基于强化学习与约束规划的组合优化求解方法

徐晨龙

组内阅读会分享

2022 年 12 月 13 日

[1]Cappart, Quentin, et al. **"Combining reinforcement learning and constraint programming for combinatorial optimization**." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 35. No. 5. 2021.

[2]https://paperswithcode.com/paper/combining-reinforcement-learning-and-1

Abstract

- **组合优化问题 Combinatorial optimization problem** 在航空领域、交通规划、金融投资等多个领域中应用广泛
  1. 目标：有限的解空间中找到最优解
  2. 挑战：面临状态空间爆炸的问题
- **深度强化学习 Deep reinforcement learning** 作为一种启发式方法求解 NP-hard 问题时具有较好的效果，但是现有研究存在问题有
  1. 专注于标准的 TSP 问题，拓展性弱
  2. 可以给出一个近似的解，但是不能证明其最优性
- **约束规划 Constraint programming** 是解决 COP 问题的通用工具
  1. 基于一个完整的搜索过程，虽然运行时间较长但是可以确保找到最优解。关键步骤其分支探索策略

本文提出一种混合的基于学习的求解方法，尝试利用约束规划和深度强化学习结合的方式来求解组合优化问题，主要是利用动态规划过程作为桥梁，并在典型的带时间窗的 TSP 问题（TSPTW）和四阶段组合投资优化问题（4POP）中验证效果

Introduction

组合优化问题描述

在应用数学和理论计算机科学的领域中，组合优化是在一个有限的解集合中找出最优解的一类问题。常见的组合优化问题 [3]包括

- 车辆路线 (Vehicle route), 在给定的约束条件下车队寻找最佳的路线问题
- 调度 (Scheduling)，为了实现某一目的而对共同使用的资源实行时间分配
- 装箱 (Bin packing)，将尽可能多的各种大小的物体装入固定数量的最大容量的箱子中

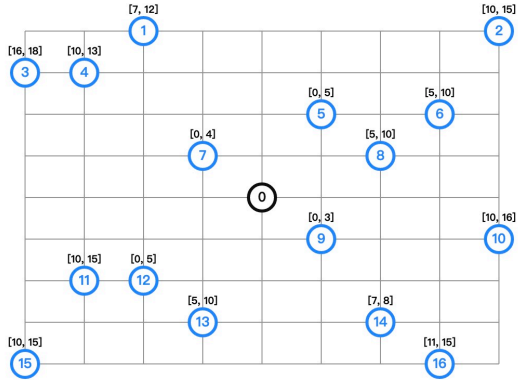
求解方法有很多，这里可以将其划分为两大类

1. 精确算法 Exact Algorithm
2. 启发式方法 Heuristic Algorithm

[3]https://developers.google.com/optimization/introduction/overview

TSPTW：带时间窗的 TSP 问题[4]

许多车辆路线问题访问固定节点需要考虑其对应的时间窗。这些问题被称为带时间窗的车辆路径问题。目标是最小化车辆的总行程时间。



[4]https://developers.google.com/optimization/routing/vrptw

精确算法是在全部的解空间中进行巧妙的枚举来得到最优解，但是在一些大型的问题中往往会出现求解时间过大。为了解决这个问题我们通常会使用**次优解**的方式来简化问题

- 优点：上述过程的简化是商业求解器 Cplex、Groubi、Gecode 的基础，让利用计算机求解组合优化问题成为可能
- 缺点：设计探索完整的解空间并有效的策略依旧是困难的，如何确定有效的启发式分支方法是 CP 求解问题的热门话题

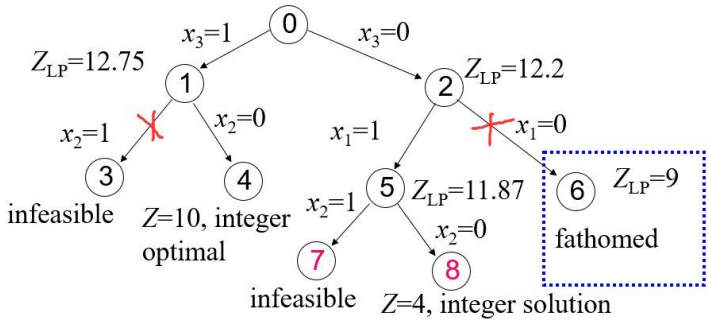
* 一个精确算法求解问题的例子：分枝定界过程

*** 非主线内容**

Maximize $4x_1+9x_2+6x_3$

Subject to $5x_1+8x_2+6x_3 \leq 12$

x_1, x_2, x_3 are binary variables



采用一种非完全遍历所有解空间而找到可行解的有效方式，

- 优点：可以求解使用 CP 无法求解的问题
- 缺点：不能证明解的最优性或者是优劣程度

常见的求解思路包括使用元启发式方法或 RL 的方式针对问题设计求解方案。利用强化学习求解的困难在于

1. 有一个很强的假设：我们生成的样本与实际问题的分布是相同的，并且生成的样本数量足够多来进行训练
2. 理论研究中强化学习针对的主要是 TSP 问题

* 一种强化学习启发式求解 TSP 问题过程

*** 非主线内容**

主要包括五个步骤: [a]

