3-расскрашиваемость графа

Крохалев Арсений

26 декабря 2017 г.

1 Введение

Существует много известных NP-полных задач, включая такие важные теоретические проблемы в графе, как раскраски и независимые множества. Человечество до сих пор не знает, существовании полиномиальных алгоритмов для этих проблем, но это не устраняет необходимости их решения как можно эффективнее. Первый не тривиальный алгоритм был предложен Лаувером (Lawer, 1937), который работает за $O\left(1.4422^n\right)$. На сегодняшний день самый быстрый известный алгоритм работает за время $O\left(1.3289^n\right)$ и предложен Ричардом Бейгелем и Дэвидом Эппстейном (Richard Beigel and David Eppstein, 2000)[1].

В данной статье будет рассмотрен алгоритм Лаувера, а также доказательство времени работы $O\left(1.4422^n\right)$ и реализация на языке C++.

2 Вспомогательные утверждения

2.1 Обозначения

$\overline{a...b} = \{x \hookrightarrow x \in \mathbb{N} \wedge x \geq a \wedge x \leq b\}$

$v, u, v_1, u_1, \ldots$ — обозначают вершины, а $e, e_1, \ldots$ — ребра. В данной работе все ребра будут не ориентированы, поэтому не теряя общности можно считать, что $e = (v, u) = (u, v)$

Графами будут обозначаться заглавными буквами G, F. $G = (V, E)$, где V — множество вершин, G — множество ребер, также примем $|V| = n$. Также иногда удобно обозначать множество вершин как V_G, а ребер E_G.

Для множества $S \subseteq V$, обозначим $G\left[S\right]$ — подграф, индуцированный на подмножество вершин S, или другими словами:

$$G\left[S\right] = \left(V_{G[S]}, E_{G[S]}\right) \quad V_{G[S]} = S,\ E_{G[S]} = \{e = (v_1, v_2) \mid e \in E_G \wedge v_1 \in S \wedge v_2 \in S\}$$

Для графа G и $v \in V$ введем $N\left(v\right)$ — множество всех смежных с v вершин. Или более формально

$$N\left(v\right) = \{u \mid \exists\, e \in E \hookrightarrow e = (v, u)\}$$

Через $I\left(G\right)$ обозначим множество всех максимальных независимых подмножеств вершин (далее MIS) в G. Также через $I^{\leq k}\left(G\right)$ — MIS, размер которых не превышает

k, а $I_v^{\leq k}(G)$ — только те подмножества, которые вдобавок содержат вершину v. По аналогии зададим $I^{=k}(G)$, $I_v^{=k}(G)$, $I^{\geq k}(G)$ и $I_v^{\geq k}(G)$

Время выполнения алгоритмов в данной статье будут иметь вид $O(p(n)c^n)$, где p — это полином и c — константа. Если мы округлим v до большего значения, то можно игнорировать полиномиальный фактор и считать, что время пропорционально c^n.

2.2 Теория

2.2.1 Максимальные независимые множества

Теорема 1. *[2] Максимальное число k-MIS в графе равно*

$$\lfloor n/k \rfloor^{(\lfloor n/k \rfloor+1)k-n} (\lfloor n/k \rfloor + 1)^{n-\lfloor n/k \rfloor k} \tag{1}$$

Доказательство. Для смежных v и w в G обозначим $G_{v\to w}$ как граф, в котором v заменено на копию w с сохранением ребра между ними.

