没有机会的游戏

MSRI 出版物

**第29**卷， 1996

# 求解莫里斯九子棋

## RALPH GASSER

摘要.我们描述了两种搜索方法的组合，用于求解莫里斯九子棋。改进的逆向分析算法计算了包含大约 10 10个状态的最终游戏数据库。然后，一个 18 层 的字母顺序搜索使用这些数据库来证明初始位置的 value 是一个平局。莫里斯九子棋是第一个非平凡的游戏被求解，似乎没有受益于基于知识的方法。

## 1. 简介

近年来，许多游戏已经求解使用计算机，包括Qubic [Patashnik 1980]，连接-4[Allen 1989;Allis 1988] 和Go-Moku [Allis 等人， 1993]所有这些游戏都是用基于知识的方法求解的。这些方法是成功的，因为所有这些游戏具有较低的决策复杂性 [Allis 等人， 1991]，也就是说，正确的着法往往很容易找到。并非所有游戏的利润都相同。例如，在跳棋中，国际象棋和围棋的多项动作通常看起来相当平等。暴力搜索通常是玩或求解此类游戏的最可行的方法。 这一观点得到了以下事实的支持：国际象棋（Hsu 1990）和跳棋（舍弗1992年）的最佳程序都严重依赖搜索。

搜索方法不仅对玩游戏很有用：它们在很多方面都无处不在。其中一些算法在计算机科学中是最容易理解的。然而，并非所有的搜索算法都同样得到很好的研究;特别是，在大型状态空间进行详尽的搜索还处于起步阶段。部分原因在于，硬软件最近才发展到一个有趣的问题触手可及的地方。由于硬件和软件的不断改进，必须不断重新评估搜索方法，并适应新的系统设计 [Stiller 1991;湖1992年\*。游戏是获得这一领域专业知识的理想选择，因为它们在限制领域进行，可以控制问题难度并衡量方法的有效性。

101

我们描述了两个暴力搜索的组合，遇到了用于求解莫里斯九子棋。改进的逆向分析算法计算了包含大约 10 10个状态的最终游戏数据库。然后，18层 alpha-beta 搜索使用这些数据库来证明初始位置的值是平局。莫里斯九子棋是第一个非平凡的游戏被求解，似乎没有从知识的方法受益。

## 2. 莫里斯九子棋

莫里斯九子棋是今天仍然玩的最古老的游戏之一[贝尔69]。在世界各地的许多历史建筑中都发现了棋盘。最古老的（大约公元前1400年）被发现雕刻在埃及一座寺庙的屋顶石棋盘上。其他人被发现像锡兰，特洛伊和爱尔兰一样广泛散落。

a

B

C

D

e

F

G

1

2

3

4

5

6

7

游戏在棋盘上玩，有24点，可以放置石头。最初，棋盘是空的，两个玩家每人都拿着九块石头。玩白石头的玩家开始。

在开场时，玩家交替放置**图1。**开场后任何空点上的石头（图1）。

放置完所有石头后，游戏进行到中盘。在这里，玩家可能会将他的一块石头滑到相邻的空点。如果在游戏期间任何时候，玩家成功地排列他的三块石头在一排 - 这被称为形成*一个三连*- 他可以移除任何对手的石头，不属于三连。在图 2 中，如果 White 通过玩到 b6 在开局处形成一个三连，他现在可以在a1 上移除 B缺少的石头，但不能移除 d2 上的石头。

一旦玩家只剩下三块石头，终盘比赛就开始了。轮到他时，有三块石头的玩家可以跳到棋盘上的任何空点。

a b d f g a b c e f g

77

66

55

44

33

22

11

**图 2.**b6 之后，W命中移除 a1 或 b4。 **图 3.**黑色没有着法。

游戏以下列方式结束：

* 少于三块石头的玩家会输。
* 不能进行合法着法的玩家会失败（图 3）。
* 如果重复一个中盘或结束游戏的位置，游戏就是平局。

两点在莫里斯九子棋爱好者中争论不休。第一个取决于观察，在摆子时，可以同时形成两个三连。应该允许对方取出一还是两块对手的石头？我们的实现只允许去除一块石头。第二点涉及玩家着法的位置刚刚形成了一个三连，但所有对手的石头也在三连里。可以搬走石头吗？在我们的实现中，可以清除任何石头。这些规则变化似乎都不太可能影响游戏的价值。

