2021-2022 春季学期创新实践 II 第一次答辩报告

面向旅行商问题的神经网络求解器复现及泛化性能测试

吕昊泽 11912814

张子研 11912324

指导老师： 唐珂教授 刘晟材老师

2022. 3. 22

目录

1 项目介绍

1.1 背景描述

1.1.1 TSP 问题

TSP(Traveling Salesman Problem) 问题，又称旅行商问题，是一个经典组合优化问题。其定义如下：一名旅行商要拜访 n 个城市，他需要选择最短的拜访路径，使得每个城市都被拜访一次，并最终回到原点。

目前解决 TSP 问题的手段有精确方法与启发式算法，其中精确算法虽然可以得到最优解，但其时间开销呈指数增大，而启发式算法可以在较短时间内获得较优解。

TSP 问题在诸多现实领域如物流、DNA 测序中都有涉及。也有许多类似的衍生问题，如非对称 TSP 问题（城市来回距离不相等）、旅行推销员问题（部分城市合成一个国家，旅行商需要在每个国家恰好访问一所城市）等。

1.1.2 传统求解器介绍

① Concorde 求解器

Concorde 求解器[3]由 David Applegate, Robert E. Bixby 等人于 1997 年开发，后两次更改，最新版本为 2003 年更新，对学术界开源，是一种组合优化问题的精确求解器。Concorde 求解器曾一度被认为是“现存的最快速的 TSP 问题求解器”。

② LKH 求解器

LKH[6]求解器基于 Lin-Kernighan 启发式算法。LK 算法使用 λ-opt，将原始解切断、重新连接获得更优解。LKH 求解器优化了 LK 算法，增加候选集合，忽略有较小概率可能构成最优解的链路。

1.2 项目简介

1.2.1 项目目标

本学期我们将基于近年来 TSP Neural Solver 中表现较为出色的 AM 模型[1]作为基础，将之与传统 TSP 求解器 Concorde[5], LKH[6]等性能相比较，研究其泛化求解性能。

1.2.2 时间安排

① 第一次答辩，我们预计将完成 AM[1]模型的代码构建，使得自身能对于 AM 模型进行研究甚至是适当的修改，尝试复现论文中的实验并以此对其性能进行初步验证。同时，完成对于 Concorde 等专业求解器的了解，并做好数据迁移工作，让 AM 模型自生成的 TSP 数据集能够被专业求解器进行求解，从而初步测试 AM 模型的性能。同时完成 TSPLIB 数据集的了解和相关迁移脚本的编写，使得后续可以在 TSPLIB 数据集上对于神经网络求解器有更深层次的性能测试。

② 第二次答辩，我们预计完成测试数据集的生成，用更高质量的数据测试 TSP 求解器，并尽可能解决当前测试下精度的问题。同时，我们也将挑选更多 TSP 求解器，用我们生成的数据集对其完成相应测试。

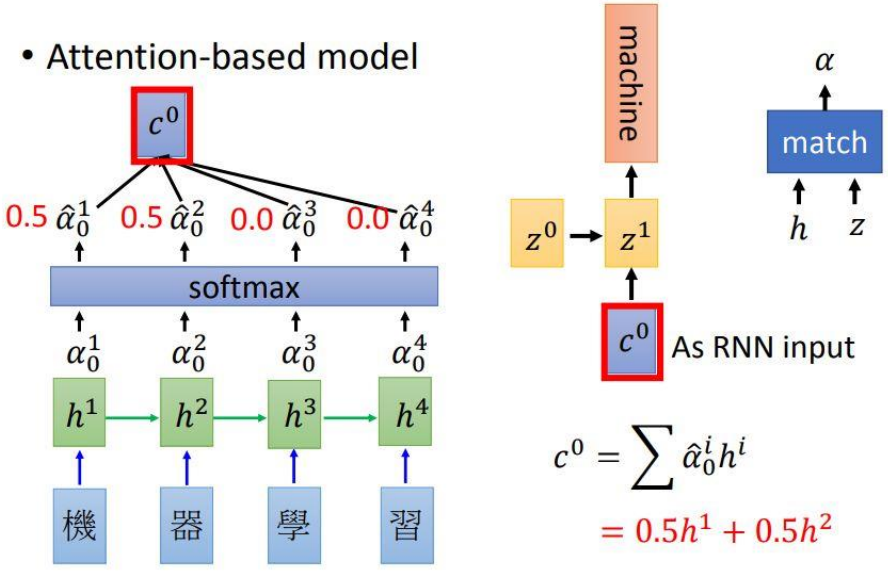
1.3 报告结构

在本篇报告中，在第二部分简介 AM 模型，在第三部分介绍 AM 模型论文实验的复现情况，在第四部分简单介绍用到的专业求解器和 TSPLIB 数据集迁移相关实验， 在第五部分将进行总结和进一步规划。

