МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

Кафедра дискретной математики и алгоритмики

САВИН

Дмитрий Дмитриевич

Влияние входных данных на эффективность алгоритмов раскраски графов

Дипломная работа

Научный руководитель:
Старший преподаватель
А.А. Буславский

Допущена к защите
«19» мая 2025г.
Заведующий кафедрой дискретной математики и алгоритмики,
доктор физико-математических наук, профессор В.М. Котов

Минск, 2025

ОГЛАВЛЕНИЕ

РЕФЕРАТ

Дипломная работа: 35 страниц, 3 таблицы, 19 источников.

Ключевые слова: ГРАФЫ, ЗАДАЧА РАСКРАСКИ, ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ.

Объект исследования: алгоритмы раскраски графов.

Цель работы: исследование влияния входных данных на эффективность алгоритмов раскраски графов.

Методы исследования: методы системного анализа, сравнительного анализа, обобщения существующих материалов.

Результаты: среди рассмотренных алгоритмов наилучшее качество решений обеспечило использование алгоритма муравьиной колонии, однако разница существенна только на графах с большим числом ребер.

Область применения: задачи оптимизации, теории расписаний, аллокации регистров.

РЭФЕРАТ

Дыпломная праца: 35 старонак, 3 табліцы, 19 крыніц.

Ключавыя словы: ГРАФЫ, ЗАДАЧА РАЗМАЛЁЎКІ, НАБЛІЖАНЫЯ АЛГАРЫТМЫ.

Аб'ект даследавання: алгарытмы размалёўкі графаў.

Мэта работы: даследаванне ўплыву ўваходных дадзеных на эфектыўнасць алгарытмаў размалёўкі графаў**.**

Метадалогія навуковага даследавання: метады сістэмнага аналізу, параўнальнага аналізу, абагульнення існуючых матэрыялаў.

Вынікі: сярод разгледжаных алгарытмаў найлепшая якасць рашэнняў забяспечыла выкарыстанне алгарытму мурашынай калоніі, аднак розніца істотная толькі на графах з вялікім лікам рэбраў.

Вобласці ўжывання: задачы аптымізацыі, тэорыі раскладаў, аллокации рэгістраў**.**

ABSTRACT

Thesis: 35 pages, 3 tables, 19 sources.

Keywords: GRAPHS, COLORING PROBLEM, INEXACT ALGORITHMS.

The object of study: graph coloring algorithms.

The aim of the study: research of the influence of input data on the effectiveness of graph coloring algorithms.

The methodology of scientific research: methods of system analysis, comparative analysis, generalization of existing materials**.**

Results: among the algorithms considered, the best quality of solutions was provided by the ant colony algorithm, but the difference is significant only on graphs with a large number of edges**.**

The fields of applications: optimization problems, theory of schedules, registers allocation.

ВВЕДЕНИЕ

Задача раскраски графа была сформулирована почти 200 лет назад, и звучит она так: «необходимо раскрасить вершины графа таким образом, чтобы никакие две смежные вершины не были покрашены в один цвет». Хотя формулировка задачи довольно проста, она до сих не имеет точного решения с полиномиальной сложностью для произвольного графа. При этом эта проблема имеет множество приложений в различных областях, включая планирование расписаний, оптимизацию ресурсов, анализ социальных сетей и многие другие. Таким образом исследование алгоритмов раскраски графов является важной темой в области теории графов и дискретной математики.

Целью данной дипломной работы является исследование влияния входных данных на эффективность алгоритмов раскраски графов.

Задачи: в ходе работы будут рассмотрены основные понятия и определения, связанные с раскраской графов, а также будут изучены основные теоретические результаты, связанные с алгоритмами раскраски графов. Кроме того, будет проведено сравнительное исследование некоторых алгоритмов раскраски графов, будут рассмотрены преимущества и недостатки в зависимости от входных данных задачи, а также их применимость в различных сценариях.

Гипотеза: различные алгоритмы позволяют эффективно находить достаточно точные решения задачи о раскраске графа при разных входных данных.

Методы исследования: сравнение, измерение, анализ.

Исследование алгоритмов раскраски графов имеет важное значение для развития теории графов и практических приложений. Полученные результаты и выводы могут быть использованы в различных областях, где требуется эффективное использование ресурсов и оптимальное планирование.

ГЛАВА 1. ЗАДАЧА РАСКРАСКИ ГРАФА

1.1 Постановка задачи

Обычно под задачей раскраски графа понимают такую задачу: необходимо раскрасить вершины графа так, чтобы никакие две вершины одного цвета не были смежными и количество цветов было минимальным.

Более формально её можно сформулировать так: пусть дан граф $G(V, E)$, где V – множество вершин графа, а E – множество рёбер. Тогда задача заключается в том, чтобы поставить каждой вершине $v \in V$ в соответствие натуральное число $c(v) \in \{1, 2, \dots, k\}$ так, что:

– k – минимально;

– $c(v) \neq c(u)$ для $\forall\{u, v\} \in E$.

В такой формулировке вместо конкретных цветов, таких как красный или зелёный, в качестве меток используются натуральные числа, что гораздо удобнее при больших размерностях задачи. Под выражением $c(v) = k$ будем понимать, что вершина v имеет цвет k.

1.2 Связанные определения

Перед рассмотрением непосредственно алгоритмов раскраски имеет смысл ввести некоторые определения.

Раскраска называется *полной*, если каждой вершине $v \in V$ поставлено в соответствие число $c(v)$, в противном случае раскраска называется *частичной*.

Раскраска называется *правильной*, если $c(v) \neq c(u)$ для $\forall\{u, v\} \in E$.

Раскраска называется *допустимой*, если она является полной и правильной.

Наименьшее k, при котором допустимая раскраска графа G существует, называют его *хроматическим числом* и обозначают $\chi(G)$. Допустимая раскраска G, в которой используется ровно $\chi(G)$ цветов называется *оптимальной*.

Цветовой класс – множество вершин графа, раскрашенных в один цвет. Для цвета $i \in \{1, 2, \dots, k\}$ цветовой класс имеет вид $\{ v \in V \mid c(v) = i \}$.

Независимое множество – подмножество взаимно несмежных вершин графа. Формально, $I \subseteq V$ независимо, если для $\forall u, v \in I$ верно $\{u, v\} \notin E$.

Клика – подмножество взаимно смежных вершин графа. Формально, $C \subseteq V$ – клика, если для $\forall u, v \in C$ верно $\{u, v\} \in E$.

1.3 Сложность задачи

Доказано, что задача о раскраске графа является NP-трудной. NP-трудными также являются связанные задачи о хроматическом числе графа, о максимальной клике и о максимальном независимом множестве графа[5]. Это значит, что алгоритм с полиномиальной сложностью для их решения либо не существует, либо ещё не изобретён. Таким образом для произвольного графа за полиномиальное время можно получить только приблизительный ответ. Тем не менее для некоторых видов графов существуют точные алгортмы, работающие за полиномиальное время.

1.4 Некоторые частные случаи с точными решениями

Пустой граф

Для любого n пустой граф на n вершинах можно раскрасить с использованием всего одного цвета, так как никакие две вершины не являются смежными. Таким образом, если граф G – пустой, то $\chi(G) = 1$. Обратное тоже верно – из $\chi(G) = 1$ следует, что G – пустой.

Полный граф

Для полного графа G на n вершинах $\chi(G) = n$, так как любые две вершины являются смежными, а значит должны быть разных цветов. Верно и обратное, если $\chi(G) = n$, то G – полный.

Двудольный граф

Для любого двудольного графа доли являются независимыми множествами, следовательно каждую из них можно раскрасить в один цвет. Таким образом если G – двудольный, то $\chi(G) = 2$. Обратное тоже верно, т.е. если $\chi(G) = 2$, то G – двудольный.

