Московский государственный университет имени М.В.Ломоносова
Факультет вычислительной математики и кибернетики
Кафедра алгоритмических языков

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

Автоматическое извлечение гиперонимов из больших текстовых корпусов.

Автор:
группа 425
Шавалиева Ралина Забировна

Научный руководитель:
должность???
Лукашевич Наталья Валентиновна

Москва, 2018

Содержание

Введение

В наше время компьютерная обработка естественного языка является одной из самых востребованных областей искусственного интеллекта. Для того, чтобы правильно распознать смысл фразы, написанной человеком, необходимо иметь дополнительные знания о каждом слове этого предложения и существующих связей между ними. К такой вспомогательной информации относится отношение *is-a*, что означает отношение обобщения. Способность к обобщению лежит в основе человеческого познания. Люди без труда могут заменить частное на общее. Например, слово «кошка» на слово «животное», «автомобиль» на «транспорт», «красный» на «цвет». Для каждой пары общее слово имеет термин *гипероним*, а частное - *гипоним*. Для каждого гипонима может существовать несколько гиперонимов, и наоборот.

КАРТИНКА

Способность успешного распознавания таких лингвистических отношений приносит вклад в такие области, как вопросно-ответная система, системы семантического поиска, управление записями (система хранения и отслеживания документов), а также информационный поиск и навигация по сайтам. Например, наличие информации, что слова «гепард» и «животное» связаны отношением *is-a*, может помочь при ответе на вопрос «какое самое быстрое животное?». А система информационного поиска сможет подобрать соответствующие сайты, при получении такого же запроса. В добавок, отношение гипоним-гипероним - это основа почти любой семантической сети и таксономии (иерархической системы отношений сущностей определенной области знаний). Наличие таких построенных структур, помогает при реферировании текста, т.е. извлечении из него основного содер-

жания или заданной информации с целью письменного изложения. Таким образом, проблема определения отношений обобщения весьма актуальна как в области лингвистики, так и в области искусственного интеллекта. На данный момент большинство ресурсов, позволяющих определять отношение обобщения, написаны вручную, что очень дорого в плане их создания и поддержки в актуальном состоянии. Ручные словари зачастую имеют не достаточно большую область покрытия. К тому же, в большинстве случаев, нет возможности выставления вероятности наличия связи непрерывной случайной величиной от 0 до 1, а лишь только дискретная степень допустимости существования отношения. Например, 0 - нет связи, 1- слабая, 2 - средняя, 3 - сильная. Таким образом, нет возможности определить, какое из слов в качестве гиперонима подходит больше, если есть несколько кандидатов имеющих одинаковую степень. Для современных задач, появляется необходимость создания автоматической системы поиска связи гипоним-гипероним. Существует два основных вида постановки задачи. Первая, наиболее распространенная, это сопоставление каждой паре слов 1 или 0, в зависимости от того, есть связь или нет. Такого рода задача бинарной классификации неоднократно подвергалась критике за её чрезмерную простоту, а также из-за невозможности сравнения кандидатов, как и в случае ручных словарей. Второй способ заключается в поиске гиперонимов, т. е. предоставлении ранжированного списка слов, которые наиболее вероятно являются гиперонимами заранее заданному гипониму. Поиск таких слов происходит из конкретного словаря, приближенному к словарю всех слов и устойчивых выражений для определенного языка. В данной работе исследуется второй вид постановки задачи - поиск гиперонимов. Рассматриваются различные алгоритмы, как с применением машин-

ного обучения, так и основанные на текстовых шаблонах. И производится сравнение полученных результатов по различиям метрикам качества.

1 Постановка задачи

Целью данной работы является реализация различных алгоритмов извлечения гиперонимов из текстовых корпусов. Исследовать алгоритм, основанный на анализе текстов по средством рукописных регулярных выражений, а также класс алгоритмов, использующих разные виды нейронных сетей, с последующим обучением на размеченных данных. Алгоритм должен для заданного гипонима составлять упорядоченный по вероятности список гиперонимов.

ЗАДАЧИ

1. Составить обзор существующих подходов к решению поставленной задачи
2. Выбор набора данных. Разделение его на обучающую и тестовую части.
3. Выбор метрик качества модели, наиболее точно отражающих главные критерии качества. Метрики должны учитывать порядок выбранных гиперонимов, так как основная задача - ранжирование списка.
4. Построение и тестирование модели, основанной на рукописных шаблонах
5. Исследование первой модели векторного представления слов, основанной на гипотезе дистрибутивной семантики. PMI + SVM.
6. Построение моделей, использующих различные комбинации векторов, полученных алгоритмом Word2Vec. Дообучение алгоритмом SGD и LambdaRank.

