基于“连珠”模式的*k*子棋机器博弈增量模型

徐长明，马宗民，徐心和
东北大学信息科学与工程学院

摘要：在连珠棋的机器博弈中，由于以交叉点为基本单位来描述或分析局面的常规方法的低效性，以及普通“模式”在知识表达、局面状态描述和局面评价能力上的有限性，本文提出了一个特殊模式——连珠，在此基础上设计了一个*k*子棋的机器博弈模型CPBIM。基于连珠，构建了若干模式库，用于保存离线生成的领域知识；通过探讨连珠之间的演化关系，得到了连珠及其内部各空交叉点的评价体系；鉴于任何局面都可等价地由若干连珠的排列所表示，本文设计了以连珠为核心的高效的局面状态描述方法；为了降低搜索树中各结点的更新代价，还提出了一种高效的增量更新方法。实验表明，在基于威胁的搜索算法中，CPBIM模型使搜索速度提高到普通模型的5倍左右。

关键字：机器博弈；六子棋；模式；知识表示；增量更新。

The Connection-Pattern Based Incremental Model in *K*-in-a-row Computer Games

Chang-ming Xu, Z.M. Ma, Xin-he Xu
College of Information Science and Engineering, Northeastern University

Abstract: Due to the ineffectiveness of the conventional method to describe and analysis a game position from the view of intersections on board, as well as the limitation of the ability to represent the knowledge, describe the state of position, and evaluate the position, this paper proposed a special pattern, called Connection, to overcome those drawbacks. Then, the *Connection-Pattern Based Incremental Model* in *k*-in-a-row games, abbreviated as *CPBIM*, is designed. Thus, a pattern database also can be built to store the domain knowledge generated off-line. Through discussing the evolutionary between each two Connections, a general method to estimate Connection and every intersection of Connection can be proposed. Because of finding that any position can be presented by the permutation of a certain number of Connections in CPBIM, we can easily design an efficient Connection-based presentation for the state of this position. Furthermore, to reduce the cost of updating the state at each tree node when making a search, we put forward an efficient incremental updating method. The results of experiments show that the search algorithm in CPBIM is 5 times faster than that in the normal model on the speed of visiting nodes of tree.

Key words: Computer games; Connect6; Pattern; Representation of Knowledge; Incremental Updating.

1 引言

作为人工智能的一个重要分支，机器博弈不断地探索和检验着依靠机器解决复杂问题的一般方法。特别是在1997年，拥有256个下棋专用芯片的IBM“深蓝”超级计算机历史性地战胜了国际象棋大师卡斯帕罗夫，曾引起广泛的轰动。由于网格、集群、云等高性能计算已渐渐普及，没有必要去设计像“深蓝”一样的下棋专用机器了，机器博弈更多地关注：如何基于通用的计算平台，构建高水准的博弈程序。由于尚未解决的棋类[1]始终是机器博弈的挑战性问题，本文面向文献[2]提出的一族*k*子棋，以尚未解决的六子棋为对象，探讨如何构建高水平的博弈程序。

Connect(*m*, *n*, *k*, *p*, *q*)表示一族*k*子棋，也称连珠棋，对弈的双方分别执黑和执白，黑先。其形式化描述*Connect*(*m*, *n*, *k*, *p*, *q*)的涵义为：1）棋盘包含*m*×*n*个交叉点。2）黑第一次下*q*颗棋子，此后，双方轮流下*p*颗棋子。3）在（水平的、垂直的、对角线方向的）任何一条线上，能率先形成本方连续不间断的*k*子序列者获胜；若双方均无获胜的可能，判和。在*k*子棋中，较为著名的是五子棋和六子棋。五子棋形式化地描述为*connect*(*m*, *n*, 5, 2, 1)，Renju和Go-Moku是两种最常见的五子棋，已经得到解决——先手必胜。由吴毅诚教授在文献[2]中提出的六子棋，可形式化地描述为*connect*(*m*, *n*, 6, 2, 1)。六子棋无禁手；一般采用19×19的棋盘；其复杂度不但远高于五子棋，也高于所有已解决的棋类。

长期以来，和文献[3-4]所介绍的方法类似，几乎所有的*k*子棋程序都以棋盘上的交叉点为单位，来设计基本数据结构和描述机器博弈模型，本文称这样建立的机器博弈模型为传统模型。同时，大多数程序也都像文献[2-4]等那样加入了一些基于模式的知识。在很多棋类中，所谓“模式”，一般指从一个局面划分出来的具有某些强关联特征的局部区域。然而，文献[2-4]中模式的定义及描述限制了它在知识表达、局面状态描述以及局面评价上的能力，因为文献[2-4]没有回答下述问题：1）局面中的任一具体模式到底包含了哪些交叉点？2）模式的分类是否涵盖了所有的模式？3）局面中某一具体模式到底属于博弈的哪一方？4）任意两个模式之间是否存在演化关系，以及理性的博弈应当如何推动模式的演化？显然，文献[2-4]所描述的模式在局面中所占据的范围是模糊的，无法单纯依赖这样的模式描述或分析局面，还必须基于交叉点来描述局面状态。总之，尽管传统模型使用了模式，但无法充分发挥其优势，而且局面状态描述还必须依赖于交叉点，

所以是低效的。针对上述缺点，本文提出了基于“连珠”模式的增量更新的k子棋机器博弈模型CPBIM(*Connection-Pattern Based Incremental Model*)。

首先，本文提出了一种特殊的k子棋模式——连珠，还把文献[2]的模式分类方法扩展到了连珠的分类上。连珠有如下特点：有规范的定义，为每个连珠划定了严格的势力范围，并明确它属于博弈的哪一方；一般地，连珠是易于枚举的；在知识表达能力上更丰富也更精确；可建立恰当的映射，使每个局面都能和若干连珠的排列建立起一一对应关系。

其次，本文设计了若干模式库——连珠或空交叉点的类型知识库。通常，模式库包含模式相关的知识，由于吴在文献[2]并未提及其六子棋程序是否建立了模式库，因此本文将所建立的模式库与其它棋类的模式库做了如下比较：1）库涵盖了所有模式的知识，这一特点与文献[5-6]所介绍的残局库类似。2）库中的知识是抽象的知识，因而是稳定的。由于采取了模式和具体局面相分离的分析方法，保证了连珠的抽象知识在最大程度上描述了各连珠内各棋子联合所形成的整体效用，因而不必像文献[6]那样进行学习。3）与文献[5-6]等相比，本文建立的模式库非常小。4）不同于文献[7]的在线学习模式库，本文的模式库都能离线生成。

一般地，着法是驱动局面演化的直接原因，本文则从着法对连珠演化的驱动作用入手，得出评价所有连珠和每个连珠内部各空交叉点的推理方法。

再次，人们习惯于把一张棋盘看作由若干交叉点交织而成的组合体。由于发现任何一个k子棋的局面可以和若干连珠的排列建立一一对应关系，本文提出基于连珠的称为CPBM（*Connection-Pattern Based Model*）的高效的k子棋机器博弈模型，这使得局面的状态描述不必依赖于单个的交叉点，而依赖于作为整体的连珠。

最后，考虑到博弈树中结点众多，降低单个结点的更新代价意义重大。以围棋机器博弈为背景，Bouzy[9]曾提出过一种增量更新方法：对交叉点进行分类，从而选择性地更新。本文提出了一种更为积极的增量更新方法，仅更新并维护局面状态发生变化的部分，从而降低更新代价。增量更新和CPBM相结合所形成的新模型称为CPBIM。

本文的主要工作在于：1）提出并形式化地描述了一个特殊的模式——连珠。2）将连珠和领域知识做了良好的结合。发现了连珠间的演化关系，建立了基于连珠或空交叉点的若干领域知识库。3）将局面以连珠为核心进行数据结构设计，并由此建立了CPBM模型。4）提出了一种更为积极的增量更新的方法，结合CPBM模型得到CPBIM模型。CPBIM在六子棋机器博弈程序——“棋天大圣六子棋”(NEUConn6)中得以实现，该程序曾获中国机器博弈锦标赛六子棋组的冠军（CCGC, 2007.10，重庆杨家坪）和第13届世界机器博弈锦标赛六子棋组的铜牌（ICGA, 2008.10，北京房山区）[13]。

2 连珠及其诸属性

2.1 连珠及其属性的定义

棋盘上的任意交叉点，或者是空交叉点（未放入棋子），或者是非空交叉点（放入了黑子或白子）。记棋子s的颜色为$color(s)\in\{'B','W'\}$，令$not(\ )$表示颜色取反，则s的对手的棋子颜色为$not(color(s))\in\{'B','W'\}$。记交叉点$i$的颜色为$color(i)\in\{'B','W'\}$。

定义1 非空交叉点的颜色。对于放入棋子s的非空交叉点i，规定$color(i)\leftarrow color(s)$。

定义2 区间。在$Connect(n, n, k, p, q)$棋盘上的一条（水平的、垂直的或斜对角的）直线内，称一个连续的交叉点的序列为区间，当且仅当它是一个至少覆盖了k个连续交叉点的，且满足下述两个条件之一的最长序列：1）它的所有交叉点都是空交叉点；2）它的每个交叉点或者是空交叉点，或者是同色的非空交叉点。

定义3 空交叉点的颜色。在某个区间中，任意空交叉点i的颜色按下述原则确定：1）若所在区间含有棋子s，则令$color(i)\leftarrow color(s)$；2）若所在区间不含棋子，且至少一个边界不是棋盘的边界，则必有某棋子s紧邻区间的边界，并令$color(i)\leftarrow not(color(s))$；3）若所在区间不含棋子，且两个边界恰为棋盘的边界，则可从黑方角度令$color(i)\leftarrow'B'$，也可从白方角度令$color(i)\leftarrow'W'$。

