Санкт-Петербургский государственный университет

Направление Математическое обеспечение и администрирование информационных систем
Профиль системное программирование

Беличенко Дмитрий Алексеевич

Улучшенный алгоритм для решения задачи Maximum Happy Vertices на дереве

Курсовая работа

Научный руководитель:
Сагунов Д. Г.

Санкт-Петербург
2020

Оглавление

Введение

Согласно отчету о состоянии цифровой сферы Digital 2020, который каждый год готовят We Are Social и Hootsuite, в январе 2020 года в мире насчитывалось 3,80 миллиарда пользователей социальных сетей, аудитория соцмедиа выросла на 9 процентов по сравнению с 2019 годом (это 321 миллион новых пользователей за год) [6].

Социальные сети имеют огромный потенциал, для того чтобы открыть двери и создать новые возможности для общения, работы и партнерства. Они устанавливают связи между людьми и дают доступ к той информации, к которой у человека обычно нет доступа.

Огромную важность приобретает изучение поведения и взаимодействия людей в социальных сетях и выявление их особенностей.

Одной из важных и известных сетевых особенностей является гомофилия. Самое простое определение гомофилии – это склонность людей со схожими характеристиками общаться друг с другом. «Рыбак рыбака видит издалека» — именно так определил гомофилию в социальных сетях в своей работе Мак Персон [17].

В то же время, гомофилия не формируется сама по себе. Как правило, существуют институциональные или географические факторы, которые способствуют созданию гомофильных связей. К примеру, люди, которые проживают на небольшом расстоянии друг от друга, работают в одной организации, скорее всего, со временем познакомятся и между ними образуется социальная связь.

Также интересен пример использования гомофилии приведенный в [16]. Была взята сеть состоящая из 27 770 вершин (то есть документов) и 352 807 направленных ребер (то есть цитат), в которой, тем не менее, только 1 214 статей имели ключевые слова, аннотированные их авторами. Задача состояла в том, чтобы предсказать ключевые слова оставшихся статей. Согласно положению о гомофилии, статьи в небольшом сообществе сети должны иметь общие ключевые слова. После реализации несложного алгоритма были успешно найдены ключевые слова для 14 123 (70 процентов) неаннотированных статей. Эксперимент

предполагает, что реальные сети действительно удовлетворяют закону гомофилии, и что закон гомофилии может быть использован в качестве принципа для предсказания общих атрибутов в большой сети.

Для наглядности приведем пример одной из известных сетевых визуализаций (рисунок 1). На картинке изображена сеть дружбы американских школьников. Цвет узлов обозначает расу школьников. Видно, что школьники одной расы тяготеют друг к другу.

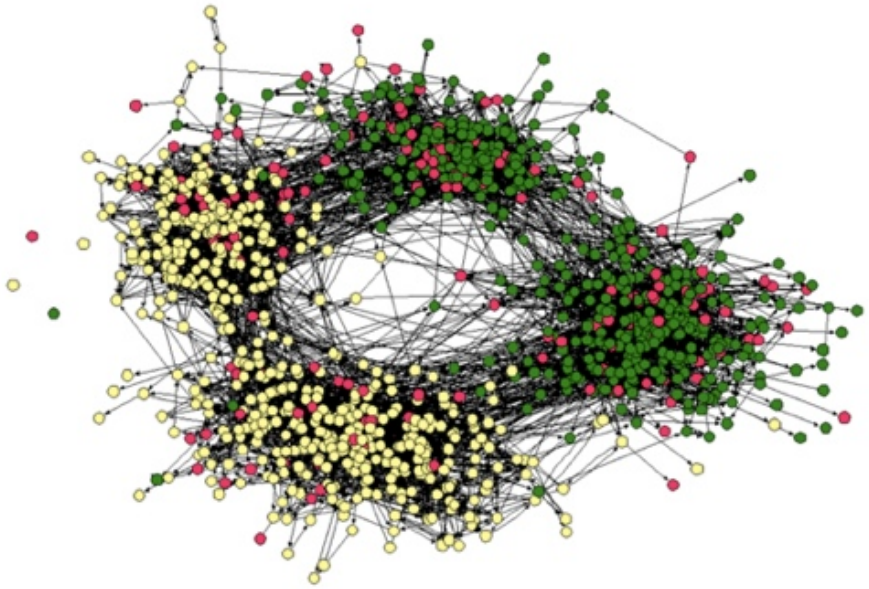

Рис. 1: Сеть дружбы американских школьников на основе данных из работы Дж. Муди [8]

Эффективным способом исследования сетей является метод представление их в виде графа.

Исследование и анализ такого графа может быть выполнен с применением соответствующих математических алгоритмов. Проблемы возникающие при изучении феномена гомофилии в крупномасштабных сетях – это две известные проблемы теоретической информатики:

— Maximum Happy Vertices (MHV)

— Maximum Happy Edges (MHE).

Задачи MHV и MHE были определены в статье [13]. В задаче MHV дан граф, некоторые вершины которого изначально покрашены в несколько цветов. Вершина в графе называется счастливой, если она и все ее соседи имеют один и тот же цвет. Ответом на задачу является такая раскраска оставшихся вершин, что количество счастливых вершин в такой раскраске максимально возможно. В задаче MHE требуется максимизировать количество счастливых ребер. Ребро считается счастливым, если его концы имеют один и тот же цвет.

Известно, что обе задачи являются NP-трудными, если в раскраске используются хотя бы три различных цвета, даже на некоторых простых классах графов [13]. Поэтому в трудах последних лет рассматриваются различные частые решения задач MHE и MHV.

В недавней статье [13] Zhang и Li, доказали, что задачи MHV и MHE являются NP-трудными в общем случае, если число цветов, в которые можно красить вершины ≥ 3. Также они предъявили алгоритмы с полиномиальным временем работы для частного случая, когда каждую вершину разрешено красить в два цвета.

