DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization

Authors: Zhiqing Sun, Yiming Yang (presented by J. Le Roux)

22/01/24

Outline

Intro

Definition

Discrete or Continuous Distributions

Predicting Assignments

Results

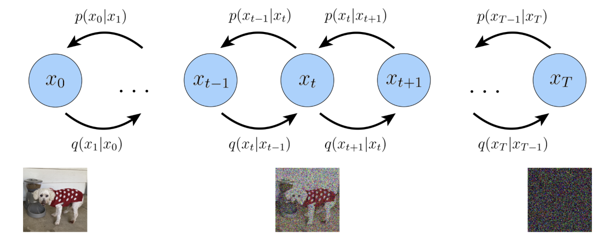
Conclusion

Intro

A mecha robot playing the guitar in a forest, low quality, 3d, photorealistic

Diffusion Models are known to be good at generating images from texts. How can they be applied to CO?

Diffusion Models (2)



- learn how to generate as denoising via distribution p (*backward*)
- from noisy examples generated by a diffusion distribution q (*forward*)

3 types of ML-based CO solvers

Autoregressive Construction Heuristics Solvers

- each time-step a new variable assignment is added to a partial solution.
- inspired by RNN, LLM...
- however high time and space complexity, sequential generation, $O(n)^2$ complexity if self-attention

Non-autoregressive (Heatmaps) Construction Heuristics Solvers

- assume conditional independence among variables, all variables assigned in parallel
- however assumption limit to overly simple distributions
- hybrid approach with active search, MCTS $\Rightarrow$ slow

Improvement Heuristics Solvers

- use MDP to iteratively refines an existing feasible solution with NN-guided operations (2-opt, node swap)
- however difficult to scale up (slow), difficult to learn (sparse rewards and sample efficiency in RL)

Definition

Notations

Generic formulation for CO, especially graph problems such as TSP and MIS.

For an instance of a problem s with N variables:

- solution space is $\mathcal{X}_s = \{0, 1\}^N$
- objective $c_s(\boldsymbol{x}) = \mathrm{cost}(\boldsymbol{x}, s) + \mathrm{valid}(\boldsymbol{x}, s)$
  - cost is a real-valued function
  - valid is a $0/+\infty$ valued function.
- we write $\boldsymbol{x}_s^* = \min_{\boldsymbol{x}} c_s(\boldsymbol{x})$ or simply $\boldsymbol{x}_0$ when s is clear from context.

ML-based approach to CO

- from s we want to predict $\boldsymbol{x}_0$
- we want to learn in a supervised framework
- MLE: we want to maximize $\mathbb{E}_{\boldsymbol{x}_0 \sim q}[\log p_\theta(\boldsymbol{x}_0)]$

Definition (1)

DMs are Latent-Variables Probabilistic Models

T noisy versions of the observations generated before we see $\mathbf{x}_0$

$$p(\mathbf{x}_0) = \int p(\mathbf{x}_0, \mathbf{x}_1 \dots, \mathbf{x}_T) d\mathbf{x}_1 \dots d\mathbf{x}_T = \int p(\mathbf{x}_{0:T}) d\mathbf{x}_{1:T}$$

We assume that we can factorize p as *denoising* T steps:

$$p_\theta(\mathbf{x}_{0:T}) = p_\theta(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t)$$

The generation is reversible

Incremental mechanism to corrupt (*diffuse* noise) an observation

$$q(\mathbf{x}_{1:T}|\mathbf{x}_0) = \prod_{t=1}^{N} q(\mathbf{x}_t|\mathbf{x}_{t-1})$$

q has no learned parameters. Its parameterization is an hyper-parameter of the system.

