Symbolic AI : Search

纲要

1. 问题表征
    1. 状态空间 / 与或图
2. 图搜索
    1. 朴素的图搜索算法
    2. 盲目搜索
    3. 启发式搜搜
3. 博弈搜索
    1. 极大极小值搜索
    2. $\alpha - \beta$ 剪枝
    3. MCTS

搜索的本质

1. 在可接受的代价下，在所有的可行解中寻找到最有或者一个近似的可行解
2. 在效率和解的结果之间做出权衡
3. 方法
    1. 确定性算法
    2. 随机化算法
4. 汉诺塔问题抽象
    - 用状态空间表示问题的本质
        - 找到初始解和终止解，找到从初始解到终止解的一条可行路径
        - 表示状态转移的过程，描述状态空间（图），在状态空间中找到一条可行解的路径
        - 状态空间的解法和求图的路径基本是一致的思想
5. 状态空间

    (S, F, C, I, G)

    - S : 状态集合
    - F : 状态转移
    - C : 代价函数
    - I : 初始解集合
    - G : 终止解集合

    以状态为节点，以后继为边集合构建图
6. 问题归约
    1. 将困难的问题归约成等价的简单的问题来求解
    2. 与或图
        - 每一个节点是一个待解决的问题或者是步骤
        - 问题的搜索
            - 与 : 问题的分解
            - 或 : 问题的转化
        - 每个节点只有两种状态
            - 可解决
            - 不可解决
        - 将图递归缩小成一个子图逐个击破

- 步骤
    - 表示每一个子问题
    - 使用与或图归约图
    - 自底向上回溯寻找可行路径，判断root节点是否是可解决的

朴素图搜索

1. 搜索的过程中扩展节点
2. 基础数据结构
    1. Open表 : 保存探索但是没有扩展的节点
    2. Close表 : 保存解决的扩展节点
3. 搜索算法
    1. 加入初始节点进入Open表
    2. 从Open选定一个节点扩展，否则回溯
    3. 将选定的节点放入Close表，将扩展出的节点放入Open表中排队
    4. 重复直到结束(找到终止节点或者Open表为空)
4. 搜索算法

   选择扩展节点的策略不同导致我们的算法的方法不同

    - 盲目搜索
        - DFS : 回溯
        - BFS : 队列的排队问题也是一个焦点
        - DLS :
            1. 受限的深度有限搜索
            2. 迭代加深搜索是在速度上接近广度优先搜索，空间上和深度优先搜索相当的搜索方式
            3. 迭代加深搜索适用于当搜索深度没有明确上限的情况
            4. 算法思想
                - 深度有限搜索在限定的层数的时候回溯
        - IDS :
            1. 迭代加深搜索
            2. 迭代加深搜索算法就是仿广度优先搜索的深度优先搜索。既能满足深度优先搜索的线性存储要求，又能保证发现一个最小深度的目标结点
    - 启发式搜索
        - 思想

          对存储的待解决的节点的排序方式上的考虑
        - 建立评估函数病每次选取最有可能扩展出理想节点的节点作为下一个带选择的节点
        - A*
            - 对状态空间的搜索
            - $f(n)=g(n)+h(n)$
                - $g(n)$: 目前节点的代价函数
                - $h(n)$: 估计代价函数,核心
                    1. 如果该函数的值确保可以小于实际的代价值的话，我们可以保证搜索是最优解(好的代价函数从来不会过高的估计代价)
                    2. 代价函数在满足上面的情况下，如果值越贴近实际代价值则可以保证速度更快，如果代价函数贴近0可以保证算法接近于Dijstka算法
                    3. 上述的条件要求是对所有的节点均成立的

        - $f(n)$: 总代价
    - 证明

$$
\begin{aligned}
f(G_2) &= g(G_2) \quad since\ h(G_2)\ is\ small \\
g(G_2) &> g(G) \quad since\ the\ G_2\ is\ unoptin \\
f(G) &= g(G) \quad since\ the\ G\ is\ the\ optin \\
f(G_2) &> f(G)
\end{aligned}
$$

        - 确定G_2节点的估值是大于最优点的估值

$$
\begin{aligned}
h(n) &\leq h^*(n) \\
g(n) + h(n) &\leq g(n) + h^*(n) \\
f(n) &\leq f(G) < f(G_2)
\end{aligned}
$$

        - 只要$h(n)$的估值是合理的(小于理想估值)，我们可以保证A*算法是最优的
            1. $h(n)$估计代价无论怎么设计，总会保证一个非最优值节点的$f(G_2) > f(G)$
            2. $h(n)$只要不会过高估计我们的代价函数，那么可以保证算法一定会在选取一个更优秀的非最优值节点(直到找到最优解)
    - 最优化的思路

