# 将逆向分析应用于莫里斯九子棋

Ralph Gasser Informatik ETH 8092 Z ü rich 瑞士

## 摘要

本文通过对莫里斯九子棋的逆向分析,给出了由此得到的结果。

首先，我介绍一些用于加速计算的方法，例如数据库的行程编码。

其次，检查结果本身。 虽然许多人直观地把莫里斯九子棋看成是一个相当简单、吸引人的游戏，但我的结果似乎是另一种建议。 一些最佳游戏显然超出了人类的能力，有时需要 150 多层的随机走子才能赢棋胜利。

# 简介

处理这个问题最初是从我的莫里斯九子棋程序发展而来的[Gasser 90]。 由于游戏的不同阶段，显然仅中盘评估功能是不够的。它在残局期间特别不适合。我决定通过详尽地分析某些残局局面来解决问题，而不是对这些情况不完全调整评估功能。

逆向分析是一种最近为解决此类详尽的分析问题而非常流行的技术。 这很适用于国际象棋，其中有些有趣的意外结果出现了。 有关摘要，请参见[Herik 86]。 其中一些结果迫使国际国际象棋联合会更改其规则[Herik 89]。

虽然许多人已经进行了逆向分析，但似乎我认为采用的算法仍有可能改进。 某些改进将非常特定于问题，而其他改进可能具有更广泛的应用程序。

沿着这些线，实施了对莫里斯九子棋的逆向分析。 我将解释如何使用运行长度压缩来加速此过程，并讨论其他更多依赖于问题的注意事项。

然后给出了莫里斯九子棋获得的结果。 尽管这些结果对问题的影响还没有任何影响，即莫里斯九子棋的完整游戏是否为游戏对象，但他们确实认为，莫里斯九子棋在可能性方面比大多数人怀疑的人都要丰富得多。 尤其是两个玩家都只有 3 枚棋子的子游戏几乎几乎没有（0.2%）和棋，但对于人类来说，发现赢棋是极其困难的。

# 莫里斯九子棋

莫里斯九子棋在一个具有 24 个点的棋盘上玩，可在其中放置棋子（图 1）。

最初，棋盘是空的，两个玩家中的每一个都会拥有 9 枚棋子。 有白棋的玩家开始。 游戏包括三个阶段：

* 开局阶段
* 中局阶段
* 残局阶段

在开局阶段，玩家交替放置

a b c d e f g

7

6

5

4

3

2

1

任何空点上的棋子。所有棋子放置完毕后，棋盘可能如图 2 所示。

现在，游戏进入中盘阶段。 在这里，一位玩家可能会将他的任何一枚棋子移动到相邻的空点上。

如果在游戏过程中，玩家成功地连续排列了三个棋子（也称为 “形成三连”），则必须吃掉对手的棋子。 同样不属于三连的任何棋子均可吃掉。 如果所有对手的棋子都是三连的一部分，则可以吃掉任何棋子。 在图 3 中，白方刚下到 b6（开局阶段），现在可以在 a1 上重吃掉黑棋，但不能在 d2 上吃掉黑棋。

图 1 空棋盘

a b c d e f g

7



6

5

4

3

2

1

图 2 开局阶段后

只要至少一名玩家剩下三个棋子，我们就会进入残局阶段。 当轮到他行棋时，有三个棋子的玩家可能会拿他的一枚棋子跳到棋盘上的任何棋盘点。如果他形成了三连，他可以照常吃掉对手的棋子。

游戏以以下方式结束：

* 第一位拥有不到三枚棋子的玩家输了。
* 第一个无法进行合法走子的玩家输棋（图 4）。
* 如果重复某个局面，则游戏为和棋。

a b c d e f g

7



6

5

4

3

2

1

a b c d e f g

7



6

5

4

3

2

1

图 3 白方走b6， a1 或 b4 可被吃掉

图 4 轮到黑方行棋，无子可走

# 逆向分析

## 什么是逆向分析 ？

“逆向分析”表达式在计算机国际象棋圈中广泛使用，它表示一种计算方法，该方法可在特定残局中找到所有可能的棋盘局面的最佳游戏。 在计算机科学中，通常使用类似的术语 “动态规划”。

