# Выявление семантических характеристик в слабоструктурированных текстовых данных.

M∀T–MΞX, курсовая работа за 8-й семестр, *Нурк С.Ю.*
Научный руководитель: Кира Вадимовна Вяткина.

---

06 мая 2010.

# Содержание

# 1 Введение

## 1.1 О задаче

В последнее десятилетие шло активное развитие интернета, в нем появилось огромное количество разнообразной и полезной информации. Она легко доступна и понятна для человека, но трудно анализируема автоматически! Каждый сайт имеет собственную структуру, глобальные идентификаторы объектов отсутствуют, в общем полная неразбериха. Эту проблему призван решить зарождающийся semantic web, но все-таки пока до его повсеместного появления еще далеко и приходится работать с тем, что есть.

В связи со слабой структурированностью больших объемов извлекаемой из веба информации возникает ряд интересных задач, главная из которых - собственно, выявление структуры, свойств и закономерностей в полученных данных.

Опишем терминологию, которой будем придерживаться в дальнейшем.

*Признак* - некоторое свойство, которым обладает объект реального мира. Каждая предметная область обладает (характеризуется) некоторым набором признаков, используемых при описании объектов ей принадлежащих (для людей - это рост, возраст, имя и т.п.; для коктейлей - ингредиенты, вид бокала и т.п.)

*Атрибутом* будем называть обыкновенную упорядоченную пару строк (соответственно *имя* и *значение* атрибута).

*Сущностью* будем называть представление объекта реального мира в виде набора атрибутов. Набор имен атрибутов в пределах одной сущности может быть не уникальным. Порядок атрибутов не имеет значения. Мы намеренно не накладываем никаких ограничений ни на состав сущности, ни на множество возможных значений атрибутов. Это даст нам возможность представления произвольной информации.

Для текстового представления набора сущностей удобно использовать XML.

Предположим, у нас имеются сущности, отражающие некоторый набор объектов реального мира, принадлежащих одной предметной области (например, коктейли).

```
entity 1 {
    Наименование: "Пина-колада"
    Ингредиенты: кокос, сок
    Способ приготовления: взболтать
    Крепость: 15%
    Итоговый объем: 150 мл
    Страна происхождения: Бразилия
}

entity 2 {
    Наименование: "Б-52"
    Ингредиенты:
    Способ приготовления: взболтать
    Крепость: 30%
    Итоговый объем:
    Страна происхождения: Германия
}

...
```

Превиденный выше пример является примером *хорошо структурированного* набора сущностей:

1. Каждый атрибут соответствует некоторому признаку реального объекта.
2. Атрибуты с одинаковым названием соответствуют признакам одного типа.
3. Имена атрибутов являются семантически осмысленными и могут считаться названием соответствующего признака.
4. Набор имен атрибутов, образующих сущность, одинаков для каждой сущности.

Все признаки можно условно разделить на несколько групп:

1. Количественные.
2. Словарные (есть фиксированный набор значений).
3. Прочие (название, способ приготовления и т.п.).

Предположим, что никакой дополнительной информации у нас нет, есть только эти сущности.
Зададимся вопросом что мы можем узнать о предметной области (а конкретнее, о признаках, для нее характерных).
Например, мы можем посчитать количество атрибутов в любой сущности (их кол-во одинаково) и сказать сколько различных признаков предметной области нашли отражение в данных.

Также для каждого имени атрибута мы, пройдя по всем сущностям, можем найти множество соответсвующих ему значений. Таким образом мы узнаем текстовое представление различных значений соответствующего признака. Проанализировав эти значения мы легко можем понять к какой из групп относится тот или иной признак (колич., словарный).
Следующий шаг - выявление интервалов, в которых лежат количественные признаки и словарей (множества значений) для словарных признаков.
В принципе, можно было бы еще посчитать параметры распределений сущностей по значениям признака, но будем считать, что и вышеперечисленного вполне достаточно. Но это был простейший случай.
Проблема в том, что автоматически собранные данные обычно выглядят не так дружелюбно, как наш пример.
Но, прежде чем рассказывать всю правду о данных, с которыми пришлось столкнуться автору на практике, сформулируем задачу, попытка решения которой приводится в данной работе.
**Задача:** по достаточно большому набору сущностей, *выявить словарные признаки*, характерные для предметной области.
Выявить словарный признак - означает найти множество его возможных значений (словарь) и предложить адекватное название признака.