7. Реализация нейронной сети, разделяющей каждое слово на 2 вектора: слово в качестве гипонима и слово в качестве гиперонима. Распараллеливание алгоритма средствами Hadoop MapReduce.

8. Сравнение результатов, полученных всеми построенными моделями.

2 Обзор существующих алгоритмов

2.1 Модель, основанная на лексико-синтаксических шаблонах

Данный метод был разработан профессором Марти Херст в 1992 году. Алгоритм является одним из первых в области автоматического определения отношения обобщения между словами. Считается, что пара слов в предложении связана отношением гипоним- гипероним, если она удовлетворяет одному из шаблонов, таких как

- $[A]$ for example $[B]$ - например
- $[A]$ such as $[B]$ - такие как
- $[A]$ include $[B]$ - включая
- $[A]$ especially $[B]$ - особенно

и др.

Здесь $[B]$ обозначает *гипероним*, а $[B]$ - список *гипонимов*. Например, «I like flowers, such as roses or peonies» - «мне нравятся цветы, такие как розы или пионы». В данном случае в качестве гиперонима выступает слово «цветы», а гипонимы - розы и пионы. Список таких словосочетаний не имеет фиксированного размера. Можно добавлять свои примеры, подходящие конкретной области или исключать неподходящие.

ПРЕИМУЩЕСТВА

- Высокая точность

НЕДОСТАТКИ

- Главный недостаток такого подхода - низкая полнота определения связей. Необходим очень большой текстовый корпус, чтобы выделить хотя бы базовые пары отношений.

- Не каждый пример связи можно описать шаблоном.

- Не улавливаются цепочки связей. То есть, если определены пары «ласточка - птица» и «птица - животное», то может не быть пары «ласточка - животное».

- Несмотря на высокую точность, встречаются случаи, когда пара слов неверно отмечена, как имеющая связь is-a. Один из таких примеров пара предложений: «...**cities** in Asian countries such as **Tokyo** ...» - «... города в странах Азии, такие как Токио ...» «...cities in Asian **countries** such as **Japan** ...» - «... города в странах Азии, таких как Япония ...» В первом случае пара «countries - Tokyo» была бы отмечена неправильно

- Для ранжирования гиперонимов не достаточно иметь только число, означающее сколько раз встретилась конкретная пара, так как это зависит от конкретного текстового корпуса.

2.2 Векторное представление слов

2.2.1 Представление слов векторами, полученными при SVM разложении матрицы PPMI

Данный подход основывается на предположении, которое носит название «дистрибутивная гипотеза»: лингвистические единицы, встречающиеся в схожих контекстах, имеют близкие значения. Для такого подхода необходимо иметь достаточно большой текстовый корпус. Существует несколь-

ко способов получение контекстов для слова. Наиболее популярный называется «window-based» метод. Задается величина ширины окна k. Далее просматриваются все предложения, содержащие конкретное слово. К примеру, интересующее нас слово W_i находится в позиции i в рассматриваемом предложении. Контекстом данного предложения к W_i будет набор слов $(W_{i-k}, \dots W_{i-1}, W_{i+1}, \dots W_{i+k})$ т.е k слов, стоящих до W_i и k слов после. Величина окна произвольная, но чаще всего выбирается от 2 до 5. Таким образом, для каждого слова строится набор его контекстов. Исследовав схожесть наборов контекстов двух разных слов, можно судить об отношении этих слов между собой. Например, в случае синонимов, наборы контекстов будут близки. Задание же функции вычисления схожести является главным параметром таких моделей. PPMI (positive pointwise mutual information) - положительная поточечная взаимная информация. Функция PPMI является одной из оценок схожести двух лингвистических единиц, например слов, контекстов, абзацев и т.п., на основе заданного текстового корпуса. Для случая вычисления взаимной информации между словом и контекстом данная функция задается следующей формулой:

$PPMI(w, c) = max(PMI(w, c), 0)$

$PMI(w, c) = \log \frac{p(w,c)}{p(w) \times p(c)})$, где

- w - слово из текстового корпуса;
- c - контекст;
- $p(w)$ - вероятность встречи слова w в корпусе = частота появления слова в корпусе деленное на общее число слов
- $p(c)$ - вероятность встречи данного контекста = частота появления контекста в корпусе деленное на общее число контекстов

