博士論文

Molecular Structure Generation Using Small Chemical Dataset

(小規模の化学データセットを利用した分子構造生成)

井上 貴央
(Takahiro INOUE)

To my parents.

Abstract

The structure generator, which generates molecular structures by computer, can propose structures satisfying the desired properties by using statistical models and datasets of the properties of interest. Therefore, structure generators are expected to streamline the process of compound design. However, due to experimental costs and other reasons, the available datasets are often small, and in such situations, the statistical model may overfit and fail to generate structures with good performance.

In this thesis, we develop a method that can generate structures with good performance when a statistical model-based structure generator is trained on a small chemical dataset consisting of about 1000 compounds with the desired properties. We proposed two methods to avoid overfitting: one is to use DAECS, a structure generator based on a statistical model without deep learning, and the other is to use data augmentation in a graph-based deep learning model to enhance the effect of transfer learning.

The structure generator DAECS tended to have low diversity of the generated structures. For this reason, we added structural modification rules that reduce the need for multiple applications of structural modification rules to a single application, and designed a seed structure selection algorithm so that the seed structures that undergo structural changes become diverse. Through the case study, it was confirmed that these methods could actually diversify the generated structures and generate new structures that are not included in the training dataset.

We also designed a method for JT-VAE, a deep structure generator, to enhance the effect of transfer learning, which uses a large molecular structure dataset along with a small chemical dataset to prevent overfitting. Specifi-

cally, we designed a data augmentation that perturbates the feature vectors for some vertices with standard normal random numbers during message passing in GNN. It was confirmed that this data augmentation enhanced the prediction performance of the QSPR model trained on a small chemical dataset and the structure generation performance of JT-VAE trained by transfer learning.

Although there remain some challenges related to the design of the structure generator and the scalability of the proposed method to training with a much smaller number of samples, it is expected to contribute to the efficiency of compound design in situations where only small-scale chemical datasets are available.

Acknowledgments

Immeasurable appreciation and deepest gratitude for the help and support are extended to the following persons who have contributed in making this dissatation possible.

My deepest gratitude goes first to Prof. Dr. Kimito Funatsu, my supervisor, who provided me an opportunity to accomplish this work. His utmost support and guidance benefited much in the completion and success of my research. It is a great honor to work under his supervision.

I would like to express my appreciation to Dr. Kenichi Tanaka, the assistant professor of Funatsu Laboratory. Thanks to his valuable insights, my work became more sophisticated.

My appreciation also extends to Prof. Dr. Yasuyuki Sakai, for his kind acceptance to Sakai–Nishikawa laboratory after Prof. Funatsu's retirement. It was due to his support that I was able to continue my research.

Special thanks go to all members of Funatsu Laboratory and Sakai–Nishikawa Laboratory, for their feedbacks, comments, encouragements, and of course friendship. Without them, my life in the Ph.D course would not have been as fun and exciting as it has been.

Last but certainly not least, I would like to thank my parents for supporting me from my hometown throughout my life.

Contents

Chapter 1

General Introduction

1.1 Computer-aided Molecular Design

Chemical products have become an indispensable part of our lives. We benefit from organic compounds, including pharmaceuticals, organic semiconductors, fragrances, and synthetic fibers, and inorganic compounds such as ceramics, electronic materials, and alloys. The creation of such functional compounds is of great scientific and industrial significance.

The stage of designing novel functional compounds with desired properties has been a bottleneck in developing chemical products [1]. Functional compounds have been developed generally in the following workflow (Figure 1.1): (1) design a candidate compound; (2) synthesize it; (3) test whether the synthesized compound has the desired properties; and (4) repeat this cycle until the synthesized compound has the desired properties. Promising compounds discovered through this cycle are then further improved and put to practical use. In the early stages of compound development, considerable trial and error is often required. Therefore, it is expected that the development cost of functional compounds can be reduced by improving the efficiency of this stage.

Chemoinformatics is an interdisciplinary field that unites chemistry and computer science, where informatics-based technologies such as machine learning are leveraged to solve problems in the field of chemistry. The discipline has attracted much attention in recent years as it provides compu-

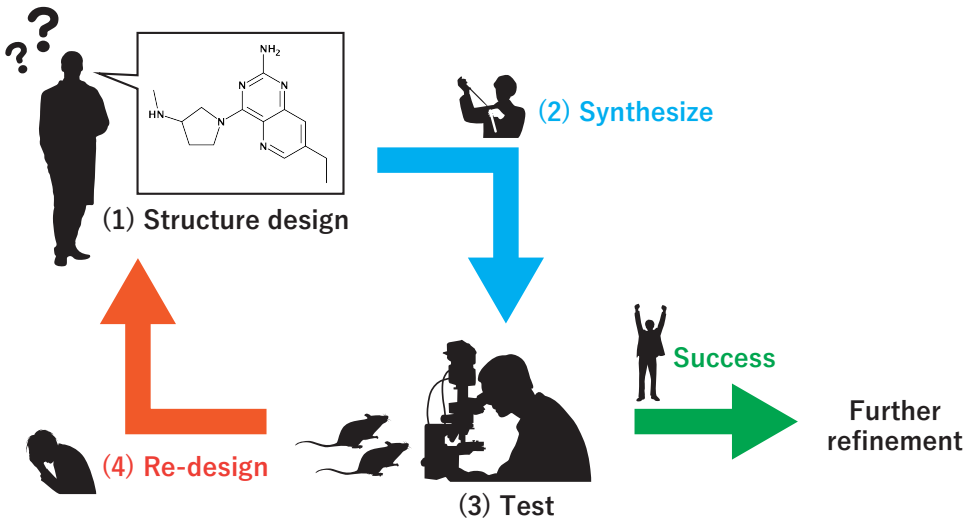

Figure 1.1: The development workflow of functional compounds.

tational methods to accelerate the development process of functional compounds. In chemoinformatics, for example, statistical models have been proposed to predict whether or not a compound has a particular property [2–6], generate the structures or compositions of promising candidate compounds [7–10], predict the product of a reaction [11–14] or the synthetic pathways of compounds [15–18], and suggest experimental designs to discover compounds with the desired properties efficiently [19]. These methods are now being used in the actual development of functional compounds [20, 21].

The format and handling of the data differ depending on whether the target compounds are organic or inorganic. In this thesis, we mainly focus on the methods for organic compounds.

One of the widely used chemoinformatics tools is the **quantitative structure–property relationship** (QSPR) model [22, 23] (Figure 1.2). The QSPR model is a statistical model that represents the statistical relationship f between the numerical vector $\boldsymbol{x}$ which describes information about a compound, and its label y, which is a numerical value that indicates the compound's property:

$$y \approx f(\boldsymbol{x}).$$

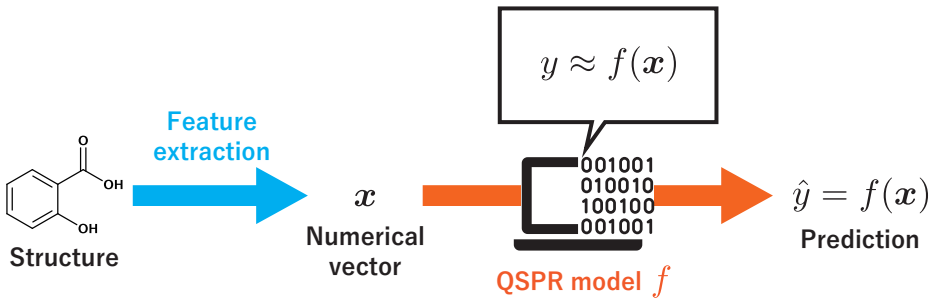

Figure 1.2: The QSPR model. The QSPR model f captures the statistical relationship between a numerical vector $\boldsymbol{x}$ extracted from the structure of a compound and its property y. The QSPR model can output the predicted property $\hat{y}$ of a compound.

In order to construct a vector $\boldsymbol{x}$ from the information of a compound, it is customary to calculate several **descriptors**, which are the characteristics of the compound (e.g., molecular weight and the number of specific substructures in the compound), or compound's **fingerprint**, which is a binary vector expressing the presence or absence of specific substructures in the compound. By training the QSPR model with a chemical dataset containing information on the physical properties of interest, i.e., by determining the model's parameters, we can predict the presence or absence of the physical properties of the compounds input to the trained QSPR model.

In addition to the QSPR model, molecular simulations such as quantum chemical calculations and molecular dynamics calculations can also be used to predict certain properties. Although it takes longer time to output the calculation results than the QSPR model, the molecular simulation can predict the properties more accurately by following the principles of physics.

One of the methods to find promising compounds with the desired properties using a trained QSPR model is the inverse QSPR. The inverse QSPR is a method that finds the descriptor vector $\boldsymbol{x}$ mapped to a given label y by the QSPR model f. When we obtain the set of descriptor vectors corresponding to the given y by the inverse QSPR, we can then obtain target compounds by recovering the structures of the compounds from their descriptor vectors. Although it is generally difficult to obtain the inverse image of y by

the QSPR model, $f^{-1}(y) = \{ \boldsymbol{x} \mid f(\boldsymbol{x}) = y \}$, or to obtain the structure of a compound corresponding to a given numerical vector $\boldsymbol{x}$, various efforts have been made so far [24–27].

Meanwhile, virtual screening using the QSPR model is another method to discover compounds with the desired properties. Virtual screening is a method to sift through many candidate compounds by various criteria such as molecular weight and the presence of specific substructures. We can efficiently discover novel compounds that may have the desired properties by using the QSPR model's predictions as the screening criterion and applying the virtual screening to a group of newly designed compounds.

In order to obtain many new molecular structures for virtual screening, **structure generators** are often used to generate molecular structures *in silico* (Figure 1.3). Many structure generators have been proposed so far. For example, the MOLGEN [28, 29] and the EnuMol [30] can enumerate the structural isomers of organic compounds. In addition, the GDB [31–33] generates an extensive range of organic compounds that contain 17 or less carbon, oxygen, nitrogen, sulfur, and halogen atoms. Because the structure generators listed here do not generate structures considering the physical properties, virtual screening is essential to find compounds that satisfy the desired properties from the compounds generated by these generators.

Meanwhile, some structure generators have been proposed to generate only promising compounds with the desired properties. For example, LigBuilder [34–36] can generate compounds that bind to a protein by using the structural information of the protein. In addition, Molgilla [37] has succeeded in generating compounds with desired properties [38] by using inverse QSPR [26]. Such structure generators are more efficient than those which enumerate all possible structures because they can restrict the generated structures by using statistical models constructed for the target properties in the generation.

In recent years, researches on **deep structure generators**, which use neural networks as statistical models, have been actively conducted, and promising results have been obtained [39–44]. While ordinary structure generators produce structures according to predetermined generation rules, deep structure generators model the data distributions of the training sam-

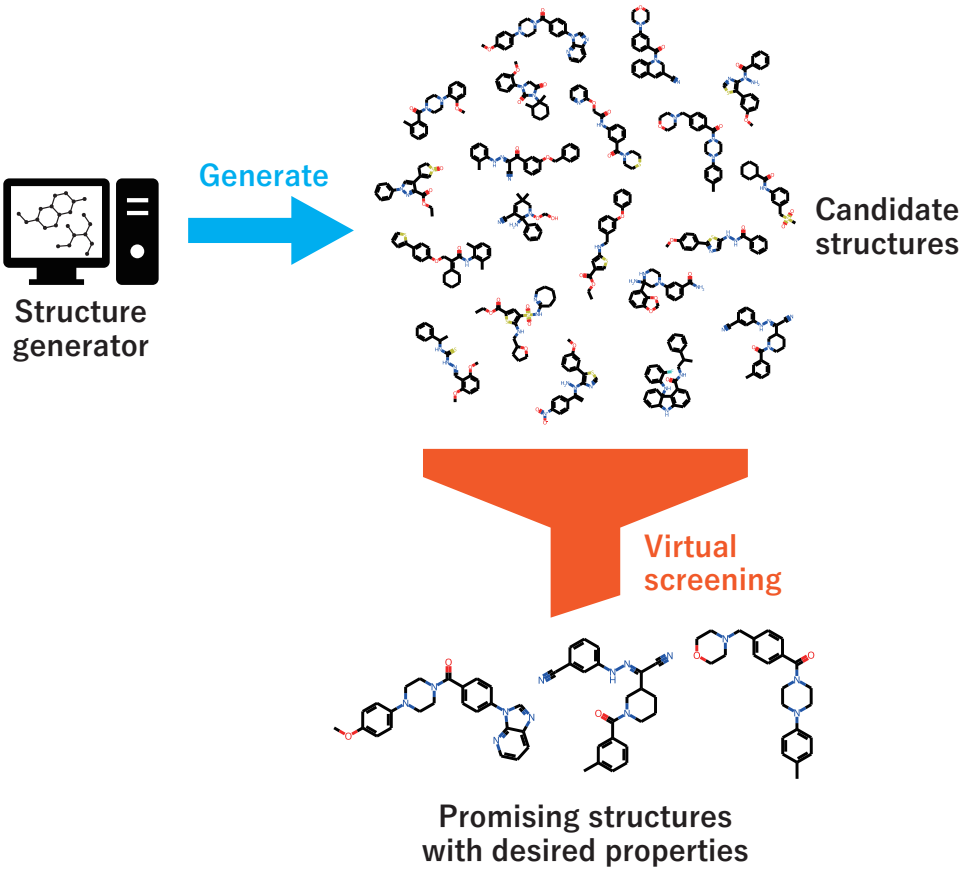

Figure 1.3: Exploration of promising molecular structures using a structure generator and virtual screening. The structure generator can produce many new molecular structures. Promising molecular structures with desired properties can be obtained by the virtual screening with suitable filters.

ples and generate molecular structures by sampling from these distributions. For this reason, the deep structure generator is more likely to generate compounds that capture the characteristics of the molecular structures in the training dataset.

Using structure generators can accelerate the process of structure design, which has so far been done only by the experience and intuition of chemists. In addition, the structure generator can sometimes generate interesting structures that chemists overlook. Therefore, the structure generator is expected to act as a strong supporter for chemists in the discovery of new functional compounds.

1.2 Challenges in Computer-aided Molecular Design

The performance of a structure generator should be defined by how efficiently it can improve the process of structural design. There are four desired features that a structure generator should achieve.

(a) The ability to generate many molecular structures with the desired properties. If this property is satisfied, it is easy to discover candidate compounds by virtual screening using a QSPR model.

(b) The diversity of the generated structures. If this property is satisfied, the generated structures are likely to include novel molecular scaffolds.

(c) The stability and synthesizability of the generated structures. If this property is satisfied, it is easy to test the physical properties of the selected candidate compounds.

(d) The high computational efficiency in generating molecular structures. If this property is satisfied, the time required for structure generation is shortened, and thus the method is easy to use.

It is not easy to satisfy these four properties simultaneously, but feature (a) can be achieved by using a statistical model. In the ordinary rule-based structure generator, by utilizing the predicted values of the QSPR

model in structure generation, we can focus mainly on promising structures. Meanwhile, the deep structure generator can also generate compounds likely to have the target properties by training with compounds having the desired property, because compounds with similar structures often have similar properties. Also, optimization methods have been developed to search for promising structures after combining a deep structure generator with a QSPR model and training them simultaneously [45].

In structure generators using statistical models, the performance of the statistical model used determines the generator's quality. In order to construct a statistical model with good performance, it is necessary to prepare a sufficient number of training samples to determine the model parameters. In particular, when we use a deep learning model, more training samples are usually required than for a model without deep learning because the number of model parameters to be adjusted is large.

However, it is not always possible to prepare enough training samples to build a model with good performance. If the data of the properties we are interested in do not exist in any database, we need to collect the data to build the model first. In this case, it is unlikely that a large amount of data can be obtained because of the considerable time and financial costs involved in the experiments when synthesizing and testing compounds. In addition, if molecular simulations can generate data on the properties of the compounds, the computational cost for a variety of compounds is enormous, and thus it is still time-consuming to obtain a large amount of data. Although it is possible to collect some data on the properties by conducting a literature survey, the number of samples obtained by this method will be a few hundred, or at most about a few thousand.

In the situation where the number of training samples is remarkably small compared to the number of model parameters, the statistical model is prone to overfit. This situation is typical especially when deep learning models are employed as statistical models. In the overfitted QSPR model, the prediction performance for the training samples will be good, but contrarily the prediction performance for the unknown samples, which are not used in training, will become poor, so that the predictions by the QSPR model are useless and property (a) above is not satisfied. In addition, the overfitted

deep structure generator tends to generate only the identical samples as the training samples or generate similar samples, and thus property (b) above is not satisfied. Hence, even if we obtain many compounds generated by a structure generator using an overfitted model, it will be difficult to discover promising compounds among them, and the compound development process will not be efficient. Therefore, in order to improve the efficiency of compound development using a statistical model-based structure generator, it is necessary to avoid overfitting of the model even when the dataset available for training is small.

1.3 Objective of This Thesis

This thesis aims to develop methods that enable statistical model-based structure generators to perform well even when trained on small chemical datasets (Figure 1.4). Here, we define a **small chemical dataset** as a dataset consisting of

(1) approximately 1000 samples and

(2) samples that are known to have properties of interest.

Considering the problems described in Section 1.2, this definition should be justified. If this objective can be achieved, it is expected that the application of the structure generator will be broadened, and thus it will be possible to discover new compounds with substantially improved functionality efficiently.

One of the simple ways to avoid model overfitting when training on small chemical datasets is to use a non-deep statistical model. When we do not use deep learning, it is easier to avoid overfitting the model because the number of model parameters is relatively small. Thus, a rule-based structure generator with a non-deep QSPR model is easier to satisfy the property (a) described in Section 1.2 by using the predictions of the model during structure generation. By improving the remaining properties (b)–(d), we can generate structures with good performance. In particular, we aim to improve property (b) to facilitate the generation of novel structures, considering the development of new compounds.

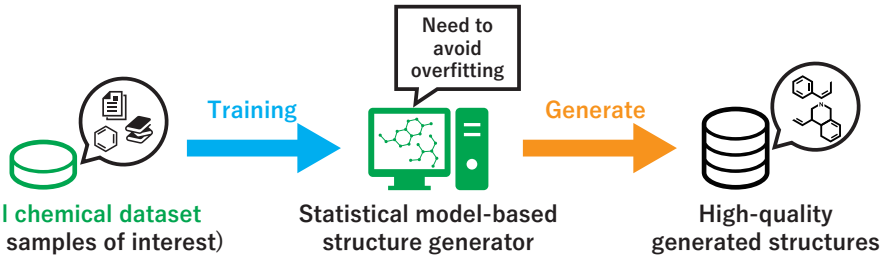

Figure 1.4: The objective of this thesis. We seek to develop methods with which statistical model-based structure generators can produce a high-quality set of generated structures by avoiding models' overfitting to the small chemical dataset.

Meanwhile, deep structure generators are prone to overfitting when trained on small chemical datasets due to many model parameters. However, the advantage of the deep structure generators is that it does not require the explicit design of the structure generation rules and can be used for any design target by simply changing the training dataset. Therefore, it is desirable to have a method to avoid overfitting even when the deep structure generator is trained on a small chemical dataset.

Transfer learning [46, 47] is one of the promising methods for training a deep structure generator on a small chemical dataset. It is a training method in which the deep statistical model is pre-trained on a large dataset other than the small chemical dataset and then trained on the target small dataset (Figure 1.5). By pre-training on the large dataset, overfitting to the small dataset can be suppressed. For the deep structure generator, training on many molecular structures that can be obtained from a database such as PubChem [48] or ZINC [49] allows the model to learn how molecular structures can be constructed. The deep structure generator is expected to be tuned to generate structures similar to the samples in the target dataset by training it on smaller chemical datasets afterward. In fact, some research [9] achieved successful generation of structures similar to the samples of the target dataset in this way.

In this thesis, we propose a method to enhance the effectiveness of transfer learning for deep structure generators. Specifically, we design a new data augmentation method for the training dataset. **Data augmentation** [50]

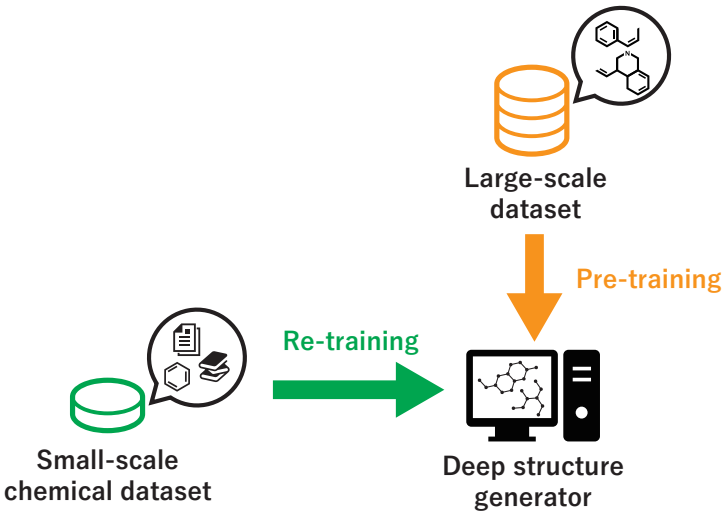

Figure 1.5: The overview of transfer learning for a deep structure generator. The deep structure generator is pre-trained on a large-scale dataset to learn how molecular structures can be constructed. The pre-trained deep structure generator is then re-trained on a small-scale chemical dataset to adapt the model parameters to the small dataset.

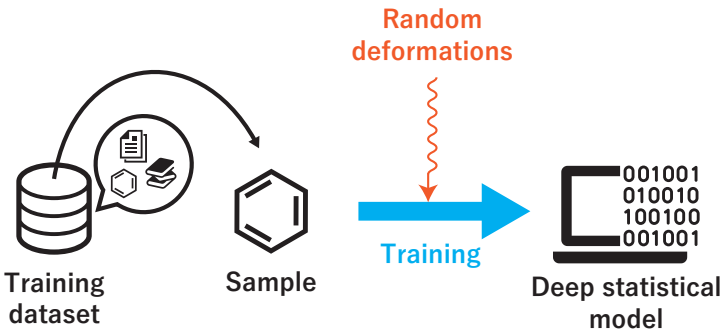

Figure 1.6: The overview of data augmentation. Data augmentation increases the apparent number of training samples by adding some random deformations to the inputs.

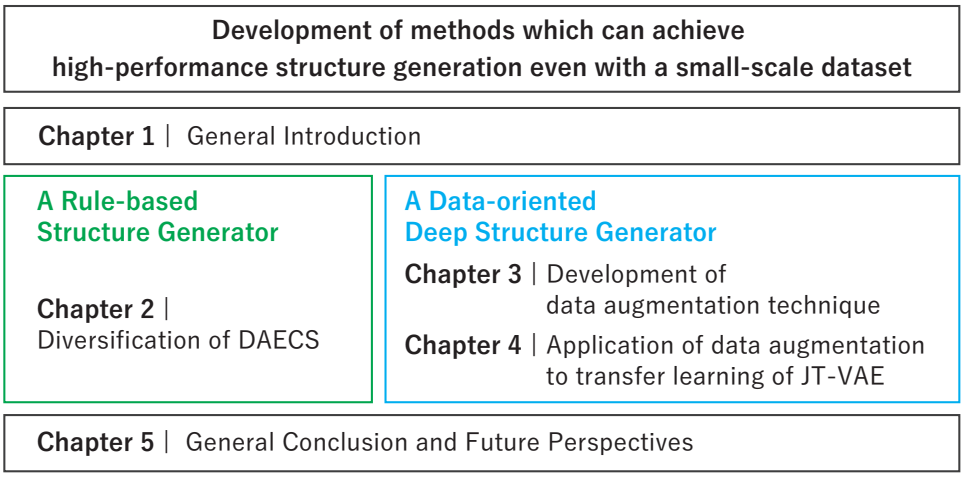

Figure 1.7: The structure of this thesis.

is an operation to increase the apparent number of samples in a training dataset by randomly deforming the samples, and it is known to be effective for training on small datasets (Figure 1.6). In particular, since the model is trained to make good predictions even when random deformations are added to the input samples, it is believed that the model can be constructed so that the essential features of the samples can be extracted. Therefore, if the data augmentation method is used for training the deep structure generator, it is expected that it will be easier to capture the features of the training dataset and generate a typical structure of the training samples.

1.4 Structure of This Thesis

This thesis is composed of five chapters (Figure 1.7). In Chapter 1 (this chapter), we describe the techniques used in computer-aided molecular design and their challenges. In order to achieve the main goal of this thesis, which is to generate structures with good performance using a small-scale chemical dataset, we use two methods: a rule-based structure generator without deep learning and a data-oriented deep structure generator using deep learning.

A Rule-based Structure Generator In Chapter 2, we propose a method that can generate diverse structures using a structure generator that does not use deep learning [51]. We choose to modify the ***de novo* design algorithm for exploring chemical space** (DAECS) [52, 53] because it has convenient features for chemists. DAECS generates a set of structures based on the given structures by repeating structural changes and structure selections. In order to increase the diversity of the structures generated by DAECS, we design new structural modification rules and structure selection algorithm.

A Data-oriented Deep Structure Generator In Chapter 3, we design a data augmentation method that can be used to train deep learning models on small chemical datasets. In particular, we develop a new data augmentation method for deep learning models that take molecular graph data as input. We construct QSPR models using the proposed method for regression task and classification task, respectively, and verify the effectiveness of the proposed method by checking their prediction performance.

In Chapter 4, we propose a method to enhance the effectiveness of transfer learning for deep structure generators. We choose to modify the **junction tree variational autoencoder** (JT-VAE) [54], which is known to be able to generate structures with good performance when using large datasets. In this thesis, we apply the data augmentation method designed in Chapter 3 to the feature extraction part of JT-VAE to capture the features of training samples more efficiently. To validate structure generation, we prepare a target small-scale chemical dataset and a large-scale dataset similar to the target dataset, and train the model by transfer learning using these datasets. The performance of the structure generation is then evaluated by metrics such as novelty and diversity of the generated structures and similarity to the datasets used for training.

Finally, in Chapter 5, we summarize the entire study in this thesis. We clarify the contributions of this thesis and discuss the remaining challenges and future perspectives. Despite some limitations of this work, we expect that this study will improve the efficiency of compound design with structure

generators that utilize small chemical datasets.

Chapter 2

Rule-based Generation with the Structure Generator DAECS

2.1 Introduction

In this chapter, we propose a statistical model-based structure generator that can successfully generate structures by utilizing a small chemical dataset. In such a structure generator, structures are generated according to predetermined generation rules while using the property prediction of the QSPR model for the generated structures. Using a non-deep statistical model with fewer parameters for the QSPR model, we can avoid overfitting the model to small chemical datasets and generate many compounds with desired properties.

Although there are many such structure generators, we chose to modify the *de novo* design algorithm for exploring chemical space (DAECS) developed by Mishima et al. [52] and improved by Takeda et al. [53] for two reasons: DAECS can generate compounds with desired properties, and users can control the structures generated by DAECS to some extent.

DAECS first visualizes the prediction results of the objective properties by the QSPR model in a two-dimensional (2D) map. Then, the user sets the coordinate of interest as a target in the visualized 2D map and gives an

initial structure to DAECS. DAECS prefers to generate compounds located near the given target coordinate on the map by repeatedly modifying the initial structure. In DAECS, the property distribution can be grasped thanks to the visualization by the 2D map. Therefore, it is easy to select a region that is predicted to have the desired property, and by setting the target coordinates to such a position, many promising compounds can be generated. In fact, when the target was set to the coordinates of the existing ligand for the histamine H_1 receptor, the docking simulation confirmed that the binding energies of some generated structures were comparable to that of the existing ligand [52], suggesting that DAECS is capable of generating molecular structures with the desired properties. Also, since DAECS generates structures based on the given initial structure, it is particularly beneficial for structural optimization, in which the input structure is modified to have the desired properties.

Among the problems of DAECS are the low diversity of the set of generated structures and the low stability and synthesizability of the generated structures. In particular, improving the diversity of DAECS-generated structures is an urgent issue because novelty in the molecular structures is necessary to develop new compounds with novel molecular skeletons.

In this study, we aimed to diversify the set of DAECS-generated structures by modifying the structure generation algorithm of DAECS. Specifically, we added the structural modification rules used in the structure generation of DAECS and modified the selection algorithm for selecting the structures to undergo a structural change from the set of generated structures. A case study using the activity data of ligands for the histamine H_1 receptor was conducted to verify whether the proposed method improves the diversity of the set of generated structures.

2.2 Methods

In this section, we first describe the generative topographic mapping, a method for dimension reduction, and then overview the DAECS algorithm. We then describe the proposed methods in detail.

2.2.1 Generative Topographic Mapping

Generative Topographic Mapping (GTM) [55] is an unsupervised learning method for dimension reduction. GTM finds the optimal nonlinear embedding from a 2D latent space into a D-dimensional descriptor space. Here, the variable belonging to the latent space is referred to as a latent variable. GTM reduces dimension of the input by finding a suitable latent variable embedded in the descriptor space. Four hyperparameters have to be determined for the GTM:

- the number of the latent grid points N_{grid},
- the regularization parameter (for preventing overfitting) λ,
- the number of the Gaussian kernels N_{kernel}, and
- the width of the Gaussian kernels γ.

The GTM model first generates N_{grid} latent variables arranged in a grid. The model assumes that a given descriptor vector $\boldsymbol{x}$ is stochastically generated as follows. First, a latent variable $\boldsymbol{s}$ is selected out of N_{grid} variables with equal probability $p(\boldsymbol{s}) = 1/N_{\mathrm{grid}}$. Next, the selected latent variable $\boldsymbol{s}$ is mapped to the descriptor space with the mapping $P_{\boldsymbol{W}}(\boldsymbol{s}) = \boldsymbol{W}\boldsymbol{\phi}(\boldsymbol{s})$, where $\boldsymbol{\phi}$ is nonlinear mapping and $\boldsymbol{W}$ is a parameter matrix. The mapping $\boldsymbol{\phi}$ consists of N_{kernel} Gaussian kernels of width γ, with each center being arranged in a grid. The model assumes that the descriptor vector $\boldsymbol{x}$ is normally distributed with mean $P_{\boldsymbol{W}}(\boldsymbol{s})$ and covariance matrix $\beta^{-1}\boldsymbol{I}$:

$$p(\boldsymbol{x} \mid \boldsymbol{s}, \boldsymbol{W}, \beta) = \mathcal{N}(\boldsymbol{x} \mid P_{\boldsymbol{W}}(\boldsymbol{s}), \beta^{-1}\boldsymbol{I}).$$

Here, β is a learning parameter. Given the training dataset, the learning parameters $\boldsymbol{W}, \beta$ can be determined with the EM algorithm [56].

After obtaining the optimal parameters $\boldsymbol{W}^*$ and β^*, the probability that each grid point $\boldsymbol{s}_i$ $(i = 1, \ldots, N_{\mathrm{grid}})$ had generated the descriptor vector $\boldsymbol{x}$ can be calculated by Bayes' theorem:

$$p(\boldsymbol{s}_i \mid \boldsymbol{x}, \boldsymbol{W}^*, \beta^*) = \frac{p(\boldsymbol{x} \mid \boldsymbol{s}_i, \boldsymbol{W}^*, \beta^*)p(\boldsymbol{s}_i)}{\sum_k p(\boldsymbol{x} \mid \boldsymbol{s}_k, \boldsymbol{W}^*, \beta^*)p(\boldsymbol{s}_k)}.$$

The 2D coordinates $\boldsymbol{z}(\boldsymbol{x})$ for the descriptor vector $\boldsymbol{x}$ can be calculated with the expectation of the grid point:

$$\boldsymbol{z}(\boldsymbol{x}) = \mathbb{E}[\boldsymbol{s}] = \sum_k \boldsymbol{s}_k \; p(\boldsymbol{s}_k \,|\, \boldsymbol{x}, \boldsymbol{W}^*, \beta^*).$$

Note that dimension reduction with GTM can be applied to the arbitrary descriptor vector $\boldsymbol{x}$, regardless of whether $\boldsymbol{x}$ is in the training dataset or not. Additionally, $\mathrm{proj}(\boldsymbol{x}) = P_{\boldsymbol{W}^*}(\boldsymbol{z}(\boldsymbol{x}))$ gives a projection of $\boldsymbol{x}$ onto the embedded 2D map in the descriptor space.

2.2.2 Overview of DAECS

In this section, we briefly review DAECS. See the original paper [52] for more detail. DAECS generates structures in two steps: the 2D map construction and the iterative structure generation (Figure 2.1).

2D Map Construction

The first step is the 2D map construction (Figure 2.2). It constructs a GTM-created 2D map to capture the distribution of training samples in the descriptor space, and then visualizes the QSPR-predicted property onto the 2D map. The algorithm trains the GTM model with the dataset for map construction to obtain the mapping $P_{\boldsymbol{W}^*}$. With the mapping $P_{\boldsymbol{W}^*}$, the descriptor vector $P_{\boldsymbol{W}^*}(\boldsymbol{z})$ for any latent variable $\boldsymbol{z}$ can be calculated. The predicted property for $P_{\boldsymbol{W}^*}(\boldsymbol{z})$ by the QSPR model, which is trained separately by using the same descriptors as the GTM model, is visualized at $\boldsymbol{z}$ with color, resulting in a 2D heatmap reflecting the QSPR-predicted property. Note that different descriptor vectors can be mapped to the same latent variable $\boldsymbol{z}$, but the visualized value on $\boldsymbol{z}$ are represented by the predicted value for $P_{\boldsymbol{W}^*}(\boldsymbol{z})$. Subsequently, a user determines the target coordinate $\boldsymbol{t}$ in the 2D map. The target $\boldsymbol{t}$ can be set at the arbitrary position around which they want to obtain structures, seeing the property landscape in the 2D map.

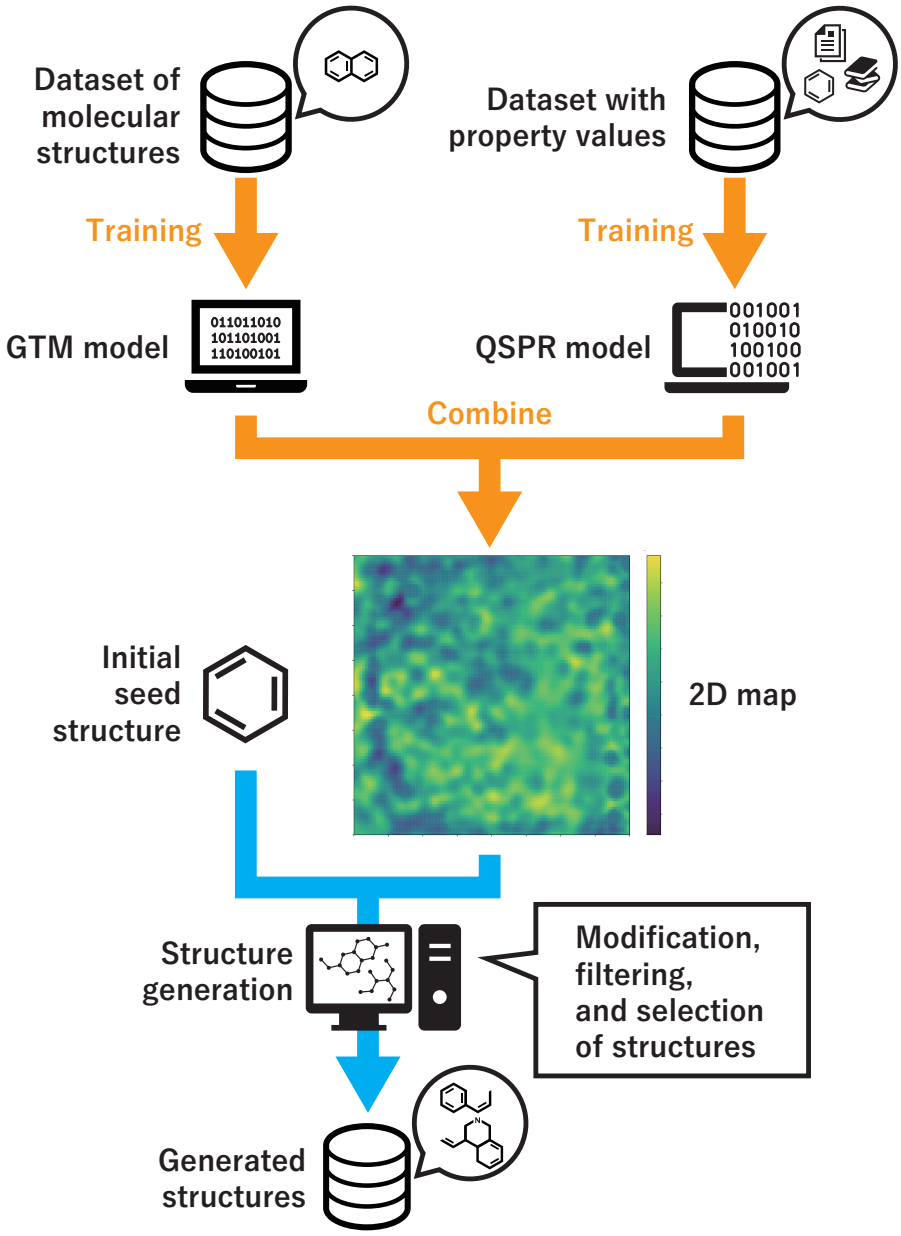

Figure 2.1: The overview of the DAECS algorithm. DAECS first constructs the 2D map from the given datasets. Then, DAECS utilizes the 2D map to generate structures by iterative modification of the given initial seed structure.