Цикл

Если граф $G = C_n$, возможны два случая:

– n чётно, тогда $\chi(G) = 2$;

– n нечётно, и в этом случае $\chi(G) = 3$.

Для доказательства можем просто выбрать одну вершину, покрасить её в цвет 1 и далее расставлять цвета, идя по циклу и чередуя цвета 1 и 2, пока не останется одна не раскрашенная вершина. Если n чётно, то оба её соседа будут цвета 1, и присвоив ей цвет 2 мы получим допустимую раскраску. В случае же нечётного n соседи последней веришны будут разных цветов, поэтому понадобится ещё один цвет 3 для построения допустимой раскраски.

Колесо

Если $G = W_n$, также возможны два случая:

– n чётно, тогда $\chi(G) = 3$;

– n нечётно, и в этом случае $\chi(G) = 4$.

Доказательство аналогично доказательству для цикла с той лишь разницей, что необходим дополнительный цвет для «центральной» вершины колеса.

Планарный граф

Наиболее значимым результатом, полученным для задачи раскраски планарного графа, является теорема о четырех красках, которая утверждает, что любой планарный граф является 4-раскрашиваемым. За время, пока гипотеза не была доказана, оценка на хроматическое число произвольного планарного графа постепенно улучшалась разными исследователями.

При доказательстве большого числа теорем для планарных графов используется следующий подход. Множество S (обычно небольших) плоских графов, называемых *конфигурациями*, *неустранимо* для класса M планарных графов, если каждый граф в M содержит конфигурацию из S в качестве подграфа. В зависимости от рассматриваемой задачи множество неустранимых конфигураций будет различаться.

Связь между этим множеством и проблемами раскраски известна уже давно, начиная с теоремы Хивуда о пяти цветах, которая основана на тривиальном факте, что каждый плоский граф имеет вершину степени не более 5. Почти все ранее известные теоремы о раскрасках планарных графов основаны на определенных множествах неустранимых конфигураций. Для доказательства зачастую используют метод сводимых конфигураций, идея которого заключается в следующем. При заданном типе раскраски сводимая конфигурация для этого типа раскраски представляет собой граф, который не может содержаться в минимальном планарном графе, который не может быть раскрашен соответсвенно требованиям задачи. Таким образом, решение задачи раскраски планарного графа с помощью метода сводимых конфигураций эквивалентно нахождению для него неустранимого множества сводимых конфигураций. Для доказательства того, что конфигурация сводима, зачастую пытаются показать, что любая раскраска некоторого подграфа может быть распространена на весь граф.

Конфигурации в неустранимом множестве могут являться сводимыми для определенной задачи раскраски или могут описывать некоторые интересные структурные свойства планарных графов. Распространенный метод доказательства неустранимости множества конфигураций, называемый разрядкой, работает следующим образом.

Предполагая существование планарного графа, не содержащего ни одной конфигурации из S, мы связываем с каждой вершиной v действительное число, которое мы называем начальным зарядом. Иногда начальный заряд присваивается также и граням. Начальные заряды определены таким образом, что их сумма отрицательна. Очевидно, что некоторые элементы имеют

отрицательный начальный заряд (являются дефицитными). Используя отсутствие в графе конфигураций, принадлежащих S, мы пытаемся перераспределить заряды (сохранив их сумму) в пользу дефицитных элементов, забирая часть заряда у элементов, имеющих положительный начальный заряд. Если нам удастся сделать новые заряды всех элементов неотрицательными, то возникает очевидное противоречие, которое завершает доказательство.

Эта простая схема на самом деле является примером обычного доказательства от противного, и она широко применяется в исследованиях графов. Хотя это широко известный метод, его применение не всегда приводит к успеху; многие известные проблемы в этой области оставались открытыми в течение многих лет. Более того, на самом деле каждая разрядка, какой бы искусственной она ни была, все равно создает множество неустранимых конфигураций, состоящее из всех тех конфигураций, которые являются дефицитными.

Доказательство непосредственно теоремы о четырех красках было получено только в 1976 году и оно строится по следующему принципу. Первоначально вводится понятие *минимального контрпримера*. Под ним понимают такой планарный граф G, что $\chi(G) \geq 5$ и любой подграф G является 4-раскрашиваемым. Для доказательства важны те факты, что любой минимальный контрпример является триангуляцией, 5-связен, а также 6-связен за исключением вершин степени 5. Далее для доказательства рассматриваются неустранимые конфигурации. Для любого плоского графа найдется подграф, изоморфный некторой конфигурации, но в то же время доказывается, что никакой минимальный контрпример к теореме не может содержать ни одной такой конфигурации. Из этого делается вывод, что контрпримера не существует[1]. Мощность множества таких конфигураций превышает 1400, поэтому доказательство проведено с помощью компьютера. В последствии доказательство было улучшено, число конфигураций было уменьшено до менее 700, однако некоторые его части все еще нельзя осуществить без помощи компьютера.

Не все ученые согласны с таким доказательством, так как в программе могла быть ошибка, к тому же сведение графов к неустранимым конфигурациям с их последующей раскраской очень сложно повторить. Тем не менее, придумать доказательство, не задействующее ЭВМ, пока не удалось. Во многом это вызвано именно большим количеством неустранимых конфигураций, рассматривать которые вручную не представляется возможным.

5-раскраска

Если нет задачи использовать в раскраске наименьшее возможное число цветов, можно применить алгоритм раскраски произвольного планарного графа в 5 цветов, работающий за линейное время[16].

Алгоритм также основан на последовательном уменьшении размерности рассматриваемого графа G. Рассматриваются два случая:

–в графе G есть вершина v степени не больше 4. Тогда строится 5-раскраска графа $G - v$, а вершине v присваивается цвет, который не встречается в ее окрестности.

–все вершины G имеют степень не меньше 5. В таком случае мы можем утверждать, что среди них есть по крайней мере две несмежные вершины u и w, ведь иначе окрестность v индуцирует K_5, что противоречит планарности G. Тогда можем получить раскраску графа H, полученного из G стягиванием вершин u, w и v. Для получения раскраски исходного графа раскрасим u и w в цвет, который получила объединенная вершина, а v – в цвет, не встречающийся в ее окрестности.

ГЛАВА 2. ТОЧНЫЕ АЛГОРИТМЫ

2.1 Размер пространства решений

Самым простым алгоритмом, который возвращает точное решение задачи, является перебор всех возможных раскрасок графа. Для определения целесообразности его использования подсчитаем размер пространства решений.

Пусть имеется произвольный граф на n вершинах. Ни одна раскраска не потребует более чем n цветов, поэтому оценим количество цветов в оптимальной раскраске числом вершин графа. Тогда каждой из n вершин можем поставить в соотвествие n цветов, итого необходимо будет перебрать n^n различных раскрасок. Это значение растёт невероятно быстро, что делает невозможным практическое использование алгоритма. На деле, даже если мы знаем, что цветов понадобится не более какого-то k, пространство решений будет иметь размер k^n, что всё ещё очень велико для $k > 1$.

Немного улучшить результат можно, избавившись от различных перестановок одной раскраски в пространстве решений. Каждую раскраску мы считали $k!$ раз, но даже так полученное число не позволяет использовать перебор на практике.

Однако это не значит, что нет возможности получить точный ответ на задачу. Существуют алгоритмы, которые оптимизируют исследование пространства решений, и которые могут давать оптимальную раскраску для небольших графов за разумное время.

2.2 Поиск с возвратом

На случай, если необходимо точное решение, пусть и с большими вычислительными затратами, существуют алгоритмы, вычисляющие точный ответ. Одним из них является поиск с возвратом.

В общем случае поиск с возвратом – название не конкретного алгоритма, а подхода к решению вычислительных задач, который заключается в попытках построить наилучшее решение из частичного. Если же частичное решение не может привести к оптимальному, происходит возврат и попытка улучшить частичное решение.

