南開大學
Nankai University

计算机学院

并行程序设计开题报告

大规模 TSP 问题并行求解研究

姓名：窦楷然

学号：2210652

专业：计算机科学与技术

2024 年 4 月 1 日

摘 要

旅行推销员问题（TSP）是一个经典的 NP 问题，它要求寻找一条访问每个给定城市恰好一次并返回起始城市的最短路径。尽管问题的陈述简单，随着城市数量的增加，寻找最优解变得极其复杂。本文探讨了 TSP 问题的不同解决方案，包括传统的精确算法和启发式算法，以及近年来随着人工智能技术发展而出现的深度学习算法。我关注于一种新的解决方案——Deep Reinforced Multi-Pointer Transformer（Pointerformer），该方法结合了深度强化学习和 Transformer 架构，通过模拟自然界现象和优化数据处理策略，展现出在大规模 TSP 实例上的优势。此外，本文讨论了并行计算技术在加速 TSP 问题解决过程中的应用，并提出了基于 Pointerformer 算法的并行实现计划，以期进一步提高求解效率和算法性能。

关键词：旅行推销员问题（TSP），启发式算法，深度学习，Transformer，并行计算

目录

1 研究课题

1.1 课题介绍

TSP 问题的提出非常简单：给定一个城市列表及每对城市之间的距离，旅行推销员需要访问每个城市恰好一次并最终返回起始城市，问题的目标是找到总旅行距离最短的路线。尽管问题陈述简单，但随着城市数量的增加，可能的路线组合数量增长的速度超出了直观的预期，使得找到最优解变得极其复杂。这种复杂性的本质反映了 TSP 的 NP 难问题属性，意味着没有已知的多项式时间算法能解决所有 TSP 实例。[2]

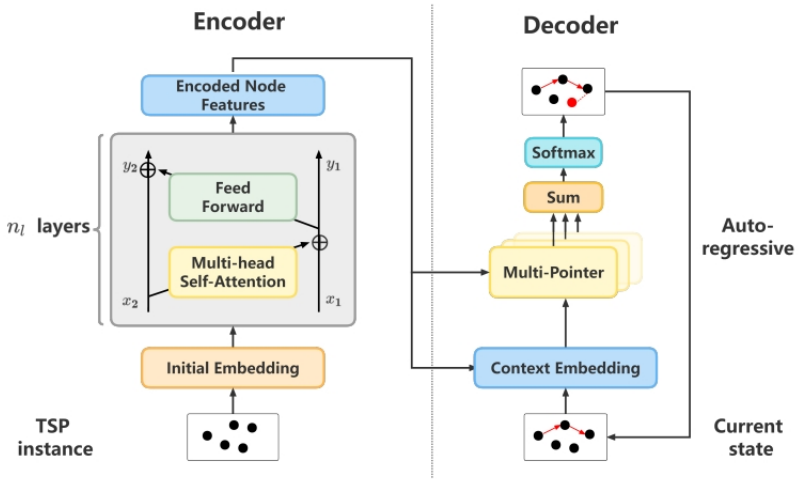

图 1.1: TSP 示意图

TSP 问题不仅仅是理论上的数学挑战，它在实际中也有着广泛的应用。例如，在制造业中，钻孔问题（如打印电路板的钻孔）可以转化为 TSP 问题，以最小化钻头移动的总距离。在物流和配送行业，如何高效地规划配送路径以最小化成本和时间，本质上也是 TSP 问题。此外，TSP 的应用还扩展到了 X 射线晶体学、仓库订单拣选、运输路线优化等领域，展现了其在解决现实世界问题中的重要价值。

随着对 TSP 的研究深入，学者们提出了多种解决方案。早期的方法主要集中在精确算法，如分枝定界、动态规划等，这些算法在小规模实例上能够找到最优解，但是当面对实际应用中常见的大规模问题时，计算时间迅速变得不可接受。因此，研究重心逐渐转移到启发式算法上，这类算法不能保证找到最优解，但在合理的时间内能够提供良好的近似解。典型的启发式算法包括遗传算法、模拟退火、蚁群算法等，它们通过模拟自然界的现象或过程来寻找解决方案，展现了解决复杂优化问题的新思路。

