成　绩	
评卷人	

研究生	李雪
学　号	2015363039

武漢紡織大學

研 究 生 课 程 论 文

论文题目	博弈树搜索技术
完成时间	2020 年 11 月 01 日
课程名称	人工智能
专　　业	电子信息
年　　级	2020 级

武汉纺织大学研究生处制

一、 引言

随着计算机技术的突飞猛进，人工智能在双人博弈的游戏上有着长足的进步。在双人博弈游戏中，经典的算法就是极大极小值算法(Min-Max)，以及对其改进后的 alpha-beta 剪枝算法，两者在很多决策游戏中都有着不错的表现。但上述算法比较依赖于与当前棋盘状态相关的评估函数。当评估的节点不是最终节点时，评估函数往往来自于人为编写的专家系统，而专家系统的优劣直接决定了评估函数的准确度。不准确的评估函数会舍去一些合理的决策分支，使得博弈结果不理想。

近十年，通过对博弈游戏的不断研究，人们在蒙特卡洛模拟评估的基础上发展出了蒙特卡洛树搜索(MTCS)算法。不同于对决策树进行简单的蒙特卡洛模拟，蒙特卡洛树搜索是一种对蒙特卡洛模拟结果进行不断反馈调整的启发式搜索算法。双人博弈游戏是指由两名玩家参与的博弈。参与的两人具有竞争关系，根据博弈规则，每人将选择自己最优的博弈行为以取得博弈的胜利。我们可以通过构建一个博弈树来展示一个博弈游戏，树的节点表示博弈游戏的局面，分支则表示一种行棋策略。

蒙特卡洛树搜索算法主要由两个问题组成：第一个是起源于多臂匪徒问题(multi—armed bandit problem)的选择策略，即上限信心界策略 Upper Confidence Bound strategy，UCB)，运用到搜索树中形成了上限信心界应用树算法口，它决定了如何从模拟的或好或坏的众多结果中“学会”提高下一次的决策；第二个是模拟策略问题，其本质是蒙特卡洛模拟评估，简单来说就是通过随机扔骰子的方式来探索博弈树，并用多次模拟后的胜率来评估每个节点的优劣，但是难点在于面对巨大的样本空间时，如何抽样才能使样本更有效且更准确。

本文主要针对 Minimax 算法、alpha-beta 剪枝算法和 MCTS 算法进行研究，并通过这三种算法实现井字棋对弈实战和算法的应用。

二、 相关工作

2.1 剪枝加速策略

(1)绝对剪枝策略

绝对剪枝可以表示为：对于一个多节点的多臂匪徒问题，若有一个节点

的访问次数达到了预计访问总次数的一半，则不再继续进行模拟测评。

由于 UCT 算法的原理是上限索引值越大的节点会被更多地访问，因此若一个节点的访问次数已经超过了总访问次数的一半，则不会再访问更多的节点。这证明该访问最多的点一定是上限索引值最大的，此时可以提前结束模拟过程，即可以节约运算资源。

(2)渐进展开式的剪枝策略

这种剪枝策略依旧是针对 UCB 策略的剪枝方法，主要思路是，K 个节点的多臂匪徒问题并不会一开始就同时访问所有节点，而是按照某一种方法对 K 个节点进行排序。设 t。表示前 i 个节点的访问总次数，则前 N 个节点的访问次数满足

如下递推公式：

$t_{i+1} = t_i + [c \times d^i]$

其中，C(C>0)和 d(d>1)的实数都是可调节参数，中括号为取整运算。这种算法可使排序靠前的节点获得更多的计算资源，从而更有可能将运算资源分配到可能比较合理的节点上。

2.2 渴望搜索算法

在使用 Alpha-Beta 剪枝算法进行搜索时的第一次调用，alpha 取负无穷，beta 取正无穷。如果将传递给一个待搜节点的 alpha 和 beta 组成的区间（alpha，beta）看成一个搜索的窗口，从剪枝的方法来看，算法最后的结果就是（alpha，beta）这个搜索窗口收敛的结果。如果在一定范围内，这个搜索得到的均衡结果能够精确的预计出来，就可以大幅度提高搜索的效率。因此我们不妨预计一个结果，再给预计的结果增加一个范围，来加大预计结果在搜索时命中的可能性。基于以上思路就产生了渴望搜索算法。下面介绍渴望搜索算法的基本流程：