2 AM 模型简介

2.1 模型背景知识简介

2.1.1 Attention 机制

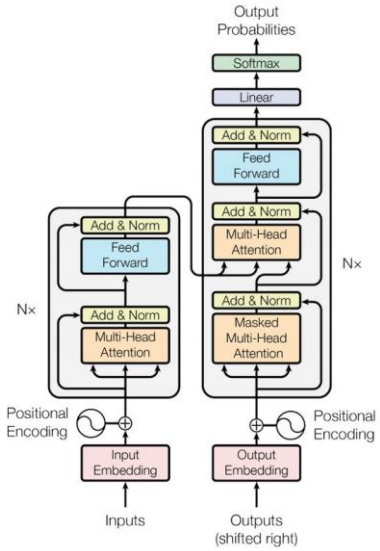


图表 1 简单展示了 Attention 机制的工作原理

Attention 机制最早被提出用来解决自然语言处理中模型无法体现对一个句子序列中不同语素的关注程度。Attention 机制通过构建 key-value query 的形式，实现了根据 key query 之间的相似程度对不同的 value 施加不同注意力的机制。如图表 1 所示，一般的做法为将 encode 解码出的各个词对应的特征向量作为 key value 对，在 Decode 过程中，不断将 output 作为 query 的形式与 key 进行比较，通过定义的距离衡量标准计算出 query 与各个 key 的相似程度，并根据其相似程度将 key 对应的 value 根据其相似程度加和，达到根据 query 选取想要关注的 value 的目的。

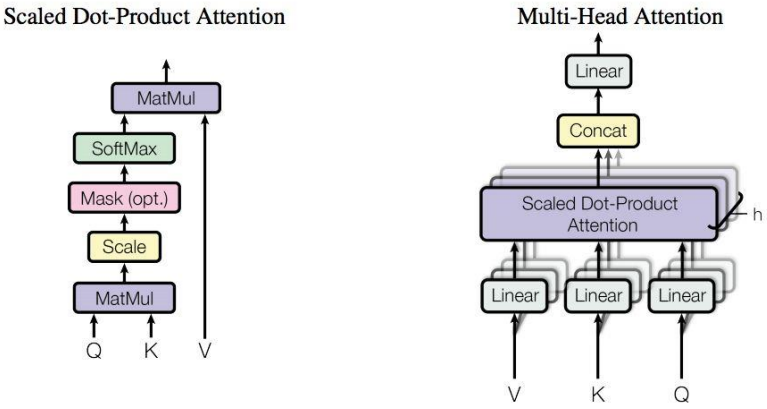
这里 key value query 如果相同即为自注意力机制。

2.1.2 Transformer



图表 2 Tranformer 架构示意图

2017 年，Google 在 seq2seq 模型的基础上，根据 Encoder-Decoder 模型提出了 Transformer[2]模型，该模型使用全 Attention 结构代替了传统的 RNN，使得并行效率得到了飞跃，在 NLP 任务上取得了巨大的突破。



图表 3 Multi-Head Attention 机制的工作原理

Transformer 大量使用了 Multi-Head Attention 机制，该机制是不同 Attention 机制的一个变体。作者先是使用前文提到的自注意力机制，对每层的 Value 都根据自注意力机制算法进行更新，同时作者又想学习 CNN 模型中卷积的多通道机制，用不同的 Self-Attention 权重对不同的信息进行提取。如图表 3 所示，使用 h 个 MLP 层对 V,K,Q 进行映射得到 h 对 V,K,Q，后分别进行 Self-Attention 运算，后再将更新过的 k 个拼接并使用 MLP 映射回原来的空间。

作者同时在模型中大量使用了 Layer Norm 技术和残差连接技术。

2.1.3 Point Network

Point Network[5]最早在 2015 年提出，当时基于 Attention 机制的 seq2seq 模型已经存在。作者察觉到 Attention 机制中的 Softmax 层输出的权重可被视为概率，从此开启了利用深度神经网络进行组合优化问题求解的一系列研究。对于 TSP 问题，Attention 机制输出的权重被认为是各个节点的选择概率，作者通过自回归的方式逐步选择节点生成 TSP 问题的解。