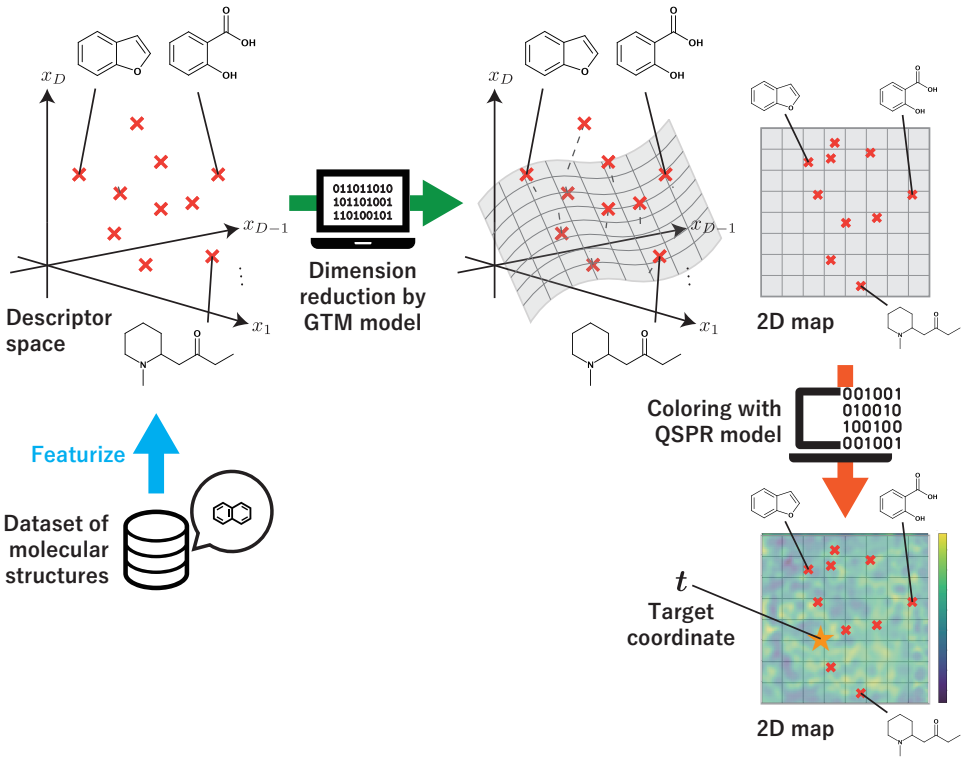

Figure 2.2: The 2D map construction step. GTM is used to capture the distribution of training samples in the descriptor space with the 2-dimensional surface. The QSPR-predicted property for each point in the surface is visualized by the 2D map with color. A user can set a target at the arbitrary position around which they want to obtain structures, seeing the property landscape in the 2D map.

Iterative Structure Generation

The second step is the iterative structure generation (Figure 2.3). DAECS prefers to generate structures around $\boldsymbol{t}$ based on some structures prepared arbitrarily by the user (we refer to these as the initial seed structures). Here, the user can select the initial seed structures by checking their coordinates on the 2D map. The algorithm for structure generation consists of three procedures:

(1) applying structural modification rules in an exhaustive manner;

(2) filtering the generated structures; and

(3) selecting the next seed structures.

The algorithm iterates these three procedures and terminates when a sufficient number of structures are obtained.

The following ten types of structural modification rules were applied (the numbers in the parentheses indicate the number of subtypes).

- Atom addition (8)
- Atom replacement (3)
- Atom insertion (3)
- Atom deletion (1)
- Bond replacement (3)
- Bond rearrangement (2)
- Ring formation (1)
- Ring dissociation (1)
- Ring aromatization (1)
- Ring saturation (1)

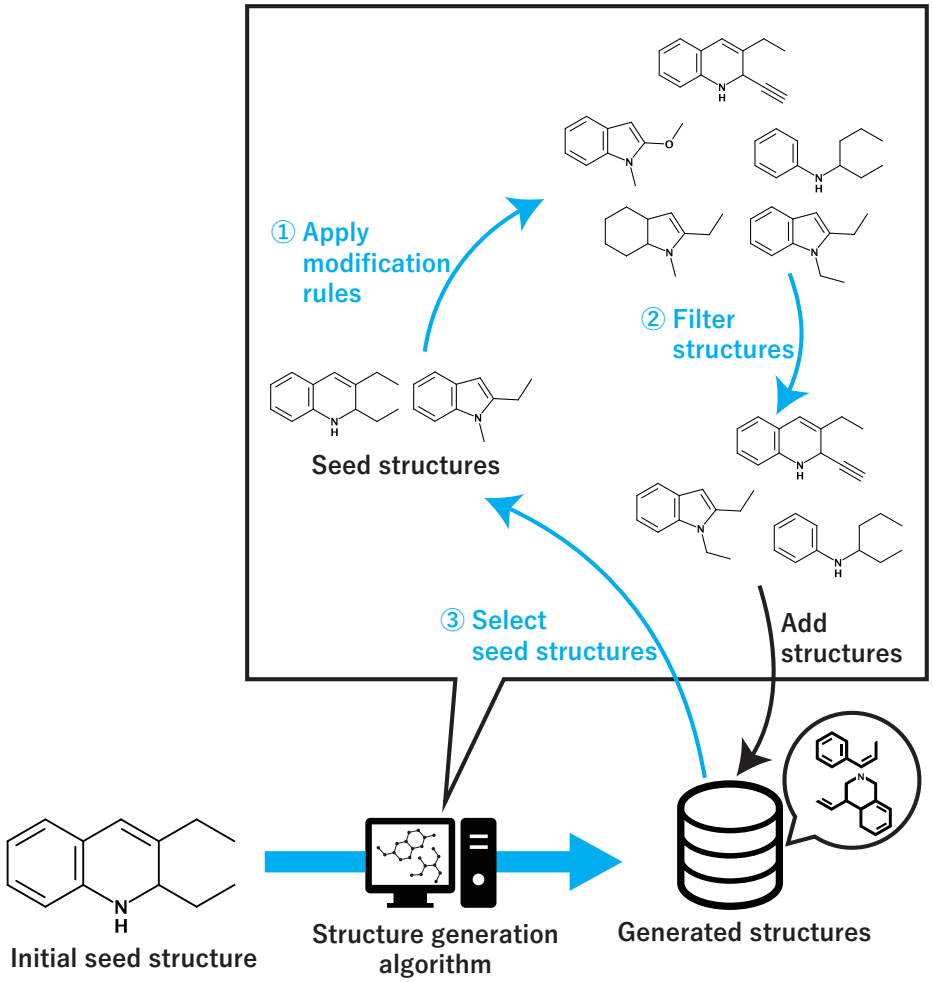

Figure 2.3: The iterative structure generation step. Blue curved arrows indicate the structure generation procedures in the second step of DAECS.

DAECS exhaustively generates new structures from each seed structure: as long as they are applicable (i.e., the resultant structure is chemically valid), these rules are applied to each part of a seed structure. Note that the rules we apply do not necessarily correspond to actual chemical reactions.

The generated structures include some unwanted structures, including those that are redundant and stereochemically unstable. These structures are discarded with some filtering rules. After filtering the structures, the remaining structures (we refer to these as candidate structures) were subjected to the seed selection algorithm (explained later) to choose K structures (K is a hyperparameter), which were used as the seed structures in the next iteration. The probability that a structure is selected is set to be higher if its descriptor is nearer to the target $\boldsymbol{t}$ on the 2D map and has a smaller projection error onto the embedded 2D map. This is because DAECS aims to generate structures whose coordinates on the 2D map are near the target $\boldsymbol{t}$. Note that a structure that is distant from the target or has a large projection error can be selected, although the probability is low.

2.2.3 Proposed Methods

In DAECS, generated structures are obtained by applying some structural modification rules. In addition, the generated structures primarily depend on the selected seed structures in each iteration. In this study, to diversify the structures generated by DAECS, we made the following modifications in the structure generation step of DAECS:

(1) addition of structural modification rules, and

(2) change of the seed selection algorithm.

The overview of the proposed methods is given in Figure 2.4, and the detailed explanation is given in the following sections.

Addition of Structural Modification Rules

Suppose that n structural modification rules are applied to a seed structure S until we obtain a chemical structure T. This means that $n-1$ intermediates are generated from S to T. DAECS generates T from S if and

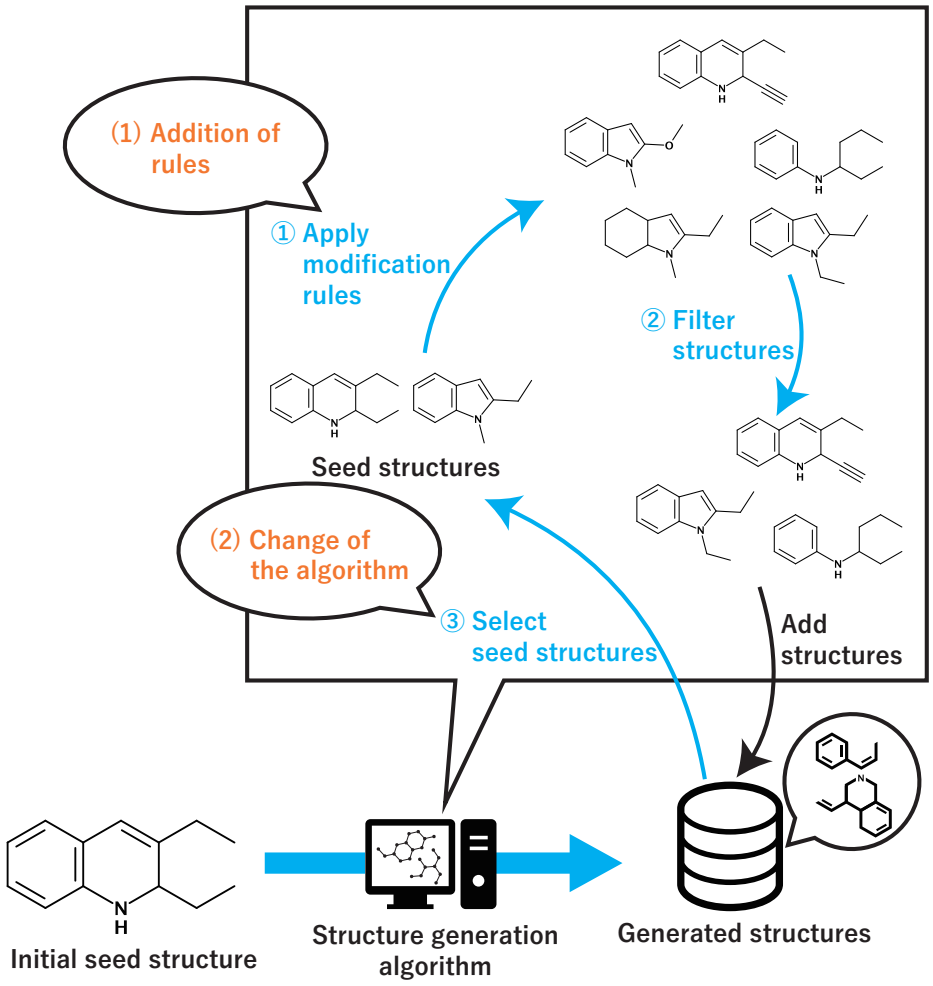

Figure 2.4: Overview of the proposed methods. Blue curved arrows indicate the structure generation procedures in the second step of DAECS. The proposed method is involved in the first and third procedure.

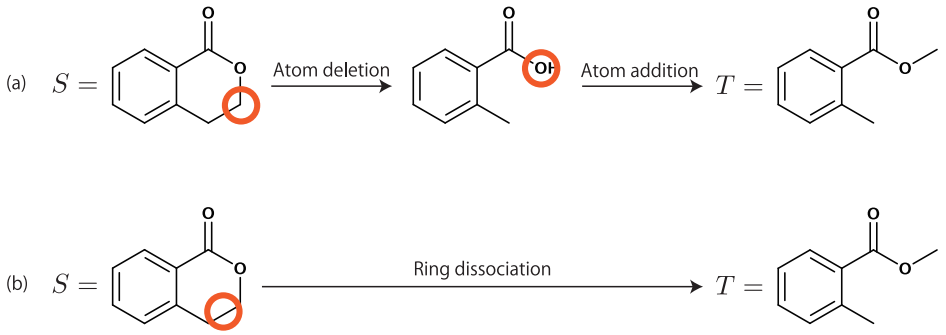

Figure 2.5: A example of multi-step structural modification. (Note that these structural modifications do not necessarily correspond to actual chemical reactions.) (a) Repetitive application of structural modification rules to obtain structure T from S. The red circles indicate the position where the designated rule was applied. In order that this modification from S to T takes place, the intermediate that is generated in the first iteration has to be selected as a seed structure in the second iteration, and the possibility was small in the previous version of DAECS. (b) Corresponding one-step rule has made this modification more likely to occur.

only if each intermediate structure from S to T is continuously selected as a seed structure in each iteration. An example is given in Figure 2.5. Since the number of candidate structures increases with each iteration, the probability that T is generated by DAECS decreases as n increases. The structural modification rules implemented in the previous version of DAECS are insufficient to make some structural modifications that need repetitive application of the existing rules.

Therefore, we designed new structural modification rules that can shortcut the multi-step structure modification in a single step. There are many such structural modification rules, but here we designed ones that are as general as possible. Namely, the structural modification rules were designed to be widely used regardless of the purpose of generation. In this study, we designed two new types of rules: bond contraction and ring mergence.

Bond Contraction The bond contraction rule selects a bond, removes the selected bond, and then the two atoms that formed the removed bond

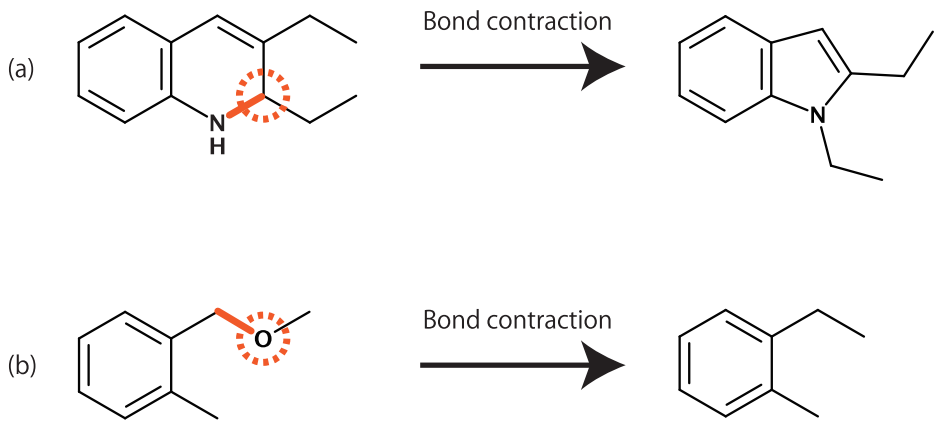

Figure 2.6: Bond contraction rules. The red bond indicates the selected bond, and the red dotted circle indicates the selected end-atom. (a) The carbon atom was removed and the side chain to the carbon atom was connected to the nitrogen atom, the other end-atom of the selected bond, resulting in smaller-membered ring. (b) The oxygen atom was removed, and the length of the straight chain was shortened.

(end-atoms) are regarded as a single atom (Figure 2.6). Here, the rule also selects either of the end-atoms to be removed. This rule can reduce the number of atoms in a structure. In particular, structural changes such as reducing the size of a ring or shortening the length of a straight chain can be performed in a single step. In the previous version of DAECS, the only rule to reduce the number of atoms in a structure was the atom deletion rule. Thus, it is expected that more small-sized structures can be obtained by adding this rule.

Ring Mergence The ring mergence rule selects a bond, and creates a new ring structure that includes the selected bond (Figure 2.7). We restricted the generated rings to four patterns: 5- to 7-membered aliphatic rings and a 6-membered aromatic ring (i.e., benzene ring). In creating aliphatic rings, all newly formed bonds are single bonds (note that the selected bond is possibly unsaturated). A single application of this type of rule greatly increases the number of atoms in a structure. For this reason, it is expected that a structure will change drastically by repetitively applying the rule. In

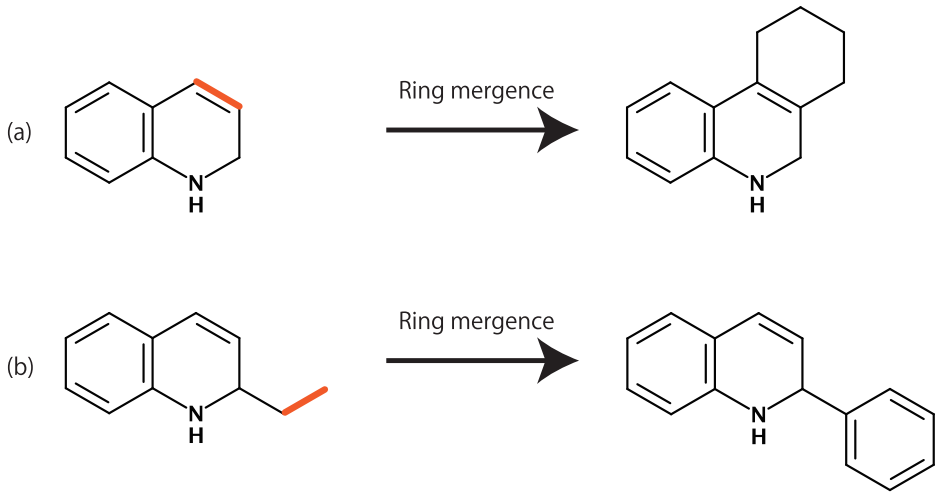

Figure 2.7: Ring mergence rules. The red bond indicates the selected bond. (a) The selected bond becomes a bond in the newly formed 6-membered ring. (b) The selected bond becomes a bond in the newly formed benzene ring.

particular, this rule enables one-step structural modification to create fused aromatic rings, such as naphthalene rings.

Seed Selection Algorithm

With the seed selection algorithm in the previous study, the structures near the target $\boldsymbol{t}$ are selected with high probability, so the selected seeds are prone to be similar to each other. Newly generated structures from similar structures are also similar to each other, which results in low diversity of the generated structures by the previous version of DAECS. In this study, we modified the seed selection algorithm to select more diverse structures.

Newly generated structures from a given seed structure are generally similar to the original structure, and they also tend to be mapped close to the seed structure on a GTM-created 2D map. Since the purpose of DAECS is to generate structures close to the target $\boldsymbol{t}$ (Figure 2.8 (a)), the selected seed structures should be mapped near the target $\boldsymbol{t}$. Therefore, we restrict the candidates of seed structures to those mapped near the target $\boldsymbol{t}$. For all N candidate structures, the distance from the target $\boldsymbol{t}$ on the 2D

map is calculated, and candidate structures are restricted to the R nearest structures from the target $\boldsymbol{t}$ ($R \in \mathbb{Z}_{>0}$ is a hyperparameter with $R < N$). We refer to the structures obtained by this approach as the restricted candidate structures. The number of restricted candidate structures R is determined so that the ratio $r = R/N$ to the total number of candidate structures N is constant.

If the two plots (i.e., two chemical structures) on the 2D map are distant to some extent, the corresponding two structures are considered to be dissimilar to each other. Therefore, we select those structures that are scattered in the restricted area in the 2D map (Figure 2.8 (b)).

A clustering method is used to select structures at scattered positions in the restricted area in the 2D map. Here, clustering is performed in the 2D map, not in the high-dimensional chemical space, to eliminate the curse of dimensionality: clustering methods do not give meaningful clusters in the higher-dimensional space. By training the clustering model with the coordinates obtained by plotting the restricted candidate structures in the map, we can divide the restricted candidate structures into K clusters (Figure 2.8 (c)). We used the k-means++ algorithm [57] for the clustering model. The structures in the same cluster are considered to be similar because their 2D coordinates are close to each other. In contrast, we consider structures in different clusters to not be so similar to each other because they are distant to each other in two dimensions. Therefore, we select the seed structures by sampling one structure from each cluster uniformly at random (Figure 2.8 (d)).

2.3 Results and Discussion

We performed structure generation to verify the effectiveness of the proposed method. As in previous studies of DAECS [52, 53], the ligand structures for the histamine H_1 receptor were generated, and the performances of the previous and proposed methods were compared.

In this experiment, we used Python 3.5.3 and `RDKit` [58] (version 2017.09.1) for implementation. We constructed both the QSPR and dimension-reduction models with all of the available `RDKit` descriptors, which are composed of

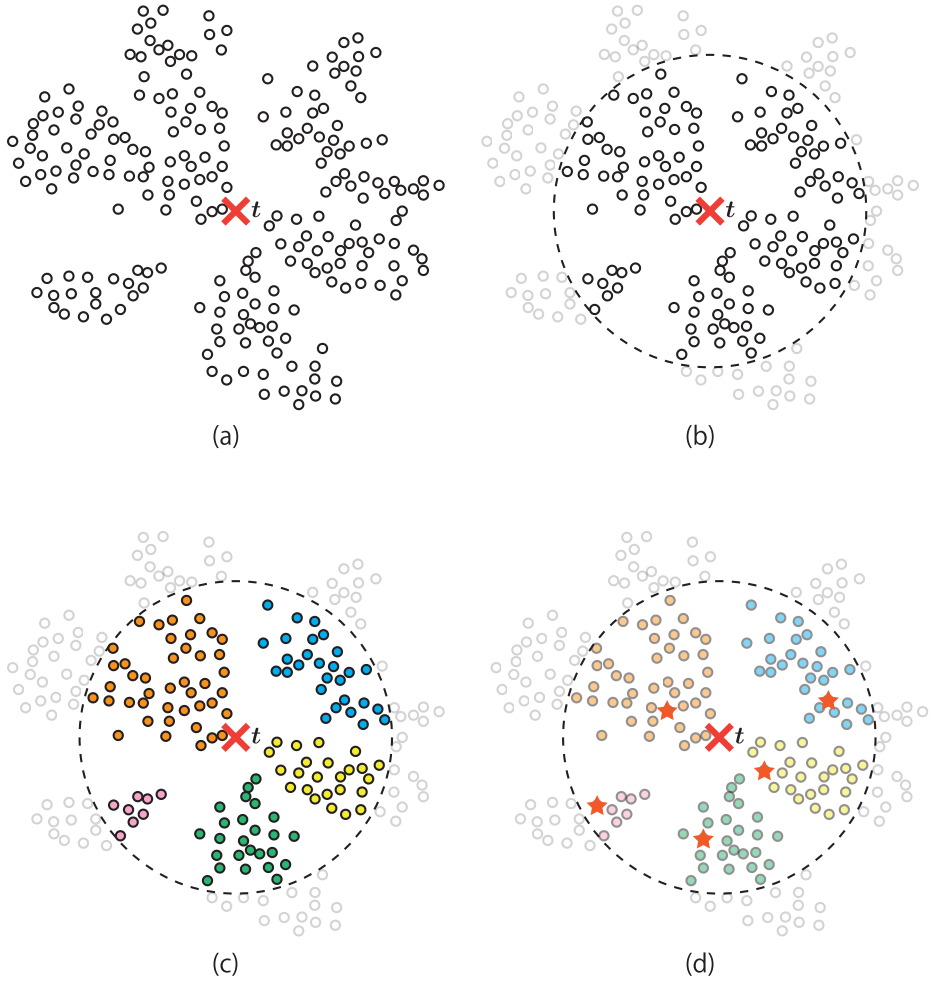

Figure 2.8: Procedures of the proposed seed selection algorithm. Solid circles indicate the positions of generated structures on the map. (a) Initial state. (b) Restricted structures. (c) Restricted structures were partitioned into $K = 5$ clusters. (d) Five structures indicated by the red stars are randomly sampled from each cluster.

200 descriptors including, e.g., molecular weight and the number of aromatic rings. The detail of the used descriptors can be found in the official website of `RDKit`. Note that the `RDKit` descriptors, which are commonly used for feature extraction of compounds, were used here, but any other descriptor can be used as long as the feature extraction method is fixed in DAECS.

2.3.1 Datasets and Models

The QSPR Model

The QSPR model is used only to visualize the property landscape in the 2D map, and the prediction results of the QSPR model are not used in the structure generation step. Therefore, the QSPR model with good accuracy is not necessary to verify the effectiveness of the proposed method. Here, we constructed the QSPR model only to complete the case study by demonstrating the workflow of structure generation using DAECS.

We used the dataset consisting of 522 known ligand structures for the histamine H_1 receptor from ChEMBL17 [59], the same dataset as the previous study [53]. Experimentally verified inhibition constant K_i is associated to each of the obtained structure. In this study, we constructed a QSPR model to predict $\mathrm{p}K_\mathrm{i}$ values for the histamine H_1 receptor. As in the previous study [53], we used support vector regression (SVR) [60], which is a non-deep method, as a learning algorithm for the QSPR model, and the Gaussian kernel as a kernel function of SVR. `Scikit-learn` (version 0.19.1) [61] was used to train the model. To determine the optimal hyperparameters for the model, a grid search with five-fold cross-validation was employed, using the mean squared error (MSE) as a loss function. Consequently, the optimal hyperparameters were determined as follows: the regularization parameter $C = 10$; the tolerance $\varepsilon = 10^{-1}$; and the width of the Gaussian kernel $\gamma = 10^{-8}$. The QSPR model was trained with these hyperparameters. With five-fold cross-validation, the MSE and the coefficient of determination R^2_CV were calculated to check the prediction accuracy of the model. Here, R^2_CV is defined by

$$R^2_\mathrm{CV} = 1 - \frac{\sum_{i=1}^{N}(y_i - \hat{y}_{\mathrm{CV},i})^2}{\sum_{i=1}^{N}(y_i - \bar{y})^2},$$

where N is the number of training data, y_i is the observed value for the i-th sample, $\hat{y}_{\mathrm{CV},i}$ is the predicted target value for the i-th sample in the validation fold, and $\bar{y}$ is the mean of the observed target value. The accuracy of the model improves as R^2_{CV} approaches 1. The calculated MSE and R^2_{CV} were 0.5689 and 0.6004, respectively. Although there is room for improvement, we used this QSPR model hereafter.

The Dimension Reduction Model

Subsequently, we trained the GTM model using 5,000 structures with a molecular weight of 500 or less from the BIOVIA Available Chemicals Directory [62], which contains commercially available compounds. Note that only molecular structure information is required for the training of GTM, and such unlabeled data can be obtained from public databases. We fixed the number of the latent grid points $N_{\mathrm{grid}} = 60^2$, and we tuned the remaining three hyperparameters $\lambda, N_{\mathrm{kernel}}$ and γ. By using `GPyOpt` (version 1.2.1) [63] combined with `GPy` (version 1.9.2) [64], Bayesian optimization was performed to determine the optimal hyperparameters for the GTM model because this optimization approach performs the hyperparameter search efficiently. The root mean squared error of midpoints (RMSEM) [65] was used as a loss function to improve the mapping accuracy not only for samples in the dataset but also for samples not included in the dataset. The RMSEM value for the k-nearest points was calculated by

$$\mathrm{RMSEM}_k = \sqrt{\frac{1}{Nk}\sum_{i=1}^{N}\sum_{\boldsymbol{m}\in M_k(\boldsymbol{x}_i)} \|\boldsymbol{m} - \mathrm{proj}(\boldsymbol{m})\|^2},$$

where N is the number of samples in the dataset, $M_k(\boldsymbol{x})$ is the set of midpoints between $\boldsymbol{x}$ and the k-nearest points of $\boldsymbol{x}$, and $\mathrm{proj}(\boldsymbol{m})$ is the point that is yielded by projecting $\boldsymbol{m}$ onto the manifold obtained by the trained GTM model. In this study, we set $k = 10$. Smaller RMSEM_{10} values indicate better accuracy of the GTM model for unknown data. The optimal hyperparameters calculated were $\lambda = 10^1, N_{\mathrm{kernel}} = 60^2$ and $\gamma = 2^{0.1044}$. These hyperparameters enabled the convergence of the RMSEM_{10} value (Figure 2.9). Using the trained GTM model and QSPR model, we visualized the distribution of predicted $\mathrm{p}K_{\mathrm{i}}$ values in a 2D map.

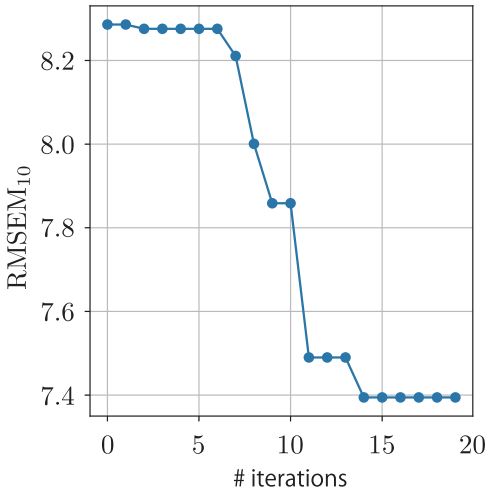

Figure 2.9: The trajectory of RMSEM_{10} in tuning hyperparameters of the GTM model. RMSEM_{10} converged after 14 iterations.

2.3.2 Metric of Structural Diversity

A chemical structure can be converted into a binary vector called a fingerprint, which represents the presence or absence of respective substructures. Fingerprints are used to calculate the Tanimoto distance, which is an index that represents the dissimilarity between two chemical structures. The distance between S_1 and S_2 is defined by the following equation:

$$\mathrm{dist}(S_1, S_2) = 1 - \frac{a}{b},$$

where a is the number of substructures that are present in both fingerprints, and b is those that are present in at least one of the two fingerprints. The Tanimoto distance ranges from 0 to 1. Values closer to 1 indicate that the two chemical structures S_1 and S_2 are more divergent. In this study, the Morgan Fingerprint [66] (2,048 bits, radius 2) was used. Note that `RDKit` descriptors, which were used to construct the QSPR model and the dimension-reduction model, is not a binary vector and therefore cannot be used to calculate the Tanimoto distance. For the set of chemical structures $\mathscr{G}$, we defined the diversity of $\mathscr{G}$ to be the mean of the Tanimoto distances between all possible combinations of two different structures in $\mathscr{G}$:

$$\mathrm{Div}(\mathscr{G}) \coloneqq \frac{1}{\binom{|\mathscr{G}|}{2}} \sum_{\{S,T\}\subseteq\mathscr{G}, S\neq T} \mathrm{dist}(S, T).$$

Here, $\binom{a}{b}$ indicates the binomial coefficient. This metric $\mathrm{Div}(\mathscr{G})$ takes a value from 0 to 1. Values closer to 1 indicate that the chemical structures in the set $\mathscr{G}$ are less similar to each other, i.e., the set $\mathscr{G}$ is more diverse.

2.3.3 Conditions for Structure Generation

Suppose that we hope to obtain an unknown chemical structure near the target $\boldsymbol{t}$ at the red cross shown in Figure 2.10 and we hope to start with an initial seed structure S_{init} with known chemical structure as shown in Figure 2.11. Note that the target $\boldsymbol{t}$ can be set at the arbitrary position on the 2D map. From 522 structures included in the training dataset of the QSPR model (not in the training dataset of the GTM), the initial seed structure S_{init} was selected as the nearest structure of the target $\boldsymbol{t}$. The number of iterations in the structure generation step was fixed to ten.

Throughout this experiment, the ten types of structural modification rules explained in Section 2.2.2 were used. In applying structural modification rules, only C, N and O atoms were considered. When using the ring formation rule, we limited the number of ring members to 5 – -7 so that the generated ring would be stereochemically stable. Note that it is still possible to generate a ring that is not 5- to 7-membered as the result of applying other rules. The structural modification rules newly designed in this study are explained in the Section 2.2.3, and we checked whether the diversity could be improved by adding these rules.

We used two filtering rules: one is to discard duplicate structures and the other is to discard structures with an unstable ring (3- or 4-membered ring with a double bond and 3- to 7-membered ring with a triple bond).

The number of selected seed structures in each iteration was set to $K = 5$. We checked that the proposed algorithm for seed selection outperformed the previous one in terms of the diversity of generated structures. We set the ratio of the number of candidate structures in the proposed algorithm to $r = 1/3$ so that the number of restricted candidate structures does not grow too rapidly.

Table 2.1 shows four conditions tested to confirm the effects of the two proposed methods (i.e., the seed selection algorithm and the structural mod-

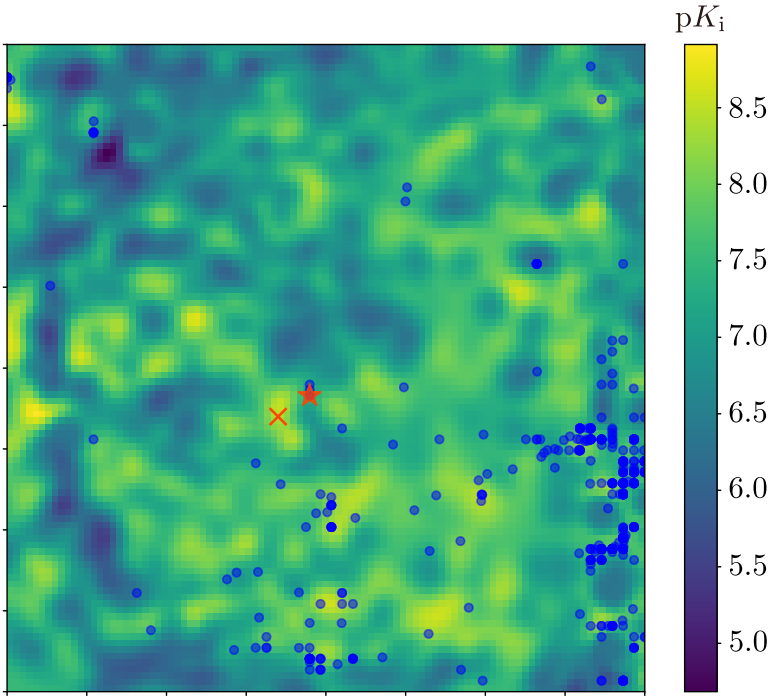

Figure 2.10: The position of the target and the initial seed structure. The red cross indicates the target $\boldsymbol{t}$, and the red star indicates the position of the initial seed structure selected, i.e., S_{init}. The blue dots are the position of 522 structures in the training dataset for the QSPR model. The background is the obtained 2D map of predicted pK_{i} values.

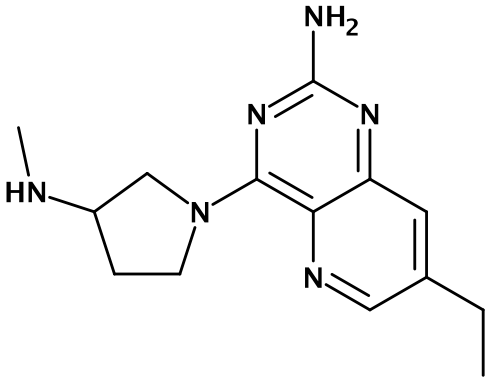

Figure 2.11: The initial seed structure S_{init}.

Table 2.1: Summary of four methods compared in the experiment.

		Set of structural modification rules	
		Previous	Proposed
Seed selection algorithm	Previous	Method P	Method R
	Proposed	Method S	Method RS

ification rules).

2.3.4 Results of Generation

Distribution of Generated Structures on the 2D Map

We checked the location of the generated structures in the chemical space. Figure 2.12 shows the plots of the structures generated by each method.

