改进大规模图组合优化的探索性强化学习方法：引入重要性评分网络

摘要

本文针对大规模图上的组合优化问题，提出了一种基于探索性组合优化（ECO-DQN）框架的新方法。通过在现有开源代码基础上添加一个用于计算节点重要性的三层MLP辅助网络，并结合softmax层为边赋予重要性分数，我们构建了一个重要性邻接矩阵A_{att}。该矩阵与原始邻接矩阵加权相加后作为输入传递给消息传递神经网络（MPNN）。实验结果表明，在求解超过500个节点的大规模图上的最大割问题时，此方法所求解的质量高于ECO-DQN方法和贪婪算法，并拥有更短的训练时间。

关键词：组合优化；深度强化学习；消息传递神经网络

1 引言

组合优化问题是计算机科学中的一类经典难题，尤其当问题规模扩大到数千乃至更多节点时，传统求解方法往往难以满足实际需求。近年来，随着深度学习技术的发展，特别是强化学习的应用，人们开始探索如何利用这些先进的算法来解决复杂的组合优化问题。然而，在处理大规模图结构的数据时，现有的图神经网络（如MPNN）面临着消息传递效率低下以及训练收敛速度慢等问题。

本研究旨在通过引入一种新的辅助网络机制——重要性评分网络，来改善MPNN在大规模图上的表现。该网络通过节点特征提取其重要性分数，来对MPNN中的邻接矩阵进行小幅修改，最终形成的加权邻接矩阵不仅保留了原始图的信息，还融入了节点间关系的重要程度，从而加速了信息传播过程并提高了模型的学习效率。

2 相关工作

组合优化问题（Combinatorial Optimization, CO）因其广泛的应用背景，如物流配送、资源分配和网络设计等，在计算机科学与运筹学领域占据重要地位。随着问题规模的增长，传统的精确算法难以在合理时间内找到最优解，因此启发式算法和近似算法逐渐成为研究热点。近年来，深度学习特别是强化学习（Reinforcement Learning, RL）技术的发展为解决这类问题提供了新的思路。

2.1 强化学习在组合优化中的应用

早期尝试将神经网络应用于组合优化的研究可以追溯到Hopfield和Tank的工作 [5]，他们使用Hopfield网络解决旅行商问题（TSP）。Zhang 和 Dietterich [9]首次将RL技术应用于组合优化，具体是针对NP-hard级别的作业车间调度问题，而TSP只是其特例。

近年来，Bello等人 [3]利用策略梯度方法训练指针网络（Pointer Networks），该网络通过softmax注意力机制选择输入序列中的元素作为输出，适用于图结构问题。Khalil等人 [6]提出了S2V-DQN框架，它结合了图嵌入网络和深度Q网络（Deep Q-Network, DQN），以自动学习不同组合优化问题的良好启发式策略。Mittal等人 [8]进一步改进了训练过程，首先训练一个图卷积网络（Graph Convolutional Network, GCN）进行图嵌入，然后训练Q网络来预测动作价值。这些方法展示了如何有效地利用深度学习模型处理图结构数据，并且在多种组合优化任务上取得了较好的结果。Barrett等人 [2]提出了ECO-DQN框架，允许代理在测试时不断改进解，通过添加或移除顶点来寻找更优的解。

2.2 图神经网络及其变体

图神经网络（Graph Neural Networks, GNNs）是一类专门设计用于处理非欧几里得结构数据（如图）的深度学习模型。Gilmer等人 [4]提出的神经消息传递框架（Neural Message Passing Framework）为GNN的具体实现提供了一个通用模板，许多常见的图网络都是其特定实例。例如，消息传递神经网络（Message Passing Neural Networks, MPNNs）通过迭代更新节点表示来捕捉局部邻域信息，这使得它们特别适合于解决基于图的组合优化问题。

Li等人 [7]结合GCN与指导树搜索（Guided Tree Search）提出了一种监督设置下的组合优化方法，虽然这种方法需要大量预解实例来进行训练，但它展示了如何将图网络与传统搜索方法相结合的可能性。Abe等人 [1]则借鉴AlphaGo Zero的思想，使用蒙特卡洛树搜索（Monte Carlo Tree Search, MCTS）作为策略改进操作，训练GCN来解决组合优化问题。尽管他们的工作没有涉及最大割问题，但这种结合搜索与学习的方法为未来的研究提供了重要见解。

