ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Государственное образовательное учреждение
высшего образования
«Московский физико-технический институт»
(Государственный университет)

Кафедра проблем физики и астрофизики

Выпускная квалификационная работа на степень бакалавра

«Способ классификации планарных графов»

студент 226 группы Слободсков И.О.
Научный руководитель:
канд. физ.-мат. наук Шейнкман О. В.

Москва, 2016

Содержание

Введение

Структура взаимосвязей между объектами может быть представлена в виде графа, что позволяет использовать общие универсальные подходы и математические методы для их анализа.

В данной работе рассматривается способ сравнения и классификации планарных графов, который учитывает только локальную структуру связей между вершинами и не зависит от размеров сравниваемых графов.

Для планарных графов исследуется способ сопоставления вершинам множества признаков, их преобразования с помощью методов математической морфологии и построения функции на основе распределения выбранных признаков по вершинам для сравнения графов.

Для построения алгоритма распознавания предлагается использование обучающей, предварительно классифицированной выборки.

1 Постановка задачи

Задача состоит в том, чтобы на основе обучающей выборки данных построить функцию, сопоставляющую каждый граф с вектором чисел. С помощью такого признакового описания в дальнейшем классифицировать графы.

Используются следующие предположения:

1. Графы планарные (можно изобразить на плоскости без пересечения рёбер).

2. Графы одного класса:

   (a) обладают схожей локальной структурой

   (b) могут быть различного размера.

3. Признаковое описание не должно явно зависеть от размера графа.

Математическая морфология используется потому, что её преобразования являются довольно простыми и при этом могут быть удобными для некоторых типов задач.

1.1 Краткое описание метода

Планарный граф является разреженным - у большинства вершин количество соседей ≈ 6. Можно по некоторым правилам раскрасить граф - сопоставить каждой вершине определённый цвет (число), затем применить последовательность преобразований, перекрашивающих граф. В каждом преобразовании новый цвет каждой вершины вычисляется в зависимости от цвета вершины и множества цветов соседних клеток. (характеристика вершины - целое число, названа цветом для большей наглядности.)

Таким образом, если последовательность состоит из n преобразований, итоговая раскраска учитывает структуру графа вплоть до соседей n-ного порядка. На используемых практических данных было достаточно $n = 3$. Не все последовательность преобразований являются полезными, но некоторые из них способны выявить различия между исходными графами.

Итак, становится необходимым сравнить уже раскрашенные графы. Для каждого графа можно выделить кластеры вершин, обладающих общим цветом, и анализировать уже распределение количества кластеров от их размера и цвета.

Сама по себе информация о кластерах не очень удобна для сравнения графов - например, между кластером из 1 или 2 клеток есть существенная разница, в то время как между кластеры из 19 и 20 клеток практически схожи – это следует из практических соображений. Кроме того, на реальных данных может получиться кластер размером с весь граф. В этом случае сравниваемые графы необходимо считать эквивалентными, не смотря на то, что максимальный размер кластера различен.

Поэтому на основе кластеров строится некоторая дискретная функция, которая учитывает данные особенности. Из практических данных было получено, что функцию можно заменить вектором из первых m значений, так как кластеры слишком большого размера уже не несут существенной информации о структуре графа.

Полученные вектора можно анализировать и сравнивать классическими способами.

Например, для того, чтобы выбрать наиболее информативные последовательности преобразований, с помощью программы были перебраны последовательности длиной от 1 до 6. Если в результате получались вектора, схожие

для одного типа графов и различные для разных - последовательность сохранялась. То, какие последовательности окажутся наиболее информативными, зависит от характера исходных данных.

Использование только одной последовательности преобразований может привести к тому, что некоторые классы будут различаться хорошо, но некоторые - нет. Чтобы различать всё, необходимо выбрать несколько последовательностей и использовать их совокупность для дальнейшей классификации. Например, можно построить дерево принятия решений.

Данный алгоритм был опробован на биологических данных. В качестве графа выступала структура клеток в плоских листиках мха. Способ сравнения основан на предположениях о локальной структуре и независимости от размера графа, которые очень хорошо подходят в данном случае - клетки могут делиться, и процесс деления клетки, скорее всего, как-то зависит от её ближайших соседей.

Впрочем, описанный метод сравнения графов рассматривает данные исключительно с математической точки зрения, и потому может быть применён и в других областях.

