PLANT WIDE STEADY STATE OPTIMIZATION USING REINFORCEMENT LEARNING

Kalpesh M Patel* and Gabriel Winter
Saudi Aramco,
Saudi Arabia 31311

Abstract

This work applies a novel systematic methodology of implementing Reinforcement Learning (RL) for plant wide steady state offline optimization of distillate and fuel oil pool in a refinery. Rather than adding to the extensive research on augmenting existing RL algorithms, the work focuses on using a systematic implementation method that incorporates domain-specific knowledge about process constraints in formulating the RL problem resulting in reduced dimensionality, involves modified exploration process, and is applicable to any existing model-free off-policy RL algorithm supporting continuous states and actions. The work demonstrates and opens up the possibility of the use of RL for plant wide or refinery wide offline optimization as the method results in smaller agent size, lesser data requirement and no need for special hardware for training and deployment.

Keywords

Plant wide Optimization, Refinery wide Optimization, Steady State Optimization, Reinforcement Learning,

1. Introduction

RL provides an alternative to conventional Process Control and Optimization that is model-free and adaptive. It has been successfully applied to playing games and gained wide spread recognition when a computer program named AlphaGo, based on RL, decisively defeated the word champion at the Chinese game of Go in 2017. Despite the success of RL in the field of games, its use in Process Industry is not trivial as Process Control and Optimization is not a game (Patel, 2022).

There has been some recent work towards optimization using RL. Powell et.al. (2020) propose RL-RTO which includes novel hybrid training techniques for the actor and critic networks effectively focusing on augmenting RL algorithms. Though the ability to handle constraints on actions is proposed, the ability to handle constraints on other process variables and the ability to handle variables to be held at targets is not addressed. Oh et.al. (2021) proposes to use a surrogate deep neural network (DNN), trained using data gathered from a first-principle mathematical model, to train RL agent. Though the example provided includes holding two product yields at their targets, the ability to handle process constraints and economic optimization is not addressed.

This work proposes a data centric approach rather than a technology centric approach. Instead of focusing on augmenting existing RL algorithms, the presented work focuses on the data(information) being fed in and coming out of RL. It uses the systematic method, introduced by Patel. (2022) and to be published in detail in a planned Journal paper, that

- Breaks down the problem into known and unknown parts based on available domain-specific

* K, M. Patel is with Saudi Aramco, P O Box 5000, Dhahran, Saudi Arabia 31311 (phone: 966-13-8801065; fax: 966-13-8744444; e-mail: Kalpesh.patel@aramco.com).

knowledge. The RL agent is made to learn only the unknown part.

- Uses a unique transformation, in formulating the RL problem.
- Uses a unique concept of Action Adjustment to handle constraints.
- Uses modified exploration and data generation resulting in richer data for training.
- Proposes unique training stop criteria for Actor network.
- Works with any model-free off-policy RL algorithm supporting continuous states and actions.

An overview of RL is provided in Section2 followed by the systematic method discussion, at a high level, in Section 3. It is used to implement RL for plant wide steady state offline optimization of the distillate and fuel oil pool in a refinery. Section 4 discusses the refinery process involved and the scope of optimization. Section 5 describes the RL problem formulation followed by discussion and analysis of results in Section 6. The conclusions and contributions of this work is presented in Section 7 followed by future work in Section 8. The acknowledgements and references are provided at the end.

2. RL overview

Reinforcement learning is a type of machine learning algorithm that does not depend on already available data sets or models to start learning. Its ability to generate data and then learn from it distinguishes it from the other types of machine learning i.e. Supervised and Unsupervised learning. As seen in Fig. 1, RL consists of an environment, which could be a game or a process, and an agent that is to be trained on the environment to achieve a certain objective. The agent reads the current state(s(t)) of the environment at time t and generates an action(a(t)), using a policy (∏). a(t) is sent to the environment for implementation. The environment in return provides the agent with its next state(s(t+1)) and a reward (r(t)) that represents the goodness of a(t) taken by the agent. ∏ is updated using information in tuple {s(t), a(t), s(t+1), r(t)} such that the cumulative sum of the rewards is maximized. ∏ update can also be done using the return (G(t)) instead of r(t). G(t) is the discounted summation of the current reward due to action taken in current state and future expected rewards due to actions taken in future expected states. The use of G(t) allows RL to consider a delayed reward in addition to the current immediate reward when updating the policy.