## 1.2 Проблемы реальных данных

В начале два слова о том откуда берутся данные о которых пойдет речь далее.
Как уже было сказано, в интернете есть большое количество интересной и важной информации. Естественно, есть большое количество разнообразных подходов к ее извлечению, а также алгоритмов, реализующих данные подходы.
[1]

Подробный рассказ о различных методах экстракции данных с html страниц выходит за рамки нашей темы, но чтобы понять ее актуальность, стоит сказать несколько слов о ее общих принципах.
Некоторые из алгоритмов предполагают долгий процесс настройки под конкретный источник (сайт), другие работают полностью автоматически, ориентируясь на особенности внутренней структуры html страниц (таблицы, "особенные"теги и т.п.). Естественно, по качеству и "чистоте"извлеченных данных первые оставляют конкурентов далеко позади, в

[1] Стоит упомянуть о том, что на момент написания работы, автор уже более года имел непосредственное отношение к реализации подобных алгоритмов и получению с их помощью полезных для общества результатов.

расплату за это скорость извлечения данных (учитывая время настройки), падает на порядок, а стоимость их извлечения (учитывая время, потраченное разработчиком) - на несколько порядков. Поэтому в тех случаях, когда это возможно, стараются применять методы полностью автоматического сбора. Именно поэтому очень важно уметь анализировать именно автоматически извлеченную информацию, ведь ее количество зачастую сильно превышает количество любой другой.
Теперь перейдем к бочке дегтя.
Будем постепенно ухудшать наш пример, приближая его к данным из реальной жизни, получаемым при автоматическом извлечении информации.

1. Имена атрибутов часто (не всегда) являются техническими, автоматически сгенерированными (в наших примерах такие имена будут иметь вид "Field№ где вместо № - уникальный в пределах сущности номер). Что еще хуже, даже если они являются частью контента страницы, это совершенно не означает, что они (с точки зрения семантики) имеют хоть какое-то отношение к значениям этих самых атрибутов. К чести методов извлечения, нужно сказать, что иногда им удается "угадать"адекватное название атрибута (мы воспользуемся этим в дальнейшем).

```
entity 1 {
    Наименование: "Пина-колада"
    Ингредиенты: кокос, сок
    Способ приготовления: взболтать
    Крепость: 15%
    Итоговый объем: 150 мл
    Страна происхождения: Бразилия
}

entity 2 {
    Наименование: "Б-52"
    Ингредиенты:
    Способ приготовления: взболтать
    Крепость: 30%
    Итоговый объем:
    Страна происхождения: Германия
}

...
```

2. Большое количество атрибутов сущности может вообще не иметь никакого отношения к признакам извлекаемого объекта, а быть просто частью "побочного"контента страницы (заголовки, реклама и т.п.)

```
entity 1 {
    Наименование: "Пина-колада"
    Ингредиенты: кокос, сок
    Способ приготовления: взболтать
    Крепость: 15%
    Итоговый объем: 150 мл
    Страна происхождения: Бразилия
}

entity 2 {
    Наименование: "Б-52"
    Ингредиенты:
    Способ приготовления: взболтать
    Крепость: 30%
    Итоговый объем:
    Страна происхождения: Германия
}

...
```

3. Некоторые сущности могут не нести в себе никакой полезной информации. Они являются "мусором побочным продуктом жизнедеятельности алгоритма извлечения.

```
entity 1 {
    Наименование: "Пина-колада"
    Ингредиенты: кокос, сок
    Способ приготовления: взболтать
    Крепость: 15%
    Итоговый объем: 150 мл
    Страна происхождения: Бразилия
}

entity 2 {
    Наименование: "Б-52"
    Ингредиенты:
    Способ приготовления: взболтать
    Крепость: 30%
    Итоговый объем:
    Страна происхождения: Германия
}

...
```

