A Dynamic Penalty Function Approach for Constraint-Handling in Reinforcement Learning

Haeun Yoo* Victor M. Zavala Jay H. Lee***

** Department of Biomolecular and Chemical Engineering, Korea Advanced Institute of Science and Technology, 291 Daehak-ro, Yuseong-gu, Daejeon, 34141, Republic of Korea (e-mail: haeungd, jayhlee@ kaist.ac.kr).*

*** Department of Chemical and Biological Engineering, University of Wisconsin-Madison, Madison, WI 53706, USA (e-mail: victor.zavala@wisc.edu)*

Abstract: Reinforcement learning (RL) is attracting attention as an effective way to solve sequential optimization problems that involve high dimensional state/action space and stochastic uncertainties. Many such problems involve constraints expressed by inequality constraints. This study focuses on using RL to solve constrained optimal control problems. Most RL application studies have dealt with inequality constraints by adding soft penalty terms for violating the constraints to the reward function. However, while training neural networks to learn the value (or Q) function, one can run into computational issues caused by the sharp change in the function value at the constraint boundary due to the large penalty imposed. This difficulty during training can lead to convergence problems and ultimately lead to poor closed-loop performance. To address this issue, this study proposes a dynamic penalty (DP) approach where the penalty factor is gradually and systematically increased during training as the iteration episodes proceed. We first examine the ability of a neural network to represent a value function when uniform, linear, or DP functions are added to prevent constraint violation. The agent trained by a Deep Q Network (DQN) algorithm with the DP function approach was compared with agents with other constant penalty functions in a simple vehicle control problem. Results show that the proposed approach can improve the neural network approximation accuracy and provide faster convergence when close to a solution.

Keywords: Reinforcement Learning, Penalty approach, Dynamic Penalty, Constraints

1. INTRODUCTION

Sequential decision-making problems such as scheduling, planning, and control involve constraints that should be respected. When the state space system model is available, the optimization problem can be formulated as follows:

$$\min_{[\boldsymbol{u_k}]_{k=0}^{T}} \sum_{k=0}^{T} l(\boldsymbol{x}(k), \boldsymbol{u}(k)) \tag{1a}$$

$$\text{s.t. } \boldsymbol{x}(k+1) = f(\boldsymbol{x}(k), \boldsymbol{u}(k)), \tag{1b}$$

$$g_i(\boldsymbol{x}(k), \boldsymbol{u}(k)) \leq 0,\ i = 1, ...v \tag{1c}$$

$$\boldsymbol{x}(0) = \boldsymbol{x}_0, \tag{1d}$$

$$\underline{\boldsymbol{x}} \leq \boldsymbol{x}(k) \leq \overline{\boldsymbol{x}}, \tag{1e}$$

$$\underline{\boldsymbol{u}} \leq \boldsymbol{u}(k) \leq \overline{\boldsymbol{u}}, \tag{1f}$$

where k is the time index, $\boldsymbol{x} \in \mathbb{R}^n$ is the state vector, $\boldsymbol{u} \in \mathbb{R}^m$ is the control input, and $l(\boldsymbol{x}, \boldsymbol{u})$ is a cost function. By solving the above problem at each time instant for the given state (estimate) as the initial state $\boldsymbol{x}_0$ and implementing the resulting $\boldsymbol{u}_0$ as the control action, one can effectively implement an optimal feedback policy. This is a basic idea behind the popular model predictive control (MPC) method. However, the model inevitably has some mismatch with the real system due to model errors, disturbances, noise, etc. The basic (deterministic) MPC formalism does not allow one to consider various uncertainties in the optimization, but instead compensate for their effect passively through the feedback. This can lead to a performance loss including constraint violations. To address this problem, reinforcement learning (RL) has been suggested as an alternative to MPC in the process system engineering field (see Yoo et al. (2020); Petsagkourakis et al. (2020b); Shin et al. (2019)).

An RL agent finds the optimal policy by exploring different parts of the state space through simulations or by probing the real system to learn the approximate value of each state. Using the Bellman optimality principle, the value estimate can be improved iteratively, with the aim of eventually identifying the optimal value (Bellman (1966)). In RL, the control problem is formulated as a Markov decision process, which is composed of the state space ($\boldsymbol{S}$) and state ($\boldsymbol{x} \in \boldsymbol{S}$), action space ($\boldsymbol{A}$) and action ($\boldsymbol{u} \in \boldsymbol{A}$), the state transition model, and the reward function ($r : S \times A \to R$) (Puterman (2014)). The state transition model (1b) is used in this study but this can be generalized to specify probabilistic transitions and/or the effect of stochastic noise terms.

