请参阅以下出版物的讨论、统计信息和作者简介：https://www.researchgate.net/publication/305413042

[规则变化对莫里斯九子棋完美游戏数据库的影响](https://www.researchgate.net/publication/305413042_The_Effects_of_Rule_Variations_on_Perfect_Play_Databases_for_Nine_Men%27s_Morris?enrichId=rgreq-a4ff2bb6f0aeef5a99a66a344e587b61-XXX&enrichSource=Y292ZXJQYWdlOzMwNTQxMzA0MjtBUzozODU1NjgzMDAxMjYyMDhAMTQ2ODkzODA1NDUxOQ%3D%3D&el=1_x_3&_esc=publicationCoverPdf)

**研究 · 2016 年 7 月**

DOI： 10.13140/RG.2.1.4972.4407

引文读取

1. 341
2. **作者：**

[韦斯利 ·卢韦尔](https://www.researchgate.net/profile/Wesley_Loewer?enrichId=rgreq-a4ff2bb6f0aeef5a99a66a344e587b61-XXX&enrichSource=Y292ZXJQYWdlOzMwNTQxMzA0MjtBUzozODU1NjgzMDAxMjYyMDhAMTQ2ODkzODA1NDUxOQ%3D%3D&el=1_x_5&_esc=publicationCoverPdf)

罗斯林学院

**4 出版物 1 引文**

[请参阅配置文件](https://www.researchgate.net/profile/Wesley_Loewer?enrichId=rgreq-a4ff2bb6f0aeef5a99a66a344e587b61-XXX&enrichSource=Y292ZXJQYWdlOzMwNTQxMzA0MjtBUzozODU1NjgzMDAxMjYyMDhAMTQ2ODkzODA1NDUxOQ%3D%3D&el=1_x_7&_esc=publicationCoverPdf)

**本出版物的一些作者也在研究这些相关项目：**

 规则变化对九人 Morris View 项目的完美游戏数据库的影响

本页面后面的所有内容均由 Wesley Loewer 于 2016 年 7 月 19 日上传。

用户已请求增强下载的文件。

**规则变化对莫里斯九子棋完美游戏数据库的影响**

# 韦斯利湾 勒韦尔

wesl @ rosslynacademy.com

*罗斯林学院，内罗比，肯尼亚*

# 摘要

当 莫里斯九子棋的游戏最初被显示为和棋时，它是使用一组规则解决的，这些规则与 Weltmu?hlespiel Dachverband（世界 莫里斯九子棋协会）批准的官方锦标赛中使用的规则略有不同。 从那时起，分析并解决了其他具有明显不同规则的 Morris 变体，如 Morabaraba 和 Lasker Morris 。 但是，尚未研究标准游戏中更为细微的规则变化的影响。 本文探讨了这些规则变化对莫里斯九子棋标准游戏的影响。

逆向分析用于为标准游戏中常用的四种规则组合创建完整的完美游戏数据库。 虽然这些数据库显示理论游戏值是针对所有四种组合和棋的，但它们还表明这些规则变体可能会对游戏的完美表现产生意外影响。

# 1. 简介

莫里斯九子棋（也称为 Mill 或 Merrills）是在两个玩家之间玩的零和完美的信息棋盘游戏。 每个玩家都有九个棋子或石头，它们以颜色（通常为白方和黑方）区分。 棋盘由三个同心正方形组成，线段连接每一侧的中点，如图 1 所示。 24 个交点和角点组成了可以走棋的位置。 按照惯例，白方首先走棋。

图 1： 莫里斯九子棋棋盘

# 2. 游戏规则

由于本文专门论述了莫里斯九子棋的规则，因此必须详细说明这些规则。 游戏具有不同的开始阶段、中间阶段和结束阶段，在这些阶段中不同的规则有效。 但是，在游戏的所有阶段，都适用以下基本规则。

1. 只要玩家设法把三枚排成一行，就称为 “形成三连”，然后该玩家就可以吃掉对手的其中一枚棋子。
2. 在三连中自身的任何对手棋子都不会被吃掉。
3. 如果玩家仅剩下两个棋子，或者在中被闷杀且无法走子，则该玩家已输棋。

## 2.1 开场游戏： 放置

在游戏的开局阶段，每个玩家轮流在位于交点或角点处的 24 个点之一上放置一枚棋子。 为了识别局面，通常使用类似于国际象棋代数记数法的记数法。 例如，假设白方的开头是 d6，黑方的后面是 c5，然后白方的回答是 d2 。 这三个层产生的棋盘如图 2 所示。 （术语 “层”是指一个玩家所执行的操作，而术语 “走子”是指两个玩家所执行的操作。 因此，如果已经走棋了 10 层，则每个玩家都已经进行了 5 个动作。 可以将层视为 “半个行棋”。）

白方

黑方

1.

d6六

c5

2.

d2

a

b

c

d

e

f

g

1

2

3

4

5

6

7

图 2： 三层后

使用黑方走棋 c4 和白方走棋 f4 继续走棋。

白黑

1

2

3

4

5

6

7

1. d6c5
2. d2c4
3. f4

abcdefg defg

图 3： 黑方走子并形成三连

在 黑方 下次走子时，她可以在 c3 上放置一个棋子以形成三连（连续三个），从而吃掉其中一个白方的棋子（如 f4），并将其从游戏中永久移除。 （为了清晰起见，本文将白方称为 “他”，黑方称为 “她”。）吃掉符号为 ×符号，如 c3 × f4，如图 4 所示。 这一开场阶段继续进行，直到两名运动员全部 9 个棋子（18 层）。

白黑

1

2

3

4

5

6

7

1. d6c5
2. d2c4
3. f4c3 × f4

a b c d e f g

图 4： 黑方刚刚形成了一台形成三连，吃掉了 f4 上的白方棋子

## 2.2 中盘： 走子

所有 18 枚棋子都放置完毕后，游戏就会过渡到走子阶段，在此阶段，玩家会将它们的棋子沿直线从一个局面移动到相邻的空局面。 图 4 中的上述游戏可以继续，直到达到图 5 中的以下状态。

a

b

c

d

e

f

g

1

2

3

4

5

6

7

图 5： 第 19 层，走子阶段的开始

此时，白方可以从 d1 走子到 a1，用 d1 – a1 表示。 现在，黑方 的唯一合法行动是 a4 – a7 。 显然，黑是没有聪明地行棋的。 由于她无法走子（如图 6 所示），因此当 a1 – a4 和黑方输棋时，白方会闷杀她。 虽然他没有吃掉到任何东西，但还是赢了白方。

黑方

白方

a4 – a7

d1 – a1

19.

a1 – a4

20.

a

b

c

d

e

f

g

1

2

3

4

5

6

7

图 6： 黑方无法走子，白方获胜

## 2.3 结束游戏： 飞棋

如果游戏继续进行而不阻塞任何一侧，则在游戏达到图 7 所示状态之前，可能会吃掉棋子。 黑方刚刚关闭了一台形成三连，吃掉了其中一个白方的棋子，仅留下三个白方的棋子。 一旦玩家下至三枚棋子，他就不再局限于沿直线走子到相邻局面，但现在可以跳转到棋盘上的任何局面。

a

b

c

d

e

f

g

1

2

3

4

5

6

7

图 7： 飞棋阶段开始

白方可以从 a7 跳到 f4 以形成三连。 如果他这样做，他将被迫吃掉 e3，因为其他棋子在三连中，因此受到保护。 但是，现在黑方也只有三个棋子，因此她也可以飞棋，最终赢得游戏，如图 8 所示。