Methods P and R use the same previous seed selection algorithm but different set of structural modification rules. The obtained distributions by them are generally similar to each other, but method R generated some new structures in the area to the right of the initial seed structure. The distribution of the generated structures did not change significantly in the method because the distribution of the selected seed structures was similar. This similarity of the seed distributions was because of the behavior of the seed selection algorithm. We obtained an analogous result when comparing methods S and RS, which use the same proposed seed selection algorithm but different set of structural modification rules.

Having compared the methods S with P, both of which use the same previous set of structural modification rules but use different seed selection algorithms, the distribution of structures near the target in method S was broader than that in method P, with many new structures generated in the area to the left of the target. The same tendency was observed when comparing the distributions of structures in method R with that in method RS, both of which use the same proposed set of structural modification rules but use different seed selection algorithms. This result suggests that changing the algorithm for selecting seed structures had a significant effect on the distribution of generated structures near the target. In addition, the

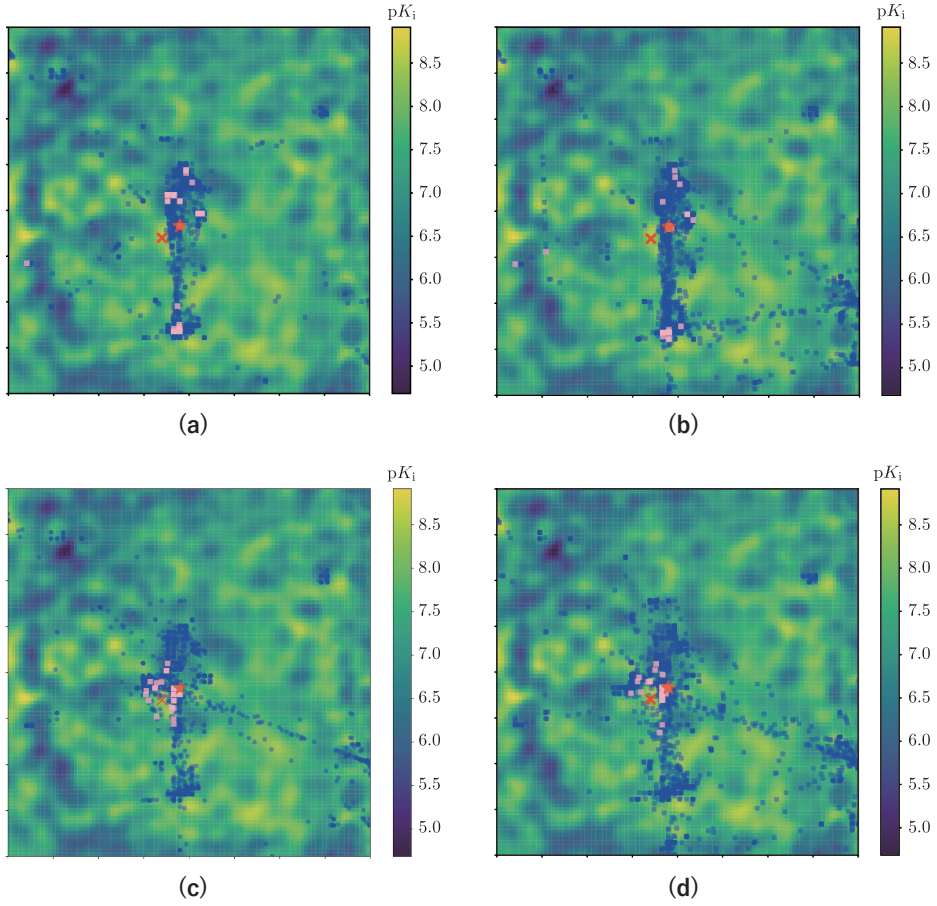

Figure 2.12: The distributions of the generated structures by each method. The red cross indicates the target $\boldsymbol{t}$, and the red star indicates the position of the initial seed structure selected, i.e., S_{init}. The blue dots indicate the positions of the generated structures, and the pink squares indicate the selected seed structures in each iteration. (a) Method P, (b) Method R, (c) Method S, and (d) Method RS.

Table 2.2: Number of generated structures and diversity indices.

	$\lvert\mathscr{G}_{\mathrm{ent}}\rvert$	$\lvert\mathscr{G}_{\mathrm{near}}\rvert$	$\mathrm{Div}(\mathscr{G}_{\mathrm{near}})$
(a) Method P	7,412	403	0.7013
(b) Method R	8,766	353	0.6551
(c) Method S	5,096	1,951	0.7574
(d) Method RS	7,139	2,343	0.7575

selected seed structures were scattered near the target as expected.

Diversity of Generated Structures around the Target

We counted the number of the structures in the following groups of generated structures: all of the generated structures ("the entire structures", $\mathscr{G}_{\mathrm{ent}}$) and the generated structures plotted closer to the target than the initial seed ("the near structures", $\mathscr{G}_{\mathrm{near}}$). We also calculated the diversity of the near structures, $\mathrm{Div}(\mathscr{G}_{\mathrm{near}})$. Table 2.2 shows the results of each method.

The proportion of the number of near structures to the number of entire structures in methods S and RS were larger than that of method P or R. This is due to the selected seeds: most of the selected seeds lay around the target (Figure 2.12 (c) (d)), resulting in more structures near the target. This suggests that the proposed seed selection algorithm contributes to the increase in number of the near structures. To explain the diversity of the near structures in each method, the distribution of Tanimoto distances between arbitrary pairs of structures taken from the near structures $\mathscr{G}_{\mathrm{near}}$ are shown in Figure 2.13.

We initially compared methods P and R, which use the same previous seed selection algorithm but differs in structural modification rules. In terms of the diversity of the near structures, method R using the proposed set of structural modification rules underperformed method P (Table 2.2), with the histogram of the Tanimoto distance peaking at around 0.7 (Figure 2.13 (b)). As the number of structural modification rules increased, the number of structures generated from the same seed structure also increased. Therefore, the number of similar structures near the target increased, which we postulate caused a decrease in diversity.

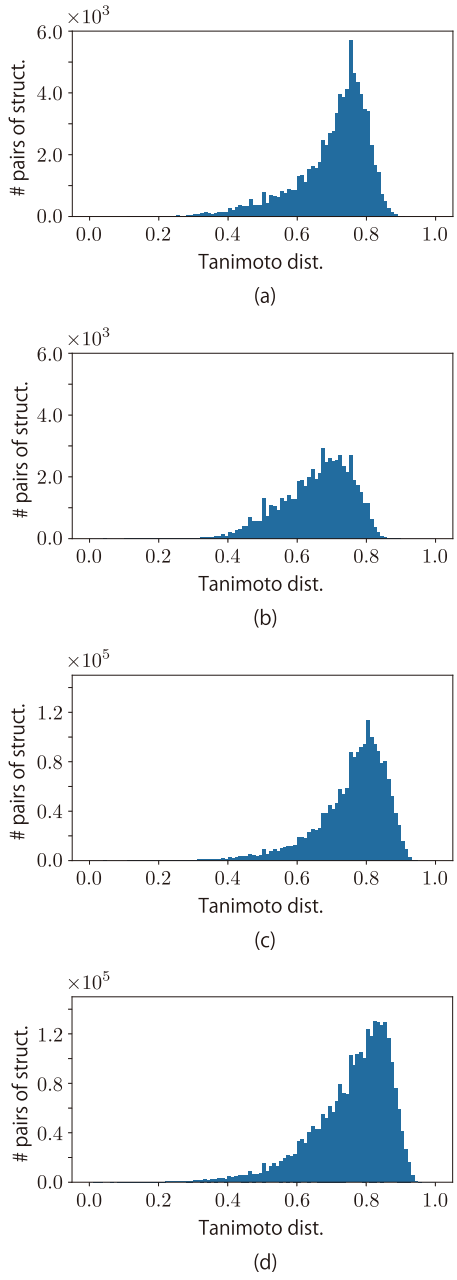

Figure 2.13: Distributions of Tanimoto distances between arbitrary structures taken from the near structures $\mathscr{G}_{\text{near}}$. (a) Method P, (b) Method R, (c) Method S and (d) Method RS.

Second, we compared methods P and S, which use the same previous set of structural modification rules but differs in seed selection algorithms. Method S outperformed method P in terms of the diversity of the near structures (Table 2.2), with the peak of the histogram in method S shifting to the right when compared with that of method P (Figure 2.13 (a) (c)). The reason for the increase in the diversity of the near structures in method S is the diverse seed structures around the target: generated structures from that seeds were also diverse and many of them lay around the target. In conclusion, the proposed seed selection algorithm can increase the diversity near the target by selecting seeds from the near structures.

Third, we compared methods S and RS, which use the same proposed seed selection algorithm but differs in structural modification rules. In terms of the diversity of the near structures, method RS slightly outperformed method S (Table 2.2), and the histograms were similar in shape, with the Tanimoto distance peaking at around 0.8 (Figure 2.13 (c) (d)). Since the distributions of the seed structures were similar between methods S and RS (Figure 2.12 (c) (d)), the major difference between these methods is the set of structural modification rules. We can conclude that the new structural modification rules actually generated the structures in the vicinity of the target. Unlike in method R, the newly added rules did not deteriorate the diversity of the near structures in method RS. This is because the selected seed were diverse and near the target thanks to the proposed algorithm, suggesting the proposed seed selection algorithm more greatly affects the generated structures than the newly added rules.

Comparison of Generated Structures Near the Target

We examined the four structures nearest to the target generated by each method (Figure 2.14). For all the methods, the molecular skeletons of the structures were similar to that of the initial seed structure. However, the aromatic ring substructures in the structures generated by method RS were not included in the training dataset. These substructures were inherited from a seed structure selected in the first iteration of method RS (Figure 2.15) and were generated by contracting the amino group bond attached to the

aromatic ring of the initial seed structure. These results suggest that the newly added rules contributed to the formation of novel structures.

2.4 Conclusion

In this study, we added structural modification rules and modified the seed selection algorithm to diversify the structures generated by DAECS. We designed two new types of structural modification rules: bond contraction and ring mergence. The modified seed selection algorithm was designed so that diverse structures near the target were selected.

The effectiveness of the proposed method was demonstrated using the histamine H_1 receptor as a case study. The proposed seed selection algorithm clearly improved the diversity of the generated structures near the target. The newly added two structural modification rules did not change the overall distribution significantly; however, we observed they contributed to the novel structures.

In the case study, a small chemical dataset consisting of 522 samples was used to build the QSPR model. In the previous studies of DAECS [52, 53], QSPR models were also constructed using the same dataset, and docking simulations predicted that some of the ligand structures generated near the target would bind to the protein. This suggests that DAECS with the proposed method will also have the ability to generate ligand structures that bind to the protein.

As the number of training samples of the QSPR model becomes even smaller, the prediction performance will inevitably become worse, and the structures generated near the target may not have the desired properties. In such a case, we can use a model that can calculate the confidence of prediction such as the Gaussian process regression, and set the target to the coordinates that are likely to have the desired physical properties.

Although the diversity of the structures generated by DAECS was improved by the proposed method, there remain problems to be solved in DAECS. First, DAECS might generate stereochemically unstable structures, such as those with overly large distortions. This problem is caused by:

(1) the structural modification rules being applied exhaustively;

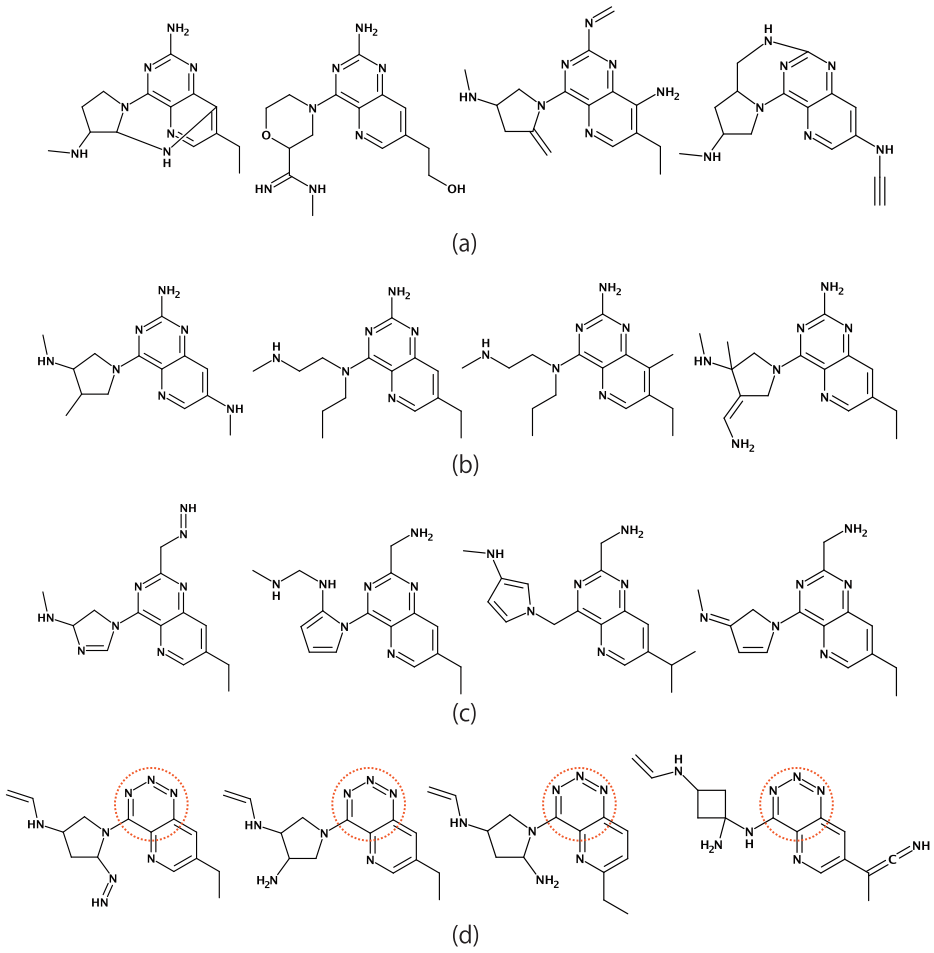

Figure 2.14: The four structures closest to the target generated by each method. (a) Method P, (b) Method R, (c) Method S and (d) Method RS. The orange dotted circle indicates the novel substructure generated by the bond contraction rule.

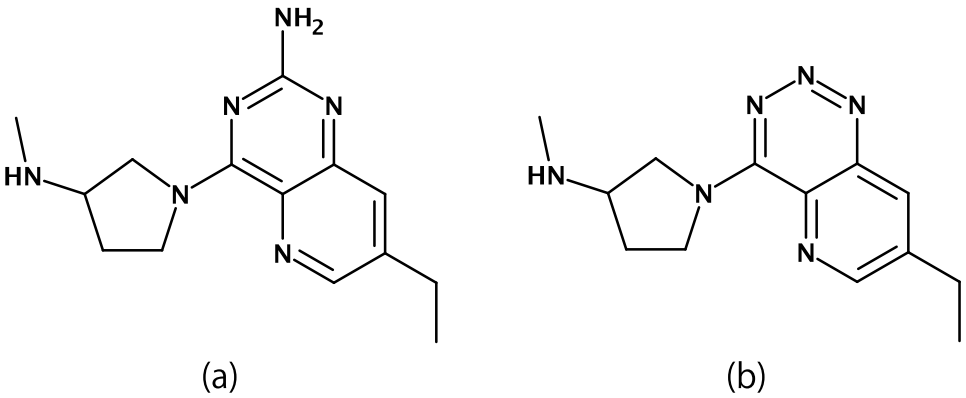

Figure 2.15: (a) The initial seed structure S_{init} (identical to Figure 2.11). (b) A seed structure selected in the first iteration of method RS.

(2) that the structural modification rules do not imitate the actual chemical reaction; and

(3) that there are few filtering rules to remove unwanted structures.

If an unstable structure was selected as a seed structure in some iteration, the stereochemical stability of the generated structures would be reduced because the generated structures based on the seed structures can also contain unstable substructures. To increase the stability of the generated structures, the structural modification rules should be improved to avoid generation of unstable structures, or to add filtering rules that remove unstable structures. Furthermore, calculation of the formation energy of the structure enables filtering of unstable structures by setting an appropriate threshold of the energy. Since it takes considerable time to calculate the formation energy of many structures correctly, it is conceivable that a machine learning method rather than calculating the energy accurately can predict the formation energy. Also, hard-to-synthesize structures might be generated by DAECS due to the same reasons. Such structures can be removed by calculating the metric for synthesizability, such as the SA score [67].

Second, DAECS can also be improved by supporting more types of atoms. All structural modification rules in DAECS can handle only carbon, nitrogen and oxygen atoms. However, many compounds contain other atoms such as sulfur, phosphorus, fluorine and chlorine atoms. For practi-

cal use, DAECS needs a mechanism that can generate structures containing these atoms, and including this mechanism will increase the diversity of the generated structures.

Third, we should consider how to apply the structural modification rules. Currently, the algorithm generates structures by applying structural modification rules exhaustively, but many structures located far from the target in the 2D map are also generated. Selection of suitable rules that are applied in each loop should remove unnecessary structures and improve computational efficiency for structure generation.

Moreover, the seed selection algorithm also needs further study. Both of the previous and the proposed methods generally select structures near the target coordinates, but it is possible that a structure distant from the target coordinate is transformed into ones near the target. Also, for the ratio parameter r in the proposed method, it is necessary to search for appropriate values to further increase the diversity of generated structures.

Chapter 3

Data Augmentation for Small-scale Datasets of Molecular Graphs

3.1 Introduction

In this and the next chapters, we deal with deep learning models as statistical models. A deep learning model consists of statistical models called **neural network**, which repeatedly performs linear and nonlinear transformations on samples. Although the number of model parameters will increase, it is possible to model complicated relationships between inputs and outputs by combining multiple neural networks.

The most notable difference between models that use deep learning and those that do not is the way of feature extraction. In a non-deep model, it is customary to use features such as descriptor vectors and fingerprints calculated from molecular structures. However, the features effective for prediction generally differ depending on what we predict, which requires us to design the features properly. Therefore, much trial and error is required to obtain a good prediction model. On the other hand, deep learning models allow direct input of the molecular representation. Such models can be trained with a dataset to extract features useful for prediction. Hence, we can obtain a general-purpose model and avoid elaborate feature engineering.

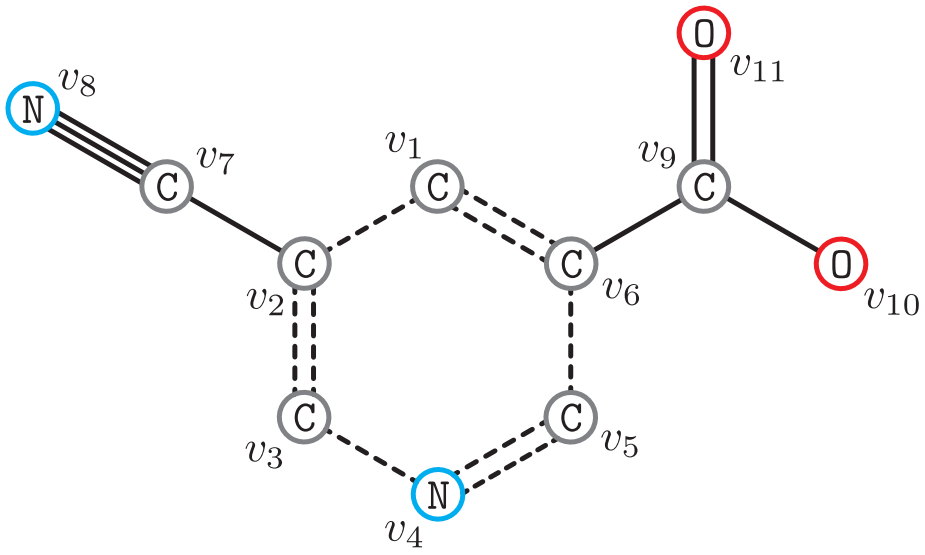

Figure 3.1: An example of a molecular graph. Circles denote vertices of the graph $(v_1, \ldots, v_{11})$ and the characters in the vertices indicate the elements of the corresponding atoms. The segments connecting the vertices are the edges of the graph, which correspond to the bonds between atoms. The number of segments indicates the multiplicity of bonds, and the type of segment indicates the presence or absence of aromaticity.

We can input the molecular structures into a deep learning model by representing them with SMILES strings [68, 69] or molecular graphs. SMILES strings are codes of molecular structures described according to a certain grammar. Because different SMILES strings may represent the same molecular structure, canonicalization of SMILES strings is often applied in preprocessing. In SMILES strings, ring structures and side chains are represented one-dimensionally, and the neighboring relationships of atoms are broken. To successfully extract features of molecular structures from SMILES strings, many models that take SMILES strings as input have been proposed by adopting methods from the field of natural language processing, and have been shown to have good prediction performance [70–73].

Meanwhile, **molecular graphs** are representations of molecular structures in which atoms and bonds correspond to vertices and edges, respectively (Figure 3.1). To molecular graph data, which constitute a molecular

graph with feature vectors attached to each vertex and edge, we can apply a deep neural network called a **graph neural network** (GNN) [74]; good progress has been made using this approach in the past few years. GNNs extract features from graph data by performing operations that do not depend on the ordering of graph vertices. Therefore, the same prediction results can be obtained for the graph data representing the same molecule, while the different prediction results can be obtained for the different SMILES strings representing the same molecule. Also, molecular graphs are expected to be easier to capture the information of molecular structures than SMILES strings because they can explicitly handle the adjacency of atoms. In addition, some studies [75–77] reported that the performance of a GNN in predicting properties of compounds is better than that of other existing feature extraction methods.

In general, as we mentioned in Chapter 1, deep learning models such as GNNs require many model parameters to be adjusted, and therefore, large datasets are needed for high-performance prediction. In the performance analyses of previous studies, datasets containing more than 100,000 samples (e.g., the QM9 dataset [78]) have often been used for training. However, in molecular design, it often happens that only a small number of the chemical property data of interest are available owing to the high cost of synthesis and testing of compounds. In such a situation where there are few available training samples, feature extraction using deep learning models tends to fail owing to overfitting.

Even in the situation that only a few training samples are available, there are cases in which a well-considered training method provides better prediction performance than conventional methods. In transfer learning [79], for example, training on the target dataset can be performed effectively by pre-training on a large dataset other than the target dataset. When using unlabeled molecular structure data obtained from a database such as PubChem [48] or ZINC [49] as a large dataset, the model is expected to acquire knowledge on the nature of the molecular structures, which facilitates training on the target dataset. Several methods for pre-training GNN models have been actively studied in recent years; e.g., self-supervised learning [80, 81] and contrastive learning [82].

Data augmentation [50] is another effective method of training on small datasets. Data augmentation is an operation that increases the apparent number of training samples by randomly deforming the samples. Data augmentation is expected to improve a model's generalization performance even when the training dataset is small, because it extracts essential features that can be successfully predicted even when the samples are slightly deformed. In particular, the advantage of data augmentation is that it does not require any additional dataset other than the training dataset and can thus be easily performed.

Admissible operations for data augmentation are limited to those that do not appreciably degrade the information of the samples. As examples, in image data augmentation, operations such as cropping and adding Gaussian noise to an image are acceptable if the object represented by the image can be correctly identified. However, operations such as filling the entire image with black pixels are inappropriate because we cannot determine the object in the deformed image.

Data augmentation can be performed for deep learning models that take SMILES strings as input by making the best of the multiplicity of the SMILES strings [83]. Although the molecular representation is affected by this augmentation, the corresponding molecular structures themselves are the same, and there is thus almost no loss of information.

Several methods for augmenting graph data have been proposed. One method of graph data augmentation is to create a sample with a slightly altered graph structure [84–88]. However, applying such a method to a molecular graph results in a molecular graph corresponding to a molecule different from the original molecule. Differences in molecular skeletons contribute to the chemical properties, and it is thus inappropriate to modify the graph structure to augment the molecular graph data.

Meanwhile, methods of graph data augmentation in which a **perturbation** is added to all the input vertex feature vectors have also been proposed [89, 90]. These methods are more suitable for the augmentation of molecular graph data because they keep the input molecular graph intact. Although these methods have been reported to improve a model's performance, large datasets with more than 100,000 samples were used to verify

the performance, and the effect of the augmentation of small datasets has not yet been investigated.

In this chapter, we designed a graph data augmentation method that improves the prediction performance of a GNN-based QSPR model for small training datasets comprising approximately 1000 samples. The proposed method deforms samples by perturbating the vertex feature vectors of the graph data rather than by modifying the graph structure. In the proposed method, elements of the perturbation vector are standard normal random numbers. In addition, instead of perturbating all the vertex features input to the GNN, the proposed method perturbates some of the hidden feature vectors in the feature extraction of the GNN.

To verify the effectiveness of the proposed method, we investigated the prediction performance for regression and classification tasks. We also investigated how the prediction performance changes when the perturbation timing is changed. As a result, we confirmed that adding perturbations immediately before the GNN readout operation improves the prediction performance the most. In particular, the data augmentation worked better for the model trained on a smaller dataset. The proposed method is not only versatile enough to be applied to many GNN models but also simple enough to be easily implemented without a large increase in computational complexity.

3.2 Methods

In this section, we first describe the graph data and then outline the GNN operations. We then describe the proposed method in detail.

3.2.1 Graph Data

A graph is a tuple $G = (V, E)$ of a set V of vertices and a set E of edges connecting the vertices. An edge $e \in E$ connecting vertices $u, v \in V$ is denoted by $e = uv$, and u and v are said to be adjacent. The set of vertices adjacent to a vertex v in graph G is called the neighborhood of v and denoted by $N_G(v)$. When graph G is obvious from the context, we simply denote the neighborhood by $N(v)$.

Graph data are the triplets $\mathcal{G} = (G, \boldsymbol{X}_V, \boldsymbol{\Xi}_E)$, where the graph $G = (V, E)$ has the information $\boldsymbol{X}_V = (\boldsymbol{x}_v)_{v \in V}$ for the feature vector $\boldsymbol{x}_v$ of each vertex $v \in V$ and $\boldsymbol{\Xi}_E = (\boldsymbol{\xi}_e)_{e \in E}$ for the feature vector $\boldsymbol{\xi}_e$ of each edge $e \in E$.

The structure data of an organic compound can also be represented as graph data by adding vertex feature vectors and edge feature vectors to the molecular graph. Discrete feature vectors, such as the types of atoms/bonds, the number of bonded hydrogens, and the presence or absence of aromaticity, are often used as feature vectors.

3.2.2 Operations of Graph Neural Networks

In a GNN, two types of operation, namely message passing and readout operations, are performed. After extracting the vertex features of the graph data by repeatedly performing message passing on the input, the readout operation is conducted to produce the feature vector of the graph data itself. Using GNNs, features that combine local information around each vertex of the graph and global information of the whole graph are extracted from the graph data.

Various types of GNN have been proposed, and the detailed operation of GNNs differs depending on the model. In this study, we used a message passing neural network (MPNN) [91] as a base model, which we describe hereafter.

The MPNN calculates the feature vector $\boldsymbol{z}_{\mathcal{G}}$ for the input $\mathcal{G} = (G, \boldsymbol{X}_V, \boldsymbol{\Xi}_E)$ in the following procedure:

(1) Each feature vector is transformed using a fully connected neural network.

(2) Message passing is repeated L times to extract vertex features (where $L \in \mathbb{Z}_{>0}$ is a hyperparameter).

(3) Feature vector $\boldsymbol{z}_{\mathcal{G}}$ is computed by reducing the information of vertex feature vectors with the readout operation.

Transformation of Feature Vectors

First, all feature vectors $\boldsymbol{x}_v$ and $\boldsymbol{\xi}_{vw}$ of the input graph data are transformed into $\boldsymbol{h}_v^{(0)}$ and $\boldsymbol{A}_{vw}$ using the functions f and g, respectively:

$$\boldsymbol{h}_v^{(0)} = f(\boldsymbol{x}_v),$$
$$\boldsymbol{A}_{vw} = g(\boldsymbol{\xi}_{vw}).$$

Here, the functions f and g are fully connected neural networks, $\boldsymbol{h}_v^{(0)}$ is a d-dimensional real vector whereas $\boldsymbol{A}_{vw}$ is a $d \times d$ real square matrix. Note that because the parameters of neural networks generally take real values, the feature vector $\boldsymbol{x}_v$ comprising discrete features is transformed by the function f into the feature vector $\boldsymbol{h}_v^{(0)}$ comprising continuous features.

Message Passing

After the feature transformation, message passing is performed L times using the transformed features. In message passing, two operations, namely aggregation and update operations, are performed alternately to transform the feature vector of each vertex into a feature vector that captures the local information of its neighborhood (Figure 3.2).

In the aggregation operation, for each vertex v of the graph, we compute a message vector that summarizes the feature vectors of vertex v's neighbors. We denote by $\boldsymbol{h}_v^{(\ell)}$ $(\ell = 0, 1, \ldots, L)$ vertex v' s feature vector after ℓ message passing operations. In the aggregation operation of the ℓ-th message passing, we compute the message vector $\boldsymbol{m}_v^{(\ell)}$ as

$$\boldsymbol{m}_v^{(\ell)} = \sum_{w \in N(v)} \boldsymbol{A}_{vw} \boldsymbol{h}_v^{(\ell-1)} \quad (\ell = 1, 2, \ldots, L).$$

We consider from how the message vector is calculated that $\boldsymbol{m}_v^{(\ell)}$ contains the information of the current feature vectors of each vertex w adjacent to v, $\boldsymbol{h}_w^{(\ell-1)}$, and the information of the incident edge vw, $\boldsymbol{A}_{vw}$.

In the following update operation, the current feature vector $\boldsymbol{h}_v^{(\ell-1)}$ of each vertex v is updated to $\boldsymbol{h}_v^{(\ell)}$ using the message vector $\boldsymbol{m}_v^{(\ell)}$ calculated above. Specifically, we use a recurrent neural network called the gated recurrent unit (GRU) [92]:

$$\boldsymbol{h}_v^{(\ell)} = \mathrm{GRU}\left(\boldsymbol{m}_v^{(\ell)}; \boldsymbol{h}_v^{(\ell-1)}\right) \quad (\ell = 1, 2, \ldots, L).$$

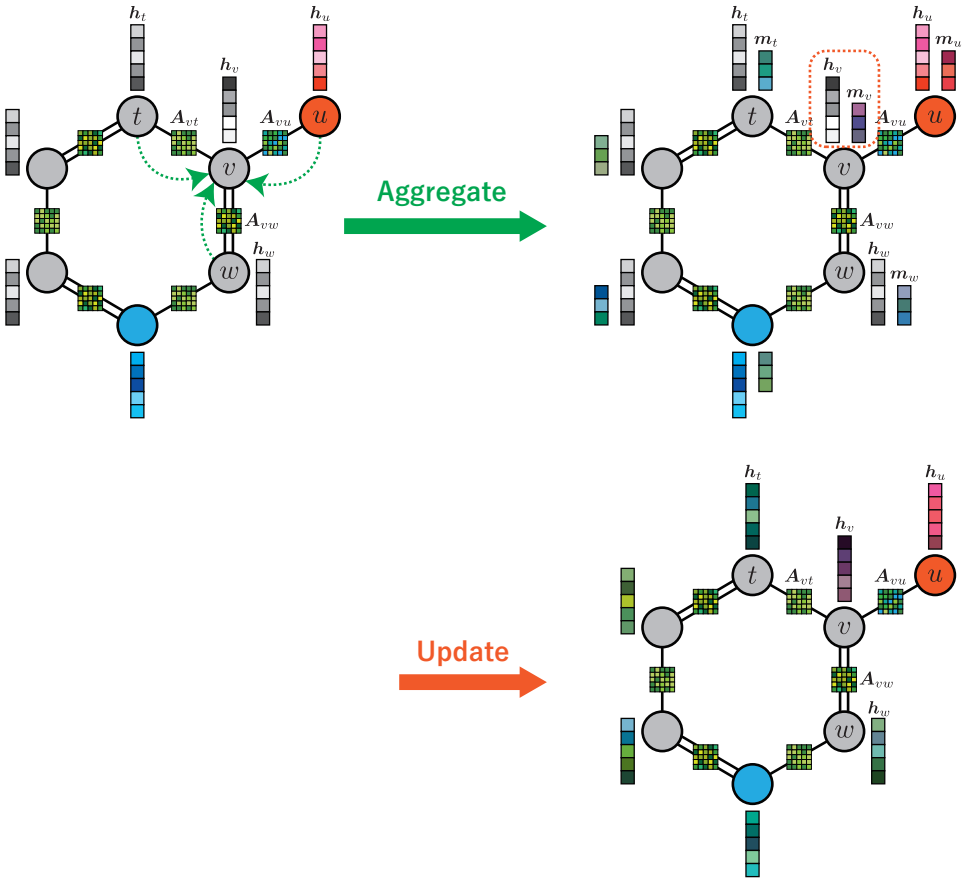

Figure 3.2: Message passing operation in the MPNN. Colors of the vertices indicate the atom types. Stacked squares near vertices indicate vertex feature vectors whereas arrayed squares on edges indicate edge feature vectors. We here focus on the operation for the vertex v. In aggregation, feature vectors of the vertices/edges adjacent/incident to vertex v are used to create the message vector for v. In the update, the message vector and the current vertex feature are used to update the feature vector of vertex v.

Here, the second argument $\boldsymbol{h}_v^{(\ell-1)}$ is a vector representing the internal state of the GRU. We note that parameter-shared GRUs are assigned to each vertex v and update operation is applied independently to the vertices. Because the the GRU can capture the evolution of each vertex feature, $\boldsymbol{h}_v^{(\ell)}$ contains information of the feature vector of vertex v up to the $(\ell-1)$-st message passing. The above message passing operation is repeated L times to calculate the final vertex feature vector $\boldsymbol{h}_v^{(L)}$.

Readout

Finally, in the readout operation, all the final vertex feature vectors computed by message passing are combined and transformed into a single feature vector $\boldsymbol{z}_{\mathcal{G}}$, namely the graph feature vector for the input $\mathcal{G}$ (Figure 3.3). Specifically, we use a neural network called Set2Set [93]:

$$\boldsymbol{z}_{\mathcal{G}} = \textsc{Set2Set}\left(\{\, \boldsymbol{h}_v^{(L)} \mid v \in V \,\}\right).$$

Set2Set can extract feature from the input set of feature vectors, i.e., the extracted feature is invariant with respect to the vertex ordering. Set2Set uses the input set to update the initial vector $\boldsymbol{q}_0^*$, which is set to zero vector, and after K times update it outputs the updated vector $\boldsymbol{q}_K^*$ as the graph feature $\boldsymbol{z}_{\mathcal{G}}$. In the k-th update of Set2Set, the vector $\boldsymbol{q}_k^*$ is calculated as follows:

$$\begin{aligned}
\boldsymbol{q}_k, \boldsymbol{\eta}_k &= \mathrm{LSTM}(\boldsymbol{q}_{k-1}^*; \boldsymbol{\eta}_{k-1}), \\
\alpha_{v,k} &= \mathrm{softmax}(\boldsymbol{h}_v^{(L)} \cdot \boldsymbol{q}_k) \quad (v \in V), \\
\boldsymbol{r}_k &= \sum_{v \in V} \alpha_{v,k} \boldsymbol{h}_v^{(L)}, \\
\boldsymbol{q}_k^* &= \boldsymbol{q}_k \parallel \boldsymbol{r}_k.
\end{aligned}$$

Here, LSTM is a recurrent neural network called the long short-term memory (LSTM) [94] and $\boldsymbol{\eta}_k$ is the vector representing the internal state of the LSTM ($\boldsymbol{\eta}_0 = \boldsymbol{0}$). Also, the binary operation $\parallel$ denotes vector concatenation. We note that $\alpha_{v,k}$ represents the contribution of vector $\boldsymbol{h}_v^{(L)}$, and the vector $\boldsymbol{r}_k$ summarizes the vertex feature vectors by taking the weighted sum based on this contribution, the operation that does not depend on the vertex ordering.

