深 圳 大 学 实 验 报 告

课程名称：算法设计与分析

实验名称：回溯法地图填色问题

学院：计算机与软件学院 专业：软件工程

报告人：郑杨 学号：2020151002 班级：腾班

同组人：陈敏涵

指导教师：李炎然

实验时间：2022.5.2~2022.5.9

实验报告提交时间：2022.5.9

教务处制

目录

一、实验目的

1、掌握回溯法算法设计思想
2、掌握地图填色问题的回溯法解法

二、实验内容

1、背景知识：
为地图或其他由不同区域组成的图形着色时，**相邻国家/地区不能使用相同的颜色**。尽可能用少的颜色种类进行填涂，可减少填图工艺复杂性。简单的“地图”（例如棋盘）仅需要两种颜色（黑白），但是复杂的地图需要更多种颜色。
2、问题描述：
我们可将地图转换为平面图，每个地区变成一个节点，相邻地区用边连接，我们要为这个图形的顶点着色，并且两个顶点通过边连接时必须具有不同的颜色。附件是给出的地图数据，请针对三个地图数据尝试分别使用 5 个（le450_5a），15 个（le450_15b），25 个（le450_25a）颜色为地图着色。
3、小规模地图如图1所示，利用四色填色测试算法的正确性；

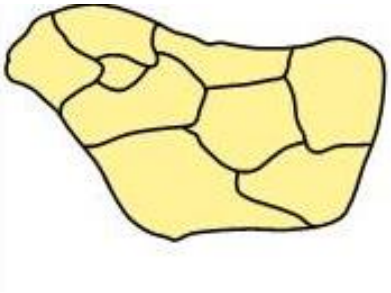

图1：小规模地图

4、对附件中给定的地图数据填涂；
5、随机产生不同规模的图，分析算法效率与图规模的关系（四色）。

三、实验步骤与结果

1 问题分析

地图涂色问题本质上是一个约束满足问题，接下来首先对约束满足问题的形式化描述做一个简单介绍，再对地图涂色问题做出形式化表示。

1.1 约束满足问题

约束满足问题被定义为一个三元组$<X,D,C>$，其中

$X=\{X_1,...,X_n\}$是变量的集合，

$D=\{D_1,...,D_n\}$是每个变量的值域集合，

C是约束的集合，也就是变量所需满足的限制条件的集合。

此问题的解定义为$Y=\{Y_1,...,Y_n\}$，满足$Y_i \in D_i$且满足C中的所有约束。

现实生活中有许多问题都可形式化描述为约束满足问题，例如以下找零钱问题：

用无限个面值为$d_1>d_2>\cdots>d_n$的最少数量的硬币找出金额为n的零钱。

该问题可以形式化为以下约束满足问题，定义三元组$<X,D,C>$为：

$X=\{X_1,...,X_n\}$，其中X_i表示面值为d_i的零钱所使用的数量；

$D=\{D_1,...,D_n\}$，其中$D_i=\{0,1,\ldots\}=N$

$C=\{\sum_{i=1}^{n}X_i d_i\}$

那么原问题的一种解法就是把该约束满足问题所有解的$\sum_{i=1}^{n}Y_i$进行比较取出最小值。

1.2 把地图着色问题表示为约束满足问题

以图 1 中的小规模地图为例，首先对该地图上的每个地方进行编号得到图 2，如下：

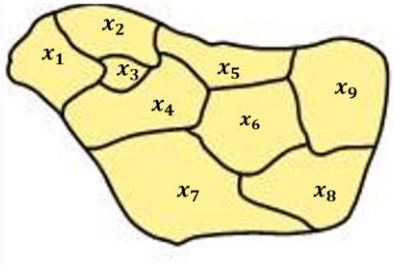

图 2：小规模地图（已编号）

那么图 2 中的小规模地图涂色问题（四色）可以形式化描述为以下约束满足问题：

$X=\{X_1,...,X_9\}$，其中X_i表示地图上x_i地区所涂的颜色；

$D=\{D_1,...,D_n\}$，其中$D_i=\{1,2,3,4\}$四种颜色；

$C=\{X_1 \neq X_2, X_1 \neq X_3, \cdots, X_8 \neq X_9\}$，所需满足的约束就是相邻的地区不能涂相同的颜色。那么目标就是找到所有满足约束C的解。