В случае задачи раскраски графа поиск с возвратом работает следующим образом. Пусть дан граф $G(V, E)$, где v_i $(1 \leq i \leq n)$ – некоторым образом упорядоченные вершины и k – число доступных цветов. Изначально можно положить $k = \infty$. Алгоритм производит серию прямых и обратных шагов. Прямые шаги окрашивают вершины по очереди, пока это возможно сделать с помощью k цветов. Обратные же шаги проходят по уже окрашенным вершинам

и определяют, можно ли было окрасить их по другому. Если так, прямые шаги снова начинаются с найденной вершины. Если найдена раскраска на k цветах, значение k уменьшается на 1 и выполнение алгоритма продолжается. Выполнение завершается, когда алгоритм достигает вершины v_1.

Из-за того, что задача NP-полная, время выполнения алгоритма может оказаться слишком большим, однако поиск с возвратом куда эффективнее обычного перебора всех возможных вариантов раскраски графа. Также выявлены эвристики, которые могут улучшить алгоритм. Например польские математики Кубале и Яковский исследовали такие эвристики как: расстановка вершин по убыванию степени; расстановка вершин так, чтобы первыми окрашивались вершины с наименьшим числом доступных цветов; попытка изначально отнести вершину к более многочисленным цветовым классам и др.[13]

Также стоит отметить, что если прервать выполнение алгоритма досрочно, то получится допустимая, хоть и не оптимальная раскраска. Таким образом поиск с возвратом можно использовать и для получения приближённого ответа.

2.3 Целочисленное программирование

Целочисленное программирование, особый случай линейного программирования, является другим способом получить точное решение задачи. Конкретные методы целочисленного программирования могут быть разными, мы же рассмотрим, как представить задачу раскраски графа в подходящем для их использования виде.

Положим, как и раньше, что граф $G(V, E)$ имеет n вершин и m рёбер. Тогда простейшая формулировка использует бинарные матрицы $X_{n\times n}$ и Y_n, которые задаются следующим образом:

$$X_{ij} = \begin{cases} 1, & \text{если вершина } v_i \text{ имеет цвет } j \\ 0, & \text{в противном случае} \end{cases} \tag{2.1}$$

$$Y_j = \begin{cases} 1, & \text{если хоть одна вершина имеет цвет } j \\ 0, & \text{в противном случае} \end{cases} \tag{2.2}$$

Целевая функция будет иметь вид:

$$min \sum_{j=1}^{n} Y_j \tag{2.3}$$

С ограничениями:

$$X_{ij} + X_{lj} \leq Y_j \quad \forall \{v_i, v_l\} \in E, \forall j \in \{1, \dots, n\}$$
$$\sum_{j=1}^{n} X_{ij} = 1 \quad \forall v_i \in V \tag{2.4}$$

У такого подхода есть существенный недостаток – перестановки цветов считаются разными ответами, хотя по сути они задают одну раскраску. Это значит, что каждую уникальную раскраску алгоритм просмотрит A_n^k раз, просто за счёт перестановок столбцов X. Для борьбы с этим можно использовать дополнительные ограничения, которые будут ограничивать количество рассматриваемых перестановок одной раскраски, например:

$$Y_j \geq Y_{j+1} \quad \forall j \in \{1, \dots, n-1\} \tag{2.5}$$

Однако и такое ограничение не решает проблему полностью, ведь мы все ещё рассматриваем каждую перестановку $k!$ раз. В таком случае можно использовать ограничения, которые представили Мендез-Диаз и Забала[17], и которые имеют вид:

$$X_{ij} = 0 \quad \forall v_i \in V, j \in \{i+1, \dots, n\}$$
$$X_{ij} \leq \sum_{l=j-1}^{i-1} X_{lj-1} \quad \forall v_i \in V - \{v_1\}, \forall j \in \{2, \dots, i-1\} \tag{2.6}$$

Такие огранчения определяют каждую раскраску уникальным образом, благодаря чему время выполнения алгоритма сильно улучшается, хотя и остаётся слишком велико для задач большой размерности.

ГЛАВА 3. ПРИБЛИЖЕННЫЕ АЛГОРИТМЫ

На практике точные алгоритмы не могут использоваться для решения задач большой размерности из-за слишком большого числа вычислений, поэтому в большинстве случаев применяются алгоритмы, которые дают приближенный ответ, но за полиномиальное время. Зачастую полученное решение обладает точностью, достаточной для решения конкретной практической задачи. Начнём их рассмотрение с конструктивных алгоритмов.

3.1 Жадный алгоритм

Жадный алгоритм является одним из самых простых, но в то же время важных алгоритмов раскраски. Алгоритм просматривает вершины по одной в определённом порядке и присваивает каждой наименьшую возможную метку. Полученное решение не всегда является наилучшим, однако далее мы покажем, что для любого графа существует порядок обработки вершин, при котором жадный алгоритм даст оптимальную раскраску. Псевдокод жадного алгоритма можно записать так:

for i from 1 to $|t|$:
  for j from 1 to $|S|$:
    if ($S_j \cup \{t_i\}$) - независимое:
      $S_j = S_j \cup \{t_i\}$
      break
    else $j = j + 1$
  if $j > |S|$:
    $S_j = \{t_i\}$
    $S = S \cup S_j$

где t – перестановка цветов, а S – изначально пустое множество, где и хранится раскраска.

Оценим сложность алгоритма. На i-той итерации для вершины перебирается не более, чем i цветов, а так как число итераций равно числу вершин в графе, временная сложность алгоритма равна $O(n^2)$.

На практике жадный алгоритм работает достаточно быстро, однако для некоторых графов его точность оставляет желать лучшего. Рассмотрим например двудольный граф $G = (V_1, V_2, E)$, где $V_1 = \{v_1, v_3, \dots, v_{n-1}\}$, $V_2 = \{v_2, v_4, \dots, v_n\}$ и $E = \left\{ \{v_i, v_j\} \mid v_i \in V_1, v_j \in V_2, i + 1 \neq j \right\}$. Если алгоритм будет рассматривать все вершины по порядку, то в результате получится раскраска из $\frac{n}{2}$ цветов. В то же время, если в алгоритме рассмотреть сначала все вершины одной доли, а затем другой, получится раскраска из 2 цветов. Таким образом выбор последовательности вершин может очень сильно влиять на результат.

Важным свойством жадного алгоритма является то, что он может улучшать уже полученные допустимые раскраски. Действительно, пусть существует допустимая раскраска S и $S_i \in S$ $(1 \leq i \leq |S|)$ – цветовые классы. Рассматривая по очереди элементы разных классов, получим допустимую раскраску S' такую, что $|S'| \leq |S|$. Пусть это неверно, и какой-то вершине $v \in S_i$ алгоритм присвоит метку $j > i$. Рассмотрим первую вершину, с которой это случилось. Это значит, что среди соседей v нашлась вершина с меткой i, но в то же время все вершины с метками меньше i в S имеют метки меньше i и в S', а $S'_i \subset S_i$, и из независимости S_i следует независимость S'_i. Пришли к противоречию, а значит исходное утверждение верно.

Отсюда следует ещё один важный факт: для любого графа существует последовательность вершин, на которой жадный алгоритм даёт оптимальную раскраску. Это следует из предыдущего утверждения, если взять в качестве S оптимальную раскраску.

