Быстрый алгоритм для решения задачи о раскраске графа с использованием битовых операций

Л.Ф.Комоско
Национальный исследовательский университет «Высшая школа экономики», Лаборатория алгоритмов и технологий анализа сетевых структур, ул. Родионова, 136, Нижний Новгород 603093, Россия
lucent.92@mail.ru

М.В.Бацын
Национальный исследовательский университет «Высшая школа экономики», Лаборатория алгоритмов и технологий анализа сетевых структур, ул. Родионова, 136, Нижний Новгород 603093, Россия
mbatsyn@hse.ru

Аннотация

В статье рассматривается задача о раскраске графа. Предложен эвристический алгоритм, позволяющий получить раскраску графа (вектор из n натуральных чисел) с помощью математических операций над битовым представлением матрицы смежности графа. Скорость и точность данного алгоритма сравнивается с этими же характеристиками известного алгоритма GIS (Greedy Independent Sets-Colour). Результаты сравнения двух алгоритмов, выполнены на графах библиотеки DIMACS. Они показывают, что предложенный эвристический алгоритм выполняет раскраску графа быстрее по сравнению со стандартным подходом к его реализации[1].

1. Введение

Задача о раскраске графа является известной задачей комбинаторной оптимизации. Это одна из двадцати одной NP-трудной задачи Ричарда Карпа (Karp, 1972). Данный класс задач известен тем, что ни одному известному алгоритму не удается найти точного решения за полиномиальное время.

Задача о раскраске графа состоит в определении минимального количества цветов, которые можно назначить вершинам графа так, что никакие две смежные из них не будут окрашены в один цвет. Для определения эффективности алгоритмов раскраски, с 1996 активно используется набор графов библиотеки DIMACS[1].

Существует большое количество эвристических алгоритмов раскраски графа[4] Большую практическую ценность на данный момент представляют следующие: жадный алгоритм раскраски (Greedy-Colour), раскраска с обменом цветами (Colour-with-Interchange), последовательная раскраска графа без упорядочивания его вершин (Random-Sequential-Colour), последовательная раскраска графа с упорядочиванием его вершин по убыванию их степеней (Largest-First-Colour), последовательная раскраска графа начиная с вершин максимальных степеней (Smallest-Last-Colour), раскраска с обменом цветами без упорядочивания вершин графа (Random-Sequential-Interchange-Colour), раскраска с обменом цветами с упорядочиванием вершин графа по убыванию их степеней (Largest-First-Interchange-Colour), раскраска с обменом цветами начиная с вершин максимальных степеней графа (Smallest-Last-Interchange-Colour), жадная раскраска графа, где его вершины упорядочиваются таким образом, что у каждой есть по крайней мере одна соседняя, покрашенная в предшествующий цвет (Connected-Sequential-Colour) , последовательная раскраска с динамическим упорядочиванием вершин графа (Saturation-Colour) и жадная раскраска независимых подмножеств (Greedy Independent Sets-Colour). Необходимо отметить, что все эти алгоритмы представляют собой

[1] Работа выполнена при поддержке Лаборатории алгоритмов и технологий анализа сетевых структур НИУ ВШЭ, грант правительства РФ дог. 11.G34.31.0057

последовательность простых алгоритмических действий, условных переходов и циклов без использования каких-либо битовых или математических операций.

В данной статье представлен алгоритм раскраски графа, основанный на битовых операциях над матрицей смежности. Рассмотрено быстродействие алгоритма и точность полученных эмпирически решений. Выявлены преимущества данной реализации алгоритма по сравнению с подходом, когда используются стандартные конструкции программирования.

2. Основные термины и определения

Графом, в общем случае, называется совокупность двух множеств: G = (V,E), где V - множество вершин, E⊂V×V - множество связей между ними (множество ребер).

Две вершины u,v ∈ V называются смежными, если они связаны ребром:

(u, v) ∈ E. Два ребра e, f ∈ E называются смежными, если они имеют общую вершину:

e ∩ f ≠ Ø.

