## R-Tree Algorithm Analyse

现有的R-Tree算法主要分为两大类：批量数据构建R-Tree 和 逐个数据构建

前者根据给定的所有批量数据，进行打包划分，逐层打包生成R-Tree结构，后者根据逐个到来的数据进行插入，增量式地构建R-Tree。考虑到批量打包划分的方式，无法应对实时插入的场景，后者逐个插入数据来构建R-Tree是主要的实现目标。

围绕 两种构建方式的全局顾及的能力、传统和强化学习方法构建的差异、强化学习状态环境和奖励设计的实现这三个方面进行分析

### 全局顾及

可以感知的是，在机器学习（主要是强化学习）方法构建R-Tree的方面，批量打包划分相比逐个数据插入的方式，更能从数据全局出发考虑，构建出更好的结构，因为一开始就具有所有数据的视野，从所有数据中找到近似最优结构的划分和打包方式，逐层向上或者向下来构建一棵R-Tree，整体数据固定，从始至终就能围绕一个最终的静态结构去实现其最优结构

批量划分打包的构建方式更能顾及全局数据的分布特征。但是这种方式无法应对增量式的数据或者实时数据，面对新增的数据，只能重新构造，完全是另一次完整的构建实现，这也是考虑逐个插入数据构建方式的最主要原因

逐个数据插入构建R-Tree的方式注定无法较好地顾及全局。

假设现在有 n 总量的数据，逐个插入这n个数据来构建R-Tree， 当前某层节点中插入第 i 个数据，选择了 “算法下最优的” 某个孩子节点进行插入，但是很有可能，在随后到来的第 j 个新数据的视角之下，之前的插入选择可能并不是最好的。对于批量划分打包的构建方式，则一开始就能处于 第 j 个数据的视角，顾及到全局。

RW-Tree中，会结合历史查询负载的记录，来预测之后可能出现的查询负载，从而实现了“即使当前插入位置不是最优，但是考虑到长远查询收益是更好的选择”这样一种思想。根本目标在于减少未来查询负载的成本（查询次数），虽然做到了忽略当前选择插入位置可能存在的 “伪全局最优” 的特征，但是 是从预测查询负载成本的方向，去选择可能的更好位置，和我们想要实现的目标不一致。并且，其查询负载是假设在历史查询负载分布稳定的理想情况下，不一定适应于现实的查询负载。

首先明确一些过程，对于当前要插入一个数据，会从根节点开始，向下不断选择要插入的孩子，最后一直走到叶子节点并插入其中。

1. 当前插入数据就是已知全部数据的最后一个（全部数据就是所有叶子节点中的数据加上当前插入数据）
2. 在插入过程中，**每一层的选择，都可能会引发树的结构变化**（被选择的孩子的MBR变化，当然如果数据MBR被包含其中，也是可以不变的）
3. **视野问题**。每一层的节点选择，传统算法中会根据孩子节点计算加入数据的面积增量、周长增量、重叠面积增量等特征，来达到 当前步中能引起最小化的结构恶化的可能（重叠面积增大、最后树高增长等）
4. **中间层的结构优化**。对于每一个到来的新数据，它和所有叶子节点在二维平面上的分布及其范围是确定的，树根节点的大小范围是确定的，树结构的重要内容都集中在中间层次上的结构，过往算法也都是在中间层次上，每走一步都会想办法引发最小的结构恶化、实现最大的结构优化，但这些操作都是局部的，视野永远只局限于所在层要选择的孩子，不会考虑已有数据的分布情况，也不会考虑树结构中其他中间层

我们想要实现的目标，是希望能够实现一个具有一定全局顾及能力的R-Tree构建方法，某些时候，当前节点的孩子选择，或许并不是局部最优，但是考虑现有全部数据，所造成的查询收益是更好的

