**莫里斯十子棋游戏**

彼得 ·斯塔尔哈克

德国 Jena 07740 Jena 大学

2003 年 5 月

摘要。 介绍了一种用于求解莫里斯十子棋的逆行算法,并给出了最重要的对策理论结果。 由 Emanuel Lasker 发明的游戏在 “莫里斯九子棋”游戏中独树一帜。它尝试为其前身赋予更多的战略潜力，以避免在游戏规则中进行过于激烈的改变的情况下玩抽奖游戏。

## **简介**

近年来，一些不重要的战略游戏已经得到了完全解决。 例如，Awari 是由 John W 解决的。 罗美因和亨利 E. Bal[J.W. 罗明角 Bal，2002]。 Qubic 由 Victor Allis[V. Allis,92]并由 Allis[V. Allis,88]和 James Allen[J. Allen，89]。 有趣的是，这两种解决方案连接了四种使用的补充方法： Allis 采用了基于知识的方法，而 Allen 首先使用暴力深度搜索。 拉尔夫 ·加瑟（Ralph Gasser）解决了莫里斯九子棋的游戏[R. Gasser,1994]使用窄 alpha-beta 数据库进行开局，使用完整的 DTC（深度到对话）数据库进行中端游戏。 在其他游戏中，如国际象棋[K. 汤普森，1986 年]和方格图案[J. Schaeffer,1997]数据库方法应用于解决重要的 endgames

介绍了一种用于创建完整 DTW（深度取胜）莫里斯十子棋数据库（包含 136,476,472,674 位）的逆行算法的开发。

## **拉斯克莫里斯**

莫里斯九子棋是两个简单规则的玩家的棋盘游戏。 它在 14 世纪很受欢迎，但发现早期版本的版本少于 9 件，可追溯到 1400 BC 。 与其他中世纪游戏一样，多年来许多不同的规则也在演变。 **莫里斯十子棋是由 Emanuel Lasker 发明的，该公司的国际象棋冠军从 1894 年到 1921 年。** 它基于莫里斯九子棋的规则。 1931 年，他在《 Brettspiele der V ö lker 》一书中描述了新变体的规则，如下所示。 （作者翻译）：

一个行棋包括在空点上放置棋子，或将已经放置的棋子滑至自由邻点，玩家可以这样做，也可以这样做。 游戏开始时，手里的棋子数可能为 9 或更高 10 。"

### abcdefg defg

723



6 

5

4

3

2

1

**图 1：** 莫里斯十子棋的典型情况。 要移动的白棋部分他手里有两个，黑棋部分剩下三个。

**莫里斯九子棋。** 游戏在棋盘上玩，如图 1 所示。 有 24 个点（交点），并且部分只能沿标记线移动。 在任意点上只能放置一个零件。 玩家从九个开始，每个玩家使用不同的颜色。通常一个玩家为白棋，另一个为黑棋。 与其他游戏不同，主板最初为空。 带有白棋棋子的玩家开始。 在开局期间，玩家交替将他们的棋子放在空点上。 每个玩家都试图在一行中获取他的三个棋子，同时阻止对手执行相同的操作。 当一个玩家连续获得三件东西时 — —这被称为形成三连 — —他可能会把他对手的一个棋子从不属于三连的棋盘上拿走。 两个玩家都把他们的所有棋子都放在船上后，将一件东西从棋盘点滑到相邻的空棋盘点上就被移走了。 玩家不断尝试把三件东西连成一排，这样他们就可以拿走不属于工厂的对手。 只要一名玩家只剩下 3 枚，他就可以将他的一枚跳到棋盘的任何空点。

游戏可能以以下方式结束：

* 少于三件的玩家输了
* 固执己见的玩家失败者

-

还有两个主题需要讨论：

1. 在开局过程中，可以同时形成两个三连。 然后应该允许玩家移除他对手的两个棋子吗?
2. 当玩家刚形成三连，但在三连外没有足够的对手片时。 然后，是否允许玩家从三连中取出一枚工件 ？

