УДК 519.17, 519.8

Лариса Федоровна Комоско, стажер-исследователь, lkomosko@hse.ru

Михаил Владимирович Бацын, ведущий научный сотрудник,

к.ф.-м.н., mbatsyn@hse.ru*

Лаборатория алгоритмов и технологий анализа сетевых структур,

Национальный исследовательский университет Высшая школа экономики,

Нижний Новгород

Эффективная раскраска графа с помощью битовых операций

Аннотация. В статье представлен новый эффективный эвристический алгоритм для решения задачи о раскраске графа. Предложенный алгоритм строит ту же раскраску графа, что и широко используемый жадный последовательный алгоритм раскраски, в котором на каждом шаге текущая вершина красится в минимальный допустимый цвет. Вычислительные эксперименты показывают, что представленный алгоритм выполняет раскраску графа гораздо быстрее по сравнению со стандартным жадным алгоритмом. Ускорение для графов библиотеки DIMACS достигает 5,6 раз.

Ключевые слова: раскраска графа; эвристика; битовые операции; жадный алгоритм; последовательная раскраска

1. Введение.

Задача о раскраске графа является известной задачей комбинаторной оптимизации. Это одна из двадцати одной NP-полной задачи Ричарда Карпа [1]. Данный класс задач известен тем, что нельзя найти точного решения за разумное время, так как пространство поиска решений увеличивается в экспоненциальной зависимости от входных данных.

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

* М.В. Бацын, +79506115777, 603093, г. Н. Новгород, ул. Родионова 136, ауд. 401
Работа поддержана грантом РНФ 14-41-00039

эффективности алгоритмов раскраски с 1996 активно используется набор графов библиотеки DIMACS.

Задача о раскраске графа имеет большое число практических приложений. Некоторые задачи теории расписаний сводятся к ней. При этом вершинами графа являются занятия, которые нужно включить в расписание, а цветами – интервалы времени, в которые параллельно проходят несколько занятий. Ребро проводится, если два занятия не могут проходить одновременно.

Задача распределения регистров процессора между переменными программы при компиляции также сводится к задаче о раскраске графа. При этом вершинами являются переменные, а цветами – регистры процессора. Ребро проводится, если времена жизни переменных в программе пересекаются.

Задача о назначении частот станциям связи моделируется с помощью раскраски графа. Станции моделируются как вершины графа, а частоты – как цвета вершин. Ребро проводится, если две станции находятся рядом и между ними возможна интерференция сигналов.

Задачи кластеризации данных могут быть представлены задачей о раскраске графа. При этом отдельные записи данных – это вершины графа, а кластеры – цветовые классы. Ребро между двумя вершинами проводится, если соответствующие записи данных сильно отличаются и не должны попасть в один кластер.

Существует большое количество точных [2,3,4] и эвристических [5,6,7,8,9,10] алгоритмов раскраски графа. Среди быстрых классических эвристик широко используются следующие: Greedy, DSATUR, GIS (Greedy Independent Sets) [11]. Необходимо отметить, что эти алгоритмы представляют собой последовательность простых действий, условных переходов и циклов без использования каких-либо математических операций.

В настоящей работе представлен алгоритм раскраски графа, основанный на битовых операциях над булевыми матрицами. Предложенный алгоритм работает только с булевыми данными и использует эффективные битовые операции процессора для их обработки. Такие операции позволяют

обрабатывать 64 вершины графа за одну операцию (при использовании 64-битной архитектуры). Для этой цели граф представляется булевой матрицей смежности, а цвета раскраски хранятся в специальной булевой матрице «запрещенных» цветов. Проведены вычислительные эксперименты, показывающие эффективность предложенного алгоритма.

