# 最大团问题的分支限界算法

## 最大团问题的定义

最大团问题是指在一个简单无向图G = (V, E)中寻找顶点最多的一个子图C，使得。在本文中，我们使用下列定义：

- 补图——给定一个图G(V, E)，它的补图定义为 (V, )，其中。

- 极大团——图的一个团，该团不能通过添加图的顶点而扩大。

- 独立集——图G = (V,E)的子图S，其中。显然，图的最大团是其补图的最大独立集。

- 候选顶点——可以用来扩充当前团的顶点，即与当前团所有顶点都邻接的顶点。

- 颜色数——在着色过程中赋予顶点的自然数。

- 染色数——对图进行着色所需的最小颜色数。

分支限界算法是求解最大团问题最常用的算法。算法步骤一般分为分支部分与限界部分，其中分支部分选择候选顶点集中的一个顶点进行分支，将该顶点添加到当前团中，在新的节点（新的递归过程）中，候选集被更新为原候选集中与该顶点邻接的所有顶点，因此候选集中所有的顶点都邻接于当前团的所有顶点。而限界部分计算当前候选集中，最大团的大小的上界，如果该上界加上当前团的顶点数不大于目前找到的最大的团，则该分支不可能产生更好的团，因此需要裁减该分支。

本文主要工作在于研究目前最先进的几个分支限界算法，并在这个基础上，构造出效率更好的求解算法。

## 下界与代价函数

分支限界法算法中，分支部分保证了算法的完整性与精确性，而限界部分则决定了算法的效率。为了提高算法效率，关键在于估算出尽可能紧的代价函数，即团大小的上界，以及尽快找到一个大的下界，即一个尽可能大的团。

估计代价函数时，一个平凡的做法是以候选集的顶点个数作为代价函数，如果当前团大小加上候选集顶点个数不超过下界，则可以当前分支无需进入下一轮迭代。然而，这样的代价函数过于松散，未能精确估计团的上界，导致算法效率不高。

### 2.1 染色数

图的染色数可以作为最大团问题的上界（命题1）。

**命题1** (Balas 和Yu 1986)[] 任意图的最大团大小不大于该图的染色数。

这个命题可以从以下事实直接得出：一个具有k 个顶点的团至少需要用k 种颜色进行着色，因为这些顶点都是两两邻接的，因而团的任意两个顶点不能具有同一种颜色。

### 2.2 贪心着色

染色数问题所对应的判定问题，本身就属于NP难问题，因此使用染色数作为最大团问题的代价函数是不实际的。另一种方案则是使用成为贪心着色[]的方法，求出染色数的近似解，作为最大团问题的代价函数。贪心着色过程遵循如下原则：

- 如果(v1,，v2)∈E，则No[v1] ≠ No[v2]

- No[v] = 1，或者No[v] = k > 1，则存在顶点v1∈NR[v]，v2∈NR[v]，...，vk-1∈NR[v]，使得No[v1] = 1，No[v2] = 2，....，No[vk-1] = k-1。

其中，No[v]表示顶点v的颜色编号，NR[v]表示顶点v在子图R中的邻居。

显然，贪心着色过程所用到的颜色数可作为最大团问题的代价函数。对于顶点个数为n的图，贪心着色过程的时间复杂度为O(n2)。

### 2.3下界与分支顺序

好的分支顺序能够使算法快速更新下界，从而在后续迭代中得以裁剪更多的分支。贪心着色过程为分支顺序做了很好的准备——优先选择颜色编码大的顶点。因为团最多只能在每个颜色编码中选取其中一个顶点，而颜色编码最大的顶点，在所有其他颜色编码中都存在与之相邻的顶点。另外，这种分支顺序使得在迭代树的同一层中，只需要进行一次贪心着色过程，因为排除了所选择的顶点后，新一轮贪心着色过程不会产生新的着色方案。

### 2.4 顶点顺序对着色的影响

贪心着色过程根据既定的顺序，依次对顶点进行着色。不同的顺序对着色效果（即所拥得颜色数）有很大的影响。直觉上，限制越多的顶点应该优先进行着色，因此，如果只考虑上界问题的话，对顶点按照度数大小降序排列，可以得到较紧的上界。这便是MCQ[]算法所采用策略。

