华南理工大学

South China University of Technology

本科毕业设计（论文）

题目：最大团问题的分支限界算法

学　院　计算机科学与工程

专　业　计算机科学与技术

学生姓名　林锦坤

学生学号　200930582489

指导教师　郑运平

提交日期　2009 年 6 月 1 日

摘　要

在最大团问题的确定性算法中，分支限界算法是最常用的策略。上界精确性和分支策略是决定分支限界算法效率的关键所在。

首先介绍了最大团问题分支限界算法的基本框架，然后引入了确定问题上界最常见的方式，即贪心着色策略，并采用一种成为重着色（re_color)的技术改进贪心着色策略，以减少分支限界算法的分支节点数量。由于贪心着色效果的好坏取决于顶点被着色的次序，因此使用了顶点动态排序以及静态顺序的顶点辅助集的技术，以获得更好的分支限界算法的上界值，并在上界准确性和获得上界所需时间之间取得了较好的权衡。另外，初始化对于分支限界算法也有相当重要的影响，所以算法也采用了与 Carraghan 和 Pardalos(1990)[1]以及 Tomita 和 Kameda(2007)[2]类似的初始化工作。

对文中描述的算法 MCSD 以及 MCSMD，我们在随机图和 DIMACS 基准图上进行了实验，并与著名的 MaxCliqueDyn 算法、MCS 算法进行比较，这些算法分别单独采用了上述的部分优化策略，实现结果表明，MCSD 与 MCSMD 相比 MaxCliqueDyn 和 MCS 取得了显著的提升，这表明了将这写策略结合的方式是可行的。同时，我们也和当今最先进的最大团确定性算法——MaxSatClique 进行了比较，这个算法是基于最大可满足性问题推理优化的，实验结果表明我们的算法与 MaxSatClique 在性能表现上不相上下。

关键词 ：最大团问题；分支限界算法；贪心着色；动态排序；重着色

Abstract

Among the exact algorithm of finding maximum clique problem, branch-and-bound algorithm is the most commonly used strategy. The performance of branch-and-bound algorithm's efficient lies on the tightness of upper bound and the method of branch.

In this paper, we first introduce the base frame of branch-and-bound algorithm for maximum clique problem. And then present the most common use way for getting upper bound, greedy coloring, and improve it with re_color which aims at decreasing the number of branching nodes. As the effect of greedy coloring depends on the sequence of vertex being colored, therefore, we make use of dynamic vertices sorting and adjunct static ordered vertices set to get tighter upper bound. With these two technology, we get a good trade-off between the tightness of upper bound and the time consume to get such upper bound. In addition, the initialization is important in branch-and-bound algorithm, so we use the similar initialization as Carraghan and Pardalos(1990)[1] and Tomita and Kameda(2007)[2].

For our algorithms MCSD and MCSMD, computational experiments were taken with DIMACES benchmark graphs and random graphs. We compare the famous algorithm MaxCliqueDyn and MCS with ours, these algorithms only use part of the improving strategy described above. As the experiments show, MCSMD and MCSD achieve Significant improvements comparing with MaxCliqueDyn and MCS, and the result indicate that the combination of improving strategy above is feasible. We also compare our algorithm with the state-of-art exact algorithm of maximum clique problem, MaxSatClique, which is based on maxsat reason rule, the experiment indicate that the performance of them is about the same.

Keyword : Maximum clique problem, branch-and-bound algorithm, greedy coloring, dynamic sorting, recolor

目 录

摘 要.....2
Abstract.....3
第一章 绪 论.....5
1.1 选题背景.....5
1.2 国内外研究现状及发展动态分析.....5
1.3 最大团问题的定义.....7
1.4 本文主要工作及内容安排.....8
第二章 分支限界算法.....9
2.1 符号定义.....9
2.2 最大团分支限界算法的基本框架.....9
2.3 分支与剪枝.....10
2.3.1 染色数作为上界的原理.....10
2.3.2 贪心着色过程.....11
2.3.3 顶点分支顺序.....12
2.3.4 重着色.....13
第三章 动态着色策略.....15
3.1 顶点的初始化顺序.....15
3.2 静态顺序的顶点辅助集.....16
3.3 邻接矩阵的重构.....17
3.4 对顶点顺序的动态排列.....17
3.5 无需参与分支的顶点.....19
第四章 实验结果.....23
4.1 用户时间.....23
4.2 对随机图的结果.....24
4.3 对 DIMACS 基准图的结果.....26
第五章 结论.....30
5.1 本文研究工作总结.....30
5.2 课题相关展望.....31
参考文献.....32
致 谢.....35

第一章 绪 论

本章首先介绍了最大团问题的基本背景，说明了最大团问题在实际与理论上的重要性。然后详细分析了最大团问题在国内外的研究现状及发展动态，介绍了在这个问题上算法的成功与不足。最后介绍了最大团问题的基本定义与本文的具体工作安排。

1.1 选题背景

最大团问题作为一个非常重要的 NP 难组合问题，在现实世界的许多应用当中有着相当广泛的运用，许多实际问题可以被描述成最大团问题（Bomze et al. 1999）[3]。在生物化学和基因组领域中，基因组映射数据的整合、非重叠局部准线、分子结构的匹配和对比、蛋白质对接等问题都可以用团检测模型来表示（Butenko et al.2006）[4]。另一个问题是寻找一个尽可能大的二进制编码，使其可以用来纠正二进制字（向量）一定数量的错误（Brouwer et al. 1990[5]; Sloane 1989[6]）。在这些二进制字中必然存在两个字在一定数量的位置上不同，所以误写的字可以被检测出来并进行纠正(Du 和 Pardalos 1999)[7]。一个团描述了一种编码的可行向量集。错误纠正编码被运用到了蜂窝电话，高速调制解调器，和 CD 播放器（比较校验和的时候）。寻找社会网络中大的聚合群体被运用于犯罪网络的分析中。另一个运用在于股市图标的团分析中。此外，最大团问题还被用于方案选择，分类理论，计算机视觉等众多领域中[8-15]。

由于最大团问题在计算上与最小顶点覆盖，最大独立集等 NP 难问题等价，并且可以在多项式时间内规约为许多著名的难题，如 Hamilton 回路问题、Hamilton 路径问题、货郎担问题等，Crescenzi，Fiorini 和 Silvestri 证明了如果最大团问题存在多项式时间的近似算法，那么任何的 NP 问题都能在准多项式时间内求解（Crescenzi et al. 1991）[16]，因此最大团在理论上也具有很大的借鉴意义。

1.2 国内外研究现状及发展动态分析

一个枚举图中所有的团的著名的算法是由 Bron 和 Kerbosch（1973）[17]提出的这个算法找出任意图中的所有极大团。因为这个算法只用到分支技术，而没有使用减少分支的限界策略，所以在较小的密集图上找出最大团将会耗费大量的运行时间。 Bron 和 Kerbosch 算法时间复杂度的最坏情况是$O(3^{\frac{n}{3}})$（Jenelius et al. 2006）[18]。

对与最大团问题，其中一个最早的分支限界算法由 Carraghan 和 Pardalos(1990)[1]年提出。该算法的主要思想是使用限界策略，当发现某个分支不可能产生比当前找到的尺寸最大的团还要大的团时，及时地裁剪掉该分支。在该算法中使用的限界技术很简单：在候选子图中，最大团的尺寸不会比该候选集顶点的数目大。该算法的另一个思想是对

顶点进行特殊的排序，以减少搜索树的大小。Carraghan 和 Pardalos(1990)[1]提出的这种排序方法也被用到了最新的一个确定性算法——Tomita(2010)[19]等人提出的 MCS 算法中。

Fahle(2002)[20]提出了一个更加高效的限界策略。该策略的主要思想是使用染色数作为最大团的上界。更加具体的说，一个图的候选顶点集按照一种启发式的顺序着色方法进行染色，其所用的颜色数（染色数的上界）作为最大团的上界。这个算法也采用了两种主要的过滤技术，这些技术主要基于以下两种现象：首先，如果候选顶点集中存在一个顶点，与候选顶点集中所有的顶点都邻接，那么这个顶点将在很多分支中被包含于团内。这个顶点应该被立即添加到当前团中，从而不需要参与分支。其次，如果候选集中的一个顶点在该候选集的度数加上当前团的尺寸，仍小于当前找到的尺寸最大的团，则应该将其排除掉，从而也不需要参与分支。

Tomita 和 Seki(2003)[21]提出的 MCQ 算法使用了不仅将着色思想作为限界策略，还将其作为分支策略。初始时顶点按照度数降序排列。在每次分支时，对候选子图进行一个标准的贪心顺序着色过程，然后优先选择颜色最大的顶点（即被赋予最大自然数的那个顶点）进行分支。