- 利用图来定义 TSP 问题
- 获取图节点和边嵌入的隐空间作为嵌入向量（GAT）
- 使用 MLP 对每个顶点进行前向传播来获得对应的概率值
- 利用经典的图搜索来得到离散解
- 上述神经网络的训练过程使用 RL

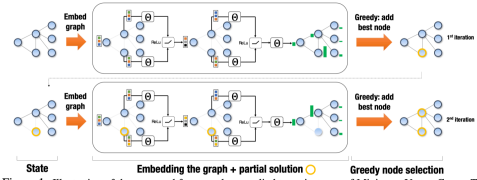


Figure 1: Illustration of the proposed framework as applied to an instance of Minimum Vertex Cover. The middle part illustrates two iterations of the graph embedding, which results in node scores (green bars).

图:

[a] https://www.chaitjo.com/post/deep-learning-for-routing-problems/?continueFlag=b220d49bda26d4033730216fbc9275d5

Introduction

本文工作与贡献

1. 对 COP 问题利用 DP 编码，将其转换为 RL 可学习的环境，以及 RL 可调整 CP 求解过程
2. 利用 DRL 中两大类的模型，一种基于 value 的 **DQN**、一种基于 policy gradient 的 PPO 对模型进行训练。[5]
3. 与现有的 CP 求解搜索策略结合，组成了 **BaB-DFS-DQN**、iterative-limited-discrpancy、Restart-search 结合的方式[6]
4. 最终的实验结果证明是有意义的
5. 代码最终开源，利用 Python 进行求解，并嵌入到 C 语言的 CP 问题求解过程

[5] paper 阅读会只介绍 DQN
[6] paper 阅读会只介绍 BaB-DFS-DQN

Section02- A Unifying Representation Combining learning and Searching

整体框架: CP 保证搜索完整性，RL 保证搜索效率

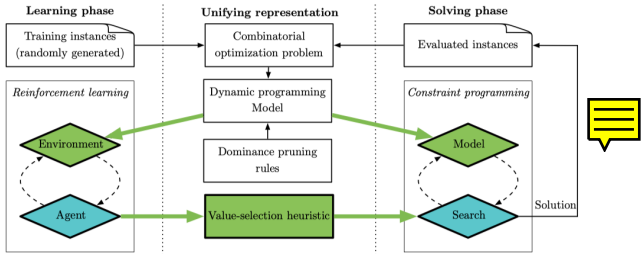


Figure 1: Overview of our framework for solving COPs.

1. COP 问题通过 DP 编码成为 RL 中的环境和 CP 中的模型
2. 在学习阶段，在 DP 编码转换得到的环境中，智能体利用 RL 方法学习某种策略，用来判断当前的最优分枝节点即最优动作
3. 在求解阶段，在 DP 编码转换的 CP 模型中，利用已有的求解器求解保证解的最优性，但是利用已经学习好的 RL agent 启发式判断分支节点

2.1 动态规划过程 DP

动态规划模型是一种将数学建模与计算机编程联系起来求解复杂优化问题的手段，简单来说是将一个大的问题转换成很多小的子问题然后将他们利用递归方程的方式联系起来最终求解，其可以描述为 $< X, S, T, R, V, P >$ 形式，也分为阶段、状态和状态转移三个要素。

名词	缩写	含义
control	C	控制变量
domain	D	变量取值范围
state	S	状态
transition	T	状态转换方程
reward	R	奖励
validity condition	V	有效性条件
dominance rule	P	剪枝条件

其中 Dominance pruning 是指在树搜索中提升搜索效率的手段。它的基本思想是，如果一个节点的值比其子节点的值都要小，那么这个节点就没有必要再进行搜索，因为它不可能成为最优解。通过这种方式，可以避免搜索冗余的节点，提高搜索效率。[7]

