**深 圳 大 学 实 验 报 告**

**课程名称： 算法设计与分析**

**实验名称： 实验三 回溯法—地图填色问题**

**学院： 计算机与软件学院 专业：计算机科学与技术（创新班）**

**报告人： XXX 学号： XXXXXXXX 班级： 高性能班**

**同组人：**

**指导教师： 杨烜**

**实验时间： 2024.4.22**

**实验报告提交时间： 2024.4.30**

**教务处制**

**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、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。 我们可能还想使用尽可能少的不同颜色进行填涂。一些简单的“地图”（例如棋盘）仅需要两种颜色（黑白），但是大多数复杂的地图需要更多颜色。

每张地图包含四个相互连接的国家时，它们至少需要四种颜色。1852年，植物学专业的学生弗朗西斯·古思里（Francis Guthrie）于1852年首次提出“四色问题”。他观察到四种颜色似乎足以满足他尝试的任何地图填色问题，但他无法找到适用于所有地图的证明。这个问题被称为四色问题。长期以来，数学家无法证明四种颜色就够了，或者无法找到需要四种以上颜色的地图。直到1976年德国数学家沃尔夫冈·哈肯（Wolfgang Haken）（生于1928年）和肯尼斯·阿佩尔（Kenneth Appel，1932年-2013年）使用计算机证明了四色定理，他们将无数种可能的地图缩减为1936种特殊情况，每种情况都由一台计算机进行了总计超过1000个小时的检查。

他们因此工作获得了美国数学学会富尔克森奖。在1990年，哈肯（Haken）成为伊利诺伊大学（University of Illinois）高级研究中心的成员，他现在是该大学的名誉教授。

四色定理是第一个使用计算机证明的著名数学定理，此后变得越来越普遍，争议也越来越小 更快的计算机和更高效的算法意味着今天您可以在几个小时内在笔记本电脑上证明四种颜色定理。

**三、问题描述**

1、对下面这个小规模数据，利用四色填色测试算法的正确性；

画着卡通人物

中度可信度描述已自动生成

2、对附件中给定的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，请解释为什么填涂颜色超过了四色；

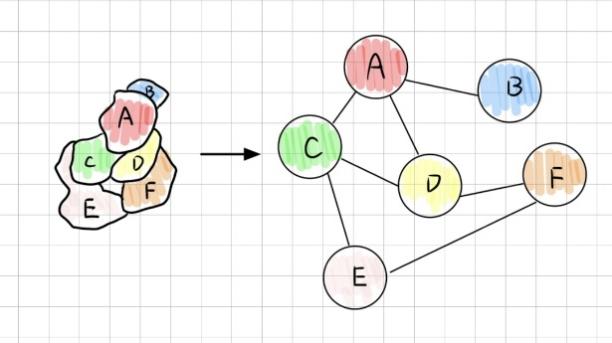
3、设计剪枝策略，分析不同的剪枝策略的效率；

4、随机产生不同规模的图，分析算法效率与图规模的关系。

**四、求解问题的算法原理描述**

**1、问题简化**

根据问题要求，我们需要将一张地图上所有区域涂色，并且要保证相邻区域的涂色不同。由于计算机难以处理地图这样具象的数据，所以我们将地图抽象为无向图，无向图中的每一个节点代表地图中每个独立的区域，节点间的无向边代表两个独立的区域相邻。故问题简化为我们要对一个无向图中的每一个节点涂色，且保证图中相邻节点的涂色不同。



**2、回溯法算法思想**

基本思想就是按照一定的约束条件沿着某条路径搜索，如果遇到不符合约束条件的节点就回退，直到找到一条满足条件的路径。

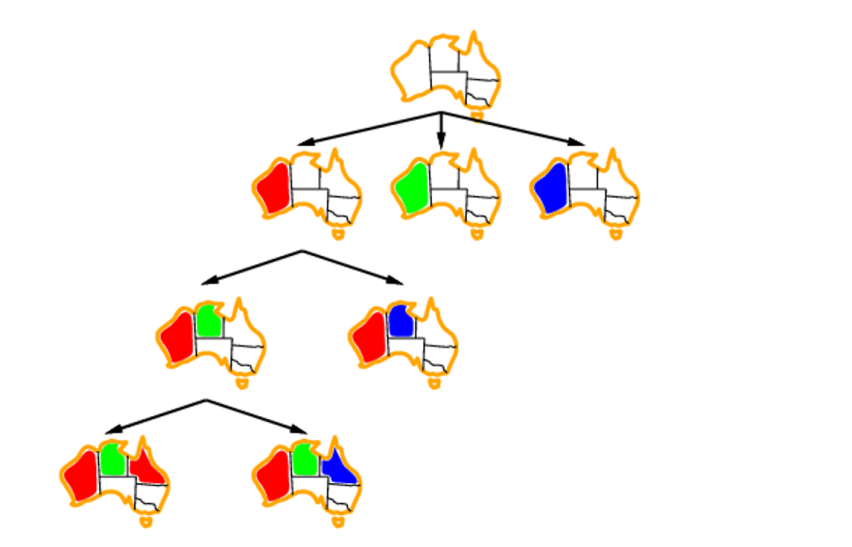
解空间可以理解为树状结构，树的每个节点表示一个状态，如果该状态符合解的状态时遍历该节点的子节点，如果该状态不符合解的状态时则不遍历子节点。

图示

描述已自动生成

**3、地图填色中的回溯法**

在地图填色问题中，每一个节点就是一个需要填色的区域，对区域不同的涂色即表示不同的状态，约束条件为相邻的区域所填颜色不能一致。



**4、填涂颜色数目**

为什么有的地图填色仅需要四种填涂颜色即可找到解，有的地图则需要四种以上的填涂颜色才能找到解？这是由地图简化后的无向图中的最大完全子图阶数所决定的。

如下图中，左边为原始地图，右边为原始地图所简化成的无向图。不难发现，该地图的最大完全子图由A、B、C、D四个节点组成（即红线相连的子图）。假设填涂颜色有红、黄、蓝，为了满足相邻节点间涂色不同的约束条件，A、B、C只能分别填涂上红、黄、蓝，但到了填涂D时，则已近无颜色可涂，即用三种颜色填涂该地图必然无解。当填涂颜色种类数为四，A、B、C、D可以分别填涂上不同的颜色且满足相邻节点间涂色不同的约束条件，此时用四种颜色填涂该地图有解，而四恰好为该最大完全子图的阶数。

图示, 工程绘图

描述已自动生成

所以，若希望对地图填色有解，则填涂颜色的种类数量必须大于等于该地图最大完全子图的阶数。

**5、对回溯法的优化剪枝**

普通的回溯法仅适合在小规模的数据集中寻找解，但当我们需要处理大规模的数据集时，则需要对现有的回溯法进行优化剪枝，以下给出几种优化剪枝的策略。

1. 最大度优先（DH）

按照最大度优先原则对地图填色即按无向图中节点的度数从大到小的顺序对每个节点进行颜色填色。

优先选择度数最大的节点涂色对其周围的节点产生的约束条件最强，使得最多的节点减少一种可使用的颜色，从而导致整棵搜索树的分支较少，减少搜索空间。也可以从另一个角度来理解，优先处理当前度最大的节点，使周围相邻节点处于更受限的状态，可能会使得不可行解更早出现，从而提前剪枝。