在我们的实施中，我们用 “是”回答第一个问题，用 “否”回答第二个问题 - 根据世界莫里斯九子棋协会的规则。 NMM 与 莫里斯十子棋之间的区别在于，在 莫里斯十子棋中，并非所有棋子都必须放在第一个移动中。 放置移动和滑动移动可以任意顺序执行。

### abcdefg defg

73



5

6

5

4

3

2

1

abcdefg defg

710



6 

5

4

3

2

1

**图 2：** 要移动的黑棋。g7xb4、f4（!）是唯一不会丢失游戏的行棋。

**图 3：** 不允许移动白棋跳转。

## **求解 Lasker Morris**

**状态空间**

主板由 24 点组成。 每个点都可以被黑棋或白棋部分占用，也可以为空。 在两个玩家的手边还剩下许多棋子。 在典型的游戏情况下，我们可以定义四个自然数字：nw、nb、nwh、nbh –其中 nw/nb 是放置在主板上的白棋/黑棋棋子的数量，nwh/nbh 是白棋/黑棋玩家手中的白棋/黑棋棋子的数量。

要计算完整状态空间中的元素数，我们首先可以看到可以将整个状态空间划分为不相交的子空间。 每个子空间都有四个参数：nw、nb、nwh、nbh 。 假设 0?nw、nb、nwh、nbh?10 和 0?nw + nwh、nb + nbh?10，我们有 4356 个不同的子空间。 现在，当一侧的游戏剩余部分少于三片时，我们可以通过拒绝子空间来减少这些子空间的数量（nw + nwh?3）。 因此，只有 3,600 个不同的子空间。

**无法到达的局面**

我们可以消除更多子空间，这些子空间仅包含在合法游戏过程中无法达到的局面。 此外，我们还可以消除在任何子空间中发现的所有不可到达的单个局面。 在实施过程中，我们不会拒绝所有这些局面，因为我们当然对他们感兴趣，因为我们对每一个局面感兴趣，其中一方都可以进行合法行棋。

### abcdefg defg

7



6

5

4

3

2

1

010

abcdefg defg

7

6

5

4

3

2

1

910

**图 4：** 白棋移动。 在 65 个天堂（无法访问）中获胜。

**图 5：** 白棋移动。 23 个天堂局面的损失。

**对称局面**

每个子空间都包含许多对称局面。 我们有五个对称轴，但这五个轴之一是冗余的，因此我们预计状态空间的大小最多可减少 16 倍。 构建能够完美解决 “对称”问题的函数可能很困难。 因此，由于快速计算至关重要，因此我们只提供半完美的解决方案，将状态空间减少 15.66 而不是 16 。 计算

子空间的大小为

??24?\*?24?nw?。使对称局面进入

"

## "

我们可以将状态空间（包括所有子空间）从约 2136 亿减少到 136.4 亿。

"

"

& # x0A;& # xA0;& # xA0;& # xA0;& # xA0;& # xA0;& # xA0;& # xA0;& # xA0;& # xA0;& # xA0;& # xA0;& #

**图 6：** 对称轴。

**Hash 函数**

我们的下一个目标是定义一个哈希函数，该函数将给定状态空间的每个特定局面（基数为 N）编译为唯一的自然数字。 如果具有这样的哈希函数，则不必存储某个局面的压缩描述及其游戏理论值，因为该描述是在哈希函数的索引中编码的。

**数据库**

最终数据库由 3600 个文件组成。 对于每个子空间，在硬盘上都有一个文件。 局面的值存储在从哈希函数返回的局面。 当局面为 “绘制”或 “丢失”时，该值为零，否则该值等于（“完全”）移动的次数，直到白棋（“移动”）可以强制获胜为止。 我们需要 8 个位来存储此值，因此完整数据库的大小约为 136 亿字节。 要确定某个局面是丢失还是绘制，我们必须应用一个层的最小 - 最大搜索。 此搜索的成本约为每个局面 10 到 30 个硬盘访问 —这等于平均局面上可能移动的次数。