1.3 约束图

在将图着色问题形式化描述为约束满足问题之后，可以利用一种特殊的数据结构——图来进行建模，进一步表示为约束图。依然以小规模地图为例（图 1 和图 2），可以**把每一个地区抽象成无向无权图中的点**，把每一个**约束抽象成无向无权图中的边**，可以得到如图 3 的约束图：

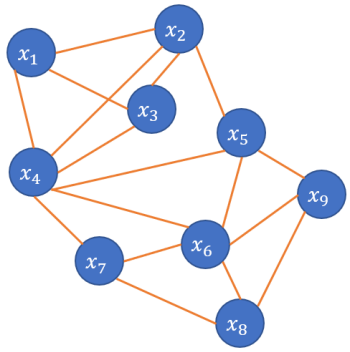

图 3：小规模地图的约束图

转化为约束图问题之后，就可以用图搜索算法对问题进行求解了，将在下一节展开叙述。在地图涂色问题中，若约束图存在多个连通分量，那么连通分量之间相互独立，可以拆解为多个子问题进行求解，所以这里只需要考虑单一连通分量的求解即可，即分析**连通的约束图**问题。

2 回溯法

2.1 算法思想

回溯法是求解约束满足问题最基本的无信息搜索算法，它包括变量分配、约束检查和回溯搜索等步骤。最朴素的回溯算法在约束图问题上的具体做法就是，以任意顶点为起始点任意分配颜色，检查是否满足约束，若当前节点无法分配颜色使得约束条件满足，则需要回溯到上一步撤销已分配颜色之后重新分配颜色。

2.2 算法流程

在之前已经假设约束图连通，那么算法流程如下：

1) 搜索当前节点的所有可行解，进入 2)
2) 若当前节点存在满足约束条件可行解（该可行解未被搜索过），那么为当前节点赋值任意一个可行解，进入 3)；否则回溯到上一节点撤销已赋值的可行解并进入 1)，若已无法回溯，则说明已搜索完所有的可行解，算法结束。
3) 若当前节点的后续节点存在，则扩展其后续节点，进入 1)；若当前节点不存在后续

