

Automated Flowsheet Synthesis via Reinforcement Learning and piece-wise linear thermodynamic Models

Quirin Göttl

Vollständiger Abdruck der vom TUM Campus Straubing für Biotechnologie und Nachhaltigkeit der Technischen Universität München zur Erlangung eines
Doktors der Ingenieurwissenschaften (Dr.-Ing.)
genehmigten Dissertation.

Vorsitz: Prof. Dr.-Ing. Michael Zavrel

Prüfer der Dissertation:

1. Prof. Dr.-Ing. Jakob Burger
2. Prof. Dr. Dominik Grimm
3. Prof. Dr.-Ing. Artur Schweidtmann

Die Dissertation wurde am 31.10.2023 bei der Technischen Universität München eingereicht und durch den TUM Campus Straubing für Biotechnologie und Nachhaltigkeit am 03.04.2024 angenommen.

Danksagung

Diese Dissertation entstand während meiner Tätigkeit an der Professur für Chemische und Thermische Verfahrenstechnik am Campus Straubing für Biotechnologie und Nachhaltigkeit der TU München. Von Juli 2019 bis August 2023 war ich dort als wissenschaftlicher Mitarbeiter angestellt. Hiermit möchte ich meine aufrichtige Dankbarkeit gegenüber all jenen zum Ausdruck bringen, die mich während dieser Zeit auf meinem akademischen Weg unterstützt haben.

Zunächst gebührt mein Dank meinem Doktorvater Prof. Dr.-Ing. Jakob Burger für die fachliche Anleitung und Unterstützung in den vergangenen Jahren. Ganz besonders möchte ich mich für Dein Vertrauen und Deine Risikobereitschaft bedanken, jemanden ohne verfahrenstechnische Grundkenntnisse einzustellen und damit diese Promotion erst zu ermöglichen. Aus meiner Sicht hat sich das Risiko auf jeden Fall gelohnt und ich kann mir vorstellen, dass Du das genauso siehst. Darüber hinaus möchte ich mich auch bei meinem Zweitbetreuer Prof. Dr. Dominik Grimm ganz herzlich für die Zusammenarbeit bedanken. Vielen Dank für die anregenden Diskussionen und letztendlich auch für die Chance, die erarbeiteten Methoden abseits von der chemischen Verfahrenstechnik einzusetzen. Aus fachlicher Sicht hat neben mir wahrscheinlich niemand mehr Bezug zum Thema dieser Arbeit als Jonathan Pirnay. Lieber Jonathan, vielen Dank für all die Diskussionen, die konstruktiven Gespräche, und für die Zusammenarbeit, die schlussendlich die Basis für den Erfolg dieser Arbeit bildete. Auch allen Kollegen, insbesondere dem CTV-Team, sei an dieser Stelle nochmal herzlichst gedankt für all die fachliche und vor allem auch moralische Unterstützung während der vergangenen Jahre.

Ein besonderer Dank geht an meine Familie, insbesondere an meine Eltern und Großeltern, für Eure bedingungslose Unterstützung und Euren unermüdlichen Glauben an mich während dieser Reise. Ihr habt mir bei den wichtigen Entscheidungen oft einen Schubs in die richtige Richtung gegeben und mich trotzdem zu nichts gezwungen. Vielen Dank, dass Ihr mir ein Studium meiner Wahl ermöglicht habt, ich glaube es war die richtige Entscheidung! Liebe Natalie, in den letzten beiden Jahren musste wahrscheinlich niemand mehr meiner Launen aushalten als Du. Vielen Dank, dass Du mich bei jedem Schritt unterstützt hast und immer für mich da warst. Ich liebe Dich. Mein Dank gilt auch meinen Freunden, die mich mit ihrem Rat, ihrer moralischen Unterstützung

und ihrer Freundschaft durch diese Zeit begleitet haben. Ihre Beiträge haben nicht nur meine Arbeit bereichert, sondern auch mein Leben.

Abschließend möchte ich mich bei all jenen bedanken, die bisher noch nicht namentlich genannt wurden und auf irgendeine Weise dazu beigetragen haben, dass mir meine Doktorandenzeit in guter Erinnerung bleibt und dass meine Promotion schlussendlich Realität geworden ist.

Siegenburg, im Mai 2024

Quirin Göttl

Abstract

Automated flowsheet synthesis is an important research field in computer-aided process engineering. The present work demonstrates how reinforcement learning, a machine learning sub-branch, can be used for automated flowsheet synthesis without any heuristics or prior knowledge of conceptual design. In reinforcement learning, the main goal is to teach an agent to master a predefined task by interacting with its environment. In the case of the present work, a steady-state flowsheet simulation serves as an environment where the agent can build up flowsheets that solve a given process problem. To set up the environment, we implement short-cut models that enable the simulation of unit operations and their underlying phase equilibria. For this purpose, the convex envelope method for constructing liquid phase equilibria is developed within this work. In this method, the composition space is discretized and the convex envelope of the Gibbs energy graph is computed. Employing the tangent plane criterion, all liquid phase equilibria can be determined robustly. For chemical systems with an arbitrary number of components, a mathematical framework is described and shown to work numerically with various examples from the literature of up to six components. On the reinforcement learning side, we stepwise present several frameworks that enable to train an agent to synthesize process flowsheets for tasks of increasing difficulty. First, a novel method named SynGameZero is developed to ensure good exploration schemes in the complex problem. Therein, flowsheet synthesis is modeled as a competitive two-player game. The agent, which consists of an artificial neural network and a tree search, is trained by playing this game against itself. The approach is shown to work for simple problems within a discrete action space. Second, the SynGameZero approach is extended by structuring the agent's actions in several hierarchy levels. This improves the approach's scalability and allows more sophisticated flowsheet problems to be considered. We successfully demonstrate the usability of the hierarchical SynGameZero approach through the fully automated synthesis of an ethyl-tert-butyl-ether process. Third, a single-player framework is derived, which combines several elements from the hierarchical SynGameZero approach and recently developed methodologies from the research field reinforcement learning. The agent is trained within a hybrid action space to set up processes for separating azeotropic mixtures out of several chemical systems.

Kurzfassung

Die automatisierte Fließbildsynthese ist ein wichtiges Forschungsgebiet in der computergestützten Verfahrenstechnik. In der vorliegenden Arbeit wird Reinforcement Learning, eine Subdisziplin des Machine Learning, für die automatisierte Fließbildsynthese eingesetzt. Hauptziel von Reinforcement Learning ist es, einem Agenten eine vordefinierte Aufgabe durch Interaktion mit seiner Umgebung beizubringen. In unserem Fall besteht die Umgebung aus einem Fließbildsimulator, in welchem der Agent Prozessschemata erstellen kann. Im Rahmen dieser Arbeit implementieren wir verschiedene Short-Cut-Methoden zur Simulation der Apparatemodelle und der zugrundeliegenden Phasengleichgewichte innerhalb des Fließbildsimulators. Zusätzlich wird eine Methodik zur Konstruktion von Flüssigphasengleichgewichten entwickelt. Hierbei wird der Kompositionsraum diskretisiert und die konvexe Hülle des Graphs der Gibbs-Energie berechnet. Mithilfe des Tangentenkriteriums können anschließend alle Flüssigphasengleichgewichte bestimmt werden. Die Gültigkeit der Methodik wird für Systeme mit einer beliebigen Anzahl an Komponenten mathematisch bewiesen. Zudem werden quantitative Ergebnisse für verschiedene Beispielsysteme mit bis zu sechs Komponenten präsentiert. Nachfolgend entwickeln wir schrittweise Methodiken im Bereich Reinforcement Learning, die es ermöglichen, einen Agenten ohne Vorkenntnisse zu trainieren, Prozesse für Probleme mit steigendem Schwierigkeitsgrad zu synthetisieren. Zunächst wird eine Methode namens SynGameZero vorgestellt, welche Fließbildsynthese als kompetitives Zwei-Spieler Spiel modelliert. Der Agent, bestehend aus einem künstlichen neuronalen Netz und einer Baumsuche, wird trainiert, indem er dieses Spiel gegen sich selbst spielt. Es wird gezeigt, dass der Ansatz für einfache Probleme innerhalb eines diskreten Aktionsraums funktioniert. Anschließend wird der SynGameZero-Ansatz erweitert, indem die Aktionen des Agenten in mehreren Hierarchieebenen strukturiert werden. Dies verbessert die Skalierbarkeit der Methodik und ermöglicht die Lösung komplexerer Probleme. Dies wird am Beispiel der vollautomatisierten Synthese eines Prozesses für die Herstellung von Ethyl-tert-Butylether gezeigt. Abschließend wird ein Ein-Spieler Ansatz beschrieben, der mehrere Elemente des hierarchischen SynGameZero-Ansatzes und kürzlich entwickelte Methoden aus dem Bereich Reinforcement Learning kombiniert. Hierbei wird der Agent in einem hybriden Aktionsraum trainiert, um Prozesse zur Trennung von azeo-

tropen Gemischen aus verschiedenen Beispielsystemen zu synthetisieren.

Declaration of Authorship

This dissertation contains material that has been published previously or that is included in submitted publications. In the following, these publications are listed together with a statement on the contributions of the author of the present dissertation.

- Q. Göttl, D.G. Grimm, and J. Burger. 2021. Automated process synthesis using reinforcement learning. *Comput. Aided Chem. Eng.*, 50, 209–214.

The author set up the implementation, carried out the experiments, and evaluated the results. The author developed the model and the algorithmic framework. The author wrote the manuscript. D.G. Grimm and J. Burger supervised the project. Material of that publication appears partly in Sections 1, 2, 3.2.2, 3.2.3, 4.2, 4.3, and 5 of this dissertation.

- Q. Göttl, Y. Tönges, D.G. Grimm, and J. Burger. 2021. Automated flowsheet synthesis using hierarchical reinforcement learning: proof of concept. *Chem. Ing. Tech.*, 93, 12, 2010–2018.

The author set up the implementation, carried out the experiments, and evaluated the results. The author developed the model and the algorithmic framework. The author wrote the manuscript. The author developed together with Y. Tönges the utilized model for cost estimation of flowsheets. D.G. Grimm and J. Burger supervised the project. Material of that publication appears partly in Sections 1, 2, 3.2.3, 4.3, and 5 of this dissertation.

- Q. Göttl, D.G. Grimm, and J. Burger. 2022. Automated synthesis of steady-state continuous processes using reinforcement learning. *Front. Chem. Sci. Eng.*, 16, 288–302.

The author set up the implementation, carried out the experiments, and evaluated the results. The author developed the model and the algorithmic framework. The author wrote the manuscript. D.G. Grimm and J. Burger supervised the project. Material of that publication appears partly in Sections 1, 2, 3.2.2, 4.2, and 5 of this dissertation.

- Q. Göttl, D.G. Grimm, and J. Burger. 2022. Using reinforcement learning in a game-like setup for automated process synthesis without prior process knowledge. *Comput. Aided Chem. Eng.*, 49, 1555–1560.

The author set up the implementation, carried out the experiments, and evaluated the results. The author developed the model and the algorithmic framework. The author wrote the manuscript. D.G. Grimm and J. Burger supervised the project. Material of that publication appears partly in Sections 1, 2, 3.2.3, 4.3, and 5 of this dissertation.

- Q. Göttl, J. Pirnay, D.G. Grimm, and J. Burger. 2023. Convex envelope method for determining liquid multi-phase equilibria in systems with arbitrary number of components. *Comput. Chem. Eng.*, 177, 108321.

The author set up the implementation, carried out the experiments, and evaluated the results. The author developed the methodology. The author wrote the manuscript. J. Pirnay revised the implementation and the mathematical framework. D.G. Grimm and J. Burger supervised the project. Material of that publication appears partly in Sections 1, 2, 3.1.2, 4.1, and 5 of this dissertation.

- Q. Göttl, J. Pirnay, J. Burger, and D.G. Grimm. 2023. Deep reinforcement learning uncovers processes for separating azeotropic mixtures without prior knowledge. *Currently under review for publication*. Preprint available under: <https://doi.org/10.48550/arXiv.2310.06415>.

That publication is a result of a shared first-authorship between the author of the present work, Q. Göttl, and J. Pirnay. Both contributed equally to the implementation, the development of the methodology, and the manuscript. D.G. Grimm and J. Burger supervised the project. Material of that publication appears partly in Sections 1, 2, 3.2.4, 4.4, and 5 of this dissertation.

Additionally, the author of the present dissertation published other research during his doctorate that is not part of the present dissertation and listed in the following:

- D. Vasiliu, Q. Göttl, S. Bröcker, and J. Burger. 2021. Multiple Solutions When Fitting Excess Gibbs Energy Models and Implications for Process Simulation. *Chem. Ing. Tech.*, 93, 3, 490–496.
- J. Pirnay, Q. Göttl, J. Burger, and D.G. Grimm. 2023. Policy-based self-competition for planning problems. *Eleventh International Conference on Learning Representations (ICLR)*.

Contents

List of Figures	XIII
List of Tables	XVII
1 Introduction	1
1.1 Automated Flowsheet Synthesis: Problem Description and Classification of Approaches	1
1.2 Machine Learning and Artificial Intelligence in Process Systems Engineering	2
1.3 Reinforcement Learning for Automated Flowsheet Synthesis	2
2 Motivation	5
2.1 Thesis	5
2.2 General Reinforcement Learning Framework and Challenges	6
3 Methodology	9
3.1 Environment: Process Simulation	9
3.1.1 General Remarks and Available Unit Operations	9
3.1.2 Modeling of Decanters: Convex Envelope Method	12
3.1.2.1 General Idea	12
3.1.2.2 Mathematical Framework	15
3.2 Reinforcement Learning Frameworks	22
3.2.1 General Remarks	22
3.2.2 SynGameZero: Proof of Concept	23
3.2.2.1 General Idea	23
3.2.2.2 Agent	24
3.2.3 SynGameZero: Integration of Hierarchical Reinforcement Learning	30
3.2.3.1 General Idea	30
3.2.3.2 Agent	31
3.2.3.3 Variation of the Hierarchical Framework	36

3.2.4	Single-Player Reinforcement Learning Framework for Automated Flowsheet Synthesis	37
3.2.4.1	General Idea	37
3.2.4.2	Agent	38
4	Results and Discussion	43
4.1	Modeling of Decanters: Convex Envelope Method	43
4.1.1	Qualitative Evaluation	43
4.1.2	Quantitative Evaluation	43
4.1.3	Analysis of the Impact of the Discretization Parameter δ	45
4.1.4	Discussion	46
4.2	SynGameZero: Proof of Concept	48
4.2.1	General Remarks	48
4.2.2	Case Study 1	49
4.2.3	Case Study 2	50
4.2.4	Discussion	52
4.3	SynGameZero: Integration of Hierarchical Reinforcement Learning	55
4.3.1	General Remarks	55
4.3.2	Original Hierarchical Framework	56
4.3.3	Variation of the Hierarchical Framework	57
4.3.4	Discussion	60
4.4	Single-Player Reinforcement Learning Framework for Automated Flowsheet Synthesis	62
4.4.1	General Remarks	62
4.4.2	Overall Performance	64
4.4.3	Comparison to Flowsheets from the Literature	65
4.4.4	Evolution of Long-Planned Recycles	66
4.4.5	Discussion	68
5	Conclusion and Outlook	71
	Bibliography	73
	Appendix	83
A	Appendix A	83
AI	Modeling of Distillation Columns: ∞/∞ -Analysis	83

B	Appendix B	87
BI	Modeling of Decanters: Convex Envelope Method	87
BI.1	Mathematics	87
BI.1.1	Simplex Geometry	87
BI.1.2	Discretization of the Composition Space	88
BI.2	Implementation	88
BI.3	Detailed Results	89
C	Appendix C	95
CI	SynGameZero: Proof of Concept	95
CI.1	Environment	95
CI.1.1	Chemical System and Unit Operations	95
CI.1.2	Cost Function	96
CI.2	Generation of the Flowsheet Matrix	97
CI.3	Implementation and Training Procedure	98
CI.3.1	Implementation	98
CI.3.2	Training Procedure	99
D	Appendix D	101
DI	SynGameZero: Integration of Hierarchical Reinforcement Learning	101
DI.1	Environment	101
DI.1.1	Chemical System and Unit Operations	101
DI.1.2	Cost Function	102
DI.2	Generation of the Flowsheet Matrix	104
DI.3	Implementation and Training Procedure	105
DI.3.1	Implementation	105
DI.3.2	Training Procedure	106
E	Appendix E	107
EI	Single-Player Reinforcement Learning Framework for Automated Flow- sheet Synthesis	107
EI.1	Environment	107
EI.1.1	Chemical Systems and Unit Operations	107
EI.1.2	Cost Functions	109
EI.2	Generation of the Flowsheet Matrix	111
EI.3	Implementation and Training Procedure	112
EI.3.1	General Implementation	112
EI.3.2	Artificial Neural Network Implementation	113

EI.3.3	Sampling of Problem Instances during Training	114
EI.3.4	Training Procedure	114

List of Figures

1	A general RL framework for AFS. The agent interacts with a process simulation as an environment and constructs a flowsheet by systematic generation. The agent can see the current state of the process and receives a reward depending on some monetary cost function.	6
2	Unit operations considered throughout this work: I) reactor, II) distillation column, III) decanter, IV) add solvent, V) mixer, VI) recycle, and VII) split.	10
3	Example for the discretization of simplices depending on the parameter δ for systems with $N = 2$ or $N = 3$ components.	15
4	Example for the convex envelope of a Δg^{mix} graph for a binary system (solid black and dashed red line segments). The homogeneous simplices are the black, solid line segments (those simplices connect only points that are direct neighbors in the discretized composition space). The red dashed line segment is a heterogeneous simplex (it spans over a multiphase region). The lower part shows the projection of the convex envelope to the discretized composition space.	16
5	Example for the classification of homogeneous (black) and heterogeneous (red) simplices for a ternary system. The homogeneous simplices connect neighboring points in the discretization space, while the heterogeneous simplices span over a larger area.	17
6	Examples for heterogeneous simplices. The homogeneous line segments are solid, and the heterogeneous line segments are dashed. The considered feed compositions are marked with the symbol x . The red lines in the first two columns show unique, linear splits of feeds inside the simplices. The last column shows two examples of simplices that cannot be modeled uniquely.	18
7	SynGameZero approach for AFS. The agent plays against itself by switching between the roles of players 1 and 2.	24
8	The agent's decision process in SynGameZero.	25

9	Example tree search at the beginning of the game (flowsheets of both players empty) with three possible actions: T, D ₁ , R.	27
10	The agent's hierarchical decision process.	32
11	Hierarchical ANN structure of the agent.	33
12	Example structure of the search tree with integrated hierarchy levels. . .	35
13	The agent's decision process.	39
14	Selection of ternary systems with UNIQUAC parameters from [62, 82] constructed with the generalized CEM at atmospheric pressure. The plots display molar fractions. The transparent red areas in the systems 1-hexanol – nitro methane – water and water – nitro methane – nonanol display three-phase regions, all other red areas display two-phase regions (for every two-phase region, a few example tie lines are plotted in black).	44
15	MD values for several choices for δ for the systems <i>n</i> -hexane – benzene – sulfolane and <i>n</i> -octane – toluene – sulfolane. MD was calculated using phase split data and parameters from [94]. Some points in the graph are marked with the symbol x, which indicates that no phase split was found for some given feed stream compositions for this choice of δ	46
16	Variation of δ for the construction of ternary diagrams for the systems <i>n</i> -hexane – benzene – sulfolane and <i>n</i> -octane – toluene – sulfolane from [94] at 298.15 K and atmospheric pressure. The red triangles are the heterogeneous simplices (i.e., simplices that span over a phase split region).	47
17	Illustrative example for the evolution of the agent during the training process in Case Study 1. Flowsheets proposed by the agent to separate an equimolar quaternary mixture are shown. Streams that leave the process without description (e.g., ABC) are empty.	50
18	Benchmark flowsheets for Case Study 2.	51
19	Example for the evolution of the agent during the training process for situation 1, Case Study 2. The 3D plots show the value of three highlighted actions of the ANN's policy output over a subset of the composition space of the feed streams ($\dot{n}_A = \dot{n}_B$, $\dot{n}_C = \dot{n}_D$) for the first action of the agent. Action 1 is mixing both feed streams. Action 2 refers to placing a distillation column of type D ₃ at the C – D feed stream. Action 3 refers to placing a reactor R at the A – B feed stream.	53
20	Benchmark flowsheets designed by the authors for the quaternary system Et – IB – nBut – ETBE.	56
21	Examples for flowsheets proposed by the trained agent.	58

-
- 22 Examples for the agent’s behavior for several feedstream combinations. a), b), c), and d) refer to the flowsheets shown in Figure 21. 59
- 23 Illustration of the agent’s evolution at different stages during training. The matrix field represents different feed stream combinations. The color code marks the winning player. The red box shows the winning flowsheet for the respective feed streams. 61
- 24 Flowsheets constructed by the trained agent for feed situations given in the literature [103–106] (training process was carried out using NPV). Flowsheet a) shows a process for the separation of Ac and Ch (feed composition: $x_{Ac} = 0.5, x_{Ch} = 0.5$) using Be as solvent. Flowsheet b) shows a process for the separation of Et and Wa (feed composition: $x_{Et} = 0.5, x_{Wa} = 0.5$) using solvent Be. Flowsheet c) shows a process for the separation of Bu and Wa (feed composition: $x_{Bu} = 0.4, x_{Wa} = 0.6$). Flowsheet d) shows a process for the separation of Py and Wa (feed composition: $x_{Py} = 0.1, x_{Wa} = 0.9$) using To as solvent. 67
- 25 Examples for the implications of recycles on the compositions of the streams, which show the planning capabilities of the trained agent. The agent constructed the displayed examples after being trained using GCF. To the left, the flowsheets without recycles are shown (to the right, the flowsheets with recycles are shown). Inside the ternary diagrams, the feed stream is marked by a black square, and the output streams are marked with a brown triangle (connected by a blue line). Panel a) shows a process for the separation of Ac and Ch using To as entrainer (feed composition: $x_{Ac} = 0.74, x_{Ch} = 0.26$). Panel b) shows a process for the separation of Wa and Py using To as solvent (feed composition: $x_{Wa} = 0.04, x_{Py} = 0.96$). . . 69
- A1 Topology of the VLE in the ternary system acetone – benzene – chloroform at 1 bar. The binary azeotrope and benzene span the distillation boundary. The arrows indicate the direction of the distillation lines toward the low-boiler. 84
- C1 Construction of the flowsheet matrix \mathbf{F} in the SynGameZero approach. \mathbf{F} contains the information of the stream table combined with structural information on the flowsheet. All entries in the matrix, which are not needed for now, are set to 0 ($\mathbf{0}$ refers to a vector consisting of as many entries equal to 0 as required for the width of the respective column). . . 98

-
- D1 Topology of the VLE in the quaternary system Et – IB – nBut – ETBE at 8 bar. The binary azeotropes and IB span the distillation boundary (gray surface). The arrows indicate the direction of the distillation lines toward the low-boiler. 103
- D2 Construction of the flowsheet matrix \mathbf{F} in the hierarchical SynGameZero approach. The matrix contains a stream table and information on the connectivity of the streams in the flowsheet. 105
- E1 Construction of the flowsheet matrix \mathbf{F} in the single-player framework. The matrix contains a stream table and information on the connectivity of the streams in the flowsheet. 113

List of Tables

1	Average performance metrics for Case Study 2.	51
2	Average performance metrics for the original hierarchical framework (the values are rounded).	56
3	Performance of the agent on the test set for both cost functions NPV and GCF. The ratio \mathcal{R} indicates how much of the input (feed and added solvent) the agent's flowsheet separates into pure components. Additionally, we report how often the agent proposes a flowsheet that separates the feed and added solvent completely into pure streams (Compl. sep.). Row 1 and 2 show the results for the agent using MCTS with a simulation budget of $K = 200$. Row 3 and 4 show the results for unrolling the policy with beam search (beam width $k = 512$).	65
B1	Results for ternary systems from [94] at atmospheric pressure. The parameter δ was set to 128, and the NRTL model was used for all systems. M describes the number of feed streams that were examined for the calculation of the MD.	90
B2	Results for quaternary systems at atmospheric pressure. The parameter δ was set to 64 for all systems. M describes the number of feed streams that were examined for the calculation of the MD.	91
B3	Results for quinary systems at atmospheric pressure. The parameter δ was set to 32, and the NRTL model was used for all systems. M describes the number of feed streams that were examined for the calculation of the MD.	93
B4	Results for systems containing six components at atmospheric pressure. The parameter δ was set to 16 for all systems. M describes the number of feed streams that were examined for the calculation of the MD.	94
C1	I_u parameters for Case Study 1 and Case Study 2.	97
C2	Component prices for Case Study 1 and Case Study 2.	97
C3	Numerical tuning parameters for Case Study 1 and Case Study 2. The parameter K specifies the depth of the tree search.	99

D1	Base values for investment costs of the units.	103
D2	Prices for components and steam.	104
E1	Considered chemical example systems and the available solvents for the single-player framework. We use the following abbreviations for the components: acetone (Ac), benzene (Be), butanol (Bu), chloroform (Ch), ethanol (Et), pyridine (Py), tetrahydrofuran (Te), toluene (To), water (Wa). The flowsheet simulation is based on phase equilibria that are constructed assuming constant conditions (temperature and pressure for liquid-liquid equilibria, pressure for vapor-liquid equilibria and distillation boundaries).	108
E2	Parameters for the cost function NPV. The base values for the investment costs of the unit operations are taken from [106] (for a mass flowrate $\dot{m}_0 = 25000 \frac{\text{kg}}{\text{h}}$). $I_{0,D}$ refers to the base value for the distillation column. $I_{0,Dec}$ refers to the base value of the decanter. Investment costs for all unit operations that are not listed are neglected. The prices for steam and components are chosen similarly as in Appendix DI.1.	110
E3	Parameters for GCF (as GCF has no unit, the parameters also have no units).	112

List of Symbols

Latin symbols

g^E	Molar excess Gibbs energy
K	Number of simulations in tree search
k	Beam width in beam search
m	Parameter for sequential halving procedure
\dot{m}	Mass flowrate
\dot{n}	Molar flowrate
\boldsymbol{p}	Filtered policy
\boldsymbol{R}	Universal gas constant
\mathcal{R}	Performance ratio
r	Reward
\boldsymbol{s}	State
v	Value, output of the value-head
\boldsymbol{y}	Normalized visit counts of tree search
N_{batch}	Batchsize
N_{layer}	Number of layers in neural network
N_{matrix}	Number of rows of flowsheet matrix
N_{memory}	Size of replay buffer
N_{node}	Width of a layer of a neural network
N_{steps}	Number of training steps

Greek symbols

δ	Discretization parameter in the CEM
π	Policy, output of the policy-head

Abbreviations

Ac	Acetone
ACN	Actor-critic network
AFS	Automated flowsheet synthesis
AI	Artificial intelligence
ANN	Artificial neural network
Be	Benzene
Bu	Butanol
Ch	Chloroform
Et	Ethanol
ETBE	Ethyl-tert-butyl-ether
GCF	Generic cost function
IB	Isobutene
LLE	Liquid-liquid equilibrium
MCTS	Monte-Carlo tree search
MD	Mean deviation
ML	Machine learning
MLP	Multi-layer perceptron
nBut	<i>n</i> -Butane
NPV	Net present value
OHE	One hot encoding
PSE	Process systems engineering
Py	Pyridine
RL	Reinforcement learning
Te	Tetrahydrofuran
To	Toluene
VLE	Vapor-liquid equilibrium
Wa	Water

1 Introduction

1.1 Automated Flowsheet Synthesis: Problem Description and Classification of Approaches

In chemical engineering, process synthesis (alternatively, flowsheet synthesis) can be defined as the act where one invents the structure and operating levels for a new chemical manufacturing process [1]. During process synthesis, the flowsheet topology is defined by specifying the placement of unit operations and recycles [2]. After chemical route synthesis, it can be seen as the second step in conceptual process design [3] and is to be distinguished from subsequent process optimization steps [4].

The process systems engineering (PSE) community has focused on computer-aided process synthesis for decades [5]. There are many methods in which the roles of humans and computers are quite different and vary in their proportions. On one end of the spectrum, humans invent flowsheets, provide mechanistic models of apparatus and physicochemical properties, and employ computers solely in simulations to evaluate and check the developed designs. On the other end of the spectrum is automated flowsheet synthesis (AFS), which we call human-aided process synthesis by a computer instead. Therein, the structure of the process and operating levels are chosen autonomously by the computer based on predefined input, i.e., available feed streams, desired products, and a predefined objective (e.g., some monetary profit). Sirola [6] classified AFS into three categories: superstructure optimization, evolutionary modification, and systematic generation. In superstructure optimization, a large flowsheet structure (the superstructure) is set up so that a large set of process alternatives can be obtained by removing parts of that structure [7, 8]. An objective function or cost function is defined and the optimal configuration for the flowsheet is determined by an optimization algorithm that uses decision variables to remove parts of the superstructure. Evolutionary modification works as follows: a process flowsheet is devised (by any method at hand), analyzed, and changed in one or more ways repeatedly to improve it. The changes are continued until no further improvement in the flowsheet can be made [9]. Systematic generation creates a flowsheet sequentially by adding unit operations from a predefined set. The

decision process is usually based on heuristics or prior knowledge. Alternatively, it is possible to derive heuristics by comparing many flowsheets systematically with the help of a computer [10]. Prominent examples of the systematic generation approach are the expert systems [11, 12]. Sometimes, two of the three categories are combined in hybrid synthesis methods [13, 14]. For further reading concerning the state-of-art of AFS, we refer to the already cited literature and review articles [5, 7, 15–18].

1.2 Machine Learning and Artificial Intelligence in Process Systems Engineering

As machine learning (ML) and artificial intelligence (AI) are rapidly expanding fields, a lot of research focuses on applying these kinds of techniques in PSE [19–25]. For example, ML approaches enable training of surrogate models for the prediction of phase equilibria [26–30] or molecular properties [31–33]. Surrogate models are also applied for process simulation and optimization to reduce computational time [26, 34–36].

AI offers, however, more potential, as stated by Dimiduk et al. [21]: "Or, how can one best apply the newest advances in ML and AI to improve materials, processes, and structures engineering results? Speculating still further, why are there no emerging AI-based engineering design systems that recognize component features, attributes, or intended performance to make recommendations about directions for final design, manufacturing processes, and materials selections or developments?" The ML technique that could address these problems is reinforcement learning (RL). The objective of RL is to teach an agent, which could, for example, consist of an artificial neural network (ANN), to master a given task through repeated interactions with its environment [37]. As the goals of RL and technical control schemes align nicely, RL is a frequently used tool in the field of optimal process control [38, 39]. Another RL application in the PSE field is the optimization of reaction pathways utilizing a variant of Monte-Carlo tree search (MCTS) [40, 41].

1.3 Reinforcement Learning for Automated Flowsheet Synthesis

The present work focuses on a combination of RL and AFS. As mentioned before, it is based on already published work of the author [42–47]. In the following, we provide a

brief overview of other RL-based AFS approaches that partially build on features of the frameworks presented in this work.

To our knowledge, Midgley [48] proposed the first approach to train an agent by RL for AFS. In this framework, the agent consists of an actor-critic network (ACN) and can set up simple distillation sequences for zeotropic process examples. Khan and Lapkin [49] demonstrated that it is possible to identify promising processing routes in hydrogen production by using an RL approach. Given the open streams in a linear scheme, the agent uses a thermodynamic graph and the Q-value to choose the units of the flowsheet. In [50], a hierarchical RL framework that decomposes process synthesis into two hierarchy levels is proposed. At the first level, a topology for several process sections (e.g., a section for separation) is constructed using a similar approach as in [49]. At the second level, these process sections are specified in detail by connecting unit operations. This part of the agent was trained using proximal policy optimization [51]. The approach was shown alongside an example process for ethylene oxide production. This work was further continued in [52], where an ontological framework supported the agent in its decisions. In [53], a different hierarchical RL framework is presented. The agent chooses an open stream at the first level and at the second level a corresponding unit. At a third level, a continuous specification for that unit is set. The flowsheets are represented as graphs and the agent consists of several graph neural networks combined with an actor-critic architecture. The agent is trained via proximal policy optimization [51] to construct a process for the production of methyl acetate.

2 Motivation

2.1 Thesis

The thesis of this work is that it is possible to train an agent from scratch via RL without using heuristics to synthesize processes for conceptual design problems in chemical engineering. The flowsheets are set up by the agent sequentially by connecting unit operations, specifying their operational parameters, and deciding on the termination of the synthesis process. This means that the agent can operate in a heterogeneous, continuously parameterized ('hybrid') action space without guidance from human engineers. During training, the agent encounters varying initial situations (i.e., feed streams in several chemical systems). It learns to approach a broad range of conceptual design problems by developing its artificial process engineering intuition.

The motivation for omitting any guidance for the agent by heuristics or prior knowledge is twofold: on the one hand, the usage of heuristics might restrict the agent's ability to come up with genuinely new designs; on the other hand, it has been shown on multiple occasions that AIs can be trained via RL without prior knowledge to outperform humans in various domains with combinatorial search spaces, e.g., [54–57]. The present work adapts and further improves on recently published ideas concerning RL and integrates those into an AFS framework. Furthermore, a flowsheet simulation based on thermodynamic short-cut models is provided within this work as an environment for the RL agent. To achieve this, existing ideas on thermodynamic short-cut models are implemented, further developed, and proven to work. In summary, this work marks a significant step towards a general process engineering AI, which can transfer its learnings from the training process onto conceptual design problems it has never encountered before.

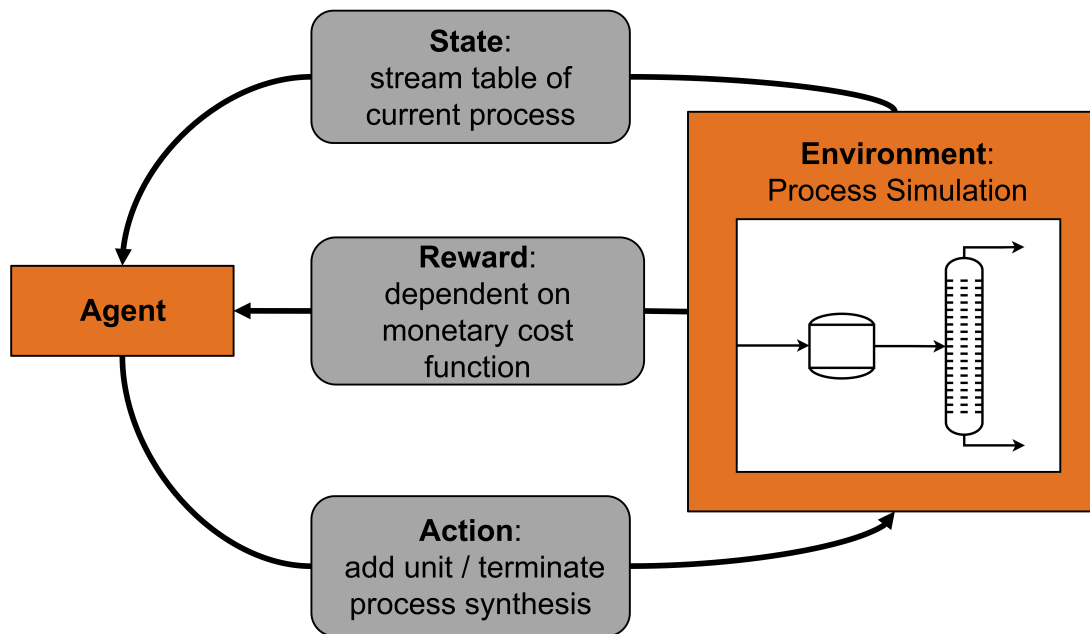


Figure 1: A general RL framework for AFS. The agent interacts with a process simulation as an environment and constructs a flowsheet by systematic generation. The agent can see the current state of the process and receives a reward depending on some monetary cost function.

2.2 General Reinforcement Learning Framework and Challenges

A general RL framework that models AFS as Markov decision process (MDP) [37] is explained alongside Figure 1. The *environment* for the *agent* is a deterministic, steady-state process simulation. The flowsheet is constructed using systematic generation [6], i.e., by sequentially connecting unit operations. The agent can observe the *state* of the environment, i.e., a stream table of the current flowsheet combined with information on the connectivity inside the process. The possible *actions* that the agent can perform are adding new unit operations to the flowsheet, setting their operational parameters if applicable, or terminating the flowsheet synthesis. After choosing an action, the environment updates the state by simulating the current flowsheet. As feedback, the agent obtains a *reward*, which is generally based on a cost function that is evaluated in the simulation (e.g., net present value (NPV) of the process).

The goal is to train the agent from scratch via RL to maximize this reward while interacting with the simulation only. To achieve this, one must overcome some significant challenges listed below.

