2021《人工智能》本科生课程

搜索与求解

教学课程组

2021年

- 参考教材：　　吴飞，《人工智能导论：模型与算法》，高等教育出版社
- 在线课程(MOOC)：　　https://www.icourse163.org/course/ZJU-1003377027
- 在线实训平台（智海-Mo）：https://mo.zju.edu.cn/classroom/class/turing_ai_2021
- 在线共享资源（智海在线）：http://www.wiscean.cn/online/intro/zju-01

提纲

一、搜索算法基础

二、启发式搜索

三、对抗搜索

四、蒙特卡洛树搜索

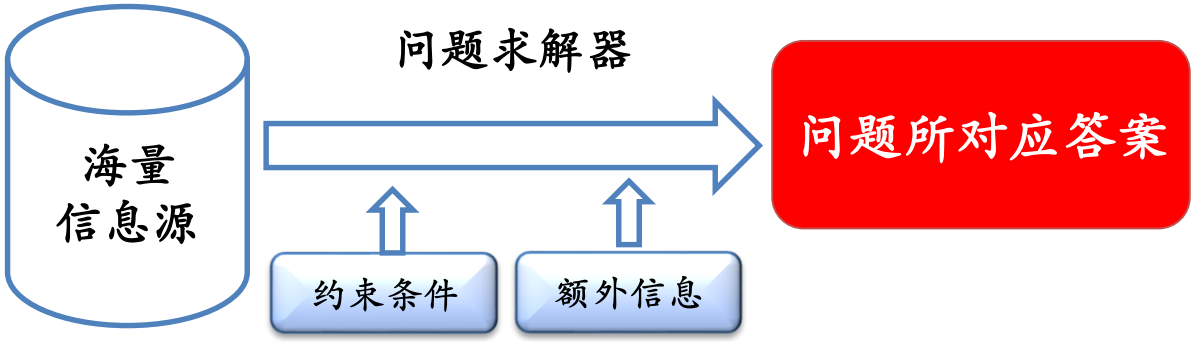
人工智能中的搜索

你见，或者不见我

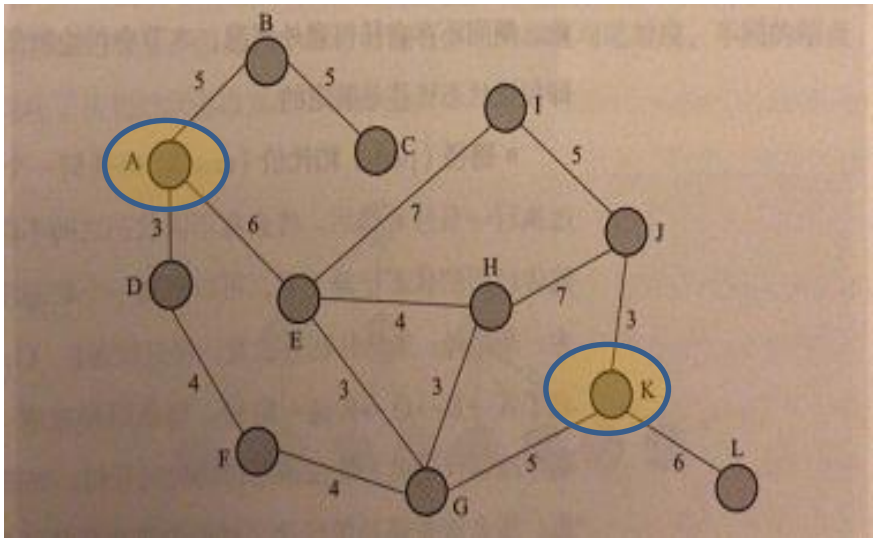
我就在那里

不悲 不喜

---扎西拉姆多多



搜索算法的形式化描述：
〈状态、动作、状态转移、路径/代价、目标测试〉



问题：寻找从城市A到城市K之间行驶时间最短路线？

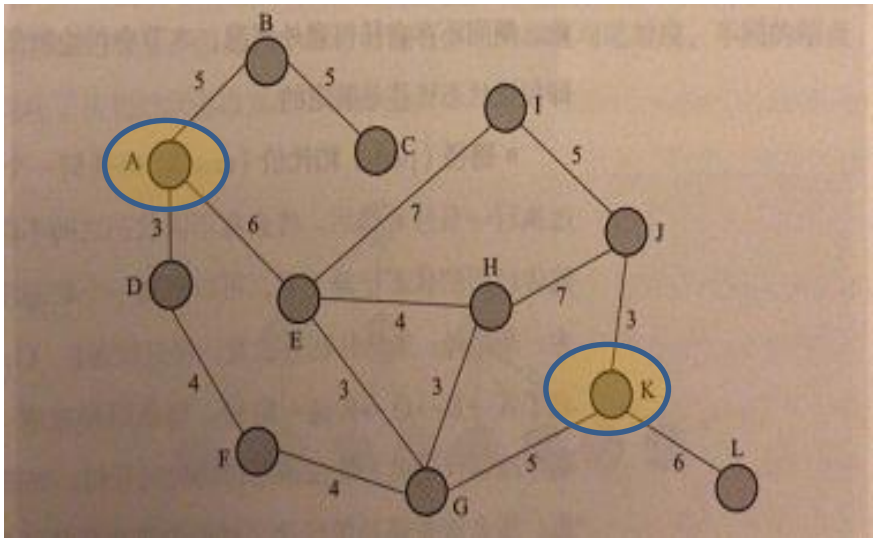
状态

对智能体和环境当前情形的描述。例如，在最短路径问题中，可作为状态。将原问题城市对应的状态称为初始状态。

动作

从当前时刻所处状态转移到下一时刻所处状态所进行操作。一般而言这些操作都是离散的。

搜索算法的形式化描述：
〈状态、动作、状态转移、路径/代价、目标测试〉



问题：寻找从城市A到城市K之间行驶时间最短路线？

状态转移

智能体选择了一个动作之后，其所处状态的相应变化。

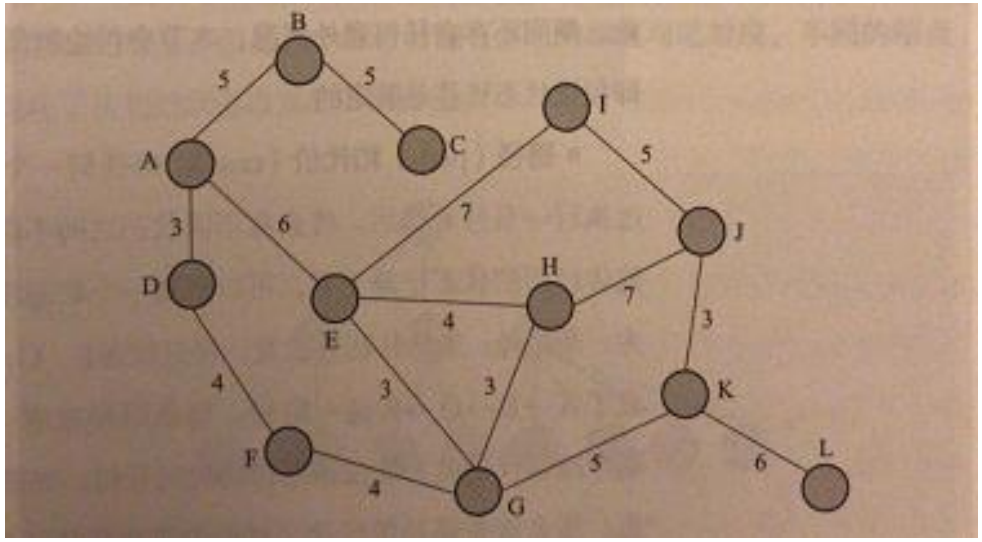
路径/代价

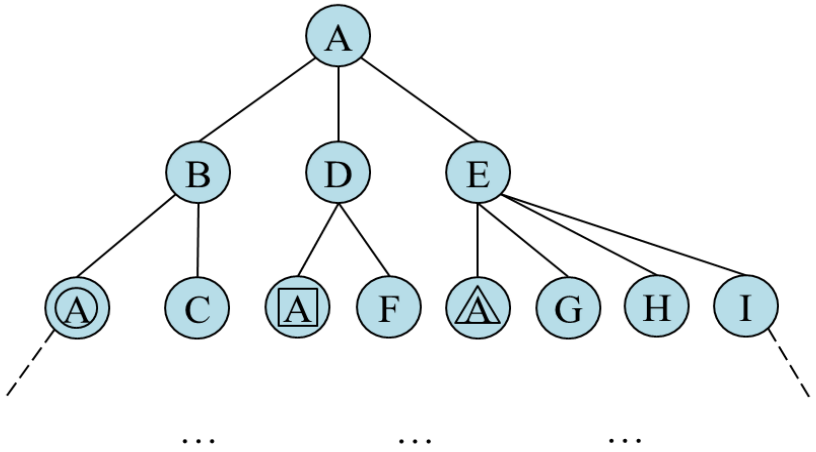
一个状态序列。该状态序列被一系列操作所连接。如从A到K所形成的路径。

目标测试

评估当前状态是否为所求解的目标状态。

搜索树：用一棵树来记录算法探索过的路径





在解决问题的过程中，搜索算法会时刻记录所有从初始结点出发已经探索过的路径，每次从中选出一条，从该路径末尾状态出发进行一次状态转移，探索一条尚未被探索过新路径。

搜索树：用一棵树来记录算法探索过的路径

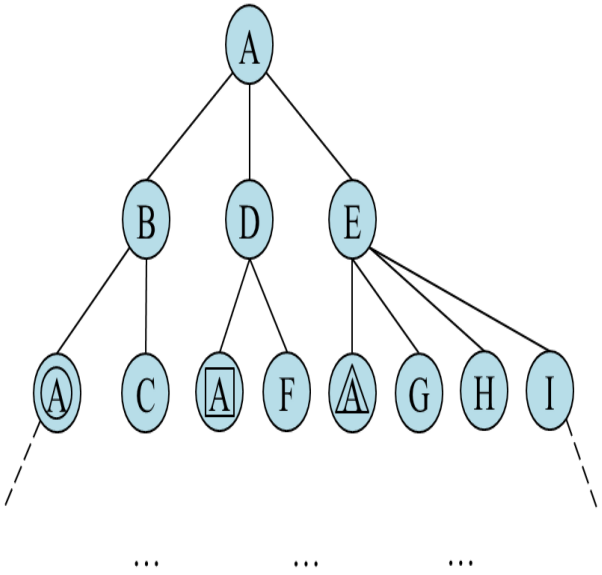


图3.2 搜索树示例（对应图3.1中路径问题。搜索树第三层中标号为*A*的三个节点由于具有从初始结点到达其的不同路径而属于不同结点）

- 搜索树第三层中有三个标号均为A的结点，它们在图中分别被圆圈、正方形和三角形框住。虽然这三个标号均为A的结点对应同一个城市，即三个标号为A的结点所对应状态相同，但是这三个节点在搜索树中却是不同结点，因为它们分别代表了从初始状态出发到达城市 A 的三条不同路径。

- 这三个结点表示的路径分别为：A → B → A、A → D → A和A → E → A。因此需要注意的是，在搜索树中，同一个标号一定表示相同的状态，其含义为智能体当前所在的城市，但是一个标号可能有多个的结点与之对应，不同的结点对应了从初始状态出发的不同路径。

- 搜索算法可以被看成是一个构建搜索树的过程，从根结点（初始状态）开始，不断展开每个结点的后继结点，直到某个结点通过了目标测试。

搜索算法的评价指标

完备性	当问题存在解时，算法是否能保证找到一个解。
最优性	搜索算法是否能保证找到的第一个解是最优解。
时间复杂度	找到一个解所需时间。
空间复杂度	在算法的运行过程中需要消耗的内存量。

完备性和最优性刻画了算法找到解的能力以及所求的解的质量，时间复杂度和空间复杂度衡量了算法的资源消耗，它们通常用O符号（big O notation）来描述。

表 3.1 搜索树中用于估计复杂度的变量含义

符号	含义
b	分支因子，即搜索树中每个节点最大的分支数目
d	根节点到最浅的目标结点的路径长度
m	搜索树中路径的最大可能长度
n	状态空间中状态的数量

搜索算法框架：树搜索

在树搜索算法中，集合$\mathcal{F}$用于保存搜索树中可用于下一步探索的所有候选结点，这个集合被称为**边缘（fringe）集合**，有时也被叫做**开表（open list）**。

函数：TreeSearch

输入：节点选择函数 pick_from，后继节点计算函数 successor_nodes

输出：从初始状态到终止状态的路径

```
1 F ← {根节点}
2 while F ≠ ∅ do
3 |   n ← pick_from(F)
4 |   F ← F − {n}
5 |   if goal_test(n) then
6 |   |   return n.path
7 |   end
8 |   F ← F ∪ successor_nodes(n)
9 end
```

