上海财经大学

毕业论文

题目：基于人工策略选择的教学优化求解算法

姓　　名 齐梓宾

学　　号 2017110800

学　　院 信息管理与工程学院

专　　业 计算机科学与技术

指导教师 郭子超

定稿日期　2021 年 4 月

基于人工策略选择的数独优化求解算法

摘 要

数独是一种数字推理游戏，玩家需要在给出部分数字的 9×9 数独初盘上，依据其限制规则还原出完整的数独终盘。程序求解数独可以使用线性规划法和回溯法等方式进行，回溯法编写方便，求解速度快，因此使用最为普遍。融合了人工策略的回溯法有着优异的求解速度，但在人工策略的选择与执行方面仍具有一定的优化的空间。本文研究发现，在 9×9 规模的数独中，人工策略的最佳选取为排除法、唯一余数法、交叉排除法和隐式数对，其中排除法和唯一余数法只在删数操作时进行触发检验，而其他策略在每层回溯中需要进行全盘搜索，回溯法优先选取候选数最少的单元进行填写，这种策略组合可以显著加快求解速度。在 16×16 的高维数独中，还可加入显式数对和隐式三数组策略进行优化，在每层回溯中仅执行一遍策略较循环执行更优。

关键词：数独，人工策略，回溯法，求解效率

Optimized Sudoku algorithm based on artificial strategy selection

Abstract

Sudoku is a numeric logic puzzle. Given an initial board with partially filled numbers, players should restore the whole board according to the game rules. Sudoku can be solved by programs using methods including linear programming and backtracking. Backtracking method is easy to implement and efficient, and so it is most commonly used. The backtracking method combined with artificial strategies has excellent runtime, but there are still rooms for improvement in the selection and implementation of artificial strategies. In this paper, we find that in 9×9 Sudoku puzzles, the best collection of artificial strategies is hidden single, naked single, intersection and hidden pair. The hidden single and naked single tests are only triggered during the removal operation, while for the other strategies, we need to search the whole board in each level of backtracking. In backtracking, the unit with the least number of candidates should be filled in first. This combination can significantly speed up the solving process. In the 16×16 high-dimensional Sudoku, the naked pair and hidden triplet strategy can be added to optimize, and it is better to execute these strategies only once in each level of backtracking than executing them in a loop.

Key words: Sudoku, artificial strategy, backtracking, solution efficiency

目 录

一、序言 ……1
二、文献综述 ……2
三、研究设计 ……3
（一）人工策略算法的选择和执行方式 ……4
（二）回溯法的优化 ……11
（三）其他优化方式 ……11
（四）数据结构与算法实现 ……11
四、实验结果与分析 ……14
（一）回溯策略的比较 ……15
（二）人工策略的比较和执行方式的选择 ……16
（三）高维度上的表现 ……25
五、结论 ……29
致谢 ……30
参考文献 ……31

一、 序言

数独（Sudoku）是一种数字逻辑游戏，它起源于 18 世纪瑞士数学家欧拉等人所研究的拉丁方阵。美国退休建筑师 Howard Garn 将其改编为数独的形式，此后数独相继流传至日本和英国。我国在 90 年代就有部分书籍刊登此类问题，2007 年 2 月我国正式引入数独，此后得到了良好的发展。

传统标准数独为包含 81 个方格的 9×9 正方形阵列，其中水平方向的每行包含九格，称为行（Row）；垂直方向的每列包含九格，称为列（Column）；按顺序每连续三行与连续三列相交的方阵包含九格，称为宫（Box）。其中独立单元格称为单元（Cell），行、列、宫称为区域（Region）。三个连续宫可以组成大行列（Chute），称为大行（Floor）和大列（Tower）。在九宫格的一些单元中存在已知数字，作为推断的线索，称为提示数（Clue）。数独的解题规则为：将数字 1~9 填入空白单元格中，使之满足 1）每个单元仅包含一位数字；2）每行包含数字 1~9 各一个；3）每列包含数字 1~9 各一个；4）每宫包含数字 1~9 各一个。正规的标准数独只有唯一解，而非正规数独存在多个解答的可能，在本文中仅研究正规数独的解答。

一个数独问题的初始状态被称为初盘， 完成状态被称为终盘（图 1-1）。数独初盘至少需要含有 17 个提示数，以保证解的唯一性（Gary McGuire et al.，2014）。数独的终盘共有 6，670，903，752，021，072，936，960（约 6.67×10^{21}）种组合， 除去可能的重复形式（如数字交换、行列交换、对称等），则存在 5，472，730，538（约 54 亿）个组合。在数独中一个空白单元可以填写的数字集合称为候选数集合，大多数的数独技巧是围绕候选数的选择与删减进行的。数独的人工解题技巧丰富，包括隐式候选数方法类（Hidden Single/Pair/Triplet/Quad），显示候选数方法类（Naked Single/Pair/Triplet/Quad），交叉排除法（Intersection），鱼方法类（X-Wing/Swordfish/Jellyfish），致命结构方法类（Deadly Pattern）和强弱链环方法类（Chains）等。数独也有多种变种题型，包括对角线数独、窗口数独、颜色数独、锯齿数独以及降维的 4×4 数独和升维的 16×16 数独等。

目前关于数独解题算法的研究包括线性规划类，人工策略类和回溯类，其中融合人工策略的回溯法求解速度最快。本文旨在在此基础上继续对人工策略的细节进行优化，尝试包含更加复杂的人工策略，以进一步优化求解速度；另外，对于人工策略的执行方式，以及回溯过程中的单元格选择和候选数选择方式，本文也进行了多种实验方案的比较与改进。

本文的主要贡献体现在：第一，实验得出了在不同规模数独中，应用的人工策略难度平衡点，即策略检索开销与效率提升的瓶颈位置。第二，本文使用的嵌入方式执行排除法和唯一余数法进一步加速了数独求解效率。第三，本文在 16×16 的高维数独上重复进行了相同实验，有助于理解高维数独的性质，填补了高维数独此方面研究的空白。经过全部优化结合后的最终算法可将搜集到全部数据集内 9×9 数独的最大求解时间降低至 3ms 以下，对于更高维度的数独也有着稳健的求解速度和普适性。

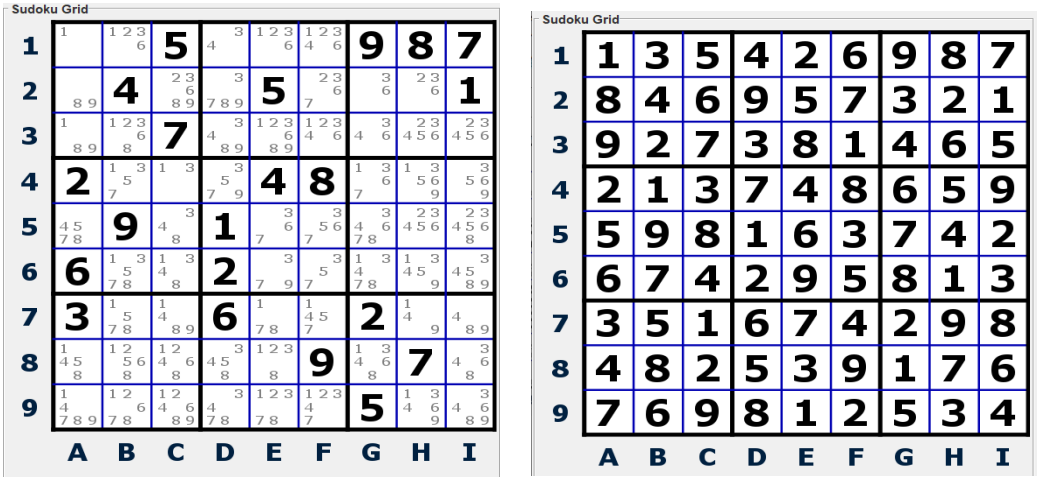

图 1-1 初盘与终盘

（图片来源：Sudoku Explainer 1.2.9.0）

二、 文献综述

数独除了作为一种益智类游戏启发人的思维外，在图像加密，关联存储等实际应用方面的发展也具有相当大的潜力（胡伟通等，2015；Thai-Son Nguyen et al.，2015；Jiann-Ming Wu et al.，2014）。因此，对数独的性质、生成以及求解方向的研究将具有重要的意义。数独的求解是数独生成和难度计算的基础，对于数独求解算法的研究方向大致分为三类：线性规划类，人工策略类和回溯类。线性规划类方法将数独难题转化为线性规划的数学形式，再进行求解。但传统整数规划解题速度较慢，经过稀疏优化提速后却无法保证求解成功（张煜东等，2011）。人工策略类方法是适合人类解题者的一系列方法，以竞赛为目的培养数独参赛者，此方面的书籍有《数独大师》等（西尾彻也，2017）。但由于人类策略多较为灵活，考虑到程序对策略触发条件的检索开销，程序应用较为复杂的人工策略往往得不偿失。回溯类方法是目前程序解题研究最多的方向，回溯法不但能保证求解成功，还有着较高的解题速度（Eric C. Chi et al.，2013）。数独问题可以转化为精确覆盖问题，因此也是 NPC 问题。Knuth 发明的舞蹈链数据结构可以高效的解决数独与精确覆盖问题，其本质与回溯法无异。

对于回溯法的优化关键在于如何剪枝，减少在不必要情景下搜索的时间浪费。目前回溯法的剪枝存在三种思路：1）识别并删除空白单元候选数集内的错误候选数，以横向减少枚举的可能性；2）在求解过程中寻找逻辑错误，提前结束错误的分支，通过纵向降低分支的深度提高搜索速度。3）调整填写单元或枚举候选数的顺序，以增加快速找到最终结果的概率。使用方法一的文献主要基于数独的基本规则，同时使用简单的人工策略对候选数进行删减，降低候选数集大小（程曦等，2011）；方法二需要在回溯的每个阶段对单元格内的侯选数集进行检查，出现无数可填或其他冲突状况时停止继续填写，及时回退（江顺亮等，2020）；也有文献利用方法三，如信息熵等方式，首先尝试填写信息增益最大的某单元或候选数，以增加提前找到最终解的概率，也获得了不错的成果（Zhai Gaoshou et al.，2013）。

