没有机会的游戏

MSRI 出版物

1996 年第 29 卷

# 求解莫里斯九子棋

## RALPH GASSER

摘要：本文描述了两种搜索方法的组合，用于解决莫里斯九子棋问题。改进的逆向分析算法计算了包含大约 10 10个状态的残局库。使用一个 18 层 的alpha-beta搜索, 并使用这些数据库，证明了初始局面的价值是和棋。莫里斯九子棋是首个并未从基于知识的方法中获益而解决的非普通游戏。

## 1. 简介

近年来，许多游戏都是用计算机解决的，包括Qubic [Patashnik 1980]， Connect-4 [Allen 1989;Allis 1988]和Go-Moku [Allis et al. 1993] 所有这些游戏都是用基于知识的方法求解的。这些方法之所以成功，是因为所有这些游戏的决策复杂度都很低 [Allis et al. 1991]，也就是说，正确的着法往往很容易找到。并不是所有的游戏都能获得相同程度的收益。例如，在跳棋中，国际象棋和围棋的多种着法看起来相当相似。蛮力搜索通常是解决此类游戏的最可行的方法。 国际象棋[Hsu 1990]和跳棋[Schaeffer 1992]的最佳程序都严重依赖于搜索，这一事实支撑了上述观点。

搜索方法不仅对游戏很有用，而且在解决问题的许多方面都很普遍。其中一些算法是计算机科学中最容易理解的算法之一。然而，并非所有的搜索算法都得到很好的研究; 特别是，在大型状态空间进行穷举搜索还处于起步阶段。其中一部分原因是硬件直到最近才发展到可以解决一些有趣的问题。由于硬件和软件的不断改进，搜索方法必须不断地重新评估和适应新的系统设计[Stiller 1991; Lake 1992]。游戏是获取该领域专业知识的理想选择，因为它们是在有限的领域中进行的，在这些领域中，问题的难度可以得到控制，方法的有效性可以得到度量。

我们描述了两种用于解决莫里斯九子棋的暴力搜索方法的组合。改进的逆向分析算法计算了包含大约 10 10个状态的残局库。先是一个18层的 alpha-beta 搜索，然后使用这些数据库，来证明初始局面的价值是和棋。莫里斯九子棋是被解决的第一个没有从基于知识的方法中获益的非普通游戏。

## 2. 莫里斯九子棋

莫里斯九子棋是今天还在玩的最古老的游戏之一[Bell 69]。在世界各地的许多历史建筑上都发现了棋盘。最古老的(约公元前1400年)是在埃及一座神庙的屋顶石板上发现的。其它的则分布在锡兰、特洛伊和爱尔兰等地。

a

B

C

D

e

F

G

1

2

3

4

5

6

7

游戏是在一个有24个点位的棋盘上进行的，棋盘上可以放置棋子。最初，棋盘是空的，两名玩家每人拿着9个棋子。拿着白棋的玩家先下棋。

在摆子时，玩家交替摆放各自的棋子在空点上（图1）。

所有棋子都摆完后，游戏进行到中盘。在这里，玩家可将他的一个棋子移动到相邻的空点。如果在游戏期间任何时候，玩家成功将自己的三枚棋子排成一排 - 这被称为形成*一个三连*， 他可以移除对方不属于任何三连的棋子。在图 2 中，如果白方在开局时通过下在 b6形成一个三连，则可以在a1 上移除对方的棋子，但不能移除 d2 上的棋子。

一旦玩家只剩下三枚棋子，残局比赛就开始了。此时，有三枚棋子的玩家可以将其中一个棋子飞到棋盘上的任何空点。

a b c d e f g a b c d e f g

7 7

6 6

5 5

4 4

3 3

2 2

1 1

**图 2** b6 之后，白方移除 a1 或 b4。 **图 3** 黑方无路可走

游戏以下列方式结束：

* 少于三枚棋子的一方输棋；
* 无路可走一方输棋（图 3）；
* 中盘或残局中，局面重复，则为和棋。

在莫里斯九子棋爱好者中在争论两点规则。首先，在摆子时，可以同时形成两个三连。应该允许移除一枚还是两枚对方的棋子？我们的实现只允许移除一枚棋子。其次，如果己方刚形成了一个三连，但对手所有的棋子也在三连里。可以移除棋子吗？在我们的实现中，可以移除任何棋子。这些规则变化似乎对局面的价值评估影响不大。