节点，则进入 4)。

4) 成功找到一组解，记录该解并回到 1)。

算法流程图如图 4 所示：

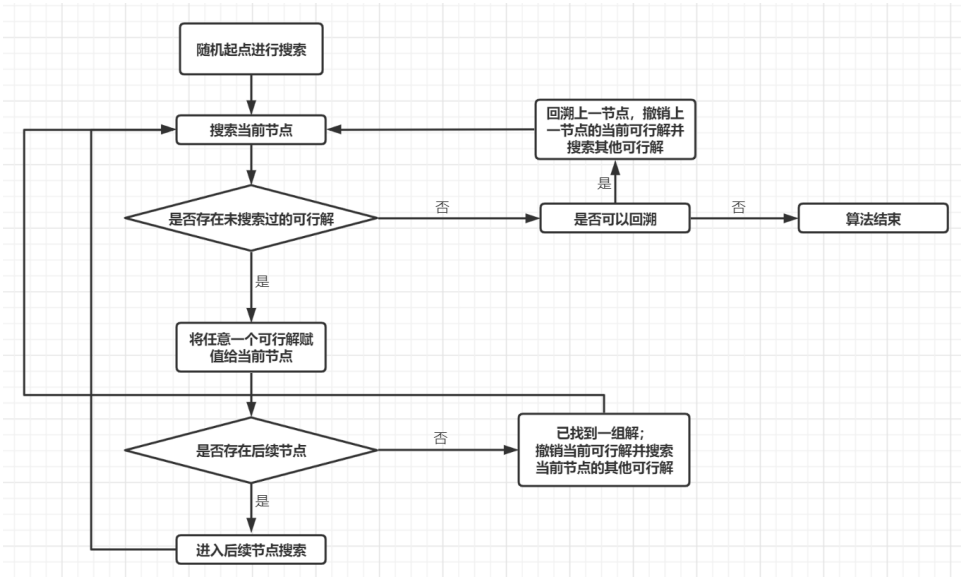

图 4：回溯算法流程图

接下来以一个具体的例子阐述回溯法的流程：

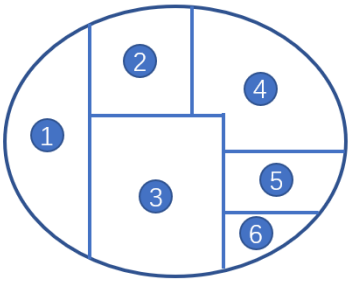

图 5：回溯法举例

使用三种颜色（红黄蓝）对图 5 例子进行染色，首先把图 5 地图表示成如图 6 所示约束图的形式。

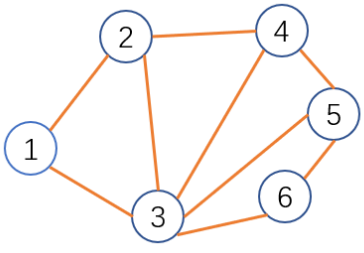

图 6：图 5 约束图

回溯法执行过程如下，首先以 1 为起始节点，对 1 赋值一个可行解（目前可行解有红黄蓝）红色，进入下个节点 2 进行搜索直至第 6 个节点，第一轮搜索不会产生回溯，第一个解如图 7 所示。

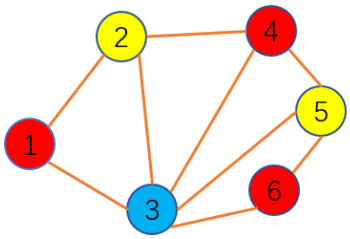

图 7：回溯法流程（1）

当算法找到第一个解时，会把节点 6 的当前可行解（红色）撤销，继续搜索节点 6 的可行解，发现不存在其他未赋值过的可行解，于是进行回溯，撤销节点 5 的当前可行解（黄色），发现节点 5 并未其他可行解，继续回溯到节点 4，仍无其他可行解，于是继续回溯到节点 3，仍无其他可行解，直到回溯至节点 2，发现节点 2 此时有第二个可行解（蓝色），故类似往下搜索可以得到第二个解，如图 8 所示。

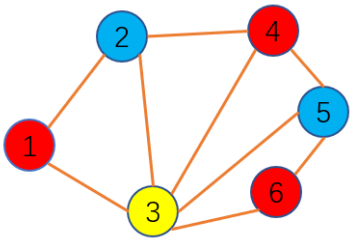

图 8：回溯法流程（2）

其他流程也是类似的操作，这里就不再赘述。

2.3 数据结构选择

回溯法执行过程中，需要频繁判断节点当前所涂颜色是否与其相邻节点冲突。在建图的时候，常用的两种方法是建立邻接矩阵与邻接表。这里采用邻接表的形式，因为可以快速遍历某个节点的相邻节点，运行时间的常数会比邻接表的形式小（邻接表对于检查某个节点需要遍历一遍所有节点）。

2.4 算法正确性验证

在如图 1 图 2 图 3 所示的小规模地图数据上做了简单回溯法的正确性测试。具体的测试方法为：对于每一个上述算法求解出来的可行解，遍历约束图中的所有边判断边的两个顶点颜色是否不同以验证正确性。正确性测试代码如图 9 所示，由于使用的是邻接表存储图，需要遍历所有节点之后找到连接某个节点的所有边。

```
for (int i = 1; i <= n; i ++ )
    for (int j = 0; j < edge[i].size(); j ++ )
    {
        int v = edge[i][j];
        if (col[i] == col[v])
        {
            right_test = false;
            break;
        }
    }
```