特别地，定义3的第3）条指出，分别从黑方和白方的角度看，某个空交叉点可以有两种颜色，这与定义4和定义5有关。

定义4 连珠。在一个区间中，所有的同色的交叉点构成的序列，称为连珠，一般记作c。特别地，称不含任何棋子的连珠为空连珠。

定义5 连珠的颜色。c的颜色，记作$color(c)$。任取c的一个交叉点，设为i，规定$color(c)\leftarrow color(i)$。不引起混淆时，用“$c$”指代执$color(c)$颜色棋子的选手，用“$c$的对手”指代执$not(color(c))$颜色棋子的选手。

连珠的颜色表明该连珠属于哪一方。黑色的连珠属于黑方，白色的属于白方。对于不含任何棋子，且两个边界恰为棋盘边界的区间，实际上包含着两个连珠，一个属于黑方，另一个属于白方。于是，不难理解定义3的第3）条所描述的交叉点可以有两种颜色。

区间与连珠的相同点在于它们均描述了一个区域；不同点在于区间仅代表着一个区域，连珠是一个着色了的序列。例如，图1恰好有4个区间，分别用椭圆线圈起来，并且标注了字母，分别为：A、B、C、D（或E）。同时，图1有且仅有5个连珠，分别为：A、B、C、D和E。根据定义1~定义5：A是非空的黑连珠；B是非空的白连珠；C是空的黑连珠；D和E是两个彼此完全覆盖的空连珠，一黑一白，不妨令D是黑连珠，E是白连珠。容易看出，区间并不关心所代表区域的归属，而连珠则必须表明所代表的序列属于哪一方。

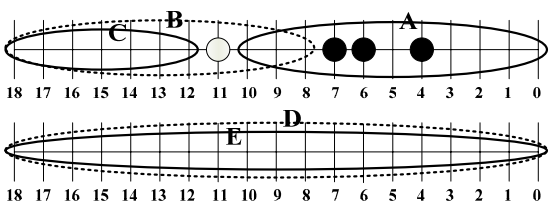


图1：在connect(19, 19, 6, 2, 1)中，线拆分成连珠的示例

请注意，图1中的连珠C只能是黑连珠。假如连珠C是白连珠，将得到矛盾的结论，因为：1）包含着它的连珠B是白色的，连珠长度|C|<|B|；2）根据连珠定义，任何连珠必然是同色交叉点的最长序列。显然，C不是

白色交叉点的最长序列，故矛盾。本节中提出的区间概念，仅用于辅助定义连珠，后续部分将不再讨论。

任何一条（水平的、垂直的、对角线的）线都可以无歧义地拆分成若干个连珠。连珠有下述涵义：首先，只有在连珠中才可能形成获胜的k子序列。其次，它严格地刻画了博弈双方各自所能影响到的最大势力范围。尽管这些势力范围之间可能会有物理上的重叠（例如，图1中的A和B，B和C分别部分地重叠；而D和E全部地重叠），各连珠之间还是保持了严格的在整体意义上的逻辑独立性。为方便进一步描述连珠，这里给出一种连珠内交叉点状态的二进制描述：令$s(c, i) = 1$，当且仅当c的第i个交叉点被一颗棋子占据；令$s(c, i) = 0$，当且仅当第i个交叉点上没有任何棋子。

定义6 连珠其它的几个重要属性。

● 连珠c的长度，记作$|c|$。c所拥有的交叉点的数目。

● 连珠c的形状，记作$\|c\|$。令$f(c) = \sum_{0\le i<|c|} s(c, i)\cdot 2^i$，$c$的形状$\|c\|$可由二元组$\|c\| = (|c|, f(c))$唯一地表示。其中，$f(c)=\sum_{0\le i<|c|} s(c, i)\cdot 2^i$是连珠中各个交叉点的二进制形式的状态表示。

● 连珠c的类型，记作$ct(c)$。后文将作详细介绍。

2.2 连珠及相关的形式化描述

形状是连珠极为重要的一个属性。根据连珠形状的定义可知：$\|c_1\|=\|c_2\|$，当且仅当1）$|c_1|=|c_2|$；和2）$f(c_1)=f(c_2)$同时成立。

定理1 连珠形状的非负整数表示。在$Connect(n, n, k, p, q)$中，令$f(c)= \sum_{0\le i<|c|} s(c, i)\cdot 2^i$，$g(c)= 2^{|c|}-2^k$，则连珠的形状$\|c\|$（这里，$\|c\|$作为一个记号，代表唯一标识$c$的非负整数编号）可由双射函数

$$\|c\|=f(c)+g(c)$$

唯一地表示。

证明：1）由$\|c_1\|=\|c_2\|$证明$f(c_1)+g(c_1)=f(c_2)+g(c_2)$。若$\|c_1\|=\|c_2\|$，必有$|c_1|=|c_2|$且$f(c_1)=f(c_2)$。因$|c_1|=|c_2|$，所以$g(c_1)=g(c_2)$，进而$g(c_1)+f(c_1)=g(c_2)+f(c_2)$。2）由$f(c_1)+g(c_1)=f(c_2)+g(c_2)$，证明$\|c_1\|=\|c_2\|$成立，也就是证明$|c_1|=|c_2|$且$f(c_1)=f(c_2)$成立。采用反证法。假设$|c_1|\ne|c_2|$成立。由$|c_1|\ne|c_2|$，不妨令$|c_1|=a$，$|c_2|=a+t$，其中$k\le a\le n$，$t$是常数，$0\le t$。令$G(a)=g(c_2)-g(c_1) = 2^{a+t}-2^a$，显然$0<G(a)$；而$F(a) = f(c_1)-f(c_2)$，必有$0<F(a)$且$0<f(c_1)$，故$F(a) \le f(c_1) \le 2^a-1$。而$F(a)-G(a) \le (2^a-1)-(2^{a+t}-2^a) = -1+2^{a+1}-2^{a+t}< 0$。故$F(a)\ne G(a)$，即$f(c_1)+g(c_1)\ne f(c_2)+g(c_2)$，这与已知$f(c_1)+g(c_1)=f(c_2)+g(c_2)$相矛盾。故$|c_1|\ne|c_2|$不成立。所以$|c_1|=|c_2|$，进而$g(c_1)=g(c_2)$。又由于$f(c_1)+g(c_1)=f(c_2)+g(c_2)$，故$f(c_1)=f(c_2)$。因此，$\|c_1\|=\|c_2\|$。

综合1）、2），定理1为真，证毕。

定理2 在$connect(n, n, k, p, q)$中，连珠的形状的总数$Sum = 2^{n+1} - 2^k$。

证明：在$connect(n, n, k, p, q)$中，所有连珠的可能长度为：$k, k+1, \ldots, n$。考虑长为k的所有连珠共有2^k个，长为$k+1$的连珠共有2^{k+1}个……依此类推。故，连珠总数$Sum = 2^k+2^{k+1}+\ldots+2^n = 2^{n+1} - 2^k$个。证毕。

推论1 定理1中代表所有连珠形状的非负整数编号必连续。

证明：由定理1知$\|c\|=f(c)+g(c)=\sum_{0\le i<|c|} s(c, i)\cdot 2^i +2^{|c|}-2^k$是$c$的编号。$0\le s(c, i)\le 1$，$k\le|c|\le n$。当$0\le i<|c|$，令$s(c, i)=0$，同时还令$|c|=k$，这时，$\|c\|$取得极小值，$\|c\| = 0$。当$0\le i<|c|$时，令$s(c, i)=1$，同时还令$|c|=n$，这时，$\|c\|$取得极大值，$\|c\| = \sum_{0\le i<n} 2^i+2^n-2^k = 2^{n+1}-2^k-1$。所以，对任意的$c$都有$0\le\|c\|\le 2^{n+1}- 2^k-1$，$\|c\|$可取得的非负整数最多只有$2^{n+1}-2^k$个。由定理1，任意两个不同的连珠编号都不相等。又由定理2，$connect(n, n, k, p, q)$恰有$2^{n+1}-2^k$个互不相同的连珠。所以，与$2^{n+1}-2^k$个连珠建立起一一对应关系的$2^{n+1}-2^k$个编号一定是从0到$2^{n+1}-2^k-1$的连续非负整数。证毕。

根据定理1、定理2和推论1，连珠的形状可采用连续的非负整数编码，其实际应用的意义在于：使得形状和内存地址之间能建立起一一对应关系。

定义7 （在一条线中）连珠的形式化描述。在$connect(n, n, k, p, q)$的一条线中，连珠可以由三元组

$c = (color(c), |c|, f(c))$，或二元组

$c = (color(c), \|c\|)$

描述。其中，$color(c)\in\{'B', 'W'\}$；$k\le|c|\le n$；$f(c) = \sum_{0\le i<|c|} s(c, i)\cdot 2^i$。

根据定义7，图1中的各个连珠可如下描述：A = ('B', 7, 208)或A = ('B', 2192)；B = ('W', 11, 8)或B = ('W', 1992)；C = ('B', 7, 0)或C = ('B', 64)；D = ('B', 19, 0)或D = ('B', 524224)；E = ('W', 19, 0)或E = ('W', 524224)。

2.3 连珠的威胁

定义8 威胁[2]。在六子棋中，考虑某个玩家——如白方，还没有形成连六。称黑方有t个威胁，当且仅当白方需要放置t颗棋子来防御对方下一着获胜。

定义8的威胁概念译自文献[2]。

定义9 连珠c的威胁数，记作$thn(c)$。称c有$thn(c)$个威胁，当且仅当c的对手需要放置$thn(c)$颗棋子来防御c在下一着获胜。

$thn(c)$是这样一个重要的度量：若轮到c的对手走棋，他至少需要$thn(c)$颗棋子才能化解c即将获胜的威胁；若轮到c走棋，只要$0<thn(c)$，c就肯定胜出。吴[2]还提出了计算威胁数目的滑动窗口法。

威胁的一个重要特征在于：当局面中出现威胁以后，对手应对的合法着法数目，远少于不存在威胁时的着法数目。善用这个特征，可以剪掉博弈树中很多不影响搜索结果的分枝，进而降低搜索树的平均分枝因子。幸运的是，在k子棋中，威胁是频繁出现的。在具有上述特点的棋类中，Allis提出一种非常重要的搜索算法——基于威胁的搜索算法[10]。运用该算法，文献[11]解决了（带禁手的）Renju文献[12]解决了（无禁手的）Go-Moku。在六子棋中，基于威胁的搜索策略也是最重要的搜索策略之一。

