Edit-Aware Generative Molecular Graph Autocompletion for Scaffold Input[(注†)]

Sheng HU†, Ichigaku TAKIGAWA†,††, and Chuan XIAO†††

† ICReDD, Hokkaido University Kita-ku, Sapporo, Japan
†† RIKEN Center for AIP Seika-cho, Kyoto, Japan
††† Graduate School of Information Science and Technology, Osaka University
1-5, Yamadaoka, Suita, Osaka, Japan
E-mail: †{hu.sheng,takigawa}@icredd.hokudai.ac.jp, ††chuanx@ist.osaka-u.ac.jp

Abstract We present a novel molecular graph generation method by auto-completing a privileged scaffold which represents a core graph substructure step-by-step. We propose a generative GNN model thus providing the ability to generate unseen molecular graphs outside the given training set. An edit-aware graph autocompletion paradigm that follows the "substructure-by-substructure" process is designed to complete the scaffold queries in multiple substructure adopt operations and allow meaningful edit operations to show the user's intention. Such operations enable the involvement of user decisions when interacting with a generative user-centered AI system, which differentiates our work from existing single-run generation paradigms. We also propose a scaffold trie for fast training pair augmentation or changing training models in real-time. Moreover, we design a top-k ranking function which considers the preferences on popularity and diversity for different applications, such as query compositions for graph database and drug discovery respectively. Such techniques enable human experts to synergistically interact with the generative models grounded on large data.

Key words Graph generation, graph neural networks, drug discovery

1 Introduction

The ultimate goal of modern drug discovery is to find the target molecules with desired chemical properties, while the potential chemical space of drug-like compounds is $10^{23} - 10^{60}$. Until recent years, such chemical space exploration was traditionally conducted by expert chemists and pharmacologists, along with huge time and monetary cost being devoted.

Visual graph query composition can assist modern drug discovery, which did not attract much attention compared with the success in the graph database community [3, 4, 7, 8, 17, 22, 23]. Instead of exploratory searching given a subgraph query in graph databases and showing matched graphs [22, 23], we prefer to use generative models to grow novel molecules on the given subgraph. In the applications of drug discovery, such a subgraph query is usually called a scaffold (i.e., privileged or bioactive scaffold), and performs as a core structure in the molecule to preserve the preferable bioactivity properties. The generated novel molecular graphs are supergraphs of the scaffold thus being guaranteed to contain the scaffold to reveal the chemical properties. Fixing the scaffold usually dramatically reduces the search space of the desired drug thus saving experts' time and cost.

Due to surprising success of deep neural network (DNN) models these days, two categories of representations used in DNN-based models emerge in the drug discovery domain. (1) simplified molecular input line entry system (SMILES) strings representation. Several early works [1, 5, 6, 15, 16] are proposed to learn the SMILES grammar using RNN architectures and then generate corresponding SMILES strings from the trained models. These methods have limitations to learning the unrelated grammar and thus have low chemical validity from generated SMILES. A recent trend is (2) undirected labeled graph representation [20]. It is more natural to learn the original graph structure by using graph neural networks (GNNs). This representation can easily achieve higher chemical validity. In this work, we adopt the graph-based representation along with GNN models as our generative models. GNN models are employed during the visual graph query composition process to generate completed candidates. A motivating example is shown in Figure 1.[(注2)]

[(注1)] This paper was accepted in DLG-AAAI'22 workshop.

[(注2)] While we only show an example for the scenario of drug discovery,

[Example 1] In Figure 1, a user wants to design a new molecule with a scaffold shown in **user's query**. This user first adopts a suggested candidate to complete the polygon and then erases a vertex to show the label on this vertex is different with his original intention. Then the system sends the input to the learned GNN and returns two candidates **rank1** and **rank2**, with detailed molecule properties (e.g., MW, logP and QED). The user can interact with **rank1** and **rank2**, such as adopt **rank1** as a new query or even erase a part. The new query will be taken as input and sent into the GNNs for further generations. This process repeats until a proper molecular graph is found.

2 Related Work

Scaffold-based Molecular Graph Generation. The idea of growing molecules on scaffolds using DNN models did not receive too much attention except the ScaffoldVAE model [14] and DeepScaffold [11]. ScaffoldVAE focused on growing side chains from Bemis-Murcko scaffolds [2], which is a special kind of scaffold that only preserves ring systems. DeepScaffold is similar with ScaffoldVAE but includes all subscaffolds in their dataset to provide the ability of growing a full molecule from a much smaller subscaffold. However, both above methods generate the final molecule in a single step without allowing users to edit on the intermediate graph to show their real intentions. This prevents the users, especially for expert chemists, from utilizing their reliable chemical intuitions and experience during the molecule design process. The generated molecules reported in both [14] and [11] also proved to be far from practical molecules used in real experimental chemistry.

