Learning TSP Requires

Rethinking Generalization

최우진

2021/10/29

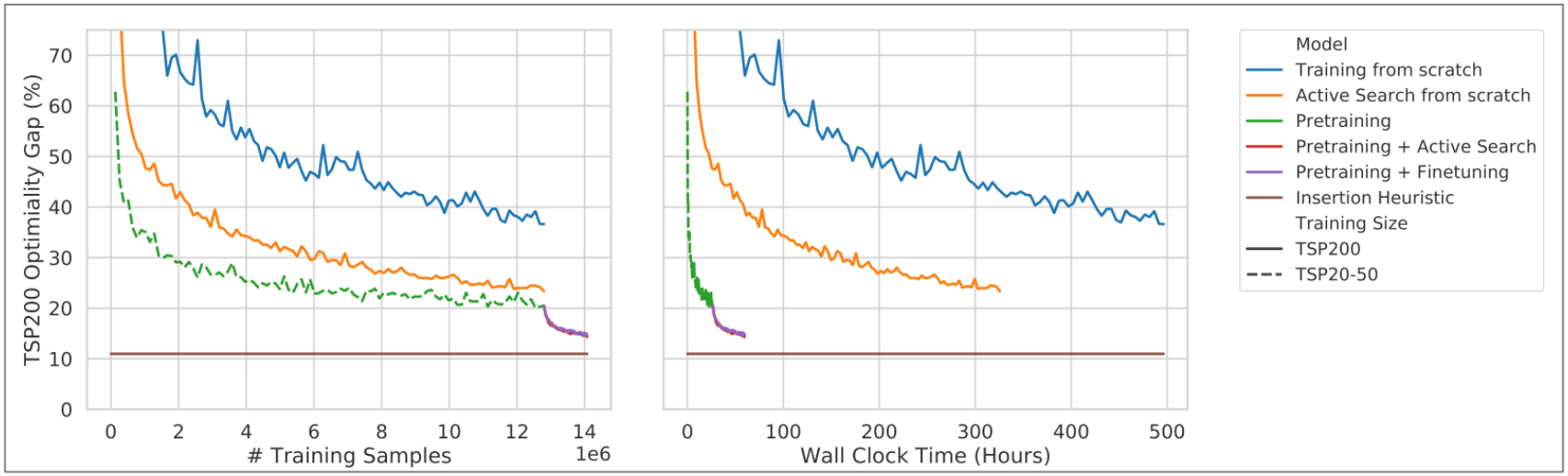
0. Abstract

- 조합 최적화 문제(TSP)에 대한 end-to-end 신경망 학습은 수백 개의 노드를 넘어서는 다루기 어렵고 비효율적이다.
- 머신러닝 방법은 작은 사이즈의 데이터 셋에서 classical solver와 성능이 비슷하지만, 학습된 정책을 더 큰 규모의 데이터 셋에 적용할 수는 없다.
- 이 논문은 대규모 TSP를 해결하기 위한 전이 학습을 활용하여, 학습에서 볼 수 있는 것보다 더 큰 인스턴스에 대한 일반화를 촉진하는 방법을 제시한다.

1. Introduction

- NP-hard 조합 문제(예: 2D 유클리드 TSP)는 최적의 해를 찾기 어렵다.
- OR Solver
  - 예: Concorde TSP solver
  - 시간이 오래 걸리고, 특정한 문제의 해결책만 제공
  - 이러한 해결책을 만들기 위해 상당한 시간과 해당 분야의 전문지식이 필요
- 기계학습에서는 조합문제를 직접 해결하는 일반적인 학습 알고리즘을 개발하고자 한다.
- End-to-end 방법은 매우 작은 데이터셋(최대 수백 개의 노드)에서 OR Solver와 비슷한 성능

1.1. Motivation



- 대규모 그래프에 대한 최신 모델의 학습시간이 오래 걸린다.
- 간단한 삽입 휴리스틱 알고리즘보다 성능이 좋지 못한다.
- 매우 작은 TSP로부터 효율적으로 학습하고 학습된 정책을 zero-shot 방식 또는 빠른 fine-tuning을 통해 규모가 더 큰 그래프로 전송

