Flowsheet generation through hierarchical reinforcement learning and graph neural networks

Stops, Laura; Leenhouts, Roel; Gao, Qinghe; Schweidtmann, Artur M.

RESEARCH ARTICLE

Process Systems Engineering

AIChE
JOURNAL

Flowsheet generation through hierarchical reinforcement learning and graph neural networks

Laura Stops [ID] | Roel Leenhouts [ID] | Qinghe Gao [ID] | Artur M. Schweidtmann [ID]

Department of Chemical Engineering, Delft University of Technology, Delft, The Netherlands

Correspondence
Artur M. Schweidtmann, Department of Chemical Engineering, Delft University of Technology, Van der Maasweg 9, Delft 2629 HZ, The Netherlands.
Email: a.schweidtmann@tudelft.nl

Abstract

Process synthesis experiences a disruptive transformation accelerated by artificial intelligence. We propose a reinforcement learning algorithm for chemical process design based on a state-of-the-art actor-critic logic. Our proposed algorithm represents chemical processes as graphs and uses graph convolutional neural networks to learn from process graphs. In particular, the graph neural networks are implemented within the agent architecture to process the states and make decisions. We implement a hierarchical and hybrid decision-making process to generate flowsheets, where unit operations are placed iteratively as discrete decisions and corresponding design variables are selected as continuous decisions. We demonstrate the potential of our method to design economically viable flowsheets in an illustrative case study comprising equilibrium reactions, azeotropic separation, and recycles. The results show quick learning in discrete, continuous, and hybrid action spaces. The method is predestined to include large action-state spaces and an interface to process simulators in future research.

KEYWORDS
artificial intelligence, graph convolutional neural networks, graph generation, process synthesis, reinforcement learning

1 | INTRODUCTION

The chemical industry is approaching a disruptive transformation toward a more sustainable and circular future.[1–3] As a major contributor to global emissions, tremendous changes are required and the

chemical industry needs to face a paradigm shift.[1] This also requires rethinking regarding the conceptualization of novel processes.[2,4] Simultaneously, innovations are pushed by new possibilities due to emerging digital technologies. Digitization and in particular artificial intelligence (AI) offer new possibilities for process design and therefore have the potential contribute to the transformation of chemical engineering.[1,3,5]

In the last decade, reinforcement learning (RL) has demonstrated its potential to solve complex decision-making problems, for example, by showing human-like or even superhuman performance in a large variety of game applications.[6–8] RL is a subcategory of machine learning (ML) where an agent learns to interact with an environment based on trial-and-error.[9] Especially since 2016, when DeepMind's

AlphaGo[10] succeeded against a world-class player in the game Go, RL has attracted great attention. In recent developments, RL applications have proven to successfully compete with top-tier human players in even real-time strategy video games like StarCraft II[11] and Dota 2.[12]

The accomplishments of RL in gaming have initiated significant developments in other research fields, including chemistry and chemical engineering. In process systems engineering, RL has been mainly applied to scheduling[13,14] and process control.[15–19] After first appearances of RL for process control in the early 1990s,[15] the development was pushed with the rise of deep RL in continuous control in games[20] and physical tasks.[21] Spielberg et al.[16] first transferred deep RL to chemical process control. In recent works, the satisfaction of joint chance constraints[17] and the integration of process control into process design tasks[18,19] via RL were considered.

In contrast to continuous process control tasks, RL in molecule design is characterized by discrete decisions, such as adding or removing atoms. Several methods use RL for the design of molecules with desired properties.[22–26] First applications generate simplified molecular-input line-entry system (SMILES) strings using RL agents with pretrained neural networks.[23,26] Zhou et al.[24] introduced a method solely based on RL, thereby ensuring chemical validity. Recently, RL based molecule design has been further enhanced in terms of exploration strategies[27] or by combining RL with orientation simulations.[28] In another approach, You et al.[22] introduced a graph convolutional policy network (GCPN) that represents molecules as graphs. It allows using graph neural networks (GNNs) to approximate the policy of the RL agent and to learn directly on the molecular graph. Using GNNs on molecule graphs to predict molecule properties[29–32] has also shown promising results besides RL. For example, Schweidtmann et al.[29] achieved competitive results for fuel property prediction by concatenating the output of a GNN into a molecule fingerprint and further passing it through a multilayer perceptron (MLP).

Graph representation and RL are also applied in other engineering fields. For example, Ororbia and Warn[33] represent design configurations of planar trusses as graphs in an RL optimization task.

Recently, important first steps have been made toward using RL to synthesize novel process flowsheets.[34–39] Midgley[34] introduced the "Distillation Gym", an environment in which distillation trains for non-azeotropic mixtures are generated by a soft-actor-critic RL agent and simulated in the open-source process simulator COCO. The agent first decides whether to add a new distillation column to the intermediate flowsheet and subsequently selects continuous operating conditions. In an alternative approach to generate process flowsheets, Khan and Lapkin[35] presented a value-based agent that chooses the next action by assessing its value, based on previous experience. The agent operates within a hybrid action space, that is, it makes discrete and continuous decisions. In a recent publication, Khan and Lapkin[40] introduced a hierarchical RL approach to process design, capable of designing more advanced process flowsheets, also including recycles. A higher level agent constructs process sections by choosing sub-objectives of the process, such as maximizing the yield. Then, a lower level agent operates within these sections and chooses unit types and discretized parametric control variables that define unit conditions. Due to the discretization, the agent operates only in a discrete action space. As another approach to

synthesize flowsheets with RL, Göttl et al.[36] developed a turn-based two-player-game environment called "SynGameZero." Thereby, they reused an established tree search RL algorithm from DeepMind.[8] Recently, Göttl et al.[37] enhanced their work by allowing for recycles and utilizing convolutional neural networks (CNNs) for processing large flowsheet matrices. Additionally, the company Intemic[38] has recently developed a "flowsheet copilot" that generates flowsheets iteratively, embedded in a 1-player-game. Intemic offers a web front-end in which raw materials and desired products can be specified. Then, a RL agents selects unit operations as discrete decisions using the economic value of the resulting process as objective. Furthermore, Plathottam et al.[39] introduced a RL agent that optimizes a solvent extraction process by selecting discrete and continuous design variables within predefined flowsheets.

One major gap in the previous literature on RL for process synthesis is the state representation of flowsheets. We believe that a meaningful information representation is key to enable breakthroughs of AI in chemical engineering.[5] Previous works represent flowsheet in matrices comprising thermodynamic stream data, design specifications, and topological information.[37] However, we know from computer science research that passing such matrices through CNNs is limited as they can only operate on fixed grid topologies, thereby exploiting spatial but not geometrical features.[41] In contrast, graph convolutional networks (GCNs) handle differently sized and ordered neighborhoods[42] with the topology becoming a part of the network's input.[43] Since flowsheets are naturally represented as graphs with varying size and order of neighborhoods, GCNs can take their topological information into account. Another gap in the literature concerns the combination of multiple unit operation types, recycle streams and a larger, hybrid action space. While previous works proposed these promising techniques in individual contributions,[34–40] they have not yet been combined to a unified framework.

In this contribution, we represent flowsheets as graphs consisting of unit operations as nodes and streams as edges (cf., References 44,45). The developed agent architecture features a flowsheet fingerprint, which is learned by processing flowsheet graphs in GNNs. Thereby, proximal policy optimization (PPO)[46] is deployed with modifications to learn directly on graphs and to allow for hierarchical decisions. In addition, we combine a hybrid action space, hierarchical actor-critic RL, and graph generation in a unified framework.

2 | REINFORCEMENT LEARNING FOR PROCESS SYNTHESIS

In this section, we introduce the methodology and the architecture of the proposed method. To apply RL to process synthesis, the problem is first formulated as a Markov decision process (MDP) which is defined by the tuple $M = \{S, A, T, R\}$. An MDP consists of states $s \in S$, actions $a \in A$, a transition model $T : S \times A \rightarrow S$, and a reward function R.[9] In the considered problem, states are represented by flowsheet graphs, while actions comprise discrete and continuous decisions. More specifically, the discrete decisions consist of selecting a new unit operation as well as the location where it is added to the

intermediate flowsheet. The continuous decisions are to define one or several specific continuous design variables per unit operation. For the environment, we implemented simple functions in Python to simulate the considered flowsheet. Finally, a reward is calculated and returned to the agent.

While most RL methods can be divided into value-based and policy-based approaches, actor-critic RL takes advantage of both concepts.[9] In contrast to value-based RL methods that cannot be easily adapted to continuous actions,[21,47] actor-critic approaches can learn policies for both, discrete and continuous action spaces and are thus also suitable for hybrid tasks.[48] Subsequently, several recent state-of-the-art policy optimization methods propose an actor-critic setup.[21,46–50] As shown in Figure 1, actor-critic agents consist of a critic that estimates the value function and an actor that decides for actions by approximating the policy.[9]

The RL framework presented in this work is derived from the actor-critic PPO algorithm by OpenAI.[46] In PPO, the objective function is clipped to prevent a collapse of the agent's performance during training. To favor exploration, an entropy term[51] is added to the loss function. Additionally, the generalized estimation of the advantage $\hat{A}$[52] is used for updating the networks.

