Representation Learning for flowsheets: generating structures for Process Synthesis

Antonio Rocha Azevedoa,b, Tahar Nabilc, Valentin Loubièreb, Benoît Valentinc, Romain Privata, Thibaut Neveuxb, Jean-Marc Commengea,\*

aUniversité de Lorraine, CNRS, LRGP, F-54000 Nancy, France

bEDF R&D Chatou, 6 quai Watier, 78400 Chatou, France

cEDF R&D, Boulevard Gaspard Monge, F-91120 Palaiseau, France

jean-marc.commenge@univ-lorraine.fr

Abstract

The research for the optimal flowsheet is a key part of process synthesis. However, optimization at a structural level is difficult due to its discrete nature, and current approaches may not guarantee a good exploration of possibilities. A continuous structural domain could be useful for systematizing this research and exploration.

In this work, a machine learning model is trained to develop a continuous representation for flowsheets: every process can be turned into a point in space, and points in space can be translated back into processes. The model’s capability of generating new processes from random points in space is assessed. Results show that, although capable of grouping similar processes together, in clusters, it may not always propose structurally feasible flowsheets. Dataset generation and hyperparameter finetuning could also be improved before the model is coupled with process synthesis approaches.

**Keywords**: Process Synthesis, Machine Learning, Artificial Intelligence, SFILES

* 1. Introduction

In process synthesis, finding the optimal flowsheet for a given application is challenging, and it may be difficult to explore the realm of possibilities. While heuristics can be used for proposing relevant alternatives, they may miss better, less intuitive ones. Superstructural (see Mencarelli et al. (2020)) and generative approaches (see Nabil et al. (2022)) search a wider range of processes, but may not ensure a systematic exploration of the domain of feasible structures – which is difficult to represent mathematically. In contrast, mapping the discrete space of process flowsheets to a continuous one could prove useful, since optimization exploration techniques could be applied.

The goal of this work is to develop a model that learns a continuous, vector-space representation of process flowsheets, from which new ones can then be sampled. In this space, it is desired that two similar processes be found close to each other, while two very different flowsheets should be further apart. The motivation behind this objective is to eventually use this model to systematize the research of the optimal flowsheet – since a continuous space would yield a better way to analyse and quantify the exploration of alternatives, and enable the use of sampling and optimization strategies for proposing new ones.

This work takes inspiration from the paper by Gómez-Bombarelli et al. (2018), in which a Variational Autoencoder was used to develop a continuous representation for molecular structures – being used for running property optimization directly in the continuous space. Here, their publicly available codes were adapted for flowsheets, with a few additions for improving model learning efficiency.

* 1. Representation Learning Model
		1. Representation Learning and the Variational Autoencoder

Representation Learning is a branch of Machine Learning that attempts to automatically identify patterns and features in raw data – and develop efficient representations for it.

The VariationalAutoencoder (VAE) is a deep learning model composed of two separate parts: an encoder and a decoder. The encoder is a dimensionality reduction model, that transforms an input (in our case a flowsheet) into a representation vector (also called a “latent” vector); while the decoder tries to reconstruct the original input from said vector. While each of these parts are used separately, for different purposes, they are trained as a single model: the VAE receives an input and tries to reconstruct it, passing through a latent space of smaller dimension (which forces the model to learn an efficient representation). The model relies solely on the similarities between flowsheets to learn.

The “variational” label, put simply, entails that noise is added to the encoded vector during training to improve learning, and that the loss (i.e., the objective function used for training the model) is composed of a reconstruction term and a regularization term. The former ensures that the model is capable of translating points from the latent space back into the correct processes; the latter forces the data to be close together in the latent space, improving the continuity of the representation. In practice, the VAE is probabilistic in nature, a property that yields attractive generative capabilities.

* + 1. Input: SFILES notation

To train a Neural Network (NN) with process structures, it is necessary to convert them into a format that can be understood by the model (a matrix). To bridge this gap, the SFILES 2.0 notation (Vogel et al., 2023a) for representing process structures is used. It mirrors the SMILES standard used for molecules (used by the molecular VAE authors) and represents flowsheets as a string of “words” (or, more generally, “tokens”) making the task analogous to a Natural Language Processing problem – which is well-studied in the Machine Learning field. A token may represent an equipment or special symbols (such as recycles and branches). When all tokens relevant to the problem are listed, each one can be associated to a (“one-hot”) vector. The string of tokens thus becomes a sequence of vectors, which makes up the model’s actual input.

* + 1. Output: decoding strategy

Because of the discrete nature of the problem, the flowsheet is decoded one word at a time. At each step, the model outputs a probability for each word and picks that step’s word randomly, according to those probabilities. Then, this token is used as an input for decoding the next one, and so on. While other probabilistic decoding strategies are possible (see Vogel et al. (2023b)), this step-by-step sampling was kept for simplicity. This means that the model’s performance is probabilistic. Because of this, when evaluating the model, encoded vectors will be run multiple times through the decoder, and results will be presented according to this number of “decoding attempts”. For comparison, a “greedy”, deterministic, strategy will also be used: in it, the token with the highest probability is picked at each step. An illustration of the complete VAE is shown in Figure 1.