首先通过一次深度为 N-1 的完全搜索（即 alpha 取负无穷，beta 取正无穷的全窗口）得到结果 x，将 x 作为我们的预计值。然后令 alpha=x-window，beta=x+window，建立一个新窗口，进行深度为 N 的搜索。其中 window 是一个较小的值，在实际中可以取一个兵的价值。最后将得到的结果和原来的 alpha 和 beta 进行比较。如果落在这个窗口范围之内就表示猜测命中，反之，

根据具体结果是偏高还是偏低分别进行处理。

(1)返回值 value 处于(x-window,x+window)区间。这种情况下，我们就可以确定要找的值就处在猜测的范围之内，可以直接使用得出此值的着法。

(2)返回值 value 大于等于 x+window。这种情况下，则可以确定要找的值落于窗口的右侧，但无法确定具体值。通常把这种情形就叫做 fail-high，此时必须再次指定搜索的范围，重新搜索才可找到所要着法的估值。由于已经知道要找的值介于 value 和正无穷之间，所以建立窗口（value，+无穷），进行新的搜索。

(3)函数的返回值 value 小于等于 x-window。这种情况下，则可以确定要找的值小于等于 x-window，同样无法确定具体值。通常把这种情形叫做 fail-low。此时也必须重新给定搜索的范围，再次进行搜索。即建立新窗口（-∞，value），进行再次搜索。

2.3 基于概率的 B*算法

通过搜索可以较为容易的发现通过回溯区间界限的方式去衡量一棵子树的潜力，未能考虑到回溯的区间内估值的分布不均匀的情况。因为父节点的边界值是通过其子节点边界值的最大值回溯得到的。这种问题的一个示例如图 2.1 所示。

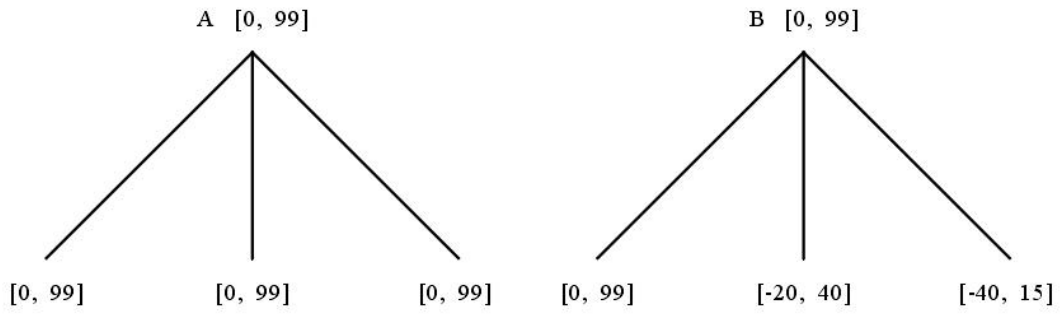


图 2.1 边界范围对节点信息描述不充分的情况

节点 A 和节点 B 回溯的各个节点的分布状态区别很大，但回溯到节点 A 和节点 B 的区间界限却是相同的，对于原始的 B*算法来说这两个节点的潜力是相同的，具有同样的地位。掩盖了事实上节点 A 和节点 B 的区别。由此说明使用分布状态来衡量子树的潜力是十分必要的。

针对 B*算法存在的问题 Palay 于 1983 年提出了一种基于概率的 B*算法(简称为 BSP 算法)。在对于由静态评估函数而引起的区间边界值不准确问题，

BSP 算法通过使用浅层搜索的方式，来产生算法中应用到的各种估值；对于乐观估值与悲观估值形成的区间边界不再采用简单加权静态评估函数的方式来估算，而是在搜索之初，让行棋方（在根节点处走棋的一方）多走一步额外的着法，来建立乐观边界值；同时，引入实际值作为节点结构的一部分，将一个节点的分布划分为乐观估值与实际值、悲观估值与实际值两部分：乐观估值与实际值之间的范围是行棋方的作用域，它试图使实际值向乐观估值逼近；悲观估值与实际值之间是对手的作用域，它试图使实际值向悲观估值靠近。实际值的估算也由浅层搜索的方式得到。对于上图中演示的问题，引入了节点估值的概率分布来代替区间界限去区别节点的潜力。