3.2 Некоторые оценки на хроматическое число

Пусть $\omega(G)$ – количество вершин в наибольшей клике графа G. Тогда $\chi(G) \geq \omega(G)$. С другой стороны, пусть $\alpha(G)$ – количество вершин в наибольшем независимом множестве графа G. Тогда $\chi(G) \geq \left\lceil \frac{n}{\alpha(G)} \right\rceil$, ведь если эта оценка не выполняется, то в графе G существует незавимое множество более, чем из $\alpha(G)$ вершин. Эти две оценки можно объединить одним выражением:

$$\chi(G) \geq \max\left(\omega(G), \left\lceil \frac{n}{\alpha(G)} \right\rceil\right) \tag{3.1}$$

Точность этих оценок варьируется для разных графов, но это не единственный их недостаток. И задача поиска наибольшей клики, и задача поиска наибольшего независимого множества являются NP-полными, что сильно ограничивает разнообразие графов, для которых применение этих оценок имеет смысл. Тем не менее, в некоторых случаях они могут быть полезны.

От оценок снизу перейдём к оценкам сверху. Пусть $\Delta(G)$ – максимальная степень вершины в G. Тогда $\chi(G) \leq \Delta(G) + 1$, так как жадный алгоритм ставит в соотвествие каждой вершине наименьшую метку, которая не встречается среди её соседей, которых в свою очередь не больше $\Delta(G)$. Эту оценку можно сделать более строгой, так в 1941 Роланд Брукс показал, что если граф G не является полным и не является циклом нечётной длины, то $\chi(G) \leq \Delta(G)$[4].

Предположим, что в любом подграфе G' графа G существует вершина со степенью не больше δ. Тогда $\chi(G) \leq \delta + 1$, т.к. на каждой итерации жадного алгоритма в подграфе нераскрашенных вершин найдётся вершина со степенью

не больше δ, которой будет присвоена метка не более $\delta + 1$. Тогда в результате получится допустимая раскраска из не более чем $\delta + 1$ цветов.

3.3 Алгоритм DSatur

Алгоритм DSatur по своей сути он очень похож на жадный алгоритм[3]. Отличаются они лишь в том, что DSatur динамически выбирает последовательность обработки вершин на основе уже полученной частичной раскраски. Записать его можно так:

```
while X ≠ ∅:
    выберем v ∈ X
    for j from 1 to |S|:
        if (S_j ∪ {v}) - независимое:
            S_j = S_j ∪ {v}
            break
        else j = j + 1
    if j > |S|:
        S_j = v
        S = S ∪ S_j
    X = X − {v}
```

где X – множество вершин графа, а S – изначально пусто и используется для хранения раскраски.

Пусть $c(v) = -1$ для всех вершин $v \in V$, которым ещё не присвоен цвет. Тогда для вершины v *насыщенностью* будем называть количество различных цветов, присвоенных её соседям, и обозначать это будем как $sat(v)$. Формально, $sat(v) = |\{ c(u) \mid u \in N(v), c(u) \neq -1 \}|$.

Суть алгоритма DSatur заключается в том, что вместо обработки всех вершин в каком-то заранее определённом порядке, на каждой итерации обрабатывается вершина с наибольшей насыщенностью. Если таких вершин несколько, выбирается та, которая имеет большую степень. Считается, что благодаря тому, что на каждой итерации мы обрабатываем вершину, для которой существует меньше всего доступных вариантов цвета, результат получается не хуже, чем при использовании жадного алгоритма.

Благодаря динамическому определению последовательности вершин у DSatur более предсказуемое поведение. Также для двудольных графов, циклов и колёс алгоритм всегда даёт оптимальную раскраску.

Сложность DSatur такая же, как и у жадного алгоритма – $O(n^2)$, хотя некоторые дополнительные расходы необходимы для отслеживания насыщенности вершин.

3.4 Алгоритм RLF (Recursive Largest First)

Алгоритм RLF был представлен в 1979 году[14], и он отличается от рассмотренных выше методов тем, что он просматривает не вершины, а цвета. На каждой итерации алгоритм определяет независимое множество и относит его к одному цвету. Далее это множество удаляется из графа и процесс продолжается на полученном меньшем подграфе. Записать псевдокод RLF можно так:

```
while X ≠ ∅:
    i = i + 1
    S_i = ∅
    while X ≠ ∅:
        выберем v ∈ X
        S_i = S_i ∪ {v}
        Y = Y ∪ N_X(v)
        X = X − (Y ∪ {v})
    S = S ∪ S_i
    X = Y
    Y = ∅
```

где X – множество вершин графа, i – рассматриваемый цвет, S и Y – изначально пустые множества, $N_X(v)$ – множество вершин графа X, смежных с v.

Для выбора вершины Лейтон использовал правила, аналогичные применяемым в алгоритме DSatur: на каждой итерации выбирается вершина, для которой доступно наименьшее число цветов. Как и в предыдущем алгоритме, в случае наличия нескольких таких вершин выбирается случайная.

Сложность алгоритма RLF – $O(n^3)$, что хуже, чем у уже рассмотренных алгоритмов, однако во многих случаях это компенсируется лучшей точностью. Также как и DSatur, алгоритм RLF даёт оптимальные раскраски для двудольных графов, циклов и колёс.

3.5 Эвристики, используемые в приближенных алгоритмах

На практике в большинстве случаев используются более сложные приближенные алгоритмы, основанные на эвристиках, благодаря которым они более эффективно исследуют пространство решений. Далее рассмотрим некоторые из них, но сначала разделим алгоритмы по рассматриваемым пространствам решений.

Первыми рассмотрим алгоритмы, которые исследуют пространства только из допустимых раскрасок. Такие алгоритмы пытаются уменьшить используемое

в раскраске число цветов и зачастую в их основе лежит жадный алгоритм. К числу таких алгоритмов относятся, например итерационный жадный алгоритм или некоторые генетические алгоритмы.

Следующими рассмотрим алгоритмы, которые работают с полными k-раскрасками, на их допустимость при этом никаких ограничений не накладывается. Работают они следующим образом: в пространстве решений ищем ту раскраску, которая окажется допустимой. В случае успеха повторяем для меньшего k, в случае неудачи признаем ответом последнее найденное решение. К числу подобных алгоритмов относятся, например, алгоритм муравьиной колонии и современные генетический алгоритмы.

Последним типом алгоритмов, который мы рассмотрим, являются алгоритмы, которые работают в пространстве неполных допустимых k-раскрасок. Подход к поиску решения похож на предыдущий: в пространстве решений ищем полную раскраску. В случае успеха повторяем для меньшего k, в случае неудачи признаем ответом последнее найденное решение. Такие алгоритмы изучены меньше всего, однако и среди них есть довольно эффективные, например алгоритм частичной раскраски, который будет рассмотрен в главе 3.10.

3.6 Локальный поиск

Перед рассмотрением конкретных алгоритмов имеет смысл рассмотреть процедуру локального поиска, которая позволяет улучшать имеющиеся решения с помощью исследования пространства решений особым образом. Хотя локальный поиск – это не какой-то алгоритм, а подход к решению задач, в работе будем понимать под локальным поиском конкретную реализацию, представленную в работе Галиньера и Хао[8]. Данный алгоритм может использоваться как самостоятельно, так и в качестве одного из этапов какого-то более сложного алгоритма раскраски. Суть его заключается в том, чтобы с использованием целевой функции определить, как наиболее эффективно исследовать пространство решений.

Алгоритм локального поиска работает в пространстве полных, не обязательно допустимых раскрасок с использованием следующей целевой функции:

$$f(S) = \sum_{\forall\{u,v\}\in E} g(u,v) \tag{3.2}$$

Где

$$g(u,v) = \begin{cases} 1, \text{если } u \text{ и } v \text{ одного цвета} \\ 0 \text{ иначе} \end{cases} \quad (3.3)$$

По сути своей f – количество ребер, по которым нарушается допустимость раскраски, т.е. если $f(S) = 0$, то S – допустимое решение. В дальнейшем под стоимостью решения будем понимать значение целевой функции на нем.

