文章编号：2095-2163(2020)04-0027-05 中图分类号：TP301.6 文献标志码：A

基于点格棋的 UCT 算法研究与分析

张宜放[1]，孟 坤[1,2]
（1 北京信息科技大学 计算机学院，北京 100101；2 北京信息科技大学 感知与计算智能联合实验室，北京 100101）

摘 要：以博弈树搜索为核心的 α-β 剪枝算法，受限于估值函数对设计者棋力水平的依赖，难以实现更进一步的提升。论文提出的 UCT（Upper Confidence Bound Apply to Tree）算法结合了 UCB 公式和蒙特卡洛树搜索算法，弱化了算法本身对估值函数的依赖性，最大化利用计算机的算力优势，提升算法的整体效率，并利用其并行化优势优化算法，基于点格棋进行了算法的实现。

关键词：UCT 算法；估值函数；点格棋

Research and analysis of UCT algorithm fordots and boxes game

ZHANG Yifang[1], MENG Kun[1,2]
（1 School of Computer, Beijing Information Science and Technology University, 100101, China;
2 Sensing and Computational Intelligence Joint Lab, Beijing Information and Science and Technology University, 100101, China）

【Abstract】 Alpha-beta pruning algorithm based on searching tree is limited by the dependence of the evaluation function on the designer's chess ability, and it is difficult to achieve further improvement. Paper puts forward the UCT (the Upper Confidence Bound Apply to Tree) algorithm is a combination of UCB formula and Monte Carlo Tree search, which weakens the dependence of the algorithm on the estimation function, maximizes the computer's computing power, enhances the overall efficiency of the algorithm, takes advantage of the parallel optimization algorithm, and realizes the algorithm based on the dots and boxes.

【Key words】 UCT algorithm; evaluation function; dots and boxes

0 引 言

最初，基于深度优先的极大极小搜索算法[1]被用作计算机博弈系统中博弈树搜索[2]的通用策略。随后著名的 α-β 剪枝[3]被提出并得到广泛使用，进行 α-β 剪枝的极大极小值搜索被称为 α-β 搜索。以 α-β 搜索为基础，衍生出了一些优秀的改进算法，如 PVS[4]、MTD(f)[5]等基于搜索空间的局部性原理进行搜索窗口优化的算法，和数据质量敏感的各种启发式或非启发式置换表优化方法[6]。在无法进行完全搜索的情况下，α-β 搜索的实际效果非常依赖其局面评估函数。长久以来，α-β 搜索的局面评估函数由专家经验与精巧的数据结构结合而成，使其算法难以快速应用到无法被完全搜索覆盖的计算机博弈问题中。

为了避免 α-β 搜索过程尤其是其局面评估过程对游戏经验的依赖[7]，各类蒙特卡洛树搜索（Monte Carlo Tree Search，MCTS）算法应运而生。通过大量随机仿真对局来模拟客观的局面胜率，进而解决博弈中局面评估难问题，具有很好的通用性和可控性。然而 MCTS 算法也苦于其天然的统计性质，搜索过程中需要在客观统计与偏好挖掘两个倾向之间寻求平衡，使得 MCTS 在大量随机模拟中，不可避免地包含了大量冗余计算[8]。UCT 算法（Upper Confidence Bound Apply to Tree）作为蒙特卡洛算法的一种延展[9]，通过 UCB 公式引导探索方向，很好地对探索和开发进行了平衡，减少了冗余计算。

UCT 的算法减少了人类关于点格棋博弈技巧和知识的了解水平对博弈系统智力水平的束缚，以求时 AI 摆脱“人”的影响。本文以点格棋为载体，通过对比 α-β 算法，从算法本身对估值函数的依赖性、阶段控制和并行化 3 个方面具体分析了 UCT 算法的优势所在，并针对 UCT 算法中存在的不足之处提出了改进策略。

