2 minutes, for 100 time steps (50 hours). In the next section, we evaluate the possible reasons for the worse behavior and discuss possible solutions.

4.4 Improvement in the Strategy

The policy improvement theorem, described earlier, indicates that the use of converged solution from Bellman iterations is expected to improve the performance over the suboptimal controller. At worst, the performance of the proposed scheme should be at par with the original suboptimal scheme. The logical reasoning behind this is that Bellman iterations should choose the original policy over all other policies that lead to a less optimal result (for mathematical proof, please see [12] or references therein).

The possible causes of error could either be presence of local minima, poor fitting of the cost approximations (by the neural network), or extrapolation to previously unvisited regions in state space. An investigation of the state space plot in Figure 4 suggests that extrapolation to previously unvisited regions of the state space could have resulted in deterioration of the controller performance. The system spans 6-D space (5 states and 1 parameter s_{2f}). All the state space plots are projections of the 6-D space on a 2-D plot, with $[s_2]$ as the abscissa and biomass $[c]$ as the ordinate. Possible remedies, which are discussed in this section, could be to restrict the control calculations to the visited part of state space or include additional data to increase the coverage as necessary.

4.4.1 Gridding and restricting the working region

In this method, the optimizer was *restricted* to search only in the visited region of the state space during both offline Bellman iterations as well as online implementation. The state space was grid into 10 grid points per state (10^5 cells) and each cell was identified as "visited" or "unvisited". A cell was said to be "visited" if it contained at least one data point. The neural network was used for the visited cells, whereas unvisited cells were associated with a very high cost, thus creating an artificial boundary to restrict the search within visited cells.

In this case, convergence of the Bellman equation occurred in 2 iterations. The neural network approximation required 4 hidden nodes. The results are shown in Figure 5 and numeric values in Table 3. The method is a clear improvement over the NMPC control law, both in performance of the controller as well as significant reduction in computation time. Corresponding state space plot is shown in Figure 6. Dots represent data points used for obtaining the cost-to-go function approximation. Diamonds represent the points visited during online implementation of the S-A-E control law. Comparing this with the previous state space plot (Figure 4), it is evident that by artificially constraining the control moves to the visited region, one obtains improved controller performance. Also note that the online optimization does not select the best among the suboptimal trajectories. Instead, it gives a trajectory which results from interpolating in the region defined by these suboptimal trajectories.

4.4.2 Increasing data coverage through additional simulations

In the previous method, we sought to restrict our controller to the visited regions of the state space. An alternative solution is to increase the coverage of the training data through additional simulations of the initial NMPC control law. Additional data is obtained from some more simulations of the NMPC law and cost iteration is performed again.

During online implementation of “unmodified” S-A-E algorithm, the controller drove the system to the unvisited region of the state space (dots in Figure 4 represent the visited region, rest is unvisited). Three such points — shown as solid discs in Figure 4 — were selected from the unvisited region. These represent $x(0)$ values for the additional simulations. We applied the original NMPC control, as before, for all the four values of s_{2f}. These 12 additional simulations resulted in 880 more data points. Cost iteration was carried out again for all the 2088 (including 1200 from the previous case) data points.

In this case, the Bellman equation converged in 4 iterations. The converged cost approximation was used in online control. Numerical and visual comparison indicate that this modification of S-A-E algorithm also results in highly improved control performance. In Figure 7, asterisks represent the 888 data point added to the original 1200 data (shown as dots). The resulting cost-to-go function approximation in this case is valid over a larger region. With these additional data points, the controller successfully avoids overshoot by identifying it as a suboptimal performance.

4.4.3 Generalized Policy Update

This technique is used to increase the coverage of the state space as in section 4.4.2. Unlike the previous scheme that requires addition of new data points and performing the entire set of iterations all over again, new points are added through policy update performed within the cost iteration loop. The need for this generalized policy update arises because in real situations, we may not want to wait for deteriorated behavior of the controller to direct additional simulations of the initial NMPC law. Moreover, the above method involves redoing the entire cost iteration procedure all over again.

The central theme of generalized policy update is to add more data points within the cost iteration loop itself through closed loop simulation of the current suboptimal controller (J^i). In this sense, it can be considered a hybrid between cost iteration approach and policy iteration[3] approach of NDP. During the cost

[3]In cost iteration, J^{i+1} represents improved cost-to-go value. Alternatively, policy iteration seeks to improve the policy by implementing it and evaluating the cost-to-go with this polity, instead of just updating the cost-to-go value [5].

iteration procedure, solving the corresponding equation (8) for each of the 1200 states ($x(k)$) gives us an improved cost-to-go (J^{i+1}) and input argument $u(k)$. If this input move was to be implemented, the system would reach a particular state $x^i(k+1|k)$ (which means the state reached by implementing control move calculated using cost-to-go function approximation $\tilde{J}^i$ at time k). If this state lies in the unvisited region, following procedure is implemented