三、 算法描述（伪代码）

3.1 Minimax 算法

在计算机博弈中，Minimax 算法需遵循极大极小原则，即：首先假定对手跟自己一样聪明，不会犯错，即对手同样总是选择对他最有利的棋步，因此在己方行棋时，要考虑到对自己最不利的情况，不采取任何冒险着法，从最坏的情况中选择最好的着法，从而依据此原则指导搜索的算法。

如下为 Minimax 算法的伪代码：

```
def Minimax(node, depth, player):
    if depth == 0 or node is a terminal node:
        return the heuristic value of node
    if player == True:
        bestValue = -∞
        for each child of node:
            v = Minimax(child, depth-1, False)
            bestValue = max(bestValue, v)
        return bestValue
    else:
        bestValue = +∞
        for each child of node:
            v = Minimax(child, depth-1, True)
```

```
            bestValue = min(bestValue, v)
        return bestValue
```

3.2 α - β 算法

α - β 算法在搜索进行中，每个搜索节点的值都在与 alpha 和 beta 进行比较，如果某个着法的结果小于或等于 alpha，那么它就是很差的着法，应该将其抛弃。如果某个着法的结果大于或者等于 beta，那么整个节点废弃，因为对手不会希望走到这个局面，它会采用别的着法来避免达到这个局面。当某个着法的结果大于 alpha 并且小于 beta 时，这个着法就是走棋一方可以考虑的着法。α - β 算法依据此原则进行指导搜索。

如下为α－β 算法的伪代码：

```
def alpha-beta(node, depth, alpha, beta, player):
    if depth == 0 or node is a terminal node:
        return the heuristic value of node
    if player:
        v = -∞
        for each child of node:
            v = max(v, alpha-beta(child, depth-1, alpha, beta, False))
            alpha = max(alpha, v)
            if beta <= alpha:
                break #beta pruning
        return v
    else:
        v = +∞
        for each child of node:
            v = min(v, alpah-beta(child, depth-1, alpha, beta, True))
            beta = min(beta, v)
            if beta <= alpha:
                break #alpha pruning
        return v
```

3.3 Monte Carlo 树搜索算法

Monte Carlo 树搜索 (Monte Carlo Tree Search, MCTS) 是一种主要应用于(但不限于)寻找最优决策(例如博弈树)的技术。从整体上而言，该方法在状态空间上执行随机采样，并动态地构建一棵搜索树。MCTS 的基本思想是使用随机模拟的方式完成对节点价值的评估，进而为节点的选取决策提供统计依据。它的核心过程是节点的选择与评估。

如下为 MCTS 算法的伪代码：

```
def MCTS(root):
    decision_time = MAX_TIME
    for time in range(decision_time):
        path = [] #for backpropagation
        node = Select(root)
        simulation_node = Expand(node)
        simulation_result = Simulate(simulation_node)
        Backpropagate(simulation_result)
    retrun a child of root, with highest number of visits

    def Select(node):
        path.append(node)
        while node is nonterminal and node is fully expanded:
            node = a best UCT child of node
            path.append(node)
        return node

    def Expand(node):
        path.append(node)
        if node is nonterminal:
            child = a random child of node
            path.append(child)
            return child
```

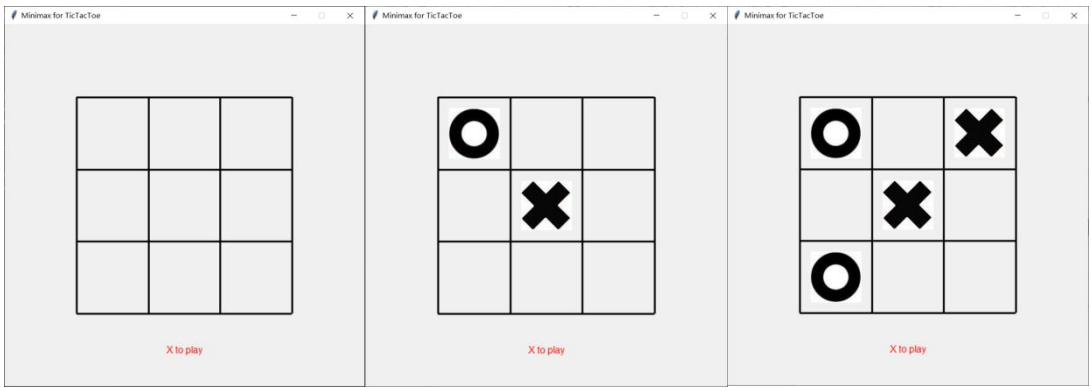
```
        else:
            return node

    def Simulate(node):
        while node is nonterminal:
            node = a random child of node
        return result(node)

    def Backpropagate(result):
        for node in path:
            update node's statistics with result
```

