建 模

基于"挖洞"思想的数独游戏生成算法

薛源海， 蒋彪彬， 李永卓
指导老师:闫桂峰， 孙华飞

(北京理工大学理学院 数学系，北京 100081)

摘要： 设计一个算法用以生成各种难度等级的数独题，通过对游戏规则的分析，首先从以下三个方面定义难度等级:已知格总数、已知格的分布和穷举搜索复杂度.本算法采用"挖洞"思想，经过以下两步生成数独题：1) 运用拉斯维加斯随机算法生成一个终盘；2) 采用以下五个操作"抹去"一部分数字来生成数独题：① 根据所需要的难度等级选取一种挖洞顺序；② 制定两个约束来控制已知格的分布；③ 通过深度优先搜索来求解，从而保证"挖去"一个数字后该数独题仍有唯一解；④ 引入剪枝技术来避免无效的"挖洞"尝试；⑤ 对"挖"好"洞"的数独题进行等效对称变换，以增加题目的多样性. 可以生成游戏者所需要的任意5种难度的数独题.经过对算法时间和空间复杂度的分析，论证了本算法的有效性. 对"挖洞法"的研究成果可总结为以下三个方面：1) 通过对"挖洞"顺序的大量试探，找到了可生成高难度数独题的"挖洞"顺序；2) 采用反证法来判断一个数独题解的唯一性；3) 通过避免"回溯"和"重填"来降低算法的运行时间.

关键词： 挖洞法;拉斯维加斯算法;剪枝;反证法

1 问题的提出

数独是一种源自18世纪末的瑞士，后在美国发展，并在日本得以发扬光大的数学智力拼图游戏[1]. 其游戏规则为：在由9个小九宫格组成的大九宫格(9格×9格)里，已填有若干数字;需用数字1～9填满剩下的空格，使得每一行和每一列都包含数字1～9，每个小九宫格里也均是数字1～9，并且每个数字在它所在的行、列以及小九宫格里面都只出现一次.该游戏环境如图1所示.

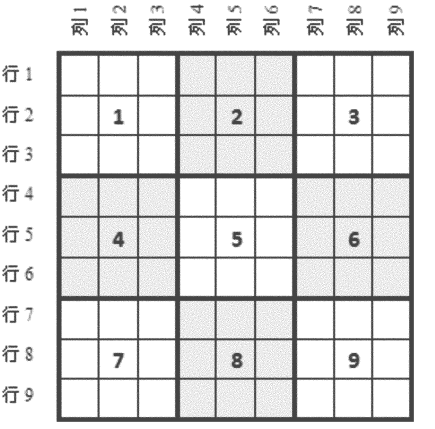


图1 数独游戏环境

近年来，数独游戏正吸引着世界上无数的游戏挑战者.他们水平各异，所以一些计算机学者们正致力于设计算法使其在尽可能短的时间内生成不同难度等级的数独题，以满足不同水平游戏者的需求[2]. 在此，本文提出了一种基于"挖洞"思想的数独题生成算法，同时考虑到三个方面要求:可变化的难度、解的唯一性和算法复杂度最小化.

收稿日期:2008-07-04

注:2008年美国数学建模竞赛(MCM)一等奖(Meritorious Winner)

2 问题的分析

要能保证算法生成的数独题具有可变化的难度和唯一解，该算法内部应该包含有对数独题的求解和评级功能.本文将该算法的设计工作分为评级、求解和生成 3 部分工作.

首先，根据游戏者需求的难度等级，我们从已知格的总数和分布以及穷举求解的搜索次数这三个方面特性，通过赋权求和来评估一个数独题的难度等级.然后，运用"反证法"思想，并结合"深度优先搜索求解器"一起论证一个"洞"被"挖去"后是否能保证该题有唯一解.最后，论证了可通过避免"重填"一个被"挖去"的格子，或者"回溯"到一个曾经无法"挖去"的格子，来降低算法的复杂性.