然而，正如前面所述，贪心着色过程在分支限界算法中，不仅仅是获取上界的手段，同时也对分支策略有重大的影响。顶点被分支的次序与其顶点的颜色编码大小顺序一致，且在同一种颜色编码的顶点中，越晚被着色的顶点越早被选择，因此，这里需要考虑新的排序准则[]。

将顶点排列成如下顺序V={V[1],V[2],...,V[n]}，使得在图G=(V, E)由顶点V’ = {V[1], V[2], ..., V[i]} (i ≤ n) 导出的子图中，使V[i]始终都是V’中度数最小的顶点。如果{V[1], V[2], ..., V[i]}中的所有顶点都有着同样的（最小的）度数，也就是说由{V[1],V[2],...,V[i]}导出的子图是正则的，则对这些顶点进行排序时没有意义的。在这种情况下，我们直接将这些顶点放置在顶点集中。虽然这个排序过程是比较耗时的操作，但是该操作只在算法初始化的时候进行，所以从全局来看，负担是相当小的。

由此便得到了MCR算法[]，该算法是MCQ算法的改进版，效率有了明显的提升。

值得注意的是，虽然在分支限界算法框架下，迭代树的同一层只需进行一次着色，然而，在不同层时，需要对新产生的候选顶点集进行贪心着色过程。贪心着色过程对顶点顺序具有部分的继承性。当着色完成后，颜色编号相同的顶点之间，原来的顺序并没有因着色而被破坏，先着色的顶点仍然在后着色的顶点之前。这样就使得下层迭代时，顶点是相对有序的。但是，这种顺序会在逐次迭代中失去，所以仅仅在初始阶段对顶点进行排序是不够的。考虑到排序操作的时间复杂度较高，必须权衡排序本身的效率负担与因排序所得的效率提升。下面分别介绍两种处理策略。

#### 2.4.1 静态顺序顶点辅助集

为了延续初始阶段顶点顺序的效果，在初始化之后，使用一个全局的顶点辅助集保存顶点的顺序。当候选顶点集需要重新着色时，根据顶点辅助集中的顺序，对候选集中的顶点进行着色。由于顶点辅助集保存了初始顶点顺序之后不再更改，因此称之为静态顺序顶点辅助集。虽然静态顺序与预期的候选集顺序不完全一致，但是作为一种近似估计，顶点辅助集较好地平衡了排序的耗费与收益。这便是MCS[]所采取的策略。MCS相比MCR增加了静态顺序顶点辅助集，以及后面即将提及的重着色技术。

#### 2.4.2 动态排序策略

相比静态顺序一经确定便不再更改的策略，动态排序策略在特定的时刻对顶点进行重新排序，使其更加适应当时的候选集。由于没有辅助顶点集，所以需要使用一些技巧来增强贪心着色过程对顶点顺序的继承性。

直觉上，当子图的顶点数量较多时，重新排序的效果会越好，因为贪心着色的继承性使得排序的影响得以蔓延到更多分支。另一方面，对顶点数量较多的图，排序所花费的时间相比于由于上界不紧而在无效分支上浪费的时间小得多。因此，我们应该对靠近搜索树根部的分支进行重新排序，而在离根部较远的地方放弃重新排序。

我们用level（层）表示从搜索树根部到当前叶子节点所进行的分支数（递归的次数）。在寻找最大团的过程中，我们必须动态地决定在搜索树的哪一个level 进行贪心着色前排序顶点。例如，对于密集图而言，最大团一般比顶点数量相同的稀疏图的最大团大。我们希望在顶点数量相同的情况下，贪心着色前的顶点排序在密集图上比稀疏图更加频繁。另外，我们也希望在边密度一致的情况下，贪心着色前的顶点排序在大图（顶点数量多的图）上进行的更频繁。