四、 算法验证（结果与分析）

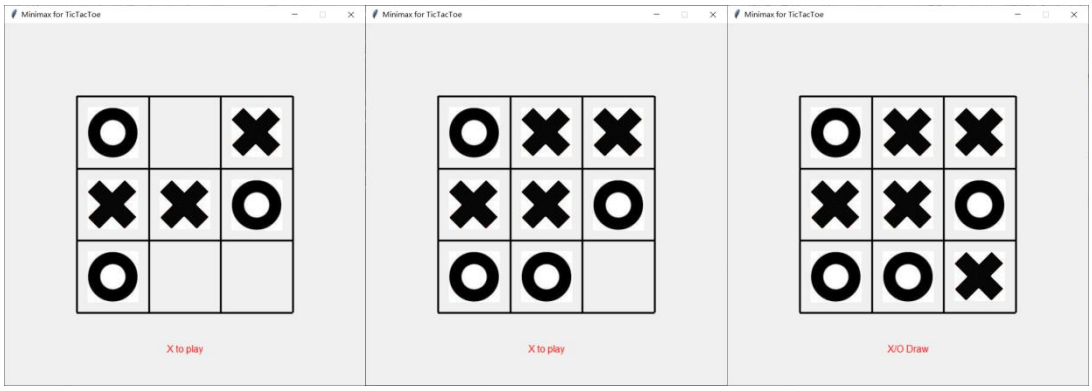
用博弈树算法实现井字棋游戏。井字棋游戏是一种简单的棋类游戏，在3*3的棋盘上，两人轮流下子，谁的棋子先连成3颗一条直线，谁就赢了，可以横着、竖着、斜着。博弈树算法是用搜索来解决这类问题的算法，井字棋游戏步数较少，较国际象棋而言，更容易用博弈树算法实现。

4.1 Minimax 算法实现

如下图如图4.1（a）所示为Minimax算法棋盘的初态，“X”方为己方，“0”方为机器端，图4.1（a）~（f）为Minimax算法双方博弈过程运行图：



(a)Minimax 算法棋盘初态　　(b)　　(c)



(d)　　(e)　　(f)

图 4.1 Minimax 算法博弈结果

分析：为便于理解 Minimax 算法井字棋博弈的执行过程，在此使用一棵假想的简单博弈树上模拟执行一次 Minimax 算法。如图 4.2 所示，展示了一棵简单的 2 层博弈树。