3 模型的假设和定义

1) 在本文中，我们将数独游戏的格盘大小定为 9 格×9 格，不涉及其他大小的格盘.

2) 文中所有关于运行时间的统计数据都是通过我们的计算机得到的；在同等的计算环境下，这些数据是可靠而具有可比性的.

4 难度等级的度量

在本节中，我们从人类的逻辑推理和计算机的穷举搜索两方面来评估一道数独题的难度.由于游戏者对未知格的推理建立在已知格所提供的信息上.而已知格的数量和分布决定了它们能为游戏者所提供信息的多少[3]. 已知格数量越少，其分布越不均匀、不对称，该数独题通过逻辑推理的求解难度就越大.

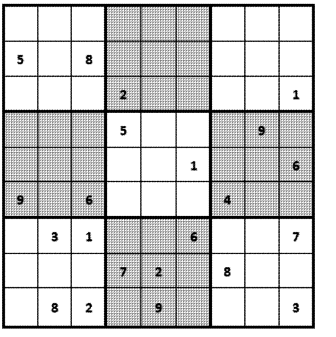


图 2 数独题评分样例

由此针对某一道数独题(见图 2) ，我们给出如下三个评定项目(见表 1) ，对其进行 0～5 分的评估，并给这三项指标赋予相应的权重，进而求和得到该题的得分(亦为 0～5 分) .将该分数的整数部分用以权衡游戏者需求的难度等级如下：等级一，低级，0～1 分；等级二，初级，1～2 分；等级三，中级，2～3 分；等级四，高级，3～4 分；等级五，骨灰级，4～5 分.

表 1 数独题评分表

评分项目	详细描述	权重	得分
已知格总数	总共 22 个已知，在等级五范围内	0.4	5
行、列中已知格数的低限	第一行中有 0 个已知，达到等级五的低限	0.3	5
穷举搜索次数	263734 次，等级四	0.3	4
总评	等级五，骨灰级	总分	4.7

5 算法详述

整个生成数独题的算法分为两步执行：先随机生成一个填满数字且符合游戏约束的终盘，而后根据所需求的难度等级"抹去"一些数字，每"抹去"一个数字就验证一次解的唯一性；如此往复直至满足所需的难度要求为止.最后通过适当的等价对称变换后输出该题.

5.1　求解算法：深度优先搜索

为了让生成算法可以判断任意一个数独题是否有唯一解，我们建立一个可以搜遍所有可行解的数独求解器. 由于我们定义的数独格盘的大小仅为 9 格×9 格，所以我们采用穷举搜索的办法，按照深度优先的搜索顺序，找出一个数独题的所有可行解.

求解器由上至下，从左至右对每个空格进行填数尝试，即填入一个 1～9 的数字，同时检验填入的数字是否满足数独游戏对每个格内数字的三个约束(数字 1～9 分别有且仅有一次出现在每行、每列和每个小九宫格中)，若能满足则视此次填入的数字可行. 若在一个空格的尝试中，1～9 均无法填入，则回溯到前一个所尝试的空格中并将其中已填入的数字替换成其他未尝试过的数字. 如此往复直至找遍所有可行解.

5.2　生成算法

借助先前建立的两个工具：难度评定准则和遍历搜索求解器，我们用以生成数独题的算法分为如下三步进行：1) 生成一个终盘；2) 在终盘上"挖洞"；3) 对"挖"好的题进行等效变换.