Для заданного решения $S = \{S_1, \dots, S_k\}$ переходы в пространстве решений выполняются путем выбора вершины $v \in S_i$, текущее назначение которой в цветовой класс S_i вызывает конфликт, и последующего переноса её в другой цветовой класс $S_j \neq S_i$. Следует отметить, что в ранних версиях этого алгоритма допускалось перемещение вершин, не участвующих в конфликтах, однако такой вариант в большинстве случаев ухудшает производительность[9].

Для оптимизации исследования пространства решений в алгоритме используются запреты на перемещение в уже исследованные его участки. Список запретов в алгоритме хранится в матрице $T_{\mathrm{n\times k}}$. Если на итерации l происходит перемещение вершины v из класса S_{i} в S_{j}, то элемент T_{vi} устанавливается равным $l + t$, где t — положительное целое число, которое мы определим далее. Это означает, что возврат вершины v в цветовой класс S_{i} запрещён на протяжении следующих t итераций (то есть v не может быть возвращена в S_{i} до итерации $l + t$). Из-за этого вообще все решения, содержащие назначение вершины v в класс S_{i}, становятся запретными на t итераций.

Как это обычно принято в методах локального поиска, на каждой итерации рассматривается всё множество соседних решений, то есть вычисляется стоимость перемещения каждой конфликтующей вершины во все остальные $k - 1$ цветовых классов. Именно этот процесс занимает большую часть времени работы алгоритма, однако его можно значительно ускорить за счёт использования эффективных структур данных.

Вместо пересчета целевой функции для всех соседних решений на каждой итерации можно произвести предварительный подсчет с помощью матрицы $C_{n\times k}$, где C_{vj} – количество вершин, принадлежащих S_j в окрестности вершины v. Таким образом мы можем вычислить целевую функцию решения, полученного в результате перемещения, за константное время. При этом после перемещения в матрице C нужно лишь изменить на 1 значения для вершин из окрестности v.

После оценки всех соседних решений алгоритм локального поиска выбирает и выполняет допустимый переход, обеспечивающий наибольшее снижение значения целевой функции (или, при отсутствии улучшений, наименьшее его увеличение). В случае равенства значений выбор осуществляется случайным образом.

Кроме того, в определённых ситуациях алгоритм может выполнять запрещённые переходы. В частности, такие переходы разрешаются, если они приводят к улучшению наилучшего найденного на данный момент решения. Это особенно полезно, если запрещённый переход позволяет достичь решения с нулевым значением функции (т.е. отсутствием конфликтов), после чего алгоритм может завершить работу. Если же все возможные переходы оказываются запрещёнными, алгоритм выбирает случайную вершину $v \in V$ и перемещает её в случайно выбранный цветовой класс. После этого список запретов обновляется в соответствии с установленными правилами.

Вернемся к параметру t. Наиболее эффективным считается использование случайной величины t, значение которой пропорционально стоимости текущего решения. Основная идея заключается в том, что при низком качестве текущего решения (при его высокой стоимости) величина t принимает большие значения, что способствует переходу алгоритма в другие области пространства решений в надежде на нахождение более качественных решений.

С другой стороны, если текущее решение обладает низкой стоимостью, алгоритм фокусируется на локальном исследовании данной области за счёт малых значений t. В частности, авторы алгоритма рекомендуют вычислять t по формуле

$$t = 0.6 \cdot f_2 + r \tag{3.4}$$

где r – произвольное число из промежутка $[0, 9]$.

3.7 Генетические алгоритмы

Генетические алгоритмы – это тип алгоритмов, вдохновлённых биологической эволюцией и оперирующих так называемой популяцией, которая представляет собой небольшое подмножество пространства решений. Во время выполнения алгоритм пытается улучшить качество решений в популяции, используя следующие операторы:

– Рекомбинация. Данный оператор создаёт новое решение, комбинируя части решений из популяции. Зачастую старые и новые решения называют предками и потомками соответсвенно.

– Мутация. Мутация вносит изменения в решения из популяции, чтобы исследовать новые области пространства решений. Изменения могут быть как случайными, так и обоснованными каким-либо алгоритмом.

Выбор решений, используемых операторами, происходит наподобие естественного отбора. Алгоритм стремится оставлять в популяции хорошие решения, избавляясь от плохих. Так, хорошие решения с большей вероятностью

участвуют в рекомбинации, а плохие с большей вероятностью замещаются новым решением. Для оценки качества решений могут использоваться различные функции, выбор которых зависит от конкретного рассматриваемого алгоритма.

Условно генетические алгоритмы можно разделить на две группы по тому, какие решения составляют популяции. Первые алгоритмы работают только с допустимыми раскрасками. Это значит, что все решения, составляющие начальную популяцию и получаемые в результате применения операторов, являются допустимыми. Вторая группа алгоритмов не требует, чтобы раскраски были правильными на каждом этапе их работы.

Так как схема решения у всех генетических алгоритмов одна, различаются они только реализацией операторов и видом функции оценки качества решений. В то же время это открывает большое пространство для исследований, ведь комбинирование различных операторов дает разные алгоритмы. Также оператор может быть реализован с использованием другого алгоритма или его части, в таком случае генетический алгоритм называют гибридным.

Алгоритм Эрбена

Алгоритм Эрбена – один из генетических алгоритмов, работающих только с допустимыми решениями[7]. Рассмотрим используемые в нем операторы.

Рекомбинация в алгоритме Эрбена проводится следующим образом. Сначала из популяции выбираются решения S_1 и S_2. Из S_2 случайным образом выбирают подмножество цветовых классов и добавляют их к копии S_1, чтобы получить потомка S'. На данном этапе некоторые вершины встречаются в S' дважды, поэтому алгоритм удаляет все цветовые классы, унаследованные от S_1, в которых содержатся дублируемые вершины. После этого с большой вероятностью S' будет неполной, поэтому она дополняется жадным алгоритмом.

Этот оператор обладает важным для генетического алгоритма свойством – если цветовой класс S_i присутствует в обоих предках, то его будет содержать и потомок.

Мутация работает аналогично: из решения удаляются случайно выбранные цветовые классы, удалённые вершины перемешиваются и добавляются обратно жадным алгоритмом.

Для оценки качества решений в алгоритме применяется следующая функция:

$$f(S) = \frac{\sum_{S_i \in S} \left(\sum_{v \in S_i} \deg(v) \right)^2}{|S|} \tag{3.5}$$

Где выражение $\sum_{v \in S_i} \deg(v)$ равно сумме степеней всех вершин, принадлежащих одному цветовому классу. Раскраска из популяции даёт

большее значение функции, если она содержит большие цветовые классы в малом количестве. Именно на основе значения f происходит выбор решений для скрещивания и удаления из популяции.

Алгоритм Мамфорд

Алгоритм Мамфорд использует тот же оператор мутации и функцию оценки решений, что и алгоритм Эрбена, но оператора скрещивания два – слияния независимых множеств и перестановки относительно точки[18].

Первый из операторов скрещивания работает на основе цветовых классов. Для начала вершины в каждом родителе упорядочиваются по своему цветовому классу, а затем две полученные последовательности случайным образом сливаются в одну, где каждая вершина встречается дважды. Далее для каждого ребенка строится своя последовательность вершин – при первое вхождение каждой вершины относят к первому потомку, а второе – к оставшемуся. Последним шагом для каждого потомка по полученной последовательности вершин строится раскраска с помощью жадного алгоритма.

Второй оператор по принципу работы очень похож на первый. Как и в предыдущем случае, перестановки вершин предков формируются по их цветовым классам. Затем случайным образом выбирается точка разделения последовательностей и части, которые находятся после этой точки, меняются местами. По полученным последовательностям с помощью жадного алгоритма строятся решения, которые и являются потомками.

В целом подобные алгоритмы считаются не очень эффективными, ведь требование к допустимости решений накладывает ограничения на возможность эффективно исследовать пространство решений.