图 9：回溯法正确性验证代码

代码运行结果如图 10 所示：

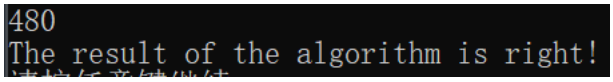

图 10：回溯法正确性验证结果

注：此正确性验证也适用于下一节的所有优化，个人也在优化之后进行了正确性验证，均可通过。

2.5 算法复杂度分析

对于节点规模为n，颜色规模为m的问题，在朴素的回溯法中，需要枚举每一个节点的每一个颜色之后进行判断，可以画出如图 11 所示的解空间，其中边表示枚举的每一种颜色，点表示所有枚举情况下的节点。可以看出，解空间是一颗$n+1$层的m叉树，于是解空间的大小为：

$$\sum_{i=0}^{n} m^i = \frac{1-m^{n+1}}{1-m} = \frac{m^{n+1}-1}{m-1}$$

故朴素回溯法的时间复杂度为：$O(m^n)$

可见算法复杂度呈指数增长，故下一节会对算法添加优化以提升性能。

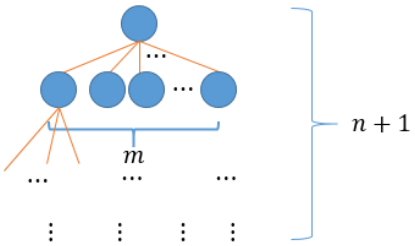

图 11：复杂度分析示意图

2.6 剪枝前的小优化

在对回溯算法进行剪枝之前，需要先提一下一些代码上的小优化。在之前的回溯算法中，对于每个结点需要枚举所有颜色之后进行可行性判断，这造成了许多不必要的枚举（可能枚举的颜色在这个结点相邻结点出现过）。本人的想法是，记录每个结点当前**可用的颜色**以及**颜色数**，每次枚举只需要枚举可用的颜色即可。这在下一节剪枝策略中也会提到。

2.7 算法效率实验测试

朴素回溯算法解决较大规模问题（如给定的第一个地图数据 le450_5a）求解效率非常低下，在有限时间内连一个解都难以跑出来。于是只能在小规模地图（图 1 所示地图）上进行效率测试，对于给定的三个较大规模地图数据只能优化之后再进行实验效率测试。朴素回溯算法在图 1 所示的小规模地图上运行 20 次之后的平均运行时间如表 1 所示：

表 1：朴素回溯算法在小规模地图下的平均运行时间

地图	平均运行时间(s)
小规模地图	0.082

3 剪枝策略

由于回溯法的搜索空间呈现一颗搜索树的形式，于是可以通过一些方法，减少搜索树一些没必要的搜索分支以提升算法性能。

3.1 推理策略-前向检查

前向检查的优化思想就是在为当前节点染色之后，检查是否存在某个相邻节点的值域为空（注意这里不需要检查所有节点，因为当前节点只会影响其相邻节点的值域），若存在某个相邻节点值域为空，则当前染色方案终会失败，可以提前回溯。仍以图 5 数据作为例子进行说明，具体过程如图 12 所示。若当前染色节点为 4 号节点，检查其相邻节点的值域发现 3 号节点的值域为空，故此时直接撤销 4 号节点的已染颜色，重新搜索可行解。

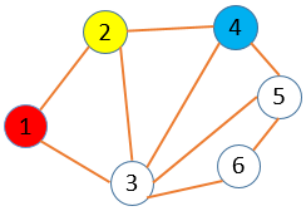

图 12：前向检查示例

3.2 排除等效冗余-颜色轮换

之前的三种剪枝优化策略都是针对于**减少不必要的搜索**进行优化的，这里介绍一个针对**等效冗余**的优化策略。该策略的出发点是颜色布局的冗余性，就是说有许多种染色方案本质上的布局是一致的，如图 13 例子所示，六种染色方案的布局一致，只是颜色的排列不同。那么只需要找出所有的颜色布局，再将结果乘上$c!$即可（c 为颜色数）。

