Probabilistic hypergraph grammars for efficient molecular optimization

Egor Kraev
Mosaic Smart Data Ltd
egor.kraev@gmail.com

Mark Harley
Mosaic Smart Data Ltd
mharley.code@gmail.com

Abstract

We present an approach to make molecular optimization more efficient. We infer a hypergraph replacement grammar from the ChEMBL database, count the frequencies of particular rules being used to expand particular nonterminals in other rules, and use these as conditional priors for the policy model. Simulating random molecules from the resulting probabilistic grammar, we show that conditional priors result in a molecular distribution closer to the training set than using equal rule probabilities or unconditional priors. We then treat molecular optimization as a reinforcement learning problem, using a novel modification of the policy gradient algorithm - batch-advantage: using individual rewards minus the batch average reward to weight the log probability loss. The reinforcement learning agent is tasked with building molecules using this grammar, with the goal of maximizing benchmark scores available from the literature. To do so, the agent has policies both to choose the next node in the graph to expand and to select the next grammar rule to apply. The policies are implemented using the Transformer architecture with the partially expanded graph as the input at each step. We show that using the empirical priors as the starting point for a policy eliminates the need for pre-training, and allows us to reach optima faster. We achieve competitive performance on common benchmarks from the literature, such as penalized logP and QED, with only hundreds of training steps on a budget GPU instance. [1]

1 Introduction

A major problem facing the exploration of novel molecules for the purposes of drug design is the vast array of potentially useful compounds – estimated to be in the range of 10^{24} and 10^{60} possible drug-like structures [1, 2]. While it is of course necessary to experimentally determine the usefulness, and safety, of candidate drugs in the laboratory, de novo drug design is an approach to finding candidate molecules through either exhaustive search, or through various generative and machine learning models. This approach takes the form of an optimization procedure over given target scoring functions, giving pre-screened, promising molecules and thereby hopefully reducing drug discovery costs.

Deep learning has now been extensively investigated for encoding and generating molecular graphs [3–12], and remains an area of active research. Typically, the approach taken has been to generate a linear molecular representation, such as the SMILES format [13], with an encoder-decoder network architecture similar to that used in machine translation [10].

This route is, however, not optimal for this problem domain. Unlike written text, a molecule's structure is non-linear – including both cycles and branches. The model is therefore forced not only to learn to optimize molecules on the given benchmark, but also to learn to generate SMILES strings

[1] All code as well as the actual grammar will be made publicly available.

corresponding to chemically valid molecules. This task is non-trivial and robs capacity of the model from the true task at hand.

A recent development which partially remedied the issue was presented by Kusner, et. al. [9], in which the authors deduce a context-free grammar (CFG) for SMILES strings. This grammar guarantees that only valid SMILES strings will be produced, however not all valid SMILES strings are chemically possible molecules and so some model capacity must still be spent on learning the subset of chemically valid SMILES.

Moving away from linear representations, Kajino [14] proposed the use of a grammar defined on a hypergraph representation of molecular structure. The molecular hypergraph grammar (MHG), a special case of an hyperedge replacement grammar (HRG) [15], uses rules which can be pictured as splitting the molecular graph at non-terminal bonds, and replacing them with another subgraph, thereby constructing any desired molecular structure while guaranteeing that only chemically valid molecules are produced. In particular, this approach does not have issues of invalid atom valences or loss of stereochemistry that other approaches suffer.

A different approach was taken by [16], who directly construct a molecular graph step by step. This has the disadvantages of a more complex action space (4 policies as compared to two in our case) as well as the fact that chemical validity must be enforced at simulation time, and the model trained to encourage valid transitions, instead of validity being guaranteed by construction as it is when using a HRG.

Brown, et. al. [17] provide an overview of the techniques used, as well as a set of benchmarks for model evaluation and the scores of several reference implementations.