Challenges on environment side

I) **Robustness of process simulation**

As the agent has no prior knowledge of AFS, it will likely propose flowsheets of low quality during the early stages of training. A robust environment is required as it is difficult to provide a resulting state and reward to the agent in case of a divergent simulation. Therefore, whenever possible, models for unit operations should be chosen to converge for most of the possible input specifications.

II) **Speed of process simulation**

In other RL applications, the agent has to interact thousands of times with the environment to master specific tasks (e.g., board games such as chess [54, 55]). Therefore, it is expected that the agent has to construct thousands of flowsheets to grasp fundamental concepts from conceptual process design. To train the agent in a reasonable time, it is essential to provide an environment that simulates the proposed designs fast, e.g., by using short-cut models for the available unit operations.

Challenges on agent side

I) **Local optima**

A general problem in RL and optimization is convergence to local optima. In our case, this problem is twofold. On the one hand, the agent must decide if a flowsheet is an optimal solution or if further exploration is needed. On the other hand, the RL framework must ensure that the agent's policy converges to a global optimum (or at least not stopping training at a local optimum).

II) **Sparse reward signal**

The reward guides the agent's training process; thus, it is advantageous to provide a reward after every action the agent takes. This is quite difficult for AFS (and most planning processes in general), as it is very hard to judge the quality of a particular action without knowing the final result, i.e., the final flowsheet. For example, think of a multi-step separation sequence that only works successfully after a recycle has been closed. After placing the first unit operation of the sequence, a constructive reward is hard to determine. Further, if the agent does not close the recycle and terminates the flowsheet synthesis instead, the process does not work. Which was the unit operation that caused the process ultimately to fail? In this case, it is not even possible to reward or punish individual actions constructively after completing the flowsheet synthesis. Consequently, we suggest not rewarding every single action but rather looking only at the final flowsheet and providing

a reward based on its value (e.g., NPV) for the agent. This leads to a relatively sparse reward signal, which may cause difficulties for the agent’s learning process.

III) **Hybrid and heterogeneous action space**

To construct a flowsheet, one needs to work in a hybrid action space consisting of discrete and continuous actions, e.g., deciding on the destination of a recycle and specification of the split ratio inside a distillation column, respectively. A lot of RL algorithms only work with either discrete or continuous actions. While much research focuses on hybrid action spaces, e.g., [58, 59], the proposed algorithms are usually tailor-made for a specific problem class. Additionally, the action space is heterogeneous, which means several types of actions cause different implications for the flowsheet, e.g., placing a unit operation to an open stream, terminating the process synthesis, or deciding on the destination of a recycle. To summarize, the structure of the action space is quite complex; therefore, great attention should be paid to this fact during the setup of the RL framework.

The subsequent chapters will address these challenges, which are structured as follows. In Chapter 3, the methodology is introduced. Section 3.1 presents the unit operations considered throughout this work. In Section 3.1.2, the development of the convex envelope method (CEM) is explained. The CEM constructs linear representations of liquid phase equilibria that are used to model decanters subsequently. In Section 3.2.2, the SynGameZero approach is introduced. It models flowsheet synthesis as a competitive two-player game and enables training an agent for AFS by RL within a discrete action space. In Section 3.2.3, the SynGameZero approach is further improved by the hierarchical decomposition of the action space. Finally, in Section 3.2.4, a single-player RL framework suitable for AFS is described. In Chapter 4, results for the methodologies described in Sections 3.1.2, 3.2.2, 3.2.3, 3.2.4 are presented and discussed. In Chapter 5, an outlook onto directions for future research and concluding remarks are provided.

3 Methodology

3.1 Environment: Process Simulation

3.1.1 General Remarks and Available Unit Operations

The environment is a deterministic, steady-state process simulation set up in Python. A set of feed streams initializes it. An action consists of terminating the process synthesis or choosing an open stream in the current flowsheet, a unit operation, and its specification. After each action, the simulation returns an updated state of the current process. The simulation solves the process-specific unit operation models in a sequential-modular way [60]. The unit operations are based on short-cut models, which allow a quick and robust evaluation.

The unit operations and specifications considered throughout this work are shown in Figure 2 and listed below. Additional information will be provided alongside the case studies in subsequent sections.

I) **Reactor**

The model for this unit operation depends on the chemical system and is explained alongside the respective case studies.

Within this work, placing a reactor is a purely discrete action without further specifications (e.g., temperature, pressure) to be set.

II) **Distillation column**

Distillation columns are modeled using the ∞/∞ -approach [61–64] by employing a linearised representation of the underlying vapor-liquid equilibrium (VLE). As the approach was adapted from [62, 63], we outline the basic concepts in the Appendix AI.

When a distillation column is placed, the ratio of distillate flowrate \dot{n}^D to feed flowrate \dot{n}^F has to be specified:

$$\frac{\dot{n}^D}{\dot{n}^F} \in [0, 1]. \quad (1)$$

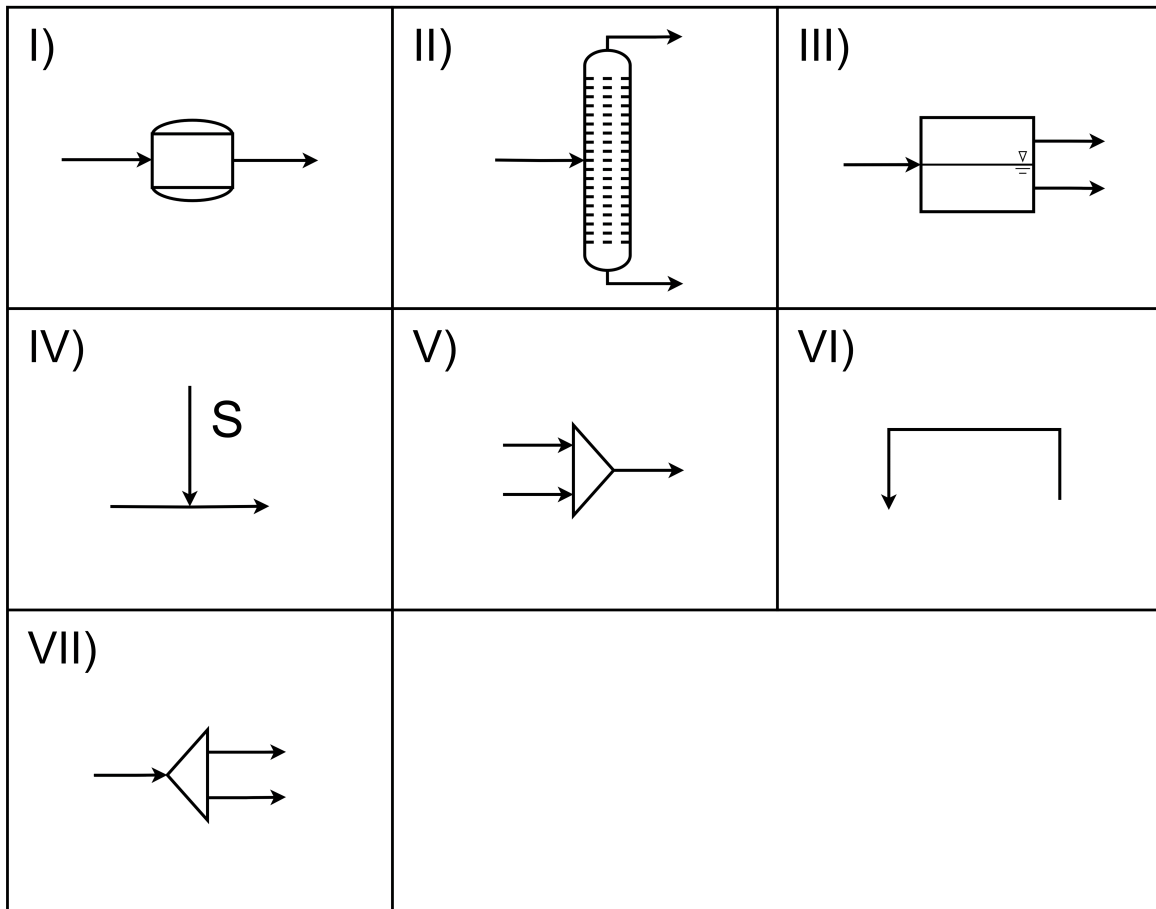


Figure 2: Unit operations considered throughout this work: I) reactor, II) distillation column, III) decanter, IV) add solvent, V) mixer, VI) recycle, and VII) split.

III) Decanter

We employ a linearised representation of the underlying liquid-liquid equilibrium (LLE) constructed by the convex envelope method (CEM) for a decanter model. Details are provided in Section 3.1.2.

Within this work, placing a decanter is a purely discrete action without further specifications (e.g., temperature, pressure) to be set. If a decanter is placed to a feed stream that does not display a liquid phase split into two phases, the environment simulates the decanter as split (see VII) Split). This ensures a constant number of two output streams for the decanter.

IV) Add solvent

It can be decided to mix a solvent to an open stream. In this case, the ratio of solvent flowrate \dot{n}^S to the flowrate of the open stream \dot{n}^F has to be specified:

$$\frac{\dot{n}^S}{\dot{n}^F} \in [0, \infty). \quad (2)$$

V) Mixer

Placing a mixer to an open stream is a discrete action requiring a discrete specification, namely, choosing another open stream to mix with. If more than two streams must be mixed, placing multiple mixers in a cascade is possible.

IV) Recycle

Throughout this work, tear streams are used to simulate recycles [60]. The underlying fixed-point problem is transformed into a root-finding problem and solved by the usage of the function `fsolve` provided within the Python package `scipy` [65].

Similarly to the mixer, recycling an open stream is a discrete action, which requires a discrete specification, namely, choosing the destination of the recycle. Possible destinations are all other closed streams (i.e., streams that are not open) in the current flowsheet.

VII) Split

This unit splits its feed stream according to a predefined ratio into two streams of identical composition as the feed stream. In the present work, this ratio is constantly set to 0.5, meaning that both output streams are identical in flowrate and composition.

As explained above (see III) Decanter), a split is only introduced to ensure a constant number of output streams for the simulation of a decanter. It is not

provided as a separate option for the agent to choose from.

3.1.2 Modeling of Decanters: Convex Envelope Method

3.1.2.1 General Idea

The underlying liquid phase equilibrium has to be modeled to simulate a decanter. While there exists a plethora of different approaches related to this topic, most of them are still based on mechanistic models. Given a g^E -model, one can try to solve an equation system based on the isoactivity condition [66–68]. As this condition is only a necessary but not sufficient criterion for stable equilibrium phases, more equations and conditions must be considered to ensure correct solutions [66]. Additionally, these methods usually rely on a good initial guess for the resulting compositions of the phases to prevent convergence to trivial solutions [66, 67]. Thus, most g^E -model-based approaches employ the tangent plane criterion [69], based on the minimization of Gibbs energy [70, 71]. For a given feed composition, the stable phases are therein found by an optimization algorithm. There exist various examples of such approaches. Listing just a few, they can be classified in deterministic [72–77] or stochastic optimization algorithms [78, 79]. We refer to the literature for a thorough summary [68, 80, 81].

Within this section, we provide a generalization of the CEM [62, 82], which is based on the tangent plane criterion [69], but can also be seen as a surrogate model, as it constructs an approximation of the phase equilibrium within a discretized space. Contrary to the approaches mentioned beforehand, the CEM constructs a liquid phase equilibria diagram for the (discretized) composition space of a given mixture as a whole and stores it in a piecewise linear representation. Afterward, phase splits can be computed robustly and fast for given feed compositions. The number of phases for a given feed is computed by the CEM and thus has not to be specified beforehand. Obviously, the construction for the whole composition space is computationally inefficient if only a few feed compositions for fixed conditions (temperature and pressure) have to be evaluated. However, it is advantageous in conceptual process design when many different process options with possible recycles have to be evaluated robustly. In the CEM, the composition space is discretized into equally distributed points, and the Gibbs energy of mixing is calculated at every point. Combined with those values, the discretization points represent a graph, which is used as a basis to construct a convex envelope around it. Given this surface, the tangent plane criterion [69] is used to determine all stable equilibrium states in the whole composition space. [62] and [82] presented the CEM to determine phase equilibrium diagrams in the liquid phase for up to four components.

While it is clear from a thermodynamic point of view, as stated in [62, 82], that the CEM applies to systems with an arbitrary number of components, such a generalization is still missing. This generalization will be provided by the present work, which presents a changed version of the CEM.

We refer to [62, 82] for a detailed derivation and explanation of the CEM and give a brief description of the general steps for a system consisting of $N \in \mathbb{N}$ components:

I) **Discretization of the composition space**

A system consisting of N components can be represented by a simplex with N vertices (for a mathematical explanation, we refer to Appendix BI.1.1). This simplex is discretized by choosing points inside it, which are uniformly distributed. This is done by specifying a minimal distance $\frac{1}{\delta}, \delta \in \mathbb{N}$ between the discretization points. Figure 3 provides a visualization of this concept. For $N = 2$ and $\delta = 4$, a binary system is discretized into 5 points with a minimal distance of $\frac{1}{4}$ between them (Figure 3 a)). For $N = 3$ and $\delta = 4$, the discretization yields 15 points in total (Figure 3 b)). Note that the number of points in the discretized space is bounded by $\mathcal{O}((\delta + 1)^{N-1})$, but alongside Figure 3 it is also easy to visualize that for $N > 2$, the number of points is less than $(\delta + 1)^{N-1}$. We refer to Appendix BI.1.2 for a detailed explanation of the discretization procedure.

II) **Determination of the Δg^{mix} -graph and the convex envelope**

For each point in the discretized space with molar fractions \mathbf{x} , the Gibbs energy of mixing [83] is calculated:

$$\Delta g^{\text{mix}}(\mathbf{x}) = g(\mathbf{x}) - \sum_{i=1}^N x_i g_i^{\text{pure}} = RT \sum_{i=1}^N \ln(x_i \gamma_i). \quad (3)$$

The molar fractions \mathbf{x} are transformed to cartesian coordinates $\mathbf{a} \in \mathbb{R}^n$ (described in Appendix BI.1.1) with $n = N - 1$. Combined with the values for $\Delta g^{\text{mix}}(\mathbf{x})$, one obtains points of the form $(\mathbf{a}, g^{\text{mix}}(\mathbf{x}))$, which represent a graph in \mathbb{R}^{n+1} . The convex envelope is constructed around the points of this graph.

III) **Classification of the simplices of the convex envelope**

The convex envelope from step II) consists of several simplices, i.e., boundary elements. For example, the simplices of the convex envelope in two dimensions are straight line segments in three dimensions triangles. Figure 4 shows an example for the convex envelope of a binary system. To obtain possible phase splits, one has to classify the simplices of the envelope into homogeneous (black, solid line segments) and heterogeneous (red, dashed line segment). A homogeneous simplex connects only points that are direct neighbors in the discretized composition space,

i.e., points with minimal distance, specified by the parameter δ (see also Figure 3). All other simplices are heterogeneous and span over a multiphase region. As shown in the lower part of Figure 4, the simplices can be classified after the projection of the convex envelope to the discretized composition space. The values for Δg^{mix} are discarded, and the remaining distances determine whether two points are neighboring. In a ternary system, the simplices are triangles consisting of straight line segments, where each line segment either connects two neighboring points in the discretization space, i.e., a homogeneous line segment, or not, i.e., a heterogeneous line segment. Simplices that connect only neighboring points and thus consist only of homogeneous line segments are called homogeneous; all other simplices are called heterogeneous. Figure 5 visualizes the classification into homogeneous (black) and heterogeneous (red) simplices for a ternary system.

For each heterogeneous simplex, one must check if it can be used to model a unique phase split in a linearised decanter model. This is shown alongside Figure 6 similarly as in [62]. The homogeneous line segments are solid, and the heterogeneous line segments are dashed. Figure 6 a) shows a split of a feed in a ternary system into two phases: a unique straight line through the feed, which ends in the compositions of the two phases. Figure 6 b) shows a split of a feed in a ternary system into three phases (a unique plane, which contains the feed and the compositions of the three phases). In Figure 6 c), there are infinitely many possibilities to draw a straight line through the feed and the two phases. Therefore, we can not model this simplex uniquely and omit it for further analysis in step IV). Figures 6 d)-f) show the same situations for systems with four components.

As mentioned in [62], heterogeneous simplices, which are not feasible for a unique, linear phase split, occur rarely and only at the boundary of multiphase regions, i.e., at locations where heterogeneous and homogeneous simplices are direct neighbors. They nearly vanish when more points are specified within the discretization. We provide detailed results on the effect of a low value for δ on the accuracy of the CEM later on.

Steps I)-III) yield the phase diagram for the given system at the temperature used to calculate Δg^{mix} . Examples are shown in the Results section. These steps have to be done only once per system at fixed conditions, i.e., temperature and pressure.

IV) Computation of phase splits

For a given feed composition, one has to check if the point lies within a feasible, heterogeneous simplex, which has been stored at step III). The occurring phase split is computed depending on the simplex structure and the lever arm rule. This step is explained in detail in the next section.

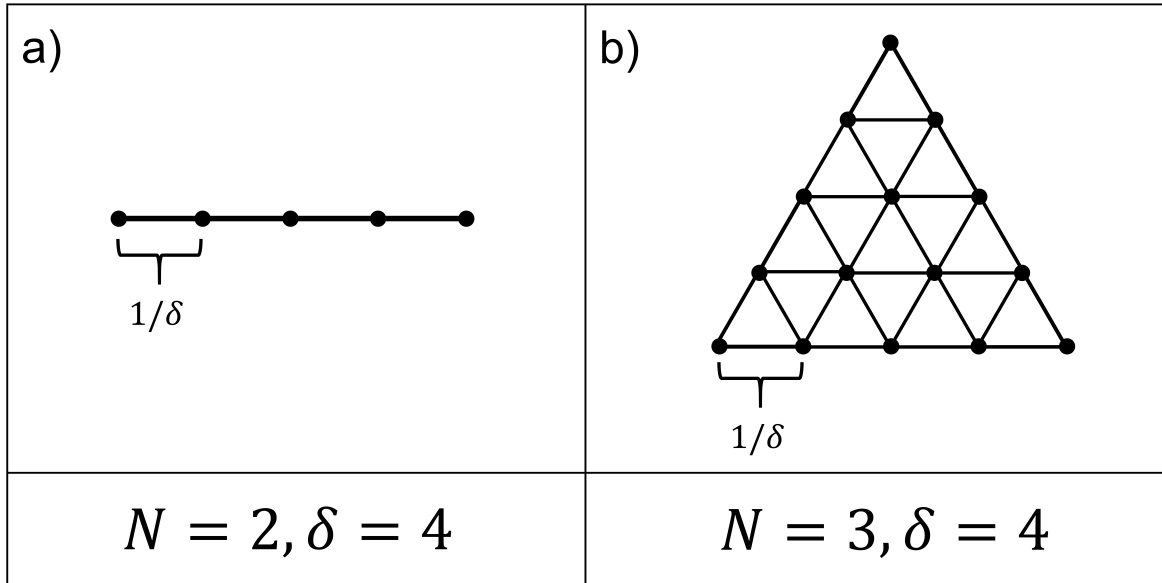


Figure 3: Example for the discretization of simplices depending on the parameter δ for systems with $N = 2$ or $N = 3$ components.

Step I) and II) can be executed for an arbitrary number of components without a change. In step III) one has to classify the simplices into homogeneous and heterogeneous. In [62], a graphical evaluation scheme for this task is provided that distinguishes several cases and does the decision on a case-by-case basis. The scheme is only presented for up to four components. An extension to more components would be very cumbersome and hard to grasp. The number of cases for the heterogeneous simplices increases rapidly (binary: 1, ternary: 3, quaternary: 8). This problem remains present in step IV) as one has to define the type of linear split for every possible simplex topology. To address these challenges, we propose a mathematical framework for the classification of the heterogeneous simplices that is general and allows the computation of splits for an arbitrary number of components and phases.

3.1.2.2 Mathematical Framework

We consider a system consisting of $N \in \mathbb{N}$ components, which is represented by a n -simplex U in \mathbb{R}^n with $n = N - 1$ (e.g., a 3-component system is represented by a triangle in \mathbb{R}^2 , i.e., a 2-simplex). Given such an n -simplex, we now want to switch easily between cartesian coordinates (i.e., coordinates in \mathbb{R}^n) and molar fractions. This can be done by using barycentric coordinates. Barycentric coordinates of a point inside a simplex are coordinates with respect to the vertices of this simplex, which are non-negative and sum up to 1. As will be seen later on, barycentric coordinates of a point inside a simplex, which represents the whole composition space, are identical with molar fractions of this

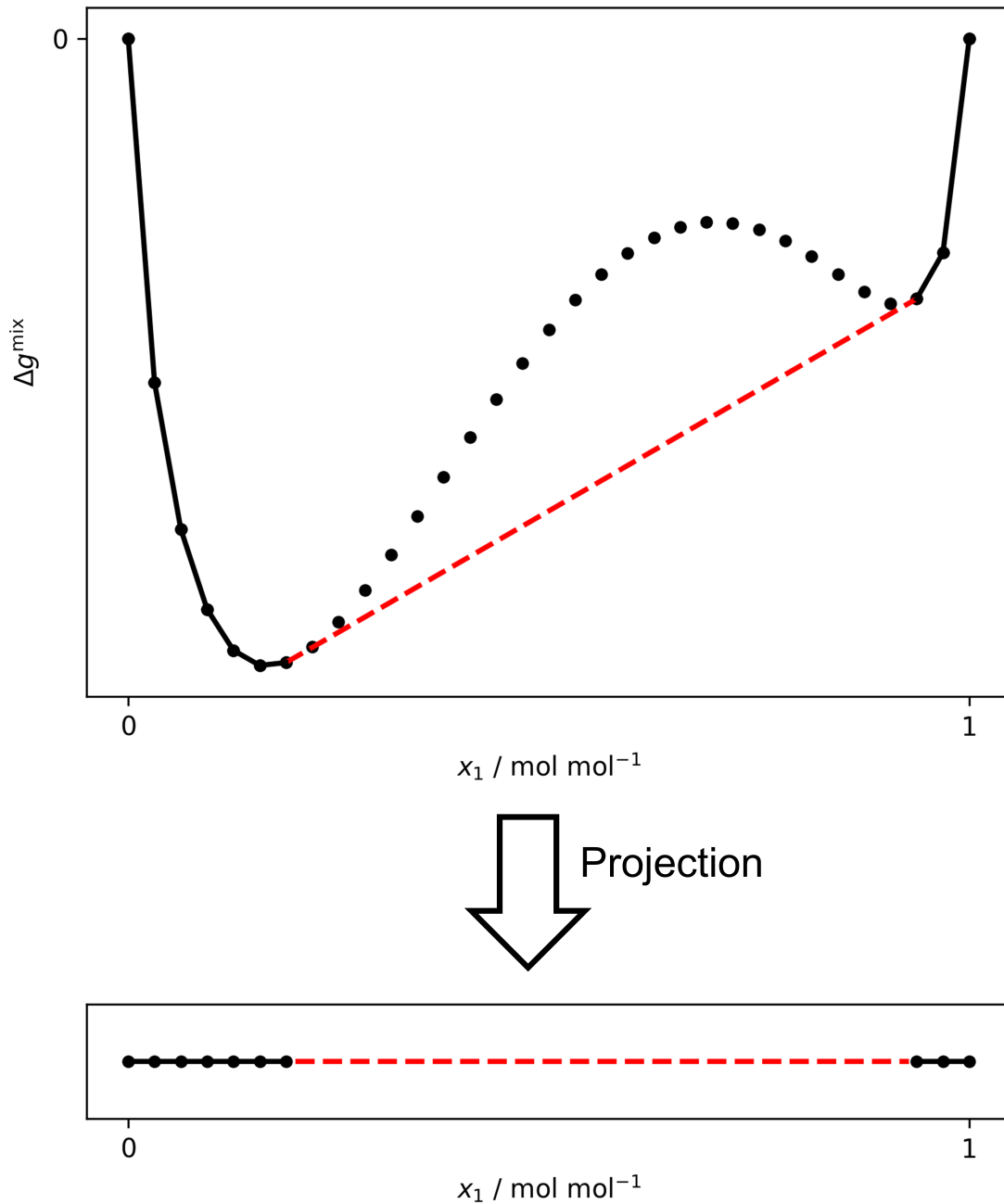


Figure 4: Example for the convex envelope of a Δg^{mix} graph for a binary system (solid black and dashed red line segments). The homogeneous simplices are the black, solid line segments (those simplices connect only points that are direct neighbors in the discretized composition space). The red dashed line segment is a heterogeneous simplex (it spans over a multiphase region). The lower part shows the projection of the convex envelope to the discretized composition space.

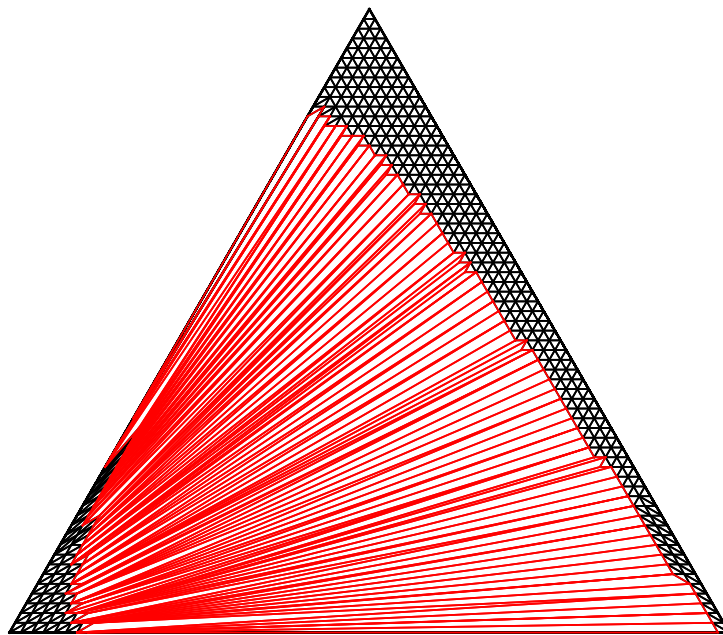


Figure 5: Example for the classification of homogeneous (black) and heterogeneous (red) simplices for a ternary system. The homogeneous simplices connect neighboring points in the discretization space, while the heterogeneous simplices span over a larger area.

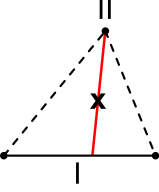
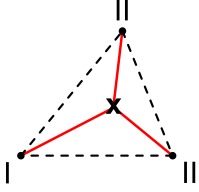
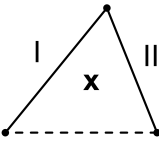
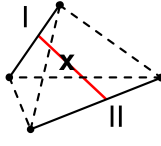
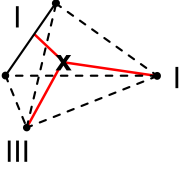
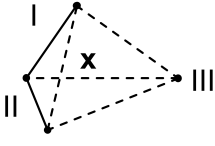
	Unique split (2 phases)	Unique split (3 phases)	No unique split
$N = 3$	a) 	b) 	c) 
$N = 4$	d) 	e) 	f) 

Figure 6: Examples for heterogeneous simplices. The homogeneous line segments are solid, and the heterogeneous line segments are dashed. The considered feed compositions are marked with the symbol x . The red lines in the first two columns show unique, linear splits of feeds inside the simplices. The last column shows two examples of simplices that cannot be modeled uniquely.

point in the composition space. For a detailed explanation of the simplex geometry used throughout this work, we refer to the Appendix BI.1.1. Note that the transformation by barycentric coordinates is linear; thus, the following geometric results should also work directly in the molar fraction space. We transform the coordinates mainly for illustration, as we believe some concepts are more straightforward to visualise this way. If not stated otherwise, we refer from now on to cartesian coordinates. We assume that the system has been discretized and that the convex envelope of the Δg^{mix} -graph has been computed. We start with defining heterogeneous simplices:

Definition 1 (Heterogeneous simplex). Consider a system consisting of N components, represented by a n -simplex U with $n = N - 1$, which was discretized with parameter δ . Let $\mathbf{h}_1, \dots, \mathbf{h}_N \in \mathbb{R}^n$ be points in the discretized space ($\mathbf{h}_1, \dots, \mathbf{h}_N$ are the cartesian coordinates after transformation of the molar fractions), which define a simplex $H = \text{conv}(\{\mathbf{h}_1, \dots, \mathbf{h}_N\})$, originating from step II) in Section 3.1.2.1. We call H *heterogeneous* or a *heterogeneous simplex* if there exist at least two indices $i, j \in \{1, \dots, N\}$ with $i \neq j$ so that \mathbf{h}_i and \mathbf{h}_j are not neighboring points in the underlying discretization of the composition space. neighboring points are defined by having a minimal distance dependent on δ in the discretized space.

From step III) in Section 3.1.2.1, we know that not every heterogeneous simplex is necessarily feasible for a unique, linearised decanter model. The following definition provides a subset of heterogeneous simplices, which can be modelled in the desired way (this will be proven at the end of this section).

Definition 2 (Phase block, isolated simplex). Let $H = \text{conv}(\{\mathbf{h}_1, \dots, \mathbf{h}_N\})$ be a heterogeneous simplex. A *phase block* is a subset $B \subseteq \{\mathbf{h}_1, \dots, \mathbf{h}_N\}$ with the following properties:

- I) For all $\mathbf{b}_i, \mathbf{b}_j \in B$ with $\mathbf{b}_i \neq \mathbf{b}_j$, it holds that \mathbf{b}_i and \mathbf{b}_j are neighboring points in the underlying discretized composition space.
- II) B is maximal w.r.t. property I), i.e., for all $\mathbf{h} \in \{\mathbf{h}_1, \dots, \mathbf{h}_N\} \setminus B$ there is at least one $\mathbf{b} \in B$ such that \mathbf{h} and \mathbf{b} are not neighboring points.

Note that a phase block $B = \{\mathbf{b}_1, \dots, \mathbf{b}_{k+1}\}$ defines a k -simplex $\text{conv}(B)$ which only consists of neighboring points.

A phase block B is called *isolated*, if for all $\mathbf{h} \in \{\mathbf{h}_1, \dots, \mathbf{h}_N\} \setminus B$ and all $\mathbf{b} \in B$ it holds that \mathbf{h} and \mathbf{b} are not neighboring points. Note that this is a stricter requirement than property II).

A heterogeneous simplex H is called *isolated* if all phase blocks in H are isolated.

Remark 1 (Decomposition property of heterogeneous, isolated simplices). It follows directly from the definition that any isolated heterogeneous $(N - 1)$ - simplex $H = \text{conv}(\{\mathbf{h}_1, \dots, \mathbf{h}_N\})$ decomposes uniquely into $m \leq N$ isolated phase blocks B_1, \dots, B_m such that $\bigcup_{i=1}^m B_i = \{\mathbf{h}_1, \dots, \mathbf{h}_N\}$. Furthermore, $B_i \cap B_j = \emptyset$ for $i \neq j$, and also $\text{conv}(B_i) \cap \text{conv}(B_j) = \emptyset$ as neighboring points have minimal distance in the discretized composition space.

These definitions can be visualised using Figure 6: the simplices in a)-f) are all heterogeneous since they contain at least one heterogeneous line segment. Examples of phase blocks are the line segment I (a 1-simplex) and the point II (a 0-simplex) in part a) of Figure 6 (these are also isolated, as the only connections to other phase blocks are heterogeneous line segments). Figure 6 part c) shows two phase blocks (the line segments I and II), which are not isolated as they are connected. Also, those two line segments do not just form one phase block since a third homogeneous line segment would be needed to represent a 2-simplex (i.e., a triangle). Note that one could also spare the classification into homogeneous and heterogeneous simplices, as a homogeneous simplex can be seen as a simplex consisting of just one phase block. We keep the distinction into

two types of simplices to emphasise the differences: a heterogeneous simplex, contrary to a homogeneous simplex, models a phase split.

Additionally, note that in Figure 6, the simplices, which can be modelled uniquely, are precisely the ones that are isolated. Before we can prove that the isolated, heterogeneous simplices are exactly the ones which can be modelled within a unique, linearised decanter model, we need to describe a phase split inside a heterogeneous simplex:

Definition 3 (Phase split simplex). Let $H = \text{conv}(\{\mathbf{h}_1, \dots, \mathbf{h}_N\}) \subset \mathbb{R}^n$ be a heterogeneous, isolated simplex, with isolated phase blocks B_1, \dots, B_m . Let \mathbf{x} be the molar fractions of a feed with cartesian coordinates $\mathbf{a} \in \mathbb{R}^n$ so that \mathbf{a} lies in H . A split of \mathbf{a} into m phases is defined by a $(m-1)$ -simplex $S = \text{conv}(\{\mathbf{p}_1, \dots, \mathbf{p}_m\}) \subseteq H$ with vertices $\mathbf{p}_1, \dots, \mathbf{p}_m \in \mathbb{R}^n$ so that the following properties are fulfilled:

- I) We have $\mathbf{a} \in S$.
- II) For all $i \in \{1, \dots, m\}$, we have $\mathbf{p}_i \in \text{conv}(B_i)$.

We call S a *phase split simplex with respect to \mathbf{a}* . The vertices of S define the compositions of the resulting phases, and we call \mathbf{p}_i a *phase*. The cartesian coordinates of the vertices $\mathbf{p}_1, \dots, \mathbf{p}_m$ can be transformed to molar fractions (i.e., the compositions of the resulting phases) via the barycentric coordinates with respect to the simplex which represents the whole N -component system. Furthermore, the split ratios $\lambda_1, \dots, \lambda_m$ are given exactly by the barycentric coordinates of \mathbf{a} with respect to the vertices of S .

For example, a phase split simplex for a given feed for two phases is a 1-simplex and, therefore, a straight line, which contains the feed and is defined by the compositions of the resulting phases. The barycentric coordinates of the feed with respect to that line yield precisely the well-known lever arm rule (for barycentric coordinates $\lambda_1, \dots, \lambda_m$ holds $\lambda_i \geq 0$ and $\sum_{i=1}^m \lambda_i = 1$).

Now consider an arbitrary feed composition within an isolated, heterogeneous simplex. This means that the simplex spans over a multiphase region, and we want to compute the compositions of the resulting phase split. The following theorem proves, on the one hand, that for every feed composition within an isolated, heterogeneous simplex exists a unique linear phase split (i.e., a phase split simplex from Definition 3). On the other hand, it also provides a formula for the computation of the compositions of the resulting phases.

Theorem 1. Let \mathbf{x} be a feed composition and $\mathbf{a} \in \mathbb{R}^n$ its representation in cartesian coordinates, located in an isolated, heterogeneous $(N - 1)$ -simplex $H = \text{conv}(\{\mathbf{h}_1, \dots, \mathbf{h}_N\})$, with $k > 1$ phase blocks B_1, \dots, B_k . For any phase block B_i , denote by $\Lambda_i = \{j \in \{1, \dots, N\} \mid \mathbf{h}_j \in B_i\}$ its corresponding index set. Let $(\tilde{\lambda}_1, \dots, \tilde{\lambda}_N)$ be the barycentric coordinates of \mathbf{a} with respect to H . Furthermore, set $\lambda_i := \sum_{j \in \Lambda_i} \tilde{\lambda}_j$.

Then $S = \text{conv}(\{\mathbf{p}_1, \dots, \mathbf{p}_k\})$ with $\mathbf{p}_i = \sum_{j \in \Lambda_i} \alpha_j \mathbf{h}_j$, where $\alpha_j = \frac{\tilde{\lambda}_j}{\lambda_i}$, is a unique phase split simplex for \mathbf{a} . The split ratios for the phases are given by $(\lambda_1, \dots, \lambda_k)$.