剪枝搜索 – 并不是其所有的后继节点都值得被探索

在某些情况下，主动放弃一些后继结点能够提高搜索效率而不会影响最终搜索结果，甚至能解决无限循环（即算法不停机）问题。

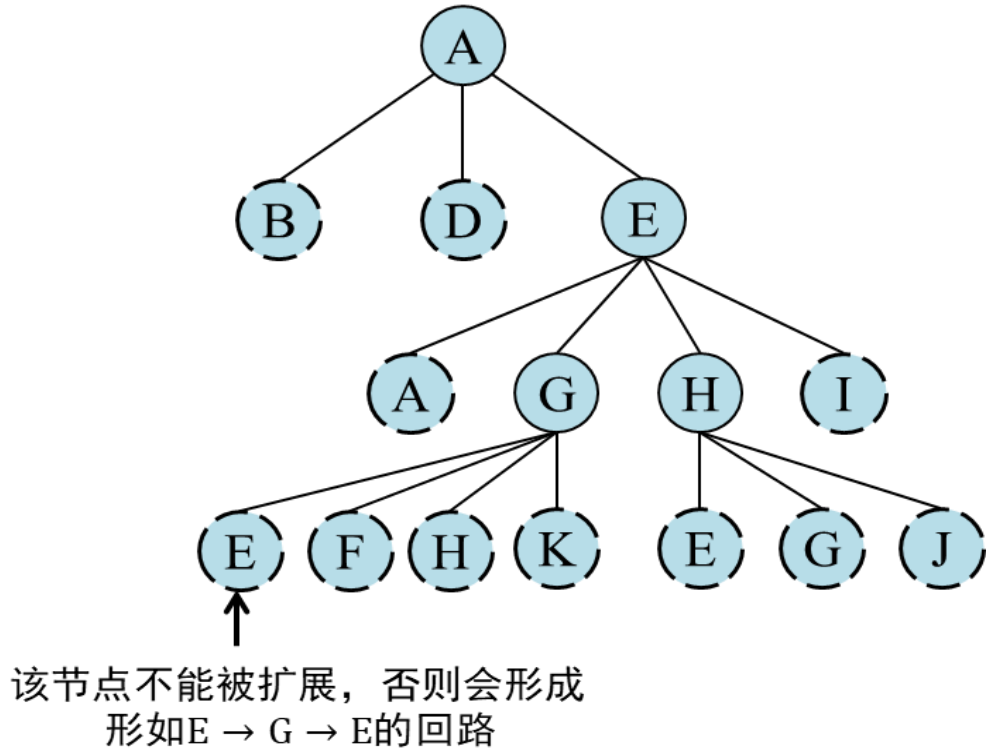


图 3.3 正在构建中的搜索树

注意到图3.3中右侧的路径为A→E→G→E→G→E→…，这意味着在某些搜索策略下（例如深度优先搜索），算法可能会沿着搜索树的右侧路径在状态E和状态G之间陷入无限循环，即出现环路或回路，搜索算法无法终止，此时算法不具有完备性。

图搜索 - 不允许环路的存在

在图搜索策略下，在边缘集合中所有产生环路的节点都要被剪枝，但不会排除所有潜在的可行解。因此在状态数量有限情况下，采用图搜索策略的算法也是完备的。

函数：GraphSearch

输入：节点选择函数 pick_from，后继节点计算函数 successor_nodes

输出：从初始状态到终止状态的路径

```
1  F ← {根节点}
2  C ← ∅
3  while F ≠ ∅ do
4      n ← pick_from(F)
5      F ← F − {n}
6      if goal_test(n) then
7          return n.path
8      end
9      if n.state ∉ C then
10         C ← C ∪ {n.state}
11         F ← F ∪ successor_nodes(n)
12     end
13 end
```

提纲

一、搜索算法基础

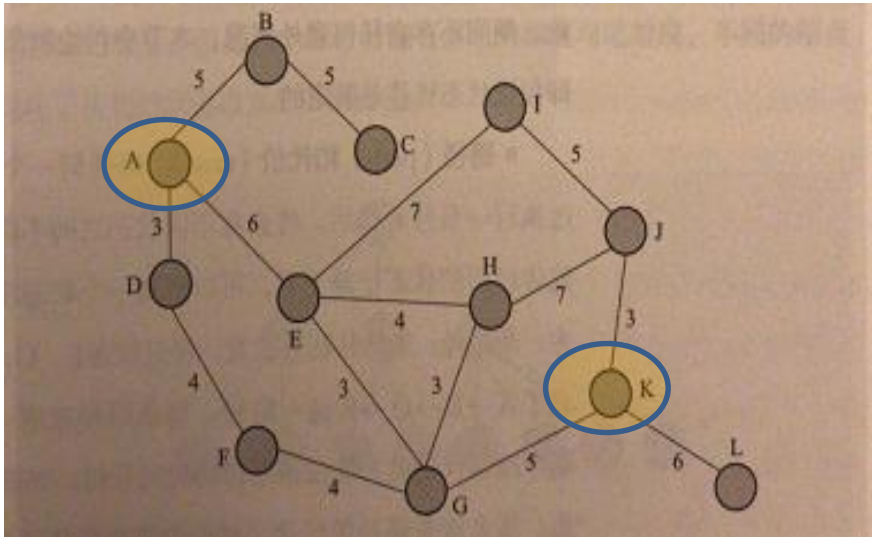
二、启发式搜索

三、对抗搜索

四、蒙特卡洛树搜索

搜索算法：启发式搜索(有信息搜索)

在搜索的过程中利用与所求解问题相关的辅助信息，其代表算法为**贪婪最佳优先搜索**(Greedy best-first search)和**A*搜索**。

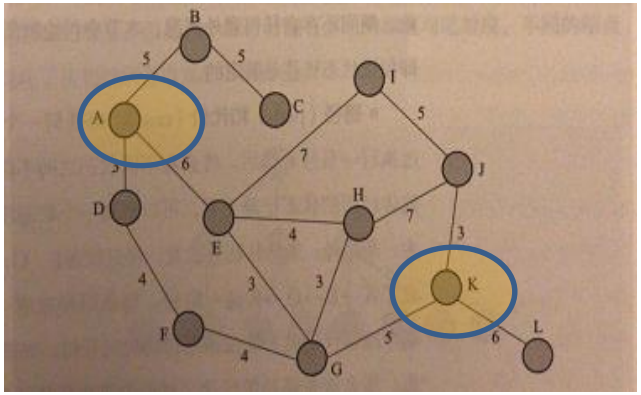


问题：寻找从城市A到城市K之间行驶时间最短路线？

搜索算法：启发函数与评价函数

辅助信息	所求解问题之外、与所求解问题相关的特定信息或知识。	
评价函数（evaluation function）f(n)	从当前节点n出发，根据评价函数来选择后续结点。	下一个结点是谁?
启发函数（heuristic function）h(n)	计算从结点n到目标结点之间所形成路径的最小代价值，这里将两点之间的直线距离作为启发函数。	完成任务还需要多少代价?

贪婪最佳优先搜索(Greedy best-first search)：评价函数f(n)=启发函数h(n)



问题：寻找从城市A到城市K之间行驶时间最短路线?

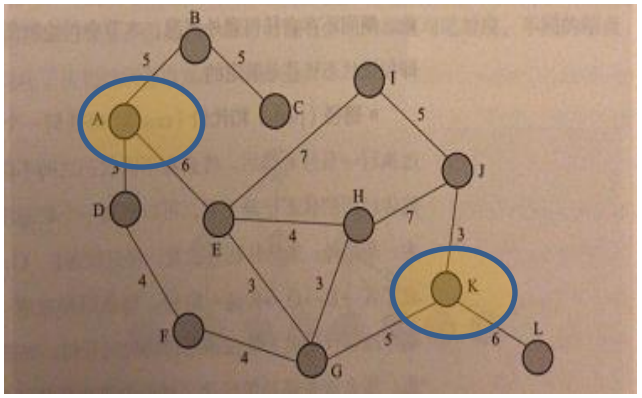
表3.2 每个状态（城市）所对应启发函数取值

状态	A	B	C	D	E	F	G	H	I	J	K	L
h(n)	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息（启发函数）：
任意一个城市与终点城市K之间的直线距离

搜索算法：贪婪最佳优先搜索

贪婪最佳优先搜索(Greedy best-first search)：评价函数f(n)=启发函数h(n)



问题：寻找从城市A到城市K之间行驶时间最短路线？

表3.2 每个状态（城市）所对应启发函数取值

状态	A	B	C	D	E	F	G	H	I	J	K	L
h(n)	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息（启发函数）：
任意一个城市与终点城市K之间的直线距离

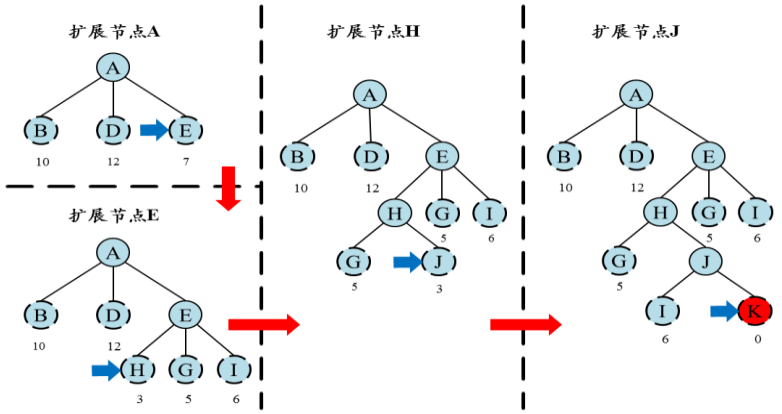


图3.4 贪婪最佳优先搜索的过程

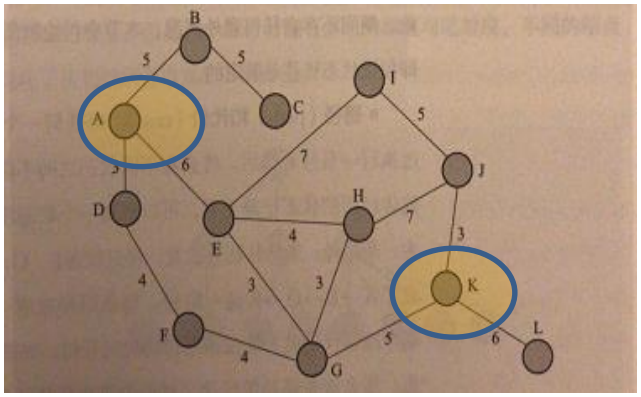
算法找到了一条从起始结点到终点结点的路径A → E → H → J → K，但这条路径并不是最短路径，实际上最短路径为A → E → G → K。

搜索算法：A*算法

评价函数：$f(n) = g(n) + h(n)$

- $g(n)$表示从起始结点到结点n的开销代价值，$h(n)$表示从结点n到目标结点路径中所估算的最小开销代价值。
- $f(n)$可视为经过结点n、具有最小开销代价值的路径。

$$\underbrace{f(n)}_{\text{评价函数}} = \underbrace{g(n)}_{\substack{\text{起始结点到结点}n\text{代价} \\ \text{(当前最小代价)}}} + \underbrace{h(n)}_{\substack{\text{结点}n\text{到目标结点代价} \\ \text{(后续估计最小代价)}}}$$



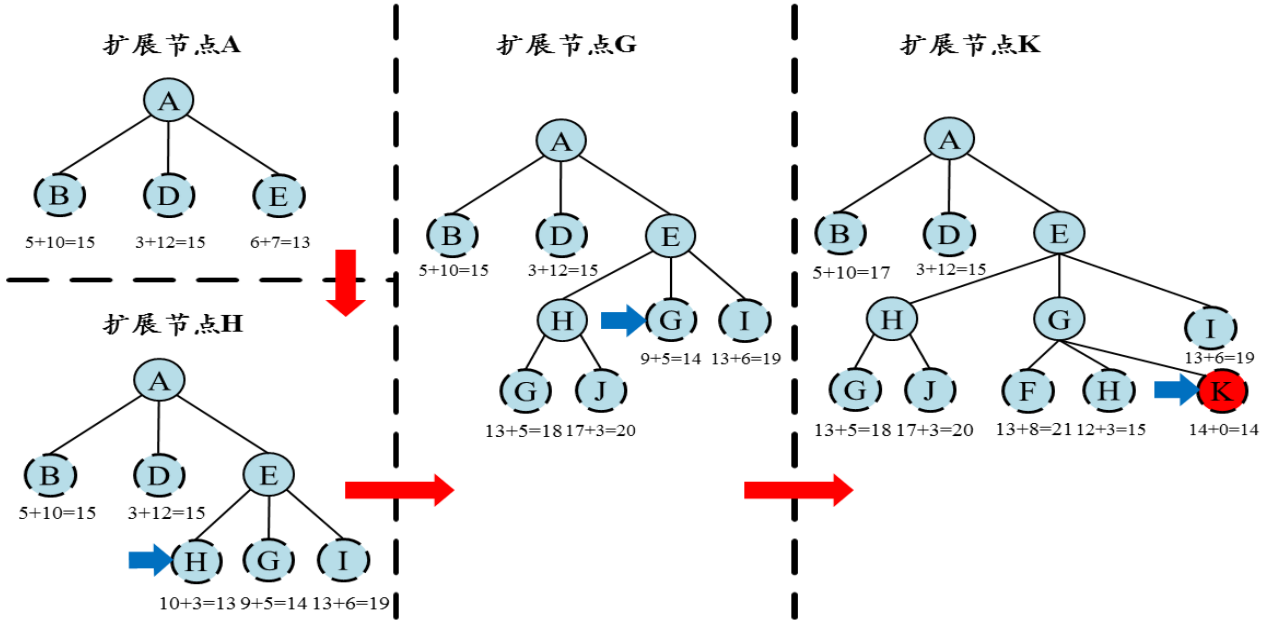
问题：寻找从城市A到城市K之间行驶时间最短路线？

表3.2 每个状态（城市）所对应启发函数取值

状态	A	B	C	D	E	F	G	H	I	J	K	L
h(n)	13	10	6	12	7	8	5	3	6	3	0	6

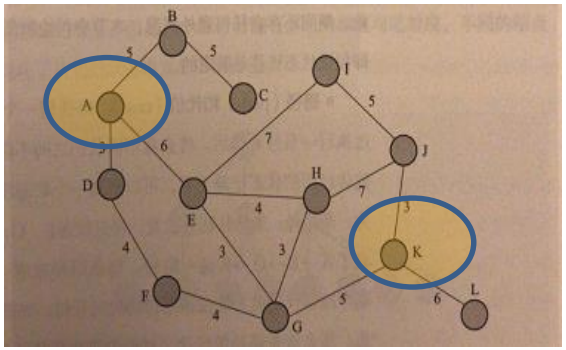
辅助信息：任意一个城市与终点城市K之间的直线距离

搜索算法：A*算法



最短路径A → E → G → K

图3.5 A*算法的搜索过程



问题：寻找从城市A到城市K之间行驶时间最短路线？

表3.2 每个状态（城市）所对应启发函数取值

状态	A	B	C	D	E	F	G	H	I	J	K	L
h(n)	13	10	6	12	7	8	5	3	6	3	0	6

辅助信息：任意一个城市与终点城市K之间的直线距离

搜索算法：A*算法性能分析

A*算法的完备性和最优性取决于搜索问题和启发函数的性质

表 3.3 本节中使用的若干符号

符号	含义
$h(n)$	节点n的启发函数取值
$g(n)$	从起始节点到节点n所对应路径的代价
$f(n)$	节点n的评价函数取值
$c(n,a,n')$	从节点n执行动作a到达节点n'的单步代价
$h^*(n)$	从节点n出发到达终止节点的最小代价