- Current state is initialized as $x_0^i = x^i(k+1|k)$
- Control move is computed as $u_0^i = \arg\min[\phi + \tilde{J}^i]$. Implementing this move results in $x_1^i = F_h(x_0^i, u_0^i)$
- If x_1^i lies in the visited region, cost-to-go at this state is given by the neural network $\tilde{J}^i(x_1^i)$. Otherwise, control moves u_1^i, u_2^i, $\ldots u_{Ni}^i$ and corresponding states x_2^i, x_3^i, $\ldots x_{Ni+1}^i$ are computed according to the steps described above until x_{Ni+1}^i lies in visited region or reaches the set point.
- Cost-to-go values J_j^{i+1} for state x_j^i are given by $J_j^{i+1} = \sum_{l=j+1}^{Ni} \{\phi(x(l), u(l))\} + \tilde{J}^i(x_{Ni+1}^i)$

The cost-to-go value is computed and this state is added to the original 1200 states. This is done for each state $x^i(k+1|k)$ that lies in the unvisted region. In other words, policy update and cost-to-go evaluation is carried out within the cost iteration loop, hence the name. Then, a new functional approximator $\tilde{J}^{i+1}$ is fitted and (equation 8) is solved again for the expanded data set. This is done iteratively until convergence. Iteration is said to be converged if sum of absolute error was less than 5% and number of data points added during the iteration was less than 6 (0.5% of 1200).

In this case, the Bellman equation and policy update converged in 7 iterations. Numerical and visual comparison indicate that this modification of S-A-E algorithm also results in highly improved control performance. We believe that this method is more rigorous and general than the previous two methods. The optimal path may lie outside the region visited by the initial NMPC control law, which would not be considered if we were to restrict the online controller by gridding. Likewise, we do not need deteriorated behavior of the controller to direct additional simulations of initial NMPC controller to increase the data coverage, which may or may not work in a generalized scenario.

5 Concluding Remarks

Application of general ideas from NDP and RL literature to improving performance of MPC for a highly nonlinear problem of steady state switching in a microbial reactor was considered. The results indicate that the S-A-E scheme developed provides a promising framework for nonlinear optimal control in a computationally amenable way. The proposed controller does not select the best amongst original suboptimal trajectories; instead it seems to interpolate in the region defined by these trajectories, directed by cost-to-go function approximation, to yield substantially improved performance. An important outcome of this work is that while the method shows great promise, one needs to be careful in using the cost-to-go function approximation. In this study, extrapolation to previously unvisited states resulted in poor performance of the "vanilla" S-A-E controller. Three different modifications were suggested. We are, however, biased towards the *generalized policy update* as a more systematic method of increasing the coverage and searching for more optimal policy even in the regions unvisited by the original suboptimal controller.

This also points to an importtant function initial suboptimal strategy plays. We think that it is not necessary for initial suboptimal strategy to be close to optimal — Bellman iterations will take care of this issue. However, it is imperative that the suboptimal strategy covers all relevant portions of the state space. Spanning relevant parts of state space becomes increasingly difficult with an increase in state dimension. Hence Principal Component Analysis (PCA) or feature extraction may be used to reduce the state dimension or better identify relevant regions of the state space.

6 Acknowledgments

We thank Dr. Ramkrishna and A. A. Namjoshi for their help with cybernetic modeling and National Science Foundation (CTS-0096326) for financial support.

7 References

References

[1] M. Morari and J. H. Lee. Model predictive control: past, present and future. *Computers and Chemical Engineering*, 23:667–682, 1999.

[2] D. Q. Mayne, J. B. Rawlings, C. V. Rao, and P. O. M. Scokaert. Constrained model predictive control: Stability and optimality. *Automatica*, 36:789–814, 2000.

[3] J. H. Lee and B. Cooley. Recent advances in model predictive control. In *Chemical Process Control - V*, pages 201–216b, 1997.

[4] R. E. Bellman. *Dynamic Programming.* Princeton University Press, New Jersey, 1957.

[5] D. P. Bertsekas and J. N. Tsitsiklis. *Neuro-Dynamic Programming.* Athena Scientific, Belmont, MA, 1996.

[6] J. M. Lee and J. H. Lee. Neuro-dynamic programming method for mpc. In *DYCOPS VI*, pages 157–162, 2001.

[7] D. S. Kompala, D. Ramkrishna, N. B. Jansen, and G. T. Tsao. Investigation of bacterial growth on mixed substrates: Experimental evaluation of cybernetic models. *Biotechnology and Bioengineering*, 28:1044–1055, 1986.

[8] J. Varner and D. Ramkrishna. Metabolic engineering from a cybernetic perspective - i: Theoretical preliminaries. *Biotechnology Progress*, 15:407–425, 1999.

[9] N. S. Kaisare, J. H. Lee, A. A. Namjoshi, and D. Ramkrishna. Control of a mammalian cell bioreactor using cybernetic model. In *AIChE Annual Meeting 2000, Los Angeles, CA*, 2000.

[10] J. H. Lee and N. L. Ricker. Extended kalman filter based nonlinear model predictive control. *Ind. Eng. Chem. Res.*, 33:1530–1541, 1994.

[11] R. Howard. *Dynamic programming and markov processes.* MIT Press, Cambridge Massachussets, 1960.

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

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

[14] B. D. Follstad, R. R. Balcarel, G. Stephanopoulos, and D. Wang. Metabolic flux analysis of hybridoma continuous culture steady state multiplicity. *Biotechnology and Bioengineering*, 63:675–683, 1999.