图示

描述已自动生成

举个例子，现在我们需要用五种颜色对下面的地图进行填色。

图示, 工程绘图

描述已自动生成

如果我们按A、B、C、D、E的顺序对地图的节点进行填色，就能得到下图中左边的搜索树；如果我们按最大度优先的原则，即E、A、B、C、D的顺序对地图的节点进行填色，就能得到下图中右边的搜索树。

图片包含 室内, 照片, 桌子, 挂

描述已自动生成

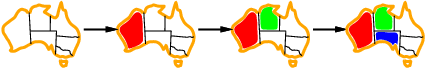
观察两个搜索树，不难发现，由于遵循了最大度优先的原则，E会最先被填涂上颜色，由于E和其他节点都相连，则其他节点能用于填涂颜色的个数减一，因为这个约束，当对第三个节点B进行涂色时，B只有三种颜色可选用于填涂，相对于按A、B、C、D、E的顺序对地图涂色涂到第三个节点时产生的分支少一，即完成了剪枝。通过以上例子可见，最大度优先确实能够减少解空间，达到提前剪枝的效果。

1. 最少可选颜色优先（MRV）

按照最少可选颜色优先的原则对地图填色即在无向图中优先选择剩余未填色节点中可填颜色最少的节点进行填涂。

优先选择剩余可用颜色最少的节点进行填色，这样可以减少剩余可用颜色最多的节点需要尝试不同颜色的次数，即减少分支。也可以从另一个角度来理解，优先处理当前剩余可用颜色最少的节点，使周围相邻节点处于更受限的状态，可能会使得不可行解更早出现，从而提前剪枝。

由于每次对节点涂不同颜色会导致相邻节点剩余可用颜色数量不同，所以在对节点涂色后，需要更新剩余未填色节点可填颜色从小到大的顺序，然后在进行对下一个节点的填色。



举个例子，现在我们需要用五种颜色对下面的地图进行填色。

图示, 工程绘图

描述已自动生成

如果我们按A、B、C、D、E的顺序对地图的节点进行填色，就能得到下图中左边的搜索树；如果我们按最少可选颜色优先的原则，即A、B、E、C、D的顺序（对A涂色后，其余节点剩余可选颜色数目相同，故下一个涂色节点选B；对B涂色后，E剩余可选颜色数目为3，C和D剩余可选颜色数目为4，故下一个涂色节点选E；对E涂色后，其余节点剩余可选颜色数目相同，故下一个涂色节点选C；最后对剩余的最后一个节点D涂色）对地图的节点进行填色，就能得到下图中右边的搜索树。

图片包含 室内, 物体, 桌子, 照片

描述已自动生成

观察两个搜索树，不难发现，由于遵循了最少可选颜色优先的原则，当填涂完A和B后，第三个涂色的节点为E，由于A和B颜色的约束，所以E可选涂色颜色数目为3，故对E涂色只会产生三条分支，相对于按A、B、C、D、E的顺序对地图涂色涂到第三个节点时产生的分支少一，即完成了剪枝。通过以上例子可见，最少可选颜色优先确实能够减少解空间，达到提前剪枝的效果。

1. 向前探测剪枝

向前探测剪枝即当某节点选择了一个颜色后，在相邻节点的可选颜色中将该颜色删去。如果在删除的过程中发现删除后没有颜色可以填涂，就放弃该节点的这个颜色。向前探查有利于我们提前发现不可行解，避免搜索该路径不必要的时间开销，从而达到剪枝效果。

举个例子，我们需要用红和蓝两种颜色对下面的地图进行填色。假设对节点1涂上红色后，其相邻节点3可选颜色则只剩蓝色。现在要对节点2进行涂色，若节点2涂上蓝色，则其相邻节点3的可选颜色中要将蓝色删去，删去后发现节点3无可选颜色，说明这是个不可行的解，随放弃该分支，即提前剪枝。若节点2涂上红色，则其相邻节点3的可选颜色中要将红色删去，删去后节点3可选颜色数目仍大于零，说明当前状态可行，即选择向该分支继续向下搜索最终解。通过向前探查，我们不难发现该方法确实可以达到剪枝效果，从而避免我们搜索无可行解的分支。

图示

描述已自动生成

1. 最小约束优先

按最小约束优先的原则对地图进行填色即在对节点进行涂色时，在可选颜色中，选择约束最小的颜色，即对相邻未涂色节点的可选颜色减少数量最小的颜色，给相邻未涂色节点更多可选颜色，从而提高找到解的可能性。但由于其给了相邻未涂色节点更多可选颜色，相当于搜索树的分支增多了，增大了解空间，可能会导致搜索解的效率下降的问题。

在对节点涂色前，我们需要先计算出每个可选颜色涂色后对相邻未涂色节点的可选颜色减少的数量，并将其记录记录下来，当全部可选颜色计算完毕后，再按可选颜色涂色后对相邻未涂色节点的可选颜色减少数量从小到大的顺序，对该节点进行涂色。

举个例子，我们需要对下面地图进行三色填涂。

图示

描述已自动生成

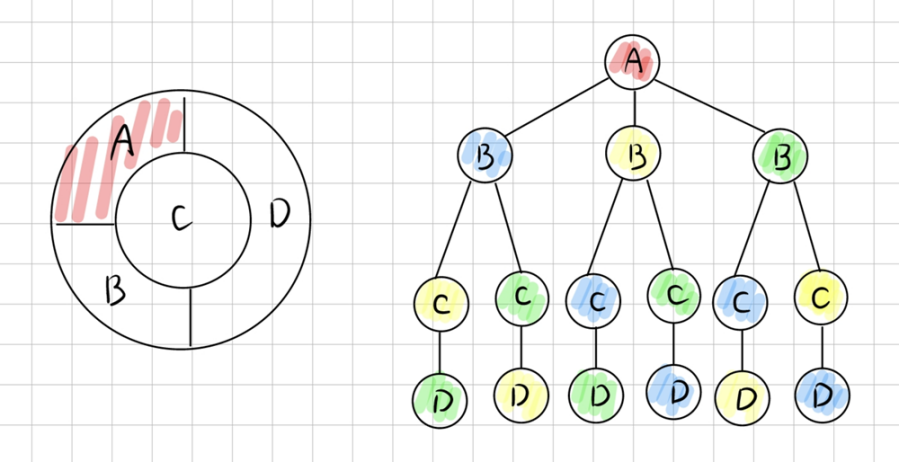
当对节点\*涂色时，若涂上红色，则相邻未涂色节点的可选颜色减少数量为1，其中节点SA可选颜色减少数量为0，可选颜色剩余蓝色；若涂上蓝色，则相邻未涂色节点的可选颜色减少数量为2，其中节点SA可选颜色减少数量为1，无可选颜色。不难发现，选择对相邻未涂色节点产生约束最小的红色进行填涂，确实能保证接下来的分支可能有解；但选择了对相邻未涂色节点产生约束最大的蓝色进行填涂，必然导致接下来的分支无解，这是因为节点SA已经无可选颜色。所以，最小约束优先原则确实能提高找到解的可能性。