Proof. We first show that S is a phase split simplex for \mathbf{a} . We certainly have $\sum_{j \in \Lambda_i} \alpha_j = 1$ by definition and $\alpha_j \geq 0$ as $\tilde{\lambda}_j \geq 0$. Hence $\mathbf{p}_i \in \text{conv}(B_i)$, and we only need to show that $\mathbf{a} \in S$. For this, notice that $\lambda_i \geq 0$ and

$$\mathbf{a} = \sum_{j=1}^N \tilde{\lambda}_j \mathbf{h}_j \stackrel{(*)}{=} \sum_{i=1}^k \sum_{j \in \Lambda_i} \tilde{\lambda}_j \mathbf{h}_j = \sum_{i=1}^k \lambda_i \cdot \sum_{j \in \Lambda_i} \frac{\tilde{\lambda}_j}{\lambda_i} \mathbf{h}_j = \sum_{i=1}^k \lambda_i \mathbf{p}_i,$$

where $(*)$ follows from the decomposition property of heterogeneous, isolated simplices (Remark 1). With the same reasoning, obtain

$$\sum_{i=1}^k \lambda_i = \sum_{i=1}^k \sum_{j \in \Lambda_i} \tilde{\lambda}_j = \sum_{j=1}^N \tilde{\lambda}_j = 1,$$

hence $\mathbf{a} \in S$ with barycentric coordinates $(\lambda_1, \dots, \lambda_k)$. For uniqueness of S , let $S' = \text{conv}(\{\mathbf{p}'_1, \dots, \mathbf{p}'_k\})$ with $\mathbf{p}'_i = \sum_{j \in \Lambda_i} \alpha'_j \mathbf{h}_j$ be another phase split simplex for \mathbf{a} with barycentric coordinates $(\lambda'_1, \dots, \lambda'_k)$ (with respect to S'). As

$$\sum_{i=1}^k \sum_{j \in \Lambda_i} \lambda'_i \alpha'_j \mathbf{h}_j = \mathbf{a} = \sum_{i=1}^k \sum_{j \in \Lambda_i} \lambda_i \alpha_j \mathbf{h}_j,$$

it must hold that $\lambda'_i \alpha'_j = \lambda_i \alpha_j$ for all $i \in \{1, \dots, k\}$ and $j \in \Lambda_i$ by the uniqueness of the barycentric coordinates of \mathbf{a} with respect to H . Hence,

$$\lambda'_i = \lambda'_i \underbrace{\sum_{j \in \Lambda_i} \alpha'_j}_{=1} = \sum_{j \in \Lambda_i} \lambda'_i \alpha'_j = \sum_{j \in \Lambda_i} \lambda_i \alpha_j = \lambda_i$$

so $\alpha_j = \alpha'_j$ for all i and $j \in \Lambda_i$, and it follows $S = S'$. \square

With Theorem 1, we know how to model unique, linear phase splits in isolated, heterogeneous simplices. Therefore, we propose the following modifications to step III) and

IV) in Section 3.1.2.1:

III) **Classification of the simplices of the convex envelope**

We check for all heterogeneous simplices, if those are isolated and if so, process those to step IV). It is easy to check that this classification does not change the methodology described in [62, 82] for binary, ternary, and quaternary systems.

IV) **Computation of phase splits**

For a given feed with molar fractions \mathbf{x} , we transform \mathbf{x} to cartesian coordinates \mathbf{a} and check if \mathbf{a} lies within an isolated, heterogeneous simplex H (with k phase blocks). If this is the case, we use Theorem 1 to obtain the split ratios $(\lambda_1, \dots, \lambda_k)$ and the cartesian coordinates $\mathbf{p}_1, \dots, \mathbf{p}_k$ of the resulting phases. Using the inverse transformation of coordinates that was used to obtain \mathbf{a} from \mathbf{x} , one can get molar fractions corresponding to each phase.

We provide qualitative and quantitative results regarding the CEM in Section 4.1. Details on the implementation are provided in the Appendix B1.2. This framework allows us to model arbitrary liquid phase splits and thus simulate decanters quickly and robustly.

3.2 Reinforcement Learning Frameworks

3.2.1 General Remarks

This section describes several RL frameworks developed for AFS one after another to address the challenges listed in Section 2.2. In the first part, we will introduce the SynGameZero approach, which models AFS as a competitive two-player game. This setup enforces exploration and allows training an agent for AFS within a discrete action space. In the second part, we describe how to improve the SynGameZero approach’s scalability by integrating the hierarchical decomposition of the action space. In the third part, a generalized RL framework for AFS is presented, which is based on several elements from the SynGameZero approach (e.g., hierarchical decomposition of the action space) but also adapts and further develops features from recently published RL algorithms. This allows a transformation back to a single-player game and the integration of hybrid action spaces.

3.2.2 SynGameZero: Proof of Concept

3.2.2.1 General Idea

Without a doubt, the introduction of AlphaZero [54, 55] marks one of the most important points of the last decade for the research field RL. It allows one to train an agent from scratch the two-player board games chess, shogi, and Go by self-play without using heuristics. To employ similar RL techniques for AFS, the basic idea of the SynGameZero approach is to transform flowsheet synthesis into a competitive two-player game. As AlphaZero was developed for games that only have discrete actions, we restrict flowsheet synthesis to a discrete action space within this section to show the concept to work.

The SynGameZero approach is explained alongside Figure 7. Both players are given the same feed stream(s) and try to create a more profitable flowsheet than the opponent. The game is turn-based, and at each turn, the current player (i.e., player 2 in Figure 7) can take an action (i.e., place a unit operation at an open stream or terminate the flowsheet synthesis) in its own flowsheet. Both players can always see their own and the opponent's flowsheet. The game ends when both players have completed their flowsheets (either by choosing the terminate action or reaching a maximum number of actions). The player with the more profitable flowsheet is the winner, measured by some monetary cost function (e.g., NPV). If the evaluation with the cost function leads to a tied game, then the player that has completed the synthesis first wins the game. Note that this prevents player 2 from copying the moves of player 1. The winner obtains the reward $r = 1$, the loser $r = -1$. The agent switches back and forth between the roles of player 1 and player 2 during the game and thus is trained via self-play.

The SynGameZero approach addresses multiple challenges mentioned in Section 2.2. First, the transformation into a competitive two-player game forces the agent to permanently look for improvements in its current policy, as it will always lose the game in the role of one player. This may not guarantee convergence to a globally optimal policy, but at least ensures continuous exploration for better alternatives. Additionally, AlphaZero was developed for two-player games with sparse rewards, which are only provided at the end of the game. This feature is beneficial for SynGameZero, as it, therefore, is suitable for AFS problems. On the environment side, we address the challenges for the moment by using simplified models for unit operations that always converge and can be simulated fast. The Appendix C1.1 provides a detailed description of those models.

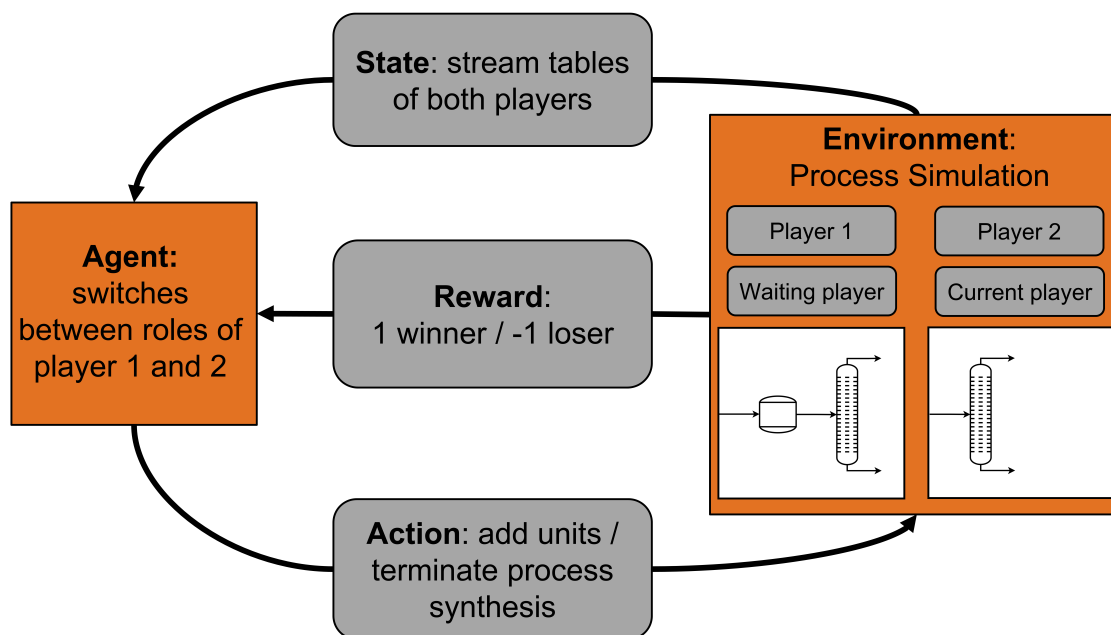


Figure 7: SynGameZero approach for AFS. The agent plays against itself by switching between the roles of players 1 and 2.

3.2.2.2 Agent

Agent Structure

Similarly, as in [54, 55], the agent consists of an ANN that interacts with a MCTS variant. The concept is explained alongside Figure 8.

As input, the agent receives the state of the game, i.e., the flowsheets of both players, which are represented as flowsheet matrices (a description of the generation of those is provided within Appendix CI.2). The ANN obtains the state as a vector, which is constructed as a concatenation of all rows of the flowsheet matrices of the current and the waiting player. Further, a vector with the length of a row of a flowsheet matrix is concatenated. It contains either only zeros (the flowsheet of the waiting player is not completed) or only ones (the flowsheet of the waiting player is completed). The ANN is an actor-critic network (ACN) [37], which means that it generates two kinds of outputs: a policy π (generated by the actor, i.e., the policy-head), which is a suggestion for the next action/decision, and an estimate of the expected reward v (generated by the critic, i.e., the value-head), which is an evaluation of the performance of the actor. The entries in π are in the range $[0, 1]$ and sum up to 1. In π , there is exactly one entry for every possible action for the agent, e.g., place a reactor to stream 2 or terminate the flowsheet synthesis. For such an action i , π_i corresponds to the probability with which the corresponding action should be executed (suggested by the ACN). Different actions

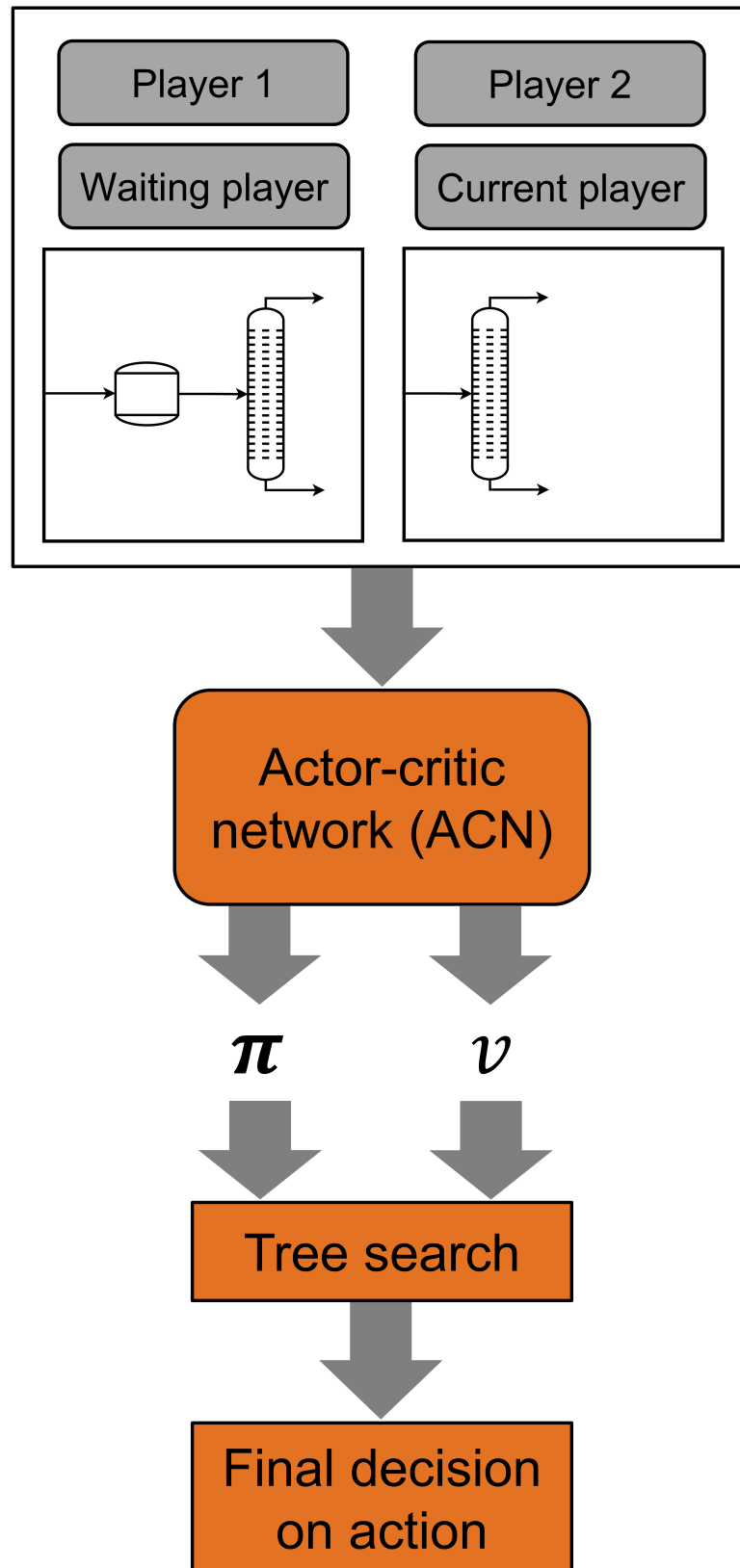


Figure 8: The agent's decision process in SynGameZero.

may be feasible at different game stages; for example, when the agent chooses to place a reactor to stream 1 as the first action, it cannot place any other unit to this stream in subsequent actions. Therefore, $\boldsymbol{\pi}$ is filtered. Entries that correspond to infeasible actions are set to 0. The remaining entries are scaled with a common factor so that the resulting filtered vector \boldsymbol{p} also sums up to 1. The scalar v is in the range $[-1, 1]$ and can be interpreted as an estimate of the current player’s reward (by the ACN) at the end of the game. Details on the implementation and hyperparameters of the ACN are provided within the Appendix CI.3.

As shown in Figure 8, the outputs of the ANN are used as inputs in a tree search that imitates a planning process. As in [54, 55], to avoid extensive computations, the tree search is a variant of MCTS, which is adaptive in depth and does not use a complete enumeration of all actions. Only promising actions are explored, where the values of \boldsymbol{p} and v are used to quantify the word promising. The tree search results are used to train the ANN and choose an action in the game. Afterward, the agent switches roles, and the same process repeats until both players finish their flowsheets. When one player finishes its flowsheet (either by choosing the terminate action or by reaching the maximum number of possible units that the flowsheet matrix can contain), the agent does not take the role of this player anymore. It just stays in the role of the player with the uncompleted flowsheet until this is finished as well. The tree search and the action selection are explained in detail in the following.

MCTS and Action Selection

To improve its performance, the agent does not always select the action with the highest entry in \boldsymbol{p} . Instead, \boldsymbol{p} and v are used as the basis for a tree search to plan several actions in advance. The tree search imitates a typical human planning process before finally deciding on an action. The tree search is explained alongside Figure 9 and an example in which the agent (for the sake of simplicity) has only three possible actions named T, D₁, and R, say terminating flowsheet synthesis, placing a distillation column on the first open stream, or placing a reactor on the first open stream. The tree consists of nodes and branches. The nodes n correspond to the states of the two-player game. The branches (n, a) correspond to the action a that the current player takes at node n . The origin of the tree is the root node I. In Figure 9, the root node I corresponds to the beginning of the game when both players have an empty flowsheet. The current player is the one who takes the next action. Its flowsheet is shown in the left half of the nodes. Since the game is turn-based, the order of the two flowsheets is switched after every action. Each node is either an explored node (e.g., node I in Figure 9) or an unexplored leaf node (e.g., node III). A node is explored if and only if the corresponding state of the game is known. To explore an unexplored leaf node, i.e., to obtain its state, the

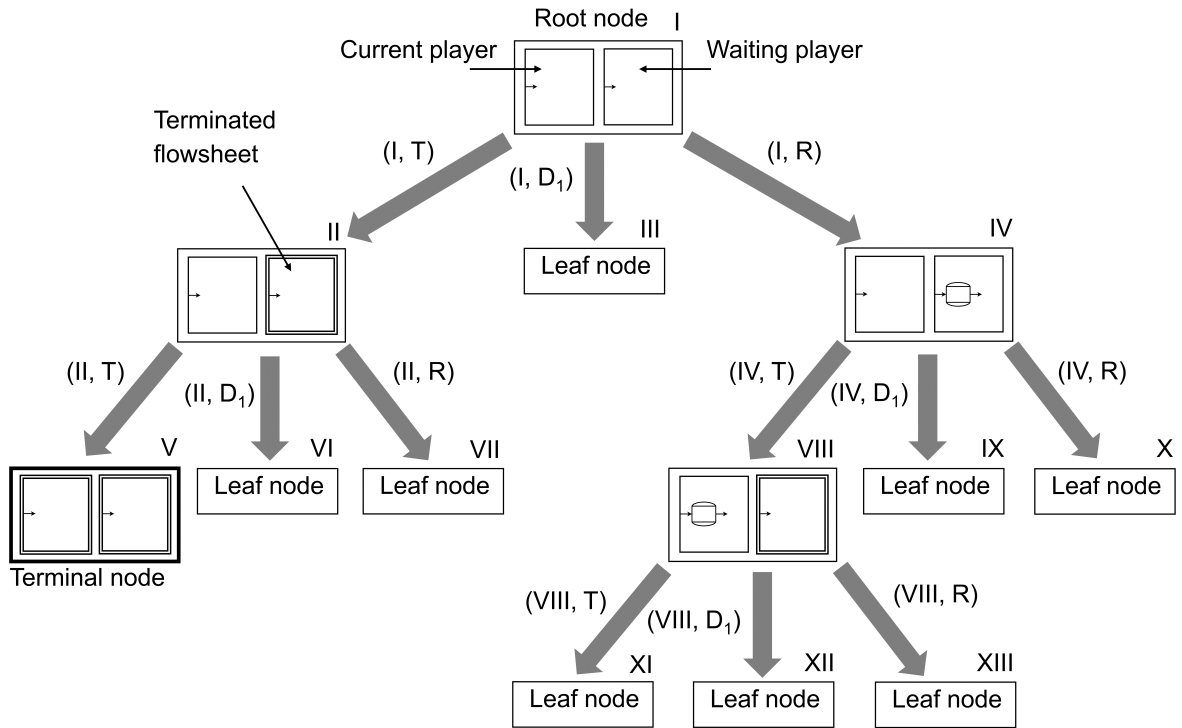


Figure 9: Example tree search at the beginning of the game (flowsheets of both players empty) with three possible actions: T, D_1 , R.

respective action has to be applied to the flowsheet of the current player in the parental node, and the flowsheet has to be evaluated in the simulation. For example, if node III in Figure 9 has to be explored, then action D_1 (adding a distillation column) has to be applied to the left flowsheet in node I. Afterward, node III would consist of the updated flowsheet matrix of the current player of node I and the unaltered flowsheet matrix of the waiting player of node I. The order of the matrices would also switch, as now it is the other player's turn. Whenever a node is explored, it is checked whether it is terminal, i.e., a node in which both flowsheets are terminated (e.g., node V in Figure 9). The termination of a flowsheet may be caused either by the action T (terminate) or by a full flowsheet matrix. The node is not terminal if at least one player has a non-terminated flowsheet (for example, node VIII). In this case, the branches of all feasible actions and the corresponding (unexplored) leaf nodes are added to the tree below that node.

Four variables ($N_{(n,a)}, W_{(n,a)}, Q_{(n,a)}, P_{(n,a)}$) are stored for every branch (n,a) in the tree. The variable $N_{(n,a)}$ counts how often this branch has been taken during the tree search. The variable $W_{(n,a)}$ is the sum of all estimated and obtained rewards beneath that branch, and $Q_{(n,a)}$ is defined as $W_{(n,a)}/N_{(n,a)}$. The values of $P_{(n,a)}$ are set to the corresponding value of the vector \mathbf{p} that is obtained by feeding the state of the node

n into the ANN. These variables are updated while the tree is constructed. They also guide both the tree extension and the agent's final decision. A new tree is initialized only once at the beginning of every game. A root node with the state vector of two empty flowsheets is placed (node I in Figure 9) and explored. The variables $N_{(n,a)}$, $W_{(n,a)}$ and $Q_{(n,a)}$ are set to 0 for the resulting branches, while the values of $P_{(n,a)}$ are obtained as described above. The tree search proceeds then in four steps:

I) **Select**

The algorithm starts at the root node and runs down one path through the tree until it arrives at a leaf node or a terminal node n_{bottom} . At each node n , the algorithm greedily selects to follow the branch (n, a) that maximizes $Q_{(n,a)} + U_{(n,a)}(\alpha)$. $U_{(n,a)}(\alpha)$ is defined as follows:

$$U_{(n,a)}(\alpha) = P_{(n,a)} \frac{\sqrt{\sum_{b \in A} N_{(n,b)}}}{N_{(n,a)} + 1} \quad \text{if } Q_{(n,a)} > \alpha, \quad (4)$$

$$U_{(n,a)}(\alpha) = 0 \quad \text{if } Q_{(n,a)} \leq \alpha, \quad (5)$$

where A is the set of all actions at this node. If there are two or more branches that maximize $Q_{(n,a)} + U_{(n,a)}(\alpha)$, the branch among them with the largest value of $P_{(n,a)}$ is taken. To enhance exploration, the above greedy selection policy is replaced during training for the root node (and only the root node) by an ϵ -greedy policy [37]. With a probability of ϵ , an entirely random (i.e., uniform distribution) branch is selected. With a probability of $1 - \epsilon$, the algorithm selects the branch using the above greedy policy that maximizes $Q_{(n,a)} + U_{(n,a)}(\alpha)$. In the present work, ϵ is set to 0.2 and α to -0.9 .

II) **Explore and/or evaluate**

If the node n_{bottom} that was found in step I) is an unexplored leaf node, then it is explored, and the resulting state is stored in n_{bottom} . Two cases might occur: Case 1: The node n_{bottom} is terminal. The winner of the game is determined by comparing the NPVs of both flowsheets. The reward is determined for the current player of the node n_{bottom} and stored in the variable V for step III). Case 2: The node n_{bottom} is not terminal. In this case, the state of the node n_{bottom} is fed into the ANN. The value v that the ANN calculates is stored in the variable V for step III).

III) **Backup**

Starting at the node n_{bottom} that was found in step I), the algorithm runs back upwards the tree until the root node. For every branch (\tilde{n}, \tilde{a}) passed on the way,

the following updates are made to the branch variables:

$$N_{(\tilde{n},\tilde{a})} = N_{(\tilde{n},\tilde{a})} + 1, \quad (6)$$

$$W_{(\tilde{n},\tilde{a})} = W_{(\tilde{n},\tilde{a})} + tV, \quad (7)$$

$$Q_{(\tilde{n},\tilde{a})} = W_{(\tilde{n},\tilde{a})}/N_{(\tilde{n},\tilde{a})}. \quad (8)$$

Herein, V is the value that has been stored in step II). The variable t takes into account that the agent switches players after every action when playing against itself. If V has been determined at a node where player 1 is the current player, then V is added ($t = 1$) at all branches that represent the actions of player 1. At the other branches, which represent the actions of player 2, V is subtracted ($t = -1$). The opposite is done when V has been determined at a node where player 2 is the current player.

IV) Play

Steps I)-III) are repeated K times as a loop before the agent finally decides on an action at the tree's root node. The decision is based on a probability vector \mathbf{y} with one entry for every feasible action at the root. The entry for action a is calculated by:

$$y_a = \frac{N_{(n_{\text{root}},a)}}{\sum_{b \in A} N_{(n_{\text{root}},b)}}. \quad (9)$$

During training, the decision in step IV) is made by randomly selecting an action using the vector \mathbf{y} as a probability distribution. After training, the moves are chosen greedily by always selecting the action a with the largest $N_{(n,a)}$. If there are two or more branches that maximize $N_{(n,a)}$, then the algorithm selects the branch among them with the largest value for $P_{(n,a)}$. The tree is shifted downwards after the action is applied to the environment. The node that is reached by the selected action becomes the new root node. The tree is cut off above. The values of the branch variables are retained.

The tree search algorithm is briefly interpreted in the following. The algorithm generally selects actions with large values of $N_{(n,a)}$. $N_{(n,a)}$ counts how often the branch has been taken during step I). For the algorithm's success, selecting promising actions in this step is crucial. This selection is based substantially on the value of $Q_{(n,a)} + U_{(n,a)}(\alpha)$, which is an estimate of the value of action a at node n , i.e., the value of the state, which is reached by selecting a . This value can only be determined at a terminal node. At other nodes, it is estimated by the ANN (value v). In step III), the best available guess for this value is backed up along the search path to improve the estimation of $Q_{(n,a)}$ as it is the average of those guesses. If the path in step I) would only depend

on $Q_{(n,a)}$, then a wrong estimate of $Q_{(n,a)}$ at the beginning of the tree search might lead to inefficient exploration. Therefore, the function $U_{(n,a)}(\alpha)$ is also considered. This function is large for actions a that have a large value $P_{(n,a)}$ but a small value $N_{(n,a)}$ compared to $\sum_{b \in A} N_{(n,b)}$. These actions are favored by the ANN but have not been explored so far. If these promising actions are not explicitly considered, they might be overlooked by chance at the beginning of the tree search. Later in the tree search, if such an action has turned out to be not constructive (the estimate $Q_{(n,a)}$ falls below the threshold α that is typically chosen rather low, e.g., -0.9), then the exploration function $U_{(n,a)}(\alpha)$ is no longer considered. This ensures that shortsighted recommendations of the ANN do not bias the tree search in the long run.

Training Process

The goal of the training process is to adjust the parameters of the ANN so that the ANN ideally predicts the consequences of a potential action up till the end of the game. Ideally, the ANN outputs a value v that correctly predicts the chances of the current player winning the game. The output \mathbf{p} should ideally be a sharp distribution with a maximum at the action that maximizes the chances of winning the game. At the start of the training process, the ANN is initialized with random weights. In training, the agent plays a large number of games against itself. The given feed stream(s) in the games are varied randomly to obtain an agent that can solve a broad class of problems. The search tree is initialized with the given feed(s) at the beginning of every game. Then, the agent plays the game until the end (both players terminated their flowsheets). Thereby, every decision that had been made in step IV) of the tree search is stored (the state \mathbf{s} at the root node, which served as input for the ANN, and the vector \mathbf{y} of the decision). After finishing the game, this data is augmented by the final reward r obtained at the end of the game. The tuples of the form $(\mathbf{s}, \mathbf{y}, r)$ are used to train the ANN.

We provide results for the proof of concept for SynGameZero in Section 4.2. Details on the implementation are provided within the Appendix CI.3.

3.2.3 SynGameZero: Integration of Hierarchical Reinforcement Learning

3.2.3.1 General Idea

In the original SynGameZero approach, the agent chooses actions from a flat action space, i.e., the next process unit, its location, and specifications were decided on simultaneously. Such a structure of the action space scales poorly when more sophisticated

problems are considered, e.g., flowsheets with recycle streams or continuous process parameters. The action space would grow exponentially, and the RL problem would become intractable due to combinatorics.

To prevent this, we integrate hierarchical RL into SynGameZero in this section. Hierarchical RL is an RL subfield with the general idea of splitting the agent’s decisions into several levels [84, 85]. We have defined a decision hierarchy for flowsheet synthesis consisting of three levels. At level 1, it is decided whether to terminate or, alternatively, select an open stream to manipulate. At level 2, it is decided on the unit operation to add to the flowsheet at the stream identified at level 1. At level 3, the specification of the chosen unit operation is determined. In this section, the agent sets only discrete specifications at level 3: the destinations of mixers and recycle streams. Levels 2 and 3 are only evoked if needed. The general concept of SynGameZero, i.e., the two-player game as shown in Figure 7, is employed on top of this hierarchical structure. The agent consists of a series of connected ANNs to provide suggestions for the decisions on every level and a tree search, which is used for planning and decision-making.

The approach is demonstrated by training an agent to design processes for ethyl-tert-butyl-ether (ETBE) synthesis [62, 64]. This process example displays a significantly increased level of complexity compared to the problems considered in the original SynGameZero approach, as the agent has to deal with azeotropic behavior, equilibrium reactions, and recycle streams. To provide a robust and fast simulation environment (challenges I) and II) on the environment side in Section 2.2), we employ short-cut models for the unit operations, which always converge, except for recycle streams. The Appendix DI.1 provides a detailed description of the available unit operations. While the action space in this example is still purely discrete, the integration of hierarchical RL provides a starting point to allow the agent to work in a hybrid action space (challenge III) on the agent side in section 2.2) in a subsequent step.

3.2.3.2 Agent

Agent Structure

The overall structure of the agent accounts for the hierarchical decomposition of the agent space. It is explained alongside Figure 10. The flowsheet matrices (for details, see Appendix DI.2) of both players are processed by a convolutional neural network (CNN) [86]. Each hierarchy level is represented by an ACN, which receives data processed by the previous networks and information on the decisions at upper hierarchy levels (this applies only to level 2 and level 3). Each ACN i generates two kinds of outputs π_i and

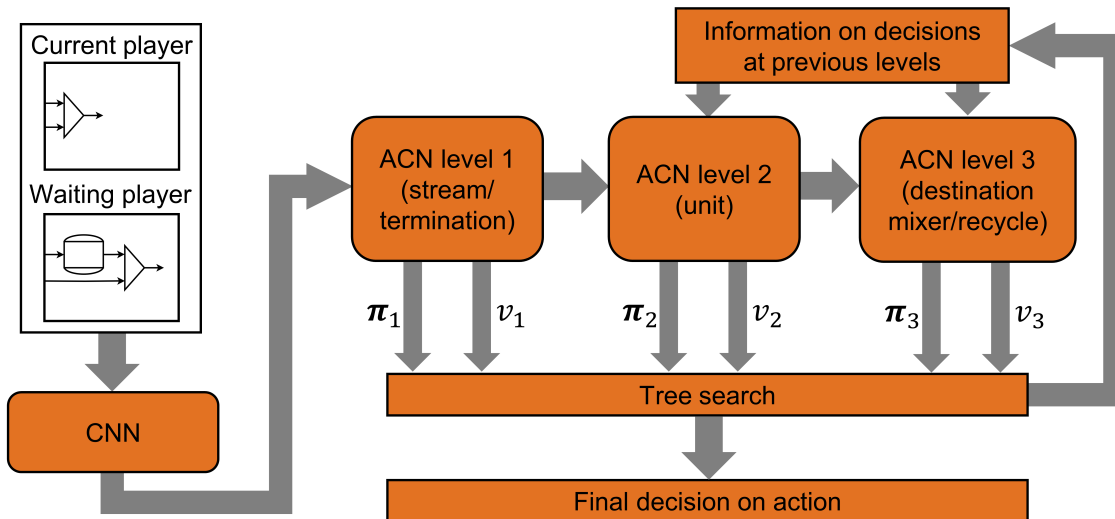


Figure 10: The agent's hierarchical decision process.

v_i , which, similarly as before, represent a suggestion for the decision at this level and an estimate of the expected reward for the current player, respectively. Those outputs guide the tree search by exploring promising flowsheet alternatives. As in the original SynGameZero approach, the final decision on an action is based on the tree search results.

ANN Architecture

The ANN architecture is shown in Figure 11. The state \mathbf{s} consists of the flowsheet matrices of both players and is processed by a CNN. Its output is mapped into a vector representation, which serves as input for the subsequent ACNs. At every level i , the respective ACN outputs π_i and $v_i \in [-1, 1]$, which will be used to guide the tree search. The entries in π_i sum up to one and represent a probability distribution, which can be interpreted as a suggestion for the decision at level i (the number of entries in π_i depends on the level i , e.g., one entry for every possible unit operation on level 2). The scalar v_i is an estimate of the reward. Before entering the tree search, the vectors π_i are filtered similarly as in the original SynGameZero approach: all entries, which correspond to infeasible decisions, are set to zero, and the resulting vector is divided by the sum of its elements so that its entries sum up to one again. It is important to mention that only decisions impossible to simulate are filtered this way (e.g., placing a reactor to a stream where already another unit is connected). There is no filter for decisions that are possible but not useful (e.g., placing a distillation column to a stream, which only consists of a pure component). The resulting vectors are referred to as \mathbf{p}_i .

In the vector π_1 , there is one entry for every possible stream in the flowsheet of the current player. Additionally, it contains one entry, which represents the termination

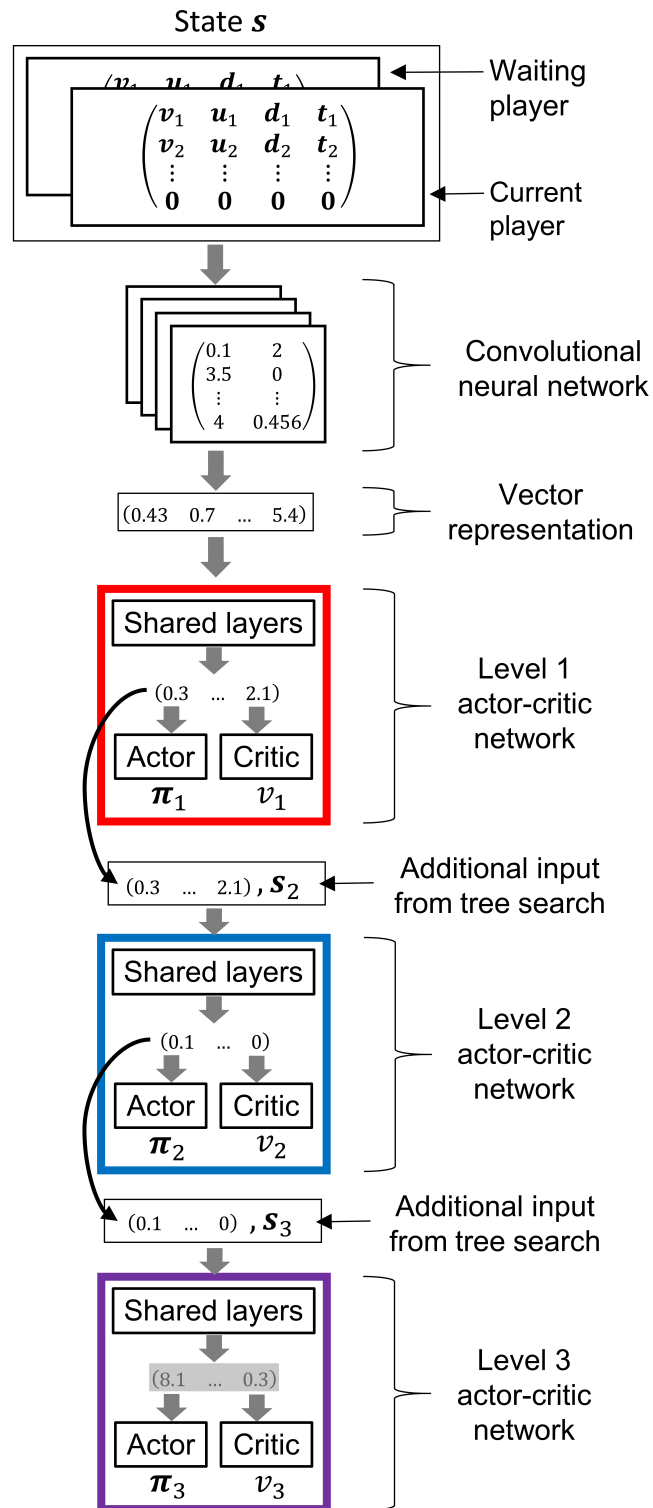


Figure 11: Hierarchical ANN structure of the agent.

action. Together with v_1 , it is used in the tree search to make a decision at level 1. If termination of the flowsheet synthesis is chosen, there is no requirement for further evaluation by the subsequent ANNs. Otherwise, the output vector of the shared layers of the ACN at level 1 is processed to the ACN at level 2. It is concatenated with a vector \mathbf{s}_2 , an OHE of the stream chosen by the tree search at level 1. In the vector $\boldsymbol{\pi}_2$, there is one entry for every possible unit operation. Together with v_2 , it is used in the tree search to determine a unit. If a reactor or distillation column is chosen, the action is fully specified and can be applied to the flowsheet of the current player. If a mixer or recycle is selected as a unit, level 3 is required to decide where to mix or recycle the chosen stream. The ACN at level 3 receives an input, which consists of the output vector of the shared block of the ACN at level 2 concatenated with a vector \mathbf{s}_3 of the same length, which indicates whether mixer or recycle was chosen (the vector is either filled with zeros or ones and provided by the tree search). The entries in $\boldsymbol{\pi}_3$ represent a probability distribution, a suggestion, where to admix or recycle the chosen stream, respectively. Similar as before, $\boldsymbol{\pi}_3$ and v_3 are used to guide the tree search, and based on its results, a decision at level 3 is determined.

Details on the ANN architecture’s implementation and hyperparameters are provided in the Appendix DI.3.