此方法的特点是执行反向计算。 一开始是立即赢棋或输棋的所有局面。 从这些角度来看，可以计算出 1 层的所有赢棋或输棋局面，然后计算出所有这些局面，即 2 层的赢棋或输棋局面等。

将此方法应用于莫里斯九子棋有两个主要原因。 首先，似乎有可能获得使用国际象棋并不容易看到的方法的新见解。 其次，莫里斯九子棋比国际象棋要简单得多。 这个较小的领域吸引了我，因为从长远来看，可能会完全解决这个游戏。

## 标准逆向分析

逆向分析通过以下三个过程完成：

* + - 初始化
    - 输棋备份
    - 赢棋备份

初始化循环通过所有棋盘并标识残局，其中一个玩家已赢棋或输棋。在莫里斯九子棋中，这些残局要么是闷杀对方的局面，要么是输棋（图 4），要么是玩家走子的局面可以形成三连，从而使对手减少到不到三个棋子的局面。

输棋备份过程会搜索玩家在指定层数中输棋的所有局面。对于这些局面，将生成所有合法的前驱局面。所有这些前驱局面都是赢棋的。

轮到白方行棋



新确定的必胜局面



轮到黑方行棋 必输局面

赢棋备份过程是处理最密集的过程。 首先，确定指定层深度的赢棋局面，并生成其前驱局面。如果前驱局面是输棋的，则其所有可能的后续任务局面都必须是赢棋的。



黑色移动

新确定失位

走子取胜局面的白方

白方行棋

赢棋局面



通过反复对所有盘面应用输棋和赢棋备份程序来完成追溯分析。 当没有发现新的赢棋或输棋时，算法将终止。

## 可能的改进

本节介绍了对上述标准方法的可能改进。 它们并非都同样适用于其他游戏或其他硬件配置，而是专门针对莫里斯九子棋和 Apple Macintosh 操作系统进行了定制。

### 初始化

在最后一节中，我们发现逆向分析的第一步是初始化所有立即赢棋/输棋局面。 作为第一种方法，可以通过查看每个可能的棋盘局面并确定它是赢棋还是输棋来解决此问题。

执行此初始化的成本可能是回归分析总运行时间的较高百分比，尤其是在大多数局面都处于和棋状态时（即，赢棋和输棋备份循环不需要花费很多时间）。 例如，莫里斯九子棋的 4-4 子游戏中，在3,225,597个局面中只有188个是不可和棋的。

要寻找替代方案，让我们看一下四个特定局面对应的四个局面。

图 5 显示了四个白棋的可能配置。 有 4845 种可能的方法可以放置四个黑棋。 如果轮到白方行棋，我们通常会计算所有 4845 可能的局面，并且每个局面都会测试白方是否被闷杀。 但是，有一种更快的方法。 图 6 显示，我们至少需要 9 个黑棋才能封住让白棋不能动，因此任何黑棋配置都不能闷杀这四个白棋。 当轮到黑方行棋时，可以使用类似的过程。 在图 7 中，我们可以看到四个白棋是如何将棋盘分成 3 个区域的。 这些区域包含 1,2 和 17 个空棋盘点。 因此，可以放置 1,2,3,17,18,19 或 20 枚黑棋，这样它们就没有相邻的空棋盘点。 但白棋不能闷杀四个黑棋。 因此，我们可以快速确定 4845 黑方配置中的任何一个都不会让黑方被闷杀。

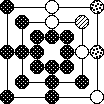




图 5 白棋配置

图 6

黑棋闷杀白方所需的最少数量

图 7 白方无法闷杀 4枚

黑棋

这种同时观察许多类似局面的方法无疑是问题所依赖的。 但是，主要想法通常可以用一种形式或另一种形式来使用。 这当然也适用于国际象棋。 例如，放置了一种颜色的所有部分后，可以确定对手 64 个可能的王局面的子集，这样，只有将王放在这个子集的某个局面上，才能发生将军。