看起是希望，能像批量打包划分那样，能看到全部数据的分布情况，那要怎样做到呢？去预测未来的新数据么？

1. 预测成本，高精度预测需要额外复杂模型，不容易融合到构建算法中，可能会增加系统计算负担
2. 偏差风险，预测不一定准确，如果未来数据分布随机，预测并无优势

总的来说，预测的手段并不靠谱， 包括之前RW-Tree基于历史查询记录预测，也是建立在理想假设之上的，没了这个假设，并不会有优势，甚至可能恶化。

那么新的问题：

既然无法很好准确地预测未来数据，但是针对当前已有数据（假设当前已有数据也是所有批量打包划分构建R-Tree方式的给定数据），那么这样两者之间的差异在哪里？

批量方式，一开始就已知了现在的所有数据，能顾及全局数据分布特征，然而动态构建R-Tree的方式，直到最后一个数据的到来，才能看到全部数据的全貌，但是过去的数据已经产生了那些数据在当时（考虑所在层孩子而进行的选择）的局部最优结构

**如何在每次新数据到来的时候感知所有数据的分布特征，使得构建过程中不仅仅局限于当前所在层的孩子节点的选择，从而做到针对现有全体数据的近似最优？**这样的话，在一定程度上或许可以减少选择时只考虑当前所在层孩子造成的局部最优，因为局部最优的累积并不一定朝着总体最优的方向逼近，最后的整体树结构不一定是不断逼近全局最优的结果。

如果构建R-Tree的时候能考虑针对全体数据的近似最优结构，那么在每一层的时候，并不是必须根据传统算法的选择策略进行（MBR的最小面积、周长、重叠面积增量），也就是说在某一层，树的插入子树的选择，并不一定服从传统算法下的选择策略，但是对于整体树的构建是有益的。

那么需要做什么才能实现这样的效果？那是接下来强化学习实现要考虑的问题。

这样的效果，传统算法中的 Hilbert R-Tree是有一些的，但是其弊端也很明显，对于树结构的变化，重新编排内部节点需要大量计算Hilbert value， 并且无法有效处理分布不均匀的数据，实时更新方面，频繁的插入删除场景也无法适应。

**总之，目标总结为构建R-Tree时考虑现有全部数据，每次子树选择的时候并不一定服从传统指标，来实现整体树结构的近似最优。**

### 传统方法和强化学习方法

**很重要的一点在于每层兄弟节点之间的重叠面积**，重叠部分可能会导致产生多条查询路径，传统方法的R\*-Tree、R+-Tree等都采用措施减小了这个指标，目的是为了使得查询尽可能有单一的查询路径，这是一个非常重要的指标。

对于面积增增量和周长增量指标，纳入考虑的原因也是为了提高空间利用效率，降低重叠，使得树的结构更加紧凑，直观地看，每个节点MBR包围其所有孩子条目，其中，MBR覆盖但是并没有孩子与之重叠的“死区面积”减小了，这样面对未来查询，可以尽早确认查询区域中没有数据从而提升查询效率（降低节点访问次数），这也是一个重要的指标。

在RLR-Tree中没有考虑直接考虑树的高度，而是考虑每次选择子树的时候各个孩子节点的填充率，尽可能的满装每个节点有助于降低树的高度。那么每层节点的总填充率是否可以纳入考虑？能体现什么？填充的满说明尽可能的划分了父节点MBR的空间范围。

**如果每一层的节点，填充又满，重叠又小（最好没有），周长、面积又小（局部更紧凑），那么下一层的孩子在各自的父节点所框定的范围下也具有这些特性，整体树结构就更能趋近最优。但是如果逐层实现这一最优特性，对后续层次上的节点的最优特性实现施加了过于严格的限制，最后对全局结构造成负面影响**。