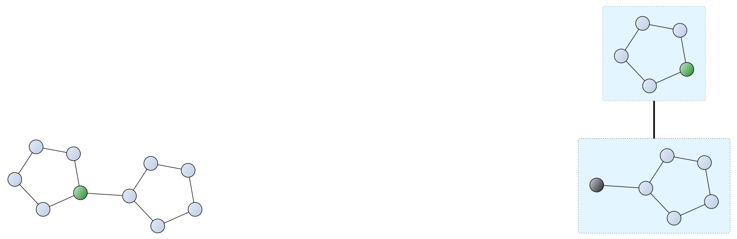
We start with an approach similar to [14], but rather than reducing all hypergraph cliques, we terminate early allowing cycles of length five and greater to remain. This permits an equally expressive grammar but allows non-trivial cycle structure to be expressed with far fewer rules, resulting in a grammar we call the hypergraph-optimized grammar (HOG). This allows the model to produce novelties without straying too far from reasonable drug-like molecules but, comes at the cost of introducing more rules. We then optimize the model's usage of the HOG by injecting conditional priors for rule selection. After parsing, we count all occurrences of rules conditional on the parent rule and the nonterminal therein that's being expanded – counting all occurrences of $(\text{parent}, \text{location}, \text{child})$ tuples – and use these frequencies as priors to the rule selection with the effect of grounding the molecular structure in the region of those seen in the training set, but allowing the model to explore further substructure. We infer this grammar from the ChEMBL training set of 1,273,104 SMILES strings provided by [12]

The second innovation we present, first adapted to this problem in [18], is the use of the Transformer architecture [19] in place of the more typically applied RNNs [20, 21] or, recently, a graph convolutional network [16]. Unlike an RNN or CGN, the Transformer's information distance between any two inputs is always one, giving the network full information about the sequence so far when selecting the next graph node and rule to expand. This will clearly have an impact on the memory performance of the algorithm, which grows as the square of the sequence length, and so it is beneficial that the HOG optimized MHG provides a concise representation of a given molecule.

Furthermore, rather than use the encoder-decoder architecture used in some previous work [6–10], we treat this problem with a reinforcement learning approach (see e.g. [22] for another RL based approach) with policies selecting directly the next grammar rule and onto which hypergraph node it should be applied. This means that we do not have to learn a latent space representation of the molecules. We train these policies using a batch-advantage modification of the policy gradient algorithm.

2 Hypergraph grammar

In order to address the issue highlighted in Sec. 1 relating to SMILES string grammars, we choose to use a *molecular hypergraph grammar* (MHG) as derived by Kajino in [14]. This works by first representing the molecular graph atoms as hyperedges and bonds as hypernodes of a molecular hypergraph. This will produce hypergraphs which are (i) 2-regular and (ii) have constrained cardinality of its hyperedges. (i) ensures that the hypergraph can be decoded back to a molecular graph and (ii) preserves the valence of each atom.



(a) Molecular hypergraph before cutting. (b) Parent and child sections of parse tree after cutting.

Figure 1: Molecular hypergraph parse tree creation step. Black node indicates the parent node that is merged into the green parent node when the child rule is applied.

The MHG is defined over these hypergraphs as a hyperedge replacement grammar (HRG) [15], a grammar generating a hypergraph by replacement of hyperedges with other hypergraphs. This approach has a number of desirable properties, such as preserving the number of hypernodes belonging to each hyperedge, thereby preserving an atom's valence – satisfying (i) above. Furthermore, stereochemisty can be encoded directly into the hyperedge replacement rules. MHG is thus guaranteed to produce only chemically valid molecules, allowing our model below to focus on optimizing the target benchmarks, without wasting network capacity on learning to generate valid outputs.

2.1 Parsing algorithm

In order to build the grammar which will be used by the RL agent outlined in 3, we construct a parse tree for each molecule in a training set. By doing so for each molecule, we can identify the set of unique hyperedge replacement rules defining the grammar. Given an input molecular graph, the algorithm to deduce the MHG rules is as follows.

