Electronic Supplementary Information for:

Scaffold-based molecular design with a graph generative model

Jaechang Lim,†[a] Sang-Yeon Hwang,†[a]
Seokhyun Moon,[a] Seungsu Kim[b] and Woo Youn Kim*[a,c]

[a] *Department of Chemistry, KAIST, Daejeon 34141, Republic of Korea*
[b] *School of Computing, KAIST, Daejeon 34141, Republic of Korea*
[c] *KI for Artificial Intelligence, KAIST, Daejeon 34141, Republic of Korea*
* Email: wooyoun@kaist.ac.kr
† These authors contributed equally to this work.

Algorithm

Algorithm 1 shows the full process of encoding and decoding molecular graphs in our model. There, concat denotes the vector concatenation, Cat denotes a categorical distribution, and $\circ$ denotes the function composition. The role and operation of other modules are introduced in the main text and will be detailed below.

Algorithm 1 Scaffold-based graph generation

Inputs: G, S, $\mathbf{y}$, $\mathbf{y}_S$ ▷ Whole/scaffold graphs and properties

$G_0 \leftarrow S$

$\tilde{\mathbf{y}} \leftarrow \text{concat}\left(\mathbf{y}, \mathbf{y}_S\right)$

if $G \neq (\emptyset, \emptyset)$ **then** ▷ Learning phase

  $\left(\mathbf{H}_{V(G)}, \mathbf{H}_{E(G)}\right) \leftarrow \mathsf{embed}(G)$

  $\mathbf{H}_{V(G)} \leftarrow \mathsf{propagate}^{(k)}\left(\mathbf{H}_{V(G)}, \mathbf{H}_{E(G)}, \tilde{\mathbf{y}}\right)$

  $\mathbf{z} \sim \mathsf{reparam} \circ \mathsf{readout}\left(\mathbf{H}_{V(G)}\right)$ ▷ Vector representation of the target graph

else

  $\mathbf{z} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}\right)$ ▷ Generation phase

end if

$\tilde{\mathbf{z}} \leftarrow \text{concat}\left(\mathbf{z}, \tilde{\mathbf{y}}\right)$

$\left(\mathbf{H}_{V(G_0)}, \mathbf{H}_{E(G_0)}\right) \leftarrow \mathsf{embed}\left(G_0\right)$ ▷ Node and edge feature vectors

$\mathbf{H}_{V(G_0)} \leftarrow \mathsf{propagate}^{(k)}\left(\mathbf{H}_{V(G_0)}, \mathbf{H}_{E(G_0)}, \tilde{\mathbf{y}}\right)$ ▷ Initial update of the scaffold nodes

$t \leftarrow 1$ ▷ Node addition counter

$v_t \sim \text{Cat} \circ \mathsf{addNode}\left(\mathbf{H}_{V(G_{t-1})}, \mathbf{H}_{E(G_{t-1})}, \tilde{\mathbf{z}}\right)$ ▷ Sample a node type or STOP

while $v_t \neq \mathsf{STOP}$ **do**

  $V(G_t) \leftarrow V(G_{t-1}) \cup \{v_t\}$ ▷ Add the new node

  $\mathbf{H}_{V(G_t)} \leftarrow \mathbf{H}_{V(G_{t-1})} \cup \left\{\mathsf{initNode}\left(v_t, \mathbf{H}_{V(G_{t-1})}\right)\right\}$ ▷ Initialize and add a new node vector

  $E_{t,0} \leftarrow E(G_{t-1})$; $\mathbf{H}_{E_{t,0}} \leftarrow \mathbf{H}_{E(G_{t-1})}$ ▷ Prepare edge additions

  $i \leftarrow 1$ ▷ Edge addition counter

  $e_{t,i} \sim \text{Cat} \circ \mathsf{addEdge}\left(\mathbf{H}_{V(G_t)}, \mathbf{H}_{E_{t,i-1}}, \tilde{\mathbf{z}}\right)$ ▷ Sample an edge type or STOP

  while $e_{t,i} \neq \mathsf{STOP}$ **do**

    $v_{t,i} \sim \text{Cat} \circ \mathsf{selectNode}\left(\mathbf{H}_{V(G_t)}, \mathbf{H}_{E_{t,i-1}}, \tilde{\mathbf{z}}\right)$ ▷ Sample a node to connect

    $E_{t,i} \leftarrow E_{t,i-1} \cup \left\{\left(v_t, v_{t,i}\right)\right\}$ ▷ Add the new edge (with type $e_{t,i}$)