MCTS and Action Selection

In general, the tree search concept is quite similar to the original SynGameZero approach. At the beginning of each game, the tree is initialized with a sole root node and used by both players to guide their decisions throughout the game. An example of a tree is shown in Figure 12. For a more straightforward illustration, the example tree shows flowsheets where only reactors (R) and mixers (M) are available as unit operations. The tree consists of nodes n and branches (n, a) , where a refers to the action decided on at node n . The root node I represents the current state of the game. Each node is either explored and thus refers to a state \mathbf{s} (the flowsheet of the current player is always displayed to the left), for example, node II, or it is an unexplored leaf node, such as node III. If a flowsheet is terminated (T), this player can no longer take any actions during this game and has to wait until the opponent’s flowsheet is finished. If both players have finished their flowsheets (node V), a terminal node is reached: no more nodes are connected below that node. The color of an explored node shows the respective decision level (red: level 1, blue: level 2, purple: level 3, black: terminal node, no more actions required). As the state \mathbf{s} only changes if a unit is placed to an open stream or the synthesis is terminated, details are only shown in the nodes at level 1 in Figure 12. A leaf node is explored by evaluating the corresponding decision from its parental node in the process simulation. This leads either to an actual change in the

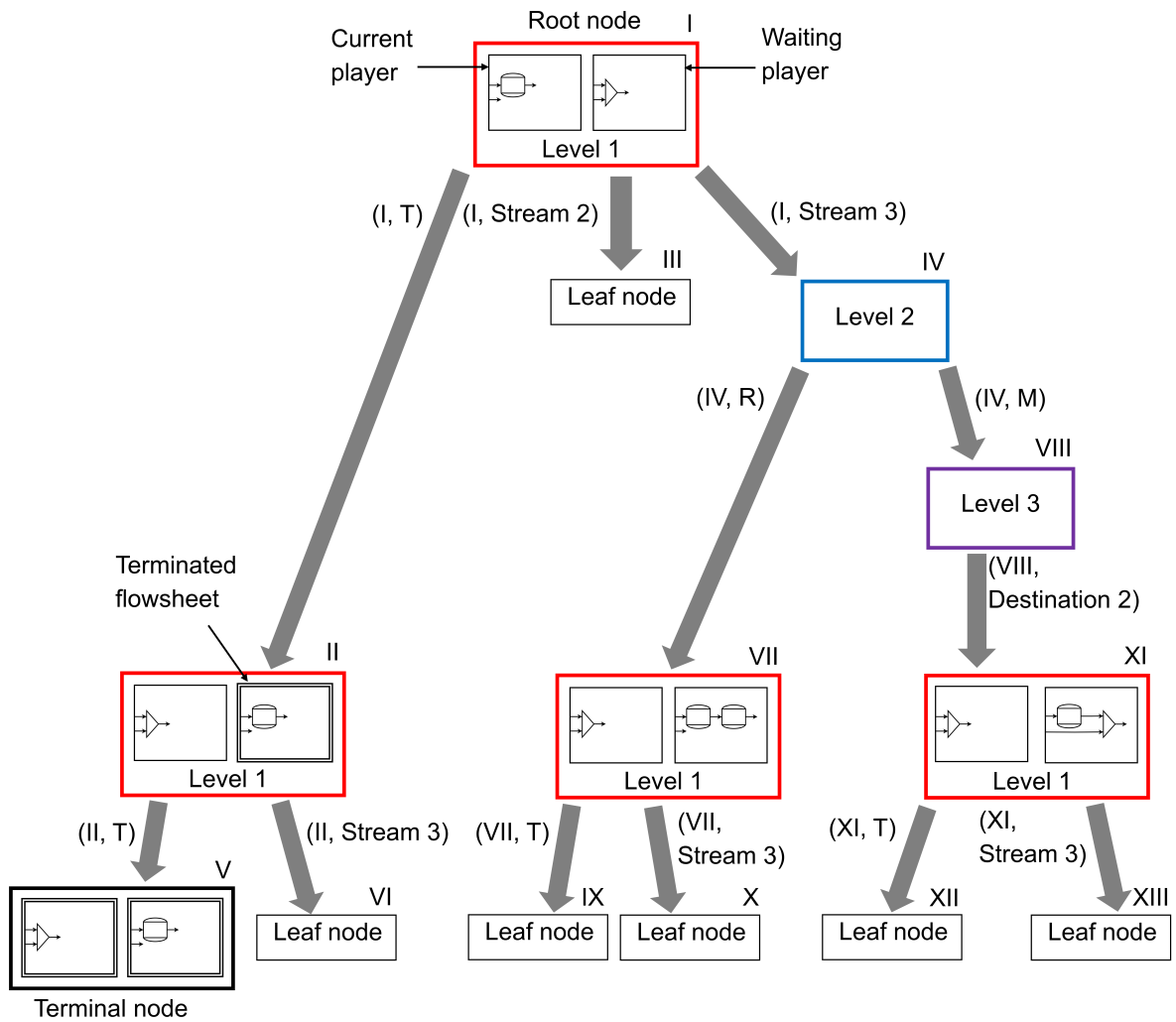


Figure 12: Example structure of the search tree with integrated hierarchy levels.

flowsheet of the current player, as can be seen at the transition from node IV to VII, where the turn changes after the reactor is placed. Or another decision is required to determine the unit or its specification, as can be seen at the transition from node IV to VIII (the turn does not switch as the agent has to decide at node VIII where to mix the chosen stream).

Whenever a leaf node is explored, the corresponding state \mathbf{s} is stored there, and it is checked whether the node is terminal. New branches corresponding to available actions and leaf nodes are connected if it is not a terminal node. At each branch (n, a) four variables $(N_{(n,a)}, W_{(n,a)}, Q_{(n,a)}, P_{(n,a)})$ are stored. $N_{(n,a)}$, $W_{(n,a)}$, and $Q_{(n,a)}$ are initialized with zero, while $P_{(n,a)}$ is set to the corresponding entry in \mathbf{p}_i , which is generated with the ACN at level i . These variables are updated during the tree search and guide the expansion and ultimate decisions made in the game. The tree search proceeds in four

steps, which are the same as in the original SynGameZero framework, only differing in the first step: the ϵ -greedy policy [37] is used at every node with ϵ constantly set to 0.1. The parameter K , which controls the depth of the tree, is set to 30. During the tree search, the simulation checks if the flowsheet is feasible whenever a leaf node is explored by placing a recycle stream somewhere in the flowsheet. If the simulation signalizes divergence, the corresponding node is cut off the tree and deleted. This prevents the generation of an infeasible flowsheet, as all decisions during the game are based on the tree search.

After the tree search, a final decision at the current root node is determined using a probability vector \mathbf{y}_i (i refers to the level of the root node) with one entry for every available branch. \mathbf{y}_i is generated by the tree search algorithm based on the visit counts $N_{(n,a)}$ of all branches starting from the root. As before, the decision is randomly made using \mathbf{y}_i as probability distribution during training. After training, the moves are chosen greedily by always selecting the largest entry of \mathbf{y}_i . When the decision has been made, the tree is shifted downwards. The node reached through the chosen branch becomes the new root node. The tree is cut off above, while the parts below the new root are retained. Note that nodes at level 2 or 3 may also become the root node. Details on the implementation of the training procedure are provided within the Appendix DI.3.

3.2.3.3 Variation of the Hierarchical Framework

In the framework described above, the agent has to distinguish between mixing two open streams and recycling an open stream to a closed stream at hierarchy level 2. At level 3, the agent decides on the destination of mixers and recycles. Both unit operations are conceptually the same, as in both cases, an open stream is mixed with another stream, which is either open or closed. Therefore, we decided to implement a variation of the hierarchical framework where the process simulation determines automatically if the chosen action leads to a mixer or a recycle and simulates the respective option. Compared to the hierarchical framework described before, there are only two differences in the implementation of this variation. On the one hand, the agent has one option less to choose from at level 2, as mixer and recycle have been combined into one option. On the other hand, the ACN at level 3 does not receive the additional input \mathbf{s}_3 , which would indicate if a mixer or recycle has been chosen at level 2 (as in the original hierarchical framework).

We provide quantitative and qualitative results for both variations of the hierarchical framework in Section 4.3.

3.2.4 Single-Player Reinforcement Learning Framework for Automated Flowsheet Synthesis

3.2.4.1 General Idea

In the previous sections, AFS was reformulated as a two-player game to be able to adapt powerful RL algorithms [54, 55] for this specific problem class. The case studies were restricted to discrete action spaces and limited to one chemical system per training procedure. In this section, we omit this reformulation into a two-player game and propose a single-player RL framework, which enables training an agent from zero knowledge to synthesize near-optimal flowsheets for multiple chemical systems, each with varying feed compositions, each requiring a substantially different conceptual approach. Contrary to the SynGameZero approach, the agent can choose from a broad range of unit specifications (continuous ranges are discretized and factorised to cover all possible options). This is possible due to integrating several features of recently published work on RL algorithms and ANN architectures.

Similarly, as in the previous sections, the flowsheet is represented as a matrix (details are provided in Appendix EI.2). The resulting state is encoded by an architecture based on the MLP-Mixer [87] - diverted from computer vision - which means that we propose to view flowsheet synthesis encoding as a sequence-to-sequence problem. This architecture maintains a global receptive field of all streams at all times. It can distill the dynamics of the environment such that the policy network alone can produce excellent flowsheets. The agent’s action space is structured similarly to the hierarchical SynGameZero approach, with the only exception being that level 3 is now separated into several sublevels, which account for choosing the specifications for the unit operations. The agent’s decision process is supported by a tree search, which is based on the algorithmic Gumbel extension [88] of the AlphaZero framework [54, 55].

Due to its complex nature, we take the separation of azeotropic mixtures as an example task for the agent. The agent is trained simultaneously on several feed compositions from four chemical systems: acetone – chloroform, ethanol – water, butanol – water, and pyridine – water. It learns to add solvents, combine them with distillation columns, decanters, and mixers, and place crucial recycle streams. This way, without prior knowledge, it discovers classical chemical engineering schemes such as azeotropic or entrainer distillation. Details on the chemical systems and the environment, which bases the flowsheet simulation as before on short-cut models, are provided within Appendix EI.1. In summary, the framework presented in this section addresses all the challenges listed in Section 2.2. To our knowledge, it is the first RL approach that enables training an

agent without prior knowledge to synthesize flowsheets within a hybrid action space for several chemical systems. Therefore, this framework marks a significant step towards generality, i.e., an agent that can transfer its learnings from the training process onto conceptual design problems it has yet to encounter.

3.2.4.2 Agent

Agent Structure

There are three general hierarchy levels conceptually defined similarly to the hierarchical SynGameZero approach. At level 1, the agent can terminate the flowsheet synthesis or select an open stream in the current flowsheet. If a stream is selected, the agent transitions to level 2, where it must choose a unit operation for the selected stream from a pool of available units. This unit operation is further specified (for mix/recycle stream) in level 3a or (for continuous specification of distillation columns/add solvent) in level 3b, which has another sublevel 3c. The unit operations and their specifications are explained in detail in the Appendix EI.1.

The overall structure of the agent is displayed in Figure 13. In a first step, the flowsheet matrix is encoded using an MLP-Mixer architecture [87]. This latent representation of the flowsheet is processed by several policy-heads and a single value-head to generate π_i and v . There is a policy-head for each hierarchy level 1, 2, 3a, and 3b. Note that policy-head 3b is used to generate π_{3b} and π_{3c} , as those outputs have the same shape and are related to each other as will be explained later. As in previous sections, the entries in $\pi_i \in [0, 1]$ sum up to 1 and are interpreted as suggestions for the next action. $v \in [0, 1]$ is an estimate of the expected reward (we norm and clip the final objective so that it is in the range $[0, 1]$; for details, see Appendix EI.3). The outputs π_i and v guide the tree search by exploring promising flowsheet alternatives. As in the SynGameZero approach, the final decision on an action is based on the tree search results.

ANN Architecture

AlphaZero-type algorithms stand and fall with the underlying neural network. In our case, it must provide the agent with latent embeddings of the flowsheet streams that capture the essence of the current state and allow the agent to derive accurate policy and value predictions. Hence, the network always needs to maintain a global view of all streams and their connections to each other, as choosing to recycle a stream can alter all present streams simultaneously. Convolutional networks and graph neural networks generally struggle to capture long-range dependencies. Therefore, we pose the task of obtaining stream representations as a sequence-to-sequence problem. Using an affine

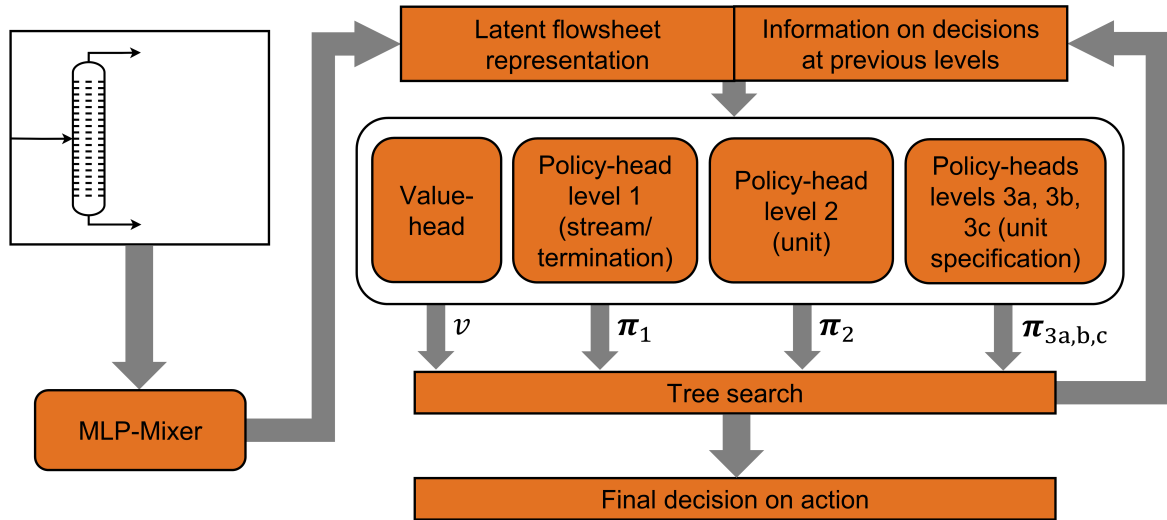


Figure 13: The agent’s decision process.

embedding, every row of the flowsheet matrix is mapped into some latent space. The resulting sequence is input for an ANN based on the MLP-Mixer [87], transforming it into an expressive sequence of latent stream embeddings. Although the MLP-Mixer might seem an unorthodox choice as it is classically used in computer vision, it has a global receptive field (as in self-attention based transformers [89]), but with only linear complexity in the number of matrix rows (as opposed to the quadratic complexity of transformers). Furthermore, it respects the sequential procedure of placing unit after unit on the flowsheet. Details on the ANN architecture and implementation are provided in Appendix EI.3.

MCTS and Action Selection

The tree search works in a similar way as in SynGameZero, but additionally, some features from the Gumbel extension [88] of the AlphaZero framework [54, 55] were adapted. Therefore, we provide a brief explanation here and refer to [88] and Appendix EI.3 for the details. In the search tree, nodes n represent states, and branches represent actions a . As in AlphaZero, a single search simulation from a root node traverses the tree until a leaf node is reached, which is evaluated by the neural network (resulting in a value v), after which the node is expanded using π_i (in the same way as in the SynGameZero approach) and the predicted value v is recursively backed up the trajectory. We store for all branches the search statistics $N_{(n,a)}$ and $Q_{(n,a)}$, where $N_{(n,a)}$ denotes the visit count and $Q_{(n,a)}$ is the estimated action value (i.e., accumulated backed up values averaged over $N_{(n,a)}$). As explained below, the main difference between Gumbel AlphaZero [88] and the original AlphaZero [54, 55] framework is the way actions are selected during the tree search.

Given a root node n (at hierarchy level i), we sample a maximum of $m = 16$ actions without replacement from the predicted policy π_i using the Gumbel-Top-k trick [90, 91]. We denote by $\text{logit}_{\pi_i}(a)$ the (unnormalized) log-probability of an action a and by $G(a)$ its sampled Gumbel noise. Using a Sequential Halving [92] procedure, a predefined budget of simulations is evenly distributed between the sampled actions, and multiple search simulations are started from each sampled action. After each Sequential Halving level, the considered root actions are pruned to the top $m \leftarrow \frac{m}{2}$ actions according to their scores:

$$G(a) + \text{logit}_{\pi_i}(a) + \sigma(Q_{(n,a)}), \quad (10)$$

where σ is the monotonically increasing linear map

$$\sigma(Q_{(n,a)}) = (50 + \max_b(N_{(n,b)})) \cdot Q_{(n,a)}, \quad (11)$$

matching the choice in [88]. When the search budget is exhausted, the agent chooses the action from the remaining unpruned actions that maximizes Equation 10. Throughout this example, we grant a budget of $K = 200$ simulations at a root node (during training and evaluation). With 16 sampled actions, this amounts to four Sequential Halving levels with 16, 8, 4, and 2 remaining root actions, where each remaining action is allocated 3, 6, 12, and 28 simulations.

The MCTS aims to construct an improved policy $\hat{\pi}_i$, which supports the agent’s decision and serves as a training target. It is constructed in a two-step process. First, a ‘completed Q-value’ $\hat{Q}_{(n,a)}$ is defined for all actions a by setting

$$\hat{Q}_{(n,a)} = \begin{cases} Q_{(n,a)} & \text{if } N_{(n,a)} > 0, \\ \hat{v}(\mathbf{s}) & \text{else.} \end{cases} \quad (12)$$

Herein, \mathbf{s} refers to the state corresponding to node n and $\hat{v}(\mathbf{s})$ is defined as the interpolation

$$\hat{v}(\mathbf{s}) = \frac{1}{1 + \sum_b N_{(n,b)}} \left(v(\mathbf{s}) + \frac{\sum_b N_{(n,b)}}{\sum_{b, \text{ s.t. } N_{(n,b)} > 0} P_{(n,b)}} \sum_{b, \text{ s.t. } N_{(n,b)} > 0} P_{(n,b)} Q_{(n,b)} \right). \quad (13)$$

Herein, $P_{(n,b)}$ is equal to the entry of π_i (policy-head output at the state \mathbf{s} of the node n) that refers to action b . Then, $\hat{\pi}_i$ is constructed by setting

$$\hat{\pi}_i = \text{softmax}(\text{logit}_{\pi_i}(\cdot) + \sigma(\hat{Q}_{(n,\cdot)})). \quad (14)$$

Informally, the improved policy $\hat{\pi}_i$ increases the logit of an action where the search

tells us that it leads on average to higher returns than expected and decreases the logit otherwise, giving zero advantage to unvisited actions. As in [88], whenever computing Equation 10 or Equation 14, we normalize the Q-values with a min-max normalization according to all Q-values encountered in the search so far.

During the search, at a non-root node \tilde{n} , we compute $\hat{\pi}_i$ according to Equation 14 and select an action from

$$\arg \min_a \sum_b \left(\hat{\pi}_{i,b} - \frac{N_{(\tilde{n},b)} + \mathbf{1}_{\{a\}}(b)}{1 + \sum_c N_{(\tilde{n},c)}} \right)^2, \quad (15)$$

where $\mathbf{1}_{\{a\}}(b) = 1$ if $a = b$, and zero otherwise. This deterministic action selection intentionally chooses the action that shifts the visit count distribution closer to $\hat{\pi}_i$.

Further details on using the results of the MCTS for the training procedure are provided within Appendix EI.3.

Pruning Procedure for Diverging Recycles

In flowsheet synthesis, one can generally distinguish between two types of infeasible actions. First, there are infeasible actions that come directly from the definition of the environment. Exhaustively, these are all closed streams in hierarchy level 1, 'add solvent' in hierarchy level 2 if a solvent has already been added, all closed streams in hierarchy level 3a when mixing streams (as an open stream can only be mixed with another open stream) and all open streams in level 3a when recycling a stream (as an open stream can only be recycled to a closed one). The second type of infeasible action occurs when the flowsheet simulation does not converge. These actions are not known upfront and must be simulated during the tree search. As in the case study for the hierarchical SynGameZero framework, recycles are the only unit operations in our framework that can lead to a divergent flowsheet simulation. Additionally, recycles are a unique action when setting up a flowsheet sequentially, as contrary to other unit operations, they can alter all streams of the process and drastically change its overall dynamics. Therefore, the challenge is how to teach the agent to avoid placing diverging recycles without learning to avoid recycles overall.

We solve this problem directly in the MCTS by pruning the corresponding node from the tree whenever a search simulation reaches a failed flowsheet and continuing the search from the simulation's last feasible actions. The main difference to the corresponding procedure in the hierarchical SynGameZero approach is that the pruning process is now repeated recursively. Whenever a divergent recycle at a state \mathbf{s} on level 3a is encountered during a search simulation, its logit is set to $-\infty$ and the action selection at \mathbf{s} is repeated, where the improved policy $\hat{\pi}_i$ is recomputed. If all actions at \mathbf{s} lead

to a diverging simulation, the node \mathbf{s} and its subtree is pruned, and the action from the parent of \mathbf{s} leading to \mathbf{s} is set as infeasible, repeating the MCTS procedure from the parent of \mathbf{s} . As the pruning is applied recursively, it only takes a small number of search simulations such that the agent never chooses to place an infeasible recycle on the flowsheet.

While this simple procedure leads to longer search times at the beginning of training, the search tree statistics directly reflect the pruning from which the policy is trained via Kullback-Leibler divergence [86] using the improved policy $\hat{\pi}_i$ as training target (details are provided in Appendix EI.3). Furthermore, as divergent actions are directly reflected in the improved policy, their infeasibility will be recursively accounted for in $\hat{\pi}_i$ at the root state, so the network is trained to give lower probability to these actions. In effect, the number of times the tree pruning needs to be executed diminishes as training progresses, while at the same time not teaching the agent to avoid recycling streams overall.

Factorised Discretization of Continuous Actions (Level 3b and Level 3c)

As mentioned, the agent chooses continuous unit specifications from factorisation of a discretized range at the hierarchy levels 3b and 3c. The procedure is explained in the following.

We discretize continuous actions from an interval $[c, d] \subseteq \mathbb{R}$ with a two-step factorisation. The agent chooses a discrete tuple $(m_1, m_2) \in \{1, \dots, L_1\} \times \{1, \dots, L_2\}$, where $L_1, L_2 \in \mathbb{N}$ are predefined integers. The tuple (m_1, m_2) translates to the element (which is the unit specification used for the flowsheet simulation)

$$c + (m_1 - 1) \cdot \frac{d - c}{L_1} + (m_2 - 1) \cdot \frac{d - c}{L_1 L_2} \in [c, d]. \quad (16)$$

In particular, the agent chooses (m_1, m_2) sequentially by first picking m_1 at action hierarchy level 3b and then m_2 at level 3c.

This factorisation is appealing as the agent can first make a coarse-grained decision via m_1 and then refine it with m_2 . Thus, the agent learns more quickly which interval ranges are suitable instead of choosing from a single discrete distribution of size $L_1 \cdot L_2$. Throughout this example, we set $L_1 = L_2 = 7$ everywhere, effectively dividing the interval $[c, d]$ into 49 evenly spaced actions.

We provide results for the single-player framework in Section 4.4.

4 Results and Discussion

4.1 Modeling of Decanters: Convex Envelope Method

4.1.1 Qualitative Evaluation

The CEM was already shown to work for ternary and quaternary systems by [62, 82]. Therefore, we show first that the generalized version of the CEM works in the same way for systems that were examined there. As no numerical data for comparison (feed streams and resulting phases) is given in [62, 82], we show ternary diagrams constructed by the generalized CEM for a selection of systems from [62, 82] using the same binary parameters for the UNIQUAC model [93]. It is difficult to display the phase equilibrium diagram for a quaternary system in a way that one can get useful information, let alone compare it to other phase diagrams. Therefore, we omit a graphical evaluation of the quaternary systems shown in [62, 82] and show later on in a numerical way that our approach works for systems with more than three components.

Figure 14 shows six ternary systems reported in [62, 82] with various types of liquid phase equilibria. Note that the systems 1-hexanol – nitro methane – water and water – nitro methane – nonanol also show a three-phase region enclosed by the two-phase regions. When comparing Figure 14 to the ternary plots shown in [62, 82], one can see that the displayed phase equilibria are the same.

4.1.2 Quantitative Evaluation

To evaluate the CEM quantitatively, we will compare its results with experimental data from several sources [94–98]. All of the listed authors provide binary interaction parameters for a g^E -model (either UNIQUAC or NRTL [93]), experimental data on occurring phase splits, and the resulting root-mean-square-error (RMSE) of the model compared to experimental data. Unfortunately, the data points generated by the models

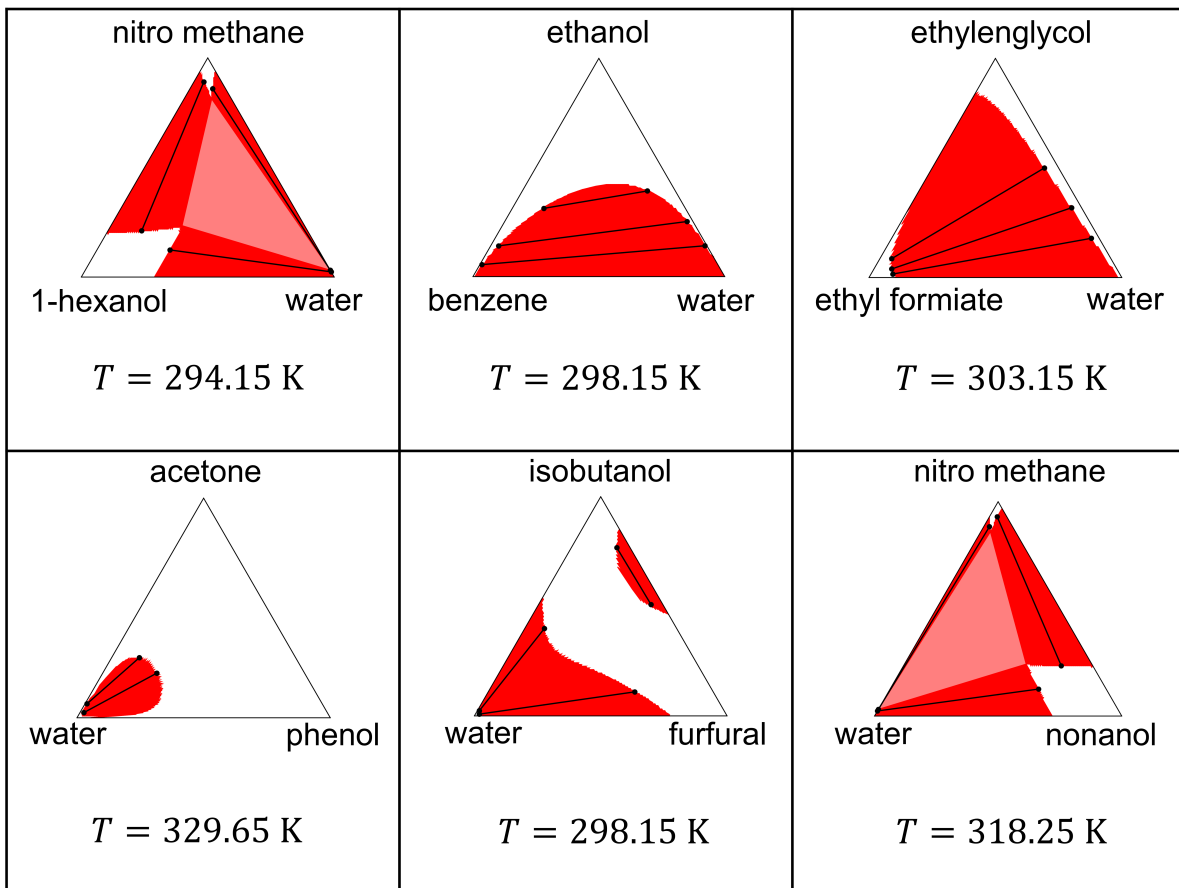


Figure 14: Selection of ternary systems with UNIQUAC parameters from [62, 82] constructed with the generalized CEM at atmospheric pressure. The plots display molar fractions. The transparent red areas in the systems 1-hexanol – nitro methane – water and water – nitro methane – nonanol display three-phase regions, all other red areas display two-phase regions (for every two-phase region, a few example tie lines are plotted in black).

were not reported. But as the RMSE is relatively low (< 0.01 for almost all directly fitted examples in [94–98]), we compare the CEM to the reported experimental data as a workaround. We measure the accuracy of the generalized CEM by mean deviation (MD) to given data:

$$\text{MD} = \frac{\sum_{i=1}^N \sum_{j=1}^M \sum_{k=1}^P |x_{ijk}^{\text{source}} - x_{ijk}^{\text{CEM}}|}{NMP}. \quad (17)$$

N describes the number of components, M the number of examined feed streams and P the number of phases (in [94–98] only splits into two phases were reported). x^{source} refers to the molar fraction reported in the literature.

Detailed results for systems, which were reported in [94–98], containing three, four, five, and six components are provided in Appendix B1.3. For all examined systems, the CEM was able to calculate the occurring phase splits with high accuracy. For ternary systems, our implementation of the CEM constructs phase equilibria in up to 2 seconds with serial execution of Step III) from Section 3.1.2.1. For systems with more components, Step III) from Section 3.1.2.1 was executed parallelized (the respective settings for parallelization are published alongside the code on GitHub). For quaternary systems, computation time per system is below 1 minute, whereas for systems containing five components, up to 3 minutes are required. For systems containing six components, constructing the phase equilibrium takes up to 50 minutes per system.

4.1.3 Analysis of the Impact of the Discretization Parameter δ

To show the impact of the discretization parameter δ on the accuracy of the calculated phase splits, we plot the MD for two ternary example systems from [94] for various choices of δ in Figure 15. For low values of δ , the CEM cannot always determine that there will be a phase split for a given feed stream composition according to the data. These points in Figure 15 are marked with the symbol x. But as expected, if δ increases, the MD decreases, and the CEM can correctly calculate phase splits for all given data points. Also, it can be seen that an accurate calculation of phase splits is already possible for relatively low values of δ as 16. The residual deviation for large δ originates from the deviation of the model and the experiments with which we compare us. Figure 16 shows the heterogeneous simplices (in red) for different choices of δ . As can be seen, when δ is increased, the boundary of the two-phase region gets smoother and approximates the shape defined by the non-discretized model.

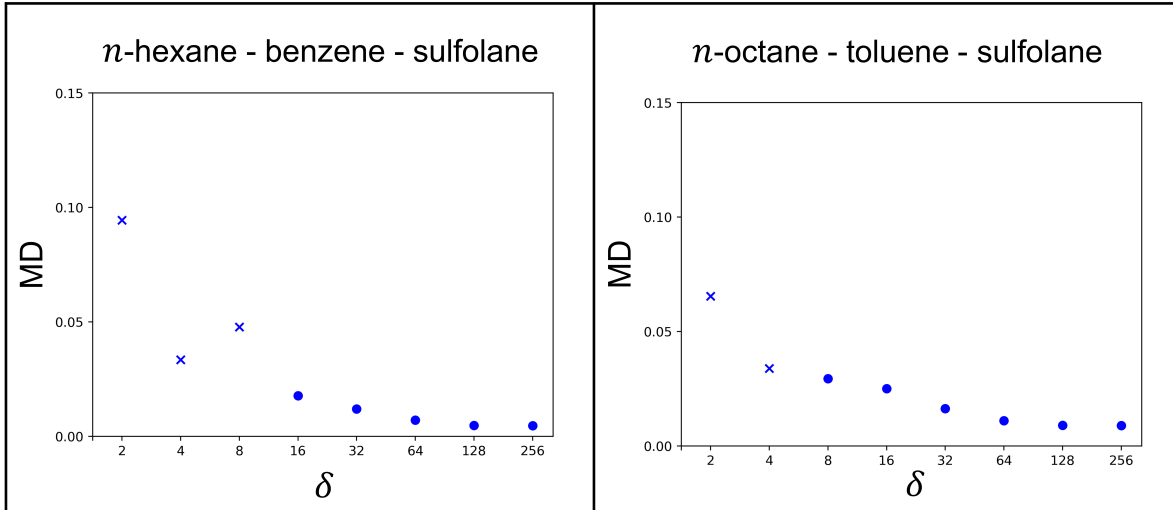


Figure 15: MD values for several choices for δ for the systems *n*-hexane – benzene – sulfolane and *n*-octane – toluene – sulfolane. MD was calculated using phase split data and parameters from [94]. Some points in the graph are marked with the symbol x, which indicates that no phase split was found for some given feed stream compositions for this choice of δ .

4.1.4 Discussion

The generalization of the CEM [62, 82] generates liquid phase equilibria for systems with an arbitrary number of components by discretization of the whole composition space and construction of the convex envelope for the Gibbs energy of mixing graph. In Sections 3.1.2 and 4.1, a theoretical framework for the extension to an arbitrary number of components is given, and the approach was proven to work for systems from the literature with up to six components.

While Theorem 1 gives the possibility to compute phase splits for systems with an arbitrary number of components, computational complexity becomes a limiting factor when systems with more than six components are examined. Particularly, the construction of the convex envelope is a challenging task. For this, we utilize the package `scipy` [65], which bases the construction of the convex envelope on the QuickHull-algorithm [99] provided by the Qhull library. The Qhull library documentation mentions that complexity increases rapidly with the number of input points (defined by the discretization of the composition space) and the dimension of the input (defined by the number of components). For the CEM, we require not only the points that form the convex envelope but also the connections between those points, i.e., the facets. According to the Qhull library documentation, this requires a lot of virtual memory and finally leads to a rapid decrease in performance. An effective implementation of a parallelized convex envelope algorithm to approach higher order systems could solve this problem. Theoret-

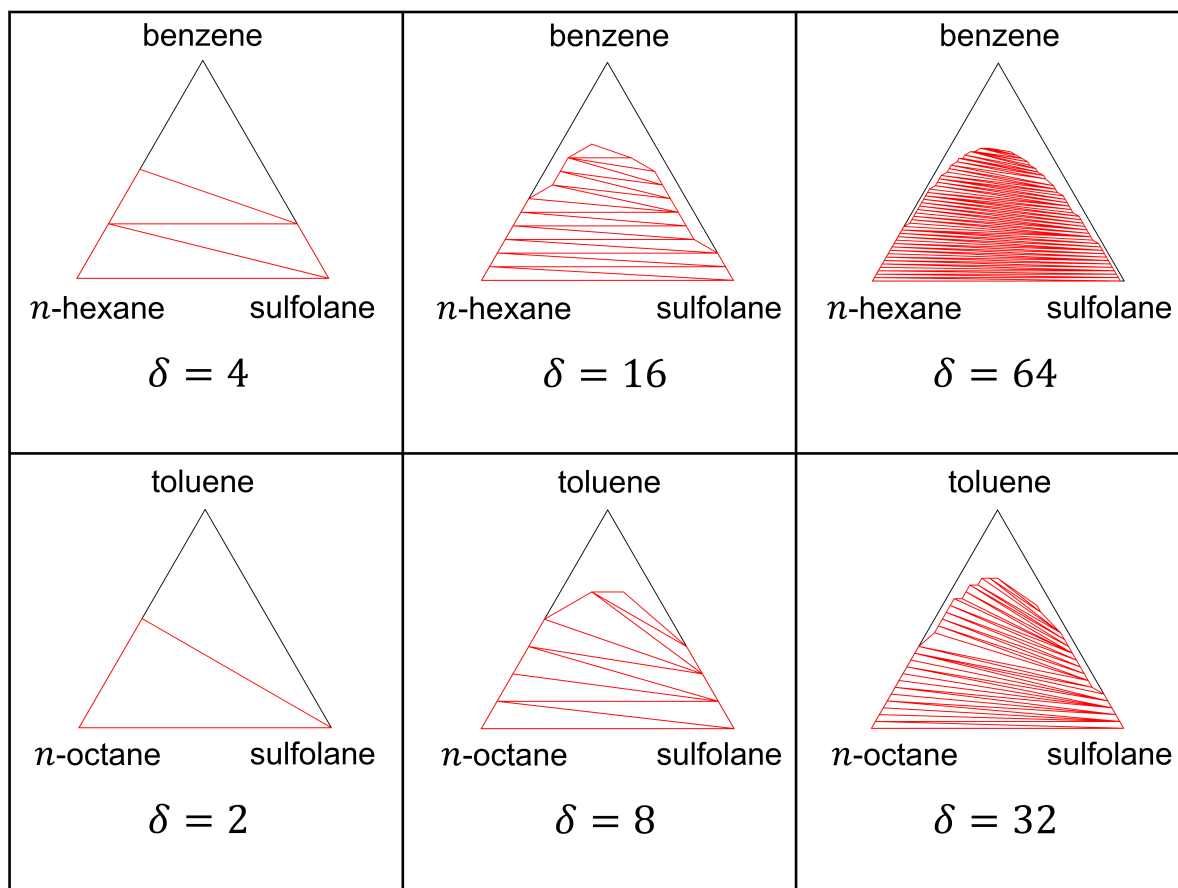


Figure 16: Variation of δ for the construction of ternary diagrams for the systems *n*-hexane – benzene – sulfolane and *n*-octane – toluene – sulfolane from [94] at 298.15 K and atmospheric pressure. The red triangles are the heterogeneous simplices (i.e., simplices that span over a phase split region).

ical research focusing on the construction of convex envelopes in a parallelized fashion is, for example, provided by [100] and [101]. Still, no implementation is available which could be easily integrated into our framework to compare the resulting performances. Additionally, it is unclear if those algorithms can construct the convex envelope with less usage of virtual memory. Other algorithms can solve specific problem classes faster than the Qhull framework [102], e.g., two-dimensional input and particular topologies. But to our knowledge, no implementation exists that exceeds the Qhull framework in terms of generality and effectiveness, as it can handle an arbitrary number of dimensions. Also, the classification of the heterogeneous simplices (Step III) in Section 3.1.2.1) is a time-consuming step, but contrary to the construction of the convex envelope, it can be easily parallelized. The discretization of the composition space and the computation of Δg^{mix} -graph do not need much computation time compared to the previously mentioned steps and are negligible.