3 连珠的估值

在大多数棋类中，如何较为准确地评价那些错综复杂的局面往往是一个极具挑战性的难题。本文把上述难题分解为两个步骤：1）把局面拆分成若干个连珠的组合。遵循第2.1节的介绍，可实现基于连珠的局面拆分。局面中所有的连珠代表了所有同色棋子间最直接和紧密的所有联系，因此将局面进行这样的拆分不会损失信息。2）估值每个连珠。若能准确地估值每个连珠，便可累加各个连珠的估值得到局面的估值。可见，准确地估值各个连珠是局面估值无法逾越的步骤。

3.1 连珠估值的二步法

在具体上下文环境（局面）中的各个连珠往往彼此影响，其估值问题仍是一个难题，因为它们的价值可能因所在的上下文环境而发生变化。依据循序渐进的原则，把复杂的连珠估值问题分解成相对容易的两个子问题：A）“抽象估值”问题。即，抛开具体局面，怎样估值各个连珠？B）“具体估值”问题。在具体的局面之中，各个连珠之间会彼此影响，围绕“抽象估值”，其实际估值可能有不同幅度的波动，“抽象估值”就是要削弱这种误差。基于上述分析，提出两步法来解决连珠估值的难题。

第一步，得到连珠的“抽象估值”。抽象估值是抛开具体局面的连珠估值，是对连珠内部的所有棋子间联络的整体效用的量化评价。对于一个理想的模式估值方案而言，它不仅应比较出任何两个模式孰优孰劣，还应通过估值之差准确地量化差距。在*connect*(19, 19, 6, 2, 1)中，连珠共有1,048,512个，其估值难度可想而知。既然如此，可否先将连珠分类，然后逐类估值呢？答案是肯定的。在第3.2节中，NEUConn6已经把1,048,512个连珠归纳成12种类型，同时保证了同类型连珠之间在价值上的差别甚微，以至于完全可以忽略。这样，可认为类型相同的连珠其“抽象价值”也相同。于是，连珠类型就有资格取代与之相关的具体连珠参与估值，在不失准确性的情况下大幅降低估值的难度。

第二步，得到连珠的“具体估值”。显然，存在游离于任何局面之外的连珠只是一种理想化的假设而已。在博弈过程中，估值任何连珠必将在它所属的局面中进行。既然“抽象估值”是排除了上下文干扰的一个基本估值，当把连珠还原到具体局面中加以考虑时，其具体估值也必然要根据典型的上下文环境，在抽象估值基础上做不同幅度的调整。实施具体估值的困难在于引起估值变化的情况特别多，需要逐一分类，以便分别地处理。但是，在比较复杂的棋类中，如六子棋，引起具体估值变化的情况庞杂而且琐碎，缺乏有效的分类方法，往往都只是借助专家们的或程序设计者的离散的领域知识，粗略地分类罢了。一般地，在*alpha-beta*这类对估值准确程度要求较高的搜索算法中，细致地调整“具体估值”都能提高程序的博弈水平。此外，也可采用高级些的方法，包括机器学习、神经元网络等，但达到一定水平后，就很难再作提高。

在连珠估值的两步法中，步骤一至关重要，而步骤二在特殊情况下甚至不是必须的。特别地，在本文的实验部分采用了基于威胁的搜索算法，利用抽象估值就足够了，步骤二可省略，故不加赘述。

3.2 连珠的类型

基于以上介绍，如何获得连珠的类型已成为两步法最关键的一环。本文将吴的模式分类方法扩展到连珠的分类上，将所有的连珠分为12种类型。其分类依据参考了文献[2]，但重新归纳为以下三个基本标准：直接（现实）威胁和间接（潜在）威胁；棋子间亲密程度；进攻能力和防守需求。

定义10 基于直接威胁和间接威胁的连珠分类。

● 作为游戏的终极目标，*‘WIN’*类型的连珠包含了一个六颗棋子的连续不间断序列，如图2(*a*)所示。

● 一个DTC(*Direct Threat Connection*)是这样的一个连珠：通过增加1颗或2颗棋子，它可以形成一个新的′*WIN*′类型连珠，如图2所示的(*b*)~(*g*)；一般地，若*c*是一个DTC，往往0<*thn*(*c*)≤2。其中，唯有′*DW*′(*Definitely Win*)类型的连珠特殊，2<*thn*(*c*)。

● 一个ITC(*Indirect Threat Connection*)是这样的一个连珠：增加1颗或2颗棋子可以形成一个新的DTC，如图2的(*h*)~(*k*)。若*c*是一个ITC，则*thn*(*c*)=0。

定义11 基于棋子亲密程度的连珠分类。

● 窗口中棋子间的亲密程度——*l*亲密窗口（0≤*l*≤6），简称*l*窗口。考虑包含6个交叉点的固定大小的窗口，从连珠的最左侧滑动到最右侧，则称某个滑动窗口为*l*窗口，当且仅当它包含*l*颗棋子。

● 连珠中棋子间的亲密程度——*l*亲密连珠（0≤*l*≤6），简称*l*连珠。称某个连珠*c*为*l*连珠，记作*l*(*c*)，当且仅当*c*中存在着*l*窗口，且不存在任何的*l*′（*l*≤*l*′）窗口。一般地，对于一个DTC连珠*c*，总有4≤ *l*(*c*) <6；对于一个DTC连珠*c*，总有*l*(*c*)<4。

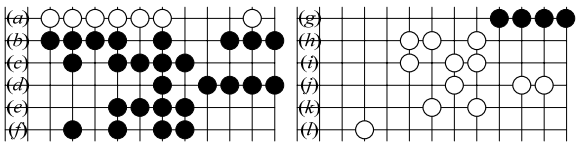
定义12 基于进攻能力和防守需求的连珠分类。

● 称*c*是一个*live-l*，当*c*是个*l*连珠，且它满足下述情形之一：1）*thn*(*c*)=2；2）*c*是一个ITC，通过给*c*增加4–*l*颗*c*的棋子，可以形成*live*-4。

● 称*c*是一个*sleep-l*，当*c*不是一个*live-l*，且满足下述情形之一：1）*l*(*c*) = 4且*thn*(*c*)=1，通过给*c*增加一颗*c*的棋子，可形成一个新的*live*-5；2）*l*(*c*) = 3，通过给*c*增加一颗*c*的棋子无论如何都不能形成*live*-4，但通过给*c*增加某两颗*c*的棋子必能形成一个新的*live*-4或*live*-5。

● 称*c*是一个*dead-l*连珠，当*c*是个*l*连珠，且*c*既不是一个*live-l*，也不是一个*sleep-l*。

简记*live-l*连珠的类型为′L_l′；*sleep-l*连珠的类型为′S_l′；*deal-l*连珠的类型为′D_l′。根据定义10~定义12的三个标准，最终得到了15种具体的连珠类型。令连珠类型的集合为：S_{CT} = {′*WIN*′, ′*DW*′, ′L_5′, ′D_5′, ′L_4′, ′S_4′, ′D_4′, ′L_3′, ′S_3′, ′D_3′, ′L_2′, ′D_2′, ′L_1′, ′D_1′, ′*E*′}。其中，′*E*′代表空连珠。NEUConn6中，几乎所有局面中的′D_2′、′L_1′、′D_1′和′*E*′类型连珠比例都是最多且重要性也最低的，因此将上述四种类型合并为类型′*O*′，最终共得到12种类型。



(*a*) *WIN*′; (*b*) *DW*′; (*c*) L_5′; (*d*) D_5′; (*e*) L_4′; (*f*) S_4′; (*g*) D_4′; (*h*) L_3′; (*i*) S_3′; (*j*) D_3′; (*k*) L_2′; (*l*) *O*′;

图2：各种类型的连珠举例

定义13 连珠的类型。在*Connect*(*n*, *n*, 6, 2, 1)中，共有12种连珠类型：′*WIN*′、′*DW*′、′L_5′、′D_5′、′L_4′、′S_4′、′D_4′、′L_3′、′S_3′、′D_3′、′L_2′、′*O*′。

令包含上述连珠类型的全集为S_{CT}。此外，还令S_{DCT} = {′*DW*′, ′L_5′, ′D_5′, ′L_4′, ′S_4′, ′D_4′}；S_{ICT} = {′L_3′, ′S_3′, ′D_3′, ′L_2′, ′*O*′}为ITC连珠的类型集合。连珠分类与吴的模式分类主要的不同在于：1）从′D_3′中分离出′S_3′，从′D_4′中分离出′S_4′；2）增加了′*WIN*′、′*DW*′类型；3）把局面中最多见的、作用却不大的几种类型归结为′*O*′类型。连珠分类的优点：1）′S_3′和′S_4′分别比′D_3′和′D_4′更有威胁，区分其间的差别，必将提高局面评价的准确性；2）

*'O'*类型包括了低有效性却高频地出现于实战局面的连珠，细分它们只会降低程序效率，并不会改善对局面的评价水平；3）任意连珠都有了一个确定的类型，从而保证分类的完备性。

3.3 求解全部连珠类型的算法

根据定义6，连珠c由颜色$color(c)$和形状$||c||$共同决定。其中，颜色仅表明它属于博弈的哪一方，而形状则蕴含着棋子间亲密程度、进攻和防守等因素，因而对价值起决定作用。相同形状的黑连珠和白连珠的类型必然是一样的，3.3节所提到的连珠均指黑连珠。

求解所有连珠类型，只要顺次执行下述步骤：1）初始化。初始化所有的连珠c的类型为未知类型，即$ct(c) \leftarrow Unknown$；2）枚举获胜的连珠。枚举出所有*'WIN'*类型连珠c，并令$ct(c) \leftarrow$ *'WIN'*；3）第一次遍历。遍历每个$ct(c) = Unknown$的连珠，对于类型为t的DTC，令$ct(c) \leftarrow t$；4）第二次遍历。遍历每个$ct(c) = Unknown$的连珠，对于类型为t的ITC，令$ct(c) \leftarrow t$。5）结束。