在已知文献中，效率最高的求解方案为江顺亮等人所研究的融合人工策略的回溯法。这种方法借鉴 Stuart 发现的数独特点，即大部分数独仅需要两到三次不得已的猜数操作，就可以用简单的人工策略求解成功。江顺亮等人的融合人工策略的回溯法，在回溯时选取候选数最少的单元格进行填写，并在回溯的每一层循环使用人工策略推进，直到人工策略无法继续推进时再执行下一次回溯进行猜数操作，在人工策略求解失败时及时进行回退操作，同时

优化了枚举深度和广度，融合了剪枝的全部三种思路。这种方法将数独的平均求解时间降低至 2ms 以下，只有少数较难的数独求解时间超过 10ms。同时该文献也与其它多种方法进行了对比，并分析了适合测试的数据集，其中 Su17 数据集（Royle 收集的包含 17 个提示数的数独 49151 个）在各方面都具有良好的特性。

三、 研究设计

参考江顺亮等人（2020）的融合人工策略的回溯法流程，本文设计算法框架如下图所示：

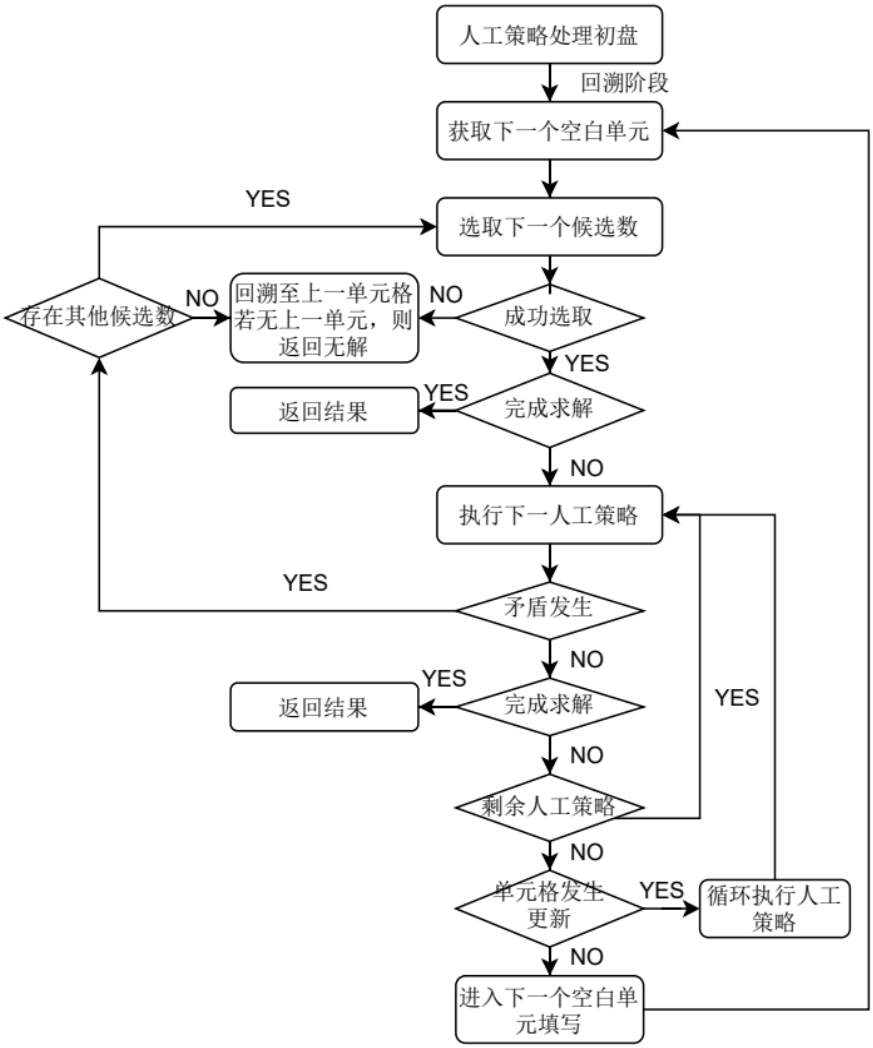

图 3-1 算法框架流程图

首先使用人工策略对数独初盘进行求解。当求解无法继续进行时，使用回溯法挑选空白单元格和候选数进行猜测填写。猜测填写结束后继续使用人工策略进行推进，如此猜测填写与人工策略反复进行，直至得到最终结果。

使用人工策略填写的优势是，可以确定其操作均符合规则，也就是人工策略填写数字的行为都是准确的（猜测过程中填写的数字会出现错误）。同时，在检测人工策略触发条件的过程中，也可以及时发现数独中出现的逻辑错误，以便及时返回上一次猜测进行答案调整。

而人工策略的劣势则是需要进行全盘查找满足触发条件的情景，若没有满足条件的情景，搜索时间就会被浪费。因此，人工策略的使用和选择就要同时权衡其搜索时间、剪枝效率和剪枝可能性。

猜测过程虽然准确率较低，但倚仗计算机的快速计算能力，可以通过遍历所有可能性找到正确解。在剩余候选数较多时对候选数组合进行遍历显然会浪费时间，因此在人工策略求解遭遇瓶颈时，使用回溯策略推进是最佳时机，对猜测单元的选择以及尝试候选数顺序的调整，也会通过改变求解路径使求解的时间发生变化。

综上所述，我们将从以下几个角度对求解策略的优化进行实验：1）人工策略的选择和执行方式；2）回溯过程中单元和候选数的选择顺序；3）其他优化方式

（一） 人工策略算法的选择和执行方式

考虑到某些复杂的人工策略具有较大的搜索开销，但仅能更新较少单元格内的少量候选数。我们在策略的选择上只选择搜索方式相对简单，而更新效果较为显著的人工策略。其中实验涉及到的策略类型如下：

1. 显式候选数方法类

显式候选数方法根据其涉及的单元格数量可分为唯一余数法、显式数对、显示三数组、显示四数组等方法。

唯一余数法：单元格中仅剩余一个备选候选数可以使用，则这个单元格只能填入此候选数。填数结束后可将与此单元处于同一区域（行、列、宫）的其他单元格中的此候选数删除。

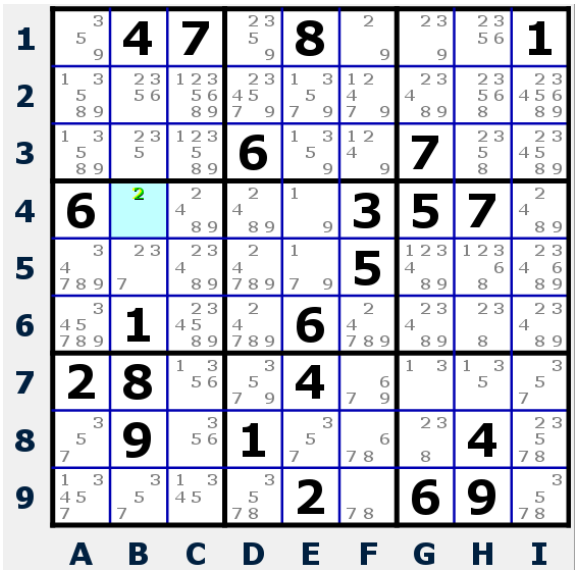

图 3-2 唯一余数法

（图片来源：Sudoku Explainer 1.2.9.0）

显式数对：为唯一余数法的进阶，涉及同一区域（行、列、宫）的两个单元格。如果同一区域的两个单元格，其候选数集合的并集恰好只包含两个数字，则意味着这两个数字仅能出现在这两个单元格中，因此可将此区域内其他单元侯选数集中出现的这两个数删除。如图 3-3 所示，在第一宫中 A2，A3 单元中仅含有 8，9 两个候选数，8 和 9 最终一定填写在这两个单元之中，因此可删除第一宫中其他位置出现的 8，9 候选数（B3 和 C2 单元）。

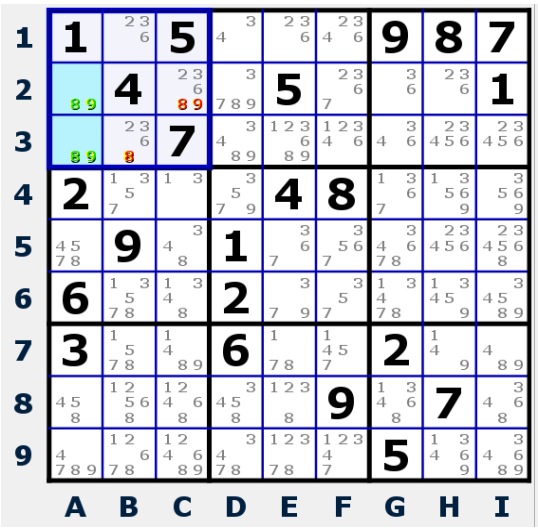

图 3-3　显式数对

（图片来源：Sudoku Explainer 1.2.9.0）

显式三数组和显式四数组的定义与显式数对相似，仅增加了涉及单元格和数字的数量，示例见图 3-4。

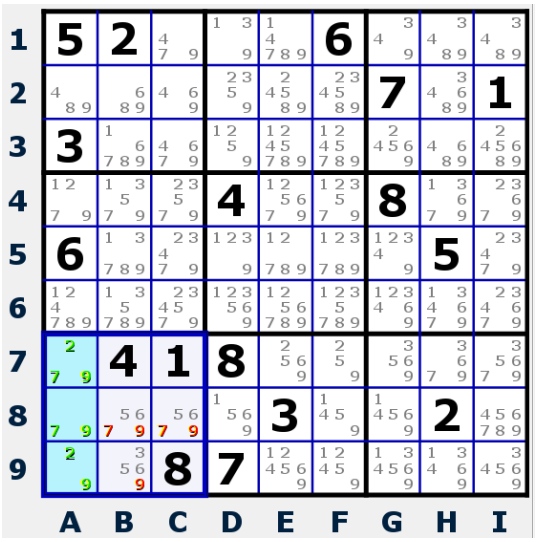

图 3-4 显式三数组

（图片来源：Sudoku Explainer 1.2.9.0）

2. 隐式候选数方法类

隐式候选数方法根据其涉及的单元格数量可分为排除法、隐式数对、隐式三数组、隐式四数组等方法。隐式候选数方法类与显式候选数方法类形成互补，因此在 9×9 规模的数独中，这两种策略的最高阶只到四阶，隐式四数组等价于显式五数组，显式四数组等价于隐式五数组。