2 Реализация

2.1 Общие положения

В процессе написания дипломной работы была создана программа на языке программирования Scala, с помощью которой был проведён анализ исходных данных, найдены наиболее удачные варианты преобразований графов и т.п. Схема преобразований данных представлена на рисунке (1).

В дальнейшем описании приведены формулы, подобранные с целью достижения максимальной понятности и математической простоты. Используемые в реальных вычислениях функции математически эквивалентны, но в целях производительности реализованы иначе. Например, функцию 3 нет смысла вычислять суммированием для каждой точки.

Граф $G(V, E)$ представлен как массив вершин, для каждой вершины хранится множество её соседей. В дальнейших формулах множества рёбер и вершин обозначаются как E и V или $Edges$ и $Vertices$ соответственно.

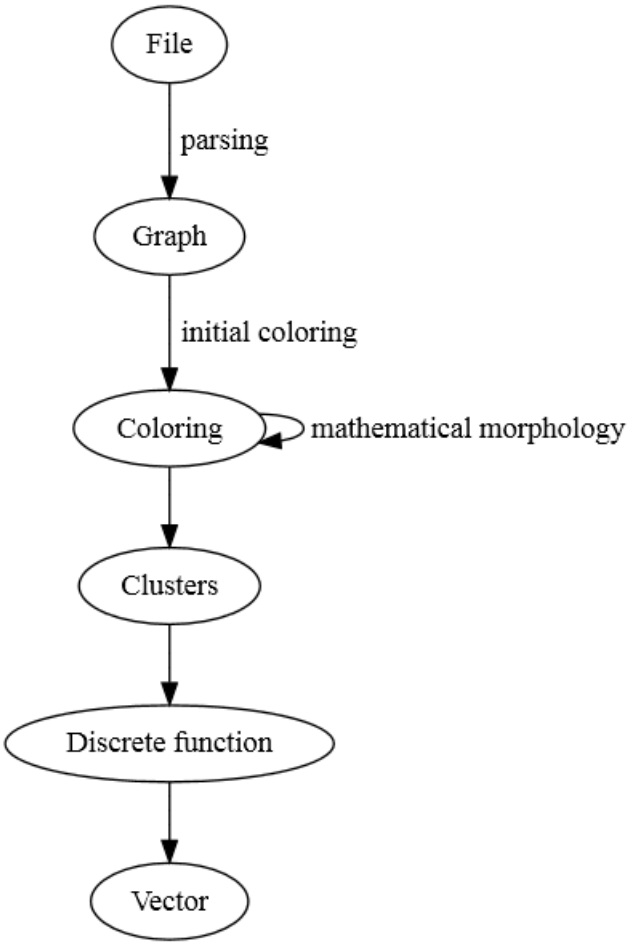

Рис. 1: Схема преобразований данных

2.2 Создание исходной раскраски

Каждой вершине сопоставляется целое число. Для удобства отладки и визуального представления происходящего можно сопоставить каждому числу цвет - в дальнейшем, при упоминании раскраски, подразумевается отображение:

$$Coloring : V \to \mathbb{Z} \tag{1}$$

В качестве исходной раскраски использовалось количество соседних вершин

$$neighborCount(v) = \sum_{e \in Edges} [v \in e] \tag{2}$$

Пример раскраски и особенности реализации содержатся в разделе 3.2 Для уменьшения количества цветов экстремальные значения (меньше 3 и больше 9 приводились к 3 и 9 соответственно).

2.3 Преобразование раскрасок (Математическая морфология)

За основу преобразования раскрасок были взяты идеи математической морфологии. Изначально мат. морфология была разработана для анализа чёрно-белых изображений (т.е., вершинам квадратной сетки сопоставлялись только два цвета).

Структура графа несколько сложнее - у вершин может быть различное количество соседей, поэтому в общем случае понятие структурного элемента и переноса не применимо. Тем не менее, можно реализовать частные случаи таких базовых операций, как opening (открытие), closing (закрытие), dilation(расширение) и erosion (сжатие).

Для вершин графа можно определить расстояние между ними, как минимальное количество рёбер, необходимое для построения пути между ними. Поэтому в качестве структурного элемента можно взять множество вершин, для которых расстояние от центральной равно нулю или единице. Этот вариант был выбран как самый простой, другие в данной работе не рассматривались.