随后在 2017 年，随着 Transformer 的提出，改进后的 Ponit Network 架构被广泛使用，本文的模型就基于此。

2.1.4 Reinforcement Learning

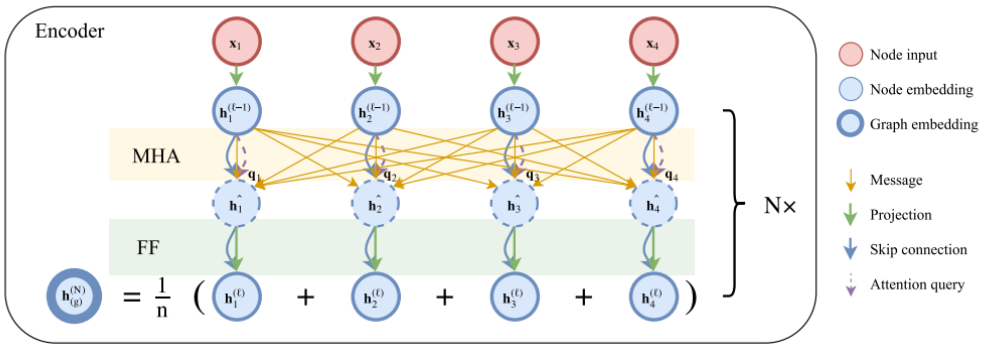
由于组合优化问题样本的稀少和难生成等特性，强化学习训练模型成为了这一系列问题的必要手段，最早由 Bello 等人于 2017 年提出[4]。基于 Pointer Network 模型的网络通常使用策略学习进行强化学习。其 Baseline 从最开始的引入 Critic 网络到 AM 模型提出的 Rollout 随着时间不断进

2.2 AM 模型简介和特点

2.2.1 AM 模型架构简介

① Encoder

如前文提到的那样，AM 模型的基本结构为采用了 Multi-Head Attention 技术的 Pointer Network，如图表 4 所示，其 Encoder 架构与前文提到的 Transformer 的 Encoder 部分结构几乎一致，都为 Multi-Head Attention 层和 Feed Forward 层组成的块的堆叠。



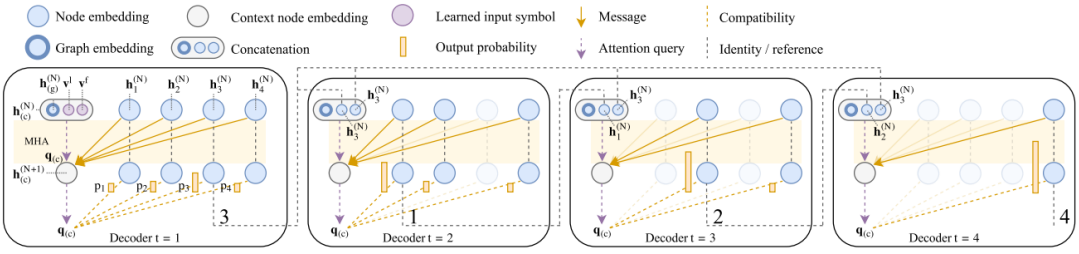
图表 4 AM 模型 Encoder 架构

但与其略有不同的是

1. 由于 TSP 问题中节点在输入序列中的 Position 对于结果没有影响，因此 AM 模型没有采用 Transformer 的 Position Encoding 技术。
2. AM 模型在保留了各个输入节点经过 Encoder 得到的 Embeddings 外号保留了其平均值，一起输入到 Decoder 中。

② Decoder

Decoder 架构与 Transformer 中略有不同，只保留了单层 Attention 层用来计算。



图表 5 AM 模型 Decoder 架构和工作示意图

从图表 5 可以看到 Decoder 中维护了一个独特的节点 Context node(c)，其由 Encoder 得到的，前一时刻选择的点，和出发地组成。在每一次计算中，将 c 作为 query， 作为 key - value，通过 Attention 机制计算 query 与 key 之间的相似度作为节点选择概率，从中选择概率最高的节点作为当前时刻的输出。后使用选出的更新 c 的值，用更新的 c 下一时刻进行下一时刻节点的选择。这里使用 mask 的形式确保选出的节点满足问题规则。