The agent gets a certain reward (or penalty) for each state visited and the summation of the discounted rewards (with a discount factor of $\gamma \in [0,1]$) from current state to the terminal state ($\sum_{k=t}^{T} \gamma^{k-t} r(x(k))$) is the value of the state. The value function, $V(x)$, is the function that maps the state to the expectation of the return value, which implies the long-term *"value"* of the state, and Q-function $Q(x,a)$, is the expected return value of an state and action pair. In the MDP formulation, the action limits of (1f) can always be satisfied by considering only the feasible actions. Meanwhile, the state bounds of (1e) and state/input constraints (1c) cannot generally be satisfied in a strict sense as they may become infeasible. To address this problem, policy gradient RL methods for constrained MDP have been proposed based on a trust region policy optimization algorithm (Achiam et al. (2017); Petsagkourakis et al. (2020a) and a Lyapunov-based approach (Chow et al. (2019)). However, these methods are only applicable within the policy gradient algorithms. Hence, we instead focus on the penalty approach, where violations are penalized with terms added to the reward function along with the original objective function (1a). This approach is more general as it can be applied to not only policy-based methods, but also other value-based RL methods.

The form of penalty function can be a uniform constant, as shown in Yang et al. (2020); Zhang et al. (2020), a linear (1-norm) function of the magnitude of the violation as shown in Ma et al. (2019), or a logistic function as shown in Pan et al. (2020); Modares et al. (2016). The penalty is typically chosen to be large enough in order to prevent constraint violation. On the other hand, an infinite penalty (as used in the barrier method for solving constrained optimization) can lead to infeasibilities and thus will not be used here. When we use a high uniform penalty value or a steep linear penalty function, the training of neural network (NN), a popular choice for representing the value (or Q) function in RL, can give convergence problems as will be demonstrated later with a simple example. On the other hand, a small penalty term can lead to unnecessary constraint violations, i.e., violations even when the constraint can be satisfied. A penalty function that strikes a right balance is needed, but this is not easy to decide *a priori*.

In this study, we propose an approach that updates the penalty factor as the training proceeds, in order to achieve a stable training result with less approximation error and to get a so-called "sufficiently feasible" policy (Deb (2000)). We will refer to this approach as the DP approach. A similar approach has been studied to update parameters during iterations in evolutionary optimization strategies (Kramer (2010); Joines and Houck (1994)). Also, Lin and Zheng (2012) suggested to gradually increase the penalty value in RL-based control to address the numerical difficulty approximating a steep function, but they do not provide any detailed procedure or systematically analyze the effect of varying the penalty parameter.

The paper is structured as follows: Section 2 describes the DP function design with constraints aggregation using the Kreisselmeier-Steinhauser (KS) function and penalty factor update rules. In Section 3, we use a simple 1-dimensional function approximation example to examine the regression accuracy achieved with several penalty functions including the DP function. In Section 4, the proposed DP function is tested on a simple vehicle control example and the results are summarized. Section 5 concludes the paper.

2. DYNAMIC PENALTY (DP) FUNCTION

In this work we use a constraint aggregation method to design an unbiased penalty for constraint violations. A systematic rule for updating the penalty factor is described for an application to any RL algorithm.

2.1 Constraint Aggregation

Let us assume that we have υ inequality constraints including the inequality constraints on the state space boundary. The RL agent gets a penalty value for each constraint violation. A constraint aggregation is needed for unbiased constraint handling when multiple constraints exist. The KS function is a widely used constraint aggregation method for gradient-based optimization (Kreisselmeier and Steinhauser (1980)). Although RL is not a gradient-based optimization algorithm, we can effectively use this method with a high aggregation parameter, ρ, making the error from the aggregation as small as possible (Poon and Martins (2007)). For constraints $g_i(x_t, u_t) \leq 0, i = 1, ..., \nu$, the aggregate function $KS[\boldsymbol{g(x,u)}]$ is:

$$KS[\boldsymbol{g(x,u)}] = \\ g_{max}(\boldsymbol{x},\boldsymbol{u}) + \frac{1}{\rho}\ln[\sum_{i=1}^{\nu} e^{\rho(g_i(\boldsymbol{x},\boldsymbol{u}) - g_{max}(\boldsymbol{x},\boldsymbol{u}))}]. \quad (2)$$

where g_{max} is the maximum of all constraints evaluated at the given state-action pair.