Таким образом, есть следующие базовые операции:

1. Dilation (расширение): все вершины, соседствующие с вершинами цвета c, перекрашиваются в этот цвет.

2. Erosion (сжатие): вершины цвета c, соседствующие с вершинами другого цвета, перекрашиваются в другой цвет.

3. Opening (открытие): последовательное применение сжатия и расширения.

4. Closing (закрытие): аналогично, применение расширения и вслед за ним - сжатия.

В случае двух цветов a и b расширение цвета a эквивалентно сжатию цвета b.

Кроме того, операцию dilation можно обобщить на раскраски с произвольным количеством цветов. Сжатие обобщить нельзя - не понятно, в какой из цветов перекрашивать клетки.

Но исходная раскраска (количество соседей) содержит несколько цветов! В данной работе использовался следующий способ:

1. К раскраске из нескольких цветов применялось до двух операций расшерения цветов.

2. Раскраска преобразовывалась в раскраску из двух цветов

3. Для раскраски их двух цветов использовалось до трёх операций типа расширения или сжатия.

Для бинаризации использовалась следующая функции:

$$newColor(baseColors, color) : \begin{cases} color \in baseColors \to 1 \\ color \notin baseColors \to 0 \end{cases}$$

Количество способов выбрать подмножество цветов, довольно велико (для n цветов будет 2^n вариантов). Чтобы не перебирать все из них, из соображений здравого смысла были выбраны самые простые варианты:

1. Множество baseColors состоит из одного или двух цветов.

2. Множество содержит значения, меньшие некоторого числа, например $\{1, 2, 3\}$.

Рассматривать вариант, когда во множестве содержатся значения, большие некоторого числа n, не имеет смысла – это эквивалентно пункту 2 и даст раскраску с инвертированными цветами.

2.4 Кластеризация

После раскраски можно выделить кластеры (множества соприкасающихся клеток одного цвета) на графе. В дальнейшем строится распределение количества кластеров от их размера и цвета: $clustersCount(color, size)$

К сожалению, сравнивать напрямую распределение размеров кластеров для различных графов нет смысла по следующим причинам:

1. Исходные графы могут быть достаточно малыми (например, количество вершин < 200), и распределение кластеров по размерам будет довольно некрасивой функцией с большим количеством нулевых значений.

2. Размер графа не должен влиять на результат сравнения. Графы с отличающимся в разы количеством вершин могут быть схожи по структуре. Если в раскрасках значительно преобладает какой-либо цвет, он образует кластер размером с граф - размеры больших кластеров не должны сильно влиять на результаты сравнения.

3. Для кластеров малого размера точное значение этого размера является важным.

4. Чем больше клеток определённого цвета, тем больше должен быть вклад этого цвета в функцию сравнения.

2.5 Более удобное представление информации о кластерах

Итак, необходимо построить функцию, у которой бы не было недостатков, описанных в разделе 2.4. Данная функция не должна сильно менять свой

вид при небольшом изменении размеров кластеров и выбрана следующим образом:

$$f(color, size) \equiv \frac{\sum_{i=size}^{\infty} i * clustersCount(color, i)}{\sum_{c \in Colors} \sum_{i=1}^{\infty} i * clustersCount(c, i)}. \quad (3)$$

Множитель в знаменателе можно считать нормировочным. Необходим для выполнения условия $\sum_{c \in Colors} f(c, 1) = 1$.

Функция (3) может быть представлена в следующем виде:

$$f(color, size) \equiv \frac{\sum_{v \in V} [cluster(v).size >= size]}{|V|}. \quad (4)$$

Из определения (3) видны следующие свойства функции:

1. $f(c, s) \geqslant 0$.
2. $f(c, s) \leqslant 1$.
3. $f(c, s)$ монотонно убывает при увеличении s и фиксированном c.
4. $\lim_{s \to \infty} f(c, s) = 0 \ \forall c$.
5. В явном виде содержит информацию, сколько вершин окрашены в данный цвет:
$$f(color, 1) = \sum_{v \in V} [Coloring(v) = color]. \quad (5)$$
6. $\sum_{c \in Colors} f(c, 1) = 1$.

Итак, эта функция не зависит от размера графа, а так же не имеет недостатков, которые были у функции распределения размеров кластеров: например, если у большого кластера окажется на одну клетку больше, функция почти не изменит свой вид.