- $p(w, c)$ - вероятность встречи пары «слово - контекст» = частота появления данной пары в корпусе деленное на общее число пар

Таким образом, PPMI вычисляется напрямую из текстового корпуса. Если слово и контекст не связаны между собой, то

$p(w, c) = p(w) \times p(c)$

и PPMI для этой пары будет равно 0. Чем выше данный показатель, тем чаще слово w появляется в сопровождении контекста $\mathbf{c}$. Строится таблица M, строки которой соответствуют словам, а столбцы - контекстам. Таблица заполняется соответствующими значениями PPMI. Вектор слова определяется величинами, расположенными в его строке. Длина вектора - количество найденных в корпусе контекстов. Таким образом, схожесть наборов контекстов для двух слов определяется близостью их векторов. Как можно заметить, величина встречаемости слова намного меньше размера множества всех контекстов. Значит, матрица M будет иметь большой размер, и при этом она будет сильно разряженной. Возникает проблема «проклятия размерности». Измерение расстояния между такими векторами будет неинформативным, т.к согласно Закону Больших Чисел, сумма n слагаемых стремится к некоторому фиксированному пределу при

$n \to \infty$,

следовательно, расстояния во всех парах объектов стремятся к одному и тому же значению. Одним из решений данной проблемы является снижение размерности алгоритмом SVD (Singular Value Decomposition). Данный алгоритм может приблизить матрицу M размера $n \times m$, некоторой другой матрицей M_k с заданным рангом k, такой, что её можно разложить в произведение трех других матриц.

$M \approx M_k = U_k \times \Sigma_k \times V_k^T$,

где U_k и V_k - две унитарные матрицы, состоящие из левых и правых сингулярных векторов соответственно, а V_k^T - это сопряжённо-транспонированная матрица к V_k

Σ_k — матрица размера $k \times k$ с неотрицательными элементами, у которой элементы, лежащие на главной диагонали — это сингулярные числа (а все элементы, не лежащие на главной диагонали, являются нулевыми)

Полученная матрица U_k будет иметь размерность $n \times k$. Строки такой матрицы будут соответствовать строкам исходной матрицы M, только размерность векторов изменится с m на k. Часть информации потеряется, но существенная её часть остается. Аналогично, матрица V_k^T несет информацию о столбцах матрицы M, только имея при этом меньший размер. Применительно к нашей задачи, разложение матрицы PPMI приведет к получению компактного векторного представления слов, сохраняющего информацию о контекстах.

Дальнейшая работа с полученными векторами будет описана в пункте 2.4.

2.2.2 WORD2VEC

Еще одним инструментом, реализующим модель векторного представления слов, является Word2Vec. Этот инструмент был разработан группой исследователей Google в 2013 году, под руководством Томаша Миколова. Word2Vec реализует две архитектуры - Continuous Bag-of-Words (CBOW) и Skip-gram. Обе архитектуры построены на нейронных сетях и основаны на дистрибутивной гипотезе. Принцип работы CBOW — предсказывание слова при данном контексте, а Skip-gram наоборот — предсказывает контекст при заданном слове. Независимо от архитектуры, модель принимает

в качестве входных параметров текстовый корпус, и формирует векторное представление каждого слова, входящего в корпус и имеющего частотность в заданном диапазоне. Применяется искусственная нейронная сеть прямого распространения (Feedforward Neural Network) с функцией активации иерархический софтмакс (Hierarchical Softmax) и/или негативное сэмплирование (Negative Sampling). Метрика близости векторов – косинусное расстояние. Таким образом, данная модель позволяет получить еще одно векторное представление каждого слова.

2.2.3 Dynamic distance-margin model

Две предыдущие модели позволили получить векторные представления слов, основываясь на наборах их контекстов. При этом каждое слово учитывалось только один раз, не было различия: слово выступает в качестве гипонима или в качестве гиперонима. Отношение is-a не является симметричным. Если пара слов $A - B$ связана отношением гипоним-гипероним, то пара $B - A$ таким отношением уже не связана. Более того, чем ближе вектор A к вектору B, тем больше вероятность, что слова и являются синонимами. Необходимо подбирать границы близости: не слишком далекие, чтобы слова имели связь друг с другом, и не слишком близкие, чтобы не получать синонимичные пары. С точки зрения построения моделей, задача поиска максимума или минимума решается проще и качественнее, чем определение таких границ. Поэтому возникает предположение о разделение одного вектора слова на два - вектор гипоним и вектор гипероним.