Definition (2)

Variational Inference

Define a family of approximations, depending on a function (here $p_\theta(x_t|x_{t+1})$)

- Finding the best approximation by solving an optimization problem.
- When applied to maximizing probability of observations (*evidence*):
  - derive a lowerbound based on a auxiliary distribution
  - called ELBo (Evidence Lower Bound) (caveat minimization/maximization)

$$
\begin{aligned}
\mathbb{E}\left[-\log p_{\boldsymbol{\theta}}(\mathbf{x}_0)\right] &\leq \mathbb{E}_q\left[-\log \frac{p_{\boldsymbol{\theta}}(\mathbf{x}_{0:T})}{q_{\boldsymbol{\theta}}(\mathbf{x}_{1:T}|\mathbf{x}_0)}\right] \\
&= \mathbb{E}_q\left[\sum_{t>1} D_{KL}[q(\mathbf{x}_{t-1}|\mathbf{x}_t,\mathbf{x}_0)\|p_{\boldsymbol{\theta}}(\mathbf{x}_{t-1}|\mathbf{x}_t)] - \log p_{\boldsymbol{\theta}}(\mathbf{x}_0|\mathbf{x}_1)\right] + C
\end{aligned}
$$

  - KL sum: denoising matching terms
  - last: reconstruction term

Remark

$$
\begin{aligned}
-\log p_\theta(x_0|x_1) &= 1 \times (-\log p_\theta(x_0|x_1)) = q(x_0|x_1,x_0)(-\log p_\theta(x_0|x_1)) \\
&= q(x_0|x_1,x_0)(0 - \log p_\theta(x_0|x_1)) \\
&= q(x_0|x_1,x_0)(\log q(x_0|x_1,x_0) - \log p_\theta(x_0|x_1)) \\
&= q(x_0|x_1,x_0)\frac{\log q(x_0|x_1,x_0)}{\log p_\theta(x_0|x_1)} = KL[q(x_0|x_1,x_0)||p_\theta(x_0|x_1)]
\end{aligned}
$$

Learning DM

Diffusion Models are optimized via MC sampling:

1. Draw one instance s randomly
2. Draw a time step t randomly between 1 and T
3. Make on gradient descent step with loss:

$$\log q(x_{t-1}|\mathbf{x_t}, x_0) - \log p_\theta(x_{t-1}|x_t)$$

   where $\mathbf{x}_t$ is sampled from q and $\mathbf{x}_T$

The exact form of the loss depends on q and p_θ

Discrete or Continuous Distributions

Discrete Case (Bernoulli Model)

let β_t the corruption ratio, for changing 0 to 1 or 1 to 0 between timesteps

$$q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}) = \mathrm{Cat}(\boldsymbol{x}_t; \boldsymbol{p} = \tilde{\boldsymbol{x}}_{t-1}\boldsymbol{Q}_t) \text{ with } \boldsymbol{Q}_t = \begin{bmatrix} (1-\beta_t) & \beta_t \\ \beta_t & (1-\beta_t) \end{bmatrix}$$

- $\tilde{\boldsymbol{x}} \in \{0,1\}^{N\times 2}$ is a one-hot encoding of $\boldsymbol{x}$
- We can *compose* timesteps:
  $q(\boldsymbol{x}_t|\boldsymbol{x}_0) = \mathrm{Cat}(\boldsymbol{x}_t; \boldsymbol{p} = \tilde{\boldsymbol{x}_0}\boldsymbol{Q}_1\boldsymbol{Q}_2 \dots \boldsymbol{Q}_t) = \mathrm{Cat}(\boldsymbol{x}_t; \boldsymbol{p} = \tilde{\boldsymbol{x}_0}\bar{\boldsymbol{Q}}_t)$
- So we can express the first part of the loss as:

$$q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \boldsymbol{x}_0) = \frac{q(\boldsymbol{x}_t|\boldsymbol{x}_{t-1}, \boldsymbol{x}_0)q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_0)}{q(\boldsymbol{x}_t|\boldsymbol{x}_0)} = \mathrm{Cat}(\boldsymbol{x}_{t-1}; \frac{\tilde{\boldsymbol{x}_t}\boldsymbol{Q}_t^\top \odot \tilde{\boldsymbol{x}_0}\bar{\boldsymbol{Q}}_{t-1}}{\tilde{\boldsymbol{x}_0}\bar{\boldsymbol{Q}}_t\tilde{\boldsymbol{x}}_t{}^\top})$$