1.2. Contributions

- 규모 불변 TSP Solver의 end-to-end 학습

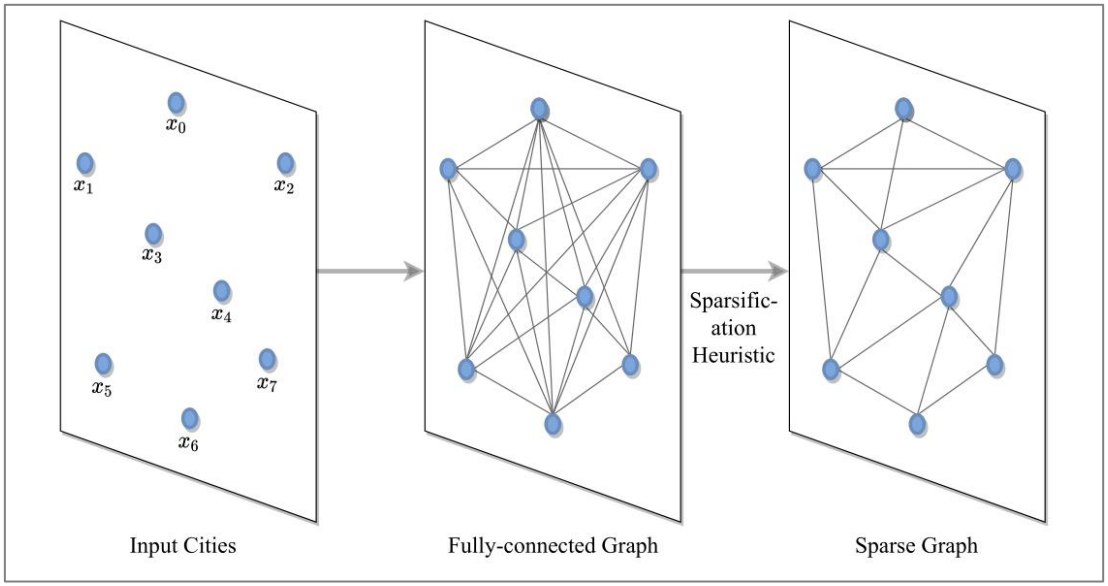
  - 평가 방법 수정: 고정 or 사소하게 작은 TSP 크기에 대한 성능 측정하지 않는다.
  - GNN 집계 함수와 정규화 일반화 성능 개선
  - 자기 회귀 디코딩: 일반화를 개선하는 순차 귀납적 편향을 사용하지만, 추론 시간이 길다.
  - 지도 학습은 사후 검색에 적합
  - 강화 학습은 정답 데이터에 의존하지 않기 때문에 더 많은 연산을 통해 더 확장이 가능하여 일반화에 좋다.

3. Neural Combinatorial Optimization Pipeline

- 많은 NP-hard 문제는 매우 구조화된 특성 때문에, 그래프에 대한 순차적 의사 결정 작업으로 공식화가 가능하다.