5.2.1　生成终盘：拉斯维加斯算法(源代码见附录)

我们采用拉斯维加斯算法随机生成一个数独题的终盘(所谓"终盘"，即数独题的答案). 首先，在一个数独"空盘"中随机选中 n 个格子，在这些格子内随机填入数字1～9；然后，调用先前的求解器对这个已知 n 格的数独题 $S(n)$ 进行求解. 完成上述两步操作的代码段为 **BOOL** las -vegas(n) ：如果 $S(n)$ 有解，则返回 **TRUE**；否则返回 **FALSE**. 拉斯维加斯算法的思想便是不断重复执行 las -vegas(n)，直至其返回 **TRUE** 为止[4]，用编程语言表示如下：

```
while(! las -vegas( n) ) ;
```

代码段 las -vegas(n) 返回 **TRUE**(即成功生成一个"终盘") 的概率 p 与 n 值有关. 通过对 n 值进行大量的尝试和调整，我们发现当 $n=11$ 时，概率 p 已能达到 99.7%，且 p 随着 n 的减小而增大，但运行时间延长. 故我们将 n 设为 11.

5.2.2　"挖洞"算法(源代码见附录)

"挖洞法"的基本思想就是"挖去"一个数独终盘上的一些格子，使其成为有唯一解的数独题. 然而"挖洞"的机制是多种多样的，为了加快出题算法的速度，我们将"贪心"机制引入到我们的"挖洞"策略当中. 整个"挖洞"算法的流程图如图 3 所示.

以下我们将对"挖洞"流程中的 5 个关键操作(如流程图中标记所示) 进行详细说明.

操作1　"挖洞"顺序

"挖洞"顺序即在一个终盘上，决定哪个位置的格子先被"挖去"，哪个格子是下一个. 该顺序对生成的题目中已知格的分布起决定性作用. 我们通过大量尝试，选取了以下四个较为有效的顺序(图 4)，并根据四者的效果将其分配给 5 个难度等级(表 2) .

顺序 1　从左到右，至顶向下(见图 4(a))

顺序 2　蛇形(见图 4(b))

顺序 3　间隔(见图 4(c))

表 2　难度等级对应的"挖洞"顺序

难度等级	"挖洞"顺序
1(低级)	全盘随机
2(初级)	全盘随机
3(中级)	间隔
4(高级)	蛇形
5(骨灰级)	从左到右，至顶向下

顺序 4　全盘随机

操作 2　"挖洞"约束

根据难度等级的定义，我们对"挖洞"操作加入两个约束来影响已知格的分布态势：

约束 1(已知格的总数)　已知格数最多为 80 格，将 0～80 大致均分为五个区间，对应到 5 个难度等级. 已知格越少则题越难. 若游戏者需要某个难度的数独题，则在其对应的区间内随机一个数作为已知格数的下界，即当"挖洞"至剩余格数达到下界时停止.

约束 2(行、列中已知格数的低限)　同理约束 1，每行、每列剩余的格数必须多于难度等级所规定的下界，否则禁止此次"挖洞"操作.

每尝试"挖去"一个格子，都要检验一次"挖去"该格后是否与这两个约束冲突. 一旦冲突则禁止此次"挖洞".

操作 3　反证法判断解的唯一性

我们借助"反证法"的思想提出了一个新的策略来随时检验当一个格子中的数字被"挖去"后，是否能保证此题还有唯一解. 该策略按步解释如下.

假设在一次"挖洞"尝试中，我们试图抹去一个填有数字 6 的格子.

步骤 1　将数字 6 依次替换为除 6 以外 1～9 的其他八个数字，检查是否违反游戏规则.

