TEAM PROJECT REPORT

曾奥涵 2018013383
刘润达 2018013412

2019 年 7 月 2 日

目录

1 介绍

给定图 G，图中的一个团是 G 的一个完全子图，其中任意两个节点之间都有边连接。在图中找到含有最大点数的团是一个经典的 NP-hard 问题，目前并未发现任何高效找到最大团的方法。现如今求最大团的算法主要分为两类，一类是确定性算法，一类是启发式算法。确定性算法能确保给出图的一个最大团，一般需要进行大量的搜索，当点数增加时所需的计算量将爆炸式增长。而启发式算法往往效率较高，能够胜任较大的数据规模，但只能给出一个相对较优的解。

对于确定性算法，本项目分别实现了 Chu-Min Li 等人与 2010 年提出的 MaxCLQ 算法 [1]，Pablo San Segundo 等人在 2015 年提出的 BBMCX 算法 [2]。在 BBMCX 算法上，我们进一步参考 Pablo San Segundo 等人在 2012 年提出的 BBMC 算法[3]，将其中按位并行的思想融入到我们的实现中，利用 `std::bitset` 对数据的存储方式进行了优化。进一步地，我们参考了 Janez Konc 等人于 2007 年的论文[4]，在染色过程中启发式地对节点进行重新排序。最终，我们的实现效率达到了接近 STOA 的水平。

对于启发式算法，本项目实现了 Wayne John Pullan 等人与 2006 年提出的 DLS 算法 [5]，实现结果基本达到论文中的水平。

2 系统结构

2.1 Graph

```
class Graph {
  public:
    int n; // number of the vertices
```

[1] Li, CM, Quan, Z. Combining graph structure exploitation and propositional reasoning for the Maximum Clique Problem. In: Proceedings of the 2010 22th IEEE international conference on tools with artificial intelligence, IEEE. 2010. p. 344–351.

[2] San Segundo, Pablo Nikolaev, Alexey Batsyn, Mikhail. (2015). Infra-chromatic bound for exact maximum clique search. Computers & Operations Research. 64. 10.1016/j.cor.2015.06.009.

[3] San Segundo P, Rodriguez-Losada D, Jimenez A. An exact bit-parallel algorithm for the maximum clique problem. Comput Oper Res 2011;38(2):571–81.

[4] Konc, J., Janei, D.; An improved branch and bound algorithm for the maximum clique problem. MATCH Commun. Math. Comput. Chem. 58: 569-590, 2007

[5] Pullan, Wayne Hoos, Holger. (2011). Dynamic Local Search for the Maximum Clique Problem. Journal of Artificial Intelligence Research. 25. 10.1613/jair.1815.

```
    std::vector<std::vector<int> > G; // adjacency matrix
    Graph();
    bool loadGraph(const char *filename); // load graph from file
    const ints& operator [] (const int &idx) const;
    ints& operator [] (const int &idx);
};
```

最基本的用于存放图的类，由于数据中多为稠密图并且需要频繁判断两节点是否相邻，故采用邻接矩阵的方式存储图。

2.2 Strategy

```
class Strategy {
  public:
    virtual std::vector<int> getMaxClique(const Graph &G) = 0;
    ↪ // get max clique
};
```

整个算法的核心部分，采用 Strategy Pattern。功能为给定一个图，求该图的最大团。

本程序共实现了 4 种 Strategy：3 种确定性算法 `MaxCLQ`, `BBMCX`, `BBMCX_BITSET` 以及 1 种启发式算法 `DLS`。

2.3 Validator

```
class Validator {
  public:
    bool check(const Graph &G, std::vector<int> maxClique); //
    ↪ check if it's a clique
};
```

2.4 Estimator

```
class Estimator {
  public:
```

```
    void estimate(Strategy *stg, const Graph &G); // estimate
     ↪  a specific strategy
};
```

用来评估一个算法在某个图上的运行效率的类，该类会衡量算法寻找最大团的运行时间。

3 算法介绍

3.1 基于色数剪枝的分支定界法

大部分求取最大团的确定性算法使用分支定界法（Branch and Bound, BnB）的结构，按照一定的顺序枚举图中的所有极大团（Maximal clique），即不是其他的团的真子集的团。在递归搜索的过程中，维护一个当前团 S 以及一个与 S 中所有点相邻的集合 V，每次搜索都考虑将 V 中的一个点 v 加入 S，同时 V 被更新为 $V \cap N_G(v)$。