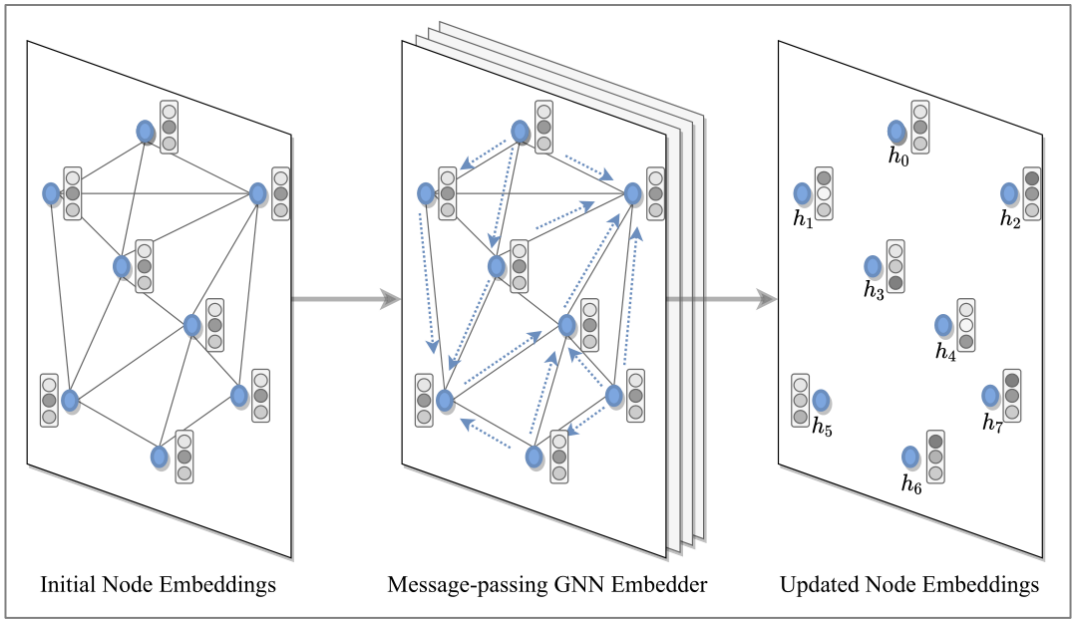
  1. 문제 정의

  - 입력을 완전 연결 그래프로 변환
  - 완전 연결 그래프를 희소 그래프로 변환



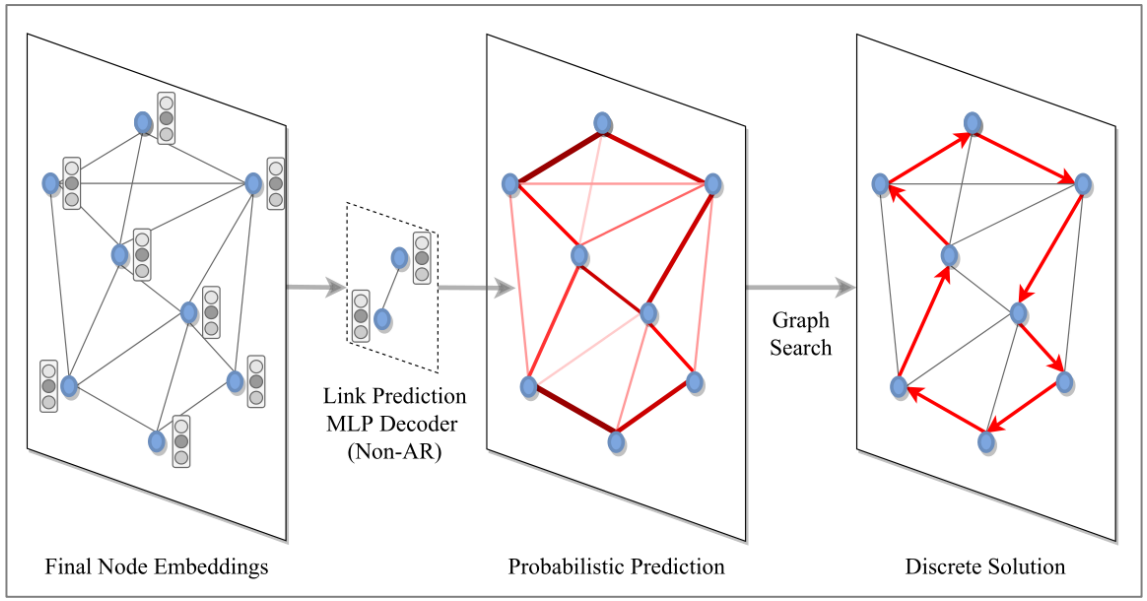
3. Neural Combinatorial Optimization Pipeline

- 2. 그래프 임베딩
  - 노드 임베딩 초기화
  - 주변에 대한 관계를 임베딩에 업데이트



3. Neural Combinatorial Optimization Pipeline

- 3. 솔루션 디코딩 & 탐색
  - 노드 임베딩의 값을 통해 해당 링크(edge)가 TSP 투어에 속하는 지에 대한 확률 계산
  - 계산된 확률을 통해 edge 선택



3.1. Problem Definition

- 2D 유클리드 TSP: 도시들의 집합과 각각의 도시들 사이의 거리가 주어질 때, 각 도시를 방문하고 원래 도시로 돌아오는 가장 짧은 경로
- $S = \{x_i\}_{i=1}^{n}, x_i \in [0,1]^2$인 n개의 도시의 완전 연결 입력 그래프가 주어지면, 노드 π의 순열(투어)을 찾는 것

$$L(\pi|s) = \|x_{\pi_n} - x_{\pi_1}\|_2 + \sum_{i=1}^{n-1} \|x_{\pi_i} - x_{\pi_{i+1}}\|_2,$$

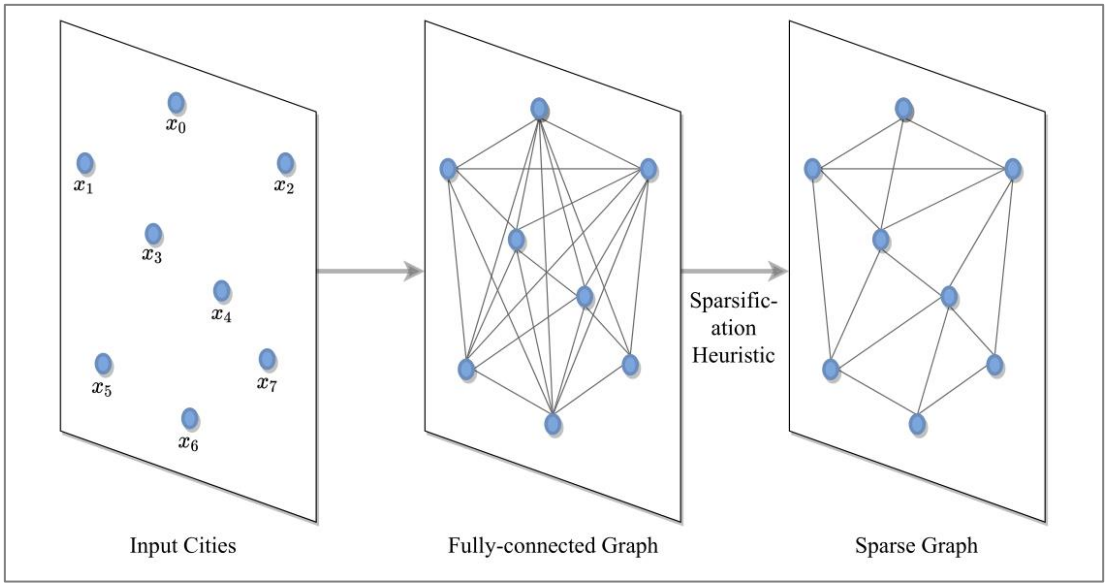
- $\|\cdot\|_2$은 L2 norm

$$L_2 = \sqrt{\sum_i^n x_i^2}$$
$$= \sqrt{x_1^2 + x_2^2 + x_3^2 + \ldots + x_n^2}$$

그림 출처: http://taewan.kim/post/norm/

3.1. Problem Definition

- Graph Sparsification
- 보통 TSP는 완전 연결 그래프로 정의된다.
- K-nearest neighbor 방법으로 그래프를 희소화하여 학습 속도 개선



3.2. Graph Embedding

- GNN 인코더는 입력 TSP 그래프의 각 노드에 대한 d차원 표현을 계산

  - 각 계층에서, 노드는 인접 계층으로부터 특징을 수집하여 재귀 메시지 전달을 통해 로컬 그래프 구조를 나타낸다.

