上海交通大学
SHANGHAI JIAO TONG UNIVERSITY

机器学习大作业报告

梁俊轩 姜楠 袁跃
522031910439 522031910437 522031910357

2025 年 1 月 5 日

我们主要采用了 Reinforcement Learning 方法，并通过热力图的形式将 BPP 二维化以降低复杂度，进而求解该问题。对于 Task1，我们使用了模仿学习中的 Behavioural Cloning 方法，利用生成数据集的特性让 Agent 向最优解靠拢。对于 Task2，我们采用树搜索算法来进行改进。对于 Task3，我们将前面通过行为克隆预训练好的策略网络与实际环境进行交互得到奖赏，利用 REINFORCE 算法进行梯度上升，使得 Agent 能够处理没遇见过的情况。

1 问题介绍

Bin Packing Problem (BPP)，即装箱问题，是一个经典的 NP-hrad 组合优化问题。在这个问题中，我们需要将一系列不同大小的物品尽可能高效地装入一个或一组箱子中，目标是最大化箱子利用率。

我们本次大作业基于监督学习和强化学习来尝试解决 BPP 问题，并且假设箱子可以在三维方向旋转 $\pm 90°$，不能将箱子放在已有箱子的下方，这些假设是为了使我们的方案更加符合现实物理世界。在 Task1 中我们需要产生数据集并实现和训练基本的策略网络，以解决将物品放入单个箱子中的问题；Task2 则需要在 Task1 的网络中加入树搜索算法以更高效的解决 BPP 问题；Task3 允许使用多个箱子，我们要将测试数据尽可能高效地装入箱子中，并输出最终每个物品的位置和旋转情况。

2 解决思路

2.1 Task1

Task1 为 BBP 问题做了引入，提出了一种较为简单的情况，即箱子体积为 $100\times100\times100$ 并且待装箱物品体积由完全的 $100\times100\times100$ 立方体随机分割而来（这里的随机应至少包含四个层次：切割次数的随机（限定范围内）、切割维度的随机、切割位置的随机、旋转方式的随机）。在这种情况下，待装箱物品理论上可以达成 100% 填充率，换言之，该背景下给定了问题的最优解，即 100% 填充率，为我们确定优化方向与模型评估提供了便利。

2.1.1 问题抽象

BPP 是一个涉及三维空间的问题，但是结合实际的物理情况考虑，我们并不能够“任意”地沿着 Z 轴放置物品，因为我们必须保证每个物品的下方有支撑（更严格地说，一定程度的支撑）。因此我们可以将 BPP 二维化 [5]，以热力图的形式表示箱子的情况，具体如下：

1) 设定一个 100×100 的二维数组 St（即离散化的平面），用以模拟箱子的俯视角。
2) 该数组中每个元素的值表示该单元格的“高度”，不能超过 100。
3) 新的物品在放置时不允许放置在该“高度”以下。

图1给出了更直观的解释：

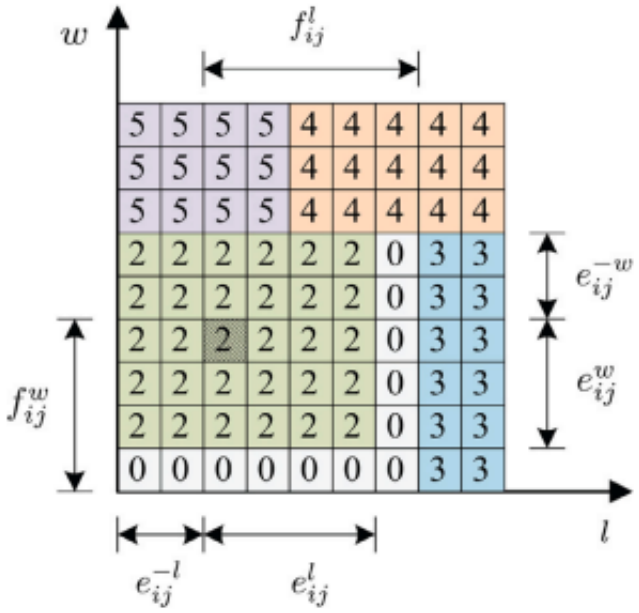

图 1 装箱情况平面化示意图 [5]