Мы хотим найти зависимость между количеством k-MIS в графе $G_{v\to w}$ и в графе G,

$$\left|I^{=k}(G_{v\to w})\right| = \left|I^{=k}(G)\right| + \dots$$

Ни одно из $k-$MIS в обоих графах не может содержать v и w одновременно, поскольку они смежны. Заметим, что $k-$MIS, содержащий w в одном из графов будет независимым множеством в другом поскольку все изменения касаются только вершины v, которая не входит в это множеств и будут являться максимальными поскольку v всё ещё не может быть добавлено. Исходя из этого, $k-$ MIS, содержащие w одинаковы в этих графах. Также k-MIS в $G_{v\to w}$, содержащие v точно такие же как содержащие w, с заменой v на w, итак

$$\left|I^{=k}(G_{v\to w})\right| = \left|I^{=k}(G)\right| + \left|I_w^{=k}(G)\right| - \left|I_v^{=k}(G)\right| + \dots$$

Все остальные $k-$MIS в G, а точнее не содержащие ни v, ни w, также являются $k-$MIS в $G_{v\to w}$ поскольку изменения коснулись только вершины v, а она в это множество не входит поэтому могли только появиться новые $k-$MIS в $G_{v\to w}$. Обозначим их количество за $f_{v\to w}^{=k}(G)$.

$$\left|I^{=k}(G_{v\to w})\right| = \left|I^{=k}(G)\right| + \left|I_w^{=k}(G)\right| - \left|I_v^{=k}(G)\right| + f_{v\to w}^{=k}(G) \tag{2}$$

Совершенно аналогично получим, что

$$\left|I^{=k}(G_{w\to v})\right| = \left|I^{=k}(G)\right| + \left|I_v^{=k}(G)\right| - \left|I_w^{=k}(G)\right| + f_{w\to v}^{=k}(G) \tag{3}$$

Теперь рассмотрим граф G из n вершин с максимальным числом k-MIS, у которого v и w — смежные. Тогда $\left|I^{=k}(G_{w\to v})\right|$ и $\left|I^{=k}(G_{v\to w})\right|$ не могут превышать $\left|I^{=k}(G)\right|$. Из равенств (2) и (3), получим $\left|I_v^{=k}(G)\right| = \left|I_w^{=k}(G)\right|$ и $f_{v\to w}^{=k}(G) = f_{w\to v}^{=k}(G) = 0$. Это обозначает, что граф G со смежыми v и w можно заменить на $G_{v\to w}$ без изменения величины $\left|I^{=k}(G)\right|$. Рассмотрим вершину $w \in V$ и $N(w) = \{v_1, v_2, \dots, v_m\}$. Последовательной заменой G на $G_{v_i\to w}$ для $i = \overline{1...m}$, мы получим вершины $\{w, v1, v2, \dots, v_n\}$ образующие отдельную компоненту в виде клики. Если проделать данную операцию для оставшихся вершин, то в итоге мы получим граф, состоящий из объединения

нескольких независимых клик. Положим количество вершин в каждой такой компоненте за $i_1, i_2, \ldots, i_l$ где $i_1 + i_2 + \cdots + i_l = n$.

$$\left|I^{=k}(G)\right| = \begin{cases} i_1 \cdot i_2 \cdot \ldots \cdot i_l & \text{если } l = k \\ 0 & \text{иначе} \end{cases} \tag{4}$$

Выражение (4) максимально, если ни один из i_j-ых не отличается более чем на единицу, а точнее достигает максимума если $k - (n \mod k) = (\lfloor n/k \rfloor + 1)\, k - n$ будут принимать значение $\lfloor n/k \rfloor$, а оставшиеся $(n \mod k) = n - (\lfloor n/k \rfloor k)$, что дает нам заявленный результат (1). ■

Алгоритм 1. *Нахождение всех k-MIS в графе*

```
MIS(S, I, k)
    if |S| = k then
        check(I ∪ S)
    else if |S| > k > 0 then
        if the largest degree in S is ≥ d then
            let v have the largest degree in S
            MIS(S \ N̄(v), I ∪ {v}, k − 1)
            MIS(S \ {v}, I, k)
        else
            let v have the smallest degree is S
            MIS(S \ N̄(v), I ∪ {v}, k − 1)
            for all w ∈ N(v) do
                MIS(S \ N̄(w), I ∪ {w}, k − 1)
```

Теорема 2. *Ни один граф не может содержать более*

$$d^{(d+1)k-n}(d-1)^{n-dk} \tag{5}$$

k-MIS, где $d \in \mathbf{N} \setminus \{0\}$ и Алгоритм 1 находит их все за a время в пределах полиномиального коэффициента нашей границы.

