AI 课程大作业: 数独求解与生成 制作报告

物理系 基科31 蒋文韬 2013011717

2015 年 7 月 4 日

1 Tasks

1.1 构思与分工

我们小组由张思源，李泽清，蒋文韬三人构成，均来自物理系基科31班。我们的选题是数独的求解与生成。求解部分大部分为使用搜索算法，考虑到整门课程内容丰富，因此我们打算加入手写与图片识别的功能，这样大作业覆盖的知识点更加全面，程序功能也更加丰富多样。

我们的分工情况大致如下：小组成员三人共同设想，讨论与改进游戏的关键算法及其代码实现。蒋文韬主要负责游戏本身有关算法与代码的实现；张思源主要负责游戏程序界面的设计，制作与实现；李泽清主要负责图像识别部分功能的代码实现。最终，数独求解与生成算法部分全为自己编写，代码约3000余行；游戏程序界面使用MFC实现，代码约8000余行；图像识别部分使用了OpenCV，代码约1000余行。

1.2 数独问题的求解与生成

数独问题的求解与生成部分首先实现了所需的描述当前数独游戏状态以及获取一些相关信息的类，并实现了所需的队列，堆栈等结构。随后根据数独游戏的特点建立搜索树。在建立搜索树的过程中我们发现了一些优化方案并进行了比较与实践。

在数独游戏的生成方面，我们的大体思路是先随机生成一个完整的数独，再在保证唯一性的前提下去掉一些已知的数字，通过去除到不同程度来控制最终数独游戏的难度。

1.3 手写与图片输入的处理

这部分代码主要由李泽清完成，主要通过kNN算法[1]，分别实现了手写单个数字的识别与输入，以及对一张完整数独图片的识别与输入。

1.4 GUI界面的设计与实现

这部分内容主要由张思源完成，将数独问题的求解与生成，以及手写与图片识别等功能整合到一起，并利用MFC实现了较好的使用体验。

[1]即K最近邻(kNN，k-NearestNeighbor)分类算法

2 代码实现与运行结果分析

2.1 数独问题的求解

2.1.1 利用广度优先与深度优先算法求解

较为简单的数独，利用逻辑推理能够较快地直接得到答案，但难度较大的数独，往往需要十分复杂而难度较大的推理。因此，我们在一开始便将思路确定为利用搜索算法解决数度求解问题，而逻辑推理则成为辅助的部分。

实现搜索算法首先需要一些数据结构，如队列与堆栈。编写好这些结构以及描述数独游戏的类后，下一个问题就是搜索树的建立，即从当前的数独问题，确定下一步对什么情况进行尝试。很自然的想法便是首先选出当前数独问题中可能填入的数字最少的位置，然后这些可能的填法便构成当前节点的子节点。搜索到无效节点即当前数独游戏中出现较为明显的矛盾，而搜索停止的条件即为数独游戏完成。因此，我实现了搜索当前数独中所有待填写位置中可能填入数字最少的函数，返回当前数独中某位置可能填入的所有数字的函数等实现相关支撑功能的函数。

利用队列与堆栈的结构，便可实现广度与深度优先算法。容易看出，深度优先算法即可以找到一个解为目标，而广度优先算法即可找出所有解。因此我们采用深度优先算法对数独问题进行求解，而采用广度优先算法判断当前数独问题是否有唯一解。不过实际测试中我们迅速发现，如果给求解算法一个空白的或是解的个数非常多的数独问题，即会得到一个十分庞大的搜索树，这时如果仍采用广度优先算法确定该问题是否唯一则毫无疑问是不现实的。稍加思考后，我们用找到解后仍继续进行搜索的深度优先搜索算法解决了这一问题。在确定唯一解这方面，持续搜索至穷尽搜索树的深度优先搜索的规模与广度优先是相当的，然而由于数独问题的特点以及我们建立搜索树的方法，深度优先即为求解优先，而通过控制搜到的解的数目来停止继续搜索即可回避开搜索树过大导致的上述问题。因此，我们采用若搜索到10个解则停止搜索的深度优先搜索算法代替了广度优先的搜索算法解决了上述问题。