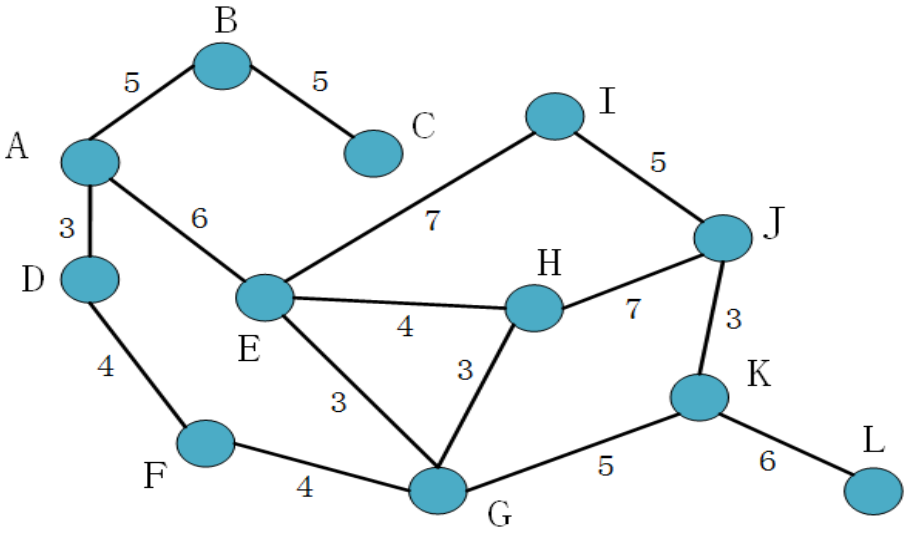
辅助信息	所求解问题之外、与所求解问题相关的特定信息或知识。	
评价函数（evaluation function）f(n)	**从当前结点n出发，根据评价函数来选择后续结点。**	**下一个结点是谁?**
启发函数（heuristic function）h(n)	**计算从结点n到目标结点之间所形成路径的最小代价值，这里将两点之间的直线距离作为启发函数。**	**完成任务还需要多少代价?**

搜索算法：A*算法性能分析

启发函数需要满足如下两种性质：

可容性（admissible）：对于任意结点n，有$h(n) \le h^*(n)$，如果n是目标结点，则有$h(n) = 0$。如表3.3所示，$h^*(n)$是从结点n出发到达终止结点所付出的（实际）最小代价。可以这样理解满足可容性的启发函数，启发函数不会过高估计（over-estimate）从结点n到终止结点所应该付出的代价（即估计代价小于等于实际代价）。

一致性（consistency）：启发函数的一致性指满足条件$h(n) \le c(n, a, n') + h(n')$，这里$c(n, a, n')$表示结点$n$通过动作a到达其相应的后继结点$n'$的代价(三角不等式原则)。



搜索算法：A*算法性能分析

满足一致性条件的启发函数一定满足可容性条件

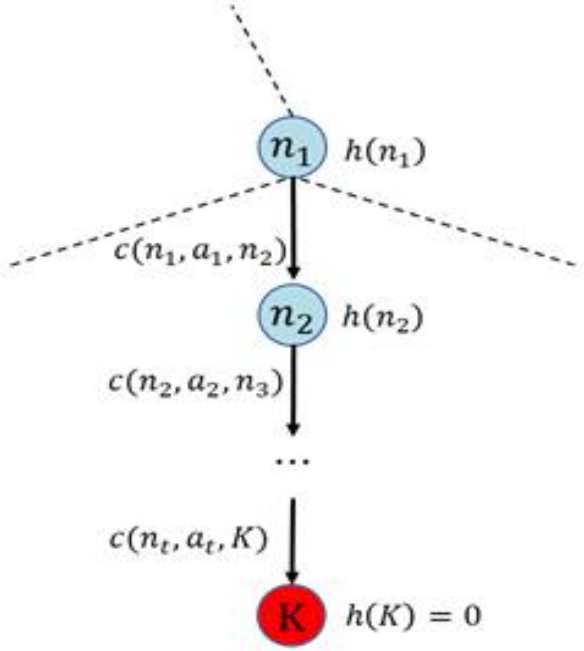


图3.6 从n_1结点到终止结点的最短路径

如图3.6中以n_1为根结点的子树，假设从该节点到达终止结点代价最小的路径为$n_1 \rightarrow n_2 \rightarrow \ldots \rightarrow$ K。由于h(K) = 0且启发函数满足一致性条件，故可以推导可容性条件如下：

$$
\begin{aligned}
h(n_1) &\le h(n_2) + c(n_1, a_1, n_2) \\
&\le h(n_3) + c(n_2, a_2, n_3) + c(n_1, a_1, n_2) \\
&\le \cdots \\
&\le c(n_1, a_1, n_2) + c(n_2, a_2, n_3) + \cdots + c(n_t, a_t, \mathrm{K}) \\
&= h^*(n_1)
\end{aligned}
$$

搜索算法：A*算法的完备性

完备性：如果在起始点和目标点间有路径解存在，那么一定可以得到解，如果得不到解那么一定说明没有解存在

在一些常见的搜索问题中，状态数量是有限的，此时图搜索A*算法和排除环路的树搜索A*均是完备的，即一定能够找到一个解。在更普遍的情况下，如果所求解问题和启发函数满足以下条件，则A*算法是完备的：

- 搜索树中分支数量是有限的，即每个节点的后继结点数量是有限的。
- 单步代价的下界是一个正数。
- 启发函数有下界。

搜索算法：A*算法的最优性

树搜索A*算法的最优性

如果启发函数是可容的，那么树搜索的A*算法满足最优性

证明：启发函数满足可容性时，假设树搜索的A*算法找到的第一个终止结点为n。对于此时边缘集合中任意结点n'，根据算法每次扩展时的策略，即选择评价函数取值最小的边缘节点，有$f(n) \leq f(n')$。

由于A*算法对评价函数定义为$f(n) = g(n) + h(n)$，且$h(n) = 0$，有$f(n) = g(n) + h(n) = g(n)$，$f(n') = g(n') + h(n')$，综合可得$g(n) \leq g(n') + h(n') \leq g(n') + h^*(n')$。

此时扩展其他任何一个边缘结点都不可能找到比结点n所对应路径代价更小的路径，因此结点n对应的是一条最短路径，即算法满足最优性。

搜索算法：A*算法性能分析

图搜索A*算法的最优性

如果A*算法采用的是图搜索策略，那么即使启发函数满足可容性条件，也不能保证算法最优性。

例子：
如果有多个结点对应同一个状态（即存在多条到达同一个城市的路径），图搜索算法只会扩展其中的一个结点，而剪枝其余的结点，然而最短路径有可能经过其中某些被剪枝结点，导致算法无法找到这些最短路径。

表 3.2 每个状态（城市）所对应启发函数取值

状态	A	B	C	D	E	F	G	H	I	J	K	L
$h(n)$	13	10	6	**12→0**	7	**8→0**	**5→0**	3	6	3	0	6

此时启发函数仍然满足可容性，但图搜索的A*算法会优先扩展结点A → D → F → G，而放弃最短路径经过的节点A → E → G。

搜索算法：A*算法性能分析

图搜索A*算法满足最优性（方法一）

当算法从不同路径扩展同一结点时，不保留最先探索的那条路径，而是保留代价最小的那条路径。

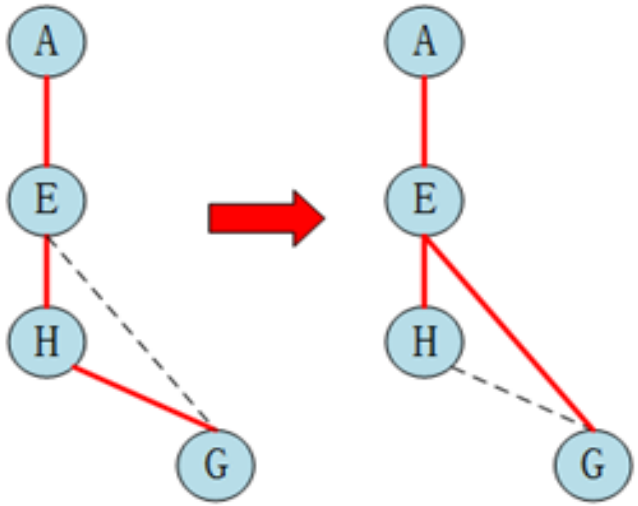


图3.7 修改后图搜索A*算法扩展A→E→G结点，红色实线表示当前搜索树中的边，虚线表示不在搜索树中的边

搜索算法：A*算法性能分析

图搜索A*算法满足最优性（方法二）：

要求启发函数满足一致性

引理3.1： 启发函数满足一致性条件时，给定一个从搜索树中得到的结点序列，每个结点是前一个结点的后继，那么评价函数对这个序列中的结点取值按照结点出现的先后次序而依次单调非减。

引理3.2： 假设状态t加入搜索树时对应的结点为n，在结点n被加入搜索树后，若对应状态t的任何结点n'出现在边缘集合中，那么必然有$g(n) \leq g(n')$。

算法的最优性：对于任意一个状态t，它第一次被加入搜索树时的路径必然是最短路径。

搜索算法：A*算法性能分析

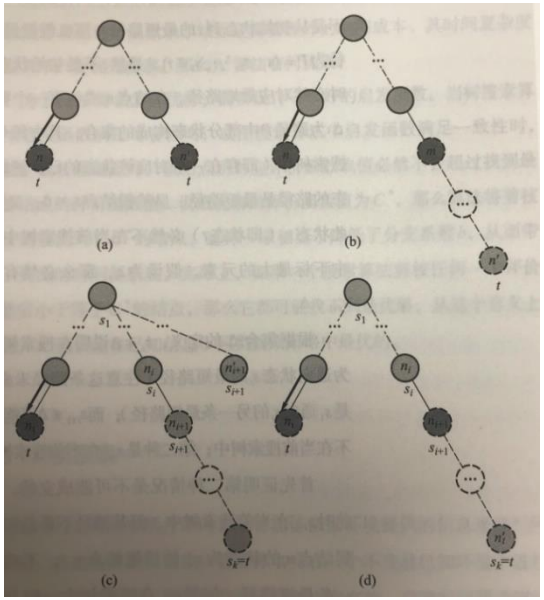
引理3.1： 启发函数满足一致性条件时，给定一个从搜索树中得到的结点序列，每个结点是前一个结点的后继，那么评价函数对这个序列中的节点取值按照节点出现的先后次序而依次单调非减。

考虑序列中结点n，对结点n采取动作a而得到的后继节点n'。由于启发函数满足一致性条件，因此有$h(n) \leq c(n,a,n')+h(n')$。于是有：

$$f(n') = g(n')+h(n') = g(n)+c(n,a,n')+h(n') \geq g(n)+h(n) = f(n)$$

搜索算法：A*算法性能分析

引理3.2： 假设状态t加入搜索树时对应的结点为n，在结点n被加入搜索树后，若对应状态t的任何结点n'出现在边缘集合中，那么必然有$g(n) \leq g(n')$。



这个性质说明，每个状态被加入搜索树时，其路径代价必然小于等于任何将来出现在边缘集合中通向该状态的路径的代价。

图3.8 图搜索A*算法最优性证明过程示意图。其中已在搜索树中的结点用实线表示，不在搜索树中的结点用虚线表示，可扩展的边缘结点用深灰色标出。节点的代号标注在结点内，结点对应的状态标注在相应结点的下方。

搜索算法：A*算法性能分析

最优性：

对于任意一个状态t，它第一次被加入搜索树时的路径必然是最短路径。

为了让A*算法发挥优势，需要设计一个好的启发函数。当树搜索算法中启发函数满足可容性时，或图搜索算法中启发函数满足一致性时，算法扩展任意结点时，该结点所对应评价函数取值必然不会超过找到最优解结点的评价函数值。

提纲

一、搜索算法基础

二、启发式搜索

三、对抗搜索

四、蒙特卡洛树搜索

对抗搜索

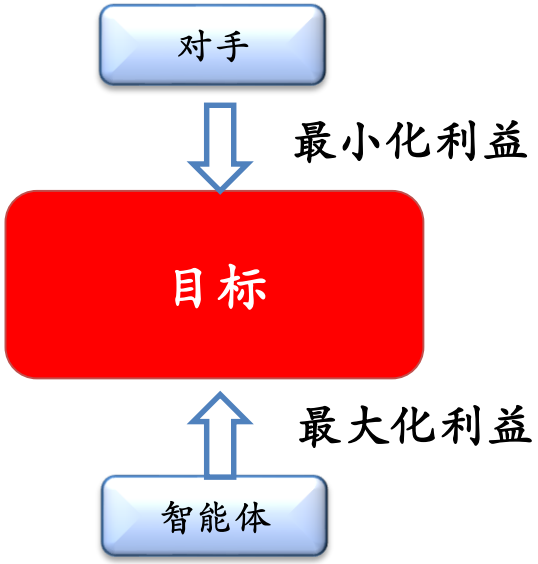
- 对抗搜索(Adversarial Search)也称为博弈搜索(Game Search)
- 在一个竞争的环境中，智能体(agents)之间通过竞争实现相反的利益，一方最大化这个利益，另外一方最小化这个利益。