Figure 1: Illustration of a VAE and example of a process represented as a SFILES string.

* 1. Case Study: Thermodynamic Cycles
		1. Methodology

To train the model, a dataset of around 300,000 randomly generated thermodynamic cycles is used with a 90 %/10 % training/validation split. These processes are composed of 7 different types of equipment: compressors, turbines, heaters, coolers, integrated heat exchangers, stream splitters and mixers. All flowsheets are *structurally* feasible – they are closed cycles and equipment connectivity is respected – but they are not necessarily well-performing or logical from a thermodynamical standpoint. This dataset stems from the works by Nabil et al. (2023) and was kindly provided by the authors. An extra dataset (hereinafter “test” dataset) with 100,000 cycles was created by one of the generative AI models mentioned in the referenced paper and is used for further analysis.

The VAE’s *process* reconstruction accuracy (i.e., the fraction of correctly reconstructed processes in a set) for each dataset is used for verifying if the model was able of learning and generalizing. Note that the reconstruction of a given process is considered as a failure as long as at least one token is incorrectly decoded in the sequence. Token-wise accuracies (the fraction of correctly decoded tokens in the set) are also presented for discussion. To analyse the model’s capacity of generating new flowsheets, points will be randomly sampled from the latent space and decoded. The fraction of valid (structurally feasible SFILES), unique, and new (not present in the training dataset) flowsheets will be used for quantifying performance.

Since the model is given multiple decoding attempts, it is necessary to define which decoded flowsheet will be assigned as final. Here, the methodology used by the molecular VAE’s authors is adapted, for taking feasibility into account. After a point is decoded multiple times, each process is screened for feasibility. Then, each one is re-encoded into latent space, using the encoder part of the VAE (without adding noise to the result). Finally, the valid process which is the closest to the sampling point is chosen as the “assigned” decoding. This methodology is illustrated in Figure 2.



Figure 2: Illustration of the structure assignment methodology for a point in latent space (2D in this example). The final decoding is the valid process which is closest to the point.

* + 1. Results
			1. Reconstruction accuracies

Models with different numbers of latent space dimensions were trained for a total of 40 epochs. They will be referenced as a function of their dimensionality (e.g., model 100D has a 100-dimensional latent space). Because GPU acceleration was not possible, training times were a limiting factor, and most hyperparameter values were kept as default (e.g., the dimension size of 196 was the default). As mentioned in section 2.1, the model is trained with a regularization term. Following the molecular VAE’s authors’ recommendation, the weight of this term is increased during training, following a sigmoid curve. Here, its value is varied from 0.001 at epoch 21 to 0.999 at epoch 31 (steeper than default values, because of the long training times). Table 1 shows results for the best two models, for the epoch (training iteration) with minimal validation loss.

Table 1: Reconstruction accuracies for the best two models.

|  |  |  |
| --- | --- | --- |
|  |  | **Reconstruction accuracy** |
| **Model** | **Decoding type** | **Training** | **Validation** | **Test** |
| 100D | Greedy (token-wise) | 98.5 % | 98.3 % | 87.8 % |
|  | Greedy (process-wise) | 85.7 % | 84.3 % | 45.4 % |
|  | 10 attempts (process-wise) | 87.2 % | 86.3 % | 45.8 % |
| 196D | Greedy (token-wise) | 98.3 % | 98.2 % | 87.0 % |
|  | Greedy (process-wise) | 80.5 % | 79.5 % | 42.0 % |
|  | 10 attempts (process-wise) | 86.3 % | 85.6 % | 45.0 % |

As expected, the models achieve better accuracies with more attempts (since greedy decoding is deterministic, it represents one decoding). Since process reconstruction fails if at least one token is incorrectly decoded, very high token-wise accuracies are needed to achieve good process-wise accuracies. It is observed that the 100D model achieves slightly better reconstruction accuracies than the 196D. However, the former peaks at epoch 23, while the latter peaks at epoch 25, when the regularization weight is higher (though not yet 1). Model 196D is thus expected to have better latent space properties. Because of the steep variation of the regularization weight, a harsh deterioration of validation accuracy and loss were observed after the best epoch, from which the models did not recover by the end of training. For smaller dimensions, the drop in accuracy is greater and the recovery is much worse – which is shown in Table 2, where validation *process* reconstruction accuracies (10 decoding attempts) are presented for both the best and last epochs. Although the 196D model should have better latent space properties by the last epoch (40), which has a smaller regularization loss, it proved much worse for generating *feasible* flowsheets from the latent space. For this reason, only the best epoch models were kept for the next analyses.

Table 2: Evolution of validation process reconstruction accuracy (10 decoding attempts) according to latent space dimension and epoch.

|  |  |  |
| --- | --- | --- |
|  |  | **Reconstruction Accuracy** |
| **Model** | **Best epoch number** | **Best epoch** | **Last epoch** |
| 20D | 22 | 70.7 % | 0.00 % |
| 50D | 23 | 83.0 % | 0.25 % |
| 100D | 23 | 86.3 % | 0.31 % |
| 150D | 22 | 83.2 % | 13.1 % |
| 196D | 25 | 85.6 % | 77.5 % |

Finally, it is observed that test accuracies are much worse than training and validation’s, which was expected since the test set was generated in a different way (likely containing flowsheets that do not resemble the ones in the training set). This indicates that the model has trouble to generalizing beyond its training distribution.