2.2 DP Function

Using the KS function, the reward function with DP term can be defined as follows:

$$R(\boldsymbol{x},\boldsymbol{u}) = l(\boldsymbol{x},\boldsymbol{u}) + p(\boldsymbol{x},\boldsymbol{u}|\mu) \quad (3a)$$
$$p(x,u|\mu) = \mu \cdot KS[\boldsymbol{g(x,u)}]\mathbf{1}_{KS[\boldsymbol{g(x,u)}]>0} \quad (3b)$$

where μ is the penalty factor which defines the slope of the penalty function. The right-hand side of 3b is calculated only when the aggregate constraint is violated. The DP function approach starts with a low value of μ, which facilitates the initial training of the NN, and gradually increases it to a large value by multiplying the DP update parameter c so that the slope becomes large to prevent constraint violations (Fig. 1 illustrates this approach). The procedure can be described as:

(1) Set $\mu := \mu_{min}$.
(2) When the loss value of the NN is less than $(100 - \alpha)\%$ of the maximum loss value after the parameter update, set $\mu := c\mu$ where c is some constant larger than 1
(3) Repeat step (2) until $\mu \geq \mu_{max}$
(4) After reaching $\mu \geq \mu_{max}$ set $\mu := \mu_{max}$
(5) Continue the training until an optimal policy is found.

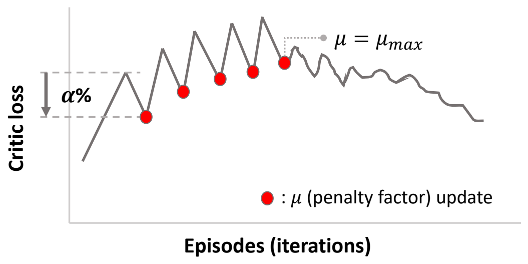


Fig. 1. Illustration of penalty update rule

3. VALUE FUNCTION REGRESSION TEST

3.1 1-Dimensional Example

A feed-forward NN is the most widely used function approximator for the value function or Q-function approximation. The agent uses the output value or the gradient of the NN to update the optimal policy. Therefore, accuracy of the approximation is important for a fast and stable agent training. We tested the regression accuracy with a priori fixed uniform penalty and linear penalty, which are the popular choices for the penalty function against the proposed dynamically varied penalty function. The test value function $V(x)$ is chosen as follows:

$$V(x) = 1 + \cos(0.5x) + 0.05(x-1)(x+2) + p(x). \quad (4)$$

We assumed that this problem has inequality constraints as $-5 \le x \le 5$ and the penalty functions are assigned as in table 1 where the $\mu_{min} = 0.1$, $\mu_{max} = 50$ and $c = 2$. This artificially chosen value function is not exactly in the same form as in the optimal control problem where the value should be calculated with respect to the cumulative rewards. Nevertheless, this analysis should be useful to understand the tendencies of the NN approximation during the agent training with the different penalty terms.

Table 1. Parameters for the penalty function

Penalty function	$\boldsymbol{p(x)}$
Uniform penalty	$50 \cdot \mathbf{1}_{x<-5,x>5}$
Linear penalty	$50((-5-x)\mathbf{1}_{x<-5} + (x-5)\mathbf{1}_{x>5})$
DP	$\mu((-5-x)\mathbf{1}_{x<-5} + (x-5)\mathbf{1}_{x>5})$

To examine the NN training performance within an RL environment, the training environment is set similarly as in RL. In every episode, 20 x points are randomly sampled from -10 to 10 and the environment calculates the $R(x)$ value ($R(x) = l(x) + p(x)$) with the assigned objective function (4) and the penalty function (table 1). Then, 20 samples are stored in a replay buffer and the NN is trained with a batch of 64 data randomly sampled from the replay buffer up to 500 episodes.

3.2 Results

Fig. 2 shows the output value of the NN during the training process. As shown in Fig. 2a, the NN approximation performance is poor at the boundary points, which causes a bias in the estimate inside the feasible region. The linear penalty case also gives a biased estimate inside the feasible region during the training and the maximum loss function is much higher than that of the DP case, as shown in Fig. 3. With the proposed scheme of dynamically varying the penalty during traning, the function is well approximated near the constraint boundary, which can reduce the bias and loss. Fig. 3 shows that the loss of NN with the DP function approach is increased during the penalty factor update period but is decreased drastically afterward. The final loss is the lowest in the case of DP, which implies the best approximation performance. From this simple test, we see the potential benefit of using a dynamically varied penalty in training the NN to learn the value function with a large penalty function.