排除法：在某一区域中，某个数字只能填写在一个位置。此区域的其他的空白单元均不含有这个候选数，排除其他位置后仅剩一个位置，故名排除法。如图 3-5 所示，B 列中只有 B2 单元格含有候选数 6，因此 6 只能填写在 B2 位置。

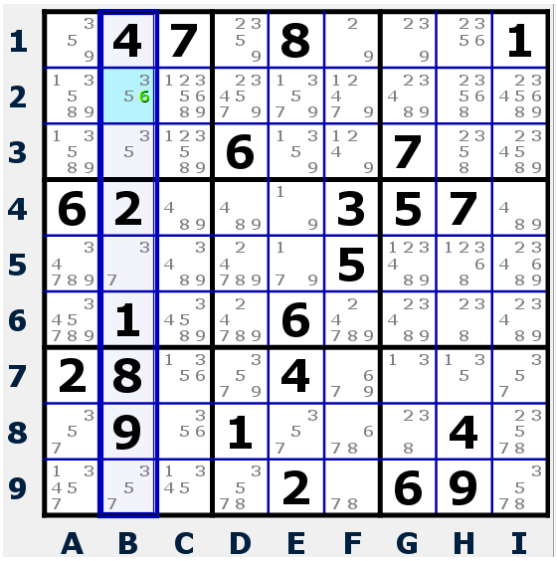

图 3-5　排除法

（图片来源：Sudoku Explainer 1.2.9.0）

隐式数对：在某一区域中，某两个数只能出现在两个单元内，其他的空白单元格均不含有这两个候选数，则构成隐式数对，可以删除这两个单元格内除了这两个数外的其他候选数。如图 3-6 所示，在第九行中，数字 1 和 4 只能出现在 A9 和 C9 两个单元，因此这两数必定填入这两个单元，因此删除 A9 和 C9 单元内其他候选数。

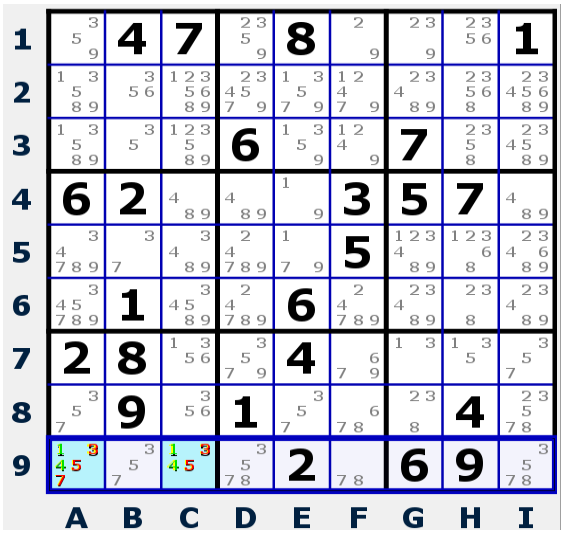

图 3-6　隐式数对

（图片来源：Sudoku Explainer 1.2.9.0）

隐式三数组和隐式四数组的定义与隐式数对相似，仅改变了涉及的单元格和数字的数量，示例见图 3-7。

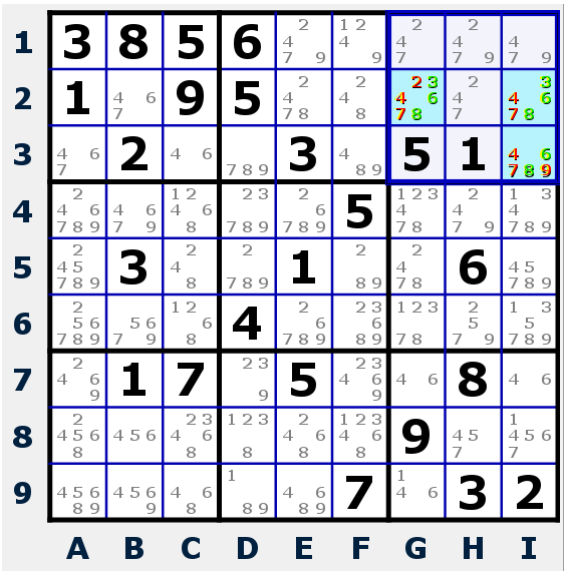

图 3-7　隐式三数组

（图片来源：Sudoku Explainer 1.2.9.0）

3. 交叉排除法

若两个区域存在交叉，且对于其中一个区域而言，某一候选数仅在交叉单元格内出现，则可以删除另一区域其他位置出现的此候选数。如图 3-8 所示，第五宫与 E 列发生交叉，对于第五宫而言，候选数 1 仅出现在与第 E 列交叉的范围内（中间的三个单元），因此数字 1 必定填入此范围内，因此第 E 列的其他位置出现的候选数 1 可被删除。交叉排除法只可发生在行与宫、列与宫的交叉范围内，这是由于行列仅有一个单元格为交叉区域，若数字唯一出现于此，可以直接应用排除法进行解答。

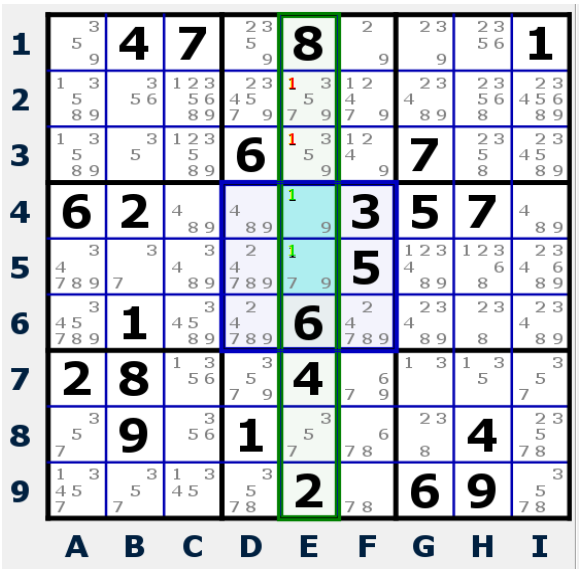

图 3-8　交叉排除法

（图片来源：Sudoku Explainer 1.2.9.0）

4. 鱼方法类

鱼方法类也被称为链列，与显式候选数和隐式候选数相似，链列也分为二阶链列，三阶链列和四阶链列，分别称为二阶鱼，剑鱼和水母。对于 9×9 规模的数独来说，四阶以上的链列都存在其低阶链列形式与之互补，因此鱼方法类的最高阶也只到四阶。

二阶鱼：如图 3-9 所示，候选数 5 在第三行与第九行均出现两次并构成矩形，因此数字 5 必然确定于矩形的一组对角上，于是可以删除矩形的竖边上其他位置出现的候选数 5。其中构成这种形态的第三行与第九行称为鱼的定义域，受到影响需要进行删除候选数操作的第 A 列与第 E 列称为鱼的删除域。

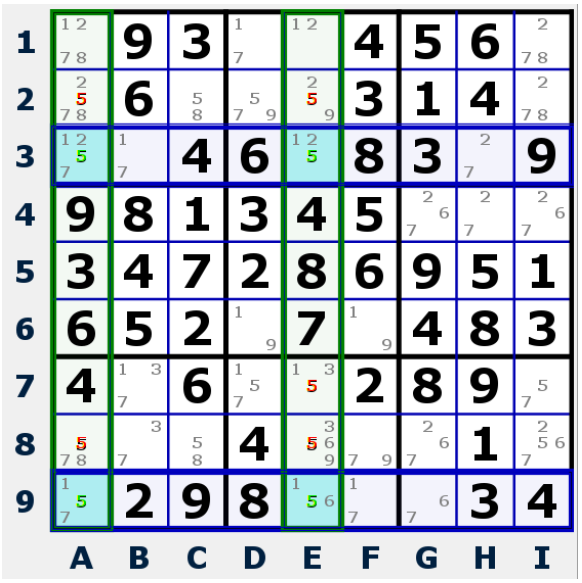

图 3-9　二阶鱼

（图片来源：Sudoku Explainer 1.2.9.0）

三阶鱼（剑鱼）：三阶鱼即在二阶鱼的基础上，将定义域与删除域扩展为三条边，四阶鱼同理，如图 3-10 所示：

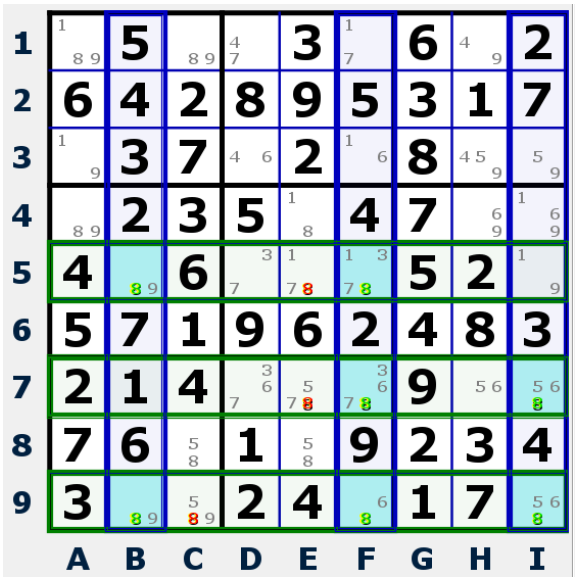

图 3-10　三阶鱼（剑鱼）

（图片来源：Sudoku Explainer 1.2.9.0）

5. 致命结构方法类

致命结构方法类主要利用了标准数独只有唯一解的特性，对于不可能发生的情景进行排除，主要方法有唯一矩形法及其变种和全双值格致死解法。

唯一矩形：如图 3-11 所示，F7，F9，I7，I9 四单元构成矩形，且其中均含有候选数 5 和 9，如果最终答案中四个单元均填入 5 或 9（F7（5），F9（9），I7（9），I9（5））就一定存在另一种对应填法 F7（9），F9（5），I7（5），I9（9）在不影响其他单元的情况下使终盘存在两种答案。由于标准数独仅存在唯一解，因此四个单元不可全部填入 5 或 9，又因为 F7，F9，I9 三个单元格只可填入 5 或 9，可以将第四单元 I7 中的候选数 5 和 9 删除，从而破坏双解结构。唯一矩形一定出现在两行、两列、两宫之中，因此可以根据此规律进行查找。