Проблемы MHV и MHE были рассмотрены для другого частного случая в статье [9]. Здесь авторы доказали, что MHV и MHE разрешимы за полиномиальное время в том случае, когда исходный граф является деревом, предъявив алгоритмы решения MHV за $O(nk \log n)$ и MHE за $O(nk)$.

В статье [11] авторы изучают древесную ширину некоторых социальных сетей, так, например, оказалось, что древесная ширина социального графа форума Mathematics Stack Exchange, в котором более миллиона вершин, не превышает 11100. Также в статье [1], авторы изучают древесные декомпозиции социальных графов, работа с которыми помогает в анализе социальных сетей. Как было показано в работе [4], алгоритм, использующий идею динамического программирования на дереве, обобщается до алгоритма с полиномиальным временем работы на графах с ограниченной древесной шириной.

Поэтому в данной работе подробно будет изучен вопрос улучшения времени работы алгоритма, использующего идею динамического программирования, для задачи MHV на графах, являющихся деревьями [9]. Так как решение задачи MHV на деревьях может быть обобщено на древесную декомпозицию графов с ограниченной древесной шириной, что внесет вклад в решение практически значимой проблемы гомофилии. Необходимо так же отметить, что графы в крупномасштабных сетях обладают внушительным размером, поэтому требуются самые быстрые алгоритмы для их изучения и исследования, и любое увеличение скорости работы при прочих равных условиях можно только приветствовать.

1. Постановка задачи

1.1. Цель работы

Дан неориентированный граф $G = (V, E)$, где $|V| = n$, с некоторой вершинной раскраской, где для некоторых вершин уже задан цвет, а остальные вершины графа могут быть покрашены в один из k цветов. Вершина $u \in V$ называется счастливой, если все вершины, соединенные с ней ребром, имеют тот же цвет, что и она. В проблеме Maximum Happy Vertices необходимо покрасить вершины без цвета так, чтобы количество счастливых вершин в графе было максимальным. Целью работы является улучшение теоретической оценки быстродействия существующего алгоритма для решения MHV на дереве, описанного в статье [9], так как этот алгоритм имеет лучшую оценку времени работы на данный момент.

1.2. Задачи

1) Осуществить сравнение и обзор существующих алгоритмов для решения задачи MHV на деревьях.

2) Разработать альтернативный алгоритм с лучшей оценкой времени работы для решения задачи MHV на дереве.

3) Сравнить время работы реализаций существующих и улучшенного алгоритма на одной машине на одном наборе исходных данных.

2. Обзор существующих решений

В общем случае, если $k \geq 3$, как было показано в статье [13], MHV это задача из класса сложности NP, то есть для нее не существует алгоритма с полиномиальным временем работы. Все алгоритмы, решающие задачу в общем случае — переборные и работают за экспоненциальное время, из-за чего их использование на практике для деревьев, в которых есть хотя бы 1000 вершин, уже недопустимо. В последнее время вышло немало статей по этой теме. Данную задачу исследовали, как с параметрической точки зрения [2] [9] [10] [5] [7], так и с целью найти приближенное решение [3], [12] [14].

Последним и лучшим результатом для решения MHV на дереве, является алгоритм описанный в статье [9]. Время его работы на дереве из n вершин, где разрешается красить каждую вершину в один из k цветов $O(nk\log(k))$, что является большим успехом по сравнению с асимптотикой для общего случая с экспоненциальным временем работы.

Идея алгоритма, описанного в статье [9] состоит в следующем.

Заметим, что если в дереве есть листья (такие вершины $u : deg(u) = 1$), для которых не задан изначальный цвет, то мы можем запомнить их количество и решить задачу для графа, в котором нет изначально непокрашенных листьев, и тогда ответом на задачу будет сумма количества неокрашенных листьев и ответа для графа, в котором их нет. Таким образом, дальше мы будем всегда решать задачу для графов, в которых нет непокрашенных листьев.

Для того, чтобы понять, какое максимальное количество счастливых вершин может быть в графе, в котором все листья предварительно покрашены, авторы предлагают вычислить на исходном графе две функции:

$T_v[i, H]$ — максимальное количество счастливых вершин в поддереве вершины v, при условии, что вершина v имеет цвет i, и она является счастливой в ее поддереве T_V.

$T_v[i, U]$ — максимальное количество счастливых вершин в поддереве вершины v, при условии, что вершина v имеет цвет i, и она не является

счастливой в ее поддереве T_V. (Такое бывает тогда и только тогда, когда одна или более вершина, смежная с v, имеет цвет, отличный от i)

Параметры U и H, отвечают за счастье вершины. Если параметр H — вершина счастливая, если параметр U — несчастливая.

Чтобы понять, как основываясь на этих функциях посчитать ответ, авторы предлагают посчитать для каждой вершины v исходного графа следующие значения:

$T_v[i, *]$ — максимальное количество счастливых вершин в поддереве вершины v T_V, где вершина v покрашена в цвет i, но она может быть как счастливой, так и не счастливой. Отсюда по определению:

$$T_v[i, *] = max(T_v[i, H], T_v[i, U]) \ (1).$$

$T_v[i, -]$ — максимальное количество счастливых вершин в поддереве вершины v - T_V, не считая саму вершину v, при условии, что вершина v покрашена в цвет i.

$$T_v[i, -] = max(T_v[i, H] - 1, T_v[i, U]) \ (2).$$

$T_v[\bar{i}, *]$ — максимальное количество счастливых вершин в поддереве вершины v — T_V, при условии, что вершина v покрашена в цвет отличный от i.

$$T_v[\bar{i}, *] = \max_{c \neq i} (T_v[c, *]) \ (3).$$

$T_v[\bar{i}, -]$ — максимальное количество счастливых вершин в поддереве вершины v — T_V, не учитывая вершину v и при условии, что вершина v покрашена в цвет отличный от i.

$$T_v[\bar{i}, -] = \max_{c \neq i} (T_v[c, -]) \ (4).$$

$T_v[*, *]$ — максимальное количество счастливых вершин в поддереве вершины v — T_V.

$$T_v[*, *] = \max_{i \in [1..k]} (T_v[i, *]) \ (5).$$

После введения всех необходимых обозначений, чтобы решить задачу нужно научиться считать $T_v[i, H]$ и $T_v[i, U]$. Для простоты будем считать, что если значение некоторой функции недопустимо, то оно будет равняться -1.

