REINFORCEMENT LEARNING FOR CHEMICAL ENGINEERING PROCESS SYNTHESIS

Authors: Laurence Midgley and Michael Thomson

Advisor: Assoc. Prof. Adeniyi Isafiade

NOVEMBER 11, 2019

CHE4045Z

Acknowledgements:

We would like to thank several people for their contribution to this project.

Foremost, Assoc. Prof. Adeniyi Isafiade for his invaluable guidance throughout the process of formulating and executing this thesis. His wealth of experience saved us from burrowing down too many tempting rabbit holes, and ultimately enabled us to produce something of a quality that we are proud of.

We would like to also thank Prof. Nicola Illing, Amelia Midgley, Ché Thomson, Philip Uken and Emma Buckland for their proof reading and incredibly helpful feedback as we reached the final sprint of the project. Their critique allowed us to view our project from new perspectives and make important changes accordingly.

Finally, we would like to thank all of our friends, family and other animals for their endless support whilst we obsessed over this thesis. From climbing to art sessions, homecooked meals, words of encouragement and everything in between, you've made our experience of final year thesis a balanced and enjoyable one.

SYNOPSIS

This thesis demonstrated, for the first time, that reinforcement learning (RL) can be applied to chemical engineering process synthesis (sequencing and design of unit operations to generate a process flowsheet). Two case studies were used, with simple toy process synthesis problems for the proof of concept.

The first case study was a toy reactors-in-series sequencing problem with only two actions ("PFR" or "CSTR") and a known solution. The RL agent applied deep-Q learning, which is a simple well-known variant of RL. The agent was able to find the optimal configuration of reactors. The application of high level RL coding libraries, together with the development of visualisation tools in this case study makes the example accessible to chemical engineers without advanced knowledge of RL.

The second case study was a toy distillation column (DC) train synthesis problem. Solving this was more complex due to the branching structure of the DC train and the hybrid action space containing both discrete and continuous actions. Consequently, this case study began to approach the open-ended domains in which RL may have an advantage over conventional approaches.

In this case study, a P-DQN agent that could produce both discrete and continuous actions was used. The agent was able to learn and outperform multiple heuristic designs, often creating unexpected configurations for the DC trains. These counter-intuitive results are an indicator of the potential for RL to generate novel process designs that would not necessarily be found by conventional methods.

In an exploration of further developments within RL's application to process synthesis: (1) We compared RL and conventional process synthesis techniques. RL has the potential to be superior due to its ability to learn and generalise within open-ended problems, while taking advantage of computers' ability to analyse large amounts of data and rapidly interact with simulations. (2) We proposed an expansion of the RL agent's action space used in the simple case studies to a more general action space for process synthesis that could be used to generate complete chemical engineering processes. (3) We highlighted that RL is well suited to process synthesis problems governed by general equations/laws like thermodynamics. (4) We critiqued the *model free* RL approach that we used and recommended that *model is given* RL should rather be used in future developments of RL's application to process synthesis, as the *model free* RL made the problem unnecessarily complex. (5) We proposed benchmarks for future RL research on process synthesis.

In the future, RL for process synthesis is well suited to take advantage of improvements in chemistry and physics simulation. This thesis hopes to stimulate research within this area – with the long-term goal of an RL agent creating novel, profitable processes.

Table of Contents

LIST OF FIGURES

LIST OF TABLES

NOMENCLATURE:

Symbols	Definition
α	Relative volatility
A	Continuous action space
A()	Advantage function
F	Compound molar flowrate
k	Discrete action
l	Loss function
K	Discrete action space
θ	DDPN network parameters
Q	Q-value
r_a	Reaction rate (of compound A)
s_t	State
V_i	Reactor volume
V()	Value function
ω	DQN network parameters
x_k	Continuous action
X_k	Continuous action space
y_t	Target value

LIST OF ABBREVIATIONS:

Abbreviations	Definition
AI	Artificial Intelligence
CSTR	Continuously stirred reactor
DC	Distillation Column
DQL	Deep Q learning
DDDPN	Deep deterministic policy network
DQN	Deep Q Network
LK	Light key
LK_{split}	Light key split
MDP	Markov decision process
MINLP	Mixed integer non-linear programming
ML	Machine Learning
NN	Neural Network
PFR	Plug flow reactor
P-DQN	Parameterised deep-Q network
RL	Reinforcement learning

Glossary

Term[1]	Definition
Action space (RL)	The set of all possible actions a reinforcement learning agent can take
Architecture (RL)	The design and shape of a neural network
Distillation Column Train	A configuration of multiple connected distillation columns used to separate out a multi-component stream
Deep (RL)	A neural network that contains multiple hidden layers
Environment (RL)	The simulation of an Markov decision process in which an reinforcement learning agent acts
Episode (RL)	One run of the MDP from the starting state to the final state (the final state is defined within the specific environment)
Heavy Key	In a separation step between 2 specified compounds, the light key is the compound with the higher boiling point.
Hybrid action space (RL)	An action space containing both discrete and continuous actions
Light Key	In a separation step between 2 specified compounds, the light key is the compound with the lower boiling point.
Markov Decision Process (RL)	A discrete time process where at each step, an agent takes an action and the process transitions (stochastically) to the next state depending only on the previous state and the action taken. The agent receives a reward after each step.
Model free (RL)	A type of reinforcement learning where the agent does not have a model for how the environment works and only observes the state and receives rewards. A model free agent learns through trial and error.
Model is given (RL)	A type of reinforcement learning where the agent is given the model of the environment which it can use to simulate results of potential future actions to better determine current actions.
Process Synthesis	Selection and design of a sequence of unit operations
Q-value	The expected reward for taking an action within an Markov decision process
Reward (RL)	A signal value given to an agent within a Markov decision process to incentivise good behaviour.
State (RL)	A description of the current step within a Markov decision process
State space	The set of all possible states within a Markov decision process
Superstructure	A structure used in numerical optimisation containing the set of all possibly useful configurations of a set of optimised variables

[1] “(RL)” after a term indicates that it is specific to reinforcement learning/machine learning

1 Introduction:

1.1 Background

Chemical engineering process synthesis is defined as the sequencing and specification of unit operations to generate process flowsheets. Process synthesis is currently performed by humans through engineering judgement, heuristic design and computer aided optimisation. Conventional computational methods of process synthesis focus on highly constrained problems. These problems are often defined within a superstructure which exhaustively specifies the possible configurations of the process. The problem is then solved using computational methods like mixed integer non-linear programming (MINLP). These techniques require the optimal solution to be imbedded within the superstructure and cannot to be applied to open-ended problems. Reinforcement learning (RL), a type of machine learning (ML) which involves an agent making decisions within an environment to maximise an expected reward, offers the potential to solve open-ended process synthesis problems.

1.2 Problem statement

Show that it is possible to apply reinforcement learning to chemical engineering process synthesis problems and map out future developments within the field.

1.3 Scope of study

The focus of this thesis is a proof of concept to demonstrate that it is possible to apply RL to process synthesis problems. The simple case studies used in this thesis are selected as a starting point for the proof of concept. Due to this simplicity, however, conventional approaches are better suited to solving the selected problems. For example, the first case study (‘Case Study 1’) can be solved using chemical engineering judgement and the second case study (‘Case Study 2’) can be solved using MINLP. Furthermore, this thesis used a *model free* RL techniques as a simplification. In review we recommend that *model is given* RL techniques would be more applicable to process synthesis problems. These simplifications are strongly justified by the fact that since RL has never been used for process synthesis before, taking an iterative approach makes more sense than starting with an exceedingly difficult process synthesis problem. This thesis discusses the more open-ended process synthesis problems for which RL may outperform conventional approaches. Solving these more complex problems by implementing RL, however, falls outside of this thesis’ scope.

1.4 Aims and objectives

- Solve a simple chemical engineering process synthesis problem with a known intuitive solution, using basic RL techniques and high level RL programming libraries to provide an easy to understand and accessible example. (Case Study 1)
- Apply RL techniques to a more complex process synthesis problem to show the applicability of RL to more open-ended process synthesis problems. (Case Study 2)
- Outline the advantages and disadvantages of applying RL to process synthesis relative to conventional methods
- Map out key problems and potential approaches to future developments of RL's application to process synthesis.

2 Literature Review:

2.1 Artificial Intelligence in Chemical Engineering

Artificial Intelligence (AI) is widely regarded as a promising area of modern tech development. This branch of computing – concerned with performing tasks that have historically relied upon human execution – is poised to facilitate the widespread creation of economic value in various industries over the years to come. It is therefore unsurprising that many chemical engineers are interested in the prospective use of AI (in particular Machine Learning (ML)) for a variety of industrial applications with the intention of solving long-standing process engineering problems (Lee et al., 2018). What may be slightly more surprising, however, is that the use of AI within the process engineering field has already been researched for some 35 years, albeit with somewhat limited historic impact within the discipline (Venkatasubramanian, 2019).

The research conducted about the potential use of AI in chemical engineering over the past four decades offers important insights into the challenges that have historically hampered progress and industrial impact in the field. Historical AI research within chemical engineering can be roughly categorised into two major phases: the expert systems era (1983-1995) and neural networks era (1990-2005) ((Venkatasubramanian, 2019). The former involved the rapid automatic matching of relevant rules-of-thumb from a vast pool of domain-specific heuristics to a particular problem in such a way that domain knowledge and order of execution could be separated, allowing for computational flexibility to solve poorly structured problems (Hayes-Roth et al., 1983). This allowed for the development of applications such as CONPHYDE, DECADE, and MODELL.LA for predicting thermophysical properties in complex fluids, designing catalysts and design modelling respectively (Venkatasubramanian, 2019). Whilst exciting, expert systems never gained widespread traction due to the extensive effort, resources and time required to build credible applications for industrial purposes. From about 1990, industry focus therefore transitioned from the top-down design paradigm of expert systems to the bottom-up paradigm of neural networks (NNs) enabled by the increasing availability of large process data sets and advances in computational technologies and methodologies (Venkatasubramanian, 2019). This allowed for the increasingly efficient upkeep and advancement of computational models. Albeit a promising computational branch, methods did not yet exist to adequately train NNs (which “learn” from repeated testing using substantial data sets) to achieve their objectives, slowing progress in the field for more than a decade. Furthermore, AI was not as impactful in the chemical engineering field as previously expected because researchers (1) had a deficiency in the necessary computational power, storage, programming environments and communication methods and (2) lacked the copious amounts of data required by existing AI methods (Venkatasubramanian, 2019).

The developments in AI and computing in general in recent years have, however, addressed the limitations listed above, paving the way for a new wave of AI development within the chemical engineering space. Since 1985, computers have achieved an approximately 150,000-fold performance/unit cost gain in terms of hardware and have witnessed equally exciting software development progress with the advent of various programming environments made possible by the likes of Python and TensorFlow. Furthermore, advances in the chemical engineering field have allowed for access to large data sources in various domains, making possible significant advances in ML in particular (Venkatasubramanian, 2019).

In addition to the reduction in factors limiting AI development within chemical engineering, limitations in conventional computing techniques are prompting a resurgence of interest in AI research. The efficiency gains made possible using model-predictive control (MPC) and optimisation techniques have been largely realised in many chemical engineering domains (Venkatasubramanian, 2019). This saturation of existing computational methods is therefore motivating researchers to address more complex and previously unsolvable decision-making problems – many of which appear to be well suited to modern AI techniques.

Three major branches of modern AI show promise in the development of the next wave of computational methods in the chemical engineering space: statistical ML, convolutional neural networks (CNNs) and reinforcement learning (RL) (Venkatasubramanian, 2019). Whilst statistical ML falls outside of the scope of this thesis, the use of RL to train CNNs and the technology's applicability to chemical engineering is explored extensively. Although the potential of these technologies within chemical engineering is broad, perhaps one of its most exciting prospective applications is for automated simulation-based modelling (e.g. for material science and chemical process synthesis).

Process synthesis can be described as the design phase during which equipment units and streams of a process are determined in order to construct a flowsheet – providing a blueprint for the later construction of chemical processes (Nishida et al., 1981). Despite process synthesis being one of the cornerstone tasks of chemical engineering, little research into the applicability of AI within process synthesis has been conducted, with the possibility of approaching process synthesis using AI techniques only alluded to. Before exploring the mechanics and applicability of specific AI techniques, it is useful to define the components of process synthesis and the current methods generally applied to this phase of design.

2.2 Chemical process synthesis

Chemical process design is a complex hierarchy of many subtasks, including reactor sequencing, material input-output-recycle structuring, separation, energy and power systems design, safety contingency planning and a range of other subtasks dependent on the requirements of any particular process at hand. Furthermore, the complexity

resulting from the design of the numerous subsystems involved in chemical synthesis is only amplified when one observes the various manners in which these subsystems interact and the iterative modelling required to articulate, coordinate, integrate and control such subsystems.

The systematic computational methods currently used to design chemical processes (see Section 2.3) were only introduced primarily in the past three decades (Barnicki and Siirola, 2004). This computational developments in the chemical process synthesis space can be largely attributed to two major driving factors: (1) the development of increasingly powerful systemic methods of process synthesis enabled by a combination of the rapid advancements in computers and expert systems and (2) the wave of development of chemical processes prompted by globalisation (Barnicki and Siirola, 2004). The synthesis aids resulting from the former of these developments are perhaps best represented by expert (heuristics based) design practices such as PIP (Kirkwood et al., 1988), planning paradigms such as the means-ends analysis of AIDES (Siirola and Rudd, 1971) and resolution theorem-proving such as utilised by BALTAZAR (Mahalec and Motard, 1977) which together enabled many of the systemic approaches used for chemical process synthesis today (Barnicki and Siirola, 2004). These developments within the field were largely driven by the demand for new approaches to process synthesis and plant design prompted by increased focus on manufacturing efficiency and the current era of industrial globalisation (Barnicki and Siirola, 2004). This trend will likely necessitate more competitive industrial cost structures, and the ability to rapidly invent and implement advantageous technologies and methodologies will become increasingly important. Given the time consuming and costly requirements of many of the current methods of process synthesis, it is therefore going to become increasingly important to develop new approaches to maintain a competitive edge in the global market.

Whilst the foundation goal of chemical process synthesis should remain the design of processes at levels of desired scale, environmental sustainability, efficiency, safety and economic viability, it is likely that the relative importance of many of these factors will change substantially in the future. This may include changes in the importance of maintaining low carbon emissions, the cost and sources of energy, the database of available unit operations, the availability and cost of various materials of construction and various other design factors. The challenge associated with these changes is that many of the heuristics used in synthesis methodologies today, particularly those based on expert systems, will have decreasing relevance, necessitating the formulation of new or adapted approaches to chemical process synthesis (Barnicki and Siirola, 2004). Furthermore, this need for new synthesis systems is further necessitated by the limitations associated with the method most commonly used for computer aided process synthesis: mixed integer non-linear programming (Kocis and Grossmann, 1987) (Angira and Babu, 2006a).

2.3 Computer Aided Process Synthesis

Although ML has not been applied to process synthesis, many other techniques of computer aided process synthesis have a long history within chemical engineering literature (Kocis and Grossmann, 1987). The most successful and common example of computer aided process synthesis is the framing of process synthesis as a mixed integer non-linear programming problem (MINLP) (Kocis and Grossmann, 1987) (Angira and Babu, 2006b). In this formulation, the pre-defined space of possible unit sequences and conditions is embedded within a superstructure (as shown in Equation (1) and Figure 1 below) and optimised using various MINLP solving methods such as Genetic Algorithm, Evolution Strategy, MINLP-Simplex Simulated Annealing and Modified Differential Evolution (Angira and Babu, 2006b). Superstructure optimisation can therefore be described as the process of eliminating the superfluous paths and unit alternatives found within the original superstructure in order to find an "optimal" solution (Barnicki and Siirola, 2004).

$$F = \min_{x,y} c^T y + f(x) \tag{1}$$

$$\text{s.t. } h(x) = 0$$

$$g(x) \leq 0$$

$$Ax = a$$

$$By + Cx \leq d$$

$$x \in X = \{x | x \in R^n,\ x^L \leq x \leq x^U\}$$

$$y \in Y = \{y | y \in \{0,1\}^m,\ Ey \leq e\}$$

For the equations above, x is a vector of continuous variables representing a process unit's operating and design conditions (e.g. temperature, size) and y is a vector representing the existence/absence of units within the superstructure, satisfying the defined constraints (e.g. mass balance and unit configuration constraints). $c^T y$ represents the fixed unit cost. $f(x)$ represents the revenues, operating costs and unit size-dependant costs.

Examples of process synthesis MINLP problems extend from simple reactor selection to more complex multi-step process configuration (Kocis and Grossmann, 1987). An example of a flowsheet configuration problem superstructure is shown in Figure 1 below. The superstructure shows that the general process major steps have been already pre-defined (e.g. compression step, reaction step). The optimisation takes place within each of these pre-defined steps, determining the scale, operating conditions and designs of each unit and the material flows through each pipeline by optimising according to desired performance criteria (Barnicki and Siirola, 2004).

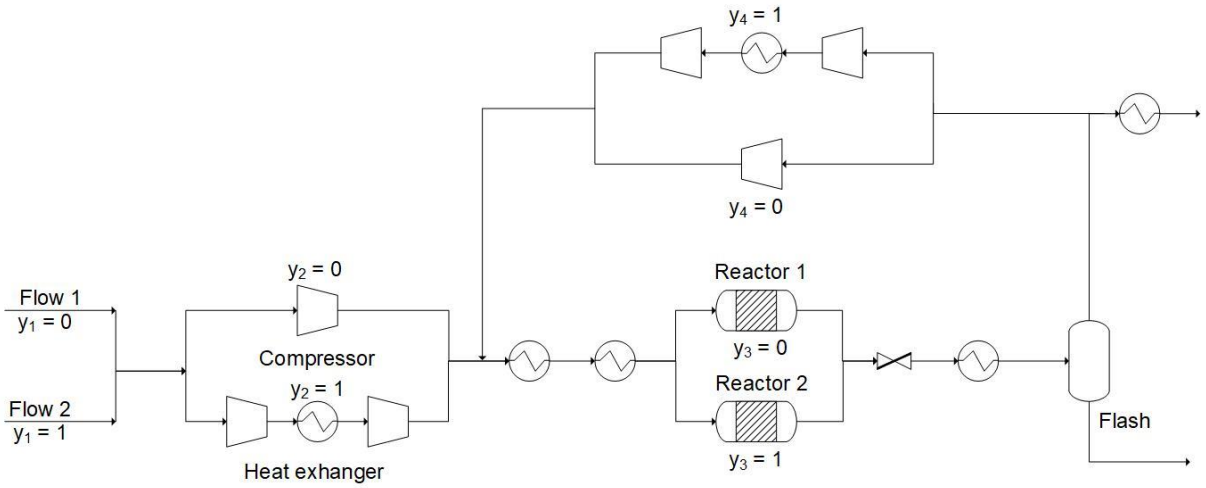

Figure 1: Superstructure example based on [1]. Where $y_i = 1$ denotes the existence of a unit/step within the optimised superstructure. A single example of each major unit type is labelled.

The main issue with this approach is that it requires the specification of the complete space of configuration possibilities. The optimal solution can only be found if it is described within a comprehensively descriptive, pre-defined superstructure. For applications such as heat exchanger network optimisation this approach is applicable as it can be certain that the optimal solution is described within the superstructure (Yee and Grossmann, 1990). This is because the space of configurations is constrained by the heating and cooling units and requirements of the predefined system. Superstructure optimisation techniques are, however, unable to solve problems of the complexity synonymous with generalised chemical process synthesis (Barnicki and Siirola, 2004). Generally, the MINLP approach has applicability in narrow, highly constrained process synthesis problems where the general possible configurations are known. In true process design - such as that conducted by human chemical engineers - the space of possibilities is, however, very large/infinite and the MINLP approach is no longer applicable as the superstructure is increasingly difficult to define and the computational requirements become excessively large. The potential use of RL to solve such process synthesis problems is promising and does not seem to have been explored in existing chemical engineering literature. In order to understand why the use of RL and its various branches are particularly well suited to chemical process synthesis, one must first understand the mechanics and capabilities of this method of computation.

2.4 Reinforcement Learning

RL is a method of Machine Learning (ML) that concerns sequential decision-making. More formally, the RL application can be described as an “agent” that is tasked to make decisions within a pre-defined environment in order to maximise a provided notion of cumulative reward (Francois-Lavet et al., 2018). The agent *learns* an appropriate behaviour (sequence of actions) by modifying its decision-making policy

incrementally. Importantly, the agent does so by using a trial-and-error approach without requiring complete control or knowledge of the relevant environment (in contrast to dynamic programming which assumes full initial knowledge of an environment) (Francois-Lavet et al., 2018). Instead, the agent need only be capable of collecting data and interacting with the environment.

The RL application is structured as a discrete time stochastic control process where the agent interacts with a pre-defined environment as shown in Figure 2. After gathering an initial observation ($\omega_0 \in \Omega$) whilst inhabiting an initial state in the environment ($s_0 \in \mathcal{S}$), the agent takes an action ($a_t \in \mathcal{A}$), following which there are three consequences: (1) a reward is obtained by the agent ($r_t \in \mathcal{R}$), (2) the state transitions to ($s_{t+1} \in \mathcal{S}$) and (3) a new observation ($\omega_{t+1} \in \Omega$) is obtained by the agent causing the cycle to repeat.

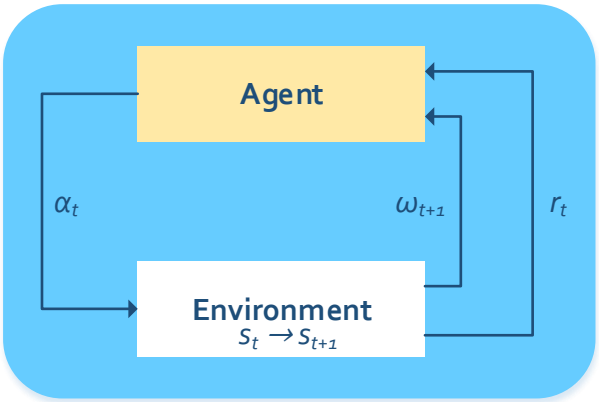

Figure 2: Interaction between agent and environment in RL (Bellman, 1958)

RL problems can generally be described as Markov Decision Processes (MDPs) (Bellman, 1957). A discrete time stochastic control process is said to be *Markovian* if the future of the process is only dependent on the current state, which captures all relevant historic information (Ashraf, 2018). This can be represented mathematically by:

$$P[St+1|St] = P[St+1|S1, \dots, St] \quad (2)$$

where St is the current state. Given that the *Markovian* property is satisfied, the MDP is a discrete time stochastic control process defined as a 5-tuple $(\mathcal{S}, \mathcal{A}, T, R, \gamma)$ where (Bellman, 1957):

- $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces respectively
- $T: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0,1]$ is the conditional set of transitional inter-state probabilities
- $R: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{R}$ is the reward function with $\mathcal{R}$ defined as a continuous set of rewards in the range $R_{max} \in \mathbb{R}^+ (\text{e.g.}, [0, R_{max}])$
- $\gamma \in [0,1)$ is the discount factor used to calculate the total discounted rewards from the current time frame

At a given time step (t), the state transition function $T(s_t, a_t, s_{t+1})$ gives the probability of transitioning to state s_{t+1}, whilst the reward function $R(s_t, a_t, s_{t+1}) \in \mathcal{R}$ gives the associated reward (Bellman, 1957).