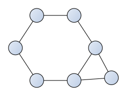
1. Turn all atoms into child nodes containing the specific atom with a parent node containing a nonterminal with identical connectivity; this means all further graph decomposition only deals with the connectivity of the graph, without regard to the actual atoms.
2. Find a node, n, at which the graph can be subdivided by cutting a single connected hyperedge
3. Split the graph at n producing a now reduced parent graph and a new child graph containing a new non-terminal node on the cut hyperedge, which we call the *parent node*
4. Iterate steps above until the parent graph can no longer be subdivided in this manner
5. Apply all of the above recursively to the new child graphs

After this, we have a parse tree where the original graph can be reconstructed by replacing the correct node in the parent graphs with children at their corresponding parent nodes. A tree constructed in this manner will contain only graphs containing a single hyperedge or graphs composed of cycles. An example of a step in the above algorithm is illustrated in Fig. 1.

2.1.1 Extraction of Hypergraph Cliques

Though the above procedure will produce a MHG able to represent any molecule, it is not the most efficient representation given the combinatorial complexity in the construction of molecular graph cycles leading to an unnecessarily large set of rules. We now continue to reduce the cycle containing graphs by indentifying any cliques of the remaining graph of length greater than or equal to three, but which do not contain the parent node. This is achieved by replacing the entire clique with a new non-terminal graph node in the parent, see Fig. 2. The child consists of the clique and all edges exiting the clique connected to the new non-terminal parent node.

After removing all such cliques, we continue to remove 2-cliques from any remaining cycles of length greater than five. This reduces the complexity of remaining cycles, and so the number of resultant rules, but discourages the production of triangles and boxes which are undesirable in output molecules from the model.


(a) Molecular hypergraph before removal of the 3-clique.

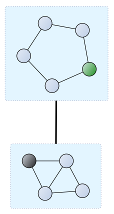

(b) Parent and child sections of parse tree after removing the 3-clique.

Figure 2: Molecular hypergraph clique extraction step. Black node indicates the parent node that is merged into the green parent node when the child rule is applied.

2.2 Using the grammar to create new molecules

After following the procedures outlined above, we can collect all unique graphs. This set of unique graphs are the rules of the HOG. In the sense of a CFG, a rule maps a non-terminal node to a hypergraph containing further terminal and non-terminal nodes. To generate a new molecule we first select a starting rule, which by construction will contain one or more non-terminal nodes. Next, for each non-terminal node in this first graph, we select a rule whose parent node will expand the non-terminal in the parent rule. We recursively apply the same procedure now over the selected child rules until all non-terminals are replaced with terminals. This is guaranteed to produce a valid molecular hypergraph.

2.2.1 Conditional and unconditional rule frequencies

To assist the model in selecting these rules, we collect the unconditional and conditional rule frequencies as they appear in the inference data set after having inferred the final set of grammar rules. Unconditional frequencies are simply the total count of appearances of each unique rule in the inference set parse trees, whereas conditional frequencies count the occurrence of each rule conditional on the given parent and the nonterminal therein, that it will expand in the parse tree. These are used as priors for rule selection, by adding their logarithm to the logits the policy model outputs, before taking the softmax to calculate rule selection probabilities. For (parent, location, child) tuples that were never observed in the training set, we set their log prior to -3, and for those that are impossible due to nonterminal mismatch, we set their log prior to -1000; thus the priors also double as masking to ensure validity.

2.2.2 Ensuring expansions terminate

The final challenge we address is making sure the rule expansion terminates before the maximum allowed number of expansion rules has been reached by the agent. To do this, we define the concept of *terminal distance* of a hypergraph node, defined as the length of the shortest sequence of rules needed to expend said node into a sub-hypergraph consisting only of terminal nodes.

We calculate this distance for each hypergraph in our set of grammar rules by means of the following dynamic programming algorithm:

1. Define a set R of all rule hypergraphs observed so far, and seed it with the root token `molecule`.
2. Initialize all parent node terminal distances to ∞.
3. Iterate over all parent nodes n_p of rules r in R, defining R_{n_p} as the set of rules with equivalent parent node n_p.
   (a) For each n_p, the terminal distance is defined as one plus the minimum of terminal distances for all possible child rules of this graph. Defining C(r) as the set of child nodes of the rule r,

$$\mathrm{TD}_n(n_p) = 1 + \min_{r \in R_{n_p}} \left(\sum_{c \in C(r)} \mathrm{TD}_r(c) \right) , \tag{1}$$

where $\mathrm{TD}_n, \mathrm{TD}_r$ are the node and rule terminal distances respectively. If the child rule, c, is terminal it is assigned terminal distance $\mathrm{TD}_r(c) = 0$.