In this work, we built a system GNNGAC (**GNN**-based **G**raph **A**uto**c**ompletion), to allow users to edit the intermediate graph candidates during the molecule design process in multiple steps, utilizing the edit operations to predict the user's real intention to improve effectiveness. We design an interactive substructure-by-substructure adopt process to verify this idea. This process guarantees the involvement of user decisions to interact with a generative user-centered AI system, which differentiates our work from previous studies that generate graphs in a single run [11,14]. Compared with generating a bunch of cluttered results, involving user decisions can exploit human's capability, i.e., domain-knowledge or experiences, to generate much more insightful candidates. Also, to make the utmost of graph training data set for efficient training, we design `scaffold-trie` for data augmentation as well as to efficiently train the GNN from the computer memory. A pairwise Tanimoto similarity-based top-k ranking algorithm is also proposed to enhance the practicability. We demonstrate our system using a real-world molecular dataset containing nearly one million graphs. Users can draw various scaffold queries in our prepared Web-based canvas and check the suggestion quality by themselves.

our autocompletion system can be simply applied to traditional graph query suggestions for graph databases.

3 Edit-Aware Graph Autocompletion

Scaffold input. Scaffolds are generally those subgraphs carrying important characteristics of molecules. The basis scaffolds of a molecule are usually the set of all unique ring systems in the molecule, while a ring system is defined as single/multiple rings sharing an internal bond. The graph representing a molecule itself is called a full molecular graph. Scaffolds can be extracted by utilizing general graph mining algorithms [9], but as for molecular applications, we use HierS [19] to obtain the scaffolds to keep in line with [11].

[Definition 1] (Scaffold input) Given a well-trained generative model $\mathcal{M}$ and a scaffold input q, the candidate graphs generated by $\mathcal{M}$ is a set $\mathcal{M}_q = \{\, g \mid q \subseteq g \,\}$, where $q \subseteq g$ means q is a subgraph of g.

Definition 1 guarantees that the generated graphs $\{g \mid g \in \mathcal{M}_q\}$ are supergraphs of q.

Autoregressive GNNs. We adopt the GNN model used in [11] and [13] for generating our graph candidates. The entire model architecture is shown in Figure 2. Compared with variational autoencoders (VAEs) or generative adversarial networks (GANs), autoregressive GNNs have unique ability to model edge dependencies, which guarantees the nodes are generated in a sequential way. We adopt a sequence-like graph knowledge representation used in [13] which builds a full molecular graph in a sequential fashion $(\langle g_0, t_0\rangle, \langle g_1, t_1\rangle, \cdots, \langle g_N, t_N\rangle)$, where g_n is a specific graph state (q equals g_0 here) and t_n is an action that transforms $g_n \rightarrow g_{n+1}$. Such a sequential molecular generative process is essentially a Markov decision processing thus being modeled as either Markovian or recurrent using a global recurrent neural network (RNN). In another word, the GNNs are used to decide whether to generate a new atom or bond along with its atom/bond type. In [13], three types of actions t_i are allowed to build a full molecule: (1) add an atom and connect it with an existing atom v (with a probability p_v^A), (2) connect an existing atom v to the new atom (with a probability p_v^C), and (3) terminate the generating process (with a probability p^*). According to Figure 2, the MLP layer outputs a tensor with a size $|V| \times (|A|+1) \times |B|$, where V is the atom node set that $v \in V$, A is the atom type set, and B is the bond type set. This tensor is further split into $[\mathbf{p}^A, \mathbf{p}^C] = [tensor^A_{|V|\times|A|\times|B|}, tensor^C_{|V|\times 1\times|B|}]$. On the