4. CASE STUDY

4.1 Vehicle Control Example

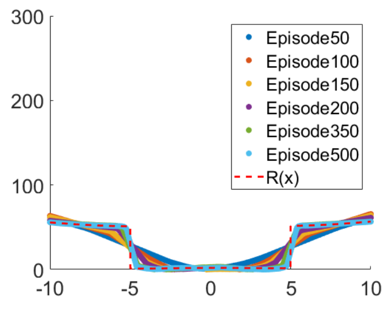
To further examine the effectiveness of the DP approach, a simple vehicle control problem described in (5) is solved with RL. We modified the problem from Grüne and Pannek (2017) to reduce the size of the feasible region and make the starting point uncertain as described in (5f). The objective is to minimize the running cost (5a). $x_1(k)$ is the position of a vehicle at time k, $x_2(k)$ is its velocity and $u(k)$ is its acceleration.

$$\min \; l(x,u) = x_1^2 + u^2 \quad (5a)$$
$$\text{s.t.} \; \begin{pmatrix} \dot{x}_1(k) \\ \dot{x}_2(k) \end{pmatrix} = \begin{pmatrix} x_2(k) \\ u(k) \end{pmatrix} \quad (5b)$$
$$-1 \le x_1(k) \le 1 \quad (5c)$$
$$-0.25 \le x_2(k) \le 1 \quad (5d)$$
$$-0.25 \le u(k) \le 0.25 \quad (5e)$$
$$x_0 \sim U((-1, 0.8), (-0.8, 1)) \quad (5f)$$

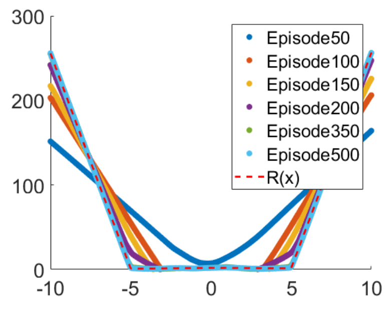
We applied the Deep-Q-Network (DQN) algorithm which uses a deep NN to approximate the Q-function and chooses the ϵ-greedy policy (Mnih et al. (2013)). For the training, we defined the discrete action space $u \in \{-0.25, -0.2375, -0.225, ..., 0.2375, 0.25\}$ and each episode was set for a time duration of 20min with 1min control interval. The deep NN consisted of three hidden layers with 64 nodes of the ReLU activation function. For constraint aggregation, (5c) and (5d) are decomposed as shown in (6) and aggregated to a $KS[\boldsymbol{g}]$ function of (2) with $\rho = 50$.

$$\begin{pmatrix} g_1 : -1 - x_1 \le 0 \\ g_2 : x_1 - x \le 0 \\ g_3 : -0.25 - x_2 \le 0 \\ g_4 : x_2 - 1 \le 0 \end{pmatrix} \quad (6)$$

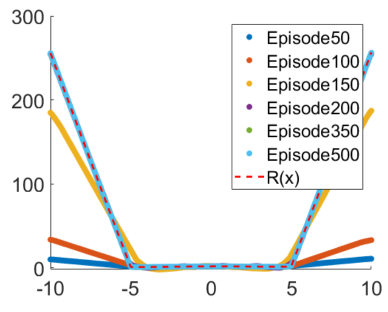
The DP was updated when the loss became lower than 30% of the maximum loss ($\alpha = 70$) and the penalty factor was doubled in each update ($c = 2$), starting from $\mu_{min} = 0.05$ up to $\mu_{max} = 20$. The value for the uniform penalty function and the penalty factor for the linear penalty were both set to 20 as shown in table 2. The training comprised 2000 episodes and the initial point was randomly selected inside the feasible region to start each episode. To test the agent's behavior during training, we tested the policy after every 100 episodes without exploration. In addition, for a rigorous comparison, we trained a set of 100 agents in the environment set with 100 different random seeds, which introduced randomness to exploration, NN initialization, buffer sampling, etc.