  - L개의 계층을 쌓으면, 네트워크가 각 노드의 L-hop 이웃으로부터 해당 노드를 표현

$$
\begin{aligned}
h_i^{\ell+1} &= h_i^{\ell} + \mathrm{ReLU}\Big(\mathrm{NORM}\Big(U^{\ell} h_i^{\ell} + \mathrm{AGGR}_{j \in \mathcal{N}_i}\Big(\sigma(e_{ij}^{\ell}) \odot V^{\ell} h_j^{\ell}\Big)\Big)\Big)\,, \\
e_{ij}^{\ell+1} &= e_{ij}^{\ell} + \mathrm{ReLU}\Big(\mathrm{NORM}\Big(A^{\ell} e_{ij}^{\ell} + B^{\ell} h_i^{\ell} + C^{\ell} h_j^{\ell}\Big)\Big),
\end{aligned}
$$

- $\boldsymbol{h_i^l, e_{ij}^l}$은 각각 L 계층의 node i의 특징, edge ij의 특징

  $U^l, V^l, A^l, B^l, C^l \in \mathbb{R}^{d\,x\,d}$: 학습 파라미터

  NORM: 정규화 층(BatchNorm, LayerNorm), AGGR: 이웃 합계 함수 (SUM, MEAN, MAX)

  σ: 시그모이드 함수, $\odot$: Hadamard product

3.3. Solution Decoding

- Non-autoregressive Decoding (NAR)
- TSP를 각 노드 연결 예측 작업으로 간주

  - 각 edge는 다른 edge와 독립적이며, 최적의 TSP 경로에 포함 여부를 계산
  - edge predictor를 최종 GNN 인코더 L 계층에 의해 생성된 노드 임베딩의 2계층 MLP로 정의

$$\hat{p}_{ij} = W_2\Big(\mathrm{ReLU}\ \big(W_1\big(\ [\ h_G,\ h_i^L,\ h_j^L\ ]\ \big)\big)\Big),\quad \text{where}\quad h_G = \frac{1}{n}\sum_{i=0}^{n} h_i^L$$

- $[\cdot,\cdot,\cdot]$: 연결 연산자, $\hat{p}_{ij}$는 Softmax를 통해 각 edge p_{ij}에 대한 확률을 계산
- NAR 디코더는 한 번에 예측을 생성하여 빠르지만, TSP 투어의 순차적 순서를 무시한다.

3.3. Solution Decoding

- Autoregressive Decoding (AR)

  - Attention or RNN을 기반으로 단계별 그래프 통과를 통해 순차 귀납적 편향을 모델링
  - 무작위 노드에서 시작하여, 각 단계에서 이웃에 대한 확률 분포를 출력하는 attention decoder를 사용
  - 결과를 예측할 때는 Greedy Search를 사용
  - 마스킹을 통해 방문한 노드를 재방문하지 못하게 제약

- 시간 t에서 노드 i의 컨텍스트 벡터

  - $\widehat{h}_i^C = \mathbf{W}_C[\mathbf{h}_G, \mathbf{h}^L_{\pi'_{t-1}}, h^L_{\pi'_1}]$이며, $\mathbf{h}^L_{\pi'_t}$: 부분 투어 π'의 노드 t의 임베딩

- 컨텍스트 벡터 $\widehat{h}_i^C$를 노드 임베딩에 대한 표준 MHA 연산 수행 (M = 8)

$$h_i^C = \mathrm{MHA}\left(Q = \hat{h}_i^C, K = \{h_1^L, \ldots, h_n^L\}, V = \{h_1^L, \ldots, h_n^L\}\right),$$

3.3. Solution Decoding

- Autoregressive Decoding (AR)

  - 각 edge e_{ij}에 대한 비정규화된 logits은 컨텍스트 h_i^C와 임베딩 h_j 사이의 최종 attention mechnism을 통해 계산

  - 계산된 logits은 모든 edge에서 softmax를 통해 확률 p_{ij}로 변환

  - tanh은 [-C, C] 내에서 logits 값을 유지하기 위해 사용 (C = 10)

$$\hat{p}_{ij} = \begin{cases} C \cdot \tanh\left(\frac{\left(W_Q h_i^C\right)^T \cdot \left(W_K h_j^L\right)}{\sqrt{d}}\right) & \text{if } j \neq \pi_{t'} \quad \forall t' < t \\ -\infty & \text{otherwise.} \end{cases}$$

3.4. Solution Search

- Greedy Search

  - AR 디코딩의 경우, 노드 i의 예측 확률은 확률 분포 p_i에서 샘플링을 통해 현재 단계에서 이동할 edge를 선택하거나 가장 가능성이 높은 edge p_{ij}를 선택한다.
  - NAR 디코딩의 경우, 서로 독립적으로 모든 edge에 걸쳐 확률을 직접 출력한다.
  - 무작위 노드에서 시작해서 방문한 노드를 마스킹하여 유효한 TSP 투어를 얻을 수 있다.

- Beam Search and Sampling

  - 디코딩을 통해 추론을 할 때, b개의 가장 가능성이 높은 투어를 유지한다.
  - b개의 투어들을 샘플링하고, 그 중에서 가장 짧은 투어를 선택한다.
  - 좋은 일반화를 위해, 큰 값의 b를 사용하지 않는다.