1. 等效冗余剪枝

等效冗余剪枝即在搜索过程中，如果我们能够判定从搜索树的当前节点上沿着某几条不同分支到达的子树是等效的，那么只需要对其中的一条分支执行搜索，然后让返回的结果乘上这几条分支的数目，即可得到这几条分支的总结果，从而无需再搜索其他分支，达到了剪枝的效果。

在地图填色中，若当前需要涂色的节点有 x 种从未被其他点使用过的颜色可以使用，则这 x 种颜色可以视为等价的颜色，直接使用其中一种颜色搜索得到可行解方案数y后，这x种颜色总可行解方案数为x\*y，从而无需再填涂其他颜色进行搜索，达到了剪枝的效果。

举个例子，我们需要对下图中的地图进行四色填色。假设当前节点A已经涂上红色，下一个涂色节点为B，通过计算我们能够得到节点B的三种可选颜色均未被其他节点使用过，再通过构建当前状态后的搜索树，我们不难发现，对节点B涂色产生的三条分支到达的子树是等效的，即节点B的三种可选颜色是等效的，因此我们只需要对节点B涂色中的一条分支进行搜索得到可行解方案数为2，再乘上节点B等效可选颜色数目3，最终就能得到当前状态下对节点B涂色后可行解方案总数2\*3=6。通过前面构建的搜索树我们能验证该方法的正确性，即说明等效冗余剪枝是有效的。



1. **算法实现的核心伪代码**

**1、回溯法（无优化）**

|  |
| --- |
| Function Dfs(x)  if x > vertices  ans++ // 到达递归终点，记录下方案  return // 回溯  for c in colors  color[x] = c // 选择颜色填涂节点x  if Check(Adjacency\_list) // 满足约束条件则继续往下搜索  Dfs(x+1)  color[x] = 0 // 恢复节点x颜色 |

**2、回溯法（最大度优先）**

|  |
| --- |
| Function Dfs(x)  if x > vertices  ans++ // 到达递归终点，记录下方案  return // 回溯  for c in colors  color[x] = c // 选择颜色填涂节点x  if Check(Adjacency\_list) // 满足约束条件则继续往下搜索  Dfs(x+1)  color[x] = 0 // 恢复节点x颜色  Sort\_by\_degree(Adjacency\_list) // 根据每个节点的度从大到小排序邻接表  Dfs(1) // 根据排序后的顺序搜索答案 |

**3、回溯法（最少可选颜色优先）**

|  |
| --- |
| Function Dfs(x)  if x > vertices  ans++ // 到达递归终点，记录下方案  return // 回溯  Sort\_by\_color(Adjacency\_list) // 根据每个剩余未涂色节点的剩余可选颜色数量  //从小到大排序邻接表  for c in colors  color[x] = c // 选择颜色填涂节点x  if Check(Adjacency\_list) // 满足约束条件则继续往下搜索  Sub\_color(Adjacency\_list) // 从相邻节点可选颜色中减去颜色c  Dfs(x+1)  Add\_color(Adjacency\_list) // 从相邻节点可选颜色中恢复颜色c  color[x] = 0 // 恢复节点x颜色 |

**4、回溯法（向前探测剪枝）**

|  |
| --- |
| Function Dfs(x)  if x > vertices  ans++ // 到达递归终点，记录下方案  return // 回溯  for c in colors  color[x] = c // 选择颜色填涂节点x  if Check(Adjacency\_list) && Forward\_detect(Adjacency\_list)  // 满足约束条件且向前探测通过则继续往下搜索  Dfs(x+1)  color[x] = 0 // 恢复节点x颜色 |

**5、回溯法（最小约束优先）**

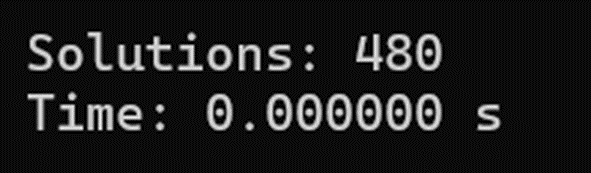
|  |
| --- |
| Function Dfs(x)  if x > vertices  ans++ // 到达递归终点，记录下方案  return // 回溯  Priority\_queue q // 维护一个最小堆，对相邻节点可选颜色影响最小的颜色处于队头  for c in colors  color[x] = c // 选择颜色填涂节点x  if Check(Adjacency\_list)  q.push(c) // 通过约束条件的颜色入队  // 优先填涂对相邻节点可选颜色影响最小的颜色  while !queue.empty()  chose\_color = q.top()  q.pop()  color[x] = chose\_color  Dfs(x+1)  color[x] = 0 // 恢复节点x颜色 |

1. **回溯法（等效冗余剪枝）**

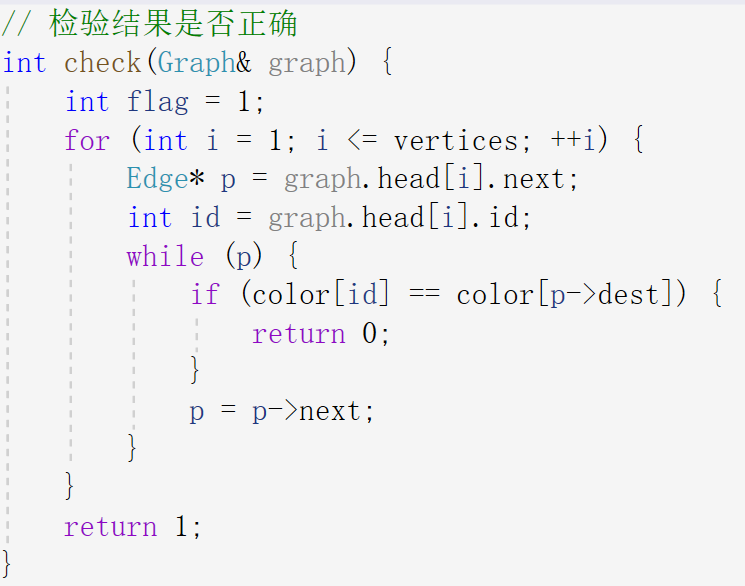
|  |
| --- |
| Function Dfs(x)  if x > vertices  ans++ // 到达递归终点，记录下方案  return // 回溯  ans = 0 // ans为填涂节点x得到解的总方案数  cnt = number\_of\_not\_use\_color() // cnt为当前节点可用且其他节点未用的颜色数目  for c in colors  color[x] = c // 选择颜色填涂节点x  // 第一次遍历到可用且其他节点未用的颜色，方案数要乘上cnt  if Check(Adjacency\_list) && Not\_use(color) && First()  ans += cnt\*Dfs(x+1)    // 其余可用颜色正常计算  eles if Check(Adjacency\_list)  ans += Dfs(x+1)  color[x] = 0 // 恢复节点x颜色  return ans // 返回总方案数 |