2. Основные термины и определения. Графом G называется совокупность двух множеств: $G=(V,E)$, где V – это множество вершин, $|V|=n$, а $E\subset V\times V$ – множество ребер между ними, $|E|=m$. Две вершины $u,v\in V$ называются смежными, если они связаны ребром: $\{u,v\}\in E$. Две ребра $e, f\in E$ называются смежными, если они имеют общую вершину: $e\cap f\neq\phi$. Вершина v инцидентна ребру e (а ребро e инцидентно вершине v), если она является концом этого ребра: $v\in e$. Степенью $\deg(v)$ вершины v в графе G называют число ребер, инцидентных этой вершине: $\deg(v)=|\{e\in E: v\in e\}|$.

Раскраской вершин графа $G=(V,E)$ называется функция $c:V\to \mathrm{N}$, которая каждой вершине графа ставит в соответствие натуральное число (цвет) так, что любым двум смежным вершинам $u,v\in V$ назначаются разные цвета: $\{u,v\}\in E\Rightarrow c(u)\neq c(v)$. Функция c называется функцией раскраски. Граф G, для которого существует раскраска из k цветов, называется k-раскрашиваемым. В этом случае функция раскраски разбивает граф G на независимые множества $V_1,...,V_k$, внутри которых никакие две вершины не связаны ребром. Наименьшее число k, для которого существует k-раскраска графа G, называется хроматическим числом графа G и обозначается $\chi(G)$. Любая раскраска графа G, имеющая ровно $\chi(G)$ цветов называется хроматической.

3. Математическая модель задачи о раскраске графа. Обозначим за k максимальное число цветов, в которые всегда с гарантией можно покрасить граф G. Это может быть тривиальная оценка сверху на число цветов: $k=n$ или более точная оценка: $k=\max\limits_{v\in V}\deg(v)+1$. Хорошо известна следующая математическая модель задачи о раскраске (см., например, [3]).

Переменные:

$x_{ih} \in \{0,1\}$, $x_{ih} = 1$, если вершине i назначен цвет h

$y_h \in \{0,1\}$, $y_h = 1$, если цвет h использован в раскраске

Целевая функция:

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

Ограничения:

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

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

Целевая 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.

4. Описание алгоритма. Разработанный алгоритм использует битовые операции над матрицей смежности и специальной матрицей запрещенных цветов. Матрица смежности A – это квадратная матрица размера $n \times n$, в которой значение элемента $a_{ij} = 1$, если вершины i и j соединены ребром, и $a_{ij} = 0$ иначе. Матрица запрещенных цветов C – это тоже квадратная матрица размера $n \times n$. Значение элемента $c_{ij} = 1$, если цвет i запрещен для вершины j, поскольку один из ее соседей уже покрашен в этот цвет.

Над матрицами A и C выполняются следующие операции. В матрице C, начиная с первой вершины графа $j = 1$ (с первого столбца матрицы), производится поиск по этому столбцу первого свободного для этой вершины цвета. То есть определяется минимальное значение i, для которого $c_{ij} = 0$ (определяется минимальный цвет i, разрешенный для вершины j). Вершине j назначается это цвет i.

Далее необходимо запретить цвет i всем соседям вершины j. Для этого выполняется битовая операция “или” над j-ой строкой матрицы смежности и i-ой строкой матрицы запрещенных цветов – операция дизъюнкции булевых

векторов, представляющих собой эти строки. Результат записывается в *i*-ю строку матрицы запрещенных цветов: $C_i = C_i \vee A_j$. Поскольку в строке A_j единицы стоят у соседей j, то в результате эти единицы появятся в строке C_i, запрещая цвет i этим соседям. Затем переходим к следующей вершине $j = j + 1$ (следующему столбцу матрицы *C*), и операции поиска свободного цвета и запрещения его для соседей j повторяются аналогичным образом.

Алгоритм останавливается, когда будут перебраны все вершины графа *G*. Раскраска, полученная в результате, описывается матрицей *C*. Для каждой вершины j первый ноль в столбце j матрицы C определяет цвет этой вершины.

Пусть дан граф G с вершинами $V = \{1,...,7\}$, $n = 7$, изображенный на рис. 1.