### 赢棋备份

在标准的 “赢棋备份”过程中，我们将通过所有棋盘来确定在指定数量的层中赢棋的局面。 对于其中的每一个，我们都会生成其可能的预处理器局面。 对于尚未确定为 ,赢棋,或 ,输棋,的每个前置任务局面，我们将生成其后续局面。 如果所有后继局面都是赢棋的，我们将有一个新确定的输棋。 此计算的复杂性如下：

O (# Wins \* # AvgPred \* # AvgSucc)，其中

#Wins：指定层数中的赢棋局面数

# AvgPred：前驱局面的平均数量

# AvgSucc：后继局面的平均数量

还有另一种实现相同结果的方法。 我们可以直接搜索文件以查找未识别的局面，而不是从赢棋的局面开始备份和检查这些局面。 对于所有这些局面，我们将对其后续局面执行相同的检查。 以这种方式执行 赢棋备份会导致以下问题：

O ( #Unidentified \* #AvgSucc )

大致来说，这表明，如果

#Win \* #AvgPred > #Unidentified.

第二种方法速度会更快。

在莫里斯九子棋中，这种情况可能会发生。 例如，在 7-3 子游戏中，我们拥有约 3000 万个局面，3 层的赢棋次数约为 900 ,000 。由于 # AvgPred 约为 50，因此使用第二种方法确定 4 层输棋是合理的。

### 行程编码

我的计算是在 Apple Macintosh 上完成的，该机器也用于其他任务，因此只有 30-40 MB 的硬盘存储容量可用。 因此，在没有某种数据压缩的情况下，计算具有 1.5 亿局面（每个局面有一个字节来区分 256 层）的数据库是不可能的。 行程编码似乎非常适合莫里斯九子棋，因为有许多和棋局面可以进行高度压缩。我还试图将 “相似”的棋盘局面转换为相邻的文件局面。 这里的 “类似”棋盘需要相同数量的层，直到游戏结束。 如果可以这样做，则行程长度压缩更有效。

标准行程编码将相同值的序列转换为仅一个值的副本加上序列的长度。对于长序列，它节省了大量的空间，但对于短序列（例如，序列长度为 1），它实际上可能会使用更多的内存。

对莫里斯九子棋数据库进行了研究，表明存在许多相当短的问题，并且某些值（对应于较低的层号的值）更频繁。 因此，将行程编码与可变长度编码结合起来可能是个好主意。 这些想法导致了下面显示的编码方案。

X

价值

值信息

长度信息

|  |  |  |
| --- | --- | --- |
| X | 1111111 | 值 - 127 |

|  |  |  |
| --- | --- | --- |
| 1 | 1 | 长度 |

这基本上是一个行程编码，但由于大多数局面都将在少量的层中和棋或赢棋/输棋，因此这些更频繁的值只使用一个字节存储。 7 位用于存储 0 到 125 层的和棋和赢棋/输棋。 对于高达 381 层，使用第二个字节。由于长度为 1 的运行次数过多，因此值信息（bit X）中的位用于指示运行长度为 1 。 如果 run-length 大于 1，则将一个字节用于从 0 到 127 的 run-lengths 。 0-16383的两个字节，甚至更长的运行时间的四个字节。 此代码具有一定的冗余性，但已证明足够。 下面列出了计算的子游戏的压缩系数。 子游戏的特征是棋盘上棋子的数量，即 5-3 子游戏有 5 枚一种颜色的棋子和 3 枚另一种颜色的棋子。

长度

长度

0

1

0

子游戏 未压缩文件大小 压缩文件大小 压缩因子

|  |  |  |  |
| --- | --- | --- | --- |
| 3-3 | 210 ,140 | 89 ,803 | 2.3 |
| 4-3 | 1 ,725, 960 | 159 ,387 | 10.8 |
| 4-4 | 3 ,667, 665 | 714 | 5136.8 |
| 5-3 | 5 ,484, 540 | 84 ,212 | 65.1 |
| 5-4 | 21 ,938, 160 | 37 ,210 | 589.6 |
| 6-3 | 14 ,319, 168 | 597 ,471 | 24.0 |
| 5-5 | 32 ,907, 240 | 97 ,427 | 337.8 |
| 6-4 | 53 ,696, 880 | 12 ,765, 238 | 4.2 |