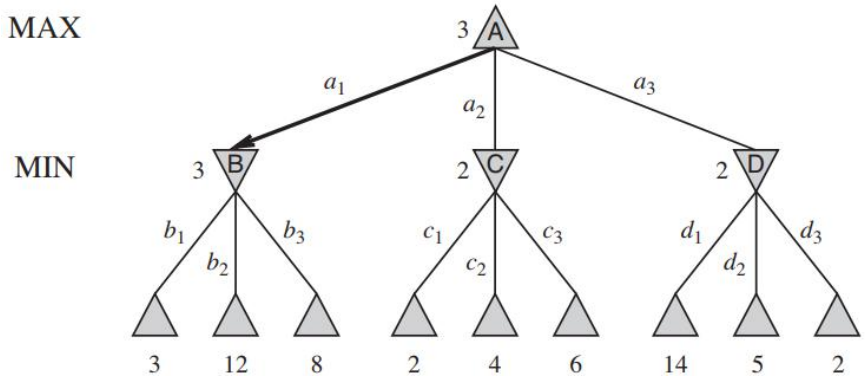


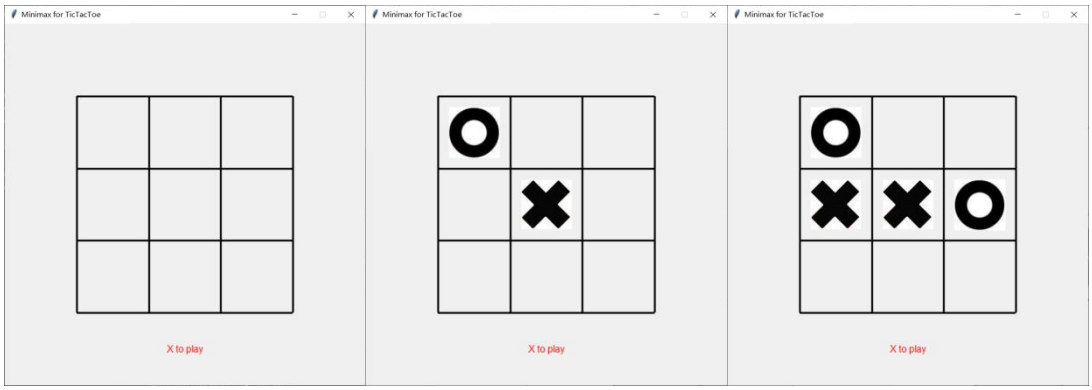
图 4.2 Two-Ply 博弈树示例

根据 Minimax 算法，MAX 方偏爱效用值大的节点，MIN 方偏爱效用值小的节点。在该图中，对于根节点 A 而言，根节点 MAX 的可能落子有 3 种选择：a1、a2 和 a3。

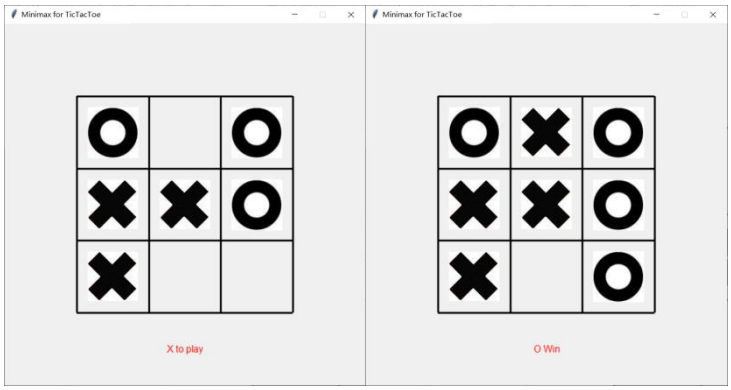
对于节点 B 而言，它的值将由 MIN 方确定：MIN 方将从 b1、b2 和 b3 中选择 b1，并将 b1 的效用值 3 作为节点 B 的值。对于节点 C 和节点 D，采用同样的方式来确定它们的值，得到的结果都是 2。对于节点 A 而言，它的值将由 MAX 方确定：MAX 方将从 a1、a2 和 a3 中选择 a1，并将 a1 的效用值 3 作为节点 A 的值。因此，在状态 A 时，MAX 方的最优落子是 a1。对于其余节点的分析，可依此类推。

4.2 α－β 算法实现

如下图如图 4.3（a）所示为 α－β 算法棋盘的初态，“X”方为己方，“O”方为机器端，图 4.3（a）~（e）为 α－β 算法双方博弈过程运行图：



(a) α－β 算法棋盘初态　　(b)　　(c)



(d)　　(e)

图 4.3　α－β 算法博弈结果

α－β 算法博弈过程分析如下：

（1）搜索深度也就是往下推算的步数是 9，叶子节点估价函数的定义为 f(board)=-1 为人玩家赢，f(board)=1 为电脑赢，f(board)=0 为平局；