借由此法，我们可以一定程度地降低 BPP 的复杂度，也可以大幅缩减模型的训练时间。

2.1.2 方法介绍

我们大体上依照 Project 任务书的提示，采用 Reinforcement Learning 方法，并结合查阅到的大量资料，最终决定采用 Behavioural Cloning（即“行为克隆”）[7] 方法，接下来将对该方法进行介绍。

(1) Reinforcement Learning 基础理论：

涉及 Reinforcement Learning 时，想到 *State* 与 *Action* 的概念是相当自然的，为了能让 Agent 能在判断时对过往的信息加以利用，我们考虑将过去 n 次 step 的热力图和箱子参数拼接起来表示为 state，在 BPP 中，二者具体定义如表1所示：

$State(s)$	过去 4 次 step 的热力图和箱子的参数
$Action(a)$	由 Agent 给出的物品放置的坐标 (i, j)，对应的旋转 r

表 1 *State* 与 *Action* 的定义

而我们的目的则是寻找一个策略 π_θ 使得以下目标函数最大化：

$$J(\theta) = \mathop{\mathbb{E}}_{s_0, a_0, \ldots s_T} [\sum_{t=0}^{T} \gamma^t r(s_t, a_t)]$$

(2) Behavioral Cloning：

基于 Reinforcement Learning 之上，Behavioral Cloning 顾名思义，关键在于“模仿与克隆”。在该方法中，我们让模型通过模仿 Expert Policy（专家策略）来进行学习，使之尽可能向 Expert Policy 靠拢。在 Task1 中由于待装箱物品是由 $100 \times 100 \times 100$ 立方体分割而来，在此过程中保存的 label 数据显然就可以作为 Expert Policy 指导模型。

(3) Replay Buffer：原始的采样数据往往存在着不容忽视的相关性问题。这种相关性会使得模型在学习时，难以精准捕捉到不同状态下的通用策略，进而导致学习效果不佳。与此同时，未经处理的数据分布处于一种较为动荡、不稳定的状态，这也给稳定的梯度更新带来极大挑战。为了有效打消采样数据之间紧密的相关性，同时促使数据分布朝着更稳定的方向发展我们采用了 Replay Buffer，将训练产生的数据保存到 Buffer 中，在 episode 结束后再从 Buffer 随机采样，对采样的数据进行更新。

Task1 的伪代码如下：

算法 1 给定最优策略的 Behaviour Cloning

Input: 数据生成得到的 label 数据整理而成的“状态-动作”组 (S, A^*);

Output: 策略 π;Repaly Buffer B

随机初始化策略 π;

while 未到达设定的 *Episode* 数目 **do**

  获取当前 State 下的 Action 预测：$A = \pi(S)$;

  将数据装入 B 中

  从 B 中随机采样

  计算损失函数：$\boldsymbol{L} = ||A - A^*||^2$;

  反向传播;

  已完成 Episode 数目 +1;

return π;

(3) 细节优化：

无论从实践角度还是 Reinforcement Learning 本身的理论出发，Agent 的策略都不可能完全与 Expert Policy 相同。在 BPP 中二者之间的细微差异需要被充分地纳入考量，在某些情况下，这些差异会导致某些物品的放置方式非法，另一些情况下，已经放置的物品的差异会导致后续物品即使被正确地放置也会变为非法。因此，我们需要着重考虑当 Agent 给出的放置方式非法时的处理。经过调研与思考后，我们得出了较为合理的处理方法，下面以伪代码的形式给出：

算法 2 Agent 给出非法放置方式时的处理

```
/*Agent 给出的放置坐标合法时无需此处理 */
if Agent 给出的放置坐标 (i, j) 非法 then
    从 (i, j) 开始由远及近启发式地寻找可以合法放置物品的位置;
    if 寻找到了合法的位置 (i′, j′) then
        将物品放置在 (i′, j′) 处;
        给予 Agent 惩罚，惩罚数值与 (i, j) 和 (i′, j′) 之间的距离成正相关;
    else
        /* 未寻找到合法位置时 */
        不给予惩罚，修正 Agent;
```