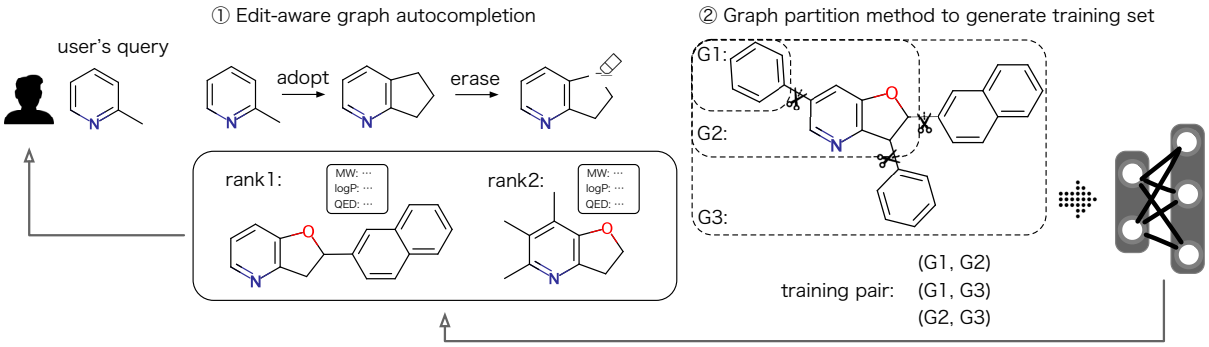

Figure 1 An Example of GNN-based Graph Autocompletion

other side, the MLP* layer outputs a scalar value $p*$ to represent the probability of termination. Eventually, after a softmax computation, $\{\mathbf{p}^A, \mathbf{p}^C, p*\}$ decides which action t_i should be taken when $g_{n+1} \leftarrow t_i(g_n)$.

User operations. When a user operates on a visual graph composition interface, there are some fundamental operations he/she can make. For example, edge addition is the most fundamental one. However, in autocompletion scenarios, the most efficient step is adopting a suitable graph suggestion to complete the current query which can accelerate the query composition dramatically [23]. Also, we consider that it is natural to allow the user to slightly modify the provided suggestion which produces the need of erasing and replacing a part of the graph. Without losing the possibility of extending in the future, we only consider three types of composition operation $\mathcal{OP} = \{\mathsf{adopt}, \mathsf{erase}, \mathsf{replace}\}$. Particularly, we call erase and replace as **edit** to allow users to perform specific operations on modifying the intermediate resulting graphs. The **edit** that user has performed is usually implicit indicators of his/her real intention. By adapting our edit-aware paradigm, the GNN model is able to take advantage of users' ideas, especially for those chemical experts who possess drug design experience. As replace can be taken as a special case of erase (e.g., replace an atom v with u can be considered as erase v first and add u from suggestions), for simplicity, we mainly focus on erase here. We also consider adopt as the context of erase to show more users' intentions. To be aware of the sequence composed by adopt and erase, we design different training methods.

Adopt training. The operation adopt means the user accepts a provided graph suggestion q' and incrementally adds a substructure Δq to current query q. To make sure q' has a strong correlation with the current query q, we need to create the training pair (q, q') and insert it into the training set. In another word, a simplest case is scaffold $\rightarrow q$ and molecule $\rightarrow q'$. Nevertheless, when a relatively larger molecule requires multiple steps to compose which results in a long composition sequence, more fine-grained training pair enumerations are expected, i.e., q and Δq becomes much smaller and the recursive case $q'' = q' + \Delta q$ must be considered.

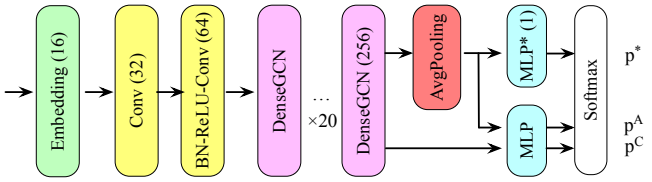

Figure 2 Model Architecture

Data augmentation. Instead of only using (scaffold $\rightarrow q$, molecule $\rightarrow q'$) pairs for training, we need to conduct data augmentation to enumerate subgraphs to handle cases such as $q'' = q + \Delta q + \cdots + \Delta q'$ For graph structures, such enumerations are impossible because of the combination explosion of subgraphs. Therefore, we decide to only preserve subgraphs with chemical significance (i.e., ring or chain systems). We choose to store each basis scaffold q and their incremental graphs $q'' = q + \Delta q + \cdots + \Delta q'$ in a trie structure according to the super-subgraph relationship in memory and assembly them as training pairs on-the-fly. The advantages of building the `scaffold-trie` is multifold: (1) the trie is memory-resident thus avoiding the I/O overhead which will deteriorate the training time-cost. (2) the granularity of increment size Δq can be controlled flexibly without rebuilding additional indices. (3) a single `scaffold-trie` can generate training pairs for different GNN models' training requirements beginning with different q for different problem settings such as macromolecules. We set a uniform threshold $\delta = |\Delta q|$ as a granularity constraint of incremental subgraphs that grows from previous intermediate graph q. The value δ can be fixed to a number or can be a variable as a ratio such as 1/2 size of the full molecular graph. To efficiently utilize the training set, we set $\delta = 1$ by default, i.e., we put the pair (q, q'') into the training set when $|\Delta q| = |q''| - |q| \geq 1$.