Эта функция уже достаточно хороша для сравнения. Например, можно ввести расстояние между раскрасками двух различных графов:

$$distance = \sqrt{\sum_{c \in Colors} \sum_{s=1}^{\infty} \left(f_1(c, s) - f_2(c, s)\right)^2 e^{-ks}}. \quad (6)$$

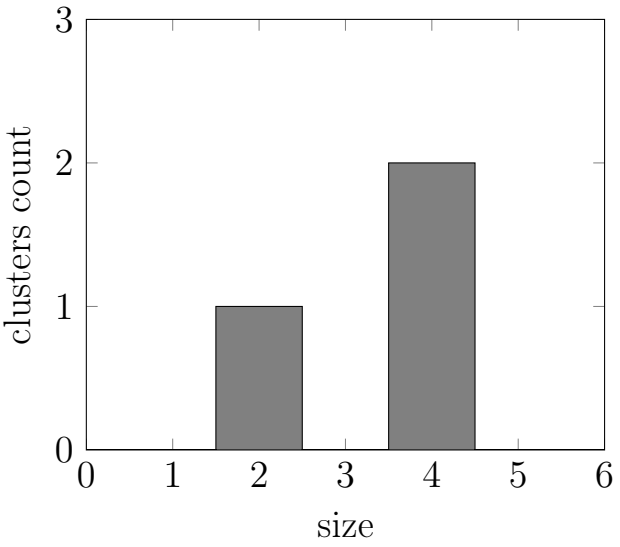

Рис. 2: Простейший пример функции распределения кластеров

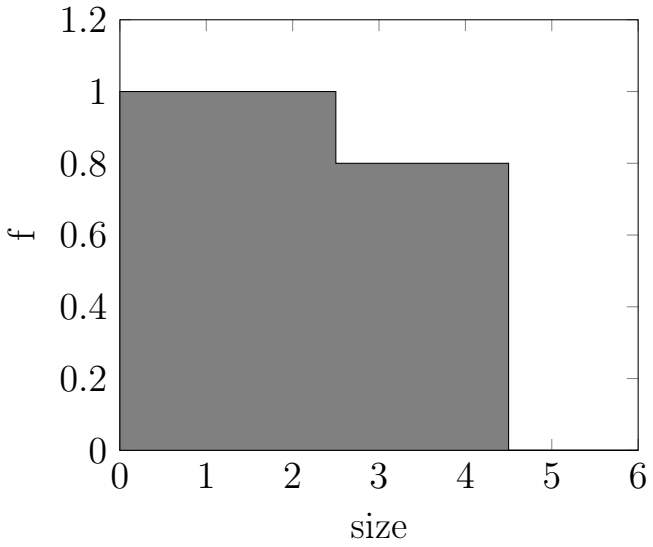

Рис. 3: Вид сопоставляемой функции f

Параметр k выбирается так, чтобы сделать более важной разницу между размерами маленьких кластеров.

Что важно, если в раскраске мало клеток цвета c – например, 0.01, то $f(c, s) <= 0.01 \ \forall s$, и разница в клетках этого цвета не может существенно повлиять на функцию расстояния.

2.6 Сравнение векторов

Как можно заметить в формуле 6, экспоненциальный множитель k делает бессмысленным вычисление функции f для больших значений s, поэтому функцию можно заменить вектором из первых n значений: $n = n(k)$.

На экспериментальных данных получилось, что при раскраске графов обычно доминирует какой-нибудь один цвет (например, 6, если граф имеет структуру сот).

Итак, при помощи последовательности преобразований и последующих действий можно получить вектор их нескольких чисел, описывающих граф, а так же метрику для них.

Вообще говоря, не обязательно использовать экспоненциальный множитель, можно подставить туда любую убывающую функцию в зависимости от особенностей входных данных.

2.7 Кластеризация результатов

Для того, чтобы оценить, какую информацию выдаёт та или иная последовательность преобразований (сопоставляющая каждому графу вектор), можно использовать кластеризацию.

В лучшем случае все объекты будут сразу разделены по классам. К сожалению, на практических данных настолько хорошей последовательности не было найдено.

В процессе работы были опробованы два алгоритма кластеризации:

2.7.1 WPGMA

(Weighted Pair Group Method with Arithmetic Mean)

Довольно простой метод кластеризации:

1. Берётся матрица расстояний между элементами.