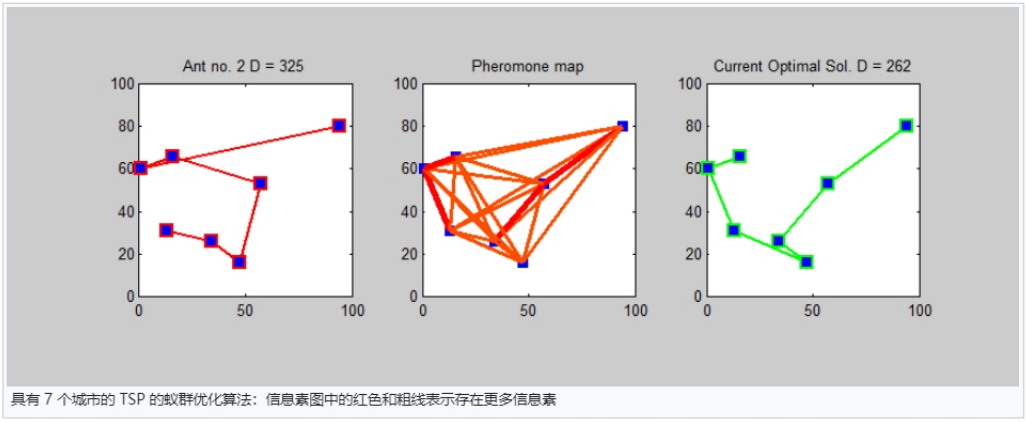

图 1.2: TSP 在城市规划中的应用

近年来，随着人工智能技术的飞速发展，深度学习（Deep Learning, DL）算法被引入到 TSP 问题的解决中。依托强大的数据处理能力和自我学习能力，DL 算法在解决 TSP 问题上展示出了前所未有的潜力。特别是深度强化学习（Deep Reinforcement Learning, DRL）和端到端的深度学习模型，为处理大规模 TSP 实例提供了新的解决方案。这些方法通过学习策略来优化路径选择过程，而不是直接搜索所有可能的路径组合，从而在较大规模的问题实例上实现了比传统方法更高效的解决方案。[1]

尽管如此，当前 DL 算法在解决 TSP 问题上仍面临挑战。例如，模型的训练和推理效率，以及如何提高模型在不同类型 TSP 实例上的泛化能力仍是研究的热点。此外，随着问题规模的增加，如何有效管理计算资源，特别是内存资源，成为了亟需解决的问题。

对于这个问题，经过一系列的资料查询，我拟采用 **Deep Reinforced Multi-Pointer Transformer** 的方法的并行模式去实现优化。这种新方法使用深度强化学习 (DRL) 来解决 TSP，旨在高效处理大规模问题。Pointerformer 在编码器中采用可逆残差网络，在解码器中采用多指针网络，有效遏制了内存消耗；它还利用特征增强来探索 TSP 对称性，以及一种增强的上下文嵌入方法来获得全面的上下文信息。该方法在小规模和大规模 TSP 实例中都显示出了有希望的结果，优于许多最先进的 DRL 方法，特别是在对更大问题的泛化方面。

1.2 算法构成

为了实现这个经典问题的求解，构建了许多种经典串行算法，这些算法都能在一定程度上降低问题的时间复杂度，以下我将列举这几种算法。

1.2.1 遗传算法

- **算法实现原理**

  动态规划求解 TSP 问题的基本思想是将问题分解为子问题，并递归地解决这些子问题。具体地，设 d_{ij} 为城市 i 到城市 j 之间的距离，n 为城市的总数。定义 $dp[S][i]$ 为从城市 i 出发，经过集合 S 中的所有城市恰好一次，最后回到起点城市的最短路径长度，其中集合 S 包含了除起点城市外的部分或全部其他城市，且起点城市固定不变。

- **状态转移方程**

  动态规划的状态转移方程如下所示：

$$dp[S][i] = \min\{dp[S - \{i\}][j] + d_{ji}\}, \quad \forall j \in S, j \neq i$$

这里，$S - \{i\}$ 表示集合 S 去掉城市 i 后的城市集合，d_{ji} 是城市 j 到城市 i 的距离。状态转移方程的含义是，要想知道从城市 i 出发，经过集合 S 中的所有城市恰好一次并返回起点的最短路径长度，我可以先考虑从某个城市 j（j 是集合 S 中除了 i 外的任意一个城市）出发，经过集合 $S - \{i\}$ 中的所有城市恰好一次并返回起点的最短路径长度，然后再从城市 j 直接到城市 i 的距离。在所有这样的路径中，长度最短的那一条就是要找的路径。

