复杂博弈搜索中极大极小算法与 UCT 算法的比较

黄　晶[1]　刘知青[2]
北京邮电大学
北邮－九鼎计算机围棋研究所
中国，北京 100876
E-mail：flyanjj@yahoo.com.cn　E-mail：zhiqing.liu@gmail.com

摘要：本文所指的复杂博弈有以下两个特点：1）搜索空间大，现有计算能力只能搜索全部搜索空间的极小一部分；2）博弈局面复杂，现有计算方法无法对博弈局面进行准确评估。文章以机器博弈的介绍为起点，从具体实现、优缺点等方面对基于静态搜索思想的极大极小算法和基于动态搜索思想的 UCT 算法进行了比较。在利用两种方法解决同一实际问题后，验证了两种算法各自的特点并在深入分析中找到了二者的不同。经过分析发现，基于动态搜索思想的 UCT 算法具有更强的灵活性和高效性，更适用于解决需要大规模搜索的复杂问题。

关键词：博弈　搜索　极大极小算法　UCT 算法

A comparative study of minimax and UCT for complicated games

Huang jing[1]，Liu zhiqing[2]
BUPT-JIU DING Computer GO Research Institute
Beijing University of Posts and Telecommunications
Beijing，China100876
E-mail：flyanjj@yahoo.com.cn[1]
E-mail：zhiqing.liu@gmail.com[2]

Abstract：The complicated games referred to by this paper have the following two characteristics：1）the search space is large and the existing computer power can only search a tiny faction of all the search space；2）the game situation is complicated and the existing method of calculation can not accurately assess the game situation. This paper begins with the introduction of computer game. From the implementation，advantages and disadvantages，it compares the minimax search algorithm which is based on the static search and the UCT algorithm which is based on the dynamic search. Through using two methods to solve the same practical problems，their respective characteristics are verified and the differences of the two methods are found. After the analysis，we find that the UCT algorithm which is based on the dynamic search is more flexibility and efficiency. The algorithm is more suited to solve complex problems which require large-scale search.

Key Words：game；search；minimax；UCT

1. 引言

人工智能诞生 50 多年以来，在知识工程、模式识别、机器学习等领域取得了蓬勃发展。其中，计算机博弈受到了人们特别多的关注。因为在计算机博弈的研究过程中，很多实际问题得以解决，这使得人们看见了电脑智能向人类智能靠近的希望。

但是，现有的大量计算机博弈程序的基本思想仍然基于静态搜索算法，它们往往依赖于较为完整的博弈树，因而缺乏灵活性，且效率不高。由于这样的缺点，使得静态搜索算法很难被用在更为复杂的博弈游戏中。为此，人们开始了对动态搜索算法的研究。目前，一些动态搜索算法已经被应用于计算机围棋中，并且获得了不错的效果。本文通过一个简单的例子，展示了两种不同的搜索方法在解决同一问题时所体现出来的不同性能，从实验角度证明动态搜索方法有着更强的灵活性和高效性。

2. 博弈及极大极小搜索

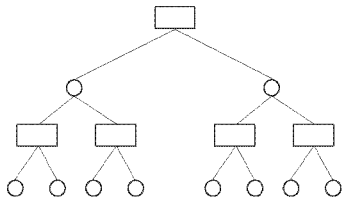

图 2-1 博弈树

博弈[1][2]，是一类富有智能行为的竞争活动（例如：下棋、打牌、战争等），最简单的博弈是双人完备信息博弈[3]。双人完备信息博弈是指两位选手对垒，轮流走步，每一方不仅知道对方已经走过的棋步，而且还能估计出对方未来的走步，对弈的结果是一方赢，另一方输，或者双方和局。双人完备信息博弈过程可以使用一棵特殊的“与或树”表示（图 2-1），这种“与或树”被称为博弈树，通过搜索博弈树实现问题求解。博弈树有下面一些特点：