RL agents use policies to select appropriate actions for given states. In stochastic cases, policies are described by $\pi(s,a): \mathcal{S} \times \mathcal{A} \rightarrow [0,1]$ where the probability that a given action (a) is chosen for a particular state (s) is denoted by $\pi(s,a)$ (Francois-Lavet et al., 2018). The overall goal of an RL agent is to find the policy that optimises expected cumulative reward (return). This reward can be referred to as Q-value, which gives the value of taking a particular action given a state when following a particular policy (Greaves, 2019).

A strategy commonly implored to choose the sequence of actions that generates the highest reward is the use of the Bellman Equation for the Q-value function (Bellman and Dreyfus, 1962). This equation can be simplified as:

$$Q(s,a) = r(s,a) + \gamma \max_{a} Q\,(s',a) \tag{2}$$

which states that the highest Q-value obtained from selecting action a when in state s is equal to the immediate reward received, $r(s,a)$, plus the maximum possible Q-value obtained from the following state (s') (Zychlinski, 2019). This method of seeking the maximal Q-value is called a *greedy* approach. In order to prevent the algorithm from continuously selecting the same best actions, an ***ε-greedy*** approach is used in which the *greedy action* is chosen with probability $p = 1 - \varepsilon$ (or a random action with probability ε), allowing the RL agent to both explore and exploit new opportunities and maximal Q-value state-action sequences respectively (Zychlinski, 2019). Q learning is the approach whereby the Q-values for all of the possible state-action pairs are stored in a Q- table, which is iteratively updated using the agent's experience.

2.5 Deep Q networks (DQNs)

Whilst Q-learning is a powerful algorithm for choosing actions, its dependency on storing all of the possible state-action pairs becomes computationally infeasible when dealing with large state-action spaces (Analytics Vidhya, 2019). In order to solve this computational problem, a NN is used to approximate the Q-value function in "deep Q-learning (DQL)" (Mnih et al., 2015a). In DQL, the state is inputted and associated Q-values for all possible actions are outputted after computing a policy approximated by a NN (known as a deep Q network (DQN)).

The NN consists of multiple layers of interconnected "neurons" that cause non-linear transformations to the input state, outputting the approximation of the Q-value for each of the possible output actions – which is then used to select the best action (Erhan et al., 2009). These layers are trained incrementally to minimise empirical error using a method of gradient descent through the backpropagation algorithm (Rumelhart et al., 1988). At every iteration the algorithm adapts the internal parameters of the NN to fit

the desired function by a predefined “learning rate” weighting. A schematic of a simple example NN can be found in Figure 3 below.

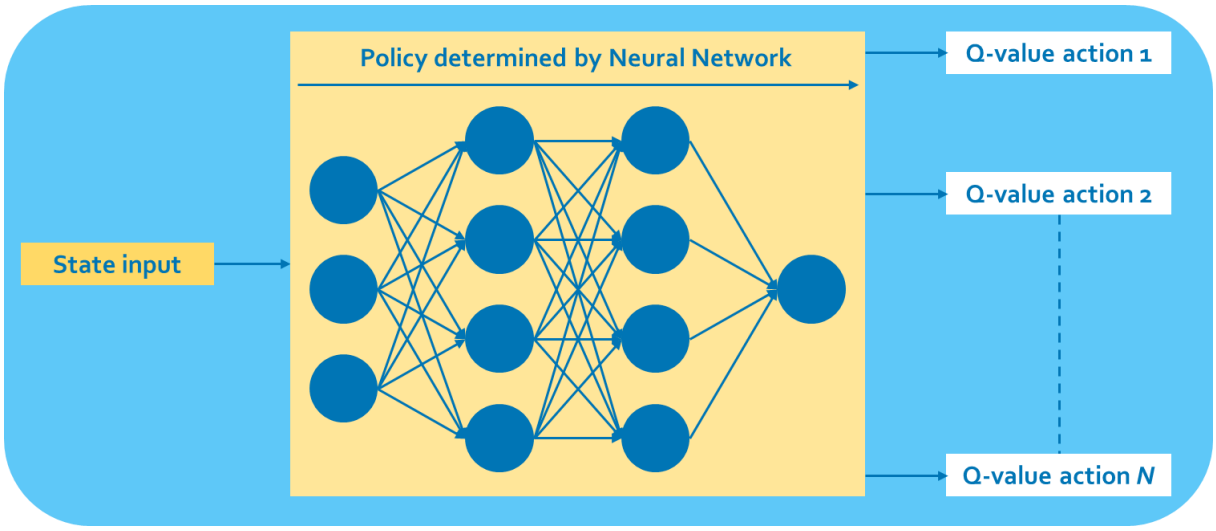

Figure 3: Example of NN architecture used in DQL

Since being devised in 2015, DQL has been used for a broad variety of real-world applications, illustrating the potential and versatility of the method. DQL has been used to outperform humans in complex games such as Go (Silver et al., 2017) and the Atari 2600 suite (Mnih et al., 2015a), traffic light control (Arel et al., 2010), personalised news recommendations (Zheng et al., 2018), 3D model generation (Choi et al., 2019), and a plethora of other applications. Given this promise and relative newness of the DQL methodology, it is likely that its potential applications within the realm of chemical engineering are both broad and largely underutilised.

2.6 Application of Reinforcement Learning to Chemical Engineering

Current proven applications of ML to chemical engineering include molecule/material design, process scheduling, fault detection, process modelling, optimization and control (Hajjar et al., 2018) (Bailey, 2019). Deep RL has been used to optimise the conditions of a chemical reactions and found that their model outperformed standard black-box optimisation algorithms in both simulated and real reactions (Zhou et al., 2017). However, deep RL and NNs in general have not been applied to process synthesis problems.

Using an RL agent allows for process synthesis enables design within a much larger space of possible configurations and is therefore better suited to complete process design problems (generating completely new processes). Further advantages are that: (1) Deep RL agents are well suited to interaction with a simulation (e.g. Aspen) as they have been largely developed through application to computer games (Aspen can be thought of as a computer game for chemical engineers). This allows for improvements alongside the development of better chemical and physical process simulators. (2) Deep RL is well suited to generalisation – allowing for easy application to synthesis of

multiple different types of chemical engineering designs. Illustrative examples of the generality of deep reinforcement learning are: Deepmind's alpha zero agent was able to achieve superhuman performance in chess, shogi and go and Deepmind's deep Q-network agent was able to achieve professional human level performance in 49 Atari games (Silver et al., 2018) (Mnih et al., 2015b). (3) Deep RL has shown that it can learn without the use of heuristics (which ultimately constrain the possibilities). For example, alpha zero did not require any hard-coded heuristics to be coded in (besides the rules of the game), and learning was entirely completed through self-play.

3 Methodology

3.1 Introduction

RL systems were designed to address two major synthesis problems in this thesis: basic reactor sequencing and distillation train design. Both problems were intentionally addressed as “toy problems”. This essentially means that each was developed as means of exploring the various potential capabilities of RL within the scope of chemical process synthesis without necessarily developing solutions that are mature enough to be implemented in real world applications. The primary focus of the experimental was therefore on attempting to build RL agents that could address the kinds of sub-problems associated with more general chemical process synthesis, therefore providing a “proof-of-concept” for later development in the space. This iterative approach to testing and developing RL solutions within a given space is typical of the RL community and will hopefully provide a useful foundation for later collaborative work.

The reactor sequencing problem was used as a means of illustrating the basic functionality of RL within the context of chemical process synthesis by framing the synthesis problem as an MDP. Importantly this problem has a known optimal solution, which allowed the learning capacity of the agent to be demonstrated and assessed. Whilst the problem itself was straightforward, the objectives achieved were significant and showed the potential of adapting existing RL architectures to a novel process synthesis application.

The DC sequencing problem was addressed to illustrate the potential capabilities of RL to solve synthesis problems more illustrative of real-world chemical process synthesis, and included comparison to solutions generated using a conventional heuristics-based approach. This problem was more characteristic of practical process synthesis problems because it involved (1) a mixed continuous and discrete action space, (2) a branching structure and (3) has a far broader and more open-ended solution space that is non-trivial to optimise using even conventional methods. This problem was solved using a combination of various new RL methods including a parameterized deep Q network (P-DQN) to allow the use of RL in a hybrid action space 2018 (Xiong et al., 2018), a duelling layer to improve the agent’s discrete policy performance (Wang et al., 2015) and asynchronous methods (Mnih et al., 2016) to allow for fast learning using a standard CPU.

The complete code for the project, including a short guide and GIFs of the agent making decisions in real time within the environment[2], was uploaded to a Github repository for open access at www.github/lollcat/RL-Process-Design.

[2] GIFs of the agent include the step by step generation of the BFD over time as well as other graphics

3.2 Case Study 1: Reactor Sequencing Problem

3.2.1 Introduction and problem definition

The objective of the reactor sequencing problem was to solve a straightforward chemical engineering process synthesis problem with a known and simple solution using basic RL architectures and high level RL programming libraries. It is important to note that whilst this problem can be more efficiently solved using other techniques (e.g. using a Levenspiel plot and engineering judgement), the purpose of Case Study 1 was to demonstrate that process synthesis problems can be framed as MDPs and solved using RL algorithms.

Custom rate equations were designed such that the "best" reactor (maximising conversion per volume) varied depending on which position in the sequence it was located (e.g. PFRs are best when the conversion is low and CSTR's are best when the conversion is high). This meant that the agent needed to learn which reactors were conducive to optimal performance at any observed state.

The Case Study 1 problem was framed as follows:

Given Equation (1) and (2) defining the reaction system, with a given the molar flowrate of compound A (F_{A0}) of 1 units.s^{-1}: Select a "PFR or CSTR" in a series of 4 reactors, each with a volume of 0.8 units^3, in order to maximise total conversion in which the ideal CSTR and PFR equations are defined by Equations (3) and (4) respectively.

$$A \rightarrow B \tag{1}$$

$$\frac{F_{A0}}{-r_a} = -10\,X^2 + 10X + 2 \tag{2}$$

$$V_n = \frac{F_{A0}(X_n - X_{n-1})}{-r_a} \tag{3}$$

$$V_n = F_{A0} \int_{X_{n-1}}^{X_n} \frac{\partial X}{-r_a} \tag{4}$$

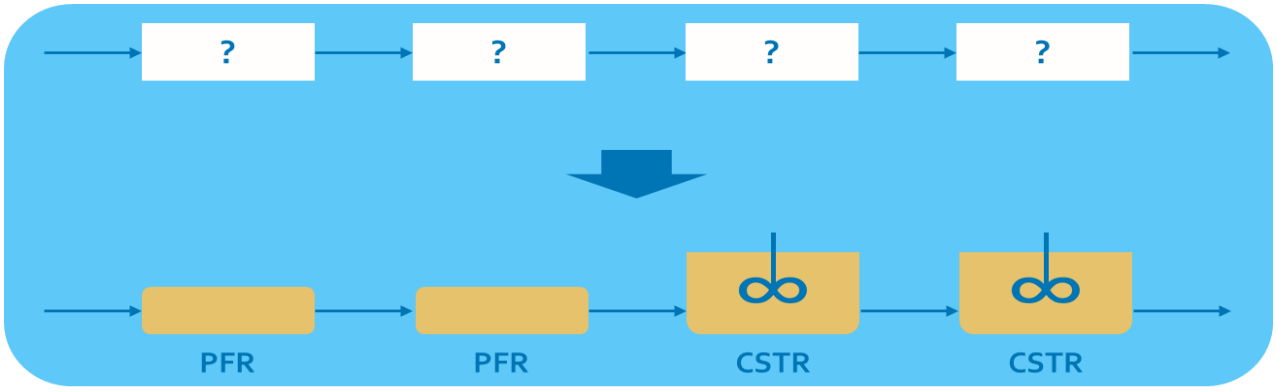

Figure 4: Reactor problem solution

The ideal solution to this problem is straightforward: it is best to choose two PFRs and then two CSTR, resulting in a final conversion of 100%. This can be intuitively represented in a Levenspiel plot, as shown in Figure 5. The volume of a PFR is represented by the area under the curve (between the inlet and outlet conversion) while the volume of a CSTR is represented by a rectangle, with a width of the change in conversion and a height of the point on the curve at the outlet conversion. This means that when the conversion is below 50%, while the curve is upwards sloping, it is better to use PFRs (as PFR get more conversion per volume), while above a conversion of 50% the graph is sloping down and CSTRs provide an improved choice.

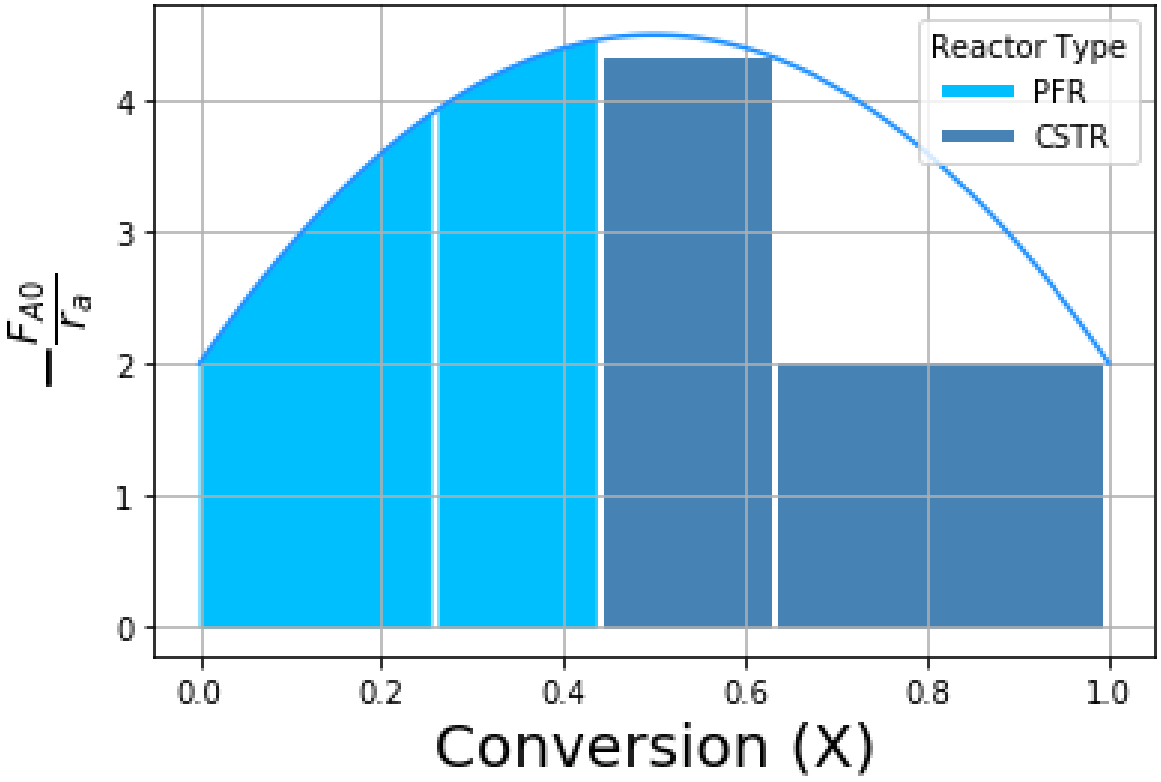

Figure 5: Livenspiel plot of Reactor Case Ideal Solution

3.2.2 Reaction Sequencing Environment MDP

As described in Section 2.4 in order to define RL problems as an MDP the following needed to be defined: **(1)** a state which is input into the agent in order to make a decision, **(2)** an action which the agent selects, **(3)** a state transition function that calculates the next state of the environment and **(4)** a reward used to incentivise the agent to select the best actions. A single step diagram of the reactor sequence MDP is shown in Figure 6 below where: **(1)** the state is the inlet conversion, **(2)** the action is the choice of reactor type (PFR or CSTR) of a fixed volume 0.8 units^3, **(3)** the state transition is the simulation of the reactors outlet conversion using the reaction system (Equation (2)) and relevant design Equation (3) or (4) for the CSTR and PFR respectively and **(4)** the reward is the increased conversion achieved by the selected reactor.

The code for the environment was written in the format of OpenAI Gym – as this is the general format used in reinforcement learning literature, and allows for integration with Keras-RL (Brockman et al., 2016).

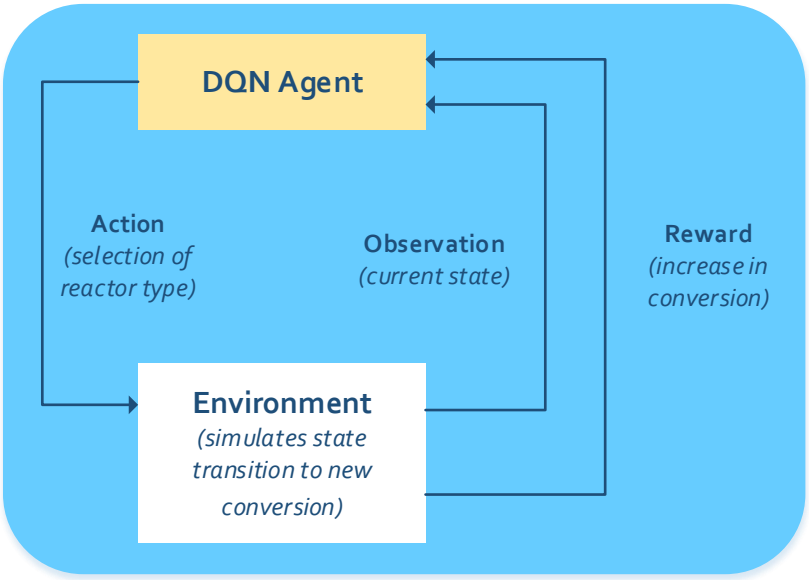

Figure 6: Reactor Sequencing MDP

3.2.3 Agent: DQN using Keras-RL

A DQN agent, as described in Section 2.5, was used to produce the two possible discrete actions (PFR or CSTR). The agent was implemented with Keras and Keras-RL libraries.

Keras is a high-level deep learning library that is used to create the NNs. Keras-RL is a library used to create deep reinforcement learning agents for application to environments in the format of OpenAI Gym and is easily integrated with Keras defined NNs. Keras-RL was used to create the DQN agent for Case Study 1.

Since Keras-RL is high-level and designed for accessibility, the code required for the DQN agent is relatively straightforward to understand. This should enable chemical engineers without in-depth knowledge of RL to apply RL to simple (discrete action space) process synthesis problems. The key parameters (including the size and shape of the NN and other hyperparameters like the learning rate) can be easily adjusted – allowing for the efficient assessment of their effects on the agent's performance. Furthermore, the simulation environment can be adapted to address other chemical engineering problems involving a discrete action space whilst using the same original agent. This in turn allows increasingly efficient experimentation with solving similar problems in the process synthesis space using RL. The goal of using Keras-RL is therefore to provide a simple, easy-to-follow RL framework for chemical engineering students and researchers interested in using RL in their respective research areas.

3.3 Case Study 2: Distillation Column Train Sequencing Problem

3.3.1 Introduction and problem definition

The objective of the DC train sequencing problem was to solve a more complex process synthesis problem. This problem could be solved using existing computational methods (such as MINLP). It does, however, represent a step towards more open-ended domains where RL may have value over conventional approaches.

The DC train synthesis problem can be considered a step closer to complete process design in the following 3 ways:

1. It contains an action space with both discrete and continuous actions (hybrid action space)[3]. Typical process design involves the selection of discrete unit types and the continuous conditions under which each unit operates (e.g. temperature, pressure, etc), resulting in a hybrid action space.
2. It involves a branching structure whereby each new column creates two new streams to be separated (unlike the reactors which were in series and had a one-one stream in to stream out mapping). This is reflective of many unit operations included in general process synthesis.
3. It is significantly more open-ended with a larger space of possible configurations compared to the reactor sequencing toy problem already described.

The DC train synthesis sample problem addressed in this thesis was adapted from process synthesis literature (Rodrigo B and Seader, 1975) and can be paraphrased as follows:

Given the hydrocarbon feed, as defined in Table 1 below, select a sequence of DCs (for each of which a light key and a light key split factor are specified) in order to optimally satisfy a pre-specified performance criterion.

Table 1: DC Train Synthesis Problem Specification

Compound	Compound symbol	Flowrate (kmol/h)
Ethane	A	3
Propylene	B	5
Propane	C	12
1-Butene	D	2
n-Butane	E	9
n-Pentane	F	7

This particular problem was chosen for several reasons. Foremost, it pertained to the separation of 6 potential products, corresponding with 42 possible distillation train sequences. This meant that the problem was complex enough to demonstrate several

[3] The discrete action is the selection of the light key. The continuous action is the selection of the split factor. Refer to the problem definition below.

of the potential capabilities of the RL agent relating to points 1-3 above, but straightforward enough to develop several base designs using the heuristics conventionally used for DC train synthesis to later benchmark the performance of the RL method. In addition to this the example problem involved compounds often involved in separations in the petrochemical industry for which pricing and physical/chemical properties are readily available online, adding to the practical validity of the solutions generated. Importantly, this particular problem was also chosen due to the fact that the mixture of hydrocarbons processed can be considered ideal, enabling ordinary distillation to be used. By limiting the solution space to ordinary distillation, the risk of breaking unforeseen physical constraints was minimised, maximising the practical reliability of the resulting DC train designs. The 7 conditions satisfied to ensure that the system can be assumed to be ideal are as follows (Seider et al., 2009):

1. The relative volatility between the light and heavy keys selected is always greater than 1.05.
2. The reboiler duties are unlikely to be excessively large, as would be the case where the two key components have a low relative volatility and the lighter of the components has a very high heat of vaporisation (e.g. water).
3. It is unlikely that the resultant tower pressures will cause the mixture to approach super-critical temperatures.
4. The overhead vapour can be partially or completely condensed to ensure that reflux can be achieved without the use of refrigerants.
5. Bottoms temperatures are not sufficiently high to cause the chemical decomposition of components in the mixture.
6. Azeotropes do not complicate the desired separations.
7. Tolerable column pressures are likely to be used (especially should operations take place under vacuum or near-vacuum conditions).

The DC sequencing for this problem was performed with the goal of maximising each of the performance metrics as defined in the “rewards” description in Section 3.3.4.

3.3.2 Column modelling and cost correlations used

Several correlations were used to model the DCs in terms of operational conditions, physical dimensions and cost. More specifically these correlations included calculating several major design specifications (height, internal diameter, average vessel thickness, number of trays, design pressure, condenser/reboiler temperatures and among several other minor specifications), capital costs (based on unit dimensions and operational conditions) and revenue associated with the sale of purified product streams. A detailed description of the methods used for modelling and costing can be found in Appendix A.

Several of the correlations and variables mentioned above were in turn used to structure the sparse reward functions discussed in Section 3.3.4. Since these reward

structures used to train the RL agent calculated rewards only at the end of each training episode, the same rewards were used to assess the heuristics-based sequences, allowing for a fair overall comparison of the various methods of sequencing used.

3.3.3 Development of heuristic base cases

Before solving the outlined DC train synthesis problem using RL, several base sequences were first generated using conventional synthesis methods. These sequences were produced in order to provide bases for later performance comparison with those proposed by the RL agent. Several heuristics found in literature (Seider et al., 2009) were used to determine these base sequences, after which the respective DCs were modelled and costed using the methods described in Section 3.3.2. These methods of modelling and costing were kept the same as those used in the RL agent's environment in order to ensure later comparisons were made fairly.