Гибридный алгоритм

Рассмотрим теперь алгоритм, который не требует допустимости решений. Гибридный алгоритм работает с полными k-раскрасками, причем не обязательно правильными[8]. Гибридным он является, потому что для улучшения решений используется алгоритм локального поиска. Рассмотрим его подробнее.

Для начала необходимо построить начальную популяцию. Для этого используется немного модифицированная версия алгоритма DSatur. Отличается от оригинального алгоритма он тем, что строит раскраску ровно в k цветов. Построение каждого решения начинается со случайной вершины, чтобы обеспечить разнообразие решений в популяции. В ходе выполнения алгоритма размер популяции остается неизменным, и хотя он может варьироваться в зависимости от конкретной задачи, в общем случае оптимальной считается популяция из десяти решений. Когда популяция построена, производится попытка улучшить каждое решение с помощью локального поиска.

Рассмотрим операторы, которые применяются в этом алгоритме. Скрещивание производится следующим образом. Первоначально из популяции

выбираем случайным образом два решения. Из их цветовых классов выбираем тот, который имеет наибольшую мощность и переносим его в потомка. Так как каждая вершина фигурирует в двух цветовых классах – по одному из каждого предка, нужно удалить вторые вхождения вершин, которые уже попали в потомка. Затем эта операция повторяется поочередно для каждого родителя, пока количество цветовых классов в потомке не достигнет k. После этого некоторые вершины в потомке могут не относиться ни к одному классу, в таком случае цвет присваивается им случайным образом. Полученное в результате решение заменяет в популяции худшего из предков, при этом допускается ситуация, в которой потомок может получиться хуже их обоих. После попадания в популяцию пробуем улучшить решение с помощью локального поиска.

Оператор мутации в явном виде в алгоритме не используется, его роль берет на себя локальный поиск, который применияется к каждому решению перед их попаданием в популяцию.

Для нахождения хроматического числа графа данный алгоритм применяется многократно, каждый раз для меньшего k. Прекращается его выполнение в том случае, когда за большое число поколений не получается улучшить значение f. Тогда хроматическим числом графа признается последнее k, для которого была найдена допустимая раскраска.

3.8 Итерационный жадный алгоритм

Итерационный жадный алгоритм был представлен в 1996 году Кулберсоном и Луо[6]. Он основан на возможности жадного алгоритма улучшать уже имеющиеся раскраски.

Сам алгоритм работает следующим образом. Для начала с помощью DSatur строится начальное решение. Затем на каждой итерации алгоритма составляется перестановка цветовых классов имеющейся раскраски и на ней запускается жадный алгоритм, который возвращает новую допустимую раскраску.

Авторы алгоритма предлагают несколько способов построения перестановки:

– перестановка по убыванию размера классов;

– перестановка цветовых классов в обратном порядке;

– перестановка цветовых классов слуайным образом.

Первый способ пытается создать большие независимые множества, второй перемешивает вершины между цветовыми классами, а третий не допускает возникновения циклов. Авторы алгоритма рекомендуют использовать на каждой итерации одно из этих правил, выбранное случайным образом с отношением вероятностей 5:5:3 соответственно.

3.9 Алгоритм муравьиной колонии

Алгоритм муравьиной колонии (АМК) – это алгоритмический подход, изначально вдохновлённый тем, как настоящие муравьи находят оптимальные пути между источниками пищи и своей колонией.

В естественной среде, когда источник пищи ещё не обнаружен, муравьи перемещаются хаотично. Однако, как только пища найдена, муравьи-первооткрыватели переносят её часть в колонию, оставляя за собой феромонный след. Другие муравьи, обнаружив этот след, с меньшей вероятностью продолжают блуждать случайным образом и вместо этого могут последовать по нему. Если они также находят тот же источник пищи, то возвращаются по феромонному следу в гнездо, дополнительно усиливая его своими собственными феромонами. Это стимулирует ещё больше муравьёв следовать по данному пути.

Кроме того, со временем феромоны испаряются, уменьшая вероятность следования по старому маршруту. Чем дольше муравей перемещается по пути, тем больше времени остаётся для испарения феромонов. Таким образом, на более коротких маршрутах концентрация феромонов растёт быстрее, что увеличивает вероятность их выбора другими муравьями и дальнейшего усиления следа. В результате такой обратной связи все муравьи в конечном итоге начинают следовать по единственному оптимальному пути между колонией и источником пищи.

Первоначально АМК применялся для решения таких задач, как задача коммивояжёра и маршрутизация транспортных средств, где требуется найти оптимальный путь обхода вершин графа, однако впоследствии метод был адаптирован и для многих других задач.

Идея подобного алгоритма для раскраски графа заключается в использовании муравьёв для генерации отдельных решений[19]. В процессе работы каждый муравей строит своё решение недетерминированным образом, используя вероятности, основанные на эвристиках, а также на качестве решений, полученных предыдущими муравьями. В частности, если предыдущие муравьи выявили особенности, которые приводят к решениям лучше среднего, текущий муравей с большей вероятностью включит эти особенности в своё решение. Это, как правило, приводит к уменьшению количества цветов по ходу выполнения алгоритма.

На каждой итерации алгоритма каждый муравей создаёт полное, хотя и не обязательно допустимое, решение. Затем детали каждого из этих решений используются вместе с «коэффициентом испарения» для обновления матрицы следов.

В начале каждой итерации текущий муравей пытается построить решение с помощью процедуры, основанной на алгоритме RLF, который, напомним,

работает путём последовательного построения каждого цветового класса в решении. Также стоит отметить, что при построении каждого класса Si ∈ S метод RLF использует два множества: X – содержит неокрашенные вершины, которые на текущем шаге могут быть добавлены в текущий цветовой класс без конфликта; Y – содержит неокрашенные вершины, которые не могут быть корректно добавлены в текущий цветовой класс.

Модификации используемого алгоритма RLF заключаются в следующем:

– В процедуре допускается использование не более чем k цветовых классов. После их построения оставшиеся вершины остаются неокрашенными;

– Первая вершина, назначаемая каждому классу S_i $(1 \leq i \leq k)$, выбирается случайным образом из множества X;

– В остальных случаях каждая вершина v назначается цветовому классу S_i с вероятностью, зависящей от следов, оставленных предыдущими муравьями.

Так как процесс построения решения, описанный выше, не является детерминированным, производится несколько попыток создания каждого цветового класса, из которых в последствии выбирается лучший.

В результате получаем правильную раскраску, которая может оказаться неполной. Если это так, всем нераскрашенным вершинам присваивается произвольный цвет так, что получается полная, не обязательно правильная раскраска, которую в последствии пытаемся улучшить с помощью локального поиска.

3.10 Алгоритм частичной раскраски

Алгоритм частичной раскраски, предложенный в 2008 году[2], работает аналогично алгоритму локального поиска, рассмотренному ранее, используя эвристику поиска с запретами для нахождения допустимой $k-$раскраски. Однако, в отличие от локального поиска, алгоритм частичной раскраски не рассматривает недопустимые решения – вместо этого вершины, которые не могут быть назначены ни одному из k цветов без возникновения конфликтов, помещаются в множество нераскрашенных вершин U. Цель алгоритма состоит в модификации частичного решения S таким образом, чтобы множества U стало пустым, что соответствует нахождению допустимой k-раскраски.

Из-за того, что алгоритм работает в другом пространстве решений, перемещение между решениями в рассматриваемом алгоритме отличается от используемого в локальном поиске. Осуществляется он следующим образом:

1 Выбирается нераскрашенная вершина $v \in U$;

2 Вершина v назначается цветовому классу $S_j \in S$;

3 Все вершины $u \in S_j$, смежные с v, переносятся из S_j в U.

После выполнения такого перехода соответствующие элементы T_{uj} в списке запретов помечаются как запрещённые на следующие t итераций алгоритма.