3 本文方法

3.1 本文方法概述

为了解决大规模图组合优化问题中消息传递效率低下的问题，本文提出了一种改进的消息传递神经网络（MPNN）架构。该架构引入了一个重要性评分辅助网络，旨在根据节点特征动态评估每个节点的重要性，并通过加权机制增强信息传播的效率。具体来说，我们设计了一个三层多层感知器（MLP）作为重要性评分模块，用于计算节点的重要性分数，并通过带温度参数的softmax层实现软注意力机制。最终，我们将边的重要性分数与原始邻接矩阵加权相加，形成一个新的加权邻接矩阵，作为输入传递给MPNN。

3.2 重要性评分模块

重要性评分模块计算每个节点 v 的重要性分数 p_v 如下：

$$\mathbf{h}_v = \mathrm{ReLU}(\mathbf{W}_1\mathbf{x}_v + \mathbf{b}_1) \tag{1}$$

$$s_v = \mathbf{W}_2 \mathbf{h}_v + b_2 \tag{2}$$

$$p_v = \frac{\exp(s_v/\tau)}{\sum_{u \in V} \exp(s_u/\tau)} \tag{3}$$

边的重要性分数通过相乘所连节点的重要性分数计算得出：

$$a_{uv} = p_u \cdot p_v \tag{4}$$

重要性邻接矩阵 A_{att} 将由归一化得到：

$$(A_{att})_{uv} = a_{uv} / \sum_{u,v} a_{uv} \tag{5}$$

并与原始邻接矩阵 A 加权结合形成最终的输入矩阵 A'：

$$A' = A + N\sigma A_{att} \tag{6}$$

其中τ和σ为可调的超参数，加权时乘上节点个数N是由于A与A_{att}的归一化方式不同。

3.3 消息传递过程

初始嵌入由节点特征向量通过初始嵌入层获得：

$$\mu_v^0 = \text{ReLU}(\theta_1 \mathbf{x}_v) \tag{7}$$

节点嵌入通过消息传递进行更新：

$$m_v^{k+1} = \text{ReLU}\left(\theta_4 \left[\frac{1}{|N'(v)|} \sum_{u \in N'(v)} \mu_u^k, \xi_v\right]\right) \tag{8}$$

$$\mu_v^{k+1} = \text{ReLU}\left(\theta_5 \left[\mu_v^k, m_v^{k+1}\right]\right) \tag{9}$$

其中 ξ_v 描述了连接到节点 v 的边：

$$\xi_v = \text{ReLU}\left(\theta_3 \left[\frac{1}{|N'(v)|} \sum_{u \in N'(v)} \text{ReLU}(\theta_2 [w_{uv}, \mathbf{x}_u]), |N'(v)|\right]\right) \tag{10}$$

最后，Q值从最终的节点嵌入中读出：

$$Q_v = \theta_7 \left[\text{ReLU}\left(\theta_6 \frac{1}{|V|} \sum_{u \in V} \mu_u^K\right), \mu_v^K\right] \tag{11}$$

在以上过程中，$N'(v)$将由在重要性评分模块得到的输入矩阵 A'所得到而非原邻接矩阵A。

4 复现细节

4.1 现有开源代码

本文所引用的开源代码来自开源项目eco-dqn（https://github.com/tomdbar/eco-dqn/）,该项目由原论文作者Thomas D. Barrett提供 [2]。

4.2 本文改进

在代码层面，本文的改进体现在对原项目的MPNN类中进行如下改动：

```
importance_scores = self.importance_scoring_network(torch.cat([
    node_features,adj], dim=-1)).squeeze(-1)
sparse_importance_scores =F.softmax(importance_scores/self.tau, dim=-1)
attention_weights = sparse_importance_scores.unsqueeze(-1) *
    sparse_importance_scores.unsqueeze(-2)
total_importance = torch.sum(attention_weights, dim=[-2, -1], keepdim=
    True)
attention_weights = attention_weights / total_importance
adj_aggregated = adj + self.delta* attention_weights
```

其中的importance_scoring_network是一个简单的三层MLP：

```
class ImportanceScoringNetwork(nn.Module):
        def __init__(self, input_dim=507, hidden_dim=512, output_dim=1):
                super(ImportanceScoringNetwork, self).__init__()
                self.fc1 = nn.Linear(input_dim, hidden_dim)
                self.fc2 = nn.Linear(hidden_dim, hidden_dim)
                self.fc3 = nn.Linear(hidden_dim, output_dim)
        def forward(self, x):
                x = F.relu(self.fc1(x))
                x = F.relu(self.fc2(x))
                x = self.fc3(x)
                return x
```