## 3. 求解莫里斯九子棋

莫里斯九子棋可以分为两个不同的阶段。开局在棋盘上放置石头，在中局和盘末放置石头。这两个阶段导致具有特定特征的状态空间图。最重要的是，开局阶段形成有向无环图，而在中局和盘末阶段着法可能发生重复局面。另一个区别是搜索深度。开局被明确定义为每条路径的 18 层深度;相比之下，在中局和盘末上花费的时间取决于所选的着法。

这些不同的属性使得需要使用不同的搜索方法。我们决定使用逆向分析来构建包含所有中局游戏位置的数据库。这似乎有利，因为逆向分析比前向搜索更高效地处理重复局面。此外，开局搜索将需要许多位置的值。由于相互依赖性，这可能意味着计算所有或几乎所有中局和盘末位置的值，这是逆向分析的理想任务。

为开局计算进一步的数据库是不合理的，因为它们的大小将甚至比中局游戏更大。此外，由于在开盘阶段仅对单个位置（空棋盘）的值感兴趣，因此必须存储任何中间位置的价值。 Alpha-beta 搜索 [Knuth 和 Moore 1975] 是此类问题的理想选择。

## 4. 中局和盘末游戏数据库

**4.1. 状态空间。**为了观察应用逆向分析是否可行，必须构建适当的状态空间。24个棋盘点中的每一个都可以空置，也可以被一块黑色或白色的石头占据，因此状态空间大小的上限为324，或近似2.8×1011，状态。通过考虑以下游戏约束，可以改进这种相当松散的约束：

* 玩家有三到九块石头在棋盘上，有不可到达的位置，和
* 棋盘是对称的。

3-3

4-3

4-4

5-3

5-4

5-5

6-3

6-4

6-5

6-6

7-3

7-4

7-5

7-6

7-7

8-3

8-4

8-5

8-6

8-7

8-8

9-3

9-4

9-5

9-6

9-7

9-8

9-9

第一个观测值允许我们将状态空间拆分为 28 个子空间。图 4 显示了这些子空间和它们的依赖关系;因此，7-5 位置（7个石头对 5 个）只能在确定所有 7-4 和 6-5 个位置的值后进行计算。

在合法博弈过程中，并非所有这些子空间中的所有状态都可到达。例如，如果一个玩家有一个三连，他的对手不能有所有九块石头在棋盘上（图5）。如果两个或三个形成的三连在棋盘上，也可以提出类似的考虑。9-9、9-8、9-7、9-6、9-5、9-4、9-3、8-8、8-7 和 8-6 子空间包含此类无法访问状态。

子空间还包含许多对称位置。通常有五个对称轴（图 6）。在

3-3子空间特殊案例中，有额外的对称性，**图4。**数据库依赖项。因为所有的环都是可互换的。由于这五个轴之一是冗余的， 例如， *π4 = π1 ◦ π2 ◦ π3*，我们可以预期约 16 倍从对称性减小状态空间大小。

考虑到所有三个削减，在中、末阶段共有7,673,759,269个状态。

对于如此大的状态空间，需要一种内存效率高的存储方法。逆向分析中使用的标准方法是构造一个完美的哈希函数，在给定状态的情况下，该函数返回一个唯一的索引。因此，无需将状态描述与状态值一起存储，因为状态将

*P*

3

*P*

1

*P*

4

*P*

2

*P*

5

**图 5.**无法到达的状态。 **图 6.**对称轴。

脚本在索引中编码。理想情况下，完美的哈希函数范围从 0 到状态数减去 1，可以快速计算。构造这样的哈希函数可能很困难。由于快速计算至关重要，因此放宽要求并允许范围稍大的哈希函数似乎是合理的。 这意味着文件中包含一些未使用的条目，但只要文件大小不会太大，这就不会妨碍计算。我们决定使用的哈希函数将 7，673，759，269 状态映射到 9，074，932，579 个索引范围。

**4.2. 计算。**逆向分析的第一步是初始化所有容易识别的赢棋和输棋位置。在莫里斯九子棋有两种类型的终局位置，两种是玩家走子的损失：玩家不能走子的位置，以及玩家少于三块石头的位置。因此，如果玩家走子被阻止，我们将该状态标记为输棋。我们也希望将玩家着法少于三块石头的位置标记为输棋，但由于我们从状态空间中消除了这些位置，因此我们标记玩家着法的位置可以损失一个三连，并将对手减少到两块。赢。我们还标记玩家在两个平地中输掉三块石头的位置。这些位置易于识别，属于以下类别之一：