[15] W.-S. Hu, W. Zhou, and L. F. Europa. Controlling mammalian cell metabolism in bioreactors. *Journal of Microbiology and Biotechnology*, 8:8–13, 1998.

[16] A. A. Namjoshi and D. Ramkrishna. Multiplicity and stability of steady states in continuous bioreactors: Dissection of cybernetic models. *Chemical Engineering Science*, 56(19):5593–5607, October 2001.

[17] A. Narang. The steady states of microbial growth on mixtures of substitutable substrates in a chemostat. *Jounal of Theoretical Biology*, 190:241–261, 1998.

State Variables	s_1	glucose (gm/L)
	s_2	arabinose (gm/L)
	e_1	key enzyme(s)-1 (gm/gm dry wt.)
	e_2	key enzyme(s)-2 (gm/gm dry wt.)
	c	biomass (gm/L)
Manipulated Variable	D	dilution rate (hr^{-1})
Controlled Variable	c	biomass
Parameter	s_{2f}	s_2 feed rate (gm/L)

Table 1: Key variables and parameters of the system

State	s_1	s_2	e_1	e_2	c
High biomass	0.035	0.081	0.0004	0.0006	0.0565
Low biomass	0.0447	0.1425	0.0007	0.0003	0.02

Table 2: Steady state values for input conditions $D = 0.8, s_{1f} = 0.078, s_{2f} = 0.146$.

Control Algorithm	Number of data points	Cost Iterations	Number of hidden nodes	Total cost (at $x(0)$)	CPU Time (seconds)‡
NMPC	- N.A. -	- N.A. -	- N.A. -	22.54	1080.3
New Scheme	1200	4	5	24.18	98.7
w/ Restricting	1200	2	4	9.06	127.7
w/ Add Sim†	2088	4	5	9.37	79.5
w/ Policy update	1395	7	9	10.32	74.12

Table 3: Details of nonlinear MPC algorithm v/s new scheme and its modifications. †Additional simulations for increasing data coverage; ‡Intel Pentium III, 800 MHz processor, 512 MB RAM, running Matlab 6 Release 12 on Windows 2000.