(a) Uniform penalty



(b) Linear penalty



(c) DP

Fig. 2. NN approximation results according to episodes progression

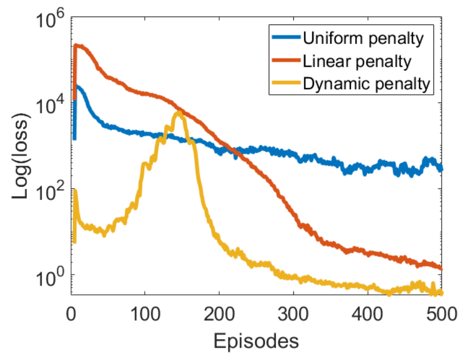


Fig. 3. NN loss according to episodes progression

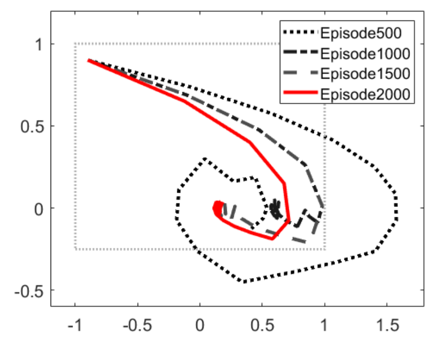


Fig. 4. Trajectories in the training process of the agent trained with a DP function. The dotted rectangle represents the feasible region.

4.2 Results

Table 3 shows that the agents trained with the uniform penalty and the linear penalty could find sufficiently feasible policies only in 42 and 48 cases out of the 100 random cases, respectively. Here, 'sufficient feasible policy' means that the policy does not violate all the constraints and achieves a sufficient low cost value. The agent trained with the uniform penalty incurred the highest average cost. Also notable is that the agent could find feasible policies in 82 cases, but almost a half of them were unable to achieve sufficiently low cost. According to table 4, the uniform penalty seems favorable for a rapid training, but the final performance is not satisfactory. These results are likely due to the interference of the bias in the estimate inside the feasible region. The agent trained with the linear penalty function performed poorly due to having inaccurate approximate values for the value function during training. This consequently resulted in the policy being updated in a wrong direction, or converging to sub-optimal policies.

The use of the DP approach was effective in finding a sufficient feasible policy rapidly and consistently. The agent trained with the DP approach could find sufficient feasible policies in 83 cases, and the average cost of that cases was lower than the results of the agent trained with other penalty functions. We also calculated the average degree of violation by averaging the positive KS function values from the converged infeasible policy. As shown in Table 3, when the agent was trained with the DP approach, the degree of violation of the infeasible policies was much lower compared to the other approaches. This means that even the infeasible policies found in the 7 cases, nearly satisfied the constraints. Table 4 also shows that sufficiently feasible policies were found within 1500 episodes in 39 cases, which is comparable to the training with the uniform penalty. Fig. 4 shows the policy improvement during training with the DP function.

Table 2. Penalty function types

Penalty function	$p(\boldsymbol{x})$
Uniform penalty	$20 \cdot \mathbf{1}_{KS(\boldsymbol{g})>0}$
Linear penalty	$20 \cdot KS(\boldsymbol{g}) \cdot \mathbf{1}_{KS(\boldsymbol{g})>0}$
DP	$\mu \cdot KS(\boldsymbol{g}) \cdot \mathbf{1}_{KS(\boldsymbol{g})>0}$

5. CONCLUSIONS

To solve constrained optimal control problems with RL, inequality constraints imposed on the state should be translated into a penalty term in the reward function. However, when the agent is trained with a priori set large uniform or linear penalty function, inaccurate NN approximations can result leading to convergence problems and poor eventual performance. To address this, we propose an approach that varies the penalty function during training in order that NNs be trained stably and rapidly. Our results show that the DP approach can reduce the approximation loss not only during the training but also at the end of the training. The DP approach was also found effective in the simple vehicle control problem tested. We trained the agent with different penalty forms, and the DP scheme showed the best performance in finding sufficient feasible policies with lowest average cost. In addition, the average degree of

Table 3. Summary of agent training results

Penalty function	**Uniform penalty**	**Linear penalty**	**DP**
Finding feasible policy/100	82	62	93
Finding sufficient feasible policy*/100	42	48	83
Average cost of the sufficient feasible policy	3.56	3.49	3.36
Average degree of violation	0.948	2.051	0.137

∗'sufficient feasible policy' means the policy does not violate all the constraints and achieves a sufficient low cost value

Table 4. Finding sufficient feasible policy until each episodes

Penalty function	**500**	**1000**	**1500**	**2000**
Uniform penalty	0	14	15	13
Linear penalty	2	8	16	22
DP	2	8	29	44

violation was lowest when the agent was trained with the DP function. The proposed approach can be applied to any RL algorithm and can help the agent that uses NN as a function approximator to be trained efficiently to represent a function that includes a steep penalty term for constraint violation. For future work, we will further analyze the effectiveness of the DP function approach for cases with model and exogenous uncertainties.