Throughout this work, the CEM was only employed to construct liquid phase equilibria. But as already mentioned in [62, 82], the CEM could also be applied to calculate arbitrary phase equilibria (e.g., vapor-liquid and solid-liquid). Integration of this feature into the present framework is out of the scope of this work but an interesting option for future research.

The CEM constructs phase equilibria for the whole composition space for fixed temperature and pressure. Therefore, it may not be useful for optimizing temperature in a decantation process. However, after the phase equilibrium has been constructed, it is possible to quickly evaluate different feed stream compositions at fixed conditions. As the construction process of the phase equilibria is robust, the CEM is particularly useful for modeling decantation processes for RL-based AFS (see challenges on the environment side in Section 2.2).

4.2 SynGameZero: Proof of Concept

4.2.1 General Remarks

A proof of concept for the SynGameZero approach is provided along the quaternary system A – B – C – D and the unit operations described in Appendix CI.1. Two case studies are presented, which differ in the cost function and the number of feed streams. In both case studies, the compositions of the feed streams are varied randomly (during training and evaluation) to enable the agent to succeed over a broad range of problems. After training is completed, the agent designs flowsheets for arbitrary feed

streams (selected inside the training intervals) by playing one game against itself and returning the flowsheet of the winning player.

To evaluate the flowsheets proposed by the trained agent, we compare them to predefined benchmark flowsheets. An evaluation set of 1000 feed stream situations is sampled randomly. The trained agent synthesizes flowsheets for all problems in the evaluation set. These flowsheets are compared to the benchmark flowsheets using the NPV. The success rate of the agent is defined using the following metrics:

$$R_1 = \frac{N_{\text{success}}}{1000}, \quad (18)$$

$$R_2 = \frac{1}{1000} \sum_{j=1}^{1000} \frac{\text{NPV}_{\text{agent},j} - \text{NPV}_{\text{benchmark},j}}{|\text{NPV}_{\text{benchmark},j}|}, \quad (19)$$

$$R_3 = \frac{1}{1000 \cdot R_1} \sum_{j \in \Lambda} \frac{\text{NPV}_{\text{agent},j} - \text{NPV}_{\text{benchmark},j}}{|\text{NPV}_{\text{benchmark},j}|}, \quad (20)$$

$$R_4 = \frac{1}{1000 \cdot (1 - R_1)} \sum_{j \in \Gamma} \frac{\text{NPV}_{\text{agent},j} - \text{NPV}_{\text{benchmark},j}}{|\text{NPV}_{\text{benchmark},j}|}. \quad (21)$$

Therein, N_{success} is the number of times the agent proposed a flowsheet that is at least as good as the best benchmark flowsheet. Thus, $R_1 \in [0, 1]$ is the overall success rate. $R_2 \in (-\infty, \infty)$ gives the average deviation for the NPVs of the agent's flowsheet and the best benchmark flowsheet. $R_3 \in [0, \infty)$ measures this average only in cases Λ when the agent was at least as good as the benchmark. $R_4 \in (-\infty, 0)$ measures the average deviation in all other (i.e., unsuccessful) cases Γ .

4.2.2 Case Study 1

In Case Study 1, there is one single feed stream, and all components have the same prices (details are provided in the Appendix CI.1). Therefore, in this case, a reactor does not increase the value of a stream (as the sum of stoichiometric coefficients is equal to 0), and a distillation sequence provides the most profitable process. During training, random quaternary and ternary feeds out of the following feed stream situations were selected: (x_A, x_B, x_C, x_D) , $(0, x_B, x_C, x_D)$, $(x_A, 0, x_C, x_D)$, $(x_A, x_B, 0, x_D)$, $(x_A, x_B, x_C, 0)$. The mole fractions of the non-zero components were chosen randomly. The total molar flow rate of the feed was always set to 1 kmol/h. For every feed stream situation listed above, we defined as a benchmark flowsheet a distillation sequence, which separates the given feed into pure components (3 distillation columns for the quaternary feed and 2 distillation columns for ternary feeds).

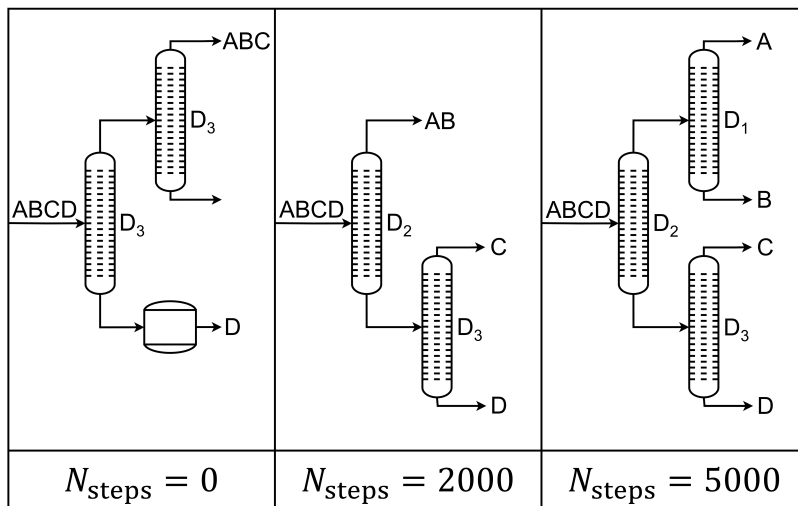


Figure 17: Illustrative example for the evolution of the agent during the training process in Case Study 1. Flowsheets proposed by the agent to separate an equimolar quaternary mixture are shown. Streams that leave the process without description (e.g., ABC) are empty.

Figure 17 shows examples of the flowsheets proposed by the agent at different stages during training. The training procedure was repeated five times. Each time, the trained agent surpassed an overall success rate of $R_1 = 0.98$. This proves that the agent displays an almost optimal behavior, as in this case study, the benchmark flowsheets already provide an optimal solution.

4.2.3 Case Study 2

In Case Study 2, the NPV parameters are changed so that A and B have a negative price. C and D are high-value products. Thus, it is worth to react A and B to C and D (details are provided in the Appendix CI.1). Four different feed stream situations are considered. Situation 1 considers two feed streams of the types: $(x_A, x_B, 0, 0)$, $(0, 0, x_C, x_D)$. Situation 2 considers two feed stream situations: $(x_A, 0, x_C, 0)$, $(0, x_B, 0, x_D)$. Situation 3 considers one feed stream of the type: $(x_A, 0, x_C, x_D)$. Situation 4 considers one feed stream of the type: $(0, x_B, x_C, x_D)$. For every game during training, one of the four situations is selected randomly. The given molar flowrates \dot{n}_i for the non-zero components are sampled randomly out of the interval $[0.2, 1.2]$ kmol/h. The agent's performance is evaluated individually for all four feed situations. The flowsheets shown in Figure 18 are defined as the respective benchmark flowsheets.

The training process was repeated five times. Table 1 shows the average of the performance metrics. Most of the flowsheets proposed by the agent were quite similar to the

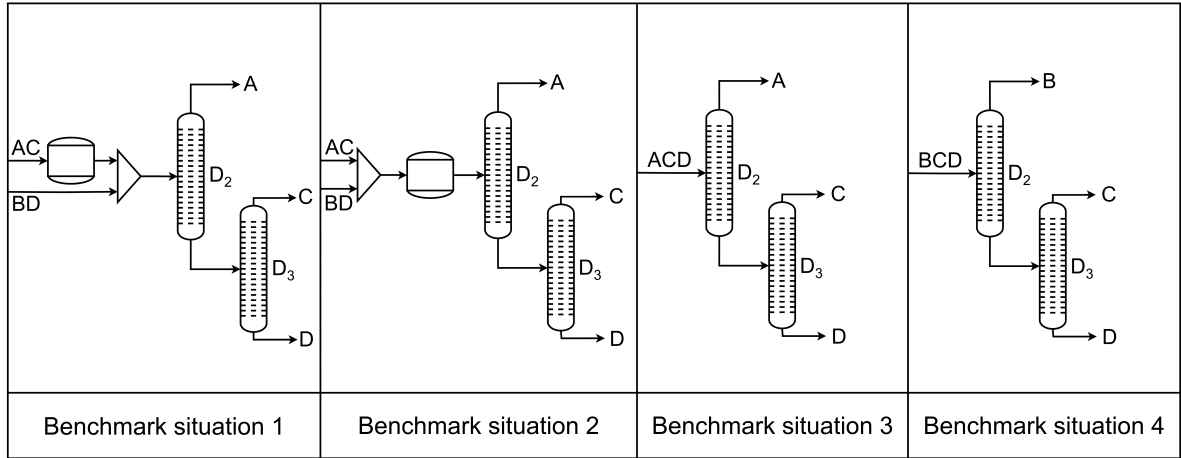


Figure 18: Benchmark flowsheets for Case Study 2.

Table 1: Average performance metrics for Case Study 2.

	R_1	R_2	R_3	R_4
Situation 1	0.84	0.08	0.12	-0.12
Situation 2	0.99	0.13	0.13	-0.01
Situation 3	1.00	0.00	0.00	-
Situation 4	1.00	0.00	0.00	-

benchmarks shown in Figure 18. In situation 1, the agent meets or beats the benchmark in 84% of the cases. In the successful cases, the possible gain over the benchmark is 12%. Although the agent is worse than the benchmark in 16% of the cases, its average NPV is still slightly better than the benchmark ($R_2 > 0$). In situation 2, the agent almost always reaches the NPV of the benchmark, and as $R_4 = -0.01$, it seems that it proposes, in the few cases where it is worse, flowsheets with very similar NPVs as the benchmarks. In situation 3 and situation 4, there is either A or B missing in the process. Thus, there is no use for a reactor. The optimal flowsheets are distillation sequences for separating the ternary mixtures. The trained agent solves the problems in these situations without any difficulty.

The evolution of the agent during training for feed stream situation 1 (feed streams: $(x_A, x_B, 0, 0)$, $(0, 0, x_C, x_D)$) is shown in Figure 19. The right column shows examples of the flowsheets the agent proposed at this stage. The left column shows 3D plots of three highlighted entries of the vector \mathbf{p} (which depends on the ANN's output $\boldsymbol{\pi}$) at the beginning of the flowsheet synthesis, i.e., the ANN's suggestions for the very first action. The data is plotted over possible feed stream flowrates. Since the space of feed

stream flowrates is 4-dimensional in this situation, we restrict ourselves for the sake of illustration to a 2-dimensional subspace in which $\dot{n}_A = \dot{n}_B$ holds in the A – B feed stream and $\dot{n}_C = \dot{n}_D$ holds in the C – D feed stream. Action 1 refers to mixing both feed streams. Action 2 refers to placing a distillation column of type D₃ at the C – D feed stream. Action 3 refers to placing a reactor R at the A – B feed stream. At the beginning of training, the actions of the agent are random. The suggested probabilities of the three highlighted actions are small as they are not significantly larger than the probabilities of any other feasible action (there are ten feasible actions for the move). The agent selects none of the shown actions. After 2000 training steps, the ANN favors as first action, placing a distillation column that splits C and D at the second feed stream (Action 2). After 6000 training steps, the agent has learned to complete the reaction part of the flowsheet before distillation is done. In the shown example, the agent prioritizes mixing (Action 1) before reaction. At the end of training, the agent has learned that bypassing the reactor with the products C and D yields a higher conversion. C and D are separated only later together with the reactor outlet.

4.2.4 Discussion

The results shown in Section 4.2 give a proof of concept for the SynGameZero approach, which enables training an agent to solve basic flowsheet synthesis problems without using prior knowledge or heuristics via RL. The agent consists of an ANN and a tree search in which the planning process is modeled within a two-player game. This setting allows the usage of a modified version of the training algorithm of the SynGameZero framework [54, 55]. The trained agent succeeds at the given problems by combining discrete actions to synthesize a flowsheet using systematic generation.

To assess the efficiency of the approach, the total number of possible flowsheets for a fixed feed composition has to be determined as follows (with a matrix size of $N_{\text{matrix}} = 10$). To the first stream, one could connect three types of distillation columns, one reactor, or a mixer to one of the streams 2 to 9 (not to stream 10, as this would result in a new stream 11, which exceeds the size of the flowsheet matrix). Additionally, it is possible to let the stream leave the process, which results in 13 possibilities; for the second stream, one arrives at 12 possibilities (as mixing to stream 1 was already counted in beforehand). This calculation can be continued until stream 8 (6 possibilities). At stream 9, there is only the possibility of placing a reactor or letting that stream leave the process (distillation columns are not possible as the state matrix can only contain one more stream). At stream 10, no unit can be placed. Therefore, this stream leaves the process. This results in roughly 50 million possible flowsheets to choose from for

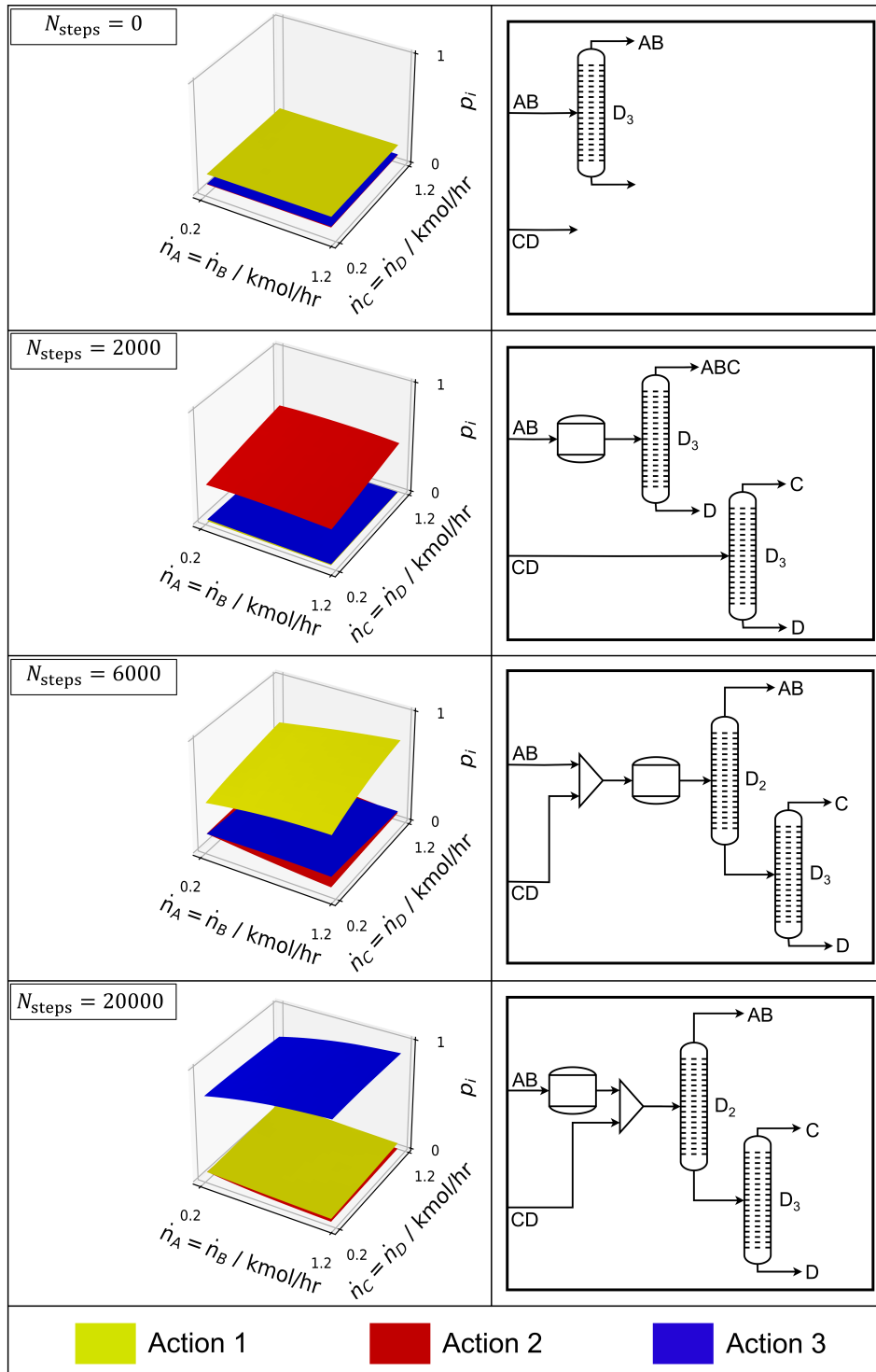


Figure 19: Example for the evolution of the agent during the training process for situation 1, Case Study 2. The 3D plots show the value of three highlighted actions of the ANN's policy output over a subset of the composition space of the feed streams ($\dot{n}_A = \dot{n}_B$, $\dot{n}_C = \dot{n}_D$) for the first action of the agent. Action 1 is mixing both feed streams. Action 2 refers to placing a distillation column of type D_3 at the C – D feed stream. Action 3 refers to placing a reactor R at the A – B feed stream.

one player and does not even count in that the feed compositions are sampled from a continuous range. Therefore, the state space for the agent is infinitely big. The tree size cannot exceed $2 \cdot K \cdot N_{\text{matrix}}$, equal to 800 for Case Study 2. In the case studies, the trees contained only an average of 200-300 flowsheets, as many were visited multiple times during the tree search. The number of total distinct flowsheets visited during the tree search is estimated to be significantly smaller than 4 million. Additionally, the policy of the agent (parameterized by the ANN) shows explicit learning behavior, as shown in Figure 19. Thus, it can be concluded that the proposed flowsheets are not found by luck or through massive enumeration in the tree search.

The unit operations considered in the presented case studies are few and basic in their modeling. This is enough for a proof of concept for the SynGameZero approach. However, the examples have several limitations. First, a larger number of process units and chemical compounds would require a larger flowsheet matrix. This would blow up both the agents input and the action space. The representation of state (see Appendix CI.2) and actions (see Section 3.2.2) is most likely not scalable to very large problems. For example, convolutions for large inputs or a hierarchical decomposition of the action space could be integrated into the framework. Second, only processes without recycles were considered. Those pose problems for the environment, as a convergent simulation can no longer be guaranteed when the agent is allowed to place a recycle at any point during the game (e.g., recycling the only stream that leaves the process). Therefore, the action space has to be redesigned so that the agent cannot propose a divergent flowsheet but is still flexible enough to develop genuinely new designs.

Finally, it is necessary to allow the agent to operate in a hybrid action space (i.e., an action space containing discrete and continuous actions) to tackle more sophisticated flowsheet problems by specifying parameters of the unit operations, for example, pressure and temperature. The presented RL algorithm operates in a discrete action space, so further developments are required. Much RL research focuses on parameterized action spaces, as this field has a broad range of applications [58, 59]. However, it is not straightforward to integrate these methods into the SynGameZero framework.

The main contribution of the SynGameZero approach is modeling flowsheet synthesis as a two-player game, which is beneficial for two primary reasons. On the one hand, it eliminates the absolute value of the environment’s native reward function. Additionally, the SynGameZero framework provides a reward only at the end of the game. As shown in Section 4.2, it is possible to train the agent successfully with this relatively sparse reward signal (addressing challenge II) on the agent side in Section 2.2). On the other hand, the framework has strong exploration abilities (addressing challenge I) on the

agent side in Section 2.2). Especially for player 2, who starts second, there is a strong motivation to explore novel actions instead of losing the game by just copying the actions of player 1. This is because player 2 has the systematic disadvantage of losing the game if it is tied. During the examples shown above, player 2 lost the majority of all games, as expected, due to this disadvantage. However, it could be often observed that player 2 wins more games during the training phases when new breakthrough actions are learned. This indicates that these actions were found first by player 2. Player 1 adopts these actions quickly by observing player 2 and benefits therefore as well. These beneficial features also make the SynGameZero method attractive for other planning processes beyond chemical engineering.

4.3 SynGameZero: Integration of Hierarchical Reinforcement Learning

4.3.1 General Remarks

The hierarchical framework is proven to work along the quaternary system Et – IB – nBut – ETBE and the process units described in Appendix DI.1. During training, before every game, two feed streams $F_1 = (x_{Et}, 0, 0, 0)$ and $F_2 = (0, x_{IB}, x_{nBut}, 0)$ are sampled randomly and provided to the agent as initial situation. The total molar flowrates are randomly sampled from the interval $[10, 110]$ kmol/h. After training is completed, the agent designs flowsheets for arbitrary feed streams (selected inside the training intervals) by playing one game against itself and returning the flowsheet of the winning player.

Without being aware of the optimal solution for every conceivable combination of feed streams, we have evaluated the agent’s performance by comparing it to benchmark flowsheets based on ETBE processes from the literature [62, 64]. The benchmark flowsheets are shown in Figure 20. The trained agent is evaluated by designing flowsheets for 1000 randomly sampled feed streams of identical format as during training. For every given set of feed streams, all benchmark flowsheets are simulated, and the best one will be compared to the flowsheet proposed by the agent. The performance is evaluated by the same performance metrics as introduced in Section 4.2.

In the following, we present results for the hierarchical SynGameZero framework and its variation, described in Section 3.2.3.3.

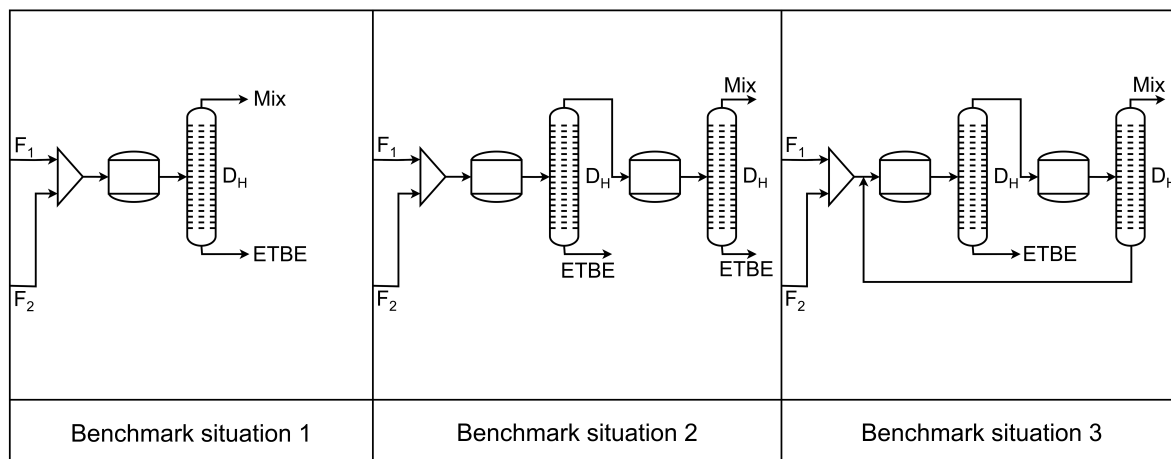


Figure 20: Benchmark flowsheets designed by the authors for the quaternary system Et – IB – nBut – ETBE.

4.3.2 Original Hierarchical Framework

In the original hierarchical framework, there is one option for placing a mixer and one option for placing a recycle in the agent’s action space. As described above, the agent is trained to construct flowsheets suitable for synthesizing ETBE. The procedures for training and evaluation were repeated five times. Table 2 lists the resulting performance metrics. In about 97% of all problems, the agent is at least as good as the benchmarks. On average, it even surpasses the benchmarks by 23% in the NPV. This is because the benchmarks only focus on the generation of pure ETBE, while the cost function also allows profitable flowsheets without even synthesizing ETBE. The margin to the benchmarks almost vanishes for feed streams where the agent proposes a worse flowsheet.

Table 2: Average performance metrics for the original hierarchical framework (the values are rounded).

R_1	R_2	R_3	R_4
0.97	0.23	0.23	0.00

Flowsheets that outperform the benchmarks are shown in Figure 21 for different feed streams. Panel a) shows a flowsheet that outperforms the NPV of the best benchmark (benchmark situation 2) by 19%. This is because excess nBut and IB are separated as pure streams and sold as products instead of only ETBE. Panel b) shows a problem with less IB. The NPV of the shown flowsheet surpasses the best benchmark flowsheet (benchmark situation 2) by 12%. Contrary to the flowsheet in panel a), IB is not present

in excess and has to be recycled to gain a higher yield of ETBE. Panels c) and d) show further situations in which there is so little Et or IB that it is not profitable anymore to produce high-purity ETBE. In panel d), the agent decides to separate the feed streams and sell the educts. In the situation in panel c), the agent puts a reactor and separates IB afterward but omits to separate pure ETBE with a column of type D_H , as the costs of the distillation column would exceed the benefit of the resulting streams. The best benchmark (benchmark situation 1) in these cases is outperformed significantly by 52% (panel c)) and 36% (panel d)).

In Figure 22, it is shown how the trained agent approaches different feed stream combinations over a part of the composition space ($\dot{n}_{IB} = \dot{n}_{nBut}$ is assumed). Except for the equimolar feeds on the diagonal, the agent always uses one of the flowsheets shown in Figure 21. A dear structure emerges, which has been learned by the agent. It proposes different flowsheets for the equimolar feeds on the diagonal, similar to the flowsheets a) and b) in Figure 21. However, they often contain an unnecessary unit (e.g., a distillation column, which already takes a pure stream as input) or an empty recycle. This is due to cheap unit operations and the fact that the agent does not often encounter these equimolar feed stream situations during training. For all shown feed stream combinations, the agent can surpass the benchmarks.

4.3.3 Variation of the Hierarchical Framework

In the variation of the hierarchical framework, the agent decides to admix an open stream to another stream in the flowsheet, which can either be open or closed. The environment determines automatically if the chosen configuration leads to a mixer or a recycle. The agent is trained to construct flowsheets suitable for synthesizing ETBE. The procedures for training and evaluation were repeated five times. The agent outperforms the benchmark flowsheets in almost every case ($R_1 = 0.9996$, it only proposes a worse process on two occasions). It surpasses the benchmarks on average by 24%, which is a slight improvement compared to the original hierarchical framework.

To uncover the importance of the two-player game setup, the agent's performance is analyzed at different stages during training for a fixed set of feed stream situations. Note that this analysis could also be conducted for the original hierarchical framework, yielding similar results.

The molar flowrate of Et is varied between 15 and 95 kmol/h. The molar flowrates of IB and nBut are set equal. They are also varied between 15 and 95 kmol/h. Figure 23 illustrates the evolution of the agent during training by showing its behavior after

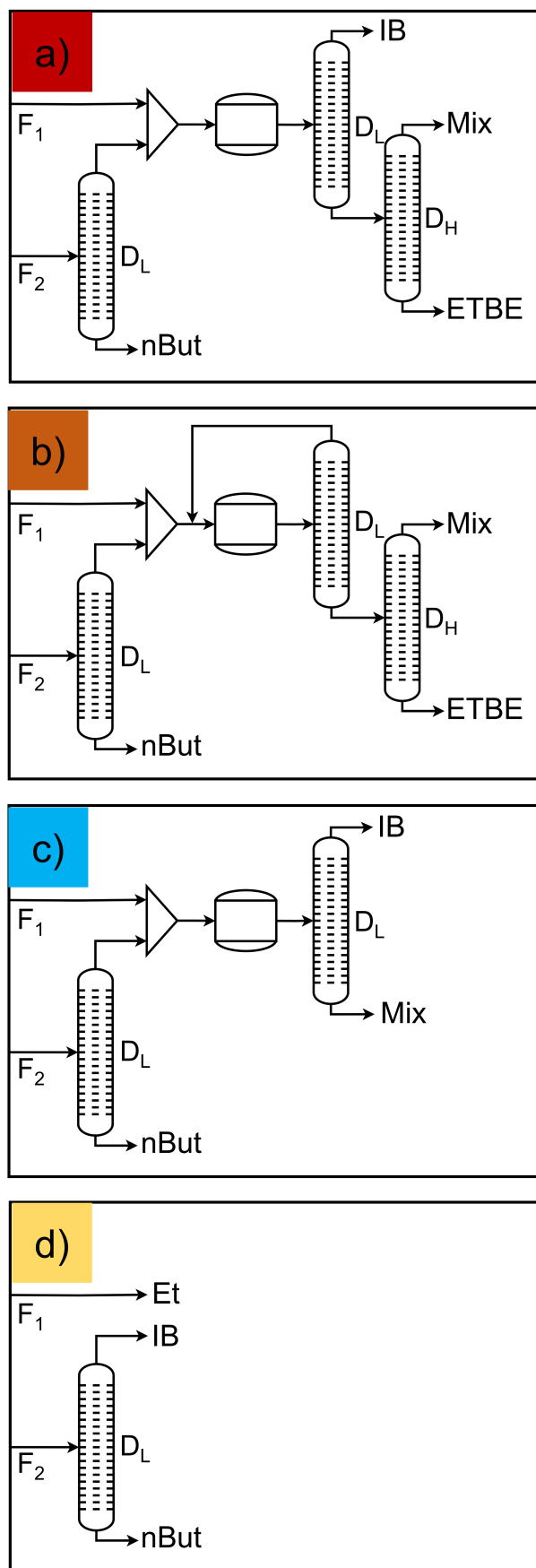


Figure 21: Examples for flowsheets proposed by the trained agent.

$\dot{n}_{IB} = \dot{n}_{nBut}$ / kmol/hr	95	c)	c)	a)	a)	other
	75	c)	a)	a)	other	b)
	55	a)	a)	other	b)	b)
	35	a)	other	b)	d)	d)
	15	other	d)	d)	d)	d)
		15	35	55	75	95
		\dot{n}_{Et} / kmol/hr				

Figure 22: Examples for the agent's behavior for several feedstream combinations. a), b), c), and d) refer to the flowsheets shown in Figure 21.

different numbers of training steps. The various feed stream situations are depicted as cells in the matrix. The winning player of every combination is indicated with a color code. The winning flowsheet (marked red) is displayed for one feed stream combination. Without any training ($N_{\text{steps}} = 0$), the agent consists of a randomly initialized ANN and a tree search. The agent's behavior is the same for all shown combinations of feed streams. In the role of player 1 it terminates the synthesis right away. In the role of player 2 it sets up the shown flowsheet and wins the game. After 100 training steps, player 1 has copied this tactic for all shown feeds and wins all games. After 1000 training steps, the game is more balanced. Both players can win in some situations. From the shown flowsheet generated by player 2, it is visible that the agent has learned to use a reactor to synthesize ETBE, which is clear progress. After 3000 training steps, player 1 is more dominant, and the flowsheets become more sophisticated. This balance change between players 1 and 2 winning in the game is observed many times during training. Typically, player 1 is copying (if needed) and using the so-far best-known tactic. Player 2 must avoid a tie and, therefore, explore alternative tactics. It is consequently mainly player 2 who uncovers novel improved tactics. Player 2 will afterward win more games than player 1 for a short period during training. Eventually, player 1 acquires the novel tactic and wins again. The bottom row in Figure 23 shows situations at the late stages of training. After 5000 steps, the complexity of the flowsheets further increases while the game is still quite balanced. For equimolar feed rates of IB and Et (i.e., on the diagonal of the matrix), the agent has learned to generate flowsheets with complete conversion of IB and Et. The chemical equilibrium is overcome by using a recycle. However, the design is still not optimal. After 10000 steps, the flowsheets are slightly more improved, and the training is completed. Player 1 wins all games. Even with further training, player 2 cannot find a better tactic. Such a constellation signifies that a local or maybe even global optimum for the performance has been reached.

4.3.4 Discussion

Sections 3.2.3 and 4.3 show how to integrate a hierarchically structured action space into the SynGameZero approach. At first, the agent chooses a location (i.e., an open stream); second, the corresponding unit operation, and third, a unit specification, if needed. The approach is demonstrated by training the agent to set up a process for ETBE synthesis. Compared to the original SynGameZero approach, this example displays a significantly increased level of complexity, as the agent has to deal with azeotropic behavior, equilibrium reactions, and recycle streams. Two variations of the framework, which differ in how the agent places mixers and recycles, are presented. The agent

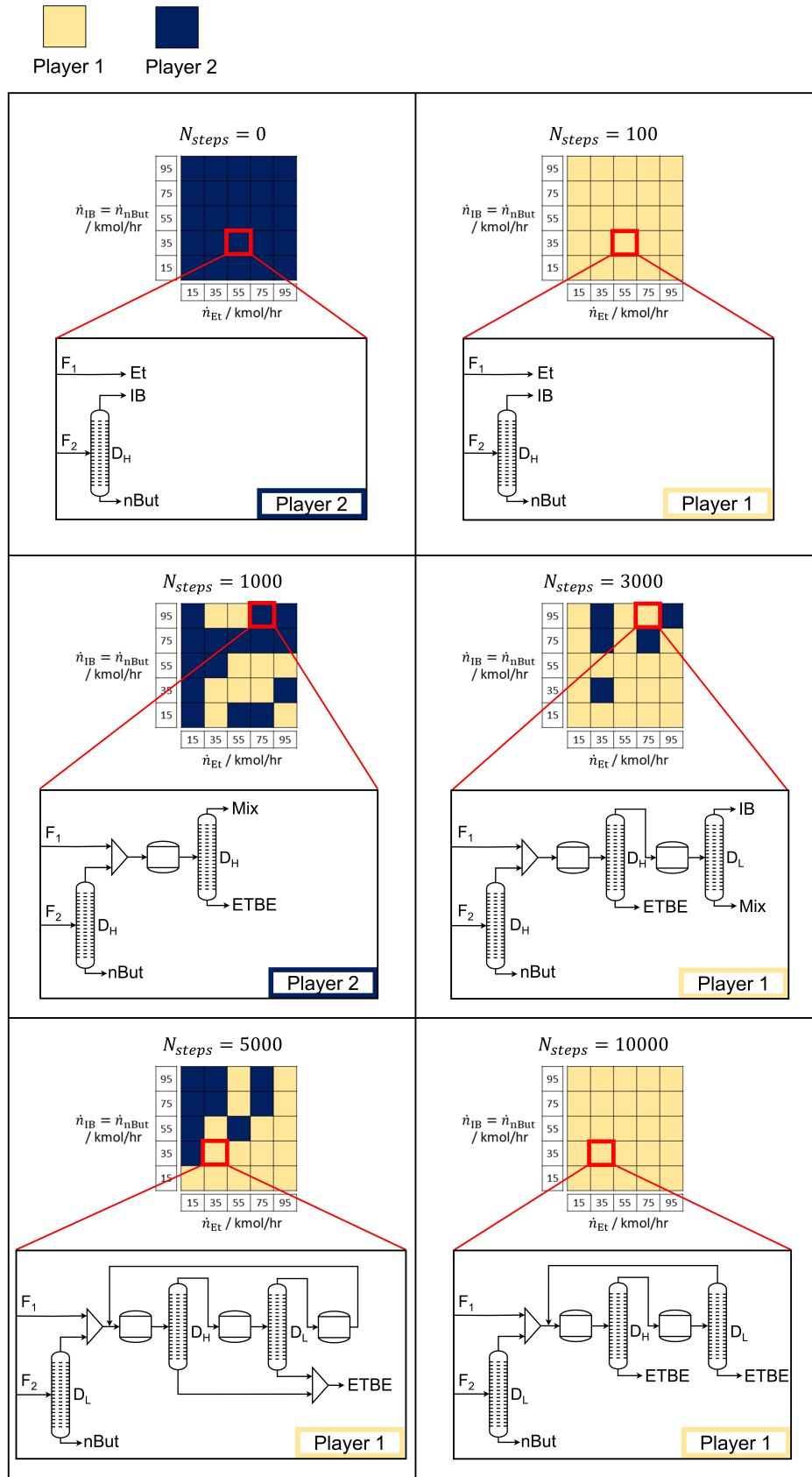


Figure 23: Illustration of the agent's evolution at different stages during training. The matrix field represents different feed stream combinations. The color code marks the winning player. The red box shows the winning flowsheet for the respective feed streams.

succeeds at the given problem in both cases, displaying similar performance metrics.

To assess the efficiency of the approach, one can compare the maximum number of flowsheets per tree to the number of possible flowsheets for a fixed pair of randomly sampled feed streams. Similarly, as described in Section 4.2.4, it can be concluded that the number of possible flowsheets exceeds the size of the tree by many orders of magnitude. Note that the possibility to place recycles to any destination increases the action space further compared to the original SynGameZero approach.