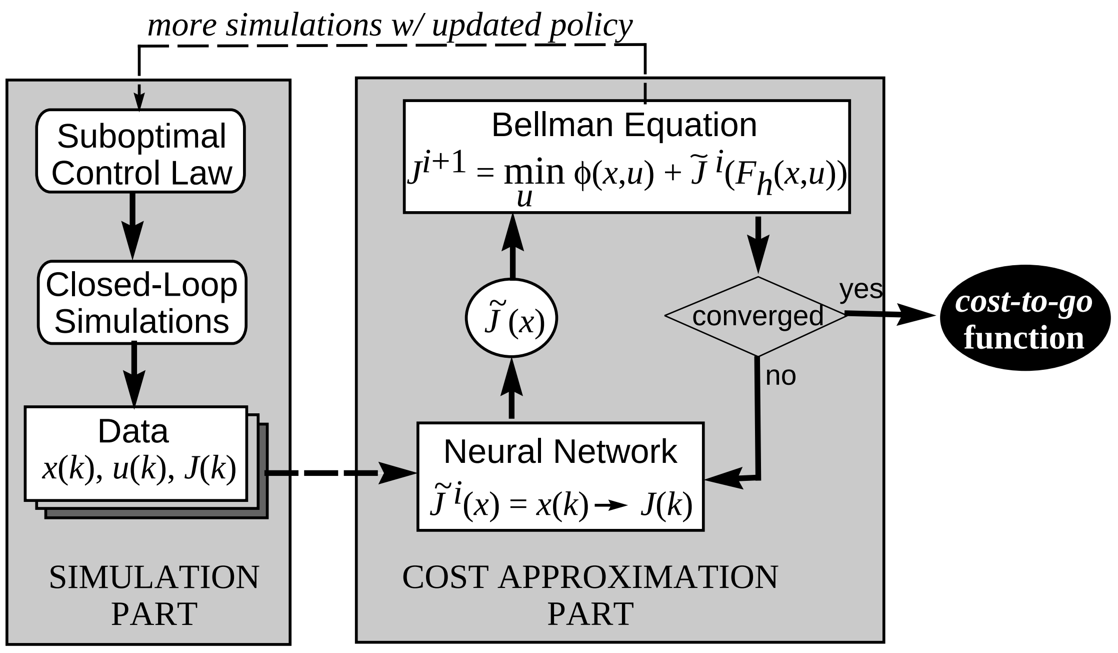

Figure 1: Architecture for offline computation of cost-to-go approximation

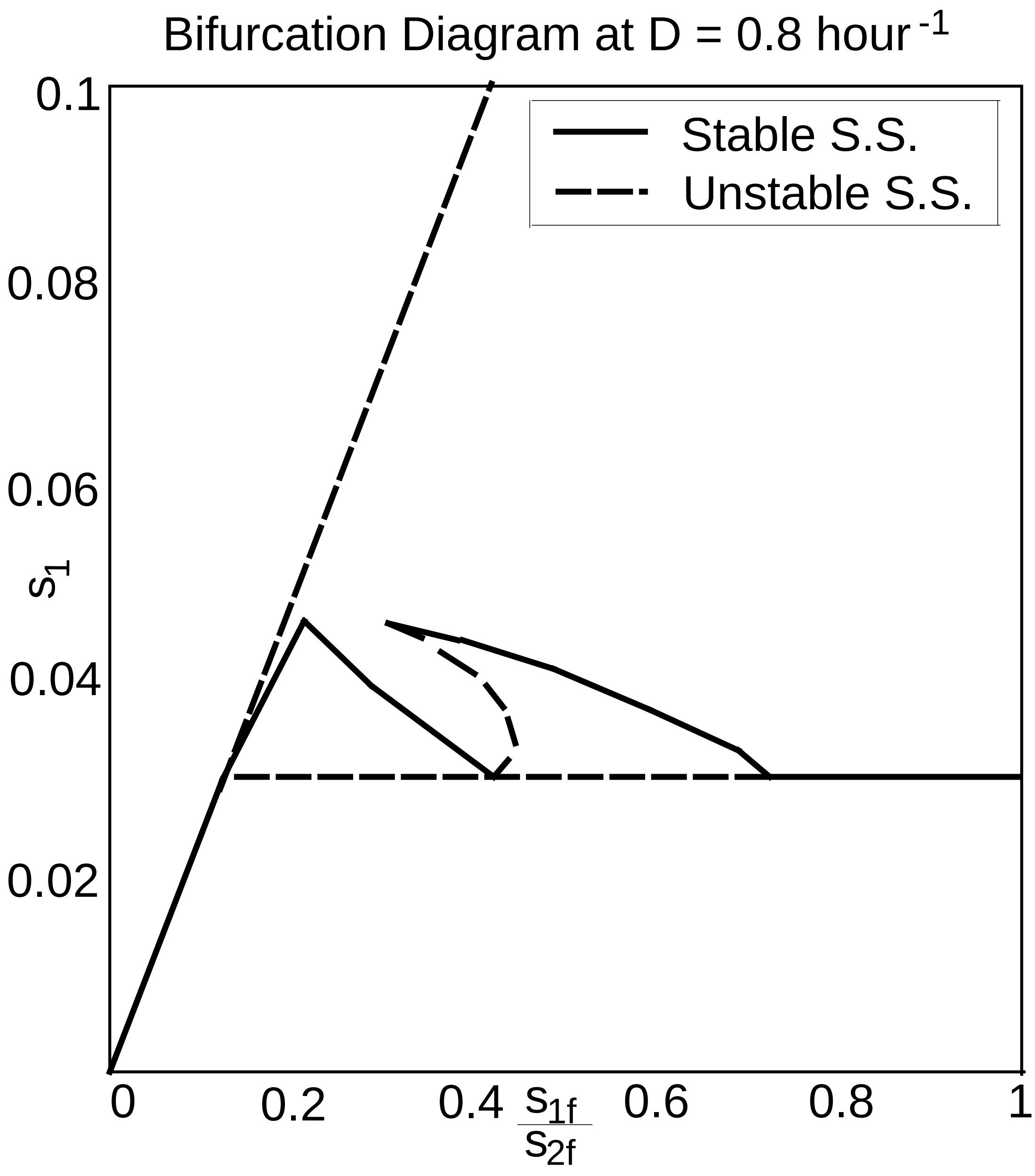

Figure 2: Steady state bifurcation diagram for Klebsiella Oxytoca growing on glucose and arabinose. Note the proximity of the steady states to turning point bifurcation. Adapted from [16]