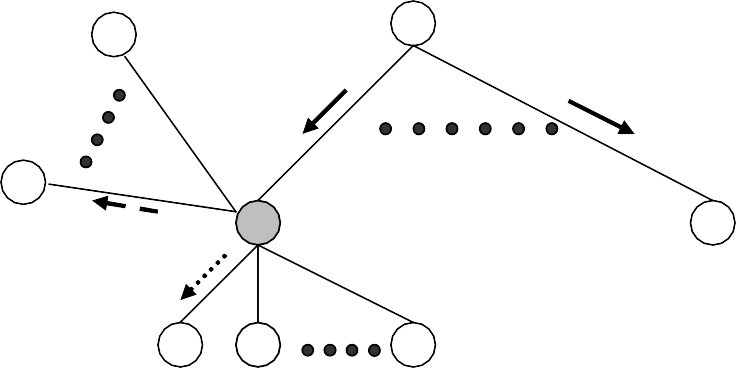
**计算**

逆行分析的第一步是初始化所有的入刀局面。 初始化完成后，会将所有剩余局面的值设置为零。 然后，迭代过程确定这些 “零”局面的实际游戏理论值。 在第 n 步，我们将计算并标记所有要移动的第 n 位（白棋）。 假设数据库中已标记了用于移动白棋的 n 中的所有 wins 。 在下一个迭代步骤中，我们将获得一个 “赢入 n”（win-In-In）局面。 该局面的所有前置任务都是黑棋的潜在损失（以 n 为单位）。 如果我们知道与实际数据库有关的前一代 q 丢失了黑棋，我们可以将 q 的所有前一代标记为 “一共 n + 1 位”的白棋候选项（移动）。 要检查 q 是否确实因黑棋而丢失，我们必须检查 q 的所有后续任务。 如果有一个后继路由器未标记为 “白赢”，则 q 不会因黑棋而损失 n 。

如果所有这些局面都已被标记为 “白赢”，则 q 将被证明是损失

移动白棋

黑棋移动



在?n + 1 中获胜

在 n 位获胜

qCandidate：n 型损耗

黑棋

向后移动

黑棋向前移动

白棋向后移动

**图 7：** 当 q 是损失时，我们可以将未进行 jet 评估的 q 的所有前置任务标记为 win-In-n + 1 。

此算法听起来很容易 –事实上，对于所有具有小状态空间的游戏来说，在计算此类数据库的当今计算机的主要内存中很容易出现这种情况。 莫里斯十子棋的状态空间为 2,134 亿

元素。 考虑到所有对称，我们可以将状态空间减少到约 136 亿的不同局面。 为了加快数据库的计算速度，我们必须避免随机磁盘访问。 即使每个局面只有一个光盘访问权限，计算时间也会增加到 40 年以上 ！ （136 亿 \* 10 毫秒）。 在我们的实施中，我们利用了数据库包含 3,600 个文件的事实。 这些文件使我们能够利用下面介绍的非常高效的缓存方法：

考虑任意子空间（nw = i, nb = j, nwh = m, nbh = n）。 从该子空间的任何局面取回任何黑棋移动后，我们只能在五个不同的子空间中找到自己（图 8）。向前移动这五个子空间（以证明损失）会使我们最多再进入 13 个子空间。 最后，从经验证的输棋地位中收回所有白棋移动也会导致最多 13 个子空间。