1 UCB 公式

1.1 UCB 公式描述

假设有一个拥有 K 只手柄的赌博机，对于每一个手臂，都有一个未知的基于随机数的奖励函数。可以做的动作是选择并拉下其中一只手柄，而由此

基金项目：北京信息科技大学 2019 年促进高校内涵发展-大学生科研训练项目（5101923400）；科技计划一般项目（KM201911232002）资助。

作者简介：张宜放（1999-），男，本科生，主要研究方向：网络工程；孟 坤（1980-），男，博士、副教授，主要研究方向：随机模型、网络性能评价、计算智能等。

收稿日期：2019-12-04

赚取随机数量的钱。问题是如何根据当前已掌握的每只手柄的收益情况来决定下次拉下哪只手柄[10]。

分析可知,对于不同手臂其所能赚取的钱是不一样的,多次拉同一个手臂其所能赚取的钱也是不一样的。玩家可以根据之前所累积的知识,拉下某只已开发过的手臂;也可以选择去探索未开发手臂,获得未知数量的钱。如果不停开发已知手臂而不去探索未知手臂,就可能会错过收益率更高的手臂;若不停探索未知手臂,就可能出现收益率一直小于当前已知最高收益率手臂,后悔值越来越大。

因此该问题可以简化为探索和开发之间的矛盾问题,即在可接受的后悔值范围内平衡探索未知手臂和开发当前收益率最高手臂之间的矛盾关系。

对于该问题模型,UCB 算法提出了一种解决方式,具体步骤如下:

(1)首先将所有手臂都尝试拉一次;

(2)选择使如下公式最大化的手臂:

$$UCB_i = X_i + \sqrt{\frac{2\ln N}{T_i}}. \quad (1)$$

式中,X_i 表示第 i 只手臂的平均收益值;T_i 表示第 i 只手臂被探索的次数;N 表示总共探索的次数。这个公式被称作 UCB 公式。

(3)拉下该手臂,更新公式(1)中对应手臂的 UCB_i 值。此时 X_i 和 T_i 会发生变化,N +1;

(4)重复步骤(2);

对于 UCB 公式(1),可以将其拆分为 2 部分,分别是 X_i 和 $\sqrt{\frac{2\ln N}{T_i}}$。前项为当前开发项,表示该手臂过去开发的平均表现;后项为探索项,做为调整值表示该手臂被探索的价值。其累加和为当前对这条手臂的评价值。

UCB 算法即利用当前掌握的知识加上调整值来平衡开发与探索之间的矛盾。在每次做出选择时,UCB 算法会挑选当前拥有最大评价值的手柄。其中的探索项调整值会随着手柄被选择次数的增加而减少,以便在选择手柄时,不过分拘泥已有的表现,适当探索其它的手柄,从而在开发和探索之间取得平衡。

1.2 UCB 公式改进

由于开发与探索之间的比例依赖于具体问题,对于不同的问题有不同的比例选择,所以在公式中的探索项引入常系数 C,以改变开发和探索之间的比例[11]。改进后的 UCB 公式如下所示:

$$UCB_i = X_i + C\sqrt{\frac{2\ln N}{T_i}}. \quad (2)$$

以点格棋为例,在前期布局时给予 UCB 公式(2)较大的常数 C 值,尽量给每一种走法探索的机会;而渐入中局,一定要获取当前局面的最优解,选择表现最好收益率最高的分支走棋,此时减小公式中的 C 值,在开发与探索的平衡中渐渐偏向开发收益率最高的手臂。

2 UCT 算法——UCB Apply To Tree

UCT 算法基于蒙特卡洛树搜索算法,并利用 UCB 公式来增量扩展搜索状态,通过多次仿真来找到当前结点的最优分支[9]。

假设已经建立了一棵庞大无比的博弈树,其叶结点包含了全部 n 层的 n 的阶乘种对局。利用 UCT 算法搜索整棵树从根结点开始的前 k 层,在模拟到第 k 层时给出估值函数来评价当前局面,必胜为 1 必败为 0,评估值作为这次探索的收益值 X,逐层回溯更新父结点的 UCB 值。若达到叶结点,表示棋局结束,直接返回胜负值 1/0 做为收益 X_i。

将 UCT 算法简化描述成如下步骤:

(1)以当前局面为根结点;

(2)生成根结点的所有子结点;

(3)利用 UCB 公式(2)计算每个子结点的 UCB_i 值,选择最大评价值的子结点进行探索;

(4)若该结点非叶结点且搜索深度未达到预设深度 k,将当前结点作为根结点重复步骤(2);