以及还有物品的旋转状态需要考虑，但并不影响上述的基本架构，因而不再赘述。至此，Behavioural Learning 算法基本实现。

2.1.3 有关数据集的问题

虽然我们的算法在理论上可行，但是在实际训练过程中发现效果并不理想。我们经过思考，认为问题在于随机分割生成物品的过程中容易产生“长条状物品”（即某一维度尺寸远大于另外两维度，呈现长条状的物品），而“长条状物品”会放大 Agent 与 Expert Policy 之间的差异所带来的影响。直观来看，若“长条状物品”在被放置时与正确位置出现偏差，则由于其在某一维度上的大尺度，会使更多数量的后续物品无法放置在正确位置甚至是无法放置，反之若众多物品中的某几个发生了偏差，则“长条状物品”也将无法放置。更进一步地，我们单独研究了数据生成部分的代码，将“长条状物品”定义为满足“某一维度尺寸不小于另外两维度尺寸之和 2 倍”的那些物品，发现若按照真随机原则对 $100 \times 100 \times 100$ 立方体进行随机切割（即保证切割次数随机、切割维度随机与切割位置随机），那么在超过 95% 的情况中会出现 1 个及以上的“长条状物品”，在超过 80% 的情况中“长条状物品”的数量超过了生成物品总数量的 10%。这对于我们的模型训练是十分不利的，因此我们在训练阶段有必要对数据生成加以限制，具体为优先选择存在较大尺寸的维度进行切割，并且限制切割位置不得过于靠近两端，从而避免“长条状物品”的出现。在这样的有限制数据集上进行充分的训练后，再适用到无限制数据集上。

2.1.4 结果

按照以上理论及步骤，最终 Task1 所得结果以图2中的热力图形式展示：
占用率为 60.6784%，放入物品数量为 47。

2.2 Task2

Task2 将在 Task1 的基础上结合树搜索算法以进一步高效地解决 BPP 问题。Task1 已经给出了完整的模型训练逻辑，我们依然延用上述的模型训练方法，在此基础上引入 search_space 和 stability scores，结合树搜索算法对 agent 的当前状态作出简单评估。

具体而言，在执行放置物品的 Action 之前，会先对当前的 State 进行稳定性评分。该评分函数会首先检测当前物品能否放置在该位置，并根据平稳与否（在物品覆盖范围内高度是否一致）给出初始化稳定得分，之后对物品上下左右四个方位的边界和顶角连续性进行稳定性评测，具体的评分细

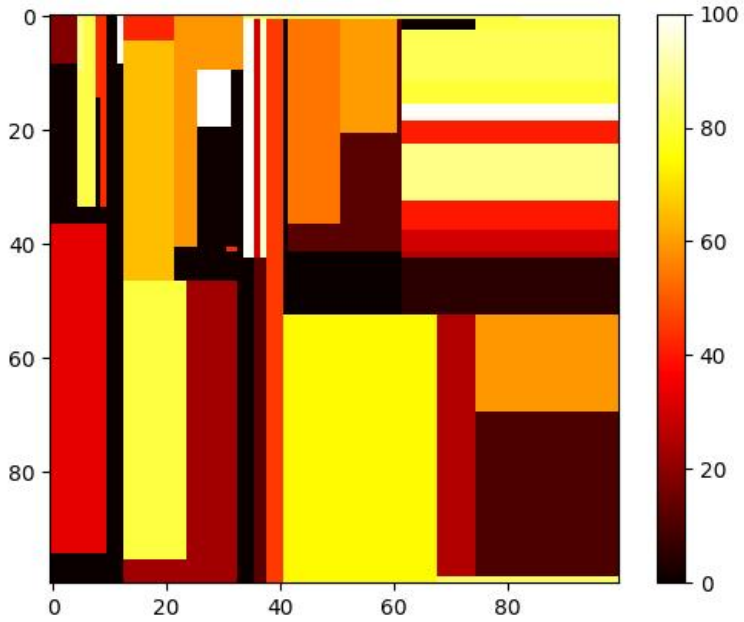

图 2 Task1 结果

则可以参考实际代码，下面给出稳定性评分的伪代码形式：

算法 3 对当前放置位置的稳定性评分

Input ：放置物品的目标位置 (i, j)，箱子的高度地图 ldc，物品的形状 $dimn$，操作区域的范围 $(currldc\_x, currldc\_y, currldc\_r)$

Output 稳定性评分 $stab_score$

:

if 物品可以放置在目标位置 (i, j) **then**

    $stab\_score = initial\_flat\_score(i, j, ldc, dimn, currldc\_r)$

    $stab\_score += calculate\_edge\_stability(i, j, ldc, dimn, currldc\_x, currldc\_y, currldc\_r)$

    $stab\_score += calculate\_continuity\_stability(i, j, ldc, dimn, currldc\_x, currldc\_y, currldc\_r)$