在全局视角下，会有许多因素需要总和考虑和平衡，尽可能靠近“每层都有较优特性”但是又为未来数据留有余地的整体结构，也就是说并非每层都是最优。

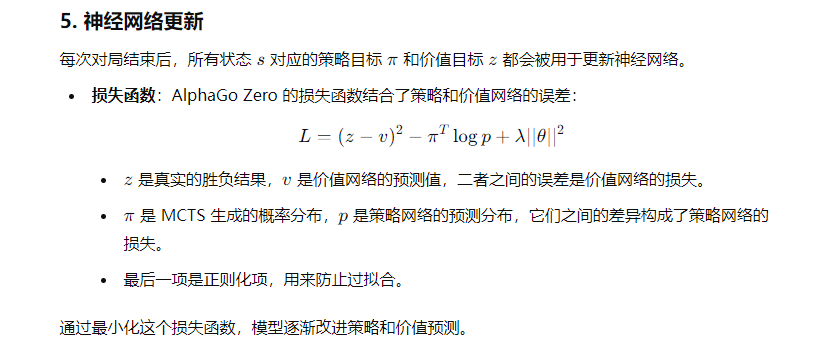
**所以从这个角度看RLR-Tree在一定程度上有所实现该平衡，每层不再是考虑最小指标，而是综合多种指标，进行排序之后暴露出一部分留给agent通过大量数据训练来进行选择决策**。但是其视野依然是局限于当前所在层的孩子节点，这种平衡留有余地，可能会得到更好的选择，但是相较于整体数据，agent看不到整体数据的分布，每层实现的较优特性仍然对后续结构具有一定的限制和约束。理论上依然有可以提升的空间。

设想这样的情况，从根节点向叶节点方向，在每一层要进入下一层子树的选择总数都有max\_entry最大节点容量个，那么每一种选择，后续又会有对应max\_entry种选择，对应树的高度，逐个插入数据构建一棵R-Tree所面临的选择可能是max\_entry的h幂次方，h是树的高度，对于数以百万计的数据，每一次将数据插入到叶子结点所面临的可能个数是无法穷尽的

对于批量打包划分构建的方式，面对给定的所有数据，如何一层层划分和打包也是面临无法穷尽的选择，PLATON算法选择蒙特卡洛搜索，使用有限的探索使得模型参数去逼近最优模型，类似于AlphaGo Zero，通过自我博弈，在每次对局的每一步棋中，都会选择最有价值的一个选择深入探索，最后使得模型的参数，无需穷尽每一种选择，但是可以近似做出最优的选择。

稍微深入看一看AlphaGo Zero的做法，采用一个策略-价值网络（都是CNN），都输入棋盘状态，一个用于输出策略，即棋盘上每个位置的概率分布，即下一步落子的选择。一个用于评估策略价值，即下一步落子的选择能有多大价值（有助于最终胜利）。然后使用MCTS搜索进行对局，记录一局过程中的统计数据，用于训练神经网络。

MTCS的过程是：选择、扩展、模拟、回溯。



MCTS从根节点开始选择，根据每个节点的UCB进行选择，UCB结合了策略网络给出的落子概率和价值网络评估的落子价值，选择一个节点后进行扩展，也就是在树中添加一个新的子节点，从新的子节点进行一次模拟，模拟方式是使用策略网络给出落子选择，直到终局结束。过程中会记录每一次落子的信息，然后根据终局胜负结果来优化策略和价值网络（参考上面公式）。然后重新从根节点开始一次MCTS。

在这个过程中，每一步的选择会有无数种可能，这里直接使用神经网络给出概率或价值，根据最后的结果与过程信息之间的误差来更新网络参数（联合更新），**即通过有限次（也是非常多次的训练）路径探索来把学习到的策略直接反映在网络参数当中，直接建立网络参数-输赢之间的映射关系**，最后的参数模型就是逼近最优策略的模型。

AlphaGoZero采用CNN来获取棋盘状态，棋盘上每个位置的状态受到相邻位置的影响很大，即局部棋形很重要，CNN卷积核可以有效捕捉局部模式，多层卷积捕捉局部到全局的特征，可以识别到不同种类的棋形特征，多层卷积逐步聚合局部信息，还能捕捉到全局态势，生成下一步落子概率分布。通过大量对局训练，CNN可以学会当前棋局状态与最终结果之间的复杂联系，从而给出落子价值的评估。

**强化学习本身面对无穷的探索空间，目的就是学得一个能够做出更好动作的近似最优策略模型。**

**回归到逐个插入数据构建R-Tree上，需要在每一层的子树选择时，评估每一种选择的可能价值，给出一个合适的策略，最后优化出具有“参数-近似最优树结构”映射关系的模型。**

### 强化学习方法实现

根据前面的分析，

那么对于R-Tree的构建，如何有效感知整个树的环境状态？（如何设计合适的全局状态）

如何使得网络能捕捉树的整体结构信息，包括每一层的局部信息和所有层的全局信息？从而捕捉全局态势，下一个孩子的选择？

如何使得网络学习到当前树的整体结构与最后更优整体之间的联系？（那么要确定最终建成树后结构优秀与否的**评价标准**）

对于这些问题的回答，不得不说，采用CNN是一种看起来非常有吸引力的选择，用CNN来捕捉整个树的环境状态，但是动态变化的R-Tree平面结构到CNN固定输入的映射是一个很大的问题。在RLR-Tree中，仅采用当前层次的一些特征作为状态，并非是感知树的整体环境的状态设计。