$$G(t) = r(t) + \sum_{k=1}^{\infty} \gamma^k r(t+k) \quad (1)$$

where $\gamma \in [0,1]$ is the discounting factor

The other feature of RL that distinguishes it from other machine learning algorithms is the concept of exploration and exploitation. Initially when the agent does not know the environment, it explores the environment by taking random actions and generates data. As the agent learns from the generated data it starts to exploit the knowledge it has by taking actions that maximize the rewards. A tradeoff exists between the two. Too much exploration will result in agent acting randomly and too much exploitation will result in agent not being able to learn the environment behavior fully. Both will ultimately result in rewards not being maximized. The appeal of RL is that it needs minimal prior information of the environment, learns by exploration and then acts by exploiting the knowledge gained. For more details on RL, readers are directed to Sutton (2018).

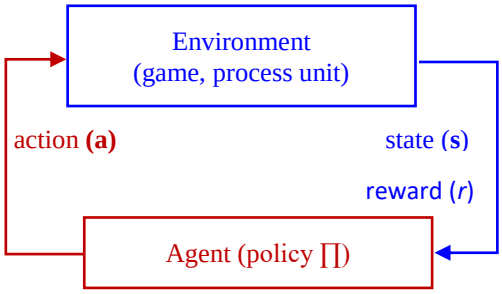

Figure 1. Agent-Environment interaction

3. Systematic Method

The details on the systematic method is planned to be published in a Journal paper. A high-level overview of the method is provided here. It begins with understanding the optimization problem at hand and formulating the RL problem (state, action and reward). We start by listing the following

- optimization objective and associated variables (included in defining reward)
- independent process handles available, or to be investigated, to achieve the objective (included in defining actions) and
- all possible constraints, that can limit the optimization, and targets that need to be honored (included in defining states and reward).

The action space consist of all independent process handles available or to be investigated. The actions could be either incremental or absolute though incremental actions are more suitable for exploration.

$$a = \{\Delta a_1, \Delta a_2, \dots, \Delta a_m\} \quad (2)$$

where m = number of independent variables

Thereafter all available domain-specific process knowledge, in the form of actions to take when a process constraint is active, is gathered. Typically, though not necessarily, this knowledge is readily available for all process constraints expected to be encountered. This, along with information on variables to be held at target if any, forms the known part of the problem. This information is used along with a unique transformation to come up with the RL state space that clearly distinguishes between situations when there are no constraints versus when there are one or more constraints active. When there are lots of

constraints, the use of the unique transformation, also results in a much smaller size of RL state space and agent.

The reward is calculated as the incremental effect of actions on the profit function shown below where p=number of product variables, q=number of feeds and r=number of utility variables.

$$Profit = \sum_{i=1}^{p}(ProductFlow_i * ProductPrice_i) - \sum_{i=1}^{q}(FeedFlow_i * FeedPrice_i) - \sum_{i=1}^{r}(UtilityFlow_i * UtilityPrice_i) \quad (3)$$

The unknown part of the problem consists of how to optimize the process when no process constraints are active along with any unavailable knowledge in handling some constraints which the RL agent is trained to learn.

Having formulated the RL problem, the agent can be trained using any off-policy model free RL algorithm that supports continuous states and actions. The exploration and exploitation are done while incorporating a unique concept of Action Adjustments to handle process constraints based on domain-knowledge already gathered. It consists of action suppression, action restriction and action enforcement under different situations.

While exploring, efforts are made not to repeat an action that has been taken before, if possible, thereby providing unique data points and improving the quality of data. This is especially the case in this work, as there is no noise in the data due to use of a simulation model.

While training, it is ensured that only data suitable for the purpose of learning the unknown part (when no constraints are active) is provided to the agent thereby enhancing the quality of data. Any reduction in quantity of data due to this is nullified by leveraging the RL problem formulation to modify the unsuitable data to make it suitable and provided to the agent.

When using an Actor-Critic RL algorithm, a simple but unique criterion is suggested to evaluate agent training and decide when to stop training phase.