③ Baseline

Baseline 的作用是估计一个样本 s 的难度，从而用来进行强化学习训练。与之前使用 Critic 模型估计 s 的难度不同，这里它使用了过往模型中在测试集表现最好的模型作为 Baseline。因为作者认为训练一个模型去估计样本的难度是复杂的，我们应该认为样本的难度是由一个应用在其身上的算法的表现决定的，因此以自身模型的最好样本为 Baseline 是合理的。

2.2.2 AM 模型创新

① 在 Decoder 端进行了创新，抛弃了简单使用 seq2seq 模型的自回归计算采用了一种观察起始点，终止点和全局信息的 Context Node 作为 Query。

② 设计了一系列的 Mask 机制，使得模型可以通过使用不同的 Mask 方法应用到不同的问题上，如 TSP, VRP 等。

③ 改进了 Baseline 的选取方法，创新性的提出了 Rollout Baseline，显著改进了强化学习的效果，并被后续模型采用。

3 论文实验复现和总结

由于本学期创新实践着眼于 TSP 问题，因此在论文实验复现部分，我对于文章中提到的有关 TSP 的相关实验进行了研究与复现，并从中得到一些 AM 模型性能的总结。

3.1 TSP 相关实验复现结果

本实验中论文使用的设备和我使用的设备如表格 1。

设备	本实验	论文实验
CPU	I7-8700H	
GPU	GTX1060(90w 6G)	GTX1080Ti

表格 1 本实验与论文实验所用设备

3.1.1 复现 Table1 即 AM 求解性能实验

① 概述

论文中 Table 1(图表 6)为作者主要的实验数据，其详细对比了 TSP, CVRP, OP 等不同问题下自身模型和其他模型的表现(时间尺度，结果尺度)。虽然作者没有指出表格数据所用的实验问题，但是通过附录和实验我认为 Table 1 中的实验数据是作者使用其代码中提供的生成数据的方法，使用随机种子 1234 进行数据生成并实验的。

Table 1: Attention Model (AM) vs baselines. The gap % is w.r.t. the best value across all methods.

	Method	$n = 20$ Obj.	$n = 20$ Gap	$n = 20$ Time	$n = 50$ Obj.	$n = 50$ Gap	$n = 50$ Time	$n = 100$ Obj.	$n = 100$ Gap	$n = 100$ Time
TSP	Concorde	3.84	0.00%	(1m)	5.70	0.00%	(2m)	7.76	0.00%	(3m)
	LKH3	3.84	0.00%	(18s)	5.70	0.00%	(5m)	7.76	0.00%	(21m)
	Gurobi	3.84	0.00%	(7s)	5.70	0.00%	(2m)	7.76	0.00%	(17m)
	Gurobi (1s)	3.84	0.00%	(8s)	5.70	0.00%	(2m)		-	
	Nearest Insertion	4.33	12.91%	(1s)	6.78	19.03%	(2s)	9.46	21.82%	(6s)
	Random Insertion	4.00	4.36%	(0s)	6.13	7.65%	(1s)	8.52	9.69%	(3s)
	Farthest Insertion	3.93	2.36%	(1s)	6.01	5.53%	(2s)	8.35	7.59%	(7s)
	Nearest Neighbor	4.50	17.23%	(0s)	7.00	22.94%	(0s)	9.68	24.73%	(0s)
	Vinyals et al. (gr.)	3.88	1.15%		7.66	34.48%			-	
	Bello et al. (gr.)	3.89	1.42%		5.95	4.46%		8.30	6.90%	
	Dai et al.	3.89	1.42%		5.99	5.16%		8.31	7.03%	
	Nowak et al.	3.93	2.46%			-			-	
	EAN (greedy)	3.86	0.66%	(2m)	5.92	3.98%	(5m)	8.42	8.41%	(8m)
	AM (greedy)	**3.85**	**0.34%**	(0s)	**5.80**	**1.76%**	(2s)	**8.12**	**4.53%**	(6s)
	OR Tools	3.85	0.37%		5.80	1.83%		7.99	2.90%	
	Chr.f. + 2OPT	3.85	0.37%		5.79	1.65%			-	
	Bello et al. (s.)		-		5.75	0.95%		8.00	3.03%	
	EAN (gr. + 2OPT)	3.85	0.42%	(4m)	5.85	2.77%	(26m)	8.17	5.21%	(3h)
	EAN (sampling)	3.84	0.11%	(5m)	5.77	1.28%	(17m)	8.75	12.70%	(56m)
	EAN (s. + 2OPT)	3.84	0.09%	(6m)	5.75	1.00%	(32m)	8.12	4.64%	(5h)
	AM (sampling)	**3.84**	**0.08%**	(5m)	**5.73**	**0.52%**	(24m)	**7.94**	**2.26%**	(1h)