以前建议使用行程编码压缩最终数据库[Marris 89]。就我所知，以前没有尝试使用压缩数据执行逆向分析。

在逆向分析期间，当我们在文件中执行顺序搜索时（例如，当我们需要具有给定值的所有棋盘局面时），运行长度压缩会加快我们的搜索速度，因为文件较小。只有当我们需要特定棋盘局面的值时，才会出现问题。 使用未压缩的数据库时，我们完全知道可以找到该值的局面。 使用运行长度压缩，我们基本上必须在整个文件中进行搜索，直到达到所需的局面。 通过在许多地方使用指向压缩文件的索引，我们可以加快此速度。 使用此索引，我们可以将顺序搜索限制为文件的较小部分。 如果压缩因子足够大，则此顺序搜索仍然比使用未压缩数据更快，因为文件的较大部分可以保存在内存缓存中，从而消除了缓慢的磁盘访问。 要记住的另一个事实是，在反向分析开始时，压缩文件的大小较小，并且增大我们确定的更多新局面。 这意味着对于较小的层深度，当发现大多数新局面时，我们仍有一个较小的文件，也就是说，快速访问。 当压缩文件随层深度的增加而增大时，我们必须检查的局面数量通常会变小，从而使快速访问变得不太重要。

# 结果

以下所有结果都是使用具有 8 MB 内存的 Macintosh IIx 获得的。运行时间从几分钟（4-4 个子游戏）到数周（6-4 子游戏）不等。 更精确的计时不可用，因为此程序仅在未使用计算机时运行。

对于每个数据库，都提供了一些简短的统计信息以及一个最困难的功能示例。 如果有多个最佳走子，则替代项将缩进。 术语如下： W-B 子游戏包括具有 W 白棋和 B 黑棋 ，也即所有棋盘局面。 5-4 子游戏包含所有具有 5 个白棋和 4 个黑棋的局面。有 5 个黑棋和 4 个白棋的局面可以通过切换棋子颜色转换为 5-4 局面。 如果玩家能够赢棋，则称为“赢棋局面”，同样也会导致输棋。 “最大赢棋层”和 “最大输棋层”列为子游戏中的某个局面提供了最长的赢棋或输棋顺序。 下此序列可能会导致关闭三连并进入不同的子游戏。 “最大赢棋层”和 “最大输棋层”测量层数量，直到最终达到赢棋/输棋,，而不是层数量，直到子游戏发生更改。

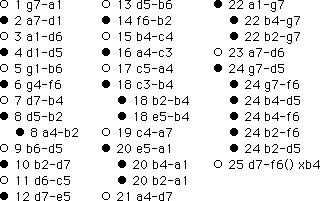
这些结果证明了[disprove 87]中的许多声明。 似乎人类对这些组合游戏并不特别擅长。

## 3-3 子游戏

这是第一个要检查的子游戏，到目前为止，它也是最有趣的子游戏，换句话说，和棋百分比非常低（约 0.2%）。 这与人类直觉矛盾，直觉是 3-3 子游戏几乎始终是一个和棋。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 赢棋数 | 和棋数 | 输棋数 | 最大赢棋层 | 最大输棋层 |
| 轮到白方行棋 | 47 167 | 96 | 9 659 | 25 | 26 |