* 对手有两个开厂机智没有石头共同;
* 对手有两个开放的三连，玩家着法不能形成一个三连;或
* 玩家必须清除阻挡对手三连的石头。

初始化设置这些胜负后，它将所有重位置的值设置为平局。然后，迭代过程确定这些"绘制"状态的实际值。作为第一步，请考虑如何在双人游戏中传播胜负。如果玩家着法的位置输棋，则所有位置的预处理器都可以标记为对手获胜。类似地，如果玩家赢得一个位置着法，所有前身都是对手的潜在损失。只有当所有的后继也为玩家赢得时，他们才是真正的损失（图7）。

检查所有后继以确定位置是否输棋，将效率低下。通过使用每个状态的 Count 字段来存储尚未证明对手获胜的后继数，可以避免这种情况。然后，我们不是在每次出现一个位置时都会对所有后继进行减号，而是简单地递减计数。当它达到零时，所有后继都被证明是对手获胜的，并且这个位置可以标记为失败。此改进的成本是使用的额外内存是 Count 数组。

由于我们计算赢和输的深度，因此使用一个字节将每个状态的值存储在 Val 数组中。实际上，Count 和 Val 数组可以使用相同的内存，因为从来不需要这两条信息

B

A

D

*...*

C

图 7.确定前驱的值。位置A是白方胜（先行），因为*一些*后继，说B，黑必败。位置 C 黑色必败，因为它*的所有*后继D白方 都赢了。

同时：仅在可能绘制位置值时，才需要 Count 数组。换句话说，Count 和 Val 值存储为"union";参见表 1。

表 2 显示了我们阻止所有状态值的算法。

第一个数据库是在1989年在MacintoshIIx上计算的，最后一个数据库是在1993年的Macintosh Quadra800上计算的。其间，该算法得到改进并移植到各种机器上。其中包括 Cray X-MP/28、DEC VAX 9000420、IBM RS/6000、16-转算器 （T805） 系统、30 处理器 MUSIC 系统 （Motorola DSP 96000） 和 DEC 3000（阿尔法）。只有DEC 3000的表现比苹果Macintosh更好。造成这种令人惊讶的结果的原因有很多。由于多用户环境，许多计算机都以较低的优先级削减了我们的代码。此外，许多功能更强大的计算机都经过优化，以处理浮点和矢量操作，而其整数性能则没有高度优化。并行机械器还因主内存不足和没有直接磁盘访问而受到影响。这使得 Macintosh 前端成为 I/O 瓶颈。

Val entry Count entry

1. = loss in 0 *.................................*
2. = win in 1 252 = “draw” (3 successors unknown) 2 = loss in 2 253 = “draw” (2 successors unknown)

3 = win in 3 254 = “draw” (1 successor unknown)

*......* 255 = “draw” (all successors unknown)

表 1.位置的所有已知信息都适合一个字节。

|  |  |
| --- | --- |
| Backupable ← []; **for** all states **do**  Val[state] ← Initialize(state); Count[state] ← 0; **if** Val[state] =6 draw **then**  Put(state,Backupable); | {initialization} |
| **end**; {if}  **end**; {for}  **while not** Empty(Backupable) **do** state ← GetState(Backupable); **for** all Predecessors(state) **do if** Val[pred] = draw **then if** Val[state] = loss **then**  Val[pred] ← win;  Put(pred,Backupable);  **else** {Val(state) = win} **if** Count[pred] = 0 **then** | {iteration} |
| **for** all Successors(pred) **do** | {count successors} |

Count[pred] ← Count[pred]+1; **end**; {for}

**end**; {if}

Count[pred] ← Count[pred]−1; **if** Count[pred] = 0 **then**

Val[pred] ← loss;

Put(pred,Backupable); **end**; {if} **end**; {if}

**end**; {if}

**end**; {for}

**end**; {while}

表2.此算法显示初始化和迭代如何确定所有状态的值。为简单起见，我们不会显示 Count 和 Val 数组的合并，也不会显示赢和输深度的计算。

**4.3. 核查。**计算这个大小几乎不可避免地包含一些错误。最常见的原因是逆向分析代码、系统软件或编译器中的软件错误。硬件问题（如磁盘错误或内存故障）也发生。由于这些原因，在30个太阳站（Balz 1994]）的集群上验证了数据库。 这大约需要三个月的时间。验证发现六个硬件错误。此外，在莫里斯九子棋代码中发现了一个错误，它允许将一些假设NS归类为平局，而不是损失。这些错误导致进一步的不一致，因此必须纠正数千个位置。