黑方

白方

a7 – f4

。

×

e3

b2 – a4

41.

g4 – b2

b6 – c4

×

f4

a

b

c

d

e

f

g

1

2

3

4

5

6

7

图 8： 黑得胜

# 3. 规则变体

如前所述，吃掉规则规定当玩家形成三连时，他可以吃掉对手的棋子；但当前位于三连中的棋子将受到保护，无法被吃掉。 在某些需要特别注意的情况下，这两个规则可能会相互冲突。

## 3.1 始终受保护或始终吃掉

图 9 中的中盘局面显示了这两个规则如何相互冲突。 白方轮到走了，他可以通过走子 d5 – e5 来形成三连。 一个规则表示，自形成三连后，白方可以吃掉一个棋子，而另一个规则则表示黑方的棋子都受到保护，无法吃掉，因为它们都在三连中。

有些人倾向于认为黑棋在形成三连中总是受到保护，因此白方无法吃掉任何东西。 其他人优先考虑以下规则：由于 白方 已形成

1

2

3

4

5

6

7

abcdefg defg

图 9： 白方走子。 白方能通过形成三连来吃掉一枚棋子东西吗 ？

他有权吃掉一些东西，因此即使他们都在形成三连里，他也可以吃掉黑棋之一。 这两种规则变体分别称为 “始终受保护”规则和 “始终吃掉”规则。

## 3.2 Take 1 或 Take 2

在开局阶段，玩家可以同时形成两个三连。 在图 10 中，通过走棋 f4 可以实现此目的。

1

2

3

4

5

6

7

a b c d e f g

图 10： 黑方可走棋 f4 键同时关闭 2 台形成三连

根据这一规则，如果黑方走棋 f4，她就可以吃掉前面所述的白方棋子之一。 其他人规定，由于黑方同时形成两个三连，因此她有权吃掉两个棋子。 这两个规则变体分别称为 “Take 1”规则和 “Take 2”规则。

**3.3 谁使用了哪些规则变体 ？**

当拉尔夫 ·加瑟（Ralph Gasser）于 1993 年最初在他的创业工作中解决了莫里斯九子棋的问题时，他使用了 “始终吃掉”（Always Capture），采用了 1 个规则组合（Gasser，1993 年）。 这就为后来的研究涉及莫里斯九子棋（Lincke,1994;Edelkamp, Sulewski, & Yu?cel,2010;G?evay & Danner,2014）的人确立了先例。 在确定他使用了哪些规则时，Gasser 表示：“这两种规则的变化似乎都不可能影响游戏的价值”（Gasser，1996）。 这是这一陈述，它为此分析提供了动力。 事实上，这一假设是否正确 ？ 它提出了一个问题：“规则变化实际上是否会影响初始理论游戏值，如果不影响，它们会对游戏产生什么影响 ？”

相比之下，在 Gasser 工作时，管理官方锦标赛的 Weltmu?hlespiel Dachverband（WMD）规则始终受到保护，采用 2 条规则（Weltmu?hlespiel Dachverband，2008）。 使用这些官方 WMD 规则的游戏首先得到了解决，并在 2000 年被 Peter Stahlhacke（Stahlhacke，2016 年）认为是他解决 Lasker Morris（Stahlhacke，2003 年）之前的一项和棋。

为了减少锦标赛的和棋次数（Lehner，2016），WMD 在 2009 年将其官方规则更改为 “始终吃掉”，请采用 2 条规则（Weltmu?hlespiel Dachverband，2010 年）。 使用新的 WMD 规则的游戏是在 2014 年首次解决的，经证明是由由 Alexander Szabari, Jozef Mičko, Ferenc Volman 和György Bándy（Brillant Mill Team，2016）组成的程序员和玩家的 “Brillant Mill Team”生成的完整的完美游戏数据库。 作为本文研究的一部分，作者于 2014-15 年生成了所有四种组合的完美游戏数据库。

为了说明这些变体是如何全部使用的，下表显示了各种研究人员、软件开发人员和在线游戏网站使用了哪些规则。

|  |  |  |
| --- | --- | --- |
|  | **始终受保护** | **始终吃掉** |
| **采用 1** | Muehle 2.4（Richard Fischer）  3 D Morris（Lobstersoft） | 加瑟  林克  Edelkamp、Sulewski 和 Yu?cel  马洛姆（G?evay & Danner）  九个男士莫里斯（Carl von Blixen）  弗廖迪  智能小游戏  孔雷加特 |
| **拍摄 2** | 穆勒达滕班克（斯塔尔哈凯）  WMD（2009 年之前） | Brillant 三连（Brillant 三连团队）  普拉约克/库尔尼克  现行规则 |

表 1： 分布各种研究人员、软件和游戏站点使用的规则

**4. 这四种游戏变化的解决程度如何 ？**

将游戏描述为已解决时，Victor Allis 列出了已广泛采用的三个解算级别（Allis,1994）。

* **超弱解：** 最初的理论游戏值是已知的。
* **弱求解：** 从最初的角度来看，有一种已知的策略来实现理论游戏的价值。
* **强解析：** 从任何合法局面来看，都有一个已知的策略来实现该局面的理论游戏价值。

Schaeffer 和 Lake 定义了一个额外的水平（Schaeffer & Lake，1996 年）。

* **超强解：** 从任何合法局面来看，都有一种已知的策略来提高恶意（非完美）对手出错的可能性，从而导致结果超过该局面的理论游戏值。

最后，G?evay 和 Danner 又增加了一个级别（G?evay & Danner，2014 年）。

* **扩展的强大解决方案：** 游戏得到了极大的解决，即使是在合法性上无法达到的局面上也是如此。

在此研究中，使用扩展的强大解决方案解决了四个规则组合中每个规则组合的数据库问题。 因此，即使不可能的局面（如图 11）仍进行了评估。

1

2

3

4

5

6

7

abcdefg defg

图 11： 局面不可能，两侧都可以走子和和棋

扩展解决方案包括棋盘上的棋子局面（如上图 11 所示）以及板上的棋子数量比可能少的开局（如第 6 层 2 与 2）。 包括这些不可能的局面的目的是使数据库更易于生成，因为该程序不再需要确定给定局面是否合法。 它还具有使比较不同规则的数据库更容易的优势，因为它们将包含所有相同的局面。 例如，图 12 显示了可以使用 “始终吃掉”规则获取的 “层 18”的局面，但不能使用 “始终受保护”规则。 尽管有这种情况，但使用这两个规则对此局面进行了计算。

白黑

1

2

3

4

5

6

7

* 1. "
  2. d7d5D7D5D7D5D7D@@
  3. f6e5 × f6
  4. g7 × e5e5 × g7 5. g7 × e5e5 × g7 6. g7 × e5e5 × g7.g7 × e5e5 × g7
  5. g7 × e5e5 × g7
  6. g7 × e5abcdefg