Степенью deg(v) вершины v в графе G называют число вершин, смежных с вершиной v: deg(v) = |{u ∈ V| (v,u) ∈ E}|.

Раскраской вершин графа G = (V,E) является функция c : V → ℕ, которая каждой вершине графа ставит в соответствие натуральное число(цвет) так, что любым двум инцидентным вершинам u, v ∈ V назначаются разные цвета: {u, v} ∈ E ⇒ c(u) ≠ c(v). Функция c называется функцией раскраски.

В данной статье функция раскраски имеет следующий вид: c: $\{0,1\}^{n\times n} \longrightarrow \mathbb{N}^n$. То есть, по матрице смежности графа мы получаем вектор его цветов.

Граф G, для которого существует раскраска из k цветов, называется k-раскрашиваемым. В этом случае функция раскраски разбивает граф G на независимые подмножества $V_1, V_2, \ldots, V_k$, для которых справедливо: $V_i \cap V_j = Ø$ и $V_1 \cup V_2 \cup \ldots \cup V_k = V$. Внутри независимого множества никакие две вершины не связаны ребром.

Наименьшее число k, для которого существует k-раскраска графа G, называется хроматическим числом графа G и обозначается χ(G). Любая раскраска графа G, для которой необходимо только k цветов, и k = χ(G), называется оптимальной[8].

3. Математическая модель задачи о раскраске графа

Имеется граф G = (V,E), где V - множество вершин, E - множество ребер между ними. Необходимо назначить каждой вершине $v \in V$ цвет так, чтобы общее количество цветов было минимальным.

Пусть:

- k – количество цветов, в которые можно раскрасить граф, причем $k \le |V|$. Разумнее сначала найти любую раскраску с помощью быстрой эвристики с некоторым числом цветов *col* и положить $k = col$, иначе необходимо положить $k = |V|$.
- x_{ih} ($i \in V$, $h = 1, \ldots, k$), где $x_{ih} = 1$, если вершине i назначен цвет h
- y_h ($h = 1, \ldots, k$), где $y_h = 1$, если цвет h использован в раскраске

$$\sum_{h=1}^{k} y_h \rightarrow \min \quad (1)$$

$$\sum_{h=1}^{k} x_{ih} = 1, \ i \in V \quad (2)$$

$$x_{ih} + x_{jh} \le y_h, \ (i, j) \in E, h = 1, \ldots, k \quad (3)$$

$$x_{ih} \in \{0, 1\}, \ i \in V, h = 1, \ldots, k \quad (4)$$

$$y_h \in \{0, 1\}, \ h = 1, \ldots, k \quad (5)$$

Функция (1) минимизирует количество цветов, используемых в раскраске. Ограничения (2) требуют, чтобы каждой вершине был назначен ровно один цвет. Ограничения (3) требуют, чтобы смежным вершинам были назначены разные цвета. Наконец, ограничения (4) и (5) задают возможные значения переменных – 0 или 1[5].

4. Алгоритм

4.1. Общая идея алгоритма

Предложенный нами алгоритм работает на основе битовых операций над матрицей смежности. A — квадратная матрица размера $n \times n$, в которой значение элемента a_{ij} равно числу рёбер из i-й вершины графа в j-ю вершину, причем $a_{ij} \in \{0, 1\}$(между двумя вершинами может быть только одно ребро). Будем считать, что любая вершина смежна сама с собой, то есть на главной диагонали матрицы стоят 1.

Над матрицей A проводятся следующие операции:

Начиная с первой строки матрицы A (i=1), производится поиск по строке первой несмежной неокрашенной вершины. Несмежной вершиной j окажется та, для которой в столбце j i-ой строки будет стоять 0. Если считать i-ю строку двоичной записью некоторого числа A_i, то математически можно инвертировать каждый его разряд ($\overline{A_i}$) и

тогда $j=[\log_2(\overline{A_i})]$, где $[x]$ – целая часть x.