针对贪心着色过程的结果与被着色顶点的顺序有着密切的关系，Konc 和 Janezic(2007)[22]提出的算法 MaxCliqueDyn 采用了一种动态排序的思想，即在分支搜索树中，对靠近根部的节点，在贪心着色开始前重新排列候选顶点集，而在离根部较远的节点则仍然采用旧的顶点顺序（即由贪心着色过程继承下来的顺序）。

Tomita 和 Kameda(2007)[2]提出的 MCR 算法和 Tomita et al.(2010)[19]提出的 MCS 算法是 MCQ 算法的进一步改进。MCR 算法和 MCQ 算法的不同之处仅在于在算法初始时顶点的次序。在 MCR 中，顶点按照 Carraghan 和 Pardalos(1990)[1]提出的排序方法进行排列。在 MCS 算法中，一种称为重着色的方法被用来改进着色过程，它尝试对颜色最大的顶点进行重新着色，赋予其更小的一种颜色。另一个最新的算法是由 Li 和 Quan(2010a)[23]以及其提升版（Li 和 Quan 2010b）[24]提出，它们使用了基于 Max-Sat 所获得的上界取代了着色得到的上界。对于 DIMACS 图，根据已发表论文中显示的结果，MCS 和基于 Max-Sat 的算法在所有的确定性算法中性能最佳。

此外，也有通过将最大团问题转化为 0-1 规划问题[25]，或通过数学方式转化为矩阵计算进行求解[26]的算法，并取得了不错的效果。在算法实现技巧上，San 等人使用了位编码以及并行编程方式[27-29]，使得算法的效率得到了可观的提升。另外，最大团问题的随机算法也有了相当好的进展[30-36]。

1.3 最大团问题的定义

最大团问题是指在一个简单无向图中寻找顶点数最多的一个团的问题。在本文中，我们使用下列定义：

- 图——简单无向图 G = (V, E)，其中 V = {1, 2, ..., n}表示图的顶点集，$E \subseteq V \times V$ 表示图的边集
- 邻接矩阵——表示图 G 的矩阵 $A = (a_{i,j})$，$\forall i, j \in V$，如果$(i, j) \in E$，则 $a_{i,j} = 1$，如果$(i, j) \notin E$，则 $a_{i,j} = 0$。
- 补图——给定一个图 G(V, E)，它的补图定义为$\overline{G}(V, \overline{E})$，其中 E = {(i, j) | (i, j) $\in (V \times V) \setminus E$}。
- 完全图——任意两个顶点都是邻接的图，即$\forall i, j \in V$，有$(i, j) \in E$。
- 团——图 G(V,E)顶点集 V 的一个子集 C，其中 C 中任意两个顶点都是邻接的。
- 最大团——顶点数最大的一个团。
- 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)顶点集 V 的一个子集 S，其中 S 中任意两个顶点都是不邻接的。
- 候选顶点——可以用来扩展当前团的顶点，即与当前团所有顶点都邻接的顶点。
- 着色——赋予图的每个顶点一种颜色（自然数），使得具有相同颜色的顶点两两不相邻接。
- 颜色——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。
- 团数——最大团的顶点数量。
- 顺序着色——对顶点一个一个地进行着色，其中顶点已经按照一定次序排列。
- 合理（紧）着色——对顶点进行顺序着色，该过程中只要可以有可能，就对顶点赋予已经存在的颜色，如果找不到这样的颜色，这采用新的颜色。
- 贪心着色——对顶点进行顺序着色，其中对每个顶点赋予最小的可能颜色值。

最大团的大小记为$\omega(G)$，最大团问题有很多种数学表示公式，其中的一种表示如下：

$$\max\{\sum^{n} x_i + x_j \le 1, \forall (i, j) \in \overline{E},\ x_i \in \{0,1\},\ i = 1,\ldots, n\}$$

在这里，如果顶点 i 属于最大团，则 $x_i = 1$，否则如果 $i \notin C$，则 $x_i = 0$。

最大独立集，即给定一个图，找出其顶点数最多的独立集的问题。最大独立集与最大团问题在计算上是相互等价的两个问题，因为图 G 的一个团等于图 G 补图的一个独立集，反之亦然。（如图 1-1 中，例图的最大团{1,2,3,4}，及其补图的最大独立集{1,2,3,4}）

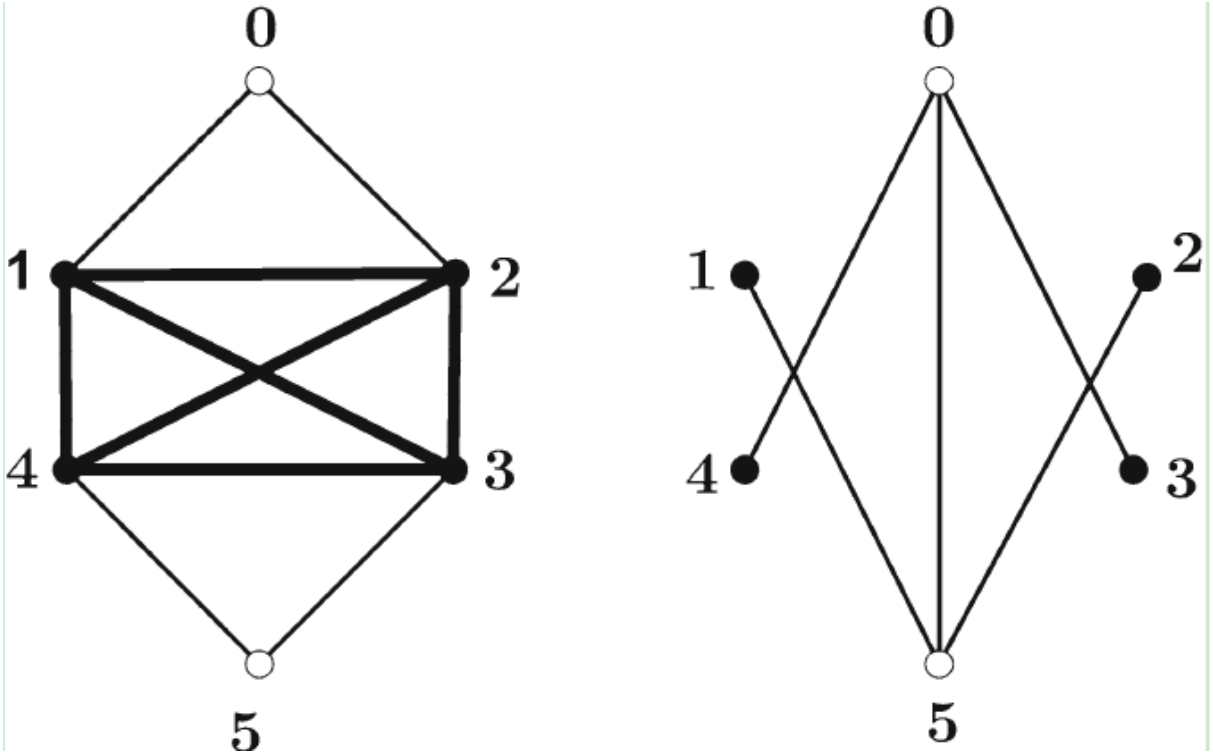

图 1-1 最大团问题和最大独立集问题

1.4 本文主要工作及内容安排

本文首先介绍了最大团问题分支限界算法的基本框架，然后引入了确定问题上界最常见的方式，即贪心着色策略，并采用一种成为重着色（Tomita et al. 2010）[19]的技术改进贪心着色策略，以减少分支限界算法的分支节点数量。由于贪心着色效果的好坏取决于顶点被着色的次序，因此本文使用了顶点动态排序（Konc 和 Janezic 2007）[22]以及静态顺序的顶点辅助集的技术（Tomita et al. 2010）[19]，以获得更好的分支限界算法的上界值，并在上界准确性和获得上界所需时间之间取得了较好的权衡。另外，无论算法的初始化对分支限界算法也有相当重要的影响，所以本文介绍的算法采用了与 MCR 算法（Tomita 和 Kameda 2007）[2]类似的初始化工作。根据顶点动态排序的不同，本文所提出的两个算法分别命名为 MCSD 和 MCSMD。

本文将按照如下方式进行组织。在第二章中描述分支限界算法以及着色策略。顶点集的初始化与动态排序策略将在第三章论述。第四章描述团检测与团顶点检测及其作用，即避免对属于当前团的顶点进行分支。第五章是计算实验结果及本文结论。