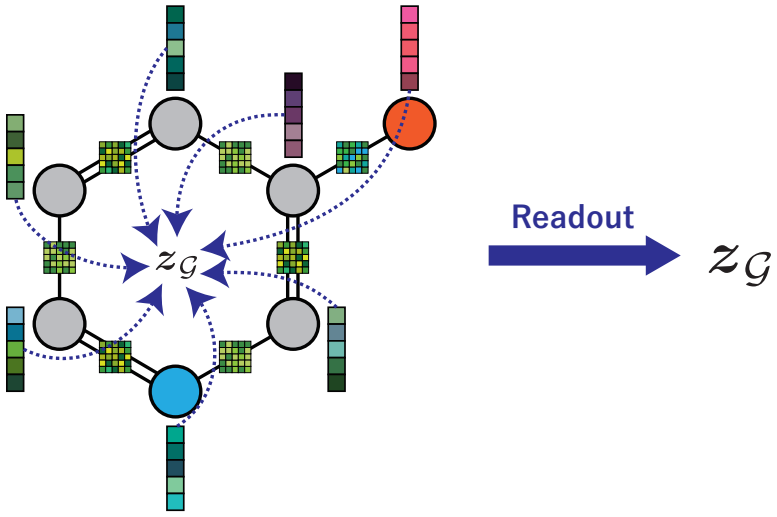

Figure 3.3: Readout operation in the MPNN. All the current features for vertices are used to compute the feature vector for the input $\mathcal{G}$

3.2.3 Proposed method: Perturbating MPNN

In the augmentation of molecular graph data, it is necessary to handle the discrete aspects of the graph structure and feature vectors. One method of augmenting graph data is to change the graph structure slightly (Figure 3.4 (a)), but as mentioned in the introduction, this method is inappropriate for molecular graph data because it changes the compounds represented by the molecular graph. Meanwhile, perturbating the vertex/edge feature vectors is also effective in augmenting the graph data. However, because the vertex/edge feature vectors for molecular graphs often include discrete features such as binary features like one-hot vectors and features with non-negative integer values, direct modification of these feature vectors may corrupt the information of the feature vectors (Figure 3.4 (b)). As an example, if the element corresponding to the atom type in the vertex feature vector is changed, the modified graph data may correspond to a different molecule with a different atom type or an inappropriate structure that does not satisfy the valence constraint. We need to avoid the above discreteness in molecular graph data.

We here focus on the operation of the MPNN. In procedure (1) of the MPNN, the feature vectors are transformed by the neural network, such

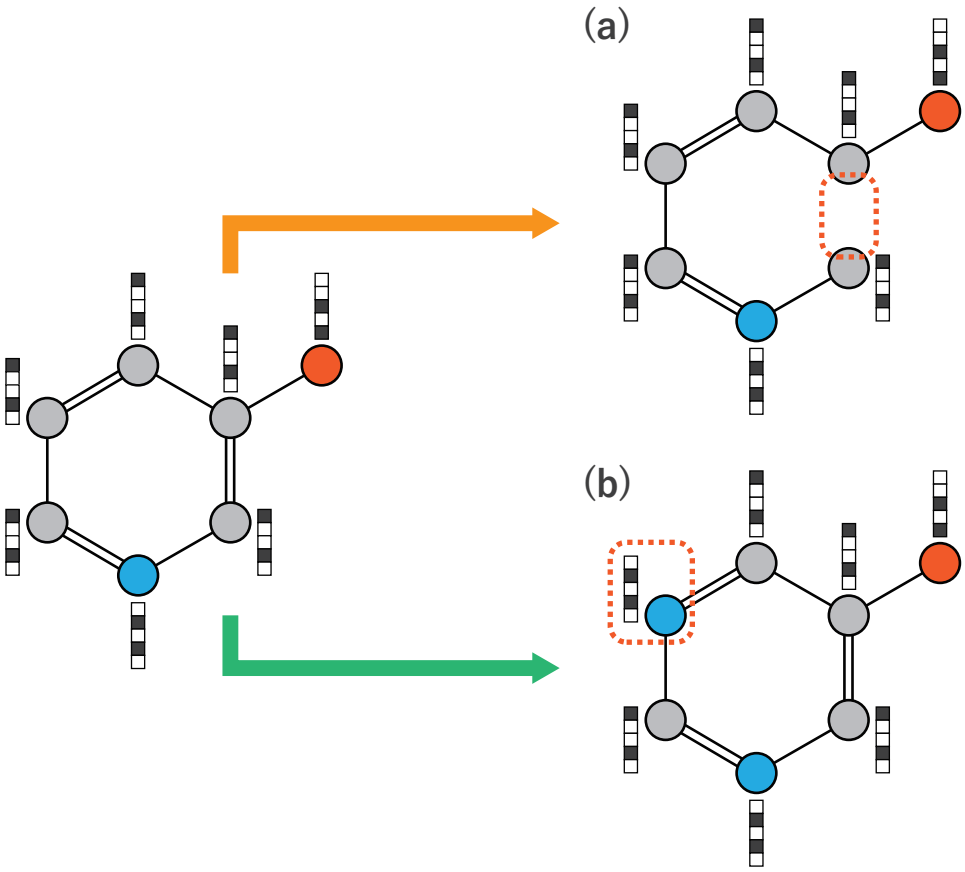

Figure 3.4: Modifications that are inappropriate for the augmentation of molecular graph data. Colors of vertices indicate the atom types. Stacked squares indicate discrete feature vectors with each element being an integer value. The modified part is circled by a red dotted line. (a) Change in the graph structure. This operation results in a different molecular graph. (b) Change in a discrete vertex feature. This operation may also result in a different molecular graph or an invalid structure.

that each element of the feature vectors is a continuous value. Similarly, the vertex feature vectors in message passing and the graph feature vectors obtained by the readout operation are also vectors with continuous values. For such continuous-valued vectors, it is expected that the information of the original data can be more easily preserved even if random changes are made to each vector. In fact, in a previous study [89], graph data were augmented by adding perturbations to the vertex feature vectors immediately after the input vertex feature vectors were transformed by the neural network; i.e., $\boldsymbol{h}_v^{(0)}$ $(v \in V)$.

In addition, if all vertex feature vectors are perturbated, the graph data may be deformed such that they do not retain the features of the original data. It is therefore desirable to control the total perturbation applied to the graph data by perturbating only some of the vertex features.

We therefore designed the perturbating MPNN (PMPNN) as a GNN model that performs graph data augmentation (Figure 3.5). The PMPNN calculates the feature vector $\boldsymbol{z}_{\mathcal{G}}$ for the input graph data $\mathcal{G} = (G, \boldsymbol{X}_V, \boldsymbol{\Xi}_E)$ in the following procedure:

(1) Each feature vector is transformed using a fully connected neural network.

(2) Message passing is repeated ℓ_{pre} times (where $\ell_{\text{pre}} \in \mathbb{Z}_{\geq 0}$ is a hyperparameter).

(3) For each vertex feature, a perturbation vector of normal random values with probability p (where $0 < p \leq 1$ is a hyperparameter) is sampled and added, with each element of the perturbation having a mean of zero and variance of 1.

(4) Message passing is repeated ℓ_{post} times to extract vertex features (where $\ell_{\text{post}} \in \mathbb{Z}_{\geq 0}$ is a hyperparameter satisfying $\ell_{\text{pre}} + \ell_{\text{post}} > 0$).

(5) Feature vector $\boldsymbol{z}_{\mathcal{G}}$ is computed by reducing the information of vertex feature vectors with the readout operation.

In procedure (3) of the PMPNN, we sample a binary value $b_v \in \{0, 1\}$ $(v \in V)$ from the Bernoulli distribution $\mathrm{B}(1, p)$ of the parameter p and

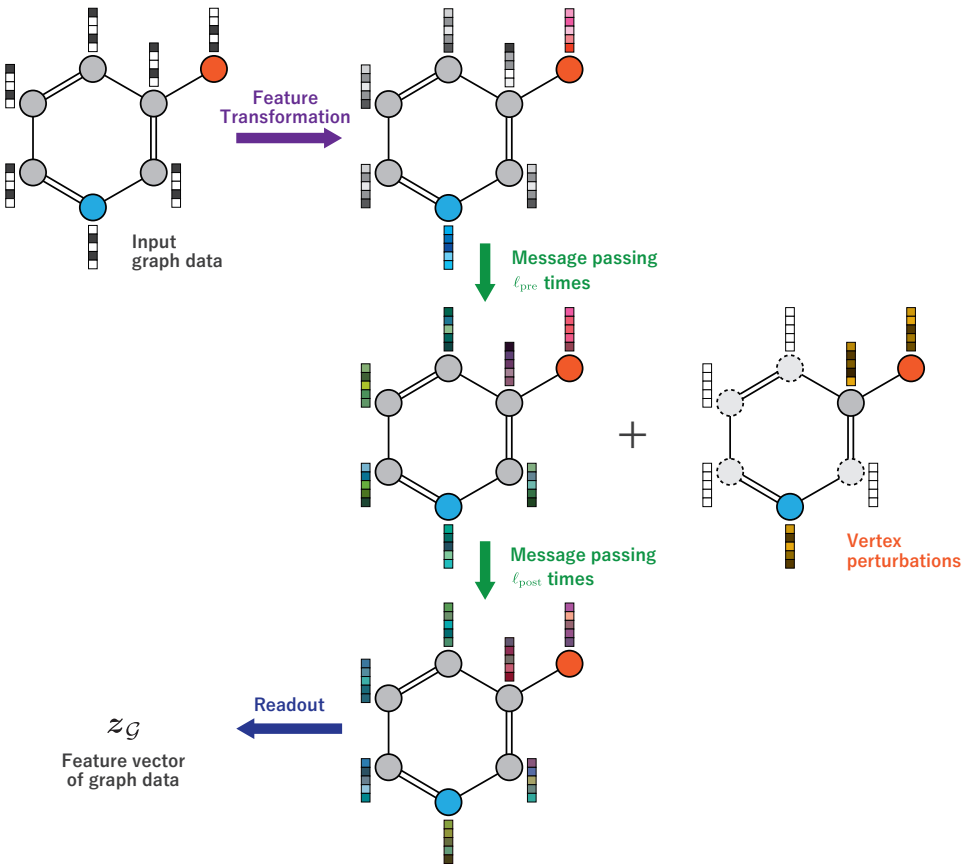

Figure 3.5: Overview of the PMPNN. Stacked squares indicate feature vectors. After ℓ_{pre} message passings, some atoms in the molecular graph are selected with probability p (where selected atoms are depicted with solid lines), and normally distributed perturbations are then sampled and added to the feature vectors.

a perturbation vector $\boldsymbol{\pi}_v \in \mathbb{R}^d$ $(v \in V)$ from the d-dimensional standard normal distribution $\mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$, and add $b_v \boldsymbol{\pi}_v$ to the vertex feature after the ℓ_{pre}-th update:

$$\begin{aligned} \boldsymbol{h}_v^{(\ell_{\text{pre}})} &= \text{GRU}\left(\boldsymbol{m}_v^{(\ell)}; \boldsymbol{h}_v^{(\ell-1)}\right) + b_v \boldsymbol{\pi}_v, \\ b_v &\sim \text{B}(1, p), \\ \boldsymbol{\pi}_v &\sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d). \end{aligned}$$

Here, $\boldsymbol{I}_d$ is the $d \times d$ identity matrix. Note that the time required for these operations is sufficiently short relative to the time required for training.

In the PMPNN, we adjust the timing of perturbating the feature vectors by fixing the number of times the message passing is applied, $L = \ell_{\text{pre}} + \ell_{\text{post}}$, and varying the hyperparameters ℓ_{pre} and ℓ_{post}. The hyperparameter p controls the total number of perturbations to be added to the graph data; i.e., for a graph with n vertices, the perturbations are added to np vertices in expectation. In particular, at the limit that $p \to 0$, the PMPNN is consistent with the MPNN with $L = \ell_{\text{pre}} + \ell_{\text{post}}$, and the PMPNN is thus an extended model of the MPNN.

3.3 Results and Discussion

To verify the effect of the proposed data augmentation method, we conducted comparison experiments with the base model, the MPNN, for two types of task: regression and classification. We confirmed the effects of the timing of the perturbation and the number of perturbations on the prediction performance. We also investigated the effect of data augmentation for different sizes of the training dataset.

We performed the experiment using Python 3.7.3. We used `RDKit` (version 2019.03.2.0) [58] to handle the organic compound data and `PyTorch` (version 1.7.0) [95], `DGL` (version 0.5.3) [96], and `DGL-LifeSci` (version 0.2.6) [97] as deep learning frameworks.

3.3.1 Datasets

All datasets used in this study were obtained from MoleculeNet [98]. The original datasets contained molecules that have atoms with incorrect valence, mixtures, and molecules with deuterium, which were removed in the preprocessing step.

Dataset Preparation for the Regression Task

For the regression task, we used 133,018 structures with calculated values of the gap between the highest occupied molecular orbital and the lowest unoccupied molecular orbital from the QM9 dataset [78] of 133,885 structures. This dataset contains a comprehensive set of structures with nine or fewer carbon, oxygen, nitrogen, and fluorine atoms and is used in many property prediction tasks.

To train on a small dataset, we created a training dataset, a validation dataset, and a test dataset in the following steps. First, 26,604 samples (20% of the total samples) were held as the test dataset for evaluating the prediction performance. For the remaining dataset comprising 106,414 samples, we repeated the operation of sampling half of the samples at random and created datasets comprising 53,207, 26,603, 13,301, 6650, 3325, 1662, and 831 samples. Each of the above eight datasets was split so that the sample ratio of the training dataset to the validation dataset was 8:2. For the experiments in the first part of Section 3.3.3, we trained the model using the datasets with 1662 samples, and for the experiments in the final part of Section 3.3.3, we trained the model with each dataset.

Dataset Preparation for the Classification Task

Meanwhile, for the classification task, we used 301,644 structures (active: 62,577, inactive: 239,067) with human TDP1 inhibitor data from the PCBA-686978 dataset of 302,175 structures. We select this dataset because it contains the most active data among the PCBA dataset.

As in the regression task, to train on a small dataset, we created the training, validation, and test datasets using the following procedure. To maintain the proportion of classes, we divided the data through stratified

sampling. First, we held 60,329 samples (active: 12,515, inactive: 47,814), which is 20% of the total samples, as the test dataset for evaluating the prediction performance. We next undersampled the inactive classes to get the same class ratio for the remaining 241,315 samples and created a dataset comprising 100,124 samples in total, including 50,062 active and 50,062 inactive samples. We repeated the operation of sampling half of the samples through stratified sampling and created datasets comprising 50,062, 25,031, 12,515, 6257, 3128, 1564, and 782 samples. The above eight datasets were divided so that the sample ratio of the training dataset to the validation dataset was 8:2. For the experiments in the first part of Section 3.3.3, we trained the model using the dataset comprising 1564 samples, and for the experiments in the final part of Section 3.3.3, we trained the model using each dataset.

Also, we used 5,673 structures (active: 911, inactive: 4,762) from the Tox21 dataset (SR-ARE) for the classification task. To train on a small dataset, we created the training, validation, and test datasets using the following procedure. To maintain the proportion of classes, we divided the data through stratified sampling. First, we held 2,270 samples (active: 365, inactive: 1,905), which is 40% of the total samples, as the test dataset for evaluating the prediction performance. We next undersampled the inactive classes to get the same class ratio for the remaining 3,403 samples and created a dataset comprising 1,092 samples in total, including 546 active and 546 inactive samples. This dataset was divided so that the sample ratio of the training dataset to the validation dataset was 8:2. We trained the model using this dataset for the experiments in the first and second part of Section 3.3.3.

In constructing the graph data, we used an 80-dimensional vertex feature vector comprising the following elements:

- Type of atom,
- Vertex degree,
- Number of bonded hydrogens,

- Formal charge,
- Number of radicals,
- Type of hybrid orbital, and
- Presence or absence of aromaticity.

As the edge feature vectors, we used a 12-dimensional feature vector comprising the following elements:

- Type of bond,
- Whether the bond is conjugated or not
- Whether the bond is in a ring or not, and
- Stereo configuration.

3.3.2 Models and Experimental Conditions

Network Architectures

To perform the regression and classification tasks, we used a network structure in which the graph data are passed through the MPNN or PMPNN for feature extraction and then passed through a two-layer fully connected neural network. All the activation functions used in the neural networks were ReLU functions. The detailed model architecture is shown in Figure 3.6.

To transform the vertex feature vectors of the MPNN and PMPNN, we used a one-layer fully connected neural network that outputs 64-dimensional vectors. For the transformation of the edge feature vectors, we used a two-layer fully connected neural network that outputs a 64×64 square matrix, and the dimension of the hidden layer of this network was set at 128. In Set2Set, which is used for the readout of both MPNN and PMPNN, the number of updates K was set at six, the number of LSTM layers at three, and the dimension of the graph feature vector at 128. In the network for regression and classification, we set the dimension of the hidden layer at 64 and that of the output at 1.

As a base model, we used the MPNN with four message passings. To make the number of message passings the same as that for the MPNN, we

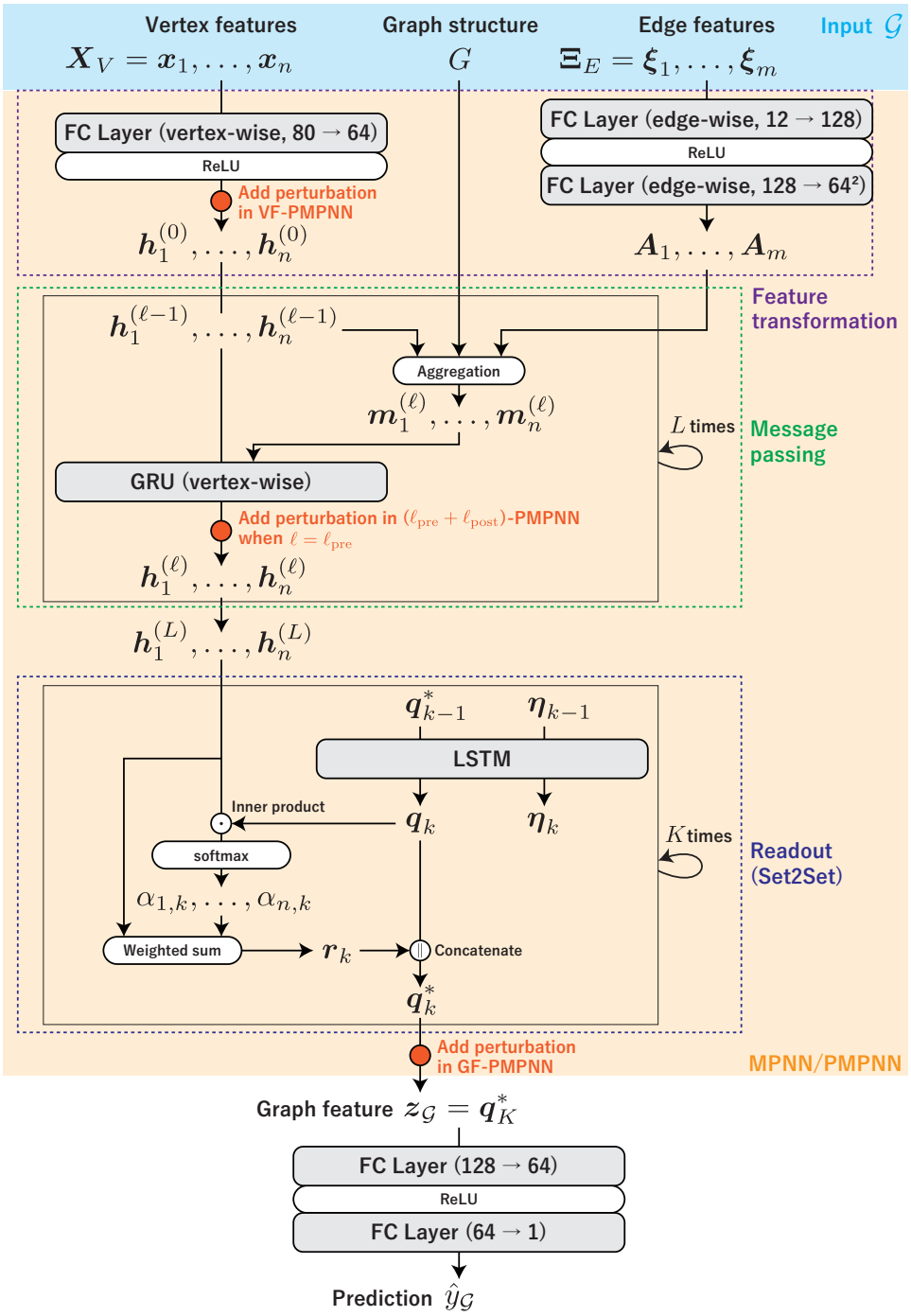

Figure 3.6: The architecture of the model we used in the case study. FC layer stands for fully connected layer and the numbers in parentheses indicate the dimensions of the input and output. Gray rounded squares indicate the presence of learning parameters. In each PMPNN, perturbation is applied at the position indicated by the red circle.

set the hyperparameter of the PMPNN as $L = \ell_{\text{pre}} + \ell_{\text{post}} = 4$. In the following, we refer to the PMPNN with hyperparameters ℓ_{pre} and ℓ_{post} as the $(\ell_{\text{pre}} + \ell_{\text{post}})$-PMPNN.

To consider the timing of perturbation addition, we also created two types of variant in the experiments described in the first part of Section 3.3.3: the VF-PMPNN, a model that adds perturbation directly to the input vertex feature vectors, and the GF-PMPNN, a model that adds perturbation to the graph feature vectors output by the MPNN. The number of message passings is again set at four for these models.

Traditional Model

For the comparison in the first part of Section 3.3.3, we also created the non-deep QSPR models. We used the Random Forest (RF) [99] for the model and the Morgan Fingerprint [66] (2,048 bits, radius 2) for feature extraction. Among the hyperparameters of the Random Forest, the number of decision trees and the maximum depth were optimized by grid search using the training and validation datasets. The search range of the hyperparameters was as follows:

- The number of decision trees: $\{\, 50 \times i \mid i = 1, \ldots, 6 \,\}$,
- The maximum depth of trees: $\{\, 5 \times i \mid i = 1, \ldots, 16 \,\}$.

After the tuning of the hyperparameters, we trained the Random Forest with the optimized hyperparameters on the training dataset only, and evaluated its prediction performance on the test dataset.

Loss Functions

As the loss function to be minimized in training, we used the mean square error (MSE) in the regression task and the mean of the binary cross-entropy loss weighted by the class ratio in the classification task. For a dataset of N samples $\mathscr{D} = \{\, (\boldsymbol{x}_i, y_i) \,\}_{i=1}^{N}$ (where $\boldsymbol{x}_i$ is i-th sample and y_i is the label for the sample $\boldsymbol{x}_i$), the mean squared error of the regression model f for $\mathscr{D}$ is

defined by

$$\mathrm{MSE}_{\mathscr{D}}[f] = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(\boldsymbol{x}_i))^2$$

and the mean weighted binary cross-entropy loss (CEL) of the classification model f for the dataset $\mathscr{D}$ is

$$\mathrm{CEL}_{\mathscr{D}}[f] = -\frac{1}{N}\sum_{i=1}^{N}\left(w_1 y_i \log f(\boldsymbol{x}_i) + w_0(1-y_i)\log(1-f(\boldsymbol{x}_i))\right),$$

where $y_i \in \{0, 1\}$ and $f(\boldsymbol{x}) \in [0, 1]$. In this experiment, the hyperparameters w_0 and w_1, which represent the weights of classes 0 and 1, were set to account for the slight bias in the class distribution of the training dataset:

$$w_i \coloneqq \frac{N_i}{\max(N_0, N_1)} \quad (i = 0, 1),$$

where N_i is the number of samples of class i in the training dataset. Note that $w_i \approx 1$ because of the application of undersampling in the classification task.

Experimental Conditions

We used Adam [100] with a learning rate of 0.0001 as the optimization algorithm for the loss function. The training batch size was set at 64. The maximum number of training epochs was set at 500, and early stopping [101] was adopted. We set the patience of early stopping at 200 in the regression task and 50 in the classification task. Data augmentation with the PMPNN was used only for training, and perturbations were set as zero vectors in the prediction for the validation and test datasets.

Metrics for Evaluation

To evaluate the prediction performance of the model, we used the mean absolute error (MAE) and the coefficient of determination R^2 for the test dataset in the regression task and the area under the curve of receiver operating characteristic (ROC-AUC) for the test dataset in the classification task. ROC-AUC and R^2 closer to 1 and an MAE closer to zero indicate better model prediction. We performed 10 trials in the following experiments.

Table 3.1: MAE (mean±standard deviation) for the test dataset in the regression task. The background color indicates the improvement of the evaluation metric over the base model (MPNN). Bold type indicates the best evaluation metric.

	No Perturbation	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 1.0$
RF	0.4511 ± 0.0015	—	—	—	—
MPNN	0.3334 ± 0.0188	—	—	—	—
VF-PMPNN	—	0.3471 ± 0.0065	0.3733 ± 0.0073	0.4106 ± 0.0121	0.4459 ± 0.0098
(0+4)-PMPNN	—	0.3593 ± 0.0045	0.3672 ± 0.0051	0.3888 ± 0.0124	0.4057 ± 0.0158
(1+3)-PMPNN	—	0.3371 ± 0.0064	0.3359 ± 0.0074	0.3421 ± 0.0068	0.3440 ± 0.0141
(2+2)-PMPNN	—	0.3245 ± 0.0090	0.3230 ± 0.0063	0.3243 ± 0.0081	0.3271 ± 0.0064
(3+1)-PMPNN	—	0.3090 ± 0.0062	0.3084 ± 0.0077	0.3072 ± 0.0075	0.3082 ± 0.0063
(4+0)-PMPNN	—	$\mathbf{0.3070} \pm 0.0082$	0.3099 ± 0.0081	0.3110 ± 0.0063	0.3164 ± 0.0080
GF-PMPNN	—	0.3400 ± 0.0104	0.3550 ± 0.0147	0.3594 ± 0.0136	0.3637 ± 0.0177

In each trial, we trained a model and computed the metrics described above. We calculated the mean and standard deviation of the metrics after completing all the trials.

3.3.3 Performances of Models

Effect of Perturbations on the Prediction Performance

We first investigated whether the proposed data augmentation performs well. Table 3.1 and 3.2 give the evaluation metrics for the regression task whereas Table 3.3 and 3.4 give the evaluation metrics for the two different classification tasks when the hyperparameters $(\ell_{\text{pre}}, \ell_{\text{post}})$ and p are varied.

The performances of (3+1)-PMPNN and (4+0)-PMPNN exceed the performance of the base model, indicating that the proposed data augmentation method works well in both regression and classification tasks. In all cases, the prediction performance tends to improve as ℓ_{post} becomes small. In particular, (4+0)-PMPNN, which includes a perturbation immediately before readout, has the highest prediction performance, which may be because the message passing did not amplify the information loss caused by the perturbation. The perturbation added to the feature vector of some vertex is propagated to the feature vectors of its neighboring vertices in the subsequent message passing. Therefore, a smaller number of message passings after the perturbation results in the perturbation effect being less spread,

Table 3.2: The coefficient of determination R^2 (mean±standard deviation) for the test dataset in the regression task. The background color indicates the improvement of the evaluation metric over the base model (MPNN). Bold type indicates the best evaluation metric.

	No Perturbation	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 1.0$
RF	0.7502 ± 0.0015	—	—	—	—
MPNN	0.8597 ± 0.0186	—	—	—	—
VF-PMPNN	—	0.8439 ± 0.0053	0.8214 ± 0.0076	0.7842 ± 0.0119	0.7522 ± 0.0107
(0+4)-PMPNN	—	0.8313 ± 0.0050	0.8251 ± 0.0056	0.8052 ± 0.0122	0.7917 ± 0.0152
(1+3)-PMPNN	—	0.8560 ± 0.0061	0.8589 ± 0.0082	0.8527 ± 0.0067	0.8505 ± 0.0138
(2+2)-PMPNN	—	0.8683 ± 0.0095	0.8707 ± 0.0062	0.8683 ± 0.0080	0.8636 ± 0.0050
(3+1)-PMPNN	—	0.8818 ± 0.0060	0.8833 ± 0.0067	0.8830 ± 0.0072	0.8823 ± 0.0070
(4+0)-PMPNN	—	$\mathbf{0.8868} \pm 0.0074$	0.8850 ± 0.0055	0.8850 ± 0.0052	0.8815 ± 0.0057
GF-PMPNN	—	0.8548 ± 0.0103	0.8418 ± 0.0131	0.8391 ± 0.0128	0.8380 ± 0.0142

Table 3.3: ROC-AUC (mean±standard deviation) for the test dataset in the classification task (PCBA). The background color indicates the improvement of the evaluation metric over the base model (MPNN). Bold type indicates the best evaluation metric.

	No Perturbation	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 1.0$
RF	0.7116 ± 0.0040	—	—	—	—
MPNN	0.7408 ± 0.0107	—	—	—	—
VF-PMPNN	—	0.7037 ± 0.0083	0.6830 ± 0.0128	0.6637 ± 0.0107	0.6673 ± 0.0060
(0+4)-PMPNN	—	0.6838 ± 0.0141	0.6656 ± 0.0056	0.6655 ± 0.0067	0.6641 ± 0.0054
(1+3)-PMPNN	—	0.7182 ± 0.0099	0.7194 ± 0.0183	0.7023 ± 0.0205	0.7187 ± 0.0147
(2+2)-PMPNN	—	0.7372 ± 0.0069	0.7329 ± 0.0067	0.7242 ± 0.0181	0.7343 ± 0.0040
(3+1)-PMPNN	—	0.7472 ± 0.0055	0.7405 ± 0.0078	0.7395 ± 0.0175	0.7419 ± 0.0083
(4+0)-PMPNN	—	0.7494 ± 0.0066	$\mathbf{0.7550} \pm 0.0038$	0.7535 ± 0.0060	0.7524 ± 0.0061
GF-PMPNN	—	0.7370 ± 0.0073	0.7263 ± 0.0233	0.7303 ± 0.0165	0.7264 ± 0.0250

Table 3.4: ROC-AUC (mean±standard deviation) for the test dataset in the classification task (Tox21). The background color indicates the improvement of the evaluation metric over the base model (MPNN). Bold type indicates the best evaluation metric.

	No Perturbation	$p = 0.25$	$p = 0.5$	$p = 0.75$	$p = 1.0$
RF	$\mathbf{0.7609} \pm 0.0031$	—	—	—	—
MPNN	0.6955 ± 0.0332	—	—	—	—
VF-PMPNN	—	0.6854 ± 0.0204	0.6882 ± 0.0089	0.6761 ± 0.0128	0.6528 ± 0.0131
(0+4)-PMPNN	—	0.6771 ± 0.0154	0.6894 ± 0.0170	0.6852 ± 0.0107	0.6804 ± 0.0080
(1+3)-PMPNN	—	0.6846 ± 0.0263	0.6716 ± 0.0134	0.6638 ± 0.0150	0.6712 ± 0.0181
(2+2)-PMPNN	—	0.6950 ± 0.0411	0.6740 ± 0.0320	0.6893 ± 0.0322	0.6586 ± 0.0129
(3+1)-PMPNN	—	0.7065 ± 0.0402	0.7169 ± 0.0372	0.6867 ± 0.0409	0.6815 ± 0.0381
(4+0)-PMPNN	—	0.7400 ± 0.0247	0.7345 ± 0.0344	0.7149 ± 0.0385	0.7340 ± 0.0346
GF-PMPNN	—	0.7143 ± 0.0332	0.7241 ± 0.0270	0.6956 ± 0.0439	0.7045 ± 0.0269

and the vertex features can thus be extracted from the original graph data without appreciable information loss. This result is consistent with the tendency that the prediction performance is improved when ℓ_{post} is small.

It is confirmed that the prediction performance of the VF-PMPNN is worse than that of the base model, which verifies that it is unreasonable to augment the data by directly perturbating the discrete input vertex features. Moreover, the prediction performance of the GF-PMPNN is also lower than that of the (4+0)-PMPNN, suggesting that it is desirable to perturbate the vertex feature vectors when augmenting the data by adding perturbations.

We note here that the setting of the probability p for the best prediction performance will depend mainly on the dataset, as shown by the fact that p for the best prediction performance differs depending on the tasks. We suppose that this is because the total amount of perturbation applied to a sample depends on the number of atoms in the molecule.

On the QM9 and PCBA datasets, the prediction performance of MPNN was better than that of Random Forest, but on the Tox21 dataset, the prediction performance of Random Forest was the best. This result suggests that the prediction performance of PMPNN could fall short of that of the traditional QSPR model depending on the dataset, although the prediction performance of MPNN was indeed improved by the data augmentation. In order to further improve the prediction performance, it will be necessary to use other training methods on small datasets such as transfer learning.

Effect of Perturbation Probability

To investigate the effect of the perturbation probability p on the prediction performance in more detail, we performed predictions on the Tox21 dataset by varying the p of the (4+0)-PMPNN in increments of 0.05. The results are shown in Figure 3.7.

For all values of the probability p of selecting vertices, the mean ROC-AUC of the (4+0)-PMPNN was improved over that of the base model (MPNN). In particular, the mean ROC-AUC was greatest when $p = 0.85$, and the improvement from the base model was confirmed even when the standard deviation was taken into account. This suggests that the pre-

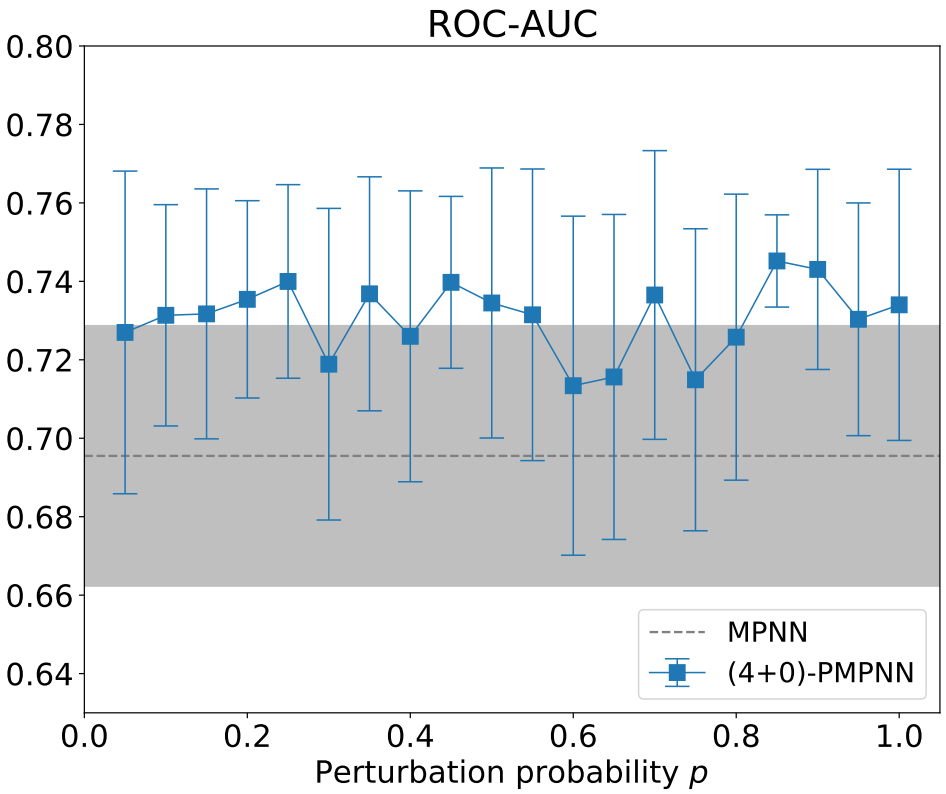

Figure 3.7: Variation in the ROC-AUC with the perturbation probability in the classification task for the Tox21 dataset. Data are averages for 10 trials. The error bars indicate the standard deviation. The grey area indicates the range where the difference from the mean of the baseline model falls within its standard deviation.

diction performance can be improved by partially perturbating the vertex features.

However, no clear relationship was found between the value of p and the prediction performance. It is thus desirable to perform the hyperparameter tuning of p through Bayesian optimization or other methods to achieve better prediction performance.

Data Augmentation Effect According to the Number of Training Samples

We then checked the effect of data augmentation when the number of training samples was varied. We fixed the models to the base model and (4+0)-PMPNN ($p = 0.25$) and compared the prediction performances of these models. The evaluation metrics for the regression task and the classification task are plotted in Figures 3.8 and 3.9, respectively.

It is confirmed that the mean prediction performance of (4+0)-PMPNN is better than that of the base model, except for the datasets with 53,207 and 106,414 samples in the regression task. In particular, the average effect of data augmentation tend to be greater in average when the number of training samples was approximately 1000, suggesting that the proposed method successfully extracts features from a small number of graph data.