Научимся считать $T_v[i, H]$ для некоторой вершины v. Обозначим за $\Gamma(v)$ множество вершин из поддерева вершины v. Заметим, что существует всего два случая:

Случай 1: Если существует вершина u, такая что между u и v существует ребро, и нет допустимого значения $T_u[i, *]$, то тогда, если мы покрасим вершину v в цвет i, это будет означать, что v перестанет быть счастливой, поэтому значение $T_v[i, H]$ станет также недопустиым.

Случай 2: Если для любой вершины $u : T_u[i, *] \neq -1$, то тогда это означает, что если мы покрасим v в цвет i, то ее "счастливость" не нарушится, и тогда, согласно определению:

$$T_v[i, H] = 1 + \sum_{u \in \Gamma(v)} T_u[i, *] \ .$$

Псеводкод процедуры подсчета $T_v[i, H]$ на Рис. 2.

Algorithm 1 Computing $T_v[i, H]$

```
1: procedure ComputeTvH(v, i)
2:     if ∀v_j, T_{v_j}[i, *] ≠ −1 then
3:         return (1 + Σ_{v_j} T_{v_j}[i, *])          ▷ Case 2
4:     else
5:         return −1                                   ▷ Case 1
6:     end if
7: end procedure
```

Рис. 2: Псеводкод процедуры подсчета $T_v[i, H]$ из статьи [9]

Теперь Научимся считать $T_v[i, U]$ для некоторой вершины v. При подсчете этой функции уже возникает 3 случая:

Случай 1: Если любая вершина поддерева вершины v, имеет тот же цвет i, что и она, то тогда мы не можем сделать ее несчастливой, и поэтому значение $T_v[i, U] = 1$ (не валидно).

Случай 2: Если в поддереве вершины v существует вершина $\overline{u}$, такая что для нее верно: $T_{\overline{u}}[*, *] \neq T_{\overline{u}}[i, *]$. То есть если вершина $\overline{u}$ в

оптимальной раскраске имеет цвет $c \neq i$, то тогда вершина v несчастливая. В этом случае согласно определению используется следующая формула:

$$T_v[i, U] = \sum_{u \in \Gamma(v)} \max(T_u[1, -], T_u[2, -], ..., T_u[i, *], ..., T_u[k, -]). \quad (7)$$

Случай 3: Если в поддереве вершины v для каждой вершины u поддерева верно $T_u[*, *] = T_u[i, *]$. То есть для каждой вершины поддерева u, если мы возьмем $T_u[i, *]$, то вершина v станет счастливой, чего мы хотим избежать при подсчете $T_v[i, U]$. Чтобы обойти эту ситуацию для вершины u, нам следует взять значение, соответствующее цвету, отличному от i. Определим для каждой вершины u: $diff(u, i)$ как

$$diff(u, i) = T_u[i, *] - T_u[\bar{i}, -] \quad (8)$$

Таким образом, в этом случае нам следует взять вершину поддерева s такую, что для нее $diff(s, i) = \min_{u \in \Gamma(v)} diff(u, i)$. Пусть $T_s[\bar{i}, -] = T_s[q, -]$, тогда итоговая формула для данного случая будет:

$$T_v[i, U] = T_s[q, -] + \sum_{u \neq s} T_u[i, *] \quad (8)$$

Псеводкод процедуры подсчета $T_v[i, U]$ на Рис. 3.

После того, как уже известно как считать, $T_v[i, H], T_v[i, U]$ можно привести псевдокод итогового алгоритма для решения MHV на дереве.

Подсчет $T_v[i, H]$ занимает $O(nk)$, так как на каждом шагу алгоритма подсчета этой функции, доступ к $T_u[i, H]$ и $T_u[i, U]$ осуществляется за $O(1)$.

Подсчет $T_v[i, U]$ занимает $O(nk \log(k))$, так как на каждом шагу алгоритма подсчета этой функции нам необходимо сортировать уже насчитаные значения функций $T_u[*, *]$ и $T_u[\bar{i}, *]$.

То есть итоговое время работы алгоритма: $O(nk \log(k))$.

Algorithm 2 Computing $T_v[i, U]$

procedure ComputeTvU(v, i)
  if every child v_j is pre-colored with color i **then**
    return -1 ▷ Case 1
  else if $\exists v_{j'}$ child of v such that $T_{v_{j'}}[*, *] \neq T_{v_{j'}}[i, *]$ **then**
    return $(\sum_{v_j} \max\{T_{v_j}[1, -], \ldots, T_{v_j}[i, *], \ldots, T_{v_j}[k, -]\})$ ▷ Case 2
  else ▷ Case 3
    for each child v_j **do**
      $\text{diff}(v_j, i) \leftarrow T_{v_j}[i, *] - T_{v_j}[\bar{i}, -]$
    end for
    $v_\ell \leftarrow \text{argmin}_{v_j} \text{diff}(v_j, i)$
    $q \leftarrow \text{argmax}_{r \neq i} T_{v_\ell}[r, -]$
    return $(T_{v_\ell}[q, -] + \sum_{v_j \neq v_\ell} T_{v_j}[i, *])$
  end if
end procedure

Рис. 3: Псеводкод процедуры подсчета $T_v[i, U]$ из статьи [9]