4. Сущности, собранные с различных источников могут очень сильно отличаться по количеству и содержанию атрибутов. При этом анализировать каждый источник по-отдельности было бы принципиально неверно, так как зачастую встречаются значения атрибутов, характерные лишь для одного источника и более нигде в данных не встречающиеся.

```
entity 1 {
    Наименование: "Пина-колада"
    Ингредиенты: кокос, сок
    Способ приготовления: взболтать
    Крепость: 15%
    Итоговый объем: 150 мл
    Страна происхождения: Бразилия
}

entity 2 {
    Наименование: "Б-52"
    Ингредиенты:
    Способ приготовления: взболтать
    Крепость: 30%
    Итоговый объем:
    Страна происхождения: Германия
}

...
```

5. Как уже было упомянуто, большое количество имен атрибутов генерируется автоматически. Легко себе представить ситуацию, в которой однотипные страницы одного сайта "сверстаны"чуть по-разному. В этом случае, в некоторые сущности, может, скажем, случайно добавиться дополнительный атрибут, который повлияет на нумерацию автоматически сгенерированных атрибутов (произойдет "сдвиг"всей нумерации или только ее части). Также подобная проблема часто случается и если значения признаков могут указываться на источнике в разном порядке. Еще один случай имеет место из-за "опциональности"некоторых признаков (их значения могут не указываться).

```
entity 1 {
    Наименование: "Пина-колада"
    Ингредиенты: кокос, сок
    Способ приготовления: взболтать
    Крепость: 15%
    Итоговый объем: 150 мл
    Страна происхождения: Бразилия
}

entity 2 {
    Наименование: "Б-52"
    Ингредиенты:
    Способ приготовления: взболтать
    Крепость: 30%
    Итоговый объем:
    Страна происхождения: Германия
}

...
```

Это далеко не полный перечень проблем, встречающихся в реальных данных, но остальные добавляют лишь некоторую головную боль в реализации описанных ниже алгоритмов, не влияя при этом на суть происходящего.
Все вышеперечисленные свойства входных данных сильно затрудняют решение задачи, делая наивный подход к анализу, описаный ранее совершенно неприменимым.

# 2 Решение проблемы

## 2.1 Тестовые данные

Прежде, чем вдавться в делати, нужно рассказать о данных, на которых производилось тестирование. Это набор сущностей в количестве 50 000+. , заданые в некотором XML формате. Эти данные были собраны с группы автоматически найденных сайтов (с использованием поисковых систем), имеющих отношение к предметной области "рестораны". Извлечение данных с сайтов также проводилось полностью автоматически. В 15 000+ различных имен атрибутов.

```
<entity id="2676793">
    <source>ekaterinburgout.ru</source>
    <attr><name>Field3</name><value>Caffe del mare</value></attr>
    <attr><name>Тип заведения:</name><value>кафе</value></attr>
    <attr><name>Адрес:</name><value>ул. Восточная, д.7/д</value></attr>
    <attr><name>Часы работы:</name><value>Ежедневно, круглосуточно.</value></attr>
    <attr><name>Кухня:</name><value>итальянская</value></attr>
    <attr><name>Кредитные карты:</name><value>Visa, MasterCard, EuroCard, Union, Maestro</value></attr>
</entity>
```

Задача остается прежней: выявить как можно больше словарных признаков, характерных для теперь уже конкретной предметной области "рестораны".

## 2.2 В двух словах

В начале кратко сформулируем этапы, на которые был разделен путь решения задачи

1. Предобработка и сбор статистики.
2. Анализ статистики.
3. Восстановление ответа.

На этапе предобработки мы, во-первых, пытаемся избавиться от побочной информации ("мусора"), а во-вторых, сокращаем пространство поиска, значительно увеличивая быстродействие алгоритма.
На втором этапе мы найдем некоторые группы слов, по возможности похожие на словари искомых признаков.
И наконец, мы восстановим ранее разбитые на отдельные слова словосочетания и попробуем предложить название для найденных признаков.
Пару "источник данных"/"имя атрибута"будем в дальнейшем называть *полем*.