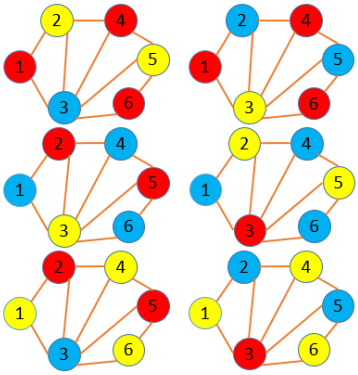

图 13：颜色轮换示意图

在具体实现上，可以把原问题拆分为若干个子问题进行求解。在算法执行过程中，我们记录当前未被使用过的颜色个数与每个颜色被使用的次数，把当前节点可涂颜色中的所有未使用过的颜色看成等效颜色，只需对其中一种颜色计算方案数之后乘以颜色数；而对于当前节点可涂颜色中的所有使用过的颜色，就需要分别计算方案数之后累加。具体例子仍以图 5 地图数据为例，过程如图 14 所示。这里假设节点按照编号从小到大的顺序进行扩展，在节点 1 时，未使用的颜色数为 3，于是只需要给节点 1 染上三个颜色中的任意颜色计算一次然后乘以 3 即可得到所有方案数。接下来到达节点 2，此时未使用颜色书为 2，于是只需要给节点 2 染上两个颜色中的任意颜色计算方案数，之后乘以 2 即可得到**以节点 2 为起始节点的子问题**的所有方案数。到达节点 1 的时候，只剩下一种颜色未使用过，直接正常计算即可。往后的节点也是类似的道理，最后返回到 2 节点时，只找到了 1 种方案，故以节点 2 为起始节点的子问题方案数为 1*2=2。之后回到节点 1，由于子问题方案数为 2，故以 1 为起始节点的原问题的方案数为 2*3=6。

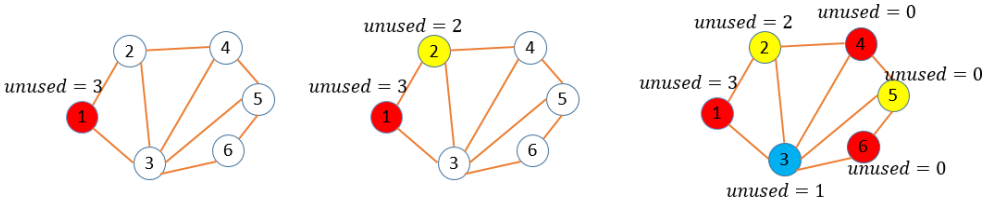

图 14：颜色轮换示意图

3.3 搜索顺序-最少可取值优先（MRV）

搜索顺序是回溯法中影响算法效率的一个重要因素，按照不同的顺序对节点进行染色，所长成的搜索树形状不是固定的，其节点与分支数量都有很大的差异。例如对于图 5 的例子而言，第一个染色变量若为 1 号节点，则生成的搜索树如图 15 左边所示，而若以 3 号节点为第一个染色变量，则生成的搜索树如图 11 右边所示。（假设同时以 4 号节点为第二个染色变量）可以看到，右边的搜索树第三层比右边的分支树少 3。这只是一个简单的例子，而在更复杂的实例中，类似这样的差异会更大。

故优化的目标是：**确定一个合理的搜索顺序使得搜索树的节点数与分支树尽可能少**。

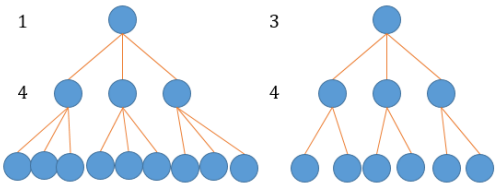

图 15：搜索顺序不同对搜索树规模的影响

在这一小节中，提出的一种搜索顺序的优化是优先搜索可取值最少的节点（MRV 准则），也即**优先选择可填颜色最少的节点进行扩展**。这样做的原因是：优先选择可填颜色少的节点相当于是最小化了当前节点的分支个数，使得当前节点受其他节点的约束最大，能够更快的检测出失败的情况，从而避免没有必要的搜索。具体例子如图 16 所示，以优化前和优化后搜索图 5 第一个解的搜索树对比为例，左图为不按照 MRV 准则进行搜索的搜索树，右图为添加了 MRV 准则优化后的搜索树。可以看出，MRV 准则明显优化了搜索树的规模。