图 12： 始终吃掉规则的受控制但合法的第 18 层局面

所遵循的唯一限制是玩家可能在棋盘上的棋子数不能超过玩家放置的棋子数。 例如，在上面的图 12 中，相同的安排不能用于第 5 层，因为白方显示为棋盘上有 3 枚，尽管他只玩了两次。

# 5. 逆行分析生成数据库

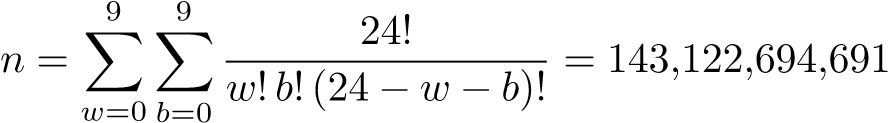
为 中盘和 残局（走子和飞棋阶段）生成数据库的方法非常相似。 因此，这两个数据库通常组合在一起，有时统称为 “中盘/残局 数据库”。

使用类似于 Gasser 方法（Gasser，1990）的技术进行倒退分析，在此方法中反向分析游戏，从游戏结束开始，向后向开始。 Gasser 对 中盘/残局 使用了逆行分析，然后从一开始就使用了 18 层向前 alpha-beta 搜索（Gasser，1996），从而解决了游戏问题。 本文研究中使用的方法是通过从游戏结尾一直到最开始应用逆行分析来解决变化，从而生成完整的完美游戏数据库。

在数据库生成器编程中采用的一般方法是支持简单性和准确性。 由于有足够的内存和计算能力（见附录 A），为了提高可靠性，我们避免了巧妙的快捷方式和内存缩减技术。

## 5.1 对称

在评估局面之前，首先需要列出所有可能的局面。 在 24 个棋盘局面上放置多达 9 个颜色的局面（无论是否合法）可通过以下公式找到：



通过利用棋盘的高度对称程度，可大幅减少需要评估的局面数量。

图 13： 板对称

图 13 所示的对称包括四个反射（垂直轴、水平轴和两个对角轴）、三个旋转（90 在这些对称中，实际上只需要四个，其余的则是冗余的。 选择的四个对称是三个反射（垂直轴、水平轴和正倾斜的对角轴）和反转。 其他对称可以表示为这四个对称的组合。 例如，旋转 90°与垂直轴上的反射相同，然后是对角线轴上的反射。

可以在中间和结束游戏中使用的另一个对称是颜色对称。 如果白方转身走棋某个局面，则这与黑方转身走棋时反转所有颜色的效果相同。

还有一个对称可以使用。 对于 3 比 3 的特殊情况，因为白方和黑方都可以飞棋，所以所有三个正方形都可以相互交换。 这将进一步减少需要评估的唯一局面的数量，但与整个数据库相比，只有一小部分。 因此，未使用这种额外的对称。

为了利用这些对称，我们创建了一个完美（可逆）哈希函数，该函数将棋盘的状态降至单个数值。 为了确定某个局面是否冗余，对该局面应用了四个变换对称的所有 16 个组合，从而创建了最多 16 个唯一的哈希值。 如果原始哈希值与这 16 个哈希值中的最小值匹配，则在数据库中使用该局面。 否则，该局面被视为冗余局面。 使用这些对称，需要评估的唯一局面总数在 残局 数据库中降至 8,904,593,601，在开局库中降至 17,874,891,168 。 （差异原因见第 5.5 节。）

## 5.2 数据表示

必须作出关于如何表示数据的两项重要决策：

1. 如何表示哈希值 ？
2. 如何表示理论游戏值 ？

### 5.2.1 哈希值

为了确定哈希值，棋盘上的三个同心方形中的每一个都被描述为 8 位数的 base-3 数字。 每个数字使用值 0 =空、1 =白、2 =黑表示哪个数字位于正方形上的 8 个局面。 这三个 8 位数字（每个正方形一个）可以单独操作，也可以合并为一个 24 位 base-3 数字。 为了提高速度和简化位操作，在应用转换时，这些 base-3 值暂时转换为 base 4 。） 选择最低有效位数字（LSD）作为正方形 d7 的顶部，沿方形顺时针方向前进到外部正方形 a7 的最高有效位（MSD），如图 14 所示。 外部正方形被认为是最不重要的 8 位数字，而内部正方形则是最重要的。

图 15 显示了一个示例板状态，其中内部方由 000000123 表示，中间方由 000200103 表示，外部方由 02001003 表示，表示为单个 24 位 base-3 哈希值 00000012 00020010 02001003 。

77

0

1

~~2~~

3

4

5

6

7

8

9

~~10~~

11

12

13

~~14~~

15

16

17

~~8~~

1

19

20

21

~~2~~

2

23

a

b

c

d

e

f

g

66

55

44

33

22

11 abcdefg

图 14： LSD（0）至 MSD（23）图 15：哈希 = 00000012 00020010 02001003

可能的最大 24 位 base-3 数字 324-1 需要 39 位，因此这些值存储为 40 位（5 字节）数字，仅浪费 1 位。

### 5.2.2 理论游戏值

为了容纳最大可能层数，使用了 16 位（2 字节）数字来存储每个局面的理论游戏值。 经证明，这是个明智的选择，因为最大层数为 331，不适合 1 字节。

由于莫里斯九子棋的优胜者始终是最终行动的玩家，因此可以通过以下两种简单方式之一来表示局面的理论游戏价值：

* 游戏结束时的无符号层数（一半走子）（奇数层是成功的，evens 是失败的）
* 游戏结束时的已签名完全走子数（肯定是胜负，负值是输棋）

选择了使用完全走子的后一个选项，因为这样可以更轻松地比较游戏值，即，肯定值优于否定值。 选择零表示和棋，以便获胜值>和棋值>输棋值。 如果要将这些值和棋在数字线上，则它们的顺序如图 16 所示。

和棋

更好

更糟的

-1000-5005001000

输棋 <0 胜> 0

图 16： 走子以赢得或放弃数字线，立即获胜 = 1

虽然以这种方式存储值很容易，但其不理想的特征是，较小的正值优于较大的正值，因为在 3 个走子中获胜比在 10 个走子中获胜要好。 输棋与负输棋相同。 为了避免这种情况，通过从 “偏移”常量中减去值来 “反转”这些值。 例如，使用偏移 1000 时，1 走子中的成功存储为 999，1 走子中的失败存储为 - 999，2 中的成功存储为 998，依此类推。 现在，可以将任意两个游戏值（无论是赢、输还是和棋）与简单有符号整数比较进行比较，如图 17 所示。

和棋

更好

更糟的

-1000-5005001000

输棋 <0 赢棋> 0

图 17： 反向转向获胜或失败，立即获胜 = 999

## 5.3 与先前实施的差异

此研究中使用的逆行分析实施与先前研究人员使用的分析实施存在一些差异。

### 5.3.1 使棋盘恢复走子前状态

在分析 checkers、Lake、Schaeffer 和 Lu 游戏中的某个局面时，他们的程序针对每个可能的走子调用了一个函数 MakeMove（），然后使用 GetValue（）获取此新局面的游戏值，然后使用 UnMakeMove（）（Lake、Schaeffer、& Lu、1994）将棋盘返回到其原始状态，如图 18 所示。

for( i = 0; i < numb; i++ ) {

MakeMove( pos, moves[ i ] );

childvalue = GetValue( ... ,Index( pos ) );

UnMakeMove( pos, moves[ i ] );

...