**六、算法测试结果及效率分析**

**1、小规模数据测试**



通过回溯法（无优化）四色填涂小规模数据，得到可行解的方案个数为480个，所用时间接近于0秒。



通过上面的代码对解进行检查，能够知道480个方案均正确，回溯法的正确性得以验证。

1. **大规模数据测试**

结合上面提到的算法和所有的优化剪枝策略，分别使用5个（le450\_5a），15个（le450\_15b），25个（le450\_25a）颜色为附件大规模数据着色。由于地图le450\_15b和le450\_25a的可行解的方案数过大，所以除了结合等效冗余剪枝的回溯法外，其余方法均记录求解le450\_5a、le450\_15b、le450\_25a第一个可行解的所用时间。

**1）回溯法（无优化）**

|  |  |  |  |
| --- | --- | --- | --- |
| 回溯法（无优化）（只找一个解） | | | |
| 地图 | Le450\_5a | Le450\_15b | Le450\_25a |
| 时间/s | ∞ | ∞ | ∞ |

通过测试可以发现，无优化的回溯法在短时间内找不到三个地图的第一个可行解。

**2）最大度优先（DH）**

|  |  |  |  |
| --- | --- | --- | --- |
| 最大度优先（只找一个解） | | | |
| 地图 | Le450\_5a | Le450\_15b | Le450\_25a |
| 时间/s | ∞ | ∞ | 0.001000 |

通过测试可以发现，结合了最大度优先的回溯法只能快速找到Le450\_25a的第一个可行解。

**3）最少可选颜色优先（MRV）**

|  |  |  |  |
| --- | --- | --- | --- |
| 最少可选颜色优先（只找一个解） | | | |
| 地图 | Le450\_5a | Le450\_15b | Le450\_25a |
| 时间/s | 6.835000 | ∞ | 0.014000 |

通过测试可以发现，结合了最少可选颜色优先的回溯法能快速找到Le450\_5a和Le450\_25a的第一个可行解。

|  |  |  |  |
| --- | --- | --- | --- |
| 最大度+最少可选颜色优先（只找一个解） | | | |
| 地图 | Le450\_5a | Le450\_15b | Le450\_25a |
| 时间/s | 7.367000 | ∞ | 0.026000 |

通过测试可以发现，结合了最大度和最少可选颜色优先的回溯法能快速找到Le450\_5a和Le450\_25a的第一个可行解，但速度要比结合了最少可选颜色优先的回溯法慢，这可能是因为两个方法的结合导致求解Le450\_5a和Le450\_25a的最佳填涂顺序被改变了。

1. **向前探测剪枝**

|  |  |  |  |
| --- | --- | --- | --- |
| 向前探测剪枝（只找一个解） | | | |
| 地图 | Le450\_5a | Le450\_15b | Le450\_25a |
| 时间/s | ∞ | ∞ | ∞ |

通过测试可以发现，结合向前探测剪枝的回溯法在短时间内找不到三个地图的第一个可行解。

|  |  |  |  |
| --- | --- | --- | --- |
| 结合DH和MRV的向前探测剪枝（只找一个解） | | | |
| 地图 | Le450\_5a | Le450\_15b | Le450\_25a |
| 向前探测（时间/s） | ∞ | ∞ | ∞ |
| DH+向前探测剪枝（时间/s） | ∞ | ∞ | 0.002000 |
| MRV+向前探测剪枝（时间/s） | 5.768000 | ∞ | 0.014000 |
| DH+MRV+向前探测剪枝（时间/s） | 6.823000 | ∞ | 0.027000 |

通过测试可以发现，结合MRV+向前探测剪枝或DH+MRV+向前探测剪枝的回溯法能快速找到Le450\_5a和Le450\_25a的第一个可行解。并且通过比较，使用了向前探测剪枝的策略能够加速使用了MRV和DH+MRV两种算法找第一个可行解的速度，说明向前探测剪枝是有效的。

1. **最小约束优先**

|  |  |  |  |
| --- | --- | --- | --- |
| 最小约束优先（只找一个解） | | | |
| 地图 | Le450\_5a | Le450\_15b | Le450\_25a |
| 时间/s | ∞ | ∞ | ∞ |

通过测试可以发现，结合最小约束优先的回溯法在短时间内找不到三个地图的第一个可行解。

|  |  |  |  |
| --- | --- | --- | --- |
| 结合DH、MRV、向前探测的最小约束优先（只找一个解） | | | |
| 地图 | Le450\_5a | Le450\_15b | Le450\_25a |
| 最小约束优先（时间/s） | ∞ | ∞ | ∞ |
| DH+最小约束优先（时间/s） | ∞ | ∞ | 0.022000 |
| MRV+最小约束优先（时间/s） | 0.009000 | ∞ | 0.019000 |
| MRV+向前探测+最小约束优先（时间/s） | 0.009000 | ∞ | 0.019000 |
| DH+MRV+最小约束优先（时间/s） | 5.498000 | 11.588000 | 0.030000 |
| DH+MRV+向前探测+最小约束优先（时间/s） | 5.086000 | 10.427000 | 0.032000 |

通过测试可以发现，结合了MRV+最小约束优先或MRV+向前探测+最小约束优先的回溯法是目前找到Le450\_5a第一个可行解最快的算法。同时，结合了DH+MRV+最小约束优先或DH+MRV+向前探测+最小约束优先的回溯法也是目前能够在短时间内找到Le450\_15b的第一个可行解的算法，这说明最小约束优先确实能够提高找到可行解的可能性。