狭路相逢勇者胜

勇者相逢智者胜

智者相逢德者胜

德者相逢道者胜



对抗搜索

- **最小最大搜索(Minimax Search):** 最小最大搜索是在对抗搜索中最为基本的一种让玩家来计算最优策略的方法.
- **Alpha-Beta剪枝搜索(Pruning Search)**: 一种对最小最大搜索进行改进的算法，即在搜索过程中可剪除无需搜索的分支节点，且不影响搜索结果。.
- **蒙特卡洛树搜索(Monte-Carlo Tree Search)**: 通过采样而非穷举方法来实现搜索。

对抗搜索

- 本书主要讨论在确定的、全局可观察的、竞争对手轮流行动、零和游戏（zero-sum）下的对抗搜索
- 两人对决游戏 (MAX and MIN, MAX先走) 可如下形式化描述，从而将其转换为对抗搜索问题

状态	状态s包括当前的游戏局面和当前行动的智能体。函数$player(s)$给出状态s下行动的智能体。
动作	给定状态s，动作指的是$player(s)$在当前局面下可以采取的操作a，记动作集合为$actions(s)$。
状态转移	给定状态s和动作$a \in actions(s)$，状态转移函数$result(s,a)$决定了在s状态采取a动作后所得后继状态。
终局状态检测	终止状态检测函数terminal_test(s)用于测试游戏是否在状态s结束。
终局得分	终局得分$utility(s,p)$表示在终局状态s时玩家p的得分。

注：所谓零和博弈是博弈论的一个概念，属非合作博弈。指参与博弈的各方，在严格竞争下，一方的收益必然意味着另一方的损失，博弈各方的收益和损失相加总和永远为“零”，双方不存在合作的可能。与“零和”对应，“双赢博弈”的基本理论就是“利己”不“损人”，通过谈判、合作达到皆大欢喜的结果。

对抗搜索：最小最大搜索

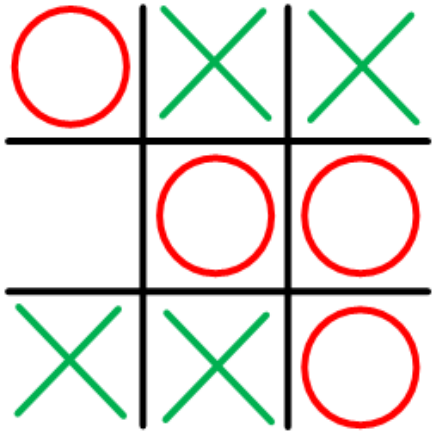
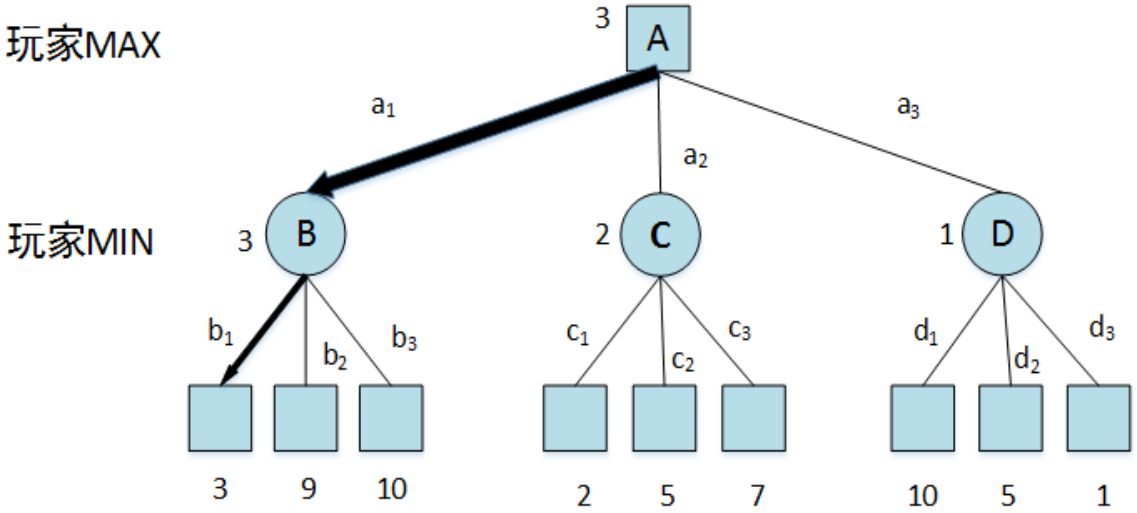


图 3.9 井字棋的盘面，此时执“〇”棋子的玩家获胜



$$\text{minimax}(s) = \begin{cases} \text{utility}(s), & \text{if terminal\_test}(s) \\ \max_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s,a)), & \text{if player}(s) = MAX \\ \min_{a \in \text{actions}(s)} \text{minimax}(\text{result}(s,a)), & \text{if player}(s) = MIN \end{cases}$$

对抗搜索：最小最大搜索

函数：MinimaxDecision

输入：当前的盘面状态s

输出：玩家 MAX 行动下，当前最优动作a^*

```
1 a* ← arg max_{a∈action(s)} MinValue(result(s, a))
```

函数：MaxValue

输入：当前的盘面状态s

输出：玩家 MAX 行动下，当前状态的得分$v = \text{minimax(s, MAX)}$

```
1 if terminal_test(s) then return utility(s)
2 v ← −∞
3 foreach a ∈ action(s) do
4 |   v ← max(v, MinValue(result(s, a)))
5 end
```

函数：MinValue

输入：当前的盘面状态s

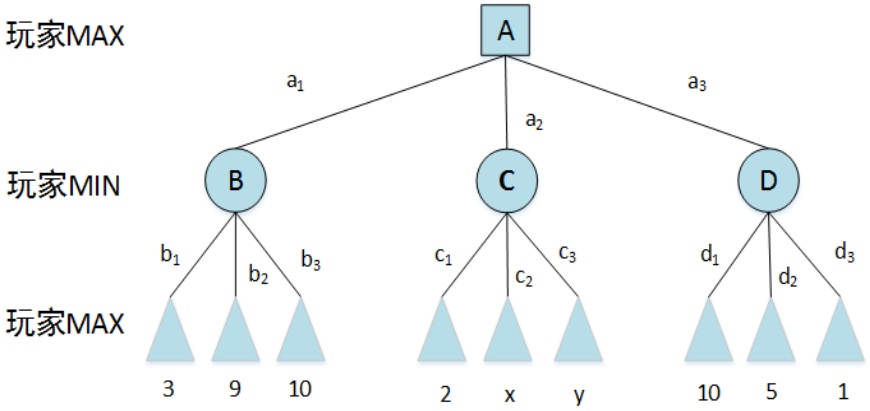
输出：玩家 MIN 行动下，当前状态的得分$v = \text{minimax(s, MIN)}$

```
1 if terminal_test(s) then return utility(s)
2 v ← +∞
3 foreach a ∈ action(s) do
4 |   v ← min(v, MaxValue(result(s, a)))
5 end
```

对抗搜索：Alpha-Beta 剪枝搜索

Alpha-Beta 剪枝搜索算法在Minimax算法中可减少被搜索的结点数，即在保证得到与原Minimax算法同样的搜索结果时，剪去了不影响最终结果的搜索分枝。

$$\begin{aligned}&\text{minimax}(A)\\&= \max(\min(3,9,10), \min(2,x,y), \min(10,5,1))\\&= \max(3, \min(2,x,y), 1)\end{aligned}$$



由于$\min(2,x,y)$的值小于等于2，所以即使不知道x和y的值，根结点的分数也可以算出来等于3。也就是说，算法没有必要计算动作c_2和c_3对应的两个子树在游戏结束时所得分数，就能决定在根结点采取动作a_1从而在游戏结束时可获得收益3，因此动作c_2和c_3所对应子树可以被剪枝。

图中MIN选手所在的结点C下属分支4和6与根结点最终优化决策的取值无关，可不被访问。

对抗搜索：Alpha-Beta 剪枝搜索

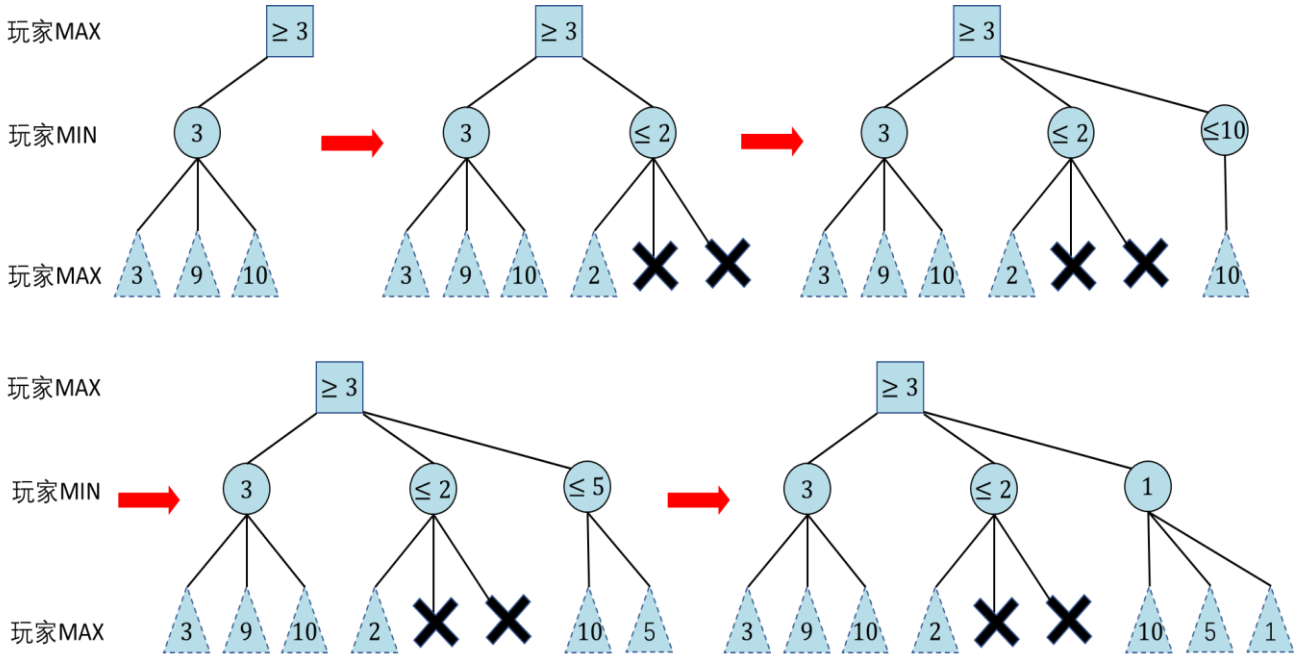


图 3.12 存在剪枝的搜索树部分扩展过程

对抗搜索：Alpha-Beta 剪枝搜索

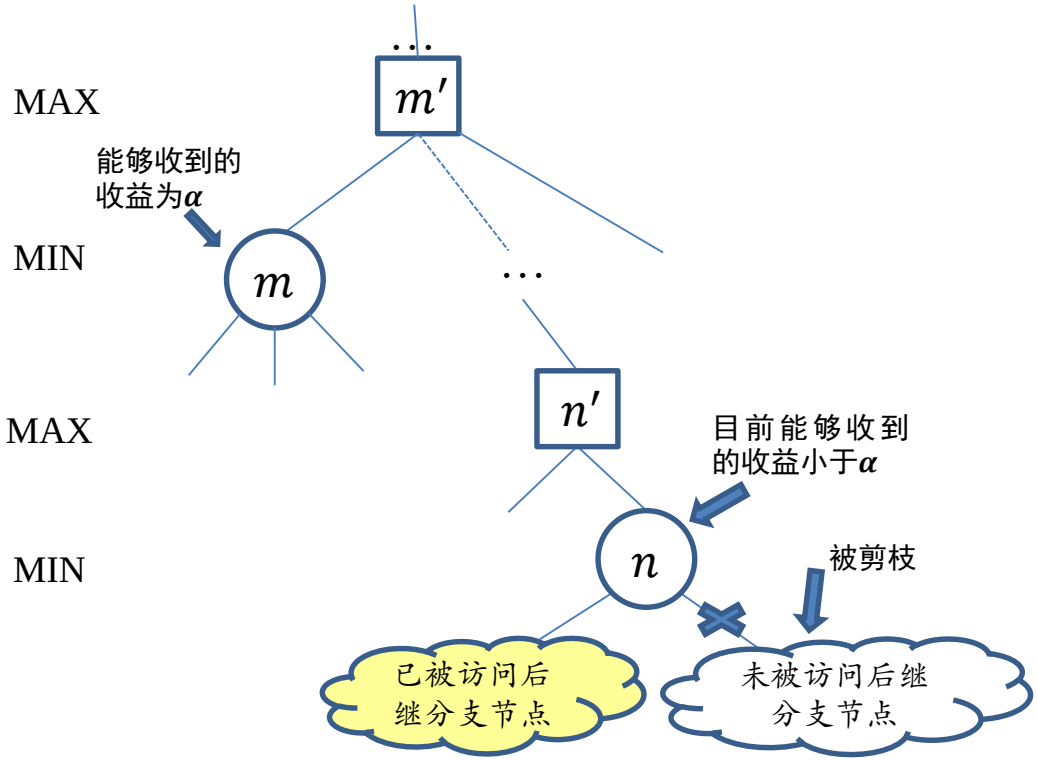


图3.14 基于MIN结点反馈收益进行剪枝(alpha剪枝)