来源储存格



n, m + 1 n, m n, m-1

图例

黑棋向后移动



i、j、m、n

I, j-1

I, j

I + 1、j-1



I + 2、j-1

I + 1、j

黑棋向前移动

13 芯

I, j-1 I, j I, j + 1



I-1、j-1

I-1、j I、j + 1



I + 1、j-1

I + 1、j I、j + 1

I + 2、j-1

I + 2、j



I-2、j

I, j + 1

**图 8：** 为了减少随机磁盘访问，我们使用临时完全加载到计算机主内存中的单元进行操作。

在迭代算法中，我们将在所有子空间上排列一个循环，以分别查找每个子空间的所有 win-In-n + 1 局面。 这使我们有机会将少数受影响的子空间完全加载到计算机的主内存中，以避免许多磁盘访问。

在下面的内容中，我们详细介绍了用于计算整个 莫里斯十子棋数据库的迭代算法。

# **步骤 0（初始化）**

此步骤仅在迭代过程开始时执行一次。 我们创建主数据库：对于每个子空间，我们在硬盘上创建一个文件。 文件大小等于子空间中考虑到所有对称的局面数。 所有字节都设置为零。 之后，我们将循环应用于所有局面，以明确查找并标记所有入局局面。

# **步骤 1**

在此步骤中，我们将创建四个位字段。 每个位字段由 3600 个文件组成 —每个子空间一个单元（文件）。 我们现在只使用一个单位位，而不是每个局面使用一个字节。 因此，一个位域的大小只是主数据库大小的 1/8，但文件数量相等。 我们一开始

用零初始化所有四个位域。 在 bitfield 3 中，我们存储了 “win-In-n”信息：当

主数据库中的相应局面表示 “入刀 - n”（win-In-in）。 位字段 4 中的位在主 db 中的相应局面发出最多 n 个移动的获胜信号时进行设置。

主数据库

### Win = nWin?n

0

0

位域 1

位域 2

位域 3

位域 4

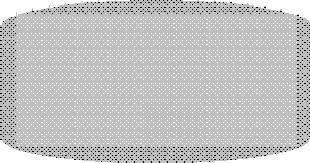
**步骤 2**

在此步骤中，我们将 n（黑棋移动和丢失）中丢失的候选存储到 bitfield 2 中。 我们在来自位域 3 的所有单元格上启动循环。 我们将位字段 3 中的一个单元和位字段 2 中的五个对应单元加载到计算机的主内存中。 然后，我们将另一个循环应用于所有 “入赢”局面，我们收回所有黑棋移动，并在这五个对应单元格中的一个单元格中将其标记为 “n 中的输棋”候选项。 完成内部循环后，我们将五个单元存储回硬盘，并切换到位字段 3 中的下一个单元。

### 

0

候选人损失 = n



Win = n

胜负

位域 1

位域 2

位域 3

位域 4

**步骤 3**

在此步骤中，我们将尝试证明步骤 2 中发现的输棋候选项。 在此步骤结束时，位域 2 包含所有经验证的 “在 n”局面出现的 “黑棋移动损耗”。 与步骤 2 类似，我们对来自位域 2 的所有单元格和一个单元格的所有局面应用两个循环。 我们提取每个候选输棋，并生成所有后继局面集。 如果这些后续局面之一未在位字段 4 的相应单元格中标记为 “白赢”，我们可以拒绝候选子。

### 

0

经验证的损失 = n

Win = n

胜负

位域 1

位域 2

位域 3

位域 4

**步骤 4**

在这里，我们将从位域 2 中加载所有单元，并从位域 1 中加载对应的五个单元。 现在，我们从 bitfield 2 中发现的所有 “loss in n”局面中收回所有白棋移动，并在 bitfield1 中将它们标记为 “win in n + 1 candidate”。

### 

候选人成功案例 = n + 1

经验证的输棋 = n

Win = n

胜负

位域 1

位域 2

位域 3

位域 4

**步骤 5**

在此步骤中，应用主数据库的更新。 当以下两个条件成立时，局面标记为 “在 n 中获胜”