[7] https://chat.openai.com/

2.1 动态规划过程 DP

如何将 TSPTW 问题用 DP 描述

阶段 STAGE：给定 n 个节点，需要有 $n+1$ 个阶段

状态 STATE：表示以当前节点作为开始，给定节点作为结束的最佳路线，包含三个部分

- 剩下的没有被访问的节点 $m_i \in P$
- 最后访问的节点 v_i
- 当前的时间 t_i

转移 Transmit: 初始状态和状态转移过程如下列公式所示 [8]

$$
\begin{aligned}
s_1 &= \{m_1 = \{2..n\},\ v_1 = 1,\ t_1 = 0\} && \text{(Initial state definition)} \\
m_{i+1} &= m_i \setminus a_i && \forall i \in \{1..n\}\ \text{(Transition function for } m_i\text{)} \\
v_{i+1} &= a_i && \forall i \in \{1..n\}\ \text{(Transition function for } v_i\text{)} \\
t_{i+1} &= \max\left(t_i + d_{v_i,a_i}, l_{a_i}\right) && \forall i \in \{1..n\}\ \text{(Transition function for } t_i\text{)} \\
V_1 &: a_i \in m_i && \forall i \in \{1..n\}\ \text{(First validity condition)} \\
V_2 &: u_{a_i} \geq t_i + d_{v_i,a_i} && \forall i \in \{1..n\}\ \text{(Second validity condition)} \\
P &: \left(t_i \geq u_j\right) \Rightarrow \left(j \notin m_i\right) && \forall i,j \in \{1..n\}\ \text{(Dominance pruning)}
\end{aligned}
$$

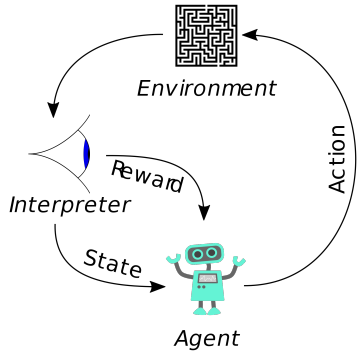
[8]利用 DP 求解 TSP 问题的过程-python
https://gist.github.com/winrey/c7de9e4743a2fc79cf8a30cfcd9c604d

2.2 强化学习环境

强化学习的目的

强调如何基于环境而行动，以取得最大化的预期利益，通常问题需要建模成为 MDP 过程，包括以下四点

1. 状态
2. 动作
3. 奖励
4. 状态转移



2.2 强化学习环境 RL Environment

将组合优化问题经过 DP 过程编码为 $< S, X, T, R, V, P >$，更进一步转换成为强化学习中的环境 $< S, A, R, T >$

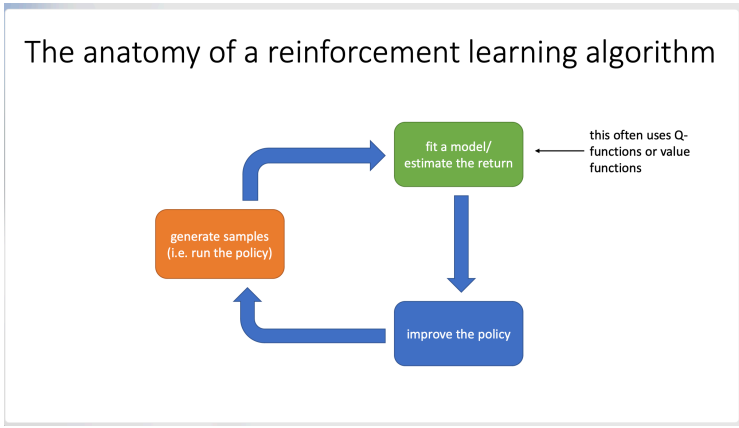
1. State, 包括原始的组合优化问题，动态规划的状态，在实践中状态通常需要使用神经网络转换成为一个 embedding 向量[9]
2. Action，也就是 DP 过程中的决策变量，同时需要考虑有效性 V 和剪枝 P 条件，$A_i = v_i | v_i \in D(x_i) \& V(s_i, v_i) = 1 \& P(s_i, v_i) = 1$
3. Transition，输入状态和输出动作的函数，和 DP 一样
4. Reward，为了满足一下的基本规则，需要对奖励有一点小小的改变
   - 完成的轨迹要优于没有完成的轨迹
   - 好的轨迹需要优于坏的轨迹

   $R(s, a) = \rho * (1 + |UB(Q_p)| + r(s, a))$

[9] https://blog.csdn.net/u010412858/article/details/77848878

2.3 RL agent DQN & PPO

目的是根据在环境中不断的探索训练智能体，**得到考虑整体奖励下，当前状态对应的最好的动作**，在 Model-free 的方法中主要分为 Value-based 和 Policy-gradient 的方法，**具体算法细节本文并不涉及**，强化学习范式[10]可以描述为：