（1）博弈的初始状态是初始节点。

（2）博弈树中的“或”节点和“与”节点是逐层交替出现的。自己一方扩展的节点之间是或的关系，对方扩展的节点之间是“与”的关系。双方轮流扩展节点。

（3）整个博弈过程始终站在某一方的立场上，所有能使自己一方获胜的终局都是本原问题，相应的节点是可解节点；所有使对方获胜的终局都是不可解节点。

求解问题时一般采用搜索博弈树的方法，办法是生成一定深度的博弈树，找出当前最好的行动方案。在此之后，在已经选定的分枝上扩展一定深度，再选择最好的行动方案，如此进行下去，直到分出胜负或者和局。每次生成博弈树的深度，当然是越大越好，但由于受计算机内存空间的限制，博弈树的深度根据实际情况而定，但每次扩展节点时必须至少扩展一层“或”节点一层“与”节点。常用的搜索有极大极小搜索法。

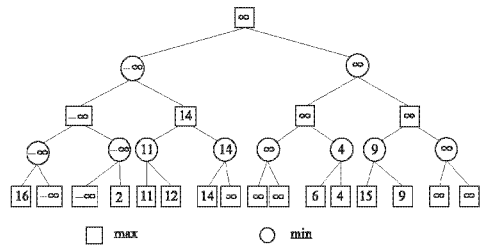

图 2–2 极大极小搜索示意图

极大极小搜索的 5 个步骤：

A. 生成整个博弈树，即扩展树的每个节点。

B. 用静态估值函数 F 对每个叶节点进行估值，得出每个终节点的评价值。

C. 用终节点的估值来得到其搜索树上一层节点的估值。

D. 重复 C 过程在 max 层取其分支的最大值，min 层取其分支的最小值，一直回溯到根

节点。

E. 最终，根节点选择分支值最大的走步走。

图 2–2 为每个节点仅有两个子节点，深度为 4 的博弈树。其中的叶节点如果不是自然终结节点，可由估值函数给出评估值，用来形象地表示己方在该叶节点时的收益。用极大极小值算法从叶节点向上，“max”节点取其子节点中的极大值，“min”节点取其子节点中的极小值，由叶节点层层递推可得到根节点的值为∞，来自于其右子节点。这个求解过程表明当前决策方应该选择其右子节点的方法，在双方都是理智决策的前提下，能保证获得收益∞。

在 John von Neumann 早期论述中[4][5]，终端节点的取值只有 1、0 和–1，代表胜、和、负。而实际的计算机博弈树构造中（如图 2–2 所示），利用极大极小算法完成搜索时，节点的值可以是灵活的，一般给出一个整数值代表一定的收益，再用“∞”代表获得全部收益，“–∞”代表没有任何收益，这就为非完整博弈树的求解（搜索）扫清了障碍，但与此同时也提出了更复杂的问题——怎样才能给出合理的估值函数。

3. 基于 UCT 算法的搜索

1）基于 rollout 思想的算法

在本文中，我们将考虑 Monte Carlo[6][7]算法（它也被称为基于 rollout 的算法）。不同于多级式树的建立，一个基于 rollout 思想的算法通过从初始状态开始重复给出抽样事件而建成前向树。而每一个事件都是状态–行为–奖励的三元序列。那些因事件被加入到树中而产生的信息，也将成为树的一部分。

我们考虑基于 rollout 思想的算法的原因是，它允许我们记录那些曾经遇到过的已抽样状态的行为评估值。因此，如果状态是重复遇到的，则这些行为评估值能够被用来帮助我们有倾向性地选择之后的行为，这样可以潜在地加速估计值收敛的速度。如果这些重复遭遇的节点在整个程序中所占例较小的话，那基于 rollout 思想的抽样过程就将退化成非选择性的 Monte-Carlo 过程。另一方面，继承状态的范围也会集中于仅有的几个状态。通过执行基于 rollout 思想的算法得到的样本将优于其他方法产生的样本。