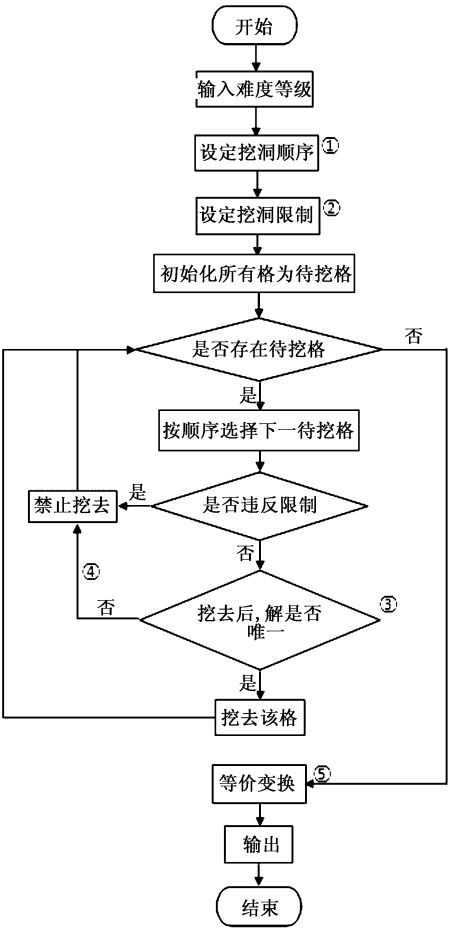


图 3　"挖洞"算法流程图

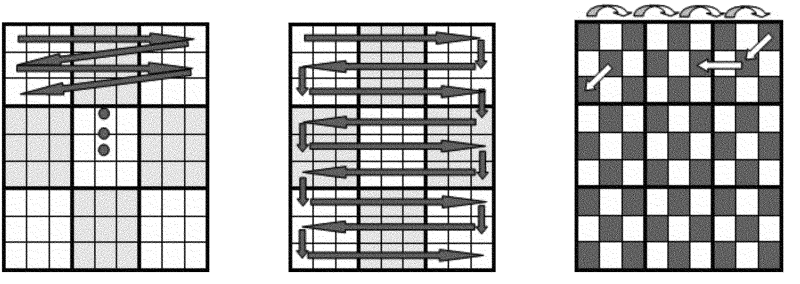


图 4　"挖洞"顺序图示

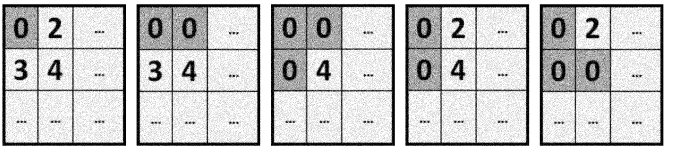
步骤 2　调用求解器对完成步骤 1 后的数独题进行求解.

步骤 3　一旦求解器能得到一个解，便立即终止求解器，同时宣布抹去 6 后得到的题至少有两解. 因为针对这一空格，原本填入 6 时已有一个来自初始终盘的解.

步骤 4　只有当所有除 6 外的八个数依次经过步骤 1 的检验，同时步骤 2 中求解器遍历搜索后回复无解，才能保证抹去数字 6 后的数独题的解具有唯一性，即判定此次"挖洞"操作是合法的.

操作 4　剪枝优化

我们通过剪枝优化来降低耗费在"挖洞"尝试中的运行时间. 该优化过程如图 5 所示.



(a)　(b)　(c)　(d)　(e)

图 5　剪枝优化过程图示

假设我们刚完成了位于(行 1，列 1)的"挖洞"，在此将空格用 0 表示(见图 5(a)). 接下来我们按顺序尝试抹去(行 1，列 2)格子中的数字 2(见图 5(b))，随后我们通过求解器发现抹去数字 2 后的数独题至少有两解. 既然抹去数字 2 后留下的题是多解的，我们论定再抹去更多的格子(如再抹去(行 2，列 1)中的数字 3)后留下的题(见图 5(c))仍是多解的. 因此，我们保留(行 1，列 2)中的数字 2，继续进行随后的"挖洞"尝试. 接着我们尝试抹去(行 2，列 1)中的数字 3(见图 5(d))，而后发现抹去 3 后的题仍有唯一解. 按照先前的描述，我们应该对下一个格子进行"挖洞"尝试. 此时，我们禁止选取曾经尝试过的格子，(行 1，列 2). 因为一旦(行 1，列 2)中的数字 2 再次被抹去，则将再次回到图 5(c)中多解的局面. 因此，我们限定成功抹去数字 3 后，禁止选取曾经被尝试过的格子(行 1，列 2)和已经被"挖"成空格的格子(行 1，列 1)作为下一个"挖洞"尝试格. 综上可知，我们规定初始终盘上的每个满格只允许做一次"挖洞"尝试，即我们至多需要做 81 次"挖洞"尝试，而且任何空格将不会被重填，这一点不同于现有的"回溯"思想.