图表 6 文中 Table 1。作者使用 AM 对比传统模型(Baseline)。其中 Obj 为模型最优解，Gap 为模型最优解和历史最优解的差距(百分率)

② 自身实验

我使用了文中提供的预训练模型和自身训练的模型对图表 6 进行了复现。受限于设备原因，我只复现了表格的部分。

	n=20			n=50			n=100		
Method	Obj	Gap	Time	Obj	Gap	Time	Obj	Gap	Time
Concorde	3.84	0%	6m						
LKH3	3.84	0%	2m						
AM(greedy)	3.84	0%	1s	5.79	1.57%	3s	8.10	4.38%	8s
AM(sample)	3.84	0%	14m						
AM(self,greedy TSP20)	3.84	0%	1s	5.97	4.73%	3s	8.97	15.59%	8s

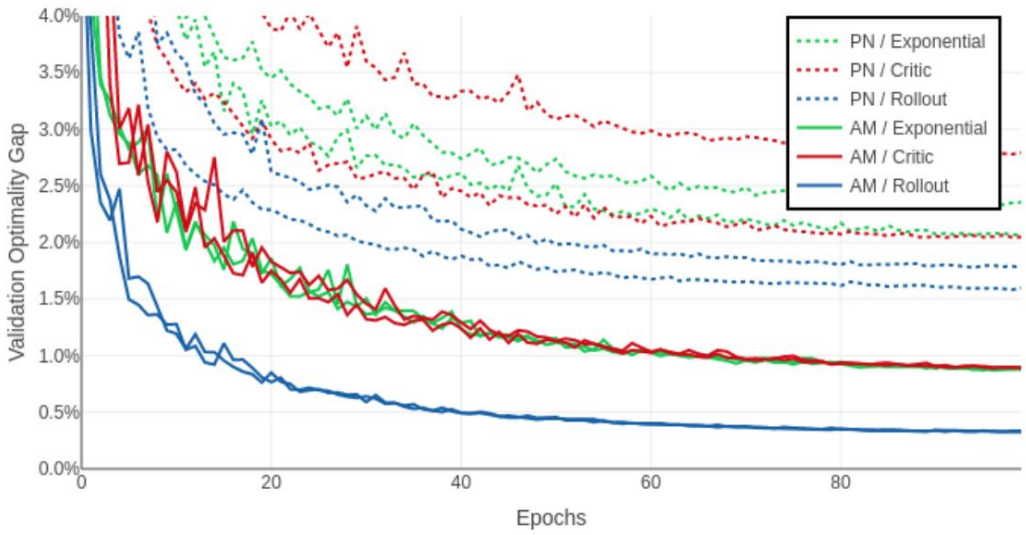
表格 2 对于文中 Table 1 的部分复现

从表格 2 中可以看到，文章中提到的表格中记录的求解性能表现均与测试结果一致，在 20，50，100 不同大小的数据集上，分别取得了 3.84,5.79,8.10 的结果，与论文中一致甚至要优于论文。运行时间由于实验设备差距的原因无法比较，但是 Neural Solver 的求解所需时间远远低于传统启发式求解器。

3.1.2 对比不同 Baseline 和 AM 与 PN 的区别的折线图 Figure 3

① 概述

在图表 7 中，论文将自身改进的模型 AM 和其原型 Pointer Network(PN) 结合不同的 Baseline 方法进行比较，展示了其提出的 Rollout Baseline 的优越性。



图表 7 不同 Baseline 与模型的结合的训练效果

② 自身实验

训练模型需要 GPU 支持，我训练一个 AM TSP20 模型花费 8h，没有获得设备支持前无法进行该实验。

3.1.3 对比不同超参数选择下模型性能的表格 Table 2(图)

① 概述

如图表 8，论文对比了不同学习率，解码层层数，Baseline 下的推断结果和

训练所需时间，作为模型超参数选择的指导。

Table 2: Epoch durations and results and with different seeds and learning rate schedules for TSP.