When the number of products requiring separation in a DC train are equal to or less than four, it is possible to cost and design all possible column sequences in order to find the optimal sequence (Seider et al., 2009). As the number of products is increased, however, this approach rapidly becomes infeasible, traditionally necessitating the use of various heuristics to sequence the columns with the intention of satisfying common performance metrics (such as cost). For the scope of this thesis, four of the most common heuristics for ordinary DC sequencing were relevant to the sample problem and used to generate the heuristic base sequences. These heuristics were the following (Seider et al., 2009):

i) Remove products individually in the distillate of each column (i.e. remove in order of increasing boiling point - referred to as the "direct sequence").
ii) Remove the components in the order of decreasing size of molar flowrate.
iii) Remove components in a way that light keys are chosen in the order of decreasing relative volatility (so that the most difficult splits are conducted once other compounds have already been removed).
iv) Separate the components in an order such that the molar flowrates of bottoms and distillates are kept as near as possible in each of the columns.

Whilst these heuristics have been developed over an extended period of time through the collective experience of a host of engineers, they are imperfect and often conflict with each other (Seider et al., 2009). It is therefore conventional to develop several sequences using each applicable heuristic after which each sequence is assessed and compared using relevant performance criteria. For this reason, a solution sequence was generated for each heuristic listed above, resulting in three different separation sequences (heuristics ii) and iv) resulted in the same separation sequence) These sequences are illustrated symbolically below where particular components are shown by their corresponding letters (as shown in Table 1) and white and light orange blocks represent non-product and product streams respectively.

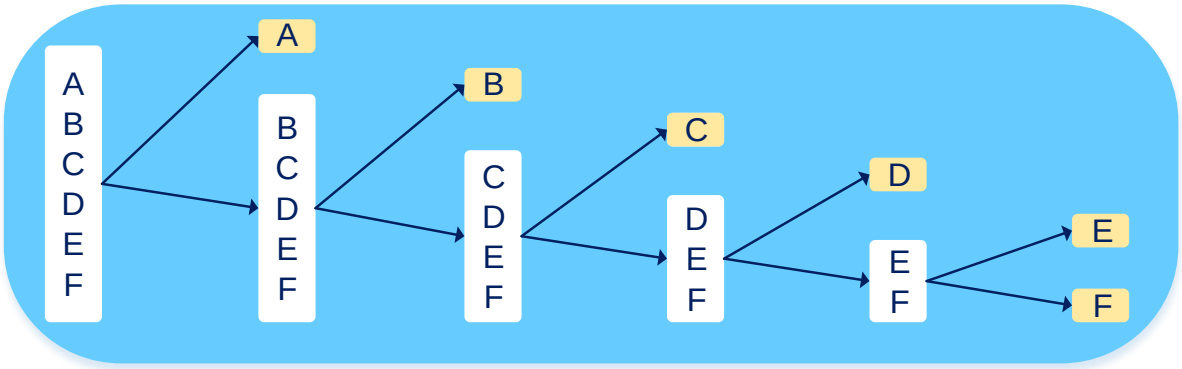

Figure 7: Separation sequence developed using heuristic i) (separation by increasing boiling point)

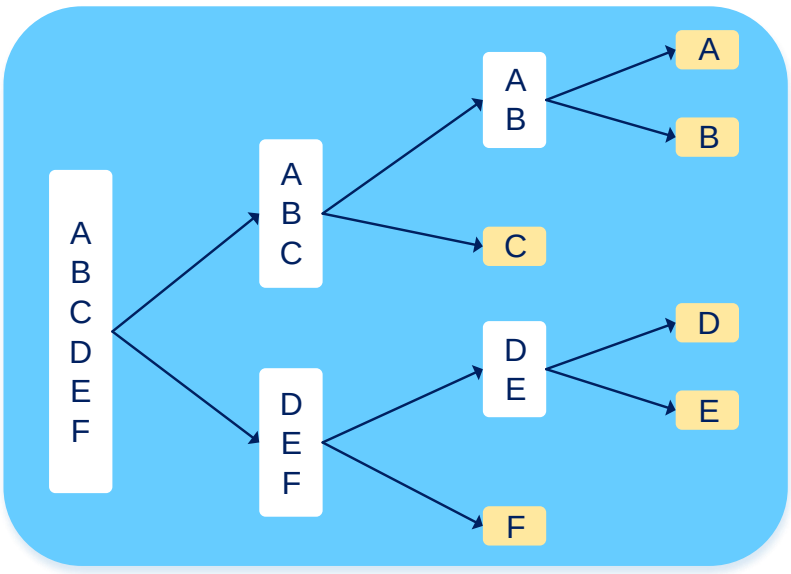

Figure 8: Separation sequence developed using both heuristics ii) and iv) (separation by decreasing molar flowrate and favouring equimolar distribution respectively)

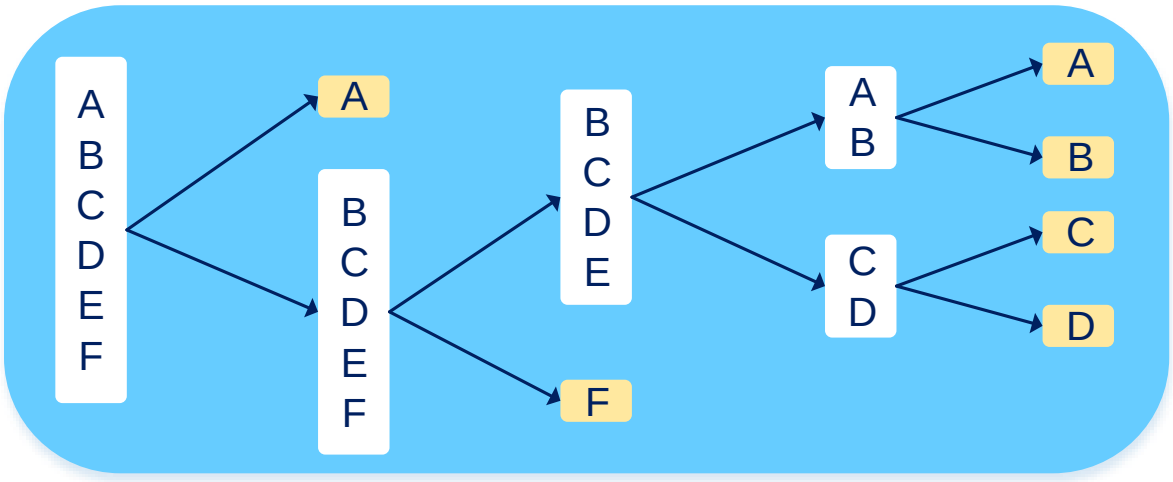

Figure 9: Separation sequence developed using heuristic iii) (separation by decreasing relative volatility)

The sequence determined using decreasing relativity as represented by Figure 9 utilises adjacent relative volatilities found in literature as presented in Table 2 (Rodrigo B and Seader, 1975). In contrast to this, Henry's and Raoult's laws and relevant Antoine equations were used by the RL agent to determine more accurate relative volatilities as a function of pressure and temperature, enabled by real-time simulation of column conditions.

Table 2: Adjacent relative volatilities used for sequencing by heuristic iii)

Adjacent relative volatility	Approximate value
α_{AB}	3.5
α_{BC}	1.20
α_{CD}	2.7
α_{DE}	1.21
α_{EF}	3.0

3.3.4 Distillation Environment MDP

As is essential for the use of RL, the DC train synthesis problem can be modelled as an MDP, for which a single step is shown in Figure 10 below. The paragraphs that follow the diagram explore the details and rationale pertaining to each aspect of this MDP structure in order to provide insight into the overall structuring of the RL system.

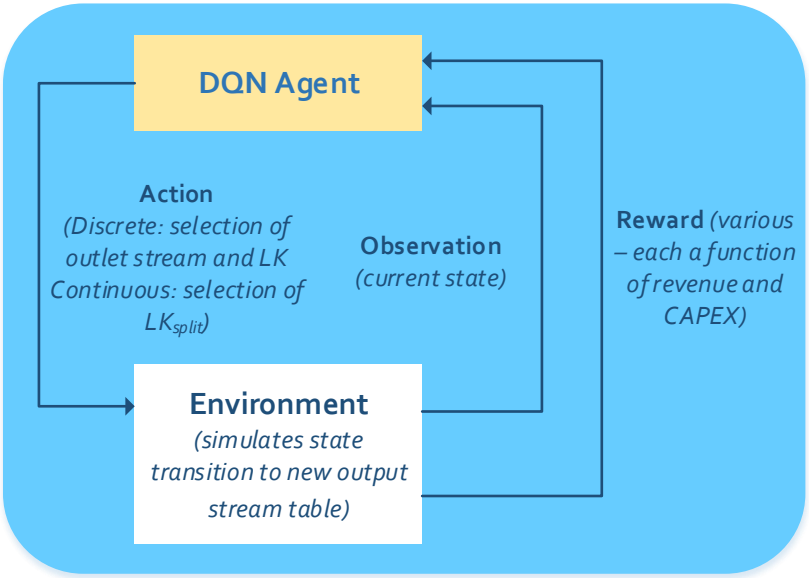

Figure 10: The distillation train synthesis problem structured as an MDP

State: The state was defined as the compound flowrates of all the streams available for further separation. In the first step of an episode, the state was the starting feed stream, after which the state was defined as all of the streams exiting columns that were available for further separation (i.e. those that weren't connected to another column). The NN required that the array representing the state that is inputted into the

NN was of a fixed shape. To ensure this format, an empty outlet-stream table was created (with 0's as entries) of *m x n* shape where *m* was the number of compounds and *n* was the maximum number of outlet streams. This stream table was then populated with new streams (tops and bottoms from the recently specified DC) whilst old streams were removed as the episode proceeded. An example of a state is shown in Table 3 for the column configuration shown in Figure 11. In this example, streams four to seven are the streams exiting columns that are available for further separation. Since there are two fewer outlet streams available for separation than the defined maximum limit in this example, there are two placeholder streams in the array with component flowrates equal to zero.

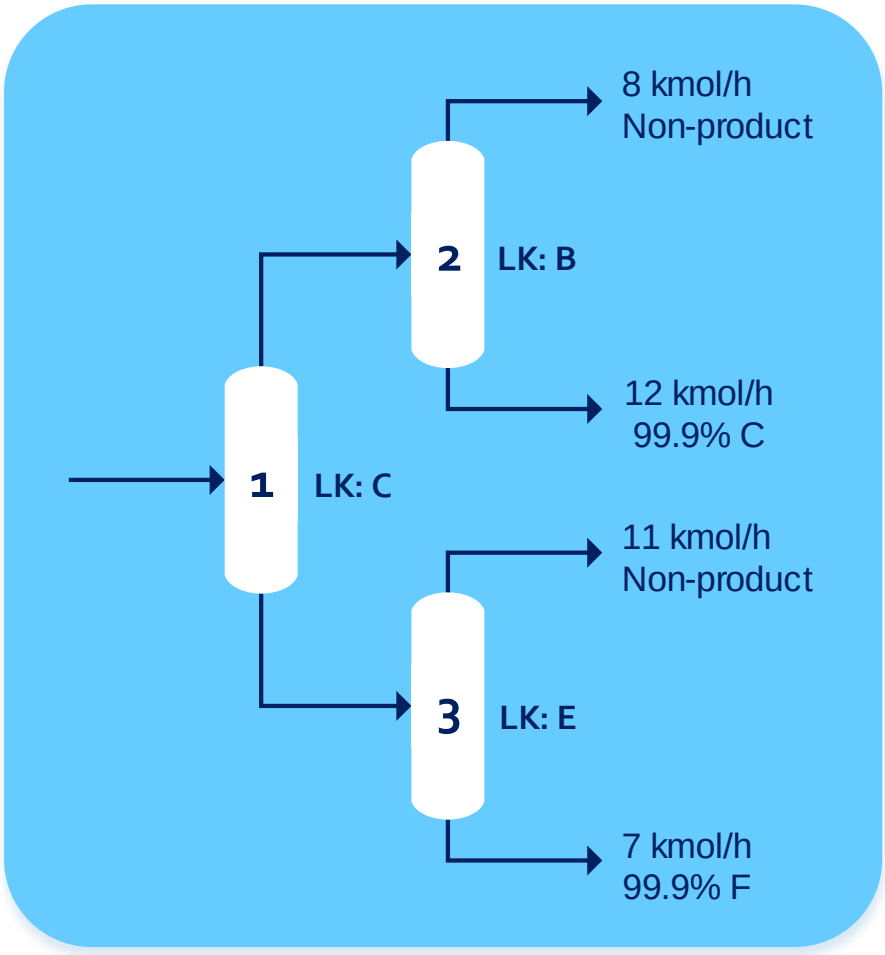

Figure 11: Example state BFD for DC train synthesis

Table 3: Example state for DC train synthesis

Number of possible streams to separate	Unit	4	5	6	7	N/A	N/A
Ethane	kmol/h	3	0	0	0	0	0
Propylene	kmol/h	5	Trace	0	0	0	0
Propane	kmol/h	Trace	12	Trace	0	0	0
1-Butene	kmol/h	0	0	2	0	0	0
n-Butane	kmol/h	0	0	9	Trace	0	0
n-Pentane	kmol/h	0	0	trace	7	0	0

Actions: The discrete actions were the selection of the outlet stream to separate from the current available outlet streams as well as the selection of the LK. The continuous action was the selection of the LK_{split}.

State transition: The state transition was the simulation of the currently specified DC whereby the dimensions and cost of the DC and the flowrates of the distillate and bottoms were calculated (as described in Section 3.3.2). The feed was then removed from the state (outlet-stream table), and the distillate and bottoms were added.

Reward: Two rewards were experimented with to test the effects of changing reward structures on agent performance, both of which were based on financial indicators calculated for each DC train developed. The reward was used to evaluate the financial viability of a given train and incentivise the agents. These rewards are both functions of the capital cost of the columns and sales value (revenue) associated with components in purified streams. Since correlations found in literature involving total annual cost (TAC) estimated the operational cost of DC trains as a function only of number of valuable products and flowrates in the feed stream to the train (using the distillation resistance method), it was assumed that TAC would remain approximately constant across various sequences and so wasn't considered in reward structuring (Lange, 2017). The alternative to this would be to calculate the column duties and estimate a utility cost. However, the TAC simplification allows for the environment to be made less complex, while maintain the general structure of the problem.

The first reward was based on expressions for the payback period of processing equipment and is shown in Equation (5). Reward one is the main metric for which the agent was evaluated. The second reward, which was based off annual profit is shown in Equation (6). The second reward is used to test the agent after the additions to the agent described in Section 3.3.6 are applied. Both reward values were constrained to a minimum value of -10, to prevent large negative rewards from destabilising the learning.

$$Reward1 = 10 - \frac{Capital\ Cost}{Revenue} \quad (5)$$

$$Reward2 = Revenue - \frac{Capital\ Cost}{10} \quad (6)$$

Episode end: The episode ends once 6 outlet streams have been produced. This was based on the assumption that because all of the components in the feed are valuable, the agent should separate each of them, resulting in 6 final streams. A further extension that was be experimented with was allowing the agent to select when to end the episode, through the addition of another discrete action. This can be seen as letting it choose when to finalise the design.

3.3.5 Baseline Asynchronous P-DQN Agent

The hybrid action space for the P-DQN agent is defined in Equation (7) as follows: The high level discrete actions, k, are selected from the discrete set [K]. The low-level continuous action, x_k, is selected from the continuous set X_k.

$$A = \{(k, x_k | x_k \in X_k, k \in K)\} \quad (7)$$

This hybrid action space presents a problem in framing these choices for a reinforcement learning agent as typical agents in RL literature are either continuous or discrete action oriented and not both. Some solutions to this include discretizing the continuous part of the action space, or converting the discrete actions into continuous actions (Xiong et al., 2018). Both of these have down sides. Discretizing the continuous action space loses information about the structure of the continuous space as well as requiring a large number of discrete actions for a reliable approximation which increases computational requirements. On the other hand, relaxing the discrete action space into a continuous set increases the complexity of the space.

Parameterized Deep Q-Networks (P-DQNs) present an efficient method of RL with a discrete-continuous hybrid action space (Xiong et al., 2018). The P-DQN architecture, shown in Figure 12 below, works through a combination of DQL and deep deterministic policy gradients (DDPG) (Mnih et al., 2015a). In P-DQN, the state is fed to a deep deterministic policy network (DDPN) which produces continuous actions which are then fed to the DQN which predicts Q-values for the possible discrete actions. The Q-values produced by the DQN also function as a critic for the DDPN – where the weights of the DDPN (θ) are updated in order to maximise the sum of the DQN values given the state and predicted continuous actions by the DDPN. This is shown by the loss function in Equation (9) below. The DQN weights (ω) are updated on how accurately it is able to predict the Q-values. The loss function for the DQN is shown in Equation (8). The weights of the DDPN and DQN can then be updated using the RMSprop optimizer (Tieleman and Hinton, 2012).

$$l_t^Q(\omega) = \frac{1}{2}\left[Q(s_t, k_t, x_{k_t}; \omega) - y_t\right]^2 \quad (8)$$

$$l_t^\Theta(\theta) = -\sum_{k=1}^{K} Q(s_t, k, x_k(s_t; \theta); \omega_t) \quad (9)$$

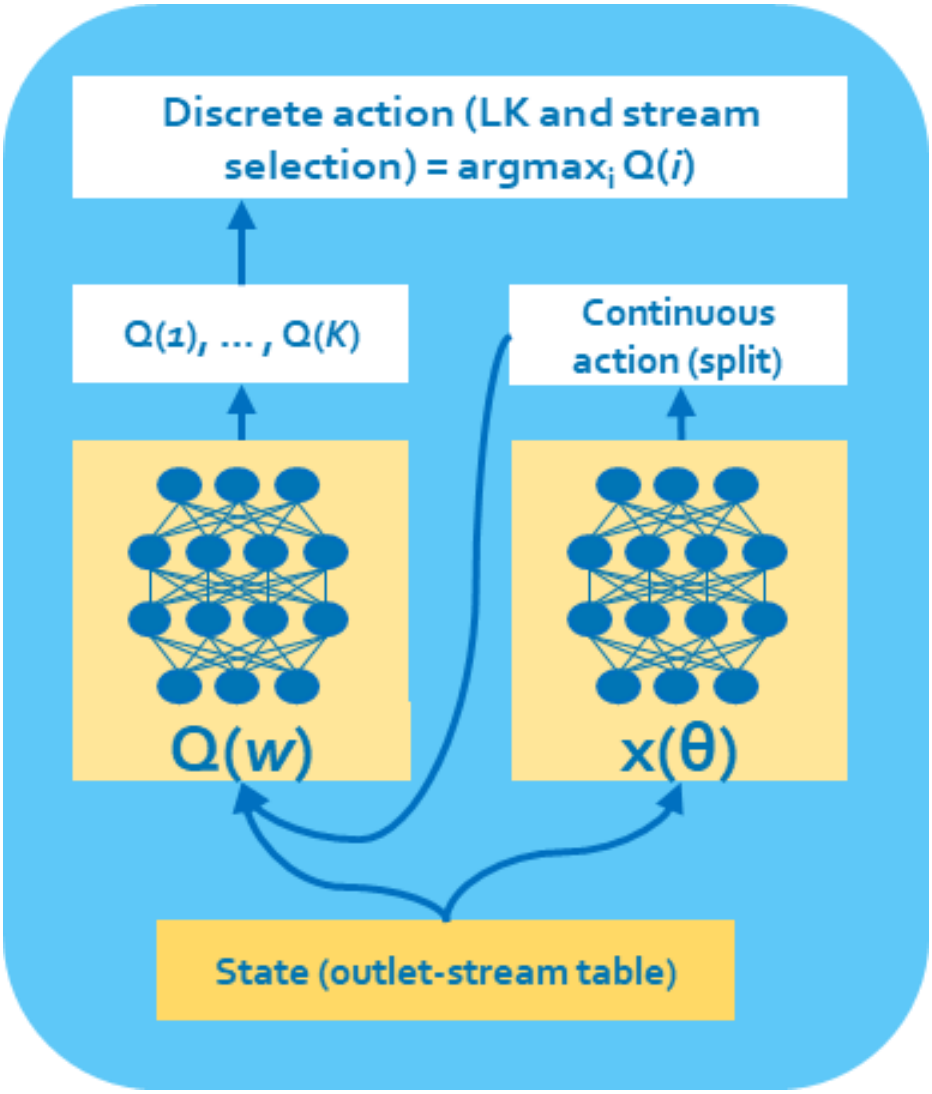

Figure 12: P-DQN Diagram Showing Overall Structure and Relation to State and Actions

For a given step there are “illegal actions” that the agent is prevented from being able to select. This includes the selection of an LK (or implied HK) for compounds with no flowrate in a given stream and the selection of placeholder streams which are yet to be filled with non-zero values.

Exploration is important for both the discrete and continuous actions as the agent has to simulate a large range of possible configurations in order to ensure that it does not

miss good possible configurations. Both the discrete and continuous action exploration is performed off-policy, meaning that it can be treated as independent to the learning.

Discrete action exploration is performed using epsilon greedy strategy (Sutton and Barto, 1998). This works through taking a random action with the probability ε or the current best action with probability (1 – ε). ε decays over the training period to allow for progressively more exploitation as the agent learns more.

Continuous action exploration is performed using the addition of noise generated by the Ornstein-Uhlenbeck process to the action predicted by the DDPN (Lillicrap et al., 2015). The Ornstein-Uhlenbeck process is temporally correlated noise that has previously been used in continuous action exploration policies in RL (Lillicrap et al., 2015)

To speed up the learning process an asynchronous version of the P-DQN can be used (Mnih et al., 2016). This means that instead of having a single agent interacting with the simulation, a master agent learns from multiple “worker” agents which interact with the simulation at once – this is represented in Figure 13 below. Each worker runs concurrently on a different thread, allowing all of the computer’s CPU processors to be utilised – speeding up the process roughly by the number of processes. The asynchronous version of P-DQN was used for all of the P-DQN problems as the resultant speedup is important for iterating through many possible versions of the problem.