为了论证以上剪枝优化技术的有效性，我们做了大量的对比实验，以生成一道骨灰级难度的题所需要的运行时间为标准，得到统计结果如表 3 所示，从而说明了引入剪枝优化可以大大降低出题的运行时间.

表 3　剪枝对运行时间的影响

是否加入剪枝优化	平均运行时间
没有剪枝	55106.23ms
加入剪枝	1088.35ms

操作 5　等效变换

为了增加所出题目的多样性，我们对完成以上操作后得到的数独题进行如下 4 种等效变换(变换后不违反游戏规则).

变换 1　互换任意两种数字的位置(例如将格盘中所有的"1"和所有的"9"互换位置).

变换 2　整体互换任意两个小九宫格行或小九宫格列(例如行 1、行 2 和行 3 组成一个小九宫格行整体)

变换 3　在同一个小九宫格行(列)中重排(互换)行(列).

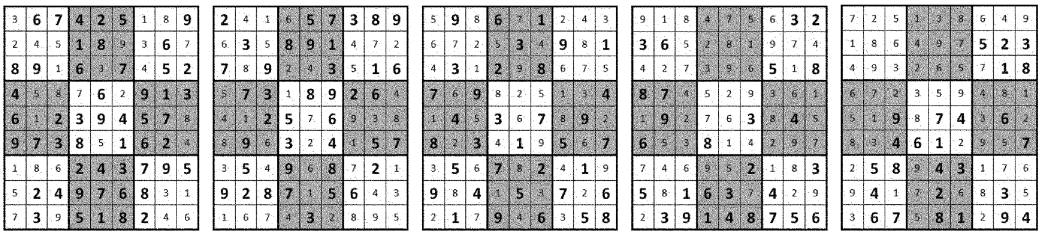
变换 4　全盘行、列互换，即矩阵转置.

6　结　　果

运用以上建立的生成算法，我们得到了五种不同难度等级的数独题如图 6 所示：

7　复杂度分析

我们建立的数独题生成算法的数据存储空间是可忽略的. 因此我们着重分析时间复杂



等级一：低级　等级二：初级　等级三：中级　等级四：高级　等级五：骨灰级

图 6　结果示例(粗体为已知格)

度,看其是否能在游戏者可忍受的时间范围内生成所需要的题.

在整个生成算法中,随机生成终盘只耗费常数级时间,可表示为 $O(1)$. 而在"挖洞"算法中,我们知道加入剪枝优化后的算法只需要做至多 81 次"挖洞"尝试,这其中调用了深度优先搜索求解器. 深度优先搜索的时间复杂度为 $\Theta(V+E)$[5]. 而我们粗略估计整个数独求解问题的复杂度 $\Theta(V+E)$ 会低于 1500000.

8 结　论

为了生成多难度等级的数独题,我们建立了数独题难度评定准则,并借用"挖洞"思想设计了数独生成算法. 现将本文中所提出的"挖洞"新机制总结如下:

结论 1　从左到右、至顶向下的"挖洞"顺序有助于生成高难度等级的数独题;而全盘随机"挖洞"有利于生成低难度等级的题目.

结论 2　运用"反证法"来验证数独题解的唯一性比遍历搜索所有解更为简便和快捷.

结论3　通过避免"回溯"到一个曾经尝试过的格和"重填"一个空格,可以大大降低算法的运行时间,这便是我们引入的剪枝优化技术.