	epoch time	$\eta = 10^{-4}$ seed = 1234	$\eta = 10^{-4}$ seed = 1235	$\eta = 10^{-3} \times 0.96^{\text{epoch}}$ seed = 1234	$\eta = 10^{-3} \times 0.96^{\text{epoch}}$ seed = 1235
TSP20	5:30	3.85 (0.34%)	3.85 (0.29%)	3.85 (0.33%)	3.85 (0.32%)
TSP50	16:20	5.80 (1.76%)	5.79 (1.66%)	5.81 (2.02%)	5.81 (2.00%)
TSP100 (2GPUs)	27:30	8.12 (4.53%)	8.10 (4.34%)	-	-
N = 0	3:10	4.24 (10.50%)	4.26 (10.95%)	4.25 (10.79%)	4.24 (10.55%)
N = 1	3:50	3.87 (0.97%)	3.87 (1.01%)	3.87 (0.90%)	3.87 (0.89%)
N = 2	5:00	3.85 (0.40%)	3.85 (0.44%)	3.85 (0.38%)	3.85 (0.39%)
N = 3	5:30	3.85 (0.34%)	3.85 (0.29%)	3.85 (0.33%)	3.85 (0.32%)
N = 5	7:00	3.85 (0.25%)	3.85 (0.28%)	3.85 (0.30%)	10.43 (171.82%)
N = 8	10:10	3.85 (0.28%)	3.85 (0.33%)	10.43 (171.82%)	10.43 (171.82%)
AM / Exponential	4:20	3.87 (0.95%)	3.87 (0.93%)	3.87 (0.90%)	3.87 (0.87%)
AM / Critic	6:10	3.87 (0.96%)	3.87 (0.97%)	3.87 (0.88%)	3.87 (0.88%)
AM / Rollout	5:30	3.85 (0.34%)	3.85 (0.29%)	3.85 (0.33%)	3.85 (0.32%)
PN / Exponential	5:10	3.95 (2.94%)	3.94 (2.80%)	3.92 (2.09%)	3.93 (2.37%)
PN / Critic	7:30	3.95 (3.00%)	3.95 (2.93%)	3.91 (2.01%)	3.94 (2.84%)
PN / Rollout	6:40	3.93 (2.46%)	3.93 (2.36%)	3.90 (1.63%)	3.90 (1.78%)

图表 8 文中 Table2 Epoch 持续时间和结果，使用不同的种子和学习速度的 TSP 问题。

② 自身实验

GPU 限制只训练了一次10^{-4}学习率下，Rollout Baseline， TSP20 的模型。

设备	Epoch Time	Learning Rate $\{10^{-4}\}$
	Seed = 1234	
TSP20(GTX1060)	7:30(batch size 512)	3.85(0.34%)
TSP100(GTX1060)	80:00(batch size 256)	

表格 3

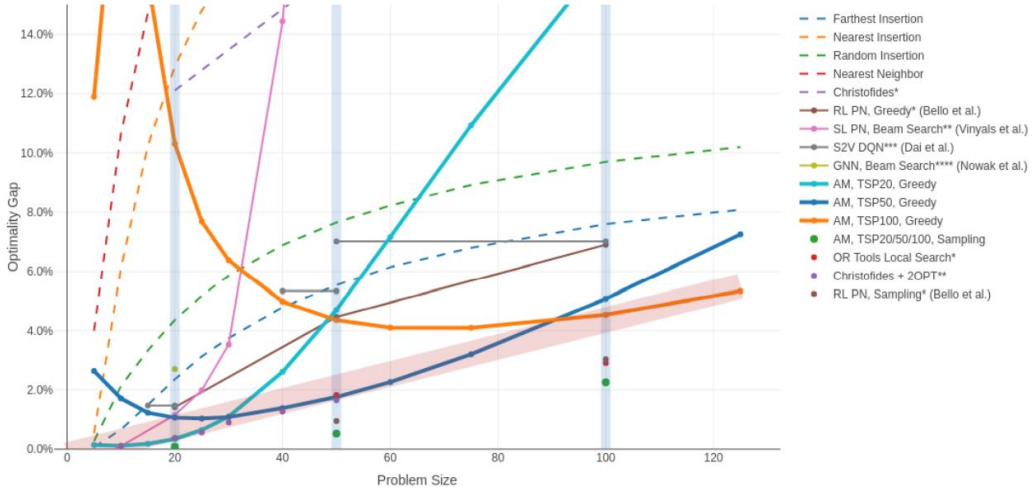
如表格 3，在 GTX1060 训练下，Batch size=512 时训练速度约为 Epoch Time = 7:30 左右，Batch size 小时速度明显下降，而其在显存允许范围内增大时训练速度无明显提升。准确率与论文中给出的一致，均为 3.85%。