        只要我们可以很好的设计我们的$h(n)$函数，我们可以保证最优并且速度非常快(不会扩展无用的节点,每次选取离最优点估计最合适的[最合适意味着可以找到最优的路径，最合适保证了我们扩展的时候的最优性]节点扩展)

博弈搜索

1. 分硬币问题举例

    定义问题状态

    $$Game\ State : (C_1, C_2, C_3, \ldots, C_n,\ \ MIN\ or\ MAX)$$

2. 思想
    - 双方行动永远选择最有优势的一步来行棋
    - 可以考虑使用与或图来求解
    - 但是因为有时候我们思考的棋的解空间是非常的巨大的，所以我们必须使用有效的手段防止我们的过度搜索
        - 截断测试 : 深度受限制的搜索
        - 评估函数 : 没有办法搜搜所有的可行解而我们利用评估函数尽可能选择可以到达最优解的路径
3. 算法
    - 极大极小值搜索
        1. 我方行棋考虑搜索当前分支叶子的极大值(我方最优步骤选取)
        2. 对手行棋考虑搜索当前分支叶子的极小值(我方最差步骤选取，相当于对方最优决策)
    - $\alpha - \beta$ 剪枝
        - 可以在极大极小值搜索的基础上，优化效率，剪去没有必要考虑的分支
    - MCTS
        - 没有评估函数，没有截断测试的随机模拟方法
        - 在叶子节点模拟后将结果作为最后的评分
        - 算法思路

1. 选择：根据权重(之前经常访问的子决策)进入最优先进入的子决策中
2. 扩展：添加新的子节点进入树中保存
3. 模拟：根据默认策略从子节点开始模拟生成运行的结果
4. 反向传播：将模拟结果备份沿途反向传播到根节点

- 伪代码

Algorithm 1 General MCTS approach.

```
function MctsSearch(s_0)
    create root node v_0 with state s_0
    while within computational budget do
        v_l ← TreePolicy(v_0)
        Δ ← DefaultPolicy(s(v_l))
        Backup(v_l, Δ)
    return a(BestChild(v_0))
```

- 选择步骤的原则
  - 最大获胜情况决定
  - 访问次数最多情况
  - 最大并且访问次数最多的子节点
  - 找到最安全的节点(最大置信下界)
- ？？？？？？？？？？UCT :对于选择和扩展步骤中我们可以使用UCT算法来最佳决策？？？？？？？？？？？

  UCB + MCTS

  1. 最大上限置信区间算法(UCB)

  $$UCT = \bar{X}_j + 2C_p\sqrt{\frac{2lnn}{n_j}}$$

     - $\bar{X}_j$: n_j节点的仿真测试平均值
     - n : 当前节点(父节点)被访问的次数
     - n_j : j号子节点被访问的次数，$n = \sum_j n_j$
     - c_p : 常量,用来平衡深度和广度

  2. 树内选择策略可以有效的将我们的搜索方向集中在更有可能获胜的分支上

     从根节点开始，对于每一个非叶子节点n的子节点n_j,根据UCB公式计算出一个评估值，根据评估值选择最优的一个子节点扩展，直到到达叶子节点

     - 如果是极大的情况(我方行棋)：选取评估值最大的子节点扩展
     - 如果是极小的情况(对方行棋)：选取评估值最小的子节点扩展

  3. 默认仿真策略

     当搜索到　　当前的树的叶子节点的时候，UCT算法执行扩展操作：

     1. 把此叶子节点允许的所有合法下一步产生的子节点， 作为新的叶子节点加入到搜索树中， 并正确初始化其X_j值和n值
     2. 对当前的子节点进行随机的模拟(当然随机的棋力不高，需要考虑人为设定的快速走子系统)
     3. 对于该节点的执行多次的仿真，确定仿真结果的平均值$\bar{X}_j$

  4. 仿真结果回传

     1. 回传直到根节点
     2. 父节点的$\bar{X}$的计算是所有子节点的$\bar{X}_j$的按照访问次数的加权平均值

AlphaGO简单了解

1. 定义游戏

- 游戏状态
- 下一步动作

$$P(next\ axtion \mid current\ state) = P(a|s)$$

2. 算法思路
    - 监督学习 :
        - 专家系统
        - 获取预测模型
        - CNN
            - 极大似然法
            - 梯度下降
    - 强化学习 :
        - 自我对弈提升实力
    - 强化学习 :
        - 评估棋局
        - MSE(均方误差最小) : 梯度下降
        - ????引入策略网络(广度)和价值网络(深度)进行评估????