（2）棋盘的位置用数字 0-8 来表示，对应的使用列表来存储；

（3）win 元组中存储的是所有可能取胜的位置情况，位置组合也是用元组表示。

（4）为了输出棋盘，设置 mark 列表[‘空白’，‘O’，‘X’]通过循环将使用的-1,1，0 转换成标记符号。

如图 4.4 所示为 α－β 算法博弈流程图：

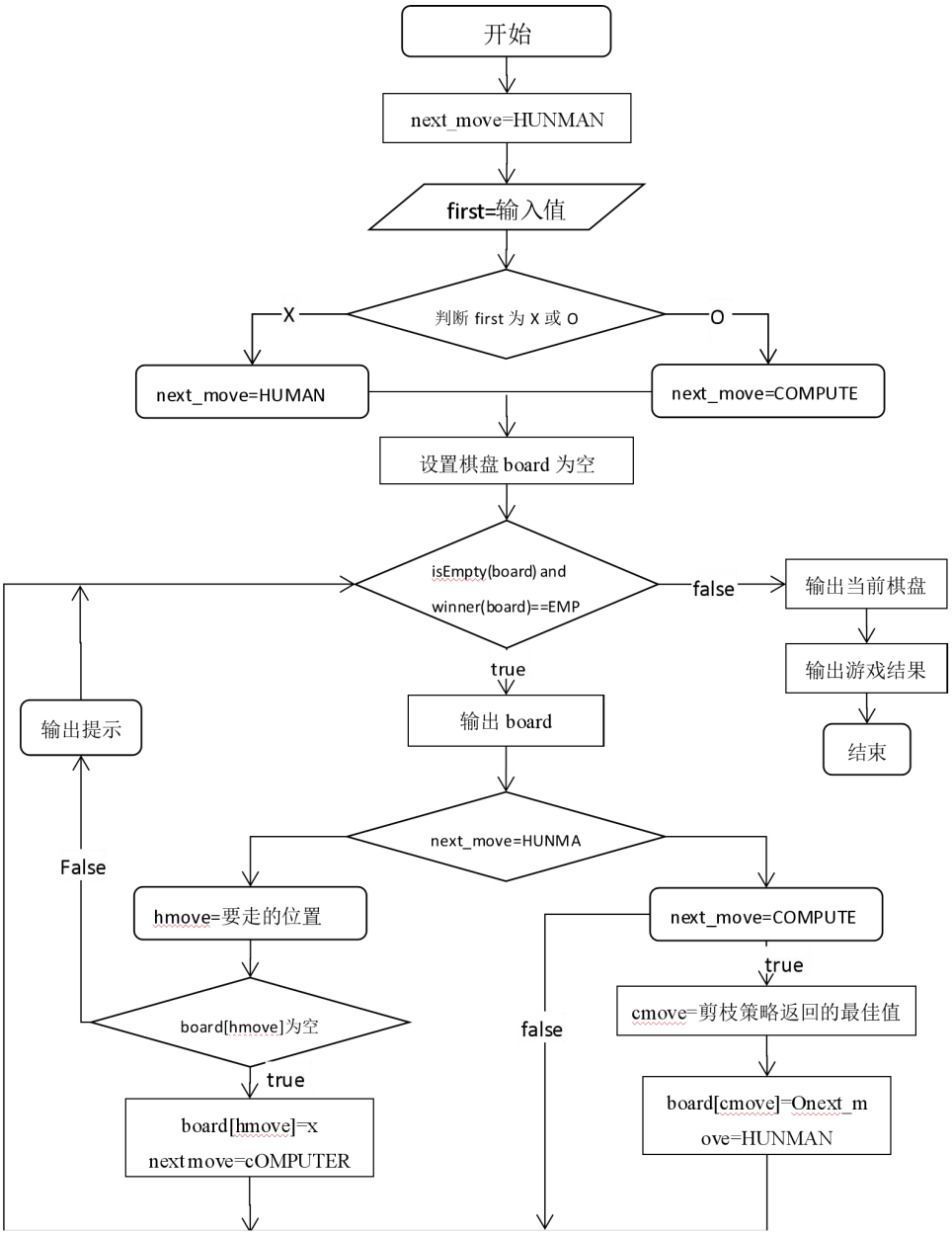
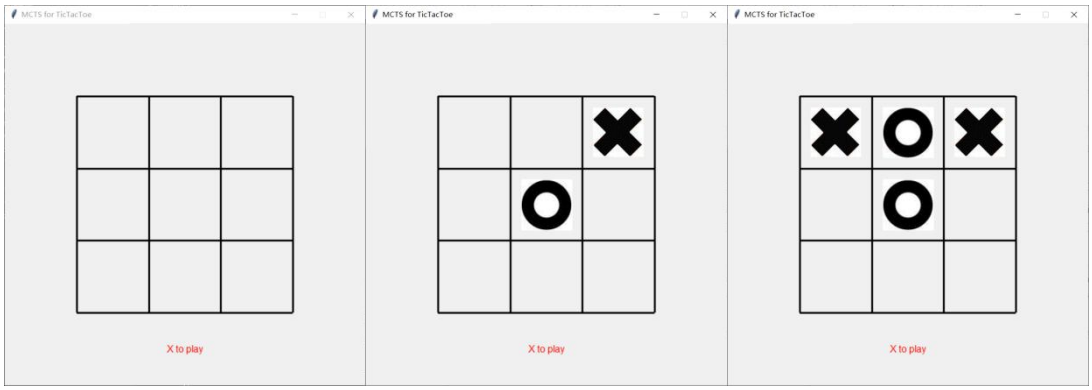