#### RLR-Tree方法分析

先回顾RLR-Tree方法，其环境和奖惩设计的思路是怎么来的。

在明确了选择增量式数据下逐个插入数据构建R-Tree的方式后，**要面对的问题是什么？**动作设计比较简单，就是每层做选择的时候选择哪一个孩子。

1. 用什么去代表agent选择动作时所观察到的环境状态？

在每一层选择孩子节点的时候，传统算法有那些加入新数据后的MBR面积、周长、节点重叠面积增量来作为选择孩子的依据，都是服务于树结构，使其朝着目标形状去构建。并且每次分裂也会导致树的部分重构。

论文中关于MDP中的状态设计，有这样一段话：“**A state captures the environment that is taken into account for decision making**. For ChooseSubtree, **it is a natural idea that a state is from the tree node whose child nodes are to be selected for inserting a new object**. The challenging question is: what kind of information should we extract from the tree node to represent the state”。

也就是说，状态捕获环境信息，什么样的环境信息，用于做决策的环境信息。**那么状态从哪里来？**插入数据需要自顶向下从根节点遍历，每个节点中都要决定选择哪个孩子插入数据。很自然的想到，**状态应该从要做出选择动作的节点中来**，然后关于提取怎样的节点信息来表示状态，就是那些指标综合的实验了。

（关于分裂的实现，也是，从要分裂的节点中提取状态，所有可能的分裂情况就是其动作选择空间）

1. 最后要优化的目标是什么？即如何实现奖励？

还要说一个“Transition” 状态转移的概念，做出动作之后，环境发生状态的变化，对于RLR-Tree，其状态变化 (s, a) ——> s， 这个 s\_ 就是所选择并进入（插入数据）的子节点，即另一个下一层的节点

关于奖励设计，文章中是这么说的：“**A reward associated with a transition corresponds to some feedback indicating the quality of the action taken at a given state**. A larger reward indicates a better quality. Since our objective is to learn to build an R-Tree that processes query efficiently, **the reward signal is expected to reflect the improvement of query performance**”

也就是说奖励能反馈，在s状态下做出a选择后转移到 s\_状态时的动作a的质量好坏，并且出于目标考虑，希望奖励能够反映查询性能的提升。

选择子树的过程中，新数据尚未完全插入到树的节点，很难直接评价某个状态下的动作选择（某个节点中的孩子选择）时好时坏。直接想法就是数据完全插入之后，执行一些随机范围查询。这样看起来，agent好像会被鼓励采取动作建成一棵尽可能有很少查询次数的树。但是有些问题：

1. 过去的动作会影响树结构 以及未来树的查询性能。 所以一个“好”的 动作可能因为过去的一些不好的动作而获得很差的奖励
2. 更重要的是，最直接的目标是要比 对比的传统树表现的更好，所以非常感兴趣，什么样的选择会使得生成的树优于对手。
3. 随着更多新数据插入R-Tree， 查询的平均次数是递增的，奖励信号也会越来越弱。之前那样直接执行范围查询 既无法区分正负奖励