[10] https://rail.eecs.berkeley.edu/deeprlcourse/

2.4 神经网络框架 Neural network structure

如果希望求解方法具有泛化，需要注意两点

1. 对于相同问题，不同数量的输入
2. 对于不同问题的输入

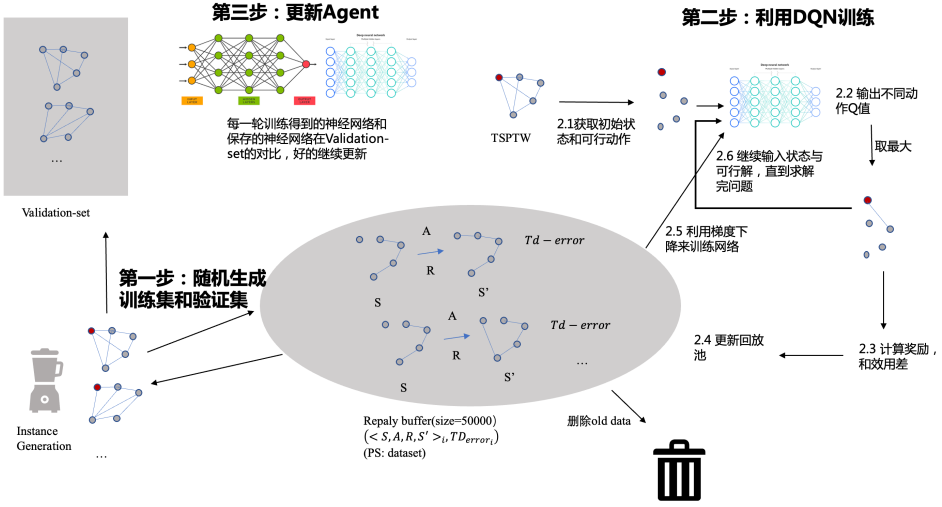
所以套用 GAT(Graph attention network) 来作为 TSP 问题的特征提取，最后利用 Softmax 输出不同行为概率

GAT=GNN+Transformer[11]

[11]Vaswani, Ashish, et al. "Attention is all you need." Advances in neural information processing systems 30 (2017).

* DQN 算法训练过程的 code

利用效用之差进行训练，如果估计当前状态 S 下采用 a 的效用，需要求解到问题结束。这里采用 $(< S, A, R, S' >, TD)$ 方式利用效用差训练; 期望得到输入状态 S 和动作 A 能够得到效用 Q 的智能体



2.5 约束规划编码 Constraint Programming Encode

如何将动态规划描述的 $< X, S, T, R, V, P >$ 转换成为约束规划的问题 $< X, D, C, O >$

这里采用的 Gecode 求解器，它的变量与 DP 过程对应的关系

- X 代表变量的集合，D 代表变量的取值范围。对变量做进一步的取份
  - 决策变量 X_i^a decision variables
  - 辅助变量 X_i^s auxiliary variables
- C 代表约束, 保证
  - 保证 DP 过程的收敛
  - 添加额外的冗余约束 redundant 加增加搜索效率
- O 代表目标函数，即最小化运行时间

$$
\begin{array}{lll}
\mathbf{x}_1^s = \epsilon & & \text{(Setting initial state)} \\
\mathbf{x}_{i+1}^s = T(\mathbf{x}_i^s, \mathbf{x}_i^a) & \forall i \in \{1, \ldots, n\} & \text{(Enforcing transitions)} \\
\texttt{validityCondition}(\mathbf{x}_i^s, \mathbf{x}_i^a) & \forall i \in \{1, \ldots, n\} & \text{(Keeping valid transitions)} \\
\texttt{dominanceCondition}(\mathbf{x}_i^s, \mathbf{x}_i^a) & \forall i \in \{1, \ldots, n\} & \text{(Pruning dominated states)}
\end{array}
$$

2.6 搜索策略 Search Strategy

将上述过程与不同的搜索策略相结合，这里仅展示 BnB 与 DQN 的结合

1. 输入 COP 和预训练好的 RL agent
2. 编码转换为 CP 问题进行求解，开始利用 BnB 进行搜索
3. 在搜索没有结束之前，每次从当前 CP 问题的状态转换为 RL 的状态；同时根据网络得到最佳动作
4. BaB 的内在的分支、回溯等都被放在 BranchAnd Update 里面, 利用 RL agent 选择的动作来更新分支定界过程
5. 额外使用一个缓存池 κ 来避免重复计算

Algorithm 1: BaB-DQN Search Procedure.