假设有一个位于MIN层的结点m，已知该结点能够向其上MAX结点反馈的收益为α（alpha）。n是与结点m位于同一层的某个兄弟（sibling）结点的后代结点。如果在结点n的后代结点被访问一部分后，知道结点n能够向其上一层MAX结点反馈收益小于α，则结点n的未被访问孩子结点将被剪枝。

对抗搜索：Alpha-Beta 剪枝搜索

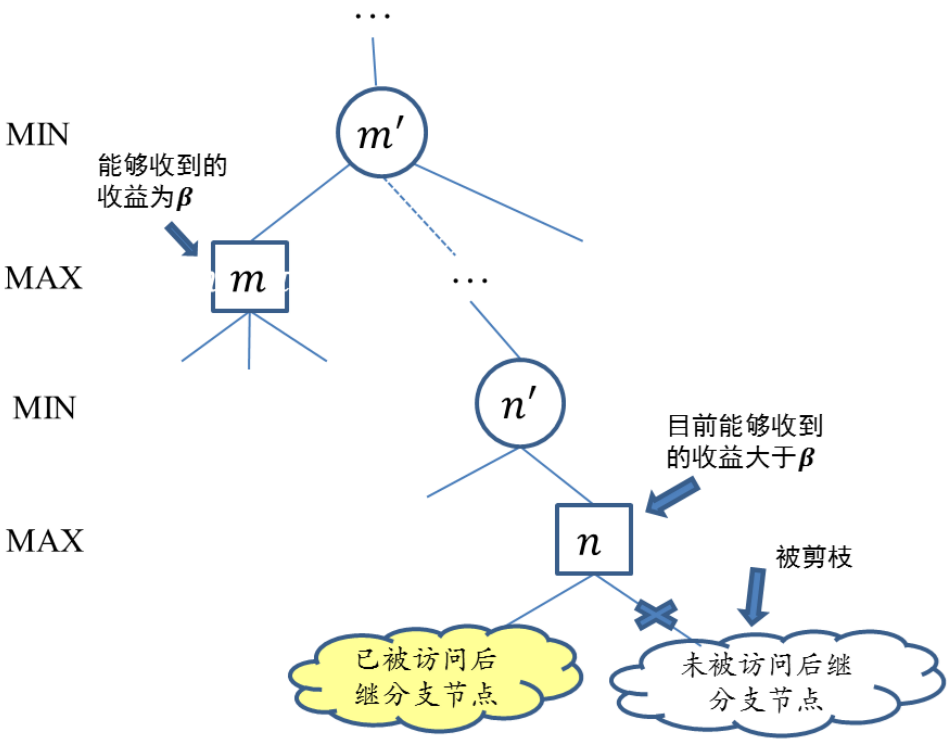


图3.15基于MAX结点反馈收益进行剪枝(beta剪枝)

考虑位于MAX层的结点m，已知结点m能够从其下MIN层结点收到的收益为β（beta）。结点n是结点m上层结点m'的位于MAX层的后代结点，如果目前已知结点n能够收到的收益大于β，则不再扩展结点n的未被访问后继结点，因为位于MIN层的结点m'只会选择收益小于或等于β的结点来采取行动。

对抗搜索：Alpha-Beta 剪枝搜索

- 对于MAX节点，如果其孩子结点（MIN结点）的收益大于当前的α值，则将α值更新为该收益；对于MIN结点，如果其孩子结点（MAX结点）的收益小于当前的β值，则将β值更新为该收益。根结点（MAX结点）的α值和β值分别被初始化为$-\infty$和$+\infty$。
- 随着搜索算法不断被执行，每个结点的α值和β值不断被更新。大体来说，每个结点的$[\alpha, \beta]$从其父结点提供的初始值开始，取值按照如下形式变化：α逐渐增加、β逐渐减少。不难验证，如果一个结点的α值和β值满足$\alpha > \beta$的条件，则该结点尚未被访问的后续结点就会被剪枝，因而不会被智能体访问。

对抗搜索：Alpha-Beta 剪枝搜索

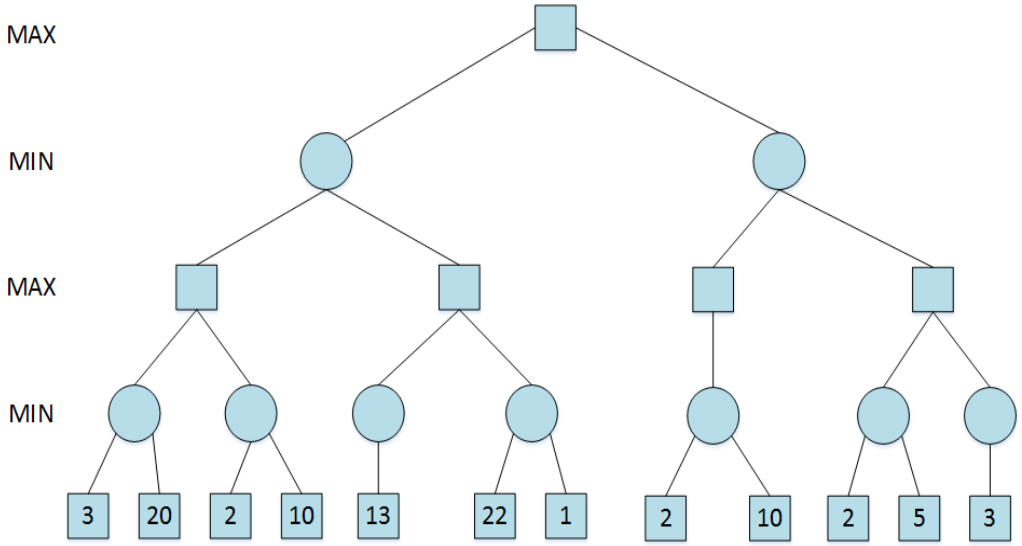


图 3.16 一棵完整的最小最大搜索树

对抗搜索：Alpha-Beta 剪枝搜索

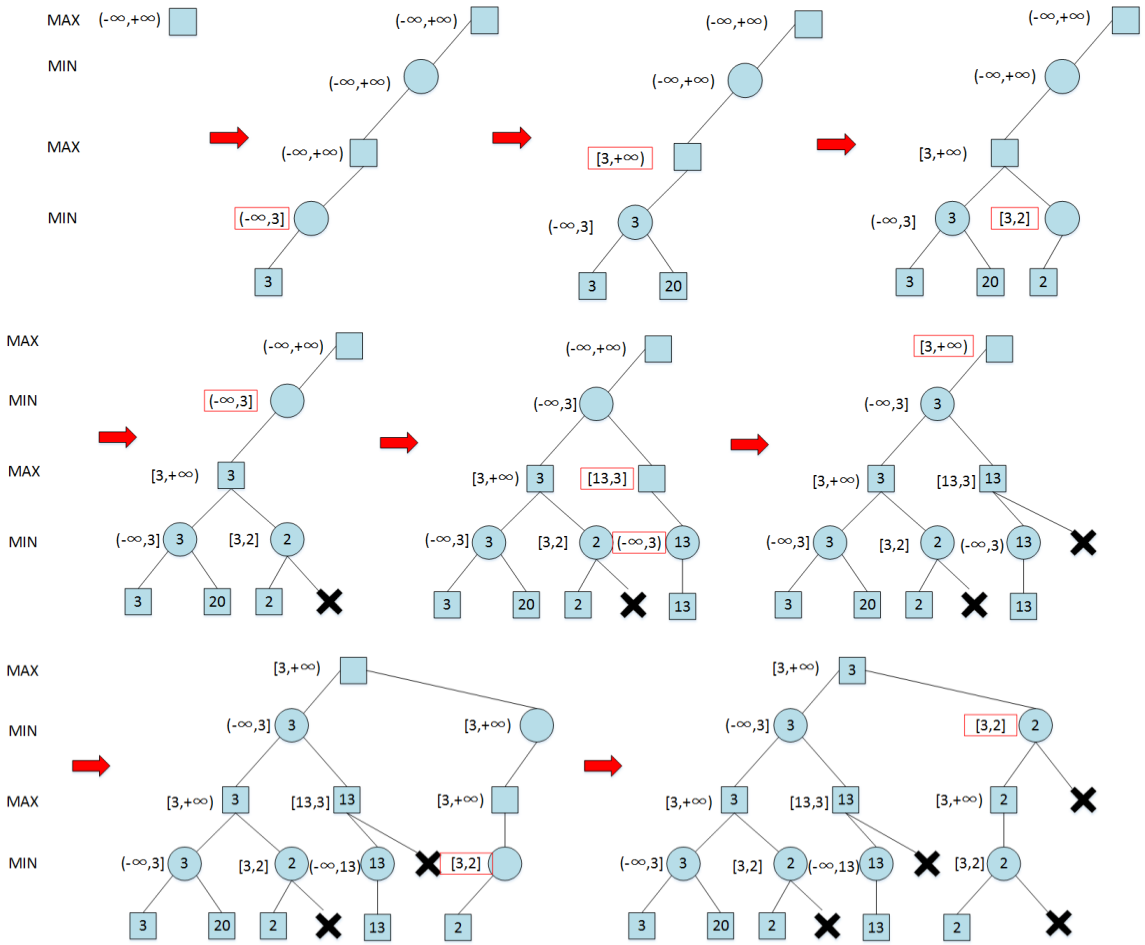


图 3.17 对图3.16中搜索树的alpha-beta剪枝

（原本Minimax搜索树中有26个结点，经过alpha-beta剪枝后只扩展（访问）了其中15个结点，可见alpha-beta剪枝技术能够有效地减少搜索树中结点的数目）

对抗搜索：Alpha-Beta 剪枝搜索

函数：AlphaBetaDecision

输入： 当前的盘面状态s

输出： 玩家 MAX 行动下，当前最优动作a^*

```
1 v, a* ← MaxValue(s, −∞, +∞)
```

函数：MinValue

输入： 当前的盘面状态s，当前节点的下界α和上界β

输出： 玩家 MIN 行动下，当前状态的得分$v = \text{minimax(s, MIN)}$和最优动作$a^*$

前置条件： $\alpha \le \beta$

```
1  if terminal_test(s) then return utility(s), null
2  v ← +∞
3  a* ← null
4  foreach a ∈ action(s) do
5      v', a' ← MaxValue(result(s, a), α, β)
6      if v' < v† then
7          v ← v'
8          a* ← a
9      end
10     β ← min(β, v)
11     if α ≥ β then return v, a*
12 end
```