Рассмотрев $d = \lfloor n/k \rfloor$ мы получим (1). Исходя из Теоремы 1 оценка (5) является точной только в случае $n/(d+1) \leq k \leq n/d$.

Доказательство. Алгоритм 1 рассматривает вершину v в случаях, когда она входит в k-MIS и когда не входит. Если I — это k-MIS, содержащее v, тогда $I \setminus \{v\}$ — это $(k-1)$-MIS в $G\left[V \setminus \overline{N}(v)\right]$, а если I не содержит v, следовательно I — является k-MIS в $G[V \setminus \{v\}]$, содержащий смежную к v вершину. Нахождение k-MIS в $G\left[V \setminus \overline{N}(v)\right]$ или $G[V \setminus \{v\}]$ совершается путем рекурсивного вызова или же путем поочередного рассмотрения каждого из соседей v. Параметр I — это независимое множество, собранное на текущий момент, которое должно быть расширено до MIS путем добавления вершин из $G[S]$, $check(I)$ возвращает I в случае если I является MIS.

Количество листьев в дереве рекурсии, запущенной на графе с n вершинами, является верхней оценкой на число k-MIS, так и время работы алгоритма (игнорируя полиномиальные факторы). Положим $T(n,k)$ — наименьшая функция, удовлетворяющая

рекуррентному соотношению:

$$T(n,k) = \max \begin{cases} T(n-(d+1),k-1) + T(n-1,k) \\ T(n-1,k-1) \\ 2 \cdot T(n-2,k-1) \\ \vdots \\ d \cdot T(n-d,k-1) \end{cases} \tag{6}$$

где $T(n,0) = T(k,k) = 1$. Тогда $T(n,k)$ будет является верхней оценкой на число листьев в дереве рекурсии. Теперь (5) является верхней оценкой на $T(n,k)$: если $n = k$, то 5 равняется $\left(d^d / (d+1)^{d-1}\right)^k$, что больше единицы для $d \geq 1$ и если $k = 0, 5$ равняется $((d+1)/d)^n$, что также больше единицы. Тогда по индукции

$$T(n-(d+1),k-1) + T(n-1,k) \leq (1+d) \cdot d^{(d+1)k-n} (d+1)^{n-dk-1}$$

что равняется (5)

$$d \cdot T(n-d,k-1) \leq d \cdot d^{(d+1)k-n-1} (d+1)^{n-dk}$$

что также равно (5). Наконец для $a < d$

$$a \cdot T(n-a,k-1) \leq \frac{a \cdot (d+1)^{d-a}}{d^{d+1-a}} \cdot d^{(d+1)k-n} (d+1)^{n-dk}$$

Последний член в точности (5), а первый равняется

$$\frac{(d+1)^{d+1-a} - (d+1-a)(d+1)^{d_1}}{((d+1)-1)^{d+1-a}}$$

и числитель представляет из себя лишь первых два члена в биномиальной записи знаменателя и как следствие все выражение не превышает единицы. Поэтому (5) является верхней границей для $T(n,k)$. ■

Теорема 3. *Равенство* (5) *является верхней оценкой на на число* MIS *размера не более* k *для любого* $d \in \mathbb{N}$, $d \geq 3$ *и модифицированный алгоритм 1 (вызывает* $check(I)$ *когда* $S = \emptyset$ *в первых двух строчках и выполняет часть с* $else$ *при* $k > 0$*) находит их все за полиномиальный фактор от* (5).