下面显示了 25 层赢棋局面的示例，以及可能赢棋的赢棋序列：

a b c d e f g

7



6

5

4

3

2

1

白方行棋， 25 层获胜

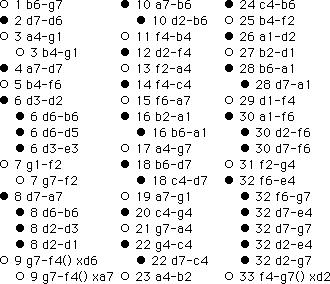
## 4-3 子游戏

这个子游戏比 3-3 有些有趣，但我们仍然可以从这个数据库中获得一些新的见解。首先，观察到拥有 4 枚棋子的玩家的最大赢棋层数为 1 ，换句话说，只有当他轮到自己的时候，他才会赢。他有一个三连，一个在实际比赛期间不会发生的相当有病理意义的情况。其次，如果是轮到黑方（3 枚棋子）行棋，他就不能

输 ！ 显然，通过飞棋能力获得的额外走子性（即剩余的 3 枚棋子）相当有价值。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 赢棋数 | 和棋数 | 输棋数 | 最大赢棋层 | 最大输棋层 |
| 白方行棋 | 75 ,397 | 681 ,906 | 3 ,095 | 1 | 32 |
| 黑方行棋 | 102,281 | 658 ,117 | 0 | 33 | - |

以下显示了 33 层赢棋局面的示例，以及可能赢棋的赢棋序列：

a b c d e f g

7



6

5

4

3

2

1

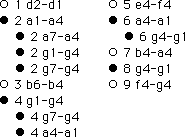
白方先行，在 33 层赢棋

## 4-4 子游戏

这是一个非常无聊的子游戏。 几乎没有任何赢棋局面（约 0.005%），甚至更少的输棋局面。但是，可以赢棋的胜利的性质与之前的不同。 赢棋不是通过吃掉对手的棋子而实现的，而是通过闷杀对方来实现的。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 赢棋数 | 和棋数 | 输棋数 | 最大赢棋率 | 最大输棋层 |
| 白方行棋 | 159 | 3,225,409 | 29 | 9 | 8 |

以下示例显示，即使是最长的赢棋序列，也很容易被人类识别。

a b c d e f g

7



6

5

4

3

2

1

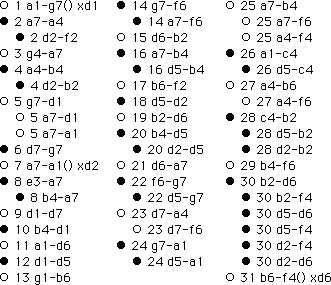
轮到白方行棋，在9层赢棋

## 5-3 子游戏

5-3 子游戏类似于 4-3 子游戏。 拥有 5 个棋子的玩家拥有许多赢棋的局面，但由于他们处于如此浅的深度，他们似乎不太可能在实际比赛中达到。拥有 3 个棋子的玩家都有有趣的赢棋序列，但其中很少有人认为这个子游戏本质上是和棋。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 赢棋次数 | 和棋数 | 输棋数 | 最大赢棋层 | 最大输棋层 |
| 白方行棋 | 580,660 | 1,999,730 | 0 | 3 | - |
| 黑方行棋 | 6 ,301 | 2,564,412 | 9 ,677 | 31 | 2 |

以下示例显示要赢棋此子游戏，必须首先将对手减少为 3 枚。 这表明一种针对不完全对手赢棋和棋局面的方法：具有 3 个棋子的玩家应尝试将对手减少到 3 个棋子，因为正确的 3-3（或 4-3）游戏比正确的 5-3 下更困难。

a b c d e f g

7



6

5

4

3

2

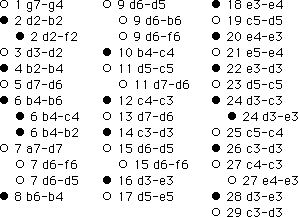
1

轮到白方行棋，31层获胜

## 5-4 子游戏

5-4 子游戏基本上也是和棋。 所有可能的胜利都是通过闷杀对方来实现的。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 赢棋数 | 和棋数 | 输棋数 | 最大赢棋层 | 最大输棋层 |
| 白方行棋 | 9 ,889 | 10,300,599 | 8 | 29 | 4 |
| 黑方行棋 | 51 | 10,308,935 | 1 ,510 | 5 | 28 |