Algorithm 3 Algorithm for MHV problem

for each $v \in V$ in post order **do**
  for $i = 1$ to k **do**
    if v is a leaf **then**
      if $\text{color}(v) = i$ **then**
        $T_v[i, H] \leftarrow 1$
        $T_v[i, U] \leftarrow -1$
      else
        $T_v[i, H] \leftarrow -1$
        $T_v[i, U] \leftarrow -1$
      end if
    else
      if v is pre-colored and $\text{color}(v) \neq i$ **then**
        $T_v[i, H] \leftarrow -1$
        $T_v[i, U] \leftarrow -1$
      else
        $T_v[i, H] \leftarrow$ ComputeTvH(v, i)
        $T_v[i, U] \leftarrow$ ComputeTvU(v, i)
      end if
    end if
  end for
end for

Рис. 4: Псеводкод итогового алгоритма из статьи [9]

3. Альтернативный алгоритм

3.1. Описание алгоритма

Для того, чтобы решить MHV на дереве, целесообразно вычислить значения следующих функций:

$F[v, c]$ — максимальное количество счастливых вершин в поддереве вершины v, при условии, что вершина v имеет цвет c.

$G[v, c]$ — максимальное количество счастливых вершин в поддереве вершины v, при условии, что вершина v имеет цвет c и вершина v является счастливой.

Также будем хранить для каждой вершины v значение $mF[v]$ — максимальное значение $F[v, c]$: $mF[v] = \max_{c \in [1..k]} F[v, c]$.

Согласно обозначениям, данным выше, ответом на задачу будет являться:

$$answer = \max_{c \in [1..k]}(G[root, c], mF[root])$$

Будем считать значения этих функций в процессе обратного обхода (начиная с листьев), посещая каждую вершину по одному разу. Не умаляя общности будем считать, что вершина с номером 1 является корнем, и алгоритм будет начинать работу именно с нее. Непосредственно перед началом подсчета проинициализируем все исходные значения функций для любых v и c как:

$$F[v, c] = G[v, c] = mF[v] = -\infty$$

Научимся считать значения $F[v, c]$, $G[v, c]$ и $mF[v]$. Для этого рассмотрим несколько случаев:

Случай 1: Если вершина v, для которой мы хотим посчитать значения этих функций является листом и у нее изначально нет цвета, то тогда, согласно определению:

$$F[v, c] = 0,\ G[v, c] = 1 \text{ для любого } c,\ mF[v] = \max_{c \in [1..k]} F[v, c] = 0$$

Случай 2: Если вершина v, для которой мы хотим посчитать значения этих функций, является листом и у нее есть изначальный цвет c, то тогда, согласно определению:

$$F[v,c] = 0,\ G[v,c] = 1\ mF[v] = \max_{c\in[1..k]} F[v,c] = 0$$

Случай 3: Если вершина v, для которой мы хотим посчитать значения этих функций, не является листом и у нее изначально нет цвета, то тогда подсчет будет производиться в три стадии:

Стадия 3.1: инициализация значений:

$$F[v,c] = 0,\ G[v,c] = 1 \text{ для любого } c \in [1..k]$$

Стадия 3.2: пересчет значений $F[v,c]$, $G[v,c]$: поскольку мы считаем значения в процессе обратного обхода дерева, это означает, что если вершина v не является листом,то для любой вершины u, находящейся в ее поддереве ($u \in \Gamma(v)$), значения уже посчитаны. Тогда пересчет, согласно определениям, $F[v,c]$, $G[v,c]$, $mF[v]$ будет осуществляться следующим образом:

$$\forall c \in [1..k] : F[v,c] = \sum_{u\in\Gamma(v)} \max(mF[v], G[v,c])$$

$$\forall c \in [1..k] : G[v,c] = \sum_{u\in\Gamma(v)} \max(F[v,c], G[v,c])$$

Стадия 3.3: пересчет значений $mF[v]$:

$$mF[v] = \max_{c\in[1..k]} F[v,c]$$

Случай 4: Если вершина v, для которой мы хотим посчитать значения этих функций, не является листом и у нее есть изначальный цвет, то тогда подсчет будет производиться в три стадии:

Стадия 4.1: инициализация значений:

$$F[v,c] = 0,\ G[v,c] = 1 \text{ для всех возможных комбинаций } (v,c)$$

Стадия 4.2: пересчет значений $F[v, c]$, $G[v, c]$: поскольку мы считаем значения в процессе обратного обхода дерева, это означает, что если вершина v не является листом, то для любой вершины u, находящейся в ее поддереве ($u \in \Gamma(v)$), значения уже посчитаны. Тогда пересчет, согласно определениям, $F[v, c]$, $G[v, c]$, $mF[v]$ будет осуществляться следующим образом:

$$F[v, c] = \sum_{u \in \Gamma(v)} \max(mF[u], G[u, c])$$

$$G[v, c] = \sum_{u \in \Gamma(v)} \max(F[u, c], G[u, c])$$

Стадия 4.3: пересчет значений $mF[v]$:

$$mF[v] = \max_{c \in [1..k]} F[v, c]$$

Ниже представлен псеводокод алгоритма, описанного выше.

Algorithm 1 improved algorithm for MHV on tree problem

```
1: for each v ∈ V in post order do
2:     for c = 1 to k do
3:         if color_v = c then
4:             G[v, c] = 1
5:             F[v, c] = 0
6:         end if
7:         if color_v is not defined then
8:             G[v, c] = 1
9:             F[v, c] = 0
10:        end if
11:    end for
12:    for each u in Γ(v) do
13:        for c = 1 to k do
14:            if color_v = c then
15:                F[v, c] = F[v, c] + max(mF[u], G[u, c])
16:                G[v, c] = G[v, c] + max(F[u, c], G[u, c])
17:            end if
18:            if color_v is not defined then
19:                F[v, c] = F[v, c] + max(mF[u], G[u, c])
20:                G[v, c] = G[v, c] + max(F[u, c], G[u, c])
21:            end if
22:        end for
23:    end for
24:    for c = 1 to k do
25:        mF[v] = max(mF[v], F[u, c])
26:    end for
27: end for
```

3.2. Доказательство корректности алгоритма

Утверждение: Алгоритм, описанный выше, вычисляет значения $F[v, c]$, $G[v, c]$ и $mF[v]$ согласно определениям для любой вершины v и цвета $c \in [1..k]$.