Далее, согласно алгоритму, все соседи вершины j добавляются к соседям вершины i (происходит стягивание вершин). В результате получается обновленная i-ая строка матрицы A, в

которой 0 стоят только в столбцах с номерами вершин, которые несмежны ни с *i*-ой, ни с *j*-ой вершинами. Математически это можно заменить операцией дизъюнкции булевых векторов, представляющих собой строки *i* и *j*.

Номера строк *i* и *j* матрицы A формируют первую цветовую группу.

Затем производим ту же операцию поиска несмежной вершины или, можно сказать, первого 0 в обновленной строке с номером i и таким образом, возможно, добавляем новые вершины в первую цветовую группу.

Набор в группу заканчивается в двух случаях: во-первых, когда в i-ой строке накапливаются все 1 – это означает, что все несмежные друг с другом и с вершиной i вершины уже попали в группу с одним цветом; во-вторых, когда все вершины графа G уже покрашены, то есть любая вершина находится в какой-либо цветовой группе.

Если набор в группу закончился по первой причине, то алгоритм продолжает работу со следующей неокрашенной строки по порядку после i-ой. Производится набор строк в новую цветовую группу согласно вышеописанной последовательности действий.

Алгоритм полностью останавливается в случае, если всем вершинам графа G назначены цвета, другими словами, все строки матрицы A распределены по цветовым группам.

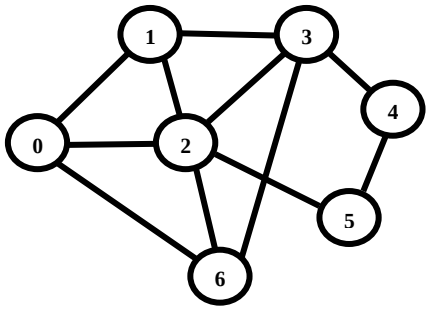
4.2. Пример работы алгоритма

Пусть дан неориентированный граф G(V,E), для которого:

V = {0,1,2,3,4,5,6}

E = {(0,1), (0,2), (0,6), (1,2), (1,3), (2,3), (2,5), (2,6), (3,4), (3,6), (4,5)}

n = 7



Построим для графа G(V,E), матрицу смежности A:

	0	1	2	3	4	5	6	
	1	1	1	0	0	0	1	**0**
	1	1	1	1	0	0	0	**1**
	1	1	1	1	0	1	1	**2**
A :=	0	1	1	1	1	0	1	**3**
	0	0	0	1	1	1	0	**4**
	0	0	1	0	1	1	0	**5**
	1	0	1	1	0	0	1	**6**

Согласно алгоритму, манипуляции с матрицей смежности начинаем с первой строки: *i*=0. Ищем в строке с номером 0 первую несмежную вершину, то есть первый ноль(слева направо). Первый ноль находится в столбце с номером 3, значит номер несмежной с *i* вершиной равен 3, то есть *j*=3.

Теперь складываем строки 0 и 3:

1110001
V 0111101
1111101

Таким образом, новая стока *i*=0 выглядит следующим образом: 1111101. Кроме того, номера строк *i* и *j* помещаются в первую цветовую группу и эти вершины считаются покрашенными: Colour1 = {0,3}.

Обновленная матрица смежности:

	0	1	2	3	4	5	6	
	1	1	1	1	1	0	1	**0**
	1	1	1	1	0	0	0	**1**
	1	1	1	1	0	1	1	**2**
A :=	0	1	1	1	1	0	1	**3**
	0	0	0	1	1	1	0	**4**
	0	0	1	0	1	1	0	**5**
	1	0	1	1	0	0	1	**6**

В обновленной строке *i* еще есть 0, поэтому набор в первую группу цвета не заканчивается. Ищем новую несмежную с *i* вершину: ее номер – 5, то есть *j*=5. Номер строки *j* добавляется в первую цветовую группу:

Colour1 = {0,3,5}.Складываем строки с номерами *i* и *j*:

1111101
V 0010110
1111111

Обновленная строка *i* вся состоит из 1, поэтому набор в цветовую группу один – закончен.