We repeat step 3. until convergence, that is until the computed terminal distances do not change between iterations. This is most efficiently implemented as a dynamic programming problem since there are many overlapping sub-problems.

Finally, we define the terminal distance of a rule hypergraph as the sum of terminal distances of the child nodes,

$$\mathrm{TD}_r(r) = \sum_{c \in C(r)} \mathrm{TD}_r(c)\,. \tag{2}$$

We use the terminal distance concept to make sure the rule expansion terminates before the maximum number of steps, in the following manner: at each step, we consider the number s of steps left and the terminal distance, td, of the sequence generated so far.

At each step, we make sure $\mathrm{td} \leq s$, using induction. First, we choose the maximum rule sequence length to be larger than the terminal distance of the root graph. Second, at each rule selection step we consider all child rules, r, who can expand the next nonterminal in the graph, and only allow those where $\Delta\mathrm{TD}_r(r) + \mathrm{td} \leq s - 1$. This must be a nonempty set because $\Delta\mathrm{TD}_r(r') = -1$ for at least one applicable rule r'. Thus, by the time we run out of steps, that is, $s = 0$, we know $\mathrm{td} = 0$, that is our token sequence consists only of nonterminals.

3 Model choice

3.1 Graph embedding

In a model that generates a discrete object via a series of decisions, we need to choose how to represent the intermediate state to feed into making the next choice. In models that produce sequences, such as SMILES strings, the natural choice is a sequence of one-hot embedded string characters. In models that construct the molecular graph directly, we have more options: we could take as our representation the sequence of the grammar rules chosen so far, like in [9, 18], or a representation of the intermediate graph state, as in [16].

We choose the latter, via a simple encoding of three concatenated parts, with a sequence of length equaling the number of nodes in the graph. That embedding is composed of the concatenation of the connectivity matrix of the graph (with values 0, 1, 2, etc. denoting no connection, single bond, double bond, etc.), the identity matrix (so that the vector knows which node it refers to, as the ordering of the sequence itself is arbitrary), and a one-hot encoding of any data on each node, such as atomic number and chirality. This is then linearly transformed to the dimension of the downstream model.

3.2 Network architecture

We choose the Transformer architecture [19] rather than the more often used RNNs, because the nodes in the graph are not naturally ordered, and as we don't add the sinusoidal position encoding from the original Transformer model to the inputs, the outputs are invariant to the ordering of the input sequence. The Transformer was used for graph inputs by [23], but never to our knowledge in the context of molecular optimization. We use model dimension 512, 6 heads, head size 64, and 5 layers.

3.3 Policy network and discriminator

We implement both a policy model and for some runs a discriminator[2] model using the above architecture. The policy model has two heads, one that supplies the logits for choosing the next nonterminal node to expand, and one that supplies the logits for the next rule to use on that node.

The discriminator network has a single head that tries to calculate the probability of whether a given molecule comes from the training dataset or has been created by the model.

[2]see [11, 24, 25] for applications of discrimator based method, though with a sequential SMILES molecular representation

3.4 Training

Most of the literature on molecular optimization either trains a VAE and then optimizes over the latent space, or uses a reinforcement learning approach; most of the reinforcement learning approaches don't exchange any information between the results of a simulation batch, with the exception of [26, 27] and [18], who simulate the whole model and then take the k results with the best reward, and rewards the log-likelihood of those decision paths; which can be regarded as a basic kind of Monte Carlo RL. That is motivated by the fact that in contrast to many other RL use cases, we don't know the reward until the end of the episode, and so it's worth trying to reward the more successful paths as a whole after the fact.

3.4.1 Non-originality penalty

To avoid the optimization getting stuck in a local minimum, we penalize repeated occurrences of a molecule. Denoting the batch size used for generator training as b, we keep buffers of b, $10 \cdot b$, and $100 \cdot b$ last unique molecules, and decrease the rewards of new molecules if these are found in the training dataset or one of these buffers.