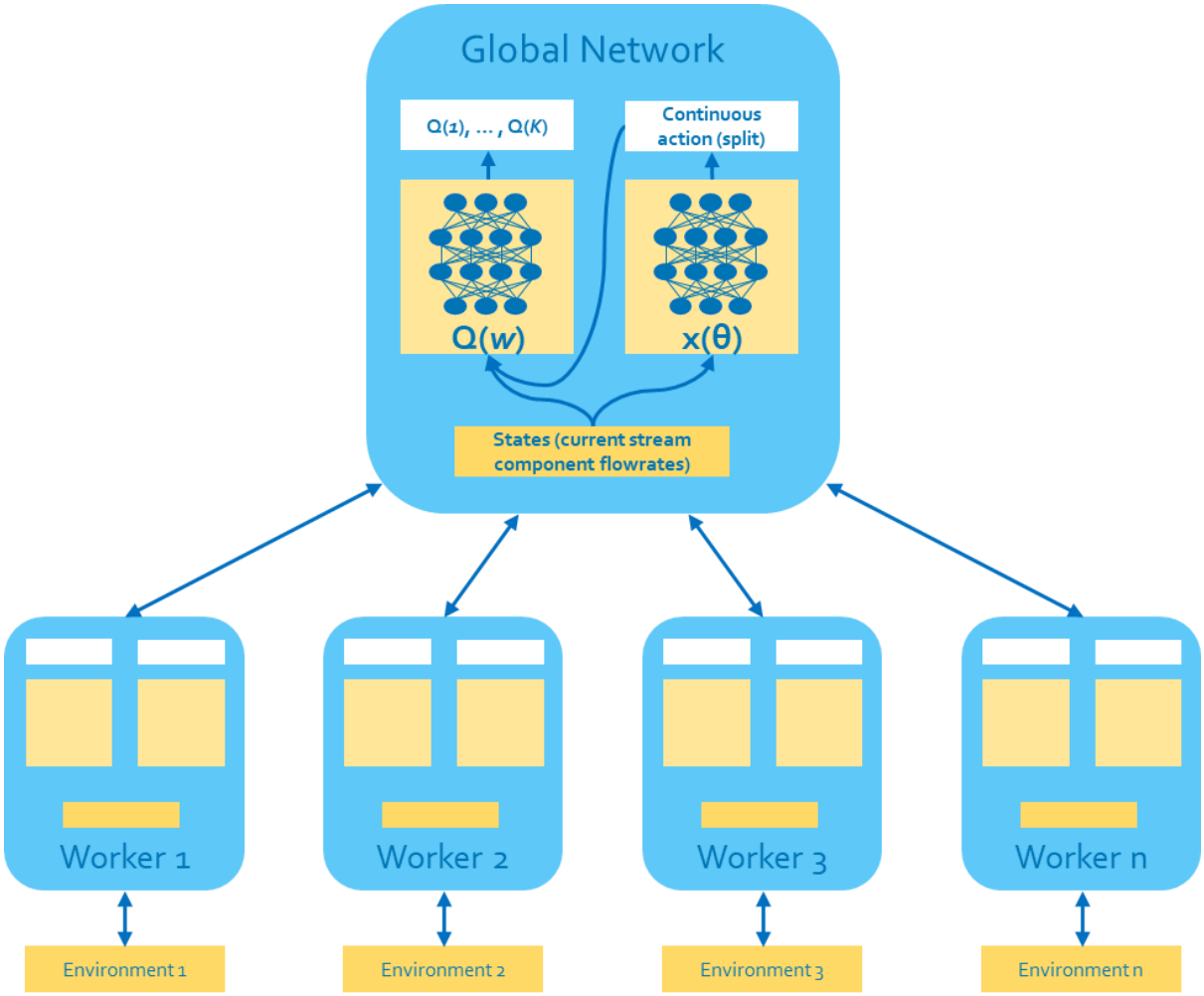

Figure 13: Asynchronous version of P-DQN agent showing multiple workers updating a global network

3.3.6 Experimental Agent Additions

To further improve the agent's performance the following further additions were experimented with:

1. **Addition of memory:** Experience replay reduces the correlations between experience (s, r, s', d tuples) (Mnih et al., 2013). This works through storing the agent's experiences over time in a memory buffer and then sampling batches of experience after each episode.
2. **Multiple explore strategies for different workers:** Giving different workers in the asynchronous P-DQN different explore strategies offers potential for more efficient exploration of the action space. This was done through changing the sample distribution from which e-greedy samples to include the following distributions: **(1)** a uniform distribution across legal actions, **(2)** a softmax distribution across the legal actions depending on the Q-values produced by the DQN and **(3)** a uniform distribution across the legal actions for the LKs that have not been previously selected within the episode. Explore strategy **(3)** was

to encourage the agent to try a sequence of different LK splits as this is a common feature of DC trains.

$$k_{explore} = \begin{cases} U(K_{new,LK}) & for\ workers\ 1-3 \\ U(K_{legal}) & for\ workers\ 3-5 \\ softmax(K_{legal}, & probability = Q)\ for\ workers\ 6-8 \end{cases} \quad (10)$$

3. **Shaped reward:** Instead of giving the reward at the end of the episode, intermediate rewards can be included to help the agent learn which intermediate actions are good. To achieve this, the sparse reward equations described in Section 3.3.4 were calculated during each step of the environment. A reward equal to the increase in the equivalent sparse reward metric over the most recent step were then assigned as the shaped reward.
4. **Selecting when to end episode:** Instead of the episode ending only after the agent has produced 6 outlet streams (corresponding to 5 DCs), an additional action was added which allows the agent to choose to end the episode. This allowed the agent to choose to design a DC train with less columns. This has the potential to be useful if a certain number of columns results in an increased revenue that does not outweigh the corresponding cost. This was also a useful extension for more complex and open-ended process synthesis problems where the end of an episode may not be easy to define.
5. **Fine-tuning the agent:** Experimenting with: different dimensions of the NN, increasing the changing the training time and adding learning rate decay.
6. **Freezing the DDPN:** As described in section 3.3.5, during training the DQN is required to have a lower learning rate than the DDPN to ensure stability of the DDPNs learning. The following alternative learning scheme is tested to see if it could result in better performance of the DQN: An additional training period, where the DDPN is frozen (network parameters set constant with no learning) was added after the initial period. This allowed for only the DQN to be trained – potentially increasing the performance of the DQN.
7. **Adding a duelling layer for DQN:** A duelling layer uses 2 additional estimators within the architecture of the agent, which are combined before the Q-value outputs (Wang et al., 2015). These estimators learn the state value function and advantage function and are combined using Equation (11) to produce Q-values. The duelling layer has been shown to increase agent performance, specifically when DQN must produce many actions. It was therefore added to test to investigate whether it improved agent performance.

$$Q(s,a,\theta,\alpha,\beta) = V(s,\theta,\beta) + \left(A(s,a,\theta,\alpha) - \frac{1}{A}\sum_{a'} A(s,a',\theta,\alpha) \right) \quad (11)$$

Where β is the state value estimator NN parameters and α is the advantage estimator NN parameters.

4 Results and Discussion:

4.1 Case Study 1: Reactor Sequencing

The DQN agent was trained with a learning rate of 0.001 and used an epsilon greedy explore strategy, where epsilon was linearly annelid from a value of 1 to a value of 0.05. After a training period of 15000 episodes, which took less than 4 minutes[4], the DQN agent was able to learn the optimal sequence of reactors (PFR, PFR, CSTR, CSTR). The increasing average reward in Figure 14, shows the improvement of the agent over the training period. Figure 15 shows a Livenspiel plot which depicts how the agent has correctly selected a PFR when the system's reaction curve is sloping upwards and a CSTR when the reaction curve is sloping downwards.

An animation which shows the generation of the BFD, and the corresponding Livenspiel plot as the agent makes decision was created and can be found on the following link: www.github/lollcat/RL-Process-Design.[5] This animation shows the agent exploring possible configurations early on in the learning period, including many clearly suboptimal decisions but with configurations improving over time as the agent learns.

Although this problem is simple by design – the results of the reactor demonstrating learning and finding the optimal solution are significant. Because RL has not yet been applied to process synthesis, showing that chemical engineering process synthesis problems can be phrased as a MDP and solved using reinforcement learning is a decisive proof of concept with implications for future research. It is important to note that this problem is not best solved using reinforcement learning techniques – and merely functions as a simple toy-problem.

This easy to understand example and solution, as well as the simple RL algorithm coded, which relies on the high level Keras-RL library makes this problem accessible to chemical engineers without advanced knowledge of RL. The code and associated guide which were developed as part of this thesis were uploaded to Github to provide an opensource resource for future development by others.

[4] All computation in this thesis was performed on an intel i7-6700HQ CPU
[5] All code for both case studies was uploaded to this Github repository

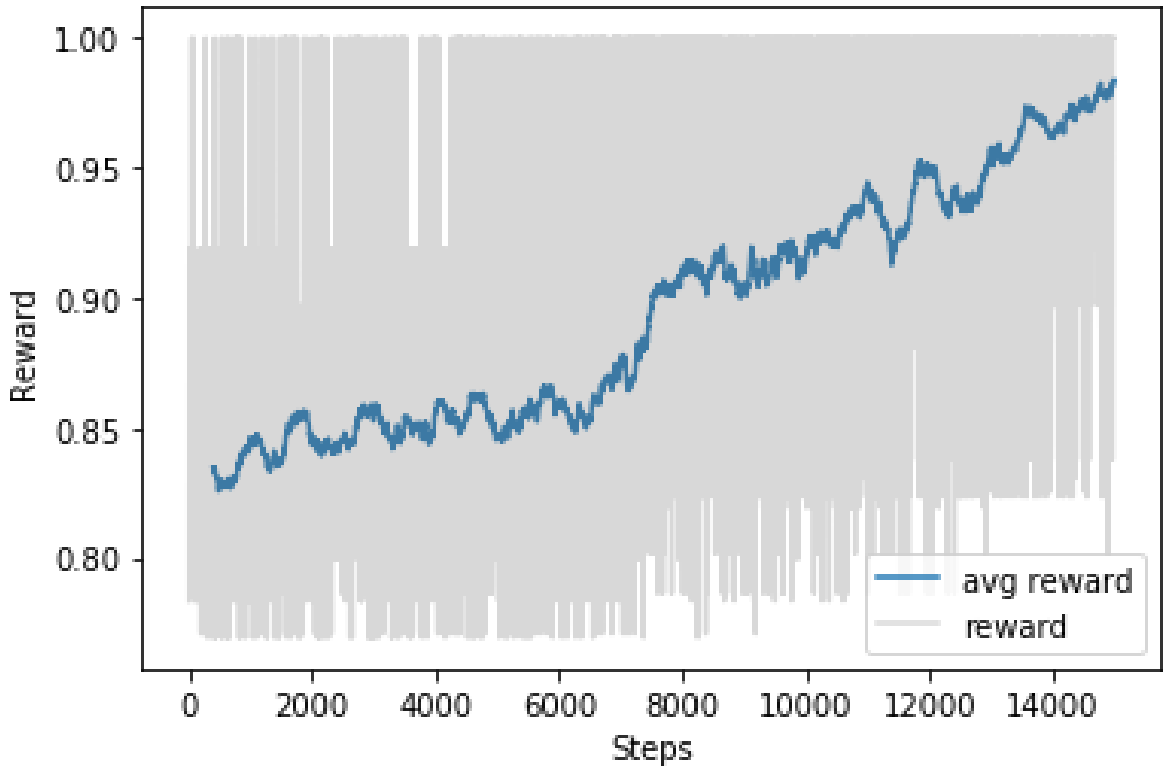

Figure 14: DQN agent learning on reactor sequencing problem

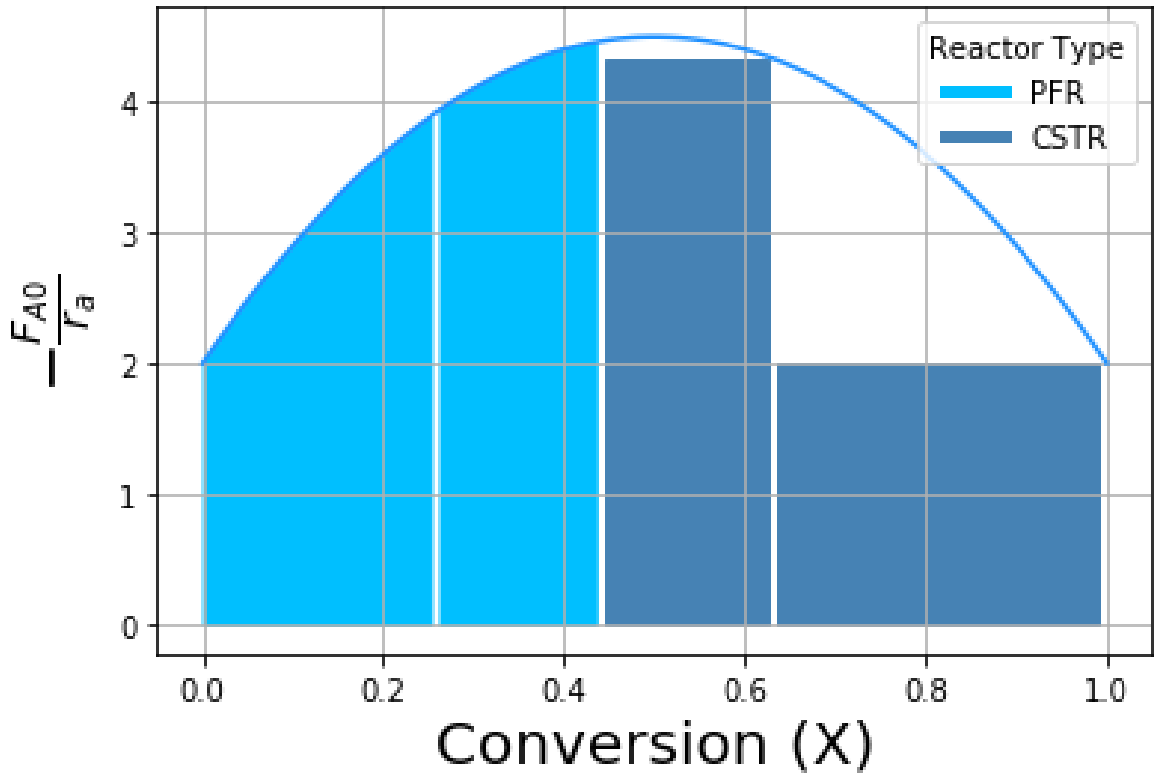

Figure 15: Livenspiel plot of agent producing the optimal solution after training

4.2 Case Study 2: Distillation Column Sequencing

4.2.1 Baseline Asynchronous P-DQN Agent

The training of the asynchronous P-DQN (with 8 workers) for 2000 episodes took under 10 minutes. The DDPN used a learning rate of 0.001, while the DQN used a learning rate of 0.0001. The DDPN contains 2 fully-connected layers of 100 and 50 nodes respectively, while the DQN is made up of 3 fully-connected layers of 100, 50 and 50 nodes respectively. These learning rates and NN shapes were used for the rest of the agents within this case study.

As shown by the increasing average reward over time in Figure 16 below, the P-DQN agent was able to learn within the environment - showing significant improvement over time.[6] The trained agent produced the DC train sequence shown in Figure 17, corresponding to a reward 1 of 9.65, outperforming all of the heuristic designs. The economic evaluation of the baseline agent's DC train is shown in Table 4 below. The agent showed some instability in the learning process, where it did not always beat heuristic designs. This instability is partially visible in the fluctuations of average reward in Figure 16.

The trained agent selected an LK_{split} of 99.9% for each of the DCs. This was common throughout the various examples in Case Study 2. This is because a high LK_{slit} allows for high recovery and purity of product. However, these solutions were not desired as they imply that the selection of the continuous action is too simple – with the correct answer almost always at the upper boundary of the continuous actions space. This is further discussed in Section 4.2.5. Unless specified – all LK_{splits} in the DC trains for the rest of the discussion are 99.9%.

The trained agent's solution shows an unexpected strategy where 4 sellable product streams are produced (instead of 6). The agent selects to repeat the selection of methane as the LK twice and never selects propylene as an LK. This results in the agent achieving a lower total revenue (R75 million) than the heuristic design cases (R109 million). However, this also results in the reduction of capital cost to R26 million – less than half that of any of the heuristic design's capital cost. This is because the propylene-propane split is expensive: propylene-propane has the lowest adjacent relative volatility. So, by avoiding the propylene-propane split, the agent is able to significantly reduce its capital cost to achieve a higher reward 1 than heuristic designs. It is important to note that the agent's goal/evaluation did not include a requirement to separate all of the compounds into separate streams (as is commonly expected within DC train synthesis problems). A version of the problem where the agent is required to separate out all of the compounds in shown in Section 4.2.3.

[6] All figures within the results section focus on the key relevant information for the discussion. The diagrams within Appendix B give further information (e.g. number of trays, outlet flowrates) for all of the generated DC train configurations.

The baseline P-DQN agent demonstrates that process synthesis problems with both discrete and continuous choices can be framed within a MDP environment in which a RL agent can successfully learn.

One key question in response to the resultant sequence produced by the standard agent would be: Given that the agent is not selecting to separate all 6 compounds – why does it need to use 5 distillation columns? The answer is, because the environment is designed to only end an episode once 5 outlet streams have been produced. This was due to the assumption that the agent would separate all of the compounds. The *allow submit* extension to the agent where the agent can choose to end an episode deals with this problem and is discussed in 4.2.2.3 below.

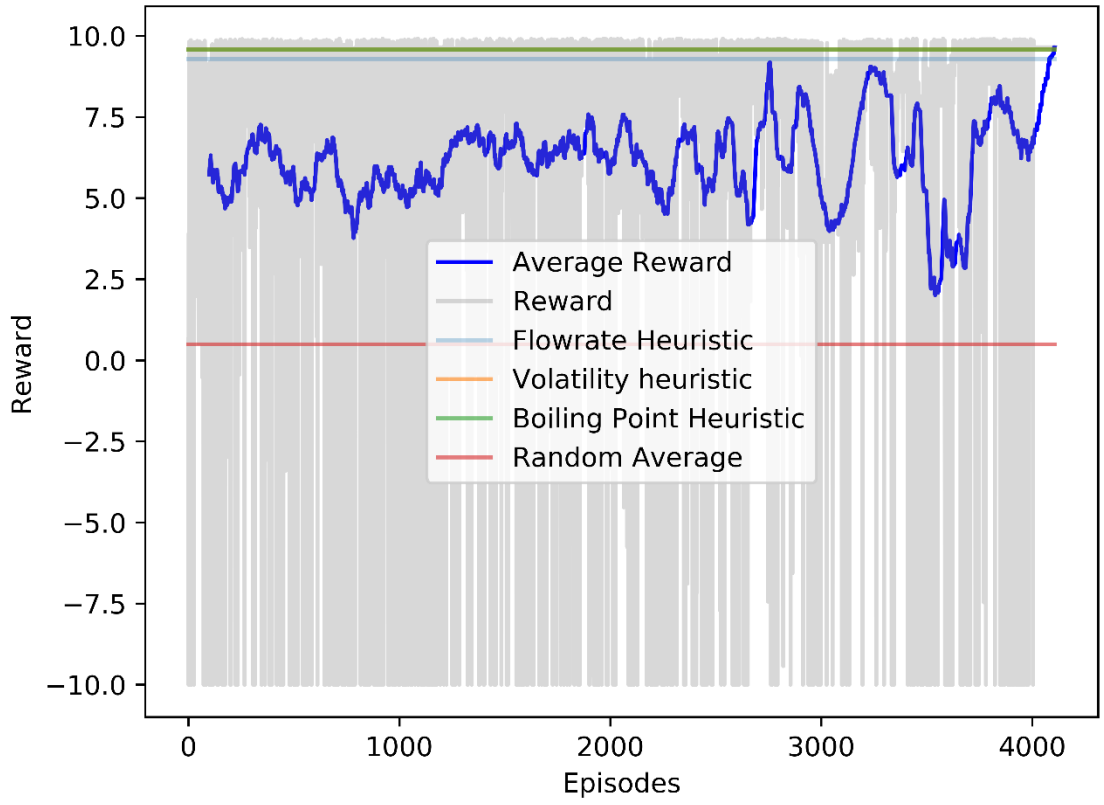

Figure 16: Asynchronous P-DQN agent learning on Distillation Sequencing Problem with Performance metric 1 [7] [8]

[7] Note that the volatility and direct sequencing heuristic lines are close in terms of both rewards so appear at the same place on the plot

[8] For all learning plots, a short additional period of 100 episodes was added to the end whereby the agent takes the action it considers to be best, to allow for the end of the graph to show the trained agent's performance

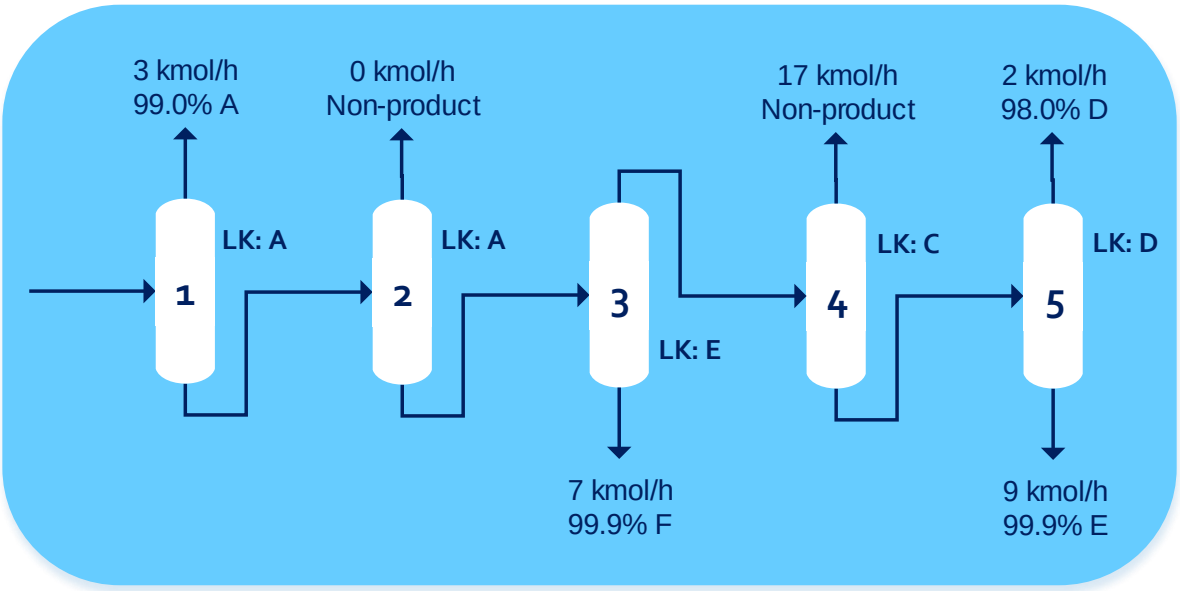

Figure 17: Trained Baseline Asynchronous P-DQN agent's Final Sequencing Solution

Table 4: DC train baseline agent economic summary

	Cost (R million)	Outlet stream Revenue (R million)
Column 1	2	1
Column 2	0.4	Non-product
Column 3	2	39
Column 4	2	Non-product
Column 5	20	18 (tops)
		16 (bottoms)
Total Capital Cost (R million)		**Total Revenue (R million)**
26		75

4.2.2 Agent Improvements

Improvements to the agent, as discussed in Section 3.3.6, were experimented with. Not all of these additions resulted in improvements in the learning. However, the addition of the training period with the DDPN weights frozen (*DDPN freeze)*, the *duelling layer*, using multiple explore strategies for different workers (*multiple explore strategies)* and letting the agent choose to end its own episode (*allow submit)* did show significant improvements. These are discussed below.

4.2.2.1 DDPN Freeze

The additional training period with the DDPN weights set constant (*DDPN freeze*) exhibited more stable learning after the freeze. Unlike the first training period in which both the continuous and discrete actions have a random component to allow for exploration, in the second training period the DDPN is simply producing the actions

which it considers to be best. This allows for more stable exploration of the discrete action space – improving the final performance of the DQN. In Figure 18 this is shown by the steady increase (less fluctuation) in the average reward after the freeze-point. The dip in the average reward after the freeze is because the agent restarted its epsilon-greedy explore strategy of the discrete action space. Furthermore, as the discrete action space requires more exploration than the continuous action space, the second learning period gives the potential for the agent to focus on discrete action space exploration.