Вектор слова w, используемый, когда данное слово будет выступать в качестве гипЕронима, обозначим за $E(w)$. Например, слово *птица* в отношении (**ласточка**, **птица**). Когда в качестве гипОнима - за $O(w)$. Пример:

в паре (*птица*, *животное*).

Предполагается, что построенная новая модель должна удовлетворять следующим трем правилам:

1. Если пара слов $u - v$ связана отношением гипоним-гипероним, то вектора $O(u)$ и $E(v)$ находятся близко друг к другу. $O(u) \approx E(v)$

2. Если пара слов $u - v$ являются ко-гипонимами, т.е. существует слово w, являющееся гиперонимом, как для слова u, так и для слова v, то должно выполняться соотношение $O(u) \approx O(v)$

3. Аналогично, если пара слов является ко-гиперонимами, то должно быть верно: $(u) \approx (v)$

Таким образом задача сводится к минимизации расстояний $O(u) \approx E(v)$, $O(u) \approx O(v)$ и $(u) \approx (v)$. В качестве обучающего множества берется набор триплетов (u, v, q), где u - гипоним, v - гипероним, а q - сколько раз пара гипоним-гипероним (u, v) встретилась в текстовом корпусе. Такой набор данных может быть собран вручную или, например, методом шаблонов, описанным в пункте 2.1. Необходимо учитывать, что для получения хороших результатов, размер текстового корпуса должен быть очень большим.

Для каждой пары $x = (u, v, q)$ из выбранного набора данных, вычисляется функция расстояния между векторами $O(u)$ и $E(v)$. В качестве функции расстояния может, например, выступать 1-норма разности векторов. Так

$f(x) = ||O(u) - E(v)||_1$

Для того, чтобы оценить величину ошибки модели на примере x, расстояние между векторами слов u и v сравнивается с расстоянием от u до произвольного слова v'. Т.е выбирается негативный пример гиперонима для

гипонима u. По предположению, построенные вектора $O(u)$ и $E(v)$ должны быть ближе, чем вектора $O(u)$ и $E(v')$. Насколько ближе они должны быть, определяется величинамиq и q'. Величина q' означает, сколько раз пара гипоним-гипероним (u, v') встретилась в текстовом корпусе. Так как слово v выбирается случайным из всего корпуса, то наиболее вероятно q равняется нулю.

3 Структура алгоритма

Поставленная задача не является простой, особенно для графов, имеющих большое количество вершин. Для ее решения граф удобно представить в виде матрицы смежности. Пусть, для удобства, элементы этой матрицы состоят из единиц и нулей. Если на пересечении строки i и столбца j стоит $1(0)$, и $i \neq j$ - это означает, что существует (не существует) ребро (i,j), если $i = j$, вершина под номером i имеет (не имеет) петлю. При отображении одной вершины в другую, в матрице смежности меняются местами строки и столбцы соответствующих вершин. Автоморфизмом является следующее отображение: $\widehat{g}A\widehat{g}^{-1} = A$, где $\widehat{g}$ - матричный вид подстановки g. Польза такого представления графа заключается в том, что для поиска автоморфизма g, являющегося отображением множества вершин графа G на себя, достаточно проверить равенство элементов матрицы смежности до и после применения подстановки g.

Идея алгоритма заключается в построении последовательности множеств частичных отображений по матрице смежности графа. Последнее множество из этой последовательности будет содержать в себе все автоморфизмы графа.

Для описания построения такой последовательности будут использоваться следующие определения и обозначения.

3.1 Проверка по критерию

Критерием h является проверка подматриц на равенство. Предположим, требуется определить, удовлетворяет ли элемент g_k^s критерию h. Для этого необходимо, чтобы элементы a_{ij}, для всех $i, j \leq k$, были равны соответ-

ствующим элементам $a_{r_i r_j}$. Подматрица g_k^s выглядит так:

$$\begin{array}{c} \\ r_1 \\ r_2 \\ \vdots \\ r_k \end{array} \begin{array}{c} \begin{array}{cccc} r_1 & r_2 & \dots & r_k \end{array} \\ \begin{pmatrix} a_{r_1 r_1} & a_{r_1 r_2} & \dots & a_{r_1 r_k} \\ a_{r_2 r_1} & a'_{r_2 r_2} & \dots & a_{r_2 r_k} \\ \vdots & \vdots & \ddots & \vdots \\ a_{r_k r_1} & a_{r_k r_2} & \dots & a_{r_k r_k} \end{pmatrix} \end{array}$$

Если хотя бы одно из равенств не выполнено, это означает, что по данной частичной подстановке хотя бы одна вершина отобразилась в другую так, что структура графа изменилась. А так как частичная перестановка g_k^s фиксирует $r_1, \dots r_k$, то структура останется измененой, что означает отображение не будет являться автоморфизмом.