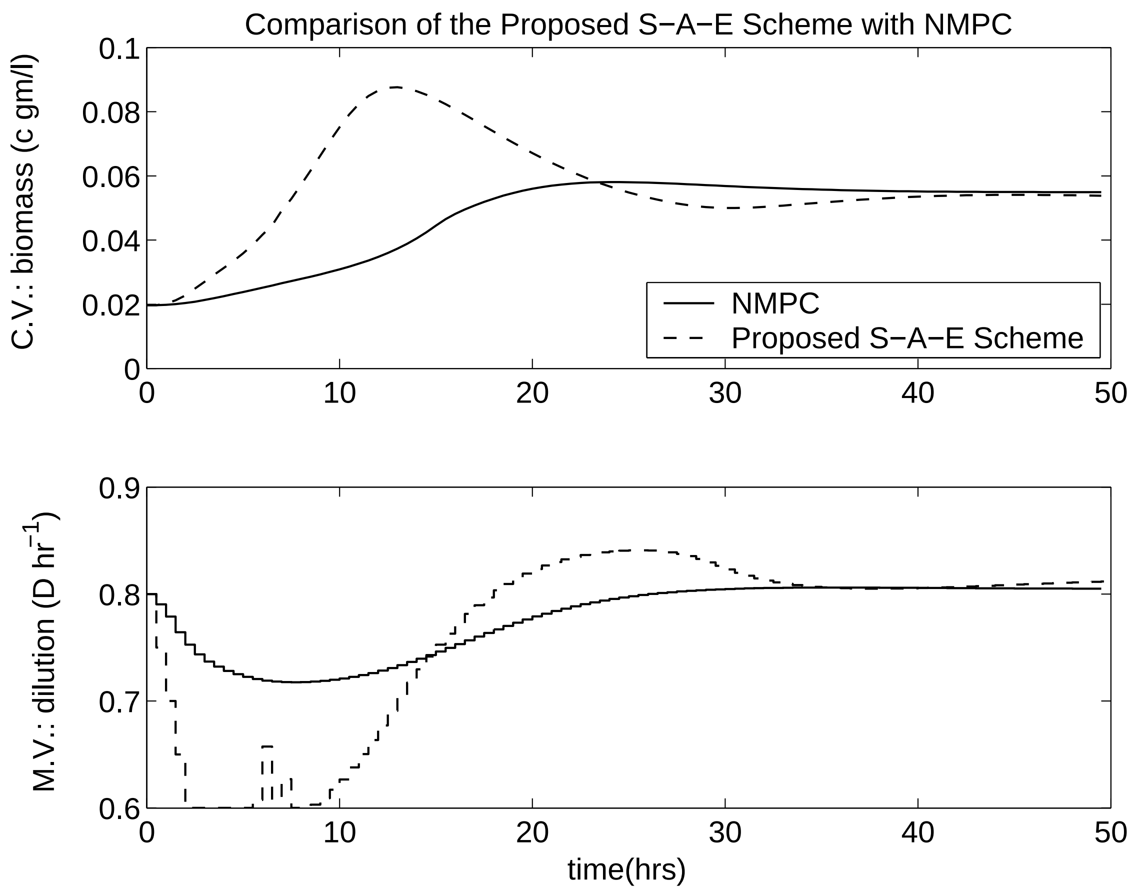

Figure 3: Comparison of the online performance of NMPC control law and proposed S-A-E approach. The S-A-E controller gives faster response initially, but causes significant overshoot, as compared to more sluggish NMPC controller. Numeric comparison between the two approaches is shown in table 3.

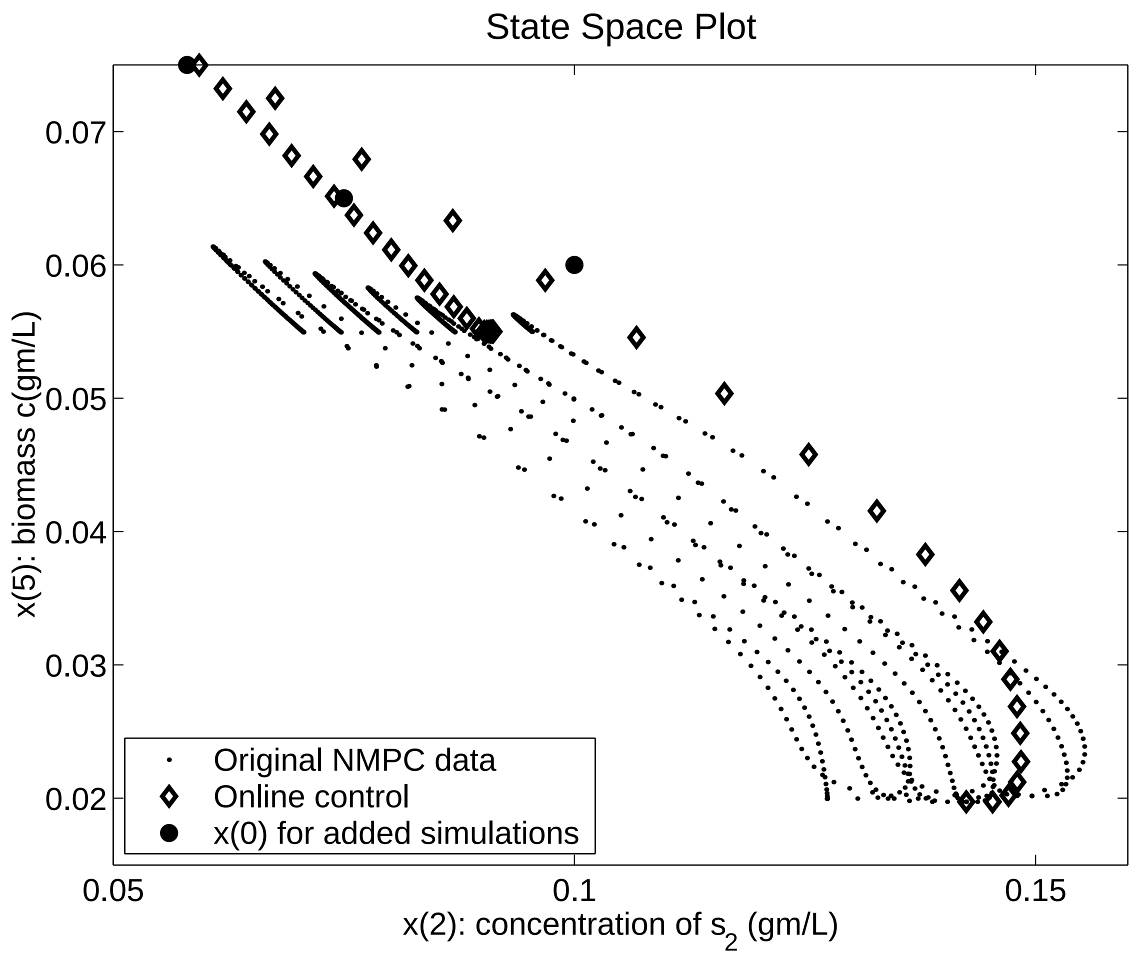

Figure 4: State space plot of states visited during online implementation (diamond). The data from NMPC control law that were used for training the cost approximation is shown as dots. Extrapolation to the unvisited states during online implementation is likely to be the cause of overshoot.