На каждой итерации алгоритма частичной раскраски рассматривается полное множество соседних решений размером $|U| \times k$. Выбор перехода осуществляется по тем же критериям, что и в алгоритме локального поиска. Аналогично, для ускорения вычислений может использоваться предварительно подсчитанная матрица C.

Начальное решение генерируется жадным алгоритмом, который может использовать не более k цветов. При этом вершины без допустимых цветов сразу помещаются в U. Основное отличие между алгоритмами заключается в способе расчета длительности запрета t. В оригинальной работе используется модификация алгоритма, где:

– При отсутствии изменений целевой функции в течение длительного времени t увеличивается (для выхода из области локального оптимума);

– При колебаниях целевой функции t постепенно уменьшается.

Однако при необходимости могут применяться и более простые схемы расчета t.

ГЛАВА 4. ГРАФЫ ДЛЯ СРАВНЕНИЯ

В этой главе рассмотрим графы, на которых будет проводиться сравнение описанных выше алгоритмов. Для этого необходимо ввести понятие случайного графа.

Случайный граф – граф на n вершинах, где между любыми двумя вершинами существует ребро с вероятностью p. Обозначается он как $G_{n,p}$. Стоит заметить, что параметр n будет не так сильно влиять на предпочтительность того или другого алгоритма, так что не имеет смысла рассматривать большое количество различных его значений. В то же время значение параметра p влияет на количество ребер, что влияет на количество ограничений на цвета и, соответственно, на поведение алгоритмов. В работе рассмотрим графы со значениями $n \in \{200, 1000\}$ и $p \in \{0.1, 0.2, \dots, 0.9\}$.

В сравнении на случайных графах есть существенная проблема – отсутствие возможности установить их истинное хроматическое число. Чтобы решить ее, введем отдельный вид графа, который назовем равномерным. Для его получения распределим все вершины произвольного графа $G = (V, E)$ на s независимых множеств мощности $\lfloor n/s \rfloor$ или $\lceil n/s \rceil$. Затем между вершинами из разных множеств будем добавлять ребро с вероятностью p так, чтобы степени вершин отличались как можно меньше. Каждый такой граф будет s −раскрашиваемым, что позволит установить эффективность алгоритмов раскраски при разных значениях s и n.

В заключении рассмотрим, как алгоритмы раскраски показывают себя на графах, которые возникали в реальных практических задачах.

ГЛАВА 5. АНАЛИЗ ЭФФЕКТИВНОСТИ

Сравнение алгоритмов разделим на две части: сравнение на случайных графах и на больших графах с известным хроматическим числом, возникавших в прикладных задачах. Сравнение будем проводить для гибридного генетического алгоритма, алгоритма муравьиной колонии и алгоритма частичной раскраски.

Рассмотрим результаты применения алгоритмов раскраски к различным случайным графам. Для каждого вида графа генерировалось по 10 экземпляров, на которых каждый алгоритм запускался 5 раз. Результаты применения алгоритмов к случайным графам изображены в таблице 4.1.

Таблица 4.1 – Результаты применения алгоритмов к случайным графам

		Алгоритм муравьиной колонии		**Гибридный генетический алгоритм**		**Алгоритм частичной раскраски**	
n	p	χ	t, мс.	χ	t, мс.	χ	t, с.
200	0.1	7	104	7	43	7	34
	0.2	11	804	11	294	11	323
	0.3	15	892	15	264	15	289
	0.4	20	1432	20	226	20	82
	0.5	25	589	25	131	25	125
	0.6	30	551	30	388	30	277
	0.7	37	1082	37	418	37	465
	0.8	46	715	46	551	46	321
	0.9	64	698	64	636	63	307
1000	0.1	23	425	22	152	22	158
	0.2	40	981	39	300	39	302
	0.3	58	826	57	498	57	488
	0.4	76	1260	76	723	76	749
	0.5	95	995	97	1133	97	987
	0.6	119	1266	124	1181	124	981
	0.7	147	1789	157	1165	156	1049
	0.8	184	1212	203	1128	200	960
	0.9	244	1613	276	1204	272	1155

Из результатов можем видеть, что на графах на 200 вершинах все три алгоритма показывают себя практически одинаково эффективно, однако на графах на 1000 вершин алгоритм муравьиной колонии показывает себя

значительно лучше. Это может быть вызвано тем, что локальный поиск не так эффективен в условиях сильных ограничений, так как можем видеть, что разрыв между алгоритмами значительно растет с ростом плотности графа. Стоит заметить, что сравнивать время выполнения алгоритмов не имеет большого смысла, так как они не могут определить, является ли найденное решение оптимальным, соответственно их выполнение продолжается до достижения заранее определенного критерия остановки, будь то ограничение на число итераций или на отсутствие улучшения решения.

Схожесть результатов алгоритма частичной раскраски и гибридного генетического алгоритма можно объяснить тем, что они оба используют локальный поиск для улучшения решений.

Если же смотреть на время, потраченное для нахождения решений, можно увидеть, что алгоритм муравьиной колонии самый затратный вне зависимости от параметров графа. Для больших графов это можно объяснить тем, что он совершает больше итераций, ведь решения у него лучше, чем у других двух алгоритмов. Однако и на небольших графах его выполнение занимает больше времени. Причиной этого может быть критерий остановки по числу итераций – его итерации могут быстрее приближать решение к оптимальному ценой большей трудоемкости каждой из них.

Теперь сравним результаты применения алгоритмов к равномерным графам на 500 вершинах. Найденное хроматическое число и время его поиска отображены в таблице 4.2.

Таблица 4.2 – Результаты применения алгоритмов к равномерным графам с $n = 500$

		Алгоритм муравьиной колонии		**Гибридный генетический алгоритм**		**Алгоритм частичной раскраски**	
s	***p***	**χ**	***t*, мс.**	**χ**	***t*, мс.**	**χ**	***t*, с.**
10	0.1	13	86	12	84	12	48
	0.3	10	295	10	452	11	381
	0.5	10	36	10	186	11	119
	0.7	10	41	10	13	10	15
	0.9	10	20	10	5	10	5
50	0.1	13	1441	13	44	13	35
	0.3	32	200	31	470	31	241
	0.5	52	750	52	288	51	683
	0.7	73	1099	76	653	72	900
	0.9	50	26	60	1167	50	1203
100	0.1	14	90	13	81	13	102

Продолжение таблицы 4.2

	0.3	32	845	31	581	31	190
	0.5	53	334	53	230	52	361
	0.7	80	1028	81	529	80	427
	0.9	100	567	127	965	124	1054

Из результатов применения алгоритмов к равномерным графам можем видеть, что в сравнении друг с другом алгоритмы ведут себя так же, как и в случае со случайными графами – на более плотных графах алгоритм муравьиной колонии показывает себя лучше всего, однако интересно в случае равномерных графов другое. По результатам, в особенности при $s = 50$, видно, что при средних значениях p алгоритмы показывают себя хуже, чем при высоких. Так, для каждого s, есть свое значение p, при котором алгоритмы показывают себя хуже всего. Это можно объяснить тем, что с какого-то момента увеличение числа ребер, а соответсвенно и числа ограничений, делает задачу поиска оптимума только проще, ведь у алгоритма остается меньше вариантов уйти от раскраски, где исходные независимые множества образуют цветовые классы. И хотя равномерные графы являются довольно искусственными, они дают представление о поведении алгоритмов при увеличении числа ограничений, когда гарантированно существует оптимальное решение задачи раскраски.

Наконец, результаты применения алгоритмов к задачам, которые возникали на практике, например в задаче аллокации регистров, отображены в таблице 4.3.