3.2 Построение множеств частичных отображений

Описание построения M'_n.

На первом этапе рассматриваются все g_1^s, образующие множество M_1. Каждый элемент g_1^s проверяется по критерию h. Элементы, удовлетворяющие h, образуют M'_1. В каждый элемент из M'_1 добавляется еще одно отображение $(2 \to r_2)$, т.о. происходит переход от g_1^s к g_2^s. Из каждого g_1^s получается разных $(n-1)$ элементов g_2^s. Всевозможные элементы g_2^s образуют M_2. Далее строится M'_2, добавляя в него только те элементы из M_2, которые удовлетворяют критерию. Аналогично получаются множества $M_3,\ M'_3,\ M_4, \dots, M_n,\ M'_n$.

Таким образом, получается последовательнось множеств частичных отображенией:

$\{M'_k\}$: $M'_1 \subseteq M'_2 \subseteq \dots \subseteq M'_n$.

Множество M'_n является множеством всех возможных автоморфизмов (или пустое, если их нет).

4 Исследование алгоритма

4.1 Определение класса решаемых задач

Предложенный алгоритм является универсальным. Используя различный формат вводимых данных, можно решать следующие задачи:

1. Нахождение группы автоморфизмов произвольной квадратной матрицы.
2. Нахождение левой и правой групп автоморфизомв произваольной матрицы (т.е. блок-схем, дискретных функций и т.д.).
3. Нахождение всех изоморфизмов матрицы A на матрицу B.
4. Нахождение всех изоморфных вложений матрицы A в матрицу B.
5. Поиск клик.
6. Решение задачи Коши в подстановках.
7. Нахождение группы автоморфизмов матрицы Адамара.
8. Нахождение всех изомофризмов кода A на код B.

В рамках дипломной работы рассматриваются первые шесть задач.

4.2 Анализ

Алгоритм применим для любых графов, но для удобство рассматриваются случайные графы, матрицы смежности которых состоят из нулей и едениц. В этом случае получена оценка сложности алгоритма в среднем.

Так как граф случайный, элементы матрицы смежности графа равны 1 или 0 с вероятностью $\frac{1}{2}$. Необходимо вычислить наиболее вероятные размеры для каждого множества M'_k.

Если рассмотреть построение множеств, не учитывая промежуточного критерия, то на k-ом шаге множество увеличивается в $(n-k)$ раз (так как для каждого элемента $g^s_{(k-1)} = (r_1, \ldots, r_{k-1})$ добавляется r_k из оставшихся $(n-k)$ вариантов): $|M_{k+1}| = |M_k|(n-k) = n(n-1)\ldots(n-k)$.

С учетом критерия получается:

1. Из построения множества M'_1 следует $|M'_1| \approx \frac{|M_1|}{2}$ (элемент $a_{11} = a_{r_1 r_1}$ с вероятностью $\frac{1}{2}$).

2. На $(k+1)$-ом шаге требуется совпадение $((k+1)+(k+1)-1)$ элементов. Так как граф случайный (матрица с равновероятным распределением 0 и 1), то на этом шаге $|M'_{k+1}| = \frac{1}{2^{2m+1}}|M_k|$ элементов.

Учитывая построение $\{M'_k\}$ и пункты 1, 2 получим
$|M'_{k+1}| \approx \frac{n!}{(n-k-1)!\ 2^{(k+1)^2}} \approx \frac{1}{e^{k+1}} \frac{n^{n+1/2}}{(n-k-1)^{(n-k-1/2)}\ 2^{(k+1)^2}}$. Последнее равенство получено используя формулу Стирлинга: $n! \approx \sqrt{2\pi n}(\frac{n}{e})^n$, при большом n.

Далее необходимо рассмотреть последовательность $\{|M'_{k+1}|\}$.
$|M'_{k+1}| = \frac{n(n-1)\ldots(n-k)}{2^{(k+1)^2}} \leq \frac{n^{k+1}}{2^{(k+1)^2}}$
Равенство $n^{k+1} = 2^{(k+1)^2}$ означает, что в множестве осталось небольшое количество элементов. Данное равенство далее будет называться стабилизацией последовательности $\{M'_k\}$, а k - значением стабилизации.