核查只检查胜负和平局是一致的。深度信息没有被处理，因为存储数据库所需的额外内存会增加磁盘 I/O 并显著减慢验证速度。目前，我们正在新的并行计算机上运行深度验证算法，该算法可以启用完整的验证n。即使深度验证没有发现其他错误，我们又如何确定不存在错误？验证背后的一个基本思想是使用不同的算法（正向搜索）和独立代码来验证数据。但是，这两种算法仍有一些通用的算法，例如文件索引函数或初始状态值。理想情况下，我们的结果应该独立验证。

## 5. 开局搜索

现在，所有中局和结束位置值都计算在数据库中，可以通过简单的 18层 alpha-beta 搜索找到初始位置的值。原则上，18 个 p 的开局可以导致除 8-3、9-3 和9-4 以外的任何数据库。这包括大约 9 GB的数据。由于我们的机器只有72兆字节，访问数据库值成为I/O瓶颈。应用以下方法来缓解此问题：

* 减小数据库的大小，
* 减少已用数据库的数量，以及
* 减少磁盘访问数。

由于开盘时没有周期，因此无需计算赢或输的深度。因此第一个想法是减少数据库的大小，将五个位置打包到e 字节 （35 = 243）。但是，为了进一步最小化存储需求，我们决定将八个状态打包成一个字节。尽管我们当时每个状态只有一个位，但如果我们执行两个 alpha-beta 搜索，这足以计算游戏理论值。在第一次搜索中，我们确定初始位置是否获胜。如果未获胜，我们将执行第二次搜索（使用不同的数据库），以确定初始位置是否输棋。

虽然这减少了数据库的大小一个8的倍数，但它仍然留下文件总计约1G字节。对莫里斯九子棋玩家，很明显，大多数

打开数据库

可换位

8

层

16

层

18

层

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 9-9 |  | 9-8 |  | 8-8 |

图8。打开搜索。

合理的游戏将导致最多一个或两个三连在开局期间形成。为此，我们决定只使用9-9、9-8和8-8数据库来证明平局结果，总共115兆字节。然后，无法再计算每个位置的正确值。但是，在使用 白方着法的位置时，我们可以计算正确值的上限，假设数据库中未在数据库中的所有位置都赢得白色位置，如果假设所有此类位置都是为黑色赢得的，则下限为白色。为了证明游戏理论值是平局，我们必须证明白棋有平局的上限和下限。

我们在 16 层级别插入了一个置换表，以进一步减少数据库访问的数量。通常，置换表从任何位置的横摆位置，甚至可能优先考虑树中较高的位置。我们选择只包括 16 个多级位置，因为减少磁盘 I/O 比树中的节点总数重要得多。参见图 8。

最后，在8层级别上访问的alpha-beta搜索访问的所有位置都存储在一个中间数据库中。这允许实时玩游戏，因为对于前 8 个层，只需对中间数据库执行搜索。以下是在两个搜索中每个搜索中访问的 8-层 位置 alpha-beta 的数量：

### 证明 没有证明

白色至少和棋 15，501 12 白色最多4，364 29

**图 9.**黑色着法。

输棋在26个plies。

**图 10.**白色着法和胜利。

187 plies，三连形成。

我们区分了至少（或最多）平局和不能证明是平局的位置。这并不意味着头寸输棋（或赢）;只是使用三个数据库不足以显示其他。

在8层水平的约350万个位置中，该表显示只有19,906个被评估。节点评估的时间从 2 秒到 15 分钟不等。在 Macintosh Quadra 800 上，总运行时间约为三周。检查表，一个 might 倾向于假设大多数位置是绘制的。情况绝不能如此;也许着法排序启发式只是成功地选择了平局着法。

## 6. 结果

**6.1. 数据库位置。**浏览最终数据库以搜索"有趣的"位置是一项艰巨的任务。由于我们没有真正的直觉，什么使一个位置有趣，我们决定收集结果在一个更统计的方式。接下来的两个图显示了具有长胜位位置的示例。图 9 中的 3-3 位置是该数据库中最长的序列的示例。图 10 中的 8-7 位置显示了一个获胜位置，其着法顺序最长，直到三连形成。