1.2.2 模拟蚁群算法

模拟蚁群算法 (ACO) 是根据对真实蚂蚁能够在不使用视觉提示的情况下找到从食物源到巢穴的最短路径的观察而开发的。为了说明“真正的”蚁群如何搜索最短路径，我在网上查到一个示例以便更好地理解。可假设下图中 A 是食物源，E 是巢穴。蚂蚁的目标是将食物带回巢穴。显然，较短的路径与较长的路径相比具有优势。

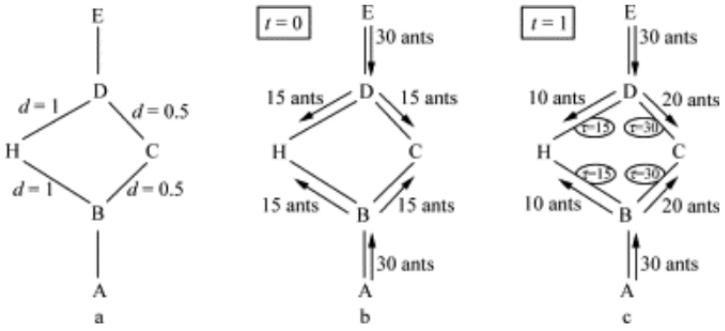

图 1.3: 蚁群算法示意图

假设在时间 t = 0 时，B 点有 30 只蚂蚁（D 点有 30 只蚂蚁）。此时，任何片段上都没有信息素踪迹，蚂蚁以相同的概率随机选择路径。因此，平均每个节点有 15 只蚂蚁将走向 H，15 只蚂蚁将走向 C。在 t = 1 时，从 A 到达 B 的 30 只新蚂蚁在通往 H 的路径上发现了一条强度为 15 的轨迹，该轨迹是由从 B 出发的 15 只蚂蚁铺设的，并且在该路径上有一条强度为 30 的轨迹到 C 的路径，是从 B 出发的 15 只蚂蚁和从 D 经 C 到达 B 的 15 只蚂蚁所铺设的路径之和。因此，选择路径的概率是有偏差的，因此前往 C 的蚂蚁的预期数量将是前往 H 的蚂蚁数量的两倍：分别为 20 和 10。对于来自 E 的 D 中新的 30 只蚂蚁也是如此。这个过程一直持续到所有蚂蚁最终都选择了最短路径。[6]

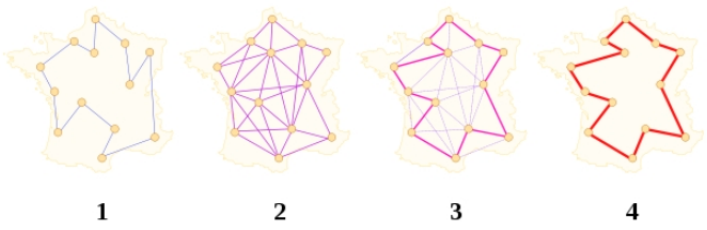

图 1.4: ACO 原理图

1.2.3 Pointerformer

通过阅读这篇论文，我发现了使用 Pointerformer 的一些基本思路。[4]

根据这篇文章，他提出了一种基于多指针变压器的可扩展 DRL 方法称为指针转换器，以端到端方式解决 TSP 问题。在遵循经典的编码器-解码器架构的同时，这种新方法采用了可逆残差网络而不是编码器中的标准残差网络，以显著减少内存消耗。其在解码器中提出了一个多指针网络，根据给定的查询顺序地生成下一个节点。除了解决内存消耗问题外，指针转换器还包含了精细的设计，以进一步提高模型的有效性。特别是，为了提高得到的解的有效性，指针转换器采用了特征增强的方法来探索 TSP 在训练和推理阶段的对称性，并采用了增强的上下文嵌入方法来在查询中包含更全面的上下文信息。

总旅行成本的计算公式为：

$$L(\tau) = cost(\tau[N], \tau[1]) + \sum_{i=1}^{N-1} cost(\tau[i], \tau[i+1]) \tag{1}$$

其中，$\tau[i]$ 表示路径上的第 i 个节点，$N = |\tau|$ 是路径的长度，$cost(i, j)$ 是从节点 i 移动到节点 j 的成本。

目标函数定义为：

$$J(\theta|s) = E_{\tau \sim p_\theta(\tau|s)}[R(\tau)] \tag{2}$$

其中，$R(\tau) = -L(\tau)$，即每一步的奖励定义为新增边的负成本。

2 可行性分析

2.1 相较于其他算法的优势

- 面对大规模实例的可扩展性和效率

  Pointerformer 在面对大规模实例展现出卓越的可扩展性，能高效解决多达 500 个节点的 TSP 实例，这得益于其在编码器中使用的可逆残差网络。同时这个算法可以显著降低了内存消耗，因为其通过可逆残差网络减少了训练和推理阶段的内存消耗，使其能更高效地处理较大实例。