2.1 | State representation

The main feature of the proposed method is the representation of the states by directed flowsheet graphs. This characteristic allows us to process the states in GNNs, thereby taking topological information into account.

Figure 2 demonstrates the graph representation of flowsheets. Feeds, products, and unit operations are represented by nodes, storing the type of unit operation and design variables. The edges include thermodynamic information about process streams, like temperature, molar flow, and molar fractions.

Intermediate flowsheets feature nodes of the type "undefined." Whenever a new unit operation is added to the flowsheet, the resulting open streams are considered as such "undefined" nodes. In subsequent steps, they represent possible locations for placing new unit operations. Consequently, adding a new unit operation practically means replacing an "undefined" node with a defined one.

2.2 | Agent

At the heart of the proposed RL method stands a hierarchical, hybrid actor-critic agent composed of multiple GNNs and MLPs. Its characteristics are introduced hereinafter.

2.2.1 | Hierarchical, hybrid action space

The architecture of the agent is decisively affected by the considered hierarchical and hybrid action space. The decision-making process is
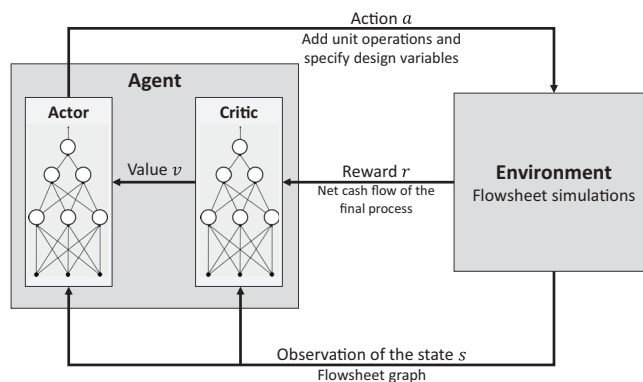
FIGURE 1 Agent–environment interaction in an actor-critic policy optimization approach for flowsheet synthesis. The agent approximates the policy and makes decisions. Meanwhile, the critic estimates the value of the environment's state using the flowsheet graph, which is used to evaluate the agent's decisions. Here, actor and critic both deploy graph convolutional neural networks.
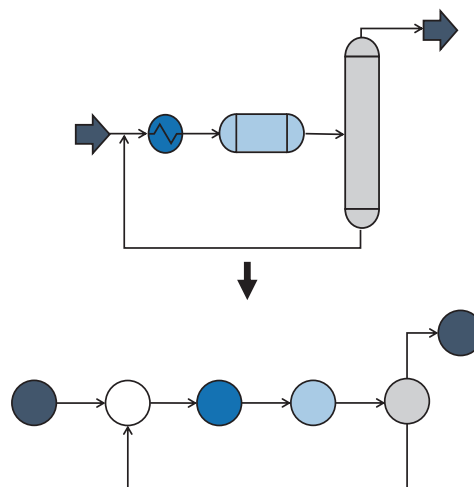
FIGURE 2 Example of a flowsheet displayed as a graph. Unit operations, feeds, and products are represented as nodes, whereas streams are represented as edges.

illustrated in Figure 3. Every action consists of three levels of decisions: (i) select a location, (ii) add a new unit operation, and (iii) define a continuous design variable.

In the first level, the agent decides for an open stream and thus for the location of the next flowsheet expansion. As discussed previously, open streams are identified by "undefined" nodes. In the second level, the agent decides which type of unit operation will be added. Thereby, the agent can choose to add a distillation column, a heat exchanger, or a reactor. Furthermore, it can decide to add a recycle by introducing a splitter and a mixer into the flowsheet. As a fifth option, the agent can declare the considered stream as a product. If a unit operation is added, the third level decision is to specify the design variables of the corresponding unit operation. Although it is possible to set multiple design variables in this step, we chose to only set one variable for simplification reasons. Thus, one characteristic variable for
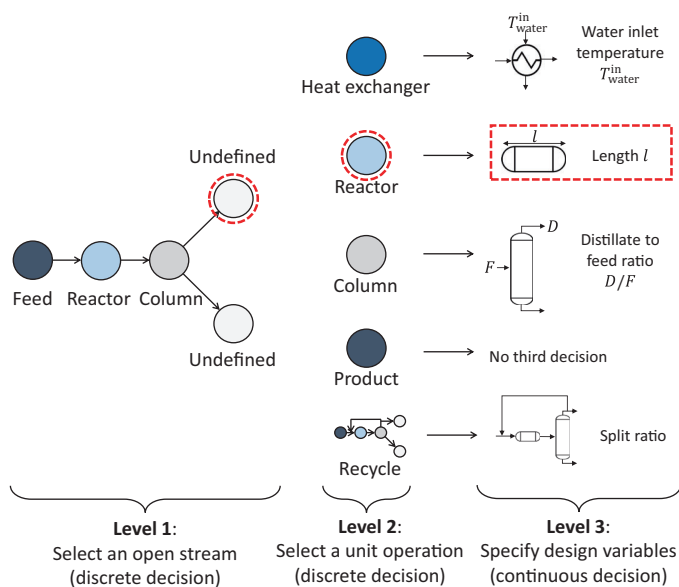
FIGURE 3 Hierarchical decision levels of the agent, starting from an intermediate flowsheet. In the first level, the agent selects a location where the flowsheet will be extended. Possible locations are open streams, represented by "undefined" nodes. In the presented flowsheet, both streams leaving the column can be chosen. Then, the agent selects a unit operation. Thereby, the options are to add a heat exchanger, a reactor, a column, a recycle, or to sell the stream as a product. Finally, a continuous design variable is selected for each unit operation. This third decision depends on which unit operation was selected previously.

each unit operation is defined in this step while all other variables are fixed. For the current implementation of the agent, the recycle stream is always inserted into the feed stream. Whereas the first two levels are discrete decisions, the third level decisions are continuous. This combination of discrete and continuous decisions is referred to as hybrid action space.

2.2.2 | Using GNNs to generate flowsheet fingerprints

In RL, every iteration of the agent-environment interaction starts with the observation of the environment's state s, as shown in Figure 1. In other approaches,[34,36,37,40] states or rather flowsheets are represented by vectors or matrices and, for example, passed through CNNs for the observation step.[37] Instead, in the herein presented approach, states are represented by flowsheet graphs (cf., in Section 2.2.1). To observe and process the therein stored information, the flowsheet graphs are passed through GCNs and encoded into a vector format called flowsheet fingerprint. The advantage of using graphs and GCNs is that it allows operating in variable neighborhoods with different numbers and ordering of nodes, thereby taking spatial and spectral information into account.[41–43] Thus, we believe that graphs and GCNs are better suited for representing and processing the branched connectivity of flowsheets than passing matrices through CNNs.

For this step, we transfer the method introduced by Schweidtmann et al.,[29] who apply GNNs to generate molecule fingerprints, to flowsheets. The approach utilizes the message passing neural network (MPNN) proposed by Gilmer et al.[30]

The overall scheme to process a flowsheet graph is displayed in Figure 4 and consists of a message passing and a readout phase. First, the flowsheet graph is processed through a GCN with several layers to exchange messages and update node embeddings. Afterward, a pooling function generates a vector format, the flowsheet fingerprint, in the readout phase. After several steps of message passing, sum-pooling is deployed for the subsequent readout phase. Thereby, the node embeddings of the last layer are concatenated into a vector format, the flowsheet fingerprint.

For every step in the message passing phase, first the node and edge features of the neighborhood of each node in the flowsheet graph are processed. Therefore, GCNs are utilized to exchange and update information in the message passing phase. The functionality of a graph convolutional layer is illustrated in Figure 5, following Schweidtmann et al.[29] The figure visualizes the procedure to update the node embeddings of the blue node. Therefore, the information stored in the yellow neighboring nodes and the corresponding edges is processed and combined to a message through the message function M. Then, the considered node is updated through the message in the update function U. In each layer of a GCN, this procedure is conducted for every node of the graph.

2.2.3 | Hierarchical agent architecture

For the architecture of the agent, a structure suggested by Fan et al.[48] for hierarchical and hybrid action spaces is used. Thereby, individual MLPs are applied for each level of decisions and one MLP is applied as a critic to evaluate the decisions.

The architecture of the actor-critic approach is illustrated in Figure 6. In the "fingerprint generation" step, the state represented by a flowsheet graph is processed to a flowsheet fingerprint through a GCN (cf., in Section 2.2.2).