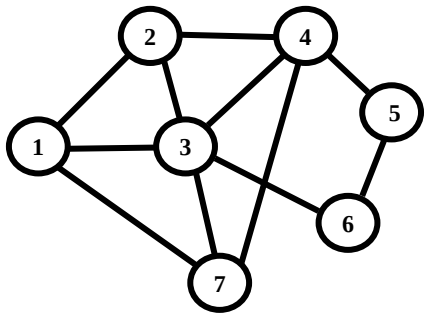

Рис. 1. Граф для демонстрации алгоритма

Построим для графа G матрицу смежности *A* и матрицу запрещенных цветов *C*:

$A =$

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	1
2	1	0	1	1	0	0	0
3	1	1	0	1	0	1	1
4	0	1	1	0	1	0	1
5	0	0	0	1	0	1	0
6	0	0	1	0	1	0	0
7	1	0	1	1	0	0	0

$C =$

	1	2	3	4	5	6	7
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

Операции с матрицей запрещенных цветов начинаются с первого столбца: $j = 1$. Ищем в столбце $j = 1$ первый свободный цвет, то есть первый ноль (сверху

вниз). Первый ноль находится в строке $i=1$, значит, вершина 1 будет покрашена в 1-й цвет. Теперь нужно запретить этот цвет для соседей вершины 1. Для этого к строке 1-го цвета $i=1$ матрицы C мы «добавляем» строку $j=1$ матрицы A:

$$\begin{array}{l} 0000000 \\ \vee\ 0110001 \\ \hline 0110001 \end{array}$$

Теперь новая строка $i=1$ матрицы запрещенных цветов выглядит следующим образом: 0110001. Таким образом, цвет 1 запрещен для вершин 2, 3 и 7 (соседей вершины 1).

Далее все те же операции повторяются для всех вершин графа $j=2,3,4,5,6,7$. Вершина 2 будет покрашена в цвет 2, этот цвет будет запрещен для ее соседей, и строка 2 матрицы C будет равна: 1011000. Вершина 3 будет покрашена в цвет 3, и после запрещения этого цвета для ее соседей строка 3 матрицы C будет равна: 1101011. Четвертой вершине будет назначен 1-й цвет, и в первой строке матрицы C этот цвет будет запрещен еще для 5-й вершины: 0110101. Вершину 5 мы покрасим во 2-й цвет, и запретим этот цвет еще для 6-й вершины: 1011010. Шестая вершина будет покрашена в 1-й цвет, но после «сложения» с 6-й строкой матрицы A строка 1 матрицы C не изменится, потому что цвет 1 уже запрещен для всех соседей вершины 6: 0110101. Вершина 7 будет покрашена во второй цвет. Так как это последняя вершина, то операцию дизъюнкции больше нет смысла выполнять. В результате матрица C будет иметь следующий вид:

$C =$

	1	2	3	4	5	6	7
1	0	1	1	0	1	0	1
2	1	0	1	1	0	1	0
3	1	1	0	1	0	1	1
4	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0

Таким образом, для раскраски графа в нашем примере необходимо 3 цвета. Номер цвета для вершины определяется первым нулем в столбце этой вершины в матрице *C*. Вершинам 1, 2, 3, 4, 5, 6, 7 назначены соответственно цвета: 1, 2, 3, 1, 2, 1, 2.