▷ **Pre:** $\mathcal{Q}_p$ is a COP having a DP formulation.
▷ **Pre:** $\mathbf{w}$ is a trained weight vector.
$\langle \mathtt{X}, \mathtt{D}, \mathtt{C}, \mathtt{O} \rangle := \mathtt{CPEncoding}(\mathcal{Q}_p)$
$\mathcal{K} = \emptyset$
$\Psi := \mathtt{BaB\text{-}search}(\langle \mathtt{X}, \mathtt{D}, \mathtt{C}, \mathtt{O} \rangle)$
while Ψ **is not completed do**
  $\mathtt{s} := \mathtt{encodeStateRL}(\Psi)$
  $\mathtt{x} := \mathtt{takeFirstNonAssignedVar}(\mathtt{X})$
  if $\mathtt{s} \in \mathcal{K}$ **then**
    $v := \mathtt{peek}(\mathcal{K}, \mathtt{s})$
  else
    $v := \mathrm{argmax}_{u \in D(\mathtt{x})} \hat{Q}(\mathtt{s}, u, \mathbf{w})$
  end
  $\mathcal{K} := \mathcal{K} \cup \{\langle \mathtt{s}, v \rangle\}$
  $\mathtt{branchAndUpdate}(\Psi, \mathtt{x}, v)$
end
return $\mathtt{bestSolution}(\Psi)$

Experiment result

1. OR-tool 和 DQN 表现最差;CP-nearest 和 PPO 也很不错，同样 CP-nearest[12]证明最优性较好，PPO 找到可行解效果较好（验证之前引言）
2. 本文提出的混合的方法显著好于这些方法，同时增加缓冲池可以提高搜索效率

Table 1: Results for TSPTW. Methods with ⋆ indicate that caching is used, *Success* reports the number of instances where at least a solution has been found (among 100), *Opt.* reports the number of instances where the optimality has been proven (among 100), and *Time* reports the average execution time to complete the search (in minutes, and only including the instances where the search has been completed; when the search has been completed for no instance *t.o.* (timeout) is indicated.

Approaches		20 cities			50 cities			100 cities		
Type	Name	Success	Opt.	Time	Success	Opt.	Time	Success	Opt.	Time
Constraint programming	`OR-Tools`	100	0	< 1	0	0	t.o.	0	0	t.o.
	`CP-model`	100	100	< 1	0	0	t.o.	0	0	t.o.
	`CP-nearest`	100	100	< 1	99	99	6	0	0	t.o.
Reinforcement learning	`DQN`	100	0	< 1	0	0	< 1	0	0	< 1
	`PPO`	100	0	< 1	100	0	5	21	0	46
Hybrid (no cache)	`BaB-DQN`	100	100	< 1	100	99	2	100	52	20
	`ILDS-DQN`	100	100	< 1	100	100	2	100	53	39
	`RBS-PPO`	100	100	< 1	100	80	12	100	0	t.o.
Hybrid (with cache)	`BaB-DQN*`	100	100	< 1	100	100	< 1	100	91	15
	`ILDS-DQN*`	100	100	< 1	100	100	1	100	90	15
	`RBS-PPO*`	100	100	< 1	100	99	2	100	11	32

[12]CP-nearest 是一个利用表格记录动作 Q 值的启发式约束规划混合求解方法

Discussion &Conclusion

- 再次强调本文不是第一种尝试使用 ML 提升组合优化求解器，但是这篇文章主要是尝试 RL 和 CP 求解的混合方式来求解组合优化问题，在学习阶段利用 DP 编码的方式训练 RL、在求解阶段依旧使用现有 CP 求解框架保证解的最优性、利用训练好的 RL agent 提升现有求解过程的效率
- 神经网络调用时间虽然相比训练很短，但是重复调用依旧有很大的计算开销，除了使用缓存池之外；进一步可以考虑使用知识蒸馏或者更紧凑的网络来等价
- 本文使用 Pybind11 来结合 Python 训练的 DQN 和 C++ 中的 CP 求解器，但是更好的解决方案是落实到单一、高效的框架之中。

* 个人感悟

1. 分享本文的目的是希望找到一篇 RL 实际应用到例子，本文在学习阶段的方式是将组合优化问题利用 DP 的方式转换为 $< S, A, R, T >$ 来进行学习训练，同时这篇文章是开源的。在阅读本文收获了优化相关精确算法和启发算法结合的混合求解思路，并看到了实实在在论文背后的实现方法（code）。
2. 这篇论文从计算机的角度来看是不是也是一类方法集成呢？另外本文虽然从 TSP 走向 TSPTW 和 4POP，但是他们仍然逃离不开典型的优化问题场景，所以交通是否更应该是对场景中遇到的困难规范性的描述并求解？