Доказательство. Докажем утверждение методом математической индукции по размеру поддерва. База индукции: если вершина v является листом, то для нее согласно определениям правильно вычисляются значения функций $F[v, c]$, $G[v, c]$ и $mF[v]$.

Если вершина v не является листом, то по индукционному предположению: $\forall u \in \Gamma(v)$ (u — ребенок v) правильные значения функций $F[u, c]$, $G[u, c]$ и $mF[u]$ уже посчитаны, так как подсчет значний осуществляется в порядке Post-order обхода дерева. Тогда пусть для вершины v значение функции $F[v, c]$ или $G[v, c]$ посчитано не согласно определениям. Согласно определению функций и правилам их пересчета, описанным выше, получается, что некорректное значение функции $F[v, c]$ или $G[v, c]$ в вершине v было получено из хотя бы одного из вычисленных не по определению значений функций для некоторой вершины $\overline{u}$, которая является ребенком v. Но согласно индукционному предположению для любой вершины $u \in \Gamma(v)$, в том числе и $\overline{u}$, значения функций получены правильно. Противоречие. Отсюда получаем, что алгоритм корректно вычисляет значения функций $F[v, c]$, $G[v, c]$ и $mF[v]$ для любой вершины v для любого цвета $c \in [1..k]$.

3.3. Время работы алгоритма

Утверждение: Алгоритм, описанный выше, вычисляет значения $F[v, c]$, $G[v, c]$ и $mF[v]$ для любой вершины v и цвета $c \in [1..k]$ за суммарное время $O(nk)$.

Доказательство.

Посчитаем суммарное время, требуемое для подсчета значений функций $F[v, c]$, $G[v, c]$ и $mF[v]$ в вершине v, для $c \in [1..k]$.

Во время подсчета $F[v, c]$, $G[v, c]$ доступ к уже посчитанным значениям для $u \in \Gamma(v)$ $F[u, c]$, $G[u, c]$ осуществляется за константное время. Тогда время, необходимое на подсчет $F[v, c]$, $G[v, c]$, вычисляется как:

$$\sum_{c=1}^{k} O(|\Gamma(v)|) = O(nk).$$

Вычисление $mF[v]$ занимает также $O(nk)$. Сам Post-order обход посещает все вершины по одному разу, поэтому он добавляет только константу к суммарному времени работы алгоритма. Отсюда итоговое время работы алгоритма: $O(nk)$.

4. Особенности реализации

Для того чтобы узнать, действительно ли полученное теоретическое улучшение времени работы алгоритма, влечет за собой уменьшение реального времени работы, были реализованы два алгоритма:

1. Алгоритм с оценкой времени работы $O(nk \log k)$, описанный в статье [9]

2. Алгоритм с оценкой времени работы $O(nk)$, описанный в предыдущем пункте.

Чтобы проверить правильность реализации, был написан генератор случайных деревьев. Каждое решение выполнялось на одном наборе тестов и был проверен каждый полученный ответ.

Тестирование времени работы проходило на случайных тестах. Каждый тест представлял из себя случайное дерево на n вершинах, в котором $[n/p]$ не имели изначального цвета (p выбирается случайно $p \in [1..n]$), а остальные вершины были изначально раскрашены равновероятно в один из k цветов. Сравнение времени работы реализаций алгоритмов осуществлялось на одном наборе случайных тестов.

Поскольку и алгоритм, описанный в статье [9], и мой имеют асимптотическую оценку времени работы, зависящую от двух параметров n и k, поэтому было решено, чтобы сравнить производительность зафиксировать параметр n достаточно большим, и смотреть, как изменяется производительность обоих алгоритмов по мере увеличения параметра k.

Для того, чтобы проверить ускорение алгоритма на практие было разработано приложение с архитектурой, изображенной на Рис. 5.