5. Псевдокод алгоритмов. Для битового представления матриц *A* и *C* каждая строка матриц разделяется на битовые части, размер которых определяется архитектурой ЭВМ (например, 64 бита). Это делается для того, чтобы можно было выполнять битовые операции над числами, обрабатывая сразу несколько вершин (64 вершины при 64-битной архитектуре процессора). Номер битовой части строки задается индексом *p* в псевдокоде, приведенном ниже. Дизъюнкция строк матрицы смежности и матрицы запрещенных цветов выполняется при помощи битовой операции "|" над 64-битными частями этих строк. Чтобы получить бит номер *b* из 64-битного числа используется битовая операция конъюнкции "&" со специальной маской, равной числу, в котором все биты – нули, кроме бита на позиции *b*, который устанавливается в 1. Такую маску можно получить с помощью операции побитового сдвига "<<" единицы на *b* битов влево (см. псевдокод ниже). Конъюнкция с такой маской равна 0, если бит *b* в 64-битном числе равен 0, иначе результатом конъюнкции является сама маска, то есть ненулевое число.

В приведенном далее псевдокоде номера вершин, цветов и битовых частей строк нумеруются, начиная с нуля. Следующие переменные используются в псевдокоде:

BITS – 8, 16, 32 или 64 в зависимости от архитектуры процессора

A_{ip} – *BITS*-битная часть *p* строки *i* матрицы смежности *A*

C_{ip} – *BITS*-битная часть *p* строки *i* матрицы цветов *C*

$parts = \lceil n / BITS \rceil$ – число битовых частей в строках матриц *A* и *C*

Алгоритм 1. Битовая жадная раскраска графа

function Bit-Greedy-Color($A[n \times n]$)

output: $C[n \times n]$

$C = 0$

for $j = 0$ to $n - 1$

for $i = 0$ to $n - 1$

$c_{ij} =$ Get-Bit(i, j, C)

if $c_{ij} = 0$ **then**

break

end if

end for

for $p = 0$ to $parts - 1$

$C_{ip} = C_{ip} \mid A_{jp}$ *// обработать BITS вершин за 1 операцию*

end for

end for

return C

end function

function Get-Bit(i, j, C)

output: c_{ij}

$p = j / BITS$ *// индекс битовой части, в которой хранится бит j*

$b = j \% BITS$ *// индекс бита j в этой части*

$mask = 1 << b$ *// маска – число, в котором только бит b равен 1*

return $C_{ip} \& mask$ *// конъюнкция с маской, равна 0, если бит b в части* C_{ip} *– ноль*

end function

Предложенный алгоритм сравнивается в работе со стандартным жадным алгоритмом раскраски Greedy-Color, чей псевдокод приведен далее. Номера вершин и цветов в этом псевдокоде начинаются с единицы. В стандартном алгоритме раскраска хранится в наборе цветовых классов $C_1, C_2, ..., C_k$, где множество C_i содержит все вершины, покрашенные в цвет i.

Алгоритм 2. Стандартная жадная раскраска графа

function Greedy-Color$(A[n \times n])$

output: $C_1, C_2, ..., C_k$

$k = 1$ *// максимальный цвет, использованный в раскраске*

for $i = 1$ to n *// для каждой вершины i*

$color = 1$ *// текущий цвет*

while Has-Neighbors(i, C_{color}, A) *// если в цвете color есть соседи i,*

$color = color + 1$ *// то переходим к следующему цвету*

end while

if $color > k$ **then**

$k = color$ *// обновляем максимальный цвет*

end if

$C_{color} = C_{color} \cup \{i\}$ *// красим вершину i в цвет color*

end for

end function

function Has-Neighbors(i, C_{color}, A)

output: true, если у вершины i есть соседи из множества C_{color}

for $j \in C_{color}$

if $A_{ij} = 1$ **then**

return true

end if

end for

return false

end function

---

6. Результаты вычислительных экспериментов. Эксперименты выполнялись на процессоре Intel Xeon X5690 3.47 GHz с 64-битной архитектурой (*BITS* = 64). Работа алгоритмов проверялась на графах библиотеки DIMACS[12]. Отметим, что оба алгоритма всегда возвращают одинаковую раскраску графа. Результаты экспериментов приведены в таблице 1. Поскольку раскраска одного графа занимает доли миллисекунд, вычисление раскраски повторялось 1 миллион раз

для каждого графа, и в таблицу заносилось общее время выполнения 1 миллиона раскрасок в секундах.