第二章 分支限界算法

本章首先对算法中所使用到的符号进行定义，然后介绍了最大团问题的分支限界算法基本框架，并详细介绍了算法中所使用的贪心着色策略，论述了其在算法中是如何同时作为分支技术与剪枝技术的。最后，我们通过重着色策略对贪心着色策略进行了改进。

2.1 符号定义

在我们算法所有过程的描述中所用到的符号遵循如下定义：

- V——图的顶点集列表
- E——图的边集列表
- $\overline{E}$——补图的边集列表
- A——图的邻接矩阵
- R——图的候选顶点集列表
- Ra——图的候选顶点集列表，其中顶点集的顺序与上次排序过程后的顺序一致
- Rp——临时变量，图的候选顶点集列表
- Rap——临时变量，图的候选顶点集列表，其中顶点集的顺序与上次排序过程后的顺序一致
- Q——全局变量，当前团
- QMAX——全局变量，目前找到的最大的团
- |X|——集合 X（数组，列表，团）)的大小
- C[k]——被着色为的顶点集
- C[k][i]——C[k]中的第 i 个顶点
- v——图的一个顶点
- No[v]——顶点 v 的颜色
- $N_R(v)$——顶点 v 在候选集 R 中的邻居
- $deg_R(v)$——顶点 v 在候选集 R 中的度数
- U®)——R 中最大团大小的上界
- ω(R)——R 中最大团的大小

2.2 最大团分支限界算法的基本框架

分支限界算法分为分支部分与限界部分，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被选中的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。

```
procedure EXPAND(R)
begin
    while R ≠ Ø do
        v := a vertex of R
        if |Q|+ UB(R) > |Qmax|              //prune
            then
            Q := Q ∪ {p};
            Rp := NR(v);
            if Rp ≠ Ø then
                EXPAND(Rp,)
            else return
            fi
            Q := Q - {p}
        else return
        fi
        R := R - {p}
    od
end {of EXPAND}
```

图 2-1 分支限界算法的基本框架

2.3 分支与剪枝

2.3.1 染色数作为上界的原理

着色过程所需要的颜色数（大于或小于染色数），可以作为最大团问题的上界（命题1）。

命题 1（Balas 和 Yu 1986）[37] 任意图的最大团尺寸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 个顶点的团只能用 k 中颜色进行着色，因为这些顶点都是两两邻接的，因而任意两个顶点不能具有同一种颜色。注意到一个图的染色数可以任意地大于它的团数（即最大团的大小）。Mycielski 的定理[38]证明了

这一事实。

定理 1（Mycielski 1955）[38] 对于任意自然数 n，都存在有限个不包含三角形的图，不能用 n 种颜色进行着色。

显然，任意一个不包含三角形的图的最大团不可能大于 2。但是该图的染色数可以远大于最大团的大小。图 2-2 显示了不包含三角的图，其染色数分别为 2,3,4，它们的最大团的大小均为 2。

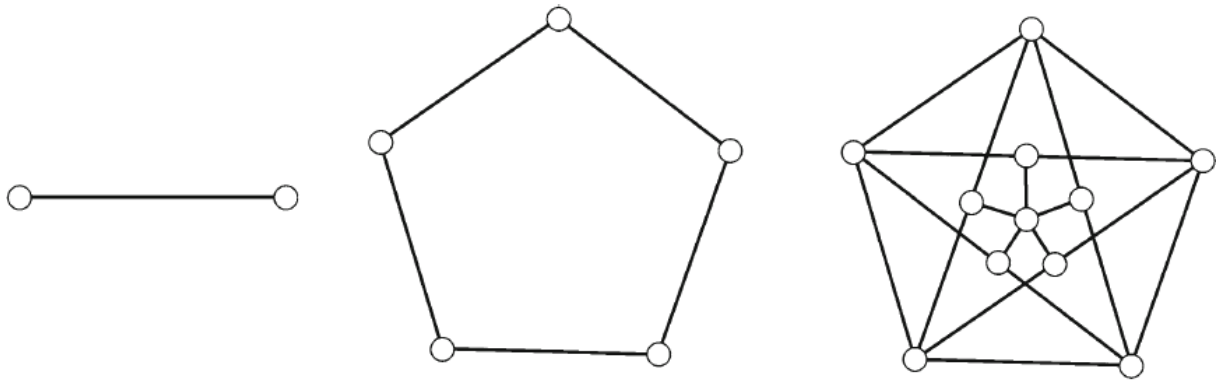

图 2-2 染色数为 2,3,4，最大团大小为 2 的 Mycielski 图

2.3.2 贪心着色过程

贪心着色过程被 Tomita 和 Seki(2003)[21]的 MCQ 算法采用，该过程遵循下列原则：

- 如果$(v_1, v_2)\in E$，则 $No[v_1]\neq No[v_2]$
- $No[v] = 1$，或者 $No[v] = k > 1$，则存在顶点 $v_1\in N_R[v]$，$v_2\in N_R[v]$，...，$v_{k-1}\in N_R[v]$，使得 $No[v_1] = 1$，$No[v_2] = 2$，....，$No[v_{k-1}] = k-1$。

于是我们知道

$$\omega(R) \leq \mathrm{Max}\{No[v] \mid v\in R\},$$

即 $\mathrm{Max}\{No[v] \mid v\in R\}$可以作为候选集 R 中最大团大小的上界，于是在分支限界算法中，当$|Q| + UB(R) > |Q_{max}|$时，可以得知该分支不可能获得更好的团，因此可以减掉这个分支。

在贪心着色的过程中，顶点被一个一个的赋予一种颜色值，其方法如下：假设顶点集 $R = \{v_1,v_2,...,v_n\}$按照这样的顺序排列。首先，让 $No[v_1]=1$。接着，如果 $v_2\in N_R[v_1]$，则让 $No[v_2]=2$，否则让 $No[v_2]=1$，……，如此进行。如图 2-3 中 Color 部分所示。

```
procedure COLOR-SORT(R, C)
begin
{COLOR}
    maxno := 0;
    C[1] := Ø;
    while R ≠ Ø do
        v := the first vertex in R;
        k := 1;
        while C[k] ∩ N_R(v) ≠ Ø
            do k := k + 1 od
        if k > maxno then
            maxno := k;
            C[maxno] := Ø;
        fi
        No[p] := k;
        C[k] := C[k] ∪ {p};
        R := R - {p};
    od
{SORT}
    i := 1;
    for k := 1 to maxno do
        for j := 1 to |C[k]| do
            R[i] := C[k][j];
            i := i + 1;
        od
    od
end {of NUMBER-SORT}
```

图 2-3 贪心着色与顶点分支顺序

2.3.3 顶点分支顺序

贪心着色过程不仅仅是一种限界技术，同时也可以作为一种不错的分支策略。因为最大团的每个顶点的颜色是不同的，即最大团只由每一种颜色值中小于 1 个的顶点组成。按照上述贪心着色方法，被赋予第 i 种颜色的顶点，都能在被赋予第 j (< i)种颜色的顶

点中找到其邻居，因而颜色数越大的顶点属于最大团的可能性越大。最好的分支方式是每次都选中了属于最大团的顶点，然而这在实际算法中是很难做到的，一种次优的方案是每次选取可能性最大的顶点，即每次选取颜色值最大的顶点。Tomita 和 Seki(2003)[21]将顶点分支顺序与限界过程无缝结合了起来，即着色完毕之后，按照颜色值的大小顺序排列顶点，更具体的，颜色值为 1 的顶点在颜色值大于 1 的顶点之前，并且颜色值相同的的顶点的顺序与其被染色的顺序一致，如图 3 中 sort 部分所示。按照这样的分支方式，我们在搜索树的同一层中只需着色一次。

2.3.4 重着色

一个精确而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，减少分支数量也能够通过其他方式进行，Tomita et al.(2010)[19]提出的重着色技术便是其中一种。

```
procedure Re-COLOR(v, R, C)
begin
    for k1 := 1 to |QMAX| - |Q| do
        if |C[k1] ∩ NR[p]| then
            v1  = the element in C[k1] ∩ NR[v]
            for k2 := k1 + 1 to |QMAX| - |Q| do
                if C[k2] ∩ NR[v] = φ then
                    {Exchange the Colors of v and v1}
                    C[No[v]] := C[No[v]] - {v};
                    C[k1] := (C[k1] - {v1}) ∪ {v};
                    C[k2] := C[k2] ∪ {v1};
                    return
                fi
            od
        fi
    od
end {of Re-COLOR}
```

图 2-4 重着色过程

按照本文之前所介绍的贪心着色与顶点分支方式，颜色值小于|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。我们称颜色值小于|QMAX| - |Q| + 1 的顶点集是被动的，而颜色值大于等