4. Experiments

- 4.1. Controlled Experiment Setup
- 1. 작은 문제 인스턴스(TSP20-50)에서 효율적으로 학습
  - 수억 개의 훈련 샘플과 모델 매개 변수 없이 사소하게 작은 TSP 학습
- 2. 학습하기 어려운 큰 인스턴스(TSP200)를 포함하여 더 넓은 범위의 크기로 일반화
  - 훈련에서 볼 수 있는 것보다 더 작고 더 큰 사례에 잘 일반화
- 3. '좋은' 일반화 정량화
  - 간단하고 학습되지 않은 furthest insertion heuristic baseline을 통해 모델 평가

4.1. Controlled Experiment Setup

- Training Datasets
  - 가변 TSP20-50 그래프를 통해 모델 학습
  - 고정 그래프 크기 TSP20, 50, 100에 대해서 비교 모델 학습

- Supervised Learning vs. Reinforcement Learning
  - 공통: batch size: 128, learning rate: 1e-4
  - 128만 개의 TSP 샘플 (Concorde Solver 사용), Adam optimizer
  - 12.8만 개의 TSP 샘플 (랜덤 생성), Epoch: 100

4.1. Controlled Experiment Setup

- Model Hyperparameters
- AR 디코더가 있는 모델 vs. NAR 디코더가 있는 모델
  - 공통: hidden dimension d = 128으로 설정, 약 350,000개의 훈련 매개 변수 생성
  - 어텐션 디코더 헤드를 사용하는 GNN 인코더 계층을 3개 사용
  - edge 예측기를 사용하는 GNN 인코더 계층을 4개 사용
- 대부분의 실험
  - MAX aggregation, BatchNorm(배치 통계 사용)을 사용한 Graph ConvNet 인코더 사용
  - AR 디코더 사용
  - 학습 방식을 비교할 때를 제외하고 모든 모델은 지도학습으로 훈련

4.1. Controlled Experiment Setup

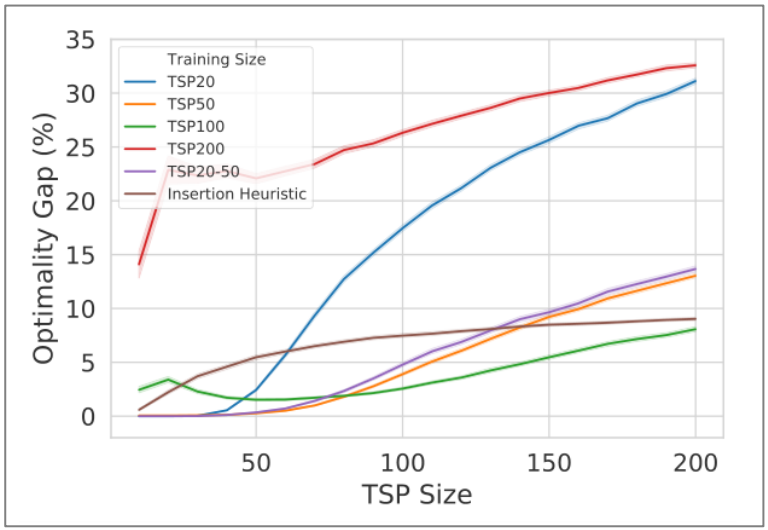
- Evaluation
- Test set

  - TSP10, 20, ···, 200 각각 1,280개의 샘플로 구성 (총 25,600개)