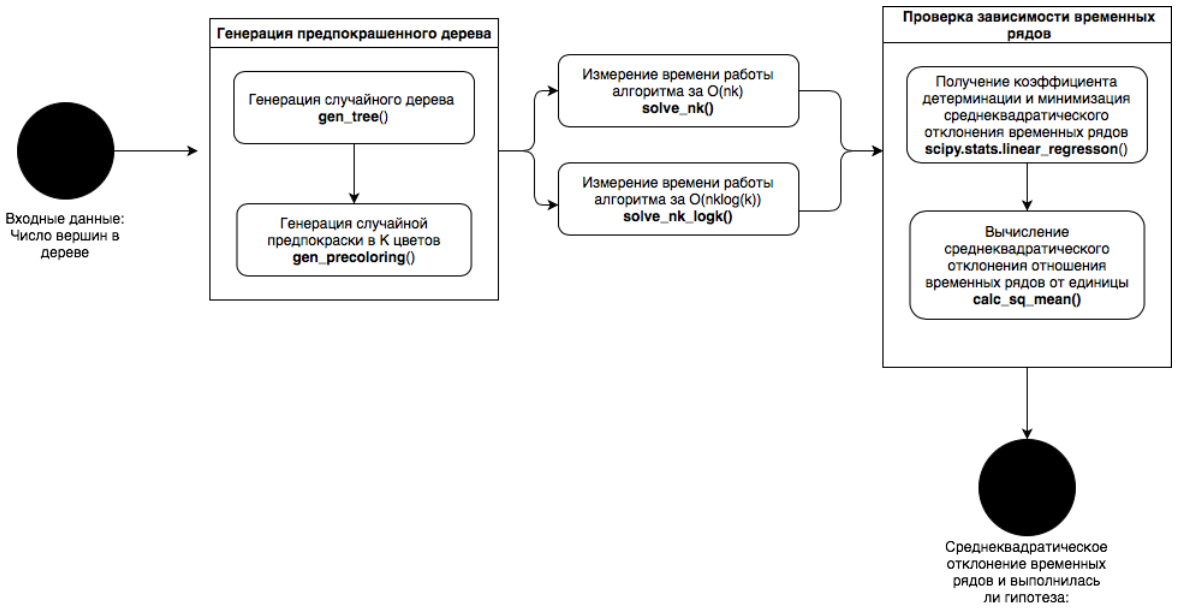

Рис. 5: Архитектура тестирующей ситсемы

На вход подается число вершин в графе — n, как аргумент командной строки. Затем генерируется случайное дерево (функция gen_tree()), для котрого для всех $k \in [2..n]$ генерируются случайные предпокраски (функция gen_precoloring()). После этого наборы полученных случайных тестов передаются в функции solve_nk() и solve_nk_logk(), для получения двух наборов времен работы на каждом тесте. Затем оба набора обрабатываются стандартной библиотекой scipy.stats [15], чтобы проверить наличие функциональной зависимости между временными рядами и найти константу минимизурющую среднеквадратическое отклонение между ними (она зависит от характеристики случайного дерева). Затем вычисляется и выводится среднеквадратическое отклонение отношения времянных рядов с учетом константы и желаемого $\log(k)$ от 1, если оно получается меньше 0.05, то считаем, что эксперимент удался.

Все вычисления производились на ЦПУ. В качестве языка реализации алгоритмов были выбраны C++, и Python 3.4.9 в силу наличия библиотеки с методами математической статистики scipy.stats [15] для проверки временных рядов.

Все материалы, с помощью которых проводились эксперименты доступны по ссылке https://github.com/mitterr/MHV-algos.

5. Сравнение времени работы алгоритмов

Все эксперименты осуществлялись на рабочей станции со следующими характеристиками:

операционная система: Mac OS 10.12.3

объем оперативной памяти: 8.0 GB

ЦПУ: 1.6 GHz Intel Core i5-5250U dual-core, 4 Logical Processor(s)

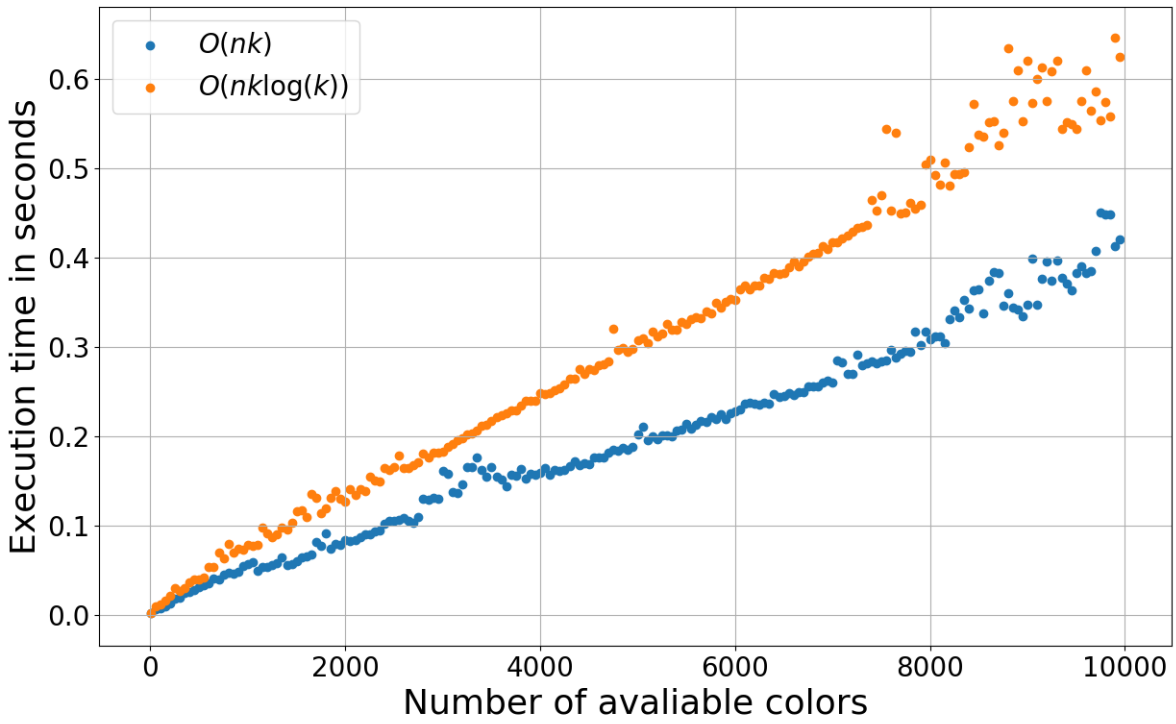

Рис. 6: Сравнение работы алгоритма из предыдщего пункта и из статьи при фиксированном $n = 10000$ [9]