a b c d e f g

7



6

5

4

3

2

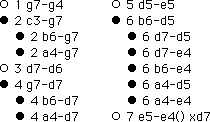
1

以 29 层的速度实现赢棋

## 6-3 子游戏

6-3 子游戏类似于 5-3 子游戏。但由于还有额外的棋子，白方不会输。 可能的胜利是在浅深处，但人类似乎很奇怪，很难找到最佳的游戏。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 赢棋次数 | 和棋数 | 输棋数 | 最大赢棋层 | 最大输棋层 |
| 白方行棋 | 2,752,371 | 4,122,949 | 0 | 7 | - |
| 黑方行棋 | 0 | 6,683,320 | 192 ,000 | - | 6 |

a b c d e f g

7



6

5

4

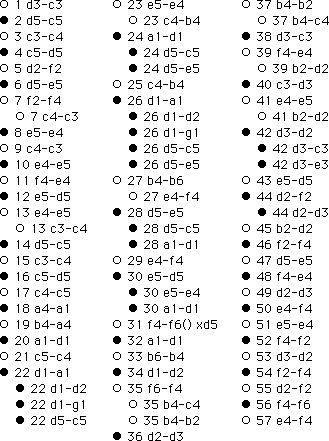
3

2

1

在 7 层中实现赢棋的白方走子

## 5-5 子游戏

所有赢棋都是通过闷杀对手来实现的。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 赢棋数 | 和棋数 | 输棋数 | 最大赢棋层 | 最大输棋层 |
| 白方行棋 | 28 ,819 | 30,881,800 | 4 ,295 | 57 | 56 |

a b c d e f g



以 57 层的速度实现赢棋

## 6-4 子游戏

这是迄今为止具有最长赢棋顺序（即 157 层）的子游戏。 我所执行的测试表明，发现赢棋序列明显超出了人类能力。

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 赢棋次数 | 和棋数 | 输棋数 | 最大赢棋层 | 最大输棋层 |
| 白方行棋 | 5,985,293 | 19,780,495 | 4 | 157 | 2 |
| 黑方行棋 | 22 | 24,115,798 | 1,649,972 | 3 | 156 |

a b c d e f g

7



6

5

4

3

2

1

遗憾的是，赢棋顺序过于精细，无法在此处显示，但可以从作者处获得。

白方先行，在 157层中夺取胜利

# 结论

这项工作的目的之一是更好地了解逆向分析的能力和限制。 使用前面所述的改进和其他 Macintosh 特定调整，逆向计算的速度提高了 10 倍以上。 进一步的工作将专注于将 莫里斯九子棋获得的经验推广到反向分析的其他应用中。

第二项相当长期的这项工作是对 莫里斯九子棋的完整分析。 这是否可以在不久的将来实现取决于游戏本身。 如果莫里斯九子棋是一个和棋，那么将和棋许多甚至大多数局面，并且行程编码方案在加速计算和减少内存空间要求方面都将有效。 目前，我似乎已经达到了 Macintosh 的限制。 将来，我必须将代码移植到更快的计算机上，或者加快计算速度。通过不区分赢棋所需的层数，而只将局面分类为赢棋、输棋或和棋，可以进一步加速。

# 参考

[Gasser 90]加瑟河 ,启发式搜索和逆向分析：它们对莫里斯九子棋的应用,，文凭论文，ETH Zü rich，1990 年 2 月。

[Herik 86] van den Herik H.J.和 Herschberg I.S., ,基于数据库的数据库,, ICCA Journal 9（1）, p. 29（1986 年 3 月）。

[Herik 89] van den Herik H.J.和 Herschberg I.S.，《重新访问的 50步 走子规则》，ICCA Journal 12（3）, p. 192（1989 年 9 月）。

[Marris 89] Marris, C.A., ,Compressing a chess-endgame database,, ICCA Journal 12（1）, p. 22（1989 年 3 月）。

[ller 87] ller, R.F., ,hle,，ECON Taschenbuch Verlag，1987 年 10 月。