A. Detailed discussion of related work

Traditional graph generative models There has been a lot of work (Erdös & Rényi, 1959; Barabási & Albert, 1999; Albert & lászló Barabási, 2002; Chakrabarti et al., 2004; Robins et al., 2007; Leskovec et al., 2010; Airoldi et al., 2009) on generating graphs with a set of specific properties like degree distribution, diameter, and eigenvalues. While some of them (Erdös & Rényi, 1959; Barabási & Albert, 1999; Albert & lászló Barabási, 2002; Chakrabarti et al., 2004) focused on the statistical mechanics of random and complex networks, (Robins et al., 2007) proposed exponential random graph models, known as $p*$ model, in estimating network models comes from the area of social sciences, statistics and social network analysis. However the $p*$ model usually focuses on local structural features of networks. (Airoldi et al., 2009; Leskovec et al., 2010), on the other hand, model the structure of the network as a whole by combining a global model of dense patches of connectivity with a local model.

Most of these classical models are hand-engineered to model a particular family of graphs, and thus do not have the capacity to directly learn the generative model from observed data.

Deep graph generative models Recent work on deep neural network based graph generative models are typically much more expressive than classic random graph models, at a higher computation cost.

Similar to visual data, there are VAE (Kingma & Welling, 2013) and GAN (Goodfellow et al., 2014) based graph generative models, like GraphVAEs (Kipf & Welling, 2016; Simonovsky & Komodakis, 2018) and NetGAN (Bojchevski et al., 2018). These models typically generates large parts (or all) of the adjacency matrix independently, making them unable to model complex graph structures.

Auto-regressive models, on the other hand, generate a graph sequentially part by part, gaining expressivity but are typically more expensive as large graphs require large numbers of generation steps. Scalability is a constant important topic in this line of work, starting from (Li et al., 2018). More recent work GraphRNN (You et al., 2018) and GRAN (Liao et al., 2019) reported improved scalability and success generating graphs of up to 5,000 nodes. Our novel BiGG model advances this line of work by significantly improving both on model quality and scalability.

Besides models for general graphs, a lot of work also exploit domain knowledge for better performance in specific domains. Examples of this include (Kusner et al., 2017; Dai et al., 2018; Jin et al., 2018; Liu et al., 2018) for modeling molecule graphs, and (You et al., 2019) for SAT instances.

B. More experiment details

Manual Batching Though theoretically Section 3.1 enables the parallelism, the commonly used packages like PyTorch and TensorFlow has limited support for auto-batching operators. Also optimizing over a computation graph with $O(m)$ operators directly would be infeasible. Inspired by previous works (Looks et al., 2017; Neubig et al., 2017; Bradbury & Fu, 2018), we enable the parallelism by performing gathering and dispatching for tree or LSTM cells. This requires a single call of `gemm` on CUDA, instead of multiple `gemv` calls for the cell operations. The indices used for gathering and dispatching are computed in a customized c++ op for the sake of speed.

Parameter sharing As the maximum depth in BiGG is $O(\log n)$, it is feasible to have different parameters per different layers for tree or LSTM cells. However experimentally it didn't affect the performance significantly. With this change, not all parameters are updated at each step, as the trees have different heights and some are updated more often than others, which could potentially make the learning harder. Also it would make the model $O(\log n)$ times larger, and limits the potential of generating graphs larger than seen during training. So we still share the parameters of cells like other autoregressive models.

B.1. BiGG setup

Hyperparameters: For the proposed BiGG, we use embedding size $d = 256$ together with position encoding for state representation. Learning rate is initialized to $1e^{-3}$ and decays to $1e^{-5}$ when training loss gets plateau.

All the experiments for both baselines and BiGG are carried out on a single V100 GPU. For the sublinear memory technique mentioned in Section 3.3, we choose k (*i.e.*, the number of blocks) as small as possible such that the training would be able to fit in a single V100 GPU. This is to make the training feasible while minimizing the synchronization overhead. Empirically the block size used in sublinear memory technique is set to 6,000 for sparse graphs.