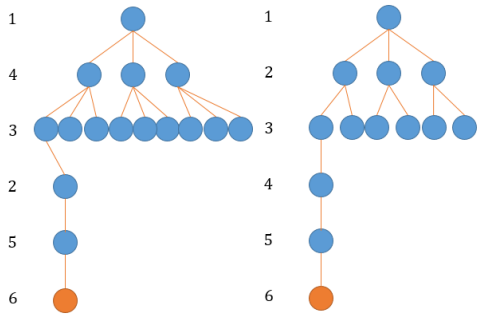

图 16：MRV 准则举例

3.4 搜索顺序-最大度数优先（DH）

最大度数优先的准则与 MRV 准则的思想一致，都是想让节点所受约束尽可能大以尽快排除不必要的搜索分支。DH 准则的意思就是优先选取度数大的节点进行搜索，在 MRV 准则的前提下，利用 DH 准则可以**使得当前节点的分支数量尽量少，且使得后续节点的分支数量也尽量少**，因为度较大的意思就是当前节点的相邻节点较多，也就是说使得当前节点对其他节点的约束最大。下面同样用图 5 问题的搜索树作为例子来进一步说明 DH 准则的优化结果，添加了 DH 准则与 MRV 准则的搜索树如图 17 所示。一开始所有节点的可选择颜色数量都相同，故选择度较大的一个节点 3 进行扩展，之后的扩展也是类似的，可以看到整棵搜索树的规模确实变小了。

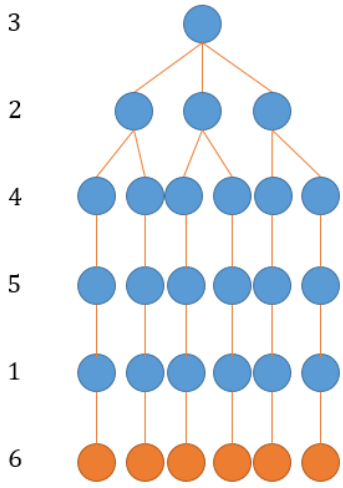

图 17：DH 准则举例

3.5 剪枝优化的实验测试

对比无优化的暴力回溯算法，MRV 和 DH 准则的应用使得算法解决较大规模问题效率得到了大幅度的提升。从原先难以找出一个解到如今在 1s 内找到所有可行解。一个数据测试小实验如表 2 所示，分别在图 1 所示的小规模地图数据上与给定的第一组地图数据（le450_5a）上跑了 20 次取平均运行时间。对于更大的地图数据（le450_15b 和 le450_25a），仍无法在有限时间内得出所有结果，于是只测试了搜索出**第 1 个解**与 **100 万个解**时所需时间（20 次取平均），测试结果如表 3 所示。

表 2：添加 MRV 与 DH 准则的回溯算法在数据集上求解所有解的平均运行时间

地图	平均运行时间(ms)
小规模地图	0.091
le450_5a	25631.646

表 3：添加 MRV 与 DH 准则的回溯法在数据集上求解 1 个解和 100w 个解的平均运行时间

地图	1解的平均运行时间(ms)	100w解的平均运行时间(s)
le450_15b	4.920	0.590
le450_25a	0.665	0.598

与表 1 的朴素回溯法比较可得，添加了 MRV 与 DH 剪枝之后的回溯算法已经得到了很好的优化，在朴素回溯法有限时间内无法求解出来的第一个给定地图数据（le450_5a）下，优化后的回溯算法可以在有限时间内（约 25s）求解出来所有可行解（共 3840 组），可以看到，可行解数量并不多，但解空间非常的庞大，朴素的回溯算法没有排除不必要的搜索，失败次数远远多于优化后的回溯算法，于是性能也有较大的差异。而在更大的数据集（le450_15b 和 le450_25a）上，至今优化后的回溯算法仍然无法在有限时间内得出所有可行解，故只测试了得出第一个解与第 100w 个解所需的时间，可以看出，在两个较大的数据集上，第一个解的求解速度有较大的差异，而第 100w 个解的求解速度却十分相似，个人猜测这与颜色数量有关，因为颜色较多时容易得到解，不容易失败。