表 2–1　Monte-Carlo 算法的伪码

```
1: function MonteCarloplanning (state)
2: repeat
3:    search (state,0)
4: until Timeout
5: return bestAction (state,0)

6: function search (state, depth)
7: if Terminal (state) then return 0
8: if Leaf (state, d) then return Evaluate (state)
9: action :=selectAction (state, depth)
10: (nextstate, reward):= simulate Action (state, action)
11: q:=reward +γsearch (nextstate, depth+1)
12: Update Value (state, action, q, depth)
13: return q
```

上表（表 2–1）是 Monte-Carlo 算法的伪码。该算法迭代产生事件，并且返回那些具有最高平均行为奖励的节点。在 Update Value 程序中，总的收益值 q 用来修正那些给定了的状态行为在指定深度的估计值。事件通过 search 函数产生。该函数递归地选择并实现行为，直到满足某个终止条件。整个算法的执行将严格依赖于怎样选择行为（第 9 行）。在非选择性的 Monte-Carlo 算法中行为将被均匀地抽样。而 UCT 算法的贡献在于它提出了很好地实现选择

行为抽样的方法。

2）随机老虎机问题和 UCB1

一个有 k 个臂（节点）的老虎机被一系列的随机收益值所表示，X_{it}，i = 1，……，k；$t \geqslant 1$，这里 i 表示第 i 个臂的编号。拉动第 i 个臂，得到 X_{i1}，X_{i2}，……。为了简单，我们假设收益在 0～1 之间。而下一个将被选择的臂的编号，既取决于过去所选择臂的情况，也受限于得到的收益值。根据这一策略，可以预期到损失值为：

$$R_n = \max_i E[\sum_{t=1}^{n} X_k] - E[\sum_{j=1}^{k} \sum_{t=1}^{T_j(n)} X_{j,t}]$$

造成这个损失的原因是以上策略不一定会选择最佳的臂。如果某策略希望造成的损失增长率不超过某个常数，那么它就应该解决继续探索和利用当前状态之间的平衡问题。

UCB1 算法最吸引人的地方是它成功的解决了继续探索和利用当前状态之间的平衡问题。它为所有臂记录了平均收益 $\bar{X}_{i,\,T_i(t-1)}$，并且选择有最大置信上界的臂：$I_t = \underset{i \in \{1,\ldots,K\}}{\operatorname{argmax}} \{\bar{X}_{i,\,T_i(t-1)} + c_{t-1,T_i(t-1)}\}$，这里 $c_{t,s}$ 是偏移序列。在我们需要的搜索中，UCB1 被用来确定内部节点将要选择的下一个尝试的行为。由于对任意给定的节点其收益的序列值是随时变化的，因此，在 UCT 中，$c_{t,s} = 2C_p\sqrt{\dfrac{\ln t}{s}}$。

3）推荐算法

就每一个已扩展的内部节点而言，在 UCT 中行为的选择问题就好比一个单独的多臂老虎机。不同的臂就相当于不同的行为，而收益相当于以该节点为起始点的路径的累积奖励值。特别地，在状态 s，深度 d 处，使 $Q_t(s,a,d) + c_{N_{s,d}(t),N_{s,a,d}(t)}$ 最大的行为将被选择。

4. 不同搜索方法应用于简单游戏的性能比较

为了显示不同的搜索方法在搜索性能、可实现性等方面的差距，以下将用它们分别实现对一个简单游戏的求解。我们希望这个游戏不需要其他任何知识，以便能更直观的凸显两种方法的区别。

游戏内容：有一堆火柴，数量为 n 根；博弈的双方轮流取走一定数量的火柴，每轮至少取走一根，最多取走 m 根（$m \leqslant n$），谁能取走最后一根火柴谁就是胜利者。该游戏不需要额外的知识，而且这是一个有确定解的问题，可以辅助验证答案的准确性。