图 18： 使用 UnMakeMove（）的代码段

此研究使用的方法是在走子之前复制原始局面。 找到新局面的值后，拷贝就被丢弃了，无需 “撤消”走子。 实际代码看起来不同，但等效概念的表示如图 19 所示。 局面存储为枚举值的 24 元素数组，每个局面需要 24-96 个字节（取决于编译器）。 由于可以执行小数据枚内存拷贝的速度，此方法证明速度更快。

for( i = 0; i < numb; i++ ) {

CopyPosition( pos\_copy, pos );

MakeMove( pos\_copy, moves[ i ] );

childvalue = GetValue( ... ,Index( pos\_copy ) );

…

图 19： 使用 CopyPosition（）的代码段

### 逆行步骤

与某些实施相比，另一个区别在于如何实际完成逆行步骤。 一种方法是扫描整个子空间，查找可能导致游戏值已确定的相对局面的局面。 另一种方法是采用游戏价值已确定的每个局面，然后通过反向应用规则确定可能导致当前局面的所有先前局面。 无论采用哪种方法，任何导致对手 “六层输棋”的局面都将成为当前玩家 “7 层以内获胜”的局面。

尽管确定玩家可以从给定局面进行的合法性走子是一枚棋子简单的事情，但确定 莫里斯九子棋的给定局面的所有可能方式是一个相当复杂的过程。 尽管后一种方法通常速度更快（Cazenave & Nowakowski，2015），但为了简化和可靠性，选择了前一种扫描整个子空间的方法。

### 5.3.3 哈希函数

与早期研究人员的方法相比，最显著的区别在于游戏值的查找方式。 通常，哈希函数会将游戏状态转换为索引，然后使用该索引查找表中的游戏值。 理想情况下，哈希函数将生成 0 到（N-1）范围内的索引，其中 N 是可能的游戏状态数。 但是，创建理想的散列函数（如这样）可能会很困难。 Gasser 从一个数学函数开始，该函数的范围超过了理想范围 18.26%，然后他显示比理想范围高 5.59%（Gasser，1995）。

在这项研究中，我们尝试了一种不同的方法。 确定了实际理想索引，从而导致数据库不包含任何无关项，而不是使用数学函数来近似理想索引。 为此，每个子空间使用了两个并行数组。 一个是一个哈希数组，其中包含每个可能的子空间哈希值的排序列表，而另一个数组包含相应的游戏值。 要查找游戏值，首先使用二进制搜索在哈希数组中查找游戏状态的哈希值。 然后使用生成的索引直接从值数组中检索游戏值。 实际上，散列数组搜索充当理想的散列函数。

当然，这种方法需要更多的哈希数组内存，但额外的内存要求（平均值 = 0.4 GB，最大值 = 2.75 GB）在硬件限制范围内。 保存这些哈希数组同样占用了更多磁盘空间。 这有点被较小的数据库文件所抵消，因为它们没有无关的条目。 由于相同的哈希文件可以同时用于 中盘/残局 数据库和所有规则变体的开局库，因此每个子空间的文件大小得到了显著减小。 例如，5-4 哈希文件可与 5-4 子空间一起用于中盘规则，9 个 5-4 子空间用于所有四个开局游戏规则组合，总共 38 个子空间共享相同的 5-4 哈希文件。 最后，哈希文件将总文件空间增加了 14%。

## 5.4 生成 中盘& 残局 数据库

一旦确定了唯一的局面，就可以开始生成 残局 数据库的过程。 由于计算了不同的白方与黑方（W-B）子空间，因此始终假设白方棋子的数量至少与黑方棋子的数量相同。 对于棋盘上有更多黑棋的情况，颜色只是颠倒了。

从棋盘上尽可能小的棋子数量开始，从 3 到 3，每个局面最初标记为和棋。 对整个 3-3 子空间进行扫描，查找可能立即获胜的局面。 然后将这些局面标记为 “在 1 层内获胜”。然后，再次扫描子空间以查找未解析的条目，在这些条目中，所有可能的走子都会导致对手 “在 1 层内获胜”。 然后，这些局面被标记为 “2 层输棋”。该流程继续： 3 层获胜，4 层失败等。 如果通过子空间的过程未做任何其他更改，则该子空间的计算已完成。 尚未确定为胜负或输棋的任何局面均留作和棋。 中盘子空间的过程（两个玩家都有 3 个以上的角色）与该过程类似，不同之处在于该过程是通过标记已输棋的局面而开始的，而这些局面已被闷杀。 （无法在结束游戏中闷杀对手。）

完成 3-3 子空间后，可以计算 4-3 子空间。 3-3 和 4-3 无法并行计算，因为 4-3 取决于 3-3 。 但是，完成 4-3 后，5-3 和 4-4 子空间都可以并行计算，因为这两个子空间都不依赖于另一个子空间。 5-4 子空间必须等待 5-3 和 4-4 计算完成，因为这两个计算都依赖于这两个计算。 完成 9-9 子空间后，整个 中盘/残局 数据库就完成了。 图 20 中的图显示了 中盘/残局 子空间的依赖关系。

算法 1 中的伪代码显示了计算 W 白方棋子的中盘数据库子空间的过程，以及用黑方走子棋盘上的 B 黑方棋子的过程。

3-3

4-3

5-3

6-3

7-3

8-3

9-3

4-4

5-4

6-4

7-4

8-4

9-4

5-5

6-5

7-5

8-5

9-5

6-6

7-6

8-6

9-6

7-7

8-7

9-7

8-8

9-8

9-9

开始

完成

图 20： 中盘/残局 数据库的评估顺序

**算法 1 W × B 子空间的中中盘数据库计算, 轮到黑方行棋**

**for** each position, *pos*0, in the W×B subspace **do**

*pos*0*GameValue* ← DRAW

**end for** *targetGameValue* ← lose in zero ply (i.e., game is already lost) **repeat**

**for** each position, *pos*0, in the W×B subspace still marked as a DRAW **do** *worst* ← impossibly high game value **for** each possible legal move from *pos*0 **do**

*pos*1 ← *pos*0 apply move to *pos*1

**if** the move does not result in a mill that can capture **then**

*worst* ← min(*worst*, game value of *pos*1 in W×B ) **else for** each possible capture from *pos*1 **do**

*pos*2 ← *pos*1 apply capture to *pos*2

*worst* ← min(*worst*, game value of *pos*2 in (W−1)×B )

## end for end if end for

**if** *worst* = *targetGameValue* **then** *pos*0*GameValue* ← OnePlyLonger(*worst*)

## end if

**end for** *targetGameValue* ← OnePlyLonger(*targetGameValue*) **until** no changes took place in last pass through W×B subspace

## 5.5 生成开局库

尽管 “始终受保护”（Always Protected）和 “始终吃掉”（Always Capture）规则选项会影响游戏的所有阶段，但 “采用 1”（Take 1）和 “采用 2”（Take 2）规则选项仅影响开局阶段。 （仅在放置棋子而非走子棋子时，才能同时关闭两个三连。） 因此，所有四个规则组合都适用于开局：