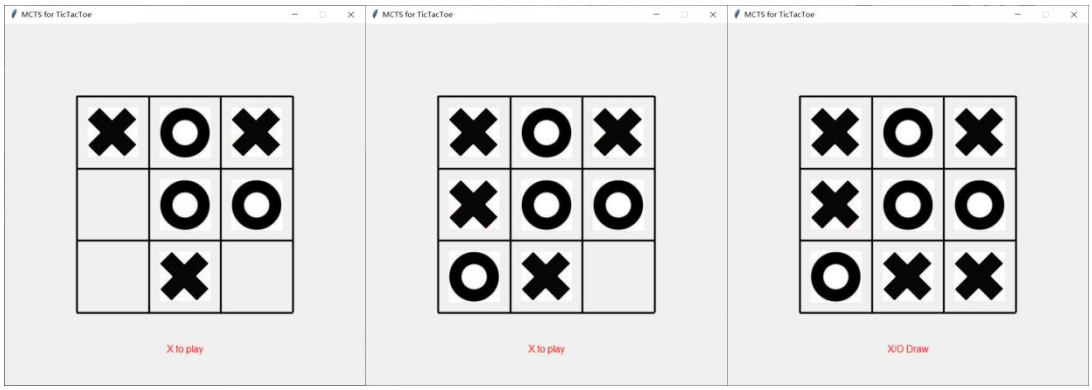
图 4.4 α－β 算法博弈流程图

4.3 Monte Carlo 树搜索实现

如下图如图 4.5(a)所示为 MTCS 算法棋盘的初态，“X”方为己方，“O”方为机器端，图 4.5（a）~（f）为 MCTS 算法双方博弈过程运行图：



(a)MCTS 算法棋盘初态　　(b)　　(c)



(d)　　(e)　　(f)

图 4.5 MCTS 算法博弈结果

分析：MCTS 算法分为 4 个阶段。

在第 1 阶段(Selection 阶段)，它从当前节点 i 开始，如果其孩子节点都已被至少访问过一次(称节点为“被完全扩展”；反之，称节点为“未被完全扩展”)，依照 UCB 算法选取最佳孩子节点进行访问。重复上述过程，一直进行到节点未被完全扩展为止，设该节点为 j。

在第 2 阶段(Expansion 阶段)，从节点 j 的未被访问的孩子节点中随机选取一个，设节点为 k。

在第 3 阶段(Simulation 阶段)，从节点 k 开始，随机模拟下棋(Playout/Simulation)，一直进行到棋局终局，得到明确的胜负平结果。在

第 4 阶段(Backpropagation 阶段)，将得分结果从节点 k 开始向上回传，一直到节点 i，更新每个节点的访问次数与胜率。循环往复执行上述 4 个阶段，直到决策(思考)时间用完为止。此时，对于节点 i 而言，一般情况下，选取访问次数最多的孩子节点作为最佳落子。

如图 4.6 所示为 MTCS 算法的一次迭代过程（四个阶段）：

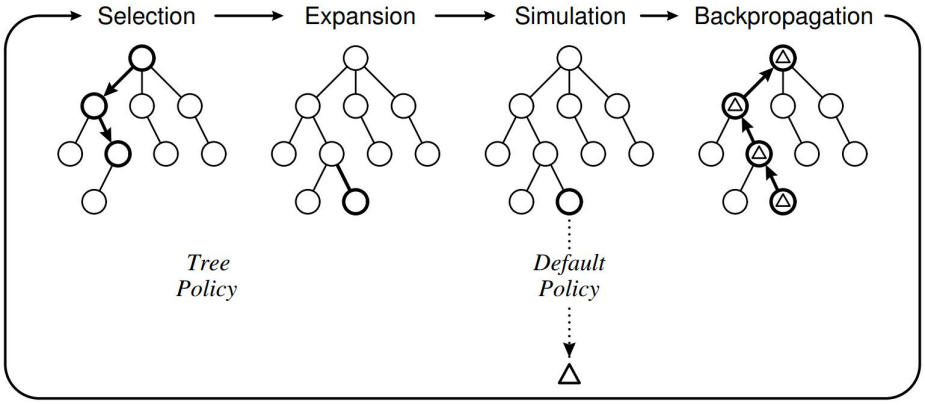


图 4.6 MCTS 算法的一次迭代过程

五、 算法应用

MCTS 可以应用到非常复杂的博弈游戏中，比如象棋，围棋，在搜索空间非常大的时候，普通的极大极小搜索树无法应用，这是由于硬件设备的限制。机器博弈系统的构成对现存的二人、零和、完备信息搜索方法给出了比较全面的讲述。二人、零和、完备信息博弈的研究已经有半个多世纪的历史，其知识结构系统，层次清晰，已经取得了许多惊人的成果。