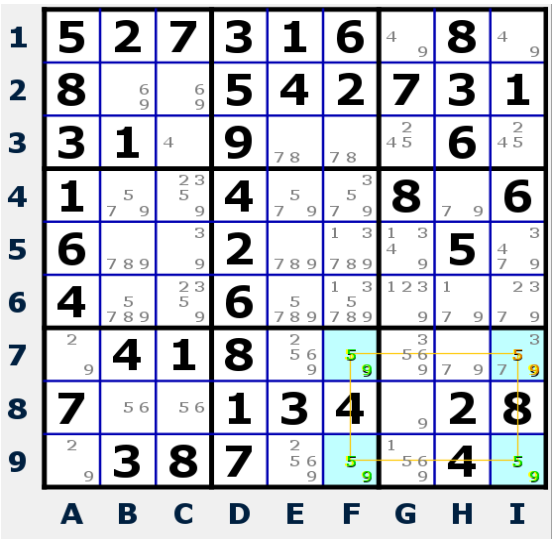

图 3-11　唯一矩形（type1）

（图片来源：Sudoku Explainer 1.2.9.0）

图 3-11 这种情景为标准的 type1 形态唯一矩形，唯一矩形还有多种形态，并可以配合隐式候选数和显式候选数等其他策略使用。图 3-12 为唯一矩形与共轭对结合的变种，对于含有候选数 3 和 8 的唯一矩形 A2，C2，C6，A6 中，矩形的下边，即第六行中候选数 3 仅出现两次，形成共轭对，且候选数 3 完全处于矩形结构中，矩形的上边只可填写 3 或 8，而下边中必然有一单元填写 3，因此下边双角中需要同时将候选数 8 删除，防止出现双解结构。

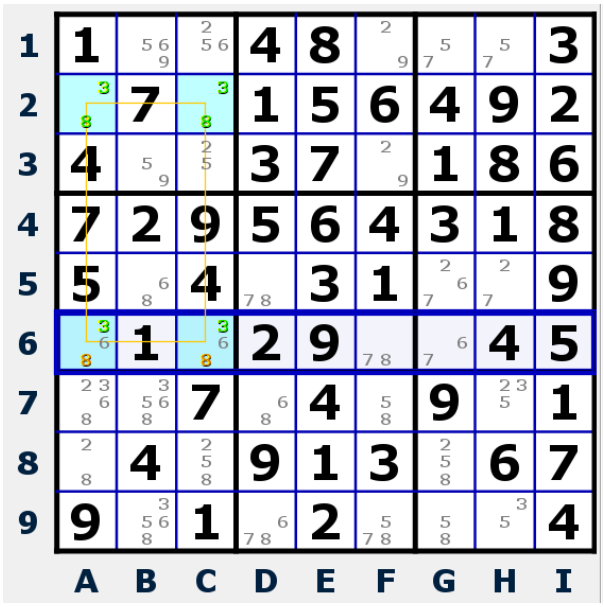

图 3-12　唯一矩形（共轭对）

（图片来源：Sudoku Explainer 1.2.9.0）

全双值坟墓：全双值坟墓与唯一矩形相似，但将形态扩展至全盘。如图 3-13 所示，若移除 I8 单元中的候选数 3，对于存在空白单元格的所有行列宫中，每个区域均形成显式数组结构，可以通过改变填写数字形成完全不同的多种可行解，为破坏此结构，I8 中只能填写数字 3。

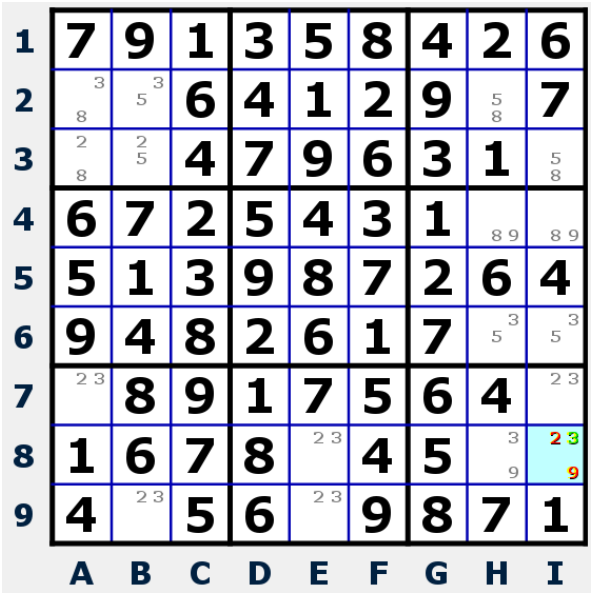

图 3-13　全双值坟墓

（图片来源：Sudoku Explainer 1.2.9.0）

人工策略算法存在两种执行方式，本文将其命名为扫描法和嵌入法。扫描法较为传统，即每种策略都首先进行全盘的扫描，如果发现符合的触发情景，则执行此策略；嵌入法则是在每次填数和删除操作时，对于潜在可能出现的策略触发情景进行局部检查，这种嵌入方法

目前多用于唯一余数法的检测，即在每次删除候选数后直接检查单元候选数是否仅剩余一个，若是，则直接填入，无需进行全盘扫描。本文将同时使用这两种执行方式进行策略的执行，从而比较其时间开销，找到最优的组合。

另外，考虑到执行人工策略的劣势主要在于，当盘面中不存在可触发策略的情景时，会白白浪费搜索时间。而在循环执行人工策略时（直到数据无更新才停止执行），在每一回溯层中都至少使得所有人工策略进行一次注定无效的搜索，这无疑增大了时间开销。作为比较，本文同时设置了在每一回溯层中仅对全部人工策略执行一遍的方式，与原方案的执行时间进行对比，寻找最优的执行方式。

（二） 回溯法的优化

回溯法中的优化目前主要涉及两个方向，1）改变填写空白单元格的顺序；2）改变尝试候选数的顺序。据此，本文设计了四种不同的回溯方式进行比较：

1. 传统回溯法（对照组）

按照空白单元格的位置顺序逐个选择（从第一行至第九行，每行从左至右）；对于候选数，进行从小到大逐个尝试候选数。

2. 空白单元选择回溯法

首先选择剩余候选数数量最少的单元格进行填写，填写顺序为从小到大选择候选数。（若效果较对照组显著更差，可以调整为首先选择候选数数量最多的单元格进行填写）

3. 空白单元选择-候选数选择回溯法

首先选择剩余候选数数量最少的单元格进行填写，首先选取全盘中已确定数字最多的侯选数进行填写。（若效果较对照组显著更差，可以调整为首先选择候选数数量最多的单元格进行填写或首先选取已确定数字最少的候选数进行填写）

4. 单一数字回溯法

首先选取全盘中已确定数字最多的数字进行填写，将全盘内全部九个相同数字填写完成后切换下一数字。填入空白单元顺序为从左至右，从上至下逐个选取可以填入此数字的空白单元进行填写。

（三） 其他优化方式

本文同时充分采用 C++语言中的位运算进行效率优化，如对于单元中存在候选数（1，4，8，9），可以使用九位二进制数字 110001001 存储，为之后的运算操作提供便捷。

（四） 数据结构与算法实现

本文建立 Sudoku 类进行数独的初始化和结果计算，其中为便捷人工策略和回溯策略的操作， Sudoku 类中共初始化了 14 个数组辅助计算，Sudoku 类的具体数据结构如下：

bool FixCell[81]： 记录单元格是否已填入数字

int FixNumber[9]：记录全盘中每个数字已确定的个数

int BitWaitCell[81]：记录每单元内可以使用的候选数（二进制存储）

int BitDisRow[9][9]：记录每行中每个数字可填写的范围分布（二进制存储，BitDisRow[0][0]=010000010 代表在第一行中数字 1 可以填写在第二列和第八列的位置）

int BitDisCol [9][9]：记录每列中每个数字可填写的范围分布（二进制存储）

int BitDisBox [9][9]：记录每宫中每个数字可填写的范围分布（二进制存储）

int NumWaitCell[81]：记录每个单元中剩余可用候选数的数量
int NumWaitRow[9][9]：记录每行中每个数字候选数剩余的数量
int NumWaitCol[9][9]：记录每列中每个数字候选数剩余的数量
int NumWaitBox[9][9]：记录每宫中每个数字候选数剩余的数量
bool FixRowNub [9][9]：记录一行中某个数字是否确定
bool FixColNub [9][9]：记录一列中某个数字是否确定
bool FixBoxNub [9][9]：记录一宫中某个数字是否确定
int Value[81] ：记录数独盘面情况
int NumFixed：记录目前已填好的单元格数量，等于 81 时解答完毕
bool Fail：记录是否出现内部矛盾情况，若存在矛盾，则置 True，否则为 False

同时本文也在全局中建立 6 个数组增强二进制和十进制转换以及行列宫索引转换效率，包括：
int BoxID[81]：返回单元格所处的宫编号
int CellBox[81]：返回单元格在宫内所处的位置编号
int BoxCell[9][9]：返回某宫内某位置的单元格的全局编号
int DigToBit[10]：返回十进制数字到二进制数字的转化（DigToBit[i] = 1<<i-1）
int BitToDig[1<<9]：返回二进制数字到十进制数字的转化（BitToDig[1<<i-1] = i）
int NumBit[1<<9]：返回二进制数字中所包含 1 的个数