后面就是熟悉的非常 novel 的奖励设计内容了…

#### 使用CNN的评估

### 环境设计想法

目标： 构建一棵好的树：每层的重叠面积越小越好

#### 状态从哪里来？

做出动作选择发生在某一层节点中。

状态仅从该节点中提取，视野也仅局限于该层插入数据会发生的变化。想要扩大视野到所有数据选择一个更“优”的位置。

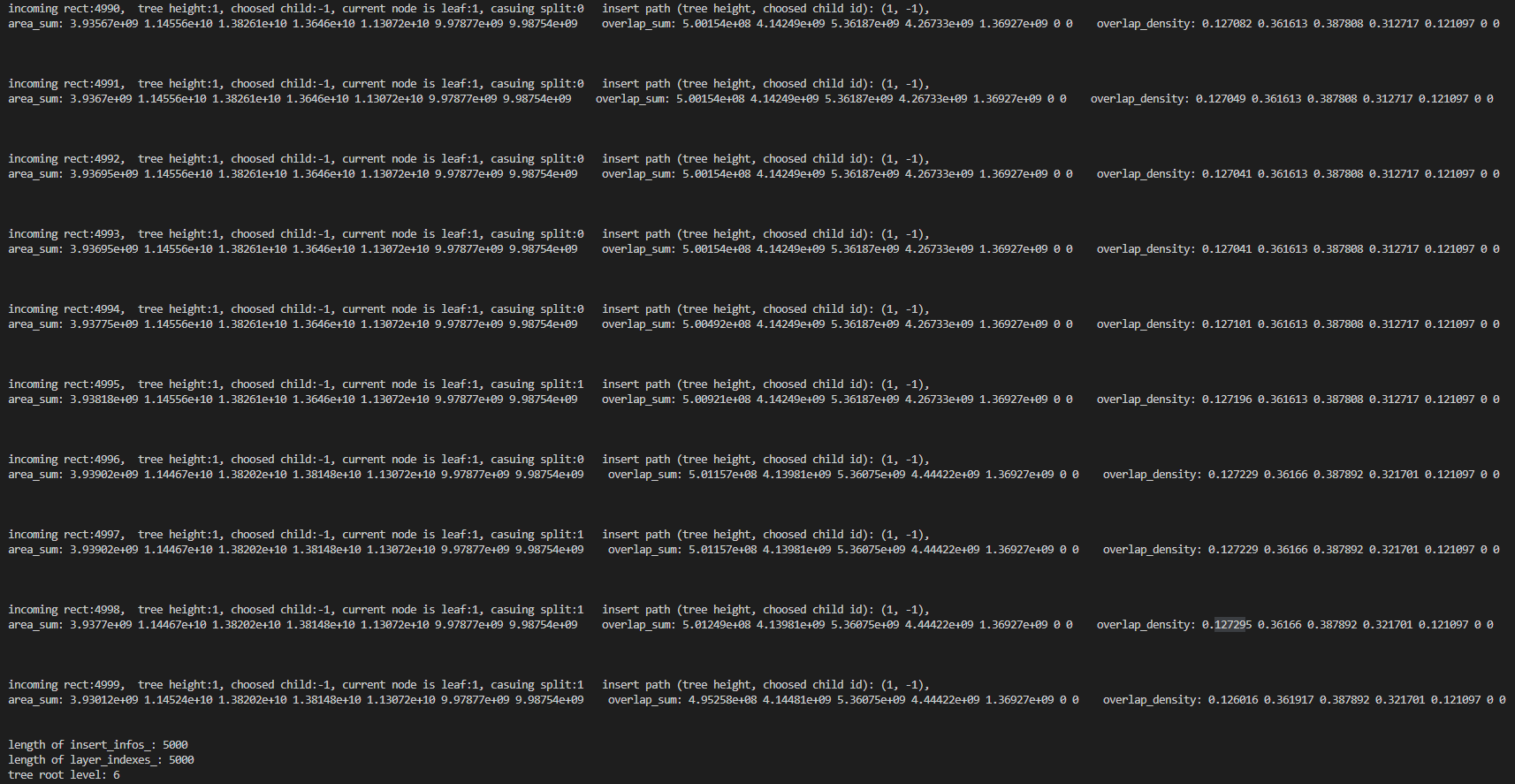
某一层插入一个数据，上面的层次中已经不会再变化，会影响的是该层次以下的所有层次的节点MBR变化（面积、周长增量、与其他兄弟的重叠面积增量等）。也就是做出一个动作，整个树环境会产生变化的是动作所在层以下的结构。并且完全插入数据后，如果叶子分裂，树环境变化可能会是整体，也可能仅仅向上数层。