Moreover, generating the training set only requires for a single full traversal of the `scaffold-trie` thus leading to a time complexity of $\mathcal{O}(|T|)$ where $|T|$ represents the number

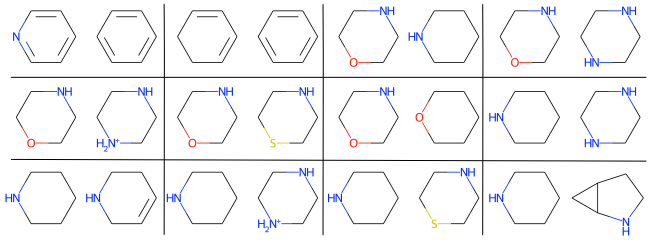

Figure 3 Graph Similarity Join Results with GED = 1

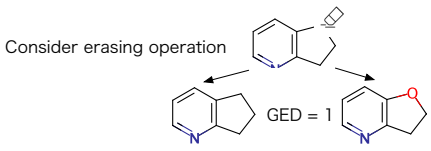

Figure 4 Erase training

of nodes in the trie. The training pair set is then taken as input fed into the GNN model for training purpose.

Erase training. We need to predict which erase edit is tending to happen and generate the corresponding training pairs. To achieve that, we generate training pairs that allow some slight mismatching to happen in the matched common fragment between basis scaffolds. The degree of similarity is measured using Graph Edit Distance (GED). We conduct a graph similarity join [24] in the scaffold set to find the scaffolds having a GED not greater than τ (default set to 1). By supposing the mismatched part is erased by users, the training is performed by inputting the training pair of (q_e, q), where q_e and q represents the one-node-erased graph (Figure 4 upper) and the original graph (Figure 4 lower) respectively.

Figure 3 shows examples of the resulting pairs after a graph similarity join with a GED equaling 1.

4 Top-k Diversifying and Ranking Results

4 1 Diversification

After running the inference process of our GNN for multiple times, we can obtain our results set R. When $|R|$ is a large number, displaying all the graphs in R will only mess up the interface with similar results. In this section, we propose a method to only choose top-k diversified graphs out of R to form the final result set R' thus $|R'| = k < |R|$.

We take advantage of the metric proposed in [19], which is called average pairwise Tanimoto (APT). Here, Tanimoto means Tanimoto coefficient which is computed using molecular fingerprint to describe how similar two molecular graphs are. By adopting APT, we sum up the computed Tanimoto coefficients between each pair of graphs appearing in R, and divide the sum by total number of pairs as shown in Equation 1.

$$APT(R) = \frac{1}{|R|(|R|-1)} \sum_{i \neq j}^{|R|} \frac{b_{i\&j}}{b_i + b_j - b_{i\&j}} \quad (1)$$

where $|R|$ represents the size of result set R, $b_i(b_j)$ is the number of bits set to 1 in result i's(j's) fingerprint, $b_{i\&j}$ means the number of bits set to 1 in the the intersection of i and j's fingerprints.

4 2 Top-k Ranking

Along with diversification, we also need to rank the results according to the substructure popularity to assist the users in easily adding the most possible subgraphs. Here, we adapt a statistic $\mathsf{sel}_\Delta(q')$ used in [23] which represents the number of supergraphs of the increments ($\Delta q'$) in the database (training set) D. This can ensure that novel molecular graphs generated will not have low ranks (which are not contained in the database). Differently, we further extend the $\Delta q'$ as $\Delta q' \cup q_\Delta$, where q_Δ represents the smallest subgraph in q which connects $\Delta q'$ to avoid meaningless increments. Note that $\mathsf{sel}_\Delta(q')$ can be efficiently computed offline using graph indexing techniques proposed in [22].

The final ranking function (util) can be written as below:

$$\mathsf{util}(R') = \frac{w}{k} \sum_{q' \in R'} \mathsf{sel}_\Delta(q') + (1-w) APT(R') \quad (2)$$

where a weight parameter $w \in [0, 1]$ is added to balance the popularity and diversity scores.

The value of w also has influences on different applications. For example, for traditional graph query suggestions on database search, a larger w is expected to add more frequent subgraph fragments to formulate the query quickly. On the contrary, drug discovery requires a smaller w to improve the diversity for generating novel molecules.