其中，步骤3）和步骤4）分别实现两个类型识别器，见算法1和算法2。算法1实现了一个能识别出一个DTC连珠的称为DTCC（*Direct Threat Connection Classifier*）的识别器；算法2实现了一个能识别出一个ITC连珠的称为ITCC（*Indirect Threat Connection Classifier*）的识别器。

算法1：DTCC的实现

```
/*注意：在连珠c内，把固定大小的（覆盖6个交叉点的）滑动窗口从最左边开始向右移动的过程中，用w表示当前窗口是上述操作中的第几个窗口（即序号）；nStonesIn(w)是w中的棋子数目；isMarked(w)指出w中有没有标记过的空交叉点。 */

1.  FOR a Connection c {
2.      nThreatWindows ← 0; isFive ← false; isFour ← false;
3.      FOR (w=0; w<|c|-6; ++w) {
4.          IF (6 = nStonesIn(w)) {
5.              ct(c) ← 'WIN';    BREAK;
6.          }
7.          ELSE IF (4 ≤ nStonesIn(w) AND false = isMarked(w)) {
8.              nThreatWindows ← nThreatWindows+1;
9.              Marking the rightmost empty intersection;
10.             IF (4 = nStonesIn(w))   isFour ← true;
11.             IF (5 = nStonesIn(w))   isFive ← true;
12.         }
13.     }

14.     IF (3≤nThreatsWindows)  ct(c) ← 'DW';
15.     ELSE IF (2 = nThreatsWindows AND true = isFive)   ct(c) ← 'L5';
16.     ELSE IF (2 = nThreatsWindows AND true = isFour)   ct(c) ← 'L4';
17.     ELSE IF (1 = nThreatsWindows AND true = isFive)   ct(c) ← 'D5';
18.     ELSE IF (1 = nThreatsWindows AND true = isFour) {
19.         FOR (i=0; i<|c|; ++i) {
20.             IF(1 = s(c, i))   CONTINUE;
21.             IF(('L4'= ct(putStone(c, i))) OR 'L5' = ct(putStone(c, i)))
                  AND ct(c) < 'S4') {
22.                 ct(c) ← 'S4';    BREAK;
23.             }
24.         }
25.     ct(c) ← 'UNKNOWN';
26. }
```

算法2: ITCC的实现

```
/*注意：nMax用于记录窗口中的棋子数目；putStone(c, i)在第i个交叉点上放入一颗本方的棋子；putStone(c, i, j)分别在第i和第j个交叉点上放入本方的棋子。 */

1.  FOR a Connection c {
2.      IF ('UNKNOWN' ≠ ct(c)))   CONTINUE;
3.      nMax ← 0;
4.      FOR (w=0; w<|c|–6; ++w) {
5.          IF (nMax < nStonesIn(w))    nMax = nStonesIn(w);
6.      }

7.      IF (3 = nMax) {
8.          FOR (i=0; i<|c|; ++i) {
9.              IF(1 = s(c, i))   CONTINUE;
10.             ELSE IF('L4' = ct(putStone(c, i)) AND ct(c) < 'L3') {
11.                 ct(c) ← 'L3';    BREAK;
12.             } ELSE {
13.                 FOR (j=i+1; j<|c|; ++j) {
14.                     IF(1 = s(c, j))   CONTINUE;
15.                     IF ('L5 =ct(putStone(c, i, j)) AND ct(c)<'S3') {
16.                     ct(c) ← 'S3';    BREAK;
17.                 }
18.             }
19.         }
20.         ct(c) ← 'D3';
21.     } ELSE IF (2 = nMax) {
22.         FOR (i=0; i<|c|-1; ++i) {
23.             FOR (j=i+1; j<|c|; ++j) {
24.                 IF(1 = s(c, i) OR 1 = s(c, j))   CONTINUE;
25.                 ELSE IF ('L4' = ct(putStone(c, i, j))) {
26.                     ct(c) ← 'L2';    BREAK;
27.                 }
28.             }
29.         }
30.     } ELSE { ct(c) ← 'O'; }
31. }
```

经历上述步骤之后，六子棋全部的1,048,512个连珠被分成12种类型。表1统计了每种类型所包含的不同的连珠形状的数目和占总数的百分比。其中，*CT*是连珠类型；*Num*是连珠形状的数目；*Per*是该类型连珠形状占总数的百分比。

表1：在*connect*(19, 19, 6, 2, 1)中连珠的相关统计

CT	Num	Per	CT	Num	Per
WIN	112896	10.767	L_3	86913	8.289
DW	192916	18.399	S_3	55085	5.254
L_5	181781	17.337	D_3	2186	0.208
D_5	45952	4.383	L_2	14425	1.376
L_4	149319	14.241	O	5239	0.450
S_4	191025	18.219			
D_4	10775	1.028		1048512	100.00

3.4 构造连珠知识库

现在，所有连珠的类型都已经求出。在搜索过程中对连珠类型的分析是异常频繁的，若空间需求可承受，考虑到3.3节的算法完全可以离线计算，应把所有形状的连珠类型存储在一个表中，称该表为CBKDB（*Connection Based Knowledge DataBase*）。这样，在线的计算就转化为离线的计算和在线的查询。对于*connect*(19, 19, 6, 2, 1)，CBKDB要存储1,048,512个表项，每一个占$\lceil \log 12 \rceil = 4$ bit，总共只需大约512Kb的内存空间。表2给出了几种k子棋知识库所需空间的大小。

构造CBKDB还有一个关键的问题：怎样为每个连珠分配一个唯一的入口地址？利用2.2节的定理1，该问题很容易解决：令$f(c)=\sum_{0\le i<|c|} s(c, i)\cdot 2^i$，$g(c)= 2^{|c|}–2^k$，任意一个连珠$c$在知识表中的入口地址为$f(c)+g(c)$。于是，CBKDB的第$f(c)+g(c)$个表项便用来存储$c$的连珠类型。

表2：几种k子棋知识库的大小

棋类	六子棋			五子棋
	19×19	15×15	13×13	15×15
大小	511.969Kb	31.969Kb	7.969Kb	31.984Kb

4 连珠之间的演化

连珠的12种类型绝非彼此孤立的，而是存在着客观联系的。任何一个连珠受新增一颗棋子的驱动，都

能演化成别的0、1或2个连珠。若所形成的不显著的演化，则它对局面的影响可以忽略；若该棋子引起了显著的演化，则该演化必须纳入到对局面的分析之中。

4.1 升变

定义14 直接升变。已知$0\le i<|c|$，$s(c, i)=0$，令$c'=putStone(c, i)$。若$ct(c)\neq ct(c')$，则称c可直接升变到c'。其中，c称为升变的原连珠；c'为升变的目标连珠。

已知$0\le i<|c|$，$s(c, i)=0$，令$c'=putStone(c, i)$。若$ct(c)=ct(c')$，则c与c'同等价值；若$ct(c)\neq ct(c')$，则c'优于c，因为刚增加的c的棋子会增强而非削弱c的进攻能力。

定义15 直接升变的可行性。若存在从连珠c到c'的直接升变，则从类型$ct(c)$到$ct(c')$的直接升变是可行的，记作$ct(c)\rightarrow ct(c')$。称$ct(c)$为原类型，$ct(c')$为目标类型。

通过程序，可找出类型间的所有可行的直接升变，它们由图3的有向无环图描述。图中，每一个箭头表示一个可能的直接升变；箭头起点处的结点代表原类型；箭头所指向的结点代表目标类型。

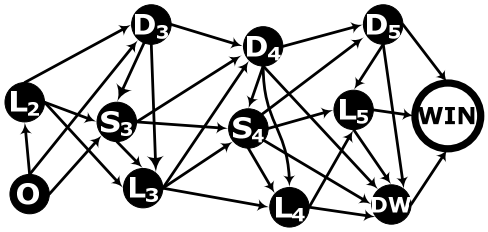


图3：连珠类型间的所有的可行的直接升变

定义16 间接升变（递归定义）。当存在c''，使得从c可直接升变或间接升变到c''，且从c''可直接升变或间接升变到c'，则c可以间接升变到c'。

定义17 间接升变的可行性。若存在从c到c'的间接升变，则从类型$ct(c)$到$ct(c')$的间接升变是可行的。

定义18 升变。直接和间接升变统称为升变。

定义19 升变的可行性。若存在从c到c'的升变，则从类型$ct(c)$到$ct(c')$的升变是可行的，记作$ct(c)\Rightarrow ct(c')$。

从图3很容易知道可行的升变有哪些。根据定义16，任何间接升变都能分解为一系列的直接升变。此外，还应注意到并非任意两个类型t和t'间都存在可行的升变，即$t\Rightarrow t'$和$t'\Rightarrow t$同时不成立。例如，$'L_4'$和$'D_5'$类型之间就不存在任何升变关系。

定义20 间接升变路径。把间接升变分解为一系列的连续的直接升变时，其分解方案往往不唯一，每一种方案都构成了一条间接升变路径。

例如，$'S_3'\Rightarrow'L_4'$存在着5条升变路径：1）$'S_3'\rightarrow'L_3'\rightarrow'L_4'$；2）$'S_3'\rightarrow'L_3'\rightarrow'S_4'\rightarrow'L_4'$；3）$'S_3'\rightarrow'D_4'\rightarrow'S_4'\rightarrow'L_4'$；4）$'S_3'\rightarrow'S_4'\rightarrow'L_4'$；5）$'D_3'\rightarrow'D_4'\rightarrow'L_4'$。

定义21 升变路径的长度。直接升变的路径长度总是1；可分解为l个直接升变的间接升变路径，其长度为l。

仍然考虑$'S_3'\Rightarrow'L_4'$，其5条间接升变路径的长度分别为3、4、4、3、3。升变路径的长度实际上表明至少需要投入多少颗棋子才能实现该升变路径。既然目标类型相同，显然路径长度越短越好。具体地，升变路径1）、4）和5）好于升变路径2）和3）。