可以看到，使用 MRV 准则与 DH 准则等类似的启发式优化虽可以产生一定效果，但是普遍性不强，算法稳定性不强。

在加上**颜色轮换优化**之后，算法速率将大幅提升，在第 4 节有对比实验验证这一结论。

4 实验测试

4.1 将所有优化策略结合起来后在三个数据上的测试

将不加颜色轮换优化前的回溯算法（仅使用 MRV 准则与 DH 准则）与加了颜色轮换优化后的回溯算法在给定的三个数据上进行对比测试。第一个数据由于可以在有限时间内跑出完备解，故测试两种算法求解完备解的平均运行时间；后面两个数据由于无法在有限时间内求出完备解，故测试两种算法求解第一个解所需的平均运行时间。测试结果如表 3 所示。可以看到，添加了颜色轮换优化后的算法运行效率比原来快了许多，在求解第一个解的问题上也有提升。

表 3：两种不同优化在不同数据集上的运行效率比较

算法	地图	平均运行时间(ms)
MRV+DH	小规模地图	0.091
	le450_5a	25631.646
	le450_15b	4.920
	le450_25a	0.665
MRV+DH+颜色轮换	小规模地图	0.006
	le450_5a	169.136
	le450_15b	3.518
	le450_25a	0.523

4.2 自拟不同规模数据进行测试

生成不同规模数据，使用添加了所有优化之后的回溯算法进行测试。

首先固定点数为 30 个点，随机生成了 10 组数据，边数从 40 开始递增，测试求解第一个解的平均运行时间，测试结果如表 4 所示。可以得出，边数对算法求解第一个解的效率影响并不大。表中所示的颜色数为最小的使得该图有解的颜色数，在所生成的图规模由稀疏到稠密变化时，所需的颜色数也在不断的增加，这是因为边数越多表示约束越大，约束越大则可行解难以求得，需要增加颜色数以得到可行解。

表 4：不同数据规模对求解第一个解的时间影响

数据规模			求解第一个解的平均运行时间（ms）
点数	边数	颜色数	
30	40	4	0.0128
	80	4	0.0143
	120	5	0.025
	160	7	0.0153
	200	7	0.0431
	240	11	0.0172
	280	13	0.0186
	320	15	0.0194
	360	19	0.0198
	400	22	0.0206

然后，固定边数为 45，随机生成了 4 组数据，点数从 10 开始递增，测试求解所有解的平均运行时间，测试结果如表 5 所示。可以看到，当点数增加时，由于图变稀疏约束减少，所需颜色数减少，求解时间变长且呈指数型增长。

表 4：不同数据规模对求解第一个解的时间影响

数据规模			所有解平均运行时间（ms）
点数	边数	颜色数	
10	45	10	0.004
15	45	5	0.421
20	45	5	624.234
25	45	4	7624.195

四、实验心得

1. 朴素的回溯法思想简单，它遍历了解空间中的所有解并一一判断是否可行，故搜索树的规模为指数级别。在数据规模较大时无法承受指数级别的算法，故需要进行优化。

2. 根据优化目的的不同，剪枝可分为很多种。本文主要介绍了两种启发式剪枝与一种数学方法剪枝，由实验结果可以得出，启发式剪枝效率不稳定而添加了数学方法之后效率提升飞快。

3. 在给定的地图数据上均得出了较快速的搜索速度，自拟了许多不同性质的地图数据进行测试，展示了不同维度对算法效率的影响。但没有生成严谨的地图数据，只是单纯生成了随机的约束图，不一定是平面图，故并不一定满足四色定理。

五、附件说明

- code（实验所编写代码）
  - basic.cpp（朴素回溯法）
  - generatedata.cpp（数据生成器）
  - optimization_MRV_DH.cpp（添加了 MRV 和 DH 的回溯法）
  - optimization_MRV_DH_LH.cpp（添加了 MRV 和 DH 和颜色轮换的回溯法）
- result（实验结果数据表）

指导教师批阅意见： 成绩评定： 指导教师签字： 年　　月　　日
备注：

注：1、报告内的项目或内容设置，可根据实际情况加以调整和补充。

2、教师批改学生实验报告时间应在学生提交实验报告时间后 10 日内。