While the main contribution of Sections 3.2.3 and 4.3 is integrating the hierarchical framework, there were further improvements compared to the original SynGameZero approach. A CNN was introduced to reduce the number of parameters, which captures the information in the state row by row. Additionally, the way of representing a flowsheet by a matrix was slightly modified to be more suitable for problems of increased size. The general idea of the two-player game combined with a tree search is still employed on top of these features, as it provides excellent exploration properties for the agent.

Looking at the challenges described in Section 2.2, the hierarchical SynGameZero approach addresses almost all except to work in a hybrid action space. To solve this issue, Sections 3.2.4 and 4.4 describe a framework that adapts several parts of the hierarchical SynGameZero approach to finally provide a general RL approach for AFS, which solves all aforementioned challenges.

4.4 Single-Player Reinforcement Learning Framework for Automated Flowsheet Synthesis

4.4.1 General Remarks

The agent's task is to separate a binary feed stream of the chemical systems listed in Table E1 (see Appendix E1.1) into its pure components. From a chemical engineering viewpoint, conceptually, quite different approaches are required for solving the separation tasks. For example, mixtures in system 1 consist of Ac and Ch and can be treated by entrainer distillation [103]. This means that solvents Be or To can be added as an entrainer to employ the curvature of the resulting ternary distillation boundaries to separate the feed stream. Contrary, mixtures in system 2 consist of Et and Wa and can be separated using (heterogeneous) azeotropic distillation [104]. This means that adding a solvent, for example, Be, results in a ternary mixture displaying liquid phase splits, which can be employed in a separation process. With minor modifications, these

techniques can also be used for system 3 and system 4, but there are other options as well. For example, feed streams in system 3 can be separated without using a solvent at all.

We train the agent separately with two types of cost functions, which are used to evaluate the resulting flowsheets. One type is NPV, based on literature about the economic analysis of chemical processes. The other type is a generic cost function (GCF) that aims at providing a general cost function for separation processes. It is handy for conceptual flowsheet synthesis, where the primary goal is not to obtain the most profitable solution but a feasible one. Note that it is possible to modify or replace the cost function arbitrarily, e.g., by introducing additional objectives such as sustainability. Both cost functions are explained in detail in Appendix E1.1.

As the raw values for NPV and GCF may be hard to interpret, we define a performance ratio $\mathcal{R} \in [0, 1]$ as a metric to evaluate a flowsheet. \mathcal{R} describes how much of the input (feed and added solvent) the flowsheet separates into pure components. As the main goal is the perfect separation into pure components, a value of \mathcal{R} close to 1 is desirable. For the definition of \mathcal{R} , consider a flowsheet with feed stream $(\dot{n}_1^F, \dot{n}_2^F, 0)$. Let \dot{n}_3^S be the accumulated amount of a solvent, which was added to the process (note that \dot{n}_3^S can also be equal to 0). Let $L = \{l_1, \dots, l_K\}$ be the set of leaving streams that meet the specification for pure streams. This means that for all $l_i = (\dot{n}_1^i, \dot{n}_2^i, \dot{n}_3^i)$ with $i = 1, \dots, K$ it holds that there exists exactly one $j \in \{1, 2, 3\}$ so that $x_j^{(m)} > 0.99$ (for NPV) or $x_j^{(n)} > 0.99$ (for GCF). We define \mathcal{R} as:

$$\mathcal{R} = \frac{1}{\dot{n}_1^F + \dot{n}_2^F + \dot{n}_3^S} \cdot \left(\sum_{i=1}^K \dot{n}_1^i + \dot{n}_2^i + \dot{n}_3^i \right). \quad (22)$$

\mathcal{R} measures how much of the input of a process is separated into pure streams ('pure stream' is defined by the respective specification of NPV or GCF).

During training, the agent encounters 50000 randomly sampled feed compositions (details on the sampling process are provided within Appendix E1.3). Finally, the agent is evaluated on a test set containing 49 feed stream situations for every chemical system from Table E1. Here, in the i -th situation of a binary system, the molar fraction of the first component is set to $x_1 = 0.02 \cdot i$. This way, it is ensured that the agent is evaluated on the whole range of the binary composition space.

We further assess whether the agent's performance relies strongly on the computationally expensive MCTS when designing a flowsheet or whether the neural network alone can capture most of the flowsheet dynamics independently. To achieve this, we discard the value network and let the agent use the policy network alone. If the network can truly

grasp the underlying chemical dynamics, then a flowsheet stemming from an action sequence with high total probability should yield a high outcome. To obtain a set of high probability sequences from the model, we unroll the policy with beam search (for a detailed description, see, for example, [91]), which is the de-facto standard sequence decoding method in natural language processing. Beam search is a pruned breadth-first search of limited width k , where at each timestep, we expand the (maximum of) k actions, leading to sequences with the highest total probability.

4.4.2 Overall Performance

Table 3 reports the agent’s performance ratio on the test set covering the full range of molar fractions. Row 1 and 2 show results for the agent using a simulation budget of $K = 200$ during MCTS (same value as during training). On average, the agent achieves a performance ratio of over 95% for all considered chemical systems. In over 60% of all test instances, the agent proposes a flowsheet that separates the feed and added solvent completely into pure streams. While the agent performs almost perfectly in some cases, it becomes clear that its performance differs from system to system (e.g., when comparing system 3 and system 4). A reason for this is the varying difficulty of the separation task. Feeds from system 3 can often be separated with fewer units or without the usage of recycles. Contrary, the location of the binary azeotropes in the other systems usually requires more sophisticated flowsheet designs. A reason for the performance differences regarding the reward choice is that the specification of a pure stream differs from NPV (mass fraction greater than 0.99) to GCF (molar fraction greater than 0.99). Still, the agent proposes flowsheets that separate large parts of the feed and added solvent into pure streams for all considered cases as \mathcal{R} is always greater than 90%.

Row 3 and 4 in Table 3 show the agent’s results when unrolling the policy network using a moderate beam width of $k = 512$. As can be seen, the agent can now master all situations with an almost perfect performance ratio. Additionally, the number of cases with complete separation increases for all cases.

These results show the suitability of the MLP-Mixer architecture for flowsheet representation. Furthermore, beam search allows generating high-quality candidate flowsheets fast (in contrast to slower MCTS), which has practical advantages, for example, when an agent proposes the conceptual design of a flowsheet that serves as an initialization for process optimization.

Table 3: Performance of the agent on the test set for both cost functions NPV and GCF. The ratio \mathcal{R} indicates how much of the input (feed and added solvent) the agent’s flowsheet separates into pure components. Additionally, we report how often the agent proposes a flowsheet that separates the feed and added solvent completely into pure streams (Compl. sep.). Row 1 and 2 show the results for the agent using MCTS with a simulation budget of $K = 200$. Row 3 and 4 show the results for unrolling the policy with beam search (beam width $k = 512$).

		All sys.	Ac, Ch	Et, Wa	Bu, Wa	Py, Wa
NPV (MCTS)	\mathcal{R}	95.9%	97.5%	97.0%	97.5%	91.6%
	Compl. sep.	60.5%	84.0%	50.0%	84.0%	24.0%
GCF (MCTS)	\mathcal{R}	97.4%	98.5%	94.5%	98.8%	97.8%
	Compl. sep.	65.5%	70.0%	30.0%	98.0%	64.0%
NPV (beam search)	\mathcal{R}	98.9%	99.5%	98.1%	99.6%	98.5%
	Compl. sep.	77.0%	94.0%	68.0%	88.0%	58.0%
GCF (beam search)	\mathcal{R}	99.0%	99.3%	97.4%	100.0%	99.4%
	Compl. sep.	78.5%	86.0%	60%	98.0%	70.0%

4.4.3 Comparison to Flowsheets from the Literature

For every system from Table E1, we evaluate the agent on a feed stream provided in the literature [103–106]. In Figure 24, we show the flowsheets constructed by the agent in those situations (trained with NPV). Similarly to the processes from the literature, the agent can separate the feed stream and the used solvent entirely in all four cases. In panel a), the agent adds Be as an entrainer to the Ac – Ch feed to employ the resulting curvature of the ternary distillation boundaries for the separation of the streams. In this ternary system, there is no liquid phase split; therefore, the usage of a decanter does not have an effect on the separation (contrary to the other systems). In panels b) and d), the agent employs (heterogeneous) azeotropic distillation for the separation task. In both cases, it adds a solvent that forms binary azeotropes with both feed components and is immiscible to Wa. This allows the separation by using a decanter combined with a distillation sequence. In panel c), the agent similarly separates Bu and Wa without using a solvent. The reason for this is that the binary system Bu – Wa already displays a liquid phase split, which allows the immediate usage of a decanter in this case. In all examples shown in Figure 24, the agent uses recycles to enable the separations and reduce waste streams. As shown in the following section, it even chooses the continuous specifications of the units so that they only make sense in combination

with those recycles. When the agent encounters different feed stream compositions as in Figure 24 or is trained with GCF, it slightly adjusts the flowsheet topology and the specifications of the unit operations.

4.4.4 Evolution of Long-Planned Recycles

Due to the sequential nature of the problem, the agent has to set up the flowsheet topology and the continuous specifications of the units upfront to make sense when a recycle is placed. We factorise the ranges for the continuous specifications into two discrete levels for fine discretization. However, even so, the number of available options for the continuous specifications is sufficiently larger than at other hierarchy levels (e.g., when the agent chooses out of the set of open streams). In the following, we will analyze how the agent learns to recycle streams and simultaneously set the continuous specifications to proper values.

During the early stages of the training process, the agent rarely places recycle streams (it evaluates their implications in the MCTS but never chooses them as final action, as they often fail). It focuses on learning about the available unit operations and their continuous specifications. Afterward, we observe that it concentrates on the placement of recycles. It learns that recycles affect streams added to the flowsheet in the future and streams that have already been dealt with in the past. It has to adjust the flowsheet topology and unit specifications to make sense in combination with a recycle stream, which will be set in the future. This implies that the specifications will only make sense once the recycle is set.

This will be shown alongside Figure 25, which shows two processes set up by the agent after being trained with GCF. Figure 25 a) shows a process for the separation of Ac and Ch using To as entrainer (contrary to Figure 24, where Be was used as entrainer). Inside this example, the agent set four continuous specifications for the units, i.e., the solvent ratio and the ratio of distillate to feed for all three distillation columns. The lower panels visualize these ratios for two of the distillation columns (marked pink and yellow). To the left, we show the flowsheet without the recycles placed by the agent. The pink column separates pure To and a binary mixture of Ac and Ch. From this mixture, it is impossible to separate Ch as a pure product by distillation because of the azeotrope (the feed is on the wrong side of the azeotrope, as shown in the ternary diagram). Therefore, no pure Ch leaves the process, and it seems that the agent sets the continuous specifications in the wrong way. After the agent places two recycle streams (shown in the right panel of Figure 25 a)), the compositions of all streams

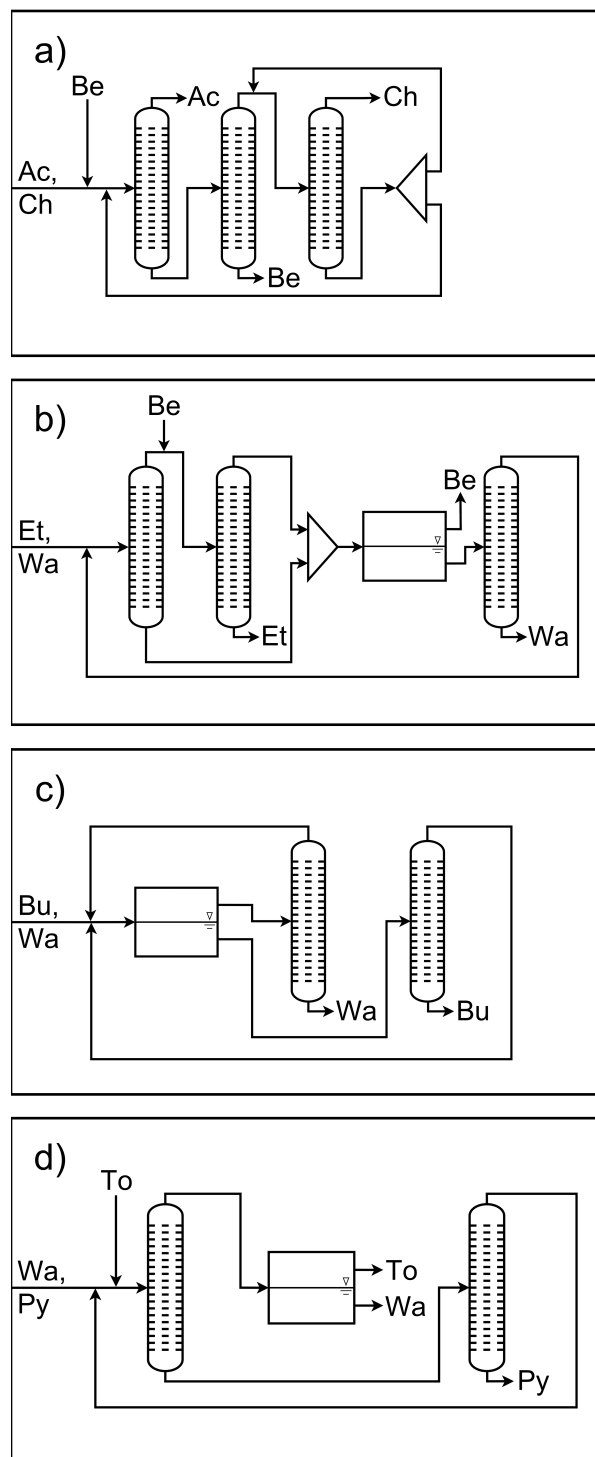


Figure 24: Flowsheets constructed by the trained agent for feed situations given in the literature [103–106] (training process was carried out using NPV). Flowsheet a) shows a process for the separation of Ac and Ch (feed composition: $x_{Ac} = 0.5, x_{Ch} = 0.5$) using Be as solvent. Flowsheet b) shows a process for the separation of Et and Wa (feed composition: $x_{Et} = 0.5, x_{Wa} = 0.5$) using solvent Be. Flowsheet c) shows a process for the separation of Bu and Wa (feed composition: $x_{Bu} = 0.4, x_{Wa} = 0.6$). Flowsheet d) shows a process for the separation of Py and Wa (feed composition: $x_{Py} = 0.1, x_{Wa} = 0.9$) using To as solvent.

of the flowsheet change drastically. It can be seen in the ternary diagrams that the previous set ratios led to the desired separation. Figure 25 b) shows a process for the separation of Wa and Py using To as solvent (also here we see that the agent chose a different design as in Figure 24). Inside this example, the agent set three continuous specifications for the units, i.e., the solvent ratio and the ratio of distillate to feed for both distillation columns. The process is displayed to the left without the later set recycle stream. The continuous specifications do not yield pure product streams inside the distillation column. Additionally, the decanter only separates pure Wa, but not pure To. To the right, the flowsheet is displayed with the chosen recycle. As in Figure 25 a), the placement of the recycle changes the compositions of all streams of the flowsheet (visualized in the ternary diagrams) and finally leads to a complete separation into pure products.

4.4.5 Discussion

The single-player RL framework for AFS introduced in Sections 3.2.4 and 4.4 enables training an agent from zero knowledge to synthesize near-optimal flowsheets for multiple chemical systems within a hybrid action space. The framework combines elements from the hierarchical SynGameZero approach (Sections 3.2.3 and 4.3), Gumbel AlphaZero [88], and the MLP-Mixer architecture [87]. The trained agent succeeds on a wide range of problems by constructing similar flowsheets as provided by the literature [103–106]. Furthermore, it was shown that the agent learns strong policies, which can be used without MCTS by unrolling with beam search.

The original and the hierarchical SynGameZero framework relied on a reformulation into a two-player game that enforced exploration. In the single-player framework, exploration is ensured by integrating the action selection proposed in Gumbel AlphaZero [88], which has two main advantages. On the one hand, the search procedure in AlphaZero does not guarantee a policy improvement when not all nodes at the root are visited [88]. While this was not an issue in the case studies used for the SynGameZero approach (original and hierarchical), it probably would have caused problems in the hybrid action space when numerous options are present at the root node. On the other hand, compared to AlphaZero, Gumbel AlphaZero was shown to work with significantly fewer simulations during training and evaluation.

The single-player framework addresses all challenges listed in Section 2.2. Nevertheless, there is room for improvement and further research. For example, the case studies in Section 4.4 consider only processes with up to three components. For a scale-up, one

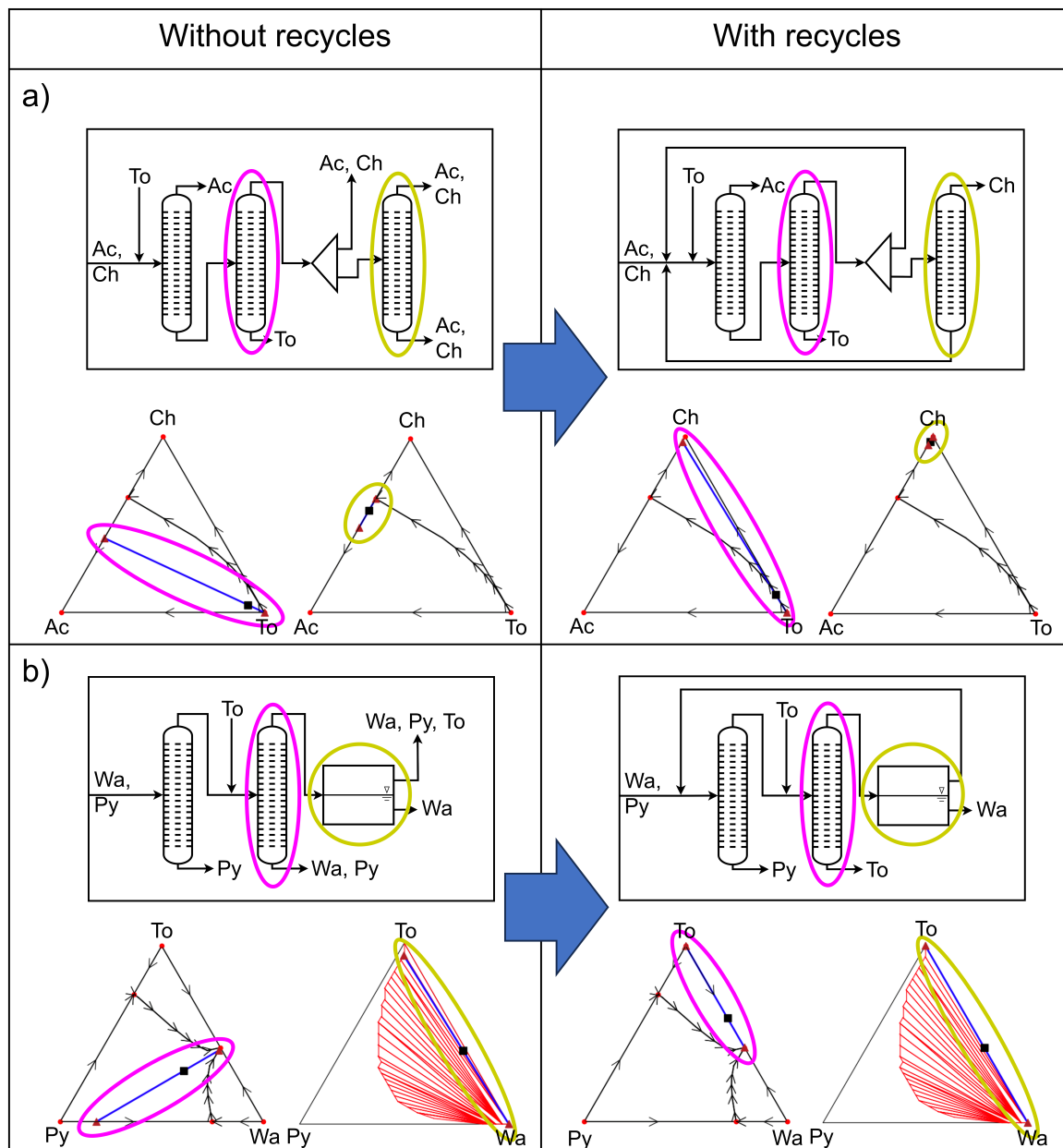


Figure 25: Examples for the implications of recycles on the compositions of the streams, which show the planning capabilities of the trained agent. The agent constructed the displayed examples after being trained using GCF. To the left, the flowsheets without recycles are shown (to the right, the flowsheets with recycles are shown). Inside the ternary diagrams, the feed stream is marked by a black square, and the output streams are marked with a brown triangle (connected by a blue line). Panel a) shows a process for the separation of Ac and Ch using To as entrainer (feed composition: $x_{Ac} = 0.74, x_{Ch} = 0.26$). Panel b) shows a process for the separation of Wa and Py using To as solvent (feed composition: $x_{Wa} = 0.04, x_{Py} = 0.96$).

would have to develop an approach for modeling VLEs and distillation columns with ∞/∞ -analysis for a variable number of components (for LLEs and decanters, this is already provided by the CEM that was introduced in Section 3.1.2). While Ryll et al. [62, 63] introduced such an approach for up to four components, it is not clear if further extensions are feasible in a computationally efficient way. Another possible addition to the single-player framework would be the integration of different unit operations, e.g., reactors and crystallizers. This extension would be straightforward, as one has to implement the respective unit operations into the existing framework. Finally, the most exciting direction for future research is undoubtedly transfer learning, i.e., evaluating the trained agent on systems it has not encountered throughout training. However, this is out of the scope of the present work, as it would require a larger number of systems than considered in the examples in Section 4.4.

5 Conclusion and Outlook

The present work presents a novel and general RL approach that enables training an agent from zero knowledge to set up processes for conceptual design problems in chemical engineering. Inside the RL framework, the agent learns only from interaction with a flowsheet simulation, i.e., by placing unit operations and deciding on their specifications. There are two main contributions of this work. On the one hand, a flowsheet simulation (i.e., the agent’s environment) based on short-cut models is provided. On the other hand, the stepwise development of the RL framework for AFS is presented.

All implemented unit operations are simulated using short-cut models to ensure a quick and robust evaluation of the proposed process designs. These short-cut models rely on phase equilibria, which are constructed by the environment automatically after providing the necessary interaction parameters of the respective chemical systems. While the implementation of most of the considered unit operations is based on already existing theoretical frameworks, a novel methodology for modeling liquid phase equilibria and decanters, namely the CEM, is developed in this work. A mathematical framework for calculating liquid phase equilibria for systems with an arbitrary number of components is provided. The approach is a generalization of a method [62, 82], which has already been shown to work for systems with up to four components. The generalization of the CEM is shown to work alongside various examples from the literature with up to six components. The computational complexity of constructing the convex envelope mainly limits application to higher order systems. To overcome this issue, further research on parallelization algorithms and effective virtual memory usage would be needed. As mentioned in [62, 82], another direction for further research could be the integration of vapor-liquid or solid-liquid equilibria into the framework.

Three RL frameworks are introduced in the present work, which incrementally address more and more of the challenges of AFS. The original SynGameZero framework models flowsheet synthesis as a competitive two-player game, which allows training an agent from scratch to set up processes using elements from the AlphaZero framework [54, 55] that was originally developed for board games such as Go and chess. Both players generate a process for the same initial set of feed streams, and the proposed designs are evaluated in a flowsheet simulation. The winner of the game, which the agent plays

against itself, is determined by comparing the NPV of the resulting flowsheets. The agent consists of an ACN interacting with an MCTS to determine promising actions. The approach is shown to work alongside simple flowsheet examples within a discrete action space. The hierarchical SynGameZero framework decomposes the action space into three hierarchy levels. This feature is employed on top of the two-player game, and the trained agent can now approach more sophisticated problem classes by determining unit specifications and placing recycles. Still, this framework works only in a discrete action space. The single-player framework combines several features from the hierarchical SynGameZero approach with recently developed methodologies from the RL research field, namely the Gumbel AlphaZero framework [88] and the MLP-Mixer [87]. It enables training the agent from scratch in a hybrid action space to set up processes for multiple chemical systems all at once. The approach was shown to work by training the agent to generate flowsheets to separate various azeotropic mixtures.

While the two-player game idea was not employed in the final RL framework, it is still an exciting option for other discrete planning problems outside of chemical engineering. For example, it has already been adapted by [107] to approach the traveling salesman problem and the job-shop scheduling problem. Further research is necessary to assess if there are more possible applications of the two-player game idea. On the AFS side, a possible direction for future research is to use the flowsheets generated by the agent as initialization for process optimization tools. However, the most exciting direction for future research is transfer learning. This means that the single-player framework could be used to train an agent on a large set of chemical systems to fulfill some predefined task, e.g., separating feed streams into pure components. Afterward, the agent is evaluated on a different set of chemical systems to see if it can transfer its learnings onto problems it has never encountered before.

In summary, this work shows that it is possible to train an RL agent without prior knowledge or heuristics to set up flowsheets for conceptual design using a simulation environment based on linearised thermodynamic models. Additionally, the developed methodologies pave the road for further research on flowsheet simulation, AFS, and the usage of RL for planning processes in general.

Bibliography

1. Westerberg, A. A retrospective on design and process synthesis. *Comput. Chem. Eng.* **28**, 447–458 (2004).
2. Siirola, J. Industrial Applications of Chemical Process Synthesis. *Adv. Chem. Eng.* **23**, 1–62 (1996).
3. Harmsen, G. Industrial best practices of conceptual process design. *Chem. Eng. Process.: Process Intensif.* **43**, 671–675 (2004).
4. O’Young, D. & Natori, Y. Process Synthesis: technology, environment and applications. *Comput. Chem. Eng.* **20**, 381–387 (1996).
5. Stephanopoulos, G. & Reklaitis, G. Process systems engineering: from solvay to modern bio- and nano technology. A history of development, successes and prospects for the future. *Chem. Eng. Sci.* **66**, 4272–4306 (2011).
6. Siirola, J. Strategic process synthesis: advances in the hierarchical approach. *Comput. Chem. Eng.* **20**, 1637–1643 (1996).
7. Chen, Q. & Grossmann, I. Recent developments and challenges in optimization-based process synthesis. *Annu. Rev. Chem. Biomol. Eng.* **8**, 249–283 (2017).
8. Yeomans, H. & Grossmann, I. A systematic modeling framework of superstructure optimization in process synthesis. *Comput. Chem. Eng.* **23**, 709–731 (1999).
9. Stephanopoulos, G. & Westerberg, A. Studies in process synthesis II, evolutionary synthesis of optimal process flowsheets. *Chem. Eng. Sci.* **31**, 195–204 (1976).
10. Zhang, T., Sahinidis, N. & Siirola, J. Pattern recognition in chemical process flowsheets. *AIChE J.* **65**, 592–603 (2019).
11. Gani, R. & O’Connell, J. A knowledge based system for the selection of thermodynamic models. *Comput. Chem. Eng.* **13**, 397–404 (1989).
12. Kirkwood, R., Locke, M. & Douglas, J. A prototype expert system for synthesizing chemical process flowsheets. *Comput. Chem. Eng.* **12**, 329–343 (1988).
13. Tula, A., Eden, M. & Gani, R. Process synthesis, design and analysis using a process-group contribution method. *Comput. Chem. Eng.* **81**, 245–259 (2015).

14. Daichendt, M. & Grossmann, I. Integration of hierarchical decomposition and mathematical programming for the synthesis of process flowsheets. *Comput. Chem. Eng.* **22**, 147–175 (1997).
15. Martin, M. & Adams, T. Challenges and future directions for process and product synthesis and design. *Comput. Chem. Eng.* **128**, 421–436 (2019).
16. Grossmann, I. & Harjunoski, I. Process systems engineering: academic and industrial perspectives. *Comput. Chem. Eng.* **126**, 474–484 (2019).
17. Montastruc, L., Belletante, S., Pagot, A., Negny, S. & Raynal, L. From conceptual design to process design optimization: a review on flowsheet synthesis. *Oil Gas Sci. Technol.* **74** (2019).
18. Mencarelli, L., Chen, Q., Pagot, A. & Grossmann, I. A review on superstructure optimization approaches in process system engineering. *Comput. Chem. Eng.* **136** (2020).
19. Stephanopoulos, G. Artificial intelligence in process engineering - current state and future trends. *Comput. Chem. Eng.* **14**, 1259–1270 (1990).
20. Stephanopoulos, G. & Han, C. Intelligent systems in process engineering: a review. *Comput. Chem. Eng.* **20**, 143–191 (1996).
21. Dimiduk, D., Holm, E. & Niezgoda, S. Perspectives on the impact of machine learning, deep learning, and artificial intelligence on materials, processes, and structures engineering. *Integr. Mater. Manuf. I.* **7**, 157–172 (2018).
22. Lee, J., Shin, J. & Realff, M. Machine learning: overview of the recent progresses and implications for the process systems engineering field. *Comput. Chem. Eng.* **114**, 111–121 (2018).
23. Venkatasubramanian, V. The promise of artificial intelligence in chemical engineering: is it here, finally? *AIChE J.* **65**, 466–478 (2019).
24. Dobbelaere, M., Plehiers, P., Van de Vijver, R., Stevens, C. & Van Geem, K. Machine Learning in Chemical Engineering: Strengths, Weaknesses, Opportunities, and Threats. *Engineering* **7**, 1201–1211 (2021).
25. Schweidtmann, A. *et al.* Machine Learning in Chemical Engineering: A Perspective. *Chem. Ing. Tech.* **93**, 2029–2039 (2021).
26. Fahmi, I. & Cremaschi, S. Process synthesis of biodiesel production plant using artificial neural networks as the surrogate models. *Comput. Chem. Eng.* **46**, 105–123 (2012).

27. Moghadam, M. & Asgharzadeh, S. On the application of artificial neural network for modeling liquid-liquid equilibrium. *J. Mol. Liq.* **220**, 339–345 (2016).
28. Reynel-Avila, H., Bonilla-Petriciolet, A. & Tapia-Picazo, J. An artificial neural network-based NRTL model for simulating liquid-liquid equilibria of systems present in biofuels production. *Fluid Ph. Equilibria* **483**, 153–164 (2019).
29. McBride, K. & Sundmacher, K. Overview of Surrogate Modeling in Chemical Process Engineering. *Chem. Ing. Tech.* **91**, 1–13 (2019).
30. Nentwich, C. & Engell, S. Surrogate modeling of phase equilibrium calculations using adaptive sampling. *Comput. Chem. Eng.* **126**, 204–217 (2019).
31. Jirasek, F. *et al.* Machine Learning in Thermodynamics: Prediction of Activity Coefficients by Matrix Completion. *J. Phys. Chem.* **11**, 981–985 (2020).
32. Yang, K. *et al.* Analyzing Learned Molecular Representations for Property Prediction. *J. Chem. Inf. Model.* **59**, 3370–3388 (2019).
33. Coley, C. *et al.* A graph-convolutional neural network model for the prediction of chemical reactivity. *Chem. Sci.* **10**, 370–377 (2019).
34. Fernandes, F. Optimization of fischer-tropsch synthesis using neural networks. *Chem. Eng. Technol.* **29**, 449–453 (2006).
35. Eason, J. & Cremaschi, S. Adaptive sequential sampling for surrogate model generation with artificial neural networks. *Comput. Chem. Eng.* **68**, 220–232 (2014).
36. Schäfer, P., Caspari, A., Kleinhans, K., Mhamdi, A. & Mitsos, A. Reduced dynamic modeling approach for rectification columns based on compartmentalization and artificial neural networks. *AIChE J.* **65**, e16568 (2019).
37. Sutton, R. & Barto, A. *Reinforcement Learning: An Introduction* (The MIT Press, Cambridge, 2018).
38. Shin, J., Badgwell, T., Liu, K. & Lee, J. Reinforcement learning - overview of recent progress and implications for process control. *Comput. Chem. Eng.* **127**, 282–294 (2019).
39. Nian, R., Liu, J. & Huang, B. A Review on Reinforcement Learning: Introduction and Applications in Industrial Process Control. *Comput. Chem. Eng.* **139**, 106886 (2020).
40. Zhou, Z., Li, X. & Zare, R. Optimizing chemical reactions with deep reinforcement learning. *ACS Cent. Sci.* **3**, 1337–1344 (2017).

41. Wang, X. *et al.* Towards efficient discovery of green synthetic pathways with Monte Carlo tree search and reinforcement learning. *Chem. Sci.* **11**, 10959–10972 (40 2020).
42. Göttl, Q., Grimm, D. & Burger, J. Automated synthesis of steady-state continuous processes using reinforcement learning. *Front. Chem. Sci. Eng.* **16**, 288–302 (2022).
43. Göttl, Q., Grimm, D. & Burger, J. Automated Process Synthesis Using Reinforcement Learning. *Comput. Aided Chem. Eng.* **50**, 209–214 (2021).
44. Göttl, Q., Tönges, Y., Grimm, D. & Burger, J. Automated Flowsheet Synthesis Using Hierarchical Reinforcement Learning: Proof of Concept. *Chem. Ing. Tech.* **93**, 2010–2018 (2021).
45. Göttl, Q., Grimm, D. & Burger, J. Using Reinforcement Learning in a Game-like Setup for Automated Process Synthesis without Prior Process Knowledge. *Comput. Aided Chem. Eng.* **49**, 1555–1560 (2022).
46. Göttl, Q., Pirnay, J., Grimm, D. & Burger, J. Convex Envelope Method for determining liquid multi-phase equilibria in systems with arbitrary number of components. *Comput. Chem. Eng.* **177**, 108321 (2023).
47. Göttl, Q., Pirnay, J., Burger, J. & Grimm, D. Deep reinforcement learning uncovers processes for separating azeotropic mixtures without prior knowledge. <https://doi.org/10.48550/arXiv.2310.06415> (2023).
48. Midgley, L. Deep Reinforcement Learning for Process Synthesis. <https://doi.org/10.48550/arXiv.2009.13265> (2020).
49. Khan, A. & Lapkin, A. Searching for optimal process routes: A reinforcement learning approach. *Comput. Chem. Eng.* **141**, 107027 (2020).
50. Khan, A. & Lapkin, A. Designing the process designer: Hierarchical reinforcement learning for optimisation-based process design. *Chem. Eng. Process.* **180**, 108885 (2022).
51. Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. Proximal Policy Optimization Algorithms. <https://doi.org/10.48550/arXiv.1707.06347> (2017).
52. Seidenberg, J., Khan, A. & Lapkin, A. Boosting autonomous process design and intensification with formalized domain knowledge. *Comput. Chem. Eng.* **169**, 108097 (2023).

53. Stops, L., Leenhouts, R., Gao, Q. & Schweidtmann, A. Flowsheet generation through hierarchical reinforcement learning and graph neural networks. *AIChE J.* **69**, 17938 (2023).
54. Silver, D. *et al.* Mastering the game of Go without human knowledge. *Nature* **550**, 354–359 (2017).
55. Silver, D. *et al.* A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* **362**, 1140–1144 (2018).
56. Fawzi, A. *et al.* Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature* **610**, 47–53 (2022).
57. Mankowitz, D. *et al.* Faster sorting algorithms discovered using deep reinforcement learning. *Nature* **618**, 257–263 (2023).
58. Hausknecht, M. & Stone, P. Deep Reinforcement Learning in Parameterized Action Space. *ICLR* (2016).
59. Vinyals, O. *et al.* Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **575**, 350–354 (2019).
60. Biegler, L., Grossmann, I. & Westerberg, A. *Systematic Methods of Chemical Process Design* (Prentice Hall PTR, 1997).
61. Bekiaris, N. & Morari, M. Multiple Steady States in Distillation: Inf/Inf Predictions, Extensions, and Implications for Design, Synthesis, and Simulation. *Ind. Eng. Chem. Res.* **35**, 4264–4280 (1996).
62. Ryll, O. *Thermodynamische Analyse gekoppelter Reaktions-Destillations-Prozesse: konzeptioneller Entwurf, Modellierung, Simulation und experimentelle Validierung (in German)* PhD thesis (University of Stuttgart, Stuttgart, Germany, 2009).
63. Ryll, O., Blagov, S. & Hasse, H. Inf/Inf Analysis of homogeneous distillation processes. *Chem. Eng. Sci.* **84**, 315–332 (2012).
64. Ryll, O., Blagov, S. & Hasse, H. Thermodynamic analysis of reaction-distillation processes based on piecewise linear models. *Chem. Eng. Sci.* **109**, 284–295 (2014).
65. Virtanen, P. *et al.* SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* **17**, 261–272 (2020).
66. Sorensen, J., Magnussen, T., Rasmussen, P. & Fredenslund, A. Liquid-liquid equilibrium data: Their retrieval, correlation and prediction, Part II: Correlation. *Fluid Ph. Equilibria* **3**, 47–82 (1979).
67. Swank, D. & Mullins, J. Evaluation of methods for calculating liquid-liquid phase-splitting. *Fluid Ph. Equilibria* **30**, 101–110 (1986).