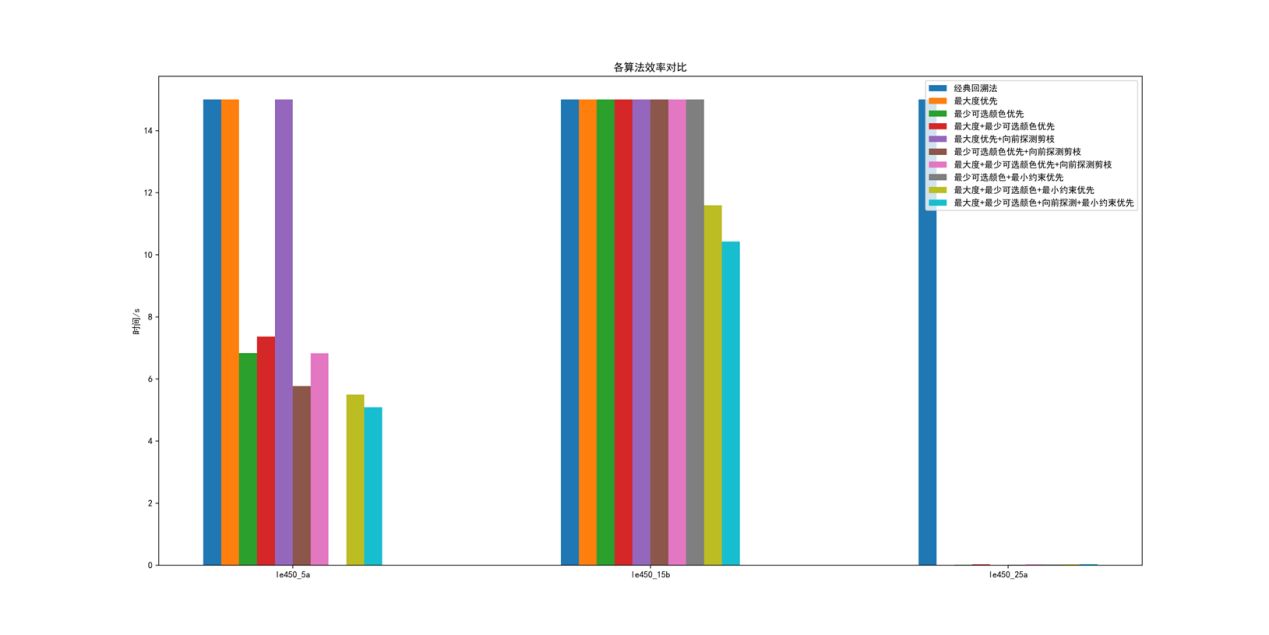
1. **等效冗余剪枝**

|  |  |  |
| --- | --- | --- |
| Le450\_5a中找全部解 | | |
| 方法 | 最大度+最少可选颜色优先+向前探测剪枝 | 最大度+最少可选颜色优先+向前探测剪枝+等效冗余剪枝 |
| 解个数 | 3840 | 3840 |
| 时间/s | 1208.806000 | 10.711000 |

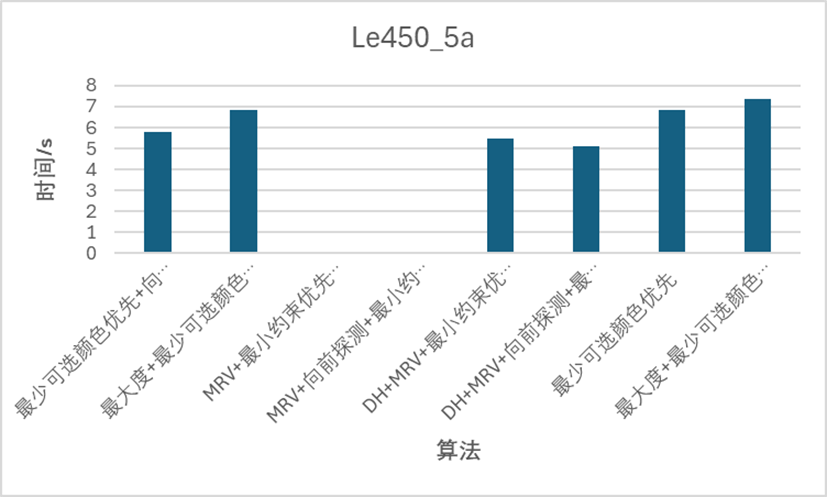
通过测试可以发现，Le450\_5a全部可行解的方案数为3840，且等效冗余剪枝能够大幅度提高回溯法找到全部可行解的速度。

1. **全部策略运行效率的比较**

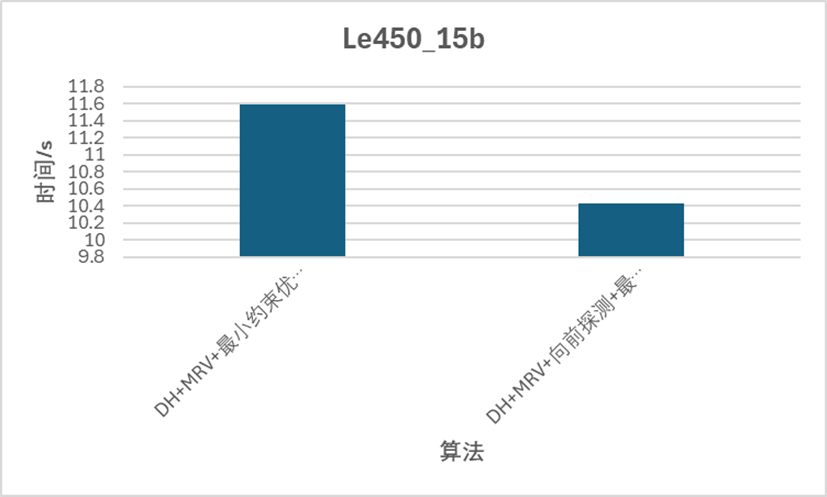
下图是结合各策略的回溯法找三个地图第一个可行解的所用时间柱状图。



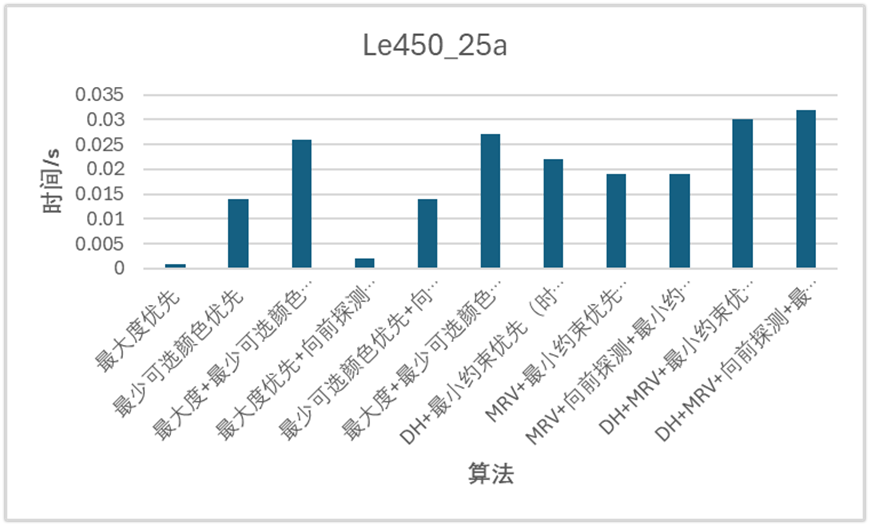
下图是结合各策略的回溯法能够在短时间内找Le450\_5a第一个可行解的所用时间柱状图，不难发现，对于Le450\_5a，结合了MRV+最小约束优先或MRV+向前探测+最小约束优先的回溯法是求解该地图第一个可行解速度最快的算法。



下图是结合各策略的回溯法能够在短时间内找Le450\_15b第一个可行解的所用时间柱状图，不难发现，对于Le450\_15b，结合了DH+MRV+向前探测+最小约束优先的回溯法是求解该地图第一个可行解速度最快的算法。