对于嵌入式人工策略的编程实现方式，本文以 RemoveCellWait 函数为例进行讲解，RemoveCellWait 函数被设计用于移除某单元的某候选数，核心代码如下：

```
int RemoveCellWait (int cell， int v)//删除一个单元中的候选数
{
    int result=0;
  if((BitWaitCell[cell] & DigToBit [v])!=0)//此单元是否含有此候选数
  {
  //移除后此单元无候选数
     if(FixCell [cell]==false && NumWaitCell [cell]==1)
    { Fail = true; return 0;}
  //移除后此行无法存在某数字
     if(FixRowNub [cell/9][v-1]==false && NumWaitRow [cell/9][v-1]==1)
    { Fail = true; return 0;}
  //移除后此列无法存在某数字
     if(FixColNub [cell%9][v-1]==false && NumWaitCol [cell%9][v-1]==1)
    { Fail = true; return 0;}
  //移除后此宫无法存在某数字
     if(FixBoxNub [BoxID [cell]][v-1]==false && NumWaitBox [BoxID[cell]][v-1]==1)
    { Fail = true; return 0;}

  BitWaitCell [cell] = BitWaitCell [cell] - DigToBit[v];
  //移除该单元特定侯选数
  BitDisRow [cell/9][v-1] = BitDisRow [cell/9][v-1] - DigToBit[cell%9+1];
  //移除该行特定侯选数分布
```

```
    BitDisCol [cell%9][v-1] = BitDisCol [cell%9][v-1] - DigToBit[cell/9+1];
    //移除该列特定侯选数分布
    BitDisBox[BoxID[cell]][v-1] = BitDisBox [BoxID [cell]][v-1]- DigToBit[CellBox[cell]+1];
    //移除该宫特定侯选数分布
    NumWaitCell[cell]--;
    //减少该单元侯选数数量
    NumWaitRow[cell/9][v-1]--;
    //减少该行特定侯选数数量
    NumWaitCol[cell%9][v-1]--;
    //减少该列特定侯选数数量
    NumWaitBox[BoxID[cell]][v-1]--;
    //减少该宫特定侯选数数量
    change=true;
    //标识符

    //嵌入执行排除法
    if(NumWaitRow[cell/9][v-1]==1)
    { result=result | DigToBit [1]; }
    if(NumWaitCol[cell%9][v-1]==1)
    { result=result | DigToBit [2]; }
    if(NumWaitBox[BoxID [cell]][v-1]==1)
    { result=result | DigToBit [3]; }
    }//if
    return result;
  }
```

函数的靠前部分检测是否满足删除候选数的条件，若出现矛盾及时终止；中间部分进行了全部关联数组的更新；尾部使用 result 返回信号，用于检测排除法情景。当删除某候选数后，若在行内此数候选数剩余数量为一，则二进制信号一号位置 1；若在列内此数候选数剩余数量为一，则二进制信号二号位置 1；若在宫内此数候选数剩余数量为一，则二进制信号三号位置 1。信号可在其他策略中被触发返回，最后一并处理，防止出现冲突。如返回的 result 信号为 010，则表示在 cell 单元中移除候选数 v 后，在 cell 这一列中就只剩下另一个唯一的单元可以填写数字 v 了。在其他函数中接收此类信号的方式如下（节选自函数 Fish）：

```
    type = RemoveCellWait (cell，value);// 删除候选数
    SingleRemain(cell);// 检验这个单元格是否可以确定
    if(Fail==true) //每次执行操作需检测矛盾是否已经出现
    {return false;}
    while(type>0)//根据 type 检验是否需要更新
    {
        tag = true;
        type_val[BitToDig[type & ~ (type -1)]-1]| = DigToBit[value];
        type = type & (type -1);
    }
```

```
if(tag)
{
check_signal (cell，type_val);
if(Fail==true)
{return false;}
}//检测更新情况
```

在人工策略函数中，type 接受 RemoveCellWait 传递回的信号，根据将信号值和涉及的候选数，将信息存储至 type_val 数组（数组索引记录行列宫信息，数组内部以二进制记录候选数值），最后将单元格编号和 type_val 数组一同传至 check_signal 函数中进行最后处理。check_signal 部分函数代码如下所示：

```
while(type_val[0]>0)//行更新
{
//还原信号内侯选数数值
value =bit2dig[ type_val[0] & ~ ( type_val[0] -1)];
if(FixRowNub[cell/9][ value-1]==false && NumWaitRow[cell/9][ value-1]==1)
{
    //计算需要更新的单元位置
     newcell=cell/9*9+BitToDig[BitDisRow [cell/9][value -1]]-1;
//执行排除法
     Exclusion(newcell，  value);

     if(Fail==true) //每次执行操作需检测矛盾是否已经出现
     {return false;}
     flag=true;
 }
   type_val[0] = type_val[0] & (type_val[0] -1);
 }
```

check_signal 函数还原了信号的所有信息，并检测现在是否仍然满足执行排除法的情景，若满足，则执行排除法。除了排除法的例子，也可以通过增加信号种类以及更改携带的信息实现其他高级策略，这里不予赘述。

四、 实验结果与分析

本实验使用设备配置为 2.6GHz CPU，8G 内存，操作系统为 Win10，开发平台是 Code：：Block 16. 01，编程语言为 C++11。实验共包含 7 个 9×9 数独数据集[1]，分别为：

1）HoDoku 软件中下载的 137 个数独，命名为 Hodoku_137；

2）andoku 软件中下载的 500 个数独，命名为 Andoku_500；

3）来自 2006 Stanford Local Programming Contest 的 91 个数独，命名为 Stanford_91；

4）GitHub 中搜索到的 hard 等级数独 95 个，命名为 hard_95；

1.数据集均已上传至 https://github.com/Rechard2017/Sudoku_dataset

5）GitHub 中搜索到的只有 17 个已知数的 difficult 等级数独 500 个，命名为 17diff_500；

6）网络上流传的最难数独 15 个（包括 Arto Inkala 的作品《AI Sudoku Top 10》中的 10 个数独），命名为 extreme_15；

7）Royle 收集的 17 个提示数的数独 49151 个，命名为 Su17_49151；

考虑到回溯策略与人工策略在执行上是相对分离的，在下述实验中我们首先找出最优的回溯策略，在此基础上逐步加入人工策略并进行执行上的调整。

（一）回溯策略的比较

按照前文所述的方式，本文设置了四组实验，记录各种回溯法在不同数据集上的表现。其中 Experiment1 为传统回溯法，Experiment2 为空白单元选择回溯法，Experiment3 为空白单元选择-候选数选择回溯法，Experiment4 为单一数字回溯法，实验结果如下表所示：

表 4-1　回溯策略比较

	Experiment1	Experiment2	Experiment3	experiment4
Average time (ms)				
Hodoku_137	17.05	2.4408759	3.482087591	14.46022
Andoku_500	0.84	0.1064	0.1097034	0.541418
Stanford_91	212.27	10.796703	10.77595604	79.10813
hard_95	322.12	12.476842	13.766	117.4232
extreme_15	11.22	1.8666667	2.598	13.71633
17diff_500	1251.46	51.6106	58.4416	22.0238
Su17_49151	1315.3	42.641655	37.54023316	43.24063
Sum	3130.26	121.93974	126.7135802	290.5137
Sum time (ms)				
Hodoku_137	2335.9	334.4	477.046	1981.05
Andoku_500	422.2	53.2	54.8517	270.709
Stanford_91	19316.8	982.5	980.612	7198.84
hard_95	30601.8	1185.3	1307.77	11155.2
extreme_15	168.3	28	38.97	205.745
17diff_500	625731	25805.3	29220.8	11011.9
Su17_49151	17h.57min+	35min	31min	35.5min
Sum	65327176	2124268.7	1877220.05	2157143
Max time (ms)				
Hodoku_137	260.67	53.9	61.1013	261.992
Andoku_500	27.06	2.58	2.0081	23.7043
Stanford_91	7708.94	194.87	196.098	1919.75
hard_95	7851.47	198.81	197.189	2686.22
extreme_15	34.85	7.88	14.43	45.0817
17diff_500	19961.7	1197.98	1377.07	787.587
Su17_49151	211464	5708.91	6663.9	11857.37
Average	35329.81	1052.1329	1215.970914	2511.672
Min time (ms)				
Hodoku_137	0.024	0.028	0.0288	0.0219

Andoku_500	0.014	0.025	0.0266	0.0159
Stanford_91	0.0739	0.038	0.064	0.098
hard_95	0.0704	0.038	0.0641	0.0926
extreme_15	0.41	0.038	0.1107	0.2119
17diff_500	1.82	0.04	0.0595	0.0373
Su17_49151	0.03	0.04	0.0366	0.0354
Average	0.3489	0.0352857	0.055757143	0.073286

根据表 4-1 的结果可以看出，经过单元格选择和候选数选择的方法均要优于传统回溯法，其中单一数字回溯法在 17diff_500 数据集上有着很好的表现，但在其他数据集上表现较差，整体而言其效率有着较大的波动性。空白单元选择回溯法和空白单元选择-候选数选择回溯法的实验结果相似，都有着较快的求解速度，同时他们在不同的数据集上表现也不同。考虑到步骤简洁性，可以认为这种效率的提高主要来源于对填写单元格顺序的选择。优先选取候选数较少的单元格进行填写有着稳定更大概率提前解出正确解，因此认为空白单元选择回溯法为效率最高的回溯方式，进一步进行后续的实验。

（二）人工策略的比较和执行方式的选择

本文首先使用扫描法循环执行作为人工策略执行方式，使用效率最高的空白单元选择回溯法作为回溯方法，根据人工策略的难易程度从简单的人工策略开始逐步加入较难的人工策略进行实验，实验结果如下：

表 4-2　人工策略比较（扫描法）

	Experiment5	Experiment6	Experiment7	Experiment8
Technique				
唯一余数法	1	1	1	1
排除法	1	1	1	1
交叉排除法	0	1	1	1
隐式数对	0	0	1	1
显式数对	0	0	0	1
Backtracking				
空白单元选择	1	1	1	1
Average time (ms)				
Hodoku_137	0.215547	0.261168	0.278832	0.332701
Andoku_500	0.01592	0.019466	0.02038	0.02258
Stanford_91	0.15956	0.126066	0.120868	0.138462
hard_95	0.165242	0.119158	0.116421	0.137895
extreme_15	0.127733	0.184533	0.259333	0.2908
17diff_500	0.060176	0.068346	0.070636	0.075988
Su17_49151	0.037209	0.037565	0.041156	0.040706
Sum	0.781389	0.816302	0.907627	1.039131
Sum time (ms)				
Hodoku_137	29.53	35.78	38.2	45.58