Таблица 4.3 – Результаты применения алгоритмов к графам из практических задач

			Алгоритм муравьиной колонии		**Гибридный генетический алгоритм**		**Алгоритм частичной раскраски**	
n	m	χ	χ	t, мс.	χ	t, мс.	χ	t, мс.
450	8263	25	25	32	25	4	25	2
450	17343	25	28	469	27	85	27	524
185	3946	31	31	37	31	2	31	1
864	18707	54	54	115	54	8	54	6
496	11654	65	65	48	65	4	65	2

По результатам можем видеть, что качество полученных решений очень близко, хотя время выполнения существенно отличается. Это можно объяснить тем, что графы, полученные из практических задач, являются довольно разреженными. В условиях, когда ребер не так много, даже приближенные

алгоритмы дают раскраски оптимальные или близкие к ним. Из-за этого и разница в качестве решений между алгоритмами может быть незаметна. В этом случае более сложные алгоритмы могут выполнятся значительно дольше, чем простые конструктивные. Это обусловлено тем, что даже если оптимальное решение будет найдено сразу, потребуется время, чтобы алгоритм остановил попытки найти решение лучше.

К результатам выше стоит добавить, что недетерминированность алгоритмов раскраски, а также множество разных параметров, которые можно менять в зависимости от задачи, не позволяют делать однозначных выводов о предпочтительности того или другого алгоритма. Изменение критерия остановки или какого-либо параметра в алгоритме может дать прибавку к эффективности даже большую, чем смена самого алгоритма. Так, гибридный генетический алгоритм и алгоритм частичной раскраски показывают примерно одни результаты при том, что работают в разных пространствах решений. В то же время качество раскраски можно улучшить, просто увеличив время, которое алгоритм будет пытаться улучшить решение.

ГЛАВА 6. ПРИМЕНЕНИЕ АЛГОРИТМОВ РАСКРАСКИ

На практике алгоритмы раскраски графов находят широкое применение в таких областях, как составление расписаний, турнирных таблиц, распределение ресурсов, аллокация регистров. Рассмотрим, каким образом практические задачи формулируются в терминах задачи раскраски.

Классическая задача о создании расписания требует распределить разные процессы по временным ячейкам, при этом некоторые процессы не могут выполняться одновременно, например из-за необходимости использовать один и тот же ресурс. Соответствующий этой задаче граф имеет вершины, оражающие процессы, причём между вершинами есть ребро, если они не могут относиться к одной временной ячейке. Структуру графа определяет специфика конкретной задачи.

Для другого примера рассмотрим задачу аллокации регистров. Компилятору нужно разместить переменные в регистрах, при этом некоторые переменные будут использоваться одновременно. Граф для такой задачи имеет вершины, соответствующие переменным, рёбра же есть между вершинами, если относящиеся к ним переменные будут использоваться одновременно. Наименьшим необходимым числом регистров будет являться хроматическое число этого графа.

Наконец, рассмотрим задачу составления рассадки. Предположим, что преподаватель знает, какие студенты могут списать друг у друга, и хочет распределить по аудитории разные варианты контрольной работы. Тогда вершины в графе будут соответствовать студентам, а рёбра будут между теми, кто может списать. Тогда хроматическое число графа – наименьшее число вариантов контрольной, при котором никто не сможет списать.

ЗАКЛЮЧЕНИЕ

В ходе данной работы было проведено исследование алгоритмов раскраски графов и их области применения. Были изучены основные понятия и определения, связанные с задачей раскраски графа, а также некоторые виды современных алгоритмов раскраски. Было проведено сравнение эффективности некоторых алгоритмов в зависимости от входных данных задачи раскраски графа.

В результате работы были получены следующие выводы:

– хотя задача NP-трудная, для некоторых видов графов она имеет быстрое и точное решение;

– на практике для графов больших размерностей нецелесообразно использование точных алгоритмов;

– для разреженных графов, часто возникающих в практических задачах, приближенные алгоритмы дают решения высокой точности;

– большое значение имеет выбор подходящего к задаче алгоритма;

– алгоритм муравьиной колонии лучше подходит для задачи раскраски графа с большим количеством ребер;

– алгоритмы на основе локального поиска быстрее находят решение задачи раскраски графа с малым числом ребер;

– кроме выбора подходящего алгоритма, для повышения эффективности поиска раскраски важен правильный подбор его параметров;

– задача раскраски графа имеет множество практических применений.

В целом, исследование алгоритмов раскраски графов представляет важную область в теории графов и дискретной математике. Полученные результаты и выводы могут быть использованы для улучшения планирования и оптимизации ресурсов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Appel, K. "Every planar map is four colorable." / K. Appel, W. Haken. – Bull. Amer. Math. Soc. 82 (5), 1976. – P. 711–712.
2. Blochliger, I. A graph coloring heuristic using partial solutions and a reactive tabu scheme / I. Blochliger, N. Zufferey. – Computers and Operations Research, 35, 2008. – P. 960–975.
3. Brélaz, D. New methods to color the vertices of a graph / D. Brélaz. – Commun. ACM, 22(4), 1979. – P. 251–256.
4. Brooks, R. L. On colouring the nodes of a network / R.L . Brooks. – Proc. Cambridge Philosophical Society, Math. Phys. Sci., 37, 1941. – P. 194-197.
5. Introduction to Algorithms / T.H. Cormen[и др.]. – The MIT Press, Cambridge, Massachusetts, 2009. – P. 1048–1086.
6. Culberson, J. Exploring the k-colorable landscape with iterated greedy / J. Culberson, E. Luo. – American Mathematical Society: Cliques, Coloring, and Satisfiability – Second DIMACS Implementation Challenge, 26, 1996. – P. 245–284.
7. Erben, E. A grouping genetic algorithm for graph colouring and exam timetabling / E. Erben. – Practice and Theory of Automated Timetabling (PATAT) III, 2079, 2001. – P. 132–158.
8. Galinier, P. Hybrid evolutionary algorithms for graph coloring / P. Galinier, J.-K. Hao. – Journal of Combinatorial Optimization, 3, 1999. – P. 379–397.
9. Galinier, P. A survey of local search algorithms for graph coloring / P. Galinier, A. Hertz. – Computers and Operations Research, 33, 2006. – P. 2547–2562.
10. Hindi, M. Genetic Algorithm Applied to the Graph Coloring Problem / M. Hindi, R. Yampolskiy. – Midwest Artificial Intelligence and Cognitive Science Conference, 2012. – P.60.
11. Jensen, T. R. Graph Coloring Problems. / T. R. Jensen, B. Toft. – Wiley-Interscience, New York, 1995. – P.31.
12. Kubale, M. Graph Colorings. / M. Kubale – American Mathematical Society, 2004. – P.23.
13. Kubale, M. GENERALIZED IMPLICIT ENUMERATION ALGORITHM FOR GRAPH COLORING / M. Kubale, B. Jackowski. – Communications of the ACM, 28(4), 1985. – P. 412–418.
14. Leighton F. A graph coloring algorithm for large scheduling problems / F. Leighton. – Journal of Research of the National Bureau of Standards, 84(6), 1979. – P. 489–506.
15. Lewis, R.M.R. A Guide to Graph Colouring: Algorithms and Applications. / R.M.R. Lewis. – Springer International Publishing, 2016. – P.20.

16. Matula, D. Two linear-time algorithms for five-coloring a planar graph / D. Matula, Y. Shiloach, R. Tarjan. – Tech. Report STAN-CS-80-830, Stanford University, 1980.
17. Méndez-Díaz, I. A cutting plane algorithm for graph coloring / I. Méndez-Díaz, P. Zabala. – Discrete Applied Mathematics, 156, 2008. – P. 159–179.
18. Mumford, C. New order-based crossovers for the graph coloring problem / C. Mumford. – In Parallel Problem Solving from Nature (PPSN) IX, volume 4193 of LNCS, Springer, 2006. – P. 880–889.
19. Thompson, J. An improved ant colony optimisation heuristic for graph colouring / J. Thompson, K. Dowsland. – Discrete Applied Mathematics, 156, 2008. – P. 313–324.