2. Ищется пара наиболее близких элементов. Пусть это будут (e_1, e_2).

3. Два этих элемента удаляются.

4. Вставляется новый элемент - пара, расстояния считаются по формуле:

$$distance(pair(e_1, e_2), e_n) = \frac{distance(e_1, e_n) + distance(e_2, e_n)}{2}. \quad (7)$$

5. Алгоритм повторяется до тех пор, пока не останется один элемент.

2.7.2 Neighbor Joining

Так же был опробован более современный метод: Neighbor Joining. В общем он похож на WPGMA, но отличается критерием для поиска пары элементов и формулой для нахождения расстояния от пары элементов до остальных.

1. Берётся матрица $D(i, j)$ расстояний между элементами i, j.

2. На основе этой матрицы вводится матрица Q:

$$Q(i, j) = (n - 2) * D(i, j) - \left(\sum_{k=1}^{n} D(k, j) + \sum_{k=1}^{n} D(i, k) \right). \quad (8)$$

3. Ищется пара $p = Pair(i, j)$ наиболее близких элементов: с наименьшим $Q(i, j)$.

4. Расстояния до пары определяются следующей формулой:

$$D(p, k) = \frac{1}{2}(D(p, i) + D(p, j) - D(i, j)). \quad (9)$$

5. Элементы i и j больше не участвуют в расчётах, вместо них добавляется элемент p.

6. Алгоритм повторяется до тех пор, пока количество элементов больше двух.

Так же в исходном алгоритме находятся расстояния от элементов i, j до $p = Pair(i, j)$ в виде:

$$D(i,p) = \frac{1}{2}\left(D(i,j) + \frac{1}{n-2}\left(\sum_{k=1}^{n} D(i,k) - \sum_{k=1}^{n} D(j,k)\right)\right) \tag{10}$$

Расстояния объектов до их пары не влияют на дальнейшие итерации, поэтому формула приведена отдельно от алгоритма.

К сожалению, ни первый, ни второй алгоритмы кластеризации не дали интересных результатов, в лучшем случае получался кластер объектов одного класса, в то время как остальные были разобросаны довольно хаотично.

2.8 Линейный классификатор

Итак, каждый алгоритм преобразований сопоставляет объекту вектор (точку в n-мерном пространстве). Если алгоритм выбран достаточно удачно, можно попробовать найти гиперплостость так, чтобы объекты классов A и B находились с различных её сторон. В таком случае для распознавания нового объекта надо будет скалярно умножить соответствующий ему вектор на нормаль к гиперплоскости и сравнить результат с некоторым пороговым значением.

В данной задаче больше двух классов для распознавания, поэтому придётся использовать несколько классификаторов.

Метод построения гиперплоскости для пары классов A и B: в качестве начального приближения в качестве нормали берётся разность для средних значений. Методом градиентного спуска данный результат улучшается. В качестве критерия оценки выбрано отношение суммы среднеквадратических отклонений от средних значений скалярных произведений к разнице между средними значениями для классов A и B.

Если на обучающей выборке для алгоритма преобразований не удалось найти плоскости, разделяющей какую-нибудь пару классов, то этот алгоритм не используется. В данной работе из 29070 алгоритмов было выбрано около 250 наилучших, которые использовались для построения классификаторов.

3 Практические результаты

В качестве исходных данных использовалась информация о взаимном расположении клеток в листьях мха. Эти листья являются плоскими - можно построить планарный граф, где каждой клетке листика соответствует вершина графа, и вершины являются соседними, если клетки имеют общую клеточную стенку.

3.1 Формат входных данных

В исходных файлах в текстовом виде содержится следующая информация:

1. Для каждой клеточной стенки:
   (a) уникальный номер - id
   (b) id двух клеток (между которыми находится эта стенка)
   (c) id двух точек
2. Для каждой точки (не путать с вершинами, эти точки - места пересечения стенок):
   (a) уникальный номер - id
   (b) координаты x, y
   (c) число стенок, которым принадлежит эта точка

В дальнейшем на основе этих данных строился дуальный граф, где вершинами являются клетки. Количество клеток в графе от 100 до 6000. Самих файлов не очень много, поэтому искусственные нейронные сети не использовались (хотя в общем, бо̀льшая часть метода применима и к ним, используемые подходы можно использовать для предварительной обработки данных).