Andoku_500	7.96	9.733	10.19	11.29
Stanford_91	14.52	11.472	10.999	12.6
hard_95	15.698	11.32	11.06	13.1
extreme_15	1.916	2.768	3.89	4.362
17diff_500	30.088	34.173	35.318	37.994
Su17_49151	1828.88	1846.34	2022.88	2000.75
Sum	1928.592	1951.586	2132.537	2125.676
Max time (ms)				
Hodoku_137	3.4842	3.035	3.855	4.08
Andoku_500	0.0693	0.094	0.105	0.11
Stanford_91	0.955	0.82	0.654	1.195
hard_95	0.891	0.848	0.66	1.364
extreme_15	0.595	0.393	0.55	0.595
17diff_500	1.316	1.105	0.524	0.825
Su17_49151	4.49	3.27	2.743	2.83
Average	1.685786	1.366429	1.298714	1.571286
Min time (ms)				
Hodoku_137	0.0116	0.01255	0.01315	0.01365
Andoku_500	0.0081	0.0086	0.0088	0.00885
Stanford_91	0.0197	0.0219	0.0266	0.0179
hard_95	0.0191	0.0197	0.02465	0.0178
extreme_15	0.01745	0.0268	0.0439	0.0483
17diff_500	0.0189	0.0219	0.0216	0.0128
Su17_49151	0.0163	0.0179	0.01835	0.01
Average	0.015879	0.018479	0.022436	0.018471

由表 4-2 可以观察到，在多数数据集上人工策略的使用并没有对解题起到显著的加速作用，而是增大了全盘扫描开销。因此实验结果表现为随着使用人工策略数量的增加，数独的平均求解时间也在增加。同时注意到除了显式数对策略外，随着人工策略的增加，各数据集中耗时最长的数独（相对较难）的求解时间在逐步降低。这也证明了人工策略对较难的数独的确具有加速作用，但对于绝大多数的简单数独来说，复杂的人工策略更可能是无谓的搜索和时间的浪费。因此需要调整人工策略的执行方式尽量降低时间的开销，才能应用更多的人工策略。

继续尝试将人工策略执行方式修改为嵌入式执行进行实验。嵌入式执行方式的关键是在每次进行填数或删数操作时，对于可能出现的其他策略的触发情景进行检查，其优势在于节省了循环全盘搜索的时间浪费，但这种方法也存在劣势，即对于疑似的情况可能会多次产生错误的策略触发判断，进而增大策略执行的时间开销。人工策略的嵌入式执行方式的实验结果如下表所示：

表 4-3　人工策略比较（嵌入法）

	Experiment9	Experiment10	Experiment11	Experiment12
Technique				
唯一余数法	1	1	1	1

排除法	1	1	1	1
交叉排除法	0	1	1	1
隐式数对	0	0	1	1
显式数对	0	0	0	1
Backtracking				
空白单元选择	1	1	1	1
Average time (ms)				
Hodoku_137	0.21	0.262482	0.306168	0.409793
Andoku_500	0.0202	0.024202	0.032442	0.044316
Stanford_91	0.2	0.241165	0.203	0.265341
hard_95	0.242	0.232937	0.219421	0.261895
extreme_15	0.187	0.184133	0.233493	0.296373
17diff_500	0.0642	0.078162	0.081244	0.101494
Su17_49151	0.038	0.046722	0.054815	0.067486
Sum	0.9614	1.069803	1.130583	1.446697
Sum time (ms)				
Hodoku_137	29.4	35.96	41.945	56.1416
Andoku_500	8.9	12.101	16.221	22.158
Stanford_91	18.2	21.946	18.473	24.146
hard_95	20.5	22.129	20.845	24.88
extreme_15	2.8	2.762	3.5024	4.4456
17diff_500	32.1	39.081	40.622	50.747
Su17_49151	1876	2296.43	2694.22	3317
Sum	1991.6	2430.409	2835.828	3499.518
Max time (ms)				
Hodoku_137	2.87	3.66	4.15	5.722
Andoku_500	0.07	0.1231	0.1101	0.1414
Stanford_91	1.12	1.38	0.856	1.126
hard_95	1.15	1.286	0.934	1.1139
extreme_15	0.59	0.9321	0.9138	0.8838
17diff_500	1.58	1.7983	0.5792	0.73
Su17_49151	4.68	5.463	2.182	3.1028
Average	1.72	2.091786	1.3893	1.831414
Min time (ms)				
Hodoku_137	0.013	0.0173	0.0251	0.034
Andoku_500	0.01	0.0129	0.0167	0.0208
Stanford_91	0.019	0.02635	0.0382	0.0495
hard_95	0.02	0.0252	0.0373	0.0485
extreme_15	0.019	0.0264	0.0396	0.0506
17diff_500	0.019	0.0277	0.0388	0.0471
Su17_49151	0.0153	0.0233	0.0314	0.038
Average	0.0165	0.022736	0.032443	0.041214

由实验结果可以看到，对比中嵌入法的表现并不突出，其时间损耗随着策略的加入增长

更为迅速。分析原因，由于交叉排除法和显隐式数对均是涉及两个单元及以上的策略，单一单元格删数行为后返回的信号无法准确预测这些策略的触发情景，而处理错误的执行信号会造成大量时间浪费。可使用嵌入执行方式的最优策略只有排除法和唯一余数法，这两种策略识别触发的准确度是比较高的。目前的研究中，大多只将唯一余数法应用嵌入执行，而排除法仍作为独立策略进行全盘扫描的执行方式。因此本文将唯一余数法和排除法均应用于嵌入执行方式，而其他的策略使用扫描法循环执行，重新实验的结果如下表所示：

表 4-4　人工策略比较（组合）

	Experiment9	Experiment13	Experiment14	Experiment15
Technique				
唯一余数法	1	1	1	1
排除法	1	1	1	1
交叉排除法	0	1	1	1
隐式数对	0	0	1	1
显式数对	0	0	0	1
Backtracking				
空白单元选择	1	1	1	1
Average time (ms)				
Hodoku_137	0.21	0.25	0.25	0.29
Andoku_500	0.0202	0.0182	0.0202	0.021
Stanford_91	0.2	0.118	0.113	0.114
hard_95	0.242	0.118	0.121	0.114
extreme_15	0.187	0.18	0.213	0.28
17diff_500	0.0642	0.058	0.06	0.07
Su17_49151	0.038	0.029	0.0277	0.0295
Sum	0.9614	0.7712	0.7772	0.9185
Sum time (ms)				
Hodoku_137	29.4	34.7	34.8	39.8
Andoku_500	8.9	9.1	10.1	10.5
Stanford_91	18.2	10.7	10.3	10.4
hard_95	20.5	11.2	11.5	10.8
extreme_15	2.8	2.7	3.2	4.2
17diff_500	32.1	29.1	30	35
Su17_49151	1876	1431.8	1366.1	1451.9
Sum	1991.6	1529.3	1466	1562.6
Max time (ms)				
Hodoku_137	2.87	2.66	3.24	3.81
Andoku_500	0.07	0.12	0.11	0.14
Stanford_91	1.12	0.66	0.74	0.71
hard_95	1.15	0.66	0.64	0.67
extreme_15	0.59	0.31	0.39	0.59
17diff_500	1.58	1.07	0.47	0.55
Su17_49151	4.68	7.18	2.49	3.52

Average	1.72	1.81	1.154	1.427
Min time (ms)				
Hodoku_137	0.013	0.013	0.012	0.014
Andoku_500	0.01	0.01	0.01	0.01
Stanford_91	0.019	0.02	0.021	0.022
hard_95	0.02	0.02	0.02	0.021
extreme_15	0.019	0.0256	0.033	0.036
17diff_500	0.019	0.022	0.025	0.028
Su17_49151	0.0153	0.016	0.016	0.016
Average	0.0165	0.0181	0.0196	0.021

可以发现组合方法执行人工策略的效果优于前两种执行方式，组合执行方法成功的提升了计算效率。另外，人工策略直到交叉排除法和隐式数对的使用都表现出效率的上升，而显式数对的加入表现出效率的降低（可能原因为显式数对出现频率较少，搜索开销较大）。可以看到最优策略组合下，全部数据集中求解时间最长的数独仅耗时 3.24ms，全部数据集的数独平均耗时都在 0.25ms 以下。

我们继续加入更加复杂的二阶鱼策略、唯一矩形类型一以及共轭对策略，观察这些策略是否能够在此基础上进一步起到加速作用，实验结果如下表所示：

表 4-5　人工策略比较（高级策略）。

	Experiment14	Experiment16	Experiment17
Technique			
唯一余数法	1	1	1
排除法	1	1	1
交叉排除法	1	1	1
隐式数对	1	1	1
显式数对	0	0	0
二阶鱼	0	1	0
唯一矩形	0	0	1
Backtracking			
空白单元选择	1	1	1
Average time (ms)			
Hodoku_137	0.25	0.271358	0.291121
Andoku_500	0.0202	0.02008	0.024651
Stanford_91	0.113	0.12051	0.116895
hard_95	0.121	0.123098	0.119527
extreme_15	0.213	0.263753	0.26066
17diff_500	0.06	0.074755	0.071568
Su17_49151	0.0277	0.030731	0.036627
Sum	0.7772	0.904286	0.921049
Sum time (ms)			
Hodoku_137	34.8	37.1761	39.8836
Andoku_500	10.1	10.0398	12.3257

Stanford_91	10.3	10.9664	10.6374
hard_95	11.5	11.6943	11.3551
extreme_15	3.2	3.9563	3.9099
17diff_500	30	37.3777	35.7839
Su17_49151	1366.1	1510.46	1800.24
Sum	1466	1621.671	1914.136
Max time (ms)			
Hodoku_137	3.24	3.1625	3.7698
Andoku_500	0.11	0.089	0.1457
Stanford_91	0.74	0.6922	0.4533
hard_95	0.64	0.68888	0.4788
extreme_15	0.39	0.5068	0.4475
17diff_500	0.47	0.6069	0.4781
Su17_49151	2.49	2.6196	2.3295
Average	1.154	1.195126	1.157529
Min time (ms)			
Hodoku_137	0.012	0.0151	0.0169
Andoku_500	0.01	0.0117	0.0133
Stanford_91	0.021	0.0236	0.0271
hard_95	0.02	0.0241	0.0279
extreme_15	0.033	0.039	0.0712
17diff_500	0.025	0.0303	0.0345
Su17_49151	0.016	0.0192	0.0204
Average	0.0196	0.023286	0.030186

由表 4-5 可以观察到，更高级策略的加入表现为时间开销的增大，结论与预想相同。对于相对简单的显式数对策略来说，已经无法起到加速作用，扫描开销更大，出现次数和实际操作行为更少的高等级策略显然无法加速解题。因此，在 9×9 维度的数独中无需进行更高级策略的测试。