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，只对那些颜色值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。

重着色的过程如图 2-4 所示，假设需要重着色的顶点为 v，其当前颜色值为 k，我们首先在颜色值为 1,2,...,|QMAX| - |Q|的顶点集中，找到只有一个顶点与 v 相邻的颜色值 k_1，记这个顶点为 v_1，接着我们在 k_1+1，k_1+2,...,|QMAX| - |Q|的顶点集中，找到可以放置顶点 v_1 的颜色值 k_2，即第一个与 v_1 没有相邻顶点的颜色值，然后将 v 着色为 k_1，将 v_1 着色为 k_2。若不存在这样的 k_1 或者 k_2，则无法对顶点 v 进行重着色，即保持 v 原有的颜色值。

第三章 动态着色策略

本章首先论述了顶点顺序对着色策略的重要影响，从而引出了我们的算法 MCSD 与 MCSMD 的顶点初始化工作。接着，我们使用了一种称为静态顺序的辅助顶点集的技术，使得顶点的初始顺序能够在算法的整个过程中得以保持，另外，为了充分利用计算机内存的局部性原则，我们对图的邻接矩阵进行了重构。本章还详细介绍了在分支过程中对顶点顺序进行动态排序的策略，该策略使得在算法分支过程中，顶点的顺序与子图保持更好的一致性，为了同时使用该策略与静态顺序的顶点辅助集，我们论述了如何将两者有机结合起来的方法。最后，我们通过一些简单的事实对算法进行了改进，使其可以避免一些不必要的分支。

3.1 顶点的初始化顺序

贪心着色过程（包括重着色）可以得到被着色的顶点集的最大团大小的上界，由上面所描述的着色过程可以得知，顶点被着色的顺序对于着色后所得到的上界（即所需的颜色数）有着重要的影响。直觉上限制越多的顶点应该首先被着色，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，可以得到更紧确的上界。

然而，我们不仅把贪心着色过程作为获取上界的手段，同时也将其作为分支策略使用。顶点被分支的次序与其顶点的颜色值大小顺序一致，且在同一种颜色值的顶点中，越晚被着色的顶点越早被选择，因此，我们对顶点的排序方式采用了与 Carraghan 和 Pardalos(1990)[1]以及 Tomita 等人(2010)[19]类似的方法。

我们将顶点排列成如下顺序 V={V[1],V[2],...,V[n]}，使得图 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。

如果所有的顶点{V[1],V[2],...,V[i]}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的最后。该初始化的过程如图 3-1 所示。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，这个负担是相当小的。

```
procedure   SORT(R)
    begin
        i := |R|;
        Rmin := set of vertices with minimum degree in R
        while |Rmin|  ≠  |R| do
            v := Rmin[1];
            V[i] := v;
            R := R - {v};
            i := i - 1;
            for j := 0 to |R| do
                if R[j] is adjacent to i then
                    degR[R[j]] := degR[R[j]] - 1;
                fi
            od
            Rmin := set of vertices with the minimum degree in R
        od
    end {of   SORT}
```

图 3-1 顶点初始化排序

3.2 静态顺序的顶点辅助集

贪心着色过程之后，顶点的被组织成按颜色值大小的顺序排列，由于同属于一种颜色的值的顶点的相对位置，与这些顶点在着色之前一致，因此贪心着色过程使得顶点的初始顺序被部分地继承下来了。这种继承使得顶点的初始化顺序的影响贯穿整个算法过程，因而也加大了初始化顺序的重要性，另外，继承下来的顺序一般认为是有助于贪心着色过程产生更好的分支和限界效果。

为了保持顶点的相对有序，比采用贪心着色过程来继承更加直接的方法，是采用一个辅助的顶点集，在算法的全过程中保持这个顶点的相对顺序不变，通过借助该辅助顶点集，我们完全保留了顶点初始化时的相对顺序，因而相比继承更加彻底，引入了该辅助顶点集之后，贪心着色过程对顶点着色的次序按照该顶点集中顶点的先后顺序进行，大大加强了贪心着色过程的效果，即有了更好的限界策略和分支策略。

为了减少不必要的遍历，在每个分支中，我们只在辅助顶点集中保留候选集的顶点，则各个分支中的辅助顶点集只使用于其所属的分支，即辅助顶点集为局部变量。由于辅助顶点集在各个分支中不尽相同，因而我们需要用到动态分配的内存，即堆（对 C++而

言，即为 new 操作），并且每个分支都需要一个这样的堆空间。动态分配内存是一个耗时的操作，尤其对于最大团分支限界算法这种分支数并不少的算法而言，动态分配内存耗费的 CPU 时间是相当可观的。我们可以使用栈内存代替堆内存，为此我们需要为每个分支都分配一个足够大的空间存储辅助顶点集，然而这将会浪费相当的内存空间。而且我们的测试用例不能大于预定的大小，使算法失去普遍性。在本文中，我们将继续采用堆，而放弃使用栈。

3.3 邻接矩阵的重构

在我们的算法中，每一个图都被表示成邻接矩阵的形式。在贪心着色的过程中，顶点按照度数降序的次序进行着色。考虑到这种情况，在本文描述的算法初始时，我们首先对顶点的邻接矩阵进行重构，使得其与 3.1 节所描述的顶点初始顺序一致，具体的，第 i 行（列）代表顶点初始排序之后的第 i 个顶点。邻接矩阵重构的过程如图 6 所示。邻接矩阵重构之后，使得我们在贪心着色过程能够更有效地利用计算机的缓存，因为这种排列顺序恰好符合计算机内存使用的局部性原理，两个先后着色的顶点被安排得尽量相近，即尽量在同一个内存页中，以减少内存换页的次数。

```
procedure Permute-Adjacency-Matrix()
begin
    A' := A;
    for i := 1 to |Va| do
        for j := 1 to |Va| do
            A[i][j] := A'[Va[i]][Va[j]];
        od
    od
    for i := 1 to |Va| do
        Va[i] := i;
    od
end { of Permute-Adjacency-Matrix}
```

图 3-2 邻接矩阵的重构

3.4 对顶点顺序的动态排列

静态顺序的顶点辅助集保存的顶点相对顺序是对于原图而言的，当我们对某个顶点进行分支时，候选顶点集被更新为原候选顶点集中与该顶点相邻接的顶点。在新的候选

集当中，顶点的正确的相对顺序往往跟静态顺序的顶点辅助集中的顺序不一致，即候选集中度数最小的顶点，与静态顺序的顶点辅助集中最后的一个顶点往往不是同一个。为了使贪心着色过程获得最佳的效果，理想情况下，候选集中的顶点应该在每次贪心着色过程前都进行一次排序。

然而，对顶点进行排序是相当耗费时间的操作，因为对顶点度数的计算以及排序的过程都需要 $O(|R|^2)$的计算时间复杂度。为了更好地权衡贪心着色过程效果提升带来的算法效率改进与获得该改进所需要的时间，一种折中的方案是，只对部分分支中的顶点进行着色前排序（Konc 和 Janezic 2007）[22]。直觉上，对顶点较多的生成子图 G(R)进行着色前排序会更好一些，因为对顶点数量越多的图，排序的影响就越大（考虑贪心着色过程对顶点顺序的继承性，或者静态顺序的顶点辅助集策略的采用，使得顶点的顺序被漫延到了后续的分支过程）。另一方面，对顶点数量多的图，贪心着色前排序可以得到更优的上界所耗费的时间，相比由于上界不够紧确而在错误的分支（即不可能产生更大的团的分支）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着色前排序，而在远离根部的位置则放弃排序操作。

需要注意的是，有了动态排序策略，并不意味着我们需要放弃静态顺序的顶点辅助集，相反，我们的动态排序策略正是运用于这样的顶点辅助集上的。

在本文描述的算法中，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 S_{old}[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 S_{old}[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 T_{limit} 来决定是否进行贪心着色前的顶点排序。当 $T[level] < T_{limit}$ 时，进行贪心着色前的顶点排序，反之，当 $T[level] \geq T_{limit}$ 时，不进行该排序过程。