  相较于 Pointerformer，ACO 和 DP 在随着问题规模的增加，ACO 的性能会降低，因为计算时间和内存需求指数级增长，同时需要为所有路径维护信息素，这在大问题中会显著增加内存使用。DP 由于时间复杂度指数级增长，在大规模 TSP 实例中实际不可行，其自身的内存需求随城市数量指数增长。

- 泛化能力和灵活性

  Pointerformer 表现出强大的泛化能力，对看过和未看过的 TSP 分布都表现良好，无需重新训练，其 DRL 方法学习了一个在不同问题实例中泛化的策略。

  相比之下，ACO 的泛化是个挑战，它高度依赖于具体问题实例，需要为每个新实例调整参数。DP 不泛化，它从头解决每个实例，不能从之前解决的实例中学习。

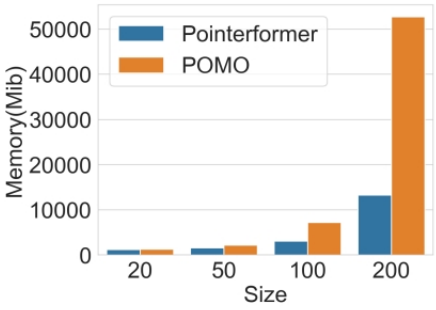

图 2.5: 相较于其他算法的优势

2.2 并行程序的优势

将 Pointerformer 算法从串行实现转变为并行程序设计，尤其是在处理大规模旅行商问题时，能够带来显著的优势。[5] 与上面 ACO 和 DP 的对比时类似，我也发现了串行实现转变为并行程序设计的时候的三个类似优势如下：

- 提高处理效率

  利用并行处理和特别设计的数据结构（如 Pointerformer 中的可逆残差网络），可以有效降低在处理大规模问题时的内存消耗。这允许算法在相同的硬件资源下解决更大的问题实例，或者使用较少的资源解决相同大小的问题，使其可以同时处理多个计算任务，显著减少求解大规模 TSP 实例所需的总时间，通过在多核 CPU 或 GPU 上并行执行 Pointerformer 的不同部分，我可以尝试结合 SIMD、Pthread/OpenMP、MPI、GPU，去更快地处理更多的数据。

- 增强模型的泛化能力

  通过并行处理，Pointerformer 可以在多个处理单元上同时训练和测试，这不仅提高了训练的效率，还有助于模型在各种不同规模和类型的 TSP 问题上获得更好的泛化能力。并行处理使得在相同时间内能够在更多的数据集上进行训练，从而提高了模型的鲁棒性和适用范围。

3 创新点

并行程序的优势在于计算速度的提升，以下是我在网上整理的创新点：

在旅行商问题的研究领域，并行计算已经成为加速求解过程的关键技术之一。最近的研究表明，通过使用并行计算框架，如消息传递接口（MPI）和开放多处理（OpenMP），可以显著提高解决 TSP 问题的效率和速度。例如，一个针对 TSP 问题的 MPI 并行化方法将城市的排列等分到多个并行环境（PEs）中，每个 PE 计算其分配的排列的最优成本和路径，然后通过主 PE 进行同步和比较以找到全局最优解。这种方法显示出，随着 PE 数量的增加，执行时间减少，加速比提高，尤其是在处理大规模 TSP 问题时表现更为明显。[3]