* + - 1. Latent space analysis

Ten thousand points were sampled from the latent space and decoded according to the methodology described in section 3.1. Table 3 presents the proportion of valid; valid *and* unique; and valid, unique *and* new processes decoded from the sampling. Note that these values consider only the 10,000 *assigned* decodings, and not the total number of decoded SFILES (10,000 times the number of decoding attempts).

Table 3: Sampling results for the best models.

|  |  |  |
| --- | --- | --- |
|  |  | **Process Percentages** |
| **Model** | **Decoding attempts** | **Valid** | **Unique** | **New** |
| 100D | Greedy | 17.7 % | 17.7 % | 17.0 % |
|  | 10 attempts | 41.5 % | 41.5 % | 40.5 % |
|  | 100 attempts | 72.4 % | 72.3 % | 71.2 % |
| 196D | Greedy | 21.9 % | 21.8 % | 20.9 % |
|  | 10 attempts | 52.3 % | 51.8 % | 50.4 % |
|  | 100 attempts | 84.6 % | 83.2 % | 81.5 % |

It is observed that the 196D model has better sampling results, as expected from its better latent space properties and despite 100D’s slightly better reconstruction accuracies. It is observed that the models are not always capable of decoding valid SFILES from arbitrary points in space: processes will often include equipment with more (or less) connections than they should, or not be cyclical (necessary in this case study’s application). The proportion of valid decodings can be improved by increasing the number of decoding attempts, but this only means that *at least one* feasible process was generated, out of all attempts. This indicates that the model may struggle to find other similar and feasible processes around the same point. When analyzing the decoded processes in latent space, it is seen that the decoded SFILES are found in clusters, rather than being evenly spaced, as illustrated in Figure 2. An example of processes decoded from a same point is shown in Figure 3.



Figure 3: Example of processes sampled from a same point in latent space.

The choice of picking the closest valid process from the sample coordinates as the assigned decoding can be the subject of discussion, especially since the encoder and the decoder are used separately. When running multiple decoding attempts, a natural option would be to pick the SFILES decoded the most often, which should relate to the decoder’s confidence – the distance serving as a tiebreaker. Also, depending on the intended use, multiple – if not all – valid decoded processes can be taken into consideration.

* + 1. Limitations

Language-based notations may allow multiple ways of representing the same flowsheets. Even if a canonical representation is defined, it is hard to guarantee that similar processes will always have similar token-by-token representations. This may cause the model to have two representations for the same process in different parts of the latent space.

The use of directed graphs would be more adequate, but graph generative models are far more complex than language-based ones. A possible middle ground for this application could be to use a graph representation as an input for the encoder, while keeping the SFILES notation for the decoder’s output.

* 1. Conclusions

In this work, the development of a continuous representation for process flowsheets is studied. It is observed that the model can generate new, valid flowsheets from the representation space, and that similar structures can be found next to each other. However, performance can still be improved: in some cases, only one feasible process was decoded in 100 attempts, while it would be desired that multiple feasible, similar processes could be sampled from a same region.

Training data generation must also be improved. Since the generated processes do not need to be simulated, data generation is inexpensive. However, the time needed to train the model was not. Adapting the model for GPU acceleration would allow the use of more data and hyperparameter optimization.

Future works could seek to improve model performance and to use its generation capabilities to feed generative approaches, to allow a more efficient exploration of alternatives. Following the molecular VAE paper, the model could also be coupled with performance data, and running optimizations directly in latent space could be tested.

References

R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, A. Aspuru-Guzik, 2018, Automatic Chemical Design Using a Data-Driven Continuous Representation of Molecules, ACS Central Science, 4, 2, 268–276

L. Mencarelli, Q. Chen, A. Pagot, I. E. Grossmann, 2020, A review on superstructure optimization approaches in process system engineering, Computers & Chemical Engineering, 136, 106808

T. Nabil, J.-M. Commenge, T. Neveux, 2022, Generative Approaches for the Synthesis of Process Structures, In Y. Yamashita & M. Kano (Eds.), Computer Aided Chemical Engineering, 49, 289–294, Elsevier

T. Nabil, M. Noaman, T. Morosuk, 2023, Data-driven structural synthesis of supercritical CO2 power cycles, Frontiers in Chemical Engineering, 5

G. Vogel, E. Hirtreiter, L. S. Balhorn, A. M. Schweidtmann, 2023a, SFILES 2.0: an extended text-based flowsheet representation, Optimization and Engineering, 24, 4, 2911–2933

G. Vogel, L. S. Balhorn, A. M. Schweidtmann, 2023b, Learning from flowsheets: A generative transformer model for autocompletion of flowsheets. Computers & Chemical Engineering, 171, 108162