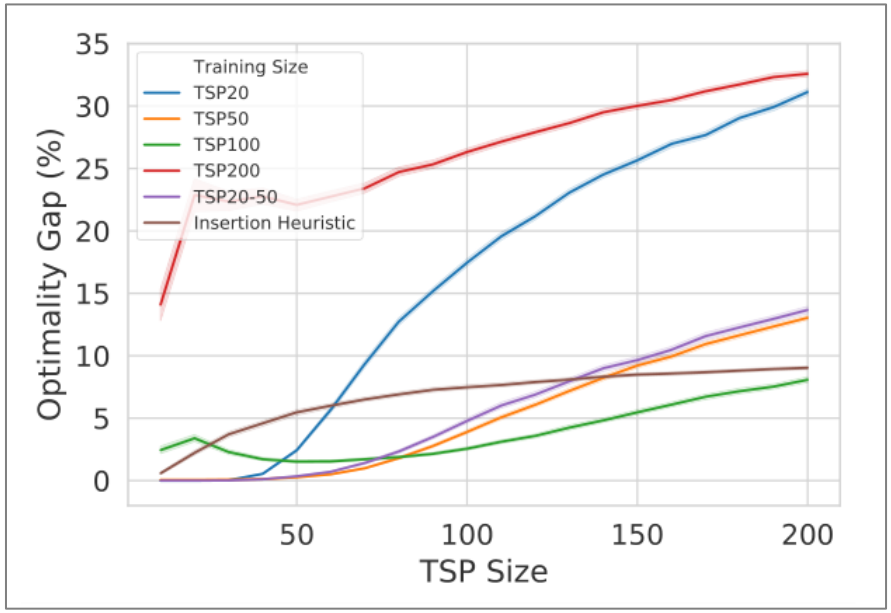
- 평가 지표

  - Concorde Solver와의 최적성 차이 = 최적의 투어 길이에 대한 예측 투어 길의 평균 백분율

- 예시

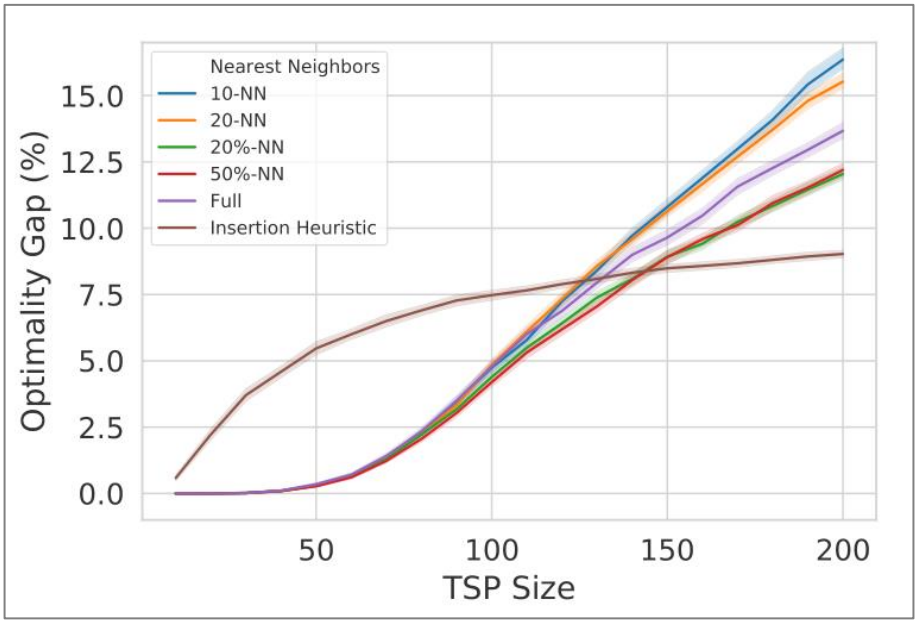


4.2. Does learning from variable graphs help generalization?



- TSP20, 50, 100, 200, 20-50 완전 연결 그래프 동일한 모델 학습
  - TSP20-50와 TSP50의 성능이 비슷하게 유지
  - 작은 TSP20에서 학습은 큰 크기로 일반화할 수 없다.
  - TSP100은 쉽게 일반화 되지만, 학습시간이 길다.
  - 신속한 실험을 위해 TSP20-50에 대해 학습을 진행한다.

4.3.What is the best graph sparsification heuristic?

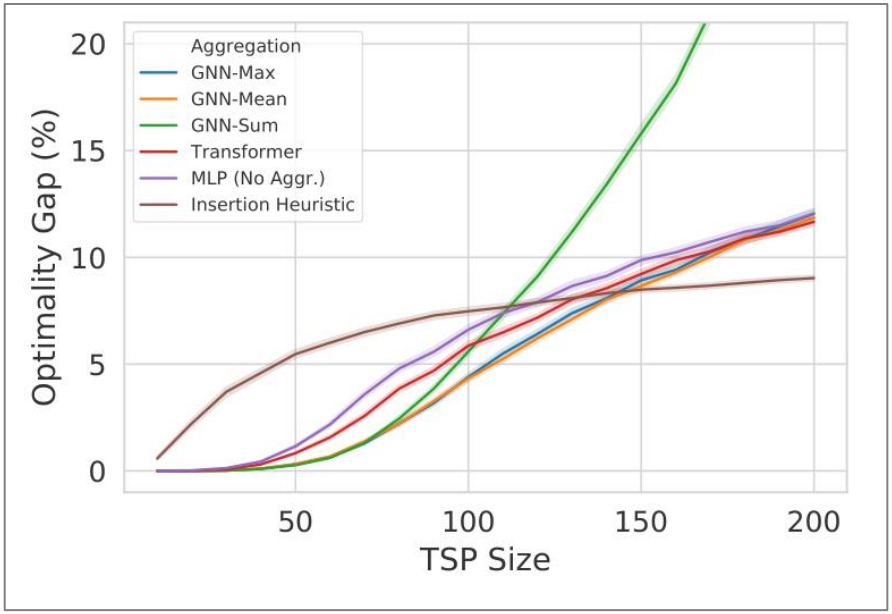


- Graph sparsification

  1. k-근접 이웃을 통해 TSPn의 각 노드를 연결하여 GNN 계층이 일정한 degree k로 고정
  2. n x k%-근접 이웃을 통해 TSPn의 각 노드를 연결하여 일정한 graph diameter 고정

- 두 방법 모두 빠른 수렴을 하지만, 2번 방법이 더 나은 일반화 성능을 보인다. (k = 20)

4.4.What is the relationship between GNN aggregation functions and normalization layer?