此外，还可按照度排序等重要性指标赋予节点重要性分数，以进一步探讨改进MPNN时的重要性准则：

```
if importance_indicator=='degree':
        importance_scores = torch.sum(adj , dim=-1).float().squeeze(-1)
if importance_indicator=='defined':
        importance_scores = self.important_cal(adj).squeeze(-1)
sparse_importance_scores =F.softmax(importance_scores/self.tau, dim=-1)
```

4.3 实验环境搭建

硬件配置：使用单张NVIDIA GeForce RTX 3090Ti进行训练。

软件环境：使用Windows11操作系统。安装最新版本的Python（建议3.8或以上）。使用Anaconda管理环境，创建一个新的虚拟环境：

```
        conda create --name eco-dqn python=3.8
        conda activate eco-dqn
```

安装PyTorch及相关依赖项，确保支持CUDA加速：

```
        pip install torch torchvision torchaudio --extra-index-url https
           ://download.pytorch.org/whl/cu113
```

安装其他必要的库（如NumPy、Matplotlib等）：

```
        pip install numpy matplotlib networkx
```

4.4 使用说明

运行以下指令以开始训练和测试过程：

```
>>> cd eco-dqn
>>> python -m experiments.ER_20spin.train.train_eco
```

```
>>> cd eco-dqn
>>> python -m experiments.ER_20spin.test.test_eco
```

或是运行项目文件中的experiments/BA_20spin/train/train_eco.py文件开始训练，运行experiments/BA_20spin/test/test_eco.py开始测试，20为节点个数，文档中有节点个数为20，40，60，100，200，500的图集可选。

4.5 结果可视化

对于最大割问题，以下代码在训练过程中间隔一定时间步进行测试，得到训练曲线。

```
def plot_learning_curve(test_save_path, folder,agent):
        data = pickle.load(open(test_save_path, 'rb'))
        data = np.array(data)
        fig_fname = os.path.join(folder, "training_curve")
        plt.plot(data[:, 0], data[:, 1])
        plt.xlabel("Timestep")
        plt.ylabel("Max Cut")
        plt.savefig(fig_fname + ".png", bbox_inches='tight')
        plt.savefig(fig_fname + ".pdf", bbox_inches='tight')
        plt.clf()
```

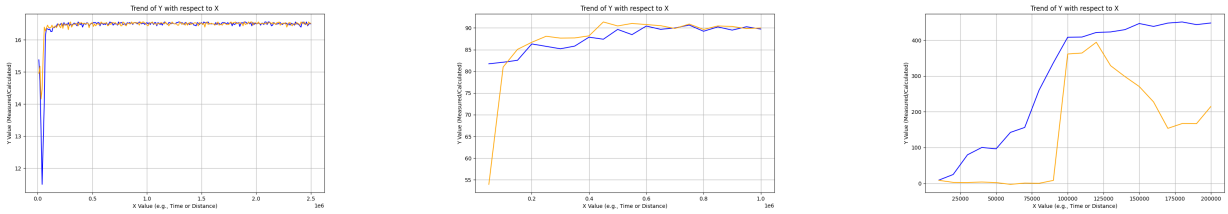
其中的测试分数由DQN类中的evaluate_agent方法得到：

```
@torch.no_grad()
def evaluate_agent(self, batch_size=None):
        if batch_size is None:
                batch_size = self.minibatch_size

        i_test = 0
        i_comp = 0
        test_scores = []
```

```
        batch_scores = [0]*batch_size

        test_envs = np.array([None]*batch_size)
        obs_batch = []

        ...

        return np.mean(test_scores)
```

5 实验结果分析

对于N=20，100，500时的可视化结果如下图所示，可见本文方法在小规模图上的解质量与ECO-DQN相当，后者已被表明在小规模图上的解质量十分接近最优结果 [2]，而在N=500的大规模图上，本文方法的求解性能优于ECO-DQN方法。



(a) (b) (c)

图 1. N=20，100，500时的训练曲线对比，其中黄色曲线为原文方法结果，蓝色曲线为本文改进后的结果。

6 总结与展望

6.1 总结

本文提出了一种改进的消息传递神经网络（MPNN）架构，通过引入重要性评分模块来动态评估节点的重要性，并结合加权邻接矩阵优化信息传播过程。实验结果表明，该方法提高了模型在超过500个节点的大规模图上求解最大割问题时的表现，具体而言：