* 始终受保护，拿走 1
* 始终受保护，拿走 2
* 始终吃掉，拿走 1
* 始终吃掉，拿走 2

开局库计算涉及到与 中盘/残局 之间的一些显著差异。 使用 残局，计算从小的 3-3 子空间开始，然后向后计算到大的 9-9 中盘子空间。 如图 21 所示，开局库计算从上一次开局的走子 ply 18 开始，该走子可能是较大的 9-8 子空间，并向后走子到游戏的开头，ply 1 始终是仅包含单个条目的小 0-0 子空间，即完全为空的棋盘。 虽然 中盘/残局 数据库总共包含 49 个子空间，但在开局库中，仅 ply 18 有 90 个子空间。 对于四个开局库中的每个数据库，总共有 615 个子空间。

层 18 层 17 层 16 层 3 层 2 层 1

90 sub-81 sub-72 sub-...4 sub-2 sub-1 subspaces 空间空间空间间距最多 9-8 至 8-8 至 8-7 最多 1-1，仅 1-0 0 0-0

图 21： 开局库的评估顺序

使用 中盘/残局 数据库，可以反转颜色，以便将 5-7 局面视为 7-5 。 对于开局库，这是不可能的，因为层规定了要走棋的颜色。 反转的颜色将与层不一致。

在中间/结束游戏中，可以在吃掉之前进行几十个走子，从而将一个子空间中的游戏带到另一个子空间中。 但是，整个开局数据库的长度仅为 18 层，每个子空间的长度仅为 1 层。 这意味着可以在单个过程中计算每个开局子空间。 例如，如果游戏处于 “层 12,6-5”局面，即使没有进行吃掉，下一个游戏也将来自 “层 13,6-6”局面 —完全不同的子空间。 此外，如果接下来的两个层层都导致吃掉，则走棋将从第 15 层 6-6 局面继续，该局面是另一个子空间。 子空间的增加使开局数据库的大小大约是中间/结束游戏数据库的两倍，但较低的深度使开局数据库的计算速度更快。

Take 2 规则的含义之一是子空间依赖于更多以前计算的子空间。 在 中盘/残局 中，子空间可以依赖之前计算的两个子空间，例如 7-6 依赖于 7-5 和 6-6 。 在 Take 2 opening 数据库中，“层 14”、“7-6”子空间取决于层 15、7-7（无吃掉）、层 15、6-7（一个吃掉）和层 15、5-7（两个吃掉）。 这意味着将更多的子空间加载到 RAM 中时，需要更多的内存。

为了与扩展强解决方案保持一致，即使棋盘上的部件数量少于实际可能的数量，也会考虑局面。 例如，对于 “层 6”，白方将已走棋 3 枚，黑方将走棋 2 枚。 在实际游戏中，如果白方已吃掉一个棋子，则棋盘可能有 3-2 或 3-1 。 出于数据库计算的目的，所有 12

子空间的计算范围为 0-0 到 3-2（0-0、1-0、2-0、3-0、0-1、1-1、2-1、3-1、0-2、1-2、2-2、3-2）。 这更易于编程，并且额外的数据库文件的大小并不重要。

算法 2 中的伪代码显示了如何生成每个开局库子空间。

**算法 2：层 P、W × B 子空间的开局数据库计算，每个局面的黑方走子× B 子空间中**

**for** each position, *pos*0, in the ply P, W×B subspace **do**

*worst* ← impossibly high game value

**for** each possible move from *pos*0 **do**

*pos*1 ← *pos*0

apply move to *pos*1

**if** the move does not result in a mill that can capture **then**

*worst* ← min(*worst*, game value of *pos*1 in P+1,W×(B+1) ) **else for** each possible capture from *pos*1 **do** *pos*2 ← *pos*1 apply capture to *pos*2

**if** rules or position dictate that only 1 capture is possible **then** *worst* ← min(*worst*, game value of *pos*2 in P+1,(W−1)×(B+1) ) **else for** each possible 2nd capture from *pos*2 **do** *pos*3 ← *pos*2 apply 2nd capture to *pos*3

*worst* ← min(*worst*, game value of *pos*3 in P+1,(W−2)×(B+1) )

## end for end if end for end if

**end for** *pos*0*GameValue* ← OnePlyLonger(*worst*)

## end for

# 6. 数据库验证

每个数据库使用不同的编译器生成两次，以避免编译器错误和硬件问题（如内存或硬盘错误）。 创建了每个文件的校验和，以便在传输到其他局面时验证完整性。 这种双重检查基本上可以消除硬件和编译器错误的可能性，但当然不会检测程序员错误。

已成功使用验证程序，但由于同一人编写了原始代码，因此无法将其视为独立验证。 要真正验证数据库，需要从外部源进行确认。 幸运的是，G?abor G?evay 和 G?abor Danner 以及 Peter Stahlhacke 同意帮助验证与其对应的特定数据库。 根据 Jonathan Schaeffer 和 Ed Gilbert 的例子，通过比较每个子空间的 win/lose/draw 频率（Schaeffer，2009，p. 478）（Gilbert，2005）。 尽管很可能会使程序出现偏移错误，从而使成功/失败/和棋记录相同。

为防止这种不可能意外匹配的可能性，通过比较层的频率分布与游戏结束的频率分布，进一步进行了验证。 例如，使用 4-4 始终吃掉中盘数据库，而不仅仅是说 win/lose/draw 频率为 159/29/3,225,409，所有 “层到端”频率的比较如表 2 所示。 请注意，wins（奇数层）的频率增加到 159，而输棋增加到 29 。 使用此附加信息，意外匹配的可能性实际上为零。 对于所有子空间，应将两个独立生成的数据库（其层频率都匹配）视为已验证（适用于所有实际用途）。

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **终局层：** | 和棋 | 已输棋 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| **频率:** | 3,225,409 | 6 | 29 | 3 | 24 | 10 | 46 | 6 | 48 | 4 | 12 |

表 2： 4-4 始终吃掉端点的层频率分布

来自 G?evay 和 Danner ’s 的 Capture 频率始终吃掉，采用 1 个开局库，并且始终吃掉与 Stahlhacke 的 中盘/残局 数据库完全匹配的中间/残局 数据库，与 Take 的始终保护 中盘/opening 数据库一样。 Stahlhacke 的格式始终受到保护，采用 2 个开局库，使其不允许轻松提取层频。 对于该开局库，基本验证是通过采用许多随机选择的局面以及一些手动选择的 “易出错”局面来完成的，并将游戏值与 Stahlhacke 的商业软件 Mu?hle Datenbank Advanced（http://muehle-24. de/）报告的值进行比较。 检测到的每个条目都是完全匹配的。 与商业软件 Brillant Mill（http://www.brillant-mill.eu/）的结果相比，始终吃掉的选定局面采用了 2 个开局数据库，并且发现它们也匹配。

在外部未验证的数据库的唯一部分是始终受保护的数据库，该数据库的开局局面为 1，此时不存在此数据库的其他已知实例。 此数据库的生成器几乎与始终受保护、占用 2 和始终吃掉、占用 1 个代码共享它的所有代码，因此可能会产生类似的结果。