После решения данного равенства, получается, что стабилизация наступает при $k \approx \log_2(n)$. Это означает, что с вероятностью близкой к единице (для случайного графа) уже на $k = \lceil \log_2(n) \rceil$ шаге мы обнаружим отсутствие или наличие автоморфизма (в статье это выдвигается как тезис).

Вычисление номера множества k в последовательности $\{|M'_k|\}$, соответствующий самому большому множеству:

Пусть $|M'_{k+1}| : |M'_{k+1}| = f(k+1)$,

$f'(k+1) = \frac{\ln(n)n^{k+1}2^{(k+1)^2} - (2k+2)\ln(2)2^{(k+1)^2}n^{k+1}}{(2^{(m+1)^2})^2}$.

Получается уравнение:

$\ln(n) - (2k+2)\ln(2) = 0 \;\Rightarrow\; k+1 \approx \frac{5}{7}\ln(n) - 1$.

Так как k и n целые, то

$k+1 = \lceil(\frac{5}{7}\ln(n) - 1)\rceil)$.

Необходимо отметить, что в тот момент, когда $k \approx \log_2(n)$, можно судить о том, существуют автоморфизмы в графе или нет. Если размер множества $M_k = 1$ (только тождественная подстановка), можно утверждать с вероятностью близкой к 1, что автоморфизмов, кроме тривиальной подстановки, не существует. Если $M_k > 1$, то, наоборот, с вероятностью близкой к 1 автоморфизм существует. Данный факт позволяет сэкономить память и уменьшить количество операций при поиске автоморфизмов в тех графах, в которых они существуют.

Опираясь на вышесказанное, получены оценки:

- значение стабилизации

  $k \approx \log_2(n)$

- номер наибольшего по количеству элементов множества

  $k \approx \frac{5}{7}\ln(n)$

- ограничение на размер множества

  $|M'_k| = \frac{n(n-1)...(n-k-1)}{2^{(k)^2}} \leq \frac{n^k}{2^{(k)^2}}$

- приблизительное количество операций в секунду:

$\frac{n^{(5/7)\ln(n)}}{2^{((5/7)\ln(n))^2}} \times n(2(\frac{5}{7}\ln(n))+1) \times \log_2(n)$

- затраты оперативной памяти:

  $2 \times \frac{n^{(5/7)\ln(n)}}{2^{((5/7)\ln(n))^2}} \times \log_2(n)$

4.3 Вероятностная сложность

Сложность всего алгоритма представляет собой сумму сложностей вычисления $M'_1 \dots M'_n$.

Количество элементов в каждом множестве (с вероятностью близкой к единице):

$|M'_k| \approx \frac{n!}{(n-k-2)!\ 2^{(k)^2}} \approx \frac{1}{e^k}\frac{n^{n+1/2}}{(n-k-2)^{(n-k-3/2)}\ 2^{(k)^2}}$

Можно оценить: $|M'_k| = \frac{n(n-1)\dots(n-k-1)}{2^{(k)^2}} \leq \frac{n^k}{2^{(k)^2}}$

Количество операций сравнения для подсчета всех множеств:

$\sum_{k=1}^{n} \frac{1}{e^k}\frac{n^{n+1/2}}{(n-k-2)^{(n-k-3/2)}\ 2^{(k)^2}} \times (n-k)(2k+1)$

Оценка (были учтены точка стабилизации и точка максимума последовательности множеств):

$n \times \frac{n^{(5/7)\ln(n)}}{2^{((5/7)\ln(n))^2}} \times n(2(\frac{5}{7}\ln(n))+1) \times \log_2(n) = O(n^2(\frac{e}{2})^{\ln(n)^2}\ln(n))$

Сложность экспоненциальная.

4.4 Теоретическая возможность распараллеливания

Каждое множество $M_i (i = 2 \dots n)$ вычисляется из M'_{i-1} путем добавления в каждый элемент (частичная перестановка, представленная вектором целых чисел) из M'_{i-1} всех возможных оставшихся чисел, которых нет в соответствующем элементе.