3.4.2 Batch Advantage

We extend the approach of [26] by introducing the concept of batch advantage, defined as the reward for a particular molecule in the batch minus the average reward for the batch. This can be regarded as a special case of the advantage concept being applied to the molecular optimization, regarding the whole sequence of decisions going into a particular molecule as one 'action'. The batch reward average can then be regarded as an estimate of the state value of the initial state, and the rewards for the batch members as their exact action values. To keep the learning rate uniform, we normalize these advantage values so that the sum of their absolute values equals 1, and use them as weights for the log likelihood of the respective molecule.

Note that batch advantage is especially effective in encouraging exploration when used together with non-originality penalty. When these two are combined, if a batch returns many molecules with very similar rewards (before applying the non-originality penalty), some of them new, and some repeated, then after applying the penalty and batch advantage, the repeated molecules' log likelihood will actually have a negative weight, so we will discourage the model from reproducing these and encourage it to learn new ones.

3.4.3 Record rewards

As a final way to encourage exploration, we keep a buffer of the 10 best rewards observed in the current simulation. In the goal-optimization benchmarks, if a new reward is bigger than the smallest of these, we add the respective log likelihood to the objective function for that iteration, to especially encourage such molecules

3.4.4 Training step

A training step for the generator consists of simply computing the objective function as above and doing one step of the Adam optimizer with a learning rate of 4e-5.

For the discriminator, the key thing was to balance a quick reaction to innovations from the generator with long-term memory. To achieve that, we use the above mentioned buffers of size b, $10 \cdot b$, and $100 \cdot b$ last unique molecules, let's call them buf_1, buf_{10}, and buf_{100}. Then each batch of b_g molecules fed to the generator consists of $0.5b_g$ molecules from the training set, $0.25b_g$ molecules from buf_1, $0.125b_g$ molecules from buf_{10}, and $0.125b_g$ molecules from buf_{100}.

4 Results

4.1 Comparison of distributions

To see whether the conditional priors are a better approximation of the training set distribution, we generate sets of 10000 random molecules from our grammar, in three versions: using equal rule probabilities; using unconditional probabilities (rule frequencies); and using conditional priors as

Table 1: GuacaMol 'Trivial' Benchmark Results

Benchmark	GuacaMol Best	PHG Result	Top Mol Score	Steps to max
logP $= -1$	1	0.999969	0.9999997	40
logP $= 8.0$	1	0.999873	0.99999997	130
TPSA (target 150)	1	0.999967	0.9999998	40
CNS MPO	1	1	1	1
QED	0.948	0.94589	0.9476	1200
C7H8N202	1	0.9896	1	130
P. MPO	0.998	0.99741	0.999184	780

Table 2: GuacaMol Distribution Benchmark Results. Compare with table 1. from [17]

Benchmark	No priors	Priors	Conditional Priors	With Discriminator
Validity	1	1	1	1
Uniqueness	0.96	0.71	0.85	0.9987
Novelty	0.96	0.7	0.81	0.9946
KL Divergence	0.2	0.59	0.71	0.79
FCD	2×10^{-5}	0.01	0.0531	0.118

discussed in Section 2.2.1. We further train a generator/discriminator pair, using the probability output by the discriminator as a reward for the generator, with repetition penalty and batch advantage-weighted log likelihood as generator objective (no rewards for record values in this case as we're interested in distribution as a whole, not the outliers).

The results are in Table 2. As we see, both the KL divergence and the Frechet ChemNet distance improve as we move from random, to unconditional, to conditional priors. These are further improved by training a generator/discriminator pair, which also greatly improves uniqueness. Finally, as expected the Validity benchmark is at 1.0 throughout, as all molecules are valid by construction.

4.2 Goal-optimization benchmarks

We first look at the goal-optimization benchmarks called 'trivial' by GuacaMol. Even though these are indeed not sufficient to measure a model's superiority for de novo drug design, they still provide an important baseline for any approach to satisfy. Here we don't use a discriminator, do use batch advantage and repetition penalty, and also use the reward for record values as described in earlier sections.