4. Process description

Figure 2 shows a block diagram of the refinery process considered in this work. It is simulated using a commercially available software and calibrated to match the actual refinery operations at steady state.

The refinery feed consists of light to heavy crude oil feed to Crude Distillation Unit (CDU) and extra light feed to Condensate Stripper (CS). The main products produced consists of 91 & 95 octane Gasoline, Jet fuel, Ultra low Sulphur diesel (ULSD), low Sulphur diesel (LSD), Fuel oil (FO) and Asphalt. The refinery consists of Catalytic reformer unit (CCR) to achieve Gasoline specifications, Diesel Hydrotreater (DHT) and Hydrocracker (HCR) to enhance distillate yields.

DHT is fed by diesel cut from CDU and Vacuum Diesel cut (VDSL) from the Vacuum Distillation Unit (VDU) producing majority of ULSD product. The remaining diesel cut from CDU is sent to LSD blender while the remaining VDSL cut from VDU is sent to FO blender.

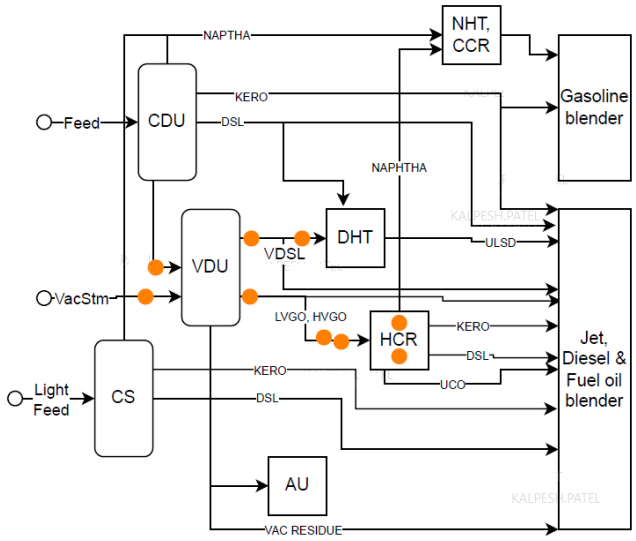

Figure 2. Refinery process block diagram

HCR upgrades the Vacuum Gas oils (LVGO and HVGO) to valuable Gasoline and Distillate blending components, while the remaining LVGO & HVGO are sent to FO blender.

Asphalt Unit (AU) is fed by VDU residue while the remaining is sent to FO blender.

The scope of optimization in this work covers the distillate and fuel oil pool primarily consisting of VDU, DHT, HCR and Diesel & Fuel oil blending system.

Table 1 shows the independent variables considered for optimization. They are also shown as Orange dots in Figure 2. Though the gasoline pool optimization is not considered in this work and the associated independent variables are held constant, the Gasoline pool and associated blending system is included in the optimization objective to address the effect of HCR naphtha changes. The feed to the refinery is also held constant in this work.

Table 1. List of Independent variables

Independent variables	Abbreviations
Vacuum feed furnace coil outlet temperature	VacCOT
Vacuum unit stripping steam	VacStm
Vacuum Diesel 95% point	VDSL_95
Heavy Vacuum gas oil 95% point	HVGO_95
Vacuum Diesel feed to DHT	VDSL_DHT
Light Vacuum gas oil feed to HCR	LVGO_HCR
Heavy Vacuum gas oil feed to HCR	HVGO_HCR
HCR 1st stage reactor inlet Temperature	HCR_INT
HCR overall conversion	HCR_CONV

5. RL problem formulation

The objective of the work is to optimize the distillate and FO blending, in order to maximize refinery profit, by adjusting the independent variables mentioned in Table 1.

The RL reward is calculated as incremental profit as in Eq. (3) where the products include that from Gasoline blending in addition to Distillate and FO blending. The feed includes both the feeds to CDU and CS. The utilities are left out of the profit function in the work as they are expected to be insignificant.