注： †第 6 行和第 11 行中对分数相同情况的处理方法与正文中不同，此处只保留一个可行解，而放弃所有分数相同的解

对抗搜索：Alpha-Beta 剪枝算法性能分析

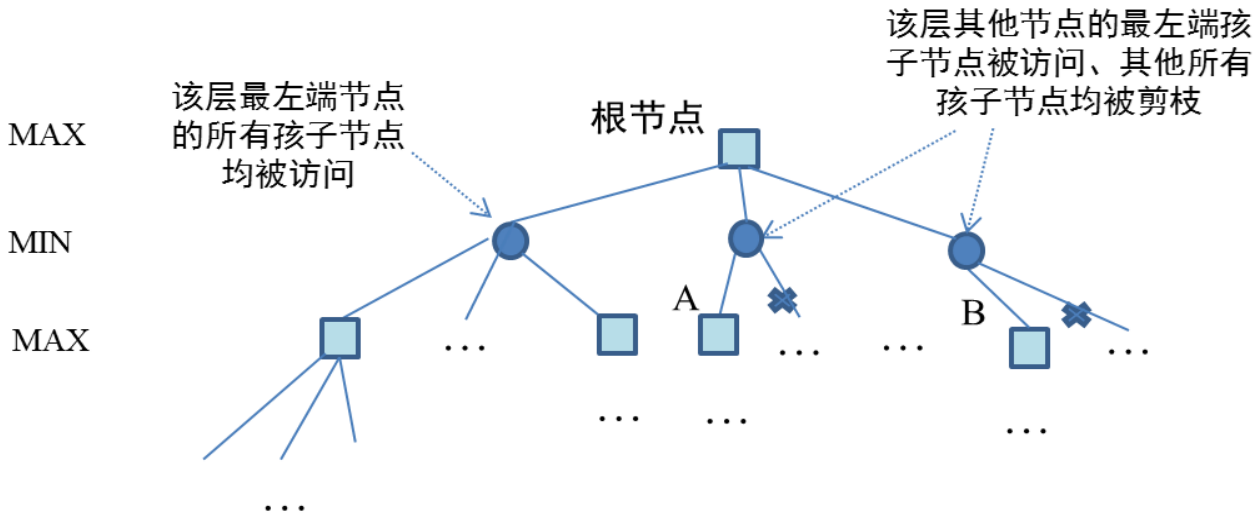


图 3.18 最优状况下的剪枝结果示意图，每一层最左端结点的所有孩子结点均被访问，其他节点仅有最左端孩子结点被访问、其他孩子结点被剪枝。

根据以上分析，似乎除了每层最左端的结点，其余结点都可以只扩展一个孩子结点，这样的算法在最优情况下是否能够达到线性（关于m）的复杂度？当然这是不可能的。

对抗搜索：Alpha-Beta 剪枝算法性能分析

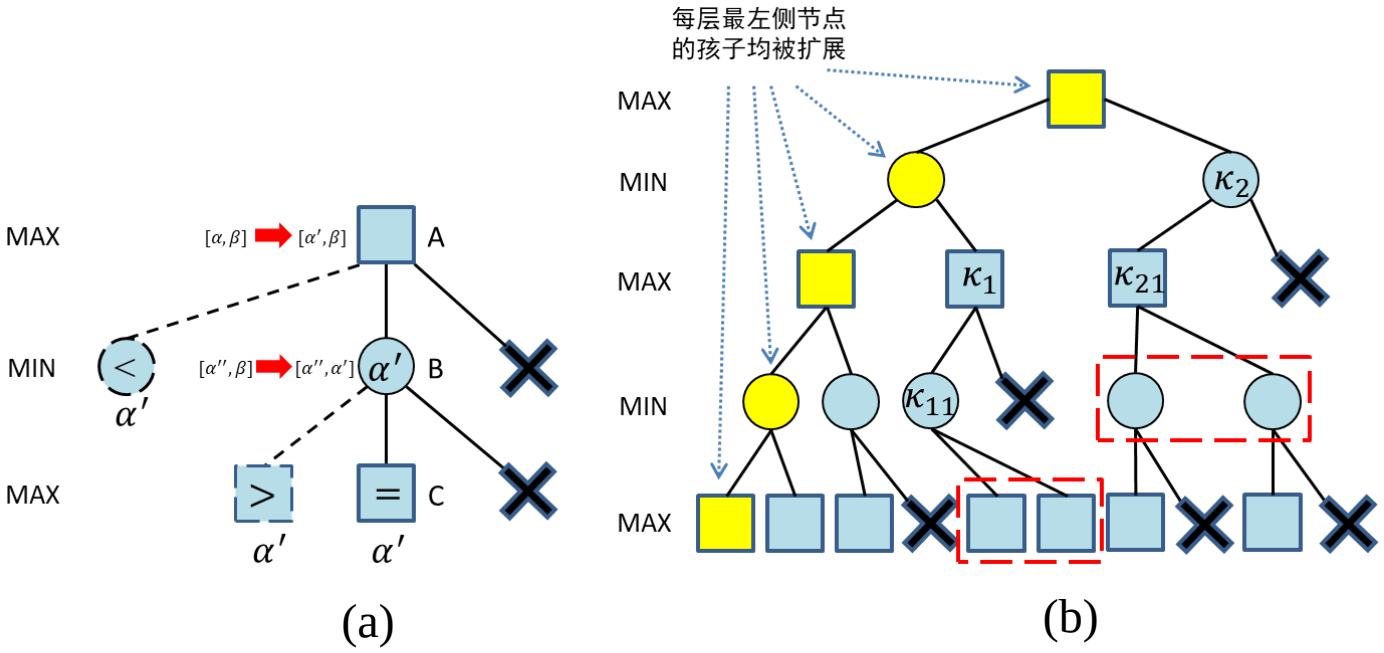


图 3.19 一种较为复杂的剪枝情况分析

- 观察：如果一个节点导致了其兄弟节点被剪枝，可知其孩子节点必然被扩展。
- 如在图3.19（b）中κ_{11}和κ_{21}两个节点分别导致了其右端的兄弟被剪枝，于是可以肯定κ_{11}和κ_{21}全部孩子节点必然被扩展。

对抗搜索：Alpha-Beta 剪枝算法性能分析

下面以图3.19（a）为示意来进行证明：

由于A结点为MAX或MIN节点时证明过程基本相同，不妨假设A结点为MAX节点。假设节点A的初是alpha和beta值为[α, β]，如果A的孩子结点B导致了B右端的兄弟结点被剪枝。由于剪枝直到B的收益分数计算完后才发生，可知结点B必然是A已扩展孩子结点中收益分数最大的结点；同时根据剪枝的发生条件，可知B的收益分数α'必然满足$\alpha' > \beta$。

假设扩展结点B时的上下界为$[\alpha'', \beta]$，其中α''来自结点B被扩展时结点A的下界alpha。根据假设，B结点至少有一个孩子结点（C结点）被扩展，因此$\alpha'' \leq \beta$，同时又根据$\alpha' > \beta$，因此推出$\alpha'' < \alpha'$。如果结点C导致B的其余子结点被剪枝，仿照对结点B的分析，可知结点C是结点B已扩展子结点中收益分数最小的一个。于是，结点B的收益分数必然是结点C的收益分数，即$\text{minimax}(C) = \text{minimax}(B) = \alpha'$，同时C的收益分数必然满足$\alpha' < \alpha''$，该式与前面的不等式矛盾。**由此不难归纳出结论：如果某个结点导致了其右侧兄弟结点被剪枝，那么该节点的所有孩子结点必然已被全部扩展**

对抗搜索：Alpha-Beta 剪枝算法性能分析

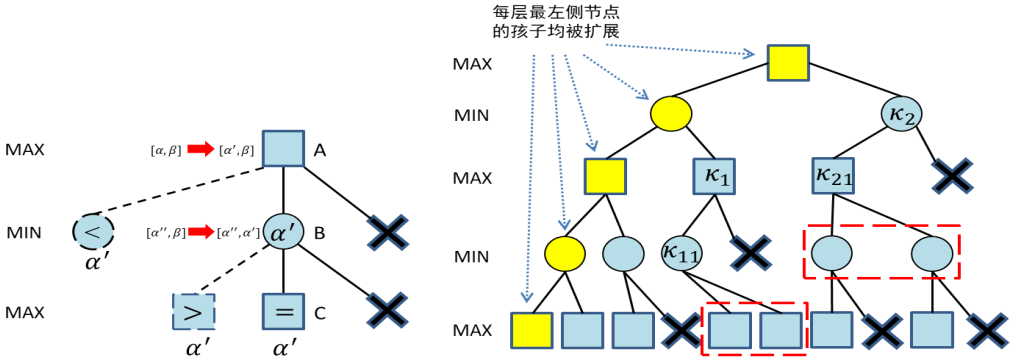


图 3.19 一种较为复杂的剪枝情况分析

图3.19（b）展示了alpha-beta剪枝算法效率最优的情况，为了方便说明，假设图中每个结点恰好有 b 个子节点（此处 b 取值为2）。在图3.19（b）中搜索树每层最左端结点的孩子结点必然全部被扩展 。其他结点的孩子结点被扩展情况分为如下两类：**1）只扩展其最左端孩子结点；2）它是其父亲结点唯一被扩展的孩子结点，且它的所有孩子结点（如果存在）一定被扩展**。如在图3.19（b）中，κ_1和κ_2两个结点属于上述第一类结点，即仅扩展它们的最左端孩子结点。κ_{11}和 κ_{21}两个结点属于上述第二类结点，即它们是其父亲结点唯一被扩展的孩子结点，且它的所有孩子结点一定被扩展。

基于以上分析，不难计算出alpha-beta剪枝在最优效率下扩展的结点数量 为$O(b^{\frac{m}{2}})/O(b^{\frac{m+1}{2}})$，比起原本 最小最大搜索的$O(b^m)$有显著提升。

提纲

一、搜索算法基础

二、启发式搜索

三、对抗搜索

四、蒙特卡洛树搜索

对抗搜索：蒙特卡洛树搜索

先考虑一个简化问题：假设智能体面前有K个赌博机，每个赌博机有一个臂膀。每次转动一个赌博机臂膀，赌博机则会随机吐出一些硬币或不吐出硬币，将所吐出的硬币的币值用收益分数来表示。现在假设给智能体$\tau(\tau > K)$次转动臂膀的机会，那么智能体如何选择赌博机、转动τ次赌博机臂膀，能够获得更多的收益分数呢?

方法：让智能体先把K个赌博机的臂膀依次转动一遍，观察在摇动每个赌博机臂膀时的收益分数，然后去转动那些收益分数高的赌博机臂膀。但是，由于从每个赌博机获得的收益分数是随机的，显然一个刚刚给用户带来可观收益分数的赌博机在下一次摇动其臂膀时就可能不会获得可观收益分数了，因此这一方法不可取。

对抗搜索：蒙特卡洛树搜索

- **状态**。每个被摇动的臂膀即为一个状态，记K个状态分别为$\{s_1, s_2, \cdots, s_K\}$，没有摇动任何臂膀的初始状态记为$s_0$。
- **动作**。动作对应着摇动一个赌博机的臂膀，在多臂赌博机问题中，任意状态下的动作集合都为$\{a_1, a_2 \cdots, a_K\}$，分别对应摇动某个赌博机的臂膀。
- **状态转移**。选择动作$a_i(1 \leq i \leq K)$后，将相应的改变为s_i。

对抗搜索：蒙特卡洛树搜索

- **奖励（reward）**。假设从第i个赌博机获得收益分数的分布为D_i，其均值为μ_i。如果智能体在第t次行动中选择转动了第l_t个赌博机臂膀，那么智能体在第t次行动中所得收益分数$\hat{r}_t$服从分布D_{l_t}，$\hat{r}_t$被称为第t次行动的奖励。为了方便对多臂赌博机问题的理论研究，一般假定奖励是有界的，进一步可假设奖励的取值范围为$[0,1]$。

- **悔值（regret）函数。**根据智能体前T次动作，可以如下定义悔值函数：

$$\rho_T = T\mu^* - \sum_{t=1}^{T} \hat{r}_t \quad (3.4.1)$$

其中$\mu^* = \max_{i=1,\cdots,K} \mu_i$。显然为了尽量减少悔恨，在每次操作时，智能体应该总是转动能够给出最大期望奖励的赌博机臂膀，但是这是不现实的，因为智能体并不知道哪个臂膀的奖励期望最大。公式（3.4.1）告诉我们，将T次操作中最优策略的期望得分减去智能体的实际得分，就是悔值函数的结果。显然，问题求解的目标为最小化悔值函数的期望，该悔值函数的取值取决于智能体所采取的策略。

对抗搜索：蒙特卡洛树搜索

- 贪心算法策略：智能体记录下每次摇动的赌博机臂膀和获得的相应收益分数。给定第$i(1 \le i \le K)$个赌博机，记在过去$t-1$次摇动赌博机臂膀的行动中，一共摇动第i个赌博机臂膀的次数为第$T_{(i,t-1)}$。于是，可以计算得到第i个赌博机在过去$T_{(i,t-1)}$次被摇动过程中的收益分数平均值$\bar{x}_{i,T_{(i,t-1)}}$。这样，智能体在第t步，只要选择$\bar{x}_{i,T_{(i,t-1)}}$值最大的赌博机臂膀进行摇动，这是贪心算法的思路。

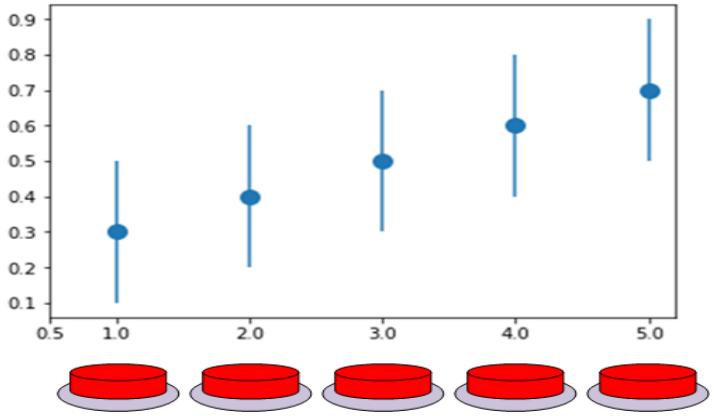
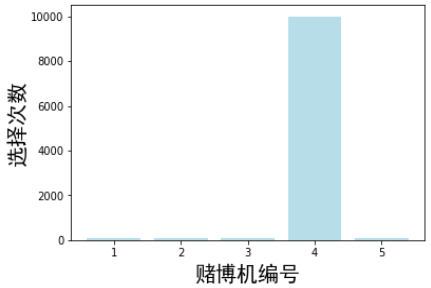
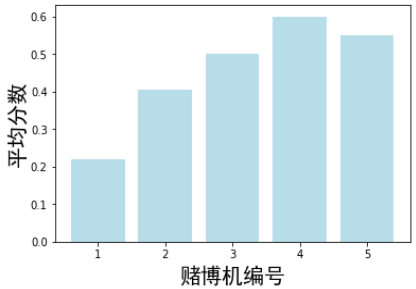


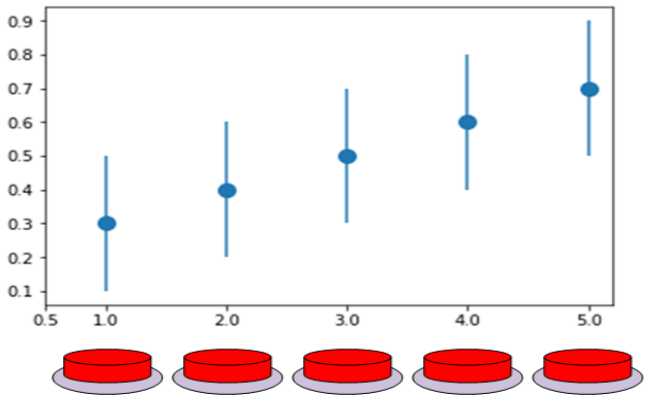
图3.20多臂赌博机问题中每个赌博机收益分数的分布情况，即第i个赌博机的得分满足均值为μ_i、方差为$\frac{1}{75}$（即支撑区间长度为0.4）的均匀分布，其中均值$(\mu_1, \cdots, \mu_5) = (0.3, 0.4, 0.5, 0.6, 0.7)$。

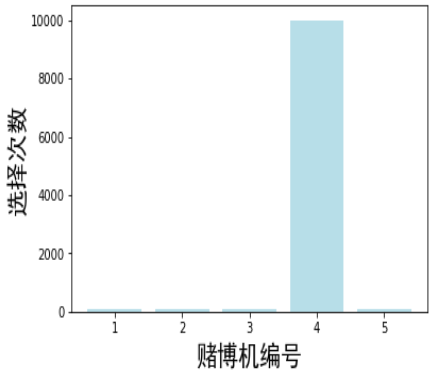


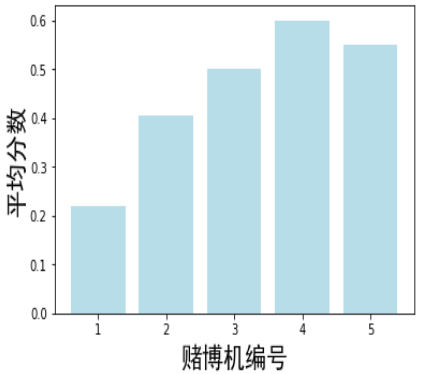


在计算机上模拟执行$T = 10000$次贪心算法，来摇动五台赌博机。模拟的结果如图3.2.1所示。可见在10000次模拟中，智能体趋向于摇动4号赌博机，其他编号赌博机被摇动次数很少。由于其他编号赌博机被摇动次数较少，可见对这些赌博机平均收益分数的估计结果与实际均值结果相差很大。