Для этого удобно использовать 2 не фиксированных по количеству строк двумерных массива размера t на n (t может увеличиваться), хранящих

частичные отображения, такого вида:

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1i} & ? & \dots & ? \\ a_{21} & a_{22} & \dots & a_{2i} & ? & \dots & ? \\ \dots & & & & & & \\ a_{k1} & a_{k2} & \dots & a_{ki} & ? & \dots & ? \\ ? & ? & \dots & ? & ? & \dots & ? \\ \dots & & & & & & \\ ? & ? & \dots & ? & ? & \dots & ? \end{pmatrix}$$

? - эти значения не известны и заполняются постепенно.

Один массив для M'_{i-1}, другой для вычисления из предыдущего M_i. Из каждой строки множества M'_{i-1} (первый массив), в соответствие с алгоритмом, получается n новых строк длины i, которые записываются во второй массив (расширяется, если нужно).

После этого, второй массив копируется в первый и запускается цикл по всем векторам. Вектор исключается из массива, если a_{ji} равен какому-либо числу в данном векторе или вектор с a_{ji} является частичной перестановкой, не удовлетворяющей критерию.

5 Модернизация

5.1 Оптимизация по времени работы

На основе результатов тестирования программы и анализа выяснилось, что эффективность алгоритма тесно связана с тем, как нумеруются вершины графа. Другими словами, время работы программы зависит от того, какой матрицей смежности (из многих) представляется граф.

Модернизация заключается в том, что на каждом этапе можно требовать, чтобы мощность множества частичных отображений была минимальна. Однако представить матрицу смежности нужным образом не представляется возможным при больших размерах, и время работы, затрачеваемой на это, превышает время работы алгоритма. Например, при $n = 100$ потребуется всего лишь 100 операций, чтобы выяснить, с какой вершины эффективнее всего начать отсчет на первой итерации. Но для того, чтобы получить выгоду на второй итерации, уже требуется более 10 000 операций [3]. Поэтому оптимальным является уменьшить только начальное множество частичных отображений.

Для получения первого множества, мощность которого будет наименьшей, в изначальной матрице меняется порядок строк/столбцов (переименование вершин), а именно, необходимо поменять строки и соответствующие им столбцы таким образом, чтобы на месте первого элемента главной диагонали стояло то значение, которое встречается меньше всех других на диагонали. То есть, если на главной диагонали 70% нулей и 30% единиц, необходимо поместить на первую позицию единицу. Эффективность данной модификации исходит из того, что первая итерация алгоритма составляет множество из тех номеров строк (столбцов), в которых значение

на главной диагонали совпадает с первым элементом главной диагонали. Значит, выбрав на эту позицию наименьший по количеству встречаний на диагонали элемент, получается наименьшее по мощности множество.

Оценки получены для случайных матриц (вероятность нуля и единицы в каждой позиции одинакова и равна $\frac{1}{2}$). Предположим, что в матрице $n \times n$ на главной диагонали находится k нулей, где $k \leq \frac{1}{2}n$ (если количество нулей больше половины, то за k обозначается количество единиц). Тогда, если на первое место главной диагонали выбирать элемент случайным образом, получим, что математическое ожидание размера полученного множества будет $\frac{k}{n} * k + \frac{n-k}{n} * (n-k)$. В модифицированном алгоритме всегда будет получаться k. Таким образом, улучшение составляет $\frac{\frac{k}{n}*k+\frac{(n-k)}{n}*(n-k)}{k} = 2\frac{k}{n} + \frac{n}{k} - 2$ раз. В случае диагонали, состоящей из одних нулей (единиц), то есть k = 0, улучшение равняется 1 (мнощьность множеств одинакова). Для получения полной оценки необходимо посчитать матожидание улучшения для произвольного числа нулей и единиц на главной диагонали. Вероятноть k нулей составляет $\frac{C_n^k}{2^n}$. Тогда матожидание улучшения $\frac{1}{2^{n-1}} + 2\sum_{i=1}^{\lceil \frac{n}{2} \rceil} \frac{C_n^k}{2^n} * (2\frac{k}{n} + \frac{n}{k} - 2)$. Далее в таблице приведены значения для различных n:

Количество вершин графа	Коэф. уменьшения мощности
4	2.12500
8	2.11198
14	1.69565
20	1.51723
50	1.27697
100	1.18312
200	1.12400

5.2 Обработка особых случаев

5.3 Ресурс параллелизма

Для поиска автоморфизмов матрицы порядка n итерационным алгоритмом в параллельном варианте требуется выполнить:

* n ярусов сравнений (количество сравнений $k = 1 \dots n : \frac{1}{e^k} \frac{n^{n+1/2}}{(n-k-2)^{(n-k-3/2)}\ 2^{(k)^2}} \times (n-k)(2k+1)$)

При классификации по высоте ЯПФ, таким образом, алгоритм имеет сложность $O(n)$.