Additionally, the updated graph resulting from the message passing phase of the fingerprint generation is passed to the "actor" step. Therein, the updated graph is further processed by an additional GCN. This represents the first level of the actor, which is to select an open stream to further extend the flowsheet. Thereby, the method takes advantage of the graph representation in which open streams end in "undefined" nodes. In the GCN of the first level decision, the number of node features is reduced to one (cf., related literature on node classification tasks[42]). Furthermore, all nodes which do not correspond to open streams are filtered out. The remaining node feature of each nodes in the last GCN layer represents its probability to be chosen as the location for adding a new unit. Then, the ID of the selected node is concatenated with the previously computed flowsheet fingerprint before it is passed on to the second and third level actors as input. The ID of a node is a numerical counter, which is assigned to the node when it is created and acts as a unique identifier.
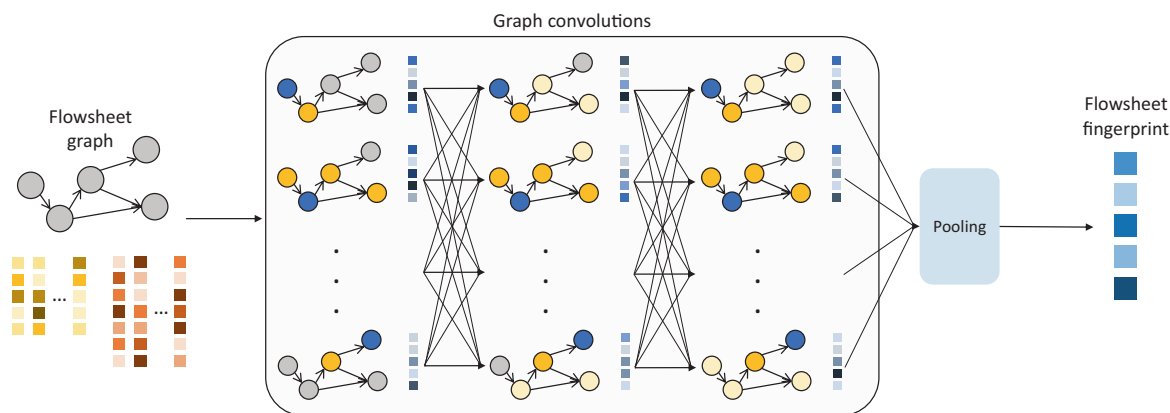
FIGURE 4 Flowsheet fingerprint generation derived from Schweidtmann et al.[29] The flowsheet graph is processed through several GCN layers to perform message passing and update node embeddings. In the readout step, a pooling function is applied, resulting in a vector format, the flowsheet fingerprint.
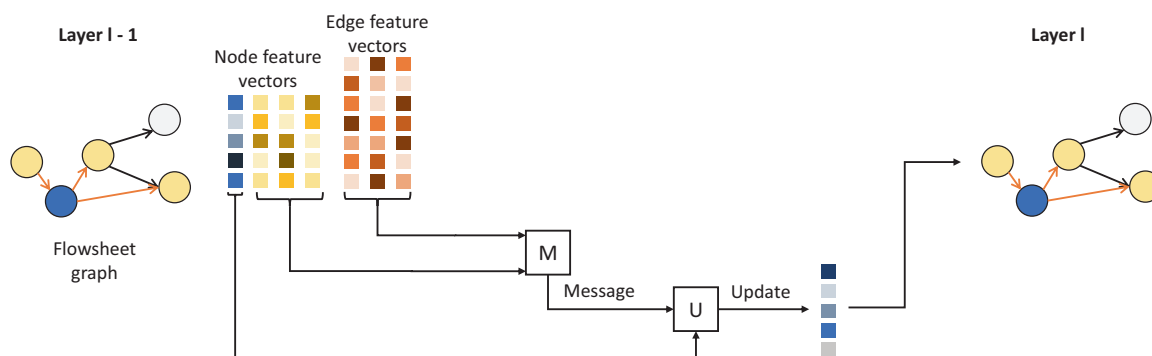
FIGURE 5 Update of the node embeddings during the message passing phase in a graph convolutional layer. The considered node is marked in blue and its neighbors in yellow. First, the information stored in the neighboring nodes and the respective edges is processed and combined through a message function M. Then, a message is generated to update the information embedded in the considered node through the update function U. The approach and its illustration follow a method proposed by Schweidtmann et al.[29]

The second level actor consists of a MLP that returns probabilities for each unit operation to be chosen. For each type of unit operation, an individual MLP is set up as the actor for the third level decision. Thereby, the third level MLPs take the concatenated vector including the flowsheet fingerprint and the ID of the selected location as an input. They return two outputs which are interpreted as parameters, α and β, describing a beta distribution $B(\alpha, \beta)$.[53] Based on this distribution, a continuous decision regarding the respective design variable is made.
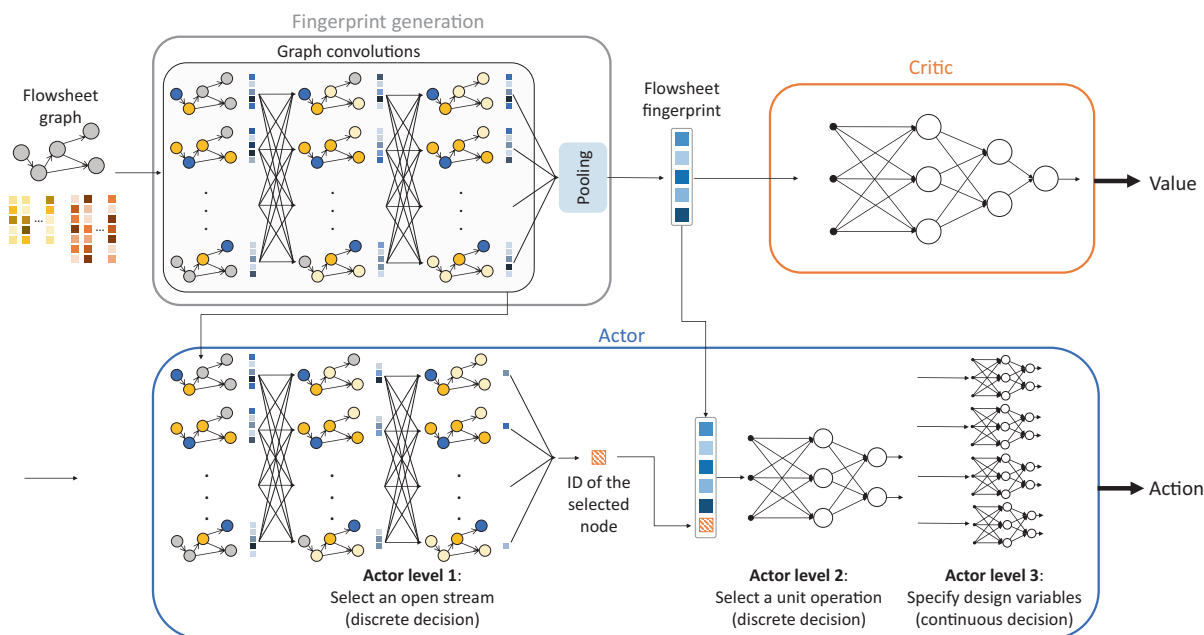
The critic that estimates the value of the original state is displayed in the upper half of Figure 6. Therefore, the flowsheet fingerprint is passed through another MLP. This value is an estimation of how much reward is expected to be received by the agent until the end of an episode when starting at the considered state and further following current policy.[9] In our approach, we utilize the value to compute the generalized advantage estimation $\hat{A}$ introduced by Schulman et al.[52] It tells whether an action performed better or worse than expected and is used to calculate losses of the actor's networks. By comparing the value to the actual rewards, an additional loss is computed for the critic.

2.3 | Agent–environment interaction

The interaction between the environment and the hierarchical actor-critic agent is further clarified in Algorithm 1. After the environment is initialized with a feed, the flowsheet is generated in an iterative scheme. The agent first observes the current state s of the environment and chooses actions a for all three hierarchical decision levels by sampling. The agent returns the probabilities and the selected actions as well as the value v of the state.

In the next step, the actions are applied to the environment. Therefore, the next state s' is computed by simulating the extended flowsheet. Additionally, the environment checks whether any open stream is left in the flowsheet, indicating that the episode is still to be completed. Since the weights of the agent's networks are randomly initialized, early training episodes can result in very large flowsheets. Thus, the total number of units is limited to 25 as additional guidance. If a flowsheet exceeds this number, all open streams are declared as products.