在运用逻辑推理进行填写方面，我们发现尽可能多地填入能够通过逻辑推理得到确定的数字能够十分有效的减小搜索树的规模，特别是在难度较大的数独问题中。以下是一个例子：

推理方法	直接搜索	使用数独基本规则	使用行列块排除
展开的节点数	8024	899	119

可以看出，使用较为简单的逻辑推理，都能使搜索的节点数有近乎数量级上的减少。因此我们打算深入探究并实现一些较高级的逻辑推理方法。

2.1.2 Bilocation Graph

在经过一些文献调研与利用网络资源进行查找后，我发现了一些数独爱好者的网站上有相当众多的高级逻辑推理求解数独的方法[2]，而就数独问题展开讨论的学术文章也为数不少。其中出现较为频繁的Bilocation Graph的概念吸引了我的注意，因此我阅读与理解了与之相关的几个逻辑推理方法，并将其进行了程序实现。

所谓Bilocation Graph，即为从一个数独问题中建立出的一个无向图，其点为满足以下条件的空白位置，条件为：与该点相连边的值为可能填入该边相连的两点中非此即彼的某一个的数。下图为一个例子：

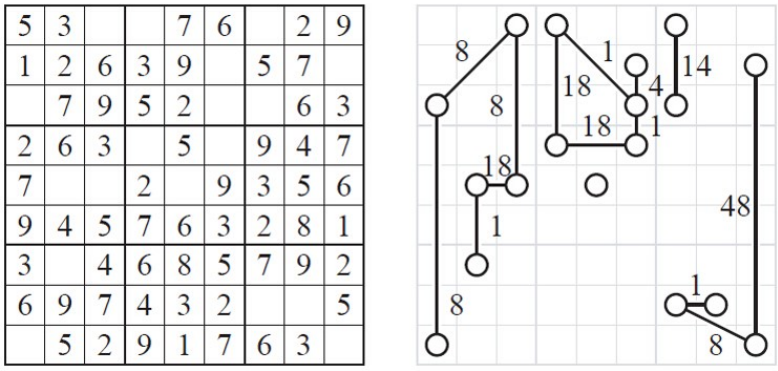

图 1: Bilocation Graph[1]

可以看出，在左图的数独问题中，对第一列中空白的两个位置，均仅可能填入4或8，因此该两个位置满足以上条件，即该两点间可连两条边，边值分别为4与8。右图即为由该数独问题生成出的Bilocation Graph的一部分。由于Bilocation Graph中边的数值仅可填入该边相连两点中非此即彼的特点，根据该图的一些特征即可能推理出一些确定的数值，如以下两种方法。

Repetitive cycle rule:

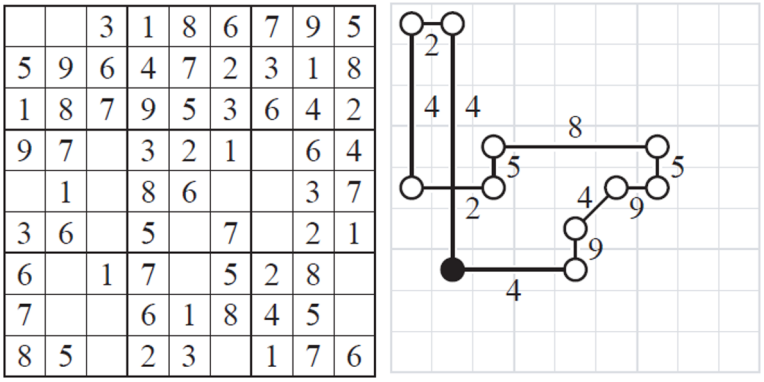

图 2: Repetitive cycle rule[1]