Доказательство. Единственное изменение в рекурсивном соотношении заключается в том, что вместо условия $T(k,k) = 1$ будет услоие $T(0,k) = 1$. Для $n = 0$ (5) эквивалентно $\left(d^{d+1} / (d+1)^d\right)^k$, что больше единицы при $d \geq 3$. Поэтому мы получим ту же оценку на время работы. ■

2.2.2 Раскраски

Раскраска графа состоит в разбиении вершин на k независимых подмножеств, в таком случае можно предположить, что одно из множеств по крайней мере размера n/k. А именно сформулируем следующую лемму:

Лемма 1. *Если $G[M]$ — это максимальный k-раскрашиваемый подграф в G и $0 < k_1 < k$, тогда в G существует максимальный k_1-раскрашиваемый подграф $G[M']$ ($M' \subseteq M$), размером хотя бы $|M| \cdot k_1/k$, такой что $G[M \backslash M']$ — это максимальный $(k - k_1)$-раскрашиваемый подграф в $G[V \backslash M']$.*

Доказательство. Среди всех k-раскрасок графа $G[M]$ выберем наибольшее множество вершин, имеющих только k_1 цветов. Тогда они образуют k_1-раскрашиваемый подграф $G[M']$ в G размера по крайней мере хотя бы $|M| \cdot k_1/k$. Получим максимальный k_1-раскрашиваемый подграф в G. Действительно, предположим, что существует такая вершина $v \in V \backslash M'$, что $M[M' \cup v]$ является k_1-раскрашиваемым. Рассмотрим 2 случая

1. $v \in M$, но тогда множество $\{M' \cup v\}$ содержится в M и включает M', что противоречит максимальности M'

2. $v \in V \backslash M$, тогда докажем, что $G[M \cup v]$ будет k-раскрашиваемым. Во-первых рассмотрим ту самую раскраску $G[M]$, благодаря которой мы выбрали максимальный k_1-раскрашиваемый подграф, обозначим эти цвета номерами $\overline{1...k_1}$, соответственно оставшиеся вершины из $M \backslash M'$ не могут быть покрашены ни в один из этих цветов, для определенности пусть им доступны цвета $\overline{k_1...k}$. Во-вторых по нашему предположению, $M[M' \cup v]$ — k_1-раскрашиваем, следовательно можно подобрать новую раскраску для $G[M' \cup v]$ так, чтобы они имели цвета только из $\overline{1...k_1}$. Наконец $G[M \cup v]$ является k-раскрашиваемым, поскольку множества $M \backslash M'$ и $M \cup v$ не имеют общих цветов. А это в свою очередь противоречит максимальности $G[M]$.

Осталось показать максимальность $G[M \backslash M']$ в $G[V \backslash M']$. Аналогично предположим, что существует такая вершина $v \in V \backslash M'$, что $G[M \backslash M' \cup v]$ является $(k - k_1)$-раскрашиваемым, а $G[M']$ — k_1-раскрашиваем, следовательно $G[M \cup v]$ будет k-раскрашиваемым. Получаем противоречие с максимальностью $G[M]$. ■

Метод 1. *Для проверки графа на k-раскрашиваемость, мы будем перебирать все MIS I размера хотя бы $\lceil n/k \rceil$ и проверять, что граф $G[V \backslash I]$ является $(k-1)$-раскрашиваемым.*

Доказательство. Коррекстность данного метода следует непосретственно из Леммы 1, если рассмотреть $M = V$ и $k_1 = 1$ ■

3 Описание алгоритма

Собственно алгоритм заключается в непосредственном применении Метода 1, а именно перебор всех независимых множеств размера не более $\lceil n/3 \rceil$, а затем проверки, что оставшийся граф можно покрасить в 2 цвета, что можно сделать за полиномиальное время.