Additionally, the environment calculates the reward that depends on whether the flowsheet is completed or not. If the net cash flow is positive, the reward equals the net cash flow. If the net cash flow is

FIGURE 6 Architecture of the deployed actor-critic agent. First, a GNN is used to process the graph representation of the flowsheet into a flowsheet fingerprint. While the critic estimates the value of the fingerprint in one linear MLP, the actor takes three levels of decisions. The first decision is to choose a location for expanding the flowsheet. Practically, this means selecting the ID of a node representing an open stream. The selected node ID is combined with the flowsheet fingerprint and passed through an MLP for the second level decision of choosing a type of unit operation. Finally, a continuous design variable of the unit is chosen. Thereby, a different MLP is used for each unit type.

negative, the reward equals the net cash flow divided by a factor 10. This procedure is implemented in order to encourage exploration of the agent. For the intermediate steps during the synthesis, process rewards of zero are given to the agent. After each iteration, the transition is stored in a batch and later used for batch learning. Thereby, the states in the transition tuples store the full flowsheet graphs using the Deep Graph Library.[54]

2.4 | Training

The presented method, including the flowsheet simulations, is implemented in Python 3.9. The training procedure is adapted from PPO by OpenAI.[46] It consists of multiple epochs of minibatch updates, whereby the minibatches result from sampling on the transition tuples stored in the memory. The agent's networks are thereby updated by gradient descent, using a loss function derived from summing up and weighting all losses of the individual actors, their entropies, and the loss of the critic.

2.5 | Case study

The proposed method is demonstrated in an illustrative case study considering the production of methyl acetate (MeOAc), a low-boiling liquid often used as a solvent.[55] In an industrial setting, MeOAc is primarily produced in reactive columns by esterification of acetic acid (HOAc).[56,57] For illustration, we consider only simplified flowsheets that use separate units for reaction and separation.

2.6 | Process simulation

For computing new states and rewards, the flowsheets generated by the agent are simulated in Python. Therefore, we implemented a model for each type of unit operation that can be selected in the second level decision. In our case study, the agent can decide to place reactors, distillation columns, and heat exchangers. Furthermore, the agent can add recycles or sell open streams as products.

2.6.1 | Reactor

The reactor is modeled as a plug flow reactor (PFR), in which the reversible equilibrium reaction shown in Equation (1) takes place.

$$HOAc + MeOH \rightleftharpoons MeOAc + H_2O \qquad (1)$$
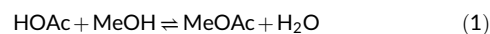
MeOAc and its by-product water (H_2O) are produced by esterification of HOAc with methanol (MeOH) under the presence of a strong acid. To calculate the composition of the process stream leaving the PFR, we formulated a boundary value problem, depending on the reaction rate, and manually implemented a fourth-order Runge–Kutta method with fixed step-size as solver. Thereby, the reactor is modeled isothermal, based on the temperature of the inflowing stream. The reaction kinetics are based on Xu and Chuang.[58]

The length of the PFR is specified by the agent as the continuous third level decision within the range of 0.05–20 m. Thereby, the relation

ALGORITHM 1   Pseudocode of the agent–environment interaction

```
done = False
while not done do
    observe state s
    actions a, probs p, value v = AGENT(s)
    new state s′, reward r, done = ENV(a)
    store transition (s, a, p, r, done) in memory
end while

function AGENT(state s)
    for level = 1,2,3 do
        probs p_level = actor(s)
        action a_level = sample(p_level)
    end for
    value v = critic(s)
    return a, p, v
end function

function ENV(actions a)
    next state s′ = SimulateFlowsheet(a)
    if no more open streams then
        done = True
        reward r = NetCashFlow(s′)
        if reward r < 0 then
            reward r = reward r/10
        end if
    else
        reward r = 0 €
    end if
    return s′, r, done
end function
```
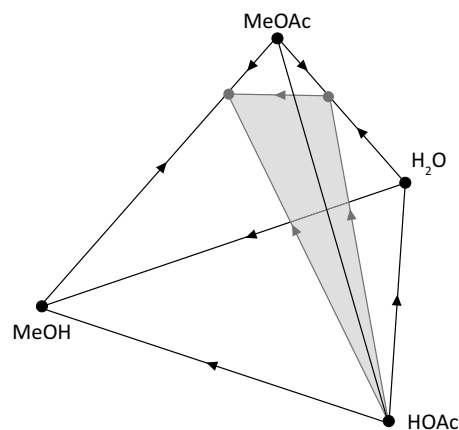

FIGURE 7   Vapor–liquid-equilibrium in the quarternary system consisting of MeOAc, HOAc, H_2O, and MeOH at 1 bar. The gray surface marks the distillation boundary spanned by the two azeotropic points and the fourth component HOAc, splitting the diagram into two distillation regimes.

stream is cooled or heated. To avoid evaporation of the process stream, the inlet water temperature is chosen within the range of 278.15–326.95 K, where the upper limit corresponds to the lowest possible boiling point of the considered quarternary system. The heat exchanger model computes the heat duty, the required heat transfer area, and the outlet temperature of the process stream. The model is based on a countercurrent flow, shell, and tube heat exchanger.[59] A typical heat transfer coefficient of 568 W K^{-1} m^2 is used.[60] Additionally, we assume that the process stream always approaches the water stream temperature within 5 K in the heat exchanger.

2.6.3 | Distillation column

The distillation column is deployed to separate the quarternary system MeOAc, MeOH, HOAc, and H_2O. The vapor–liquid equilibrium of the system is displayed in Figure 7. It contains two binary minimum azeotropes between MeOAc and H_2O, and respectively between MeOAc and MeOH. As shown in Figure 7, the azeotropes split up the separation task into two distillation regimes. To simplify the problem, we follow the assumption made by Göttl et al.[37] that the distillation boundary can be approximated by the simplex spanned between both azeotropes and the fourth component, HOAc.

We implemented a shortcut column model using the ∞/∞ analysis.[61–63] The only remaining degree of freedom in the ∞/∞ model is the distillate to feed ratio D/F. It is set by the agent in the continuous third level decision within a range of 0.05–0.95.

2.6.4 | Recycle

The agent can also select to recycle an open process stream back to the feed stream. Thereby, the ratio of the considered stream that will

of the cross-sectional area A of the PFR to the molar flow $\dot{N}$ passing through it is fixed to $A/\dot{N} = 0.1 m^2 smol^{-1}$. Notably, the length of the reactor significantly influences the conversion in the PFR. In addition, the equilibrium of the considered reaction depends on the temperature of the process stream, which thus affects the reaction rate and the conversion in the PFR. Thereby, the temperature of the process stream can be influenced by heat exchangers upstream of the reactor.

2.6.2 | Heat exchanger

In the heat exchanger, heat is transferred between the process stream and a water stream. The continuous third level decision specifies the inlet temperature of the water and thus also whether the process

be recycled is selected by the agent in the third level decision. For practical reasons, the recycled ratio must always lie between 0.1 and 0.9. The recycle is modeled by adding a splitting unit and a mixing unit to the flowsheet. First, the considered stream is split up in a recycle stream and a purge stream. The latter one ends in a new "undefined" node. To simulate the recycle, a tear stream is initialized. Then, the Wegstein method[64] is used to solve the recycle stream flow rate iteratively. When the Wegstein method is converged, the tear stream is closed and the recycle stream is fed into the feed stream by the mixing unit. This method is based on the implementation of flexsolve.[65]

2.7 | Reward

The reward assesses the economic viability of the generated process, following Seider et al.[60] for calculating annualized cost and Smith[59] for estimating unit capital costs. After completing a flowsheet by specifying all open streams as products, the agent receives a final reward. This final reward r represents an approximate net cash flow of the process within one year. If this net cash flow is negative, it is reduced by a factor 10 to encourage exploration of the agent. The economic value of incomplete flowsheets is more difficult to estimate because it may depend on future actions. Thus, a reward of zero is given after every single action since the actual value of an action can only be assessed when an episode is complete. As shown in Equation (2), the final reward includes costs for units and feeds as well as revenue for sold products.

$$r = \sum P_{products} - \sum C_{feed} - \sum (U + 0.15 \cdot I)_{units} \qquad (2)$$

The values of the products are estimated by an s-shaped price function P, depending on the purity of the considered streams. The pure component price C is used to compute the cost of the raw material stream. The annualized cost is computed by adding the annual utility costs U and the total capital investment I multiplied by a factor 0.15.[60] Furthermore, the reward is used to teach the agent to make feasible decisions. Whenever infeasible actions are selected that cause the simulation to fail, for example, if the reactor simulation fails due to bad initial values in the solver, the episode is interrupted immediately and a negative reward of -10 Mio € is given. When the agent decides to not add units at all and just sell the feed streams, the same penalty is given to prevent the agent from falling into this trivial local optimum.

Notably, the considered case study is meant to facilitate illustration and the considered parameter values for prices are only approximations.

3 | RESULTS AND DISCUSSION

In this section, we present and analyze the learning behavior of the developed agent. For investigating all single parts of the agent, the training procedure was first conducted in a discrete action space,

consisting of the first and second hierarchical decision levels. Afterward, the same procedure was conducted in a continuous action space which only includes the third decision level. Finally, all decision levels are combined to the hybrid action space. In all runs, the environment was initialized with a feed consisting of an equimolar binary mixture of MeOH and HOAc. The feed's molar flow rate was set to 100 mol s^{-1} and its temperature to 300 K.