在左图的数独问题中，通过生成出的Bilocation Graph，我们找出如右图所示的一条回路。该回路的特点是除一点外，其余各点前后相邻的边边值不同，而仅一点前后两边边值相同。由Bilocation Graph中边值填入相连两点中非此即彼的特点，我们观察与两个边值为4的边相接的点，图中已标黑。该两边上的4要么均填入黑点所处位置，要么填入各自相连的另一点中。而若填入各自相连的另一点中时，填入4的该两点即可确定分别与之相连的“2”与“9”仅可继续填入之后的点中。依此递推，即回在回路中推得矛盾，因此可得4仅可填入黑点所处位置中。

由上述讨论即可看出，如果我们在Bilocation Graph中找到了一条上述特点的回路，则该回路中与相同边值的边相连的点处应填的数即可确定。

Conflicting paths rule:

与Repetitive cycle rule类似的，观察下图的数独问题与从该数独问题生成出的Bilocation Graph中选出的路径。

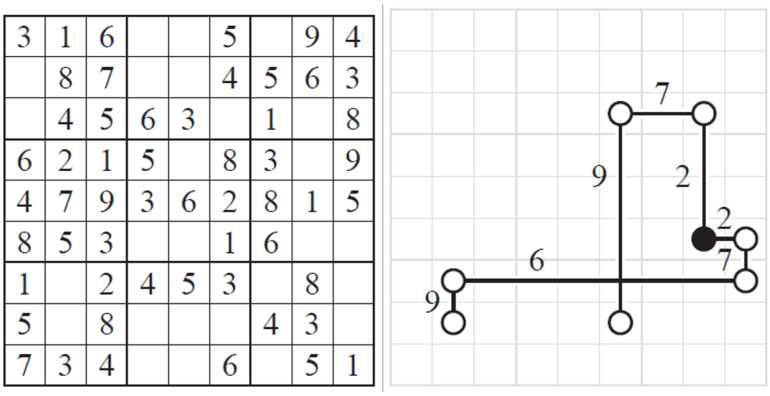

图 3: Conflicting paths rule[1]

右图中路径的特点除了与Repetitive cycle rule中的回路相似的两点之外，其首尾两点处于同一行、同一列或同一块中，且与二者分别相连的边的边值相等。对黑点进行完全类似的讨论，可以看出，若2不填入黑点所处的位置，则递推下来可得该路径的首尾的“9”均需填入首尾两端的点中，而该两点处于同一行中，出现矛盾。因此，可得2仅可填入黑点所处位置中。

由上述讨论即可看出，如果我们在Bilocation Graph中找到了一条上述特点的路径，则该路径中与相同边值的边相连的点处应填的数即可确定。

在上述讨论的基础上，我实现了通过邻接表描述无向图的一个类，并在此基础上实现从一个数独问题生成出其对应的Bilocation Graph。在该Bilocation Graph的基础上即可进行路径搜索算法的实现。通过选择出所有可能满足黑点条件的点作为起始点(即Bilocation Graph与相同边值的边相连的点)，将与之通过相同边值相邻的二点分别作为起点与终点以路径上相邻边边值不同为条件进行搜索，即可实现Repetitive cycle rule。

而Conflicting paths rule的实现则较为复杂。我起先采用类似Repetitive cycle rule的实现方法，先寻找可能满足黑点条件的点作为起始点，再向两端搜索至找到满足首尾两点条件时停止。但程序运行并不正确。由于没有较好的可视化方案，在图上的搜索过程极难进行监控并找出bug，因此我更换思路重新写了一种算法，即从任一点作为起点出发，不加上路径上相邻边边值不同的条件，而当已搜索的路径上出现相同边值的边时再加上该条件，最后仍以首尾两点满足条件时结束，否则更换起点。这样实现出的程序运行仍旧有误。由于debug实在困难，因此最终放弃了对Conflicting paths rule的实现。