    $\mathbf{H}_{E_{t,i}} \leftarrow \mathbf{H}_{E_{t,i-1}} \cup \left\{\mathsf{initEdge}\left(e_{t,i}, \mathbf{H}_{V(G_t)}\right)\right\}$ ▷ Initialize and add a new edge vector

    $i \leftarrow i + 1$

    $e_{t,i} \sim \text{Cat} \circ \mathsf{addEdge}\left(\mathbf{H}_{V(G_t)}, \mathbf{H}_{E_{t,i-1}}, \tilde{\mathbf{z}}\right)$ ▷ Sample a next edge type or STOP

  end while

  $\mathbf{H}_{E(G_t)} \leftarrow \mathbf{H}_{E_{t,i-1}}$

  $E(G_t) \leftarrow E_{t,i-1}$

  $G_t \leftarrow \left(V(G_t), E(G_t)\right)$

  $t \leftarrow t + 1$

  $v_t \sim \text{Cat} \circ \mathsf{addNode}\left(\mathbf{H}_{V(G_{t-1})}, \mathbf{H}_{E(G_{t-1})}, \tilde{\mathbf{z}}\right)$ ▷ Sample a next node type or STOP

end while

$G_t^* \sim \text{Cat} \circ \mathsf{selectIsomer}\left(G_t, \tilde{\mathbf{z}}\right)$ ▷ Assign the stereoisomerism

return G_t^*

Graph representation of molecules

In our graph representation $G = (V(G), E(G))$ of a molecule, the nodes $v \in V(G)$ represent the atoms, and the edges $(v, w) \in E(G)$ represent the bonds. We regard each node as attributed with an atom type and each edge as attributed with a bond type.

In the present work, we used the atom types in an indexed family $\mathcal{A} = (A_i) =$ (C, N, O, F, P, S, Cl, Br) and the bond types in another indexed family $\mathcal{B} = (B_i) =$ (single-bond, double-done, triple-bond). The symbols in $\mathcal{A}$ indicate the corresponding elements in the periodic table. For the initial representation of whole-molecules and scaffolds, we use an extended family $\mathcal{A}^*$, which includes all the elements of $\mathcal{A}$ and additionally chirality (R, S, or none), formal charge, and aromaticity; also, we use an extended family $\mathcal{B}^*$, which includes the three bond types and stereoisomerism (E, Z, *cis*, *trans*, or none). We used RDKit [1] to preprocess molecules into graphs.

To prepare node feature vectors $\mathbf{h}_v$ and edge feature vectors $\mathbf{h}_{vw}$, we embed node and edge types via two networks:

$$\mathbf{h}_v = \mathrm{MLP}^n\left(\mathbf{h}_v^{0*}\right) \tag{1}$$

$$\mathbf{h}_{vw} = \mathrm{MLP}^e\left(\mathbf{h}_{vw}^{0*}\right). \tag{2}$$

$\mathbf{h}_v^{0*}$ is a raw feature vector representing the type of v based on $\mathcal{A}^*$, and similarly $\mathbf{h}_{vw}^{0*}$ is a raw feature vector of (v, w) based on $\mathcal{B}^*$. For each of MLP^n and MLP^e, we used a single linear layer with output dimension 128. The result of embedding all elements of a graph G becomes

$$\left(\mathbf{H}_{V(G)}, \mathbf{H}_{E(G)}\right) = \mathsf{embed}\left(G\right). \tag{3}$$

We use the same module embed to embed all whole-molecules, scaffolds, and stereoisomers.

Graph propagation and readout

The graph propagation module

$$\mathbf{H}'_{V(G)} = \mathsf{propagate}\left(\mathbf{H}_{V(G)}, \mathbf{H}_{E(G)}, \mathbf{c}\right) \tag{4}$$

consists of the following processes:

$$\mathbf{m}_{u\to v} = \mathrm{ReLU} \circ \mathrm{MLP}^m \circ \mathrm{concat}\left(\mathbf{h}_u, \mathbf{h}_v, \mathbf{h}_{uv}, \mathbf{c}\right) \tag{5}$$

$$\mathbf{m}_v = \sum_{u:(u,v)\in E(G)} \mathbf{m}_{u\to v} \tag{6}$$

$$\mathbf{h}'_v = \mathrm{GRUCell}\left(\mathbf{m}_v, \mathbf{h}_v\right), \tag{7}$$

where $\circ$ is the function composition, ReLU is the rectified linear unit [2], $\mathbf{c}$ is a condition vector, and GRUCell is a gated recurrent unit cell [3] (accepting $\mathbf{m}_v$ as the input and $\mathbf{h}_v$ as the hidden state). For MLP^m we used one linear layer with output dimension 128. We had MLP^m and GRUCell use a different set of parameters in different rounds of iterated propagation.