The proposed learning process and the agent architecture include several hyperparameters that are listed in Table S1. The selected hyperparameters are based on literature 29,30,46,66.

3.1 | Flowsheet generation in a discrete action space

To investigate the agent's behavior in a discrete action space, the third level actor was deactivated and only the first and second level decisions were conducted. Thus, in each step, the agent selected a location for a new unit operation as well as its type. Thereby, fixed values for the unit's continuous design variables were used. They are displayed in Table 1.

Throughout the presented case study, constant pressure of 1 bar was assumed. The agent was trained 20 times in 10,000 episodes each, with the procedure described previously.

Figure 8 shows the learning curve of the agent in the discrete action space. The plotted curve represents the mean learning curve of all 20 single runs and the gray area displays the standard deviation. Thereby the curves are smoothed by taking the average score over 50 episodes.

The displayed scores correspond to the reward, which is the estimated net cash flow of the final process. Thus, they are a measure of the economic viability of the final process.

During the first 2000 episodes, the learning curve rises steeply. In this early training stage, the agent produces predominantly long flowsheets and often reaches the maximum allowed number of unit operations. However, throughout the training the agent learns that shorter flowsheets are economically more valuable. Soon, the agent mainly produces flowsheets with a positive score, meaning that the final process is economically viable. After approximately 4000 episodes, the learning curve converges to a mean score of approximately 22.

The best flowsheet the agent generated throughout the 20 training runs is displayed in Figure 9. The depicted process first uses a reactor (R1) to produce MeOAc and its side product H_2O from the feed (F1). Then, the resulting quarternary mixture is split up in two distillation columns. The distillate (P1) of the first column (C1) is enriched with MeOAc but also includes MeOH and H_2O. The bottom product of the first column is further split up in a second column (C2) to produce a mixture of H_2O and HOAc in the distillate (P2) and pure HOAc in the third product stream (P3). Ninety percent of the latter product is recycled and mixed with the feed stream. During the training, the agent learned, for example, that heat exchangers do not add value to the flowsheet. This flowsheet scored a reward of 39.86.

**TABLE 1** Fixed continuous design variables for each unit type during the training in a discrete action space

| Unit operation	Design variable	Symbol	Unit	Fixed value
Heat exchanger	Water inlet temperature	T_{water}^{in}	K	305
Reactor	Reactor length	l	m	10
Column	Distillate to feed ratio	D/F	—	0.5
Recycle	Recycling ratio	—	—	0.9

Note: This selection replaces the third level decision.
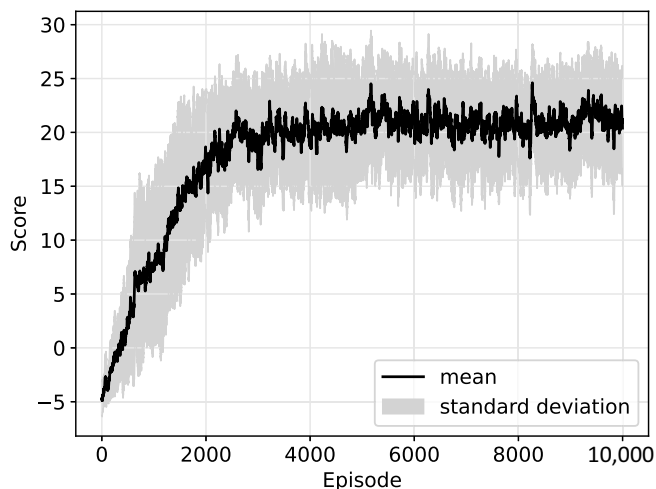
FIGURE 8 Learning curve of the agent in a discrete action space. The mean learning curve and its standard deviation from 20 training runs over 10,000 episodes each are displayed. The learning curve shows the scores of the generated flowsheets, averaged over 50 episodes. The score of each episode corresponds to the reward, which is the estimated net cash flow. An episode is a sequence of actions to generate a flowsheet, starting with a feed.
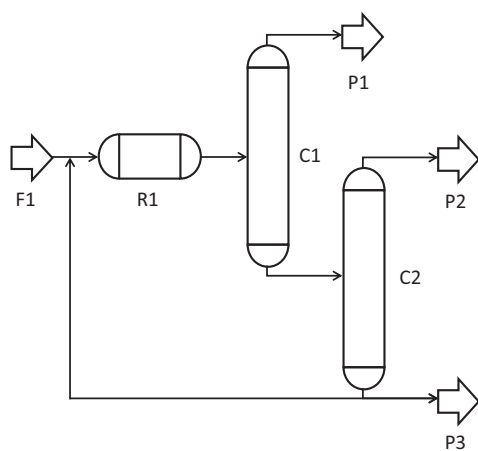
FIGURE 9 Best flowsheet generated by the agent in a discrete action space within 20 training runs of 10,000 episodes each. In a reactor (R1), MeOAc, and its side product H_2O are produced from the feed (F1). Then, the resulting quarternary mixture is split up in two columns (C1 and C2). Parts of the third product stream (P3) are recycled and mixed with the feed stream.
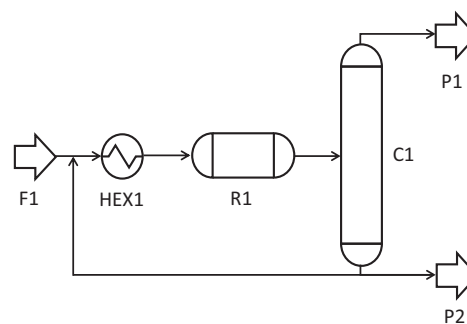
FIGURE 10 Fixed flowsheet structure during the training in a continuous action space. It consists of a heat exchanger (HEX1), a reactor (R1), and a column (C1). The bottom product (P2) is split up and partially recycled.

All 20 individual training runs found similar best flowsheets, with the mean best flowsheet scoring a reward of 39.85.

3.2 | Flowsheet generation in a continuous action space

The third level actor was investigated by deactivating the first and second level actors and thus only including continuous decisions. Therefore, the sequence of unit operations in the flowsheet was fixed, as shown in Figure 10, and only the continuous design variables defining each unit were selected by the agent. Within this structure, the agent was trained for 10,000 episodes each in 20 runs. Similar to the findings in the discrete action space, the agent learns quickly at the beginning of the training. After a steep increase, the policy starts to converge to a score of approximately 43 and is almost constant after 5000 episodes. The mean learning curve of the continuous agent and its standard deviation are displayed in Figure 11, showing the scores of the flowsheets, smoothed by taking the average over 50 episodes.

The best flowsheet the agent found throughout all 20 training runs scored a reward of 44.25. The mean best flowsheet of all 20 runs scored a slightly lower reward of 44.23. Generally, the continuous agent shows a very stable and reproducible learning behavior. However, the considered continuous action space problem is rather simple and could also be solved with established optimization algorithms. To assess the performance of the RL method in the continuous variable space, the problem was reformulated as an optimization problem with

four variables and solved using the "optimize" library from SciPy.[67] First, the problem was analyzed using the local optimizer "minimize" and the method by Broyden, Fletcher, Goldfarb, and Shanno (BFGS), which is a quasi-Newton method with good performance for non-smooth optimizations.[68] Since the considered problem contains multiple local optima, the results from the BFGS optimization highly depend on the initial values. As the RL agent does not require any initial values but chooses the first investigated variables randomly, a similar procedure was chosen for the BFGS method. The optimization was conducted 20 times, using random initial values within the boundaries of the considered design variables. The mean optimal reward and the standard deviation of the BFGS method and the RL agent are compared in Table 2. The local optimizer shows a poorer and less reproducible performance than RL. Even though the best of the 20 optimization runs matched the optimal reward of 44.25 found

by RL, the mean optimal score of the BFGS method was significantly lower with a reward of 38.29. Table 2 also shows a much higher standard deviation of the BFGS method. These results elucidate the high dependency of the local optimizer on the initial values and the requirement for global optimization strategies. Thus, the problem was also optimized using the global optimization algorithm dual annealing (DA) by SciPy[67] to generate a benchmark for the considered continuous task. DA is derived from the generalized simulated annealing algorithm by Xiang et al.[69] and combines a stochastic global optimization algorithm with local search. Analogously to the RL and BFGS methods, the optimization with DA was conducted 20 times. In each run, the number of function evaluations was limited to 10,000 to ensure comparability to the RL agent. The mean optimum and the standard deviation of the DA method is also displayed in Table 2 and compared with the results from BFGS and RL. The best optimum found in the 20 optimization runs scored a reward of 44.27, thereby exceeding the best reward from the RL agent by 0.045%. The mean optimum from all 20 runs also marginally exceeds the RL agent with a score of 44.25. Even though the DA optimization slightly outperforms RL, Table 2 shows that the deviations between RL and DA are almost neglectable whereas both clearly outperform the local optimization with BFGS. While optimization with DA led to similar results in the presented continuous variable space, neither DA nor BFGS can cope with the discrete or hybrid decision tasks of the RL agent since this would require a complex reformulation of the problem to a superstructure optimization task.