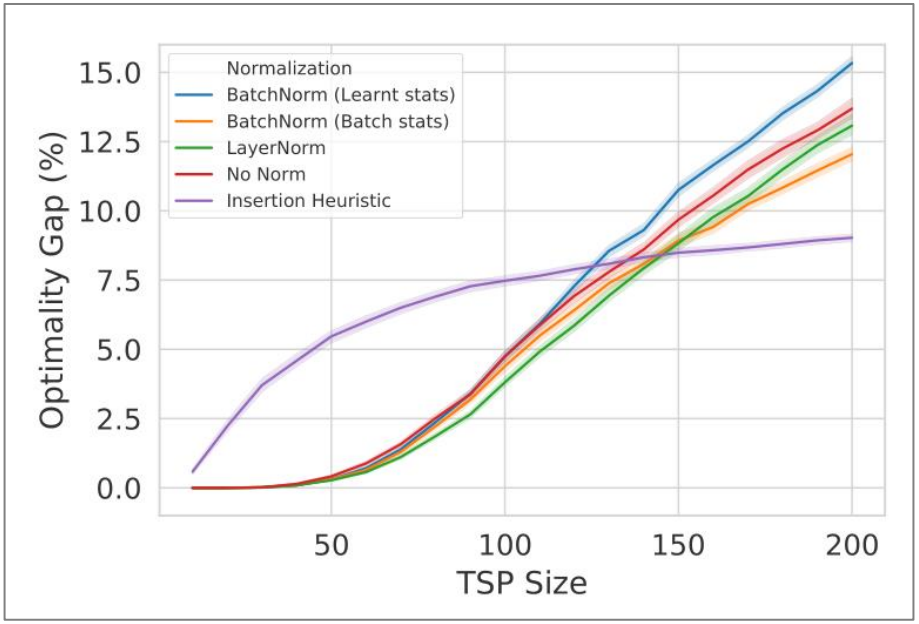


- GNN aggregation functions

  - SUM 성능이 제일 좋지 못하고, 나머지는 MLP보다 성능이 약간 좋다.

  - MAX aggregator가 일반화에 좋은 성능을 보여서 추가 실험에 사용

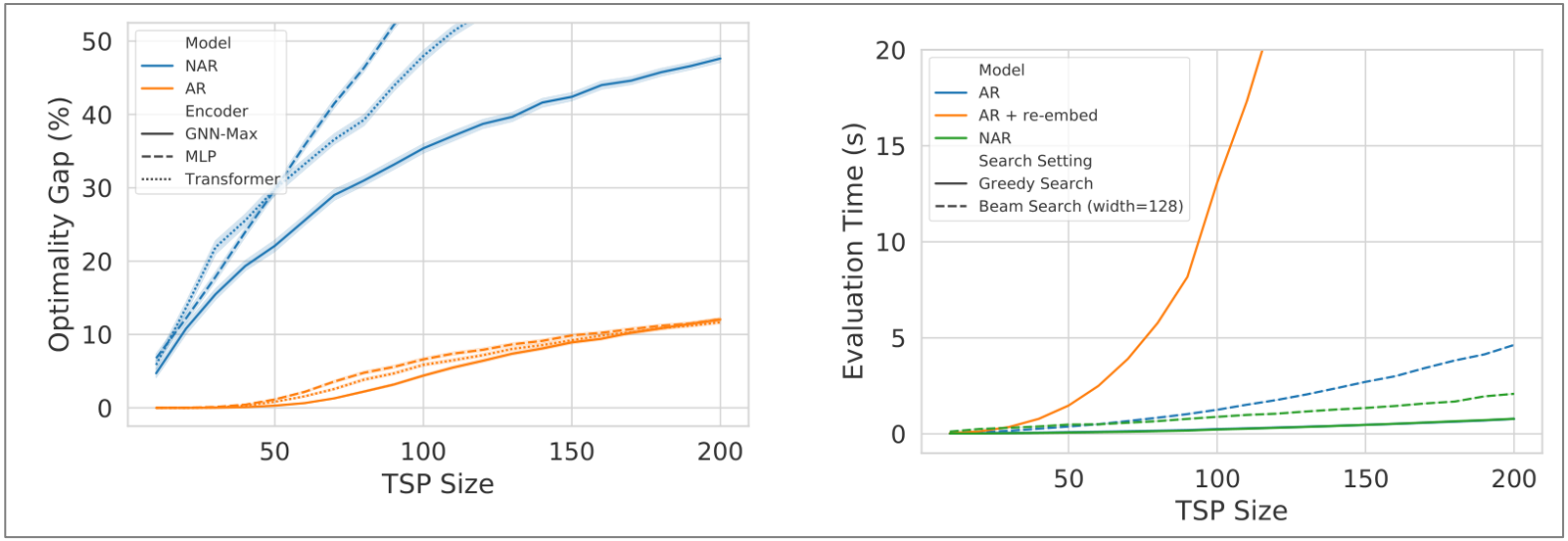
4.4.What is the relationship between GNN aggregation functions and normalization layer?