Finding the most optimal set R' out of R means $\mathsf{util}(R')$ should be larger than any other alternative set R'' with the same set size. This process proved to be a NP-hard problem, which is a reduction from the maximum independent set problem shown in [22]. Similar with [22], we use a greedy algorithm (Algorithm 1) to obtain the optimal solution. When $|R'| = k$, the time complexity of finding the APT(R') is $\mathcal{O}(k \times |R| \times S_{Tanimoto})$, where $|R|$ represents the unique graphs generated and $S_{Tanimoto}$ means the Tanimoto similarity computation.

5 Experiments

The experiments were carried out on a Linux server with an Intel Xeon Silver 4214 2.20GHz Processor and 192GB RAM, running Ubuntu 18.04.5. The training details can be found in Appendix. Both adopt training and erase training are finished in 16 hours. Other training statistics are shown

Algorithm 1: Top-k (R)

```
R' ← ∅ ;                                   /* final set R' */
foreach ⟨i, j⟩ ∈ R do
    Compute S_Tanimoto(i, j);
Choose the largest S_Tanimoto(i, j) and R' ← R' ∪ i ∪ j;
for n = 1 ··· k − 2 do
    Find an i' to make the util(R' ∪ i') the largest;
    R' ← R' ∪ i';
return R';
```

Table 1 Training Statistics

Training set	Training time	# of q	# of q'
Adopt	16 hours	303,706	1,271,948
Erase	16 hours	87,106	

in Table 1. # of q and # of q' represents the number of unique graphs used in $\langle q, q' \rangle$ pairs where $q \subseteq q'$. In erase training model, we used 87,106 similar pairs of graphs having GED $\tau = 1$. For the top-k ranking, we set the weight $w = 0.4$ as it performs the best on the query sets.

Training set. **ChEMBL**[注3] is a public dataset containing both structural and bioactivity information for molecules. We only kept molecules that containing elements $\{H, C, N, O, F, P, S, Cl, Br, I\}$ and eliminated those whose Quantitative Estimate of Drug-likeness (QED) are lower than 0.5. After that, we totally collected 923,786 unique molecules. The whole **ChEMBL** dataset is used as the training data in the GNN training process.

Query set. For the purpose of simulating the drug discovery process, we employed a different dataset **PubChem** [7] as the target query set to avoid the graph distribution bias. The purpose of the simulation is to generate each target query from the scaffold input. We used three query sets from [22] by extracting subsets from **PubChem**, each of which contains 100 query graphs, with the query size ranging from 8 to 16 (Q8, Q12, Q16) to keep in line with [22]. The query size stands for the number of edges here. We replaced those graphs having appeared in the training data to ensure each query is unseen in the training set. All experimental results are computed by averaging the sum on each query set, respectively.

5 1 Suggestion quality evaluations

Simulation experiment setup.

To generate each query molecular graph, we started with a random basic scaffold graph. In each step, we sent the intermediate graph as the input into GNNGAC. Next, the largest useful graph suggestion was selected. In the case that some suggestions had a GED equivalent to 1 with the query graph, we used the trained erase model to simulate the erase operations. In the case that no suggestion was useful, a random edge towards the query molecular graph will be added to current graph. The completion will be a success if the query molecular graph is reconstructed by the generated graphs. Particularly, when a generated graph is a super graph of the query molecular graph, it is also taken as a success because such as a case is especially meaningful in drug discovery settings. The number of generations in each inference process is set to 1,000 according to the practical response time experimental results (details in Appendix) to meet the interactive time requirement.

[注3] https://www.ebi.ac.uk/chembl/

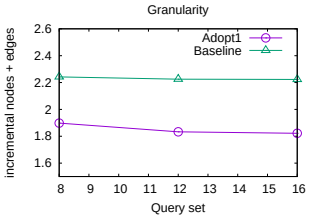

Figure 5 Granularity of Increments

To evaluate the efficiency and effectiveness thoroughly, we adopt the same evaluation metrics used in existing studies [11, 22]. We evaluate: (1) **Granularity and success rate**. Granularity, which represents the incremental subgraph size on average, measures how much a graph suggestion can extend the current scaffold. The incremental subgraph size is computed by summing the number of incremental nodes and edges. Small granularity leaves more choices for the users to choose and edit on. Large granularity causes more radical suggestions and might fail to return any useful result. We use success rate to show the proportion of queries where at least one useful suggestion is provided. (2) $|\Delta E|$ and **TPM**. $|\Delta E|$ represents the average number of useful edges provided by the adopted suggestions until the generation ends. TPM (total profit metric) is a metric taken from [21], which calculates a ratio of saved mouse clicks to all the clicks input manually. (3) **Chemical validity and uniqueness**. We check if a molecular graph is chemically valid using the Chem.Sanitize API in RDKit [10]. Then the validity and uniqueness are computed by the ratio of number of valid (unique) molecules to the total number of generations, respectively.