# 7. 结果： 规则变化的影响

开局库证明游戏是规则变体的所有四个组合的绘图。 许多相关问题自然如下：

* 这些变化对游戏的每个阶段有多少影响 ？
* 游戏中的差异对游戏有多早的影响 ？
* 游戏中的差异能在多长时间内改变最终结果 ？

## 7.1 对中端游戏的影响

在实际游戏中，“始终受保护”和 “始终吃掉”规则之间的游戏差异通常并不明显，除非结束游戏中的一个或两个玩家都小于三枚（Schu?rmann & Nu?scheler，1980，p. 44）。 但是，在完美的 play 数据库中，规则差异的影响在整个游戏中都可见。 （同时形成三连只能在开场游戏中形成，因此 Take 1 和 Take 2 规则对中间/结束游戏没有影响。）

在 残局 中，规则中的差异似乎不会对更均匀游戏（3 对 3 到 6）的结果产生很大的影响，但它确实会对更不一致的局面（3 对 7、8 或 9）产生明显的差异。 （详见附录 B 。） 图 22 显示了一个罕见的 7-3 示例，其中规则确实有所差异。

1

2

3

4

5

6

7

abcdefg defg

图 22： 白方走子和和棋（始终受保护），或 17 分获胜（始终吃掉）

在 3-3 子空间中，没有一个事例，在这种情况下，成功/失败/和棋结果是不同的，但在这种情况下，结果更容易实现，如图 23 所示。

1

2

3

4

5

6

7

abcdefg defg

图 23： 在 23（始终受保护）或 1（始终吃掉）中实现走子和成功的白方

在使用任一侧走棋的 4-4 子空间和使用黑方走棋的 4-3 中，两个数据库之间根本没有差异。 在使用白方走棋的 4-3 子空间中，只有 760,398 个局面中的 748 个局面会立即成功，而始终吃掉则始终受到保护。

在许多情况下，从 “始终保护”更改为 “始终吃掉”会将结果从 “和棋”更改为 “成功”或 “失败”。 但是，中间/结束游戏中很少出现结果颠倒的情况，其中输棋变为成功，反之亦然。 事实上，如果两个玩家的棋盘上都少于 6 件，就没有这样的情况了。 在转换结果的扩展解决方案中间/终端游戏数据库中，总共只有 3533 个案例，其中绝大多数是在实际游戏中无法达到的局面。 其中，远程甚至只能有 433 个这样的局面。 图 24 显示了扩展解决方案数据库中的一个示例，在该示例中，白方 wins 始终受保护，但在 Always Capture 中输棋。

1

2

3

4

5

6

7

abcdefg defg

图 24： 在 23（始终受保护）和 30（始终吃掉）中走子并赢得胜利的白方

表 3 显示了从 “始终受保护”规则到 “始终吃掉”规则的更改如何影响中间/结束游戏数据库中所有局面的结果的摘要。

|  |  |
| --- | --- |
| **局面合计** | 8,904,594,601 |
| **输棋和棋** | 29145,537 |
| **赢得胜利** | 34,254,690 |
| **和棋输棋** | 350087 |
| **取胜** | 450042 |
| **得胜输棋** | 2717 |
| **得失** | 816 |

表 3： 残局 结果差异从 “始终保护”更改为 “始终吃掉”。 显示的结果与当前玩家有关。

游戏理论家的好奇心之一是游戏结束时的最大层数。 对于 “始终受保护”（Always Protected）规则，“中间/结束”（中盘/残局）的最大层数为 329，对于 “始终吃掉”（Always Capture）规则，最大层数为 204 。 （有关每个子空间最大层数的详细信息，请参见附录 C 。）

这些庞大的层数远远超过了人类的能力。 对玩家更感兴趣的是，通过使用一个规则或另一个规则，游戏变得更容易获胜的频率。 大约 0.3%的中间/结束游戏局面具有不会改变结果但相差超过 30 层的游戏值。 其中，使用 “始终受保护”（Always Protected）和 “2/3 且始终吃掉”（2/3 with Always Capture），大约 1/3 更容易获胜。

## 7.2 对开场游戏的影响

开局式游戏受 “始终受保护与始终吃掉”规则选择以及 “Take 1 与 Take 2”规则选择的影响。 此调查的目标之一是确定这些规则在游戏中的影响程度。

### 7.2.1 始终受保护与始终吃掉规则

“始终受保护”（Always Protected）和 “始终吃掉”（Always Capture）规则选项通常被认为主要影响 残局，因此，在黑方第二次走子之前，此规则可能会影响理论游戏的价值，这一点有些令人惊讶。 通过第 5 层，有 13 个局面从 “始终保护”更改为 “始终吃掉”，将 2 个局面更改为 “和棋”或 “和棋”。 这些示例如图 25 所示。 在较晚的开口层，这种情况会更频繁地发生，但相对较少的情况仍然存在。 直到 14 年，在扩展解决方案中，结果完全从输棋转变为成功，反之亦然。

77

a

b

c

d

e

f

g

66

55

44

33

22

11 abcdefg

图 25： 左图：第 4 层，黑方在 79（始终受保护）或 55（始终吃掉）中走子和获胜。 右：第 5 层，走子和和棋的白方（始终受保护），或在 67（始终吃掉）中获胜。

### 7.2.2 采用 1 个规则与采用 2 个规则

“Take 1”（Take 1）与 “The 2”（are 2）规则选项确定当两个铣削同时关闭时要吃掉的数量。 同时出现形成三连的最早时间是第 9 层（白方的第 5 次走子）。 令人惊讶的是，此规则的影响可能会对游戏产生较早的影响，而第 5 层可能已经改变了结果。 图 26 所示局面为采用 Take 1 规则的绘图，但采用 Take 2 规则的绘图获胜。

1

2

3

4

5

6

7

abcdefg defg

图 26： 第 5 层，白方走子和和棋（采用 1）或获胜（采用 2）

这使得图 26 特别有趣的是，如果两个玩家从这一阶段开始都能很好地行棋，那么就不会真正形成同时的形成三连，甚至需要直接被黑方闷杀。 只有在路上同时出现形成三连威胁的情况下，才足以造成黑方的情况，从而完全避免这种情况。 如果黑方未考虑 “采用 2”规则，则白方最终会在第 17 层同时形成形成三连。

这与经验丰富的参与者一致。 WMD 总裁 Daniel Lehner 说，当被问及比赛比赛中同时出现形成三连时，“我个人从未见过一场比赛，一位运动员能够同时关闭两台形成三连。 这将是一个巨大的优势，对手将始终试图闷杀它 ”（Lehner，2016 年）。

### 7.2.3 共同制定的规则组合

查看各种规则对开局的总体影响的一种方法是检查 “层 18”的统计信息，即开局的最后一次走子，以查看规则是否更改了和棋次数。 表 4 显示 “始终吃掉”稍微减少了绘图数量，采用了 2 条规则。

|  |  |  |  |
| --- | --- | --- | --- |
|  | **取胜白方** | **以黑方取胜** | **和棋** |
| **始终受保护，拿掉 1 个** | 17.34 | 62.53 | 20.13 |
| **始终受保护，拿掉 2 个** | 17.28 | 62.67 | 20.05 |
| **始终吃掉，拿掉 1** | 17.57 | 62.88 | 19.56 |
| **始终吃掉，拿掉 2** | 17.50 | 63.02 | 19.48 |