However, when the number of training samples was smaller, the standard deviation of the prediction performance tended to be larger. When the standard deviation is large, the prediction performance might be worse than that of the base model. In such a situation, it is better to compensate for the variation in prediction by building an ensemble of several PMPNNs.

The degradation of the prediction performance on the two datasets of the regression task can be attributed to the poor setting of the hyperparameter p for the two datasets and the consequent large deformation due to the perturbation. It is thus expected that the prediction performance can be improved by adjusting the hyperparameter p appropriately.

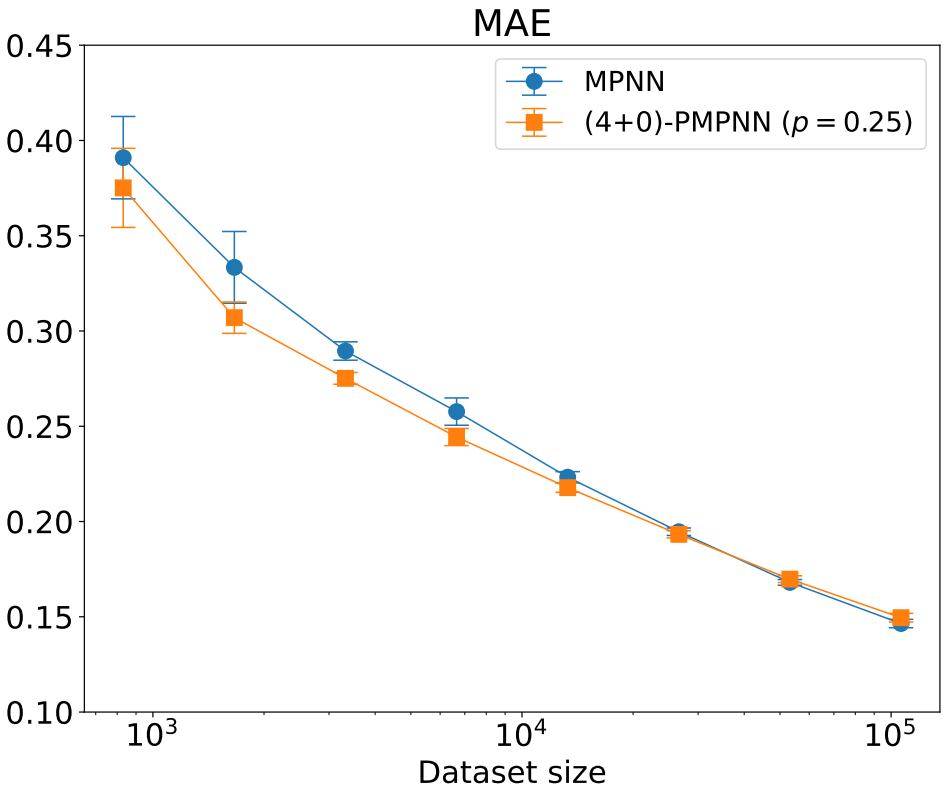

Figure 3.8: Variation in the MAE with the number of training samples in the regression task. Data are averages for 10 trials. The error bars indicate the standard deviation.

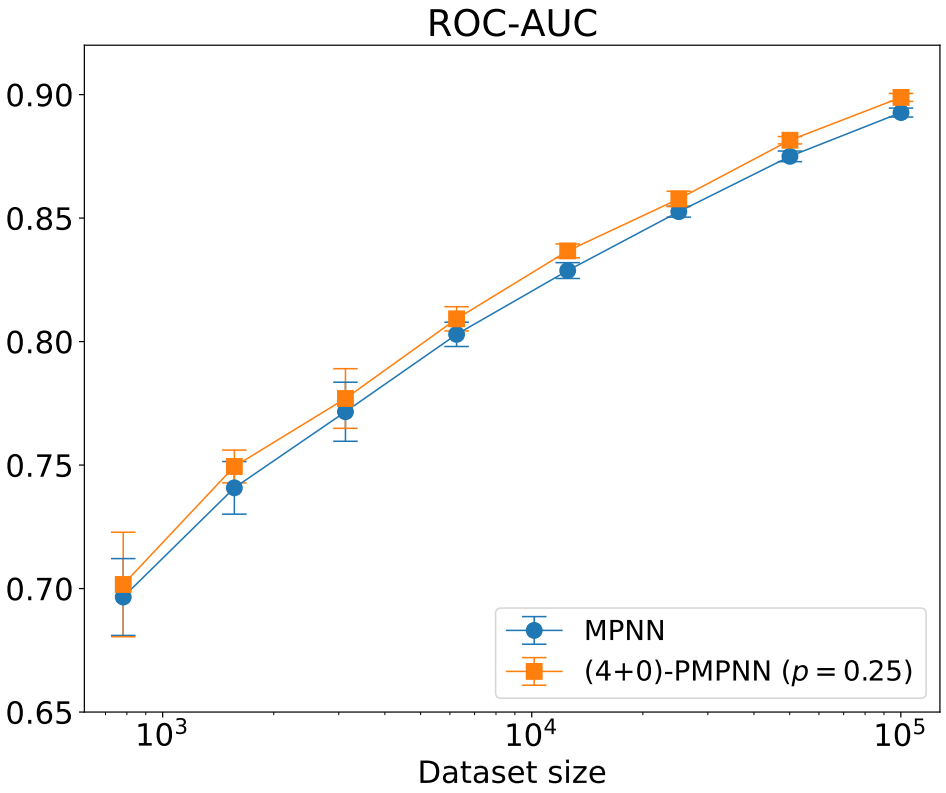

Figure 3.9: Variation in the ROC-AUC with the number of training samples in the classification task (PCBA). Data are averages for 10 trials. The error bars indicate the standard deviation.

3.4 Conclusion

In this study, we designed a data augmentation method, namely the PMPNN, for feature extraction from a small-scale graph dataset of approximately 1000 samples. In the PMPNN, perturbation vectors comprising normal random numbers with a mean of zero and variance of 1 are added to the vertex feature vectors during message passing of the MPNN, and the computational complexity is almost the same as that of the MPNN. The degree of perturbation of the samples is adjusted by introducing a hyperparameter p that represents the perturbation probability.

We confirmed that the proposed method can successfully augment graph data by comparing the proposed method with the base model, namely the MPNN, for both regression and classification tasks. In particular, we found that the perturbation just before readout enhances the effect of data augmentation. An analysis of the prediction performance for a varying number of training samples suggests that the proposed method is particularly effective in extracting features from a small-scale graph dataset of approximately 1000 samples. The appropriate value of the hyperparameter p may vary depending on the dataset, and it is thus desirable to tune the hyperparameter using methods such as Bayesian optimization to achieve better prediction performance.

In this study, we modified the MPNN to obtain the PMPNN, but adding perturbations during message passing is a general method that can be applied to any GNN model. We thus believe that we can augment data in the same way when using GNN models such as the graph convolutional network [102] and the graph isomorphism network [103]. In particular, because the graph convolutional network is a typical model used in theoretical studies of GNNs [104], there is a possibility that the effect of data augmentation by perturbation, which was only experimentally verified in this study, can be theoretically analyzed.

Although the perturbations added by the PMPNN were all standard normal random numbers, the degree of perturbation may be inappropriate for some samples. We therefore expect the prediction performance to be further improved by adjusting the perturbation value according to the feature

vector. In addition, it will be a future work to check the behavior when the distribution of the perturbations is changed to a different distribution such as the uniform distribution.

Finally, it is noted that the data augmentation proposed in this study is designed to make the best use of the available graph dataset and does not necessarily lead to a dramatic improvement in prediction performance. For further improvement in the prediction performance, other methods, such as transfer learning on a large-scale unlabeled graph dataset, are needed. Verifying the effect of data augmentation when the proposed method is used together with such transfer learning is a future topic of research.

Chapter 4

Data-oriented Generation with a Graph-based Deep Structure Generator

4.1 Introduction

A deep structure generator is a neural network model that can generate molecular structures similar to one of the training samples without setting explicit structure generation rules. The number of studies on deep structure generators has been increasing in the last few years, and various deep structure generators have been proposed so far [105, 106]. In particular, string-based deep structure generators that use string representations of molecules like SMILES strings for model input and output, and graph-based deep structure generators that use molecular graphs for model input and output have become the mainstream.

String-based deep structure generators are easy to implement structure generation mechanisms by leveraging neural networks, which are commonly used in natural language processing. For example, the ChemTS [39] can generate compounds with desired properties by using a recursive neural language model, which predicts the next symbol from an input SMILES substring, and Monte Carlo tree search [107], which is often used in gaming AIs. In addition, the chemical variational autoencoder [45], which is a model

that combines a network that converts SMILES strings into numerical vectors and a network that generates SMILES strings from an input numerical vector, can generate SMILES strings by sampling numerical vectors after adequate training. However, when SMILES strings are used for input and output, the generated strings, because of the SMILES string grammar, do not always become valid SMILES strings, i.e., SMILES strings corresponding to actual compounds. For this reason, methods that explicitly take the SMILES grammar into account [40, 108] or string representations that are robust against grammatical errors [109, 110] have been used.

On the other hand, graph-based deep structure generators [111] tend to have a more complicated structure generation mechanism than string-based models. However, they can handle molecular substructures, making it easier to check valence constraints during the generation process and thus generating more valid molecular graphs, i.e., graphs corresponding to actual compounds. In fact, many graph-based deep structure generators, such as the deep generative model of graphs [42], NeVAE [112], GraphNVP [113], GraphCNF [114], and GraphEBM [115], can generate more valid molecular structures than the SMILES string-based model. In particular, the graph-based deep structure generators such as the junction tree variational autoencoder (JT-VAE) [54], the constrained graph variational autoencoder [116], the graph convolutional policy network [44], and MoFlow [117], achieve 100% valid generation of molecular graphs. Because molecular graphs are expected to be easier to capture the information of molecular structures than SMILES strings, as described in Section 3.1, we believe that graph-based deep structure generators can generate structures with a good performance by capturing the features of training samples.

However, most of the deep structure generators proposed so far have been evaluated on large datasets such as the QM9 and ZINC datasets, which consist of more than 100,000 samples. As described in Chapter 1, the number of compounds of interest for training is often limited in compound design, and for such small chemical datasets, there is a possibility that the deep structure generator suffers from overfitting, resulting in generating only the same samples as the training samples or many similar samples.

Transfer learning can be effective in suppressing overfitting. When using

transfer learning in training a deep structure generator, we pre-train the model on a large dataset from a database such as PubChem or ZINC to learn how to construct molecular structures, and then train it on a small dataset of interest so that it can generate structures that are typical of the target dataset. By increasing the number of samples used for training, the effect of overfitting is expected to be suppressed.

Data augmentation is another method that can suppress overfitting by increasing the apparent number of training samples. In Chapter 3, we designed a method for data augmentation by adding perturbations to the GNN when extracting features from samples. Since GNNs are used for feature extraction from training samples in the graph-based deep structure generator, we believe that data augmentation can be applied similarly and thus help the model improve the generation performance.

Therefore, in this chapter, we aim to improve the effectiveness of transfer learning for deep structure generators by using the data augmentation method designed in Chapter 3. Here, we use JT-VAE [54] as a base model. It has been reported that JT-VAE can generate 100% valid molecular graphs by checking the validity of the structures during the structure generation. JT-VAE is also distinctive in that it is relatively easy to generate stereochemically stable structures, although it does not explicitly consider stability, because it assembles molecular structures based on the substructures in the training dataset. Furthermore, Bayesian optimization can be used to search for compounds with the desired properties. For these reasons, we adopted JT-VAE as the base model.

In JT-VAE, two types of GNNs are used for feature extraction from training samples. For each of them, we created a model that performs data augmentation by adding perturbations. In order to validate the effect of data augmentation, we pre-trained the model on a set of lead-like compounds obtained from the ZINC20 database [49], and then trained the model on the PCBA dataset to generate structures. The performance of the structure generation was evaluated by checking the metrics such as novelty, diversity, uniqueness, and similarity of the generated compounds to the training dataset, and the generated structures.

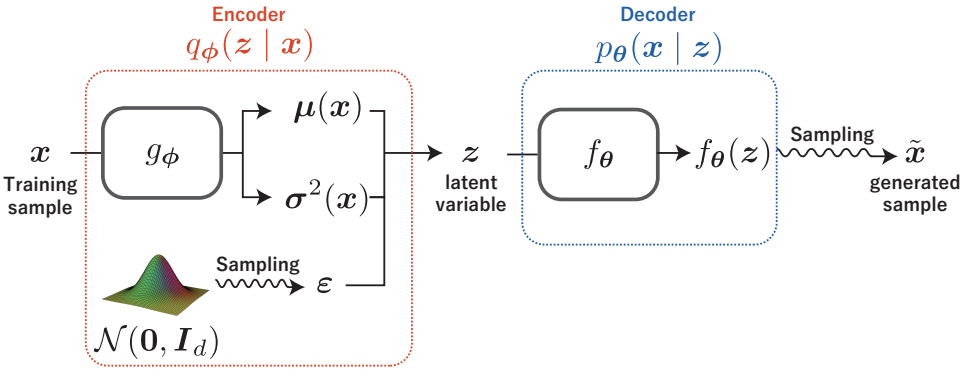

Figure 4.1: The VAE architecture (Translation of Figure 2.10 in the book [119] into English). $f_{\boldsymbol{\theta}}$ and $g_{\boldsymbol{\phi}}$ indicates neural networks.

4.2 Methods

In the following, we first explain the architecture of the model and the training method of the variational autoencoder. Then, we describe the JT-VAE architecture and how to modify the model to use the data augmentation in JT-VAE.

4.2.1 Variational Autoencoder

Assume that every sample follows some probability distribution (the **sample generating distribution**) $p_{\text{data}}(\boldsymbol{x})$. The **variational autoencoder** (VAE) [118] is a deep generative model that explicitly models this distribution $p_{\text{data}}(\boldsymbol{x})$ (Figure 4.1). In VAE, the sample generating distribution is approximated by a nonlinear latent variable model $p_{\boldsymbol{\theta}}(\boldsymbol{x})$ ($\boldsymbol{\theta}$ is a model parameter). That is, we introduce a d-dimensional unobservable latent variable $\boldsymbol{z} \in \mathbb{R}^d$ that follows the prior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{0}, \boldsymbol{I}_d)$, and the sample $\boldsymbol{x}$ is condidered to follow the conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$. In this model, given a latent variable $\boldsymbol{z}$, we can generate a new sample $\tilde{\boldsymbol{x}}$ that approximately follows the sample generating distribution. The conditional distribution $p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})$ is modeled by a neural network, and this neural network that stochastically generates samples from a given latent variable $\boldsymbol{z}$ is called the **decoder** of the VAE.

Because the parameter $\boldsymbol{\theta}$ that makes it easier to generate the training dataset cannot be determined analytically, the posterior distribution $q_{\boldsymbol{\phi}}$ of

the latent variable given the sample $\boldsymbol{x}$ is auxiliarily introduced ($\boldsymbol{\phi}$ is a model parameter). In particular, this distribution is set to be a multidimensional normal distribution with $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{\mu}(\boldsymbol{x}), \mathrm{diag}(\boldsymbol{\sigma}^2(\boldsymbol{x})))$. The distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ is also modeled by a neural network, and this network is called the **encoder** of the VAE. The encoder is responsible for transforming the input sample $\boldsymbol{x}$ into a latent variable $\boldsymbol{z}$ that can generate $\boldsymbol{x}$.

In the training of VAEs, the log-likelihood of a sample $\boldsymbol{x}$, $\log p_{\boldsymbol{\theta}}(\boldsymbol{x})$, which measures the degree of how likely it is that the sample $\boldsymbol{x}$ is derived from the model $p_{\boldsymbol{\theta}}$, is indirectly maximized by maximizing a quantity called the variational lower bound for each sample. This training strategy aims to generate a sample that is similar to the training sample from the model $p_{\boldsymbol{\theta}}$. The variational lower bound for a sample $\boldsymbol{x}$ is expressed as follows.

$$\mathcal{L}_{\boldsymbol{\phi},\boldsymbol{\theta}}(\boldsymbol{x}) = \mathbb{E}_{q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})}[\log p_{\boldsymbol{\theta}}(\boldsymbol{x} \mid \boldsymbol{z})] - D_{\mathrm{KL}}[q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x}) \| p_{\boldsymbol{\theta}}(\boldsymbol{z})]. \tag{4.1}$$

Here, $D_{\mathrm{KL}}[q(\boldsymbol{x}) \| p(\boldsymbol{x})]$ is the Kullback–Leibler divergence defined as below:

$$D_{\mathrm{KL}}[p(\boldsymbol{x}) \| q(\boldsymbol{x})] = \mathbb{E}_{p(\boldsymbol{x})}\left[\log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\right] = \int p(\boldsymbol{x}) \log \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})} \, \mathrm{d}\boldsymbol{x}.$$

The first term of Eq. (4.1) is the expectation with respect to the encoder distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ of the log-likelihood for $\boldsymbol{x}$ obtained from the latent variable $\boldsymbol{z}$. This term can be regarded as a measure of whether the latent variable $\boldsymbol{z}$ which is stochastically determined from the sample $\boldsymbol{x}$ is likely to reconstruct the original sample $\boldsymbol{x}$ or not. The second term of Eq. (4.1) measures whether the encoder distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ approaches the prior distribution $p_{\boldsymbol{\theta}}(\boldsymbol{z}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{0}, \boldsymbol{I}_d)$, which serve as a regularization term that restricts $\boldsymbol{\phi}$.

Eq. (4.1) is calculated as follows. First, the second term of Eq. (4.1) is calculated by passing the training sample $\boldsymbol{x}$ through the encoder and obtaining $\boldsymbol{\mu}(\boldsymbol{x})$ and $\boldsymbol{\sigma}^2(\boldsymbol{x})$. Then, the latent variable $\boldsymbol{z}$ sampled from the encoder distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$ is passed through the decoder to calculate the first term of Eq. (4.1).

A VAE trained in this way can generate a sample $\tilde{\boldsymbol{x}}$ similar to $\boldsymbol{x}$ from the latent variable $\boldsymbol{z}$ which is sampled from the encoder distribution $q_{\boldsymbol{\phi}}(\boldsymbol{z} \mid \boldsymbol{x})$. This can be interpreted that the latent variable $\boldsymbol{z}$ summarizes the character-

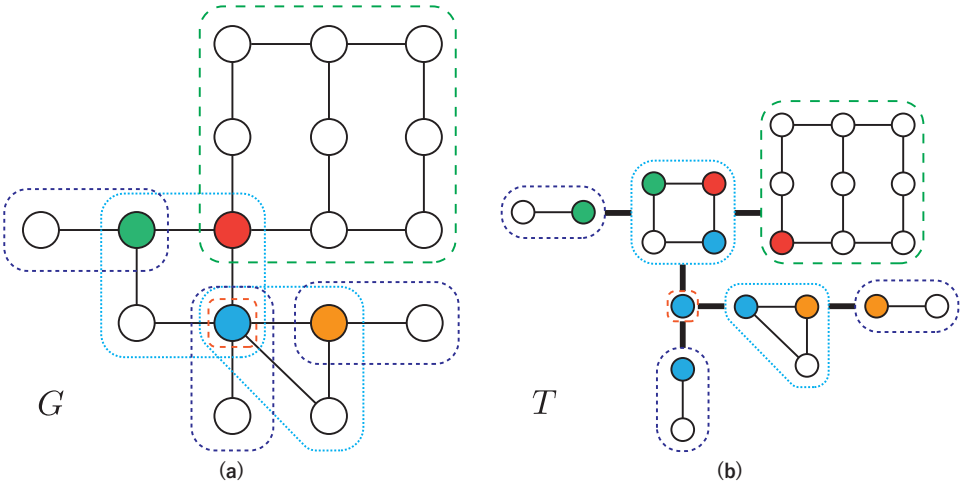

Figure 4.2: Tree decomposition of a graph (Figure 3.11 in the book [119]). (a) The given graph G. (b) A junction tree T corresponding to G

istics of the training sample $\boldsymbol{x}$, i.e., the latent space captures the characteristics of the training dataset. Therefore, if we generate a sample from a latent variable after training the VAE on the training dataset, we can construct a sample similar to a training sample.

4.2.2 Overview of Junction Tree Variational Autoencoder

JT-VAE is a deep structure generator using two VAEs. In JT-VAE, structures are generated by combining substructures, such as rings. JT-VAE uses a coarse-grained tree graph of a molecular graph as an auxiliary graph to capture the connections between substructures of the molecule.

Tree Decomposition

For coarse-graining of molecular graphs, JT-VAE uses a tree decomposition of the graph (Figure 4.2). For a graph $G = (V, E)$, let $\mathcal{C} = \{ C_1, \ldots, C_m \}$ be the set of subgraphs of G, and let $T = (\mathcal{C}, F)$ be a tree (a connected graph without any cycles), where each $C_i \in \mathcal{C}$ is a vertex. Here, the pair $(T, \mathcal{C})$ is said to be a **tree decomposition** of G when the following two conditions hold:

(1) The union of all C_i is equal to G, i.e., $\bigcup_{i=1}^{m} C_i = G$,

(2) For any $C_i, C_j, C_k \in \mathcal{C}$, if the C_i, C_j-path on T contains C_k, then $V(C_i) \cap V(C_j) \subseteq V(C_k)$ (the notation $V(H)$ denotes the set of vertices of the G's subgraph H).

Condition (1) guarantees that any vertex v of G is contained in some C_i and any edge e of G is contained in some C_j. Condition (2) states that the vertex v which is commonly contained in $V(C_i)$ and $V(C_j)$ is always also contained in $V(C_k)$, meaning that T properly reflects the adjacency of the original G. In this sense, T obtained by a tree decomposition of G can be regarded as a coarse-graining of G. The tree T in the tree decomposition of a graph G is called a **junction tree**, and $C \in \mathcal{C}$ is called a **cluster**. In general, a tree decomposition of G is not unique. In JT-VAE, a certain algorithm [54] to generate one of the junction trees is used.

Vocabulary Extraction and Construction of Junction Tree Data

In JT-VAE, it is necessary to define the substructures to be used for structure generation in advance. For this purpose, we perform the tree decomposition of each molecular graph in the training dataset and record all the clusters found. The set of clusters obtained in this way is called the **vocabulary**, and the substructures in the vocabulary are used for structure generation.

Once the vocabulary is obtained, for the graph data $\mathcal{G} = (G, \boldsymbol{X}_V, \boldsymbol{\Xi}_E)$ with $G = (V, E)$, the graph data corresponding to its junction tree $T_G = (\mathcal{C}_G, F_G)$ is set to $\mathcal{T}_G = (T_G, \boldsymbol{Y}_{\mathcal{C}_G}, \boldsymbol{O})$. Here, the feature vector $\boldsymbol{y}_i$ of vertex $i \in \mathcal{C}_G$ is a one-hot vector that indicates which substructure of the vocabulary i represents, and all the edge feature vectors are set to be zero. In addition, the root $r \in \mathcal{C}_G$ of the junction tree is chosen arbitrarily among the T_G's leaves (vertices with only one incident edge). The root r is used for feature extraction from the junction tree and for the generation of the junction tree.

JT-VAE Architectures

JT-VAE consists of two kinds of VAEs, one for molecular graph data $\mathcal{G}$ and the other for its junction tree data $\mathcal{T}_G$ (Figure 4.3). The encoder of the VAE for the molecular graph data determines the distribution of the latent vector $\boldsymbol{z}_\mathcal{G}$ for $\mathcal{G}$, $q_{\boldsymbol{\phi}_\mathrm{g}}(\boldsymbol{z} \,|\, \mathcal{G})$. The encoder of the VAE for the junction tree

data also determines the distribution of the latent vector $\boldsymbol{\zeta}_{\mathcal{T}_G}$ for the junction tree $\mathcal{T}_G$, $q_{\boldsymbol{\phi}_\mathrm{t}}(\boldsymbol{\zeta} \mid \mathcal{T}_G)$. That is, the latent vectors $\boldsymbol{z}_\mathcal{G}$ and $\boldsymbol{\zeta}_{\mathcal{T}_G}$ are calculated independently from $\mathcal{G}$ and $\mathcal{T}_G$, respectively.

Meanwhile, there is a dependency between the distributions determined by the two decoders. The decoder for the junction tree data determines the distribution of the junction tree data $\mathcal{T}$ for the latent vector $\boldsymbol{\zeta}$, $p_{\boldsymbol{\theta}_\mathrm{t}}(\mathcal{T} \mid \boldsymbol{\zeta})$. The decoder for molecular graph data then determines the distribution of the molecular graph $\mathcal{G}$ for the latent vector $\boldsymbol{z}$ and the sampled $\mathcal{T}$, $p_{\boldsymbol{\theta}_\mathrm{g}}(\mathcal{G} \mid \boldsymbol{z}, \mathcal{T})$. Thus, when generating a molecular structure from the given latent variables $\boldsymbol{z}$ and $\boldsymbol{\zeta}$, JT-VAE first samples a junction tree $\mathcal{T}$ from $\boldsymbol{\zeta}$, and then samples the molecular structure $\mathcal{G}$ from $\boldsymbol{z}$ and the sampled $\mathcal{T}$. As in the usual VAE, the prior distributions of the latent variables $\boldsymbol{z}$ and $\boldsymbol{\zeta}$ are both set to the multidimensional standard normal distribution.

The Encoder for Molecular Graph Data In the encoder for molecular graph data, the feature vector $\boldsymbol{h}_\mathcal{G}$ for the input graph data $\mathcal{G}$ is first calculated by GNN. Unlike the MPNN used in Chapter 3, the GNN used here updates the edge feature vectors instead of the vertex feature vectors in message passing. In the readout step, the vertex feature vectors are calculated using the edge feature vectors incident to each vertex, and then the feature vector $\boldsymbol{h}_\mathcal{G}$ for $\mathcal{G}$ is obtained by averaging the vertex feature vectors.

The obtained feature vector $\boldsymbol{h}_\mathcal{G}$ is then input into a fully connected neural network to obtain the mean $\boldsymbol{\mu}(\mathcal{G})$ and variance $\boldsymbol{\sigma}^2(\mathcal{G})$ of the encoder distribution $q_{\boldsymbol{\phi}_\mathrm{g}}$. That is, the encoder distribution for the molecular graph data is set to $q_{\boldsymbol{\phi}_\mathrm{g}}(\boldsymbol{z} \mid \mathcal{G}) = \mathcal{N}(\boldsymbol{z} \mid \boldsymbol{\mu}(\mathcal{G}), \mathrm{diag}(\boldsymbol{\sigma}^2(\mathcal{G})))$.

The Encoder for Junction Tree Data The encoder for junction tree data also uses GNN to calculate the feature vector $\boldsymbol{\eta}_{\mathcal{T}_G}$ for the input junction tree data $\mathcal{T}_G$. Like the encoder for molecular graph data, the edge feature vectors are updated instead of the vertex feature vectors in message passing. In the readout step, the vertex feature vectors are computed using the feature vectors of the edges connected to each vertex. Finally, the vertex feature vector of the root r, which is defined during the junction tree construction, is output as $\mathcal{T}_G$'s feature vector $\boldsymbol{\eta}_{\mathcal{T}_G}$.

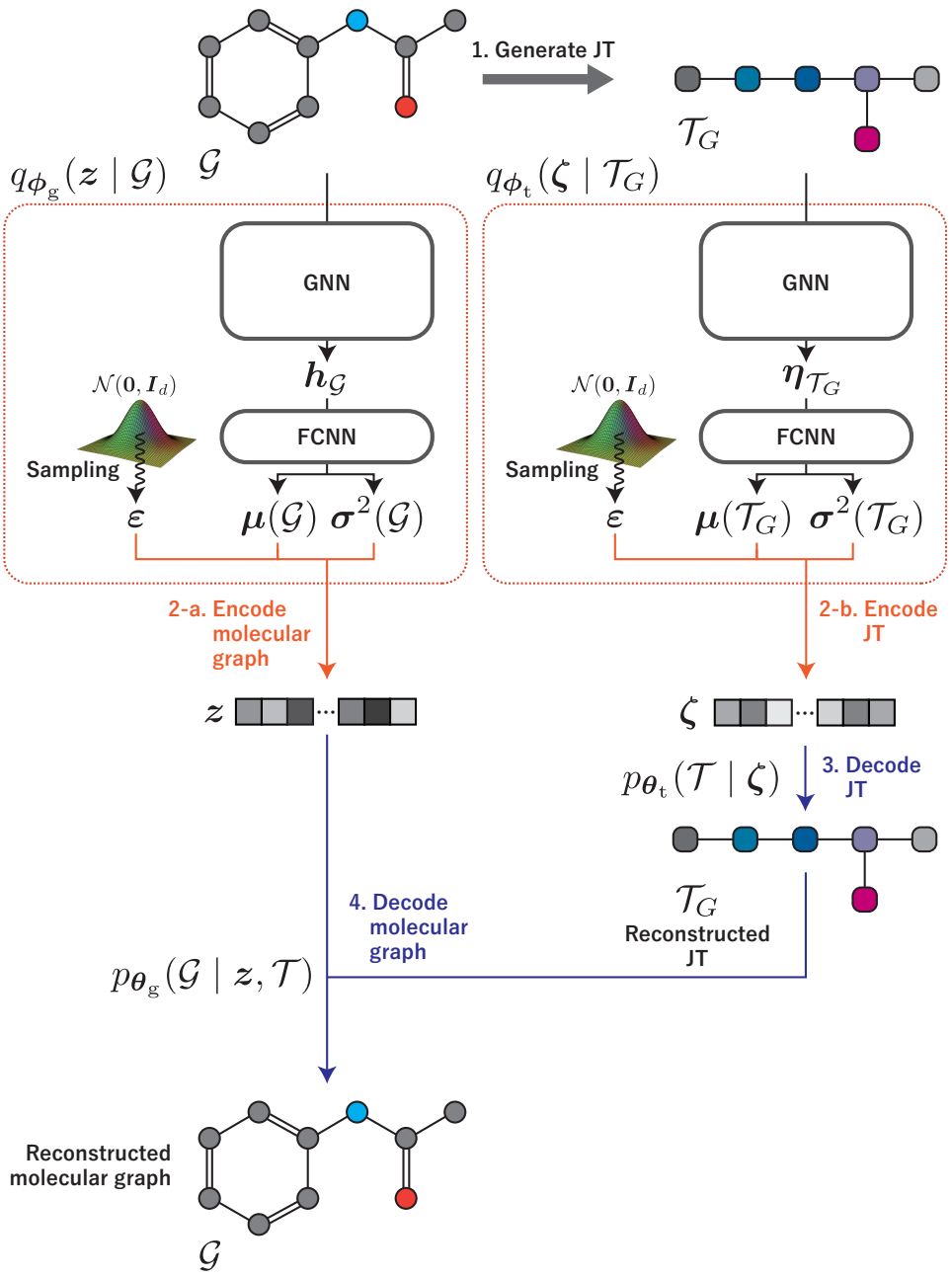

Figure 4.3: The JT-VAE architecture. FCNN denotes a fully connected neural network. d is the dimension of the latent variables.

The obtained feature vector $\boldsymbol{\eta}_{\mathcal{T}_G}$ is then input into a fully connected neural network to obtain the mean $\boldsymbol{\mu}(\mathcal{T}_G)$ and variance $\boldsymbol{\sigma}^2(\mathcal{T}_G)$ of the encoder distribution $q_{\boldsymbol{\phi}_\mathrm{t}}$. That is, the encoder distribution for the molecular graph data is set to $q_{\boldsymbol{\phi}_\mathrm{t}}(\boldsymbol{\zeta} \mid \mathcal{T}_G) = \mathcal{N}(\boldsymbol{\zeta} \mid \boldsymbol{\mu}(\mathcal{T}_G), \mathrm{diag}(\boldsymbol{\sigma}^2(\mathcal{T}_G)))$.

The Decoder for Junction Tree Data The decoder for junction tree data constructs the junction tree data $\mathcal{T} = ((\mathcal{C}, F), \boldsymbol{Y}_\mathcal{C}, \boldsymbol{O})$ based on the sampled latent variable $\boldsymbol{\zeta}$. In the generation of the junction tree $T = (\mathcal{C}, F)$, the root r is first generated, and the substructure corresponding to the root r is stochastically chosen. After this, the addition of vertices and the selection of substructures corresponding to the added vertices are repeated stochastically in depth-first search order from the root, and the resulting junction tree is output. The substructures corresponding to the vertices of the junction tree are selected from the vocabulary so that no invalid molecular graphs are generated by considering the valence constraint.

The sampled latent variable $\boldsymbol{\zeta}$ is used to determine the topology of the junction tree and the substructure corresponding to each vertex stochastically. This means that the junction tree data $\mathcal{T}$ is sampled from the latent variable $\boldsymbol{\zeta}$.

The Decoder for Molecular Graph Data In the decoder for molecular graph data, the junction tree data $\mathcal{T} = ((\mathcal{C}, F), \boldsymbol{Y}_\mathcal{C}, \boldsymbol{O})$ with its root r and the sampled latent variables $\boldsymbol{z}$ are used to construct the molecular graph data $\mathcal{G} = ((V, E), \boldsymbol{X}_V, \boldsymbol{\Xi}_E)$. Because the substructures to be used are already specified in the junction tree data $\mathcal{T}$, this decoder recursively combines the substructures from its root in a stochastic manner and outputs a valid molecular graph as soon as it is generated. If an invalid molecular graph is generated during the generation, the decoder cancels the previous selection and performs the sampling again, so that a valid molecular graph is always generated.

The sampled latent variable $\boldsymbol{z}$ is used to determine how to combine the substructures stochastically. This means that the molecular graph data $\mathcal{G}$ is sampled from the latent variable $\boldsymbol{z}$ and the junction tree data $\mathcal{T}$.

Training Scheme of JT-VAE

In JT-VAE training, the network parameters are determined to maximize the sum of the variational lower bounds for the molecular graph data and junction tree data, as in the usual VAE training. To calculate the sum of the variational lower bounds, we first input the molecular graph data and the corresponding junction tree data of the training samples into the encoder to calculate the Kullback–Leibler divergence term, and then input the sampled latent variables into the decoder to calculate the reconstruction error term. When the VAE is trained in this way, the latent space captures the characteristics of the training dataset.

4.2.3 Perturbating JT-VAE

In JT-VAE, the encoder extracts features from training samples to obtain latent variables. Since the two encoders in JT-VAE use GNNs, the graph data augmentation method proposed in Chapter 3 can be applied to this part (Figure 4.4). In the following, we refer to this model as the perturbating JT-VAE (P-JT-VAE).

Specifically, in P-JT-VAE, the following procedure is used for data augmentation during training. First, we apply the message passing to the molecular graph data in the encoder's GNN. To the feature vectors of each edge just before the readout, we add a perturbation vector following the multidimensional standard normal distribution with probability p_{g}. Then, from the feature vector $\boldsymbol{z}_{\mathcal{G}}$ of the graph data obtained after the readout, we calculate the mean $\boldsymbol{\mu}(\mathcal{G})$ and variance $\boldsymbol{\sigma}^2(\mathcal{G})$ of the distribution, and calculate the corresponding Kullback–Leibler term of the divergence. Similarly, we apply the message passing to the junction tree data in the encoder's GNN. To the feature vector of each edge just before the readout, again, we add a perturbation vector that follows the multidimensional standard normal distribution with probability p_{t}. Then, from the feature vector $\boldsymbol{\zeta}_{\mathcal{T}_G}$ of the junction tree data obtained by the readout, we calculate the mean $\boldsymbol{\mu}(\mathcal{T}_G)$ and variance $\boldsymbol{\sigma}^2(\mathcal{T}_G)$ of the distribution, and calculate the corresponding term of the corresponding Kullback–Leibler divergence. The remaining terms are calculated in the same way as in the usual JT-VAE.