2.2 数独游戏的生成

在数独游戏的生成方面，我们的大体思路是先随机生成一个完整的数独，再在保证唯一性的前提下去掉一些已知的数字，通过去除到不同程度来控制最终数独游戏的难度。

为防止每次DFS得到的结果雷同，我们事先随机填入5 ~ 8个没有矛盾的数字，再通过DFS得到一个完整的数独。通过去除到不同程度，我们控制出以下5个难度：

- ○Naive : 控制空白个数：35~40个
- ○Easy : 逐个去掉数字至无法根据基本的游戏规则完成数独
- ○Normal : 逐个去掉数字至无法根据行列块的排除法推导完成数独
- ○Hard : 逐个去掉数字至无法根据行列块的排除法以及Repetitive cycle rule完成数独
- ○Sometimes Naive : 逐个去掉数字至刚好保证唯一性

下面是生成出的各难度的数独问题的示例：

		1	4		8	6		9
	4	9			6		5	
6		2			5	8	4	3
7		8	6			5		
4	9	3	5	8			6	1
			1	7	4		3	
3	2	5						
				5		2		6
9		7	2	4	1	3		

图 4: Naive

3	2	1		7		5	6	
				3	1		4	
		4		6	2	9	1	3
	7							4
6		8			3			
2		3			5			7
5		2	4			1		
			1		6			9
1	6		3					

图 5: Easy

3	2	1		7	5	8		
		4			8		3	1
		8					7	9
		6		8	9			
1							2	
	9	2		5			4	6
				9	1			
	6							
2					6			

图 6: Normal

	5		8	4	6			
7		2		1		8		
				9	4			3
4	3	7			5		2	
			2			7		
2								
1			4		9			
8				6			4	7

图 7: Hard

	2			7				4
	4	1	9					
9	8			3		5		
						7	6	
					1			
	9			5				
						4		6
	1		2					
	6	4				8		5

图 8: Sometimes Naive

最终生成出的游戏难度大致符合设置的难易程度。需要一提的是，数独的难度大致随空白个数的增加而增加，但并不完全是这样。另一方面，上述难度划分方式更适用于我们编写的数独求解程序，而对玩家则不一定较为相符。因此我们的数独游戏生成方案的难度划分也仅可作参考一用，还有值得改进之处。

3 总结与感想

这次人工智能导论课程大作业，我们三人组成小组，将任务制定并分配为数独的求解与生成，手写与图像识别，以及程序界面这三个部分，并在共同讨论关键算法，解决所遇困难，并互相整合工作的基础上分工完成。

由于编程经验与水平的不足，我们在完成程序的过程中遇到了不少困难，也从中收获良多。如Open CV环境的设置与运用，MFC的使用，均为实际工作中遇到的与程序本身功能与算法关系不是十分紧密但对完整程序的完成来说必不可少的部分。这也是编写如课程小作业等小程序与编写一个有完整功能且使用较为方便的程序不同的地方。而我这部分程序出于自己想练练的目的没有使用任何现成的代码，因此测试过程中发现了十分严重的内存泄漏，最终花费了若干个下午逐渐解决，通过这一问题让我更加清晰地认识到自己编写的一些容器的架构的不合理之处，对合适的问题采用合适的结构等问题，以及使用已有的被较广泛使用的代码在可靠性方面的优势。

而通过这次大作业，我们对人工智能课程的知识点也有了更进一步的认识理解与实际应用的经验，如用C++语言实现堆栈，队列等结构后再实现搜索算法；运用Open CV通过kNN算法实现手写与图片的数字识别等。而整个大作业中遇到的诸多问题也还是有尚未解决的，如2.1节中提到的可视化困难带来的debug困难，以及最后用Visual Studio编译出的可执行文件无法单独运行这一可能因涉及一些环境参数而导致的问题。人工智能这一领域的问题与方法实在过于丰富，这次大作业也仅是冰山一角，这学期的课程也仅是一个开始，在以后的学习与工作中，通过这门课程与作业所理解的一些知识与所熟悉的一些方法与经验也必将派上用场。

参考文献

[1] David Eppstein. Nonrepetitive Paths and Cycles in Graphs with Application to Sudoku, 7 2005.

[2] Andrew Stuart. sudokuwiki.org - Strategy Families, 4 2008.