Обновленная матрица смежности:

	0	1	2	3	4	5	6	
	1	1	1	1	1	1	1	**0**
	1	1	1	1	0	0	0	**1**
	1	1	1	1	0	1	1	**2**
A :=	0	1	1	1	1	0	1	**3**
	0	0	0	1	1	1	0	**4**
	0	0	1	0	1	1	0	**5**
	1	0	1	1	0	0	1	**6**

Так как набор в первую цветовую группу закончен, будем составлять новую – следующую цветовую группу.

Теперь *i* принимает значение 1, так как следующая неокрашенная после 0-ой стоки – строка с номером 1. Для строки *i*=1 первый

несосед – вершина с номером 4: *j*=4. Складываем эти строки и формируем вторую цветовую группу: Colour2 = {1,4}.

```
 1111000
V0001110
 1111110
```

Обновленная *i*-ая строка: 1111110

Обновленная матрица смежности:

A :=

	0	1	2	3	4	5	6	
~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	**0**
1	1	1	1	1	1	0	**1**
1	1	1	1	0	1	1	**2**
~~0~~	~~1~~	~~1~~	~~1~~	~~1~~	~~0~~	~~1~~	**3**
~~0~~	~~0~~	~~0~~	~~1~~	~~1~~	~~1~~	~~0~~	**4**
~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~	**5**
1	0	1	1	0	0	1	**6**

Ищем новую несмежную с *i* вершину: ее номер – 6, то есть *j*=6. Номер строки *j* добавляется во вторую цветовую группу:

Colour2 = {1,4,6}.Складываем строки с номерами *i* и *j*:

```
 1111110
V1011001
 1111111
```

Обновленная строка *i* вся состоит из 1, поэтому набор в цветовую группу два – закончен.

Обновленная матрица смежности:

A :=

	0	1	2	3	4	5	6	
~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	**0**
~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	~~1~~	**1**
1	1	1	1	0	1	1	**2**
~~0~~	~~1~~	~~1~~	~~1~~	~~1~~	~~0~~	~~1~~	**3**
~~0~~	~~0~~	~~0~~	~~1~~	~~1~~	~~1~~	~~0~~	**4**
~~0~~	~~0~~	~~1~~	~~0~~	~~1~~	~~1~~	~~0~~	**5**
~~1~~	~~0~~	~~1~~	~~1~~	~~0~~	~~0~~	~~1~~	**6**

Так как набор во вторую цветовую группу закончен, будем составлять новую – следующую цветовую группу.

Теперь *i* принимает значение 2, так как следующая неокрашенная после 1-ой стоки – строка с номером 2. Для строки *i*=2 первая и единственная несмежная вершина – вершина номер 4: *j*=4, но она уже окрашена Следовательно, в третью цветовую группу попадает единственная вершина в номером 2: Colour3 = {2}, и алгоритм полностью заканчивает свою работу, так как всем вершинам назначены цвета.

Ответ: число цветов раскраски для графа G(V,E): 3

Вершинам назначены цвета следующим образом:

0 - 1
1 - 2
2 - 3
3 - 1
4 - 2
5 - 1
6 - 2

4.3. Псевдокод разработанного алгоритма

S_k - независимое множество вершин , покрашенных в цвет k.

```
begin
1.  k := 1;
2.  for  i=0 to n
3.      if i ∉ V  then  continue
        /*vertex i is coloured*/
4.      Sk := Sk ⋃ {i};
5.      V := V\{i};
6.      while  i has not coloured non-neighbours   /* row i contains 0*/
7.          for  j=i+1  to  n-1
            /* find the first non- neighbor*/
8.              if  aij = 0 AND j ∈ V  then  break
9.          end for
10.         if  j = n  then  break
11.         Sk := Sk ⋃ {j};
12.         V := V\{j};
13.         Ai := Ai ∨ Aj;
14.     end while
15.     k := k+1;
16. end for
end
```

4.4. “Битовая” реализация разработанного алгоритма

Для повышения эффективности работы разработанного алгоритма, а именно, улучшение такого его свойства как скорость работы, была реализована версия алгоритма, основанная на работе с битовым представлением матрицы смежности A графа G(V,E).