    $stab\_score -= calculate\_penalty(i, j, currldc\_x, currldc\_y, ldc)$

else

    $stab\_score = -10$

return $stab\_score$

在实现稳定性评分的基础上，我们再结合树搜索算法引入 search_space。我们并未实现显式的树结构，而是直接利用树搜索的想法，在初步估计物品的放置位置后，如果稳定性得分较低，则在 search_space 搜索以找到更高得分的放置位置，这种方法类似于在解空间树中从当前节点向多个子节点扩展，同时如果当前位置的得分高于之前位置的最高得分，则直接舍弃之前的无效位置分支，只

保留当前位置的信息以实现剪枝操作。每当找到最佳位置后，调用 step 函数更新当前装箱状态，同时将放置当前物品后的临近位置加入到 search_space 中，即在树的某个节点上生成一个新的子状态。

我们在实际代码中两次运用了树搜索的思想，第一次是在随机初始化一个放置位置时，此时的 search_arr 我们设置为初始放置位置 ±6 范围内的区域，如果在搜索完该区域后仍然无法找到合适位置，则在根据之前已放置物品生成的 search_space 内再次利用上述树搜索算法进行寻找。

结合上述思想，我们给出一次树搜索算法的伪代码形式：

算法 4 利用树搜索算法寻找合适位置

Input ：搜索空间 $search\_space$, 物品的形状 $dimn$

Output 放置位置 (x, y)

:

随机初始化放置位置 (x_0, y_0)

$score = get\_stability\_score(x_0, y_0, dimn)$

if $score$ 低于目标值 **then**

  for (x, y) *in* $search\_space$ **do**

    $new\_score = get\_stability\_score(x, y, dimn)$

    if $score < new\_score$ **then**

      $x_0 = x, y_0 = y$

根据 (x_0, y_0) 更新 $search\_space$

return (x_0, y_0)

根据上述思想与方法，最终 Task2 得到的结果以图3中的热力图形式展示：

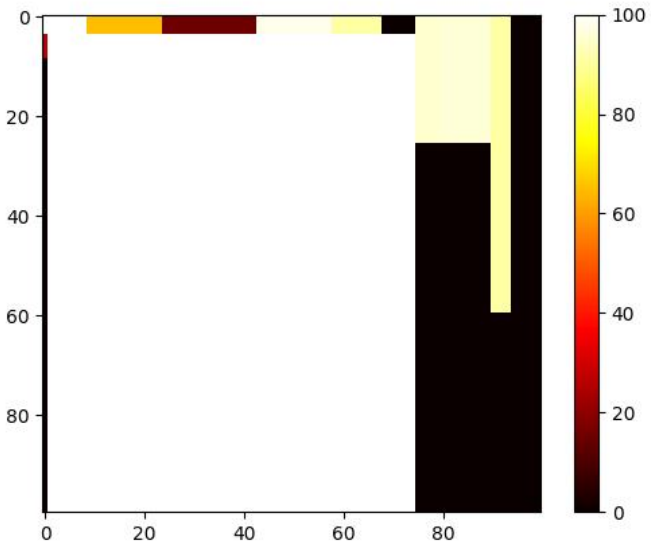

图 3 Task2 结果

占用率达到了 78.78%，产生的 14 个物品仅有 1 个没有放入。

2.3 Task3

面对更加复杂多变的情况，如不同尺寸的箱子和不同大小的物品，行为克隆的表现并没有那么好，因其本质是监督学习，单纯模仿人类专家的动作来学习策略，面对未知的情况就无能为力。

近来的一些工作注重于使用强化学习算法来解决这个问题 [3],[5],[8]。但是问题中涉及的箱子尺寸较大且要考虑旋转，单纯使用如 PPO[6]、A3C[4] 等强化学习算法对于 agent 而言很难去寻找出一个好的策略。事实上我们也曾经尝试过，在探索部分我们会展示使用 Maskable PPO[2] 实现的结果。但这不是我们在 Task3 使用的算法。

前面我们已经实现了一个基于行为克隆的策略网络，让它可以在专家数据集里进行训练，并在 Task2 部分我们实现了 stability scores 去结合树搜索方式简单评估 agent 当前状态的奖赏，那么我们可以基于此使用策略梯度 (REINFORCE) 算法，训练一个 Agent 去与真实的环境进行交互，从而在已经从人类专家动作预训练得来的策略网络进行进一步优化。