表 4： 在第 18 层赢得百分比

请注意，表 4 中显示的百分比是 ply 18 数据库子空间中因白方或黑方而获胜的条目的百分比。 这并不意味着，在两个黑方玩家之间的游戏中，获胜的概率为 63%（除非玩家在开场白中随机走棋，然后在中间/结束游戏中完全走棋）。 同时，这确实使专家相信黑方有一点优势。 虽然白方可以首先攻击，但黑方具有在开局阶段放置最后一枚棋子东西的优势（Schu?rmann & Nu?scheler，1980，p. 。

图 27 显示了从四个规则组合中的一个规则到另一个规则组合的更改如何影响开局数据库中所有局面的结果的摘要。 正如可以看到的那样，始终受保护与始终吃掉规则选择的影响比 Take 1 与 Take 2 规则选择的影响更大，从而减少了和棋次数，并且对反转成功/输棋结果的影响较小。

加瑟

WMD（先前）

WMD（当前）

**图例**

输棋提款

赢得胜利

提款输棋

取胜

得胜输棋

得失

吃掉

Take1

吃掉

Take2

受保护

Take1

受保护

Take2

10,344,735

49,443,990

17,815,119

10,444,080

3,579,118

294239

43,235,866

55576,554

1522,569

4,564,919

5194

4,294

10,546,982

49,324,170

1829919

10,666,789

748,014

305,878

42,806,161

55,534,331

1,523,401

4708,241

5398

4225

52,998,682

104,265,753

19126,009

14,500,050

3,653,281

306,932

60426,250

64,751,294

11,340,127

526642,011

30383

3,683,920

图 27： 由于从一个规则更改为另一个规则而导致的结果差异，在 17,874,891,168 个开局库条目中。 显示的结果与当前玩家有关。

## 结论

总体结论是，四个规则组合中的任何一个都不影响初始理论游戏值 — —该游戏始终是一种方法。

从 “始终保护”规则更改为 “始终吃掉”规则所产生的整体影响具有预期的效果，即稍微减少绘图数，而不会产生明显逆转游戏结果的不利影响。

使用 Take 2 规则而不是 Take 1 规则对游戏的影响微乎其微，但稍微进一步减少了和棋次数。

# 确认

我希望感谢 Rosslyn Academy IT 部门的 Alan Davis 使用其规格使得此项目更易于管理的备用服务器。

我还想感谢 G?abor G?evay、G?abor Danner 和 Peter Stahlhacke 在独立验证数据库的过程中的大量合作。

另外，我们还向 G 'abor G' evay 和 G 'abor Danner 致谢了有关本白皮书的宝贵建议。

# 附录 A 硬件和软件

残局 数据库于 2014-15 年在采用双英特尔至强 E5620 四核 2.40 GHz 处理器（共 8 个物理内核）、运行 Windows server 2012 的 96 GB RAM 的服务器上生成。 这些规范允许整个子空间以及子空间依赖关系完全加载到 RAM 中，而无需数据压缩。 例如，在计算 9-8 子空间时，9-7 和 8-8 子空间以及哈希表也加载到内存中，因此此程序实例需要大约 15 GB 的空间。 8 个内核和 96 GB RAM 仍然允许同时计算多个子空间。

出于确认目的，始终受保护数据库和始终吃掉结束游戏数据库均使用两个不同的 C 编译器 Microsoft C 18.0 和 GCC（MinGW）4.8.2 进行了两次计算。 残局 数据库及其确认的整个运行时间大约需要两个半月。

2015 年，开局游戏数据库是在英特尔 i7-3770 四核 3.40 GHz 处理器（4 个物理内核）上生成的，并使用 Microsoft C 18.0 编译器运行 Windows 10 的 24 GB RAM 上生成的。 由于开局库生成器可能需要多达 18 GB 的 RAM，因此无法并行运行多个实例，但所有四个规则组合的数据库在大约 10 天内生成。

为进行确认，所有四个开局库再次在同一台物理机上重新计算，但使用 GCC 4.8.2 C 编译器在 Virtualbox 下的 Linux Mint 17 上运行。

# 附录 B. 赢得/输棋/和棋数据

表 5 和表 6 显示了 “始终受保护”（Always Protected）和 “始终吃掉”（Always Capture）规则的成功/失败/和棋百分比的摘要。 该表假定它的走子方向为白方。 棋盘上白方部分的数量显示在左列下方，黑方部分的数量显示在顶行上。 符号 “0 +”用于表示舍入为 0.0%的小非零值。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **W\B** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **3** | 82.9/16.9/0.2 | 13.5/0/86.5 | 0.2/0.4/99.4 | 0/2.4/97.6 | 0/39.2/60.8 | 0/76.6/23.4 | " |
| **4** | 9.8/0.4/89.8 | 0 +/0 +/100.0 | 0 +/0 +/100.0 | 0 +/4.4/95.6 | 0 +/71.1/28.9 | 0 +/91.1/8.9 | 0 +/99.5/0.5 |
| **5** | 22.3/0/77.7 | 0.1/0 +/99.9 | 0.1/0 +/99.9 | 0 +/4.4/95.5 | 0 +/59.7/40.3 | 0 +/86.8/13.2 | 0 +/98.1/1.9 |
| **6** | 39.8/0/60.2 | 16.9/0 +/83.1 | 17.0/0 +/83.0 | 10.9/2.4/86.7 | 5.5/36.6/57.9 | 1.9/69.8/28.3 | 0.5/93.2/6.3 |
| **7** | 88.1/0/11.9 | 91.3/0 +/8.7 | 87.4/0 +/12.6 | 73.2/1.0/25.8 | 46.6/14.8/38.6 | 21.1/41.7/37.2 | 7.8/76.0/16.2 |
| **8** | 98.4/0/1.6 | 98.7/0 +/1.3 | 97.9/0 +/2.1 | 93.4/0.3/6.4 | 79.7/4.3/16.1 | 53.6/18.0/28.4 | 28.3/46.9/24.9 |
| **9** | 100.0/0/0 + | 100.0/0 +/0 + | 99.9/0 +/0.1 | 99.2/0.1/0.7 | 95.4/1.1/3.5 | 82.5/6.6/10.9 | 58.6/22.3/19.1 |