(5)直到遇到叶结点或搜索深度达到预设深度 k,将当前局面评估值或胜负结果逐级回溯,更新每一级祖先结点的 UCB_i 值;

(6)否则,对这个叶结点进行模拟对局,得到胜负结果,将这个收益按更新到该结点及其每一级祖先结点上去;

(7)回到步骤(1)(除非时间结束或者达到预设循环次数);

(8)从根结点的子结点中挑选使公式(2)最大化的,作为下一步 AI 走棋的选择。

3 UCT 算法与 α-β 剪枝算法对比

3.1 对估值函数的依赖性

3.1.1 极大极小搜索算法

极大极小搜索算法是大多数传统算法,如 α-β 剪枝算法的理论基础[2]。其是一种对人类思维的模仿,在机器对某个局面进行分析的时候,假设一方总选择使自己优势最大化的一步,而对手选择使自己损失最小化的一步。如图 1 所示。

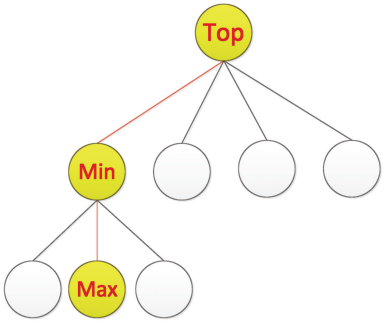

图 1　极大极小搜索树

Fig. 1　MAX-MIN Search Tree

以当前局面建立根结点，使用深度优先遍历整棵搜索树。在每次到达叶结点或搜索深度达到预设深度进行回溯时，对父结点进行判断，如果父结点轮到我方走棋，则判断子结点评估值是否大于父结点当前记录最优评估值，若大于则更新，并记录该子结点为当前我方最优解；反之轮到对手走棋，则判断当前子结点评估值是否小于当前父结点当前记录最优评估值，若小于则更新，并记录该子结点为当前对方最优解。轮到我方走棋的父结点最优评估值初始为负无穷，对方则为正无穷。直到回溯到根结点，选择当前所有子结点中评估值最高的结点作为我方下一步走棋选择即可。

而由极大极小搜索算法衍生出的 α-β 剪枝等传统算法，是在对博弈树进行遍历时所使用的剪枝策略，通过减去不可能选择的分支来提高搜索的效率，可以看做是对极大极小算法的改进和扩展。

3.1.2　估值函数与 α-β 剪枝算法

估值函数是为了对当前局面做出定量分析而设计的，其根据当前局面的优劣给出局面的评估值，必胜为 1 必败为 0，一般情况介于 0-1 之间。估值函数多是由该领域的专家，通过对局面某些特征值分析后得出的结论，可以视为人类经验的总结。

但人类的计算能力也是有限的，当人类计算至某一局面时，会自发地给出一个局面的感性评价来判断是否要选择走这一步，而估值函数所做就是在模拟计算机计算到一定深度的局面时，给出对该局面定性的分析从辅助 AI 决定下一步的走法选择。

在人类棋手之间，胜负的关键在于计算能力和比赛经验，计算能力越强算的越深，对局面的分析也就越准确，经验越丰富的棋手对局面的判断越准确也就更可能主导局势的发展。对计算机同理，在无法进行硬件性能（算力）提升的前提下，估值函数对局面分析的准确性将直接主导极大极小搜索算法的水平，是否能在有限的搜索深度下找到最准确的解。但对于从初始局面构建的一棵博弈树，其结点数随着深度的增大呈指数级上升。那么在无法完全遍历整棵博弈树时，仍非常依赖估值函数。

3.1.3　UCT 算法与 α-β 剪枝算法在估值函数依赖性方面的对比

假设有一个非常精确的估值函数，能以 100% 正确率给出当前任意局面的评估值，那么只需要将其应用到任意搜索算法中去，AI 每步必定能准确算出评估值最大的一步，使一方优势最大化，且这一步必定正确。很显然这样的估值函数是不存在的，其总会存在些误差，对某些特定局面的评估值不那么准确。那么此时，对某一局面的判断失误，会被逐层回传父结点，对整个局面的判断误差被逐层放大至根结点，导致人们做出错误的选择。所以，估值函数的参数也是影响机器棋力的一大重要因素。