68. Teh, Y. & Rangaiah, G. A Study of Equation-Solving and Gibbs Free Energy Minimization Methods for Phase Equilibrium Calculations. *Chem. Eng. Res. Des.* **80**, 759 (2002).
69. Baker, L., Pierce, A. & Luks, K. Gibbs Energy Analysis of Phase Equilibria. *Soc. Pet. Eng. J.* **22**, 731–742 (1982).
70. Gibbs, J. A method of geometrical representation of the thermodynamic properties of substances by means of surfaces. *Trans. Conn. Acad. Arts Sci.* **2**, 382–404 (1873).
71. Gibbs, J. On the Equilibrium of Heterogeneous Substances. *Trans. Conn. Acad. Arts Sci.* **3**, 108–248 (1876).
72. Michelsen, M. The isothermal flash problem. Part I. Stability. *Fluid Ph. Equilibria* **9**, 1–19 (1982).
73. Michelsen, M. The isothermal flash problem. Part II. Phase-split calculation. *Fluid Ph. Equilibria* **9**, 21–40 (1982).
74. McDonald, C. & Floudas, C. Global optimization for the phase and chemical equilibrium problem: Application to the NRTL equation. *Comput. Chem. Eng.* **19**, 1111–1139 (1995).
75. McDonald, C. & Floudas, C. Global optimization for the phase stability problem. *AIChE J.* **41**, 1798–1814 (1995).
76. Mitsos, A. & Barton, P. A dual extremum principle in thermodynamics. *AIChE J.* **53**, 2131–2147 (2007).
77. Wasykiewicz, S., Li, Y., Satyro, M. & Wasykiewicz, M. Application of a global optimization algorithm to phase stability and liquid–liquid equilibrium calculations. *Fluid Ph. Equilibria* **358**, 304–318 (2013).
78. Rangaiah, G. Evaluation of genetic algorithms and simulated annealing for phase equilibrium and stability problems. *Fluid Ph. Equilibria* **187–188**, 83–109 (2001).
79. Bonilla-Petriciolet, A., Rangaiah, G. & Segovia-Hernandez, J. Constrained and unconstrained Gibbs free energy minimization in reactive systems using genetic algorithm and differential evolution with tabu list. *Fluid Ph. Equilibria* **300**, 120–134 (2011).
80. Zhang, H., Bonilla-Petriciolet, A. & Rangaiah, G. A Review on Global Optimization Methods for Phase Equilibrium Modeling and Calculations. *The Open Thermodynamics Journal* **5**, 71–92 (2011).

81. Piro, M. & Simunovic, S. Global optimization algorithms to compute thermodynamic equilibria in large complex systems with performance considerations. *Comput. Mater. Sci.* **118**, 87–96 (2016).
82. Ryll, O., Blagov, S. & Hasse, H. Convex envelope method for the determination of fluid phase diagrams. *Fluid Ph. Equilibria* **324**, 108–116 (2012).
83. Rowlinson, J. & Swinton, F. *Liquids and Liquid Mixtures* (Butterworth-Heinemann, 1982).
84. Barto, A. & Mahadevan, S. Recent Advances in Hierarchical Reinforcement Learning. *Discrete Event Dyn.* **13**, 341–379 (2003).
85. Russell, S. & Zimdars, A. Q-decomposition for reinforcement learning agents. *ICML*, 656–663 (2003).
86. Bishop, C. *Pattern Recognition and Machine Learning* (Springer, New York, 2006).
87. Tolstikhin, I. *et al.* MLP-mixer: An all-MLP Architecture for Vision. *NeurIPS*, 24261–24272 (2021).
88. Danihelka, I., Guez, A., Schrittwieser, J. & Silver, D. Policy improvement by planning with gumbel. *ICLR* (2022).
89. Vaswani, A. *et al.* Attention is all you need. *NeurIPS*, 5998–6008 (2017).
90. Yellott Jr., J. The relationship between Luce’s choice axiom, Thurstone’s theory of comparative judgment, and the double exponential distribution. *J. Math. Psychol.* **15**, 109–144 (1977).
91. Kool, W., van Hoof, H. & Welling, M. Stochastic Beams and Where to Find Them: The Gumbel-Top-k Trick for Sampling Sequences Without Replacement. *ICML* (2019).
92. Karnin, Z., Koren, T. & Somekh, O. Almost Optimal Exploration in Multi-Armed Bandits. *ICML* (2013).
93. Prausnitz, J., Lichtenthaler, R. & Azevedo, E. *Molecular Thermodynamics of Fluid-Phase Equilibria* (Prentice Hall PTR, 1999).
94. Chen, J., Duan, L., Mi, J., Fei, W. & Li, Z. Liquid–liquid equilibria of multi-component systems including n-hexane, n-octane, benzene, toluene, xylene and sulfolane at 298.15 K and atmospheric pressure. *Fluid Ph. Equilibria* **173**, 109–119 (2000).

95. Chen, J., Mi, J., Fei, W. & Li, Z. Liquid-Liquid Equilibria of Quaternary and Quinary Systems Including Sulfolane at 298.15 K. *J. Chem. Eng. Data* **46**, 169–171 (2001).
96. Yuan, S., Li, S., Yin, H. & Chen, Z. Liquid-Liquid Equilibria for Systems of Ethanol + Hexanol + Heptanol + Decane + Undecane + Water at 298.15 K under Atmospheric Pressure: Experiment and Simulation. *J. Chem. Eng. Data* **63**, 1851–1858 (2018).
97. Yuan, S., Chen, Y., Yin, H. & Chen, Z. Liquid-Liquid Equilibria for Systems of Ethanol + Octanol + Nonanol + Dodecane + Tridecane + Water at Different Temperatures under Atmospheric Pressure. *J. Chem. Eng. Data* **64**, 3008–3017 (2019).
98. Yuan, S., Bao, G., Yin, H. & Chen, Z. Liquid-Liquid Equilibrium Data and Process Simulation for Separating the Mixture of Decanol + Undecanol + Tetradecane + Pentadecane. *J. Chem. Eng. Data* **65**, 5154–5175 (2020).
99. Barber, C., Dobkin, D. & Huhdanpaa, H. The Quickhull algorithm for convex hulls. *Mathematical Software* **22**, 469–483 (1996).
100. Amato, N., Goodrich, M. & Ramos, E. Parallel algorithms for higher-dimensional convex hulls. *Proceedings 35th Annual Symposium on Foundations of Computer Science* (1994).
101. Blleloch, G., Gu, Y., Shun, J. & Sun, Y. Randomized Incremental Convex Hull is Highly Parallel. *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures*, 103–115 (2020).
102. Chadnov, R. & Skvortsov, A. Convex hull algorithms review. *The 8th Russian-Korean International Symposium on Science and Technology* (2004).
103. Wang, Y.-H. & Chien, I.-L. Unique Design Considerations for Maximum-Boiling Azeotropic Systems via Extractive Distillation: Acetone/Chloroform Separation. *Ind. Eng. Chem. Res.* **57**, 12884–12894 (2018).
104. Kunnakorn, D. *et al.* Techno-economic comparison of energy usage between azeotropic distillation and hybrid system for water-ethanol separation. *Renew. Energ.* **51**, 310–316 (2013).
105. Luyben, W. Control of the Heterogeneous Azeotropic n-Butanol/Water Distillation System. *Energ. Fuel.* **22**, 4249–4258 (2008).
106. Chen, Y.-C., Li, K.-L., Chen, C.-L. & Chien, I.-L. Design and Control of a Hybrid Extraction/Distillation System for the Separation of Pyridine and Water. *Ind. Eng. Chem.* **54**, 7715–7727 (2015).

107. Pirnay, J., Göttl, Q., Burger, J. & Grimm, D. Policy-Based Self-Competition for Planning Problems. *ICLR* (2023).
108. Petlyuk, F. *Distillation Theory and Its Application to Optimal Design of Separation Units* (Cambridge University Press, 2004).
109. Rockafellar, T. R. *Convex Analysis* (Princeton University Press, 1970).
110. Abadi, M. *et al.* TensorFlow: Large-scale machine learning on heterogeneous systems. <https://doi.org/10.48550/arXiv.1603.04467> (2015).
111. Kingma, D. & Ba., J. Adam: A method for stochastic optimization. *ICLR* (2015).
112. Vila, M., Cunill, F., Izquierdo, J.-F., Tejero, J. & Iborra, M. Equilibrium constants for ethyl tert-butyl ether liquid-phase synthesis. *Chem. Eng. Commun.* **124**, 223–232 (1993).
113. Daniel, G. & Jobson, M. Conceptual Design of Equilibrium Reactor-Reactive Distillation Flowsheets. *Ind. Eng. Chem. Res.* **46**, 559–570 (2007).
114. Domingues, L., Pinheiro, C. & Oliveira, N. Economic comparison of a reactive distillation-based process with the conventional process for the production of ethyl tert-butyl ether (ETBE). *Comput. Chem. Eng.* **100**, 9–26 (2017).
115. Towler, R. & Sinnott, G. *Chemical Engineering Design: Principles, Practice and Economics of Plant and Process Design* (Butterworth-Heinemann, 2022).
116. Paszke, A. *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. *NeurIPS*, 8024–8035 (2019).
117. Aspen Technology Inc. *Aspen Plus (version 8.8)* 2015.
118. Linstrom, P. & Mallard, W. *NIST Chemistry WebBook* (National Institute of Standards and Technology, Gaithersburg MD, 20899, 2023).
119. DDBST GmbH. *Dortmund Data Bank 2023*. www.ddbst.com.
120. Krey, U. & Owen, A. *Basic Theoretical Physics* (Springer, Berlin, Heidelberg, 2007).

Appendix

A Appendix A

AI Modeling of Distillation Columns: ∞/∞ -Analysis

Throughout this work, linearised representations for VLEs and distillation lines are used. This form of representation allows fast and robust modeling of a distillation column using the ∞/∞ -approach [61–63], assuming an infinite number of stages and total reflux. These assumptions model a thermodynamic limiting case, which has been shown to display similar behavior as real distillation columns. For a general overview regarding phase diagrams used to analyse distillation processes, we refer to [108]. For a detailed description of the ∞/∞ -approach, we refer to [62, 63] and outline the basic concepts in the following for the ternary system shown in Figure A1.

In Figure A1, distillation lines for the ternary system acetone – benzene – chloroform at 1 bar are displayed. The red points indicate singular points, i.e., pure components and azeotropes. The binary maximum azeotrope and benzene span the distillation boundary, which separates the composition space into two distillation regions. The arrows indicate the direction of the distillation lines toward the low-boiler.

Assuming an infinite number of stages and total reflux allows specification of a distillation column with one parameter: the ratio of distillate to feed flowrate \dot{n}^D/\dot{n}^F . Given a feed stream composition, distillate and bottom product can be determined using the following rules:

- 1) Feed, distillate, and bottom product are located on one straight line, which satisfies the lever arm rule and the specified value of \dot{n}^D/\dot{n}^F .

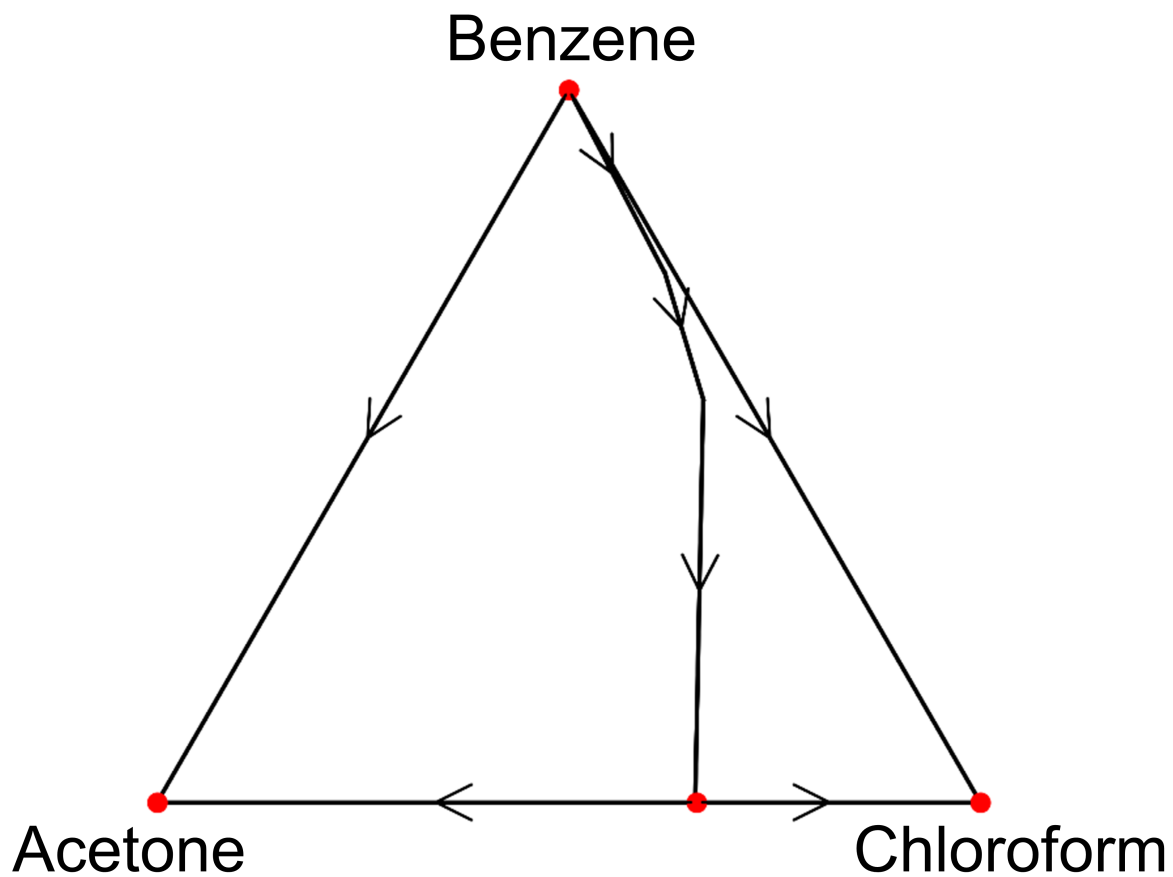


Figure A1: Topology of the VLE in the ternary system acetone – benzene – chloroform at 1 bar. The binary azeotrope and benzene span the distillation boundary. The arrows indicate the direction of the distillation lines toward the low-boiler.

-
- II) The distillate and bottom product are on the same distillation line. Note that this implies that the distillate and bottom product are in the same distillation region.
 - III) One of the following three cases applies: the distillate is the local low-boiler (i.e., the low-boiler of this distillation region); the bottom product is the local high-boiler (i.e., the high-boiler of this distillation region); the distillation line, where distillate and bottom product are located on, passes a saddle point (i.e., a singular point, which is neither the low- or high-boiler of this distillation region).

Using this concept, distillation columns can be modelled quickly and robustly, even for systems that display complicated, azeotropic behavior.

A flowsheet simulation based on the methodology described in [62, 63] was implemented for the present work. The linearised representations of VLE and corresponding distillation lines are generated in an automated way after providing necessary property data (e.g., binary interaction parameters for a g^E -model as NRTL [93]).

B Appendix B

BI Modeling of Decanters: Convex Envelope Method

BI.1 Mathematics

BI.1.1 Simplex Geometry

Definition B1 (Convex envelope/hull). Let $k, n \in \mathbb{N}$ and $\mathbf{u}_1, \dots, \mathbf{u}_k \in \mathbb{R}^n$. The *convex envelope* or *convex hull* of $\mathbf{u}_1, \dots, \mathbf{u}_k$ is the smallest convex set in \mathbb{R}^n , which contains $\mathbf{u}_1, \dots, \mathbf{u}_k$. We refer to it with $\text{conv}(\{\mathbf{u}_1, \dots, \mathbf{u}_k\})$.

Definition B2 (k -simplex). Let $k, n \in \mathbb{N}$ with $k \leq n$ and $\mathbf{u}_1, \dots, \mathbf{u}_{k+1} \in \mathbb{R}^n$ so that $\mathbf{u}_1 - \mathbf{u}_{k+1}, \dots, \mathbf{u}_k - \mathbf{u}_{k+1}$ are linearly independent. A k -simplex U with vertices $\mathbf{u}_1, \dots, \mathbf{u}_{k+1}$ is defined as the convex envelope $\text{conv}(\{\mathbf{u}_1, \dots, \mathbf{u}_{k+1}\})$ of the points $\mathbf{u}_1, \dots, \mathbf{u}_{k+1}$. We often say U is represented by $\mathbf{u}_1, \dots, \mathbf{u}_{k+1}$.

Remark B1 (Simplex representation of N -component system). A system consisting of $N \in \mathbb{N}$ components can always be interpreted as a n -simplex in \mathbb{R}^n with $n = N - 1$. For example, a 3-component system can be visualized in \mathbb{R}^2 as a 2-simplex (e.g., an equilateral triangle). When mixtures in a N -component system are examined, usually molar fractions are used (e.g., to calculate thermodynamic properties). To use results from geometry inside the simplex representing the system, one has to transform between molar fractions and cartesian coordinates. This can be done using barycentric coordinates (for a proof, see, for example, [109]).

Lemma B1 (Barycentric coordinates). Let $k, n \in \mathbb{N}$ with $k \leq n$ and $\mathbf{u}_1, \dots, \mathbf{u}_{k+1} \in \mathbb{R}^n$ be the vertices of a k -simplex U . For every point $\mathbf{a} \in U$ there exist unique $\lambda_1, \dots, \lambda_{k+1} \in [0, 1]$ so that

$$\mathbf{a} = \sum_{i=1}^{k+1} \lambda_i \mathbf{u}_i \quad \text{and} \quad \sum_{i=1}^{k+1} \lambda_i = 1. \quad (\text{B1})$$

$\lambda_1, \dots, \lambda_{k+1}$ are called *barycentric coordinates* of \mathbf{a} with respect to U .

Remark B2. Consider a N -component system, represented by a n -simplex U with $n = N - 1$. Barycentric coordinates $\lambda_1, \dots, \lambda_N$ of a point $\mathbf{a} \in U$ just describe the position of \mathbf{a} with respect to the simplex' vertices. For a possible mixture inside the system, described by molar fractions (x_1, \dots, x_N) , it holds

$$\sum_{i=1}^N x_i = 1 \quad (\text{B2})$$

and $x_i \geq 0$ for all $i = 1, \dots, N$. Because of the uniqueness statement in Lemma B1, it follows immediately that the molar fractions (x_1, \dots, x_N) are the barycentric coordinates of a point $\mathbf{a} \in U$. This yields a possibility for transformation between molar fractions and cartesian coordinates by writing Equation B1 in matrix form (and computing the left-inverse of the left matrix).

BI.1.2 Discretization of the Composition Space

Remark B3. A binary system can be discretized by specifying a minimal distance $1/\delta$, with $\delta \in \mathbb{N}$, for neighboring points. The whole discretization space then contains $\delta + 1$ points. In general, when a system with $N \in \mathbb{N}$ components is considered, we define the set of discretization points by

$$\mathcal{P}_\delta := \left\{ \left(\frac{p_1}{\delta}, \dots, \frac{p_N}{\delta} \right) \mid p_i \in \mathbb{N} \text{ for } i = 1, \dots, N \text{ and } \sum_{i=1}^N p_i = \delta \right\}. \quad (\text{B3})$$

Note that the elements of \mathcal{P}_δ are the molar fractions \mathbf{x} of the discretized points. As explained before, molar fractions $\mathbf{x} \in \mathcal{P}_\delta$ are identical to barycentric coordinates and thus can be transformed to cartesian coordinates with the help of Lemma B1. It is easy to see that every binary subsystem of \mathcal{P}_δ consists once again of $\delta + 1$ points, but the total number of points increases exponentially with N . When applying the CEM to a system with N components, we construct a n -simplex ($n = N - 1$) with unitary edge length, representing the whole composition space. This space then is discretized, as explained above.

BI.2 Implementation

The CEM framework was implemented with the programming language Python. The convex envelope is found by the usage of the QuickHull-algorithm [99], which is provided by the Qhull library within the package `scipy` [65]. An implementation, which allows execution of step III) from Section 3.1.2.1 parallelized, is published via GitHub:

<https://github.com/grimmlab/cem>. All results for the CEM were generated on a machine with an AMD EPYC 7542 32-core (64 threads) processor and 512 GB of RAM.

BI.3 Detailed Results

Here, we report detailed results obtained for systems containing up to six components examined in [94–98]. Additionally to MD, we report the computation time (CT) needed for our implementation of the CEM to construct the respective phase equilibria. For ternary systems, the classification of the simplices of the convex envelope was executed serially. For systems containing more than three components, the classification step was executed parallelized (the respective settings for parallelization are published alongside the code on GitHub). Table B1 shows results for several ternary systems containing combinations of *n*-hexane, benzene, sulfolane, toluene, xylene, and *n*-octane. NRTL parameters and experimental data regarding those systems were taken from [94]. As can be seen, the calculated phase splits match the experimental data quite well. Table B2 shows results for quaternary systems from various sources [94–98]. NRTL and UNIQUAC parameters were used to calculate the occurring phase splits. Compared to ternary systems, δ was set to a lower value to limit the computational effort. Still, our approach was able to calculate the experimental data with excellent accuracy. Table B3 shows results for quinary systems from [94, 95] using the NRTL model. The CEM was able to calculate the occurring phase splits with high accuracy. Table B4 shows results for systems containing six components, as presented in [96–98]. The discretization parameter δ was set to a lower value to control computational complexity. Still, the CEM was able to calculate the occurring phase splits with an acceptable accuracy.

Table B1: Results for ternary systems from [94] at atmospheric pressure. The parameter δ was set to 128, and the NRTL model was used for all systems. M describes the number of feed streams that were examined for the calculation of the MD.

System	$T \setminus K$	M	$CT \setminus s$	MD
<i>n</i> -hexane – benzene – sulfolane	298.15	10	1.8	0.005
<i>n</i> -hexane – toluene – sulfolane	298.15	10	1.7	0.004
<i>n</i> -hexane – xylene – sulfolane	298.15	10	1.7	0.006
<i>n</i> -octane – benzene – sulfolane	298.15	10	1.7	0.005
<i>n</i> -octane – toluene – sulfolane	298.15	10	1.7	0.009
<i>n</i> -octane – xylene – sulfolane	298.15	10	1.7	0.005

Table B2: Results for quaternary systems at atmospheric pressure. The parameter δ was set to 64 for all systems. M describes the number of feed streams that were examined for the calculation of the MD.

System	$T \setminus K$	M	g^E -model	CT \ s	MD
<i>n</i> -hexane – benzene – xylene – sulfolane [94]	298.15	5	NRTL	41.6	0.003
<i>n</i> -hexane – <i>n</i> -octane – benzene – sulfolane [94]	298.15	5	NRTL	39.2	0.005
<i>n</i> -octane – toluene – xylene – sulfolane [94]	298.15	5	NRTL	41.4	0.004
heptane – benzene – toluene – sulfolane [95]	298.15	5	NRTL	41.6	0.011
heptane – octane – <i>m</i> -xylene – sulfolane [95]	298.15	5	NRTL	39.7	0.007
hexane – heptane – toluene – sulfolane [95]	298.15	5	NRTL	38.2	0.005
ethanol – heptanol – decane – water [96]	298.15	21	UNIQUAC	49.4	0.007
ethanol – heptanol – undecane – water [96]	298.15	22	UNIQUAC	50.4	0.008
ethanol – hexanol – decane – water [96]	298.15	24	UNIQUAC	48.6	0.005
ethanol – hexanol – undecane – water [96]	298.15	29	UNIQUAC	49.4	0.007
ethanol – nonanol – dodecane – water [97]	293.15	21	NRTL	49.6	0.016
ethanol – nonanol – dodecane – water [97]	298.15	13	NRTL	50.5	0.016
ethanol – nonanol – dodecane – water [97]	303.15	17	NRTL	48.8	0.011
ethanol – nonanol – tridecane – water [97]	293.15	21	NRTL	49.8	0.009

Table B2 (continued): Results for quaternary systems at atmospheric pressure. The parameter δ was set to 64 for all systems. M describes the number of feed streams that were examined for the calculation of the MD.

ethanol – nonanol – tridecane – water [97]	298.15	18	NRTL	49.7	0.013
ethanol – nonanol – tridecane – water [97]	303.15	14	NRTL	49.6	0.009
ethanol – octanol – dodecane – water [97]	293.15	22	NRTL	46.4	0.011
ethanol – octanol – dodecane – water [97]	298.15	16	NRTL	49.6	0.013
ethanol – octanol – dodecane – water [97]	303.15	17	NRTL	49.6	0.009
ethanol – octanol – tridecane – water [97]	293.15	21	NRTL	48.5	0.009
ethanol – octanol – tridecane – water [97]	298.15	14	NRTL	48.7	0.013
ethanol – octanol – tridecane – water [97]	303.15	12	NRTL	48.5	0.012
ethanol – decanol – pentadecane – water [98]	293.15	23	UNIQUAC	49.7	0.006
ethanol – decanol – pentadecane – water [98]	298.15	22	UNIQUAC	49.5	0.006
ethanol – decanol – pentadecane – water [98]	303.15	24	UNIQUAC	49.7	0.007
ethanol – decanol – tetradecane – water [98]	293.15	21	UNIQUAC	47.8	0.007
ethanol – decanol – tetradecane – water [98]	298.15	24	UNIQUAC	49.4	0.007
ethanol – decanol – tetradecane – water [98]	303.15	23	UNIQUAC	49.5	0.006
ethanol – undecanol – pentadecane – water [98]	293.15	24	UNIQUAC	49.4	0.007

Table B2 (continued): Results for quaternary systems at atmospheric pressure. The parameter δ was set to 64 for all systems. M describes the number of feed streams that were examined for the calculation of the MD.

ethanol – undecanol – pentadecane – water [98]	298.15	21	UNIQUAC	49.7	0.007
ethanol – undecanol – pentadecane – water [98]	303.15	22	UNIQUAC	49.5	0.006
ethanol – undecanol – tetradecane – water [98]	293.15	23	UNIQUAC	49.4	0.006
ethanol – undecanol – tetradecane – water [98]	298.15	23	UNIQUAC	49.1	0.007
ethanol – undecanol – tetradecane – water [98]	303.15	23	UNIQUAC	48.0	0.007

Table B3: Results for quinary systems at atmospheric pressure. The parameter δ was set to 32, and the NRTL model was used for all systems. M describes the number of feed streams that were examined for the calculation of the MD.

System	$T \setminus K$	M	$CT \setminus s$	MD
<i>n</i> -hexane – <i>n</i> -octane – benzene – toluene – sulfolane [94]	298.15	4	170.4	0.010
heptane – octane – benzene – <i>m</i> -xylene – sulfolane [95]	298.15	5	161.0	0.012
hexane – heptane – toluene – <i>m</i> -xylene – sulfolane [95]	298.15	5	165.6	0.015

Table B4: Results for systems containing six components at atmospheric pressure. The parameter δ was set to 16 for all systems. M describes the number of feed streams that were examined for the calculation of the MD.

System	$T \setminus K$	M	g^E -model	CT \ s	MD
ethanol – hexanol – heptanol – decane – undecane – water [96]	298.15	22	UNIQUAC	3019.5	0.032
ethanol – octanol – nonanol – dodecane – tridecane – water [97]	293.15	20	NRTL	3000.0	0.036
ethanol – octanol – nonanol – dodecane – tridecane – water [97]	298.15	22	NRTL	2960.1	0.028
ethanol – octanol – nonanol – dodecane – tridecane – water [97]	303.15	18	NRTL	2884.6	0.029
ethanol – decanol – undecanol – tetradecane – pentadecane – water [98]	293.15	20	UNIQUAC	2245.7	0.021
ethanol – decanol – undecanol – tetradecane – pentadecane – water [98]	298.15	24	UNIQUAC	2224.7	0.020
ethanol – decanol – undecanol – tetradecane – pentadecane – water [98]	303.15	23	UNIQUAC	1438.2	0.020

C Appendix C

CI SynGameZero: Proof of Concept

CI.1 Environment

CI.1.1 Chemical System and Unit Operations

In the following, we provide detailed information on the environment for the proof of concept for SynGameZero. The environment consists of a steady-state flowsheet simulation initialized by a set of feed streams. In this example, the processes operate in a model system of four compounds A – B – C – D (in boiling order). The system is zeotropic and thus can be separated by distillation only. The following unit operations are available to the agent:

I) Reactor

If A and B are present in the feed stream of the reactor, the following reaction is observed:



The reactor is a continuous stirred tank reactor with the conversion of A given by a kinetic of first order in A and B:

$$\dot{n}_A^{\text{in}} - \dot{n}_A^{\text{out}} = 5 \frac{\text{kmol}}{\text{h}} x_A^{\text{out}} x_B^{\text{out}}. \quad (\text{C2})$$

Therein, 'in' and 'out' specify quantities at the reactor inlet and outlet. The variables \dot{n}_i and x_i denote component i 's molar flow rate and mole fraction, respectively. The conversion of the other components B, C, and D is calculated by the stoichiometry of Reaction C1. This unit operation is denoted as R.

II) Distillation column

As mentioned, the distillation columns are modeled using the ∞/∞ -approach [61–63]. As the system is zeotropic, it is possible to define perfectly sharp splits. Thus,

the agent has three discrete options denoted as D_1 (split A - BCD), D_2 (split AB - CD), and D_3 (split ABC - D).

III) Mixer

Place a mixer to mix two streams. This unit operation is denoted as M.

The actions D_1 , D_2 , D_3 , and R can be applied to any single open stream, whereas M requires two open streams as input.

CI.1.2 Cost Function

The NPV is used to evaluate the obtained processes. Since the degree of detail in its calculation is not relevant for the presented methodology and the process models are rather basic, a rather simple scheme is used to calculate the NPV:

$$\text{NPV} = - \sum_{u \in U} I_u + 10a \sum_{o \in O} c_o. \quad (\text{C3})$$

It combines the investment costs I_u of every unit u with the yearly operational cash flows c_o multiplied by ten years (a factor that lumps the depreciation period and interest rates). The investment costs of the units are assumed flat and independent of size and operation parameters. The yearly operational cash flows c_o consider only cost and revenues from all open material streams o leaving the process. Further operational costs of the units (e.g., the steam cost for the distillation) are neglected for simplicity. The cash flows of the open streams are calculated as follows. If a stream contains a pure component i :

$$c_o = \dot{n}_i \cdot p_i \cdot 8000 \frac{\text{h}}{\text{a}}. \quad (\text{C4})$$

where \dot{n}_i is its molar flowrate in kmol/h and p_i is the price of component i . If an open stream is not pure, then its yearly cash flow is:

$$c_o = \sum_{i \in \{A, B, C, D\}} \dot{n}_i \cdot \min(p_i, 0) \cdot 8000 \frac{\text{h}}{\text{a}}. \quad (\text{C5})$$

The minimum function ensures that the cash flow of mixed streams is never positive. If the stream contains a compound of negative price p_i (e.g., a hazardous compound for which disposal has to be paid), then the cash flow becomes negative. The values/costs of the feed stream(s) are not considered explicitly in the formulas, as they are constant for both players and, therefore, have no influence on finding the optimal process. However, the agent may select the trivial process of placing no process unit. In this case, any feed is an open stream leaving the process and is included in the determination of the NPV.

Table C1: I_u parameters for Case Study 1 and Case Study 2.

	$I_R \setminus \text{k€}$	$I_D \setminus \text{k€}$	$I_M \setminus \text{k€}$
Case Study 1	10000	10000	1000
Case Study 2	10000	10000	1000

Table C2: Component prices for Case Study 1 and Case Study 2.

	$p_A \setminus \text{€/kmol}$	$p_B \setminus \text{€/kmol}$	$p_C \setminus \text{€/kmol}$	$p_D \setminus \text{€/kmol}$
Case Study 1	1	1	1	1
Case Study 2	-0.125	-0.125	1	1

In this case study, two different choices for the parameters I_u (see Table C1) and p_i (see Table C2) are discussed to demonstrate the interchangeability of the cost function.

CI.2 Generation of the Flowsheet Matrix

The state of a flowsheet is stored in the flowsheet matrix \mathbf{F} . The construction of \mathbf{F} from the simulation results of the environment is explained in Figure C1. Every stream in the flowsheet refers to one row of \mathbf{F} . \mathbf{F} has a fixed number of rows N_{matrix} ; if there are less streams in the flowsheet, the remaining rows are filled with zeros. The number N_{matrix} is an upper limit for the size of the flowsheet (i.e., the number of streams). If the matrix is full, the process synthesis is terminated automatically by the environment. This limitation is due to technical reasons. In practice, N_{matrix} has to be chosen large enough to accommodate the optimal flowsheet comfortably. For this example, we set it to $N_{\text{matrix}} = 10$.

Every row of \mathbf{F} is composed of a set of vectors that are explained along the first row in Figure C1 for stream 1 (reactor input) of the shown flowsheet. \mathbf{v}_1 contains the molar fractions of all compounds followed by the total molar flow rate of stream 1. The vector \mathbf{u}_1 is a one hot encoding (OHE) that specifies the process unit at the streams destination. It has $N_{\text{unit}} = 4 + N_{\text{matrix}}$ entries. The first four entries refer to distillation splits D_1 , D_2 , D_3 and reactor R, respectively. The last N_{matrix} entries are relevant if the stream is connected to a mixer. The entry for the corresponding unit is set to 1, and all other entries are set to 0. In the case of the mixer, the $(4+k)$ th entry is set to one, indicating that the other stream to the mixer is stream k . For example, \mathbf{u}_1 in Figure C1 indicates

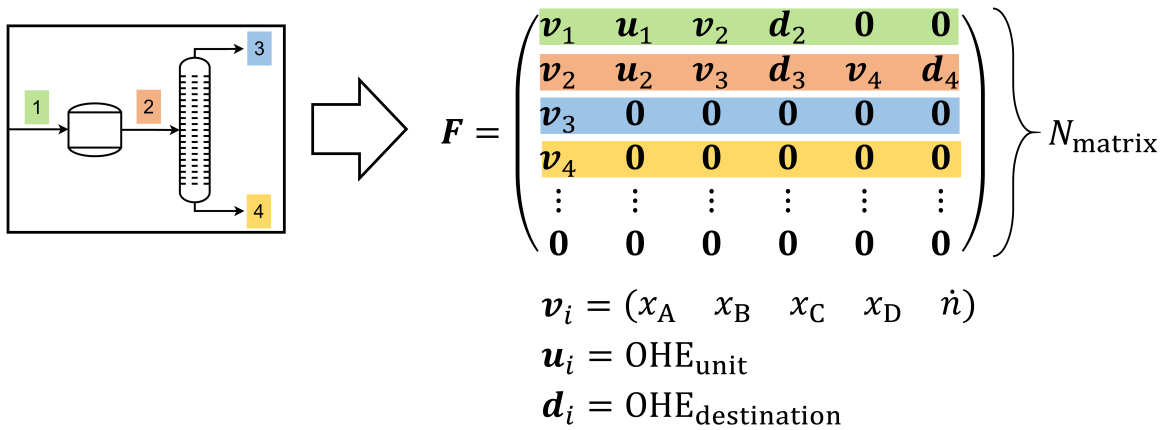


Figure C1: Construction of the flowsheet matrix \mathbf{F} in the SynGameZero approach. \mathbf{F} contains the information of the stream table combined with structural information on the flowsheet. All entries in the matrix, which are not needed for now, are set to 0 ($\mathbf{0}$ refers to a vector consisting of as many entries equal to 0 as required for the width of the respective column).

that stream 1's destination is a reactor R. If no unit is connected to a stream i , then all entries of \mathbf{u}_i are set to 0. The last four vectors of each row contain information on the subsequent streams that leave the process unit and the streams destination. Let us say these are streams m and n . The first and the third of the four vectors are copies of \mathbf{v}_m and \mathbf{v}_n , respectively. The second and fourth vectors are OHEs to the numbers m and n , respectively. They are vectors with N_{matrix} entries, all of them 0 but the m -th or n -th entries, respectively, which are 1. If there is no destination process unit (e.g., streams 3 or 4) or the process unit has only one output stream (e.g., stream 1), all four or the latter two vectors are filled with zeros, respectively.

CI.3 Implementation and Training Procedure

CI.3.1 Implementation

The framework for SynGameZero was implemented with Python. All parts of the framework that are related to ML were implemented using methods from the Python package `tensorflow` (version 1.9.0) [110].

The ACN is a multi-layer perceptron (MLP) [86] with N_{layer} fully connected hidden layers and an actor-critic output [37]. Every hidden layer has N_{node} nodes with ReLU activation. The policy head $\boldsymbol{\pi}$ consists of one entry for every possible action (as the flowsheet can theoretically consist of up to N_{matrix} open streams and one termination

Table C3: Numerical tuning parameters for Case Study 1 and Case Study 2. The parameter K specifies the depth of the tree search.

	N_{games}	N_{memory}	N_{batch}	N_{layer}	N_{node}	K
Case Study 1	5000	256	32	2	32	20
Case Study 2	20000	256	32	2	64	40

action is needed, there are $N_{\text{action}} = (N_{\text{matrix}} \cdot N_{\text{unit}}) + 1$ actions). A softmax activation ensures that the entries in $\boldsymbol{\pi}$ are in the range $[0, 1]$ and sum up to 1. A tanh activation ensures that the output of the value-head, v , is in the range $[-1, 1]$.