## 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。开局在棋盘上放置棋子，在中盘和残局移动棋子。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，而在中局和残局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，在中局和残局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和残局局面的数据库。这是最佳选择，因为逆向分析比前向搜索能更高效地处理循环局面。此外，开局搜索将需要许多局面的价值。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残局局面的价值，这是逆向分析的理想任务。

为开局计算更多的数据库是不合理的，因为数据库大小将甚至比中局残局数据库更大。此外，由于在开局阶段仅对单个局面（空棋盘）的值感兴趣，因此无需存储中间局面的价值。 Alpha-beta 搜索 [Knuth 和 Moore 1975] 是此类问题的理想选择。

## 4. 中局和残局数据库

**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点都可以是空置，也可以被一枚黑棋或白棋占据，因此状态空间大小的上限为324，或近似2.8×1011个状态。这是比较疏松的，可以通过考虑以下游戏限制条件来改善界限：

* 玩家有3到9枚棋子在棋盘上
* 有不可到达的局面
* 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

第一个观察结果允许我们将状态空间拆分为 28 个子空间。图 4数据库依赖项 显示了这些子空间和它们的依赖关系; 因此，7-5 局面（7个棋子对 5 个）只能在确定所有 7-4 和 6-5 个局面的值后进行计算。

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，如果一方有一个三连，则对方不可能所有9没棋子都在棋盘上（图5）。如果2个或3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局面。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，

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*，我们可以预期可以借助对称性减小将状态空间大小降低到原来的1/16。

使用上述三种裁剪方案，在中局和残局阶段共有7,673,759,269个状态。

对于如此大的状态空间，需要一种空间高效的存储方法。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，无需存储状态描述与状态值，因为状态描述在索引中编码。

*P*

3

*P*

1

*P*

4

*P*

2

*P*

5

**图 5.**无法到达的状态 **图 6.**对称轴

理想情况下，完美的哈希函数范围从 0 到状态数减去 1，并且可以快速计算。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局面。在莫里斯九子棋有两种类型的终局局面，两种是玩家走子的输棋：玩家不能走子的局面，以及玩家少于3枚棋子的局面。因此，如果玩家不能走子，我们将该状态标记为输棋。我们也希望将玩家着法少于3枚棋子的局面标记为输棋，但由于我们从状态空间中消除了这些局面，因此我们标记玩家着法的局面可以损失一个三连，并将对手减少到2枚作为赢棋。我们也会标记3枚棋子的玩家在两轮中输掉的局面。这些局面很容易辨认，属于下列类别之一:

* 对手有两个没有共同棋子的三连;
* 对手有两个三连，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棋子。

初始化设置这些胜负后，它将所有剩余局面的价值设置为和棋。然后，迭代过程确定这些"和棋"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。如果玩家着法的局面输棋，玩家可以移动，所有的局面的前身可以标记为胜利的对手。类似地，如果玩家赢得一个局面着法，所有前驱都是对手的潜在输棋。只有当所有的后继都为玩家赢得胜利时，他们才是真正的输棋（图7）。

检查所有后继以确定局面是否输棋是效率低下的。可以通过对每个状态使用Count字段来避免，以存储尚未被证明是对手获胜的后继者的数量。然后，我们只要简单地递减计数，而不是在每次出现一个局面时都会对所有后继进行检查。当Count达到零时，所有后继都被证明是对手获胜的，并且这个局面可以标记为输棋。这种改进的代价是为Count数组使用了额外的内存。

由于我们计算赢和输的深度，所以在Val数组中使用一个字节来存储每个状态的值。在实践中，Count和Val数组可以使用相同的内存，因为这两个信息是从不需要的。

B

A

D

*...*

C

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

同时：只有在仍可以和棋局面的值时，才需要 Count 数组。换句话说，Count 和 Val 值存储为"union";参见表 1。

表 2 显示了用于确定所有状态值的算法。

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，该算法得到了改进，并移植到了各种机器上。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 (Alpha)。只有DEC 3000的表现比苹果Macintosh更好。导致这种意外结果的原因有很多。许多计算机由于其多用户环境，而以较低的优先级执行了我们的代码。此外，许多功能更强大的计算机都经过优化，以处理浮点和向量操作，而其整数性能却没有得到很好的优化。并行计算机还因主内存不足，无法直接访问磁盘。这使得 Macintosh 前端成为 I/O 瓶颈。

Val entry

0 = loss in 0

1 = win in 1

2 = loss in 2

3 = win in 3

Count entry

*: : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : : :*

252 = \draw" (3 successors unknown)

253 = \draw" (2 successors unknown)

254 = \draw" (1 successor unknown)

255 = \draw" (all successors unknown)