最后考虑到上文中使用扫描法循环执行的所有人工策略，在回溯的每一层中均至少存在一次无效的策略触发情景扫描。因此，我们将策略在回溯的每一层中仅执行一遍进行实验，以检测策略的循环执行是否值得，实验结果如下：

表 4-6　人工策略比较（执行一次）

	Experiment 9	Experiment 18	Experiment 19	Experiment 20
Technique				
唯一余数法	1	1	1	1
排除法	1	1	1	1
交叉排除法	0	1	1	1
隐式数对	0	0	1	1
显式数对	0	0	0	1
Backtracking				
空白单元选择	1	1	1	1

Average time (ms)				
Hodoku_137	0.214599	0.244044	0.228438	0.250812
Andoku_500	0.0178	0.02084	0.02047	0.022489
Stanford_91	0.2	0.111232	0.095736	0.108385
hard_95	0.215789	0.105663	0.098158	0.100621
extreme_15	0.186667	0.182613	0.192947	0.275047
17diff_500	0.0642	0.057916	0.055674	0.063462
Su17_49151	0.038168	0.029292	0.027941	0.029328
Sum	0.937223	0.7516	0.719364	0.850144
Sum time (ms)				
Hodoku_137	29.4	33.434	31.296	34.3613
Andoku_500	8.9	10.42	10.235	11.2447
Stanford_91	18.2	10.1221	8.712	9.863
hard_95	20.5	10.038	9.325	9.559
extreme_15	2.8	2.7392	2.8942	4.1257
17diff_500	32.1	28.958	27.837	31.731
Su17_49151	1876	1439.72	1373.33	1441.49
Sum	1987.9	1535.431	1463.629	1542.375
Max time (ms)				
Hodoku_137	2.87	2.85	2.8644	3.1313
Andoku_500	0.07	0.0978	0.08885	0.1293
Stanford_91	1.12	0.6322	0.6248	0.659
hard_95	1.15	0.6631	0.5663	0.5779
extreme_15	0.59	0.3781	0.348	0.5663
17diff_500	1.58	0.9413	0.3397	0.5076
Su17_49151	4.68	2.97	2.1855	2.6614
Average	1.72	1.218929	1.002507	1.176114
Min time (ms)				
Hodoku_137	0.013	0.0151	0.0153	0.0154
Andoku_500	0.01	0.0116	0.01125	0.01155
Stanford_91	0.019	0.0227	0.023	0.02475
hard_95	0.02	0.0225	0.0248	0.024
extreme_15	0.019	0.0379	0.0359	0.0392
17diff_500	0.019	0.0255	0.0284	0.0273
Su17_49151	0.0153	0.0183	0.0185	0.0191
Average	0.0165	0.021943	0.02245	0.023043

可以观察到执行一次人工策略的方式在某些数据集上的确具有加速效果，并可将所有数据集中最难数独的求解时间降低至 3ms 以下，但其加速效果在整体上观察并不明显。人工策略仍然是从显式数对策略开始出现时间上的反弹。

最后，鉴于前文中空白单元选择回溯法和空白单元选择-候选数选择回溯法的实验结果较为相似，本文在最优的策略组合上更换回溯方式进行稳健性检验。同时考虑到在回溯过程中我们首先选用侯选数最少的单元（平均候选数个数为 2~3 个），因此在每一层回溯中存在的平行状态平均也为 2~3 个。借鉴 Stuart 的发现，大部分数独仅需要两到三次不得已的猜

数操作即可求解成功，即只需要 2 到 3 层回溯，平均而言需要存储的数独状态不超过 9 个。因此可以尝试使用广度优先算法，代替深度优先算法对状态集进行搜索，算法流程图如下：

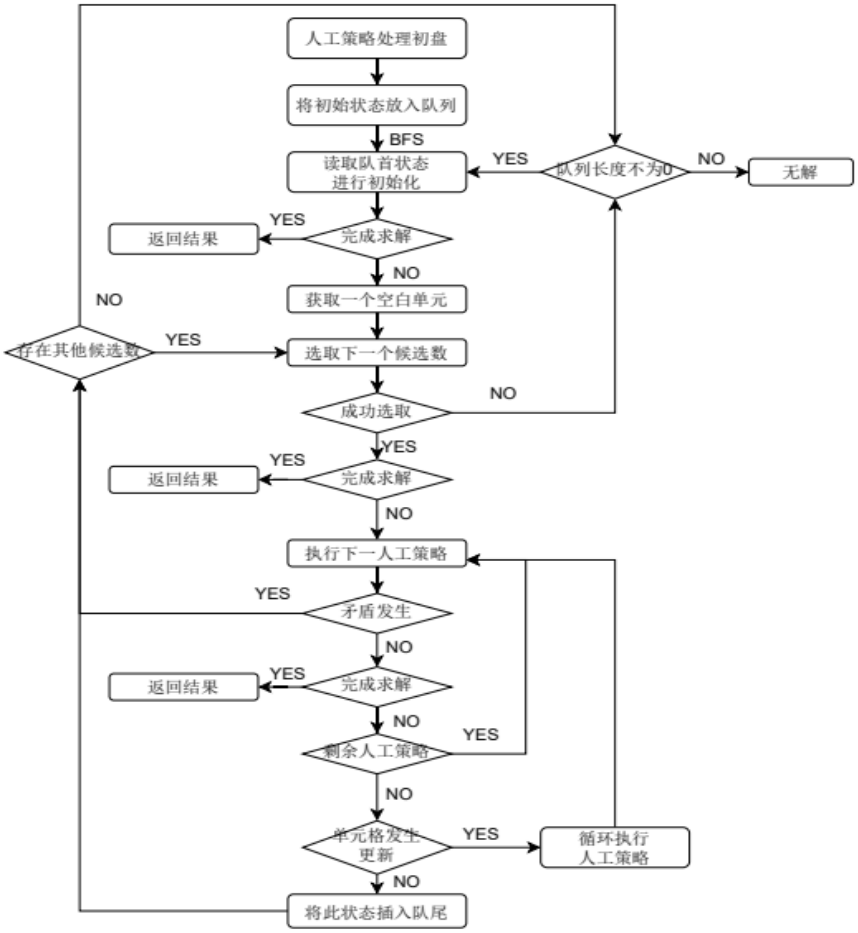

图 4-1 广度优先算法框架流程图

如果采用优先队列数据结构，可以选取最接近最终结果的状态进行优先处理，这里选取所有数独状态中，空白单元格数量最少的数独状态进行优先处理，将这种方式称为状态选择回溯。将普通广度优先和状态选择回溯两种算法一同加入到稳健性检验中，实验结果如下表所示：

表 4-7 稳健性检验（回溯法替换）

	Experiment19	Experiment 21	Experiment22	Experiment23
Technique				
唯一余数法	1	1	1	1
排除法	1	1	1	1
交叉排除法	1	1	1	1
隐式数对	1	1	1	1
显式数对	0	0	0	0
Backtracking				
空白单元选择	1	0	0	0
空白单元选择-候选数选择	0	1	0	0

普通广度优先	0	0	1	0
状态选择回溯	0	0	0	1
Average time (ms)				
Hodoku_137	0.228438	0.392521	0.292357	0.581551
Andoku_500	0.02047	0.021115	0.029142	0.022416
Stanford_91	0.095736	0.117616	0.155912	0.143915
hard_95	0.098158	0.102274	0.141128	0.147
extreme_15	0.192947	0.21514	0.334533	0.238787
17diff_500	0.055674	0.057399	0.075542	0.071909
Su17_49151	0.027941	0.032492	0.031946	0.032799
Sum	0.719364	0.938557	1.06056	1.238377
Sum time (ms)				
Hodoku_137	31.296	53.7754	40.0529	79.6725
Andoku_500	10.235	10.5575	14.5709	11.2078
Stanford_91	8.712	10.7031	14.188	13.0963
hard_95	9.325	9.716	13.4072	13.965
extreme_15	2.8942	3.2271	5.018	3.5818
17diff_500	27.837	28.6996	37.7708	35.9547
Su17_49151	1373.33	1596.99	1570.16	1612.08
Sum	1463.629	1713.669	1695.168	1769.558
Max time (ms)				
Hodoku_137	2.8644	7.9706	4.0413	15.3826
Andoku_500	0.08885	0.1231	0.1486	0.1297
Stanford_91	0.6248	0.5868	1.3821	0.784
hard_95	0.5663	0.631	1.225	0.7851
extreme_15	0.348	0.6171	0.6825	0.5275
17diff_500	0.3397	0.3597	0.7972	0.9311
Su17_49151	2.1855	4.2581	0.9988	3.2318
Average	1.002507	2.078057	1.325071	3.110257
Min time (ms)				
Hodoku_137	0.0153	0.0134	0.0171	0.0144
Andoku_500	0.01125	0.011	0.0145	0.0111
Stanford_91	0.023	0.0229	0.0251	0.0261
hard_95	0.0248	0.0249	0.0245	0.0271
extreme_15	0.0359	0.0347	0.0369	0.039
17diff_500	0.0284	0.0295	0.0299	0.0308
Su17_49151	0.0185	0.0186	0.019	0.0183
Average	0.02245	0.022143	0.023857	0.023829

可以观察到，替换回溯法后的数独求解时间未能优于使用空白单元选择回溯法的结果，可以认为空白单元选择回溯法是简洁、稳健、高效的回溯方式。

综上所述，9×9 规模下的数独最优的解题算法组合为，使用嵌入执行方式执行排除法与唯一余数法，扫描方式循环或单次执行交叉排除法与隐式数对，使用空白单元选择回溯法融合人工策略进行回溯。这种组合可以在绝大多数数据集中最大化效率，做到最快速的解题。

（三）高维度上的表现

接着，我们将算法应用于更高维度的 16×16 的数独上，来观察需要对策略和策略的执行方式做出何种调整以达到最优。16×16 的数独较 9×9 的数独来说，全盘搜索需要耗费更长的时间，但在增大的盘面中策略的出现频率也会增加，因此结论可能呈现出与 9×9 数独不同的特性，比如某些策略的效果不佳或可以应用更复杂的策略。本文使用网络爬虫，爬取来自 https://www.menneske.no/sudoku/4/eng/中的 34135 个 16×16 数独作为数据集，命名为 Sixteen_34135 进行上述实验，由于回溯法在不同维度的数独中其时间复杂度并没有本质上区别，因此回溯法只沿用上述实验中效率最高的空白单元选择回溯法，实验结果如下表所示：