对于任意一个图 G 而言，若图的色数为 $\chi(G)$，图的最大团大小为 $\omega(G)$，则有：

$$\chi(G) \geq \omega(G)$$

因此，若当前的团大小为 $|S|$，节点集合 V 的导出子图的色数为 $\chi(G_V)$，那么当前能找到的最大团的上界是：

$$|S| + \chi(G_V)$$

如果这个上界没有超过当前找到的的最大团大小 $|S_m|$，那么无论如何搜索，当前分枝能找到的极大团的大小都不会大于 $|S_m|$，所以可以剪枝。

3.2 MaxCLQ 算法

MaxCLQ 算法是由 Chu-Min Li 和 Zhe Quan 在 2010 年提出的一种确定性最大团算法。在用色数定界的基础上，利用 Partial MaxSat 问题的求解，得到了一个低于色数的上界。

2015 年，Alexandre Prusch Zuge 等人的论文指出[6]，MaxCLQ 算法也可以不使用 SAT 问题描述而纯粹用图论语言描述。在对图进行染色后，最大团中每个颜色的点最多出现一个。如果我们能找到一个颜色的集合，使得不存在同时包含这些颜色的点的团，那么最大团中至少不含这个集合中颜色的点的一个，最大团的上界可以在色数的基础上减一。如果能找出 k 个不相交的这种集合，最大团的上界就可在色数的基础上减小 k，这便是低于色数剪枝边界的来源。

为了寻找这种集合，MaxCLQ 算法借助了 SAT 问题中一种叫做 Unit Propagation 的思想，关注那些只含一个点的团从而推出矛盾。

3.3 BBMCX 算法

BBMCX 算法是 Pablo San Segundo 等人在 2015 年的一种确定性最大团算法，也是到目前为止效率最高的一种确定性算法。BBMCX 算法运用了与 MaxCLQ 算法非常类似的思想，同样是在搜索过程中利用低于色数的边界剪枝。但在寻找这个边界的过程中却更加高效。简单来说，BBMCX 算法只考虑包含 3 个颜色的集合，这种形式上的简洁性大大提升了寻找这些集合的效率。

3.3.1 启发式节点排序

在染色的过程中，如果能够按照节点度数从大到小的顺序依次染色，那么会得到更好地染色结果。然而，对于节点集合 V 的导出子图而言，重新计算节点度数的代价是 $O(|G_V|^2)$ 的。这就意味着我们必须启发式地重新计算度数代价，Janez Konc 等人分析得出，只有在搜索树的最浅几层重新计算节点才是划算的。于是我们提出这样一种策略，根据输入数据的规模，动态计算重排序的阈值 k，在搜素树中 $< k$ 的所有层重新计算节点顺序，$\geq k$ 的所有层按照原先的顺序排序。该优化加入后，搜索树的大小有了一定的减小，总的计算效率得到了一定的提升。

3.3.2 利用 `bitset` 维护颜色集合

在 BBMCX 的算法执行过程中，需要寻找包含三个颜色的集合。对于点 v，第一个颜色的成立条件是存在颜色 k_1，使得 $|C_{k_1} \cap N(v)| = 1$，记重合的

[6]Züge, Alexandre Prusch, and Renato Carmo. "Maximum clique via maxsat and back again." Matemática Contemporânea 44 (2016): 1-10.

元素为 w。第二个颜色的成立条件是存在颜色 k_2，$|C_{k_2} \cap N(v) \cap N(w)| = 0$。条件中涉及大量的集合运算，如果单纯通过遍历的形式来判断，复杂度较高。我们在染色的过程中，用 `std::bitset` 动态维护每个颜色对应的节点集合，同时预处理出每个节点相邻节点的 `bitset`，这样就能够高效的利用 `bitset` 按位并行计算的优势判断成立条件。

加入 `bitset` 优化后，程序的运行效率有了约 30% 的提升，具体可参见后续实验结果。

3.4 DLS 算法

DLS（Dynamic Local Search）算法是 Wayne John Pullan 和 Holger H Hoos 在 2006 年提出的解决最大团问题的启发式算法。

作为一种启发式算法，DLS 不一定能找出最优解，但是能在较好的时间效率下找出一个较优解，能处理确定性算法所不能处理的数据规模。

3.4.1 Local Search

局部搜索（Local Search）指的是这样一类启发式算法；从一个解出发，对其不断修改，希望得到更优的解。模拟退火（Simulated Annealing）、爬山算法（Hill Climbing）都属于局部搜索。对于最大团问题，局部搜索是最成功的启发式算法框架。最大团问题的一个解被描述成一个点集，局部搜索的过程就是不断对这个点集进行小的修改来获得更好的解。

3.4.2 DLS 算法的基本框架

算法的基本框架由两个过程组成，维护一个团 C，初始化为一个随机的点 x：

- 扩展（expand）：在图 G 中按照一定顺序将能够加到 C 里的点都加到 C 中，得到一个极大团（不是任何其他团的真子集），然后进入平移阶段。
- 平移（plateauSearch）：对于一个不能再加入其他点的团 C，如果存在一个不在 C 中的点 v，使得 v 和 C 中的 $|C|-1$ 个点都相邻，而只和 C 中的一个点 p 不相邻，那么把 C 中的点 p 替换成点 v，或许就可以找到其他点继续加到 C 中。平移阶段的目的就是通过若干次替换，使得又有点能够加到 C 中，此时再进入扩展阶段。