- from $\boldsymbol{x}_T$ and this definition, we can sample any x_t, then we train a neural network with parameters θ to predict $p_\theta(\tilde{\boldsymbol{x}_0}|\boldsymbol{x}_t)$
- Then, when generating a test solution, we can derive:

$$p_\theta(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t) = \sum_{\tilde{\boldsymbol{x}_0}} q(\boldsymbol{x}_{t-1}|\boldsymbol{x}_t, \tilde{\boldsymbol{x}_0})p_\theta(\tilde{\boldsymbol{x}_0}|\boldsymbol{x}_t)$$

Continuous Case (Gaussian Models)

By-The-Book application of DMs

- $\hat{x}_T$ is sampled from a $\mathcal{N}(0; I)$ and $\hat{x}_0$ is rescaled from $\{0, 1\}$ to $\{-1, 1\}$,
- With β_t the corruption ratio at timestep t:

$$q(\hat{\mathbf{x}}_t|\hat{\mathbf{x}}_{t-1}) \coloneqq \mathcal{N}(\hat{\mathbf{x}}_t; \sqrt{1-\beta_t}\hat{\mathbf{x}}_{t-1}, \beta_t\mathbf{I})$$

Via Gaussian properties

we define $\alpha_t = 1 - \beta_t$ and $\bar{\alpha}_t = \alpha_1 \cdots \alpha_t$. We obtain:

$$q(\hat{\mathbf{x}}_t|\hat{\mathbf{x}}_0) \coloneqq \mathcal{N}(\hat{\mathbf{x}}_t; \sqrt{\bar{\alpha}_t}\hat{\mathbf{x}}_0, (1-\bar{\alpha}_t)\mathbf{I})$$

Learning

Distance between *gaussians*, with same mean: amounts to predicting *the expected noise*

$$\tilde{\boldsymbol{\epsilon}}_t = (\hat{\mathbf{x}}_t - \sqrt{\bar{\alpha}_t}\hat{\mathbf{x}}_0)/\sqrt{1-\bar{\alpha}_t} = \bar{f}_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t, t)$$

Generation: p_θ becomes a Gaussian

$$p_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_{t-1}|\hat{\mathbf{x}}_t) = q\left(\hat{\mathbf{x}}_{t-1}|\hat{\mathbf{x}}_t, \frac{\hat{\mathbf{x}}_t - \sqrt{1-\bar{\alpha}_t} f_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t, t)}{\sqrt{\bar{\alpha}_t}}\right)$$

then final $\hat{x}_0$ is clipped to $\{0, 1\}$

Predicting Assignments

Neural Parameterization

To sum up, the model has to parameterize:

Discrete Case $p_\theta(\tilde{x_0}|x_t)$

$nn_\theta(x_t, t)$ returns 2 logits per variable that are passed through softmax to define p

Continuous Case

$$\widetilde{\boldsymbol{\epsilon}}_t = (\hat{\mathbf{x}}_t - \sqrt{\bar{\alpha}_t}\hat{\mathbf{x}}_0)/\sqrt{1-\bar{\alpha}_t} = f_{\boldsymbol{\theta}}(\hat{\mathbf{x}}_t, t)$$

$nn_\theta(x_t, t)$ returns 1 real number per variable used to parameterize a Gaussian:

Defined as a Graph Neural Network

Anisotropic

$$\hat{\boldsymbol{e}}_{ij}^{\ell+1} = \boldsymbol{P}^\ell \boldsymbol{e}_{ij}^\ell + \boldsymbol{Q}^\ell \boldsymbol{h}_i^\ell + \boldsymbol{R}^\ell \boldsymbol{h}_j^\ell,$$
$$\boldsymbol{e}_{ij}^{\ell+1} = \boldsymbol{e}_{ij}^\ell + \mathrm{MLP}_e(\mathrm{BN}(\hat{\boldsymbol{e}}_{ij}^{\ell+1})) + \mathrm{MLP}_t(\mathbf{t}),$$
$$\boldsymbol{h}_i^{\ell+1} = \boldsymbol{h}_i^\ell + \alpha(\mathrm{BN}(\boldsymbol{U}^\ell \boldsymbol{h}_i^\ell + \mathcal{A}_{j\in\mathcal{N}_i}(\sigma(\hat{\boldsymbol{e}}_{ij}^{\ell+1}) \odot \boldsymbol{V}^\ell \boldsymbol{h}_j^\ell)))$$