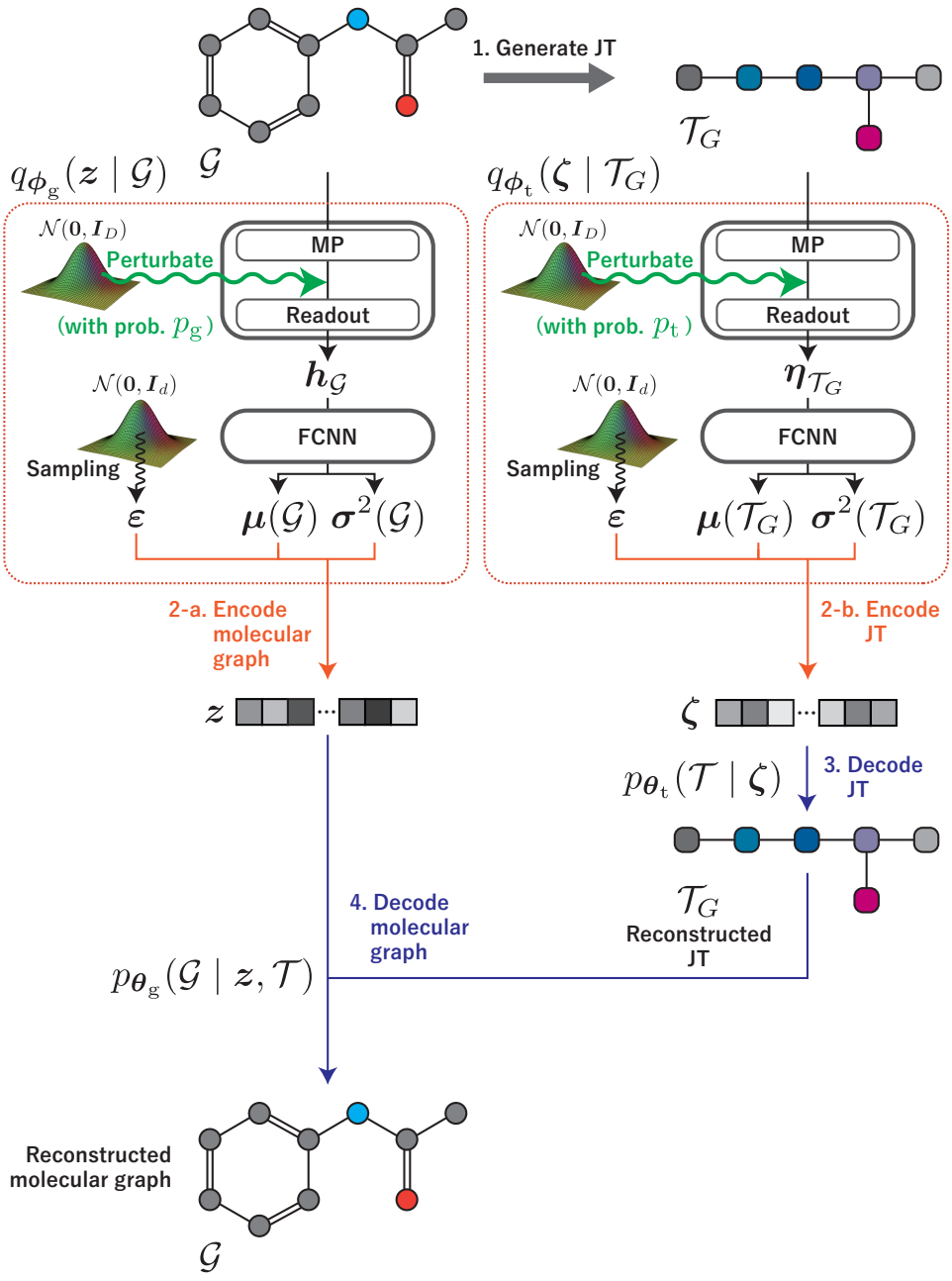

Figure 4.4: The P-JT-VAE architecture. MP denotes the message passing operations, and FCNN denotes a fully connected neural network. D is the dimension of the hidden vectors and d is the dimension of the latent variables.

In this study, the timing of the perturbation is fixed to just before the readout, based on the results of Chapter 3. Also, since we have already studied the effect of perturbations following the multidimensional standard normal distribution in Chapter 3, we again chose this distribution for the distributions of perturbations.

4.3 Results and Discussion

In order to verify the effect of data augmentation by perturbation on the performance of structure generation, we conducted a case study. We trained JT-VAE and P-JT-VAE using transfer learning, and evaluated the performance of structure generation by various metrics. In P-JT-VAE, we constructed three models—two models perturbated only by one encoder and one models perturbated by both encoders—and considered the effect of the perturbation on the structure generation performance.

The experiments were implemented in Python 3.7.3, based on the implementation published by the authors of the JT-VAE paper [54]. We used `RDKit` (version 2019.03.2.0) [58] to handle the organic compound data and `PyTorch` (version 1.7.0) [95] as a deep learning framework.

4.3.1 Datasets

We used 62,577 active compounds of human TDP1 inhibitors in the PCBA-686978 dataset as the target dataset. This dataset is identical to the dataset used for the classification task in Section 3.3. A small chemical dataset was created by randomly sampling 1000 compounds from the 62,577 active compounds. In the following, we refer to this dataset as the PCBA-1k dataset. In addition, as sub-datasets of the PCBA-1k dataset, three smaller chemical datasets with sample sizes of 500, 100, and 50 were also created. We call these datasets PCBA-0.5k, PCBA-0.1k, and PCBA-0.05k, respectively.

On the other hand, the samples used for transfer learning were obtained from the lead-like compounds of ZINC20 [49], which are similar to the samples of the target dataset and thus will help the transfer learning work well. A large dataset of 100,000 samples obtained by random sampling from

7,524,597 lead-like compounds was used for transfer learning. In the following, we refer to this dataset as the ZINC-100k dataset. We have confirmed that there is no common compound between the ZINC-100k dataset and the PCBA-1k dataset.

For the training of the model, we created vocabularies from the ZINC-100k dataset and the PCBA-1k dataset. From the ZINC-100k dataset, we obtained the vocabulary containing 526 substructures, and from the PCBA-1k dataset, we obtained the vocabulary containing 196 substructures. There are 181 substructures common to these two vocabularies. A total of 541 substructures were used for the training vocabulary throughout the experiments.

In order to investigate the distributions of compounds in the PCBA-1k and ZINC-100k datasets, feature vectors were created using 196 `RDKit` descriptors, and then reduced the dimensionality to two by Principal Component Analysis (PCA). The distribution of the compounds in each dataset is visualized in Figure 4.5. Although the contribution of the two components of PCA is low, the overlap between the compound distributions of the PCBA-1k and ZINC-100k datasets is observed. This, together with the large number of common substructures in the vocabularies, suggests that the ZINC-100k dataset is sufficiently similar to the PCBA-1k dataset to be suitable for pre-training.

In constructing the graph data, we used an 39-dimensional vertex feature vector comprising the following elements:

- Type of atom,
- Vertex degree,
- Formal charge,
- Chirality, and
- Presence or absence of aromaticity.

As the edge feature vectors, we used a 11-dimensional feature vector comprising the following elements:

- Type of bond,

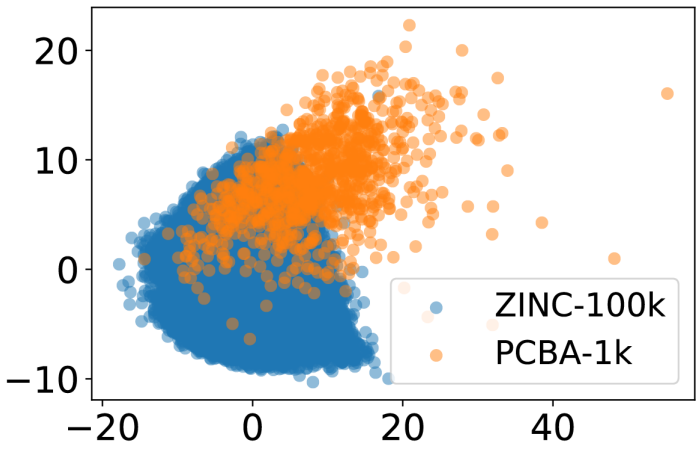

Figure 4.5: The distributions of compounds in the PCBA-1k and ZINC-100k datasets. The percentage of variance explained by two components was 18.2%.

- Whether the bond is in a ring or not, and
- Stereo configuration.

These are the same as the original implementation by the JT-VAE authors.

4.3.2 Models and Experimental Conditions

Network Architectures

The network structure of JT-VAE and P-JT-VAE is the same as the implementation by the JT-VAE authors. The dimension of the vertex/edge feature vector is set to 128, and the dimension of the latent variable is set to 56. The two perturbation probabilities of P-JT-VAE, p_{g} and p_{t}, are set to either 0 or 0.25 (here, one of them always takes a non-zero value).

Experimental Conditions

The training batch size was set to 40 for the ZINC-100k dataset and we pre-trained for two epochs (5,000 parameter updates) on the ZINC-100k dataset. Then, we trained for 100 epochs (2,500 parameter updates) on the small chemical datasets (PCBA-1k, PCBA-0.5k, PCBA-0.1k, and PCBA-0.05k). In order to fix the number of updates of the parameters to 2,500,

the training batch size was set to 40, 20, 4, 2, respectively. Adam [100] was used as the optimization algorithm for the loss function. The learning rate was set at 0.001, and the learning rate was multiplied by 0.95 for every 4000 parameter updates, as implemented by the JT-VAE author. In addition, gradient clipping [120] was performed with its threshold set at 50 to prevent the training from becoming unstable due to the large gradient used for parameter updates.

The authors of JT-VAE pointed out that the model's performance is adversely affected when the Kullback–Leibler divergence term in the loss function of JT-VAE becomes large in the early stage of training. Therefore, similar to the training method by the JT-VAE authors, we use a loss function in which the Kullback–Leibler divergence term is multiplied by $\beta = 0$ until the parameters are updated 4000 times, and thereafter β is increased by 0.002 for every 1000 parameter updates.

Metrics for Evaluation

The set of structures included in ZINC-100k and the set of structures included in PCBA-xk ($x = 1, 0.5, 0.1, 0.05$) are denoted by $\mathscr{Z}$ and $\mathscr{P}_{x\mathrm{k}}$, respectively, and the set of all these structures is denoted by $\mathscr{A}_{x\mathrm{k}} = \mathscr{P}_{x\mathrm{k}} \cup \mathscr{Z}$. The set of known active compounds that are not included in PCBA-xk but are included in the original PCBA dataset is denoted by $\overline{\mathscr{P}_{x\mathrm{k}}}$, and the list of generated structures (a set that allows duplication of elements) is denoted by $\mathscr{G}$.

To evaluate the performance of the structure generation for some $x \in \{\, 1, 0.5, 0.1, 0.05 \,\}$, we used the following metrics that have been used in some papers [112].

Novelty For a set of training samples $\mathscr{D} \in \{\, \mathscr{P}_{x\mathrm{k}}, \mathscr{A}_{x\mathrm{k}} \,\}$, we denote the novelty of $\mathscr{G}$ by

$$\mathrm{Nov}_{\mathscr{D}}(\mathscr{G}) \coloneqq \frac{|\mathscr{D} \cap \mathscr{G}|}{|\mathscr{G}|}.$$

The closer the value is to 1, the higher the percentage of novel structures that are not included in the training sample.

Uniqueness We denote the uniqueness of $\mathscr{G}$ by

$$\mathrm{Uni}(\mathscr{G}) \coloneqq \frac{|\mathrm{set}(\mathscr{G})|}{|\mathscr{G}|},$$

where $\mathrm{set}(\cdot)$ represents the operation to remove duplicates from the given list. The closer the value is to 1, the fewer duplicates there are in the generated structure and thus the higher the generation efficiency.

Diversity This metric $\mathrm{Div}(\mathscr{G})$ is the same as defined in Section 2.3.2 (except that we compute it for $\mathrm{set}(\mathscr{G})$, which excludes duplicate structures). The closer the value is to 1, the more diverse the structures in the generated structures are.

Similarity For the dataset $\mathscr{D} \in \{\, \mathscr{P}_{x\mathrm{k}}, \mathscr{L}, \overline{\mathscr{P}_{x\mathrm{k}}} \,\}$, the similarity of $\mathscr{G}$ is calculated as the average Tanimoto similarity between $\mathscr{D}$ and $\mathrm{set}(\mathscr{G})$:

$$\mathrm{Sim}_{\mathscr{D}}(\mathscr{G}) \coloneqq \frac{1}{|\mathscr{D}||\mathrm{set}(\mathscr{G})|} \sum_{S \in \mathscr{D}} \sum_{T \in \mathrm{set}(\mathscr{G})} (1 - \mathrm{dist}(S,T))\,,$$

where $\mathrm{dist}(S,T)$ is the Tanimoto distance between two structures S and T. The closer the value is to 1, the more similar the generated structures are to the dataset $\mathscr{D}$. Note that, when $\mathscr{D} = \overline{\mathscr{P}_{x\mathrm{k}}}$ (the set of the known active compounds), the value $\mathrm{Sim}_{\mathscr{D}}(\mathscr{G})$ suggests how likely it is that $\mathscr{G}$ will contain compounds with the desired properties.

In addition, we designed the following metrics to check the degree to which we can generate molecules with novel substructures that are not included in the target dataset:

Substructure Novelty We defined the substructure novelty of the list of generated structures as

$$\mathrm{SNov}(\mathscr{G}) \coloneqq \frac{|\mathscr{S}_{x\mathrm{k}}|}{|\mathrm{set}(\mathscr{G})|},$$

where $\mathscr{S}_{x\mathrm{k}} \subseteq \mathrm{set}(\mathscr{G})$ is the set of structures that have substructures not included in the PCBA-xk vocabulary. The closer the value is to 1, the more likely the generated structures are to have new substructures.

Table 4.1: Evaluation metrics (mean ± standard deviation) of the structure generation (trained on the PCBA-1k dataset). The background color indicates the improvement of the evaluation metric over the base model (JT-VAE). Bold type indicates the best evaluation metric.

	$\mathrm{Nov}_{\mathscr{P}}$	$\mathrm{Nov}_{\mathscr{A}}$	Uni	Div	$\mathrm{Sim}_{\mathscr{Y}}$	$\mathrm{Sim}_{\mathscr{P}}$	$\mathrm{Sim}_{\overline{\mathscr{P}}}$	SNov	Succ
JT-VAE	**1.000**	0.998	0.854	0.858	0.113	0.123	0.122	0.003	**0.002**
	±0.000	±0.001	±0.030	±0.006	±0.002	±0.001	±0.001	±0.002	±0.001
P-JT-VAE	0.996	0.996	**0.923**	0.850	0.119	0.133	0.132	0.009	**0.002**
($p_\mathrm{g} = 0.25, p_\mathrm{t} = 0.25$)	±0.001	±0.001	±0.009	±0.007	±0.003	±0.003	±0.002	±0.002	±0.001
P-JT-VAE	**1.000**	**0.999**	0.847	**0.864**	0.110	0.121	0.119	0.005	**0.002**
($p_\mathrm{g} = 0.25, p_\mathrm{t} = 0.0$)	±0.000	±0.001	±0.022	±0.005	±0.004	±0.003	±0.003	±0.001	±0.001
P-JT-VAE	0.998	0.998	0.895	0.841	**0.122**	**0.136**	**0.135**	**0.014**	0.001
($p_\mathrm{g} = 0.0, p_\mathrm{t} = 0.25$)	±0.001	±0.001	±0.011	±0.005	±0.001	±0.001	±0.001	±0.006	±0.001

Success Rate We defined the success rate of the list of generated structures as

$$\mathrm{Succ}(\mathscr{G}) \coloneqq \frac{|\mathrm{set}(\mathscr{G}) \cap \overline{\mathscr{P}_{x\mathrm{k}}}|}{|\mathrm{set}(\mathscr{G})|},$$

Note that $\mathrm{set}(\mathscr{G}) \cap \overline{\mathscr{P}_{x\mathrm{k}}}$ denotes the set of the generated compounds that are not contained in the training dataset and are known to be active. The closer the value is to 1, the more likely the generated structures are to have the desired properties.

4.3.3 Performances of Models

We trained each model with the PCBA-1k dataset ($x = 1$) to generate 1000 structures in three trials.

Comparison of Metrics

We first compared the metrics of the generation. The results are shown in Table 4.1.

In terms of novelty, there was little difference between the models with and without data augmentation, and both models were able to generate novel structures. However, the substructure novelty of the model with data augmentation for the junction tree encoder was larger than that of JT-VAE. Namely, the data augmentation for junction trees facilitated the generation of truly novel structures, as shown in Figure 4.6, with substructures that are

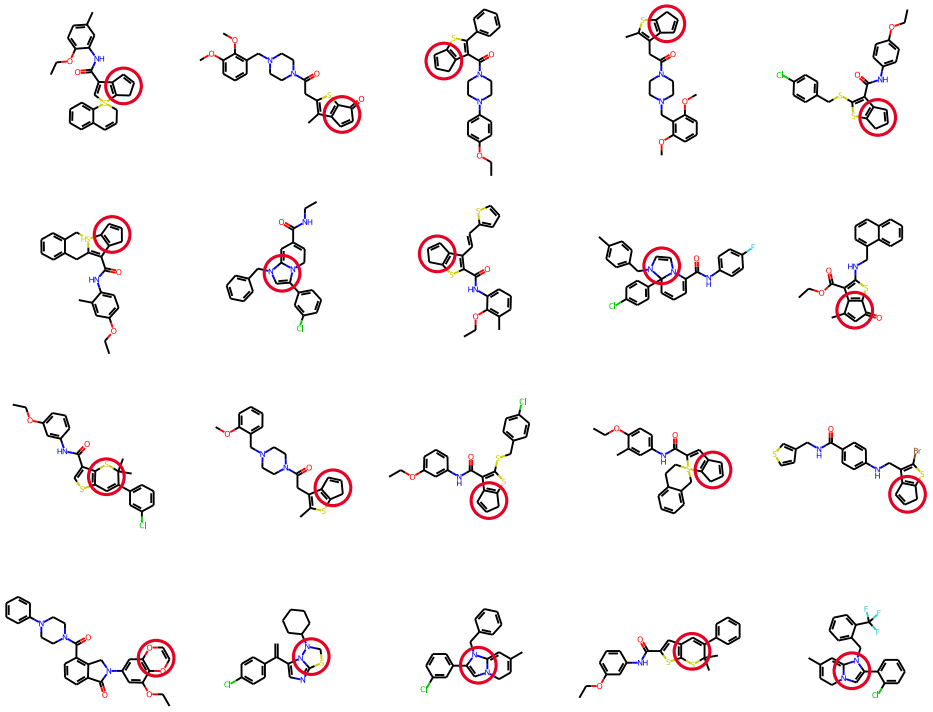

Figure 4.6: Structures with novel substructures generated by P-JT-VAE ($p_{\mathrm{g}} = 0.0, p_{\mathrm{t}} = 0.25$). The red circles indicates the novel substructures.

not included in the target vocabulary but included in the ZINC-100k dataset, which suggests that the data augmentation had successfully captured the structural features of the ZINC-100k dataset. In particular, we believe that the data augmentation for the junction tree was more successful because the junction tree defines most of the information of the generated structures.

In terms of uniqueness, the model with the data augmentation using the junction tree data encoder gave better results than JT-VAE, which indicates that the data augmentation for junction trees improves the efficiency of structure generation. This is because junction trees with various patterns can be generated by capturing the features of the samples in the ZINC-100k dataset, which reduced the number of duplicates.

On the other hand, the diversity was lower than that of JT-VAE for the data augmentation to junction trees and higher than that of JT-VAE for the data augmentation to molecular graphs. This may be due to the char-

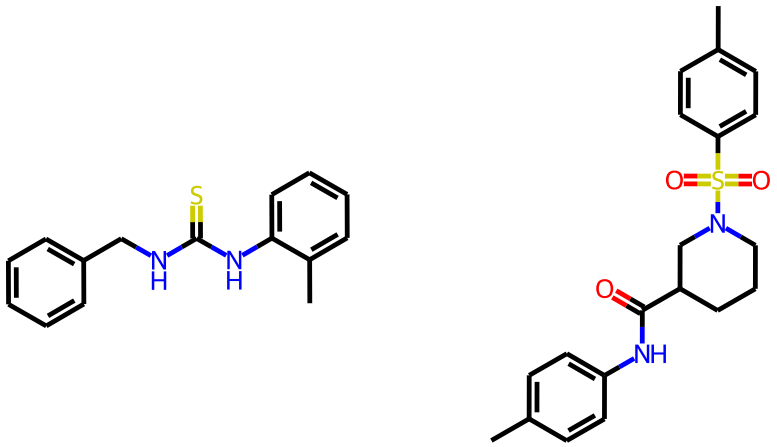

Figure 4.7: Structures generated by P-JT-VAE ($p_g = 0.0, p_t = 0.25$) and known to be active.

acteristics of the Tanimoto distance, which is calculated from the structure of the neighboring subgraphs of each atom. It is easier to obtain different neighboring subgraphs when there is a variation in the way of connecting the substructures defined by the junction tree.

The data augmentation for the junction tree improved the similarity for the training datasets, both for the ZINC-100k dataset and for the target PCBA-1k dataset. It is considered that the data augmentation makes it easier to capture the features of the training dataset because the data augmentation is used to obtain informative latent variables that are easy to reconstruct the training samples even when perturbations are added. In particular, the reason why the data augmentation for the junction tree improved the similarity of the training dataset more is that the feature extraction of the junction tree was successful, and thus the molecular structure can be grasped more comprehensively.

Finally, the similarity to known active compounds was also improved by augmenting the data to a junction tree. This suggests that it is possible to generate a structure with the desired properties by training on the target dataset, although the success rates are comparable. In fact, P-JT-VAE ($p_g = 0.0, p_t = 0.25$) successfully generated the known active compounds as shown in Figure 4.7.

Generated Structures and Molecular Weight Distribution

The 25 randomly selected structures from the 1000 generated structures by each model are shown in Figures A.1 to A.4, respectively. In structures generated by JT-VAE and P-JT-VAE augmenting only the molecular graph data ($p_g = 0.25, p_t = 0.0$), many relatively small and simple molecular structures were found, whereas in structures generated by the two P-JT-VAE that augments the junction tree data ($p_t = 0.25$), there were many large molecules connected with several ring structures. The datasets used for training mainly consist of compounds with their molecular weights ranging from about 200 to 600. Therefore, the high novelty in JT-VAE and P-JT-VAE ($p_g = 0.25, p_t = 0.0$) is considered to be due to the generation of small molecules that are not included in the dataset.

In order to confirm the characteristics of the generated structures in detail, box plots of the molecular weight distributions of the compounds in each training dataset and the generated structures by each model are shown in Figure 4.8. From the plot of JT-VAE and P-JT-VAE which augment only the molecular graph data, we can see that the molecular weight distributions of the generated structures are shifted downward, suggesting that the characteristics of the training dataset are not well captured. On the other hand, the two P-JT-VAEs that augment the junction tree data successfully capture the molecular weight distribution of the target PCBA-1k dataset. In addition, as mentioned above, these models were also able to generate compounds with substructures included in the ZINC-100k vocabulary, implying that the junction tree data augmentation successfully captures the features of the small-scale target dataset and transferred large-scale dataset.

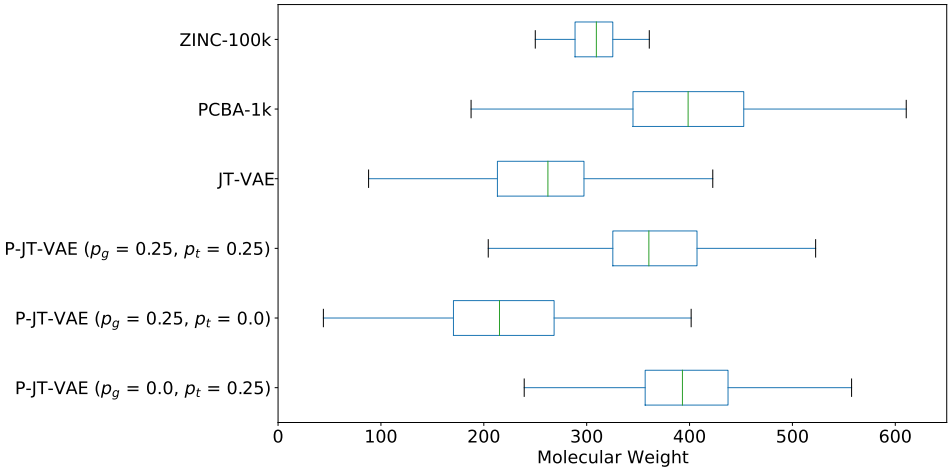

Figure 4.8: Boxplots of each set of structures with respect to molecular weight. P-JT-VAE ($p_{\mathrm{g}} = 0.25, p_{\mathrm{t}} = 0.25$) and P-JT-VAE ($p_{\mathrm{g}} = 0.0, p_{\mathrm{t}} = 0.25$) successfully capture the distribution of molecular weight in the PCBA-1k dataset.

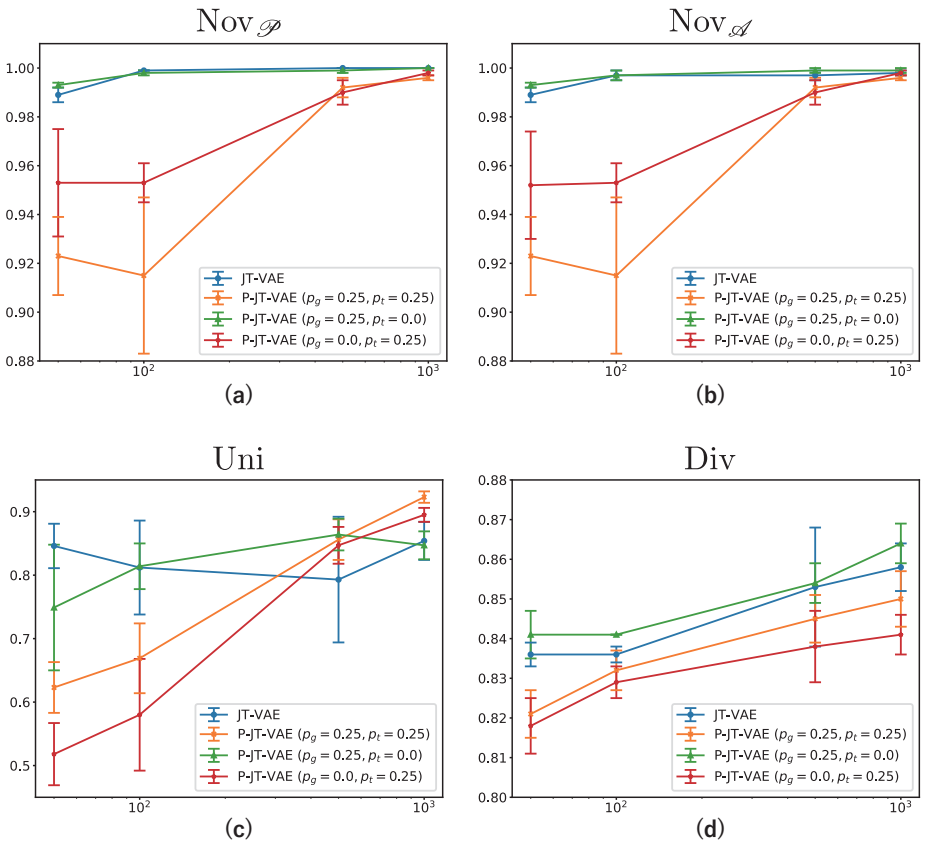

Figure 4.9: Variation in the metrics with the number of training samples. Data are averages for 3 trials. The error bars indicate the standard deviation. (a) Nov$_{\mathscr{P}}$. (b) Nov$_{\mathscr{A}}$. (c) Uni. (d) Div.

Effect of the Number of Training Samples

Finally, we investigated the relationship between the number of samples in the small chemical dataset used for the training and the performance of structure generation. We trained the pre-trained models on the PCBA-0.5k, PCBA-0.1k, and PCBA-0.05k datasets, respectively, and generated 1000 samples per trial. The results are shown in Figures 4.9 to 4.11, and the examples of generated structures are shown in Appendix.

Overall, we confirmed the same trend as the case when the number of training samples was 1,000: the models that perturbates the features on the junction tree tend to generate structures that are similar to the

training samples and the known active compounds (Figure 4.10), although the success rate was comparable across all models (Figure 4.11 (b)). This result will support the hypothesis that the junction tree data augmentation successfully captures the features of the training datasets.

In particular, when data augmentation is applied to the junction tree data, the similarity to the known active compounds $\mathrm{Sim}_{\overline{\mathscr{P}}}$ was found to be comparable regardless of the number of training samples. Considering that the ZINC-100k dataset used for pre-training was similar to the PCBA-1k dataset to some extent, and that the similarity to the ZINC-100k dataset was higher, this is probably because the feature extraction in pre-training was particularly successful due to the augmentation of the junction tree data.

However, novelty, uniqueness, and diversity for such models were lower than other models when the number of training samples is smaller than 500, suggesting a tendency of slight overfitting (Figure 4.9). Also, the substructure novelty for such models were sometimes worse than that for JT-VAE (Figure 4.11 (a)). Note that the substructure novelty has a larger value when the number of training samples is smaller, because the number of substructures in the training samples is smaller.

Also, the characteristics of the generated structures also showed the same tendency as when the number of training samples was 1,000: in structures generated by JT-VAE and P-JT-VAE with $p_{\mathrm{g}} = 0.25, p_{\mathrm{t}} = 0.0$, many relatively small and simple molecular structures were found, whereas in structures generated by the two P-JT-VAE with $p_{\mathrm{t}} = 0.25$, there were many large molecules connected with several ring structures. The 25 randomly selected structures from the 1000 generated structures by each model are shown in Appendix.

These results showed that even when the number of training samples is extremely small, the combination of pre-training and data augmentation is capable of generating similar structures to the targeted small chemical datasets. However, since it tends to slightly overfit the training dataset, it is desirable to have at least 500 training samples.

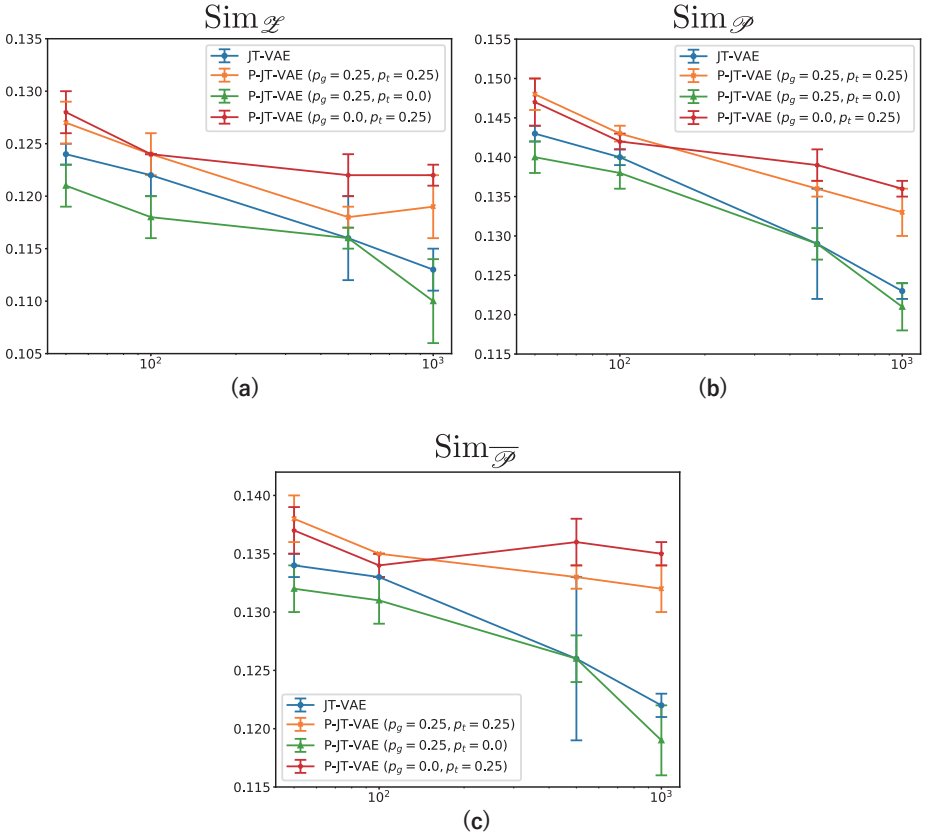

Figure 4.10: Variation in the metrics with the number of training samples (cont.). Data are averages for 3 trials. The error bars indicate the standard deviation. (a) Sim$_{\mathscr{L}}$. (b) Sim$_{\mathscr{P}}$. (c) Sim$_{\overline{\mathscr{P}}}$.

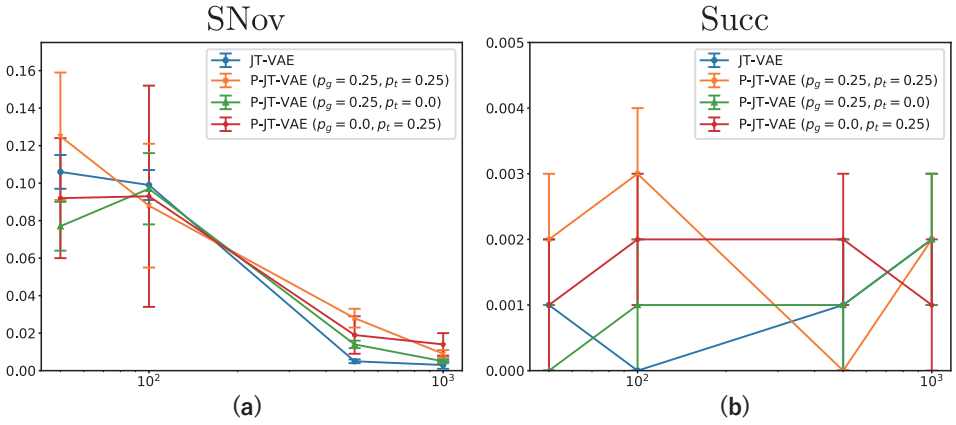

Figure 4.11: Variation in the metrics with the number of training samples (cont.). Data are averages for 3 trials. The error bars indicate the standard deviation. (a) SNov. (b) Succ.

4.4 Conclusion

In this study, we propose a method to utilize data augmentation in transition learning of JT-VAE to generate structures using a small chemical dataset successfully. We applied the graph data augmentation proposed in Chapter 3 to the feature extraction of two encoders in JT-VAE. We found that the data augmentation, especially for junction trees, improved the uniqueness of the generated structures and the similarity to the training dataset, and made it easier to generate novel structures with substructures that are only included in the transferred training samples. The reason for this is that the data augmentation has provided latent variables that can summarize the information of the training samples reasonably and that the junction tree largely determines the information of the molecular structure. Even when the size of the chemical dataset used for training is further reduced, the generation results are similar to those obtained when 1,000 training samples are used. Considering the efficiency of structure generation, it is desirable to have at least 500 training samples.

In this study, we improved the performance of structure generation by augmenting data by perturbations. However, it will be necessary to study the effects of changing the position of perturbation in the model and changing the perturbation probability to the generation performance. In addition, it would be good to study how the perturbations give specific structural characteristics to the generated structures in the future.

Applying data augmentation to the graph-based deep structure generator like that of the QSPR model, we improved the performance of structure generation by transfer learning. Therefore, the graph-based deep structure generator is expected to be successfully used in compound development, where the available data is often scarce. For further improvement of the performance, it may be necessary to modify the loss function, change the dataset and the task used for transfer learning, and so on, in addition to data augmentation.

Also, we transferred a dataset similar to the small-scale chemical dataset. However, depending on the purpose of generation, it is not always possible to obtain a large dataset similar to the target compounds. Thus, it would

be good to consider the case where the source dataset is small or the source dataset is not similar to the target dataset, and to investigate what kind of source dataset the generation works better with.

Here, we applied data augmentation based on JT-VAE. We expect the same data augmentation to be applied to other graph-based deep structure generators where feature extraction from training samples is performed, such as Hier-VAE [121]. We leave this verification as future work.

Chapter 5

General Conclusion and Future Perspectives

5.1 Summary

The structure generator, which generates molecular structures by computer, can propose structures satisfying the desired properties by using statistical models and datasets on the properties of interest. Therefore, structure generators are expected to streamline the process of compound design. However, due to experimental costs and other reasons, the available datasets are often small. In such situations, the statistical model may be overfitted, and the structure generator may not be able to generate structures with good performance.

In this thesis, we develop methods that can generate structures with good performance when statistical model-based structure generators are trained on a small chemical dataset consisting of about 1,000 compounds with the desired properties. We proposed two methods to avoid overfitting: one is to use a structure generator based on a statistical model without deep learning, and the other is to perform data augmentation with a deep learning model that takes molecular graphs as input.