分析：这是一个有确定解的问题，即如果 n 是（m+1）的整数倍，则后手一定赢，因为无论先手取走多少，后手都可以根据情况使自己与先手所取火柴根数之和等于（m+1），这样后手必然能拿到最后一根；而其余情况下都是先手获胜，因为只要第一次先手取走了 n%（m+1）根火柴，所剩下的火柴根数一定是（m+1）的整数倍，所以结果也是确定的。

1）使用极大极小搜索算法

为了能够体现极大极小搜索算法在面对复杂问题时的局限性，在这里并不使用上面的分析（因为现实中大部分复杂问题是不能轻易找到确定解的）。于是，我们尝试去找一个评估函数给出特定层数的评估值，但是除了利用确定结果外，我们很难给出适当的评估函数。所以，在实验中，采用了 John von Neumann 在其早期论述中提到的方法，终端节点的取值只有 1、0 和–1，代表胜、和、负，每次搜索都要获得最终结果才算结束。

这样的方法得到的结果是绝对的正确解，但是代价也是巨大的。

图 4–1 给出了在火柴根数不同每次至多能取 3 根时，节点个数的增长图。从图中可以看出，空间的增长速度是指数型的：当初始火柴根数为 20 时，节点数为 133079；初始火柴根数为 21 时，节点个数是初始火柴根数为 20 时的 1.9 倍（252396）；而当初始火柴根数变为 26 根时，节点个数已经增加到了 6301680 个。

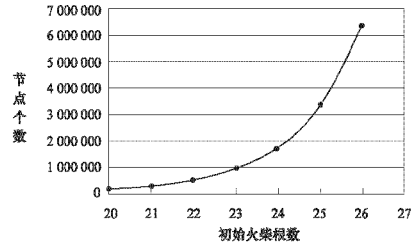

图 4–1　极大极小搜索算法火柴根数与节点

若初始火柴根数为 n 时所需要扩展的节点个数为 $T(n)$，每次至多取 m 根火柴，则有：

$$T(n)=1+\sum_{i=1}^{m}T(n-i) \qquad ①$$

$$T(n-1)=1+\sum_{i=1}^{m}T(n-i-1) \qquad ②$$

$$T(i)=1 \text{ 其中} i\in[0,m] \qquad ③$$

由①②可推导出：

$$T(n)-T(n-1)=T(n-1)-T(n-m-1) \qquad ④$$

所以

$$\begin{aligned} &T(n)=2T(n-1)-T(n-m-1)\\ &T(n)\leqslant 2T(n-1)\quad (n\geqslant m) \end{aligned} \qquad ⑤$$

由③⑤可以推出$T(n)\leqslant 2^{n-m}$（$n\geqslant m$）。从该式可以看出，节点的个数大致满足以 2 为底的指数分布，这与实验所示规律相符。

与此同时，从图 4–2 看出，时间的变化也满足指数的增长：当初始火柴为 20 根时，做出一次选择耗时仅 0.484 秒；但是，当火柴根数为 26 时，耗时已经超过 20 秒。说明在利用极大极小算法时，所用时间约等于 1.9^（n–20）*0.484 秒。也就是说，在最初节点数较少的时候，该算法还能保证一定的有效性，可是当节点扩充到一定程度后，时间上的消耗会过大。

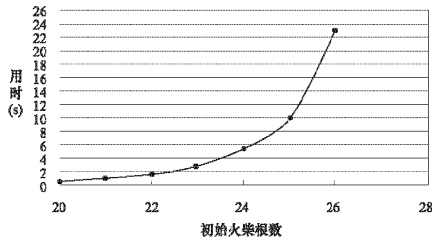

图 4–2　极大极小搜索算法火柴根数与用时

2）用 UCT 算法进行搜索

从图 4–3 可以看出，在用时只有极大极小搜索算法用时 40%的情况下，UCT 搜索几乎能够给出完全正确的答案，并且在用时为极大极小搜索算法用时 30%时，正确率已经高于 80%。由此我们能看到 UCT 算法效率上的优势。