Каждая строка матрицы смежности разделена на битовые части, размерность которых определяется архитектурой ЭВМ, где будет запускаться алгоритм (32 или 64 бит). Номер битовой части строки определяется индексом *p*. Строка матрицы делится на части по 32(64) бита, чтобы можно было применить математические операции над числами, обрабатывая сразу 32(64) вершины.

Одним из факторов, увеличивающих скорость работы этой версии алгоритма, является функция поиска первой несоседней вершины для вершины *i*, то есть первого 0 в строке *i*, или первой

единицы в инвертированной строке – getLeastSignificantOne().

Операция $\log_2$ для нахождения первой несоседней неокрашенной вершины в строке матрицы смежности оказалась слишком медленной, и была найдена специальная операция bsf – bit scan forward, поддерживаемая большинством современных процессоров, в частности процессорами Intel x86. Эта операция возвращает первый значащий бит (то есть первую единицу) двоичного числа. Таким образом, для несоседней неокрашенной вершины справедливо равенство: j=bsf($\overline{A_j}$), где $\overline{A_j}$ - строка *j* инвертированной матрицы смежности A. Заметим, что в семействе процессоров Intel iCore операция bsf требует всего 1 такт процессора.

В операционных системах Linux существует встроенная функция ffs(), которая содержит в своей реализации операцию bsf и возвращает номер позиции первой единицы в двоичного слова. Данную функция была использована для реализации битовой версии алгоритма, так как готовая реализация этой функции эффективно поддерживает различные архитектуры процессоров: как 32-х, так и 64-х битных. getLeastSignificantOne() – реализация функций ffs, ffsl, ffsll.

Важно отметить, что функция находит первую 1 в битовом слове, а нам нужен первый 0. Поэтому для использования функции getLeastSignificantOne() в программе, понадобилось использовать инвертированную матрицу смежности вершин $\bar{A}$, где 0 – означает, что вершины смежны, а 1 – что несмежны. Соответственно на главной диагонали инвертированной матрицы должны стоять нули.

Кроме того, важным аспектом битовой версии алгоритма является реализация метода умножения (конъюнкции) строк матрицы смежности, которое производится при помощи битовой операции &, что в свою очередь, также уменьшает время работы алгоритма. Операция конъюнкции возникает в результате инвертирования матрицы: $A_i \vee A_j = \overline{\overline{A_i} \wedge \overline{A_j}}$ (формула де Моргана).

4.5. Псевдокод “битовой” версии разработанного алгоритма

BITS – 32 или 64 в зависимости от архитектуры;

$A_{i,p}$ – 32(64)-х битная часть *p* строки *i* инвертированной матрицы;

parts – число частей *p* в строке матрицы, *parts*=[n/BITS]

clearBit($M_{i,j}$, k) */*clears bit number “k” to 0 in order to avoid situations when one vertex is in two colour groups*/*