我们采用总共三种奖赏，第一种是体积奖赏 R_{Vol}，鼓励 agent 去学习盒子较大时的放置策略，第二种是平坦奖赏 R_{Flat}，鼓励 agent 优先填满相同高度的空间，第三种是 Task2 使用的 $R_{Stability}$，其中 R_{Vol} 和 R_{Flat} 如下所示：

$$R_{Vol} = \frac{v_{Box,t}}{B \times L \times H} \tag{1}$$

$$R_{Flat} = \frac{h_{max} - h_{min}}{H} \tag{2}$$

$v_{Box,t}$ 是当前盒子的体积，B、L、H 分别是箱子的宽、长、高，h_{max} 代表箱子已经填上的最高高度，h_{min} 代表箱子已经填上的最低高度。

总共的奖赏是三种奖赏的加权求和：

$$r_t = \beta_1 R_{Vol} + \beta_2 R_{Flat} + \beta_3 R_{Stability} \tag{3}$$

在实验中奖赏权重如表**??**所示：

表 2 奖赏权重

β_1	β_2	β_3
0.3	0.4	0.5

有了以上的奖赏设计，其伪代码如下所示：

算法 5 REINFORCE Based on Behaviour Cloning

Input ：策略网络 $\pi_\theta(a|s)$, Replay Buffer B, 学习率 α, episodes 总数目 N, N_1, 奖赏权重 β_1,β_2,β_3

Output 最优策略网络 $\pi_\theta(a|s)$

:

初始化 $\pi_\theta(a|s)$, B

初始化专家数据集 D

for $n = 1$ **to** N **do**

  获取当前 State 下的 Action 预测： $A = \pi(S)$;

  将数据装入 B 中

  从 B 中随机采样

  计算损失函数： $\boldsymbol{L} = ||A - A^*||^2$;

  反向传播;

  已完成 Episode 数目 +1;

for $n = 1$ **to** N_1 **do**

  初始化状态 s_0

  for $t = 0, 1, \cdots, T$, 从 $a_t \sim \pi_\theta(a|s_t)$ 采样 actions， 生成轨迹 $\tau = \{s_0, a_0, r_0, s_1, a_1, r_1, \cdots, s_T, a_T, r_T\}$

  $G_t \leftarrow \sum_{k=t}^{T} \gamma^{k-t} r_k$ for $t = 0, 1, \cdots, T$

  for $t = 0$ **to** T **do**

    $\nabla_\theta \ln \pi_\theta(a_t|s_t) \leftarrow \theta \leftarrow \theta + \alpha \cdot G_t \cdot \nabla_\theta \ln \pi_\theta(a_t|s_t)$

return $\pi_\theta(a|s)$

通过以上方式训练好的 agent，可以面对不同箱子的情况。那么对于多个箱子的情况，采取贪心的策略，即先填满一个再去填满下一个，我们使用作业要求所提供的盒子数据，最后实验结果表3所示。

表 3 多个箱子放置情况对比

算法	体积占用率
Behaviour Cloning	61.22%
REINFORCE Based on Behaviour Cloning	64.09%

让 agent 在真实环境下进行强化学习的探索之后，学习到了更多的策略，也使得最后体积占用率也有所提升。

3 探索

3.1 Maskable PPO

在一开始，我们曾经尝试采用 Maskable PPO 来解决装箱问题。Maskable PPO 可以在每次 step 前根据当前状态，生成一个可供 agent 选择的 action mask，使得 agent 可以根据 action mask，直接在可以放下盒子的 action 之间选择，同时 PPO 是目前强化学习普遍采用的 Policy Based 的算法。

但是实际上情况并没有如自己预想的。首先，在 Task1 中要求的箱子尺寸为 $100\times100\times100$，在这里为了能生成 mask 实际上是只考虑离散值的情况，即所放置的位置和盒子尺寸都是由整数来表示。agent 的离散 action 是有 100×100=10000 种选择的，此外又要考虑到 6 种旋转的情况，则总共是 60000 种选择。如此多选择就会导致每次生成 mask 的所需要的计算资源庞大，同时 agent 也不一定能完全探索出真正好的策略。

在这里我们附上在 $100\times100\times100$ 的单个箱子上，花费一周时间在一张 GeForce RTX 3090 训练 1000 个 epochs 后的结果，总共有 48 个盒子，最后只能放得下 16 个盒子，且空间占用率只有 10%。