Graph	***n***	***m***	**Bit-Greedy-Color, sec**	**Greedy-Color, sec**	**Speedup**
brock200_1	200	14834	12	34	2,8
brock200_2	200	9876	10	32	3,2
brock200_3	200	12048	8	42	5,3
brock200_4	200	13089	12	32	2,7
brock400_1	400	59723	42	124	3,0
brock400_2	400	59786	38	108	2,8
brock400_3	400	59681	40	118	3,0
brock400_4	400	59765	38	102	2,7
brock800_1	800	207505	110	434	3,9
brock800_2	800	208166	110	438	4,0
brock800_3	800	207333	112	420	3,8
brock800_4	800	207643	112	496	4,4
c-fat200-1	200	1534	10	16	1,6
c-fat200-2	200	3235	6	16	2,7
c-fat200-5	200	8473	12	24	2,0
c-fat500-1	500	4459	16	68	4,3
c-fat500-10	500	46627	66	124	1,9
c-fat500-2	500	9139	26	78	3,0
c-fat500-5	500	23191	46	106	2,3
C1000.9	1000	450079	268	710	2,6
C125.9	125	6963	8	12	1,5
C250.9	250	27984	20	40	2,0
C500.9	500	112332	78	174	2,2
dsjc1000.1.col.txt	1000	49629	72	240	3,3
dsjc1000.5.col.txt	1000	249826	140	790	5,6

dsjc500.1.col.txt	500	12458	24	70	2,9
dsjc500.5.col.txt	500	62624	46	204	4,4
frb30-15-1	450	83198	22	44	2,0
frb30-15-2	450	83151	26	56	2,2
frb30-15-3	450	83216	22	42	1,9
frb30-15-4	450	83194	22	46	2,1
frb30-15-5	450	83231	22	45	2,1
gen200_p0.9_44	200	17910	12	32	2,7
gen200_p0.9_55	200	17910	12	28	2,3
gen400_p0.9_55	400	71820	52	100	1,9
gen400_p0.9_65	400	71820	48	88	1,8
gen400_p0.9_75	400	71820	60	108	1,8
hamming10-4	1024	434176	2,7	4,8	1,8
hamming6-2	64	1824	1,2	2,8	2,3
hamming6-4	64	704	28	44	1,6
hamming8-2	256	31616	12	34	2,8
hamming8-4	256	20864	3,4	4,3	1,3
johnson16-2-4	120	5460	27	25	0,9
johnson32-2-4	496	107880	0,5	0,4	0,8
johnson8-2-4	28	210	2,5	3,6	1,4
johnson8-4-4	70	1855	8	19	2,5
keller4	171	9435	108	224	2,1
keller5	776	225990	50	68	1,4
MANN_a27	378	70551	1,6	1,8	1,2
MANN_a45	1035	533115	86	358	4,2
MANN_a9	45	918	146	514	3,5
p_hat1000-1	1000	122253	188	606	3,2
p_hat1000-2	1000	244799	14	48	3,4
p_hat1000-3	1000	371746	14	48	3,4
p_hat1500-1	1500	284923	26	60	2,3

p_hat1500-2	1500	568960	26	100	3,8
p_hat1500-3	1500	847244	40	132	3,3
p_hat300-1	300	10933	56	146	2,6
p_hat300-2	300	21928	56	218	3,9
p_hat300-3	300	33390	66	192	2,9
p_hat500-1	500	31569	112	320	2,9
p_hat500-2	500	62946	76	168	2,2
p_hat500-3	500	93800	12	28	2,3
p_hat700-1	700	60999	10	18	1,8
p_hat700-2	700	121728	16	30	1,9
p_hat700-3	700	183010	16	30	1,9
san1000	1000	250500	20	26	1,3
san200_0.7_1	200	13930	22	44	2,0
san200_0.7_2	200	13930	34	96	2,8
san200_0.9_1	200	17910	28	74	2,6
san200_0.9_2	200	17910	26	62	2,4
san200_0.9_3	200	17910	54	110	2,0
san400_0.5_1	400	39900	10	38	3,8
san400_0.7_1	400	55860	16	24	1,5
san400_0.7_2	400	55860	28	126	4,5
san400_0.7_3	400	55860	36	112	3,1
san400_0.9_1	400	71820	12	34	2,8
sanr200_0.7	200	13868	10	32	3,2
sanr200_0.9	200	17863	8	42	5,3
sanr400_0.5	400	39984	12	32	2,7
sanr400_0.7	400	55869	42	124	3,0