After training, the agent created the DC train shown in Appendix B1.1 Figure 30, achieving a reward of 9.73, outperforming all of the heuristic designs. Although the second training period required additional runtime, the algorithm requires less computation per step after the freeze as only the DQN's weights are updated. Only an additional 6 minutes of training are required for another 4000 episodes with *DDPN freeze*.

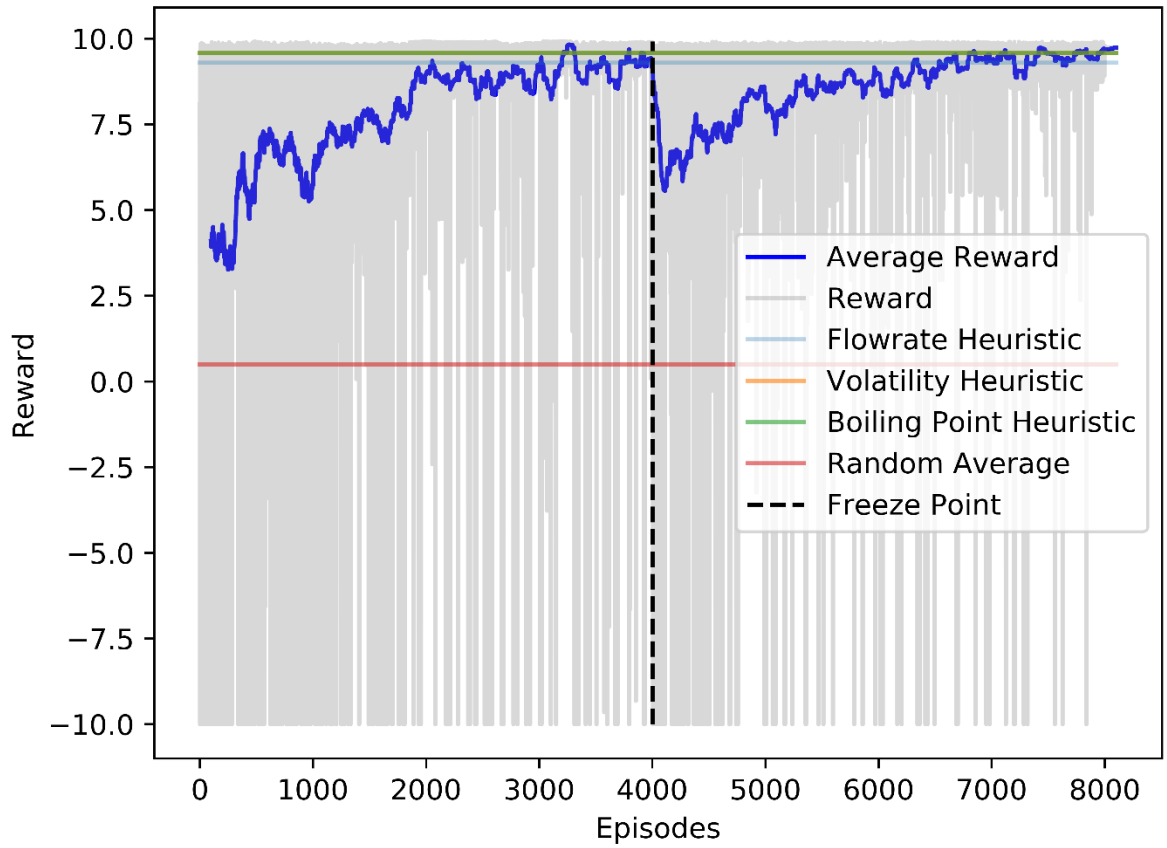

Figure 18: Asynchronous P-DQN agent learning on Distillation Sequencing Problem with Performance metric 1

4.2.2.2 Duelling Layer

The duelling layer successfully improved the agent's performance in terms of speed, consistency and the final solution. This is exemplified in in Figure 19 below where the agent was able to outperform heuristics on reward 1 in half the training time (2000

episodes). In the below example, the trained agent achieved a reward of 9.87, using the sequence shown in Appendix B, Figure 35.

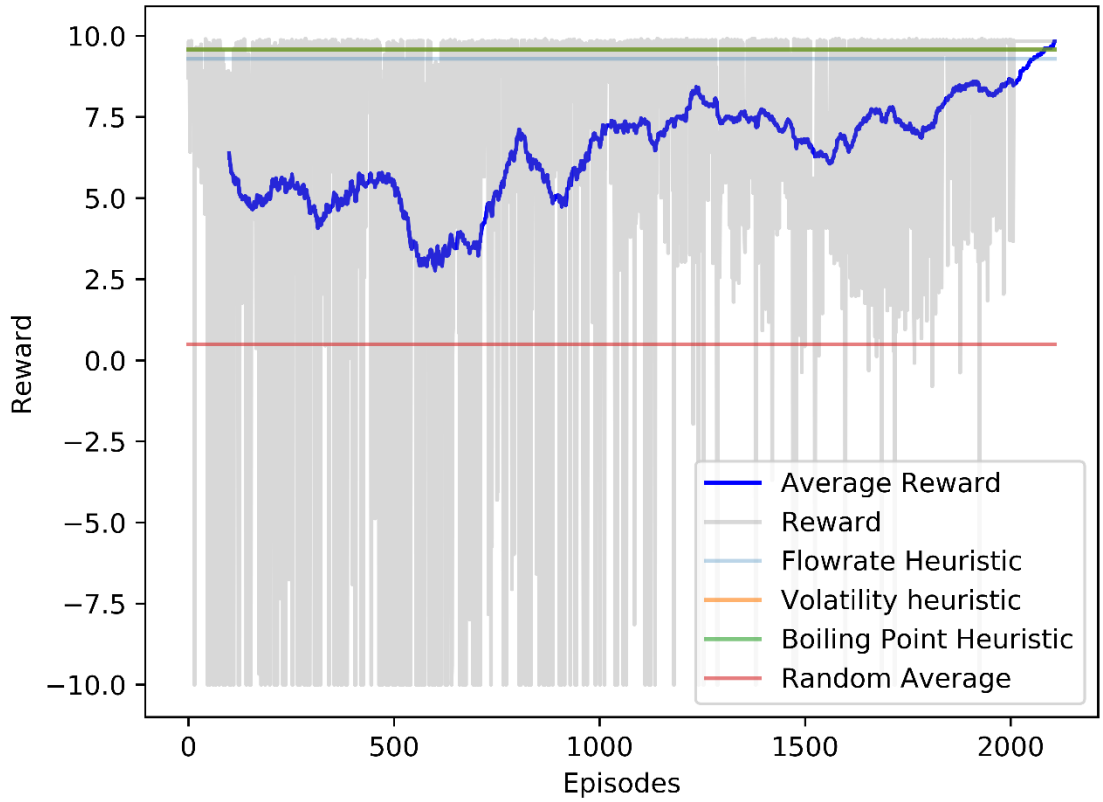

Figure 19: P-DQN Agent with duelling layer for half the training period

4.2.2.3 Allow Submit and Multiple Explore Strategies

The baseline agent did not obtain good performance with the addition of allowing the agent to end its own episode (*allow submit*). However, when the additional *allow submit* action was added to the agent using the *DDPN freeze*, a *duelling layer* and *multiple explore strategies* – the agent was able to outperform all of the previous agents (and all of the heuristics) to achieve a reward 1 of 9.93. The *multiple explore strategies* were of critical importance in getting the agent experience a wider variety of possible sequences – in order to explore the more complex *allow submit* discrete action space. The agent uses the novel strategy of only using one DC to separate out n-pentane, as shown in Figure 20 below. This corresponded to a revenue of R 39 million with a capital cost of only R 3 million – which creates an extremely high reward1 (see Equation (5)). The agent's training is shown in Figure 21 below.

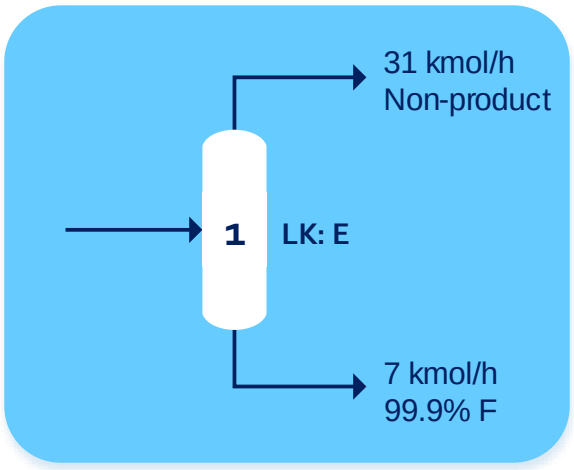

Figure 20: DC train configuration of P-DQN Agent with duelling layer, freeze, multiple exploration and allow submit

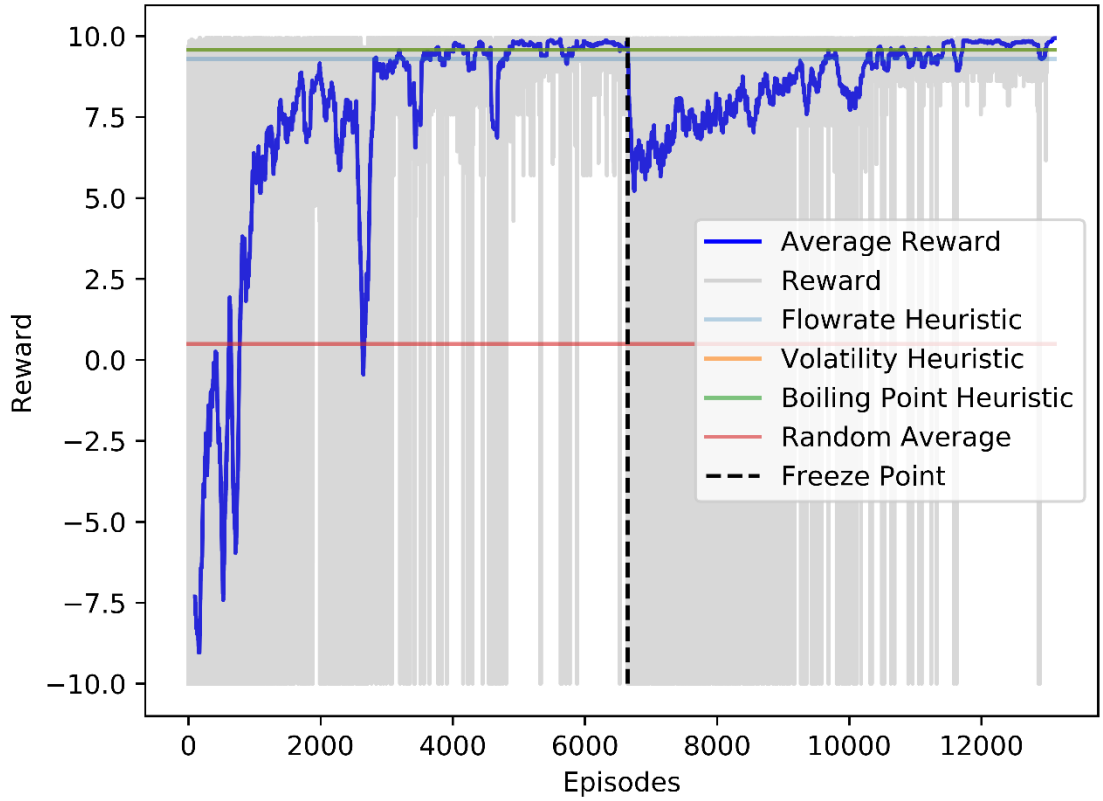

Figure 21: P-DQN Agent training with duelling layer, freeze, multiple exploration and allow submit

4.2.3 Illustrative Extensions for 6 Product Streams

An expected critique of the improved agent's novel solutions to the problem above (especially the improved agent's solution in Section 4.2.2.3) is that they are not satisfying the expectation of 6 product streams. The reason that it not satisfying this constraint is that this specification was not included within the problem definition. This was because the goal of the project was to give the agent an open-ended problem.

The agent is therefore unaware of the 6-product stream expectation and successfully seeks to maximise its reward by producing less product. However, this additional constraint can be easily added to the agent's choices. Furthermore, it simplifies the problem for the agent as the constraints to the actions mean that exploration of the action space is simpler. In response to the expected critique, the 6-product stream constraint was added to the problem definition and solved as shown below. In the agent training plot (Figure 22), the simpler action space is visible by the lower variance in the agent's rewards (the grey line). The configuration of the DC train is shown in Figure 23 below. Interestingly, the agent selects the same sequence of unit separations as the separation by boiling point heuristic – which was the best performing heuristic in terms of reward 1.

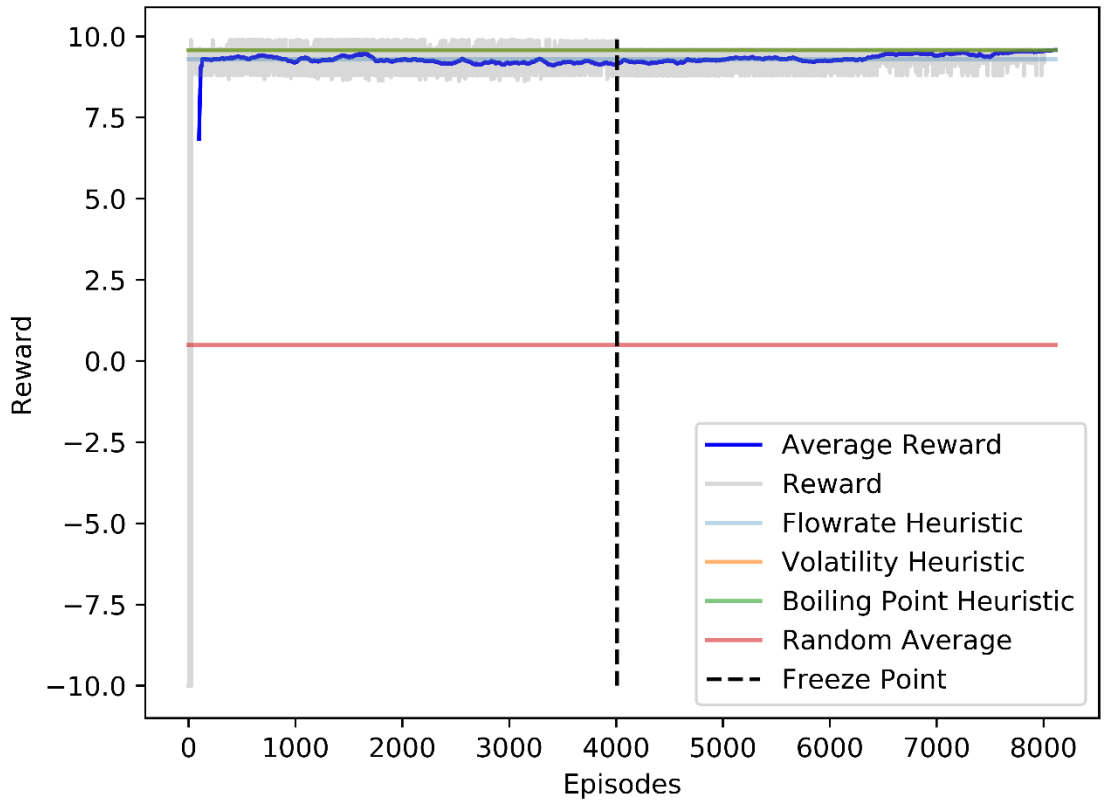

Figure 22: Training with agent constrained to separate all compounds

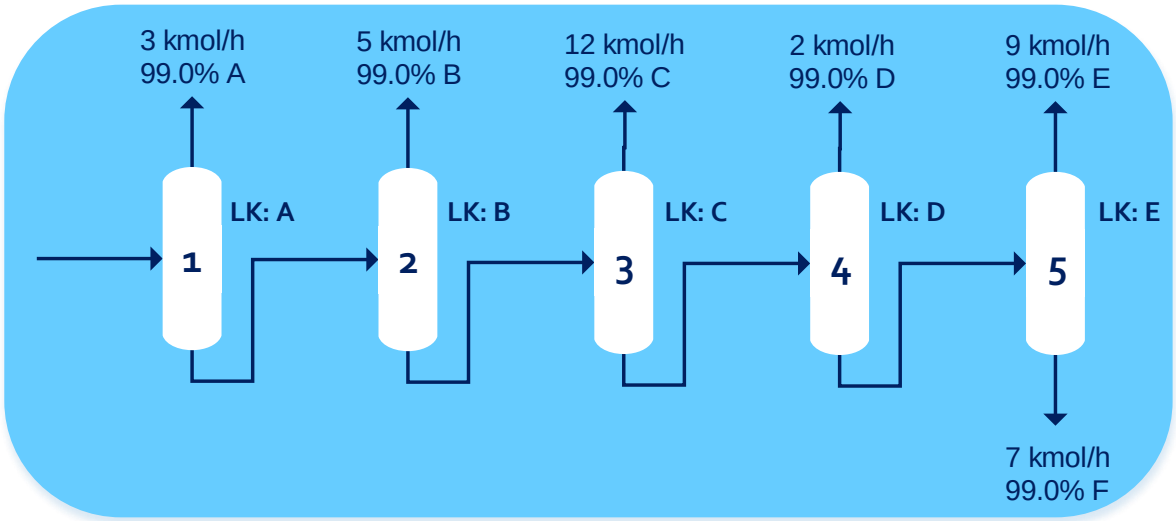

Figure 23: DC train configuration with agent constrained to separate all compounds.

4.2.4 Reward 2

4.2.4.1 Baseline Agent

The baseline agent showed a significantly decreased performance in training with reward 2 and was not able to outperform heuristics. However, the agent did demonstrate improvement over the training period, as shown by the increasing average reward in Figure 24 below. The agent achieved a final reward 2 of 7.9 using the DC train shown in Figure 25. Unlike other DC trains, the agent selected varying LK_{splits} (not always 99.9%) – this is shown in Appendix B2 Figure 37.

The reason for the moderate performance with reward 2 was due to inadequate exploration of the DC train configurations, specifically within the discrete action space. In Figure 24 this shown by the lack of overlap between the rewards (grey lines) of the agent over the training period, and the heuristic benchmark lines. The agent trained with the reward 2 exhibited maintained a good performance in terms of reward 1, achieving a reward 1 of 9.13 which was only slightly below the heuristic performance. The visa versa was not true, and sequences generated using reward 1, typically did not achieve a high reward 2.

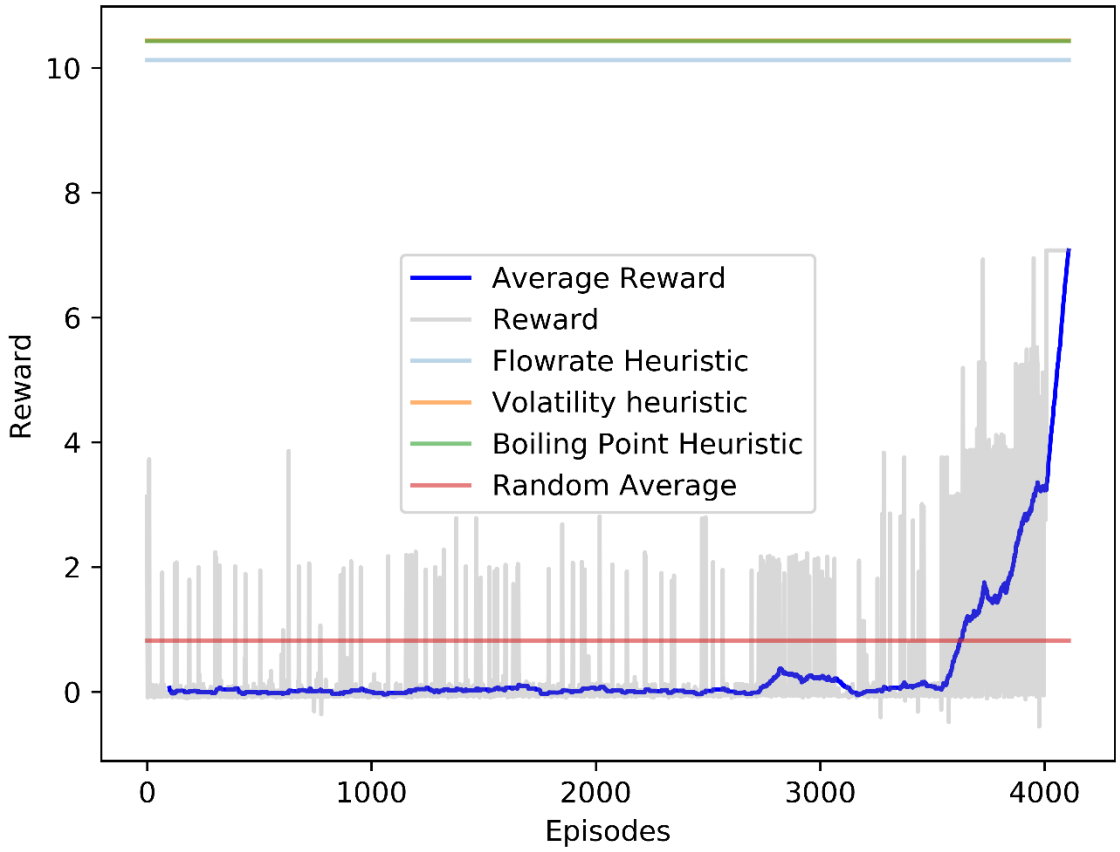

Figure 24: Baseline Agent's training on reward 2

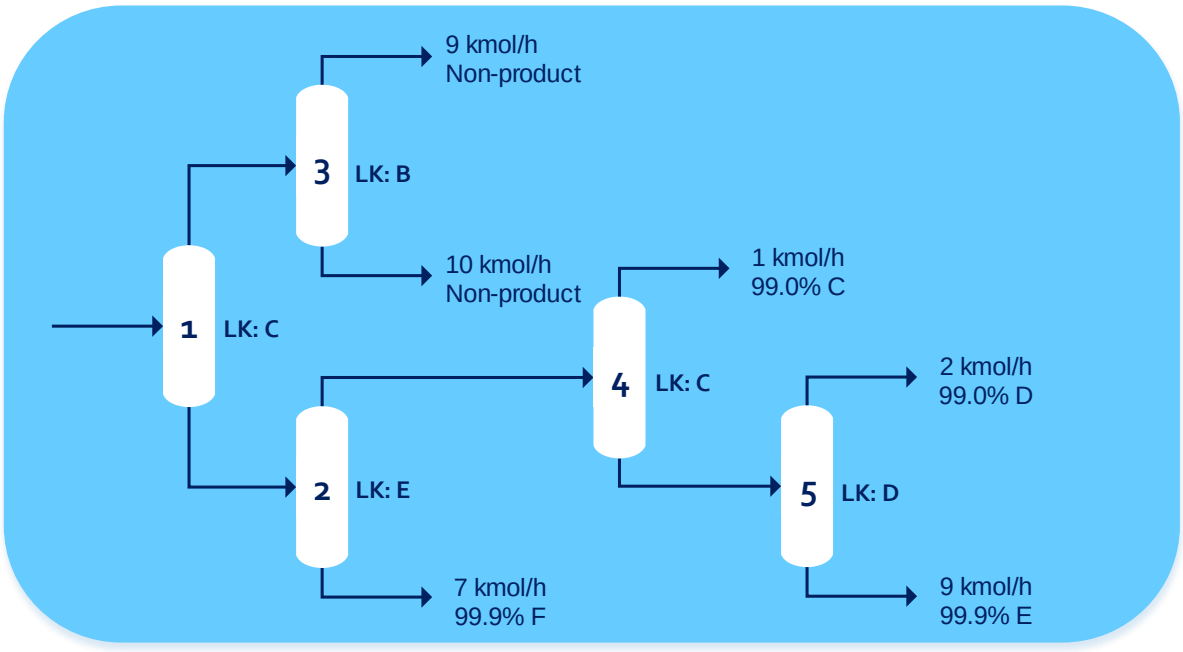

Figure 25: Baseline Agent's DC configuration after training with reward 2

4.2.4.2 Improved Agent

As in Section 4.2.2.3 the potential improvements described in Section 3.3.6 were experimented with. The following combination of improvements were applied: *duelling*

layer, freeze and *multiple explore strategies.* Furthermore, it was noted that adding the constraint of separating all of the compounds improved the agent's performance with respect to reward 2, so this was also included. The trained agent was able to outperform all of the heuristics, narrowly beating the boiling point separation heuristic to achieve a reward of 10.44. Figure 26 shows the agent's training and Figure 27 shows the DC train configuration designed by the trained agent. The agent produces 6 product streams, resulting in the same revenue as the heuristic design cases (R109 million). However, the agent is able to slightly lower capital costs of R0.2 million less than the lowest capital cost of the heuristics. Furthermore, although the agent was now no longer trained on reward 1, it also outperformed all of the heuristic designs with respect to reward 1, achieving a reward 1 of 9.58.