Method	VIG		VIG			LCG
	Clustering	Modularity	Variable α_v	Clause α_v	Modularity	Modularity
Training	0.50 ± 0.07	0.58 ± 0.09	3.57 ± 1.08	4.53 ± 1.09	0.74 ± 0.06	0.67 ± 0.05
CA	0.33 ± 0.08 (34%)	0.48 ± 0.10 (17%)	6.30 ± 1.53 (76%)	N/A	0.65 ± 0.08 (12%)	0.53 ± 0.05 (16%)
PS(T=0)	0.82 ± 0.04 (64%)	0.72 ± 0.13 (24%)	3.25 ± 0.89 (9%)	4.70 ± 1.59 (4%)	0.86 ± 0.05 (16%)	**0.64 ± 0.05 (2%)**
PS(T=1.5)	0.30 ± 0.10 (40%)	0.14 ± 0.03 (76%)	4.19 ± 1.10 (17%)	6.86 ± 1.65 (51%)	0.40 ± 0.05 (46%)	0.41 ± 0.05 (35%)
G2SAT	0.41 ± 0.09 (18%)	0.54 ± 0.11 (7%)	**3.57 ± 1.08 (0%)**	4.79 ± 2.80 (6%)	0.68 ± 0.07 (8%)	0.67 ± 0.03 (6%)
BiGG	**0.50 ± 0.07 (0%)**	0.58 ± 0.09 (0%)	**3.57 ± 1.08 (0%)**	**4.53 ± 1.09 (0%)**	0.73 ± 0.06 (1%)	0.61 ± 0.09 (9%)
BiGG-0.1	**0.50 ± 0.07 (0%)**	**0.58 ± 0.09 (0%)**	**3.57 ± 1.08 (0%)**	**4.53 ± 1.09 (0%)**	**0.74 ± 0.06 (0%)**	0.65 ± 0.07 (7%)

Table B.1. Training and generated graph statistics with 10 SAT formulas used in You et al. (2019). Baselines results are directly copied from You et al. (2019).

Method	VIG		VIG			LCG
	Clustering	Modularity	Variable α_v	Clause α_v	Modularity	Modularity
Test-8	0.50 ± 0.05	0.68 ± 0.19	4.66 ± 1.92	6.33 ± 3.23	0.84 ± 0.02	0.75 ± 0.05
G2SAT	0.22 ± 0.10 (56%)	**0.74 ± 0.11 (9%)**	**4.66 ± 1.92 (0%)**	**6.60 ± 4.15 (5%)**	**0.80 ± 0.09 (5%)**	**0.68 ± 0.04 (9%)**
BiGG	**0.37 ± 0.19 (26%)**	0.28 ± 0.12 (58%)	**4.66 ± 1.92 (0%)**	2.74 ± 0.37 (57%)	0.44 ± 0.07 (48%)	0.48 ± 0.09 (36%)

Table B.2. Test and generated graph statistics with 8 SAT formulas from https://github.com/JiaxuanYou/G2SAT. G2SAT and BiGG are trained on 24 sat formulas as in Table 2

B.2. Baselines setup

We include all the baseline numbers from their original papers when applicable, as the most experimental protocols are the same. We run the following baselines when the corresponding numbers are not reported in the literature:

- Erdős–Rényi: for the results in Table 3, we estimate the graph edge density using the training grid graphs, and generate new Erdős–Rényi random graphs with this density and compare with the held-out test graphs;
- GraphRNN-S (You et al., 2018): the result on Lobster random graphs in Table 1 is obtained by training the GraphRNN using official code [1] for 3000 epochs;
- GRAN (Liao et al., 2019): the results of GRAN in Table 3 are obtained by training GRAN with grid graph configuration for 8,000 epochs or 3 days, whichever limit the model reaches first.
- G2SAT (You et al., 2019): For results in Table 2 and Table B.2, we run the code [2] for 1000 epochs. For the test graph comparison in Table B.2, we use the 8 test graphs as 'template-graphs' for G2SAT to generate graphs.

B.3. More experimental results on SAT graphs

In main paper we reported the results on 24 training instances with the evaluation metric used in You et al. (2019). Here we include the results on the subset which is used in the original G2SAT paper (You et al., 2019). This subset contains 10 instances, with 82 to 1122 variables and 327 to 4555 clauses. This result in graphs with 6,799 nodes at most. Table B.1 summarizes the results, together with other baseline results that are copied from You et al. (2019). We can see our model can almost perfectly recover the training statistics if the generative process is biased more towards high probability region (*i.e.*, BiGG-0.1 which has 90% chance to use greedy decoding at each step). This again demonstrate that our model is flexible enough to capture complicated distributions of graph structures. On the other hand, as Table B.2 shows, the G2SAT beats BiGG on 4 out of 6 metrics. Since G2SAT is a specialized generative model for bipartite graphs, it enjoys more of inductive bias towards this specific problem, and thus would work better in the low-data and extrapolation scenario. Our general purposed model thus would be expected to require more training data due to its high capacity in model space.