**理想情况下，在某一层的某个节点插入一个数据，由环境变化而捕捉到的状态，应该涉及到上述内容**。

这些环境变化真的有用么？每层插入一个数据，引发的MBR变化可能非常微小甚至没有。继续向下会引起下一层节点的MBR的变化，每一层的变化如何综合起来？构成一个整体的变化？状态又如何固定长度？（固定数量的只有横向的节点最大容量，纵向无论树高还是节点数，都是变化的）

### 需要确定的内容

1. 指标观察（每插入一个数据观察一次）：
   1. 层节点总面积、层节点总重叠面积、层间总结点面积占比
   2. 层间总面积比值
   3. 计算时间：随着数据量的增加，总计算时间变化
2. 传统R-Tree构建过程中，每次插入一个数据，树的一些特征是如何变化的？（每层节点总面积、每层节点总重叠面积占比总面积、层间总面积比值），这些比值和查询之间有什么关联？



在传统树中观察相关数据可知：

在构建树的过程中，…. 没有明显规律。但是可以明确的是，越靠近叶子结点的中间层的层重叠面积占比都是比较大的，反而比叶子结点的重叠面积占比大。也就是说，传统方法建树，层数越高，在向下选择插入子树的时候越容易造成中间层重叠面积增大，因为由已知数据形成的结构固定了下来，后续插入数据即使能够更好地被划分，也无法重构了。

但是每次选择动作就计算一次全局相关指标，计算量巨大，时耗高

可以重点关注如何减少中间层的重叠面积

一开始就朝着局部最优的方向进行插入孩子的选择，后续对整体最优的限制会比较大。

**为什么会这样？**

一开始，根节点分裂，形成两个孩子，那么后续节点会继续在这两个孩子中选择插入位置，，新的孩子只会在已经存在的叶子节点满装后分裂产生，实际上对后续结构产生了很大的限制。但是由底层叶子结点满装后分裂向上新增父节点，这又是维持树结构平衡的重要手段。

如果选择子树的过程中，位于的某一个节点容量未达到最大，而待插入数据加入任何孩子都会引发孩子的MBR扩增，**那么不如独立成该节点的一个孩子节点？但是这样又可能会导致存在不满足最小容量的节点**。

不管这些的话，若当前节点的达到最大容量，即使加入某个孩子依然会引发MBR面积、重叠面积扩增，此时再进行选择，不一定就按照指标选取局部最优，这个选择的策略由神经网络给出。

**当前节点的所有孩子，其中心位置的坐标是否可以纳入考虑？**

如果可以，怎样向下探索更多层次？每个孩子都有（中心点， 长， 宽），新数据如果被包含在一个孩子当中，加入其他孩子势必引发重叠面积增加。新数据如果没被包含，则加入后引发的重叠面积、面积、周长增量是否应该被优先综合考虑？如果是，则倾向于更大的局部最优构造。如果能向下多探几层中心坐标，观察分布是否可行？如，agent在选择的时候，根据传入的状态，我可以看到这个孩子所划定的范围下，总共有多少孩子，总共面积密度是多少？

如果考虑中间层的重叠面积，那么极有可能是由于叶子结点（本身众多兄弟叶子就有一定重叠面积，但是那是由于自顶向下进行局部最优选择时累积的划分情况）从叶子节点层次往上的中间节点，实际上重叠情况更加严重。

那么当一开始选择子树的时候，就关注叶子结点上一层的节点的重叠情况，旨在降低导数第二层的重叠面积占比，那么最后整体构造会不会好一些？

1. 新思想：**延迟插入**（存够一定数量数据，再进行插入，从数据层面增加全局视野，哪怕视野窗口只有2个，即存两个插入一次）**只是辅助手段**。
2. 新思想： 主动分裂。
3. 采取从上至下插入孩子的构建方式，但是我有一个构建目标来指导每一层子树的选择，那就是综合最小化中间层的重叠面积占比，在插入一个数据的过程中，先在一开始观察数据分布和叶子结点的装载情况，实际上就是从现有数据的全局出发得到一个期望插入的位置，然后让深度强化学习的agent在自上而下插入过程中每一次的子树选择中执行策略网络给出的动作，这个动作并不一定遵循局部最优的指标，但是经过大量数据训练，能够使得策略尽可能让数据插入到期望位置附近