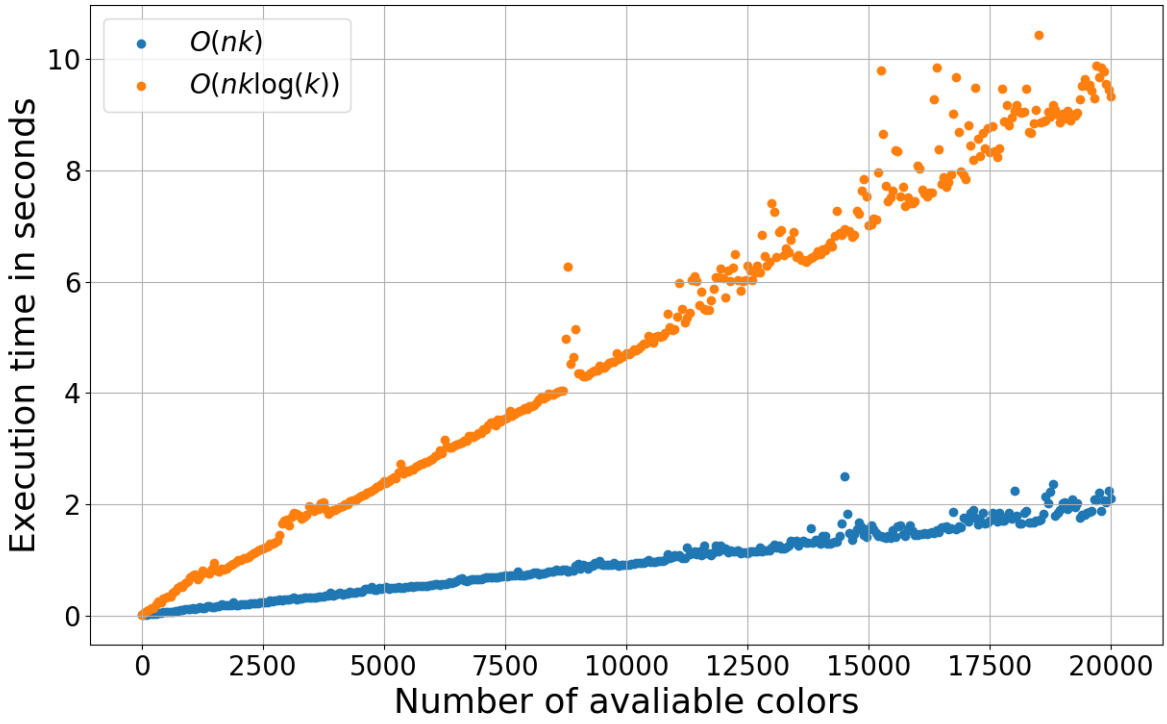

Рис. 7: Сравнение работы алгоритма из предыдщего пункта и из статьи при фиксированном $n = 25000$ [9]

На примере достаточно большого $n = 25000$ изучим результаты эксперимента. Если рассматривать график, где $n = 25000$, становится понятно, что при фиксированном значении n, при увеличении k, алгоритм, со сложнотстью $O(nk)$, начинает работать заметно быстрее, чем алгоритм, предложенный в статье [9]. Выигрыш по времени при $n = 25000$ и $k = 20000$ составляет примерно в 5 раз.

Если сравнивать графики при $n = 25000$ и при $n = 10000$, то становится понятно, что чем больше значение n, тем больше выигрыш по времени работы.

Пусть A_k — время работы алгоритма со сложнотстью $O(nk)$ на случайном дереве с фиксированным n и выбранным k.

Пусть B_k — время работы алгоритма со сложнотстью $O(nk\log(k))$ на случайном дереве с фиксированным n и выбранным k.

Было проделано несколько экспериментов, при достаточно больших n. В последующих абзацах идут результаты, полученные при $n = 25000$ и постепенно увеличивающимся k на 50 до 20000 (см Рис. 6).

Чтобы показать, что выигрыш в практическом времени работы действительно пропорционален $O(\log(k))$, достаточно показать, что $B_k = C\log(k)A_k$, где C — некоторая константа, не зависимая от k, которая зависит от характеристики случайного дерева.

Для того, чтобы убедиться, что между A_k и B_k есть стойкая функциональная зависимость, то есть, что $B_k = f(k)A_k$, необходимо вычислить коэффициент детерминации R^2. Для его вычисления была использована open-source библиотеку языка Python scipy.stats [15]. Коэффициент $R^2 = 0.963 > 0.9$, то есть можно говорить о стойкой функциональной зависимости между A_k и B_k.

Чтобы проверить, что $f(k) = C\log(k)$, вычислим C. С помощью линейной регрессии (из библиотеки scipy.stats), вычислим такую C, что она минимизирует среднеквадратическое отклонение $A'_k = A_k\log(k)$ и B_k. Было получено $C = 1.681$.

Осталось убедиться, что $T_k = \frac{B_k}{C\log(k)A_k}$ находится в окрестности 1. Среднеквадратическое отклонение T_k и 1 оказалось исключительно небольшим $\sigma = 0.027$.