对抗搜索：蒙特卡洛树搜索







- 不足：忽略了其他从未摇动或很少摇动的赌博机，而失去了可能的机会。**智能体错误地认为5号赌博机不如4号赌博机，因而无法做出更好的选择**

- 上述困境体现了**探索**（exploration）和**利用**（exploitation）之间存在对立关系。贪心算法基本上是**利用**从已有尝试结果中所得估计来指导后续动作，但问题是所得估计往往不能准确反映未被（大量）探索过的动作。因此，需要在贪心算法中增加一个能够改变其“惯性”的内在动力，以使得贪心算法能够访问那些尚未被（充分）访问过的空间。

对抗搜索：蒙特卡洛树搜索

ϵ-贪心算法： 在探索与利用之间进行平衡的搜索算法。在第t步，ϵ-贪心算法按照如下机制来选择摇动赌博机：

$$l_t = \begin{cases} \text{argmax}_i \, \bar{x}_{i,T(i,t-1)}, & \text{以} 1-\epsilon \text{的概率} \\ \text{随机的} i \in \{1,2,\cdots,K\}, & \text{以} \epsilon \text{的概率} \end{cases}$$

即以$1-\epsilon$的概率选择在过去$t-1$次摇动赌博机臂膀行动中所得平均收益分数最高的赌博机进行摇动；以ϵ的概率随机选择一个赌博机进行摇动。

不足： 与被探索的次数无关。可能存在一个给出更好奖励期望的动作，但因为智能体对其探索次数少而认为其期望奖励小。因此，需要对那些探索次数少或几乎没有被探索过的动作赋予更高的优先级。没有将每个动作被探索的次数纳入考虑，这似乎是不合理的。另一方面，简单地优先去尝试探索次数少的动作也未必合理，因为只需要少量的探索次数，就能够较好估计方差较小的动作的奖励期望。

对抗搜索：蒙特卡洛树搜索

上限置信区间算法（Upper Confidence Bounds, UCB1）：为每个动作的奖励期望计算一个估计范围，优先采用估计范围上限较高的动作。

动作1的奖励期望取值的不确定度（估计范围）虽然最大，但是因为其均值太小，因此UCB1算法不优先考虑探索动作1。动作2和3的奖励期望的均值相同，但是动作2的奖励期望取值的不确定度（估计范围）更大，于是因为置信上限更大，动作2会被UCB1算法优先考虑。

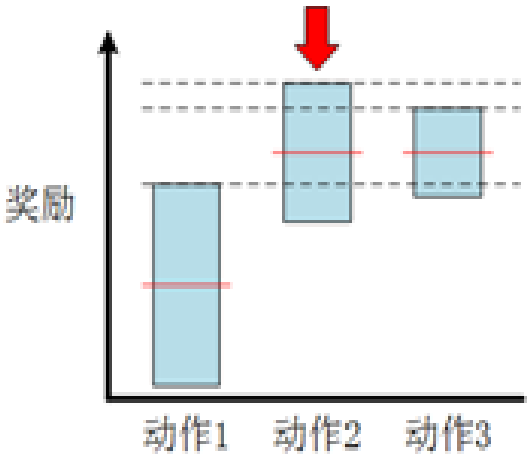


图 3.23 UCB1算法的策略示意图

对抗搜索：蒙特卡洛树搜索

霍夫丁不等式（Hoeffding’s inequality）

假设算法在过去第t次已经对动作a_i探索了$T_{(i,t-1)}$次，在当前问题中对应摇动了第i个赌博机的臂膀$T_{(i,t-1)}$次，执行动作a_i所收到收益分数的均值为$\bar{x}_{i,T_{(i,t-1)}}$。这$T_{(i,t-1)}$次动作可以看作$T_{(i,t-1)}$个取值范围在[0,1]的独立同分布随机变量的样本，根据霍夫丁不等式（Hoeffding’s inequality），有

$$P\left(\mu_i - \bar{x}_{i,T_{(i,t-1)}} > \delta\right) \leq e^{-2T_{(i,t-1)}\delta^2} \qquad (3.4)$$

这个不等式的含义是第i个动作的奖励期望μ_i大于$\bar{x}_{i,T_{(i,t-1)}} + \delta$之和的概率不超过$e^{-2T_{(i,t-1)}\delta^2}$，其中$\delta > 0$。因此只需找到一个$\delta$，使得不等式（3.4）右侧足够小，即可认为$\bar{x}_{i,T_{(i,t-1)}} + \delta$是$\mu_i$的一个上界。

这里可找到一个随时间增长快速趋近于0的函数t^{-4}，令$e^{-2T_{(i,t-1)}\delta^2} = t^{-4}$，则$\delta = \sqrt{\frac{2\ln t}{T_{(i,t-1)}}}$，因此$\mu_i$的上界为$\bar{x}_{i,T_{(i,t-1)}} + \sqrt{\frac{2\ln t}{T_{(i,t-1)}}}$。当然，要选择一个随着$t$取值增大而收敛到0的函数，未必一定要选$t^{-4}$，如果换成关于$t$的其它幂函数，结果将会和当前$\delta$取值相差一个常数系数，因此奖励期望的上界也可以写成$\bar{x}_{i,T_{(i,t-1)}} + C\sqrt{\frac{2\ln t}{T_{(i,t-1)}}}$，其中$C$是一个预先指定的超参数，可以理解为用来调节探索和利用两者权重的因子。

对抗搜索：蒙特卡洛树搜索

上限置信区间算法（Upper Confidence Bounds, UCB1）：为每个动作的奖励期望计算一个估计范围，优先采用估计范围上限较高的动作。

UCB1算法的策略可以描述为，在第t次时选择使得（3.5）式子取值最大的动作a_{l_t}，其中l_t由如下式子计算得到：

$$l_t = \text{argmax}_i \, \bar{x}_{i,T_{(i,t-1)}} + C\sqrt{\frac{2\ln t}{T_{(i,t-1)}}} \qquad (3.5)$$

在过去第t次已经对动作a_i探索了$T_{(i,t-1)}$次，

在当前问题中对应摇动了第i个赌博机的臂膀

$T_{(i,t-1)}$次，执行动作a_i所收到收益分数的均值

为$\bar{x}_{i,T_{(i,t-1)}}$

对抗搜索：蒙特卡洛树搜索

对搜索算法进行优化以提高搜索效率基本上是在解决如下两个问题：**优先扩展哪些节点以及放弃扩展哪些节点**，综合来看也可以概括为如何高效地扩展搜索树。如果将目标稍微降低，改为求解一个近似最优解，则上述问题可以看成是如下探索性问题：算法从根节点开始，每一步动作为选择（在非叶子节点）或扩展（在叶子节点）一个孩子节点。可以用执行该动作后所收获奖励来判断该动作优劣。奖励可以根据从当前节点出发到达目标路径的代价或游戏终局分数来定义。算法会倾向于扩展获得奖励较高的节点。

算法事先不知道每个节点将会得到怎样的代价（或终局分数）分布，只能通过采样式探索来得到计算奖励的样本。**由于这个算法利用蒙特卡洛法通过采样来估计每个动作优劣，因此它被称为蒙特卡洛树搜索（Monte-Carlo Tree Search）算法[Kocsis 2006]。**

对抗搜索：蒙特卡洛树搜索

- **选择（selection）**：选择指算法从搜索树的根节点开始，向下递归选择子节点，直至到达叶子节点或者到达具有还未被扩展过的子节点的节点L。这个向下递归选择过程可由UCB1算法来实现，在递归选择过程中记录下每个节点被选择次数和每个节点得到的奖励均值。
- **扩展（expansion）**：如果节点L不是一个终止节点（或对抗搜索的终局节点），则随机扩展它的一个未被扩展过的后继边缘节点M。
- **模拟（simulation）**：从节点M出发，模拟扩展搜索树，直到找到一个终止节点。模拟过程使用的策略和采用UCB1算法实现的选择过程并不相同，前者通常会使用比较简单的策略，例如使用随机策略。
- **反向传播（Back Propagation）**：用模拟所得结果（终止节点的代价或游戏终局分数）回溯更新模拟路径中M以上（含M）节点的奖励均值和被访问次数。

对抗搜索：蒙特卡洛树搜索

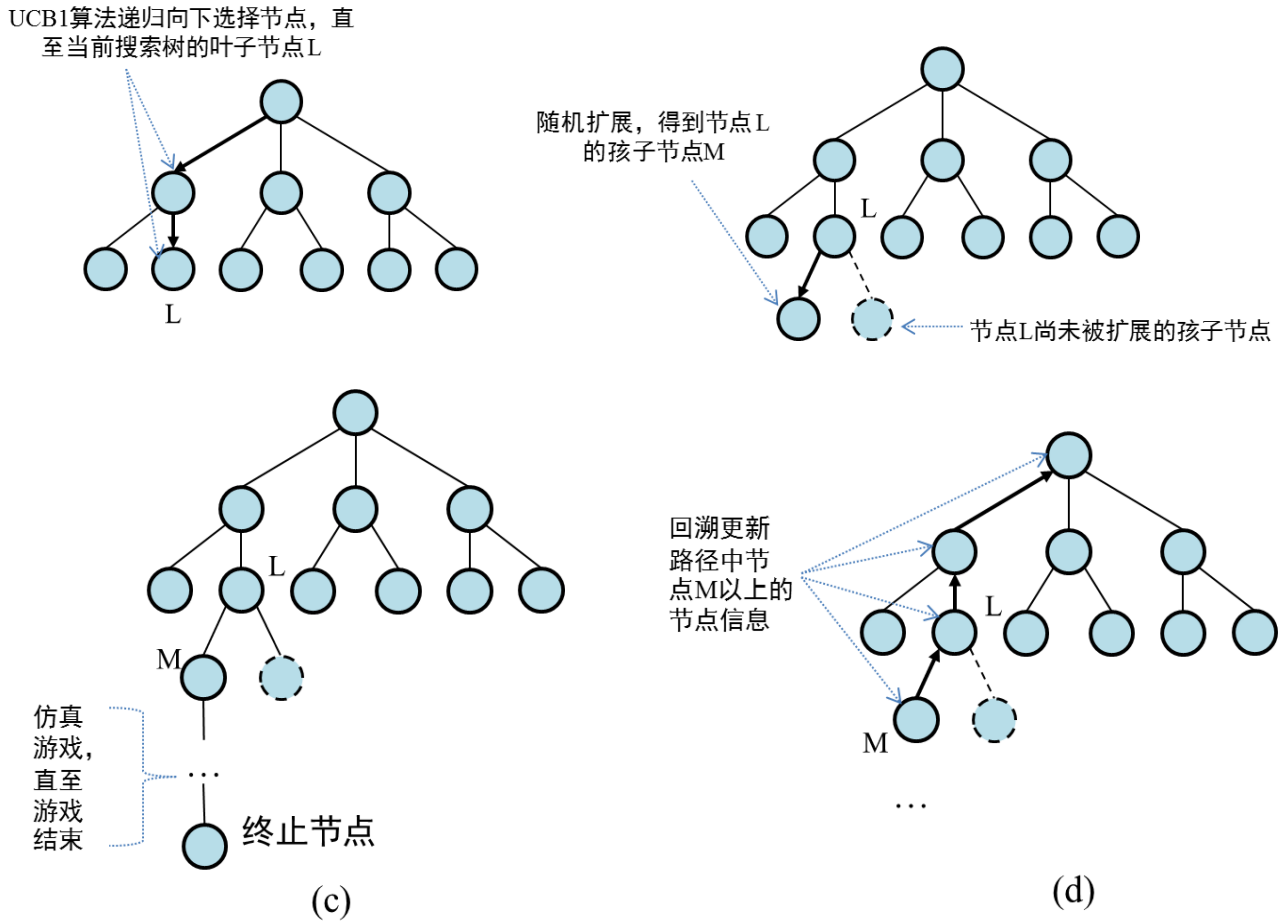


图 3.24 蒙特卡洛树搜索的四个步骤

对抗搜索：蒙特卡洛树搜索

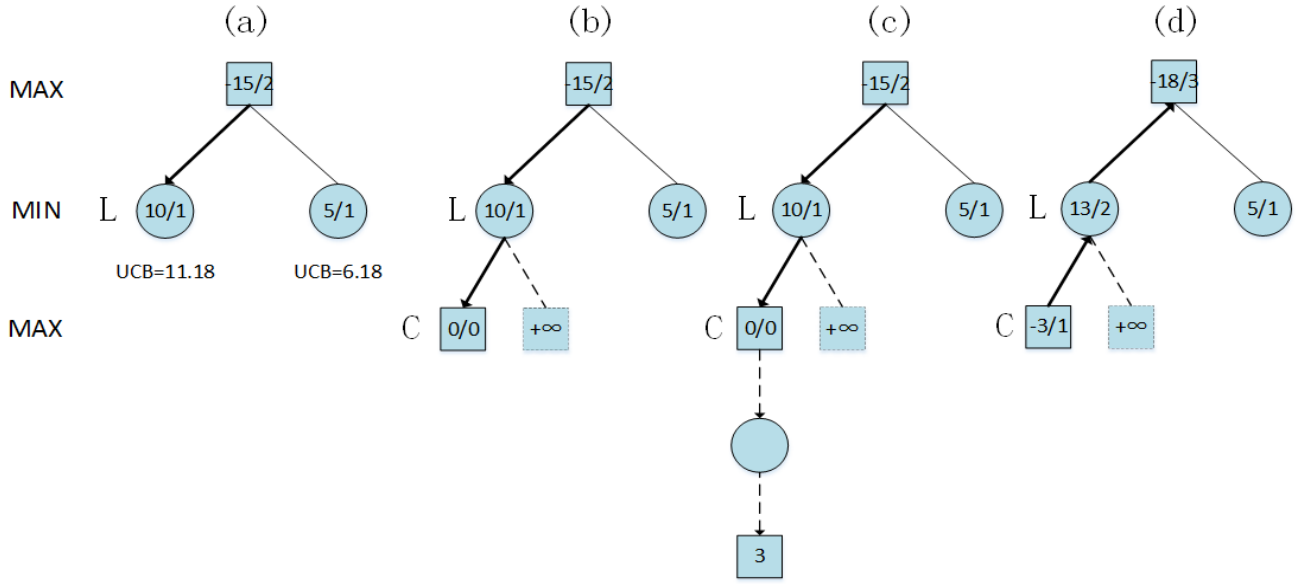


图 3.25 蒙特卡洛树搜索算法。结点中数字代表总收益分数/被访问次数。

计算第二层节点的UCB值：左侧结点为$\frac{10}{1}+\sqrt{\frac{2\ln 2}{1}}=11.18$，右侧结点为$\frac{5}{1}+\sqrt{\frac{2\ln 2}{1}}=6.18$，因此算法选择第二层左侧的结点L，由于该节点有尚未扩展的子结点，因此选择阶段结束。