Table 3 lists the continuous design variables of the best flowsheet the RL agent observed throughout the training runs and compares them with the optimum found by DA, which can be assumed to be the global optimum of the continuous action space problem. The comparison shows only slight deviations in the design variables. In the heat exchanger (HEX1), the feed is only slightly heated before entering the reactor for both methods. With a length of 7.33 and 7.88 m, respectively, the reactor (R1) is relatively short compared to the allowed length range of 0.05 m to 20 m. A shorter reactor means a lower conversion but also lower costs. The column (C1) is characterized by the distillate to feed ratio D/F of 0.58. As a result, MeOAc is enriched in the distillate which also contains MeOH and H_2O. The bottom product is a mixture of MeOH and HOAc. In the investigated flowsheet shown in Figure 10, the bottom product is partially recycled to the feed. Remarkably, the recycled ratio is set to the lower boundary value of 0.1 by both, RL and DA. These results indicate that the recycle does not add significant value to the illustrative flowsheet used for this study.

FIGURE 11 Learning curve of the agent in a continuous action space. Analogously to Figure 8, the mean learning curve and its standard deviation from 20 training runs over 10,000 episodes each are displayed. It shows the scores of the generated flowsheets, averaged over 50 episodes.

TABLE 2 Mean and standard deviations of the optimal scores found by the RL agent and optimization with BFGS and DA

	RL	BFGS	DA
Mean optimum	44.23	38.29	44.25
Standard deviation in %	0.034	38.58	0.021

Note: For all methods, the optimization was conducted 20 times with a maximal episode number of 10,000.

| Unit operation	Design variable	Symbol	Unit	RL	DA
Heat exchanger (H1)	Water inlet temperature	T^{in}_{water}	K	308.8	305.0
Reactor (R1)	Reactor length	l	m	7.33	7.88
Column (C1)	Distillate to feed ratio	D/F	—	0.58	0.58
Recycle	Recycled ratio	—	—	0.10	0.10

TABLE 3 Optimal continuous design variables found by the continuous RL agent and the global optimizer DA after 20 runs of 10,000 episodes each

3.3 | Flowsheet generation in a hybrid action space

After the previous sections have shown that all three actors are able to learn separately, they are combined hereinafter. Therefore, the
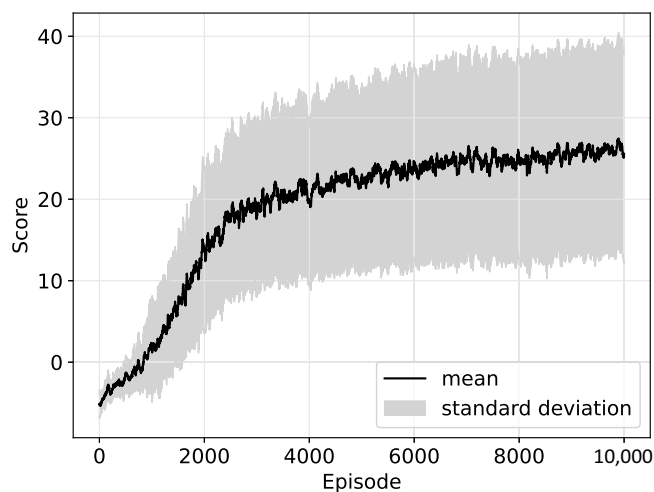
FIGURE 12 Learning curve of the agent in a hybrid action space. Analogously to Figures 8 and 11, the mean learning curve and its standard deviation from 20 training runs over 10,000 episodes each are displayed, showing the scores of the generated flowsheets, averaged over 50 episodes.
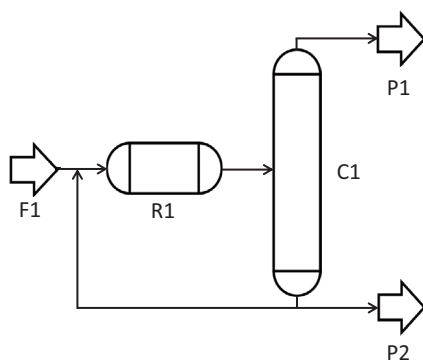
FIGURE 13 Best flowsheet generated by the agent in a hybrid action space within 20 training runs of 10,000 episodes each. First, MeOAc and its side product H_2O are produced from the feed (F1) in a reactor (R1). Then, the mixture is split up in a column (C1).The first product (P1) is enriched with MeOAc but also includes MeOH and residues of H_2O. The second product (P2) is a mixture of HOAc and MeOH. Thereby, 10% of P2 are recycled and mixed to the feed stream.

hybrid agent, combining all previously described elements, is trained 20 times for 10,000 episodes each.

The resulting mean learning curve of the 20 individual runs is displayed together with the standard deviation in Figure 12, showing the scores of the flowsheets generated during the training, smoothed by taking the average over 50 episodes.

Despite the complexity of the hybrid problem, the agent is learning fast and quickly produces flowsheets with a positive value. After a steep increase, the learning curve slowly converges to a score of approximately 27. As expected, the standard deviation is significantly larger compared to the solely discrete and continuous problems. Still, the agent converged toward positive scores in all single training runs, meaning that the generated flowsheets are economically viable. The best flowsheet the agent found during the training scored a reward of 44.67, which exceeds the scores of all flowsheets found during the solely discrete and continuous considerations. However, the mean optimum found by the hybrid agent in the 20 training runs is lower, with a score of 38.78.

The best flowsheet the agent observed during training is shown in Figure 13. The continuous design variables the agent selected for this best flowsheet are shown in Table 4.

The feed (F1) is fed directly into a reactor (R1) where MeOAc and H_2O are produced from esterification of HOAc with MeOH. With a length of 10.23 m, the reactor is larger compared to the best flowsheet generated with the continuous agent which results in a higher conversion but also higher costs. In the next step, the resulting quarternary mixture is split up in a column (C1). Thereby, the split ratio used by the hybrid agent corresponds to the results from the continuous agent. In the distillate of the column (P1), MeOAc is enriched but it also includes MeOH and residues of H_2O. The bottom product of the column (P2) contains HOAc and MeOH. Thereby, 10% of the bottom product are recycled and mixed back to the feed. Whereas 10% are the lower boundary of the split ratio in the recycle, the agent also had the option to not use a recycle at all. Thus, the agent found that the recycle does add value to the flowsheet, however only when small amounts are recycled. The sequence of unit operations found by the hybrid agent slightly differs from the best flowsheet generated by the discrete agent, were two columns were used. The desired product MeOAc is completely in the distillate and the bottom product consists of less valuable chemicals. Thus, the agent learnt that the second column does not add economic value.

TABLE 4 Continuous design variables selected by the hybrid agent in the best flowsheet observed during training

Unit operation	Design variable	Symbol	Unit	Best run
Reactor (R1)	Reactor length	l	m	10.23
Column (C1)	Distillate to feed ratio	D/F	—	0.58
Recycle	Recycled ratio	—	—	0.10

4 | DISCUSSION

Overall, the learning curves shown in the previous sections indicate that all parts of the agent learn quickly. A comparison with local and global optimization methods showed that the RL agent clearly outperforms local optimization in continuous action space tasks and almost reaches the performance of the global optimizer DA. However, in contrast to the optimizers, RL is also capable of solving discrete and hybrid tasks without further need for complex reformulation of the problem. Furthermore, it has been shown that the results from the RL agent are reproducible in all considered tasks. Still, the standard deviations are becoming larger when considering the complex task of finding an optimal flowsheet in a hybrid action space. It is assumed that the learning behavior is not yet optimal since the hyperparameters have not been optimized for this first fundamental study. In future works, it is advised to conduct an extensive hyperparameter study to investigate their influence on the learning behavior.

Compared to other approaches, the main contribution of the presented method is the representation of flowsheets as graphs and combining GNNs with RL. GNNs have already shown promising performance in various deep learning tasks.[42] One of their key advantage is that they are able to process the topological information of the graphs.[43] Since the structural information about flowsheets is automatically captured in the graph format, GNNs can take advantage of this structure. Deriving fingerprints from graphs with GNNs has already shown promising results in the molecule field.[29,70,71] Here, we transfer the methodology to the flowsheet domain. During the implementation and analysis of the training procedure, the graph presentation of the flowsheets has proven to be handy. The graphs generated by the agent can be visualized easily and thus immediately give an insight into the process and its meaningfulness. An additional advantage of the approach is its flexibility. Through its hierarchical structure, the different components of the agent can be easily decoupled and new parts can be added. By using a separate MLP for each unit operation in the third level decision, the number of the continuous decisions can vary for the different unit operations. In the presented work, only one continuous decision is made for each unit operation but the agent architecture allows including more decisions within this step. By allowing for more unit operations and setting more design variables, the action space and thus the complexity of the problem should be increased for future investigations.