The action (a) consists of incremental changes to the nine independent variables listed in Table 1.

$$a = \{\Delta a_1, \Delta a_2, \dots, \Delta a_9\} \quad (4)$$

There are no variables to be held at targets. The constraints considered in the work include the following

- *VDSL to FO (VDSL_FO) ≥ VDSL_FO_LL*
- *LVGO to FO (LVGO_FO) ≥ LVGO_FO_LL*
- *HVGO to FO (HVGO_FO) ≥ HVGO_FO_LL*
- *Jet product (JET) ≥ JET_LL*
- *low limit ≤ independent variables ≥ high limit*

(5)

The state (s) consists of variables that represent the process state including information on the above constraints.

$$s = \{s_1, s_2, \dots, s_9\} \quad (6)$$

The RL agent was trained using the Deep Deterministic Policy Gradient (DDPG) algorithm as presented by Lillicrap et.al. (2016). The python code implementing the algorithm is taken from GitHub and modified to do the following

- Disable the built-in exploration functionality that uses Ornstein-Uhlenbeck noise process.
- Modify the learning process as per the requirements of the developed method.

DDPG is a model free off-policy RL algorithm for learning using continuous states and actions employing Actor-Critic methodology. The Actor proposes an action given a state while the Critic estimates the reward given a state and an action. Both Actor and Critic are deployed as neural networks with 2 and 3 hidden layers with 32 nodes each as shown in Fig. 3. The orange nodes represent that for the state s, the blue nodes represent that for actions a and the black node represents the reward r.

As the domain-specific knowledge related to handling all the constraints mentioned in Eq. (5) is available, the unknown part of the problem solely consists of learning how to operate the refinery when no constraints are active. In this specific case all the elements the state (s), mentioned in Eq. (6), are zeros. Hence, the parts of the Actor and Critic neural networks, shown in Fig. 3, related to the state (s) have no effect and the associated weights are not to be learnt. This further simplifies the RL agent learning.

6. Results

The agent training was performed on laptop with 1.9GHz Intel core i7, 8GB RAM Central Processing Unit (CPU) while interacting with the process simulation model also running on the same laptop. A 3-step approach is followed for agent training.

- Generate data for training while exploring the process.
- Preprocess to remove bad data and replace unsuitable data by suitable data.
- Offline agent training and hyperparameter tuning.

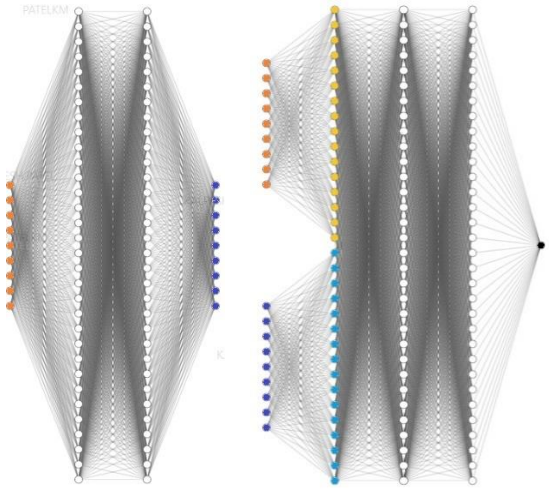

Figure 3. Actor & Critic neural networks

As the action is incremental, each of the 9 elements can be either increased, decreased or held constant simultaneously. There is a total of 19,683(3^9) possible combinations though all of them are not implementable due to presence of constraints. The agent interactions with simulation model was executed for 1200 runs, providing 1200 data points, which is approximately 6% of 19,683 combinations. Though, this step requires the most time as the simulation model takes about half a minute for each run, it is a onetime activity. Figure 4a shows the 9 independent variables and Figure 4b shows the 4 constraint variables trend when agent was exploring. The absolute values of the independent variables have been scaled so that low limits are 0% and high limits are 100%. Similarly, the absolute values of constraints variables are adjusted so that the low limits are 0 kbpd. Note that the limits on both the variables are honored when exploring.

Subsequent data preprocessing is done to remove any outliers. Offline agent training and hyperparameter tuning was then performed with agent using half the data as training data and the remaining half as validation data. Figure 5a&b shows the trend of Critic and Actor losses respectively. The critic loss reduces to a very small value for both training (CL) and validation data (CL_val) after 40 steps indicating critic network is trained reasonably well. The occasional spikes seen in CL_val are due to some data

points not getting modelled well. Thereafter at step#100, the actor network training starts where the actor loss (AL) too settles at a negative value within the next 30 steps indicating actor network is also trained. A few other criterions (to be published in the detailed Journal paper) are also used to evaluate critic and actor network learning. Once the hyperparameters are fixed, the RL agent training takes only 2 minutes on the laptop.