我们引入了全局变量S[level]和Sold[level]，分别表示从根部到当前level，并且包括当前level 所进行的步数总和，以及从根部到前一level，并且包括上前一level所进行的步数总和，其中Sold[level]是用来辅助S[level]的计算的。我们引入变量T[level]表示到当前level 为止所进行的步数与当前搜索树所进行的所有步数之间的比值，即T[level] =S[level] / ALL\_STEPS，其中ALL\_STEPS 是一个全局变量，算法每进行一个分支步，其值都增加1，我们在每个分支步中都重新计算这些值。另外，我们需要一个变量Tlimit来决定是否进行贪心着色前的顶点排序。当T[level] < Tlimit 时，进行贪心着色前的顶点排序，反之，当T[level] ≥ Tlimit 时，不进行该排序过程。

在每一level 中，我们的算法首先更新从根部到该level 中的步数总和S[level] :=S[level] + S[level - 1] - Sold[level]，并且更新前一level 的步数总和Sold[level] := S[level - 1]。如果算法在该level 时需要继续递归到下一level，则将S[level]的值加1。关于level 的总步数计算的例子见图2-1。

  
**图 2-1** level 的总步数计算。列表示 level，即从根部到当前位置的递归深度。算法的搜索路径以向右的箭头表示，而向左下的箭头则表示回溯。箭头处的数值表示S[level ](Sold level) 。S[3] = 5 是通过如下方式计算的， S[3] = S[3] + S[2] - Sold[3] = 2 + 4 - 2 = 4，因为递归需要继续前进（向右的箭头），S[3] = S[3] + 1 = 5，Sold[3] = S[2] = 4。

其中参数Tlimit的值由顶点度数的计算方式，顶点的排序准则，以及着色的策略共同决定。在MCQdyn算法[]中，Tlimit设定为0.025。

值得注意的是，贪心着色之后，算法不会将颜色编号No[v] < |Qmax| - |Q| + 1的顶点加入到当前团中，其中|Q|和Qmax分别为当前团和目前找到的最大团。为了加强着色过程对顶点顺序的继承性，MCQdyn算法[]在着色之后，颜色编号No[v] < |Qmax| - |Q| + 1的顶点仍按照着色前的顺序排列，而其他顶点则按照颜色编号排列。这样就使得着色之后颜色编码小的所有顶点的顺序能被后续迭代完整地继承。

MCQdyn算法是在MCR算法采用了动态排序策略所改进而得到的。

## 重着色

一个精确且获取代价小的上界，是减少不必要分支从而提升分支限界算法性能的关键。然而，减少分支数量也能通过其他方式进行。MCS[]所用到的重着色技术便是其中一种。

根据前文所介绍的贪心着色与顶点分支方式，颜色编码小于|Q­max| - |Q| + 1 的顶点不需要直接参与分支，因为仅由这些顶点加上当前团Q 是不可能生成比目前最优的团更好的结果的。我们称颜色编码小于|Qmax| - |Q| + 1 的顶点集是被动的，而颜色编码大于等于|Qmax| - |Q| + 1 的顶点集是主动的。重着色技术正是基于如下事实而设计的：被动顶点越多，所需要的分支数越少。重着色对主动顶点进行重新着色，使其由主动的变成被动的。但由于重着色是比较耗时的操作，因此我们在着色过程中，只对那些颜色编码等于目前的颜色数量的顶点进行重着色，即No[v] = maxno 的顶点，以取得减少分支带来的优化与减少分支所需的代价之间更好的权衡，其中maxno是在贪心着色过程的某一时刻，最大的颜色编码。

重着色过程如下：假设需要重着色的顶点为v，其当前颜色编码为k，我们首先在颜色编码为1, 2, ..., |Qmax| - |Q|的顶点集中，找到只有一个顶点与v 相邻的颜色编码k1，记这个顶点为v1，接着我们在k1+1，k1+2,...,|Qmax| - |Q|的顶点集中，找到可以放置顶点v1 的颜色编码k2，即第一个与v1 没有相邻顶点的颜色编码，然后将v 着色为k1，将v1 着色为k2。若不存在这样的k1 或者k2，则无法对顶点v 进行重着色，即保持v 原有的颜色编码。

## 四、Maxsat推理