4.2 升变间的比较和连珠类型的抽象估值

若原类型和目标类型均已确定，那么升变路径越短当然越好。事实上，升变之间的比较，下述情况更常见：原类型是确定的，且相关的升变存在多个，到底哪种升变才是最佳的？

定义22 相同原类型的直接升变间的可比较性。若t为直接升变的原类型，$\{t_0', t_1', \ldots, t_i, \ldots, t_n'\}$是$t$可直接升变的目标类型的集合，若对任意的$0\le i<j\le n$，总有$ct(t_i)\Rightarrow ct(t_j)$或者$ct(t_j)\Rightarrow ct(t_i)$，则称这两个升变之间是可比较的；否则称这两个升变之间是不可比较的。

根据4.1节，从升变的角度考虑，若$ct(c)\Rightarrow ct(c')$，则说明类型$ct(c')$好于$ct(c)$，以及c好于c'。换言之，升变关系表达了优劣意义上的一种比较关系。在图3中，考虑两个例子。第一个例子：$'L_3'\rightarrow'D_4'$、$'L_3'\rightarrow'S_4'$、$'L_3'\rightarrow'L_4'$是以$'L_3'$为原类型的所有的直接升变，注意到$'D_4'\rightarrow'S_4'$、$'S_4'\rightarrow'L_4'$，因此，$'L_3'$的所有的（3个）升变之间是可比较的。据此，可以得出如下判断：$'L_3'$升变成$'L_4'$是最好的，升变成$'S_4'$次之，升变成$'D_4'$更次之。第二个例子：$'D_4'\rightarrow'D_5'$、$'D_4'\rightarrow'S_4'$、$'D_4'\rightarrow'L_4'$是$'D_4'$为原类型的所有的直接升变，注意到$'D_5'$和$'L_4'$之间不存在升变关系，因此，$'D_4'$的两个升变之间是不可比较的。既然两个升变之间是不可比较的，那么，就无法断定：$'D_4'$升变成$'D_5'$好呢，还是升变成$'L_4'$好？

定义23 连珠c的最佳直接升变。设$\{c_0, c_1, \ldots, c_i, \ldots, c_n\}$是可由$c$直接升变的目标连珠的集合，若$j\neq i$且$0\le j\le n$时，总有$ct(c_i)\Rightarrow ct(c_j)$或$ct(c_i)=ct(c_j)$成立，则称从$c$到$t_j$的升变是$c$的最佳升变。

定义23指明一个连珠有最佳升变的条件。其中，条件“$ct(c_i)=ct(c_j)$”表明可有多个连珠同时成为c的最佳升变的目标连珠。

理想地，若任何两个类型之间都是可比较的，那么任意两个类型之间的优劣意义上的差距就可以被量化。注意到图3不是一个单侧连通图，因为$'L_4'$和$'D_5'$之间不是单侧连通的。为了让它成为单侧连通图，通过添加一个虚构的关系，让$'L_4'$和$'D_5'$成为单侧连通的。考虑到$'D_5'$只需要再增加一颗子就称为*WIN*类型，而$'L_4'$还需要增加两颗棋子才能成为*WIN*类型，令虚构出来的升变关系为$'L_4'\rightarrow'D_5'$似乎更合理，如图4所示。图4是一个单侧连通图，表示一个严格偏序关系。

根据图4，定义连珠类型在上的一个严格偏序关系$'\angle'$为：$O\angle L_2\angle D_3\angle S_3\angle L_3\angle D_4\angle S_4\angle L_4\angle D_5\angle L_5\angle DW\angle WIN$。这里，介于两个类型之间的$'\angle'$符号应读作“严格劣于”。同样也存在两个一般的规则：1）$dead\text{-}l\angle sleep\text{-}l\angle live\text{-}l$；2）$T\text{-}l_1\angle T\text{-}l_2$，其中$2\le l_1<l_2<5$，并且$T$取*dead*，*sleep*，或*live*之中的某一个。

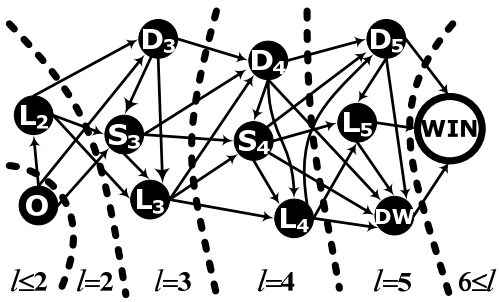


图4：连珠类型间的所有的可行的直接升变

根据图4，令连珠类型的抽象估值的一个严格偏序关系$'\langle'$为：$val(O)\langle val(L_2)\langle val(D_3)\langle val(S_3)\langle val(L_3)\langle val(D_4)\langle val(S_4)\langle val(L_4)\langle val(D_5)\langle val(L_5)\langle val(DW)\langle val(WIN)$。上述偏序关系代表了连珠类型的优劣，其中$val(T)$代表的类型$T$的“抽象估值”。图4还反映了连珠类型抽象估值的两个一般的规则：1）$val(dead\text{-}l)\langle val(sleep\text{-}l)\langle val(live\text{-}l)$；2）$val(T\text{-}l_1)\langle val(T\text{-}l_2)$。一般地，可令$val(O)=0$，表明$O$类型连珠几乎没有价值。

4.3 直接升变的简化

图4存在着冗余的升变，包括：1）为了建立类型间的严格偏序关系，虚构出来的$'L_4'\rightarrow'D_5'$的升变。2）$'D_5'\rightarrow'L_5'$、$'D_5'\rightarrow'DW'$、$'L_5'\rightarrow'DW'$这3种升变也是无意义的，因为$'D_5'\rightarrow'WIN'$、$'L_5'\rightarrow'WIN'$已经能直接获胜，经由$'L_5'$或$'DW'$再升变为$'WIN'$显然是多余的。所有的有效升变可由图5中实线箭头和结点所构成的有向无环图表示，虚线箭头表示冗余的升变。

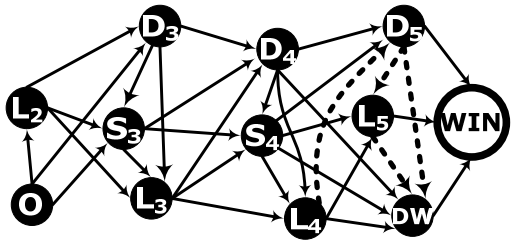


图5：去除冗余的直接升变图

在六子棋中，由于一次要选出两个空交叉点组合成一个着法(m_1, m_2)，其中，m_1表示第一次选出的空交叉点，m_2表示第二次选出的空交叉点，因此它的直接升变应分两种情况讨论。当选第一个空交叉点m_2时，还可以下两颗棋子，走棋那只能再下一颗棋子，在程序中，常以二维表加以描述，如表3。当选第一个空交叉点m_1时，还可以下两颗棋子。表4仅给出了选m_2与选m_1时升变的变化部分。在表3和表4中，小写的类型代表升变的原类型；大写的类型代表目标类型。$'\times'$表示升变不可行，$'\surd'$表示升变可行。六子棋程序可依赖上表对升变的可行与否做出判断。

表3：直接升变关系的二维表描述（灰色单元格仅适用于m_2）

	WIN	*DW*	L_5	D_5	L_4	S_4	D_4	L_3	S_3	D_3	L_2
dw	√	×	×	×	×	×	×	×	×	×	×
l_5	√	×	×	×	×	×	×	×	×	×	×
d_5	√	×	×	×	×	×	×	×	×	×	×
l_4	×	√	√	×	×	×	×	×	×	×	×
s_4	×	√	√	×	×	×	×	×	×	×	×
d_4	×	√	×	×	√	×	×	×	×	×	×
l_3	×	×	×	×	√	√	√	×	×	×	×
s_3	×	×	×	×	×	√	√	√	×	×	×
d_3	×	×	×	×	×	×	√	√	√	×	×
l_2	×	×	×	×	×	×	×	√	√	√	×
o	×	×	×	×	×	×	×	×	√	√	√

表4：一部分直接升变关系的二维表描述（仅适用于m_1）

	WIN	*DW*	L_5	D_5	L_4	S_4
l_4	×	×	√	×	×	×
s_4	×	×	√	√	×	×
d_4	×	×	×	√	×	×

4.4 降格

定义24 直接降格。$0\le i<|c|$，$s(c, i)=0$，在c的第i个交叉点上放入棋子s且$color(s) = not(color(c))$，把c被切割成两部分，分别设为c_1（覆盖交叉点0~i–1）和c_2（覆盖交叉点i+1~|c|-1）。若$ct(c_1)\angle ct(c)$且$ct(c_2)\angle ct(c)$，称c被直接降格为c_1和c_2。

直接降格是从c的对手的角度考虑，在c中新增c的对手的棋子，看能否把c削弱成为较差的类型。直接降格和直接升格有很大不同：直接升格只能遵照图3~图5所描述的规律；而直接降格，则可能将某种（除了$'WIN'$、$'DW'$外的）类型降格为任何更差的类型。

5 空交叉点的类型

5.1 空交叉点的价值

棋局的进一步演化主要取决于博弈双方即将选择哪些着法，那些有子的交叉点已经失去被选择的机会。

定义25 非空交叉点*i*的价值。连珠中的非空交叉点i的价值是0。

定理 3 已知$0\le i<|c|$，$s(c, i)=0$。在c的第i个交叉点上放入棋子s，有：A）若$color(s) = color(c)$，能使c升变。B）若$color(s) = not(color(c))$，能使c降格。那么：1）“A）成立”和“B）成立”是等价命题；2）“A）不成立”和“B）不成立”是等价命题。

证明：1）由“A）成立”证明“B）成立”。若第i个交叉点放入c的对手的棋子，使c降格，则第i个交叉点必位于包含棋子数最多的某个（包含6个交叉点的）窗口中。于是，i中放入c的棋子，必会使得c升格。同理可证，若“B）成立”则“A）成立”。2）由反证法易证，过程略。证毕。