Плюс ко всему если в оценку (5) подставить $d = 3$ и $k = \lceil n/3 \rceil$, то получим как раз заявленное $O\left(3^{n/3}\right)$

4 Реализация

Для начала опишем некоторый класс **Graph**. Непосредственно сам алгоритм будет скрываться во внутреннем классе **Colorability3**, который будет описан ниже.

```
class Graph {
public:
```

Добавим возможность создание графа из файла, а также при помощи генерации случайного графа с вероятностью ребра **edge_chance**. Плюс к этому добавим возможность сохранения графа в файл.

```
    Graph(std::string filename);
    Graph(int vertex_count, double edge_chance);
    ~Graph();

    void store(std::string filename);
```

Основной метод проверки, который просто создать класс **Colorability3** и вызовет его метод проверки.

```
    bool is3Colorable();
```

Ну и последнее это список смежности и основной класс **Colorability3**.

```
private:
    std::vector<std::vector<int>> adjacency_list_;

    class Colorability3;
};
```

Теперь представим прототип класса проверки на раскрашиваемость.

```
class Graph::Colorability3 {
```

Для конструирования ответа потребуется только список смежности. А для его вывода будем использовать метод **check**.

```
public:
    Colorability3(const std::vector<std::vector<int>>*
                  adjacency_list);
    ~Colorability3();

    bool check();
```

Далее нам потребуется несколько параметров, будем хранить их непосредственно в классе, чтобы не сохранять на стэке вызовов.

vertex_count_ — общее число вершин,
vertexes_degree_ — текущие степени вершин в оставшемся графе,
adjacency_list_ — список смежности графа,
used_ — использованные вершины (или дополнение к множеству S),
used_count_ — число использованных вершин,
independent_ — вершины, образующие независимое множество.

```
private:
    int vertex_count_;
    std::vector<int> vertexes_degree_;

    const std::vector<std::vector<int>>* adjacency_list_;
    std::vector<int> used_;
    int used_count_;
    std::vector<bool> independent_;
```

Пара функций для нахождения вершины с максимальной и минимальной степенью в оставшемся графе

```
    int argmaxVertexDegree();
    int argminVertexDegree();
```

Метод для проверки графа, без независимого множества на 2-раскрашиваемость

```
    bool isRest2Colorable();
```

Далее идет ряд вспомогательных функций, смысл который в добавлении или изъятии вершин из использования, а также обновление степеней вершин в оставшемся графе. Токность алгоритма заключается в том, что **used_** сохраняет количество раз, которое вершина была использована, соответсвенно, чтобы вернуть её возвращаясь из рекурсии, нужно изъять ее ровно такое же количество раз.

```
    void use(int v);
    void unuse(int v);
    void useNeighbors(int v);
    void unuseNeighbors(int v);
    void useWithNeighbors(int v);
    void unuseWithNeighbors(int v);

    void eraseVertexesDegree(int v);
    void insertVertexesDegree(int v);
```

Наконец, последняя и основная функция перебора необходимых для алгоритма независимых множеств.

```
    bool MIS(int k);
};
```

Наибольший интерес представляет именно последний метод MIS, представляющий из себя Алгоритм 1 модифицированный с помощью Теоремы (3).

```
bool Graph::Colorability3::MIS(int k) {
```

В случае, если свободные вершины закончились или мы собрали независимое множество нужной длины, то необходимо проверить, что оставшийся граф можно расскрасить в 2 цвета.

```
    if (used_count_ == vertex_count_ || k == 0) {
        if (isRest2Colorable()) {
            return true;
```

```
        }
    } else if (k > 0) {
```

Иначе, находим вершину с наибольшей степенью.