表 5： 始终受保护的中间/结束游戏的舍入获胜/失败/和棋百分比

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **W\B** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **3** | 82.9/16.9/0.2 | 13.5/0/86.5 | 0.2/0.4/99.4 | 0/2.8/97.2 | 0/47.3/52.7 | 0/90.8/9.2 | 0/100.0/0 + |
| **4** | 9.9/0.4/89.7 | 0 +/0 +/100.0 | 0 +/0 +/100.0 | 0 +/6.4/93.6 | 0 +/73.0/27.0 | 0 +/93.0/7.0 | 0 +/99.9/0.1 |
| **5** | 22.5/0/77.5 | 0.1/0 +/99.9 | 0.1/0 +/99.9 | 0 +/6.7/93.3 | 0 +/60.4/39.6 | 0 +/87.6/12.4 | 0 +/98.5/1.5 |
| **6** | 40.0/0/60.0 | 23.2/0 +/76.8 | 23.9/0 +/76.1 | 14.9/3.4/81.7 | 7.1/37.0/55.9 | 2.4/70.4/27.2 | 0.7/93.5/5.8 |
| **7** | 91.1/0/8.9 | 92.3/0 +/7.7 | 87.9/0 +/12.1 | 73.7/1.2/25.1 | 47.0/15.0/38.0 | 21.3/42.1/36.6 | 7.9/76.3/15.8 |
| **8** | 99.7/0/0.3 | 99.2/0 +/0.8 | 98.1/0 +/1.9 | 93.7/0.3/6.0 | 80.0/4.3/15.7 | 53.9/18.2/28.0 | 28.4/47.1/24.5 |
| **9** | 100.0/0/0 + | 100.0/0 +/0 + | 99.9/0 +/0.1 | 99.3/0.1/0.7 | 95.5/1.1/3.4 | 82.7/6.6/10.7 | 58.8/22.4/18.8 |

表 6： 始终吃掉中间/结束游戏的舍入获胜/失败/和棋百分比

# 附录 C 游戏结束时的最大层数

表 7 和表 8 显示了 “始终保护”和 “始终吃掉”规则中每个中间/结束游戏子空间的最大层数。 棋盘上白方部分的数量显示在左列下方，黑方部分的数量显示在顶行上。

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **W\B** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **3** | 26 | 33 | 31 | 6 | 38 | 40 | 40 |
| **4** | 32 | 9 | 28 | 300 | 300 | 214 | 142 |
| **5** | 3 | 29 | 57 | 312 | 308 | 302 | 232 |
| **6** | 7 | 301 | 311 | 321 | **329** | 323 | 321 |
| **7** | 39 | 299 | 303 | 328 | 318 | 321 | 321 |
| **8** | 41 | 213 | 285 | 322 | 320 | 315 | 320 |
| **9** | 39 | 95 | 163 | 318 | 312 | 319 | 319 |

最大值 = 329

表 7： 中盘/残局 始终保护游戏结束时的最大层数

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **W\B** | **3** | **4** | **5** | **6** | **7** | **8** | **9** |
| **3** | 26 | 33 | 31 | 6 | 30 | 34 | 30 |
| **4** | 32 | 9 | 28 | 156 | 112 | 112 | 110 |
| **5** | 3 | 29 | 57 | 162 | 160 | 160 | 114 |
| **6** | 7 | 157 | 163 | 167 | 184 | 186 | 173 |
| **7** | 31 | 111 | 159 | 185 | 181 | **204** | 202 |
| **8** | 33 | 111 | 153 | 185 | 203 | 196 | 202 |
| **9** | 25 | 103 | 113 | 172 | 201 | 201 | 191 |

最大值 = 204

表 8： 中盘/残局 始终吃掉游戏结束时的最大层数

图 28 显示了中盘局面，在游戏结束之前，在始终受保护规则和始终吃掉规则中，这些局面具有最大数量的层。

77

66

55

44

33

22

11

abcdefgabcdefg defgabcdefg

图 28： 游戏结束时的最大层数。 左（Left）: 在 329 年（始终受保护）中走子并赢得胜利。 右（Right）: 将在 204 年（始终吃掉）走子和输棋为黑方。

表 9 显示了开始数据库中每个层编号在游戏结束时的最大层数。 这些值用于扩展强解决方案，因此它们包括在实际走棋中无法达到的局面。

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **层** | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | **最大值** |
| **Prot-1** | - | 35 | 38 | 125 | 245 | 263 | 283 | 297 | 330 | 329 | **331** | 330 | 329 | **331** | 330 | 329 | 328 | 329 | **331** |
| **Prot-2** | - | 35 | 38 | 125 | 245 | 263 | 283 | 297 | 330 | 329 | **331** | 330 | 329 | **331** | 330 | 329 | 328 | 329 | **331** |
| **Capt-1** | - | 35 | 37 | 125 | 124 | 179 | 187 | 186 | 188 | 197 | 201 | 204 | 203 | 202 | **206** | 205 | 204 | 204 | **206** |
| **Capt-2** | - | 35 | 37 | 125 | 124 | 179 | 187 | 186 | 188 | 197 | 201 | **204** | 203 | 202 | 201 | 203 | **204** | **204** | **204** |

表 9： 在游戏结束时开局游戏的最大层

# 参考

阿利斯湖 V. （1994 年）。 *在游戏和人工智能中搜索解决方案。* Ph.D.林堡大学计算机科学系论文

Brillant 三连团队（2016 年）。私人通信..

卡泽纳夫角 诺瓦科夫斯基河 （2015 年）。 Woodpush 的逆行分析。 *没有机会的游戏，63,47 – 56 。*

埃德尔坎普山 苏莱夫斯基角 & Yu?cel, C. （2010）。 双玩家游戏的 GPU 探索，具有完美的哈希函数。 为召开第三届年度组合搜索研讨会（SOCS-10）。

加瑟河 （1990 年）。 将逆行分析应用于九个男士莫里斯。

加瑟河 （1993 年）。 莫里斯九子棋是个和棋活动。录制游戏摘要。

加瑟河 （1995 年）。 *利用计算资源进行高效的全面搜索。* Ph.D.

博士论文，瑞士联邦理工学院 Zu?rich.

加瑟河 （1996 年）。 解决 9 个男士莫里斯的问题。 *没有机会的游戏，29，101 – 113 。*

格韦湾 E.、 丹纳湾 （2014 年）。 计算九人 Morris、Morabaraba 和 Lasker 的超强和扩展解决方案。 *CoRR（在计算智能和游戏中的 AI 的 IEEE 交易中出现），abs/1408.0032 。*

吉尔伯特河 （2005）。 Kingsrow 。http://edgilbert.org/Checkers/KingsRow.htm 。

莱克河 Schaeffer, J., & Lu, P. （1994 年）。 使用工作站网络解决大量的逆行分析问题。

莱纳角 （2016 年）。私人通信..

林克角 （1994 年）。 以莫里斯九子棋为例，完美地玩耍。 ETH Zu?rich 计算机科学系硕士论文

舍费尔湾 （2009）。 *向前一个跳转： 方格图案（第 2 版）上的电脑完美。* 斯普林格

Schaeffer, J., 莱克河（& Lake） （1996 年）。 解决棋盘格游戏的问题。 *不可能玩的游戏，29,119 – 133 。*

舒尔曼角 Nu?scheler, M. （1980 年）。 *所以，Gewinnt Man Mu?hle 。* 拉文斯布格尔

斯塔尔哈克山口 （2003）。 拉斯克 ·莫里斯的比赛

斯塔尔哈克山口 （2016 年）。私人通信..

Weltmu?hlespiel Dachverband（2008 年）。 Das Mu?hlespiel（Morris 游戏）。http://www.muehlespiel.eu/images/pdf/WMD Spielregeln.pdf 。

Weltmu?hlespielDachverband（2010 年）。 Turnierreglement（比赛条例）。

http://www.muehlespiel.eu/images/pdf/WMD Turnierreglement.pdf 。

[查看发布统计信息](https://www.researchgate.net/publication/305413042)