定理3从升变和降格的角度说明：理性的博弈双方“英雄所见略同”。即，对于连珠中的某一空交叉点i，它或者应受到博弈双方同时的关注，或者不应受到任何一方的关注。所以，对于连珠c中的任何空交叉点，从c的或者从c的对手的角度评价，其结果是一致的。

定义26 空交叉点*i*的类型，记作*ti*(*i*)。若在连珠c中的一个空交叉点i中放入一颗c的棋子使c升变为c'，则$ti(i)\leftarrow ct(c')$；否则，$ti(i)\leftarrow 'O'$。

令所有的空交叉点类型集合为S_{IT}，那么$\forall ct(c')\in S_{IT}$，都有$ct(c')\in S_{CT}$，故$S_{IT}\subseteq S_{CT}$。定义26使得空交叉点的类型集合和连珠的类型集合相同，简化了六子棋中的类型体系。

定义27 空交叉点*i*的价值。若在连珠c中的一个空交叉点i中放入一颗c的棋子，使得$c\rightarrow c'$成立，令该交叉点的价值为$v(c, i) = val(tc(c'))$；否则令$v(c, i) = 0$。

普遍认为六子棋的难度在于分枝因子，其原因有二：其一，棋盘大，有361个交叉点，其中的每一个都是合法的候选点；其二，六子棋每次下两颗子。假设下每一颗子的时候，平均约有R个候选点，则每个局面平均约有R^2个候选着法（即，点对 (m_1, m_2)）。根据定义26和定义27，着法列表可以安全地拒绝那些价值为0的空交叉点，而不必担心因此有所遗漏。常见的选择候选着法的方法，如文献[3]等，往往依据局面中棋子密集程度，在其周围画出一定范围内，其中的空交叉点将都纳入候选着法列表。该方法不但会导致某些重要

的空交叉点被遗漏，也会导致有些不必要的空交叉点又被纳入着法列表。

算法3实现了一个能识别出空交叉点类型的称为TCIC（*Type Classifier for Intersection of Connection*）的识别器。和求解所有连珠类型一样，求解所有空交叉点的类型也可以离线计算，并建立一个空交叉点类型知识库。在六子棋中，其大小为10Mb左右。

算法3: TCIC的实现

```
1.  FOR a Connection c {
2.      IF (ct(c) = 'WIN')  CONTINUE;
3.      ELSE {
4.          FOR each intersection i {
5.              IF (0 = s(c, i)) {
6.                  IF (ct(c)<ct(putStone(c, i)))    ti(i) ← ct(putStone(c, i));
7.                  ELSE                             ti(i) ← 'O';
8.              }
9.          }
10.     }
11. }
```

5.2 空交叉点的复合类型

至此，任何一个连珠中的任何一个交叉点都有了各自的类型。不过，各个交叉点的评价最终还必须在具体上下文（局面）中进行。设轮到黑走棋，空交叉点i在四个黑连珠中的类型分别为$t_{b,1}$, $t_{b,2}$, $t_{b,3}$, $t_{b,4}$；在四个白连珠分别为$t_{w,1}$, $t_{w,2}$, $t_{w,3}$, $t_{w,4}$。考虑到着法选择的时候，要考虑到进攻和防守的因素，将每个点都有两个复合类型，分别从黑方和白方的角度考虑。点i的黑色复合类型表示为向量$(t_{b,1}, t_{b,2}, t_{b,3}, t_{b,4})$，白色复合类型表示为向量$(t_{w,1}, t_{w,2}, t_{w,3}, t_{w,4})$。向量中的每个分量是符合类型的分类型。因为$(t_{b,1}, t_{b,2}, t_{b,3}, t_{b,4})$中的任意一个$t$都有12种类型，故黑色的复合类型约有$12^4$=20736种。若将各个分类型的组合相同的复合类型视为相同类型，则有C(12+4–1, 4) = C(15,4) = 15!/(4!×(15–4)!) = 1365种，其中，C(n, r) = n!/(r!×(n –r)!)是组合数公式，表示在n个不同的元素中任取r个元素的不同组合总数。

在基于威胁的搜索算法中，评估1365种复合类型，仍可采用分类的方法。这时的分类实质是依据经验的一种知识启发。NEUConn6根据经验把复合类型大致分为五类，并令复合类型集合S_{MT} = {'*WIN*', '*GOOD*', '*NORMAL*', '*BAD*', '*USELESS*'}，还建立了一个复合类型知识表*MultiTbl*[T_1][T_2][T_3][T_4] = {mt_0, mt_1, mt_2, …, mt_i, …, mt_{20736}}，其中0≤i<20736时，mt_i∈S_{MT}。*MultiTbl*[ ]的大小为12^4×⌊log12⌋/8=10kb左右。若在*alpha-beta*这类强调估值准确性的搜索算法中，除了采取上述措施外，还应对1365种复合类型进行细致的估值。

引入复合类型及其分类的优点：1）很低的代价就能提高着法排序。2）安全地剔除无用的空交叉点。

6 模型的数据结构设计

基于交叉点的局面状态表示是最常用的方法。在*Connect*(n, n, k, p, q)中，若n×n个交叉点的状态确定，则棋盘的状态亦随之确定。这种表示最大的缺点是它导致棋子之间的联系是松散的。以连珠为核心来描述的机器博弈模型，要确定局面的状态，只要确定表示它各个连珠的状态即可。该方法可使基于棋子间联系的评价、分析都变得容易且高效。本节主要讨论如何以连珠为核心描述CPBM模型。

6.1 潜在的连珠

首先，介绍如何用连珠来描述一条线。

定义28 （在一条线中）潜在的连珠，记作*uc*。已知*connect*(n, n, k, p, q)的一条x长线最多可容纳s个黑连珠或t个白连珠，则它有潜在的s个黑连珠和t个白连珠。

在一条线中，令实际存在的黑连珠数目为s'，白连珠数目为t'；潜在的黑连珠数目为s，潜在的白连珠数目为t。显然，根据定义28，s'≤s，t'≤t。接着，建立s'个黑连珠和s'个潜在黑连珠之间的一一对应关系，以及t'白连珠与t'个潜在的白连珠之间的一一对应关系。考虑为每个潜在连珠这样赋值：若某个潜在的连珠*uc*和某个实际存在的连珠c建立了对应关系，则令*uc*←c；否则，令*uc*←ϕ。其中，值ϕ表示不存在连珠。这样，所有实际存在的连珠就填充到对应的潜在的连珠中去了，线也就可由连珠表示。接下来必须弄清楚，一条x长线到底最多能容纳多少个黑连珠和白连珠，见定理4。

定理4 给定一条x长的线，$\gamma(x, k)$=⌊(x+1)/(k+1)⌋。它最多能容纳$\gamma(x, k)$个黑连珠或$\gamma(x, k)$个白连珠。这里，⌈d⌉取十进制数d的整数部分。

证明：由于颜色的对称性，要证明上述定理，只要证明在x长的线中最多有$\gamma(x, k)$个黑连珠即可。在一条x长的线内，令t是黑连珠的数目。考虑极端情况，用t–1颗白子分割整条线得到：1）t–1个k长的黑连珠；以及2）一个命名为c_0的(x–(k+1)×(t–1))长的黑连珠。这时，k≤|c_0|≤2k，且此时的t必是黑连珠的最大数目。注意到，既然|c_0|从k变化到2k的时候，t值始终保持不变，不妨令|c_0|=k。于是，x–(k+1)×(t–1) = k。经计算t=⌊(x+1)/(k+1)⌋=$\gamma(x, k)$。证毕。

NEUConn6按如下方法用连珠表示一条x长线：数组*Line*[2×$\gamma(x, k)$]保存潜在的连珠，前$\gamma(x, k)$个单元表示$\gamma(x, k)$个潜在的黑连珠，后$\gamma(x, k)$个单元表示潜在的白连珠。若线内实际存在s个黑连珠$uc_{b,1}$, $uc_{b,2}$, …, $uc_{b,s}$，和t个白连珠$uc_{w,1}$, $uc_{w,2}$, …, $uc_{w,s}$，则：1）当0≤i≤t–1，*Line*[i]←$uc_{b,i}$，当t≤i≤$\gamma(x, k)$–1，*Line*[i]←ϕ；2）当$\gamma(x, k)$≤ i ≤$\gamma(x, k)$+s–1，*Line*[i]←$uc_{w,i}$；当$\gamma(x, k)$+s≤i≤2×$\gamma(x, k)$–1，*Line*[i]←ϕ。例如，在2.1节的图1中，对于连珠A所在的线，有$\gamma(x, k)$= γ(19, 6)=⌊(19+1)/(6+1)⌋= 2，所以用*Line*[4]表示：*Line*[0] ← A，*Line*[1] ← C，*Line*[2] ← B，*Line*[3] ← ϕ。而连珠D（不妨令D为黑连珠，E为白连珠）所在的线也可以用*Line*[4]表示：*Line*[0] ← D，*Line*[1] ← ϕ，*Line*[2] ← E，*Line*[3] ← ϕ。

其次，可证明*connect*(n, n, k, p, q)的棋盘上有固定数目的线。很明显，*connect*(n, n, k, p, q)有n条水平的，n条垂直的，2条对角方向的n长线。另外，对所有的i(k≤i<n)，都有4条对角方向的i长线。共计$\chi(n, k)$=6n –4k+2条线。

综上，一个局面可以用$\chi(n, k)$条线表示，而每一条线又可以用$\gamma(n, k)$个潜在的连珠表示，那么，一个棋盘恰好可以由$\lambda(n, k)$ = 2$\chi(n, k)$·$\gamma(n, k)$个潜在的连珠表示。

6.2 连珠和潜在连珠在局面中的形式化描述

博弈过程中，连珠的演化是由空交叉点落子引起的；因此，在已知点的编号的情况下，必须能知道哪些潜在的连珠会发生变化。所以，在一个局面中，每个潜在连珠由定义29形式化地描述。