- vectors of size 256, 12 layers!
- $\mathbf{t}$ is the *sinusoidal* representation of t
  - $\mathbf{t}[2i] = \sin(t/T^{2i/256})$
  - $\mathbf{t}[2i+1] = \cos(t/T^{2i/256})$

Init

- TSP: e_{ij}^0 distance (i, j) and h_i^0 is the sinusoidal for timestep forall i
- for MIS e_{ij}^0 are zeros h_i^0 are the costs

From $p(x_0)$ to Assignment

Naive sampling from obtained distributions do not perform well... **:(**

Heatmaps

- discrete: $p_\theta(\mathbf{x}_0 = \mathbf{1}|s)$
- continuous $0.5(\hat{\mathbf{x}}_0 + 1)$

TSP Decoding

$A_{i,j}$ the heatmap

1. greedy decoding, rank edges by \$($A_{i,j}$ + $A_{j,i}$) / || c_i - c_j ||, add them one by one if no conflict (+option 2-opt)
2. MCTS, k transformation are sampled guided by heatmap

MIS

1. greedy decoding from heatmap A_i

Results

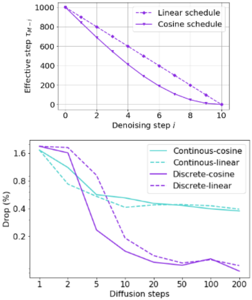


Figure 1: Comparison of continuous (Gaussian noise) and discrete (Bernoulli noise) diffusion models with different inference diffusion steps and inference schedule (`linear` v.s. `cosine`).

Table 1: Comparing results on TSP-50 and TSP-100. $*$ denotes the baseline for computing the performance gap. $\dagger$ indicates that the diffusion model samples a single solution as its greedy decoding scheme. Please refer to Sec. 4 for details.

ALGORITHM	TYPE	TSP-50 LENGTH↓	TSP-50 GAP(%)↓	TSP-100 LENGTH↓	TSP-100 GAP(%)↓
CONCORDE*	EXACT	5.69	0.00	7.76	0.00
2-OPT	HEURISTICS	5.86	2.95	8.03	3.54
AM	GREEDY	5.80	1.76	8.12	4.53
GCN	GREEDY	5.87	3.10	8.41	8.38
TRANSFORMER	GREEDY	5.71	0.31	7.88	1.42
POMO	GREEDY	5.73	0.64	7.84	1.07
SYM-NCO	GREEDY	-	-	7.84	0.94
DPDP	1k-IMPROVEMENTS	5.70	0.14	7.89	1.62
IMAGE DIFFUSION	GREEDY†	5.76	1.23	7.92	2.11
OURS	GREEDY†	**5.70**	**0.10**	**7.78**	**0.24**
AM	1k×SAMPLING	5.73	0.52	7.94	2.26
GCN	2k×SAMPLING	5.70	0.01	7.87	1.39
TRANSFORMER	2k×SAMPLING	5.69	0.00	7.76	0.39
POMO	8×AUGMENT	5.69	0.03	7.77	0.14
SYM-NCO	100×SAMPLING	-	-	7.79	0.39
MDAM	50×SAMPLING	5.70	0.03	7.79	0.38
DPDP	100k-IMPROVEMENTS	5.70	0.00	7.77	0.00
OURS	16×SAMPLING	**5.69**	**-0.01**	**7.76**	**-0.01**

TSP (2)

Table 2: Results on large-scale TSP problems. RL, SL, AS, G, S, BS, and MCTS denotes Reinforcement Learning, Supervised Learning, Active Search, Greedy decoding, Sampling decoding, Beam-search, and Monte Carlo Tree Search, respectively. * indicates the baseline for computing the performance gap. Results of baselines are taken from Fu et al. [27] and Qiu et al. [92], so the runtime may not be directly comparable. See Section 4 and appendix for detailed descriptions.