[1] https://github.com/JiaxuanYou/graph-generation
[2] https://github.com/JiaxuanYou/G2SAT

References

Airoldi, E. M., Blei, D. M., Fienberg, S. E., and Xing, E. P. Mixed membership stochastic blockmodels. In *Advances in Neural Information Processing Systems 21*, pp. 33–40, 2009.

Albert, R. and lászló Barabási, A. Statistical mechanics of complex networks. *Rev. Mod. Phys*, 2002.

Barabási, A.-L. and Albert, R. Emergence of scaling in random networks. *science*, 286(5439):509–512, 1999.

Bojchevski, A., Shchur, O., Zügner, D., and Günnemann, S. Netgan: Generating graphs via random walks. *arXiv preprint arXiv:1803.00816*, 2018.

Bradbury, J. and Fu, C. Automatic batching as a compiler pass in pytorch. In *Workshop on Systems for ML*, 2018.

Chakrabarti, D., Zhan, Y., and Faloutsos, C. R-mat: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pp. 442–446. SIAM, 2004.

Dai, H., Tian, Y., Dai, B., Skiena, S., and Song, L. Syntax-directed variational autoencoder for structured data. *arXiv preprint arXiv:1802.08786*, 2018.

Erdös, P. and Rényi, A. On random graphs i. *Publicationes Mathematicae Debrecen*, 6:290–297, 1959.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

Jin, W., Barzilay, R., and Jaakkola, T. Junction tree variational autoencoder for molecular graph generation. *arXiv preprint arXiv:1802.04364*, 2018.

Kingma, D. P. and Welling, M. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.

Kipf, T. N. and Welling, M. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016.

Kusner, M. J., Paige, B., and Hernández-Lobato, J. M. Grammar variational autoencoder. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pp. 1945–1954. JMLR. org, 2017.

Leskovec, J., Chakrabarti, D., Kleinberg, J., Faloutsos, C., and Ghahramani, Z. Kronecker graphs: An approach to modeling networks. *The Journal of Machine Learning Research*, 11:985–1042, 2010.

Li, Y., Vinyals, O., Dyer, C., Pascanu, R., and Battaglia, P. Learning deep generative models of graphs. *arXiv preprint arXiv:1803.03324*, 2018.

Liao, R., Li, Y., Song, Y., Wang, S., Hamilton, W., Duvenaud, D. K., Urtasun, R., and Zemel, R. Efficient graph generation with graph recurrent attention networks. In *Advances in Neural Information Processing Systems*, pp. 4257–4267, 2019.

Liu, Q., Allamanis, M., Brockschmidt, M., and Gaunt, A. Constrained graph variational autoencoders for molecule design. In *Advances in Neural Information Processing Systems*, pp. 7795–7804, 2018.

Looks, M., Herreshoff, M., Hutchins, D., and Norvig, P. Deep learning with dynamic computation graphs. *arXiv preprint arXiv:1702.02181*, 2017.

Neubig, G., Goldberg, Y., and Dyer, C. On-the-fly operation batching in dynamic computation graphs. In *Advances in Neural Information Processing Systems*, pp. 3971–3981, 2017.

Robins, G., Pattison, P., Kalish, Y., and Lusher, D. An introduction to exponential random graph (p*) models for social networks. *Social Networks*, 29(2):173–191, 2007.

Simonovsky, M. and Komodakis, N. Graphvae: Towards generation of small graphs using variational autoencoders. In *International Conference on Artificial Neural Networks*, pp. 412–422. Springer, 2018.

You, J., Ying, R., Ren, X., Hamilton, W. L., and Leskovec, J. Graphrnn: Generating realistic graphs with deep auto-regressive models. *arXiv preprint arXiv:1802.08773*, 2018.

You, J., Wu, H., Barrett, C., Ramanujan, R., and Leskovec, J. G2sat: Learning to generate sat formulas. In *Advances in Neural Information Processing Systems*, pp. 10552–10563, 2019.