The readout module summarizes node features via the gated pooling:

$$\begin{aligned}\mathbf{h}_G &= \mathsf{readout}\left(\mathbf{H}_{V(G)}\right) \\ &= \frac{1}{|V(G)|}\sum_{v\in V(G)} \sigma\big(\mathrm{MLP}_2^r\left(\mathbf{h}_v\right)\big) \odot \mathrm{MLP}_1^r\left(\mathbf{h}_v\right),\end{aligned} \tag{8}$$

where σ is the sigmoid function, and $\odot$ is the elementwise product. For each of MLP_1^r and MLP_2^r, we used a single linear layer.

We had propagate and readout in different modules have different sets of parameters. For instance, all propagate involved in addNode, addEdge, selectNode, and selectIsomer have different MLP^m and GRUCell. In addition, we used two different output dimensions for readout: when reading-out the node features of any transient graph in the building process, we set the dimension equal to that of a node feature vector (i.e., 128); when encoding a whole-molecule graph, we set double.

Encoding

With a whole-molecule graph G, we sample a latent vector $\mathbf{z}$ by applying the reparametrization trick [4]:

$$\boldsymbol{\mu}_G = \mathrm{MLP}^{\mu}\left(\mathbf{h}_G\right) \tag{9}$$

$$\boldsymbol{\sigma}_G = \exp\left\{\mathrm{MLP}^{\sigma}\left(\mathbf{h}_G\right)/2\right\} \tag{10}$$

$$\boldsymbol{\epsilon} \sim \mathcal{N}\left(\mathbf{0}, \mathbf{I}\right) \tag{11}$$

$$\mathbf{z} = \boldsymbol{\mu}_G + \sigma_G \odot \boldsymbol{\epsilon}, \tag{12}$$

where $\mathcal{N}\left(\mathbf{0}, \mathbf{I}\right)$ is the standard normal distribution. In the last line we omitted the graph dependence of $\mathbf{z}$ for simplicity. For each of MLP^{μ} and MLP^{σ}, we used one linear layer with output dimension 128. Note that in Algorithm 1 we used $\mathsf{reparam}$ to concisely express the sampling process.

Decoder modules

The node addition module computes atom type probabilities as

$$\begin{aligned}\hat{\mathbf{p}}^{an} &= \mathsf{addNode}\left(\mathbf{H}_{V(G_t)}, \mathbf{H}_{E(G_t)}, \mathbf{z}\right) \\ &= \mathrm{softmax} \circ \mathrm{MLP}^{an} \circ \mathrm{concat}\Big(\mathsf{readout} \circ \mathsf{propagate}^{(k)}\left(\mathbf{H}_{V(G_t)}, \mathbf{H}_{E(G_t)}, \mathbf{z}\right), \mathbf{z}\Big),\end{aligned} \tag{13}$$

where G_t is a transient graph in the building process. For MLP^{an}, we used three linear layers with ReLU activations. The output dimensions of the layers were all 128. The length of the vector $\hat{\mathbf{p}}^{an}$ is $|\mathcal{A}| + 1$. The computed probabilities define a categorical distribution $\mathrm{Cat}\left(\hat{\mathbf{p}}^{an}\right)$, from which we sample an index i:

$$i \sim \mathrm{Cat}\left(\hat{\mathbf{p}}^{an}\right) \quad (1 \leq i \leq |\mathcal{A}| + 1). \tag{14}$$

If $i \leq |\mathcal{A}|$, the model adds a new node with the i-th chemical element A_i, or else the building process terminates.

If a new node w is to be added to a transient graph G_t, the node initialization module computes a corresponding feature vector as follows:

$$\begin{aligned}\mathbf{h}_w &= \mathsf{initNode}\left(w, \mathbf{H}_{V(G_t)}\right) \\ &= \mathrm{MLP}_2^i \circ \mathrm{concat}\Big(\mathsf{readout}\left(\mathbf{H}_{V(G_t)}\right), \mathrm{MLP}_1^i\left(\mathbf{h}_w^0\right)\Big).\end{aligned} \tag{15}$$

In the last line, $\mathbf{h}_w^0$ is a raw feature representing the new node's type based on $\mathcal{A}$ (note the absence of an asterisk, unlike the one in eqn 1). We used one linear layer for each of MLP_1^i and MLP_2^i.