begin

1. k := 1;
2. **for** i=0 to n-1
   3. **if** i ∉ V **then** continue
      */*vertex i is coloured*/*
   4. $S_k := S_k \cup \{i\}$;
   5. V := V\{i};
   6. p := 0;
      /* start from the first part */
   7. **while** true
      8. bitIndex = *getLeastSignificantOne($A_{i,p}$);*
      9. **if** bitIndex = 0 **then**
         /* no ones */
      10. p := p+1;
      11. **if** p = parts **then** break
      12. continue
      13. **end if**
      14. nonNeighbour := BITS * p + (bitIndex - 1);
      15. **if** nonNeighbour ∉ V **then**
      16. *clearBit($A_{i,p}$, bitIndex - 1);*
      17. continue;
      18. **end if**
      19. $S_k := S_k \cup \{$ nonNeighbour $\}$;
      20. V := V\{ nonNeighbour };
      21. **for** q= p to parts-1
      22. $A_{i,q} = A_{i,q} \& A_{nonNeighbour,q}$;
      23. **end for**
   24. **end while**
   25. k := k+1;
26. **end for**

end

5. Практические результаты

Разработанный алгоритм был реализован в двух версиях. В одном случае каждая строка матрицы смежности представляет собой набор нулей и единиц и математические операции не используются. Другая реализация алгоритма основана на битовом представлении строк матрицы смежности и математических операциях: конъюнкции и bsf.

Эксперименты были проведены на процессоре Intel Core i3 CPU 2.27GHz на 32-разрядной операционной системе Linux (BITS = 32). Работа алгоритмов проверялась на графах DIMACS.

Отметим, что точность обоих реализаций алгоритма одинакова и сравнима с алгоритмом GIS[6].

Битовая математическая реализация алгоритма должна показать результаты по времени лучшие, чем другая версия алгоритма. Это объясняется тем, что уменьшается общее количество шагов алгоритма: битовое

представление строк матрицы смежности позволяет избежать перебора элементов матрицы смежности благодаря использованию функции ffs(), возвращающей номер позиции первого значащего бита, для поиска первой несоседней вершины; также операция конъюнкции обрабатывает за 1 такт сразу 32(64) вершины.

Результаты экспериментов приведены в таблице 1(Приложение 1).

Из таблицы видно, что оба алгоритма находят хроматическое число не всегда точно, то есть иногда определяют количество цветов неправильно. Что касается скорости работы, битовая реализация алгоритма оказалась эффективнее, показала лучшие результаты в плане скорости поиска решений. Ускорение составляет от 2 до 53 раз. Среднее ускорение: 16.9 раз

6. Заключение

В данной статье представлен эвристический алгоритм решения задачи о раскраске графа. Идея алгоритма заключается в применении математических операций над матрицей смежности графа в ее битовом представлении.

Основными достоинствами разработанного алгоритма являются:

- работа алгоритма с битовым представлением матрицы смежности графа;
- быстрый поиск первой несоседней вершины при помощи функции ffs();
- побитовая операция сложения строк матрицы смежности.

Был проведен сравнительный анализ скоростей работы “математической” версии алгоритма с обычной реализацией этого же алгоритма. Результаты подтвердили тот факт, что “математическая” реализация алгоритма обладает более высокой скоростью вычислений. Полученные в работе результаты можно использовать при решении задач, таких как: составление расписаний, задачи о максимальной клике.

Список литературы

[1] D. Cosmin Porumbela, *A search space “cartography” for guiding graph coloring heuristics*, Computers &Operations Research, 2010, №37, 769-778.

[2] *Intel® 64 and IA-32 Architectures*, Optimization Reference Manual, C.3 Latency and Throughput, 2012, 800.

[3] Isabel Méndez-Díaz, *A Branch-and-Cut algorithm for graph coloring*, Discrete Applied Mathematics, 2009, №5, 826–847.

[4] A. Kosowski, *Classical Coloring of Graphs* , AMS Contemporary Math, 2011, 1-20.

[5] E. Malaguti, *An exact approach for the Vertex Coloring Problem*, Discrete Optimization, 2011, №8,174-190.

[6] A. Radin *Graph Coloring Heuristics from Investigation of Smallest Hard to Color Graphs*, 2008, 119.

[7] D. J. A. Welsh, M. B. Powell, *An upper bound for the chromatic number of a graph and its application to timetabling problems*, The Computer Journal, 1967, 10, 85-86.