3.2 Создание *.svg изображений

Человеческий глаз очень хорошо распознаёт повторяющиеся паттерны в изображении, поэтому есть смысл в визуализации графов. Для удобства от-

ладки и наглядного представления промежуточных результатов была реализована возможность сохранить раскраску графа в виде изображения.

Формат svg выбран в связи с тем, что он векторный - граф естественным образом описывается в виде рёбер и вершин - итоговое изображение занимает небольшой объём и отображается при любом масштабе.

При визуализации как раз рисуются клеточные стенки. Клетки закрашиваются цветами, которые тем или образом сопоставлены вершинам дуального графа.

Сопоставленные вершинам целые числа преобразуются в цвета по следующему правилу:

$$\begin{cases} 0 \to white \\ 1 \to grey \\ 2, 3 \to red \\ 4 \to orange \\ 5 \to yellow \\ 6 \to green \\ 7 \to deepskyblue \\ > 7 \to blue \end{cases} \tag{11}$$

Как видно на рис. 4, большинство клеток зелёного цвета, так как эти клетки имеют по шесть соседей.

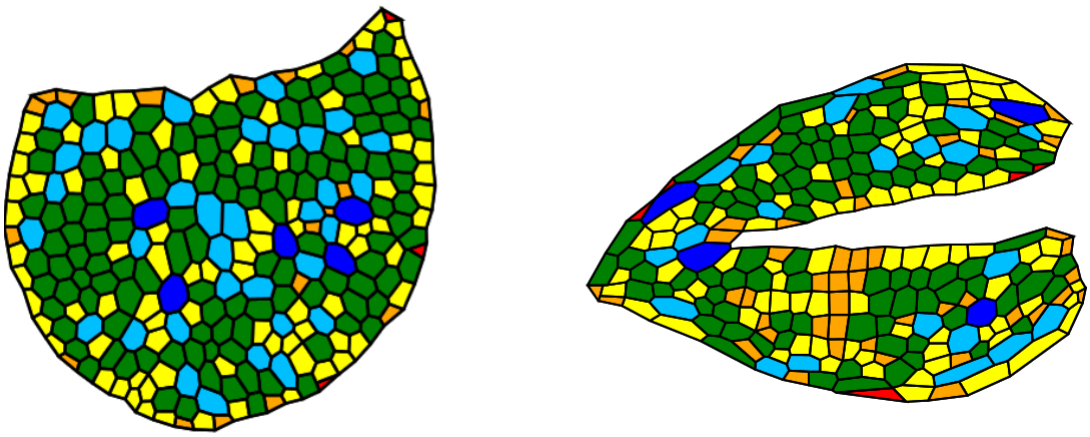

Рис. 4: Пример изначальных раскрасок

3.3 Выбор последовательности преобразований

Некоторые методы перекрашивания могут выявить различия между структурой графов. Например, на рис. 4 различия для двух образцов не очень заметны. (Если бы не было цветов, они были бы незаметны вовсе).

На рис. 5 показан результат применения преобразования "расширение цвета 4(оранжевого)". Различие между первым и вторым графом становится более явным.

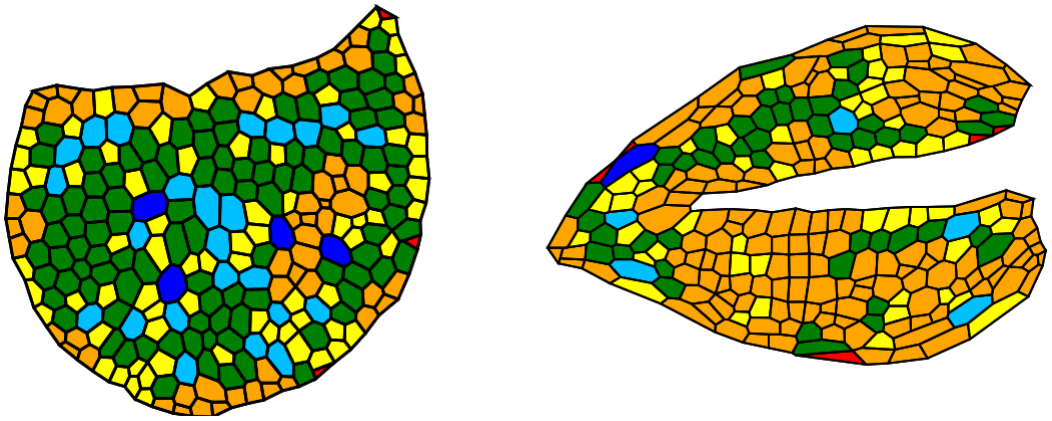

Рис. 5: Преобразование начальных раскрасок "расширением"оранжевого цвета

Эффективность можно повысить при применении последовательности из нескольких преобразований цветов (необязательно одних и тех же)