## 2.3 Предобработка

В начале мы убираем из наших данных сущности и атрибуты, которые являются продуктом различных известных недочетов конкретных методов экстракции.
Затем убираем из данных те поля, которые встречаются лишь в малой доле (скажем, менее чем в четверти) сущностей с соответствующего источника. 15 000+ полей осталось 5 000+.
Теперь мы разбиваем значения всех атрибутов на слова (обрезая окончания и игнорируя союзы и предлоги) и для каждого слова считаем количество раз, которое оно всречается на том или ином источнике. Затем составляем список тех слов, которые встречаются достаточно часто на достаточно большой доле источников (скажем, не менее 10 раз на пятой части всех источников). Всего получили список из 9 000+ различных слов.
Это были шаги первичной фильтрации.
**Замечание:** сущностей много и все их загрузить в память было бы крайне расточительно, поэтому каждую сущность мы обрабатываем отдельно, сохраняя в памяти лишь некоторые промежуточные результаты.
Теперь логично было бы сделать следующее: для каждого из отобранных слов посчитать количество раз, которое оно встретилось на каждом из оставшихся полей и представить результат в виде матрицы целых неотрицательных чисел $S$ (строки соответствуют словам, а столбцы - полям). Но мы сделаем чуть хитрее, приняв во внимание некоторую специфику решаемой задачи. Заметим, что значения словарных признаков редко содержат более трех слов. Также стоит отметить, что если значений несколько (например, типов кухни у одного ресторана), то для их разделения в тексте адекватные люди обычно используют определенные знаки препинания (такие как '.', ',', ';') или союз 'и'. Поэтому в процессе подсчета, мы будем в начале разбивать значения атрибута по этим стандартным разделителям, а затем игнорировать строки, длиннее 3-х слов.
Теперь, имея матрицу $S$, мы производим вторичную фильтрацию:

1. обнуляем позиции в матрице, имеющие значения не более $n$ (чтобы отфильтровать "случайные"появления слова в том или ином поле)

2. отбрасываем столбцы, в которых встречается менее $k$ неотрицательных значений, где $k$ - ограничение на минимальный размер искомых словарей

3. отбрасываем строки, в которых встречается менее $m$ неотрицательных значений, где $m$ - ограничение на минимальное количество

источников, на которых должен указываться каждый из искомых признаков.

Чтобы не пропустить интересных закономерностей, при эксперименте все параметры принимали совсем небольшое значение, а конкретно - 3. Но даже такие слабые ограничения позволили уменьшить количество рассматриваемых полей до 786, а слов - до 1201.

## 2.4 Анализ статистики

На этом этапе нам нужно решить задачу кластеризации: разбить множество всех слов на непересекающиеся группы (кластеры), так что элементы внутри группы тесно связаны между собой, тогда как элементы из разных групп "далеки"друг от друга.
[2]

После этого останется лишь отбросить неинтересные для нас кластеры (например, кластеры, состоящие из одного элемента).
Прежде всего, для того, чтобы иметь возожность прокластеризовать набор элементов, нужно ввести либо некоторую функцию (меру) похожести (similarity measure) на множестве пар элементов. Мера похожести принимает значения от 0 до 1, при этом чем значения ближе к 1 и дальше от 0, тем элементы более схожи между собой.
В нашем случае логично вводить какие бы то ни было отношения между словами на основании информации об их использовании в тех или иных полях. У нас уже есть информация такого рода - это информация из матрицы $S$. Но перед тем как ей воспользоваться, стоит ее *нормализовать*.
Это необходимо сделать по двум причинам:

1. некоторые значения признака объективно встречаются гораздо реже, чем другие (например, мексиканская кужня менее распространена, чем европейская)

2. на некоторых источниках сущностей гораздо больше, чем на других, следовательно и значения в соответствующих столбцах на порядок больше