下图是结合各策略的回溯法能够在短时间内找Le450\_25a第一个可行解的所用时间柱状图，不难发现，对于Le450\_25a，结合了DH的回溯法是求解该地图第一个可行解速度最快的算法。



1. **算法效率与图规模的关系**

下面通过随机生成地图来找算法效率与图规模的关系，以下是随机地图的生成代码。



1. **边疏密对回溯法效率的影响**

分别用无优化的回溯法、结合DH+MRV+向前探测剪枝的回溯法、结合DH+MRV+向前探测剪枝+最小约束优先的回溯法，用12色搜索节点数为200、边数为不同规模的随机生成地图的第一个可行解，每个规模的边数测试5次，记录下每次测试所用时间。

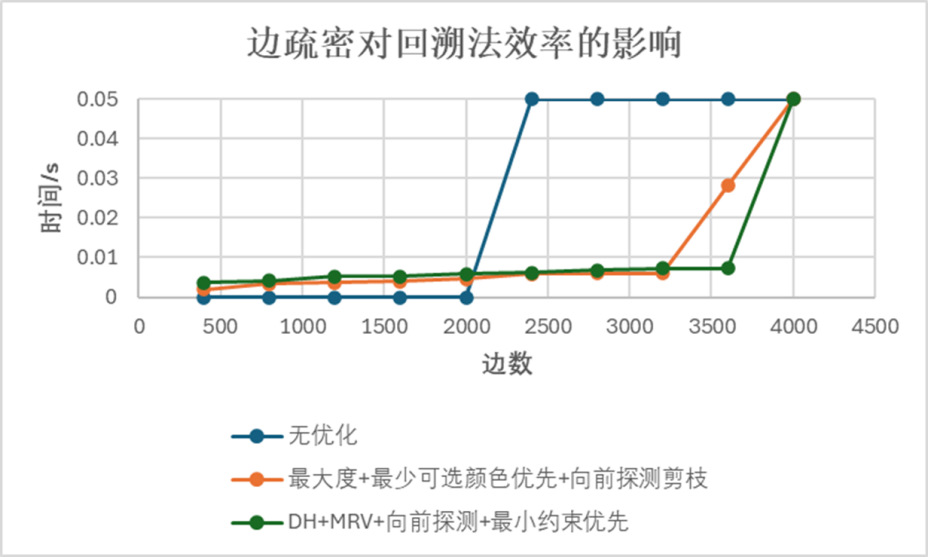
以下为测试结果。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 边疏密对回溯法（无优化）效率的影响（节点数为200，可填颜色数为12） | | | | | | | | | | |
| 边数 | 400 | 800 | 1200 | 1600 | 2000 | 2400 | 2800 | 3200 | 3600 | 4000 |
| 测试1用时/s | 0 | 0 | 0 | 0 | 0 | 0 | ∞ | ∞ | ∞ | ∞ |
| 测试2用时/s | 0 | 0 | 0 | 0 | 0 | 0 | 7.529000 | ∞ | ∞ | ∞ |
| 测试3用时/s | 0 | 0 | 0 | 0 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |
| 测试4用时/s | 0 | 0 | 0 | 0 | 0 | 4.152000 | ∞ | ∞ | ∞ | ∞ |
| 测试5用时/s | 0 | 0 | 0 | 0 | 0 | 0 | ∞ | ∞ | ∞ | ∞ |
| 平均用时/s | 0 | 0 | 0 | 0 | 0 | ∞ | ∞ | ∞ | ∞ | ∞ |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 边疏密对回溯法（最大度+最少可选颜色优先+向前探测剪枝）效率的影响（节点数为200，可填颜色数为12） | | | | | | | | | | |
| 边数 | 400 | 800 | 1200 | 1600 | 2000 | 2400 | 2800 | 3200 | 3600 | 4000 |
| 测试1用时/s | 0.002000 | 0.003000 | 0.003000 | 0.004000 | 0.005000 | 0.005000 | 0.006000 | 0.007000 | 0.008000 | ∞ |
| 测试2用时/s | 0.002000 | 0.004000 | 0.004000 | 0.004000 | 0.004000 | 0.005000 | 0.007000 | 0.005000 | 0.057000 | ∞ |
| 测试3用时/s | 0.002000 | 0.004000 | 0.004000 | 0.004000 | 0.005000 | 0.005000 | 0.006000 | 0.006000 | 0.041000 | 16.894000 |
| 测试4用时/s | 0.002000 | 0.003000 | 0.004000 | 0.004000 | 0.004000 | 0.008000 | 0.005000 | 0.006000 | 0.008000 | ∞ |
| 测试5用时/s | 0.002000 | 0.003000 | 0.003000 | 0.004000 | 0.005000 | 0.006000 | 0.006000 | 0.006000 | 0.027000 | ∞ |
| 平均用时/s | 0.002000 | 0.003400 | 0.003600 | 0.004000 | 0.0046 00 | 0.005800 | 0.006000 | 0.006000 | 0.028200 | ∞ |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 边疏密对回溯法（DH+MRV+向前探测+最小约束优先）效率的影响（节点数为200，可填颜色数为12） | | | | | | | | | | |
| 边数 | 400 | 800 | 1200 | 1600 | 2000 | 2400 | 2800 | 3200 | 3600 | 4000 |
| 测试1用时/s | 0.004000 | 0.004000 | 0.005000 | 0.005000 | 0.005000 | 0.006000 | 0.007000 | 0.006000 | 0.007000 | 1.372000 |
| 测试2用时/s | 0.004000 | 0.004000 | 0.005000 | 0.006000 | 0.006000 | 0.007000 | 0.007000 | 0.007000 | 0.019000 | ∞ |
| 测试3用时/s | 0.003000 | 0.004000 | 0.006000 | 0.005000 | 0.006000 | 0.007000 | 0.007000 | 0.009000 | 0.010000 | ∞ |
| 测试4用时/s | 0.004000 | 0.005000 | 0.005000 | 0.006000 | 0.006000 | 0.007000 | 0.007000 | 0.008000 | 0.009000 | ∞ |
| 测试5用时/s | 0.004000 | 0.004000 | 0.005000 | 0.006000 | 0.007000 | 0.007000 | 0.007000 | 0.007000 | 0.021000 | ∞ |
| 平均用时/s | 0.003600 | 0.004200 | 0.005200 | 0.005200 | 0.005800 | 0.006200 | 0.006800 | 0.007200 | 0.007200 | ∞ |

根据上面测试结果做出曲线图如下。



对于无优化的回溯法，其求解稀疏图的第一个解的速度是最快的，但随着边规模的增大，图稠密到一定程度时，无论边规模再怎么增大，无优化的回溯法也无法短时间内求出第一个解。对于结合DH+MRV+向前探测剪枝的回溯法和结合DH+MRV+向前探测剪枝+最小约束优先的回溯法，其不管是求解稀疏图还是稠密图，其速度都是相对较快的，直到边规模增到非常大时，才无法在短时间内求得第一个解。此外，能够观察到，图逐渐稠密，算法运行的时间也逐渐变长。