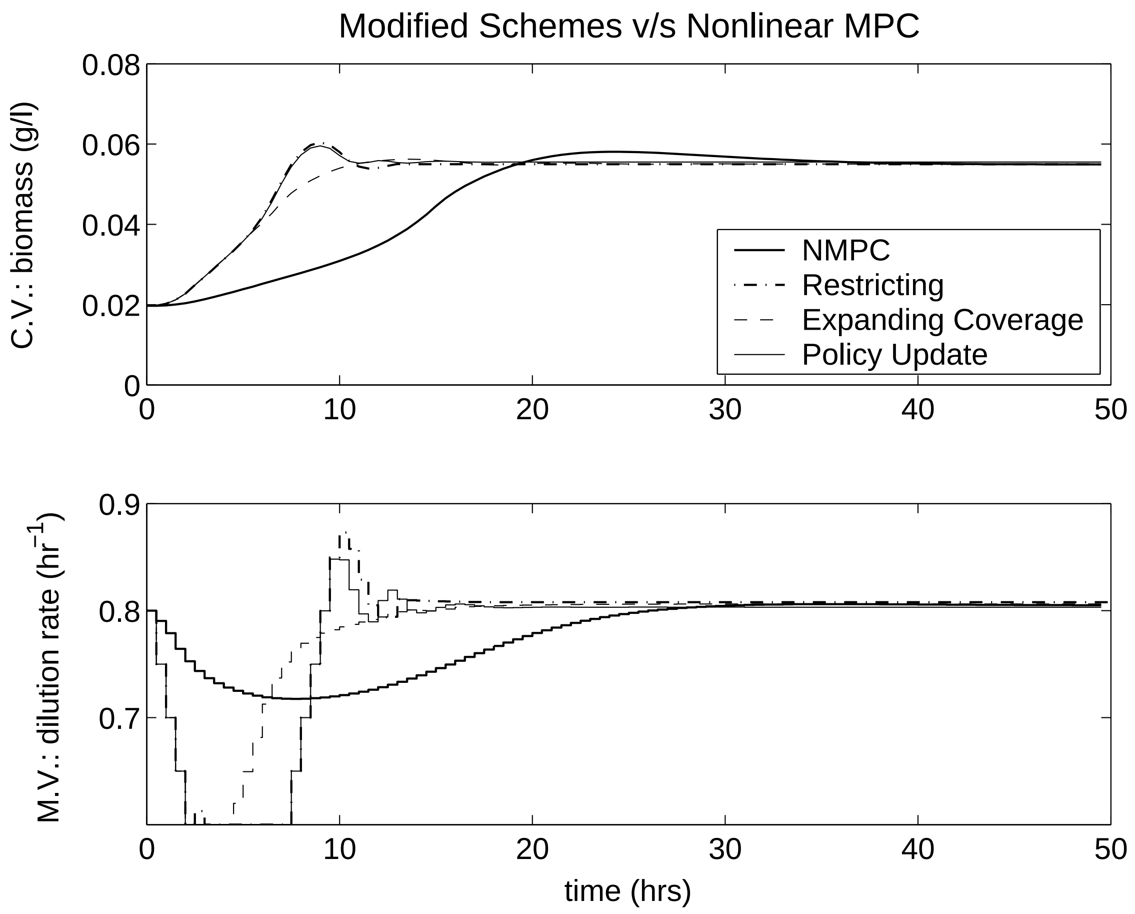

Figure 5: Performance of the modified schemes. NMPC control law is shown as thick solid line for comparison