3.4.3 DLS 算法的反复迭代与 Penalty 机制

仅仅进行一次扩展与平移的交替过程，很可能无法得到一个很好的解。比较简单的想法，是更换换团 C 的初始点 x，重新再做一次迭代。但仅仅这样做，算法还是很容易被“困在”某几个结果并不好的局部最优解。为此，在迭代过程中要给每个点赋予一个动态变化的权重 Penalty，在扩展和平移时，如果有多个点可供选择，那么会选择权重最小的点。

权重的计算方式为：算法开始时全部初始化为 0，每一轮扩展-平移迭代后，得到的团中包含的每个点的 Penalty 加一。经过特定的轮数（Penalty Delay）后，所有 Penalty 大于零的点的权重都减一，作用是让算法“遗忘”久远的迭代带来的权重。

3.4.4 DLS 算法维护集合的方式

DLS 算法需要多次对集合进行增删操作。为了提高效率，该算法使用两个数组描述一个集合：第一个数组 a[] 存储所有的集合元素，第二个数组 b[] 则表示每个元素在 a 中的下标（不存在则为 -1）。维护这样的数据结构，可以将集合增加元素和删除元素的时间复杂度都降为 $O(1)$，在 `dls_set.h` 头文件中实现了这样的集合。

```
// partial code from dls_set.h
class dls_set{
public:
    int* lst,*ind; // list, index
    int N, sz;
    dls_set();
    dls_set(int n);
    void add(int x);
    void del(int x);
};
```

3.4.5 DLS 算法的其他实现细节

在平移阶段时，为了防止在同一个位置周围换来换去，限制每个点只能被换进来一次。且如果平移阶段开始时团里所有的点都被换出去了，就直接结束平移阶段。

为了控制时间，记录向团 C 中加入点、交换点的次数 cntSteps，当达到一个预先设定的 maxSteps 时就退出算法。

结束一轮扩展-平移的迭代，重新选择一个点开始时，选择上一轮迭代中最后被加入到团中的点作为起始点。

论文中还建议，为了加速，可以使用指针代替下标访问所有数组中的元素。在实现中，为了保持较好的可读性和降低调试难度，我们没有加入这个优化。

4 实验

4.1 实验结果

所有算法均在同一框架下运行，运行时间不包括数据的读入时间。运行实验的机器的 CPU 为 2.5 GHz Intel Core i7 (i7-4980HQ)，编译选项开启 `-Ofast` 优化。所有的实验时间均限定在 12 小时之内。对于 frp100-40 数据集，由于图中点数高达 4000，我们的所有确定性算法均不能在合理的时间之内给出结果，故我们取程序运行 12 小时的最优解作为实验结果。实验结果如下表所示：

从表 1 可以看出，相较于 MaxCLQ 算法，BBMCX 算法有着明显的优势，同时，加入了 `bitset` 优化后的 BBMCX 算法相较朴素的 BBMCX 算法有着 25% ∼ 400% 的提升。至于启发式算法 DLS，在数据规模较小的时候无论是在执行实现还是运行结果上都没有优势，而在 frp100-40 数据集上，DLS 算法表现优异，不仅用时较短，且结果也是 4 个算法中最优的。

Name	ω	**MaxCLQ**	**BBMCX**	**BBMCX_B**	**DLS**
brock200_1	21	5.848s	1.377s	**0.409s**	3.95s(21)
brock200_2	12	0.131s	0.027s	**0.008s**	3.27s(10)
brock200_3	15	0.365s	0.092s	**0.023s**	3.77s(14)
brock200_4	17	1.18s	0.336s	**0.076s**	5.12s(17)
brock400_1	27	5614.92s	586.18s	**345.48s**	15.82s(23)
brock400_2	29	2118.35s	242.39s	**128.36s**	24.95s(24)
brock400_3	31	3623.74s	386.20s	**218.69s**	30.37s(23)
brock400_4	33	2082.06s	207.90s	**153.17s**	14.99s(24)
brock800_1	23	$> 12h$	7342.49s	**6821.66s**	31.22s(20)
brock800_2	24	$> 12h$	9229.95s	**7210.39s**	29.19s(20)
brock800_3	25	$> 12h$	7183.32s	**5525.27s**	25.73s(19)
brock800_4	26	$> 12h$	8257.82s	**6219.36s**	24.88s(18)
frp100-40	100	$\sim 12h(80)$	$\sim 12h(82)$	$\sim 12h(83)$	**1h(84)**

表 1: 在不同的数据集上，各算法的用时对比