Таблица 1. Результаты вычислительных экспериментов

Ускорение разработанного алгоритма по сравнению со стандартным отражено в столбце Speedup и достигает 5,6 раз. Среднее ускорение составляет 2,6 раза.

7. Заключение. В работе представлен эвристический алгоритм для решения задачи о раскраске графа. Идея алгоритма заключается в применении эффективных битовых операций процессора над булевыми матрицами: матрицей смежности графа и специальной матрицей запрещенных цветов. Основными достоинствами разработанного алгоритма являются:

- Работа алгоритма с битовыми представлениями матриц, что резко сокращает требуемый объем памяти.
- Побитовая операция сложения строк матрицы смежности и матрицы запрещенных цветов, что значительно сокращает время вычислений.

Был проведен сравнительный анализ времени работы предложенного алгоритма с широко известным жадным алгоритмом Greedy-Color. Алгоритм Bit-Greedy-Color показал более высокую скорость вычислений – в 2,6 раза быстрее в среднем. Полученные в работе результаты можно использовать при решении ряда задач дискретной оптимизации. К задаче о раскраске графа могут быть сведены различные задачи, возникающие при планировании производства, составлении расписаний, кластеризации данных, распределении регистров процессора при компиляции, назначении частот станциям связи, оптимизации количества фаз сигналов светофоров и др. Кроме того, решение задачи о раскраске графа используется как верхняя граница в точных алгоритмах решения задачи о максимальной клике и ее аналогах: задачи о максимальном независимом множестве и задачи о минимальном вершинном покрытии. При этом раскраску различных подграфов исходного графа необходимо вычислять большое число раз, поскольку дерево решений в различных методах ветвей и границ, ветвей и отсечений и др. содержит большое число узлов.

Список литературы

1. **Karp R.M.** Reducibility Among Combinatorial Problems // Complexity of Computer Computations, 1972, P. 85-103.
2. **Méndez-Díaz I., Zabala P.** A Branch-and-Cut algorithm for graph coloring // Discrete Applied Mathematics, 2006, V. 154, N. 5, P. 826-847.
3. **Malaguti E., Monaci M., Toth P.** An exact approach for the Vertex Coloring Problem // Discrete Optimization, 2011, V. 8, N. 2, P. 174-190.
4. **San Segundo P.** A new DSATUR-based algorithm for exact vertex coloring // Computers and Operations Research, 2012, V. 39, N. 7, P. 1724-1733.
5. **Welsh D. J. A., Powell M. B.** An upper bound for the chromatic number of a graph and its application to timetabling problems // The Computer Journal, 1967, V. 10, N. 1, P. 85-86.
6. **Matula D. M., Marble B. G., Isaacson J. D.** Graph Coloring Algorithms // Graph Theory and Computing, 1972, P.109-122.
7. **Johnson D. S.** Worst case behavior of graph coloring algorithms // Proceedings of the Fifth Southeastern Conference on Combinatorics, Graph Theory and Computing - Canada. Utilitas Mathematica Publishing, 1974, P. 513-528.
8. **Brelaz D.** New methods to color the vertices of a graph // Communications of ACM, 1979, V. 22, N. 4, P. 251-256.
9. **Radin A. A.** Graph Coloring Heuristics from Investigation of Smallest Hard to Color Graphs. New York, USA. MS Thesis. Rochester Institute of Technology, 2000, P. 119.
10. **Porumbela D. C., Hao J. K., Kuntz P.** A search space “cartography” for guiding graph coloring heuristics // Computers & Operations Research, 2010, V. 37, N. 10, P. 769-778.
11. **Kosowski A., Manuszewski K.** Classical Coloring of Graphs // Graph Colorings, 2004, V.352, P. 1-20.
12. **DIMACS challenges**. http://dimacs.rutgers.edu/Challenges/