对于 α-β 剪枝算法所采用的搜索策略，每个局面只会被模拟一次，一旦这个由估值函数引发的错误值更新了父结点的数据，就会被逐层回传，导致根结点认为在这一分支发现了更优的一步，从而引起判断的失误。对于 UCT 算法，每个局面有多次被模拟到的机会，父结点的回溯策略也不是简单的返回评估值最大的结点，而是返回具有最优分支的结点。换句话说，即便在某一分支中估值函数给出了错误的判断，只要其它分支表现的足够好，UCB 值较大，AI 也不会选择具有最优解的错误分支。举例而言，当 AI 算到走 1 号边能有一个必胜的局面，而走 2 号边会出现 90%的优势局面，α-β 剪枝算法会选择前者，而 UCT 算法会选择后者。对于 α-β 剪枝算法，如果选择 1 号边出现的唯一必胜局面评估值有误，将会直接进入错误的分支。对于 UCT 算法，无论 1 号边的必胜局面正确与否，都不会影响最终的选择；而假如 2 号边模拟的局面中存在某些局面有误，也不会大幅度改变该分支的胜率（就目前来看机器 1 min 仿真次数可以在十万至百万级别，仅几个局面的错误判断几乎不会影响 AI 对整体局势的判断）。

α-β 剪枝算法所要找到的是具有最优解的分支，而 UCT 算法则是为了寻找表现最好的分支，即便最优解可能不在该分支。在估值函数表现不稳定或不够准确时，就很难直观的找到最优解，而使用 UCT 算法可以很好地中和估值函数带来的误差，引导程序向整体局面更优的方向走棋。

3.2　搜索控制策略对比

3.2.1　α-β 算法

α-β 剪枝算法的控制策略主要依靠搜索深度

和迭代次数 2 种：

一般用于区分局面的阶段，如开局、中期、残局阶段，传统剪枝算法使用的是搜索深度控制策略。当搜索深度越深，估值函数的准确性就越高，当搜索到叶结点时，就不需要估值函数直接反馈比赛胜负即可，此时完全不存在估值函数造成的误差；但搜索层数越深迭代次数也就越多，耗时越多。

相比于搜索深度控制，迭代次数控制更像一种无奈之举。在没办法大量剪枝的时候，强制在达到规定迭代次数的时候终止程序，以确保不浪费过多的时间。迭代控制有一定不可预知性，可能在还没进入最优解，所在分支迭代就结束了，但至少可以保证在不超时的情况下，选择一个次优解。所以一般剪枝算法在设计时，都会在深度控制之外设置迭代控制，作为“保险装置”。

对于 α-β 剪枝算法，当算法被迭代控制直接中断时，未探索的分支相当于被剪枝，其状态树尚未生成也就无从判断是否错过最优解。而且同一时刻，剪枝算法对子结点探索程度不相同，强行终止算法可能会出现某一子结点已完成探索而仍有子结点未开始探索的情况发生。所以剪枝算法在设计之初就必须规划好时间和搜索深度的分配，但对于点格棋，局面可能存在大量等价边[12]剪枝，实际应用中根本无法判断程序在某一阶段探索一定深度所需消耗的时间。

3.2.2 UCT 仿真

UCT 算法的控制策略可以分为搜索深度、仿真次数、仿真方向 3 种：

前 2 种控制方法思路基本同剪枝算法，不同的是剪枝算法是迭代次数，而 UCT 算法是仿真次数（或者说是模拟对局的次数）。但迭代次数的控制实质上是用限制迭代次数去控制时间，而 UCT 算法则是直接控制时间去限制仿真次数，在进行调整时会更加灵活。

UCT 算法中可以通过改变 UCB 公式来进行搜索方向的引导。通过调整平衡系数和修正收益大小，都可以直接影响根结点对仿真方向的判断，从而达到提前收敛或是增加搜索广度的效果，相比 α-β 剪枝搜索的按序遍历而言，调整和控制更加灵活。