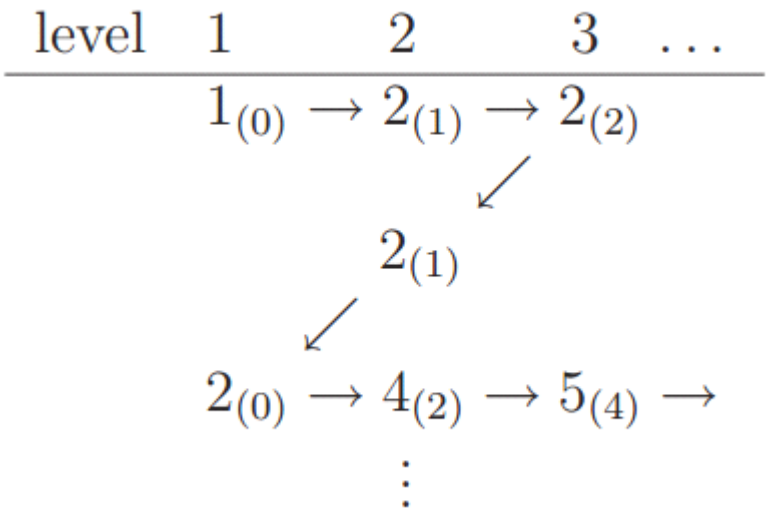

图 3-3 level 的总步数计算。列表示 level，即从更部到当前位置的递归深度。算法 的搜索路径以向右的箭头表示，而向左下的箭头则表示回溯。箭头处的数值表示 $S[level]_{(S_{old}[level])}$。S[3] = 5 是通过如下方式计算的， S[3] = S[3] + S[2] - S_{old}[3] = 2 + 4 - 2 = 4，因为递归需要继续前进（向右的箭头），S[3] = S[3] + 1 = 5，S_{old}[3] = S[2] = 4。

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] - S_{old}[level]，并且更新前一 level 的步数总和 S_{old}[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 3-3。

T_{limit} 的值由顶点度数的计算方式，顶点的排序准则，以及着色的策略决定，在 Konc 和 Janezic（2007）[22]中，顶点度数使用邻接矩阵计算、顶点按照度数降序排列，而着色则采用普通的贪心着色策略，在这种情况下，Konc 和 Janezic[22]通过实验找出的最优 T_{limit} 值为 0.025。在本文中，顶点度数的计算与 Konc 和 Janezic（2007）[22]相同，着色策略我们采用的是贪心着色策略加上重着色技术，而在顶点排序准则上（在分支过程中，而非初始化），我们采用两种方案，即按顶点降序排列的方式，以及按本文 3.1 节所述的排序方式。原则上，对 T_{limit} 进行精心的选择，能够使动态排序策略的效果达到最优，然而在本文中，我们将继续沿用 Konc 和 Janezic（2007）[22]的值，T_{limit} = 0.025。

3.5 无需参与分支的顶点

通过染色数来定义团是完全有可能的：团是一个染色数等于其顶点数的图。根据这点，我们可以通过着色发现一个图是团（命题 2）。

命题 2 如果一个有 k 个顶点的图在合理染色过程中只用到 k 种颜色，那么这个有 k 个顶点的图是完全的，即这些顶点形成了一个团。

用反证法进行证明：假设这样的图是不完全的。这以为着它至少拥有两个不相邻接的顶点。记这两个顶点为 i 和 j，并且假设在我们的合理着色过程中，顶点 i 在顶点 j 之前被着色。因为总共用了 k 种颜色对 k 个顶点进行着色，因此每个顶点都独自占用一种颜色。然而在我们的合理着色过程中，顶点 j 可以被着色为与顶点 i 同样的颜色，这与使用了 k 中颜色的条件是矛盾的。

度数越高的顶点经常是形成更大的团的元素。例如一个与图中所有其他的顶点都邻接的顶点是所有极大团（包括最大团）的元素。这样的顶点在对图进行贪心着色后被快速的找到（参见命题 3 和命题 4）。

命题 3 如果一个顶点与图中所有的顶点都邻接，那么这个顶点在图的任何着色当中，都独自占用了一种颜色。

证明：任何其他顶点都不能在着色过程中被染上与该顶点同样的颜色，因为它们与这个顶点是相邻接的。

命题 4 对于任意贪心着色过程，如果一个顶点独自占用了一种颜色，那么它将与所有颜色值比它大的顶点相邻接。

通过反证法证明：假设存在一个顶点 i 独自占用了一种颜色 k，并且顶点 j 是颜色值比 i 大的且与 i 不相邻接的顶点。则顶点 j 应该被着色为颜色 k 或者其他比 k 小的颜色，因为除了顶点 i 之外，没有其他顶点被染色成 k，而顶点 i 与顶点 j 是不相邻接的，按照贪心着色过程，我们总是对一个顶点染色为可能的最小的颜色值。这与顶点 j 被着色成比 k 大的颜色值是矛盾的。

在贪心着色过程完成之后，我们检查着色所用的颜色数与顶点的个数是否一致，如果一致，则可以确定这些顶点形成了一个团，直接将它们加入到当前团之中，不必对它们进行分支。团检测的过程如图 3-4 所示。

```
procedure Detect_Clique(R)
begin
    maxno := max{k | |C[k]| ≠ 0};
    if |R| = maxno then
        if |Q| + |R| > |QMAX| then
            QMAX := Q ∪ R;
        fi
        for i := 1 to |R| - 1 do
            ALL_STEPS := ALL_STEPS + 1;
            S[level] := S[level] + 1;
            level := level + 1;
            S[level] := S[level] + S[level - 1] - S_old[level];
            S_old[level] := S[level - 1];
        od
        return true;
    fi
end { of  Detect_Clique }
```

图 3-4 团检测

与 df 算法（Fahle 2002)[20]以及 Mikhail Batsyn（2013）[39]的算法一样，我们在每次着色过后，检查是否有候选顶点与所有其他的候选顶点都相邻接，从而将其直接添加到当前团中，避免对这些顶点进行分支。与 df 算法不同的是，我们能够使用命题 3 和命题 4 快速的检查出这些顶点，因为我们只需要检查那些独自占用一种颜色值得顶点，并且对于这样顶点，我们只需检查颜色值比它小的顶点是否都与其相邻接。

为了保持第三章中动态着色策略的一致性，我们在检测顶点是否组成团以及检测是否可以将某个顶点直接添加进当前团的过程中，需要保持对 S[level]以及 S_{old}[level]、ALL_STEPS 的更新。另外，在团顶点检测的过程中，由于顶点只是部分地加入到当前团中，所以其他顶点仍然将参与分支，因此除了在候选顶点集中，将那些可以直接添加到当前团的顶点删除掉外，我们还必须将其从静态顺序的辅助顶点集中将这些顶点删除。

在团顶点检测过程中，我们必须记录被检测出来的可以直接添加到当前团的那些顶点，以便在递归回溯时删除这些顶点，因为这些顶点并不是通过直接参与分支被加进来的。例如在某个分支过程中，顶点 i 和顶点 j 被检测出可以直接添加到当前团中，在接下来的迭代中，顶点 k 也被加入到了当前团中。如果在回溯时需要删除顶点 k，且需要

进一步回溯时，那么顶点 i 和顶点 j 也需要从当前团中被移除。我们使用全局变量 CLIQUE_VERTEX 记录这些顶点。团顶点检测的过程如图 3-5 所示。

```
procedure Detect-Clique-Vertices(R)
begin
    maxno := max{k | |C[k]| ≠ 0};
    for i := 1 to maxno do
        if |C[k]| > 1 then
            continue;
        fi
        v = C[k][1];
        if v adjacent to all vertices with color less than k then
            Q := Q + v;
            CLIQUE_VERTEX := CLIQUE_VERTEX + v;
            R := R - v;
            Ra := Ra - v;
            if |R| > 0 then
                    ALL_STEPS := ALL_STEPS + 1;
                S[level] := S[level] + 1;
                level := level + 1;
                S[level] := S[level] + S[level - 1] - Sold[level];
                Sold[level] := S[level - 1];
            fi
        fi
    od
end { of Detect-Clique-Vertices}
```

图 3-5 团顶点检测

第四章 实验结果

本文所描述的算法是主要从 MaxCliqueDyn（Konc 和 Janezic 2007）[22]以及 MCS（Tomita et al. 2010）[19]改进而形成的。因而在本章中，我们的主要比较对象是 MaxCliqueDyn（Konc 和 Janezic 2007）[22]以及 MCS（Tomita et al. 2010）[19]，据我所知，MaxCliqueDyn 和 MCS 是最大团确定性算法领域中最先进的两个算法之一。另外，为了确认顶点初始化对算法性能的影响，我们在 MaxCliqueDyn 的基础上加上初始化操作，并让这个算法参与我们的计算实验比较。

本文在动态排序中，使用了两种排序的准则，一种是按照顶点度数降序排列，另一种则采用了与 Carraghan 和 Pardalos(1990)[1]以及 Tomita 等人(2010)[19]类似的排序方法，在这两个算法当中，都采用了本文其他章节所采用的全部技术，并且采用 gcc -O3 进行编译，我们分别称这两个算法为 MCSD 和 MCSMD。