As Table 1 shows, we achieve competitive performance on all of these benchmarks, and notably succeed in doing so after merely hundreds of batches, each batch having size 15. In particular, we achieved the same top QED value as [16].

For continuity with prior literature, we also ran the penalized logP optimization as optimized by [9, 16], and many others. In this case we did use a discriminator as part of the penalty, to reward similarity to training set molecules. While our approach does not allow to directly specify maximum number of atoms, we achieved that by adding a term to the objective function penalizing number of atoms greater than a set limit. When we limit the number of atoms to 38, as in [16], we achieve a reward of 8.27 within 960 training steps, as always without pre-training.

Experimenting with the number of atoms limit, we found that it was the determining factor in the maximum score achieved, thus we conjecture that the increase in the penalized score that [16] achieved (from between 5 and 6 to just under 8) was to a large extent due to a larger maximum number of atoms compared to earlier models.

5 Discussion

When training a model, it's best to start with the best inductive bias we can, so the model's training can focus on the metrics of interest, rather than for example chemical validity. To achieve that in the case of molecular optimization, we first constructed a hypergraph grammar for molecular optimization, by inferring it from a training dataset similarly to [14]. That approach allows us to guarantee that all molecules generated by the model are chemically valid by construction, without the need for runtime checks such as done by [16]

We then compared evenly distributed rule probability priors to those only using total rule frequency, and to conditional priors also taking into account the parent rule and the nonterminal therein that's being expanded. We showed that using these conditional priors to simulate random molecules gives us the best approximation to the training set distribution, and thus the best starting point for optimization.

Further, we introduced batch advantage, a simple and natural modification to the policy gradient approach, and showed that it encouraged exploration and sped up convergence, especially when used together with a penalty for repetition.

Having good priors makes it easier for a model to do objective optimization. That allowed us to achieve competitive results without pre-training on all the 'trivial' goal-oriented GuacaMol benchmarks, in a very small number of steps, as well as on the penalized logP objective common in the literature; our approach showed promise, but was less successful, on the more advanced GuacaMol benchmarks - we conjecture that a more sophisticated reinforcement learning algorithm, such as PPO, is needed there.

We are also, to our knowledge, the first to apply the Transformer architecture to graph embeddings for the purpose of molecular optimization. Comparison with the graph convolutional networks for that purpose will be the subject of further work.

References

[1] W. P. Walters, “Virtual chemical libraries,” *Journal of Medicinal Chemistry*, vol. 62, no. 3, pp. 1116–1124, 2019. PMID: 30148631.

[2] L. Ruddigkeit, R. van Deursen, L. C. Blum, and J.-L. Reymond, “Enumeration of 166 billion organic small molecules in the chemical universe database gdb-17,” *Journal of Chemical Information and Modeling*, vol. 52, no. 11, pp. 2864–2875, 2012. PMID: 23088335.

[3] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, “Convolutional networks on graphs for learning molecular fingerprints,” in *Advances in Neural Information Processing Systems 28* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 2224–2232, Curran Associates, Inc., 2015.

[4] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, “Molecular graph convolutions: moving beyond fingerprints,” *Journal of Computer-Aided Molecular Design*, vol. 30, pp. 595–608, Aug. 2016.

[5] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, “Neural message passing for quantum chemistry,” *CoRR*, vol. abs/1704.01212, 2017.

[6] H. Dai, Y. Tian, B. Dai, S. Skiena, and L. Song, “Syntax-directed variational autoencoder for structured data,” *CoRR*, vol. abs/1802.08786, 2018.

[7] W. Jin, R. Barzilay, and T. S. Jaakkola, “Junction tree variational autoencoder for molecular graph generation,” *CoRR*, vol. abs/1802.04364, 2018.

[8] M. Simonovsky and N. Komodakis, “Graphvae: Towards generation of small graphs using variational autoencoders,” *CoRR*, vol. abs/1802.03480, 2018.

[9] M. J. Kusner, B. Paige, and J. M. Hernández-Lobato, “Grammar Variational Autoencoder,” *ArXiv e-prints*, Mar. 2017.