- **创新点**：通过一个简单的三层MLP作为重要性评分模块，用于计算节点的重要性分数，并通过softmax层实现软注意力机制。这使得模型能够更关注于那些对优化目标贡献较大的节点，在只对原图结构进行小幅修改的前提下加速信息的有效传播。
- **实验验证**：在随机生成的测试图集上的测试结果表明，本文方法得到的解质量优于ECO-DQN方法和贪婪算法。
- **实际应用潜力**：本文方法有潜力推广至其它在图上定义的组合优化问题，特别是那些具有相似结构的任务，如蛋白质折叠和投资组合优化。

6.2 展望

尽管本文方法已经取得了一定的成功，但仍有若干方向值得进一步深入研究和探索。以下是几个可能的研究方向：

6.2.1 进一步的性能比较

为了更全面地评估本文方法的有效性，未来可以将其与其他传统算法（如模拟退火、遗传算法等）以及最新的深度学习方法（如基于注意力机制的Transformer、图卷积网络等）进行全面对比。具体来说：

- **基准测试**：在多个标准基准数据集上进行系统性的性能评估，包括不同规模和类型的图结构。
- **应用场景扩展**：针对特定领域的问题（如交通流量优化、电力系统调度等），开展更多实证研究，以验证模型的实际应用价值。
- **综合评价指标**：除了准确性和效率外，还可以引入其他评价指标（如鲁棒性、可解释性等），以提供更全面的性能对比。

6.2.2 节点重要性指标与可解释性探讨

当前的方法通过神经网络动态评估节点的重要性，但这种评估方式的可解释性仍需进一步探讨。未来可以将神经网络学习到的动态重要性与传统的静态指标（如节点度数、介数中心性等）进行对比，分析其差异和优势。具体来说：

- **静态指标对比**：详细对比神经网络学习到的重要性和传统静态指标之间的异同，探讨其背后的原因。
- **可解释性研究**：利用可视化工具和技术（如SHAP值、LIME等），揭示神经网络如何根据输入特征动态调整节点的重要性评分。
- **案例分析**：通过对具体实例的深入分析，展示神经网络如何捕捉复杂图结构中的关键节点及其邻居关系，进而提升优化效果。

6.2.3 其它组合优化问题推广

本文方法已经在最大割问题上取得了良好效果，但其在其他组合优化问题（如旅行商问题TSP、车辆路径问题VRP等）上的表现仍需进一步验证。具体来说：

- **问题适应性**：研究如何将本文方法应用于不同类型和复杂度的组合优化问题，探讨其适应性和通用性。
- **任务特定优化**：针对每个具体问题的特点，设计相应的改进策略，如引入特定的损失函数或奖励机制，以进一步提升性能。

- **跨域应用**：探索本文方法在不同领域（如物流配送、资源分配等）中的应用潜力，解决实际场景中的复杂优化问题。

综上所述，本文提出的改进方法为解决大规模图组合优化问题提供了一种新的思路。未来的研究将继续围绕上述方向展开，旨在进一步提升模型的性能和适用范围，推动相关领域的理论和技术发展。

参考文献

[1] K. Abe, Z. Xu, I. Sato, and M. Sugiyama. Solving np-hard problems on graphs by reinforcement learning without domain knowledge. *arXiv preprint arXiv:1905.11623*, 2019.

[2] Thomas Barrett, William Clements, Jakob Foerster, and Alex Lvovsky. Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pages 3243–3250, 2020.

[3] I. Bello, H. Pham, Q.V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[4] J. Gilmer, S.S. Schoenholz, P.F. Riley, O. Vinyals, and G.E. Dahl. Neural message passing for quantum chemistry. In *Proceedings of the 34th International Conference on Machine Learning (ICML)*, pages 1263–1272, 2017.

[5] J.J. Hopfield and D.W. Tank. "Neural" Computation of Decisions in Optimization Problems. *Biological Cybernetics*, 52(3):141–152, 1985.

[6] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 6348–6358, 2017.

[7] Z. Li, Q. Chen, and V. Koltun. Combinatorial optimization with graph convolutional networks and guided tree search. In *Advances in Neural Information Processing Systems (NeurIPS)*, pages 539–548, 2018.

[8] A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh. Learning heuristics over large graphs via deep reinforcement learning. *arXiv preprint arXiv:1903.03332*, 2019.

[9] W. Zhang and T.G. Dietterich. A reinforcement learning approach to job-shop scheduling. In *International Joint Conference on Artificial Intelligence (IJCAI)*, pages 1114–1120, 1995.