* + 主数据库中的当前状态为零
  + 位字段 1 中的相应位被设置（win-In-n + 1 候选）

候选人成功案例 = n + 1

主数据库

位域 1

**改进与验证**

在我们的实施中，我们应用了更多改进。 我们连接了步骤 5 和步骤 1，因此在更新后不必再次读取主数据库。 其次，我们在一个位域的单元上管理了一条特殊路径，以减少加载和存储五个对应单元时的数据流量。 其想法是，两个相邻细胞的五个单元的集合不相交。 因此，切换到另一个单元时，我们可以保留主内存中的公共单元。

十子数据库计算于 2002 年 11 月到 2003 年 3 月。 在具有 1 GB 主内存的 Athlon XP 1533 MHZ（XP1800 +）计算机上，计算大约需要 4 个月。 该程序以 C 语言编写并使用 GNU C++ 编译。对于 GUI，我们使用 Borland CPP Builder 。

在计算过程中，我们遇到了一些硬件问题。 涉及的四个硬盘中有三个出现故障。 由于备份硬盘也有缺陷，因此必须再次计算完整的数据库。 更改所有驱动器后，计算不会出现任何问题。 2002 年早期和中期，我们用 7 子、8 子和 9 子计算了莫里斯十子棋。 所有数据库都包含在其后续数据库中。 由于对文件的比较没有任何差异，我们很有可能我们的最终数据库不会因硬件故障而出现任何错误。

但是，我们如何确保没有软件错误 ？ 目前，我们没有正式证明我们创建所有数据库的计划没有错误。 但另一方面，我们在过去三年的测试工作中没有发现任何异常。 这项测试工作包括数百场上次使用 Morris-GUI 玩过的游戏。 此 GUI 始终应用一个或两个 ply minimax 搜索来确定每个可能移动的值。 在此搜索中，它将执行深度检查，以将为原点局面存储的值与每次移动后的局面值进行比较。 当然，当独立的数据库程序重新计算游戏时，这将非常放心。

1. **结果**

到今天为止，莫里斯十子棋（Lasker Morris）是一个相对较小的游戏。 因此，我们将我们的结果与著名的母局莫里斯九子棋进行比较。 如果两侧都正确玩，那么 莫里斯十子棋的初始局面是一个绘图。

我们对莫里斯九子棋 的结果基于我们自己的 DTM 数据库，该数据库是在 2000 年年初计算得出的。

### abcdefg defg

72



4

6 

5

4

3

2

1

abcdefg defg

7



6

5

4

3

2

1

**图 9：** 莫里斯十子棋的最大局面。 白棋移动。 在 171 年获胜。

**图 10：** 莫里斯九子棋的最大局面。 白棋移动。 在 165 年获胜。

莫里斯十子棋（Lasker Morris）是一场抽奖活动。 因此，查看错误发生的时间有多早是有趣的。 图 11（12）显示了在早期出现黑棋错误后，两（四）名男子坐在船上的最长获胜距离。 这两个局面都吸引了 莫里斯九子棋。 莫里斯九子棋最早可能发生的错误是第二次移动白棋，即 1.c3 f4 2. f2?

图 5（第 2 节）显示了单个部件的功率。 其中，莫里斯十子棋在多数子面前获胜。

### abcdefg defg

799



6

5

4

3

2

1

abcdefg defg

788



6

5

4

3

2

1

**图 11：** 要移动的带白棋的洞口局面。 在 72 年获胜。

**图 12：** 要移动的带白棋的洞口局面。 在 132 年获胜。

图 13 至 16 显示了所有局面的获胜距离分布情况（白棋移动）。 水平轴显示以（满）移动为单位的距离，垂直轴标有局面数。 我们可以看到莫里斯十子棋和九子 ·莫里斯之间的相似之处。 即使 lokal 峰值大约位于同一局面，与 莫里斯十子棋之间的距离也较高，大约有 4 个向右移动。

在拉莫里斯十子棋 51.47%的所有局面以白棋获胜，以白棋移动，而莫里斯九子棋获胜的可能性仅为 46.55%。 百分比上的差异看起来很小，但这是我们实际观察的一个指标，即 莫里斯十子棋比莫里斯九子棋 更清晰。