UCT 算法还可以直接通过仿真方向来判断是否已经找到最优分支。不同于剪枝算法找到最优解的策略，UCT 算法的核心目的是找到最优分支，那么可以预设在达到一定仿真次数后，若某一分支被多次仿真后 UCB 值仍高居不下（某方向仿真次数越多调整项值越小，UCB 值越接近真实评估值），就可以认为该分支为最优分支。

UCT 算法拥有传统剪枝算法无可比拟的随时中断特性[13]。在判断时间不足时，使用 UCT 算法的程序可以直接强制中断，同时返回一个相对不错的次优解。这是由于蒙特卡洛模拟具有的天然统计性质，且搜索初期 UCB 公式中仿真次数 N 较小，后项探索项影响因素远大于前项开发项，可大致认为对子结点的探索次数服从均匀分布，所以即便此时中断算法，也不会出现某一分支几乎未被探索而错过最优解的情况。

3.3 UCT 算法的并行化优势

UCT 算法主要以仿真模拟对局为主，非常适合做并行化[14]。在拥有多个 CPU 核心的情况下，通过并行的展开多个线程，分别进行不同对局模拟。某一线程在模拟完成后，先给公共资源区（整棵博弈树）加 X 锁，修改部分结点 *UCB* 值，该线程再展开下一轮模拟[15]。在这一过程中，不需要访问博弈树的线程仍可以继续工作。UCT 单次仿真平均用时见表 1。

表 1 UCT 单次仿真平均用时

Tab. 1 Average running time of UCT each simulation μs

	单线程	双线程	四线程
用时	353.2	180.4	101.3

由于锁机制的存在，实际应用中每次模拟并不完全独立，因而性能的优化效果会有所减弱。

对于 α-β 剪枝算法，其搜索过程独立性没有 UCT 仿真高，对公共资源区的使用也不像 UCT 仿真在整次模拟结束后才加锁更新结点权值，而是在每一次回溯时就更新父结点权值。在做并行化时，其每一个线程对公共资源区的访问频率更高，导致公共资源区被频繁加锁，其它线程等待时间增加，CPU 利用率下降，并行化的优化效果较弱。

4 实验结果与分析

测试棋盘大小为 6×6，比赛单方总时限为 15 min。6×6 点格棋棋盘共 60 条边，假设一方走 30 步（会有得子连走情况，一方走棋未必为 30 步），标准单步时限即为 15 min/30 步，即 30 s。

测试硬件环境如下：i7，6700HQ，主频 2.6GHz，内存 12G，显卡 960M，四核八线程。

在使用相同估值函数且算法均完成并行化的情况下，将 UCT 算法与 α-β 剪枝算法实现的程序进行对弈测试。30 s 蛙限不同算法对弈结果见表 2。

进一步进行测试，分别给予程序 5 s 和 120 s 的

不同时限进行对弈。见表 3 和表 4。

表 2　30 s 时限下不同算法对弈结果

Tab. 2　Game results of different algorithms within 30 s

算法	单步时限/s	胜利局数	失败局数	胜率/%
α-β 算法	30	1	9	10
UCT 算法	30	9	1	90

表 3　5 s 时限下不同算法对弈结果

Tab. 3　Game results of different algorithms within 5 s

算法	单步时限/s	胜利局数	失败局数	胜率/%
α-β 算法	5	0	10	0
UCT 算法	5	10	0	100

表 4　120 s 时限下不同算法对弈结果

Tab. 4　Game results of different algorithms within 120 s

算法	单步时限/s	胜利局数	失败局数	胜率/%
α-β 算法	120	3	7	30
UCT 算法	120	7	3	70

由表 3、表 4 可以看出在估值函数不稳定且均使用相同估值函数的情况下，UCT 算法能明显规避误差，显著提高胜率，尤其是在搜索时间较短时，UCT 算法的优势越发明显，其短时内可得到次优解和随时中断的特性被进一步放大。当每步时限增大时，UCT 算法苦于蒙特卡洛模拟的固有问题，在大量统计量中难以避免的出现冗余计算，算法效率略显下降，不过仍明显强于 α-β 剪枝算法。