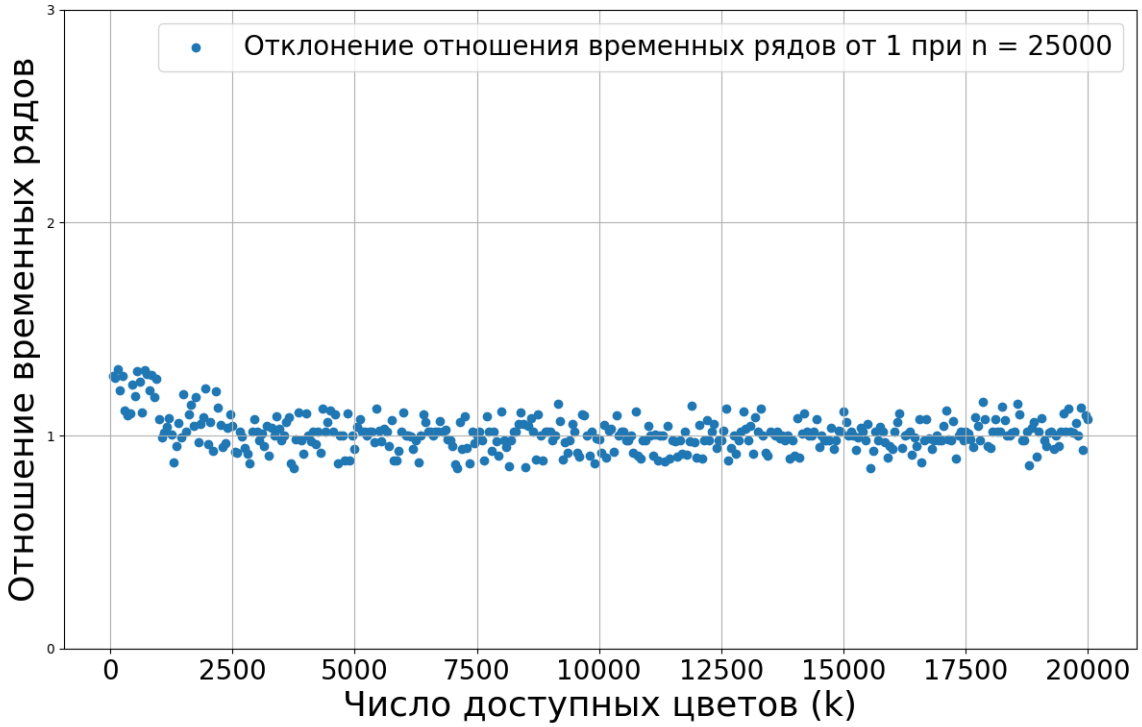

Рис. 8: отклонение $T_k = \frac{B_k}{C \log(k) A_k}$ от 1 при $n = 25000$

n	R^2	σ
25000	0,962	0,027
20000	0,962	0,029
15000	0,958	0,031
10000	0,967	0,033
5000	0,959	0,039

Рис. 9: Таблица запусков тестирующей программы при разных n

Основываясь на результатах экспериментов при разных n, можно заметить, что $\sigma \leq 0.05$. То есть можно утверждать, что доказанное теоретическое ускорение времени работы действительно влечет за собой реальное улучшение в $\log(k)$ раз.

Заключение

В данной работе были получены следующие результаты:

1. Произведен обзор существующих алгоритмов для решения задачи MHV на дереве.

2. Был предложен альтернативный алгоритм с оценкой времени работы $O(nk)$, что являтся улучшением по сравнению с $O(nk \log k)$.

3. Доказана корректность и оценка времени работы, разработанного алгоритма.

4. Произведено экспериментальное исследование алгоритмов, решающих поставленную задачу, показавшее, что предложенный алгоритм имеет не только лучшую теоретическую оценку времени работы, но и на практике показывает значительный рост производительности. Также показано, что выигрыш в скорости работы алгоритма тем значительней, чем больше количество вершин в дереве.

Также хочу определить несколько направлений будущих исследований. Разработанный алгоритм можно обобщить на древесное разложение произвольного графа, что позволит применить его к социальным сетям. В настоящее время ведется работа над разработкой алгортма, решающего MHV на дереве за $O(n \log(n))$.

Список литературы

[1] Aaron B. Adcock Blair D. Sullivan Michael W. Mahoney. Tree decompositions and social graphs. IMIS 2014.

[2] Agrawal A. On the parameterized complexity of happy vertex coloring. In: Intenational Workshop on Combinatorial Algorithms. pp. 103–115. Springer (2017).

[3] Angsheng Li Peng Zhang. Algorithmic Aspects of Homophyly of Networks. — 2018. — Access mode: `https://arxiv.org/pdf/1207.0316.pdf`.

[4] Bodlaender Hans L. Dynamic programming on graphs with bounded treewidth. ICALP 1988 pp. 105-118.

[5] Choudhari J. Reddy I.V. On structural parameterizations of happy coloring, empire coloring and boxicity. In: WALCOM: Algorithms and Computation, pp. 228–239. Springer International Publishing (2018).

[6] Digital-2020. Linear Time Algorithms for Happy Vertex Coloring Problems for Trees. — 2020. — Access mode: `https://wearesocial.com/blog/2020/01/digital-2020-3-8-billion-people-use-social-media`.

[7] Misra N. Reddy I.V. The parameterized complexity of happy colorings. In: International Workshop on Combinatorial Algorithms. pp. 142–153. Springer (2017).

[8] Moody James. Race, School Integration, and Frendship Segretgation in America. — 2001. — Access mode: `https://www.journals.uchicago.edu/doi/abs/10.1086/338954`.

[9] N. R. Aravind1 Subrahmanyam Kalyanasundaram1 Anjeneya Swami Kare, Lauri Juho. Algorithms and hardness results for happy coloring problems. Springer. — Springer, 2016. — Arxiv : https://www.iith.ac.in/ subruk/pdf/happyColoring.pdf.

[10] N. R. Aravind1 Subrahmanyam Kalyanasundaram1 Anjeneya Swami Kare, Lauri Juho. Algorithms and hardness results for happy coloring problems. — 2017. — Access mode: https://arxiv.org/pdf/1705.08282.pdf.

[11] Silviu Maniu Pierre Senellart Suraj Jog. An Experimental Study of the Treewidth of Real-World Graph Data (Extended Version). ICDT 2019.

[12] Zhang P. Jiang T. Li A. Improved approximation algorithms for the maximum happy vertices and edges problems. In: International Computing and Combinatorics Conference. pp. 159–170. Springer (2015).

[13] Zhang P. Li A. Algorithmic aspects of homophyly of networks. In: Theoretical Computer Science 593, 117-131 (2015).

[14] Zhang P. Xu Y. Jiang T. Li A. Lin G. Miyano E. Improved approximation algorithms for the maximum happy vertices and edges problems. In: Algorithmica 80(5), 1412–1438 (2018).

[15] source Python library Open. https://docs.scipy.org/doc/scipy/reference/stats (requested 2020-06-09).

[16] CAngsheng Li Jiankou Li Yicheng Pan Pan Peng. Small community phenomenon in networks: mechanisms, roles and characteristics. manuscript (2012).

[17] Что такое гомофилия. — 2020. — Access mode: http://asocialnetworks.blogspot.com/2016/01/blog-post.html.