Есть различные способы нормализовать матрицу $S[n \times m]$, причем эффективность использования каждого из них сильно зависит от того, как именно будет затем считаться функция похожести. Вот способы нормализации, испробованные автором:

[2] "Internal homogenety and external separation"[1]

1. $N[i,j] = \frac{S[i,j]}{||S[i,]||}$
2. $N[i,j] = \frac{S[i,j]}{||S[i,]||*||S[,j]||}$
3. $N[i,j] = -\log \frac{S[i,j]}{||S[i,]||*||S[,j]||}$ [3]
4. вначале каждый элемент $S[i,j]$ домножается на $\log \frac{n}{w_j}$, где $w_j$ - количество ненулевых элементов $j$-го столбца, а затем - 1)
   [3]

Здесь $N[n \times m]$ - нормализованная матрица, $S[i,]$ - $i$-я строка, а $S[,j]$ - $j$-й столбец матрицы $S$. В качестве $||.||$ может использоваться произвольная векторная норма, но наиболее популярными являются $L_1$ ($||v|| = \sum v[i]$) и $L_2$ (евклидова $||v|| = \sqrt{\sum v[i]^2}$) нормы.
Перед нами все еще стоит серьезный выбор: выбор способа подсчета меры похожести для строк нормализованной матрицы. Их можно разделить на 2 класса: работающие непосредственно с веществennозначными векторами и работающие с бинарными векторами (можно трактовать как множества). Для перехода от вектора к множеству используем некоторое пороговое значение $d$ (может быть выбрано константой, а может зависеть от среднего значения ненулевых элементов матрицы).

$$v[i] := \{^{1, v[i] > d}_{0, v[i] < d}$$

Некоторые меры похожести для веществennозначных векторов:

1. Функция косинуса угла (cosine similarity measure) - $frac \sum_i x_i y_i ||x|| ||y||$ ([5])
2. Гауссово ядро (Gaussian kernel) - $e^{-||x-y||^2/\varepsilon}$, где $\varepsilon$ - некоторая константа ([4])

**Замечание 1:** здесь и далее $||.||$ - евклидова норма.
**Замечание 2:** для применения Гауссового ядра критичным шагом является предварительная евклидова нормировка векторов ($v := v/||v||$), даже если на этапе нормирования был выбран другой способ. Дело в том, что значение этой функции оказывается близко к 1 в случае сравнения векторов малой длины, вне зависимости от их направления. Но тогда возникает другой нюанс: минимальное значение функции в этом случае

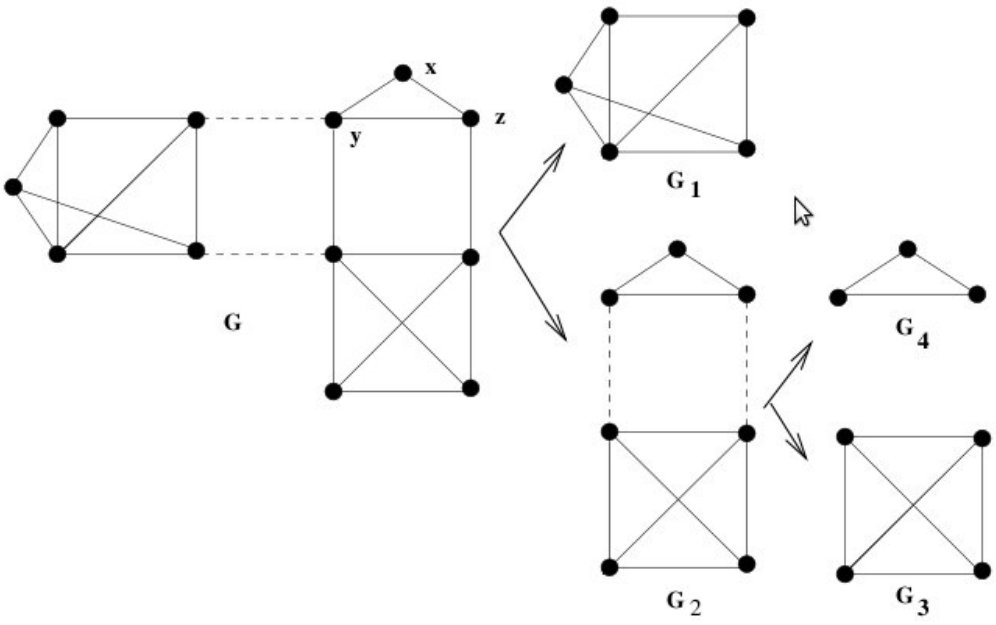
[3] полный аналог идеи использования Inverse Document Frequency из лекции 2 [5]