L.F. Komosko, research intern, lkomosko@hse.ru
M.V. Batsyn, leading research fellow, candidate of physical-mathematical sciences, mbatsyn@hse.ru
Laboratory of Algorithms and Technologies for Networks Analysis,
National Research University Higher School of Economics,
Nizhny Novgorod

Efficient graph coloring by means of bitwise operations

Abstract. Graph coloring problem is one of the classical combinatorial optimization problems. This problem consists in finding the minimal number of colors in which it is possible to color vertices of a graph so that any two adjacent vertices are colored in different colors.

The graph coloring problem has a wide variety of applications including timetabling problems, processor register allocation problems, frequency assignment problems, data clustering problems, traffic signal phasing problems, maximum clique problem, maximum independent set problem, minimum vertex cover problem and others.

In this paper a new efficient heuristic algorithm for the graph coloring problem is presented. The suggested algorithm builds the same coloring of a graph as does the widely used greedy sequential algorithm in which at every step the current vertex is colored into minimal feasible color. Computational experiments show that the presented algorithm performs graph coloring much faster in comparison with the standard greedy algorithm. The speedup reaches 5,6 times for DIMACS graphs.

Keywords: graph coloring; heuristic; bitwise operations; greedy algorithm; sequential coloring

References

1. **Karp R.M.** *Reducibility Among Combinatorial Problems*. Complexity of Computer Computations, 1972, P. 85-103.
2. **Méndez-Díaz I., Zabala P.** *A Branch-and-Cut algorithm for graph coloring*. Discrete Applied Mathematics, 2006, V. 154, N. 5, P. 826-847.

3. **Malaguti E., Monaci M., Toth P.** *An exact approach for the Vertex Coloring Problem.* Discrete Optimization, 2011, V. 8, N. 2, P. 174-190.
4. **San Segundo P.** *A new DSATUR-based algorithm for exact vertex coloring*. Computers and Operations Research, 2012, V. 39, N. 7, P. 1724-1733.
5. **Welsh D. J. A., Powell M. B.** *An upper bound for the chromatic number of a graph and its application to timetabling problems*. The Computer Journal, 1967, V. 10, N. 1, 85-86.
6. **Matula D. M., Marble B. G., Isaacson J. D.** *Graph Coloring Algorithms*. Graph Theory and Computing, 1972, P.109-122.
7. **Johnson D. S.** *Worst case behavior of graph coloring algorithms*. Proceedings of the Fifth Southeastern Conference on Combinatorics, Graph Theory and Computing - Canada. Utilitas Mathematica Publishing, 1974, P. 513-528.
8. **Brelaz D.** *New methods to color the vertices of a graph*. Communications of ACM, 1979, V. 22, N. 4, P. 251-256.
9. **Radin A. A.** *Graph Coloring Heuristics from Investigation of Smallest Hard to Color Graphs*. Thesis. Rochester Institute of Technology, 2000.
10. **Porumbela D. C., Hao J. K., Kuntz P.** *A search space "cartography" for guiding graph coloring heuristics*. Computers & Operations Research, 2010, V. 37, N. 10, P. 769-778.
11. **Kosowski A., Manuszewski K.** *Classical Coloring of Graphs*. Graph Colorings, 2004, V.352, P. 1-20.
12. **DIMACS challenges**. *http://dimacs.rutgers.edu/Challenges/*