[10] R. Gómez-Bombarelli, D. K. Duvenaud, J. M. Hernández-Lobato, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, “Automatic chemical design using a data-driven continuous representation of molecules,” *CoRR*, vol. abs/1610.02415, 2016.

[11] G. Lima Guimaraes, B. Sanchez-Lengeling, C. Outeiral, P. L. Cunha Farias, and A. Aspuru-Guzik, “Objective-Reinforced Generative Adversarial Networks (ORGAN) for Sequence Generation Models,” *arXiv e-prints*, p. arXiv:1705.10843, May 2017.

[12] P. Pogány, N. Arad, S. Genway, and S. D. Pickett, “De novo molecule design by translating from reduced graphs to smiles,” *Journal of Chemical Information and Modeling*, vol. 59, no. 3, pp. 1136–1146, 2019.

[13] D. Weininger, “Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules,” *J. Chem. Inf. Comput. Sci.*, vol. 28, no. 1, pp. 31–36, 1988.

[14] H. Kajino, “Molecular hypergraph grammar with its application to molecular optimization,” *CoRR*, vol. abs/1809.02745, 2018.

[15] F. Drewes, H.-J. Kreowski, and A. Habel, “Hyperedge replacement graph grammars,” in *Handbook of Graph Grammars and Computing by Graph Transformation* (G. Rozenberg, ed.), pp. 95–162, River Edge, NJ, USA: World Scientific Publishing Co., Inc., 1997.

[16] J. You, B. Liu, R. Ying, V. Pande, and J. Leskovec, “Graph convolutional policy network for goal-directed molecular graph generation,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS’18, (USA), pp. 6412–6422, Curran Associates Inc., 2018.

[17] N. Brown, M. Fiscato, M. H. Segler, and A. C. Vaucher, “Guacamol: Benchmarking models for de novo molecular design,” *Journal of Chemical Information and Modeling*, vol. 59, no. 3, pp. 1096–1108, 2019.

[18] E. Kraev, “Grammars and reinforcement learning for molecule optimization,” *CoRR*, vol. abs/1811.11222, 2018.

[19] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *CoRR*, vol. abs/1706.03762, 2017.

[20] X. Yang, J. Zhang, K. Yoshizoe, K. Terayama, and K. Tsuda, “Chemts: an efficient python library for de novo molecular generation,” *Science and Technology of Advanced Materials*, vol. 18, no. 1, pp. 972–976, 2017. PMID: 29435094.

[21] M. Olivecrona, T. Blaschke, O. Engkvist, and H. Chen, “Molecular de novo design through deep reinforcement learning,” *Journal of Cheminformatics*, vol. 9, 04 2017.

[22] M. Popova, O. Isayev, and A. Tropsha, “Deep reinforcement learning for de novo drug design,” *Science Advances*, vol. 4, no. 7, 2018.

[23] W. Kool, H. van Hoof, and M. Welling, “Attention, Learn to Solve Routing Problems!,” *arXiv e-prints*, p. arXiv:1803.08475, Mar 2018.

[24] Sanchez-Lengeling, Benjamin, Outeiral, Carlos, Guimaraes, G. L., Aspuru-Guzik, and Alan, “Optimizing distributions over molecular space. an objective-reinforced generative adversarial network for inverse-design chemistry (organic),” Aug 2017.

[25] E. Putin, A. Asadulaev, Y. Ivanenkov, V. Aladinskiy, B. Sanchez-Lengeling, A. Aspuru-Guzik, and A. Zhavoronkov, “Reinforced adversarial neural computer for de novo molecular design,” *Journal of Chemical Information and Modeling*, vol. 58, no. 6, pp. 1194–1204, 2018. PMID: 29762023.

[26] M. H. S. Segler, T. Kogej, C. Tyrchan, and M. P. Waller, “Generating focussed molecule libraries for drug discovery with recurrent neural networks,” *CoRR*, vol. abs/1701.01329, 2017.

[27] D. Neil, M. Segler, L. Guasch, M. Ahmed, D. Plumbley, M. Sellwood, and N. Brown, “Exploring deep recurrent models with reinforcement learning for molecule design,” 2018.