另外一个最先进的算法，是 Li 和 Quan （2010b）[24]提出的 MaxCLQ 算法，这个算法是基于最大满足性问题推理规则得出的。我们也将这个算法加入计算实验的比较之中。

4.1 用户时间

在最大团的准确性算法的对比当中，为了方便引用不同论文的数据，我们用一个基准程序运行若干基准测试图，从而得出我们所使用的计算机的 “用户时间”。然后根据其他论文中的用户时间，便可以知道我们所用机器的运算速度，与其他论文作者所使用的机器运算速度之间的比值关系，从而可以将他们的结果规约到本文中使用。

我们所使用的是 i5 2.5GHz 的 CPU，以及 4GB 内存的计算机。在该计算机中，运行基准程序 dfmax，对 DIMACS 测试图例 r100.5，r200.5，r300.5，r400.5，r500.5，所耗费的时间分别是 0，0.02，0.19，1.19，4.48。由于对 r100.5 的运行时间是 0，我们无法通过比值的方式得出我们的计算机与其他机器速度之间的比例，因此我们不考虑该测试图。其他机器的运行时间，以及这些时间与我们机器所用的时间的比值如表 4-1 和表 4-2 所示。其中 MCS（Tomita et al. 2010）[19]中的用户时间有误，经由与作者沟通之后，其用户时间更正为表 4-2 中的数据。

从表 4-1 和表 4-2 得知，我们的机器比 MaxCLQ 所用的机器快 1.462 倍，比 MCS 所使用的机器快 1.928 倍。因此他们论文中的数据需要分别除于相应的倍数才能用于本文中。

表 4-1 Li 和 Quan （2010b）中的用户时间

Graph	r200.5	r300.5	r400.5	r500.5
MaxCLQ	0.033	0.27	1.6939	6.28
Ratio with ours	1.65	1.42	1.3773	1.402
average ratio	1.462			

表 4-2 Tomita et al. 2010 中的用户时间

Graph	r200.5	r300.5	r400.5	r500.5
MCS	0.0415	0.359	2.21	8.47
Ratio with ours	2.075	1.8895	1.857	1.8906
average ratio	1.928			

4.2 对随机图的结果

我们首先在随机图上进行算法结果的比较。在比较过程中，我们不考虑顶点个数小于 200 的图，因为这些图对于这些算而言都太过简单，即都能在 1 秒之内被解决，因此没有比较的意义。而顶点数大于 500 的随机图我们也不考虑，因为测试这些图需要耗费较长的时间，往往一个例子所占用的时间都是按照小时计算的。在我们的统计中，对于任意特定顶点个数和边密度的随机图所得到的结果，都是用 3 个随机图的平均结果而得的。MaxCliqueDyn、MaxCliqueDyn 加初始化、MCSD 以及 MCSMD 在随机图上的结果如表 4-3 所示，其中 num.steps 列表示搜索树的大小，以递归次数来衡量。

从表 4-3 中可以看出，MaxCliqueDyn 的初始化操作在随机图上并没有明显的改进。将重着色的技术与动态排序策略结合起来后得到的 MCSD 无论在递归次数还是在用耗费的 CPU 时间上，都显著地小于 MaxCliqueDyn，更进一步的，在动态排序策略中改进排序准则后，所得到的 MCSMD 相比 MCSD 无论在递归次数还是在 CPU 时间上，都提升了不少。因此，在随机图上 MCSMD 和 MCSD 相比 MaxCliqueDyn 以及 MaxCliqueDyn 性能更好。

表 4-3 MaxCliqueDyn、MaxCliqueDyn 加初始化、MCSD 以及 MCSMD 在随机图上的比较

Graph		MaxCliqueDyn		MaxCliqueDyn + init_sort		MCSD		MCSMD	
n	p	num.steps	CPU time	num.steps	CPU time	num.steps	CPU time	num.steps	CPU time
200	0.6	36890	0.05	30591	0.05	18677	0.04	16340	0.04
	0.7	121188	0.23	116383	0.22	83013	0.20	61224	0.18
	0.8	1039561	2.4	1191612	2.53	570699	1.85	429400	1.60
	0.9	11292751	45.61	10275509	36.74	4349660	21.94	2997978	18.12
	0.95	7157365	49.64	7782767	46.16	3381974	22.80	2686360	22.13
300	0.6	293652	0.56	284600	0.55	187533	0.52	163429	0.49
	0.7	3932576	8.07	3956420	7.86	2115353	6.36	1717364	5.83
	0.8	97714062	272.62	94837439	245.81	43342876	171.41	32564730	145.86
500	0.5	829658	1.74	803278	1.69	572323	1.68	516816	1.60
	0.6	10637340	24.24	10684616	23.93	6557461	21.16	5642928	20.69
	0.7	289984697	791.43	310023669	806.34	168935496	641.01	138001245	605.51

规约之后 MCS 和 MaxSatClique 在随机图上的运行时间如表 4-4 所示。可以看出 MCS 几乎在所有的实例在都表现得不如 MCSD 和 MCSMD。这表明在随机图实例上，MCSD 与 MCSMD 更优于 MCS。

表 4-4 规约后 MCS 和 MaxSatClique 在随机图上的运行时间，及其与 MCSD、MCSMD 的比较

Graph		MCS	MaxSatClique	MCSD	MCSMD
n	p	CPU time	CPU time	CPU time	CPU time
200	0.8	2.33	1.55	1.85	1.60
	0.9	38.38	6.83	21.94	18.12
	0.95	30.60	1.64	22.80	22.13
300	0.6	0.52	1.02	0.52	0.49
	0.7	6.22	7.04	6.36	5.83
	0.8	204.36	108.28	171.41	145.86
500	0.5	1.45	4.27	1.68	1.60
	0.6	20.75	37.12	21.16	20.69
	0.7	798.24	784.54	641.01	605.51

然而，在于 MaxSatClique 的比较中，MCSD 与 MCSMD 都比 MaxSatClique 好的例子有 5 个，而 MaxSatClique 比 MCSD 和 MCSMD 都好的例子有 4 个。而且彼此都有显著好于对方的例子，如 n = 200, p = 0.9 以及 n= 300, p = 0.8 的例子中，MaxSatClique 显著优于其他两个算法，而在 n = 500, p = 0.6 以及 n = 500, n = 0.7 的例子中，MCSD 与 MCSMD 显著优于 MaxSatClique。

4.3 对 DIMACS 基准图的结果

通过用户时间对 Konc 和 Janezic（2007）[22]中数据的规约后，以及在我们所使用的计算机上用 MaxCliqueDyn 运行若干 DIMACS 基准图后，我们发现几乎在所有的 DIMACS 基准图实例上，本文描述的算法 MCSD 和 MCSMD 都显著地优于 MaxCliqueDyn，因此，我们在本节中，只将 MaxCliqueDyn 与用了初始化将其改进之后的算法进行对比，即用 Carraghan 和 Pardalos(1990)[1]以及 Tomita 等人(2010)[19]类似的排序准则初始化顶点，并且在算法开始时对顶点进行一次贪心着色过程，做这样的对比，是为了确定初始化是否对算法带来了改进。在其他算法的比较中，我们不考虑 MaxCliqueDyn。

表 4-5 MaxCliqueDyn 与其初始化改进之后的版本

Graph	w	MaxCliqueDyn		MCSD	
name		num.steps	CPU time	num.steps	CPU time
brock200_2	12	3619	0.01	2702	0
brock200_4	17	48397	0.09	54195	0.09
brock400_2	29	44602709	165.75	27699632	128.55
brock400_4	33	45795157	142.35	12118105	52.55
brock800_2	24	1297531152	5309.34	971297327	4412.61
brock800_4	26	564317467	2549.93	322769789	1952.56
MANN_a27	126	49439	1.86	48075	1.78
MANN_a45	345	4647163	1194.75	4569457	1201.49
p_hat300-1	8	2159	0	1635	0
p_hat300-2	25	7698	0.02	6998	0.03
p_hat300-3	36	633573	2.41	645367	2.6
p_hat700-1	11	28060	0.06	26316	0.07
p_hat700-2	44	1104290	6.82	1257534	9.65
p_hat700-3	62	292993278	3050.37	283878927	3189.58

MaxCliqueDyn 与用初始化改进之后的算法的对比如表 4-5 所示。我们可以看到用初始化进行改进之后，所需的分支次数有了明显的减少。对于 CPU 时间来说，用初始化改进之后的算法在 brock 基准用例上性能提升了不少，而在 p_hat 基准用例则性能有所下降，在本文中，我们认为这些性能下降在可接收范围之内。