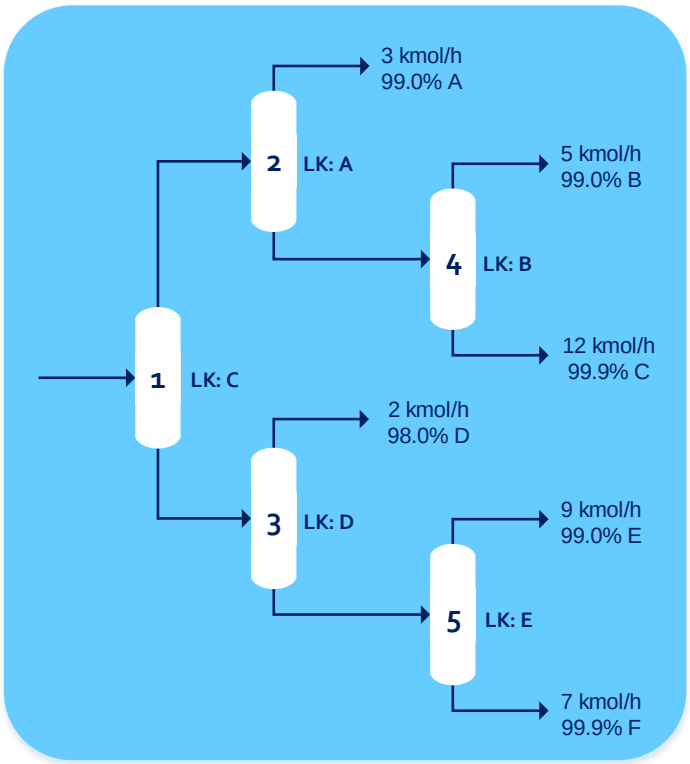

Figure 26: Improved agent's DC train configuration after training with reward 2

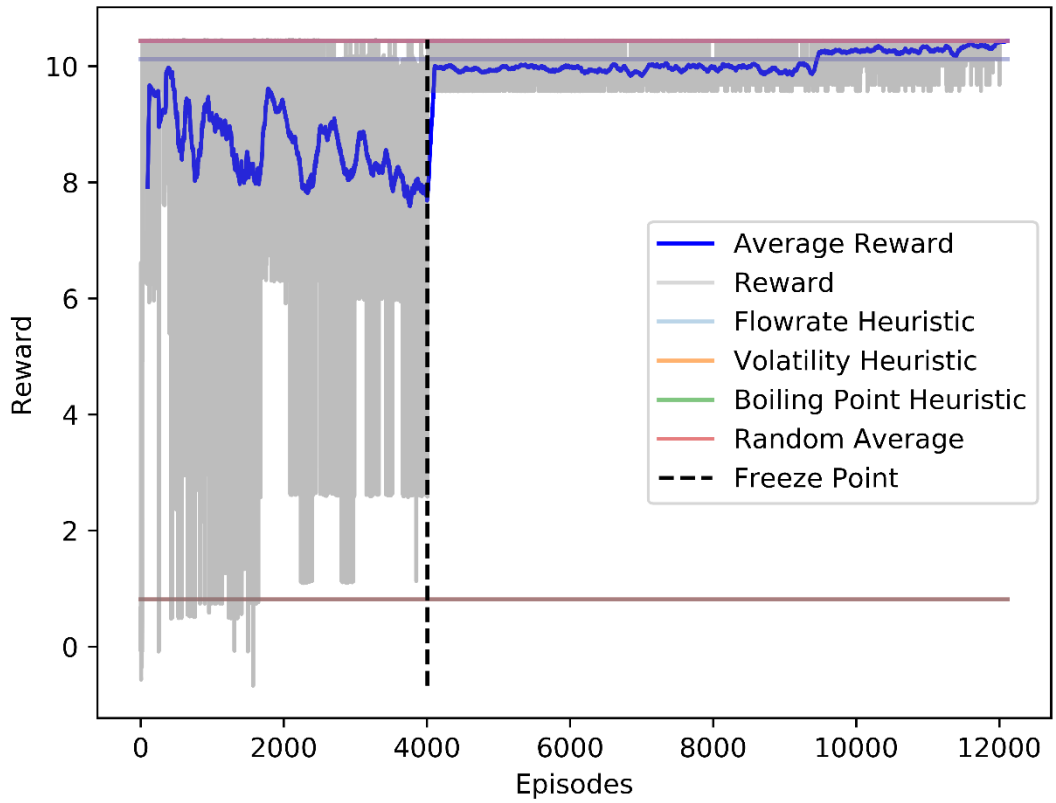

Figure 27: Improved agent's training on reward 2

4.2.5 Critique and Potential Improvements

The following critiques and potential improvements to the DC train synthesis problem are given below:

1. **Lack of difficulty within the continuous action selection:**

Although the agent was able to demonstrate learning within the continuous action space, the resultant answers (selecting a LK_{split} of 99.9% each time) are relatively simple.

To better demonstrate the P-DQN agent's learning/decision making capabilities – the environment/problem definition could be changed to give the agent a harder trade-off on the selection of the LK_{split} such that the best LK_{split} was not at the edge of the action space and that the agent is forced to select differently depending on the context of the environment's state. One example of a change to the problem that may achieve this is to superficially increase the height related cost to the DC, which would discourage a high LK_{split}. It is important to note that this critique is of the problem and the agent's environment and not the agent itself.

2. **Giving the agent more information about the environment:**

Instead of feeding the agent a state containing only the outlet stream table, more information from the environment could be given to the agent though the addition of feature representation to the state. For example, the selling price of the current outlet

streams in the outlet stream table could help the agent make more informed decisions about which compounds to produce.

3. Increasing consistency in performance:

The agents sometime experienced instability in the learning process and were not able to always learn the best sequence every run. This is not necessarily a problem as multiple agents can be run – while the best agents are selected. However, generally improving the stability and consistency of the learning process would be a useful further extension of this thesis.

4. Further experimentation with reward structure

The attempted shaped reward did not successfully improve the agent. More experimentation with nested rewards could help allow the agent to better learn about the effects of intermediate decisions.

5. **Changing the network architecture:**

To allow the initial layers of the DQN to represent the state better structurally, the DQN architecture could be changed. We propose the DQN structure shown in Figure 28 below to achieve this. Within this alternate architecture, before entering the 3 fully connected DQN layers, each stream in the state (representing the outlet-stream table) is connected to 2 fully connected sub-layers before the outputs of these sub-layers are flattened and concatenated. Previously the state was simply flattened before entering this agent's NN – this loses the structure of the stream table. The aim of this is to allow each of the sub-layers to learn a representation of a stream. An extension of this thesis would be to test out alternative DQN structures instead of the structurally simple DQN of 3 fully connected layers that was used.

Another addition would be to repeatedly randomly shuffle the stream location within the state (and corresponding Q value output) such that the order to which streams exist in the stream table is no longer a factor. The shuffle would have to be done such that the DQN's output Q-value to state relation is kept intact (e.g. if stream "1" is shuffled to the end of the outlet-stream table, then the end Q-values should still correspond to the original stream). In the state used in Case Study 2, the streams towards the end of the outlet-stream table are generally filled with placeholder 0 values (as they only become filled with values towards the end of an episode when many outlet streams exist). The streams towards the beginning of the outlet-stream table are generally filled with actual values from stream flows. Shuffling the streams would remove this correlation – it would allow for all streams (both actual flows and placeholder streams) to have an equal likelihood of being in any position within the state/outlet-stream table. This change in combination with the proposed architecture change may allow the agent to better learn and act on the information given by the compound stream flowrates.

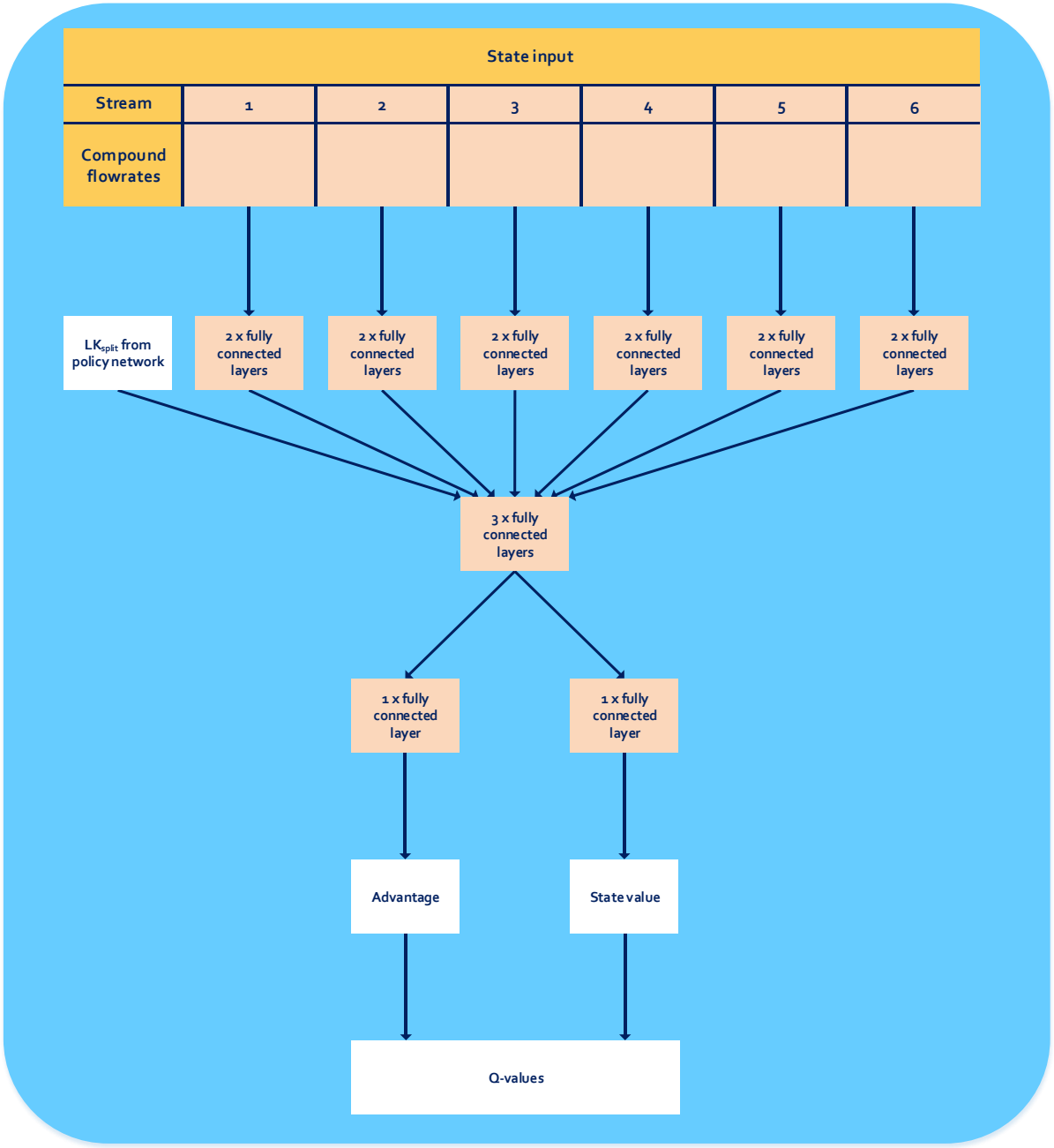

Figure 28: Alternative Agent NN architecture aimed at better representing the structure of the steam state

6. **<u>Unrealistic number of trays in environment:</u>**

Many of the DCs had an unrealistic number of trays (<100). This was because the environment relied on only on simple general equations like the Fenske equation while additional constraints to the DC's dimensions were not included. This could be solved through adding constraints to the environment. However, as the focus of this thesis is a proof of concept on toy-problems – this is not considered to be a major issue.

4.3 Comparison of RL to other Process Synthesis Techniques

Current process synthesis is performed through a combination of engineering judgement and computational methods. As describe in Section 2.6 RL, has the advantage over conventional computational process synthesis techniques because:

1. It can be used on more open-ended problems closer to complete process synthesis as performed by human chemical engineers.
2. Deep RL agents are well suited to interaction with a simulation (e.g. Aspen) as they have been largely developed through application to computer games (Aspen can be thought of as a computer game for chemical engineers). This allows for improvements alongside the development of better chemical and physical process simulators.
3. Deep RL is well suited to generalisation – allowing for easy application to synthesis of multiple different types of chemical engineering designs

RL based process synthesis has the advantage over humans of being able to more rapidly explore many possible process configurations. Human design through chemical engineering knowledge, judgement and heuristic design can be used for designing a process, however, as the problem becomes more complex humans become less and less likely to generate a top performing design. Human and heuristic DC train design often assume that all compounds in the starting stream are worth separating to produce pure compound products. Under this assumption, the number of possible sequences for an increasing number of compounds is shown in the Table 5 below. However, if this assumption is relaxed – and any number of possible pure compounds can be produced, the number of possible sequences increases even further. Case study 2 in this report let the agent choose whether or not to separate out all the compounds – and in doing this the agent was able to make the trade-off of whether some compounds are not worth producing. Humans/heuristic design would find it exceedingly difficult to make these trade-offs.

Table 5: Number of possible different distillation train sequences possible (ordinary distillation)

Number of Compounds	Number of separators in train	Number of different sequences possible separating all compounds (Seider et al., 2009)
2	1	1
3	2	2
4	3	5
5	4	14
6	5	42
7	6	132
8	7	429
9	8	1430

4.4 Future developments

4.4.1 Framing Human Level Process Synthesis as an MDP

We propose that the actions that a human chemical engineer takes when performing complete process synthesis could be phrased in terms of the action space shown in Figure 29 for an RL agent. This action space is significantly larger than that of the problems presented in this thesis and would therefore present a far more difficult RL problem with a significant computational requirement. Efficient exploration would present a key problem within complete process synthesis problems as the space of possible configurations is extremely large.

This action space has a hierarchical structure. The lower level actions depend on which high level action was selected. For example, if the high-level “add unit” action was selected, the mid-level choices would be the selection of the stream to add the unit to and which type of unit, while the lowest level would include the selection of the unit specifications/operating conditions.

Each of the Case Studies can be represented as existing is sub-sections of the action space in Figure 29 as follows: Both Case Studies assumed a given starting stream, with only the “add unit” high level action included. Case Study 1 only included the low-level action of selecting a unit type (PFR or CSTR) while the unit specifications were assumed and the stream was the outlet stream of the previous reactor. Case Study 2 included the selection of the stream from previous DC outlet streams to add the unit to and the specification of the unit (LK and LK_{split}). Thus, together Case Study 1 and 2 include all 3 major types of lower level actions within the high level “add unit” action.

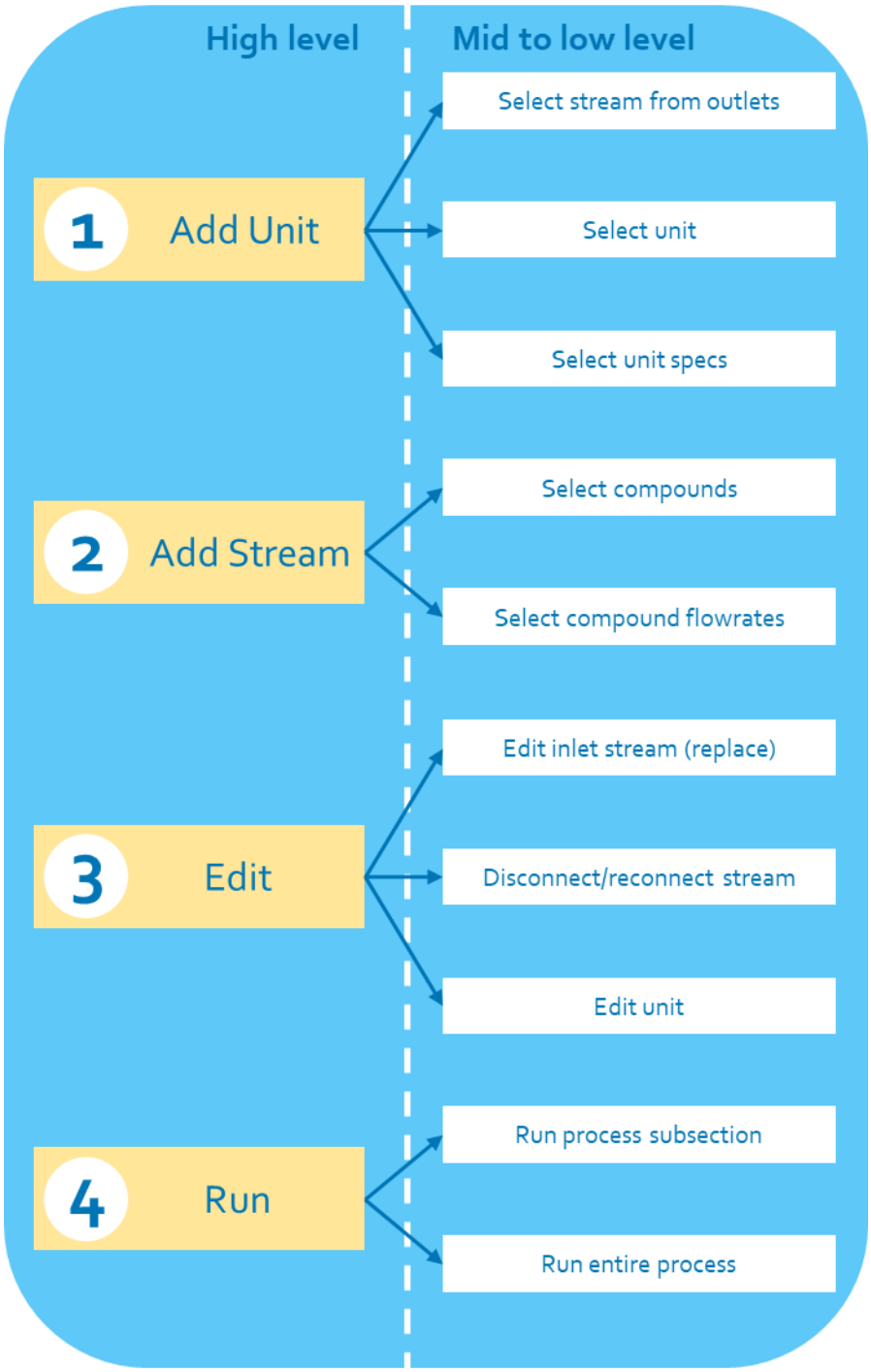

Figure 29: Action space for Complete Process Synthesis

4.4.2 Process Simulation and RL

Process simulation can be likened to RL environments that rely predominantly on equations of physics. Examples of this include *Inverted Pendulum* and *Mountain Car* (Brockman et al., 2016).

RL is applicable to processes where the complete system can be simulated using general physics and chemistry equations. This includes multi-phase systems, heating and cooling, and reactions. Systems which rely largely on real-life experimental data

specific to the context on the simulation are not well suited to reinforcement learning as the agent has to be able to try out different configurations and get immediate rewards. Bioprocesses are examples of systems not well-suited for design via reinforcement learning. This is because although there are general equations describing how to model the systems, these equations rely on experimental data specific to the configuration of the system. This makes it impossible for the agent to test a large variety of possible configurations and get a reward within a reasonable period of time.

It is important to note that additional constraints would have to be added to the agent to ensure that all of its actions don't cause results that are unrealistic for an actual process but allowed within the base simulation. An example of this could be that the agent creates a DC with an unfeasible height in order to achieve a very pure component which happens to correspond to a large reward. There are 2 possible ways of adding such constraints – which are dependent on the context of the scenario. **(1)** If the constraint is a *hard* and cannot be violated then the agent can be prevented from taking such actions. For example, it is physically impossible to have a DC of 1000 m tall, so the agent should not be able to take actions that result in this. **(2)** If the constraint is *soft* (is reasonable that sometimes it may be violated) then an additional punishment (corresponding to the negative effects of the violation) can be added. For example, it is hypothetically possible to have a DC of 100 m – however this would require non-standard structures to ensure that it would work. These non-standard structures would greatly increase its cost. This could be solved by ensuring that the costing equations and corresponding reward incorporate this additional cost.

This thesis utilised a custom-made process simulator that was designed for interaction with a RL agent. Commercial process simulators like Aspen have a design that is focused on use via the user interphase (UI) but do allow for creation and manipulation of flowsheets via code/application program interphase (API). There are however opensource chemical engineering simulators that do allow for flowsheet design via code, for example DWSIM (Wagner, 2019). A further extension of this project would be to use these simulators as a basis for the environment that the RL agent interacts with. This would require an extra layer of code on top of the base simulator that frames interaction with the simulator as an MDP.

This is a strong critique of current commercial simulators, which do not provide flexible ways of creating process flowsheets without the use of the UI. Creating an easy to use API for manipulating process flowsheets would have a large benefit for RL and the development of computation centric chemical engineering in general.

4.4.3 Agent Developments

The agents and approaches used within this thesis could be improved through combining them with other techniques. An in-depth evaluation of which techniques would be well suited to this was considered to be outside of the scope of this thesis,

however on key potential improvement to the agent is given below followed by a list of other potential improvements.

Process synthesis differs from many other RL problems in that the environment is deterministic and the agent can be given any of the internal information within the environment that is useful (e.g. in computer games the agent is often only given the raw pixels as the state value). The key learning that needs to take place by the RL agent to perform well in RL synthesis problems is in selecting the actions that best yield the long-term reward. However, the agent does not necessarily have to learn an approximation of the state transition function. The agent theoretically only has to learn the value function of a given state. This thesis used *model free* RL where the agent did not have access to any internal information to the mechanics of the environment. This can be seen as making the problem unnecessarily difficult. A key further extension of this thesis would be to use a *model is given* version of RL where the agent has access to state transitions and only has to learn the state-value function. In *model is given* problems, search techniques can be combined with the use of a NN to learn the state-value function.