图 11 显示了玩家要着法的赢持位置的百分比。它们根据棋盘上的石头数量进行分组，并相应地规范化。图中显示，获胜概率与宝石差异密切相关：宝石越多越好。在大约七块石头上似乎有一个切口，所以少一点就很难赢。统计数据还显示，3-3数据库似乎很特别。这可能是因为两个玩家都可以跳，这在天性上使这是一个不同的游戏。人们必须记住，这些统计数字可能具有误导性。例如，在4-3数据库中，如果玩家有四块石头要着法，他似乎只有很小的获胜机会。仔细观察数据库后，我们看到所有的胜利都是微不足道的位置，玩家可以立即形成三连。

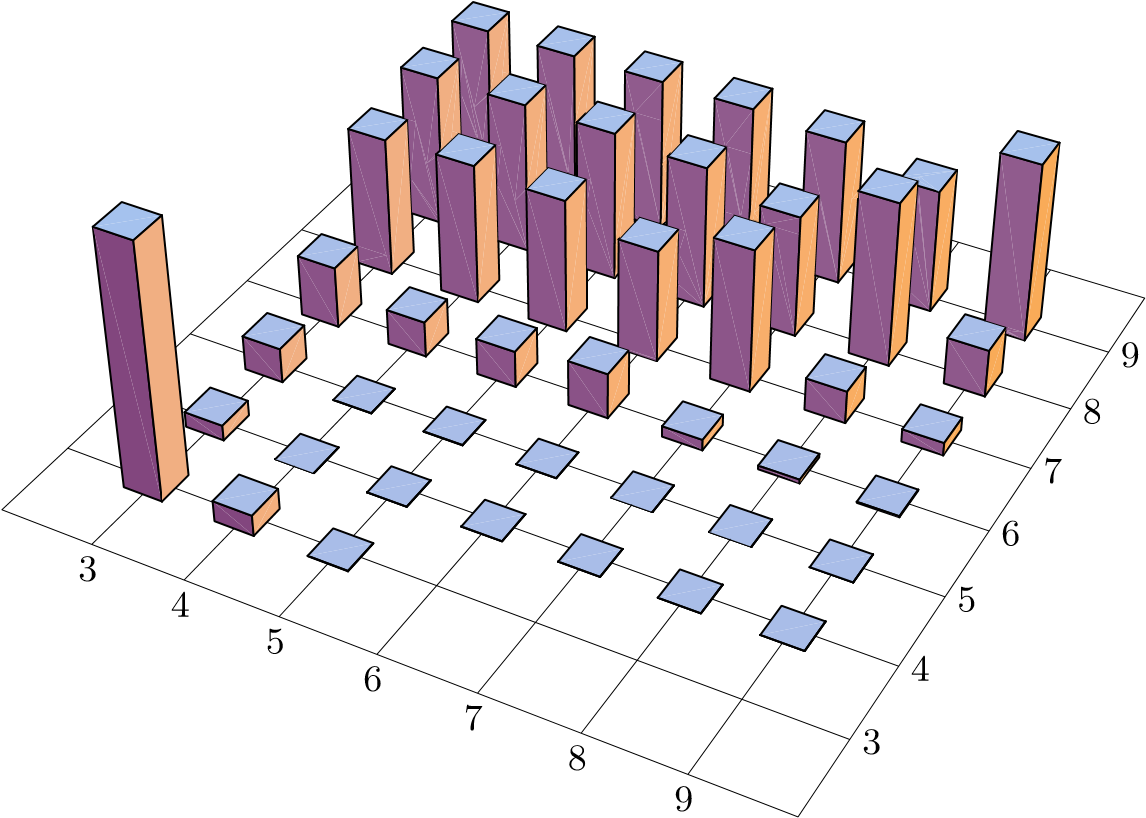


图11。玩家着法的赢百分比，作为初始宝石数的函数。3-3场比赛的最高百分比约为83%。缺少列（如 3-9 场比赛）表示百分比为零。

这些位置不会发生在真正的游戏中，所以实际上玩家与三块石头有一个小的优势。类似的保留适用于所有这些数据库。不清楚如何从其余部分过滤 r alistic 位置。

**6.2. 开局。**现在也可以检查一些常见的开局位置。在同样匹配的玩家之间，游戏在开局时往往非常吸引人，也就是说，大多数可能的动作在平局中恢复，如图12中的例子一样。然而，偶尔会发生失误。图13显示了1990年在伦敦举行的第二届计算机奥林匹克竞赛中，我们的莫里斯九子棋项目与迈克·桑利（英国冠军）的比赛。这个节目当时仍然以启发式为基础，赢了四场比赛，平了两场。