Furthermore, the reward function will require additional attention. Giving rewards is not straightforward in the considered problem since it is hard to assess the value of an intermediate flowsheet. Still, it is crucial for the performance of the RL algorithm. In the presented work, the reward function is only an estimation of economic assessments that neglects multiple cost factors in real processes. However, for future developments, investigating ways of reward shaping[72] will be an interesting aspect that can stabilize the training process especially when the size of the considered problem gets larger.

5 | CONCLUSION

We propose the first RL agent that learns from flowsheet graphs using GNNs to synthesize new processes. The deployed RL agent is hierarchical and hybrid meaning it takes multiple dependent discrete and continuous decisions within one step. In the proposed methodology, the agent first selects a location in an existing flowsheet and a unit operation to extend the flowsheet at the selected position. Both selections are discrete. Then, it takes a continuous decision by selecting a design variable that defines the unit operation. Naturally, each sub-decision strongly depends on the previous one. Thereby, flowsheets are represented as graphs, which allows us to utilize GNNs within the RL structure. As a result, our methodology generates economical valuable flowsheets only based on experience of the RL agent.

In an illustrative case study considering the production of methyl acetate, the approach shows steep, mostly stable, and reproducible learning in discrete, continuous, and hybrid action spaces. Furthermore, a comparison with established optimization algorithms for the exclusively continuous action space was conducted. It was shown that RL outperforms local optimization with BFGS and almost matches the results from global optimization with DA. However, in contrast to the optimization algorithms, RL is applicable to the discrete and hybrid action spaces without need for any problem reformulation. This work is a fundamental study that demonstrates that graph-based RL is able to create meaningful flowsheets. Thus, it encourages to incorporate AI in chemical process design.

A further advantage of the presented approach is that the proposed architecture is a good foundation for further developments like enhancing the state-action space. Thus, the selected structure of the agent is predestined for increasing the complexity and solving more advanced problems in the future. A subsequent step following this paper should be to implement an interface to an advanced process simulator. This will tremendously increase the complexity of the problem but also allow for easier extension of the action space and more rigorous simulations.

AUTHOR CONTRIBUTIONS

Laura Stops: Investigation (equal); methodology (equal); software (equal); validation (equal); visualization (equal); writing – original draft (equal). **Roel Leenhouts:** Investigation (equal); methodology (equal); software (equal); validation (equal); visualization (equal); writing – review and editing (equal). **Qinghe Gao:** Investigation (supporting); methodology (equal); project administration (supporting); software (equal); supervision (supporting); validation (supporting); writing – original draft (supporting); writing – review and editing (supporting). **Artur M. Schweidtmann:** Conceptualization (lead); formal analysis (equal); funding acquisition (lead); investigation (supporting); methodology (equal); project administration (lead); supervision (lead); validation (equal); writing – review and editing (lead).

DATA AVAILABILITY STATEMENT

Research data are not shared.

ORCID

Laura Stops https://orcid.org/0000-0001-8827-5145
Roel Leenhouts https://orcid.org/0000-0002-9873-8657
Qinghe Gao https://orcid.org/0000-0002-2350-4776
Artur M. Schweidtmann https://orcid.org/0000-0001-8885-6847

REFERENCES

1. Fantke P, Cinquemani C, Yaseneva P, et al. Transition to sustainable chemistry through digitalization. *Chem*. 2021;7(11):2866-2882. doi:10.1016/j.chempr.2021.09.012
2. Meramo-Hurtado SI, González-Delgado ÁD. Process synthesis, analysis, and optimization methodologies toward chemical process sustainability. *Ind Eng Chem Res*. 2021;60(11):4193-4217. doi:10.1021/acs.iecr.0c05456
3. Mohan S, Katakojwala R. The circular chemistry conceptual framework: a way forward to sustainability in industry 4.0. *Curr Opin Green Sustain Chem*. 2021;28:100434. doi:10.1016/j.cogsc.2020.100434
4. Martinez-Hernandez E. Trends in sustainable process design–from molecular to global scales. *Curr Opin Chem Eng*. 2017;17:35-41. doi:10.1016/j.coche.2017.05.005
5. Schweidtmann AM, Esche E, Fischer A, et al. Machine learning in chemical engineering: a perspective. *Chem Ing Tech*. 2021;93(12):2029-2039. doi:10.1002/cite.202100083
6. Mnih V, Kavukcuoglu K, Silver D, et al. Playing atari with deep reinforcement learning. arXiv Preprint arXiv: 1312.5602. 2013.
7. Kempka M, Wydmuch M, Runc G, Toczek J, Jaskowski W. ViZDoom: a doom-based AI research platform for visual reinforcement learning. 2016 IEEE Conference on Computational Intelligence and Games (CIG). IEEE, Santorini Island, Greece. 2016.
8. Silver D, Hubert T, Schrittwieser J, et al. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science*. 2018;362(6419):1140-1144. doi:10.1126/science.aar6404
9. Sutton RS, Barto A. *Reinforcement Learning: An Introduction*. 2nd ed. The MIT Press; 2018.
10. Silver D, Huang A, Maddison CJ, et al. Mastering the game of Go with deep neural networks and tree search. *Nature*. 2016;529(7587):484-489. doi:10.1038/nature16961
11. Vinyals O, Babuschkin I, Czarnecki WM, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*. 2019;575(7782):350-354. doi:10.1038/s41586-019-1724-z
12. Open AI, Berner C, Brockman G, et al. Dota 2 with large scale deep reinforcement learning. arXiv Preprint arXiv: 1912.06680. 2019.
13. Lee YH, Lee S. Deep reinforcement learning based scheduling within production plan in semiconductor fabrication. *Expert Syst Appl*. 2022;191:116222. doi:10.1016/j.eswa.2021.116222
14. Hubbs CD, Li C, Sahinidis NV, Grossmann IE, Wassick JM. A deep reinforcement learning approach for chemical production scheduling. *Comput Chem Eng*. 2020;141:106982. doi:10.1016/j.compchemeng.2020.106982
15. Hoskins JC, Himmelblau DM. Process control via artificial neural networks and reinforcement learning. *Comput Chem Eng*. 1992;16(4):241-251. doi:10.1016/0098-1354(92)80045-b
16. Spielberg SPK, Gopaluni RB, Loewen PD. Deep reinforcement learning approaches for process control. 2017 6th International Symposium on Advanced Control of Industrial Processes (AdCONIP). IEEE, Taipei, Taiwan. 2017.
17. Mowbray M, Petsagkourakis P, del Rio-Chanona EA, Zhang D. Safe chance constrained reinforcement learning for batch process control. *Comput Chem Eng*. 2022;157:107630. doi:10.1016/j.compchemeng.2021.107630
18. Sachio S, del Rio-Chanona EA, Petsagkourakis P. Simultaneous process design and control optimization using reinforcement learning. *IFAC-PapersOnLine*. 2021;54(3):510-515. doi:10.1016/j.ifacol.2021.08.293
19. Sachio S, Mowbray M, Papathanasiou MM, del Rio-Chanona EA, Petsagkourakis P. Integrating process design and control using reinforcement learning. *Chem Eng Res Des*. 2022;183:160-169. doi:10.1016/j.cherd.2021.10.032
20. Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning. *Nature*. 2015;518(7540):529-533. doi:10.1038/nature14236
21. Lillicrap TP, Hunt JJ, Pritzel A, et al. Continuous control with deep reinforcement learning. arXiv Preprint arXiv: 1509.02971. 2015.
22. You J, Liu B, Ying Z, Pande V, Leskovec J. Graph convolutional policy network for goal-directed molecular graph generation. In: Bengio S, Wallach H, Larochelle H, Grauman K, Cesa-Bianchi N, Garnett R, eds. *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc; 2018.
23. Popova M, Isayev O, Tropsha A. Deep reinforcement learning for de novo drug design. *Sci Adv*. 2018;4(7):eaap7885. doi:10.1126/sciadv.aap7885
24. Zhou Z, Kearnes S, Li L, Zare RN, Riley P. Optimization of molecules via deep reinforcement learning. *Sci Rep*. 2019;9(1):10752. doi:10.1038/s41598-019-47148-x
25. Cao ND, Kipf T. MolGAN: an implicit generative model for small molecular graphs. arXiv Preprint arXiv: 1805.11973. 2018.
26. Olivecrona M, Blaschke T, Engkvist O, Chen H. Molecular de-novo design through deep reinforcement learning. *J Chem*. 2017;9(1):48. doi:10.1186/s13321-017-0235-x
27. Pereira T, Abbasi M, Ribeiro B, Arrais JP. Diversity oriented deep reinforcement learning for targeted molecule generation. *J Chem*. 2021;13(1):21. doi:10.1186/s13321-021-00498-z
28. Jeon W, Kim D. Autonomous molecule generation using reinforcement learning and docking to develop potential novel inhibitors. *Sci Rep*. 2020;10(1):22104. doi:10.1038/s41598-020-78537-2
29. Schweidtmann AM, Rittig JG, König A, Grohe M, Mitsos A, Dahmen M. Graph neural networks for prediction of fuel ignition quality. *Energy Fuel*. 2020;34(9):11395-11407. doi:10.1021/acs.energyfuels.0c01533
30. Gilmer J, Schoenholz SS, Riley PF, Vinyals O, Dahl GE. Neural message passing for quantum chemistry. In: International conference on machine learning. PMLR. 2017.
31. Coley CW, Barzilay R, Green WH, Jaakkola TS, Jensen KF. Convolutional embedding of attributed molecular graphs for physical property prediction. *J Chem Inf Model*. 2017;57(8):1757-1772. doi:10.1021/acs.jcim.6b00601
32. Yang K, Swanson K, Jin W, et al. Analyzing learned molecular representations for property prediction. *J Chem Inf Model*. 2019;59(8):3370-3388. doi:10.1021/acs.jcim.9b00237
33. Ororbia ME, Warn GP. Design synthesis through a Markov decision process and reinforcement learning framework. *J Comput Inf Sci Eng*. 2022;22(2):021002. doi:10.1115/1.4051598
34. Midgley LI. Deep reinforcement learning for process synthesis. arXiv Preprint arXiv: 2009.13265. 2020.
35. Khan A, Lapkin A. Searching for optimal process routes: a reinforcement learning approach. *Comput Chem Eng*. 2020;141:107027. doi:10.1016/j.compchemeng.2020.107027
36. Göttl Q, Grimm DG, Burger J. Automated synthesis of steady-state continuous processes using reinforcement learning. *Front Chem Sci Eng*. 2022;16(2):288-302. doi:10.1007/s11705-021-2055-9
37. Göttl Q, Tönges Y, Grimm DG, Burger J. Automated flowsheet synthesis using hierarchical reinforcement learning: proof of concept. *Chem Ing Tech*. 2021;93(12):2010-2018. doi:10.1002/cite.202100086