As we found that our GNNGAC finished the correct suggestions on most queries within two steps of adopt, we only show the results of the first and second adopt operations as Adopt1 and Adopt2.

Autocompletion granularity and success rate.

We compare with the baseline method [11] which produces

Table 2 Success Rate

Method	baseline	GNNGAC
success rate	63%	**100%**

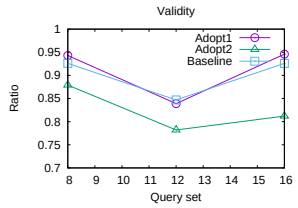

Figure 6 Validity

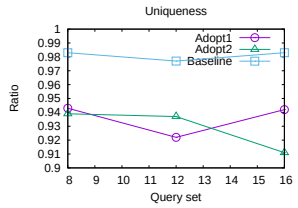

Figure 7 Uniqueness

Table 3 Autocompletion Effectiveness

$\|q\|$	$\|\Delta E\|$-AutoG	$\|\Delta E\|$-ours	TPM-AutoG	TPM-ours
8	3.3	**18.0**	45%	**73**%
12	5.1	**16.1**	44%	**61**%
16	6.7	**18.1**	42%	**72**%

a full molecular graph *all-at-once* in a single run. To avoid the large molecular graphs generated which dominate the average number, we only sum the sizes of the smallest 10% incremental graphs. The result in Figure 5 shows that the GNNGAC method can indeed generate fine suggestions compared with the baseline method thanks to the fine-grained data augmentation from the scaffold-trie. The incremental subgraph size is reduced by nearly 20%. Smaller increments mean less risk to cause a suggestion failure. This is reflected by the success rate in Table 2, where GNNGAC has a 37% lead. This is because the *all-at-once* strategy usually generates larger molecules which might save a lot of clicks if it finds the answer but is also easier to fail if missing the answer. Also, GNNGAC allows erase operation thus helping those mismatched suggestions with *GED*=1 revive as useful suggestions which improve the success rate by much.

$|\Delta E|$ and TPM.

For such metrics, we compare with AutoG [22] which is the state-of-the-art approach of graph query autocompletion for graph databases (details in Appendix). We do not compare with [11] because it is meaningless under different success rates. Note that $|\Delta E|$ might be greater than the size of $|q|$ because when a generated molecular graph contains the query graph, it will be a successful completion. Generating a super graph of the query graph is especially meaningful in drug discovery settings. The results in Figure 3 show that our GNNGAC method can provide much more useful edges than AutoG, which is mainly because traditional database indexing techniques can only generate very small graph fragments and glue them on the current scaffold in a enumerated way. Our GNNGAC also has a large lead on TPM over AutoG, showing that our method can save more clicks for the users.

Autocompletion validity and uniqueness.

RDKit tools [10] are utilized to check validity and uniqueness of generated molecules. We compare with the baseline method [11] which produces a full molecular graph *all-at-once* in a single run. The result in Figure 5 1 shows that the GNNGAC method sacrifices some validity performance on Adopt2 because the output scaffolds from Adopt1 might contain very rare results. The edit operations from users may also break the validity after Adopt1. Overall, GNNGAC does not deteriorate too much ($< 15\%$) compared with the *all-at-once* approach after interacting with the users' editing. The decreases on uniqueness are due to the small granularity which causes more small intermediate graphs. As we are not going to find the correct molecules in a single run, such decreases ($< 10\%$) are acceptable.

Acknowledgment

This work is partly supported by JSPS KAKENHI Grant Number JP21K17745, JP21K12041, JP20H00323, JP20H00605, JP20H05962, JP17H06099, JP18H04093, JP19K11979 and JST CREST Grant Number JPMJCR18K.

References

[1] J. Arús-Pous, A. Patronov, E. J. Bjerrum, C. Tyrchan, J.-L. Reymond, H. Chen, and O. Engkvist. Smiles-based deep generative scaffold decorator for de-novo drug design. *Journal of Cheminformatics*, 12(1):1–18, 2020.