图12。以黑色开局位置以着法。任何着法都会导致平局位置。

9

5

**图13.**白色 9 输。 **图14。**白色 5 输。

由于初始位置是平局，因此自然要问的是错误发生多久。图 14 中的位置显示了我们发现的最早输棋着法之一。此位置反驳一个常见的误解，即形成三连是一个理想的开放策略。怀特在前两个动作中遵循这一策略，而布莱克则忽略了潜在的三连威胁。随后，怀特的第三个动作输了，尽管他现在很容易在开局时形成两个三连。

## 7. 结论

莫里斯九子棋是一个平局。这一结果是使用阿尔法-贝塔搜索和终局游戏数据库的组合实现的。我们新的高效逆向分析算法允许在台式计算机上求解 10个 10个数据库状态。这使得莫里斯九子棋成为第一个经典棋盘游戏，它不从基于知识的方法中获利。

在管理和使用大型预先计算数据库方面获得的专门知识也可应用于其他问题。许多游戏和谜题从结合数据库与搜索中获益，例如阿瓦里，跳棋，15-Puzzle或魔方。由于游戏和谜题是一个受限的问题领域，未来的工作将研究如何使更一般的优化问题可以适应这个应用程序。

## 引用

[艾伦1989]J. D. Allen，"关于连接四的计算机求解方案的说明"，第134-135页，《人工智能的启发*式编程：第一次计算机奥林匹克竞赛》（*由D.N.L.Levy和D.F.Beal编辑），埃利斯·霍伍德，奇切斯特，恩格，1989年。

[Allis 1988]L. V. Allis，"一种以知识为基础的连接四方法。游戏被求解：白赢"，M.Sc，数学和计算机科学学院，Vrije大学，阿姆斯特丹，1988年。

[Allis等人1991年]L. V. Allis， H. J. 范登赫里克，和 I. S.赫施贝格，"哪些游戏会存活"，第 232-243 页*人工智能的启发式编程：第二届计算机奥林匹克竞赛*（由D.N.L.Levy和D.F.Beal编辑），埃利斯·霍伍德，奇切斯特，英格兰，1991年。

[Allis等人，1993年]L. V. Allis， H. J. 范登赫里克，和M.P.H.亨詹斯，"去-莫库求解新的搜索技术"，第1-9页*Proc. AAAI 秋季游戏研讨会：规划和学习*，AAAI 新闻技术. 报告 FS93-02， 门洛公园， 加利福尼亚状态.

[巴尔兹 1994] G.巴lz，"状态数据库的验证"，文凭证书，ETH祖希里奇，1994年。

[贝尔1969]R. C.*贝尔，《许多文明之门的棋盘和桌*游戏》，牛津大学出版社，牛津大学，1969年。

[徐1990]F. H. Hsu，"阿尔法-贝塔搜索的大规模并行化：计算机国际象棋的算法研究"，博士，卡内基-梅隆大学，匹兹堡，1990年。

[克努斯和摩尔1975]D.E.Knuth和R.W.摩尔，"α-贝塔修剪分析"，人工智能6 1975，293-326。

[湖等人1992]R. Lake，P. Lu和J.Schaeffer，"使用逆向分析来求解大型组合搜索空间"，第181-188页，在1992*年《千国运动年度报告：利用杀手微（*由E.D.Brooks等人编辑）》中，劳伦斯·利弗莫尔状态实验室（UCRL-ID-107022-92），1992年。

[帕塔什尼克 1980] O.帕塔什尼克，""库比奇： 4 × 4 × 4 脚趾"，*数学。马格***53**（1980），202-216。

[舍弗等人1992年]J. Schaeffer等人，"世界锦标赛口径跳棋计划"，人工智能53（1992年），273-290。

[斯蒂勒1991]L. 斯蒂勒，"大规模并行架构上的组图和计算对称性"，J. 超级计算5（1991年），99-117。

RALPH GASSER

理论信息研究所

开德诺西斯切理工学院霍赫舒勒 （ETH） |

8092 苏黎世

瑞士