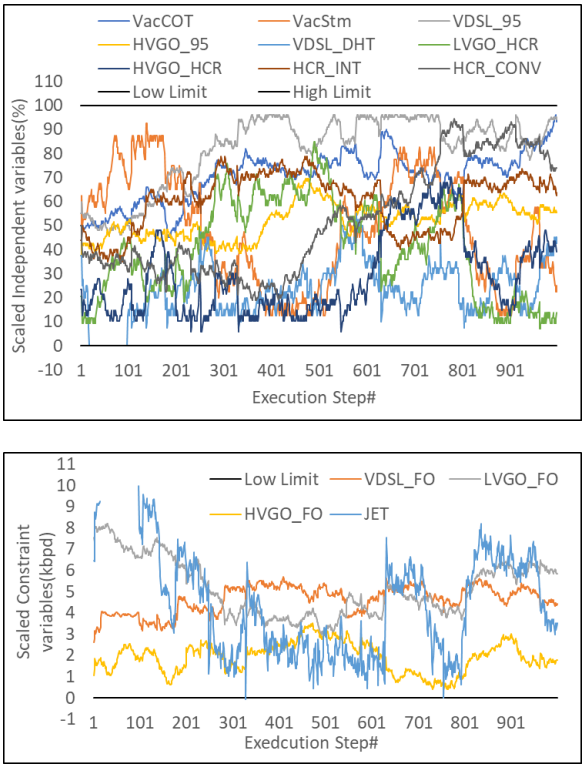

Figure 4. Agent exploration honors variable limits; a) Independent variables; b) Constraint variables

Table 2 summarizes the final RL hyperparameters used in this work. Table 3 summarizes the learnings of the RL agent in terms of direction in which it recommends each of the 9 action elements or independent variables to be changed to increase the operating profit. Under the heading of "Optimized run" it also lists the total change on each independent variable that the trained agent implemented, in a sequence of small increments, in order to optimize the process given the limits on independent and constraint variables. RL agent increased the VacCOT, VDSL_95 and HVGO_95 in order to increase the VDSL, LVGO and HVGO product draws in VDU. It then diverts the increase in the product draws to DHT and HCR thereby resulting in an increase in ULSD production. RL agent also increased the HCR_INT while decreasing the HCR_CONV which resulted in higher aromatics content in its Naphtha product resulting in a switch from 91 octane Gasoline to 95 octane Gasoline.

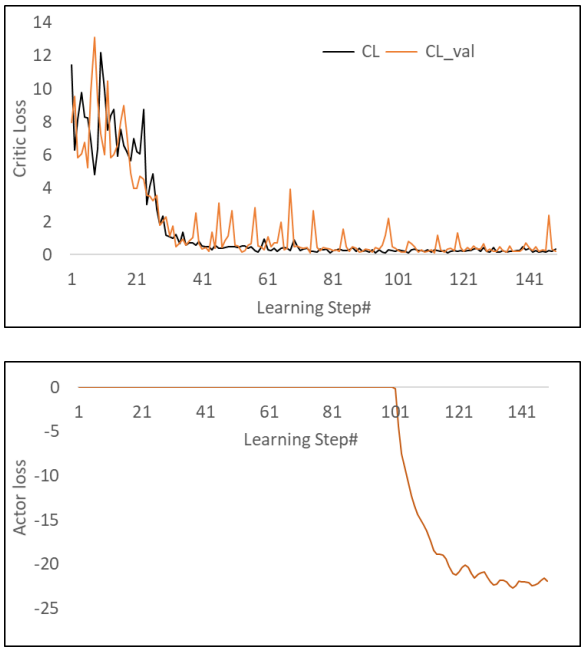

Figure 5. a) Critic loss for training & validation data; b) Actor loss

Table 2. Summary of RL hyperparameters

Hyperparameter name	Value
Buffer size	1200
Batch size	32
Learning rate (Critic)	0.0075
Learning rate (Actor)	0.1
Discount factor	0
#Learning steps	150
Actor training start step#	100