Очевидно, что значительная часть последовательностей преобразований графа является бесполезной. Например, применение операции «расширение зелёного цвета» за 1-2 раза может залить все клетки зелёным цветом и с точки зрения алгоритма все графы будут одинаковы.

Важной задачей является нахождение именно таких последовательностей преобразований, которые дают раскраски, схожие для листьев одного вида и различных для разных.

Есть большое количество уже распознанных данных - на них можно проверить данные и выбрать наиболее информативные.

Написанная программа перебрала различные последовательности преобразований, на основе лучших последовательностей была сделана функция сравнения.

3.4 Результат

К сожалению, данный метод хорошо распознаёт только часть классов. Например, на тестовых данных для «Hypopterygium flavolimbatum» одиночная проверка с вероятностью 5-10 % относит его к другому классу и с вероятностью 3-10 % ошибочно относит объект другого класса к данному. Ещё можно отметить «Brium Moravicum»: около 20 % и 10 % ошибок соответственно.

4 Заключение

В данной работе описан возможный метод классификации планарных графов на основе анализа результатов морфологических преобразований. Разработана программа, с помощью которой производились и анализировались промежуточные действия и алгоритмы, а так же проверена работа метода на реальных данных.

В качестве данных использовалась структура клеток в плоских листьях мха, распознаваемые классы - биологические роды и виды.

Получены следующие результаты:

1. Проведён анализ графов, содержащих от 100 до 6000 вершин. Методы математической морфологии оказались достаточно удобными для решения задачи, алгоритмы работы с ними просты в реализации и обладают вычислительной сложностью $O(n)$ или $O(n \log n)$

2. Проанализовано подмножество функций, сопоставлющее объектам признаковое описание в виде векторов чисел.

   (a) Методы кластеризации, использующие расстояния между этими векторами, оказались не подходящими для классификации в целом и могут использоваться только для предварительного анализа функций.

   (b) Установлено, что одна последовательность преобразований графа не даёт достататочного количества информации для распознавания большого числа классов, но может применяться для разбиения объектов на два подмножества с помощью линейного классификатора.

(c) Установлено, что достаточно эффективную классификацию можно произвести на основе дерева принятия решений, которое использует результаты нескольких преобразований. Количество преобразований в лучшем случае $O(\log ClassesCount)$, в худшем $O(ClassesCount)$.

3. На обучающей выборке найдены наиболее удачные последовательности преобразований, и на основе их комбинации построено дерево принятия решений, относящее объекты к соответствующим классам.

4. Программа позволяет взять новые данные и частично распознать роды и виды листьев мха по структуре клеток, что может использоваться для предварительной обработки и классификации.

Список литературы

[1] Sokal R and Michener C (1958). «A statistical method for evaluating systematic relationships». University of Kansas Science Bulletin 38: 1409–1438.

[2] Martin Simonsen, Thomas Mailund, Christian N. S. Pedersen (2008). «Rapid Neighbour Joining» Proceedings of WABI 5251: 113–122. doi:10.1007/978-3-540-87361-7_10.

[3] «Лекции по линейным алгоритмам классификации» К. В. Воронцов

[4] Isabelle Bloch «Mathematical Morphology», CNRS UMR 5141 LTCI.

[5] S. Dasgupta, C. H. Papadimitriou, and U. V. Vazirani «Algorithms», July 18, 2006.

[6] Graham, Knuth, Patashnik «Concrete Mathematics» Addison-Wesley, 1994

[7] Robert Sedgewick, Kevin Wayne «Algorithms», Addison-Wesley, 2011.

[8] G. Sapiro, R. Kimmel, D. Shaked, B. Kimia, and A. M. Bruckstein. «Implementing continuous-scale morphology via curve evolution.» Pattern Recognition, 26(9):1363-1372, 1993.