定义29 （一个局面中）潜在连珠的形式化描述。在*connect*(*n, n, k, p, q*)中，一个潜在的连珠*uc*可以由下述六元组

uc = (*color*(*uc*), *lineID*(*uc*), *uID*(*uc*), *from*(*c*), |*c*|, *f*(*c*))，或五元组

uc = (*color*(*uc*), *lineID*(*uc*), *uID*(*uc*), *from*(*c*), ||*c*||)

表示。其中，*color*(*uc*)∈ {′*B*′, ′*W*′}；*lineID*(*uc*)是*uc*所在的线在局面中的编号，且0≤ *lineID*(*uc*)< $\chi(n, k)$；*uID*(*uc*)是一条线内同色连珠的线内编号，且0≤ *uID*(*uc*)< $\gamma(n, k)$；*from*(*c*)是*c*的第一个交叉点在棋盘中的编号，0≤*from*<361}；*k*≤|*c*|≤*n*；$f(c) = \sum_{0\le i<|c|} s(c, i)\cdot 2^i$。

定义29中，带下划线的*color*(*uc*)，*lineID*(*uc*)和*uID*(*uc*)共同描述了局面中一个唯一的潜在连珠*uc*，而*from*(*c*)，|*c*|和*f*(*c*)则共同描述了局面中一个唯一的实际存在的连珠*c*。

定义30 基于连珠的局面表示。*connect*(*n, n, k, p, q*)中的一个局面可以由下述数据结构表示：

Position[*n_color*][*n_line*][*n_uc*] = {

(*from*(c_0), |c_0|, *f*(c_0)),

(*from*(c_1), |c_1|, *f*(c_1)),

…,

(*from*($c_{\lambda(n,k)-1}$), |$c_{\lambda(n,k)-1}$|, *f*($c_{\lambda(n,k)-1}$))

}。其中，连珠颜色共有*n_color*=2种，一个局面中的线共*n_line* =$\chi(n, k)$条，每条线内共有*n_uc*=$\gamma(n, k)$个潜在的连珠。

6.3 NEUConn6局面的基本数据表示

在*k*子棋中，对于局面评价和分析、着法生成、局面状态描述等，相比较而言，以连珠为基本单位和以交叉点为基本单位的数据表示，前者具有很大的优势。因为，前者可以把对局面中普遍存在的棋子间最紧密的联系——连珠作为结果保存起来。但是，着法的评价和选择，还必须以交叉点为单位，所以，在以连珠为核心的CPBM模型中，尽管局面的表示是基于连珠的，也必须构造一个以空交叉点为单位的（复合）类型表，从而方便确定候选着法的范围，并准确地评价候选着法。CPBM模型的数据结构设计如图6所示。

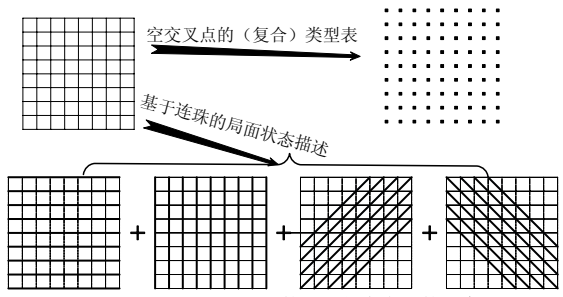


图6：*connect*(9, 9, 6, 2, 1)的局面状态表示的示意图

注意，空交叉点的（复合）类型表与传统的基于交叉点的局面状态表示是有区别的：基于交叉点的局面状态表示是指每个交叉点都有3种可能状态，分别表示有黑子、有白子和无子，必须构造一个表来确定具体交叉点的具体状态；而空交叉点的（复合）类型表，则用于保存每个空交叉点在4个方向、黑白两色连珠中的类型，或者复合空交叉点的类型。从本质上看，空交叉点的复合类型也是由连珠推导而来，因此，它的类型表或复合类型表也是仍未脱离以连珠为核心设计模型的数据结构的思想。

在图6的模型中，描述局面相关的数据结构既有以交叉点为单位的，又有以连珠为单位的。它们从不同侧面反映局面的特征。当局面发生变化的时候，必须保证两种数据结构同时更新，保持状态的一致性。

7 增量方法

增量方法可以降低博弈树中每个结点处的更新代价。在CPBM模型方便了增量方法的应用，但增量方法并不局限于CPBM模型，任何适合增量更新的棋类博弈程序都可以应用增量方法。

7.1 棋盘的基本操作

了解和更新棋盘状态，主要更新表示棋盘状态的数据结构。在基于连珠的棋盘表示法中，对*Position*[ ]的操作相对复杂些，主要包括：更新、查询。

● 更新局面中的某个连珠*c*。求得*c*所对应的*uc*，再令*Position*[*color*(*uc*)][*lineID*(*uc*)][*uID*(*uc*)] ← (*from*(*c*), |*c*|, *f*(*c*))。

● 空交叉点*i*放入棋子*s*引起的更新。1）同色连珠更新。若*s*不落在任何潜在连珠内，不会影响基于连珠的局面状态，不必更新；否则，*s*必落在某一个*uc*内，令*Position*[*color*(*uc*)][*lineID*(*uc*)][*uID*(*uc*)] ← (*from*(*c*), |*c*|, *f*(*c*)+2^i)即可；2）异色连珠的更新。与同色连珠的更新类似但要复杂些，这是因为*s*会把*c*切割成两部分，可能：两者均非连珠；恰好有一个是连珠；两者均为连珠。

● 某连珠*c*变为*c*′，更新其内部的各个空交叉点类型。无交叉点类型知识库时，要对*c*′内部的各个空点*i*都执行一遍算法3；否则，直接从交叉点类型知识库直接读取*ti*(*putStone*(*c*′, *i*))相应的类型即可。

7.2 增量的状态改变

在博弈树搜索过程中，每个结点处都由部分的状态更新代替整盘的状态更新，结果却完全相同，但它所带来的效率提升作用是可观的。

第一，向局面中放入一颗棋子所带来的状态更新。考虑棋盘中一个空交叉点*i*中放入一颗棋子*s*。根据定义30，最多会影响到*i*所在的4条线上的连珠。假设在*i*中放入*s*之前，包含*i*的同色连珠有*x*(*x*≤4)个，包含*s*的异色连珠也有*y*(*y*≤4)个，那么在*i*中放入*s*之后，仍有*x*(*x*≤4)个包含*i*的同色连珠，有至多2*y*(2*y*≤8)个异色连珠。最终，会影响到至多*x*+2*y*(*x*+2*y*≤12)个连珠，不超过局面中潜在的连珠总数的12/((6*n*–4*k*+2)$\gamma(n, k)$)×100%。至多影响*connect*(19, 19, 6, 2, 1)中约3.26%的潜在连珠。此外，*i*中放入一颗棋子*s*，最多能影响4*n*-3个空交叉点，约占总数的(4*n*-3)/361×100%。至多影响*connect* (19, 19, 6, 2, 1)的73个的空交叉点，约占总数的20.22%。

第二，把刚放入的那颗棋子从局面中取出所带来的状态更新。明显地，拿掉一颗棋子是摆放一颗棋子的逆操作。当拿掉棋子并更新状态时，最容易想到的方法就是对摆放该棋子操作的每一个指令都执行一遍逆操作。那么，拿掉一颗棋子将和增加一颗棋子的代价基本相同。7.3介绍了更快速的状态恢复方法。

7.3 增量的状态恢复

由7.2节可知，每颗六子棋棋子一般只引起局面中小范围变化，状态改变的连珠最多约占总数的3.26%，类型改变的空交叉点最多约占总数的20.22%。本文把发生变化部分的旧状态记录为一个*deta*，图8(C)将变化后的新状态用一个圆抽象表示，六子棋中的*deta*可以是变化的交叉点和连珠的旧状态。在深度优先搜索的博弈树中，为了维护*deta*，增加了一个称为*Deta*[MAXPLY]的栈结构，如图8(C)。每向*ply*层的一个局面中增加一颗棋子，都将变化前的旧状态*deta*保存到*Deta*[*ply*]中，图8(A)描述了图7中的一系列着法m_0~m_2引起的局面状态的增量更新过程。当撤销着法序列m_0~m_2的时候，只要将deta覆盖到局面中原来的（由着法所在的交叉点编号确定的）位置上即可，其撤销过程如图8(B)所示。增量方法优于非增量方法，原因在于：前者的保存和恢复的代价远比后者计算代价低，但空间代价却不一定比后者高。上述增量方法适于DFS (*Depth First Search*)，不适于BFS(*Breadth First Search*)。

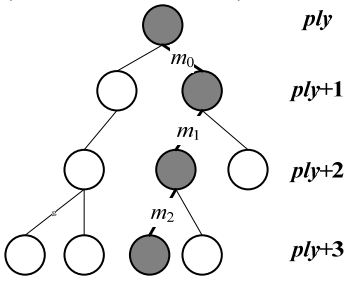


图7：博弈树中的着法序列

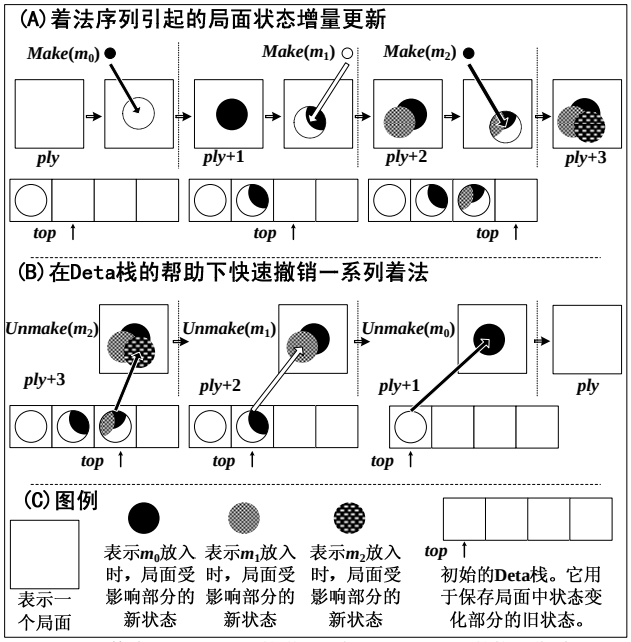


图8：着法m_0~m_2对局面的增量更新，以及m_0~m_2的快速撤销

7.4 增量的着法生成

着法生成与搜索算法密切相关。好的着法生成器具有以下特征：1）能够在不访问到无用着法的情况下，就从着法列表中剔除掉无用着法。2）所提供的着法列表，其着法顺序基本按照从优到劣的顺序。3）能够分步执行着法生成，而不必一次生成着法。