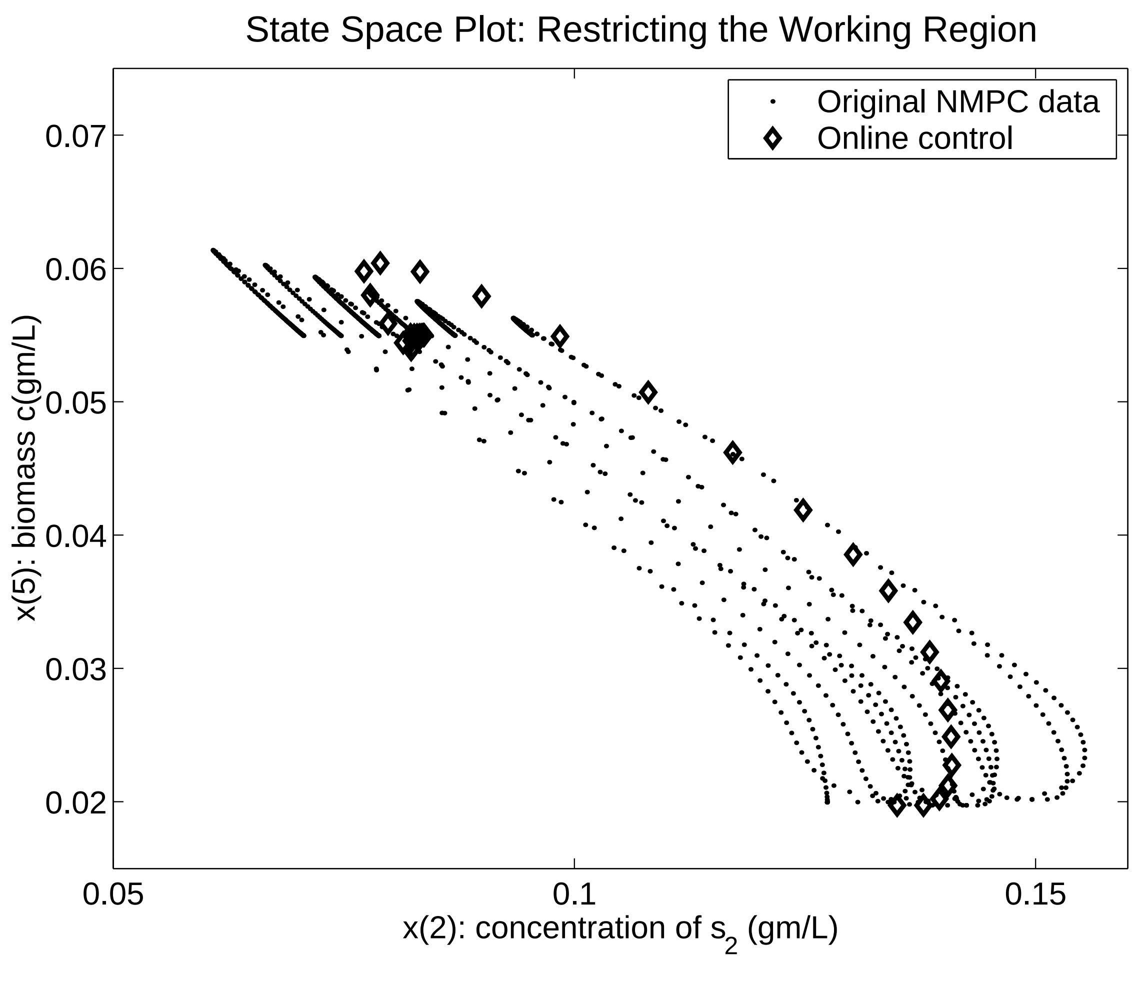

Figure 6: State space plot of states visited during online implementation (diamond). The data from original suboptimal NMPC is shown as dots. The system is “constrained” to the visited region of the state space