尝试训练了 TSP 100 问题，但是 Batch size=512 显存溢出，Batch size=256 时，Epoch Time = 80：00 左右，由于时间成本没有训练完成。

3.1.4 比较不同模型的在不同大小问题下的泛化程度 Figure 5

① 概述

图表 9 中，作者对比了不同模型在不同问题大小上泛化程度。如将用 Graph size = 20 训练的模型去解决 size = 100 的问题，看其泛化性如何。作者测试了 Graph size = [5, 10, 15, 20, 25, 30, 40, 50, 60, 75, 100, 125]。

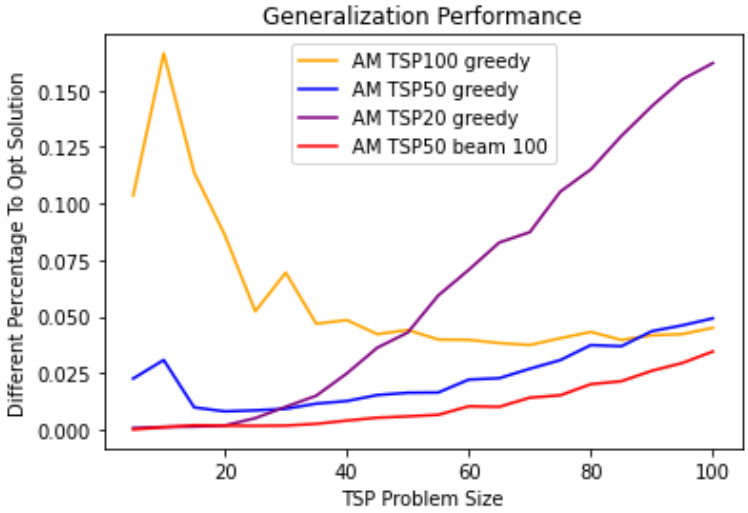


图表 9 不同模型在不同大小数据集上的泛化性能测试[1]

② 自身实验

本实验由于时间所限，只对图中与 AM 模型相关的三条曲线进行了相关实验。首先通过模型自带的数据生成函数生成数据 的对应数据集，其中每个数据集包含个测试样例。将数据转化成 TSPLIB 数据格式使用 Concorde 求解器求解出各个数据集的解，每个数据集取平均作为 Opt Solution。后分别使用模型预训练好的 TSP 20, 50, 100 模型去求解对应数据集，将得到的结果与 Opt Solution 进行比较，获得泛化性能曲线。

除此之外，我还对于 AM TSP50 模型进行了额外测试，使用 Beam search width = 100 的方式尝试获得更好的泛化性能。



图表 10 自身对论文中泛化性能测试的复现

在图表 10 中可以看到，三个使用不同 Graph size 训练的 AM 模型表现出来的泛化性能曲线与论文趋势保持一致。数值上也基本保持一致，如 AM TSP20 在 n=20 的问题集上与 Opt Solution 的差距都接近而在 n=50 的问题集上都接近。

而通过图片我们也可以看出，使用不同 Graph size 训练的模型拥有不同的泛化性能，其在解决自身训练对应大小的问题集上优于其他模型。而三个模型中，AM TSP50 表现出了最好的泛化性能，而在此基础上，使用 Beam Search 的 AM TSP50 模型表现出了更好的性能表现，其在全部数据集上的表现都优于其他模型，与 Baseline 的差距始终在 5%以内。这证明了使用 Beam Search 等牺牲时间的方式可以提升解的质量，获得接近于专业求解器的解。

3.2 论文实验复现总结

现阶段，我根据自身设备条件，结合传统启发式求解器 Concorde, LKH, 对论文所提及的实验进行了最大程度的复现。根据复现结果我们可以看出，现阶段 AM 模型为代表的神经网络求解器在随机生成的数据集上。当 Graph size 较小时，求解质量与传统专业求解器相差无几，然是在 Graph size 较大时，和传统专业求解器的求解质量仍有差距，但差距已经在逐步缩小。而在求解时间上，神经网络求解器相较于传统方法，有着数量级级别的优势。

而通过对论文中提及的有关泛化性能的测试，我们了解到 AM 模型在不同 Graph size 的数据集上表现出了较好的泛化性能，其中表现最好的 AM TSP50 模型与最优解的差距始终不超过 5%。而如果我们使用 Beam Search 等方式，可以进一步用时间损失的方式提升求解质量。这一特性也让神经网络求解器的应用场景更加广泛。

4 专业求解器与 TSPLIB 相关

4.1 专业求解器

本实验现阶段用到的专业求解器主要为 LKH 和 Concorde，这些求解器都编写于 21 世纪初，为传统的组合优化问题求解器。本实验使用的求解器下载连接请见[5],[6].