The edge addition module $\mathsf{addEdge}$ computes $\hat{\mathbf{p}}^{ae}$ in the same way as eqn 13 but with an MLP of different weights. If the sampled index $i \sim \mathrm{Cat}\left(\hat{\mathbf{p}}^{ae}\right)$ is less than or equal to $|\mathcal{B}|$, the model adds a new edge with the i-th bond type B_i. If $i = |\mathcal{B}| + 1$, the model stops the edge addition.

To describe the node selection, let us suppose a new node w was added to a transient graph G_{t-1} so that $V(G_t) = V(G_{t-1}) \cup \{w\}$ and $E(G_t) = E(G_{t-1})$. If a new edge is determined to be added, the module $\mathsf{selectNode}$ first updates the node features as

$$\mathbf{H}'_{V(G_t)} = \mathsf{propagate}^{(k)}\left(\mathbf{H}_{V(G_t)}, \mathbf{H}_{E(G_t)}, \mathbf{z}\right) \tag{16}$$

and computes the selection probability for each existing node through the following steps:

$$\hat{p}_u^{sn'} = \mathrm{MLP}^{sn} \circ \mathrm{concat}\left(\mathbf{h}'_u, \mathbf{h}_w, \mathbf{z}\right) \quad \forall u \in V(G_t) \setminus \{w\} \tag{17}$$

$$\hat{\mathbf{p}}^{sn} = \mathrm{softmax}\left(\hat{\mathbf{p}}^{sn'}\right). \tag{18}$$

Then from the resulting categorical distribution $\mathrm{Cat}\left(\hat{\mathbf{p}}^{sn}\right)$, the model samples a node and connects it with w (i.e., add the resulting edge to $E(G_t)$).

The edge initialization module $\mathsf{initEdge}$ computes edge feature vectors in the same way as eqn 15. The differences are that different MLPs are used and that $\mathbf{h}_w^0$ is replaced by a raw representation of the chosen bond type.

The complete specification of a molecular graph should assign the extended types in $\mathcal{A}^*$ and $\mathcal{B}^*$ to its elements. Motivated by the strategy of Jin et al. [5], our model assigns only the basic types in $\mathcal{A}$ and $\mathcal{B}$ during graph building and specifies stereoisomerism at the final stage of generation. Given a graph G, the isomer selection module selectIsomer prepares the set of all possible stereoisomers of G enumerated by RDKit. The graphs in the resulting set $\mathcal{I}(G)$ consist of nodes and edges that are fully typed according to $\mathcal{A}^*$ and $\mathcal{B}^*$. For each isomeric graph $I \in \mathcal{I}(G)$, selectIsomer estimates the selection probability through

$$\left(\mathbf{H}_{V(I)}, \mathbf{H}_{E(I)}\right) = \mathsf{embed}\left(I\right) \tag{19}$$

$$\mathbf{H}'_{V(I)} = \mathsf{propagate}^{(k)}\left(\mathbf{H}_{V(I)}, \mathbf{H}_{E(I)}, \mathbf{z}\right) \tag{20}$$

$$\mathbf{h}_I = \frac{1}{|V(I)|} \sum_{v \in V(I)} \mathbf{h}'_v \tag{21}$$

$$\hat{p}_I^{si} = \sigma \circ \mathrm{MLP}^s \circ \mathrm{concat}\left(\mathbf{h}_I, \mathbf{z}\right). \tag{22}$$

Note that multiple $I \in \mathcal{I}(G)$ can be valid for one G. For instance, there can be some G whose stereocenters are only partially labeled by its data source, and in such case isomers with different labels on the same unlabeled stereocenters can all be regarded as valid. When generating a new molecule, however, we want our model to predict one isomer without ambiguity. Therefore, in the generation phase, the model normalizes the probabilities $\hat{p}_I^{si}$ by $\sum_I \hat{p}_I^{si}$ and then samples one plausible isomer from the resulting categorical distribution.