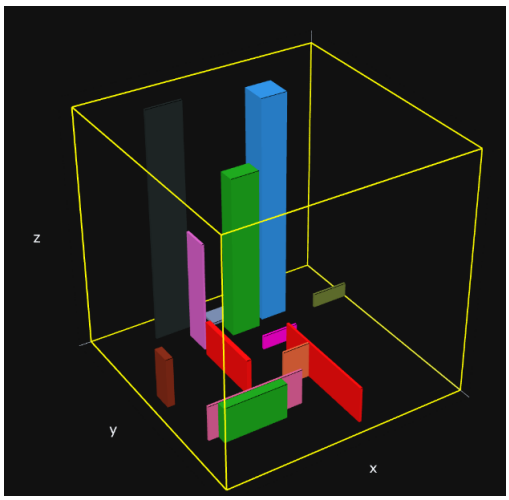

图 4 Maskable PPO 测试结果

3.2 预测 Stability Score

我们不应仅仅克隆动作，还可以克隆动作及 Stability Score，这样智能体就能了解到，即便在（x，y，rotation）方面仅有细微差异，稳定性评分也会有很大变化，伪代码如下:

算法 6 给定最优策略和稳定性得分的 Behaviour Cloning

Input: 数据生成得到的 label 数据整理而成的 “状态-动作” 组 (S, A^*, R^*);

Output: 策略 π;Replay Buffer B, 稳定性得分 R

随机初始化策略 π;

while 未到达设定的 *Episode* 数目 **do**

  获取当前 State 下的 Action 预测：$A = \pi(S)$;

  将数据装入 B 中

  从 B 中随机采样

  计算损失函数：$\boldsymbol{L} = ||A - A^*||^2 + ||R - R^*||^2$;

  反向传播;

  已完成 Episode 数目 +1;

return π;

3.3 Inverse Reinforcement Learning

此外，我们有考虑到模仿学习的另外一种方法——Inverse Reinforcement Learning(逆向强化学习)[1]，不需要设置 reward function，而是从专家的行为中推测出奖励函数。也就是说，智能体试图学习一个奖励模型，使得在这个奖励模型下，专家的行为是最优的。与行为克隆不同，IRL 不直接模仿专家的动作，而是通过学习奖励函数来理解专家为何采取这些动作。一旦学习到奖励函数，智能体就可以通过传统的强化学习方法来优化自己的策略，从而在相同或相似的环境中做出决策。但出于时间关系，没有去采用这个方法。

4 总结

这次作业我们围绕利用机器学习解决装箱问题（BPP）展开，利用行为克隆与强化学习的结合详细应对不同任务场景，当然这里有很多需要改进的地方，例如强化学习算法还可以再进一步进行优化，但总的来说收获颇丰，也认识到机器学习在解决实际问题上能发挥巨大作用。

5 小组分工

Name	Student ID	Score	Work
梁俊轩	522031910439	70%	所有 Task 的代码、训练，idea 探索，论文 Task3 和探索部分撰写
姜楠	522031910437	15%	Task1 代码完善，论文 Task1 部分撰写
袁跃	522031910357	15%	Task2 代码完善，论文 Task2 部分撰写

参考文献

[1] Dylan Hadfield-Menell, Anca Dragan, Pieter Abbeel, and Stuart Russell. Cooperative inverse reinforcement learning, 2024.

[2] Shengyi Huang and Santiago Ontañón. A closer look at invalid action masking in policy gradient algorithms. *The International FLAIRS Conference Proceedings*, 35, May 2022.

[3] Alexandre Laterre, Yunguan Fu, Mohamed Khalil Jabri, Alain-Sam Cohen, David Kas, Karl Hajjar, Torbjorn S. Dahl, Amine Kerkeni, and Karim Beguir. Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization, 2018.

[4] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning, 2016.

[5] Quanqing Que, Fang Yang, and Defu Zhang. Solving 3d packing problem using transformer network and reinforcement learning. *Expert Systems with Applications*, 214:119153, 2023.

[6] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017.

[7] Faraz Torabi, Garrett Warnell, and Peter Stone. Behavioral cloning from observation, 2018.

[8] Jingwei Zhang, Bin Zi, and Xiaoyu Ge. Attend2pack: Bin packing through deep reinforcement learning with attention, 2021.

A 附录

具体代码详见：https://github.com/hitefork/ML-proj-BPP