对于 MCS 与 MCSD 以及 MCSMD，我们不仅比较它们在这些实例的 CPU 时间，同时也比较它们解决这些实例所需要的递归次数，MCS 的递归次数是从 Mikhail Batsyn 等人（2013）[39]中引入了。这三个算法在递归次数上的比较如表 4-6 所示。

如表 4-6 所示，我们知道无论是 MCSD 还是 MCSMD，在递归次数上都显著地小于 MCS，在许多实例上所需的分支数只有 MCS 的一半或更少。这说明将动态着色策略引进 MCS，并与 MCS 静态顺序的辅助顶点集相结合的技巧在减少分支数上是成功。

表 4-6 MCS 与 MCSD、MCSMD 在递归次数上的比较

Graph	MCS	MCSD	MCSMD
name	num.steps	num.steps	num.steps
brock400_1	88555048	73451443	58983645
brock400_2	34145195	16373021	13574692
Brock400_3	66379744	6120088	5040024
brock400_4	29696341	6497954	5865953
brock800_2	972110520	598045306	505023165
brock800_4	424176492	195822785	173377989
MANN_a27	33345	25759	23902
MANN_a45	221476	958654	1062570
p_hat300-3	565792	281087	205296
p_hat500-2	89836	53221	40875
p_hat500-3	17259920	8761290	6312690
p_hat700-1	29656	19031	17426
p_hat700-2	670369	463928	329870
p_hat700-3	98911559	96398867	69848881
p_hat1000-2	25209207	14749574	9888902
san400_0.7_1	64568	18762	12850
san400_0.7_2	23471	150158	78228
san400_0.7_3	253044	742771	260854
san400_0.9_1	20537	5780895	6536431
sanr200_0.7	115666	67297	53848
sanr200_0.9	8103466	2797389	1924603
sanr400_0.7	51507583	24278583	20394027

MaxSatClique、MCS、MCSD 以及 MCSMD 在 CPU 时间上的比较如表 4-7 所示。

对于本文实现的 MCSD 与 MCSMD，后者有 15 个测试用例是快于前者的，而后者只有 5 个用例是优于前者的，因此我们认为 MCSMD 是优于 MCSD 的，即我们在动态排序过程中所使用的排序准则是优于按照度数降序排列的。在之后的算法比较中，我们只考虑其他算法与 MCSMD 的比较。

表 4-7 规约后 MCS 和 MaxSatClique 在 DIMACS 基准图上的运行时间，及其与 MCSD、MCSMD 的比较，其中在 Tomita 等人（2010）中，MCS 没有 p_hat700-1 的试验结果

Graph	MaxSatClique	MCS	MCSD	MCSMD
name	CPU time	CPU time	CPU time	CPU time
brock400_1	253.65	359.44	297.8	276.19
brock400_2	122.23	154.05	92.63	82.97
Brock400_3	198.40	242.74	41.01	37.11
brock400_4	114.43	331.43	37.23	39.27
brock800_2	5259.92	4340.25	3165.95	4299.37
brock800_4	2653.90	2073.13	1413.31	1432.27
MANN_a27	0.45	0.41	0.43	0.43
MANN_a45	174.88	145.75	96.08	116.02
p_hat300-3	1.42	1.30	1.76	1.52
p_hat500-2	0.62	0.36	0.4	0.37
p_hat500-3	38.27	77.80	84.03	67.2
p_hat700-1	0.55	——	0.08	0.09
p_hat700-2	3.33	2.90	5.15	3.89
p_hat700-3	706.57	1240.66	1502.13	1086.77
p_hat1000-2	100.21	114.63	160.43	124.44
san400_0.7_1	0.14	0.28	0.1	0.1
san400_0.7_2	0.06	0.07	0.71	0.49
san400_0.7_3	0.51	0.73	2.58	1.29
san400_0.9_1	1.35	0.05	91.96	96.93
sanr200_0.7	0.26	0.18	0.19	0.17
sanr200_0.9	3.91	21.27	14.9	12.55
sanr400_0.7	96.44	93.88	81.74	74.09

考虑 MCS 与 MCSMD，在 brock、MMAN、sanr 实例上，MCSMD 整体上性能优于 MCS。而在 san 以及部分的 p_hat 实例上，MCSMD 表现得并不如 MCS，然而在这两组实例中，除了 san400_0.9_1（MCS 显著地优于 MCSMD)以及那些能够在 1 秒之内求解的实例外，这两个算法性能的差别在较小的范围之内。因此，从整体上而言，MCSMD 是优于 MCS 的。

对于 MaxSatClique 与 MCSMD，前者在 11 个用例上优于后者，而后者在 12 个用例上优于前者，而且两者都有明显优于对方的用例，例如 MaxSatClique 在 p_hat500-3、p_hat700-3 与 san400_0.9_1 等用例上显然占优，而 MCSMD 在 brock、MANN_a45 以及 sanr400_0.7 等用例上显然占优。因此，我们可以说这两个算法的性能不相上下。

第五章 结论

5.1 本文研究工作总结

本文首先描述了最大团问题的分支限界算法的基本框架，然后介绍了在最大团问题上，贪心着色过程是如何被用作限界策略与分支策略的。在这个基础之上，我们使用了重着色技术来改进贪心着色过程，将在朴素的贪心着色过程中原本是“主动的”顶点，在重着色过后变成被动的，从而避免了不必要的分支。

我们也介绍了在贪心着色的过程中，顶点被着色的顺序的重要性，从而介绍了一种不同于按度数降序排列的排序准则，并将其用于初始化顶点顺序。通过了这种初始化，使得算法在 brock 等 DIMACS 基准图上有了较好的性能改善。考虑到着色过程对顶点初始顺序的继承的不彻底性，我们使用了一种称为静态顺序的辅助顶点集来记录顶点的初始顺序，使得往后的每次着色过程都按照按顶点集的顺序进行。

同时，为了充分地利用计算机内存使用的局部性原则，我们在初始化时对邻接矩阵进行了重新排列，使得顶点编号的顺序与初始化后静态顺序的顶点辅助集中的顺序一致。

考虑到顶点排序后能够明显的提升贪心着色过程的性能，而顶点排序又是一个比较耗费 CPU 时间的操作，因此我们采用了动态排序的策略，在顶点个数较多的分支中对顶点进行重新排序，而在顶点度数较少的分支中，保持顶点原有的顺序。具体的，我们对搜索树中的每一层，都记录从本部到该层（包括该层）所执行的分支步数总和，以及目前为止搜索树总共执行的分支步数总和，通过这两者的比例而决定是否进行重新排序。这样的策略使得我们的算法能够自适应密集图与稀疏图，顶点较多的图以及顶点较少的图。

有了动态排序的策略并不意味着我们不再需要静态顺序的辅助顶点集，相反，我们应该考虑这两者的有机结合。一方面，动态排序使得我们的顶点顺序与当前子图保持一致，另一方面，静态顺序的辅助顶点集避免顶点的顺序被贪心着色过程所破坏。因此，在我们算法的过程中，每次动态排序直接作用到静态顺序的辅助顶点集中，使得辅助顶点集中的顺序尽可能与子图保持一致。通过上述所有策略的结合，得出了本文所描述的算法 MCSD。

考虑到初始化中所采用的排序准则所带来的作用，我们将该准则引入了动态排序中，从而得出本文的另一个算法 MCSMD。使得该算法比 MCSD 算法有了更进一步的改进。

我们通过计算实验的方式，将本文的算法 MCSMD、MCSD 与著名的 MCS、MaxCliqueDyn 以及 MaxSatClique 算法进行比较，这三个算法是我所知道的目前最优秀的算法。单从算法所需的分支次数来看，我们的新算法 MCSMD 与 MCSD 比 MCS、

MaxCliqueDyn 有了显著地改进，由于我们缺少 MaxSatClique 算法在分支上的数据，因此无法与之进行比较。

在 CPU 时间的比较上，我们的新算法在几乎所有的测试用例上都显著地优于 MaxCliqueDyn 算法，而本文其中的一个新算法 MCSMD 从整体表现性能上看，也明显地优于 MCS 算法。在于 MaxSatClique 算法的比较中，无论在随机图上还是 DIMACS 基准图上，MCSMD 与 MaxSatClique 都有各自占用的例子，而且无论在占优的数量以及占优的程度上，两者都差不多，因此我们可以说 MCSMD 与 MaxSatClique 这两个算法在性能表现上不相上下。

5.2 课题相关展望

Li 和 Quan 2010b[24]的 MaxSatClique 算法采用了最大可满足性问题的推理规则，该规则可以使得贪心着色过程所得的上界更加紧确，然而同时也放弃了贪心着色过程分支与限界无缝结合的优点，使得在搜索树每一层中都需要执行多次贪心着色过程（在本文中的算法以及 Tomita 等人（2007, 2010)[2,19]的算法中，一层只需执行一次贪心着色过程）。因此，接下来的工作中，我们可以考虑如何在使用最大可满足性问题的推理规则的同时，保持贪心着色过程的优点。

另外，目前最大团问题的分支限界算法所使用的数据结构都较为朴素。而在最大可满足性问题上，数据结构给算法的性能带来了显著地改进，我们在这方面加以借鉴。最后，最大团问题算法中所使用的分支方法、算法的重启策略、失败顶点的学习（与最大可满足性问题的失败子句学习相对应）也是算法改进的方向。

参考文献

〔1〕 Carraghan R, Pardalos PM, An exact algorithm for the maximum clique problem. Oper Res Lett 9(6), 1990:375－382.