表 1. 局面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. 验证。**这种规模的计算几乎不可避免地会包含一些错误。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）。由于这些原因，在30个Sun 集群上验证了数据库（Balz 1994]）。 这大约需要三个月的时间。验证发现6个硬件错误。此外，在莫里斯九子棋代码中发现了一个错误，它允许将某些局面归类为平局，而不是输棋。这些错误导致进一步的不一致，因此必须更正数千个局面。

验证只检查赢棋、输棋、和棋是否一致。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，可以实现完整的验证。即使深度验证没有发现其他错误，我们又如何确定不存在任何错误呢？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常用的函数，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，所有中局和残局局面价值都计算在数据库中，可以通过简单的 18层 alpha-beta 搜索找到初始局面的值。原则上，18 层的开局可以通向除 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72MB，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。

由于开局时没有循环，因此无需计算赢或输的深度。因此，第一个想法将5个局面压缩到1个字节 （35 = 243）来减少数据库大小。但是，为了进一步最大限度地降低存储需求，我们决定将8个状态打包成一个字节。虽然我们每个状态只有一个字节，但如果我们执行两个 alpha-beta 搜索，这足以计算游戏理论价值。在第一次搜索中，我们确定初始局面是否获胜。如果未获胜，我们将执行第二次搜索（使用不同的数据库），以确定初始局面是否输棋。

虽然这减少了数据库的大小减少8倍，但文件大小依然有1GB字节左右，对莫里斯九子棋而言，很明显，最合理的对弈将导致在开局阶段最多形成1到2个三连。

开局数据库

可置换

8

层

16

层

18

层

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

图8 开局搜索

为此，我们决定只使用9-9、9-8和8-8数据库，总共115MB，来证明和棋结果。然后，无法再计算每个局面的正确值。但是，在轮到白方行棋的局面时，我们可以计算正确值的上限，方法是假设所有不在数据库中的局面对白方而言均获胜，如果我们认为所有这样的局面对黑方而言都获胜，则计算出下限。为了证明游戏理论值是和棋，我们必须证明白棋同时具有和棋的上限和下限。

我们在 16 层插入了一个置换表，以进一步减少数据库访问的次数。通常，置换表包含任何层的局面，甚至可能优先于树中更高的层。我们选择只包含 16 层的局面，因为于树中的节点总数相比，减少磁盘 I/O要重要得多。参见图 8。

最后，8层alpha-beta搜索访问的所有局面都存储在一个中间数据库中。这使得游戏可以实时进行，因为对于前 8 层，只能对中间数据库执行搜索。以下是alpha-beta在两次搜索中所访问的8层局面的数量:

已证明 未证明

白方至少和棋 15,501 12

### 白方至多和棋 4,364 29

### 29

**图 9.** 轮到黑方，在26层内输棋

**图 10.** 轮到白方走，会胜利，

187 层内形成三连。

我们会区分哪些局面至少(或者最多)是和棋的，哪些局面不能。这并不意味着局面输棋（或赢），只是使用三个数据库不足以显示其他情况。

在8层约350万个局面中，该表显示只有19906个局面被评估。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。仔细观察这张表，你可能会倾向于认为大多数局面是和棋的。这绝不能是事实;也许着法排序启发式只是成功地选择了和棋着法。

## 6. 结果

**6.1. 数据库局面。**筛选最终数据库以搜索“有趣的”局面是一项艰巨的任务。由于我们对局面的关注没有真正的直觉，因此我们决定以更多的统计方式收集结果。接下来的两个图显示了具有长胜位局面的示例。图 9 中的 3-3 局面是该数据库中最长的序列的示例。图 10 中的 8-7 局面显示了一个获胜局面，其着法顺序最长，直到三连形成。

图 11 显示了为玩家移动而赢得的局面的百分比。它们根据棋盘上的棋子数量进行分组，并相应地进行规范化。从图中可以看出，获胜概率与棋子差异有密切的相关性：棋子越多越好。似乎有一个大约7枚棋子的分界点，所以拥有的棋子越少，获胜就越困难。统计数据还显示，3-3数据库似乎是特殊的。这可能是因为两个玩家都可以跳，本质上这是一个不同的游戏。必须记住，这些统计数字可能产生误导性。例如，在4-3数据库中，如果玩家有4枚棋子要移动，似获胜机会很小。仔细观察数据库后，我们看到所有的胜利都是很简单的局面，玩家可以立即形成三连。