A successful example of *model is given* RL is the use of Monte-Carlo tree search in combination with a NN in Deepmind's Alpha Zero agent which was applied to the games of chess, shogi, go (Silver et al., 2018). A NN which takes in board positions as the state outputs action probabilities, as well as the value function for the expected outcome of the current position. Monte-Carlo tree search was then used to search through future possible situations based on the current board position, in order to more accurately determine the best action. Within process design, a similar approach could be used where search is used to look ahead multiple actions via simulation and use this to update which current action is best.

Further List of Potential Improvements:

- Usage of heuristics and incorporation of engineering knowledge (e.g. for more efficient exploration of the action space)
- Incorporating level of hierarchy of the action to the structure of the NN.
- Could an agent select even higher-level choices? E.g. instead of conditions selecting an outcome for a reactor like "maximise selectivity" – which is then performed by a standard optimisation algorithm?

4.4.4 Milestones

Milestones and benchmarks have been important within the development of RL and ML. For progress within RL's application to process synthesis the following 2 major milestones/benchmarks that would prove that RL has strong application over conventional approaches are proposed: **(1)** If RL was able to solve a process synthesis problem that were previously unsolvable using conventional computational techniques and **(2)** if RL was used to generate a novel section of a real-life process that outperforms older designs.

5 Conclusion and Recommendations

In this thesis, it was demonstrated for the first time that reinforcement learning can be applied to process synthesis problems.

5.1 Case Study 1

In Case Study 1, in finding the known optimal sequence of reactors, it was demonstrated that RL can be applied to simple process synthesis problems. The simple example of RL's application to process synthesis was made easy to understand for chemical engineers without advanced knowledge of RL through: **(1)** Design of visualisation tools – creation of GIFs showing the generation of the BFD and corresponding Livenspiel plot during episodes), **(2)** Use of simple and widespread DQN agent coded using the high level Keras-RL library.

5.2 Case Study 2

The DC train synthesis problem, with the inclusion of branching and the hybrid action space, applied RL to a more difficult domain. This approached more open-ended domains in which RL may have an advantage over conventional approaches. The improved P-DQN agent was able to learn within the more complex environment and outperformed multiple heuristic designs in terms of two different rewards. The agent often created unexpected configurations for the DC trains which were able to produce large rewards. The counter-intuitive results to the DC train synthesis problem show the potential for RL to generate novel process designs that would not necessarily be possible through heuristic design or obvious to a design chemical engineer.

Multiple critiques and possible improvements to Case Study 2 were given in Section 4.2.5. It is recommended these are considered in future work. The largest critique of Case Study 2 was that the resulted LK_{split} solutions were too easy to find (as the maximum split of 99.9% was selected each time). To better demonstrate the ability of the P-DQN agents' performance with respect to the continuous action space, the environment should be refined (e.g. through increasing the DC height related cost) such that the agent is forced to make more difficult trade-offs on the selection of the LK_{split}.

5.3 Final Remarks

It is important to note that the problems solved this thesis are not best solved with RL. Nevertheless, they are a rather selected starting point to show that RL can be applied to process synthesis problems. This thesis has both shown that process synthesis problems can be solved with RL and it has taken key steps towards applying RL to open-ended complete process synthesis problems (where RL may have an advantage over conventional approaches). In a comparison between RL and conventional process synthesis techniques (namely design by humans and design by non-RL computational techniques) it was shown that RL has significant potential advantages against previous techniques of process synthesis. In comparison to typical

computational methods of process synthesis (e.g. MINLP), RL is able to be applied to more open-ended domains – due to its ability to learn and generalise. In comparison to humans using heuristics and engineering judgement, RL is able to test many more process configurations as it is able to take advantage of computers ability to analyse large amounts of data and rapidly interact with simulations.

In an exploration of further developments within RL's application to process synthesis in general the following key points were made:

1. It was shown how the actions that a human chemical engineer takes when performing complete process synthesis could be phrased in terms of an action space for an RL agent.

2. It was highlighted that RL is well suited to process synthesis problems governed by general equations/laws like thermodynamics and not well suited to processes like bioprocessing where simulation requires context dependent experimental data.

3. Current commercial simulators like Aspen were critiqued for their lack of integration with external programs which is of importance as process synthesis becomes more automated (regardless of whether this automation is RL based).

4. The *model free* RL approach used within this thesis was critiqued for making the problem unnecessarily complex. It is recommended that *model is given* RL is used in future developments of RLs application to process synthesis.

5. 2 major milestones/benchmarks that would prove that RL has strong application over conventional approaches would be **(1)** If RL was able to solve a process synthesis problem that were previously unsolvable using conventional computational techniques and **(2)** if RL was used to generate a novel section of a real-life process that outperforms older designs.

In the future, RL for process synthesis is well poised to take advantage of improvements in chemistry and physics simulation. This thesis hopes to potentially stimulate research within this area – with the long-term goal of an RL agent creating novel, profitable processes.

6 References

Analytics Vidhya, 2019. Introduction to Deep Q-Learning for Reinforcement Learning (in Python). Analytics Vidhya. URL https://www.analyticsvidhya.com/blog/2019/04/introduction-deep-q-learning-python/ (accessed 9.19.19).

Angira, R., Babu, B.V., 2006a. Optimization of process synthesis and design problems: A modified differential evolution approach. Chemical Engineering Science 61, 4707–4721.

Angira, R., Babu, B.V., 2006b. Optimization of process synthesis and design problems: A modified differential evolution approach. Chemical Engineering Science 61, 4707–4721.

Arel, I., Liu, C., Urbanik, T., Kohls, A., 2010. Reinforcement learning-based multi-agent system for network traffic signal control. IET Intelligent Transport Systems 4, 128–135.

Ashraf, M., 2018. Reinforcement Learning Demystified: Markov Decision Processes (Part 1) [WWW Document]. Medium. URL https://towardsdatascience.com/reinforcement-learning-demystified-markov-decision-processes-part-1-bf00dda41690 (accessed 9.18.19).

Bailey, M.P., 2019. Artificial Intelligence: A New Reality for Chemical Engineers - Chemical Engineering | Page 1. URL https://www.chemengonline.com/artificial-intelligence-new-reality-chemical-engineers/?printmode=1 (accessed 9.14.19).

Barnicki, S.D., Siirola, J.J., 2004. Process synthesis prospective. Computers & Chemical Engineering 28, 441–446.

Bellman, R., 1958. Dynamic programming and stochastic control processes. Information and control 1, 228–239.

Bellman, R., 1957. A Markovian decision process. Journal of mathematics and mechanics 679–684.

Bellman, R.E., Dreyfus, S., 1962. Applied Dynamic Programming, Princeton Unlv. Press, Princeton, NJ.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W., 2016. OpenAI Gym.

Choi, J., Hyun, M., Kwak, N., 2019. Task-oriented Design through Deep Reinforcement Learning. arXiv preprint arXiv:1903.05271.

Coker, A.K., 2007. Rules of Thumb: Summary, in: Ludwig's Applied Process Design for Chemical and Petrochemical Plants, Volume 1 (4th Edition). Elsevier.

Corripio, A.B., Evans, L.B., 1981. Estimate Costs of Pressure Vessels via Correlations. Chemical Engineer 88, 145–146.

EIA, 2018. Prices for hydrocarbon gas liquids - U.S. Energy Information Administration (EIA) [WWW Document]. URL https://www.eia.gov/energyexplained/hydrocarbon-gas-liquids/prices-for-hydrocarbon-gas-liquids.php (accessed 11.7.19).

Engineering Toolbox, 2019. Gross and Net Heating Values for some common Gases [WWW Document]. URL https://www.engineeringtoolbox.com/gross-net-heating-values-d_420.html (accessed 11.7.19).

Erhan, D., Bengio, Y., Courville, A., Vincent, P., 2009. Visualizing higher-layer features of a deep network. University of Montreal 1341, 1.

Francois-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J., 2018. An Introduction to Deep Reinforcement Learning. FNT in Machine Learning 11, 219–354. https://doi.org/10.1561/2200000071

Greaves, J., 2019. Understanding RL: The Bellman Equations. URL https://joshgreaves.com/reinforcement-learning/understanding-rl-the-bellman-equations/ (accessed 9.18.19).

Hajjar, Z., Tayyebi, S., Ahmadi, M.H.E., 2018. Application of AI in Chemical Engineering. Artificial Intelligence: Emerging Trends and Applications 399.

Hayes-Roth, F., Waterman, D.A., Lenat, D.B., 1983. Building expert system.

ICIS, 2018. US propylene market to remain volatile through 2018 | ICIS [WWW Document]. URL https://www.icis.com/explore/resources/news/2018/07/12/10240640/us-propylene-market-to-remain-volatile-through-2018/ (accessed 11.7.19).

Kirkwood, R.L., Locke, M.H., Douglas, J.M., 1988. A prototype expert system for synthesizing chemical process flowsheets. Computers & chemical engineering 12, 329–343.

Kocis, G.R., Grossmann, I.E., 1987. Relaxation strategy for the structural optimization of process flow sheets. Industrial & engineering chemistry research 26, 1869–1880.

Lange, J.-P., 2017. Don't Forget Product Recovery in Catalysis Research—Check the Distillation Resistance. ChemSusChem 10, 245–252.

Lee, J.H., Shin, J., Realff, M.J., 2018. Machine learning: Overview of the recent progresses and implications for the process systems engineering field. Computers & Chemical Engineering 114, 111–121.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D., 2015. Continuous control with deep reinforcement learning. arXiv preprint arXiv:1509.02971.

Mahalec, V., Motard, R.L., 1977. Procedures for the initial design of chemical processing systems. Computers & Chemical Engineering 1, 57–68.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., Kavukcuoglu, K., 2016. Asynchronous methods for deep reinforcement learning, in: International Conference on Machine Learning. pp. 1928–1937.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., 2015a. Human-level control through deep reinforcement learning. Nature 518, 529.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., 2015b. Human-level control through deep reinforcement learning. Nature 518, 529.

Naylor, L., 2018. Europe PE prices languish as crude prices reach highest point since 2014 [WWW Document]. Icis. URL https://www.icis.com/explore/resources/news/2018/05/17/10222550/europe-pe-prices-languish-as-crude-prices-reach-highest-point-since-2014 (accessed 11.7.19).

Nishida, N., Stephanopoulos, G., Westerberg, A.W., 1981. A review of process synthesis. AIChE Journal 27, 321–351.

Rodrigo B, F.R., Seader, J.D., 1975. Synthesis of separation sequences by ordered branch search. AIChE Journal 21, 885–894.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., 1988. Learning representations by back-propagating errors. Cognitive modeling 5, 1.

Seider, W.D., Seader, J.D., Lewin, D.R., 2009. PRODUCT & PROCESS DESIGN PRINCIPLES: SYNTHESIS, ANALYSIS AND EVALUATION, (With CD). John Wiley & Sons.

Siirola, J.J., Rudd, D.F., 1971. Computer-aided synthesis of chemical process designs. From reaction path data to the process task network. Industrial & Engineering Chemistry Fundamentals 10, 353–362.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. Science 362, 1140–1144.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., 2017. Mastering the game of go without human knowledge. Nature 550, 354.

Sutton, R.S., Barto, A.G., 1998. Introduction to reinforcement learning. MIT press Cambridge.

Tieleman, T., Hinton, G., 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural networks for machine learning 4, 26–31.

Venkatasubramanian, V., 2019. The promise of artificial intelligence in chemical engineering: Is it here, finally? AIChE Journal 65, 466–478.

Wagner, D., 2019. DWSIM - Chemical Process Simulator [WWW Document]. URL http://dwsim.inforside.com.br/wiki/index.php?title=DWSIM (accessed 10.18.19).

Wang, Z., Schaul, T., Hessel, M., Van Hasselt, H., Lanctot, M., De Freitas, N., 2015. Dueling network architectures for deep reinforcement learning. arXiv preprint arXiv:1511.06581.

Xiong, J., Wang, Q., Yang, Z., Sun, P., Han, L., Zheng, Y., Fu, H., Zhang, T., Liu, J., Liu, H., 2018. Parametrized deep q-networks learning: Reinforcement learning with discrete-continuous hybrid action space. arXiv preprint arXiv:1810.06394.

Yaws, C.L., Narasimhan, P., Gabbula, C., 2009. Yaws' Handbook of Antoine Coefficients for Vapor Pressure (2nd Electronic Edition). Knovel New York.

Yee, T.F., Grossmann, I.E., 1990. Simultaneous optimization models for heat integration—II. Heat exchanger network synthesis. Computers & Chemical Engineering 14, 1165–1184.

Zheng, G., Zhang, F., Zheng, Z., Xiang, Y., Yuan, N.J., Xie, X., Li, Z., 2018. DRN: A deep reinforcement learning framework for news recommendation. Presented at the Proceedings of the 2018 World Wide Web Conference, International World Wide Web Conferences Steering Committee, pp. 167–176.

Zhou, Z., Li, X., Zare, R.N., 2017. Optimizing chemical reactions with deep reinforcement learning. ACS central science 3, 1337–1344.

Zychlinski, S., 2019. Qrash Course: Reinforcement Learning 101 & Deep Q Networks in 10 Minutes [WWW Document]. Medium. URL https://towardsdatascience.com/qrash-course-deep-q-networks-from-the-ground-up-1bbda41d3677 (accessed 9.18.19).

Appendix A: Column modelling and cost correlations

Column operating conditions and number of trays:

The modelling of each column in the train followed a design procedure of several steps. The first step was to select an LK and LK_{split} given a feed stream of known composition and flowrate. For the RL agent, these were selected as actions whilst for the heuristics cases the LK was pre-specified and the LK_{split} selected for each was assumed to be 99.9% as this maximised the reward functions specified in Section 3.3.4. After selecting both an LK and LK_{split} for each column, the flowrate and composition of its tops and bottoms could be determined. Following this, the column's operating pressure and bottoms temperature could be calculated. Since cooling water is regarded as the most cost-effective condenser utility, the condenser was assumed to operate ideally at 60°C. This temperature was then used to calculate the pressure at which the incoming vapour mixture completely condenses (p_{bub}), which was assumed to be equivalent to the operational pressure of the column (p_o). p_{bub} was calculated as

$$p_{bub} = \sum_i p_i^{vap} z_i \tag{12}$$

where p_i^{vap} and z_i are the vapour pressure (kPa at 60°C) and mole fractions of component i in the vapour feed to the condenser respectively. p_i^{vap} for each compound was calculated using the Antoine equation and relevant constants (Yaws et al., 2009).

The temperature at the top of the column (T_{top}) was assumed fixed and equal to the dew point (T_{dew}) of the distillate stream, which can be determined by solving Equation (13),

$$\sum_i \frac{z_i}{K_i(T_{dew})} - 1 = 0 \tag{13}$$

where K_i can be expressed for component i as

$$K_i = \frac{p_i^{vap}}{p_{column}} \tag{14}$$

Likewise, the temperature of the reboiler (T_{bot}) can be estimated by the bubble point of the bottoms stream (T_{bub}), which can be determined by solving Equation (15).

$$\sum_i K_i(T_{bub}) - 1 = 0 \tag{15}$$

The geometric mean relative volatilities of each neighbouring pair of the compounds separated in each column ($(\alpha_{ij})_m$) were calculated using Equations (16) and (17) with i and j representing the light and heavy component in each pair respectively.

$$(\alpha_{ij})_m = \sqrt{(\alpha_{ij})_{top}(\alpha_{ij})_{bot}} \tag{16}$$

$$\alpha_{ij} = \frac{K_i(T)}{K_j(T)} \tag{17}$$

The minimum number of stages in each column (N_{min}) could then be used using the Fenske equation (Equation (18)) assuming total reflux

$$N_{min} = \frac{log\left[\left(\frac{d_{LK}}{d_{HK}}\right)\left(\frac{b_{HK}}{b_{LK}}\right)\right]}{log\left(\alpha_{LK,HK}\right)_m} \tag{18}$$

where d_i and b_i represent the distillate and bottoms molar flowrates respectively whilst subscripts LK and HK representing the selected light and heavy key component respectively.

The minimum reflux ratio (R_{min}) was then determined using Equation (19),

$$R_{min} = \frac{\frac{L_f}{D}\left[\left(\frac{d_{LK}}{f_{1,LK}}\right) - \left(\alpha_{LK,HK}\right)_F\left(\frac{d_{HK}}{f_{1,HK}}\right)\right]}{\left(\alpha_{LK,HK}\right)_F - 1} \tag{19}$$

where L_f and D represent the total molar flowrates of the liquid feed and distillate respectively and $f_{1,i}$ is the molar flowrate of component i in the liquid feed. The actual reflux ratio (R) was then determined as 1.3 times R_{min}.

The actual number of stages in the column (N) was determined using the Gilliland correlation (Equation (20))

$$\mathrm{Y} = 1 - \exp\left[\left(\frac{1 + 54.4 \cdot \mathrm{X}}{11 + 117.2 \cdot \mathrm{X}}\right)\left(\frac{\mathrm{X} - 1}{\mathrm{X}^{0.5}}\right)\right] \tag{20}$$

where

$$\mathrm{Y} = \frac{N - N_{min}}{N + 1} \tag{21}$$

$$\mathrm{X} = \frac{R - R_{min}}{R + 1} \tag{22}$$

Column dimensions:

As was the case with operating conditions and number of trays, the height and internal diameter of each column were determined following several conventional design steps. Due to the design literature used originating predominantly from the United States, American Engineering System Units were used for dimension calculations and later converted back to SI units.

The height of each column (H) was calculated assuming tray spacings of 22 inches, and vapour and liquid allowances of 4 and 6 feet respectively (Coker, 2007).

The internal diameter of each column (D_i) was calculated using Equations (23) and (24)

$$D_i = \sqrt{\frac{4A}{\pi}} \quad (23)$$

$$A = \frac{\dot{V}}{\mu} \quad (24)$$

where A is the internal tray area. The volumetric flow rate ($\dot{V}$) was approximated using the molar flowrates and densities of the components in the tops stream assuming vapour state at typical column pressures and the internal linear vapour velocity (μ) was assumed to be 0.6 m/s, as is typical for ordinary distillation at moderate pressures (Coker, 2007).

Column capital costing:

The capital cost of each column was calculated as a function of D_i, H, N, p_o and T_{bot} as previously defined. In order to maintain consistency, pressures and temperatures were converted from SI to American Engineering System Units respectively. All capital costing correlations were taken from (Seider et al., 2009) unless stated otherwise. The design pressure (P_d) of each column was determined to be equal to 10 or 1.1P_o when P_o was determined to be less than 10 psig or greater than 1000 psig respectively whilst it was otherwise determined using Equation (25).

$$P_d = \exp[0.60608 + 0.91615[\ln(P_o)] + 0.0015655[\ln(P_o)]^2] \quad (25)$$

The design temperature (T_d) of each column was calculated by adding 50°F to T_{bot}. The resulting T_d was then used to determine the maximum allowable stress of each column vessel (S) as per Table 6.

Table 6: Maximum allowable distillation vessel stresses categorised by temperature

Temperature	Maximum Allowable Stress (psi)
-20 to 650	15000
700	15000
750	15000
800	14750
850	14200
900	13100

Using this value a method for determining the average wall thickness of each column (t_P) was determined using Equation (26) as proposed by (Corripio and Evans, 1981)

$$t_P = \frac{P_d D_i}{2SE - 1.2P_d} \quad (26)$$

where E is the weld efficiency and is equal to 0.85, assuming the use of carbon steel as the material of construction. The weight of the vessel and heads in pounds (W) could then be calculated as

$$W = 3.408\pi(12D_i + t_P)(L + 0.8D_i) \quad (27)$$

assuming no strong winds, earthquakes or corrosion will affect the columns being designed.

The vertical vessel purchase cost of each column in US Dollars in 2003 ($C_{P,2003}$) could then be calculated as a function of W as

$$C_{V,2003} = \exp[7.0374 + 0.18255[\ln(W)] + 0.02297\ln(W)\}^2] \quad (28)$$

whilst the added cost for ladders and platforms ($C_{PL,2003}$) could be calculated as

$$C_{\mathrm{PL},2003} = 237.1(D_i)^{0.63316}(L)^{0.80161} \quad (29)$$

After calculating the vessel purchase and added costs, the base tray cost in 2003 ($C_{BT,2003}$) was calculated based on N as

$$C_{BT,2003} = 369\exp(0.1739D_i) \quad (30)$$

The cost for installed trays ($C_{T,2003}$) was then calculated as

$$C_{T,2003} = NF_{NT}F_{TT}F_{TM}C_{BT,2003} \quad (31)$$

where F_{TT} and F_{TM} are factors for tray type and material of construction respectively whilst F_{NT} is a factor calculated conditionally as a function of number of trays. Trays were assumed to be basic sieve trays constructed of carbon steel and so the former two factors were assumed equal to 1. For $N \geq 20$, F_{NT} was assumed equal to one whilst for $N < 20$, F_{NT} was calculated by

$$F_{NT} = \frac{2.25}{1.0414^N} \quad (32)$$

The total capital cost in Rands in 2019 ($C_{tot,2019}$) was then calculated as

$$C_{tot,2019} = x_{rand}(C_{T,2003} + C_{PL,2003} + C_{V,2003})\frac{CEPCI_{2019}}{CEPCI_{2013}} \quad (33)$$

where x_{rand} is the US Dollar/Rand exchange rate (taken as R15.23/USD on 01/10/2019) whilst relevant Chemical Engineering Plant Cost Index ($CEPCI$) values were used to scale capital cost from 2003 to 2019. $CEPCI_{2013}$ and $CEPCI_{2019}$ were taken as 394 and 613.16 respectively.

Column product revenue:

In order to construct later reward functions based on both capital cost and separation of products into streams pure enough to sell, the sales values of each of the components in the feed stream were found. These values, along with their sources are shown in Table 7.

Table 7: Market prices of compounds separated

Compound	Selling price ($/million BTU)	Heating value (MJ/kg)	Source
Ethane	2.54	51.9	(EIA, 2018)
Propylene	17.58	49.0	(ICIS, 2018)
Propane	4.27	50.4	(EIA, 2018)
1-Butene	29.47	48.5	(Naylor, 2018)
n-Butane	5.31	49.4	(EIA, 2018)
n-Pentane	13.86	48.6	(EIA, 2018)

Using the molar weights and gross heating values (Engineering Toolbox, 2019) of each compound shown in Table 8 the selling price of each compound could be calculated on the basis of molar flowrates. The annual value of product streams of over 96% purity could then be calculated assuming 8000 hours of annual operation time.