8000

**百万**

7000

6000

5000

4000

3000

2000

1000

### "

**图 13：** 莫里斯十子棋获胜距离分布。

600

**百万**

500

400

300

200

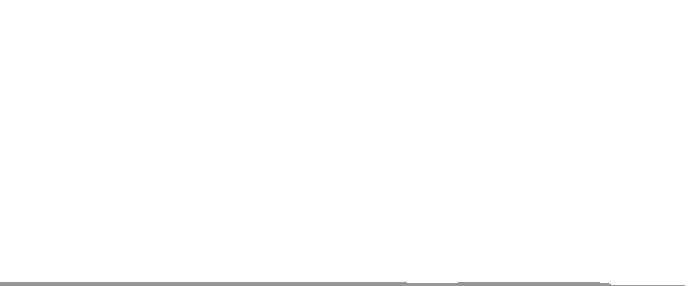
100

0

### 156763068

**图 14：** 莫里斯九子棋获胜距离分布。

18



**百万**

16

14

12

10

8

6

4

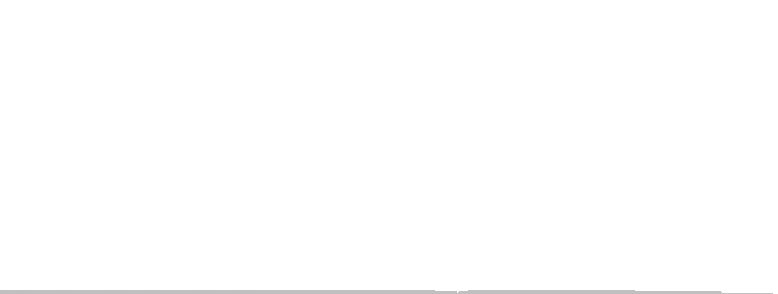
2

0

### 35 44 497685102119171

**图 15：** 莫里斯十子棋获胜距离分布。 从图 13 中放大分布。

2



**百万**

1.8

1.6

1.4

1,2

1

0.8

0.6

0.4

0.2

0

### 27374681115165

**图 16：** 莫里斯九子棋获胜距离分布。 从图 14 中放大分布。

## **结论**

莫里斯十子棋（Lasker Morris）的游戏是一场游戏。 现在提供了一个具有所有可能的 136 亿局面游戏理论值的 DTW（深度 to win）数据库。 对于未来，我们计划寻找并实施一种策略，该策略可以提高战胜虚假对手而获得超过游戏理论值的机会。