Если степень больше трех, то пытаемся сначала добвить её, исключив смежные вершины, а затем просто убрать данную вершину и исследовать граф без нее.

```
int v = argmaxVertexDegree();

if (vertexes_degree_[v] >= 3) {
    useWithNeighbors(v);

    independent_[v] = true;
    if (MIS(k - 1)) {
        return true;
    }
    independent_[v] = false;

    unuseNeighbors(v);

    if (MIS(k)) {
        return true;
    }

    unuse(v);
} else {
```

А если меньше трех, то пробуем по-очереди добавить в независимиое множество сначала её, а потом каждую из её соседей.

```
    v = argminVertexDegree();

    useWithNeighbors(v);

    independent_[v] = true;
    if (MIS(k - 1)) {
        return true;
    }
    independent_[v] = false;

    unuseWithNeighbors(v);

    for (auto to_it = (*adjacency_list_)[v].begin();
         to_it != (*adjacency_list_)[v].end();
         to_it++)
    {
        int to = *to_it;
```

```
                if (!used_[to]) {
                    useWithNeighbors(to);

                    independent_[to] = true;
                    if (MIS(k - 1)) {
                        return true;
                    }
                    independent_[to] = false;

                    unuseWithNeighbors(to);
                }
            }
        }
    }
    return false;
}
```

Полную версию реализации, а также пример запуска можно найти по ссылке.

5 Тестирование и аналитика

Для тестирования был написан класс, решающий данную задачу простым перебором. Исходный код находится в том же репозитории, что и само решение. Также было проделано измерение времени работы на графах размера не более 70, результаты выполнения представлены в виде графика ниже.

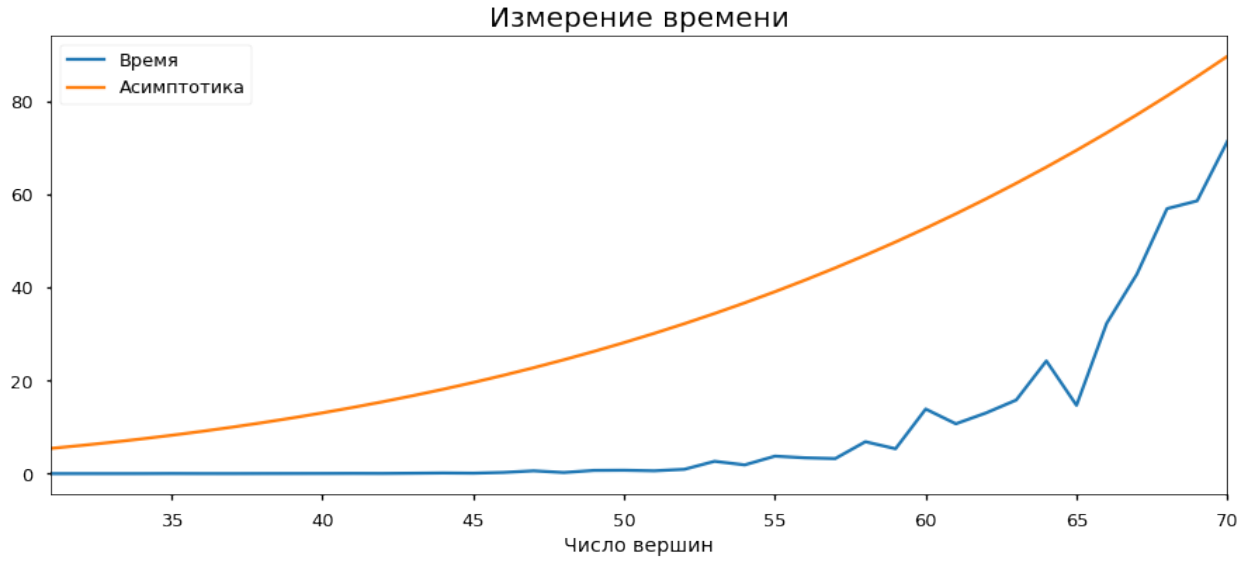

Рис. 1: Время работы

В роли асимтотики выступает функция $0.00004 \cdot n^2 3^{n/3}$

Содержание

Список литературы

[1] David Eppstein Richard Beigel. *3-Coloring in Time $O(1.3289^n)$*. 2000.

[2] Jesper Makholm Byskov. *Enumerating maximal independent sets with applications to graph colouring*. 2004.