Table 8: Gross heating values and molar weights of compounds separated

Compound	Gross heating value (MJ/kg)	Molar weight (kg/kmol)
Ethane	51.9	30.07
Propylene	49.0	42.08
Propane	50.4	44.097
1-Butene	48.5	56.108
n-Butane	49.4	58.124
n-Pentane	48.6	72.15

Appendix B:

B1: Heuristic Design DC Train Configurations Diagrams

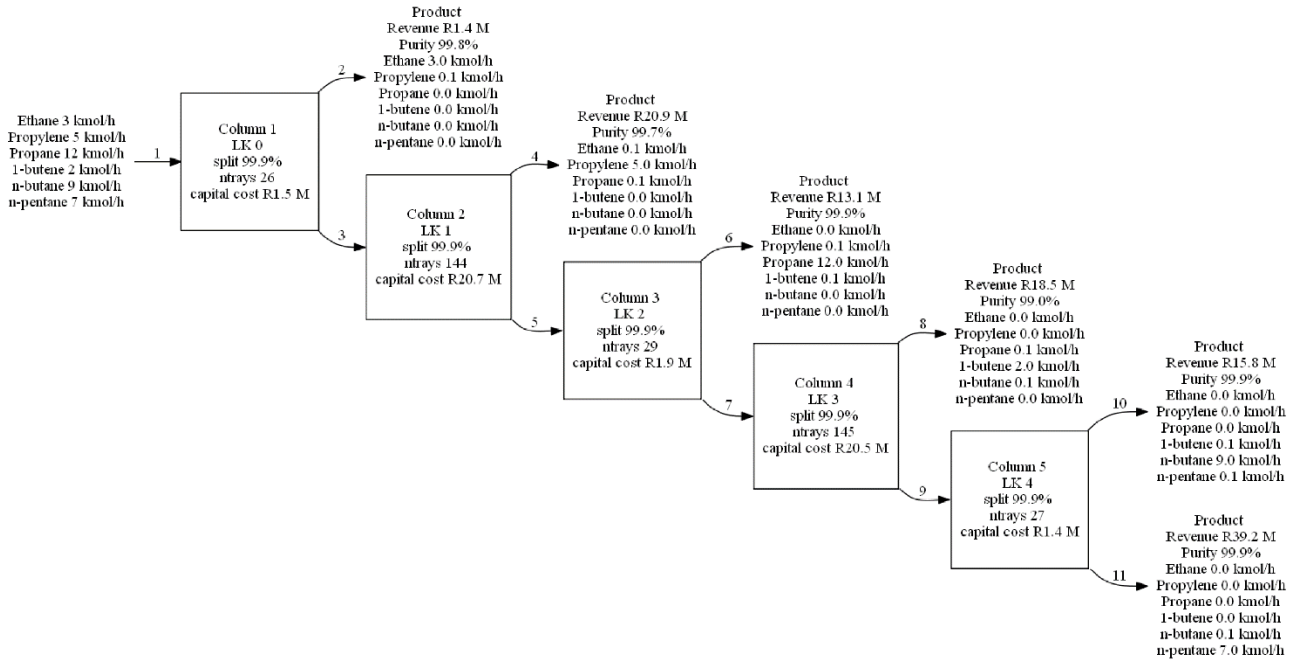

Figure 30: Separation by boiling point heuristic design configuration

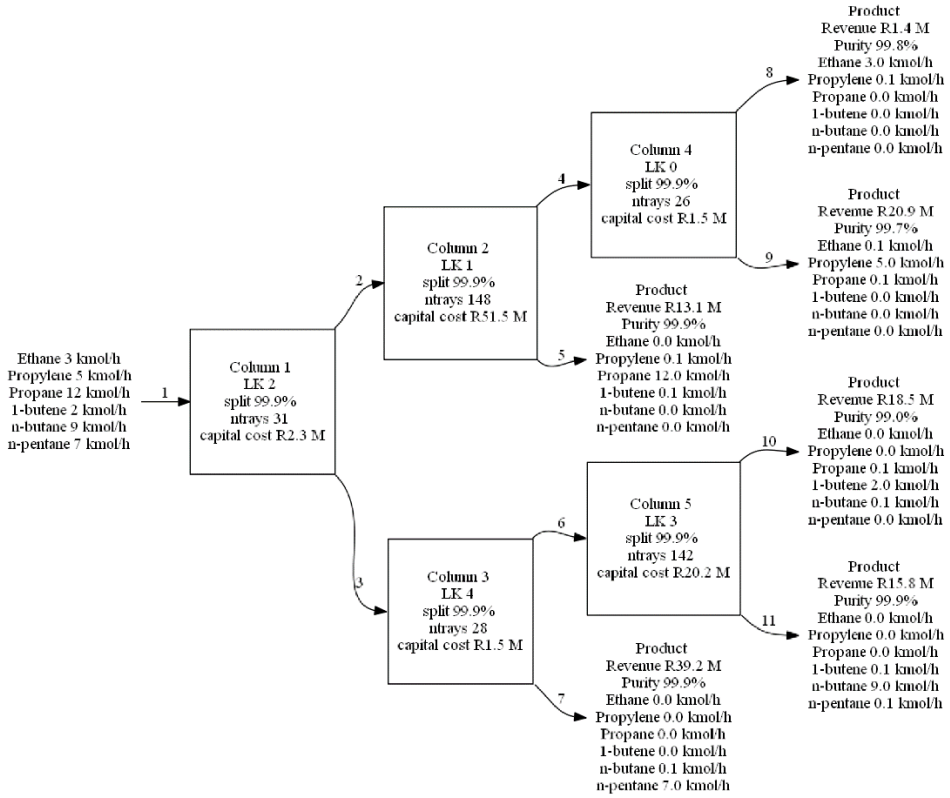

Figure 31: Separation by flowrate heuristic design configuration

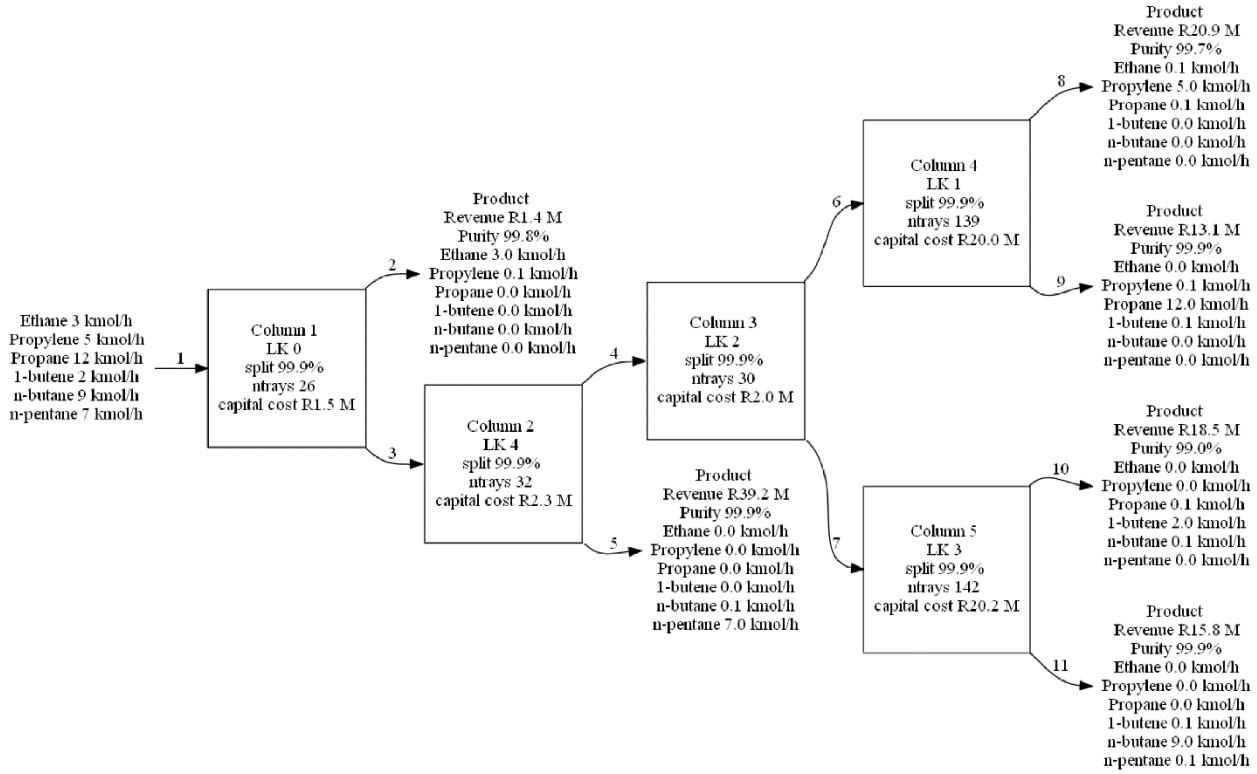

Figure 32: Separation by relative volatility heuristic design configuration

B1: Agent Generated DC Train Configurations Diagrams

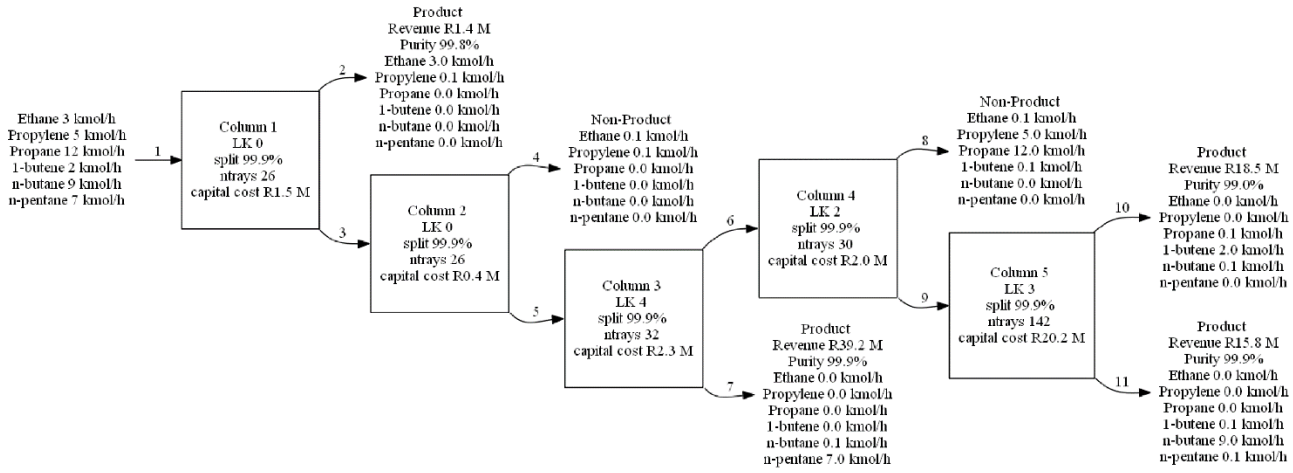

Figure 33: Baseline Agent Reward 1 DC train configuration

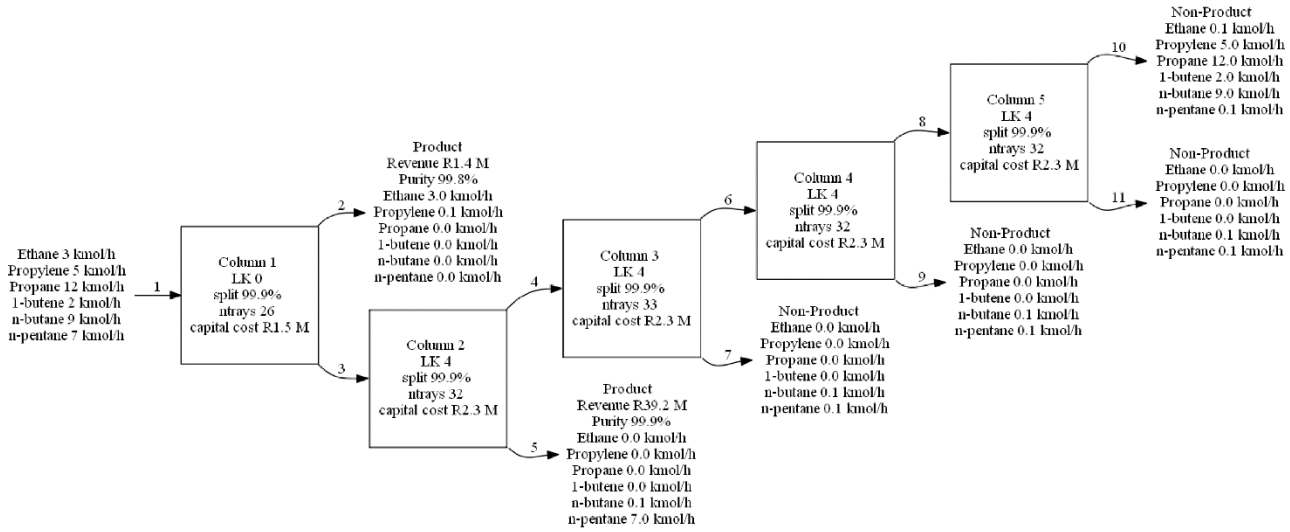

Figure 34: With Freeze Agent Reward 1 DC train configuration

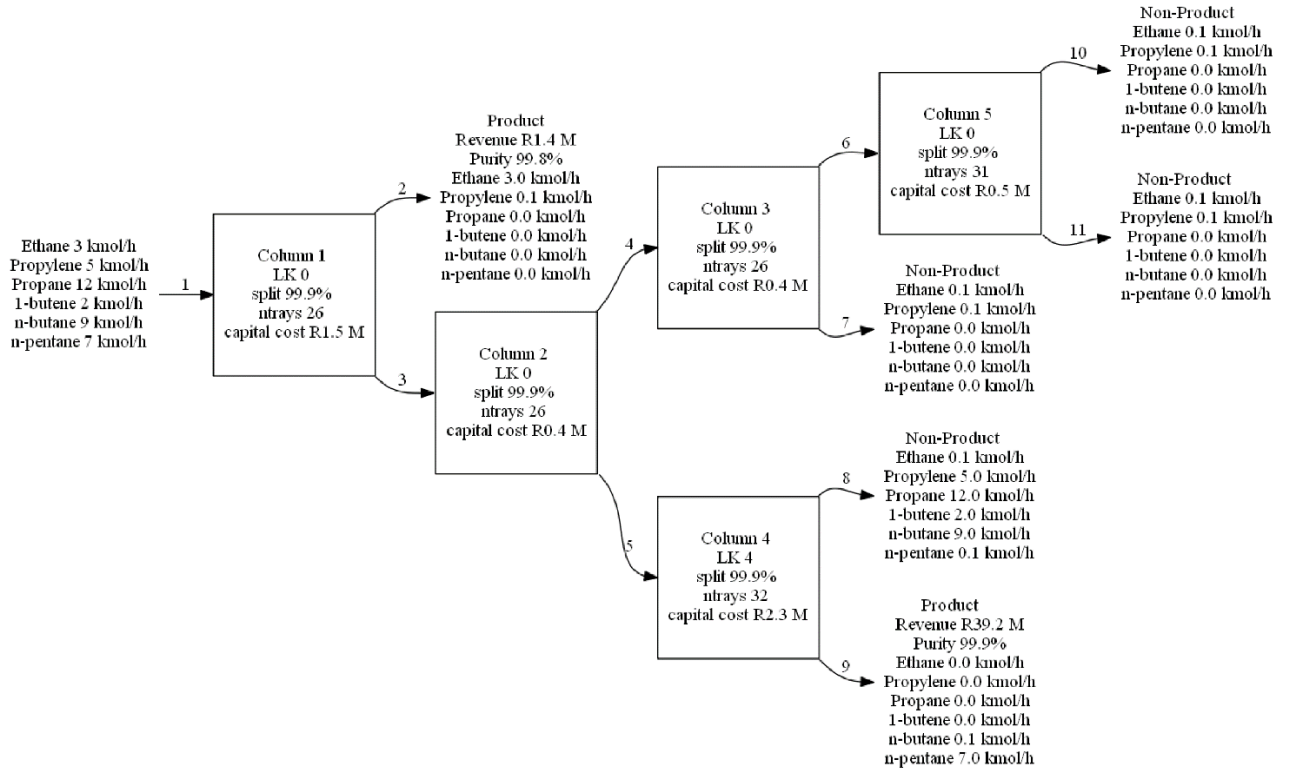

Figure 35: With Duelling Layer Agent Reward 1 DC train configuration

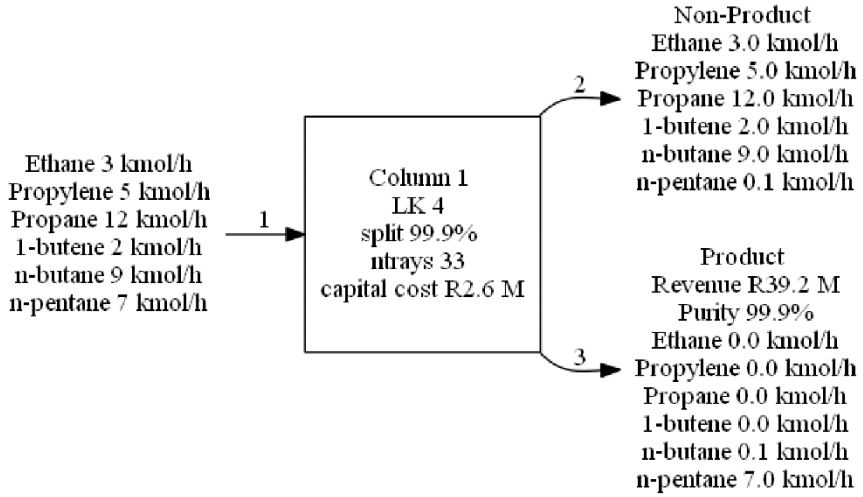

Figure 36: With all improvements Agent Reward 1 DC train configuration

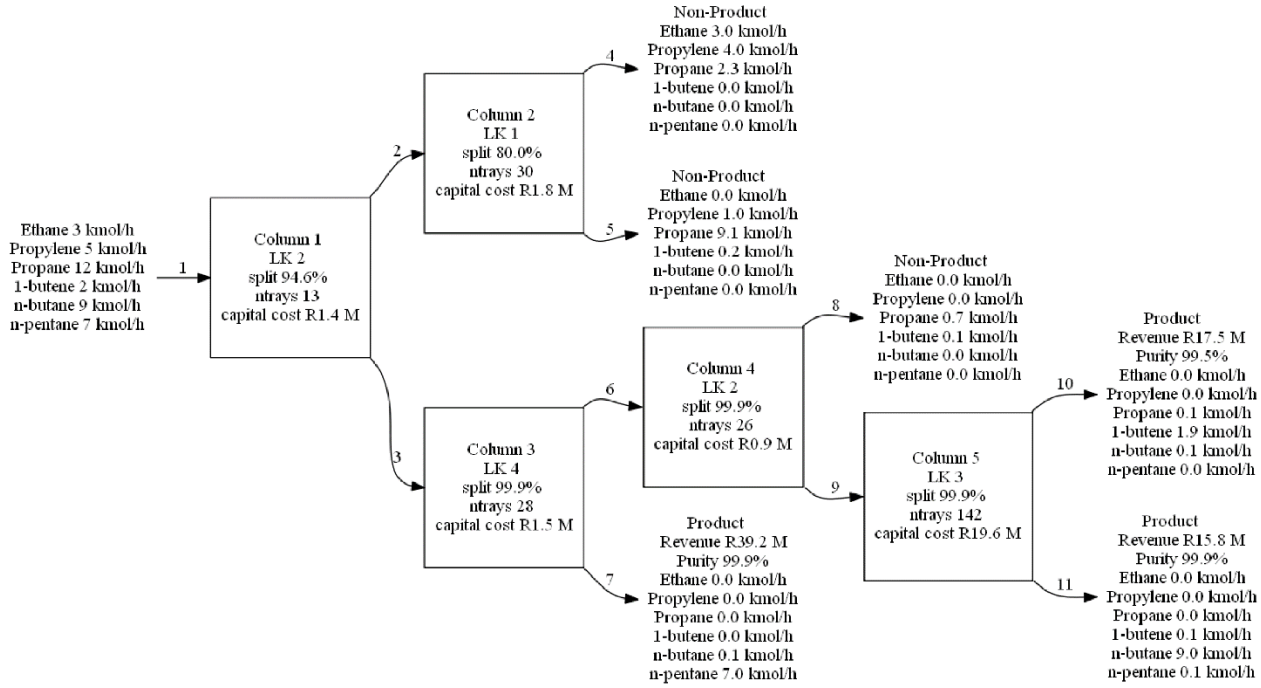

Figure 37: Baseline Agent Reward 2 DC train configuration

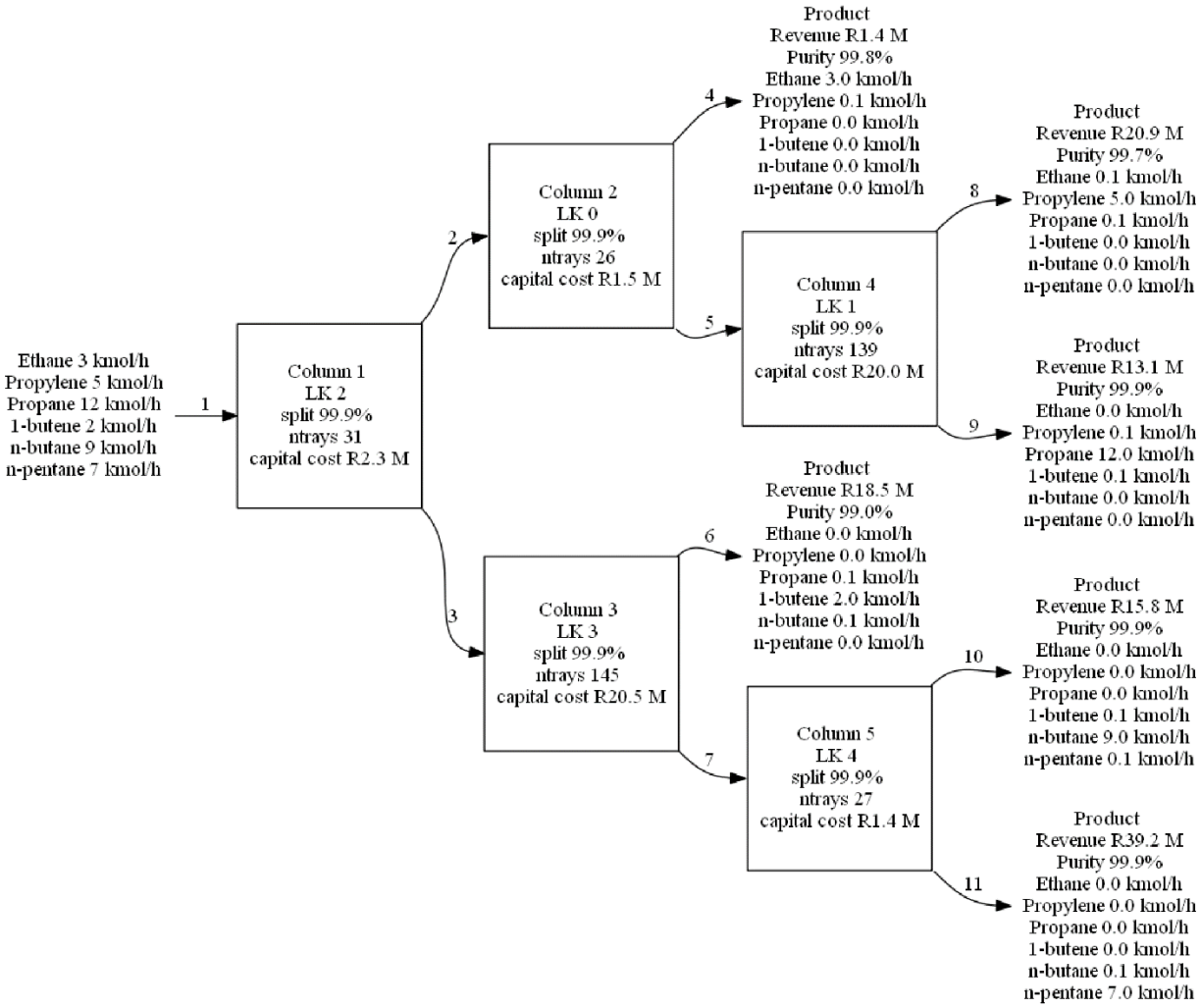

Figure 38: Improved Agent Reward 2 DC train configuration