所以，无优化的回溯法在稀疏图的求解上的速度是最快的，但其所能处理边规模的范围较小；而结合DH+MRV+向前探测剪枝的回溯法和结合DH+MRV+向前探测剪枝+最小约束优先的回溯法在稀疏图和稠密图的求解上的速度都是相对较快的，其所能处理边规模的范围也较大，这可能是因为当图稀疏，使得问题比较简单，解空间较小，导致剪枝优化等策略的执行反而会减慢算法的运行速度。

1. **节点数量对回溯法效率的影响**

分别用无优化的回溯法、结合DH+MRV+向前探测剪枝的回溯法、结合DH+MRV+向前探测剪枝+最小约束优先的回溯法，用15色搜索边数为2000、节点数量为不同规模的随机生成地图的第一个可行解，每个规模的节点数量测试5次，记录下每次测试所用时间。

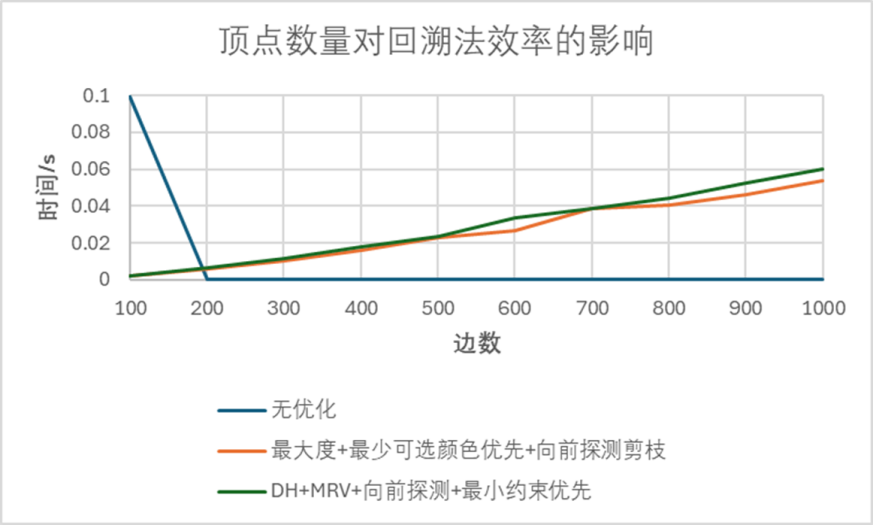
以下为测试结果。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 节点数量对回溯法（无优化）效率的影响（边数为2000，可填颜色数为15） | | | | | | | | | | |
| 节点数 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 |
| 测试1用时/s | ∞ | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.001000 |
| 测试2用时/s | ∞ | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 测试3用时/s | ∞ | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 测试4用时/s | ∞ | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.001000 |
| 测试5用时/s | ∞ | 0.001000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.001000 | 0.000000 | 0.000000 |
| 平均用时/s | ∞ | 0.000200 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000200 | 0.000000 | 0.000400 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 节点数量对回溯法（最大度+最少可选颜色优先+向前探测剪枝）效率的影响（边数为2000，可填颜色数为15） | | | | | | | | | | |
| 节点数 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 |
| 测试1用时/s | 0.001000 | 0.006000 | 0.011000 | 0.015000 | 0.019000 | 0.028000 | 0.033000 | 0.042000 | 0.045000 | 0.058000 |
| 测试2用时/s | 0.002000 | 0.005000 | 0.009000 | 0.019000 | 0.021000 | 0.029000 | 0.035000 | 0.039000 | 0.047000 | 0.052000 |
| 测试3用时/s | 0.002000 | 0.005000 | 0.013000 | 0.015000 | 0.023000 | 0.025000 | 0.033000 | 0.044000 | 0.046000 | 0.051000 |
| 测试4用时/s | 0.002000 | 0.006000 | 0.009000 | 0.015000 | 0.023000 | 0.025000 | 0.047000 | 0.038000 | 0.046000 | 0.051000 |
| 测试5用时/s | 0.002000 | 0.006000 | 0.009000 | 0.015000 | 0.028000 | 0.025000 | 0.044000 | 0.039000 | 0.046000 | 0.057000 |
| 平均用时/s | 0.001800 | 0.005600 | 0.010200 | 0.015800 | 0.022800 | 0.026400 | 0.038400 | 0.040400 | 0.046000 | 0.053800 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 节点数量对回溯法（DH+MRV+向前探测+最小约束优先）效率的影响（边数为2000，可填颜色数为15） | | | | | | | | | | |
| 节点数 | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | 1000 |
| 测试1用时/s | 0.003000 | 0.007000 | 0.011000 | 0.016000 | 0.025000 | 0.036000 | 0.039000 | 0.045000 | 0.051000 | 0.060000 |
| 测试2用时/s | 0.001000 | 0.006000 | 0.011000 | 0.017000 | 0.025000 | 0.034000 | 0.039000 | 0.043000 | 0.051000 | 0.059000 |
| 测试3用时/s | 0.003000 | 0.006000 | 0.012000 | 0.019000 | 0.023000 | 0.032000 | 0.040000 | 0.044000 | 0.052000 | 0.061000 |
| 测试4用时/s | 0.002000 | 0.006000 | 0.012000 | 0.020000 | 0.022000 | 0.036000 | 0.038000 | 0.044000 | 0.053000 | 0.061000 |
| 测试5用时/s | 0.001000 | 0.006000 | 0.012000 | 0.016000 | 0.022000 | 0.031000 | 0.038000 | 0.045000 | 0.056000 | 0.059000 |
| 平均用时/s | 0.002000 | 0.006200 | 0.011600 | 0.017600 | 0.023400 | 0.033800 | 0.038800 | 0.044200 | 0.052600 | 0.060000 |

根据上面测试结果做出曲线图如下。



对于无优化的回溯法，除了刚开始时节点较少导致图太稠密而无法短时间内求得第一个解外，随着顶点数量规模的增大，其都能最快地找到第一个可行解。对于结合DH+MRV+向前探测剪枝的回溯法和结合DH+MRV+向前探测剪枝+最小约束优先的回溯法，则是随着顶点数量规模的增大，算法运行的时间也逐渐变长。

所以，无优化的回溯法在稀疏图的求解上的速度是最快的，其速度受节点规模变化的影响相对较小；而结合DH+MRV+向前探测剪枝的回溯法和结合DH+MRV+向前探测剪枝+最小约束优先的回溯法则是随着顶点数量规模的增大而降低求解速度，说明其速度受节点规模变化的影响相对较大。

1. **可选颜色数量对回溯法效率的影响**

分别用无优化的回溯法、结合DH+MRV+向前探测剪枝的回溯法、结合DH+MRV+向前探测剪枝+最小约束优先的回溯法，用不同规模的可选颜色数量搜索边数为2000、节点数为200的随机生成地图的第一个可行解，每个规模的可选颜色数量测试5次，记录下每次测试所用时间。