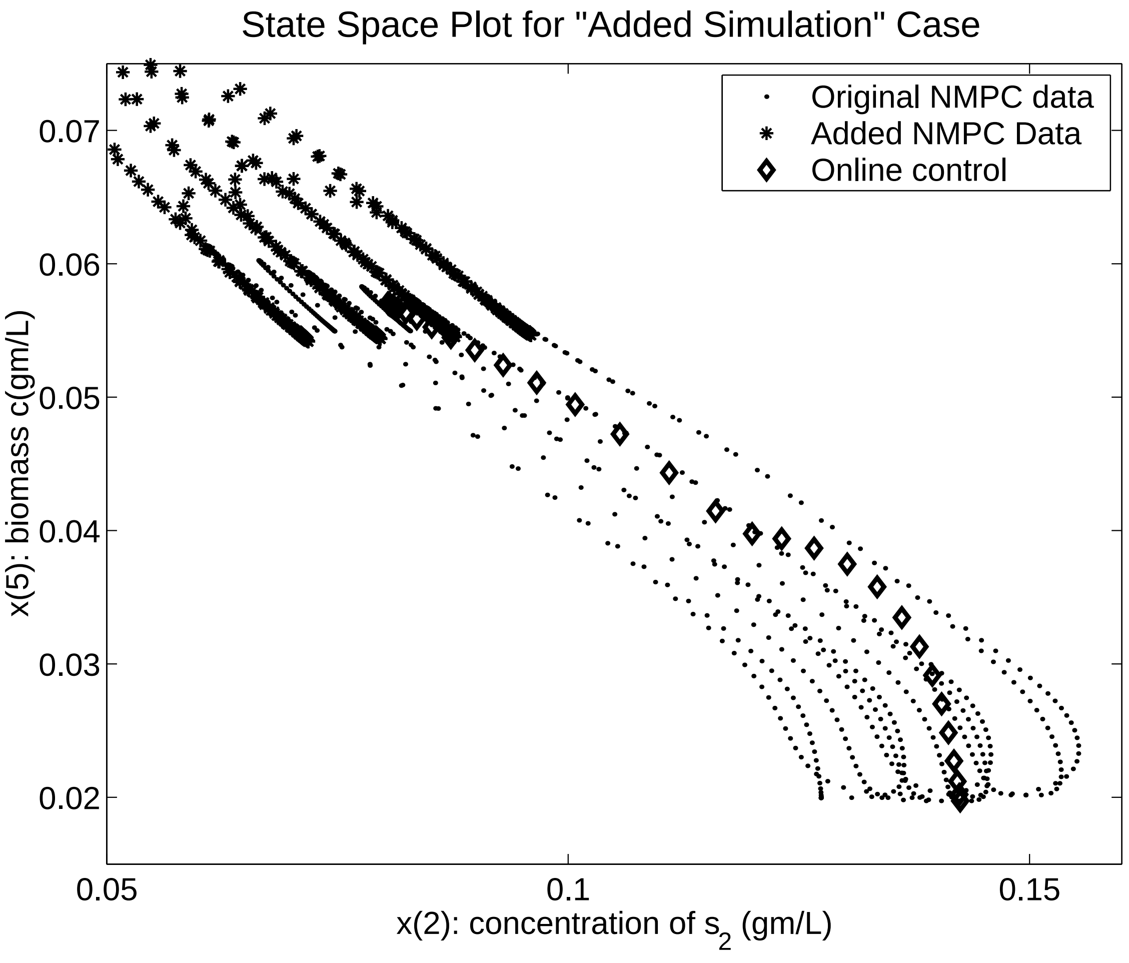

Figure 7: State space plot of states visited during online implementation (diamond). The data from original suboptimal NMPC is shown as dots. Asterisk (*) represent the extra points added through additional simulations of NMPC control law.

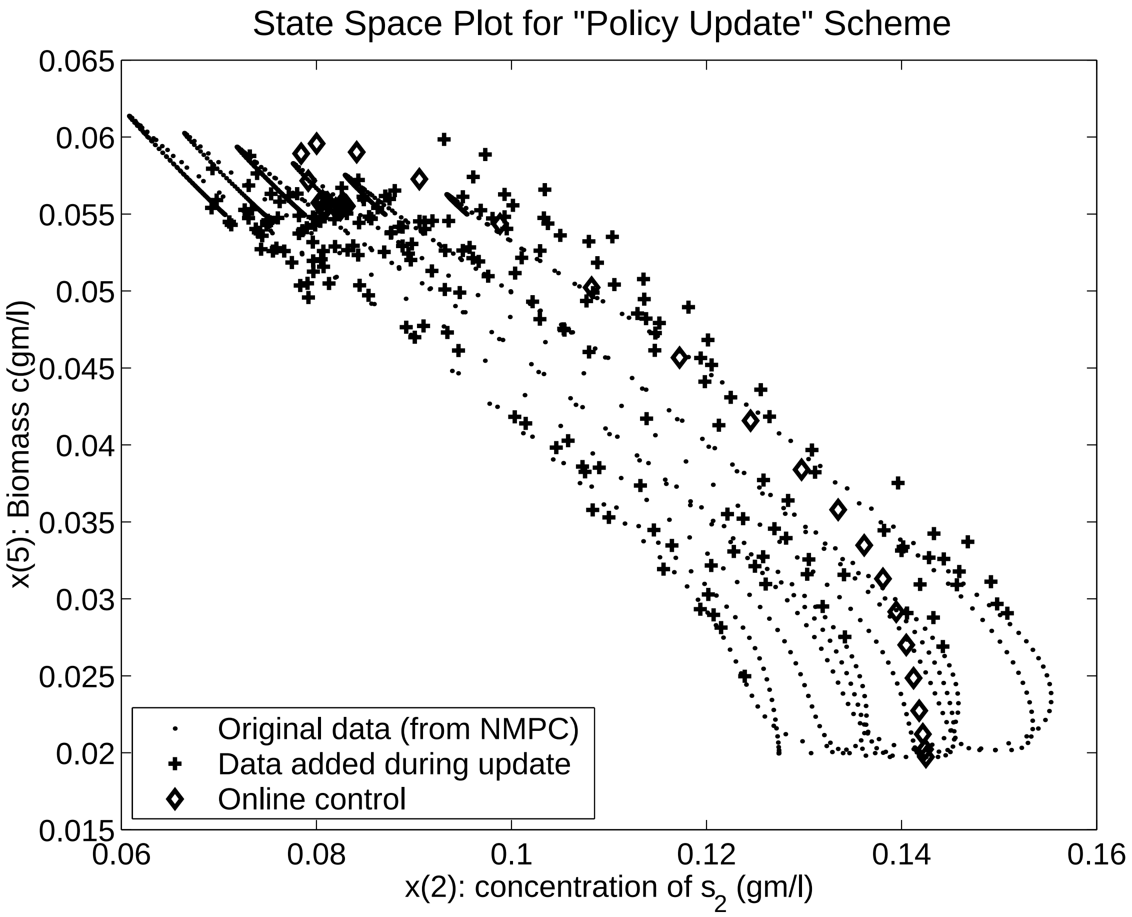

Figure 8: State space plot of states visited during online implementation (diamond). The data from original suboptimal NMPC is shown as dots. During the iteration, additional data points added are shown as crosses (+). In all, 195 data points were added in the 7 iterations performed until convergence.