[2] G. W. Bemis and M. A. Murcko. The properties of known drugs. 1. molecular frameworks. *Journal of medicinal chemistry*, 39(15):2887–2893, 1996.

[3] S. S. Bhowmick, B. Choi, and C. Dyreson. Data-driven visual graph query interface construction and maintenance: Challenges and opportunities. *PVLDB*, 9(12):984–992, Aug. 2016.

[4] S. S. Bhowmick, B. Choi, and C. Li. Graph querying meets hci: State of the art and future directions. In *ACM SIGMOD 2017*, page 1731–1736, New York, NY, USA, 2017.

[5] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song. Syntax-directed variational autoencoder for structured data. *ICLR 2018*, 2018.

[6] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS central science*, 4(2):268–276, 2018.

[7] W. Han, J. Lee, M. Pham, and J. X. Yu. igraph: A framework for comparisons of disk-based graph indexing techniques.

Proc. VLDB Endow., 3(1):449–459, 2010.

[8] K. Huang, H. Chua, S. S. Bhowmick, B. Choi, and S. Zhou. CATAPULT: data-driven selection of canned patterns for efficient visual graph query formulation. In P. A. Boncz, S. Manegold, A. Ailamaki, A. Deshpande, and T. Kraska, editors, *ACM SIGMOD 2019*, pages 900–917, 2019.

[9] W. Jin, R. Barzilay, and T. Jaakkola. Junction tree variational autoencoder for molecular graph generation. *ICML 2018*, 2018.

[10] G. Landrum. Rdkit: Open-source cheminformatics software. 2016.

[11] Y. Li, J. Hu, Y. Wang, J. Zhou, L. Zhang, and Z. Liu. Deepscaffold: a comprehensive tool for scaffold-based de novo drug discovery using deep learning. *Journal of Chemical Information and Modeling*, 60(1):77–91, 2019.

[12] Y. Li, O. Vinyals, C. Dyer, R. Pascanu, and P. W. Battaglia. Learning deep generative models of graphs. *CoRR*, abs/1803.03324, 2018.

[13] Y. Li, L. Zhang, and Z. Liu. Multi-objective de novo drug design with conditional graph generative model. *Journal of cheminformatics*, 10(1):33, 2018.

[14] J. Lim, S.-Y. Hwang, S. Moon, S. Kim, and W. Y. Kim. Scaffold-based molecular design with a graph generative model. *Chemical Science*, 11(4):1153–1164, 2020.

[15] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen. Molecular de-novo design through deep reinforcement learning. *Journal of cheminformatics*, 9(1):48, 2017.

[16] M. H. Segler, T. Kogej, C. Tyrchan, and M. P. Waller. Generating focused molecule libraries for drug discovery with recurrent neural networks. *ACS central science*, 4(1):120–131, 2018.

[17] C. Wang, M. Xie, S. S. Bhowmick, B. Choi, X. Xiao, and S. Zhou. Ferrari: an efficient framework for visual exploratory subgraph search in graph databases. *The VLDBJ*, pages 1–26, 2020.

[18] S. Wang, W. Guo, H. Gao, and B. Long. Efficient neural query auto completion. In *ACM CIKM 2020*, pages 2797–2804, 2020.

[19] S. J. Wilkens, J. Janes, and A. I. Su. Hiers: hierarchical scaffold clustering using topological chemical graphs. *Journal of medicinal chemistry*, 48(9):3182–3193, 2005.

[20] X. Xia, J. Hu, Y. Wang, L. Zhang, and Z. Liu. Graph-based generative models for de novo drug design. *Drug Discovery Today: Technologies*, 2020.

[21] X. Xie, Z. Fan, B. Choi, P. Yi, S. S. Bhowmick, and S. Zhou. PIGEON: progress indicator for subgraph queries. In J. Gehrke, W. Lehner, K. Shim, S. K. Cha, and G. M. Lohman, editors, *31st IEEE International Conference on Data Engineering, ICDE 2015, Seoul, South Korea, April 13-17, 2015*, pages 1492–1495. IEEE Computer Society, 2015.

[22] P. Yi, B. Choi, S. S. Bhowmick, and J. Xu. Autog: a visual query autocompletion framework for graph databases. *The VLDBJ*, 26(3):347–372, 2017.

[23] P. Yi, B. Choi, Z. Zhang, S. S. Bhowmick, and J. Xu. Gfocus: User focus-based graph query autocompletion. *IEEE TKDE*, pages 1–1, 2020.

[24] X. Zhao, C. Xiao, X. Lin, W. Wang, and Y. Ishikawa. Efficient processing of graph similarity queries with edit distance constraints. *The VLDB Journal*, 22(6):727–752, 2013.