Learning

We used for learning the molecule dataset described in Datasets and experiments in the main text. The dataset consists of a set of scaffold molecules, $\mathcal{S}$, and a collection of each scaffold's whole-molecules, $\mathcal{D}(\mathcal{S}) = \{\mathcal{D}(S) : S \in \mathcal{S}\}$, where $\mathcal{D}(S)$ is a set of whole-molecules of scaffold S. We can arrange $\mathcal{S}$ and $\mathcal{D}(\mathcal{S})$ into an indexed family $((S_i, G_i))_{1 \leq i \leq \sum_{S \in \mathcal{S}} |\mathcal{D}(S)|}$, each of whose elements is the pair of a scaffold and one of its whole-molecules. There can be duplicates of scaffolds or whole-molecules over different pairs, but each pair (S_i, G_i) itself is unique.

The individual loss l_i due to each pair (S_i, G_i) is a weighted sum of three losses: the graph building loss l_i^{build}, the isomer selection loss l_i^{isomer}, and the posterior approximation loss l_i^{KL}. The second of the three is set to be

$$l_i^{\mathrm{isomer}} = -\sum_{I \in \mathcal{I}(G_i)} \left(p_I^{si} \log\left(\hat{p}_I^{si}\right) + \left(1 - p_I^{si}\right) \log\left(1 - \hat{p}_I^{si}\right)\right), \tag{23}$$

where p_I^{si} is the true probability of selecting I. The third of the three reads [4]

$$l_i^{\mathrm{KL}} = -\frac{1}{2} \sum_j \left(1 + \log\left(\sigma_{G_i,j}^2\right) - \mu_{G_i,j}^2 - \sigma_{G_i,j}^2\right), \tag{24}$$

where $\mu_{G_i,j}$ and $\sigma_{G_i,j}$ are the j-th elements of $\boldsymbol{\mu}_{G_i}$ and $\boldsymbol{\sigma}_{G_i}$, respectively (eqn 9 and 10).

To describe the graph building loss, let us express the stepwise transitions from S_i to G_i by a finite sequence $(G_{i,0}, G_{i,1}, \cdots, G_{i,T})$, where $G_{i,0} = S_i$ and $G_{i,T} = G_i$. The transition from $G_{i,t}$ to $G_{i,t+1}$ conforms to the true probability vector $\mathbf{p}_{i,t}$, which has one unity value for the correct building action and zeros for the others. During learning, the model reconstructs each G_i from S_i by estimating a sequence $(\hat{\mathbf{p}}_{i,0}, \hat{\mathbf{p}}_{i,1}, \cdots, \hat{\mathbf{p}}_{i,T-1})$. Then the individual graph building loss can be defined by

$$l_i^{\mathrm{build}} = -\sum_t \mathbf{p}_{i,t} \cdot \log\left(\hat{\mathbf{p}}_{i,t}\right). \tag{25}$$

We minimized $\sum_i \left(l_i^{\mathrm{build}} + l_i^{\mathrm{isomer}} + \beta l_i^{\mathrm{KL}}\right)$ to optimize our model and maximize the log-likelihood objective. We used 0.1 for the weight β. As for k, the number of iterations of propagate, we set $k = 3$ for the initial propagation of whole-molecule graphs and scaffold graphs and $k = 2$ for addNode, addEdge, and selectNode.

Finally, we remark the effect of node and edge orderings. Sequential generation of graphs requires their elements to be ordered. Different orderings amount to different sequences of graph transitions for the same (S_i, G_i). Similarly to Li et al. [6], we trained two models using a fixed ordering for one and

using random orderings for the other. We evaluated the two models in terms of the descriptors used in Results and Discussions in the main text and confirmed that different orderings cause no significant change in performance. Therefore, we used the fixed ordering (assigned by RDKit when reading SMILES data) for all the results.

References

(1) *RDKit: Open-Source Cheminformatics*, `www.rdkit.org`.

(2) V. Nair and G. E. Hinton, Proceedings of the 27th International Conference on Machine Learning, Haifa, Israel, 2010, pp. 807–814.

(3) K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk and Y. Bengio, Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 2014, pp. 1724–1734.

(4) D. P. Kingma and M. Welling, 2nd International Conference on Learning Representations, Conference Track Proceedings, Banff, AB, Canada, 2014.

(5) W. Jin, R. Barzilay and T. Jaakkola, Proceedings of the 35th International Conference on Machine Learning, ed. J. Dy and A. Krause, PMLR, Stockholmsmässan, Stockholm Sweden, 2018, vol. 80, pp. 2323–2332.

(6) Y. Li, O. Vinyals, C. Dyer, R. Pascanu and P. Battaglia, 6th International Conference on Learning Representations, Workshop Track Proceedings, Vancouver, BC, Canada, 2018.