在CPBIM模型中，能更容易地分阶段生成着法，因为一次生成一条线上的候选着法。在生成一条线上的着法之前，可以判断其中是否存在需要的可行着法。只有存在可行着法的时候，着法生成操作才执行，这样，一次仅生成部分合法着法就能节省计算，因为如果已经展开的着法产生了一个剪枝，剩余着法的生成就变成了多余的计算。

8 实验和相关讨论

8.1 实验的建立

首先，实验程序的核心搜索算法是基于威胁的搜索[10]。通过该搜索算法可知：存在着使走棋方获胜的双威胁着法序列，或者不存在这样的序列；前者成立，算法返回赢，后者成立，算法返回不能赢。若搜索耗尽了预设的时间或空间，则失败，所付出的搜索努力也就失去实质意义。采用该算法主要因为：1）它有助于更多地关注模型的本质，忽略次要问题的干扰。比如，基于威胁的搜索只要判断叶子结点的值是：赢、输、和棋或未知即可，回避了复杂的叶结点估值问题。2）它是*k*子棋中非常有用的算法，运用该算法曾解决了著名的五子棋[12]。考虑到六子棋也广泛地存在威胁，实验程序采用了一种简单有效的双威胁搜索算法[2]。

实验涉及四个不同版本的程序：1）Raw——即采用了原始模型的版本，有如下特点：a）基于交叉点的局面状态描述和相关的数据结构设计；b）未实现连珠及其空交叉点的知识库，因而需要在线计算；c）未采用增量更新方法；2）CPBM——采用CPBM模型的版本；3）IU——Raw版本基础上实施了增量更新；4）CPBIM——在CPBIM基础上实施了增量更新。

此外，上述四个程序共同采用的技术包括：1）采用了同形表，用于消除对相同局面的重复搜索。在实验程序中，同形表是一个含有2Mb个入口地址的双层哈希表，它占用了256Mb内存。2）采用深度优先的搜索，因为它更适合增量估值。3）采用迭代加深，原因有二：其一，为了找路径最短的解；其二，可以更大程度地从同形表获益。4）基于领域知识的启发。好的着法的顺序常常可以引发更多的剪枝，为此，程序中运用了常见的历史启发和杀手启发等方法。

测试集有两种。一种是完全由无法双威胁胜的局面构成的测试集，简称非获胜测试集；另一种是完全由可双威胁获胜的局面构成的测试集，简称获胜测试集。对于搜索路径上的每颗棋子都算做一层。在基于双威胁的搜索中，六子棋的每个回合下来（进攻2颗子，防守2颗子）就涉及4层。因此，所统计的测试结果，其层数都是4的倍数。非获胜测试集由7个子集构成，每个子集都含有80个局面。对于每个测试子集的80个局面，在达到某个最大深度的时候，搜索就会因为知道解一定不存在而结束，该深度称为最大搜索深度。这7个子集的最大搜索深度分别为4层、8层、12层、……28层。获胜的测试集由5个子集构成，每个测试子集也包含80个局面。每个测试子集的80个局面的解的深度都相同（也就是找到解时候，搜索所必须达到的最大深度），分别为16层、20层、……28层。

图9给出了在非获胜测试集中的测试结果，图10给出了获胜测试集中的测试结果。图中，*x*轴是各个子集的最大搜索深度，*y*轴是搜索各个子集中的80个局面所需要的总时间。对于非获胜的测试局面，当深度是4层

或者8层的时候，由于计算机时钟的精度，统计结果稍有误差，当深度超过8层以后，程序之间的效率差别明显，误差可以忽略。对于获胜局面，12层以内就获胜的过于简单，不足以说明搜索算法的效率，因此，深度少于16层获胜的没有对比它们的实际运行时间。

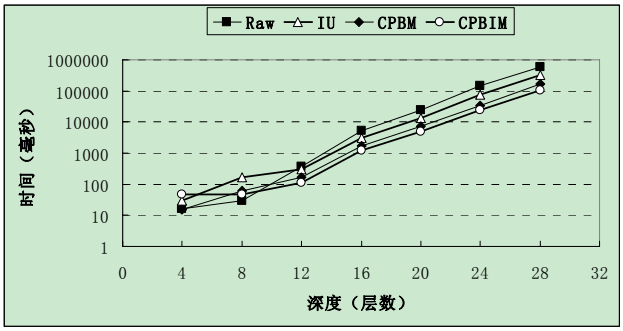


图9：在不同深度的非获胜测试集中，各程序的时间开销

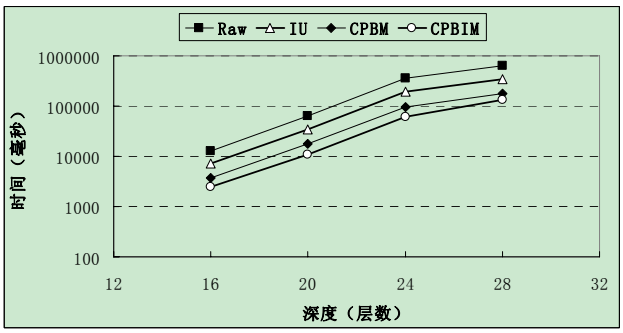


图10：在不同深度的获胜测试集中，各程序的时间开销

图9（*x*轴12层~28层之间的曲线）和图10（*x*轴16~28层之间的曲线）的结果显示：Raw所需要的时间最多，IU所用的时间次之，CPBM所用的时间更少，而CPBIM所用时间最少。

为了更直观地给出四个版本的搜索速度差异，在以上测试过程中统计了各个实验程序搜索速度的一个重要指标*nps*（*nodes per second*），它表示程序在一秒钟平均搜索的结点数目，统计结果见表5。*rate*表示以Raw版本为基本参照，各版本程序间的相对速度。CPBM的搜索速度是Raw的近3倍；增量搜索的速度是Raw的近1.8倍；CPBIM的搜索速度是Raw的5倍多。由此可见，CPBIM模型在*k*子棋机器博弈中的实际应用效果是很理想的。本节的测试是在如下配置的机器上分别进行的：时钟频率3.00GHz，内存2.00G，操作系统Windows XP。尽管测试机器的CPU是具有两个对称处理器的，但四个版本的测试程序都未采用SMP的并行处理技术。

表5：搜索算法的速度和对比

	Raw	**IU**	**CPBM**	**CPBIM**
nps	40*k*	71*k*	117*k*	201*k*
rate	1	1.775	2.925	5.025

8.2 结论和展望

首先，本文提出了连珠的概念，定义和描述了它的若干属性。接着，深入研究了与连珠相关的评价问题，并提出了空交叉点的评价方法。随后，设计了一个高效的基于连珠的*k*子棋机器博弈模型CPBM。最后，在此模型基础上进一步提出了一种更为积极的增量更新的方法，形成了CPBIM模型。为了减少其它因素对实验结果的影响，选择了基于双威胁的搜索作为核心搜索算法。实验结果表明，在CPBIM中的双威胁搜索是它在传统模型中的速度的5倍左右。

NEUConn6擅长短时间内求解双威胁获胜局面，但是，在评价不可直接通过双威胁获胜的叶子结点时，它仅仅采用了一层的搜索简单地代替了估值函数，因此在实战中，棋力并不太高。即便如此，在国际国内的竞赛中，它的水平还能居于上乘，这主要归功于CPBIM模型在搜索效率和抽象知识评价方面的优异表现。未来拟加强它的alpha-beta搜索能力，并通过机器学习、神经元网络等方法来提高实战中的学习和进化能力。尽管我们相信其它的搜索算法，如*alpha-beta*及其变种、*Proof Number*、*lambda*搜索等，在*k*子棋的CPBIM中也会有更高的搜索效率，但是到底效果如何，还有待进一步地研究和测试。

9 参考文献

[1] M.J.H. Heule, L.J.M. Rothkrantz, Solving games Dependence of applicable solving procedures. *Science of Computer Programming*, 2007, 67(1): 105–124

[2] I-Chen Wu and Dei-Yen Huang. A new family of *k*-in-a-row games. In *Proceedings of The 11th Advances in Computer Games Conference*, 2005: 88-100

[3] 张颖, 6子棋启发式搜索算法的优化与设计, 西北师范大学学报(自然科学版). 2008, 44(4): 25-44.

[4] 李果, 六子棋计算机博弈及其系统的研究与实现. 硕士论文, 重庆大学, 2007

[5] L. Stiller, Multilinear algebra and chess endgames. *Games of No Chance*, 1996

[6] R. Lake, J. Schaeffer, Lu P. (1994). Solving large retrograde-analysis problems using a network of workstations. In *Advances in Computer Chess 7*, 1994: 135-162

[7] A. Felner, U. Zahavi, R. Holte, j. Schaeffer. Dual lookups in pattern databases. In *IJCAI*, 2005

[8] A. Junghanns, J. Schaeffer. Single-Agent Search in the presence of deadlocks. In *AAAI*, 1998

[9] Bruno Bouzy, Incremental updating of objects in indigo. in *Proceedings of the fourth Game Programming Workshop*, 1997: 179-88

[10] L.V. Allis, *Searching for solutions in games and artificial intelligence*. Ph.D. thesis, Maastricht, University of Limburg, 1994

[11] János Wágner, István Virág, Solving Renju, *ICGA Journal*. 2001, 24 (1): 30–34

[12]L.V. Allis, H.J. van den Herik, M.P.H. Huntjens, Go-Moku solved by new search techniques, *Jorurnal of Computational Intelligence*. 1995, 12 (1): 7–24

[13]http://www.grappa.univ-lille3.fr/icga/tournament.php?id=186&lang=4