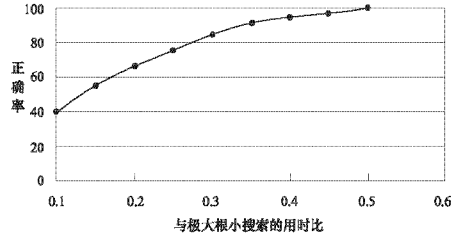

图 4-3　n=25，m=3 时 UCT 算法正确率与月时关系图

从图 4-4 我们可以看到，采用 UCT 方法后，空间的消耗也远小于极大极小搜索算法的消耗。这是因为 UCT 算法本身在继续探索和利用当前节点方面有着很好的平衡。在树的建立之初，好的节点可能被更多地访问，而那些没有被尝试过的节点也可能在一定的探索后被访问到。而树在建立到一定程度后，一些看起来绝对坏的节点将不会再被扩展。这使得更大规模的搜索成为可能，减小了内存对搜索的限制。

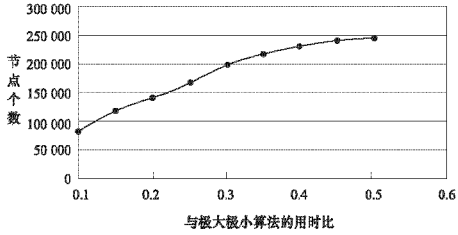

图 4-4　n=25，m=3 时 UCT 算法节点数与月时关系图

由于本题对于获胜方而言，每一步都是有固定唯一解的，所以 UCT 搜索相较于极大极小搜索在时间上的优势未能淋漓尽致地体现出来。但是，实际问题中，获胜的方式往往不止一种，相对最优解就已经足够。而且 UCT 搜索可以完全脱离知识，只用反复地随机模拟就能估计出结果。即使在面对一些确定解问题（例如：某些棋类的残局问题），适当的模拟时间就可以使得求解到的答案具有很高的正确率。因此，我们可以省去寻启发式函数的过程；同时，我们可以控制搜索时间，在任意时间内我们都可以得到反馈的结果。而在空间上 UCT 搜索方法更是优势明显。所以，是值得在更加复杂的问题中使用的方法。

5. 总结与展望

从以上分析我们可以看出：基于动态搜索的 UCT 算法具有高效性和灵活性等优点，更适用于需要大规模搜索的复杂问题。但是，由于其搜索的过程是基于模拟和评估，所以答案的正确率并没有静态搜索算法——极大极小搜索算法好。怎样将两个算法更好的结合，是以后可以继续研究的方向。

参 考 文 献

[1] Cohen P R，Feigenbaum E A．The handbook of artificial intelligence [M]．New Jersey：Addision Wesley，

1982：45–80.

[2] 陆汝铃. 人工智能. 科学出版社，2000.

[3] Nilsson N J. Artificial Intelligence：A new Synthesis. 机械工业出版社，2000.

[4] Neumann J von.John von Neumann Collected Works（ed.A.H.Taub）. Pergamon Press，Oxford，1963，6：1–26.

[5] Tinne Hoff Kjeldsen. John yon Neumann's Conception of the Minimax Theorem：A Journey Through Different Mathematical Contexts. Arch. Hist. Exact Sci 56（2001）39–68，Springer-Verlag 2001.

[6] R. Coulom. Elecient selectivity and backup operators in monte-carlo tree search. InP. Ciancarini and H. J. van den Herik，editors，Proceedings of the 5th International Conference on Computers and Games，Turin，Italy，2006，To appear.

[7] L. Kocsis and C. Szepesvari. Bandit based monte-carlo planning. In 15th European Conference on Machine Learning（ECML），pages 282–293，2006.