〔2〕 Tomita E, Kameda T, An efficient branch-and-bound algorithm for finding a maximum clique with computational experiments. J Global Optim 37(1), 2007:95－111.

〔3〕 Bomze I, Budinich MPMP, PelilloM, The maximum clique problem. In: Handbook of combinatorial optimization. Kluwer Academic Publishers, Boston. 1999.

〔4〕 Butenko S, Wilhelm WE, Clique-detection models in computational biochemistry and genomics. Eur J Oper Res 173(1), 2006:1－17.

〔5〕 Brouwer A, Shearer J, Sloane N, Smith W, A new table of constant weight codes. IEEE Trans Inf Theory 36(6), 1990:1334－1380.

〔6〕 Sloane NJA, Unsolved problems in graph theory arising from the study of codes. Graph Theory Notes NY 18(11), 1989:11－20.

〔7〕 Du D, Pardalos PM, Handbook of combinatorial optimization, Supplement, vol A. Springer, New York, 1999.

〔8〕 张焕萍,王惠南,卢光明,钟元,张志强.基于互信息的差异共表达致病基因挖掘方法[J].东南大学学报(自然科学版),2009,第 1 期：151-155.

〔9〕 汤亚波,徐守时.一种卫星遥感图像目标位置快速精校正的新方法[J].遥感学报,2005,第 6 期：653-658.

〔10〕 张雁,焦方正,卢昕玮,黄永宣.采用染色划分改进的 RLS 算法及性能分析[J].软件学报,2011,第 10 期：2305-2316.

〔11〕 焦清局,刘太岗,郑小琪,连爱娥,黄继风.拟南芥花药基因的共表达分析[J].生物信息学,2010,第 4 期：350-355.

〔12〕 Etsuji Tomita, Tatsuya Akutsu, Tsutomu Matsunaga, Efficient Algorithms for Finding Maximum and Maximal Cliques: Effective Tools for Bioinformatics in “Biomedical Engineering, Trends in Electronics, Communications and Software,” Anthony N. Laskovski (Ed.), InTech, 2011：625-640

〔13〕 孙伟,张更新,边东明,苟刘士新,宋健海.求解资源受限项目调度问题的约束规划／数学规划混合算法[J].控制理论与应用,2011,第 8 期：1113-1120.

〔14〕 刘士新,宋健海.求解资源受限项目调度问题的约束规划／数学规划混合算法[J].控制理论与应用,2011,第 8 期：1113-1120.

〔15〕 张雁,焦方正,卢昕玮,黄永宣.采用染色划分改进的 RLS 算法及性能分析[J].软件学报,2011,第 10 期：2305-2316.

〔16〕 P. Crescenzi, C. Fiorini and R. Silvestri, A Note on the approximation of the maximum clique problem, Inform. Process. 1991.Lett.40,P1~5.

〔17〕 Bron C, Kerbosch J, Algorithm 457: finding all cliques of an undirected graph. Commun ACM 16(9), 1973:575 - 577.

〔18〕 Jenelius E, Petersen T, Mattsson L, Importance and exposure in road network vulnerability analysis. Transport Res A Policy Pract 40(7), 2006:537 - 560.

〔19〕 Tomita E, Sutani Y, Higashi T, Takahashi S, Wakatsuki M, A simple and faster branch-and-bound algorithm for finding a maximum clique. In: Proceedings of the 4th international conference on algorithms and computation, WALCOM'10. Springer-Verlag, Berlin, Heidelberg, 2010:191 - 203.

〔20〕 Fahle T, Simple and fast: improving a branch-and-bound algorithm for maximum clique. In: Proceedings of the 10th annual European symposium on algorithms, ESA '02. Springer-Verlag, London, 2002:485 - 498.

〔21〕 Tomita E, Seki T, An efficient branch-and-bound algorithm for finding a maximum clique. In: Proceedings of the 4th international conference on discrete mathematics and theoretical computer science, DMTCS' 03. Springer-Verlag, Berlin, Heidelberg, 2003: 278 - 289.

〔22〕 Konc, J., & Janezic, D, An improved branch and bound algorithm for the maximum clique problem. proteins, 4, 5. 2007.

〔23〕 Li CM, Quan Z, Combining graph structure exploitation and propositional reasoning for the maximum clique problem. In: Proceedings of the 2010 22nd IEEE international conference on tools with artificial intelligence, Vol 01, ICTAI' 10. IEEE, Arras, 2010a: 344 - 351.

〔24〕 Li CM, Quan Z, An efficient branch-and-bound algorithm based on maxsat for the maximum clique problem. In: Proceedings of the twenty-fourth AAAI conference on artificial intelligence, AAAI-10. AAAI Press, Atlanta, 2010b:128 - 133.

〔25〕 Harary, Frank, and Ian C. Ross. "A procedure for clique detection using the group matrix." Sociometry 20, no. 3, 1957: 205-215.

〔26〕 Pardalos, Panos M., and Gregory P. Rodgers. "Computational aspects of a branch and bound algorithm for quadratic zero-one programming." Computing 45.2, 1990: 131-144.

〔27〕 San Segundo, Pablo, et al. "An improved bit parallel exact maximum clique algorithm." Optimization Letters, 2013: 1-13.

〔28〕 San Segundo, Pablo, Diego Rodríguez-Losada, and Agustín Jiménez. "An exact bit-parallel algorithm for the maximum clique problem." Computers & Operations Research 38.2, 2011: 571-581.

〔29〕 Rossi, Ryan A., et al. "A Fast Parallel Maximum Clique Algorithm for Large Sparse Graphs and Temporal Strong Components." arXiv preprint arXiv:1302.6256 2013.

〔30〕 Michel Gendreau, Patrick Soriano and Louis Salvail, Solving the maximum clique problem using a tabu search approach, Annals of Operations Research 41, 1993:385-403.

〔31〕 Gary A. Kochenberger, Jin-Kao Hao, Zhipeng Lü, HaiboWang, Fred Glover, Solving large scale Max Cut problems via tabu search, J Heuristics, 2011

〔32〕 Franco Mascia, Analysis of reactive search optimisation techniques for the maximum clique problem and applications, 4OR-Q J Oper Res, 2012:217–218

〔33〕 Bharath Pattabiraman, Md. Mostofa Ali Patwary, Assefaw H. Gebremedhin, Wei-keng Liao, and Alok Choudhary, Fast Algorithms for the Maximum Clique Problem on Massive Sparse Graphs, Optimization Methods and Software, Vol. 00, No. 00, 2012:1–14

〔34〕 Qinghua Wu, Jin-Kao Hao, Fred Glover, Multi-neighborhood tabu search for the maximum weight clique problem, Ann Oper Res2012:611–634

〔35〕 Una Benlic, Jin-KaoHao, Breakout Local Search for maximum clique problems, Computers & Operations Research 40, 2013:192–206

〔36〕 Pardalos, Panos M., and Jue Xue. "The maximum clique problem." Journal of global Optimization 4.3, 1994:301-328.

〔37〕 Balas E, Yu CS, Finding a maximum clique in an arbitrary graph. SIAM J Comput 15(4), 1986:1054–1068.

〔38〕 Mycielski J, Sur le coloriage des graphes. Colloq Math 3, 1955:161–162.

〔39〕 Mikhail Batsyn, Boris Goldengorin, Evgeny Maslov. Panos M. Pardalos, Improvements to MCS algorithm for the maximum clique problem. J Comb Optim DOI, 2013.

致　谢

感谢郑运平老师给予我的悉心指导。
感谢蔡少伟师兄给我提供许多丰富的参考资料与建议。
感谢 Mikhail Batsyn 在算法实现细节上讨论。
感谢 Tomita 在算法比较上给予我必要的信息。