CI.3.2 Training Procedure

The agent plays N_{games} of games against itself during training. The compositions of the feed stream(s) are randomly varied to obtain an agent that can solve a broad class of problems. The search tree is initialized with the given feed(s) at the beginning of every game. Then, the agent plays the game until the end. Thereby, every decision that had been made in step IV) of the tree search is stored. Stored are the state \boldsymbol{s} at the root node, which served as input for the ANN, and the vector \boldsymbol{y} , which is based on the tree search results. After finishing the game, the data is augmented by the final reward, r , which indicates the winning player. The tuples of the form $(\boldsymbol{s}, \boldsymbol{y}, r)$ are stored in a memory of size N_{memory} . After every game, a batch of N_{batch} tuples is sampled randomly to perform two optimization steps. The first one with respect to the loss function l_1 and the second one with respect to the loss function l_2 :

$$l_1 = (v - r)^2, \quad (\text{C6})$$

$$l_2 = \sum_i (\pi_i - y_i)^2. \quad (\text{C7})$$

The ANN was trained using Adam [111] as optimizer with a constant learning rate of 10^{-4} . The gradients were clipped to a maximum value of 5 to prevent instabilities during training. The numerical tuning parameters are listed in Table C3.

D Appendix D

DI SynGameZero: Integration of Hierarchical Reinforcement Learning

DI.1 Environment

DI.1.1 Chemical System and Unit Operations

In the following, we provide detailed information on the environment for the ETBE synthesis process example from Section 3.2.3. The processes are set up in a steady-state flowsheet simulation within a quaternary system consisting of ethanol (Et), isobutene (IB), *n*-butane (nBut), and ethyl-tert-butyl-ether (ETBE). Starting from two feed streams (the first one containing pure Et, the second one a mixture of IB and nBut), the goal is to synthesize ETBE. Short-cut apparatus models for conceptual design in the spirit of Ryll et al. [62, 64] provide a robust and rapid simulation of the proposed processes. For this matter, some simplifying assumptions are made for the available unit operations (e.g., the temperatures of the streams are not taken into account for the simulation):

I) Reactor

In the reactor, the following reversible reaction takes place [62, 64]:



It is assumed that the reactor is operated at 50 °C and equilibrium is reached. Ideal liquid phase behavior is assumed, and the chemical equilibrium constant at 50 °C ($K_x = 111.1$) is adapted from [112]. This unit operation is denoted as R.

II) Distillation column

The distillation columns are modeled using ∞/∞ -analysis (assuming an infinite number of stages and total reflux) [61–64]. The columns are operated at 8 bar [62, 64]. Two binary minimum azeotropes (in the systems Et – ETBE at $x_{\text{Et}} = 0.63$ mol/mol and Et – nBut at $x_{\text{Et}} = 0.06$ mol/mol) occur in the quaternary system

[113]. The system is separated into two distillation regions as shown in Figure D1. For the sake of simplicity, it is assumed that the simplex spanned by the azeotropes and pure IB defines the distillation boundary. To separate mixtures by distillation, the agent can choose between two different split types: D_L refers to a column that separates the local low-boiler as distillate from the feed stream of the column with the highest possible yield. The split D_H separates the local high-boiler as bottom product with the highest possible yield. For further information regarding ∞/∞ -analysis, we refer to Appendix AI and [61–63].

III) Mixer

The agent can decide to mix two open streams. M denotes this unit operation.

IV) Recycle

The agent can recycle any open stream to any closed stream in the process. Rec denotes this unit operation.

In the hierarchically structured action space, the agent chooses an open stream (or the terminate action) at level 1, and one of the above-listed units (R, D_L , D_H , M, and Rec) at level 2. If M or Rec are chosen at level 2, the agent has to select a destination (an open stream for M and a closed stream for Rec, respectively) at level 3.

DI.1.2 Cost Function

The resulting flowsheets are evaluated using the NPV:

$$\text{NPV} = - \sum_{u \in U} I_u + 10a \left(- \sum_{u \in U} C_{\text{op},u} + \sum_{o \in O} c_o \right). \quad (\text{D2})$$

For every process unit u in the set U of all process units present in the flowsheet, the variables I_u and $C_{\text{op},u}$ refer to the total investment costs and annualized operating costs, respectively. For every open stream o out of the set O of all open streams in the flowsheet, c_o describes the annualized operational cashflow.

The investment costs are estimated from the feed mass flowrate of the units. For reactors and distillation columns, base values for total investment costs are adapted from Domingues et al. [114] for a mass flow of $\dot{m}_{0,u} = 9000$ kg/h (the base values are listed in Table D1). For a unit u , scaling to other mass flowrates is done using the power rule [115]:

$$I_u = I_{0,u} \cdot \left(\frac{\dot{m}_u}{\dot{m}_{0,u}} \right)^{0.6}. \quad (\text{D3})$$

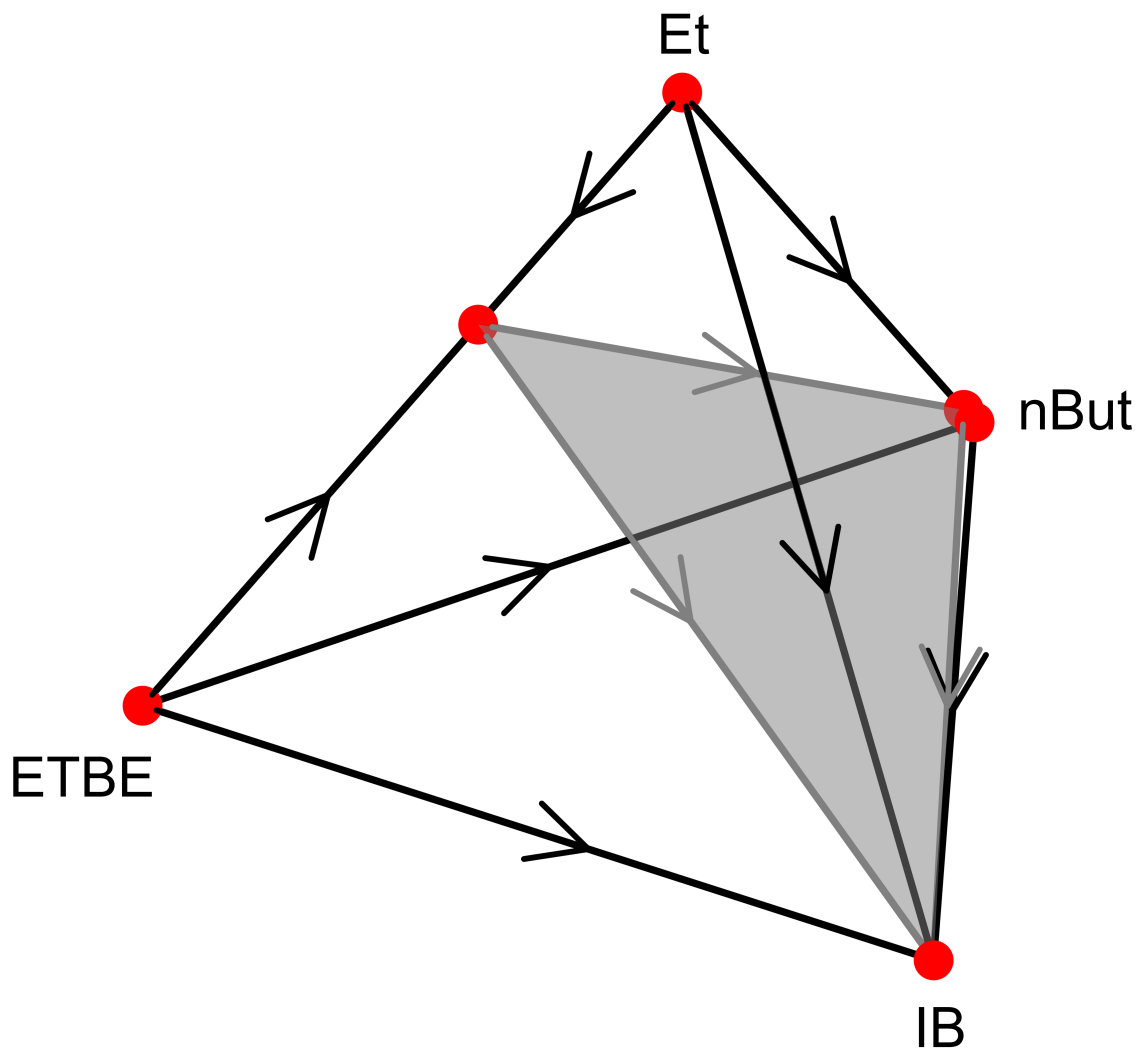


Figure D1: Topology of the VLE in the quaternary system Et – IB – nBut – ETBE at 8 bar. The binary azeotropes and IB span the distillation boundary (gray surface). The arrows indicate the direction of the distillation lines toward the low-boiler.

Table D1: Base values for investment costs of the units.

$I_{(0,R)} \setminus \text{k€}$	$I_{(0,D)} \setminus \text{k€}$	$I_{(0,M)} \setminus \text{k€}$	$I_{(0,Rec)} \setminus \text{k€}$
64	594	0	0

Table D2: Prices for components and steam.

$p_{\text{Et}} \setminus \text{€}/\text{kg}$	$p_{\text{IB}} \setminus \text{€}/\text{kg}$	$p_{\text{nBut}} \setminus \text{€}/\text{kg}$	$p_{\text{ETBE}} \setminus \text{€}/\text{kg}$	$p_{\text{steam}} \setminus \text{€}/\text{kg}$
0.75	0.5	0.5	1.27	0.04

Operating costs $C_{\text{op},u}$ comprise only steam costs for distillation columns. The reboiler duties of the distillation columns are estimated from the simple assumption that the distillate has to be evaporated twice:

$$\dot{Q}_{\text{Reboiler}} = 2 \cdot \sum_i \dot{m}_{i,\text{distillate}} \cdot \Delta h_{i,v}^{(m)}. \quad (\text{D4})$$

Therein, $\dot{m}_{i,\text{distillate}}$ and $\Delta h_{i,v}^{(m)}$ are the distillate’s mass flowrate and enthalpy of evaporation of component $i \in \{\text{Et}, \text{IB}, \text{nBut}, \text{ETBE}\}$, respectively. Assuming constant steam costs, $C_{\text{op},u}$ follows as:

$$C_{\text{op, distillation column}} = p_{\text{steam}} \cdot \frac{\dot{Q}_{\text{Reboiler}}}{\Delta h_{\text{water},v}^{(m)}} \cdot 8000 \frac{\text{h}}{\text{a}}. \quad (\text{D5})$$

Prices for steam and the components are listed in Table D2 and based on data found in the literature [114] (an equal price was assumed for IB and nBut). If $x_i^{(m)} > 0.99$ for any component i , c_o is calculated as:

$$c_o = p_i \cdot \dot{m}_i \cdot 8000 \frac{\text{h}}{\text{a}}. \quad (\text{D6})$$

Streams with less purity are discounted with a factor of 0.5):

$$c_o = 0.5 \cdot \sum_i p_i \cdot \dot{m}_i \cdot 8000 \frac{\text{h}}{\text{a}}. \quad (\text{D7})$$

DI.2 Generation of the Flowsheet Matrix

The flowsheet matrix is slightly modified compared to the original SynGameZero approach (see Appendix CI.2). The concept is explained alongside Figure D2. The flowsheet matrix \mathbf{F} has a fixed number of rows $N_{\text{matrix}} = 16$, as the agent can only process inputs with a fixed size. All entries in \mathbf{F} are initially set to zero and only altered according to changes upon flowsheet synthesis. Every row refers to a potential stream in the flowsheet, and therefore, N_{matrix} serves as an upper limit for the size of the flowsheet

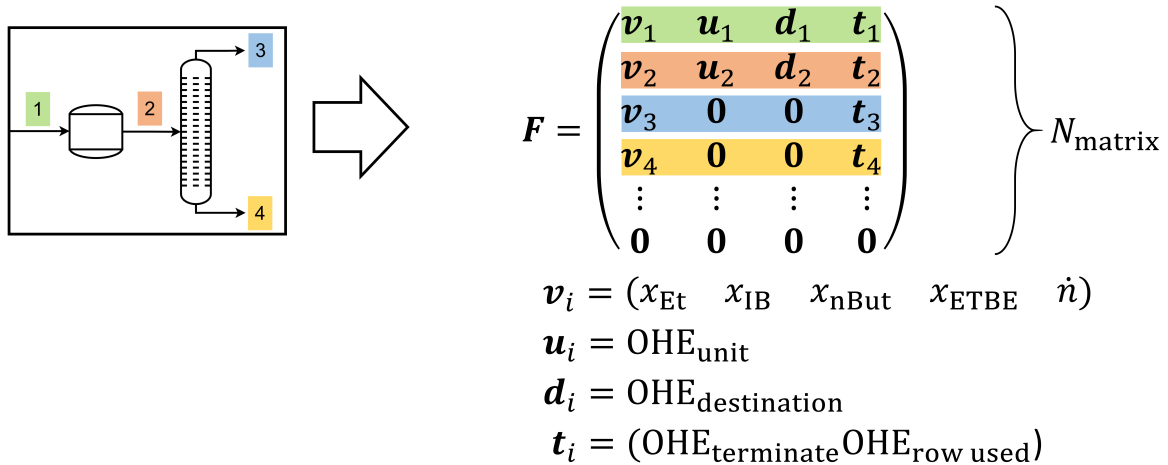


Figure D2: Construction of the flowsheet matrix \mathbf{F} in the hierarchical SynGameZero approach. The matrix contains a stream table and information on the connectivity of the streams in the flowsheet.

(the environment terminates the synthesis automatically if it is exceeded).

Every row of \mathbf{F} comprises a set of vectors. The vectors \mathbf{v}_i and \mathbf{u}_i work in the same way as explained in Appendix C1.2. The vector \mathbf{u}_i has $N_{\text{unit}} = 5$ entries (available unit operations: R, D_L, D_H, M, Rec). The vector \mathbf{d}_i is an OHE of the connectivity of the output stream(s) of the unit specified in \mathbf{u}_i . It has N_{matrix} entries, and if the output streams of the corresponding unit are stored in the n -th and m -th row of \mathbf{F} , the n -th and m -th entries of \mathbf{d}_i are set to 1, respectively (this is also done to store the connectivity of recycle streams). The vector \mathbf{t}_i has two entries, which are either equal to 0 or 1. The first entry is set to 1 if the flowsheet synthesis was terminated (marked in every row of \mathbf{F}). The second entry is set to 1 if row i refers to an actual stream in the flowsheet. This allows the distinction of streams that are part of a flowsheet structure but have no flowrate yet from unused rows in the matrix.

DI.3 Implementation and Training Procedure

DI.3.1 Implementation

The hierarchical framework and SynGameZero were implemented with Python (published via GitHub <https://github.com/grimmlab/SynGameZero> together with an implementation of the variant mentioned in Section 3.2.3). All parts of the framework that are related to ML were implemented using methods from the Python package `pytorch` (version 1.8.0) [116].

In the following, we explain the ANN architecture implementation, shown in Figure 11. The CNN consists of two layers of convolutional filters, which use ReLU activations. The state \mathbf{s} is provided in the 3-dimensional shape $(2, N_{\text{matrix}}, N_{\text{length row}})$ (note that $N_{\text{length row}} = 5 + N_{\text{unit}} + N_{\text{matrix}} + 2$). Since the information in the flowsheet matrices is stored in a heterogeneous way (flowrates to the left, connectivity to the right), the filters have to be applied over the rows as a whole. In the first layer, 16 filters (with a shape of $(1, N_{\text{length row}})$) are applied and create an output of the shape $(2, N_{\text{matrix}}, 16)$. The second layer applies eight filters (with a shape of $(1, 16)$) and generates an output of the shape $(2, N_{\text{matrix}}, 8)$. This output is flattened and used as an input vector for the ACN at level 1. All ACNs have the same structure: 2 fully connected shared layers with 48 nodes and ReLU activations. At every level, the policy-head uses a softmax activation and the value-head a tanh activation to generate $\boldsymbol{\pi}_i$ and v_i , respectively.

DI.3.2 Training Procedure

During training, the agent plays $N_{\text{games}} = 10000$ games against itself and stores the following results of the tree search: at each root node, which is passed during a game, tuples of the form $(\mathbf{s}, \mathbf{s}_2, \mathbf{s}_3, \mathbf{y}_i, r)$ are created. \mathbf{s} refers to the state at the root node, while \mathbf{s}_2 and \mathbf{s}_3 refer to the additional inputs for the ACNs at levels 2 and 3, respectively. Similarly, as in the original SynGameZero approach, the vector \mathbf{y}_i represents the distribution of the visit counts at the root node. The variable r refers to the reward, which is obtained at the end of the game. For every level i , a memory of size $N_{\text{memory}} = 640$ is created to store these tuples. Batches of size $N_{\text{batch}} = 64$ are sampled from the memories to perform two optimization steps (one for each of the loss functions below).

$$l_{i,1} = (v_i - r)^2, \quad (\text{D8})$$

$$l_{i,2} = \sum_j (\pi_{i,j} - y_{i,j})^2. \quad (\text{D9})$$

For every level i , the gradients are backpropagated starting at the respective ACN until the top of the CNN. The ANN was trained using Adam [111] as optimizer with a constant learning rate of 10^{-4} . After each game, these steps are performed for every level i that was at least used once during this game. For example, it might be possible that no mixers or recycles are placed during a game, and therefore no new tuples $(\mathbf{s}, \mathbf{s}_2, \mathbf{s}_3, \mathbf{y}_3, r)$ would be stored for level 3. The optimization steps for level 3 are suspended in this case to prevent overfitting.

E Appendix E

EI Single-Player Reinforcement Learning Framework for Automated Flowsheet Synthesis

EI.1 Environment

EI.1.1 Chemical Systems and Unit Operations

In the following E, we provide detailed information on the environment of the framework described in Section 3.2.4. In particular, we consider four chemical example systems from the literature [103–106], which are listed in Table E1. Some unit operations are simulated based on phase equilibria, which are modeled using the Non-Random-Two-Liquid (NRTL) g^E -model [93]. The NRTL parameters for all considered binary subsystems are taken from Aspen Plus [117]. Within all of the considered chemical systems, we assume constant conditions as described in Table E1 (pressure and temperature were chosen to be located inside the range of the NRTL parameters).

The agent interacts with a steady-state flowsheet simulation, which simulates the chosen unit operations sequentially. The unit operations are based on short-cut models, which allow a quick and robust evaluation. Besides, from recycle streams, all models were chosen to always converge. The following unit operations and specifications are available as actions to the agent:

1) **Distillation column**

When a distillation column is chosen at level 2, it has to be further specified at level 3b (with sublevel 3c) by setting the continuous ratio of distillate flowrate \dot{n}^D to feed flowrate \dot{n}^F :

$$\frac{\dot{n}^D}{\dot{n}^F} \in [0, 1]. \quad (\text{E1})$$

As in previous examples, this unit operation is modeled using a linearised representation of the VLE and ∞/∞ -analysis (see Appendix AI and [61–64]). This unit operation is denoted as D.

Table E1: Considered chemical example systems and the available solvents for the single-player framework. We use the following abbreviations for the components: acetone (Ac), benzene (Be), butanol (Bu), chloroform (Ch), ethanol (Et), pyridine (Py), tetrahydrofuran (Te), toluene (To), water (Wa). The flowsheet simulation is based on phase equilibria that are constructed assuming constant conditions (temperature and pressure for liquid-liquid equilibria, pressure for vapor-liquid equilibria and distillation boundaries).

	System 1	System 2	System 3	System 4
Feed stream components	Ac, Ch	Et, Wa	Bu, Wa	Py, Wa
Available solvents	Be, To	Be, To, Te	Ac, Be, To	To
Temperature	323.15 K	338.15 K	338.15 K	338.15 K
Pressure	1 bar	1 bar	1 bar	1 bar

II) Decanter

No further specification is required when a decanter is chosen at level 2, as constant temperature and pressure are assumed throughout the simulation. The decanter splits the feed stream according to the underlying liquid phase equilibrium, which is constructed by usage of the CEM (see Section 3.1.2 and [62, 82]). The considered chemical systems in this example (listed in Table E1) display liquid phase splits into two phases at most. As mentioned in Section 3.1, the decanter is simulated as split (with a split ratio of 0.5) if there is no liquid phase split for the given feed composition. This ensures a constant number of output streams for this unit operation within the flowsheet simulation. This unit operation is denoted as Dec.

III) Add solvent

Table E1 provides a list of available solvents for the considered chemical example systems. For the sake of simplicity, the total number of different components per flowsheet is limited to 3 in this example. Thus, the agent can choose this unit operation once per flowsheet synthesis process. The solvent is added to the open stream, chosen at level 1. At level 3b and 3c, one has to specify the continuous ratio of solvent flowrate \dot{n}^S to the flowrate of the open stream \dot{n}^F :

$$\frac{\dot{n}^S}{\dot{n}^F} \in [0, 10]. \quad (\text{E2})$$

In this example, the ratio is limited to 10, giving ample space to solve the considered problems. In general, however, larger ratios can be allowed. This unit

operation is denoted as Add.

IV) Mixer

In this case, the open stream chosen at level 1 is mixed with another open stream, which will be specified at level 3a. Therefore, the specification is a discrete decision. This unit operation is denoted as M.

V) Recycle stream

In this case, the open stream chosen at level 1 is recycled back. Similarly to 'Mixer', the (discrete) destination is specified at level 3a. This unit operation is denoted as Rec.

EI.1.2 Cost Functions

NPV

The NPV is calculated similarly as in Appendix DI.1:

$$\text{NPV} = - \sum_{u \in U} I_u + 10a \left(- \sum_{u \in U} C_{\text{op},u} + \sum_{o \in O} c_o \right). \quad (\text{E3})$$

U is the set of all unit operations used in the process. For every unit operation $u \in U$, I_u describes the total investment costs and $C_{\text{op},u}$ the annual operating costs, respectively. For every stream $o \in O$, which leaves the process, c_o describes the operational yearly cash flow. Herein, O is the set of all streams which leave the process.

The investment costs are scaled depending on the feed mass flowrate of the unit operations according to the power rule [115]:

$$I_u = I_{0,u} \cdot \left(\frac{\dot{m}_u}{\dot{m}_{0,u}} \right)^{0.6}. \quad (\text{E4})$$

The base values for the investment costs of the unit operations are taken from [106] (for a mass flowrate $\dot{m}_0 = 25000 \frac{\text{kg}}{\text{h}}$). All parameters and prices for the cost function NPV are provided within Table E2.

The operating costs comprise only steam costs for distillation columns and costs for added solvents. Steam costs are calculated as follows:

$$C_{\text{op, distillation column}} = p_{\text{steam}} \cdot \frac{\dot{Q}_{\text{Reboiler}}}{\Delta h_{\text{water},v}^{(m)}} \cdot 8000 \frac{\text{h}}{\text{a}}. \quad (\text{E5})$$

Table E2: Parameters for the cost function NPV. The base values for the investment costs of the unit operations are taken from [106] (for a mass flowrate $\dot{m}_0 = 25000 \frac{\text{kg}}{\text{h}}$). $I_{0,D}$ refers to the base value for the distillation column. $I_{0,Dec}$ refers to the base value of the decanter. Investment costs for all unit operations that are not listed are neglected. The prices for steam and components are chosen similarly as in Appendix DI.1.

$I_{0,D} \setminus \text{k€}$	$I_{0,Dec} \setminus \text{k€}$	$p_{\text{steam}} \setminus \text{€/kg}$	$p_{\text{feed component}} \setminus \text{€/kg}$	$p_{\text{solvent}} \setminus \text{€/kg}$
1000	200	0.04	0.5	0.05

Therein, $\Delta h_{\text{water},v}^{(m)}$ is the enthalpy of evaporation of water. The reboiler duty $\dot{Q}_{\text{Reboiler}}$ of a distillation column is estimated from the simple assumption that the distillate has to be evaporated twice:

$$\dot{Q}_{\text{Reboiler}} = 2 \cdot \sum_i \dot{m}_{i,\text{distillate}} \cdot \Delta h_{i,v}^{(m)}. \quad (\text{E6})$$

Therein, $\dot{m}_{i,\text{distillate}}$ and $\Delta h_{i,v}^{(m)}$ are the distillate's mass flowrate and enthalpy of evaporation of component i . For an arbitrary component i , $\Delta h_{i,v}^{(m)}$ is estimated, by calculating the energy, which is required to heat component i at ambient conditions (298.15 K, 1 bar), in liquid form, to its boiling point and adding the heat of evaporation. Heat capacities were taken from [118, 119]. The heat of evaporation was computed using the Antoine equation (parameters provided within [117]) and the Clausius-Clapeyron equation [120].

When a solvent is added, one has to pay for the amount of solvent:

$$C_{\text{op, add solvent}} = p_{\text{solvent}} \cdot \dot{m}_{\text{solvent}} \cdot 8000 \frac{\text{h}}{\text{a}}. \quad (\text{E7})$$

Consider a stream $o \in O$, which leaves the process. A positive value for c_o is assigned if it is (almost) a pure stream. Therefore, if the mass fraction for some component i is greater than 0.99, we consider o to be pure and calculate c_o as:

$$c_o = p_i \cdot \dot{m}_o \cdot 8000 \frac{\text{h}}{\text{a}}. \quad (\text{E8})$$

Therein, p_i is the price of the respective component and \dot{m}_o the mass flow rate of stream o . If the defined specification (some component i with a mass fraction greater than 0.99) is not fulfilled, c_o is set to 0. Note that the definition of c_o implies that one can add a solvent for free if it is entirely separated after using it for separation purposes.

Generic Cost Function (GCF)

GCF is a generic cost function particularly useful for conceptual flowsheet synthesis. It assigns a score (without a monetary unit) to a given flowsheet according to the primary goal of separating the feed stream into pure components. Secondary objectives are, for example, to use as few unit operations as possible and to obtain the added solvent as pure stream after usage. We use a different notation to prevent any ambiguity between NPV and GCF.

$$\text{GCF} = - \sum_{u \in U} C_u - C_{\text{added solvent}} + \sum_{o \in O} g_o. \quad (\text{E9})$$

U is the set of all unit operations used in the process. For every unit operation $u \in U$, C_u describes the total costs. We assign constant costs for every unit operation. Contrary to NPV, we omit scaling of unit costs with size and steam costs for distillation columns. $C_{\text{added solvent}}$ refers to the costs for added solvents. For every stream $o \in O$, which leaves the process, g_o describes the gain from that stream. Herein, O is the set of all streams which leave the process.

When a solvent is added, one has to pay for the amount of solvent:

$$C_{\text{added solvent}} = w_{\text{solvent}} \cdot \dot{n}_{\text{total added solvent}}. \quad (\text{E10})$$

Similarly, as before, $C_{\text{added solvent}}$ can be neglected by separating all used solvent as pure stream.

For a product stream $o \in O$, its gain g_o is set to a positive value if it is a pure stream. The specification for a pure stream for GCF is a molar fraction greater than 0.99 for an arbitrary component i . If component i is a solvent, g_o is calculated as:

$$g_o = w_{\text{solvent}} \cdot \dot{m}_o. \quad (\text{E11})$$

If i is a feed component, we define:

$$g_o = w_{\text{feed component}} \cdot \dot{m}_o. \quad (\text{E12})$$

All parameters for GCF are provided in Table E3.

EI.2 Generation of the Flowsheet Matrix

The flowsheet matrix is generated in a slightly modified way compared to the SynGameZero approach. The concept is explained alongside Figure E1. Every stream

Table E3: Parameters for GCF (as GCF has no unit, the parameters also have no units).

C_D	C_{Dec}	C_{Add}	C_M	C_{Rec}	$w_{\text{feed component}}$	w_{solvent}
10	5	0.5	0.5	0.5	1000	100

corresponds to a row in the flowsheet matrix and is structured similarly: the vector \mathbf{v}_i contains molar fractions, molar flowrate, mass flowrate, and the vector \mathbf{y} . \mathbf{y} contains critical temperature, critical pressure, and the acentric factor for every component, which is present in the flowsheet. Additionally, it contains the activity coefficients at infinite dilution for each binary subsystem, which is present in the flowsheet. The vector \mathbf{u}_i consists of an OHE for the connected unit, a variable for the continuous specification of this unit, and a binary variable, which marks if this unit requires a continuous specification. The vector \mathbf{d}_i is an OHE, which describes the connectivity of the stream(s) that leave the unit connected in row i . The vector \mathbf{t}_i consists of three binary variables, which mark if stream i is a feed stream, if the flowsheet synthesis is terminated, and if this row is used. Additionally, one could add the temperature and pressure of the streams in the respective rows. But as those are chosen to be constant within the example systems, this is omitted for now.

In this example, a flowsheet’s maximum number of units is set to 10. This means that if the agent does not decide to terminate earlier, the flowsheet synthesis finishes automatically after placement of 10 unit operations (this limits the size of the flowsheet matrix automatically to $N_{\text{matrix}} = 21$).

EI.3 Implementation and Training Procedure

EI.3.1 General Implementation

The implementation, used to generate all results shown in Section 4.4, is published via GitHub <https://github.com/grimmlab/drl4procsyn>, where we refer to for the details. The agent was trained with an AMD EPYC 7543 32-core processor and two NVIDIA RTX A5000, each with 24GB of memory. Training from zero knowledge for $N_{\text{games}} = 50000$ episodes takes about one day.

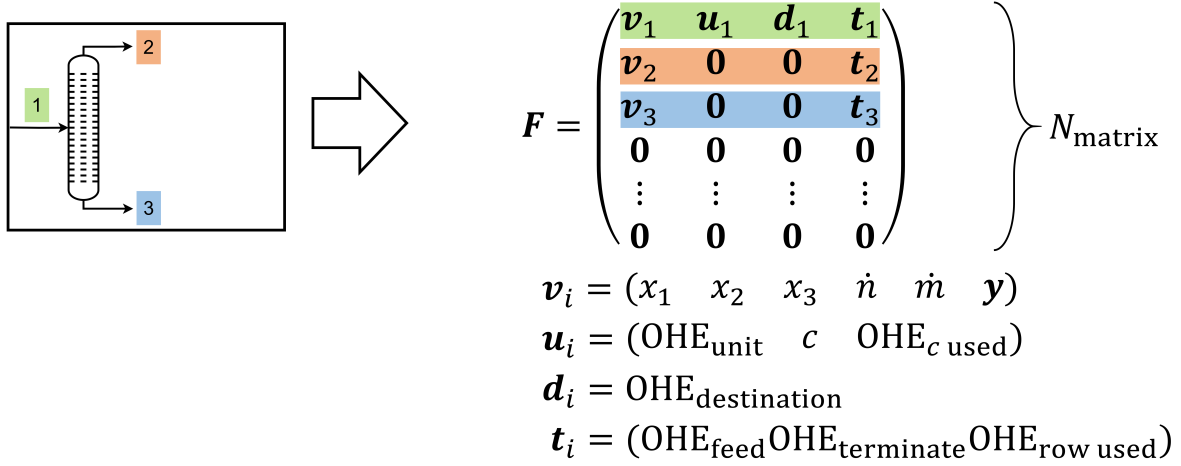


Figure E1: Construction of the flowsheet matrix F in the single-player framework. The matrix contains a stream table and information on the connectivity of the streams in the flowsheet.

EI.3.2 Artificial Neural Network Implementation

The neural network takes as input a state and outputs logits for all action hierarchy levels and the predicted value simultaneously. It consists of two identically structured MLP-Mixer [87] networks with separate weights (one for the policies, one for the value) for representing the streams, followed by a total of four separate policy-heads for each level and a single value-head.

The following describes the input and how it is passed through the network. In this example, the latent space \mathbb{R}^d is of dimension $d = 128$ everywhere. Let $F \in \mathbb{R}^{N_{\text{matrix}} \times n}$ be a flowsheet matrix with rows $\mathbf{w}_1, \dots, \mathbf{w}_{N_{\text{matrix}}}$ (by abuse of notation, let n be the length of \mathbf{w}_i). We embed each stream $\mathbf{w}_1, \dots, \mathbf{w}_{N_{\text{matrix}}}$ into \mathbb{R}^d with two learnable affine maps $H^v, H^\pi: \mathbb{R}^n \rightarrow \mathbb{R}^d$.

The two MLP-Mixer [87] networks MLP-Mixer^v and MLP-Mixer^π have identical architectures but different network parameters. The affinely embedded sequence is passed to the MLP-Mixer networks, transforming the sequence into latent streams:

$$(\tilde{\mathbf{w}}_i^v)_{i \in \{1, \dots, N_{\text{matrix}}\}} := \text{MLP-Mixer}^v \left((H^v(\mathbf{w}_i))_{i \in \{1, \dots, N_{\text{matrix}}\}} \right) \in \mathbb{R}^{N_{\text{matrix}} \times d} \quad (\text{E13})$$

and analogously

$$(\tilde{\mathbf{w}}_i^\pi)_{i \in \{1, \dots, N_{\text{matrix}}\}} := \text{MLP-Mixer}^\pi \left((H^\pi(\mathbf{w}_i))_{i \in \{1, \dots, N_{\text{matrix}}\}} \right) \in \mathbb{R}^{N_{\text{matrix}} \times d}. \quad (\text{E14})$$

Both MLP-Mixers consist of 5 mixer blocks whose design follows the original architecture [87]. In each mixer block, we use layer normalization, a hidden dimension of 512 in the

feature mixing MLP, and a hidden dimension of $2 \cdot N_{\text{matrix}}$ in the token mixing MLP.

The latent streams are shared with the policy-heads and the value-head. Each head combines the latent streams to a final vector and enriches it with information from previous hierarchy levels (and the current objective obtained if terminating the synthesis for the value head). The resulting vector is passed to a final MLP predicting the probability distribution over the hierarchical level (in the case of the policy head) or outputting a scalar value estimation.

EI.3.3 Sampling of Problem Instances during Training

During training and evaluation, we set the molar flowrate of the feed stream for all instances to a constant value of $1 \frac{\text{Mmol}}{\text{h}}$ (this value is chosen similarly to the processes examined in [103–106]). However, we note that the mass flowrate varies from instance to instance. To generate a (random) problem instance, we sample the chemical system from $\{1, 2, 3, 4\}$ with a uniform distribution. To obtain the feed composition, we sample the molar fraction of the first component x_1 with a uniform distribution from $(0, 1)$. The molar fraction x_2 of the second component reduces to $x_2 = 1 - x_1$.

EI.3.4 Training Procedure

We train the agent for $N_{\text{games}} = 50000$ episodes, where 50 parallel actor processes generate trajectories with frozen network parameters. After each episode, the actor sends the trajectory of states, actions, final reward, and improved policies at all states to a replay buffer process. We normalize and clip NPV and GCF by the profit obtained in a perfect separation without any costs, such that the observed reward r lies in $[0, 1]$ (where 1 is an unattainable upper bound).

For each action hierarchy level, a separate network training process samples uniformly random batches of states of size $N_{\text{batch}} = 128$ with replacement from the replay buffer and performs an optimizer step with respect to the value and policy on the given level. It ignores a hierarchy level if the number of trajectories containing at least one action from that level does not exceed a predefined threshold. This avoids overfitting for later levels at the beginning of training. We experienced that a threshold of 50 is generally enough. One optimizer step for all hierarchy levels (including skipped ones) constitutes one training step. All 100 training steps, the updated network parameters are distributed to the actor process to refresh their frozen network copies. Furthermore, after every 7500 steps, the performance of the current agent is evaluated on a pre-generated random but

fixed validation set of 200 initial states. The best-performing agent on the validation set is eventually used for testing after training. The training process uses the improved policy $\hat{\pi}_i$ to update the network’s policy π_i using a Kullback-Leibler divergence [86] loss $\text{KL}(\hat{\pi}_i||\pi_i)$, and the value prediction v is updated via the squared error $(v - r)^2$. The strong advantage of using the improved policy $\hat{\pi}_i$ as a training target for the network via Kullback-Leibler divergence is that $\hat{\pi}_i$ incorporates rich information from the search, as opposed to classically training the network to predict the single action which the agent takes in the environment. We leverage this effect and mask infeasible actions everywhere they are encountered in the tree by setting their corresponding logit to $-\infty$ before computing $\hat{\pi}_i$. Through this, the network learns to reduce the predicted probability for infeasible actions, better capturing the system’s dynamics. We use the Adam optimizer [111], and losses and optimizer steps are computed batchwise. The training process periodically disseminates the updated network parameters back to the actor processes to use for further episodes.

Student theses

The following student thesis was prepared under the supervision of the author of the present doctoral thesis in the frame of his research:

- A. Guellouz: Implementation of a flowsheet simulator for a process for ETBE synthesis. Bachelor thesis, Laboratory of Chemical Process Engineering (CTV), Technical University of Munich (2021).