При классификации по ширине ЯПФ его сложность $O(n(\frac{e}{2})^{\ln(n)^2} \ln(n)))$.

6 Практическое применение

Заключение

В результате данной работы разработан модернезированный итерационный алгоритм поиска автоморфизмов и изоморфизмов графов, основанный на алгоритме, предложеном в статье [1]. Получены оценки сложности алгоритма при больших размерах графа (количество вершин $n > 1000$).

Реализована программа, решающая поставленную задачу на персональном компьютере, при $n \leq 500$, а также вычислены значения затрачиваемых ресурсов, при работе программы на суперкомпьютере (при $n \leq 10^4$).

Список литературы

[1] *Егоров В.Н., Егоров А.В.* Группы автоморфизмов и изоморфизм комбинаторных объектов и алгебраических структур // *Ломоносовские чтения, Научная конференция, Москва, факультет ВМК МГУ имени М.В.Ломоносова.* — 2016.

[2] *Егоров В.Н., Марков А.И.* О гипотезе Адама для графов с циркулянтными матрицами смежности вершин ДАН СССР // *Доклады Академии наук.* — 1979. — Т. 249, № 3. — С. 529–532.

[3] *Luks Eugene M.* Isomorphism of graphs of bounded valence can be tested in polynomial time // *Journal of Computer and System Sciences.* — 1982. — Pp. 25:42–64.

[4] *Рейнгольд Э., Нивергельт Ю., Део Ю.* Комбинаторыне алгоритмы. Теория и практика. — Мир, Москва, 1980.

[5] *Babai Laszld.* On the complexity of canonical labeling of strongly regular graphs // *SIAM, J. Comput. 9(1).* — 1980.

[6] *Goldberg M.K.* A nonfactorial algorithm for testing isomorphism of two graphs // *Discrete Appl. Math., 6.* — 1983. — P. 229–236.

[7] *Егоров В.Н.* О группах автоморфизмов матриц // *Прикладная дискретная математика.* — 2010.

[8] Алгоритмы: построение и анализ. / Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн.

[9] *Воеводин В.В.* Математические основы параллельных вычислений. — Изд. Моск. ун-та, 1991.

[10] *Воеводин В.В., Воеводин Вл.В.* Параллельные вычисления. — БХВ - Петербург, 2002.

[11] Практика суперкомпьютера «Ломоносов» / Вл. Воеводин, С. Жуматий, С. Соболев и др. // *Открытые системы.* — 2012.

Приложение А

Приложение Б

Ссылка на дипломную работу с программой на github: https://github.com/fullincome/university

Схема реализации представлена на языке C++

```
#define GRAPH_SIZE 300
#define ROW_ARR_SIZE 10000
#define COL_ARR_SIZE GRAPH_SIZE

const int N = GRAPH_SIZE;

vector<array<unsigned short,COL_ARR_SIZE>>
Mi(ROW_ARR_SIZE, COL_ARR_SIZE);
vector<array<unsigned short,COL_ARR_SIZE>>
M'i(ROW_ARR_SIZE, COL_ARR_SIZE);
unsigned char **matrix;

struct_M'i make_M'i(Mi) {
    struct_M'i M'i;
    for (auto x: Mi) {
        for (int i = 0; i < n; ++i) {
            if (check_h(x) && !eq(x, i)) {
                M'i.push_back(x.add(i));
            }
        }
    }
```

```
    return M'i;
}

struct_Mi make_Mi(M'i) {
    struct_Mi Mi;
    for (auto x: M'i) {
        for (int i = 0; i < n; ++i) {
            Mi.push_back(x.add(i));
        }
    }
    return Mi;
}

int main() {

    //fill matrix
    init_automorphisms(matrix);

    //parallelization of program
    switch(process) {
    case 0:
        begin = 0;
        end = stop_0;
    case 1:
        begin = start_1;
        end = stop_1;
```

```
    ...
    case N:
        begin = start_n;
        end = M_0.size();

    //main cycle
    for (i = begin; i < end; ++i) {
        Mi = make_Mi(M'i-1);
        M'i = make_M'i(Mi);
    }

    //results save (automorphisms output)
    final_automorphism(M'i);
}
```