[8] D. M. Matula, B. G. Marble, J.D. Isaacson, *Graph Coloring Algorithms.* In Graph Theory and Computing, Academic Press, 1972, 109-122.

[9] D. Brelaz, *New methods to color the vertices of a graph*, Communications of ACM, 1979, 22, 251-256.

[10]D.S. Johnson, *Worst case behavior of graph coloring algorithms*. Proceedings of the Fifth Southeastern Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica Publishing, 1974, 513-528.

7. Приложение

“Таблица 1. Сравнение результатов работы алгоритмов”

G(V,E)	V	E	χ(G)	Colours	time	bittime
frb30-15-1	450		30	30	4070	280
frb53-24-1	1272		53	53	155540	2910
frb59-26-1	1534		59	59	247770	4120
frb59-26-5	1534		59	59	248450	4160
frb100-40.clq(1)	4000	7425226	100	100	2940	80
brock200_2.clq	200	9876		36	970	140
zeroin.i.1.col	211	4100	49	49	1360	210
brock400_2.clq	400	59786		100	9380	1180
brock800_3.clq	800	207333		143	153650	2880
c-fat200-5.clq	200	8473		68	1480	260
c-fat500-10.clq	500	46627		126	21450	2040
gen200_p0.9_44.clq	200	17910		76	1710	270
gen400_p0.9_75.clq	400	71820		143	12950	1620
hamming6-2.clq	64	1824		30	100	32
hamming10-4.clq	1024	434176		128	247060	4580
johnson32-2-4.clq	496	107880		30	6160	480
keller5.clq	776	225990		175	166170	4540
MANN_a81.clq(1)	3321	5506380		1134	22770	450
p_hat300-3.clq	300	33390		85	4570	500
p_hat700-2.clq	700	121728		114	81910	1730
p_hat1000-1.clq	1000	122253		69	126790	2600
p_hat1500-3.clq(10)	1500	847244		326	12810	170
san200_0.9_3.clq	200	17910		73	1640	240
san400_0.9_1.clq	400	71820		163	14710	1940
sanr200_0.9.clq	200	17863		82	1880	270
sanr400_0.7.clq	400	55869		91	8690	1090
anna.col	138	493	11	12	190	40
david.col	87	406	11	12	90	20
fpsol2.i.3.col	425	8688	30	30	4220	290
games120.col	120	638	9	9	180	20
homer.col	561	1629	13	15	5860	250
huck.col	74	301	11	11	60	20
inithx.i.3.col	621	13969	31	31	16270	680
jean.col	80	254	10	10	80	20
le450_5d.col	450	9757	5	18	3340	200
le450_15a.col	450	8168	15	22	4040	260
le450_25b.col	450	8263	25	27	4840	300
miles250.col	128	387	8	9	180	20
miles750.col	128	2113	31	34	380	90
miles1000.col	128	3216	41	44	460	110
miles1500.col	128	5198	71	76	640	160
queen6_6.col	36	290	7	11	20	10
queen8_12.col	96	1368	12	15	130	40
queen9_9.col	81	1056	10	16	120	30
queen10_10.col	100	2940	10	16	170	30
queen13_13.col	169	6656	13	21	450	80
queen15_15.col	225	10360	15	25	1020	120
myciel3.col	11	20	4	4	0	0
myciel7.col	191	2360	8	8	290	40
mulsol.i.5.col	185	3973	31	31	630	110
zeroin.i.2.col	211	3541	30	30	870	120
zeroin.i.3.col	206	3540	30	30	800	130

G(V,E) – неориентированный граф, V – количество вершин, E – количество ребер, χ(G) – хроматическое число, Colours –количество цветов, полученное эвристикой, time - время работы обычной реализации алгоритма в мс (запуск 1000 раз), bittime - время работы “битовой” реализации алгоритма в мс (запуск 1000 раз)