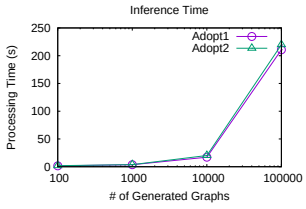

Figure A·1 Inference time

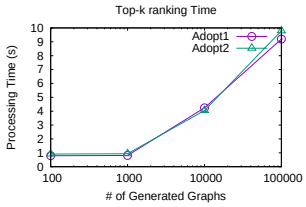

Figure A·2 Ranking time

付　　録

1 Appendix

Training details. The training is performed on two Nvidia Quadro RTX 8000 GPUs. The GNN training algorithms were implemented in PyTorch while `scaffold-trie` is a memory-resident index implemented in C++. We adopt the default parameters used in the autoregressive model [11] with a 20-layer DenseNet, setting growth rate and bottleneck layer output size as 24 and 96, respectively. The model is optimized with a log-likelihood loss function used in [13] and [12]. Hyperparameter k_{hyper} and α that represents # of samples generated from importance sampling and the degree of uncertainty in route sampling are set to 5 and 0.5, respectively.

System response time. We report the time-cost of the inference phase on average of the whole query set of Q8 as shown in Figure 1. An approximate linear growth w.r.t. the # of generated graphs is witnessed for both Adopt1 and Adopt2 models. Erase is not plotted because of negligible time-cost ($< 1s$) compared with adopt. Consider the practical scenario, choosing to generate 1,000 graphs for a adopt operation seems ideal as it results in 3.6s and 3.8s for Adopt1 and Adopt2, respectively. Time-cost of top-k ranking is reported in Figure 1. Similar with inference time, we observed a nearly linear growth on the ranking time when # of generated graphs is varied. When # of generated graphs equals 1,000, the time is nearly 1s thus being practical in real-world scenario. We set # of generated graphs to 1,000 in other experiments.

Graph Query Autocompletion for Graph Databases. The problem of graph query autocompletion (GQAC) originates from the domain of query formulation on graph databases [3, 4, 18]. The aim of GQAC system is to provide graph query suggestions given a partially-composed graph search query on a visual interface (GUI), especially for the search in a large graph database. Yi *et al.* [22] proposed the first GQAC system, AutoG, to index frequent graph features and how they combine together using a network index. After that, they improved the ranking method of AutoG and proposed GFocus [23], which can utilize focus patterns from users to improve the effectiveness. However, the graph suggestions provided are combinations of frequent graph fragments appearing in the graph databases, thus lacking the ability to generate novel graphs outside the databases.

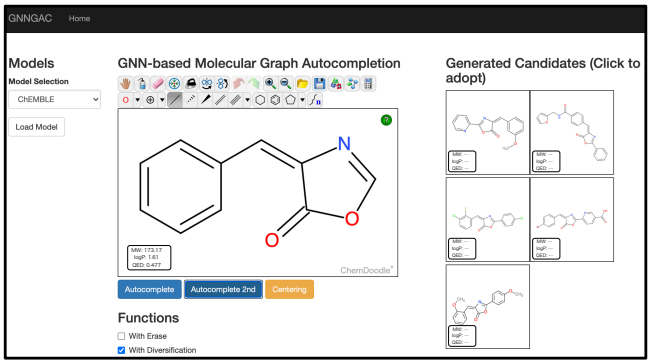

Figure A·3 A Screenshot

2 System Architecture

Front end. The front end is a Web-based graphical interface for drawing queries and navigating through graph candidates. The editor is a canvas where users can input the scaffold queries using various canned patterns or drawing edge-by-edge. Particularly, erase operation is listened to trigger the stroke predictions after erasing. The user can click on the candidate to adopt the completion and add it to the working canvas.

Back End. Four modules are included in the back end. (1) GNN module, (2) Graph generation module, (3) Top-k ranking module, and (4) Query logging module.

3 Demonstrations

A hands-on demo. To verify the idea of multi-step graph generations, we prepared a demonstration system. We set up the system as shown in Figure A·3. In Figure A·3, button Autocomplete means adopt1 operation and Autocomplete 2nd will trigger adopt2 operation. After checking the checkbox With Erase, the erase toolkit will trigger erase operation. In this demo, we aim to show that GNNGAC gives graph suggestions (right panels in Figure A·3) in an interactive way. The users can take the reference of the chemical properties (MW, logP, QED) for judgement in drug discovery.