In Chapter 2, we proposed a method to generate structures with good performance using the structure generator DAECS, which uses the QSPR model with a small number of model parameters and is resistant to over-

fitting. Since the existing DAECS tends to have low diversity in the set of generated structures, we added structural modification rules that reduce the need for multiple applications of structural modification rules to a single application, and designed a new seed structure selection algorithm that makes the seed structures that undergo structural modification diverse. The case study confirmed that these methods actually diversified the generated structures and generated novel structures that were not included in the training dataset.

In Chapters 3 and 4, we designed a general data augmentation method for graph-based deep learning models and applied it to a graph-based deep structure generator. In Chapter 3, we first designed a data augmentation method for the GNN-based QSPR model. In the proposed method, the data augmentation is performed by perturbating the feature vectors for some vertices with standard normal random numbers during the message passing of GNNs in order to avoid collapsing the structure of the input molecular graph. To confirm the effectiveness of the proposed method, we compare the performance of the two tasks, regression and classification, and find that the perturbation just before the GNN readout operation improves the prediction performance. We found that the prediction performance is likely to be improved particularly when the number of training samples is small.

In Chapter 4, we design a method to improve the effectiveness of transfer learning, which uses a large-scale molecular structure dataset and a small-scale chemical dataset, to prevent overfitting for the deep structure generator JT-VAE. Specifically, the data augmentation method for molecular graph data designed in Chapter 3 is used for feature extraction at two encoders of JT-VAE. We conducted a case study and found that the data augmentation for junction trees improved the generation performance. In particular, the data augmentation for junction trees contributed to better capture the characteristics of the dataset, such as the molecular weight distribution of the training dataset.

5.2 Contribution of This Thesis

There are two major contributions of this thesis. The first is the enhancement of the practicality of the structure generator DAECS. As described in Chapter 2, DAECS can generate compounds with the desired properties even when using small chemical datasets, and the property visualization by the 2D map further enhances its usability. In the previous version of DAECS, the diversity of the generated structures was low because the structures did not change significantly after a single application of the structure modification rule, and the seed structures selected in order to generate compounds close to the target coordinates on the 2D map were similar to each other. From the viewpoint of the development of innovative compounds, this is a severe problem that affects the practicality of DAECS. In this thesis, we have solved this problem and confirmed that DAECS could actually generate novel structures that are not included in the dataset. We believe this will be an important step toward the practical application of DAECS in compound design.

The other is that we provided a method for augmenting graph data with perturbations and verified its effectiveness on small datasets. In order to successfully train a deep learning model with many model parameters on a small chemical dataset, it is necessary to take some measures to prevent overfitting, as in the case of transfer learning, where a large dataset is used for training. Data augmentation is one such measure, and it is often used in training models that use SMILES strings [83]. However, most of the data augmentation methods proposed for GNNs [84–88], which use graph data as input, change the graph structure itself, which is inappropriate for chemical datasets, where the topology of molecular graphs is essential. In this thesis, we propose a general-purpose data augmentation method for graph data by perturbation of feature vectors, and verify the effects of perturbation timing and perturbation probability on small chemical datasets using the QSPR model. We also applied the data augmentation method to a deep structure generator and confirmed that it could enhance the effect of transfer learning in structure generation by capturing the characteristics of the training datasets well. This study confirms that the proposed data augmentation by

perturbation works successfully on small chemical datasets, and is expected to be widely used for training GNNs on chemical datasets. In particular, it is noteworthy that the proposed data augmentation facilitates the use of the deep structure generator in compound design, where it is often necessary to generate structures similar to known structures in order to find ones with desired properties.

5.3 Challenges and Perspectives

In section 1.2, we listed four features that a structure generator should have. We reiterate them below.

(a) The ability to generate many molecular structures with the desired properties.

(b) The diversity of the generated structures.

(c) The stability and synthesizability of the generated structures.

(d) The high computational efficiency in generating molecular structures.

In order to satisfy these properties, statistical model-based structure generators are often used. In this thesis, we have developed a method to improve the performance of such statistical model-based structure generators so that they can satisfy these features even when the statistical model is trained on a small chemical dataset. However, there are still many points to be improved in terms of practical applications in compound design (Figure 5.1).

First, regardless of the number of training samples, the design of the structure generator itself poses some remaining challenges. For DAECS, features (a) and (b) are satisfied even when training on a small chemical dataset. However, the application of the structure modification rule sometimes generates structures that are difficult to synthesize or sterically unstable. If such structures are selected as seed structures, the synthesizability of the entire set of generated structures may be reduced. As a possible solution, we can set more sophisticated filtering rules. For example, a synthesizability metric such as SA score [67] or retrosynthesis analysis can be used to filter

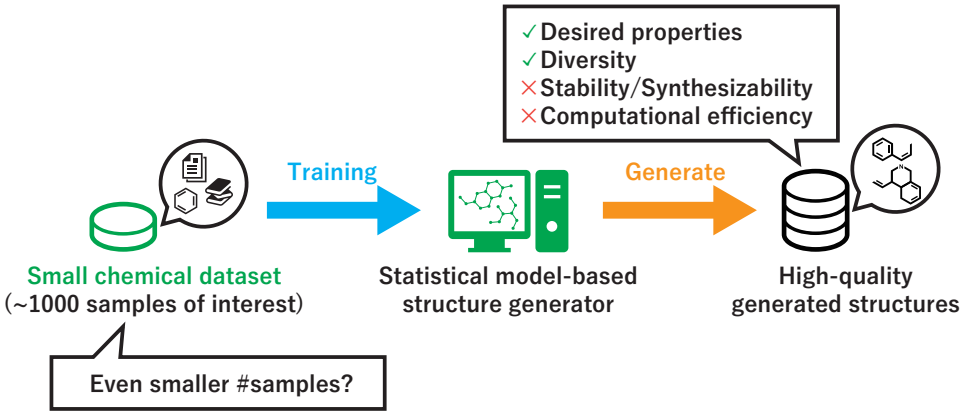

Figure 5.1: Challenges for constructing high-performance structure generator with a small chemical dataset.

out hard-to-synthesize molecules, and a QSPR model that predicts formation energy can be used to filter out sterically unstable structures. There is also room for improvement in the generation efficiency: the same structure can be output repeatedly, and many unnecessary structures that are far from the target coordinates on the 2D map are generated. Meanwhile, in JT-VAE, the data augmentation makes it easier to generate structures similar to the samples in the small-scale chemical dataset, which improves feature (a). It is also reported in the paper [54] that the Bayesian optimization in the latent space can generate structures with explicitly desired properties. For feature (b), we have confirmed that there is a certain degree of diversity in the generated structures. However, for feature (c), although we can guarantee a certain degree of sterical stability because generated structures use the substructures in the training dataset, the synthesizability of the generated structures is not explicitly considered. Also, the structure generation procedure by the decoder is complicated, and there is room for improvement in the generation efficiency. The issues described here should be solved in the future to find practical applications in compound design.

There is also an important challenge on how to deal with an even smaller number of samples. In this thesis, we defined a small-scale chemical dataset as a dataset containing about 1,000 samples. However, in practical compound design, there may be situations where only a minimal chemical dataset

consisting of a few hundred samples is available. In Chapter 4, we conducted the case study where the number of training samples are extremely small and found that our method was applicable, although some metrics for generation tended to be worse. In such a situation, if we want to generate the higher-quality set of structures, it may be necessary to devise a different training strategy, such as using a different transfer learning approach or adopting the framework of few-shot learning, which is used for training with an extremely small number of samples. In addition, it is considered necessary to devote more effort to data collection. Therefore, it would be effective to use online learning or active learning frameworks, which allow us to train the model while acquiring new samples.

Appendix A

Generated Structures in the Experiments of Chapter 4

A.1 Trained on the PCBA-1k dataset

The 25 structures generated by each model trained on the PCBA-1k dataset is shown in Figure A.1 to A.4.

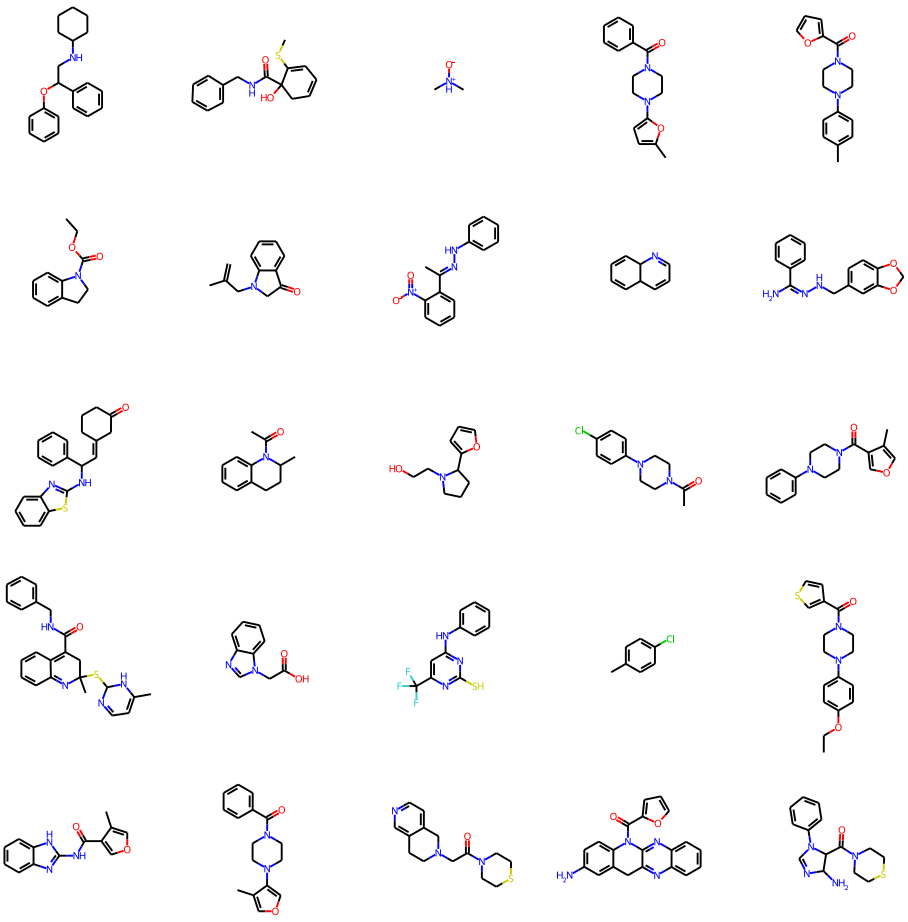

Figure A.1: 25 structures generated by JT-VAE.

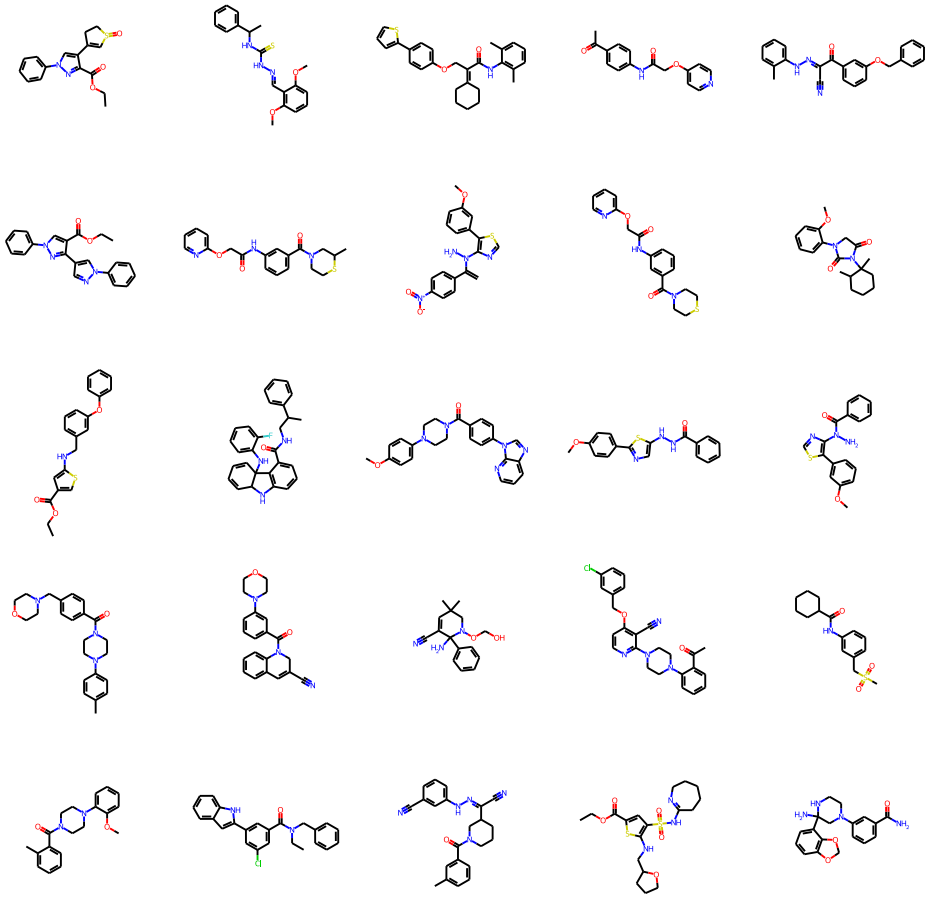

Figure A.2: 25 structures generated by P-JT-VAE ($p_g = 0.25, p_t = 0.25$).

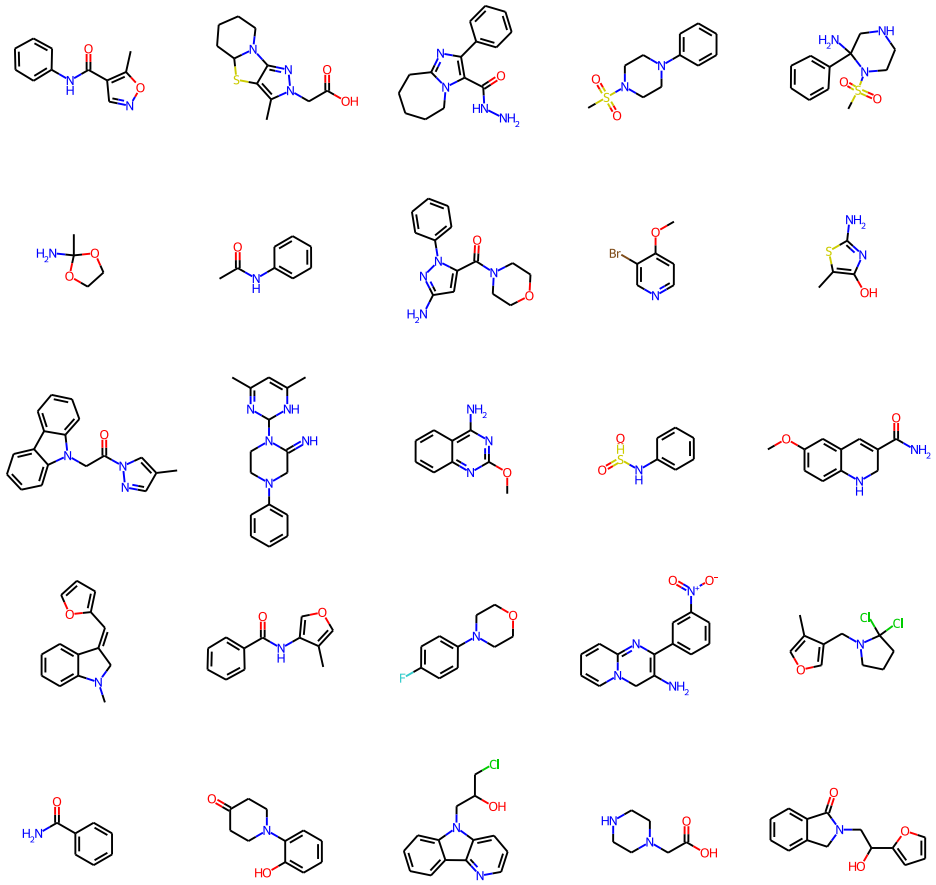

Figure A.3: 25 structures generated by P-JT-VAE ($p_g = 0.25, p_t = 0.0$).

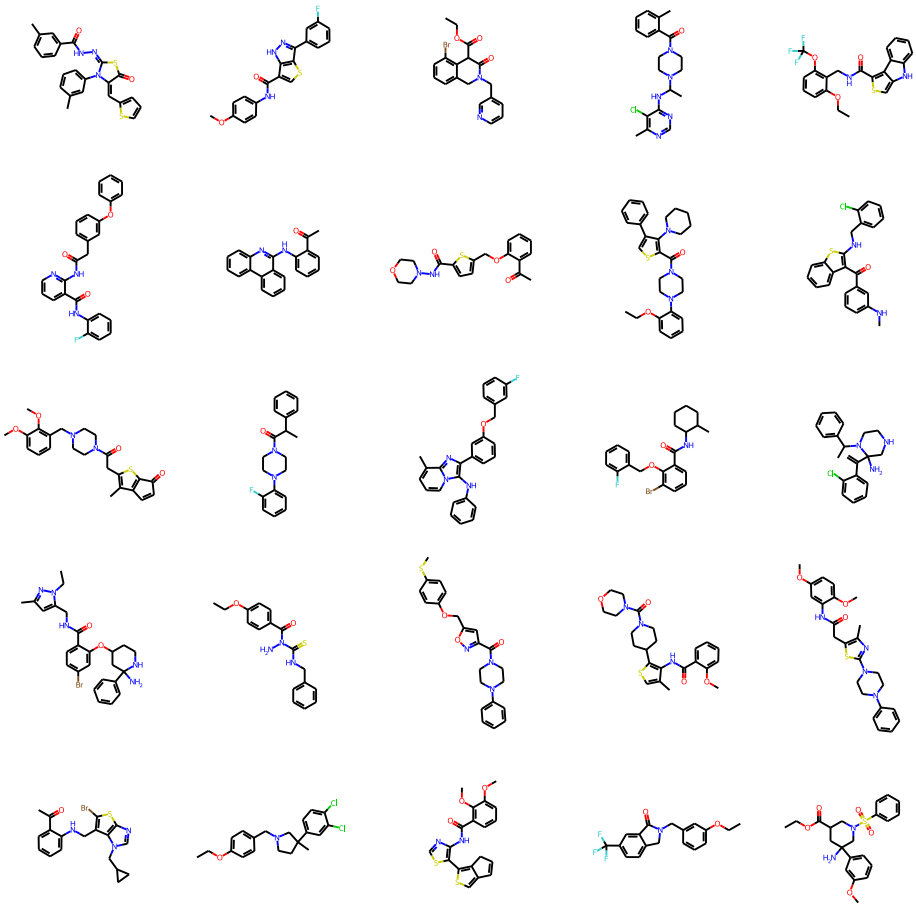

Figure A.4: 25 structures generated by P-JT-VAE ($p_g = 0.0, p_t = 0.25$).

A.2 Trained on the PCBA-0.5k dataset

The 25 structures generated by each model trained on the PCBA-0.5k dataset is shown in Figure A.5 to A.8.

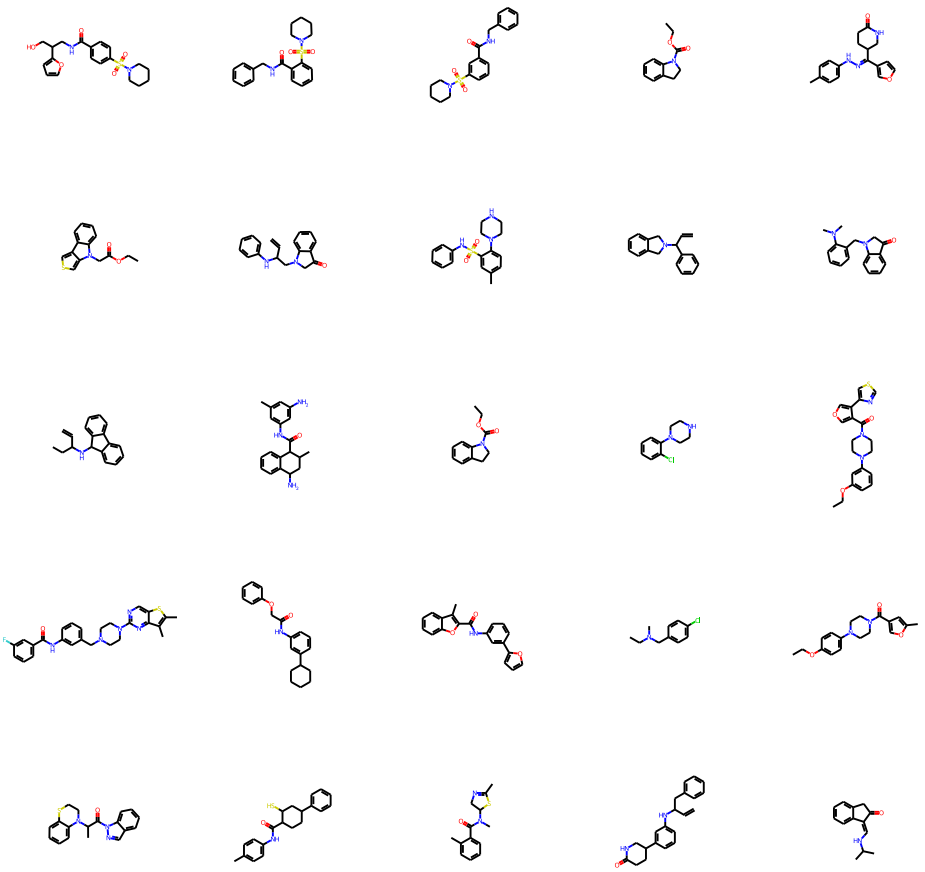

Figure A.5: 25 structures generated by JT-VAE.

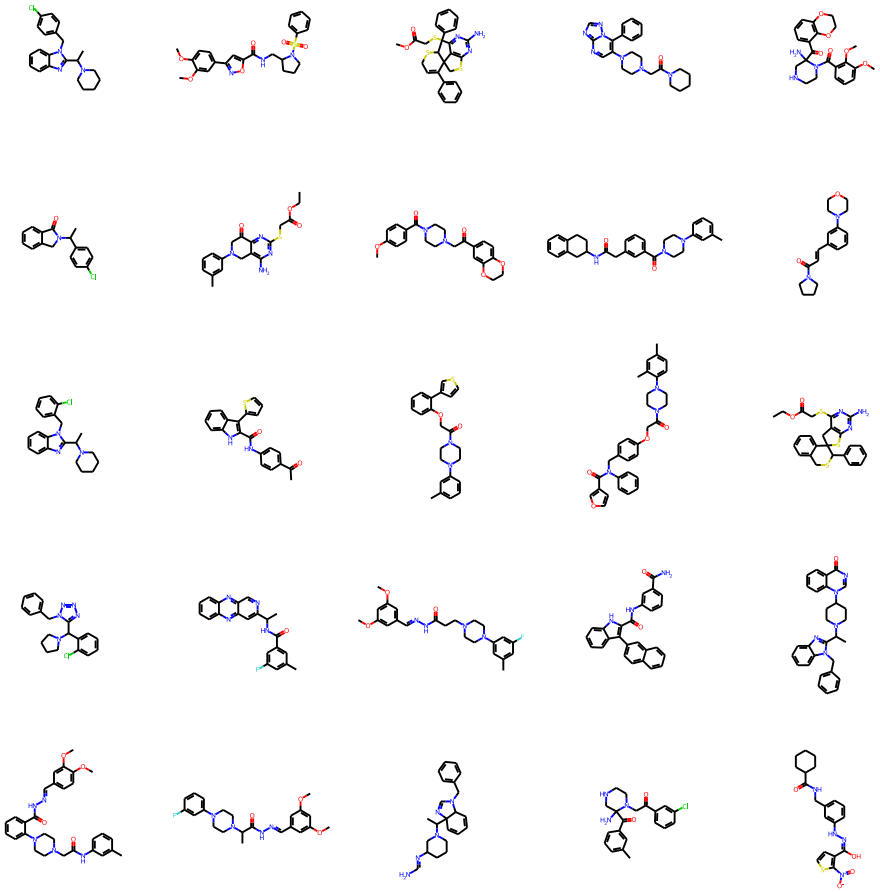

Figure A.6: 25 structures generated by P-JT-VAE ($p_g = 0.25, p_t = 0.25$).

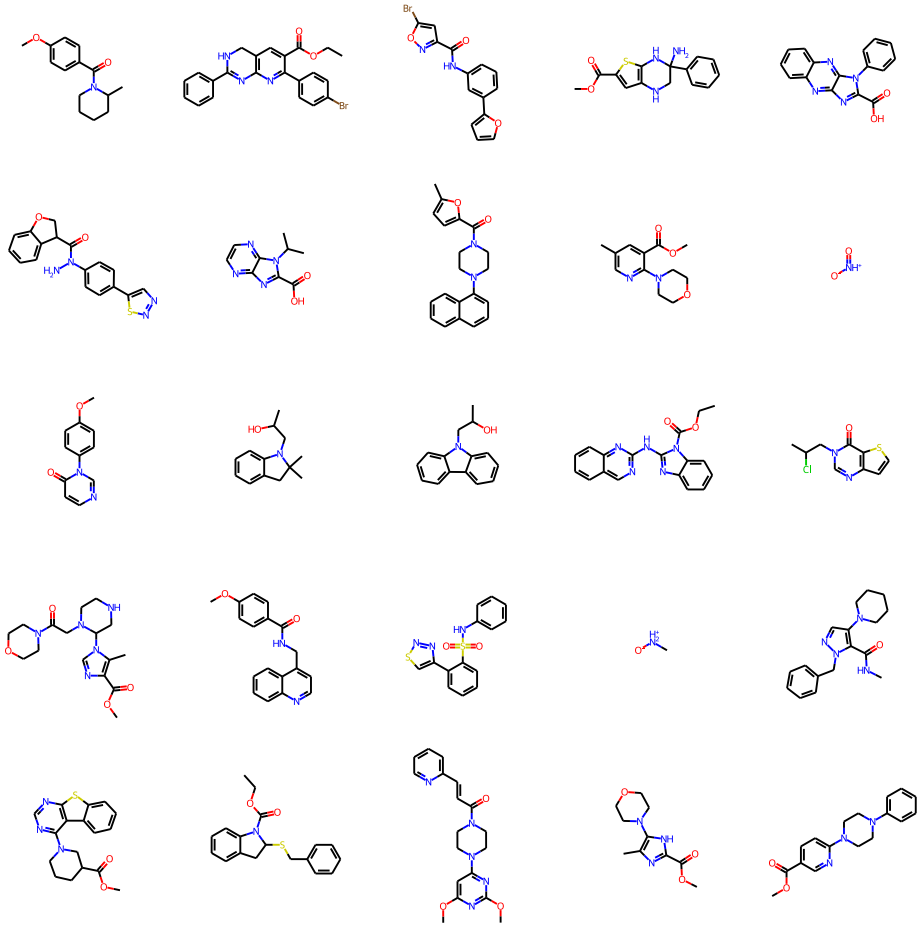

Figure A.7: 25 structures generated by P-JT-VAE ($p_g = 0.25, p_t = 0.0$).

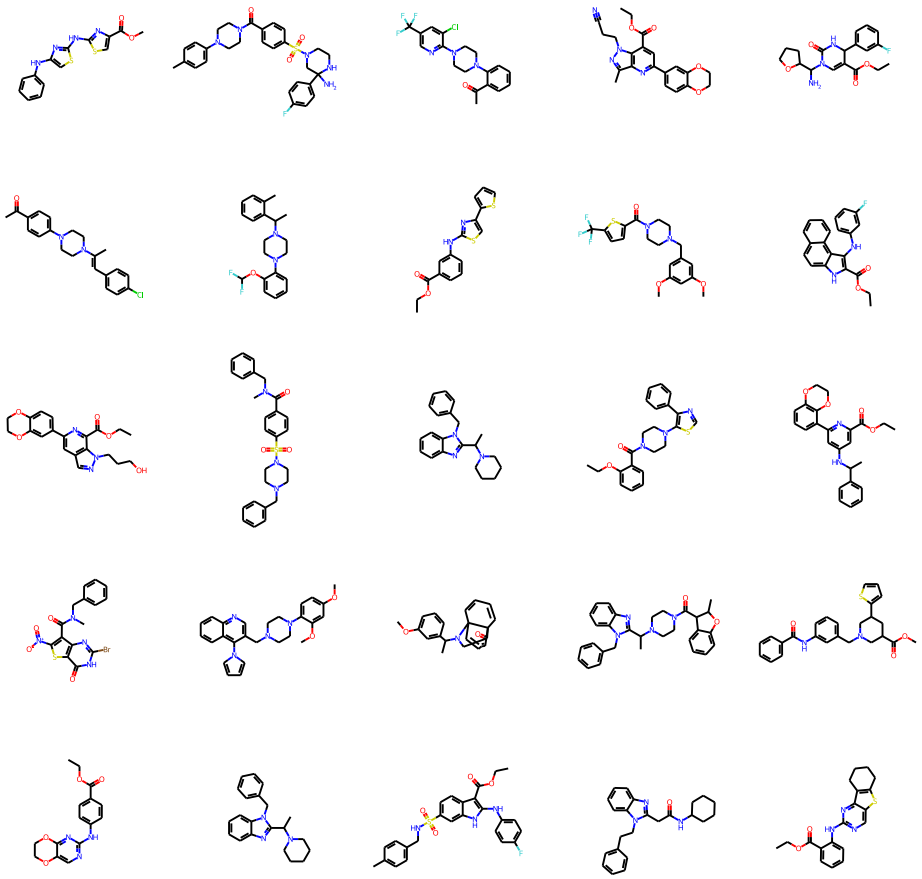

Figure A.8: 25 structures generated by P-JT-VAE ($p_g = 0.0, p_t = 0.25$).

A.3 Trained on the PCBA-0.1k dataset

The 25 structures generated by each model trained on the PCBA-0.1k dataset is shown in Figure A.9 to A.12.

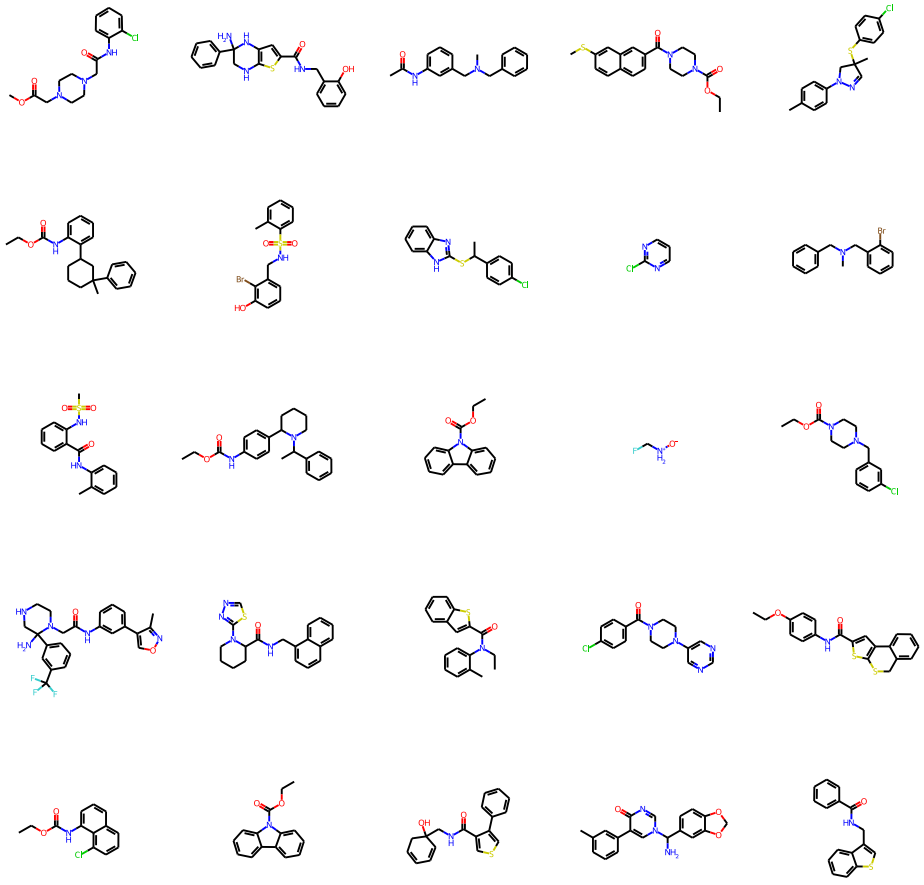

Figure A.9: 25 structures generated by JT-VAE.

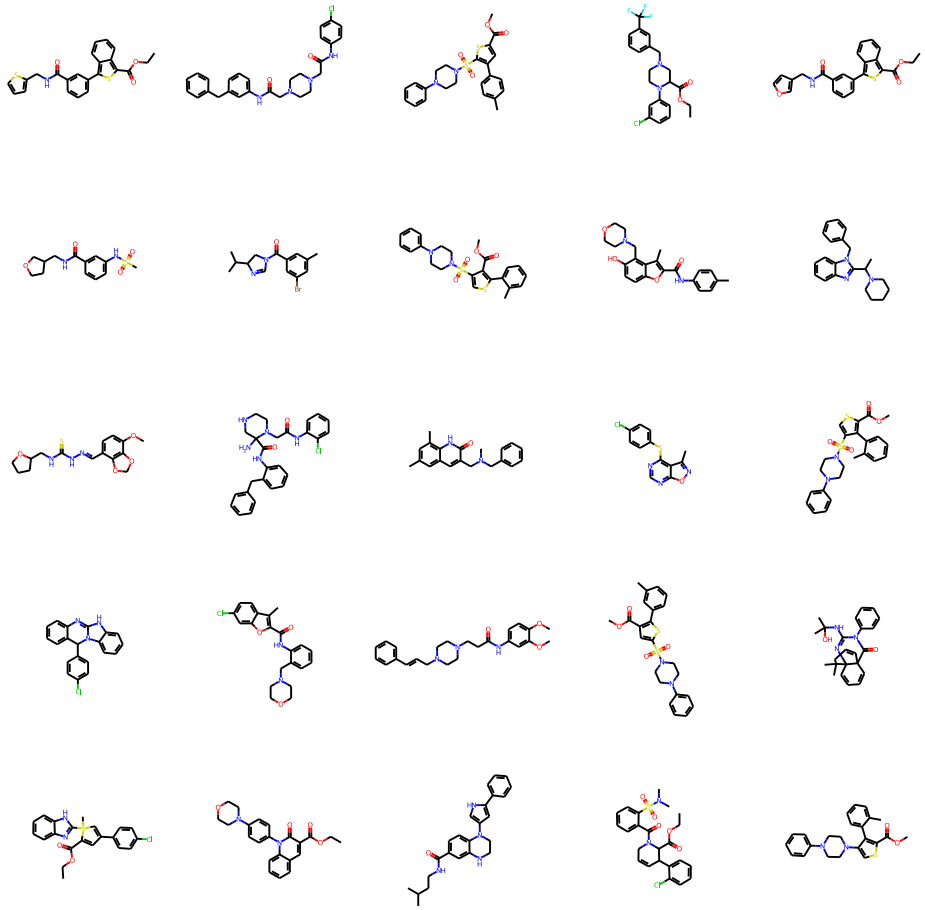

Figure A.10: 25 structures generated by P-JT-VAE ($p_g = 0.25, p_t = 0.25$).