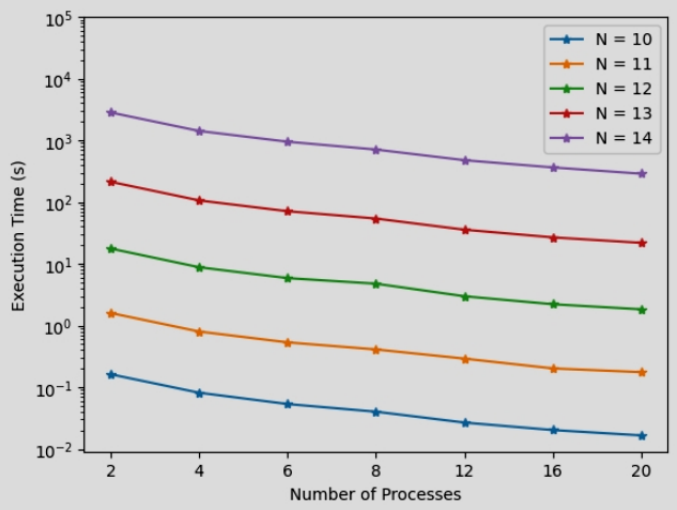

图 3.6: 执行时间随 MPI 进程数的变化

我进一步的分析还比较了 MPI 和 OpenMP 在 TSP 问题上的表现。其论文中结果表明，MPI 实现的效率普遍高于 OpenMP 实现，这可能是由于 OpenMP 线程的额外开销大于 MPI 中的进程间通信开销。此外，通过计算 Karp-Flatt 指标（一个衡量并行计算中串行分数的指标），研究进一步证明了 MPI 方法在并行计算中的高效性。[7]

4 课题研究计划

针对 Pointerformer 算法的并行实现计划，我觉得 Pointerformer 算法的关键在于其基于 Transformer 的架构，用于解决旅行商问题，在训练过程中利用深度强化学习来优化策略。

首先我会花时间了解 Pointerformer 算法修改成并行模式的主要原理。包括在编码器中使用可逆残差网络代替标准残差网络显著减少了内存消耗。我查阅资料后发现这一修改对于扩展到大型 TSP 实例至关重要 [4]，因为它在反向传播过程中减少了内存占用，使模型能够有效处理更大的输入序列（TSP 实例）。引入解码器中的多指针网络旨在通过给定查询顺序生成下一个节点，以进一步最小化编解码器架构的内存消耗，通过采用单头注意机制，它有效地控制了内存使用，同时保持或提高了模型在 TSP 解决方案上的性能。

然后我会想办法实现这些。我想利用 SIMD 优化与向量化处理，利用 SIMD（单指令多数据）指令集对 Pointerformer 的关键操作（如注意力机制）进行向量化处理，以提高数据处理的效率和降低内存访问开销。同时根据 Pointerformer 在不同阶段的计算特性，采用数据并行与模型并行策略，使用 MPI 实现不同计算节点间的数据分发与收集，特别是在处理大规模 TSP 问题时，将数据集分割并在多个节点上并行处理。然后利用 GPU 的并行计算能力，将 Pointerformer 的 Tensor 操作优化为适合 GPU 的操作，利用 CUDA 或其他 GPU 编程框架，对整个算法流程进行加速，重点关注矩阵计算和反向传播等部分。

在实现的过程中，我会不断查阅资料，在现有的框架上进行优化，也逐步修改现在制定计划的一些小错误或者不合理的地方。如果学有余力，我将探寻更多的优化方式。

参考文献

[1] Abelardo Araujo, Charles RM Bangham, Jorge Casseb, Eduardo Gotuzzo, Steve Jacobson, Fabiola Martin, Augusto Penalva de Oliveira, Marzia Puccioni-Sohler, Graham P Taylor, and Yoshihisa Yamano. Management of ham/tsp: systematic review and consensus-based recommendations 2019. *Neurology: Clinical Practice*, 11(1):49–56, 2021.

[2] Zhang-Hua Fu, Kai-Bin Qiu, and Hongyuan Zha. Generalize a small pre-trained model to arbitrarily large tsp instances. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 7474–7482, 2021.

[3] Amey Gohil, Manan Tayal, Tezan Sahu, and Vyankatesh Sawalpurkar. Travelling salesman problem: Parallel implementations & analysis. *arXiv preprint arXiv:2205.14352*, 2022.

[4] Yan Jin, Yuandong Ding, Xuanhao Pan, Kun He, Li Zhao, Tao Qin, Lei Song, and Jiang Bian. Pointerformer: Deep reinforced multi-pointer transformer for the traveling salesman problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 8132–8140, 2023.

[5] Hua Yang, Minghao Zhao, Lei Yuan, Yang Yu, Zhenhua Li, and Ming Gu. Memory-efficient transformer-based network model for traveling salesman problem. *Neural Networks*, 161:589–597, 2023.

[6] Jinhui Yang, Xiaohu Shi, Maurizio Marchese, and Yanchun Liang. An ant colony optimization method for generalized tsp problem. *Progress in Natural Science*, 18(11):1417–1422, 2008.

[7] Pramod Yelmewad and Basavaraj Talawar. Parallel iterative hill climbing algorithm to solve tsp on gpu. *Concurrency and Computation: Practice and Experience*, 31(7):e4974, 2019.