4.2 TSPLIB

TSPLIB 是来自各种来源和各种类型的 TSP(和相关问题)的示例实例库，其规范了 TSP 问题的相关格式，并提供了经典的 TSP 问题数据集和历史最优解，而本实验使用的实验数据均遵循其规定中的 EUC_2D 标准，即点之间的距离为欧氏距离。由于接下来实验的需要，我们需要使用来自 TSPLIB 数据库的数据对模型进行实验，但因为神经网络模型的输入一般为归一化数据，因此我们也要对来 TSPLIB 数据集的数据进行归一化操作。

4.2.1 AM 模型运行 TSPLIB 问题

为了便于比较专业求解器和神经网络求解器的结果，我没有对 TSPLIB 数据集中的数据用最大值进行归一化，而是采用大于最大值的值进行归一化。而由于专业求解器对于 EUC_2D 数据计算欧氏距离的时候采用四舍五入的模式，而神经网络求解器采用浮点数进行运算，为最大限度保留精度，我选择在专业求解器允许的情况下对数据进行放大，以减小四舍五入带来的误差。

① CH130.tsp 数据集实验

这里我对 ch130.tsp 数据集进行了实验，将数据集缩小 1000 倍输入 AM 模型，最终结果如下：

模型	Cost	Time	Opt
历史最优解	6.110		
Concorde	6.110	0.27s	0%
LKH	6.110	0.22s	0%
AM TSP 20 (Greedy)	7.30	0s	19.47%
AM TSP 50 (Greedy)	6.570	0s	7.52%
AM TSP 100 (Greedy)	6.501	0s	6.39%
AM TSP 100 (beam search)	6.285	1s	2.86%

表格 4 不同模型在 ch130 数据集上的表现

在表格 4 看到对于 ch130.tsp 数据集进行求解，AM 模型的求解质量与最优解仍有一定差距，但是在运行时间上仍有优势。在使用 Beam Search 提升解质量后，差距缩小，但是时间可能超过专业求解器耗时。

4.2.2 专业求解器运行生成数据集

自生成的数据为0-1均匀分布的浮点数集，由于专业求解器的四舍五入模式，使得需要对数据进行放大。在专业求解器允许的范围内，为了方便起见，前文中的数据都被放大了1000倍后交由专业求解器求解。

5 现阶段总结与规划

5.1 现阶段总结

现阶段，我们对于神经网络求解器求解组合优化问题的背景知识进行了一定程度的了解，从Neural Solver的优秀模型AM出发，完成了对其泛化性能和求解性能的基本测试，完成了数据集的相互转换，实现自生成数据集在Concorde，LKH等专业求解器的成功运行，为下一步实验做出准备

5.2 下一阶段安排

下一阶段，我们预计将生成更多测试数据，使用传统TSP求解器求出其标准解作为Baseline，用于测试TSP求解器。同时我们将着手解决由于传统TSP求解器中求解EUC_2D中四舍五入带来的精度损失。最后，我们希望引入更多的TSP求解器，用生成的数据集对其性能进行测试。

6 引用

[1] Kool W, Van Hoof H, Welling M. **Attention, learn to solve routing problems!** In: *7th International Conference on Learning Representations, ICLR, 2019*

[2] Vaswani A, Shazeer N, Parmar N, et al. **Attention is all you need.** In: *Advances in Neural Information Processing Systems, NIPS, 2017.*

[3] Vinyals O, Fortunato M, Jaitly N. **Pointer networks.** In: *Advances in Neural Information Processing Systems, NIPS, 2015*

[4] Bello I, Pham H, Le Q V., Norouzi M, Bengio S. **Neural combinatorial optimization with reinforcement learning.** In: *5th International Conference on Learning Representations, ICLR 2017*

[5] **Concorde Home.** (2022). Retrieved 24 March 2022, from *https://www.math.uwaterloo.ca/tsp/concorde.html*

[6] **LKH-3** (Keld Helsgaun). (2022). Retrieved 24 March 2022, from

http://webhotel4.ruc.dk/~keld/research/LKH-3/