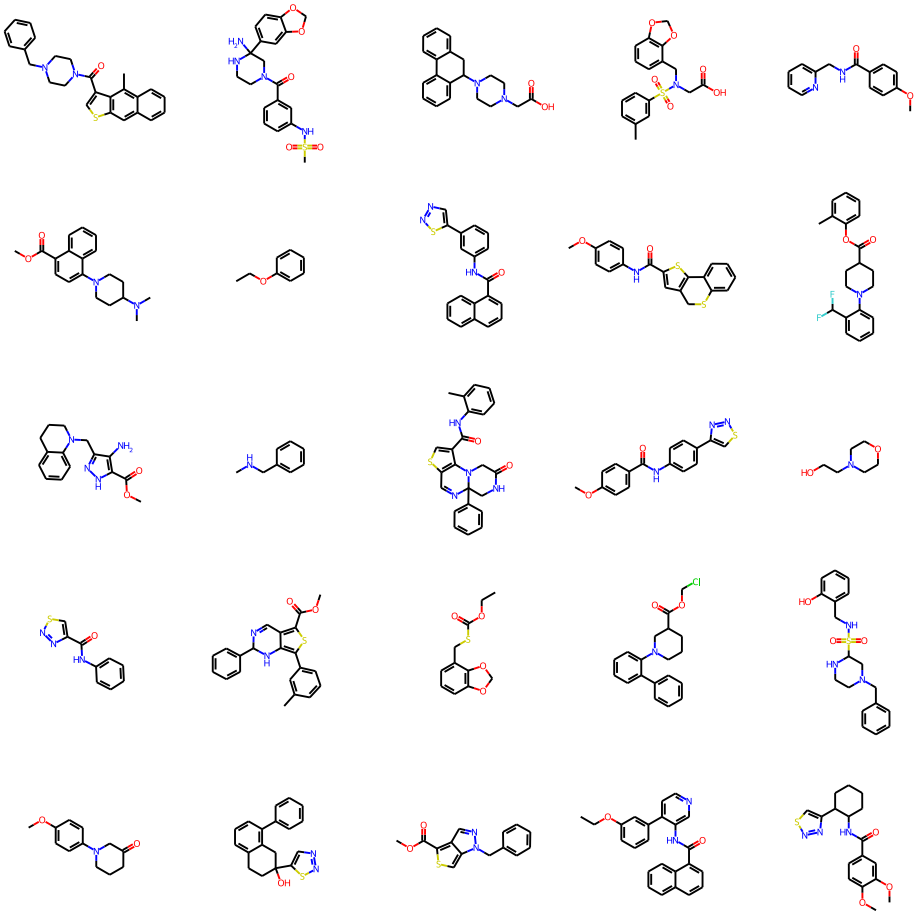

Figure A.11: 25 structures generated by P-JT-VAE ($p_g = 0.25, p_t = 0.0$).

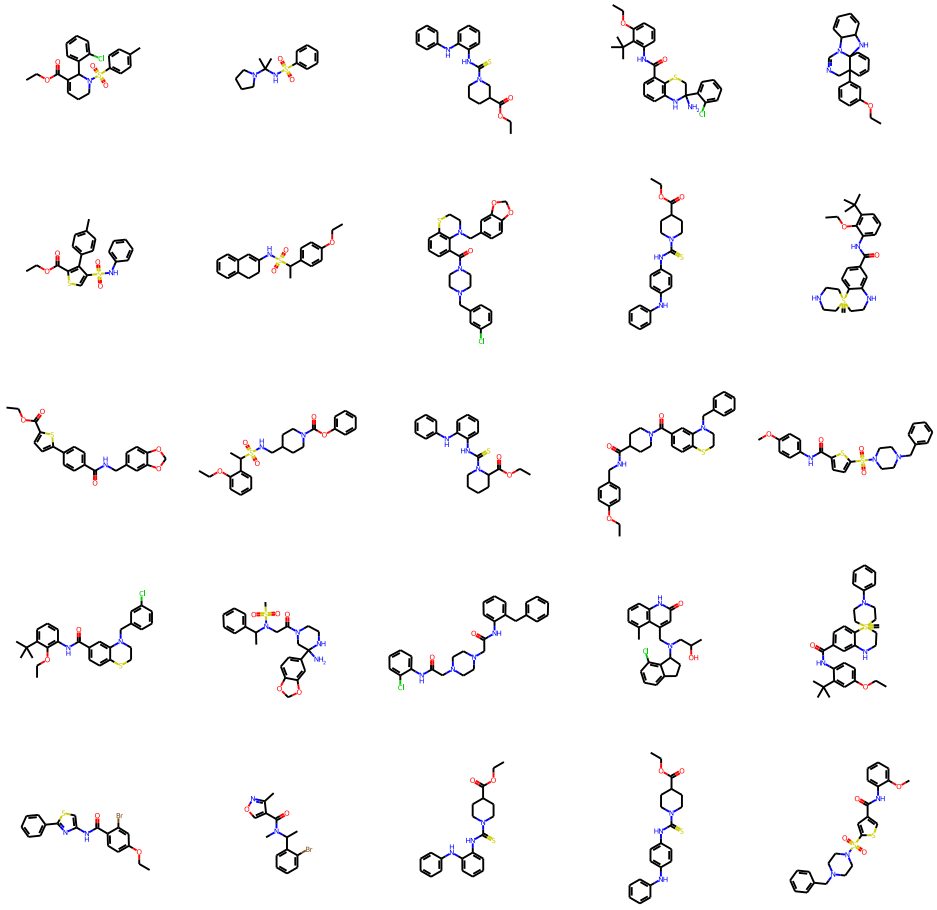

Figure A.12: 25 structures generated by P-JT-VAE ($p_g = 0.0, p_t = 0.25$).

A.4 Trained on the PCBA-0.05k dataset

The 25 structures generated by each model trained on the PCBA-0.05k dataset is shown in Figure A.13 to A.16.

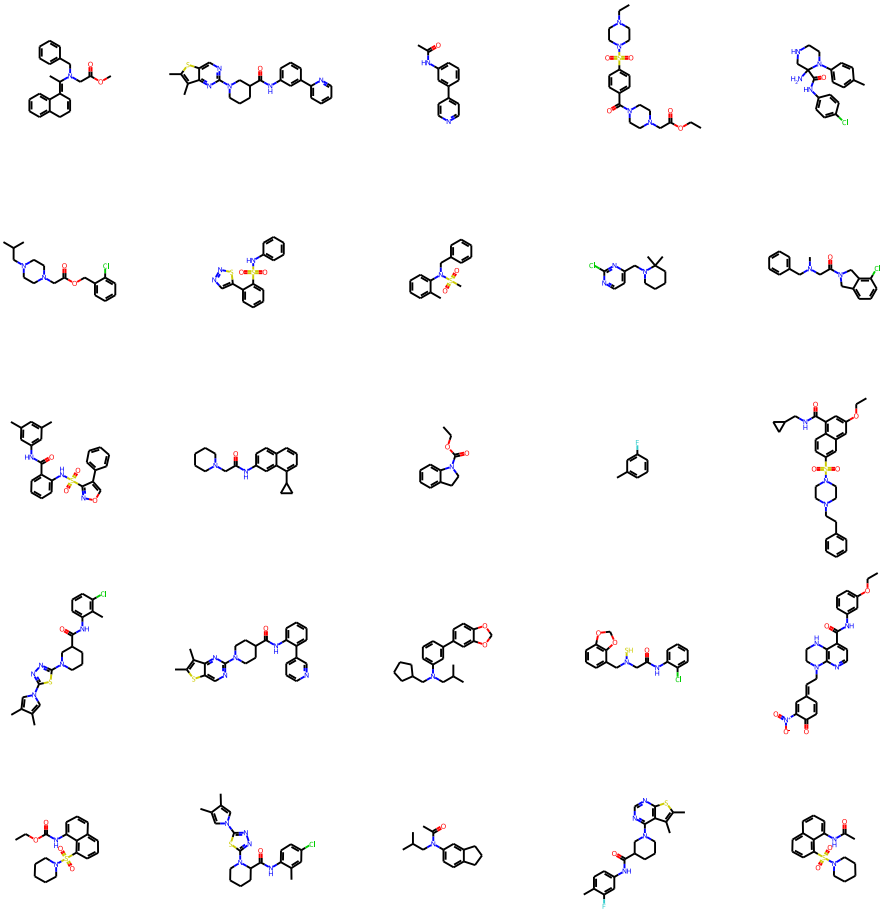

Figure A.13: 25 structures generated by JT-VAE.

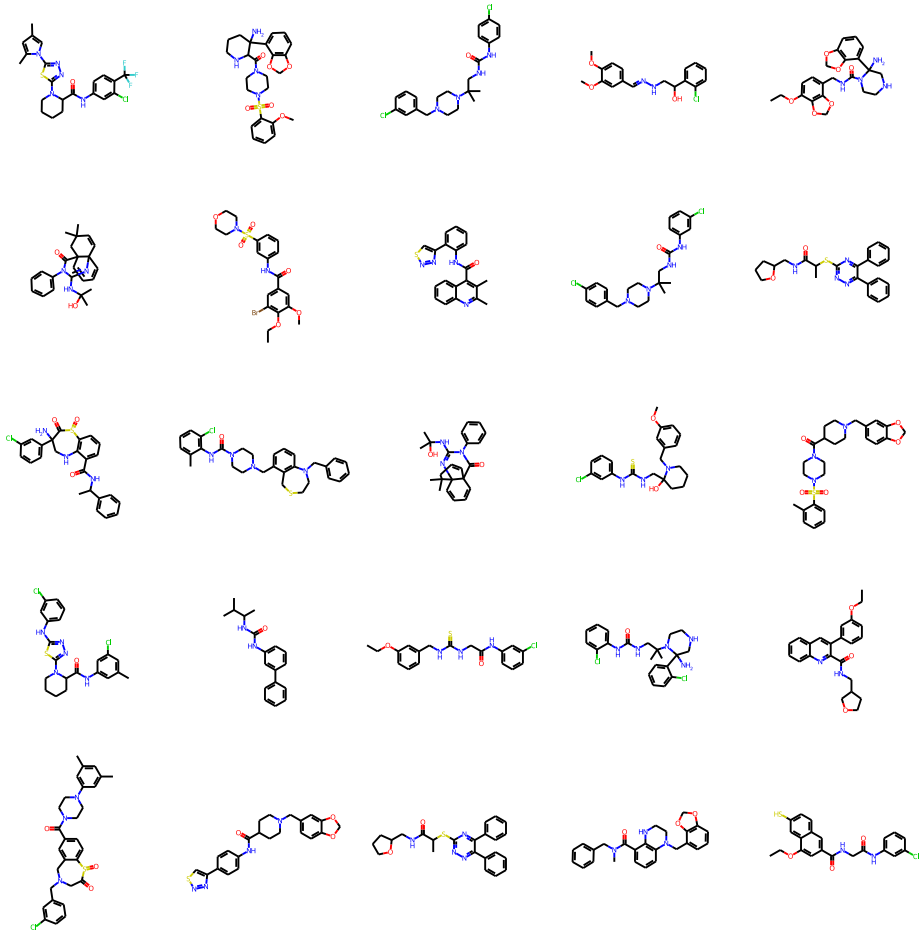

Figure A.14: 25 structures generated by P-JT-VAE ($p_g = 0.25, p_t = 0.25$).

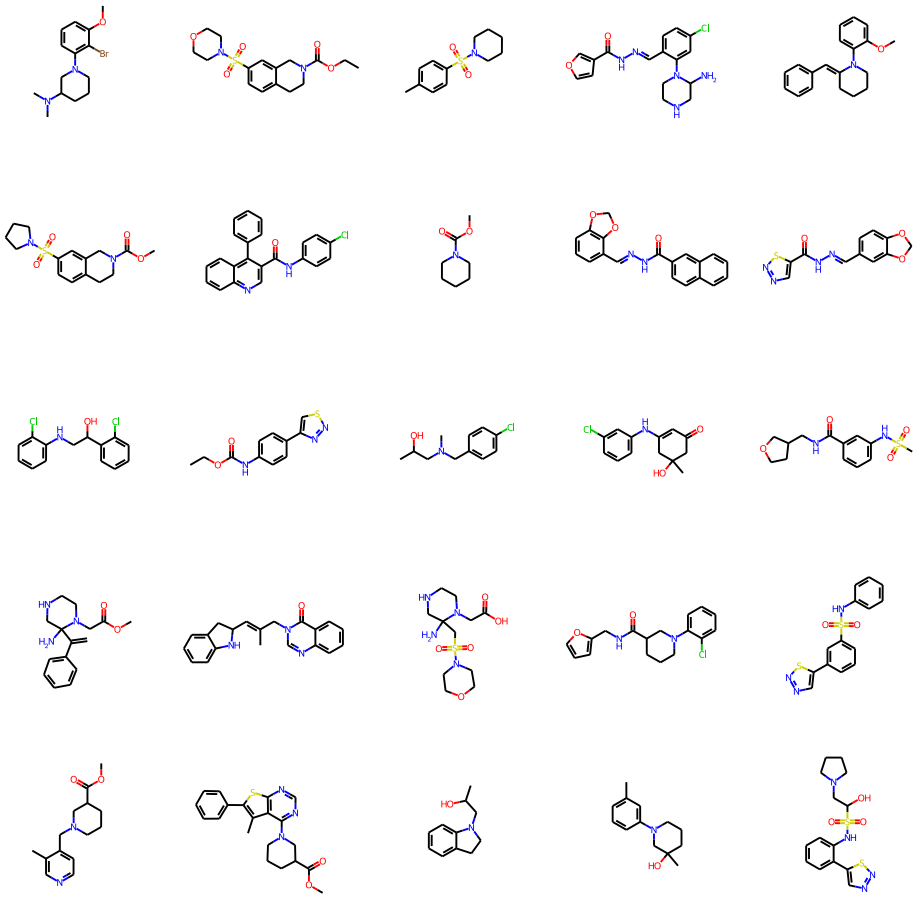

Figure A.15: 25 structures generated by P-JT-VAE ($p_g = 0.25, p_t = 0.0$).

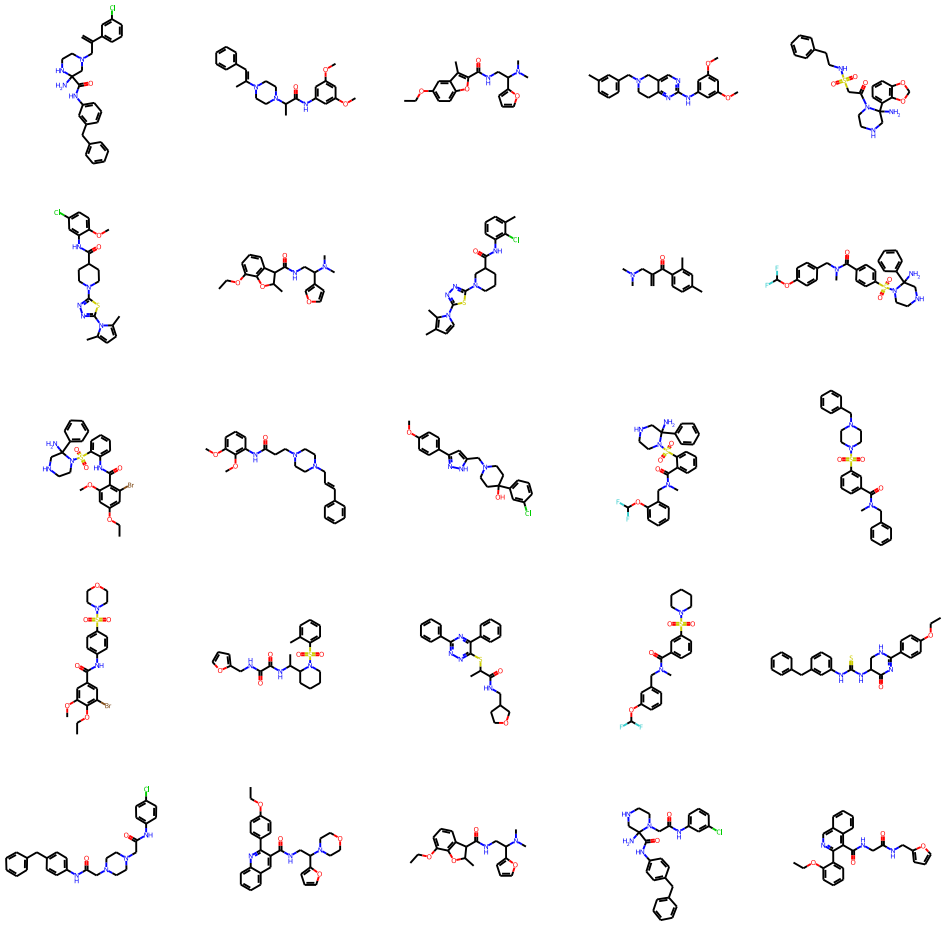

Figure A.16: 25 structures generated by P-JT-VAE ($p_{\mathrm{g}} = 0.0, p_{\mathrm{t}} = 0.25$).

References

[1] G. B. Olson. "Designing a new material world". In: *Science* 288.5468 (2000), pp. 993–998.

[2] W. Liu et al. "QSPR study on glass transition temperatures of polystyrenes by using Quantum-Chemistry descriptors". In: *Computers and Applied Chemistry* 22.9 (2005), p. 753.

[3] P. G. Seybold. "Quantum Chemical-QSPR Estimation of the Acidities and Basicities of Organic Compounds". In: *Advances in quantum chemistry* 64 (2012), pp. 83–104.

[4] B. Shin et al. "Self-attention based molecule representation for predicting drug-target interaction". In: *Machine Learning for Healthcare Conference*. PMLR. 2019, pp. 230–248.

[5] B. R. Beck et al. "Predicting commercially available antiviral drugs that may act on the novel coronavirus (SARS-CoV-2) through a drug-target interaction deep learning model". In: *Computational and structural biotechnology journal* 18 (2020), pp. 784–790.

[6] Y. Xu, J. Pei, and L. Lai. "Deep learning based regression and multiclass models for acute oral toxicity prediction with automatic chemical feature extraction". In: *Journal of Chemical Information and Modeling* 57.11 (2017), pp. 2672–2685.

[7] P. S. Kutchukian, D. Lou, and E. I. Shakhnovich. "FOG: Fragment Optimized Growth algorithm for the de novo generation of molecules occupying druglike chemical space". In: *Journal of chemical information and modeling* 49.7 (2009), pp. 1630–1642.

[8] A. Kadurin et al. "druGAN: an advanced generative adversarial autoencoder model for de novo generation of new molecules with desired molecular properties in silico". In: *Molecular pharmaceutics* 14.9 (2017), pp. 3098–3104.

[9] S. Zheng et al. "QBMG: quasi-biogenic molecule generator with deep recurrent neural network". In: *Journal of Cheminformatics* 11.1 (2019), pp. 1–12.

[10] S. R. Krishnan et al. "Accelerating de novo drug design against novel proteins using deep learning". In: *Journal of Chemical Information and Modeling* 61.2 (2021), pp. 621–630.

[11] P. Schwaller et al. ""Found in Translation": predicting outcomes of complex organic chemistry reactions using neural sequence-to-sequence models". In: *Chemical science* 9.28 (2018), pp. 6091–6098.

[12] P. Schwaller et al. "Molecular transformer: a model for uncertainty-calibrated chemical reaction prediction". In: *ACS central science* 5.9 (2019), pp. 1572–1583.

[13] C. W. Coley et al. "A graph-convolutional neural network model for the prediction of chemical reactivity". In: *Chemical science* 10.2 (2019), pp. 370–377.

[14] K. Do, T. Tran, and S. Venkatesh. "Graph transformation policy network for chemical reaction prediction". In: *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2019, pp. 750–760.

[15] K. Funatsu and S.-I. Sasaki. "Computer-assisted organic synthesis design and reaction prediction system,"AIPHOS"". In: *Tetrahedron Computer Methodology* 1.1 (1988), pp. 27–37.

[16] S. Zheng et al. "Predicting retrosynthetic reactions using self-corrected transformer neural networks". In: *Journal of Chemical Information and Modeling* 60.1 (2019), pp. 47–55.

[17] E. Kim et al. "Valid, Plausible, and Diverse Retrosynthesis Using Tied Two-Way Transformers with Latent Variables". In: *Journal of Chemical Information and Modeling* 61.1 (2021), pp. 123–133.

[18] C. Yan et al. "RetroXpert: Decompose retrosynthesis prediction like a chemist". In: *arXiv preprint arXiv:2011.02893* (2020).

[19] A. Nakao, H. Kaneko, and K. Funatsu. "Development of an adaptive experimental design method based on probability of achieving a target range through parallel experiments". In: *Industrial & Engineering Chemistry Research* 55.19 (2016), pp. 5726–5735.

[20] S. Takamoto et al. "PFP: Universal Neural Network Potential for Material Discovery". In: *arXiv preprint arXiv:2106.14583* (2021).

[21] G. V. Huerta et al. "Calculations of Real-System Nanoparticles Using Universal Neural Network Potential PFP". In: *arXiv preprint arXiv:2107.00963* (2021).

[22] A. A. Toropov and A. P. Toropova. "QSPR/QSAR: State-of-art, weirdness, the future". In: *Molecules* 25.6 (2020), p. 1292.

[23] E. N. Muratov et al. "QSAR without borders". In: *Chemical Society Reviews* 49.11 (2020), pp. 3525–3564.

[24] M. I. Skvortsova et al. "Inverse problem in QSAR/QSPR studies for the case of topological indexes characterizing molecular shape (Kier indices)". In: *Journal of Chemical Information and Computer Sciences* 33.4 (1993), pp. 630–634.

[25] T. Akutsu et al. "Inferring a graph from path frequency". In: *Discrete Applied Mathematics* 160.10-11 (2012), pp. 1416–1428.

[26] T. Miyao, H. Kaneko, and K. Funatsu. "Inverse QSPR/QSAR analysis for chemical structure generation (from y to x)". In: *Journal of chemical information and modeling* 56.2 (2016), pp. 286–299.

[27] Y. Shi et al. "An Inverse QSAR Method Based on a Two-Layered Model and Integer Programming". In: *International journal of molecular sciences* 22.6 (2021), p. 2847.

[28] C. Benecke et al. "MOLecular structure GENeration with MOLGEN, new features and future developments". In: *Fresenius' journal of analytical chemistry* 359.1 (1997), pp. 23–32.

[29] T. Grüner, R. Laue, and M. Meringer. "Algorithms for group actions: homomorphism principle and orderly generation applied to graphs". In: *DIMACS Series in Discrete Mathematics and Theoretical Computer Science* 28 (1997), pp. 113–122.

[30] Y. Ishida et al. "Improved algorithms for enumerating tree-like chemical graphs with given path frequency". In: *Genome Informatics 2008: Genome Informatics Series Vol. 21.* World Scientific, 2008, pp. 53–64.

[31] T. Fink and J.-L. Reymond. "Virtual exploration of the chemical universe up to 11 atoms of C, N, O, F: assembly of 26.4 million structures (110.9 million stereoisomers) and analysis for new ring systems, stereochemistry, physicochemical properties, compound classes, and drug discovery". In: *Journal of Chemical Information and Modeling* 47.2 (2007), pp. 342–353.

[32] L. C. Blum and J.-L. Reymond. "970 million druglike small molecules for virtual screening in the chemical universe database GDB-13". In: *Journal of the American Chemical Society* 131.25 (2009), pp. 8732–8733.

[33] L. Ruddigkeit et al. "Enumeration of 166 billion organic small molecules in the chemical universe database GDB-17". In: *Journal of Chemical Information and Modeling* 52.11 (2012), pp. 2864–2875.

[34] R. Wang, Y. Gao, and L. Lai. "LigBuilder: a multi-purpose program for structure-based drug design". In: *Molecular modeling annual* 6.7 (2000), pp. 498–516.

[35] Y. Yuan, J. Pei, and L. Lai. "LigBuilder 2: a practical de novo drug design approach". In: *Journal of chemical information and modeling* 51.5 (2011), pp. 1083–1091.

[36] Y. Yuan, J. Pei, and L. Lai. "Ligbuilder v3: a multi-target de novo drug design approach". In: *Frontiers in chemistry* 8 (2020), p. 142.

[37] T. Miyao, H. Kaneko, and K. Funatsu. "Ring system-based chemical graph generation for de novo molecular design". In: *Journal of computer-aided molecular design* 30.5 (2016), pp. 425–446.

[38] T. Miyao, H. Kaneko, and K. Funatsu. "Ring-System-Based Exhaustive Structure Generation for Inverse-QSPR/QSAR". In: *Molecular informatics* 33.11-12 (2014), pp. 764–778.

[39] X. Yang et al. "ChemTS: an efficient python library for de novo molecular generation". In: *Science and technology of advanced materials* 18.1 (2017), pp. 972–976.

[40] H. Dai et al. "Syntax-directed variational autoencoder for structured data". In: *arXiv preprint arXiv:1802.08786* (2018).

[41] O. Prykhodko et al. "A de novo molecular generation method using latent vector based generative adversarial network". In: *Journal of Cheminformatics* 11.1 (2019), pp. 1–13.

[42] Y. Li et al. "Learning deep generative models of graphs". In: *arXiv preprint arXiv:1803.03324* (2018).

[43] H. Kajino. "Molecular hypergraph grammar with its application to molecular optimization". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 3183–3191.

[44] J. You et al. "Graph convolutional policy network for goal-directed molecular graph generation". In: *Proceedings of the 32nd International Conference on Neural Information Processing Systems*. 2018, pp. 6412–6422.

[45] R. Gómez-Bombarelli et al. "Automatic chemical design using a data-driven continuous representation of molecules". In: *ACS central science* 4.2 (2018), pp. 268–276.

[46] C. Tan et al. "A survey on deep transfer learning". In: *International conference on artificial neural networks*. Springer. 2018, pp. 270–279.

[47] F. Zhuang et al. "A comprehensive survey on transfer learning". In: *Proceedings of the IEEE* 109.1 (2020), pp. 43–76.

[48] S. Kim et al. "PubChem in 2021: new data content and improved web interfaces". In: *Nucleic Acids Research* 49.D1 (2021), pp. D1388–D1395.

[49] J. J. Irwin et al. "ZINC20—A Free Ultralarge-Scale Chemical Database for Ligand Discovery". In: *Journal of Chemical Information and Modeling* (2020).

[50] C. Shorten and T. M. Khoshgoftaar. "A survey on image data augmentation for deep learning". In: *Journal of Big Data* 6.1 (2019), pp. 1–48.

[51] T. Inoue et al. "Improvement of the Structure Generator DAECS with Respect to Structural Diversity". In: *Molecular Informatics* 40.4 (2021), p. 2000225.

[52] K. Mishima, H. Kaneko, and K. Funatsu. "Development of a new de novo design algorithm for exploring chemical space". In: *Molecular Informatics* 33 (2014), pp. 779–789. ISSN: 18681751. DOI: 10.1002/minf.201400056.

[53] S. Takeda, H. Kaneko, and K. Funatsu. "Chemical-Space-Based de Novo Design Method to Generate Drug-Like Molecules". In: *Journal of Chemical Information and Modeling* 56 (2016), pp. 1885–1893. ISSN: 15205142. DOI: 10.1021/acs.jcim.6b00038.

[54] W. Jin, R. Barzilay, and T. Jaakkola. "Junction tree variational autoencoder for molecular graph generation". In: *International Conference on Machine Learning*. PMLR. 2018, pp. 2323–2332.

[55] C. M. Bishop and M. Svensén. "GTM: The Generative Topographic Mapping". In: *Neural Computation* 10.1 (1998), pp. 215–234. DOI: 10.1029/2004TC001640.

[56] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the royal statistical society. Series B (methodological)* (1977), pp. 1–38.

[57] D. Arthur and S. Vassilvitskii. "k-means++: The advantages of careful seeding". In: *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics. 2007, pp. 1027–1035.

[58] G. Landrum. *RDKit: Open-source cheminformatics*. (Accessed on 01/20/2021). URL: http://www.rdkit.org.

[59] D. Mendez et al. "ChEMBL: towards direct deposition of bioassay data". In: *Nucleic acids research* 47.D1 (2019), pp. D930–D940.

[60] H. Drucker et al. "Support Vector Regression Machines". In: *Advances in Neural Information Processing Systems 9*. Ed. by M. C. Mozer, M. I. Jordan, and T. Petsche. MIT Press, 1997, pp. 155–161.

[61] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[62] *BIOVIA Available Chemicals Directory*. URL: http://accelrys.co.jp/products/collaborative-science/biovia-available-chemicals-directory/.

[63] The GPyOpt Authors. *GPyOpt: A Bayesian Optimization framework in Python*. 2016. URL: http://github.com/SheffieldML/GPyOpt.

[64] GPy. *GPy: A Gaussian process framework in python*. since 2012. URL: http://github.com/SheffieldML/GPy.

[65] M. Arakawa, T. Miyao, and K. Funatsu. "Development of Drug-likeness Model and Its Visualization". In: *Journal of Computer Aided Chemistry* 9 (2008), pp. 70–80.

[66] H. L. Morgan. "The generation of a unique machine description for chemical structures-a technique developed at chemical abstracts service." In: *Journal of Chemical Documentation* 5.2 (1965), pp. 107–113.

[67] P. Ertl and A. Schuffenhauer. "Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions". In: *Journal of Cheminformatics* 1.1 (2009), pp. 1–11.

[68] D. Weininger. "SMILES, a chemical language and information system. 1. Introduction to methodology and encoding rules". In: *Journal of chemical information and computer sciences* 28.1 (1988), pp. 31–36.

[69] D. Weininger, A. Weininger, and J. L. Weininger. "SMILES. 2. Algorithm for generation of unique SMILES notation". In: *Journal of chemical information and computer sciences* 29.2 (1989), pp. 97–101.

[70] G. B. Goh et al. "Smiles2vec: An interpretable general-purpose deep neural network for predicting chemical properties". In: *arXiv preprint arXiv:1712.02034* (2017).

[71] M. Hirohara et al. "Convolutional neural network based on SMILES representation of compounds for detecting chemical motif". In: *BMC bioinformatics* 19.19 (2018), pp. 83–94.

[72] E. J. Bjerrum and B. Sattarov. "Improving chemical autoencoder latent space and molecular de novo generation diversity with heteroencoders". In: *Biomolecules* 8.4 (2018), p. 131.

[73] S. Honda, S. Shi, and H. R. Ueda. "SMILES transformer: pre-trained molecular fingerprint for low data drug discovery". In: *arXiv preprint arXiv:1911.04738* (2019).

[74] F. Scarselli et al. "The graph neural network model". In: *IEEE transactions on neural networks* 20.1 (2008), pp. 61–80.

[75] D. Duvenaud et al. "Convolutional networks on graphs for learning molecular fingerprints". In: *Proceedings of the 28th International Conference on Neural Information Processing Systems-Volume 2*. 2015, pp. 2224–2232.

[76] K. Yang et al. "Analyzing learned molecular representations for property prediction". In: *Journal of Chemical Information and Modeling* 59.8 (2019), pp. 3370–3388.

[77] Ł. Maziarka et al. "Molecule attention transformer". In: *arXiv preprint arXiv:2002.08264* (2020).

[78] R. Ramakrishnan et al. "Quantum chemistry structures and properties of 134 kilo molecules". In: *Scientific Data* 1 (2014).

[79] R. S. Simões et al. "Transfer and multi-task learning in QSAR modeling: advances and challenges". In: *Frontiers in pharmacology* 9 (2018), p. 74.

[80] W. Hu et al. "Strategies for pre-training graph neural networks". In: *arXiv preprint arXiv:1905.12265* (2019).

[81] P. Li et al. "Learn molecular representations from large-scale unlabeled molecules for drug discovery". In: *arXiv preprint arXiv:2012.11175* (2020).

[82] Y. Fang et al. "Knowledge-aware Contrastive Molecular Graph Learning". In: *arXiv preprint arXiv:2103.13047* (2021).

[83] E. J. Bjerrum. "SMILES enumeration as data augmentation for neural network modeling of molecules". In: *arXiv preprint arXiv:1703.07076* (2017).

[84] Y. Rong et al. "Dropedge: Towards deep graph convolutional networks on node classification". In: *arXiv preprint arXiv:1907.10903* (2019).

[85] D. Chen et al. "Measuring and relieving the over-smoothing problem for graph neural networks from the topological view". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 04. 2020, pp. 3438–3445.

[86] Y. Zhang et al. "Bayesian graph convolutional neural networks for semi-supervised classification". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 5829–5836.

[87] T. Zhao et al. "Data augmentation for graph neural networks". In: *arXiv preprint arXiv:2006.06830* (2020).

[88] J. Zhou, J. Shen, and Q. Xuan. "Data Augmentation for Graph Classification". In: *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*. 2020, pp. 2341–2344.

[89] K. Kong et al. "FLAG: Adversarial Data Augmentation for Graph Neural Networks". In: *arXiv preprint arXiv:2010.09891* (2020).

[90] H. W. Chung, A. Datta, and C. Waites. "GABO: Graph Augmentations with Bi-level Optimization". In: *arXiv preprint arXiv:2104.00722* (2021).

[91] J. Gilmer et al. "Neural message passing for quantum chemistry". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1263–1272.

[92] K. Cho et al. "Learning phrase representations using RNN encoder-decoder for statistical machine translation". In: *arXiv preprint arXiv:1406.1078* (2014).

[93] O. Vinyals, S. Bengio, and M. Kudlur. "Order matters: Sequence to sequence for sets". In: *arXiv preprint arXiv:1511.06391* (2015).

[94] S. Hochreiter and J. Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[95] A. Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035.

[96] M. Wang et al. "Deep Graph Library: A graph-centric, highly-performant package for graph neural networks". In: *arXiv preprint arXiv:1909.01315* (2019).

[97] M. Li et al. "DGL-LifeSci: An open-source toolkit for deep learning on graphs in life science". In: *arXiv preprint arXiv:2106.14232* (2021).

[98] Z. Wu et al. "MoleculeNet: a benchmark for molecular machine learning". In: *Chemical science* 9.2 (2018), pp. 513–530.

[99] L. Breiman. "Random forests". In: *Machine learning* 45.1 (2001), pp. 5–32.

[100] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[101] L. Prechelt. "Early stopping-but when?" In: *Neural Networks: Tricks of the trade*. Springer, 1998, pp. 55–69.

[102] T. N. Kipf and M. Welling. "Semi-supervised classification with graph convolutional networks". In: *arXiv preprint arXiv:1609.02907* (2016).

[103] K. Xu et al. "How powerful are graph neural networks?" In: *arXiv preprint arXiv:1810.00826* (2018).

[104] K. Oono and T. Suzuki. "Graph neural networks exponentially lose expressive power for node classification". In: *arXiv preprint arXiv:1905.10947* (2019).

[105] D. C. Elton et al. "Deep learning for molecular design—a review of the state of the art". In: *Molecular Systems Design & Engineering* 4.4 (2019), pp. 828–849.

[106] D. Schwalbe-Koda and R. Gómez-Bombarelli. "Generative models for automatic chemical design". In: *Machine Learning Meets Quantum Physics*. Springer, 2020, pp. 445–467.

[107] C. B. Browne et al. "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in games* 4.1 (2012), pp. 1–43.

[108] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato. "Grammar variational autoencoder". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 1945–1954.

[109] M. Krenn et al. "Self-Referencing Embedded Strings (SELFIES): A 100% robust molecular string representation". In: *Machine Learning: Science and Technology* 1.4 (2020), p. 045024.

[110] N. O'Boyle and A. Dalke. "DeepSMILES: An Adaptation of SMILES for Use in Machine-Learning of Chemical Structures". In: *ChemRxiv preprint* (2018).

[111] X. Xia et al. "Graph-based generative models for de Novo drug design". In: *Drug Discovery Today: Technologies* (2020).

[112] B. Samanta et al. "NeVAE: A deep generative model for molecular graphs". In: *Journal of machine learning research. 2020 Apr; 21 (114): 1-33* (2020).

[113] K. Madhawa et al. "GraphNVP: an Invertible Flow-based Model for Generating Molecular Graphs". In: *arXiv preprint arXiv:1905.11600* (2019).

[114] P. Lippe and E. Gavves. "Categorical normalizing flows via continuous transformations". In: *arXiv preprint arXiv:2006.09790* (2020).

[115] M. Liu et al. "GraphEBM: Molecular Graph Generation with Energy-Based Models". In: *Energy Based Models Workshop-ICLR 2021*. 2021.

[116] Q. Liu et al. “Constrained Graph Variational Autoencoders for Molecule Design”. In: *The Thirty-second Conference on Neural Information Processing Systems* (2018).

[117] C. Zang and F. Wang. “MoFlow: an invertible flow model for generating molecular graphs”. In: *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 2020, pp. 617–626.

[118] D. P. Kingma and M. Welling. “Auto-encoding variational bayes”. In: *arXiv preprint arXiv:1312.6114* (2013).

[119] 船津公人, 井上貴央, 西川大貴. **詳解 マテリアルズインフォマティクス — 有機・無機化学のための深層学習**. 近代科学社 Digital, 2021.

[120] M. Abadi et al. “Deep learning with differential privacy”. In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 308–318.

[121] W. Jin, R. Barzilay, and T. Jaakkola. “Hierarchical generation of molecular graphs using structural motifs”. In: *International Conference on Machine Learning*. PMLR. 2020, pp. 4839–4848.