表 4-8　16×16 人工策略比较（扫描法）

	Ex24	Ex25	Ex26	Ex27	Ex28	Ex29
Technique						
唯一余数法	1	1	1	1	1	1
排除法	1	1	1	1	1	1
交叉排除法	0	1	1	1	1	1
隐式数对	0	0	1	1	1	1
显式数对	0	0	0	1	1	1
隐示三数组	0	0	0	0	1	1
显示三数组	0	0	0	0	0	1
Backtracking						
空白单元选择	1	1	1	1	1	1
Average time (ms)						
Sixteen_34135	0.97797	0.473268	0.285283	0.229459	0.24109	2.532752
Sum time (ms)						
Sixteen_34135	33383	16155	9738.14	7832.57	8229.62	86455.5
Max time (ms)						
Sixteen_34135	485.544	744.626	686.036	134.87	151.432	1542.55
Min time (ms)						
Sixteen_34135	0.0212	0.0238	0.0253	0.0264	0.0263	0.0267

表 4-9　16×16 人工策略比较（嵌入法）

	Experiment 30	Experiment 31	Experiment 32	Experiment 33
Technique				
唯一余数法	1	1	1	1
排除法	1	1	1	1

交叉排除法	0	1	1	1
隐式数对	0	0	1	1
显式数对	0	0	0	1
Backtracking				
空白单元选择	1	1	1	1
Average time (ms)				
Sixteen_34135	0.658711	0.736408	0.527807	0.746313
Sum time (ms)				
Sixteen_34135	22485.1	25137.3	18016.7	25475.4
Max time (ms)				
Sixteen_34135	429.242	462.479	363.846	546.497
Min time (ms)				
Sixteen_34135	0.0269	0.0365	0.054	0.0799

表 4-10　16×16 人工策略比较（组合）

	Ex30	Ex34	Ex35	Ex36	Ex37	Ex38
Technique						
唯一余数法	1	1	1	1	1	1
排除法	1	1	1	1	1	1
交叉排除法	0	1	1	1	1	1
隐式数对	0	0	1	1	1	1
显式数对	0	0	0	1	1	1
隐示三数组	0	0	0	0	1	1
显示三数组	0	0	0	0	0	1
Backtracking						
空白单元选择	1	1	1	1	1	1
Average time (ms)						
Sixteen_34135	0.65871	0.31237	0.15850	0.13372	0.13435	1.09679
Sum time (ms)						
Sixteen_34135	22485.1	10662.9	5410.5	4564.76	4586.3	37439.1
Max time (ms)						
Sixteen_34135	429.242	596.21	233.277	44.2709	43.1068	703.093
Min time (ms)						
Sixteen_34135	0.0269	0.0279	0.0289	0.0294	0.0295	0.0299

表 4-11　16×16 人工策略比较（高级策略）

	Experiment36	Experiment 39	Experiment 40
Technique			
唯一余数法	1	1	1
排除法	1	1	1
交叉排除法	1	1	1
隐式数对	1	1	1

显式数对	1	1	1
隐示三数组	0	0	0
显示三数组	0	0	0
二阶鱼	0	1	0
唯一矩形	0	0	1
Backtracking			
空白单元选择	1	1	1
Average time (ms)			
Sixteen_34135	0.133727	0.140074	0.203211
Sum time (ms)			
Sixteen_34135	4564.76	4781.42	6936.62
Max time (ms)			
Sixteen_34135	44.2709	50.5279	51.4019
Min time (ms)			
Sixteen_34135	0.0294	0.0295	0.0381

表 4-12　16×16 人工策略比较（执行一次）

	Ex30	Ex41	Ex42	Ex43	Ex44	Ex45
Technique						
唯一余数法	1	1	1	1	1	1
排除法	1	1	1	1	1	1
交叉排除法	0	1	1	1	1	1
隐式数对	0	0	1	1	1	1
显式数对	0	0	0	1	1	1
隐示三数组	0	0	0	0	1	1
显示三数组	0	0	0	0	0	1
Backtracking						
空白单元选择	1	1	1	1	1	1
Average time (ms)						
Sixteen_34135	0.65871	0.30009	0.14411	0.12147	0.12295	0.87928
Sum time (ms)						
Sixteen_34135	22485.1	10243.8	4919.25	4146.54	4196.94	30014.4
Max time (ms)						
Sixteen_34135	429.242	586.85	195.51	37.3388	41.2979	470.027
Min time (ms)						
Sixteen_34135	0.0269	0.028	0.0288	0.0295	0.0294	0.0299

表 4-13　16×16 稳健性检验（回溯法替换）

	Experiment43	Experiment46	Experiment47	Experiment48
Technique				
唯一余数法	1	1	1	1
排除法	1	1	1	1

交叉排除法	1	1	1	1
隐式数对	1	1	1	1
显式数对	1	1	1	1
Backtracking				
空白单元选择	1	0	0	0
空白单元选择-候选数选择	0	1	0	0
普通广度优先	0	0	1	0
状态选择回溯	0	0	0	1
Average time (ms)				
Sixteen_34135	0.121474733	0.126565	0.126284	0.175447
Sum time (ms)				
Sixteen_34135	4146.54	4320.31	4310.7	5988.88
Max time (ms)				
Sixteen_34135	37.3388	41.975	31.7503	253.457
Min time (ms)				
Sixteen_34135	0.0295	0.0294	0.0325	0.031

由上述全部实验结果可以观察到，在 16×16 数独中，组合执行策略也较单独使用扫描法和嵌入法更具优势。与 9×9 数独不同的是，16×16 数独可以应用的策略更为复杂，可以使用显式数对策略和隐式三数组策略进行加速，直到应用显式三数组策略时才表现为耗时的显著增加。复杂的鱼策略和唯一矩形策略同样由于效率较低，不能在 16×16 数独上起到较好的加速作用。而在每层回溯中执行一次策略较循环执行策略的调整方式，在 16×16 数独中有着较好的效果。可见减少无效的策略情景扫描对于高维数独来说是十分必要的，因为高维数独有着更大的盘面、更长的搜索时间开销。

综上所述，16×16 规模下的数独最优的解题算法组合为，使用嵌入执行方式执行排除法与唯一余数法，扫描执行方式在每层回溯中单次执行交叉排除法、隐式数对和显示数对策略，使用空白单元选择回溯法融合人工策略进行回溯。这种策略组合在 16×16 规模的数独中平均求解时间为 0.12ms，耗时最长的数独解题时长为 37ms。

五、 结论

使用融合人工策略的回溯法求解数独有着良好的解题速度，同时通过改变回溯法和人工策略的执行细节可以进一步优化求解时间。对于回溯法，首先选取候选数最少的单元格进行填写是最优的回溯执行方式。对于人工策略的执行来说，排除法和唯一余数法策略的触发只涉及单个单元格，因此可以在每次进行填数或删数操作时及时进行更新，即嵌入其他函数中的嵌入式执行方法。而其他较为复杂的策略需要在每层回溯进行全盘扫描执行，这种组合执行方式可以较为显著的提升数独算法的运行速度。另外，在数独盘面较大的情况下，可以在每层回溯中仅执行一遍人工策略进行减枝，通过降低不必要搜索开销的方式优化求解速度。9×9 数独的最优人工策略组合为排除法、唯一余数法、交叉排除法、隐式数对，16×16 数独可以在此基础上增加显式数对和隐式三数组策略以达到最优效率。

进行优化后的融合人工策略的回溯法进一步提升了数独求解的效率。对于 9×9 规模的数独，相较于原算法，优化后的算法将相同数据集内的数独求解时间上限由 10ms 降低至 3ms，全数据集数独平均求解时间 0.029ms，其中对于性质较好的 Su17_49151 数据集，原文平均求解时间为 0.085ms，本文将其降低至 0.027ms，约占原时间开销的 30%。对于 16×16 规模的高维数独，本文算法仍然具有很快的求解速度，平均求解时间为 0.12ms，耗时最长的 16×16 数独解题时长仅为 37ms。本文结论有助于加速数独的求解和生成算法，对数独在其他领域的应用也具有一定启示。

参考文献

[1]程曦，肖华勇，吴林波.数独求解的候选数优化算法设计[J].科学技术与工程，2011，11(26)：6409-6412.
[2]胡伟通，李明楚，郭成，袁理峰.一种利用数独和细胞自动机的秘密图像共享方案[J].小型微型计算机系统，2015，36(06)：1271-1275.
[3]江顺亮，唐祎玲，徐少平，叶发茂等.融合人工求解策略的数独回溯求解法[J].计算机应用研究，2020，037(002)：481-485.
[4]西尾彻也(日)，《数独大师-专业篇》，科学出版社，2017.
[5]张煜东，王水花，霍元恺，吴乐南.一种基于稀疏优化的数独求解新方法[J].南京信息工程大学学报(自然科学版)，2011，3(01)：23-27.
[6]Donald E. Knuth. Dancing links[A]. in Jim Davies，Bill Roscoe，Jim Woodcock (eds.). Millennial Perspectives in Computer Science：Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare[C]. Palgrave，2000：187–214.
[7]Eric C. Chi，Kenneth Lange. Techniques for solving Sudoku puzzles? [EB/OL]. (2013-05-16). . https：/ / arxiv. org / abs/ 1203. 2295.
[8]Gary McGuire and Bastian Tugemann and Gilles Civario. There Is No 16-Clue Sudoku：Solving the Sudoku Minimum Number of Clues Problem via Hitting Set Enumeration[J]. Experimental Mathematics，2014，23(2)：190-217.
[9]Jiann-Ming Wu and Pei-Hsun Hsu and Cheng-Yuan Liou. Sudoku associative memory[J]. Neural Networks，2014，57：112-127.
[10]Syndicated Puzzles Inc. A new metric for difficult Sudoku puzzles?[EB / OL]. [2018-07- 05]. http： / / www. sudokuwiki. org / A New Metric for Difficult Sudoku Puzzles.
[11]Thai-Son Nguyen and Chin-Chen Chang. A reversible data hiding scheme based on the Sudoku technique[J]. 2015，39：109-116.
[12]Zhai Gaoshou，Zhang Junhong. Solving Sudoku puzzles based on customized information entropy[J]. International Journal of Hybrid Information Technology，2013，6(1)：77-92.