Table 3. Optimization direction for Independent variables

Independent variable	Optimization Direction	Optimized run
VacCOT	Increase	+1 degf
VacStm	Decrease	-1600 lb/hr
VDSL_95	Increase	+2.1 degf
HVGO_95	Increase	+2.1 degf
VDSL_DHT	Increase	+1 kbpd
LVGO_HCR	Increase	+1 kbpd
HVGO_HCR	Increase	+1 kbpd
HCR_INT	Increase	+1 degf
HCR_CONV	Decrease	-1 %

The effect of the changes implemented by RL agent mentioned in Table 3 on the change in profit function is

shown in Figure 6. It amounts to tens of millions of $ benefit per year.

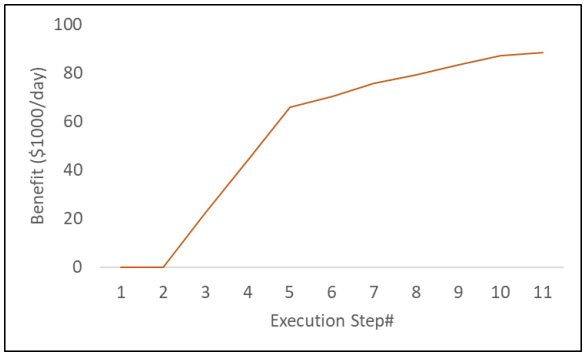

Figure 6. Change in refinery profit due to RL agent action

Product and/or feed pricing changes in the future only require a recalculation of the rewards, with the new profit calculated by Eq. (3), and retraining the RL agent which takes 2 minutes. There is no need to rerun the simulation model. Likewise, the systematic method allows adding more constraints in the RL formulation anytime without affecting the RL agent.

7. Conclusions and Contributions of this work

The work demonstrates that RL is capable of learning from a process simulation model and optimizing the process on a plant wide or refinery wide scale. The key is to break down the optimization problem in known and unknown parts based on known domain-specific knowledge while making the RL agent learn only the unknown part. Such an approach results in smaller agent size, lesser data requirement and no need for special hardware. Note that even though the number of data points available for learning is around 6% of the total action combinations possible, as mentioned in Section 6, the RL agent was able to learn, generalize and optimize the process.

The approach is compatible with the work of Oh et.al. (2021) where the use of a surrogate DNN is proposed instead of the process simulation model.

8. Future work

The work will be continued further as follows

- Present the optimized run in Table 3, recommended by the trained RL agent, to the refinery management for implementation and audit of expected benefit.
- Increase the scope of optimization to cover Gasoline blending and associated independent variables to make it full refinery wide steady state optimization.
- Investigate the options to deploy it online to make it a Real Time Optimizer (RTO).

Acknowledgments

The authors gratefully acknowledge the support provided by Saudi Aramco management in pursuing the work.

References

Github. DDPG algorithm python program available at https://github.com/keras-team/keras-io/blob/master/examples/rl/ddpg_pendulum.py

Lillicrap, T.P., Hunt, J. J., Pretzel, A., Hees, N., Erez, Y., Silver, D. and Wierstra, D. (2016). Continuous control with deep reinforcement learning. *International Conference on Learning Representations*. San Juan, Puerto Rico. https://arxiv.org/abs/1509.02971

Oh, D, Adams, D., Vo, N. D., Gbadago, D. Q., Lee C., Oh M. (2021). Actor-critic reinforcement learning to estimate the optimal operating conditions of the hydrocracking process. *Computers & Chemical Engineering. 149. 107280*. https://doi.org/10.1016/j.compchemeng.2021.107280

Patel, K. M. (2022). Safe, Fast and Explainable Online Reinforcement Learning for Continuous Process Control. In Proceedings of the 7th IEEE International Symposium on Advanced Control of Industrial Processes. Vancouver, Canada.

Powell, K. M., Machalek, D., Quah, T. (2020). Real-time optimization using reinforcement learning. *Computers & Chemical Engineering.*, 143, 107077. https://doi.org/10.1016/j.compchemeng.2020.107077

Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.