38. Intemic. Flowsheet Copilot. https://intemic.com. 2022, accessed Nov. 2 , 2022.

39. Plathottam SJ, Richey B, Curry G, Cresko J, Iloeje CO. Solvent extraction process design using deep reinforcement learning. *J Adv Manuf Process*. 2021;3(2):3:e10079. doi:10.1002/amp2.10079

40. Khan AA, Lapkin AA. Designing the process designer: hierarchical reinforcement learning for optimisation-based process design. *Chem Eng Process: Process Intensif*. 2022;141:108885. doi:10.1016/j.cep.2022.108885

41. Bruna J, Zaremba W, Szlam A, LeCun Y. Spectral networks and locally connected networks on graphs. arXiv Preprint arXiv: 1312.6203. 2013.

42. Zhou J, Cui G, Hu S, et al. Graph neural networks: a review of methods and applications. *AI Open*. 2020;1:57-81. doi:10.1016/j.aiopen.2021.01.001

43. Bronstein MM, Bruna J, Cohen T, Veličković P. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. arXiv Preprint arXiv: 2104.13478. 2021.

44. Zhang T, Sahinidis NV, Siirola JJ. Pattern recognition in chemical process flowsheets. *AIChE J*. 2019;65(2):592-603. doi:10.1002/aic.16443

45. Friedler F, Aviso KB, Bertok B, Foo DCY, Tan RR. Prospects and challenges for chemical process synthesis with P-graph. *Curr Opin Chem Eng*. 2019;26:58-64. doi:10.1016/j.coche.2019.08.007

46. Schulman J, Wolski F, Dhariwal P, Radford A, Klimov O. Proximal policy optimization algorithms. arXiv Preprint arXiv: 1707.06347. 2017.

47. Mnih V, Badia AP, Mirza M, et al. Asynchronous methods for deep reinforcement learning. Balcan MF, Weinberger KQ, eds. Proceedings of the 33rd International Conference on Machine Learning. PMLR, New York, USA. 2016.

48. Fan Z, Su R, Zhang W, Yu Y. Hybrid actor-critic reinforcement learning in parameterized action space. arXiv Preprint arXiv: 1903.01344. 2019.

49. Fujimoto S, van Hoof H, Meger D. Addressing Function Approximation Error in Actor-Critic Methods. Dy J, Krause A, eds. Proceedings of the 35th International Conference on Machine Learning. PMLR, Stockholm, Sweden. 2018.

50. Haarnoja T, Zhou A, Hartikainen K, et al. Soft actor-critic algorithms and applications. arXiv Preprint arXiv: 1812.05905. 2018.

51. Williams RJ, Peng J. Function optimization using connectionist reinforcement learning algorithms. *Conn Sci*. 1991;3(3):241-268. doi:10.1080/09540099108946587

52. Schulman J, Moritz P, Levine S, Jordan M, Abbeel P. High-dimensional continuous control using generalized advantage estimation. arXiv Preprint arXiv: 1506.02438. 2015.

53. Gupta AK, Nadarajah S. *Handbook of Beta Distribution and Its Applications*. CRC Press; 2004.

54. Wang M, Zheng D, Ye Z, et al. Deep graph library: a graph-centric, highly-performant package for graph neural networks. arXiv Preprint arXiv: 1909.01315. 2019.

55. National Center for Biotechnology Information. PubChem compound summary for CID 6584, methyl acetate. Accessed January 8, 2022. https://pubchem.ncbi.nlm.nih.gov/compound/Methyl-acetate.

56. Huss RS, Chen F, Malone MF, Doherty MF. Reactive distillation for methyl acetate production. *Comput Chem Eng*. 2003;27(12):1855-1866. doi:10.1016/s0098-1354(03)00156-x

57. Agreda VH, Partin LR. Reactive distillation process for the production of methyl acetate. Patent, US4435595A; 1982.

58. Xu ZP, Chuang KT. Kinetics of acetic acid esterification over ion exchange catalysts. *Can J Chem Eng*. 1996;74(4):493-500. doi:10.1002/cjce.5450740409

59. Smith R. *Chemical Process: Design and Integration*. 2nd ed. Wiley; 2016.

60. Seider WD, Lewin DR, Seader JD, Widagdo S, Gani R, Ng KM. *Product and Process Design Principles: Synthesis, Analysis, and Evaluation*. 4th ed. Wiley; 2017.

61. Bekiaris N, Meski GA, Radu CM, Morari M. Multiple steady states in homogeneous azeotropic distillation. *Ind Eng Chem Res*. 1993;32(9):2023-2038. doi:10.1021/ie00021a026

62. Ryll O, Blagov S, Hasse H. ∞/∞-Analysis of homogeneous distillation processes. *Chem Eng Sci*. 2012;84:315-332. doi:10.1016/j.ces.2012.08.018

63. Burger J, Hasse H. Multi-objective optimization using reduced models in conceptual design of a fuel additive production process. *Chem Eng Sci*. 2013;99:118-126. doi:10.1016/j.ces.2013.05.049

64. Wegstein JH. Accelerating convergence of iterative processes. *Commun ACM*. 1958;1(6):9-13.

65. Cortes-Pena Y. flexsolve: flexible function solvers. https://github.com/yoelcortes/flexsolve. 2019, accessed Nov. 2 2022.

66. AurelianTactics. PPO hyperparameters and ranges. https://medium.com/aureliantactics/ppo-hyperparameters-and-ranges-6fc2d29bccbe. 2018, accessed Nov. 2, 2022.

67. Virtanen P, Gommers R, Oliphant TE, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. *Nat Methods*. 2020;17:261-272. doi:10.1038/s41592-019-0686-2

68. Nocedal J, Wright SJ. Numerical optimization. Scholars portal. 2006.

69. Xiang Y, Sun D, Fan W, Gong X. Generalized simulated annealing algorithm and its application to the Thomson model. *Phys Lett A*. 1997;233(3):216-220. doi:10.1016/S0375-9601(97)00474-X

70. Duvenaud DK, Maclaurin D, Iparraguirre J, et al. Convolutional networks on graphs for learning molecular fingerprints. In: Cortes C, Lawrence N, Lee D, Sugiyama M, Garnett R, eds. *Advances in Neural Information Processing Systems*. Curran Associates, Inc.; 2015:28.

71. Kearnes S, McCloskey K, Berndl M, Pande V, Riley P. Molecular graph convolutions: moving beyond fingerprints. *J Comput Aided Mol Des*. 2016;30(8):595-608. doi:10.1007/s10822-016-9938-8

72. Ng AY, Harada D, Russell S. Policy invariance under reward transformations: Theory and application to reward shaping. Proceedings of the Sixteenth International Conference on Machine Learning. Morgan Kaufmann, San Francisco, USA. 1999.

SUPPORTING INFORMATION

Additional supporting information can be found online in the Supporting Information section at the end of this article.