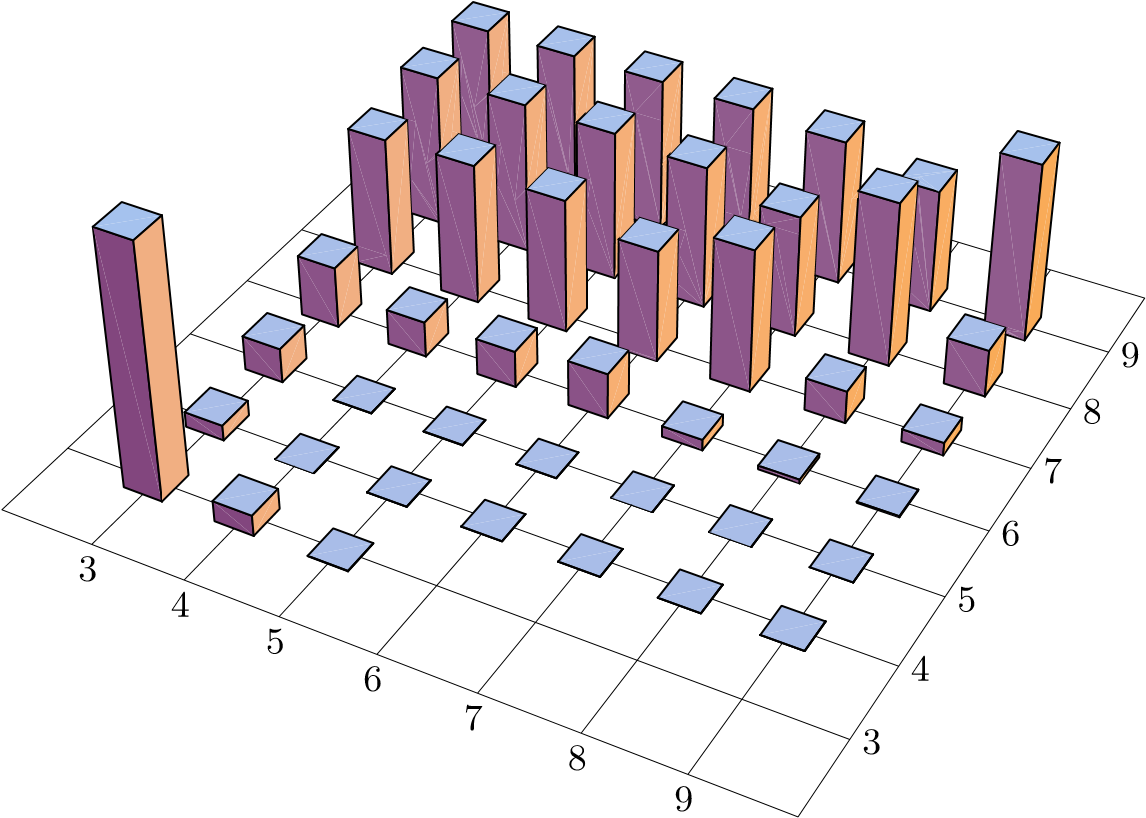


图11。玩家着法的赢百分比，作为初始棋子数的函数。3-3场比赛的最高百分比约为83%。缺少列（如 3-9 场比赛）表示百分比为零。

这些局面不会发生在真正的游戏中，因拥有3个棋子的玩家实际上具有较小的优势。类似的保留适用于所有这些数据库。目前还不清楚如何从其余局面过滤真实局面。

**6.2. 开局。**现在也可以检查一些常见的开局局面。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，大多数可能的走法都是和棋 ，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）的一场比赛中所处的局面。该程序当时仍然基于启发式技术，赢了四场比赛，和了两场。

图12 开局局面，轮到黑方行棋。任何着法都会导致平局。

9

5

**图13** 白色9 输。 **图14** 白 5 输。

由于初始局面是平局，一个自然要问的问题是多久才会出现一个错误。图 14 中的局面显示了我们发现的最早输棋着法之一。这就消除了常见的误解，即形成三连是一个理想的开放策略。白方在前两个动作中遵循这一策略，而黑方则忽略了潜在的三连威胁。随后，白方的第三步输了，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alpha-beta搜索和残局的组合实现的。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无法从基于知识的方法中获益的经典棋类游戏。

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。许多游戏和拼图从结合数据库与搜索中获益，例如Awari, Checkers, the 15-Puzzle or Rubik 's Cube。由于游戏和拼图是一个受限的问题领域，未来的工作将检查如何使更多的一般优化问题能够适应此方法。

## 引用

[Allen 1989] J. D. Allen，"关于connect-four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。

[Lake 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