- Normalization

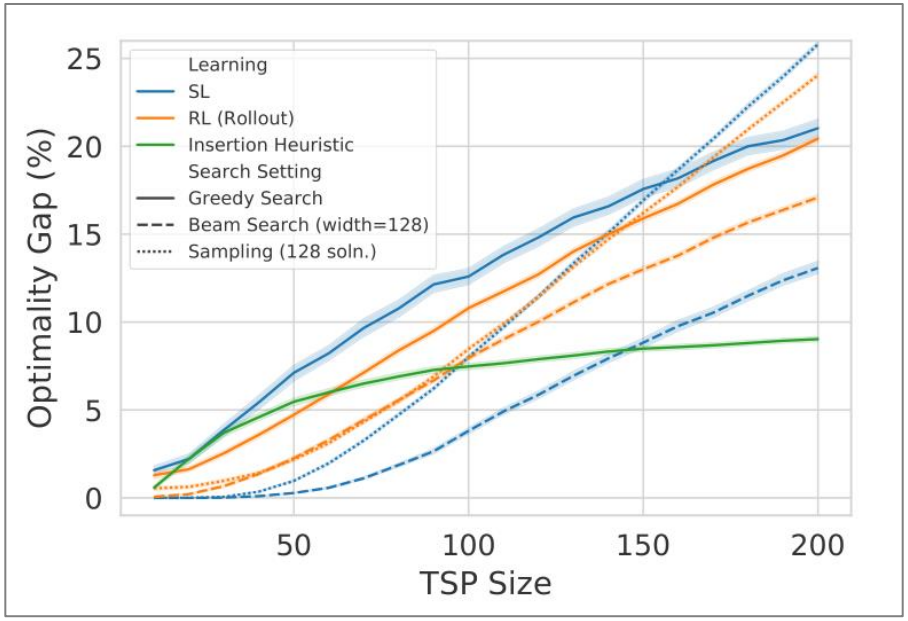
  1. 학습 데이터로부터 평균과 분산을 학습하는 표준 BatchNorm
  2. 배치 통계량을 사용하는 BatchNorm (정규화가 제일 잘되어 추후 실험에 사용)
  3. 임베딩 차원을 정규화하는 LayerNorm

4.5.Which decoder has a better inductive bias for TSP?



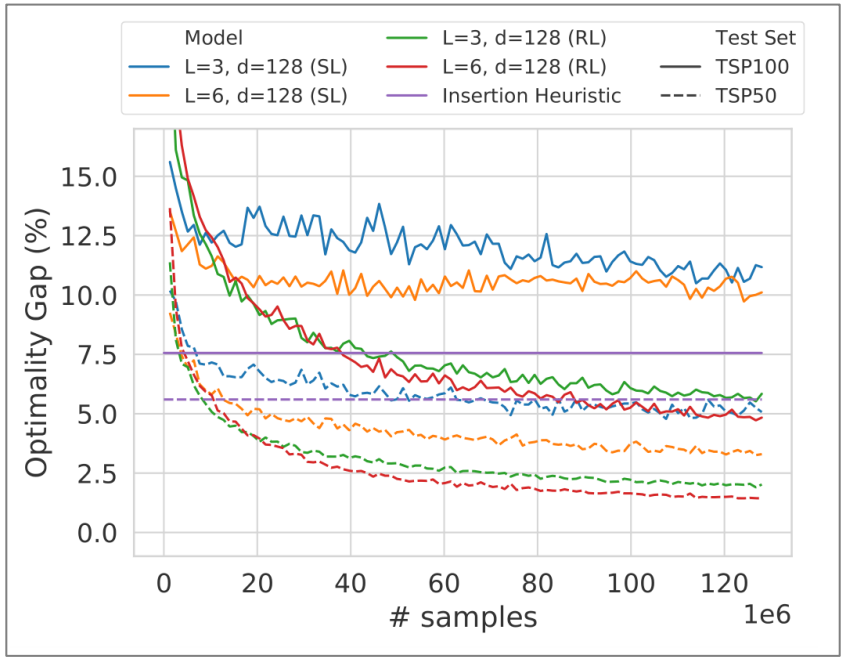
- AR decoder vs. NAR decoder
  - AR decoder가 NAR decoder보다 잘 훈련데이터를 적합 시킬 수 있다.
  - 그래프 정보가 없어도 순차 디코딩이 TSP에 더 좋은 성능을 보인다.
  - NAR decoder가 AR decoder보다 추론 속도가 빠르다.

4.6. How does the learning paradigms impact the search phase?



- Supervised Learning vs. Reinforcement Learning
  - RL 모델이 Greedy Search, Sampling에서 성능이 좋다.
  - SL 모델은 Beam Search에서 성능이 좋다.

4.7. Which learning paradigm scales better?



- 학습 데이터 샘플의 수가 늘어날 수록 성능이 좋아진다.
- TSP100에서 샘플수가 많아져도 SL은 변화가 없지만, RL은 지속적으로 성능이 개선된다.
- 따라서, RL 모델이 큰 TSP를 학습하기에 좋은 모델이다.

5. Conclusion

- TSP와 같은 조합 문제에 대한 학습 기반 모델은 작은 데이터 셋에 대해 좋은 결과를 보여주었지만, 큰 데이터 셋에 대한 훈련은 오래 걸리기 때문에 end-to-end 학습 방법을 사용하기 어렵다.
  - TSP에서 200개 노드를 초과하는 경우, 간단한 삽입 휴리스틱 방법을 능가할 수 없다.
- 작은 TSP에서 모델을 효율적으로 학습하고, 학습된 정책을 zero-shot 방식이나 fine-tuning을 통해 더 큰 그래프로 전송하는 모델을 통해 해결해보고자 한다.

Rethinking Generalization

감사합니다