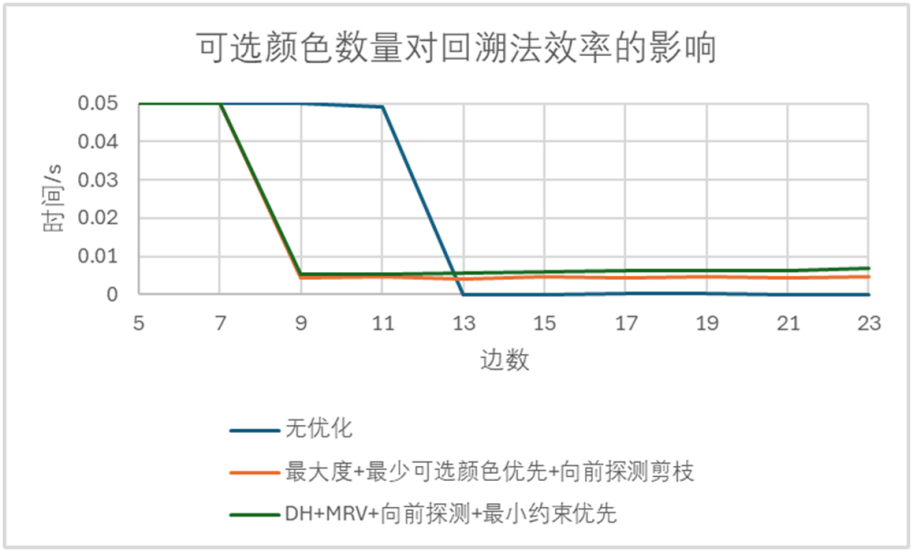
以下为测试结果。

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 可选颜色数量对回溯法（无优化）效率的影响（边数为2000，顶点数为200） | | | | | | | | | | |
| 可选颜色数 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 |
| 测试1用时/s | ∞ | ∞ | ∞ | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.001000 | 0.000000 | 0.000000 |
| 测试2用时/s | ∞ | ∞ | ∞ | 0.007000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 测试3用时/s | ∞ | ∞ | ∞ | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 测试4用时/s | ∞ | ∞ | ∞ | 0.238000 | 0.000000 | 0.000000 | 0.001000 | 0.000000 | 0.000000 | 0.000000 |
| 测试5用时/s | ∞ | ∞ | ∞ | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 |
| 平均用时/s | ∞ | ∞ | ∞ | 0.049000 | 0.000000 | 0.000000 | 0.000200 | 0.000200 | 0.000000 | 0.000000 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 可选颜色数量对回溯法（最大度+最少可选颜色优先+向前探测剪枝）效率的影响（边数为2000，顶点数为200） | | | | | | | | | | |
| 可选颜色数 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 |
| 测试1用时/s | ∞ | ∞ | 0.004000 | 0.005000 | 0.004000 | 0.005000 | 0.004000 | 0.004000 | 0.004000 | 0.005000 |
| 测试2用时/s | ∞ | ∞ | 0.005000 | 0.004000 | 0.004000 | 0.004000 | 0.005000 | 0.005000 | 0.005000 | 0.004000 |
| 测试3用时/s | ∞ | ∞ | 0.005000 | 0.005000 | 0.004000 | 0.006000 | 0.004000 | 0.005000 | 0.004000 | 0.005000 |
| 测试4用时/s | ∞ | 14.696000 | 0.004000 | 0.005000 | 0.005000 | 0.004000 | 0.004000 | 0.005000 | 0.005000 | 0.005000 |
| 测试5用时/s | ∞ | 26.390000 | 0.004000 | 0.004000 | 0.004000 | 0.005000 | 0.005000 | 0.005000 | 0.004000 | 0.005000 |
| 平均用时/s | ∞ | ∞ | 0.004400 | 0.004600 | 0.004200 | 0.004800 | 0.004400 | 0.004800 | 0.004400 | 0.004800 |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 可选颜色数量对回溯法（DH+MRV+向前探测+最小约束优先）效率的影响（边数为2000，顶点数为200） | | | | | | | | | | |
| 可选颜色数 | 5 | 7 | 9 | 11 | 13 | 15 | 17 | 19 | 21 | 23 |
| 测试1用时/s | ∞ | ∞ | 0.006000 | 0.006000 | 0.005000 | 0.006000 | 0.006000 | 0.007000 | 0.006000 | 0.008000 |
| 测试2用时/s | ∞ | ∞ | 0.005000 | 0.005000 | 0.005000 | 0.006000 | 0.006000 | 0.006000 | 0.006000 | 0.007000 |
| 测试3用时/s | ∞ | ∞ | 0.005000 | 0.005000 | 0.006000 | 0.006000 | 0.008000 | 0.006000 | 0.007000 | 0.006000 |
| 测试4用时/s | ∞ | ∞ | 0.005000 | 0.006000 | 0.006000 | 0.006000 | 0.006000 | 0.007000 | 0.006000 | 0.007000 |
| 测试5用时/s | ∞ | ∞ | 0.005000 | 0.005000 | 0.006000 | 0.006000 | 0.006000 | 0.006000 | 0.006000 | 0.007000 |
| 平均用时/s | ∞ | ∞ | 0.005200 | 0.005400 | 0.005600 | 0.006000 | 0.006400 | 0.006400 | 0.006200 | 0.007000 |

根据上面测试结果做出曲线图如下。



对于无优化的回溯法，直到可选颜色数量的规模较大时，才能在短时间内找到地图的第一个可行解。对于结合DH+MRV+向前探测剪枝的回溯法和结合DH+MRV+向前探测剪枝+最小约束优先的回溯法，当可选颜色数量的规模较小时，也能在短时间内找到地图的第一个可行解。且当可选颜色数量规模到达一定程度时，其规模的变化对以上三种算法的运行速度几乎没有产生影响。

所以，由于有各种剪枝优化策略的加持，增大了找到可行解的可能且缩小了解空间，使得结合DH+MRV+向前探测剪枝的回溯法和结合DH+MRV+向前探测剪枝+最小约束优先的回溯法能够在可选颜色受限的情况下，在短时间内找到第一个可行解。而无优化的回溯法则在更大程度上受限于较少可选颜色的数量，使得其无法在短时间内找到第一个可行解。

**七、经验总结**

通过这次实验，我们可以深刻认识到并非简单地叠加各种优化手段就一定能确保提升算法的执行效率。实际上，面对不同性质的问题，每一种问题都有自己特有的结构和难点，因此最有效的解决方案往往是独一无二的。这就意味着我们在寻求解答问题的方法时，应当立足于问题本身的实际情况，切忌盲目地罗列和堆砌各类优化技巧。正确的做法是通过反复的实验验证和测试，以此来甄别和挑选真正适用于该问题的优化方法。简言之，解决问题要讲求针对性，依据问题的具体情况灵活选择和应用优化策略。

|  |
| --- |
| 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。