对抗搜索：蒙特卡洛树搜索

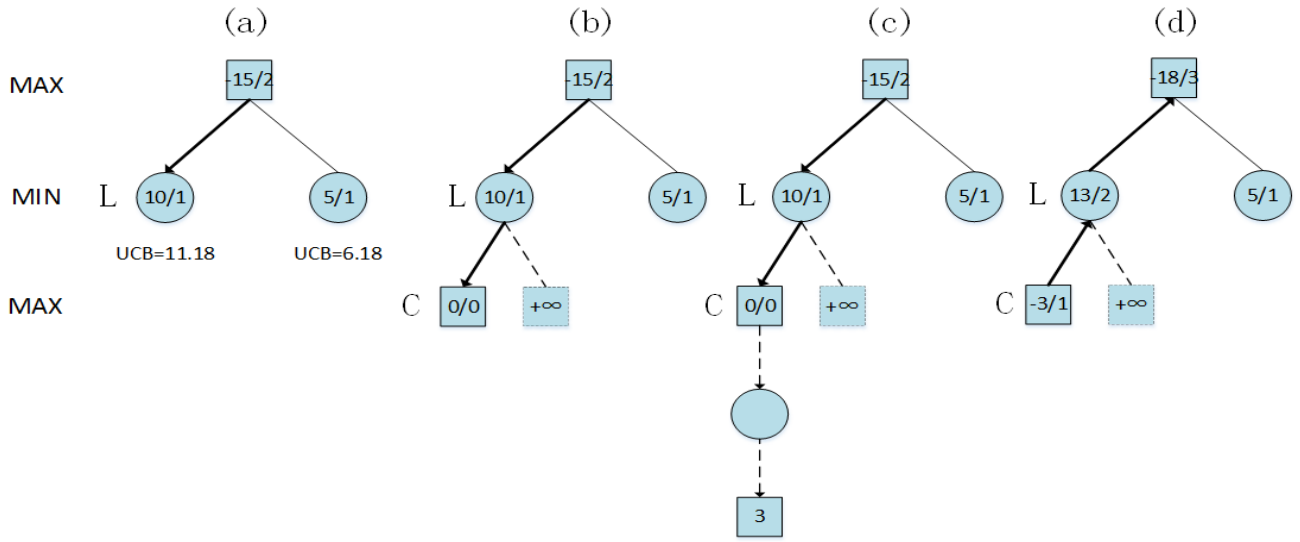
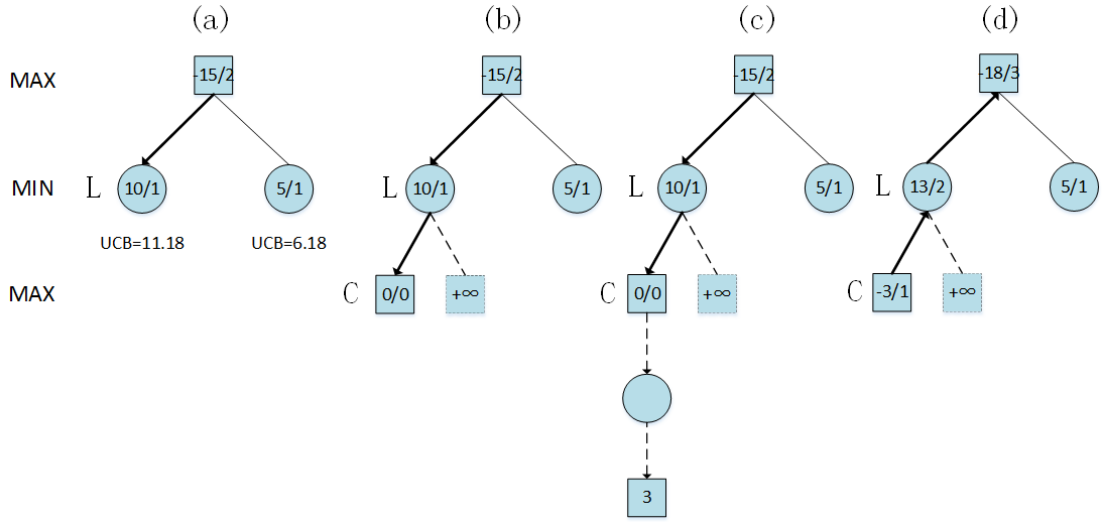


图 3.25 蒙特卡洛树搜索算法。结点中数字代表总收益分数/被访问次数。

在图3.25(b)中，算法随机扩展了L的子结点C，将其总分数和被访问次数均初始化为0。注意，为了清晰地展示算法选择扩展的结点，图3.25(b)画出了L的其他未被扩展的子结点，并标记其UCB值为正无穷大，以表示算法下次访问到L时必然扩展这些未被扩展的结点。图3.4.6(c)中采用随机策略模拟游戏直至完成游戏。当游戏完成时，终局得分为3。

对抗搜索：蒙特卡洛树搜索



在图3.4.6(d)中C结点的总分被更新为-3，被访问次数被更新为1；L结点的总分被更新为13，被访问次数被更新为2；根结点的总分被更新为-18，被访问次数被更新为3。在更新时，会将MIN层结点现有总分加上终局得分分数， MAX层结点现有总分减去终局得分分数。这是因为在对抗搜索中，玩家MIN总是期望最小化终局得分，因此在MIN层选择其子结点时，其目标并非选取奖励最大化的子结点，而是选择奖励最小化的结点，为了统一使用UCB1算法求解，算法将MIN层的子结点（即MAX层结点）的总分记为其相反数。

对抗搜索：蒙特卡洛树搜索

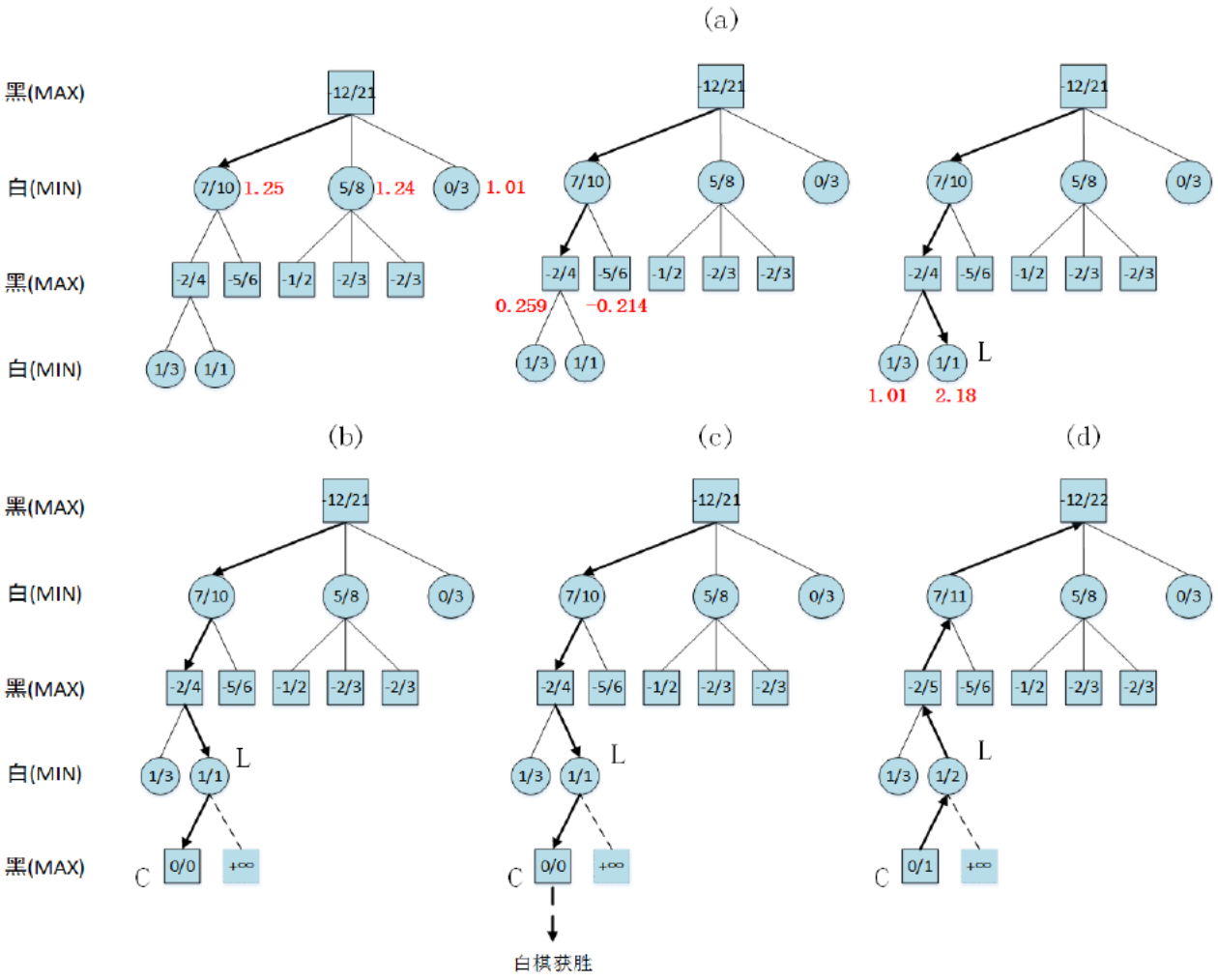


图 3.26 一个更复杂的蒙特卡洛树搜索的例子，
UCB值在对应节点边用红色字体标出

对抗搜索：蒙特卡洛树搜索

函数：UCTSearch

输入： 当前状态s_0

输出： 玩家 MAX 行动下，当前最优动作a^*

```
1 v_0 ← create_node(s_0)
2 while 未达到最大迭代次数 do
3 |   v_l ← SelectPolicy(v_0)
4 |   s_t ← SimulatePolicy(v_l.state)
5 |   BackPropagate(v_l, s_t)
6 end
7 a* ← UCB1(v_0, 0)
```

函数：SelectPolicy

输入： 选择的起始节点v_0

输出： 选择步骤的结束节点v

```
1 v ← v_0
2 while not terminal_test(v.state) do
3 |   if v 存在未被扩展的子节点 then
4 |   |   return Expand(v)
5 |   else
6 |   |   v ← UCB1(v, C_p)
7 |   end
8 end
```

函数：SimulatePolicy

输入： 状态s_0

输出： 模拟的终止状态s

```
1 s ← s_0
2 while not terminal_test(s) do
3 |   a ← 从 action(s) 中随机采样
4 |   s ← result(s, a)
5 end
```

函数：BackPropagate

输入： 反向传播更新的起始节点v，终局状态s_t

```
1 while v is not null do
2 |   v.N ← v.N + 1
3 |   v.Q ← v.Q − utility(s_t, player(v.state))
4 |   v ← v.parent
5 end
```

函数：Expand

输入： 节点v

输出： 未被扩展的后继节点v'

```
1 a ← action(v.state) 中随机的未探索动作
2 v' ← create_node(result(v.state, a))
3 v'.N ← 0
4 v'.Q ← 0
5 v'.parent ← v
6 v'.children ← ∅
7 v.children ← v.children ∪ {v'}
```

函数：UCB1

输入： 节点v，超参数c

输出： 置信上限最大的动作a^*

$$a^* \leftarrow \arg\max_{v' \in v.children} \frac{v'.Q}{v'.N} + c\sqrt{\frac{2\ln v.N}{v'.N}}$$

作业内容

黑白棋（Mini AlphaGo）

黑白棋(Reversi)，也叫苹果棋，翻转棋，是一个经典的策略性游戏。一般棋子双面为黑白两色，故称“黑白棋”。因为行棋之时将对方棋子翻转，变为己方棋子，故又称“翻转棋”(Reversi)。使用8x8的棋盘,由两人执黑子和白子轮流下棋，最后子多方为胜方。

实验要求

- 使用MCTS算法实现Mini AlphaGo for Reversi
- MCTS算法部分需要自己实现，尽量不使用现成的包，工具或者接口
- 在博弈过程中，Mini AlphaGo每一步所花费的时间以及总时间需要显示出来
- 需要有简单的图形界面用于人机博弈交互
- 使用Python语言

11月3日（周二）晚上8点前完成。