与现代优化算法,如"遗传算法"相比,我们基于"挖洞"思想的数独生成算法耗费了较少的运算时间和存储空间. 下面给出我们的算法和文献[6]所用的"遗传算法"在求解同一组数独的对比实验. 文献[6]在试验部分求解了来自网站[7]所提供的 27 道数独题(共 9 级不同的难度,每个难度等级 3 个题) . 我们使用 2.66G Hz Pentium 4 处理器的PC 机同样求解了这些题目. 结果我们的算法只用了 4.509 秒,平均每题耗时为 0.167 秒. 而文献[6]的"遗传算法"(3.0 GHz Pentium 4) 求解这些题却用了 111 秒,平均每题 4.11 秒[6]. 在数据存储方面,我们的算法只需要存储 5 个 9×9 的二维数组. 而文献[6]的"遗传算法"在每次迭代中需要存储 21 个"染色体",每个"染色体"长度为81 个单位,这还未包括其余在"交叉"、"变异"操作中的临时存储. 所以可以看出,我们的算法在运算时间和存储空间上要优于文献[6]所提出的"遗传算法".

参考文献:

[1] Wei-Meng Lee. Programming Sudoku (Technology in Action) [M]. Apress, 2006.

[2] Bertram Felgenhauer and Frazer Jarvis[EB/OL]. http://www.shef.ac.uk/~pm1afj/sudoku/.

[3] Gordon Royle. Minimum Sudoku[EB/OL]. http://people.csse.uwa.edu.au/gordon/sudokumin.php

[4] Alsuwaiyel M H. Algorithms Design Techniques and Analysis [M]. London: World Scientific Publishing Company, 1998.

(Seccond Edition)[M]. Cambridge, USA: The MIT Press, 2002.

[6] Timo Mantere, Janne Koljonen. Solving, rating and generating sudoku puzzle with GA[C]// IEEE Congress on Evolutionary Computation, 2007, 25-28, 1382-1389.

[7] Aamulehti: Sudoku online[EB/OL]. http://www.aamulehti.fi/sudoku/

Sudoku Puzzles Generating: From Easy to Evil

XUE Yuan-hai, JIANG Biao-bin, LI Yong-zhuo
Advisor: YAN Gui-feng, SUN Hua-fei

(Department of Mathematics, Beijing Institute of Technology, Beijing 100081, China)

Abstract: Sudoku puzzle becomes worldwide popular among many players in different intellectual levels. The task is to devise an algorithm that creates Sudoku puzzles in varying level of difficulty. With the analysis of the game rules, we first define the difficulty level from four aspects as: total given cells, distribution of given cells and complexity of enumerating search.

By the guidance from the definition of difficulty level, the algorithm for generating puzzles is developed with the "dig-hole" strategy on a valid grid. Thus, the algorithm developed in two steps: to create a valid grid by Las Vegas algorithm, and then to generating puzzles by erasing some digits using five operators:

- Determine a sequence of digging holes according to the desirable difficulty level,
- Set two restrictions to guide the distribution of given cells,
- Judge whether a puzzle being dug out has a unique solution by a solver built using Depth-First Search,
- Add pruning technique to avoid digging an invalid cell, and
- Perform propagating at a dug-out puzzle to raise the diversity of the output puzzle.

Using our developed algorithm, we generate Sudoku puzzles in any five difficulty levels. The difficulty level of output puzzles can be adjusted by a desirable difficulty value input by players. The complexity of the algorithms in space and time is analyzed to demonstrate the effectiveness of the algorithms.

Our main contributions in exploring the "dig-hole" strategy are summarized as following three works: to do a massive research on the sequence of digging holes and how it affects the algorithm to create a evil-level puzzle with minimal given cells, to invent a skill for judging the solution's uniqueness of a puzzle being dug out by the reduction to absurdity, and to reduce the computational time by avoiding backtracking to an explored cell and refilling an empty cell.

Keywords: dig-hole strategy; las vegas algorithm; pruning; reduction to absurdity