另外机器博弈在非完备信息游戏的博弈研究方向也取得了一些成果，如拼字游戏 Scrabble，桥牌和扑克都可以实现人机对战的过程。有一些程序还相当强大，但比之完备信息的博弈，它的研究还远远不够广泛。

但是需要特别指出的是，真正应用系统论、控制论和博弈论的方法来探讨象棋博弈问题就是在国际上也不多见。如何通过形式化和模型化来提升象棋博弈问题的研究，如何将象棋机器博弈的研究成果应用到社会安全与军事博弈当中，都是既有理论意义又有应用前景的科研项目。离散事件动态系统、模糊逻辑、神经网络、模式识别、参数估计、离散优化、数据挖掘等在机器博弈这一崭新领域大有用武之地。

六、 结论与展望

通过学习和阅读《博弈树搜索技术》相关文献，了解到 Minimax 算法、Alpha-Beta 剪枝算法和蒙特卡洛算法的各优缺点的应用形式，由于剪枝效果的存在，与极大极小算法相比，Alpha-Beta 剪枝算法搜索的效率有所提升。但遗憾的是，在节点展开的过程中可以发现 Alpha-Beta 剪枝算法十分依赖节点的展开顺序。

MCTS，即蒙特卡洛搜索树算法，大体可分为四步，选择，扩展，模拟，回传。通过不断重复这四步，也就可以不断扩展这棵搜索树，最后到达时间限制或者到达模拟次数限制之后，最终可以形成一棵不对称的树。由于每个节点记录了选择的次数数据，因此树构建完成后可以选择根节点下一层中选择次数最大的子节点作为落子，因为我们知道，虽然一开始的选择节点是很随机的，但是通过回传操作，节点的价值是不断更新的，因此最终选择次数最多的节点我们认为可能是效果比较好的节点，事实也证明这是完全正确的想法。

本篇论文简明地阐述了机器博弈算法的相关原理。总体来说，中国象棋的机器博弈还处在起步阶段，许多关键技术的研究还不够成熟，许多国际象棋成熟的做法我们还需要结合中国象棋的特点加以完善。不过，挑战中国象棋大师、特级大师和冠军的时刻指日可待。

七、 参考文献

[1] 蒋加伏，陈蔼祥，唐贤英. 基于知识推理的博弈树搜索算法[J]. 计算机工程与应用，2004，40(001):74-76,156.

[2] 林建伟. 博弈树搜索技术在牌类网络游戏中的应用[D]. 汕头大学.

[3] 肖齐英，王正志. 博弈树搜索与静态估值函数[J]. 计算机应用研究，1997(4):74-76.

[4] 焦尚彬，刘丁. 博弈树置换表启发式算法研究[J]. 计算机工程与应用，2010(06):42-45.

[5] 张振，庞海. 机器博弈及其搜索算法的研究[J]. 软件导刊，2008.

[6] 蔡自兴 ,徐光 . 人工智能及其应用 [M]. 北京: 清华大学出版 社 , 2003.

[7] 徐心和，王骄. 中国象棋计算机博弈关键技术分析[J]. 小型微型计算机系统，2006(06):961-969.

[8] 王骄，王涛，罗艳红，et al. 中国象棋计算机博弈系统评估函数的自适应遗传算法实现[J]. 东北大学学报(自然科学版)，2005(10):34-37.

[9] 闵文杰. 六子棋计算机博弈关键技术研究[D]. 重庆交通大学.

[10] 杜小勤。《人工智能》课程系列，Part I: Python 程序设计基础，

2018/06/13。

[11] 杜小勤。《人工智能》课程系列，Part II: Python 算法基础，2018/07/31。

[12] 杜小勤。《人工智能》课程系列，Chapter 5：博弈树搜索技术，2018/10/23。

[13] 博弈树, 置换表, 搜索引擎,等. 哈希技术在中国象棋机器博弈系统中的应用研究[J]. 科学技术与工程, 2008, 8(17):4869-4872.

[14] 高强.一种混合博弈树算法在中国象棋人机博弈中的应用研究.硕士论文,大连交通大学,2006