5　结束语

计算机博弈是一个复杂和具有挑战的课题，对与博弈论的学习和研究具有深远的意义。UCT 算法减少了人类关于点格棋博弈技巧和知识的了解水平对博弈系统智力水平的束缚，而机器博弈未来的发展方向，也正是让机器实现自我博弈、自我学习，真正摆脱“人”的影响。本文以点格棋为载体，通过对比 α-β 算法，从算法本身对估值函数的依赖性、阶段控制和并行化 3 个方面分析了 UCT 算法的优势所在，并提出了改进策略。

此次研究虽小有成果，但仍存在一些不足有待进一步的研究和改进。在通过 UCB 公式中引入平衡系数后，该系数的设计方式偏向于动态调整，由 AI 自发的根据当前局面判断平衡开发与探索之间的比例关系，但目前仍无法设计出一套可准确调整平衡系数值的控制函数。最终采用静态控制的策略，通过理论分析和大量实验得出各阶段平衡系数的相对合理值。

参考文献

[1] 林尧瑞，马少平. 人工智能导论[M]. 北京：北京：清华大学出版社，1989

[2] 孙伟，马绍汉. 博弈树搜索算法设计和分析[J]. 计算机学报，1993(5)：361-369.

[3] ZHANG Congpin, CUI Jinling. Improved Alpha-Beta Pruning of Heuristic Search in Game-Playing Tree[C]//IEEE. Computer Science and Information Engineering. 2009：672-674

[4] 邹竞，马华，谢鲲. 一种基于 OpenMP 的并行混合 PVS 算法[J]. 计算机应用研究，2016，33 (1)：56-59.

[5] ZHANG R, LIU C, WANG C. Research on connect 6 programming based on MTD (F) and Deeper-Always Transposition Table[C]//2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems. IEEE, 2012, 1：206-208.

[6] 张明亮. 一种新的博弈树搜索算法及其应用研究[D]. 苏州：苏州大学，2007.

[7] 张明亮，吴俊，李凡长. 五子棋机器博弈系统评估函数的设计[J]. 计算机应用，2012，32(7)：1969-1972.

[8] 季辉，丁泽军. 双人博弈问题中的蒙特卡洛树搜索算法的改进[J]. 计算机科学，2018，45(1)：140-143.

[9] GELLY S, WANG Y, MUNOS R, et al. Modification of UCT with patterns in Monte-Carlo Go[D]. INRIA, 2006.

[10] Auer, Peter, Nicolo Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem[J]. Machine learning 47.2-3 (2002)：235-256.

[11] GELLY S, WANG Y. Exploration exploitation in go：UCT for Monte-Carlo go[C]//NIPS：Neural Information Processing Systems Conference On-line trading of Exploration and Exploitation Workshop. 2006.

[12] 丁濛，张亦鹏，李淑琴. 棋盘局面数据标定方法研究[J/OL]. 计算机应用研究：1-5.

[13] ZHUANG Yimeng. Improving Monte-Carlo tree search for dots-and-boxes with a novel board representation and artificial neural networks[J]. IEEE CIG, 2015：314-321

[14] Coquelin, Pierre-Arnaud, and Rmi Munos. Bandit algorithms for treesearch[J]. arXiv preprint cs/0703062 (2007)

[15] CHASLOT G M J B, WINANDS M H M, VAN DEN H J, et al. Parallel Monte-Carlo tree search[J]. Lecture Notes in Computer Science, 2008, 5131：60-71.

(上接第 26 页)

[3] ADOMAVICIUS G, TUZHILIN A. Toward the next generation of recommender systems：a survey of the state-of-the-art and possible extensions[J]. IEEE Transactions on Knowledge and Data Engineering, 2005, 17(6)：734-749.

[4] BERNERSLEE T, HENDLER J, LASSILA O. The semantic web [J]. Scient-ific American，2001，284(5)：34-43.

[5] LINDEN G, SMITH B, YORK J. Amazon. com recommendations：Item-to-item collaborative filtering[J]. IEEE Internet Computing, 2003, 7(1)：76-80.
2017，28(3)：708-720 (in Chinese).

[6] 刘知远，孙茂松，林衍凯，等. 知识表示学习研究进展[J]. 计算机研究与发展，2016，53(2)：247-261.

[7] BORDES A, USUNIER N, GARCIA-DURAN A, et al. Translating embeddings for modeling multi-relational data[C]// Advances in neural information processing systems. 2013：2787-2795.