Algorithm	Type	TSP-500			TSP-1000			TSP-10000		
		Length ↓	Gap ↓	Time ↓	Length ↓	Gap ↓	Time ↓	Length ↓	Gap ↓	Time ↓
Concorde	Exact	16.55*	—	37.66m	23.12*	—	6.65h	N/A	N/A	N/A
Gurobi	Exact	16.55	0.00%	45.63h	N/A	N/A	N/A	N/A	N/A	N/A
LKH-3 (default)	Heuristics	16.55	0.00%	46.28m	23.12	0.00%	2.57h	71.77*	—	8.8h
LKH-3 (less trails)	Heuristics	16.55	0.00%	3.03m	23.12	0.00%	7.73m	71.79	—	51.27m
Farthest Insertion	Heuristics	18.30	10.57%	0s	25.72	11.25%	0s	80.59	12.29%	6s
AM	RL+G	20.02	20.99%	1.51m	31.15	34.75%	3.18m	141.68	97.39%	5.99m
GCN	SL+G	29.72	79.61%	6.67m	48.62	110.29%	28.52m	N/A	N/A	N/A
POMO+EAS-Emb	RL+AS+G	19.24	16.25%	12.80h	N/A	N/A	N/A	N/A	N/A	N/A
POMO+EAS-Tab	RL+AS+G	24.54	48.22%	11.61h	49.56	114.36%	63.45h	N/A	N/A	N/A
DIMES	RL+G	18.93	14.38%	0.97m	26.58	14.97%	2.08m	86.44	20.44%	4.65m
DIMES	RL+AS+G	17.81	7.61%	2.10h	24.91	7.74%	4.49h	80.45	12.09%	3.07h
Ours (DIFUSCO)	SL+G†	18.35	10.85%	3.61m	26.14	13.06%	11.86m	98.15	36.75%	28.51m
Ours (DIFUSCO)	SL+G†+2-opt	**16.80**	**1.49%**	**3.65m**	**23.56**	**1.90%**	**12.06m**	**73.99**	**3.10%**	**35.38m**
EAN	RL+S+2-opt	23.75	43.57%	57.76m	47.73	106.46%	5.39h	N/A	N/A	N/A
AM	RL+BS	19.53	18.03%	21.99m	29.90	29.23%	1.64h	129.40	80.28%	1.81h
GCN	SL+BS	30.37	83.55%	38.02m	51.26	121.73%	51.67m	N/A	N/A	N/A
DIMES	RL+S	18.84	13.84%	1.06m	26.36	14.01%	2.38m	85.75	19.48%	4.80m
DIMES	RL+AS+S	17.80	7.55%	2.11h	24.89	7.70%	4.53h	80.42	12.05%	3.12h
Ours (DIFUSCO)	SL+S	17.23	4.08%	11.02m	25.19	8.95%	46.08m	95.52	33.09%	6.59h
Ours (DIFUSCO)	SL+S+2-opt	**16.65**	**0.57%**	**11.46m**	**23.45**	**1.43%**	**48.09m**	**73.89**	**2.95%**	**6.72h**
Att-GCN	SL+MCTS	16.97	2.54%	2.20m	23.86	3.22%	4.10m	74.93	4.39%	21.49m
DIMES	RL+MCTS	16.87	1.93%	2.92m	23.73	2.64%	6.87m	74.63	3.98%	29.83m
DIMES	RL+AS+MCTS	16.84	1.76%	2.15h	23.69	2.46%	4.62h	74.06	3.19%	3.57h
Ours (DIFUSCO)	SL+MCTS	**16.63**	**0.46%**	**10.13m**	**23.39**	**1.17%**	**24.47m**	**73.62**	**2.58%**	**47.36m**

Conclusion

Summary

- A lot of Maths!
- SOTA results on 2 benchmarks (with lot of competitors)
- modelisation tailored for graph problems
- A new? GNN architecture

Questions

- Can we take into account decomposition in this framework (cf. recent works in NLP)?

SSD-LM: Semi-autoregressive Simplex-based Diffusion Language Model for Text Generation and Modular Control

Xiaochuang Han[♠] **Sachin Kumar**[♣] **Yulia Tsvetkov**[♠]
[♠]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♣]Language Technologies Institute, Carnegie Mellon University
{xhan77, yuliats}@cs.washington.edu[♠] sachink@cs.cmu.edu[♣]