## **附录**

表 1 显示了一种最佳的移动序列，以确保图 11 的游戏理论值与双方完美的游戏效果。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 1. g4 ！ | 答 7 | 26. | d2-f2 | e4-e3 | 51. | e5-e4 | d2-d3 |
| 2. d2 ！ | a1 | 27. | g4-g7 | e3-e4 | 52 。 | c5-d5 | d3-d2 |
| 3. a4 ！ | f4 | 28 。 | c5-d5 | c4-c3 | 53. | d5-e5 | d2-d3 |
| 4. d3 ！ | a1-d1 | 29. | g7-d7xe4 | b4-b2 | 54. | " | d3-d2 |
| 5. c3 ！ | f6 | 30 。 | a4-b4 | b2-d2 | 55 。 | e4-e3 | d2-d3 |
| 6. e3xf6 | f6 | 31. | d5-c5 | d2-d1 | 56 。 | e5-e4 | d3-d2 |
| 7. c3-c4 ！ | d6xa4 | 。 | d7-a7 | d1-a1 | 57. | e3-d3 | a4-a1 |
| 8. c4-c3xd1 ！ | b6-b4 | 33. | d3-e3 | c3-d3 | 58 。 | e4-e5 | a1-a4 |
| 9. d1xd6 ！ | c4 | 34. | e3-e4 | d3-d2 | 59. | d7-g7 | a4-a1 |
| 10. e3-e4 ！ | a7-a4xc3 | 。 | c5-d5 | a1-d1 | 。 | g7-g4 | a1-a4 |
| 11. d2-f2 ！ | b4-b6 | 36 。 | " | f4-g4 | 61. | g4-g1 | d2-d1 |
| 12. f2-d2xc4 ！ | a4-b4 | 37. | f2-f4 | g4-g7 | 。 | d3-d2 | a4-a1 |
| 13. d2-b2 | d6xb2 | 38 。 | e4-e3 | d2-d3 | 63. | f4-g4 | a1-a4 |
| 14. d3-d2 ！ | d6-d5 | 39 。 | d5-c5 | d3-c3 | 。 | g4-g7 | a4-a1 |
| 15. d7-d6 ！ | " | 。 | c5-c4 | c3-d3 | 。 | e5-e4 | f6-f4 |
| 16. d6 ！ | b4-c4 | 41. | c4-c3 | g7-g4 | 。 | g7-d7 | a1-a4 |
| 17. b4! | c4-c3 | 42. | d7-g7 | g4-g1 | 67 。 | d7-a7 | f4-g4 |
| 18. d1-a1 | c3-d3 | 43. | e3-e4 | g1-d1 | 68 。 | e4-f4 | g4-g7 |
| 19. a1-a4 | d3-e3 | 44. | c3-c4 | d3-c3 | 。 | f4-g4 | a4-a1 |
| 20. e4-e5 | e3-d3 | 45. | e4-e5 | d1-a1 | 。 | " | g7-d7 |
| 21. b4-c4 | " | 46. | g7-d7 | a1-a4 | 71. | g4-g7 | d7-a7 |
| 22. c4-c3 | d5-c5 | 。 | d7-a7 | c3-d3 | 。 | g7-d7 # |  |
| 23. e5-d5 | c5-c4 | 48. | e5-d5 | d3-d2 |  |  |  |
| 24. d5-c5 | d3-e3 | 49 。 | c4-c5 | d2-f2 |  |  |  |
| 25. c3-d3 | e3-e4 | 50 。 | d5-e5 | f2-d2 |  |  |  |

**表 1：** 最佳移动序列。 如果移动是唯一能保持成功的移动，则会用感叹号标记该移动。 在他的前 17 次行动中，白子必须在 15 次游戏中找到独一无二的胜利。

## **参考**

[R. Gasser，1995 年]利用计算资源提高效率

*全面搜索。* PhD 论文，ETH，瑞士联邦理工学院，苏黎世。 *ICCA 日记帐，卷18 号* 2, pp.85-86 。 ISSN 0920-234 X 。

[J. Schaeffer，1997]领先一步： *挑战跳棋中的子类霸权。* Springer-Verlag，纽约，纽约。 ISBN 0-3879-4930-5 。

[J.W. 罗明角 Bal,2002] Awari 已解决。ICGA 日记帐，卷25 号 3，第 162-165 页

[J. Allen，1989 年]。 关于 Connect-Four 计算机解决方案的注释。 *子工智能中的启发式编程：第一台计算机 olympiad（eds.* D.N.L. 征税和付款 第 134-135 页）。 Ellis Horwood 有限公司， Chichester，英国。 ISBN 0-7458-0778-X 。

[L.V. Allis,1988]基于知识的 Connect 4 方法： *这场游戏结束了，白得胜。* M.Sc. 论文，Vrije 大学。 报告编号 IR-163、《数学和计算机科学学院》、《 Vrije 大学》、《阿姆斯特丹》

[K. 汤普森，1986 年）。 某些终端游戏的逆行分析。 ICCA 日记帐，卷9、编号 3, pp.131-139 。

[Em. 拉斯克,1931]布雷特施皮勒 - 德尔沃尔克