ACKNOWLEDGEMENTS

This research was supported by Korea Institute for Advancement of Technology (KIAT) grant funded by the Korea Government(MOTIE)(P0008475, The Competency Development Program for Industry Specialist). Victor Zavala acknowledges support from the members of the Texas-Wisconsin-California control consortium.

REFERENCES

Achiam, J., Held, D., Tamar, A., and Abbeel, P. (2017). Constrained policy optimization. In D. Precup and Y.W. Teh (eds.), *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 22–31. PMLR, International Convention Centre, Sydney, Australia. URL `http://proceedings.mlr.press/v70/achiam17a.html`.

Bellman, R. (1966). Dynamic programming. *Science*, 153(3731), 34–37.

Chow, Y., Nachum, O., Faust, A., Duenez-Guzman, E., and Ghavamzadeh, M. (2019). Lyapunov-based safe policy optimization for continuous control. *arXiv preprint arXiv:1901.10031.*

Deb, K. (2000). An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2-4), 311–338.

Grüne, L. and Pannek, J. (2017). Nonlinear model predictive control. In *Nonlinear Model Predictive Control*, 45–69. Springer.

Joines, J.A. and Houck, C.R. (1994). On the use of non-stationary penalty functions to solve nonlinear constrained optimization problems with ga's. In *Proceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence*, 579–584. IEEE.

Kramer, O. (2010). A review of constraint-handling techniques for evolution strategies. *Applied Computational Intelligence and Soft Computing*, 2010.

Kreisselmeier, G. and Steinhauser, R. (1980). Systematic control design by optimizing a vector performance index. In *Computer aided design of control systems*, 113–117. Elsevier.

Lin, W.S. and Zheng, C.H. (2012). Constrained adaptive optimal control using a reinforcement learning agent. *Automatica*, 48(10), 2614–2619.

Ma, Y., Zhu, W., Benton, M.G., and Romagnoli, J. (2019). Continuous control of a polymerization system with deep reinforcement learning. *Journal of Process Control*, 75, 40–47.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602.*

Modares, H., Nageshrao, S.P., Lopes, G.A.D., Babuška, R., and Lewis, F.L. (2016). Optimal model-free output synchronization of heterogeneous systems using off-policy reinforcement learning. *Automatica*, 71, 334–341.

Pan, A., Xu, W., Wang, L., and Ren, H. (2020). Additional planning with multiple objectives for reinforcement learning. *Knowledge-Based Systems*, 193, 105392.

Petsagkourakis, P., Sandoval, I.O., Bradford, E., Zhang, D., and Chanona, E.A.d.R. (2020a). Constrained reinforcement learning for dynamic optimization under uncertainty. *arXiv preprint arXiv:2006.02750.*

Petsagkourakis, P., Sandoval, I.O., Bradford, E., Zhang, D., and del Rio-Chanona, E.A. (2020b). Reinforcement learning for batch bioprocess optimization. *Computers & Chemical Engineering*, 133, 106649.

Poon, N.M. and Martins, J.R. (2007). An adaptive approach to constraint aggregation using adjoint sensitivity analysis. *Structural and Multidisciplinary Optimization*, 34(1), 61–73.

Puterman, M.L. (2014). *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons.

Shin, J., Badgwell, T.A., Liu, K.H., and Lee, J.H. (2019). Reinforcement learning–overview of recent progress and implications for process control. *Computers & Chemical Engineering*, 127, 282–294.

Yang, T., Zhao, L., Li, W., and Zomaya, A.Y. (2020). Reinforcement learning in sustainable energy and electric systems: A survey. *Annual Reviews in Control.*

Yoo, H., Kim, B., Kim, J.W., and Lee, J.H. (2020). Reinforcement learning based optimal control of batch processes using monte-carlo deep deterministic policy gradient with phase segmentation. *Computers & Chemical Engineering*, 107133.

Zhang, P., Li, H., Ha, Q., Yin, Z.Y., and Chen, R.P. (2020). Reinforcement learning based optimizer for improvement of predicting tunneling-induced ground responses. *Advanced Engineering Informatics*, 45, 101097.