отстоит от 0 на величину, зависящую от размерности пространства, нужно либо учитывать это в дальнейшем, либо сдвигать и масштабировать все значения, чтобы вновь получить интервал $[0, 1]$.
Для задания меры похожести для множеств удобно использовать следующую нотацию: через $n_{11}$-количетво позиций, на которых в обоих векторах стоят единицы, через $n_{10}$ количество позиций, на которых в первом - 1, а во втором - 0 и, наконец, $n_{01}$ - аналогично. Тогда все наиболее популярные меры похожести являются частными случаями следующей формулы: $fracn_{11}n_{11} + \omega * (n_{10} + n01)$. Увеличивая $\omega$, мы увеличиваем "пенальти"за каждую позицию несовпадения. При $\omega = 1/2$, получившаяся мера носит название коэффициента Дайса, при $\omega = 1$ - Сокаля, а при $\omega = 2$ - Жаккарда.
В ходе экспериментов, было решено работать непосредственно с вещественнозначными векторами, так как это избавляет нас от необходимости выбора некоторого порогового значения, и вместе с тем повышает универсальность алогоритма. По этой же причине, а также соображениям из Замечания 2, в качестве меры похожести была выбрана именно функция косинуса.
Чтобы не задумываться об этом в дальнейшем, сразу же предподсчитаем значение меры похожести между всеми парами строк матрицы $N$ (самое время напомнить, что строка матрицы соответствует информации об использовании соответствующего слова) и запишем результаты в матрицу $A$.
Теперь перейдем к наиболее интересной части, а именно кластеризации. Из всех испробованных подходов, лучше всего себя зарекомендовал метод, основанный на статье [6].
Авторы предлагают интересный теоретико-графовый подход, позволяющий разбить данный **неориентированный** граф с **невзвешенными** ребрами на *"хорошо"* связанные подграфы (Highly Connected Subgraphs). *"Хорошо"* связанным назовем граф, степень связности которого (размер минимального множества ребер, удаление которого ведет к потери связности) $> n/2$ , где $n$ - количество вершин графа, $n > 1$.
Чтобы от графа с весами на ребрах, задаваемого матрицей $A$, перейти к невзвешенному графу, оставим в нем лишь те ребра, вес которых превосходит некоторое пороговое значение (для экспериментов использовалось значение 0.7).
В начале опишем сам алгоритм, затем перечислим свойства производимых им кластеров и, в заключении темы, поговорим о методах оптимизации алгоритма и его реализации.
*Разрезом* графа в называется множество ребер, при удалении которого нарушается связность графа. *Минимальным разрезом* называется раз-

рез, содержащий наименьшее количество ребер (не обязательно уникален).
Предположим, что у нас есть процедура MINCUT($G$), возвращающая $H$, $\bar{H}$, $C$, где $C$ - некоторый минимальный разрез, разделяющий $G$ на подграфы $H$ и $\bar{H}$.
Опишем процедуру процедуру HCS($G$)

1: $(H, \bar{H}, C) \longleftarrow$ MINCUT($G$)
2: **if** $G$ является $> n/2$ связанным **then**
3: return $G$
4: **else**
5: HCS($H$)
6: HCS($\bar{H}$)
7: **end if**

**Утверждение:** Время работы алгоритма ограниченно $2N \times f(n, m)$, где $N$ - количество итоговых кластеров, а $f(n, m)$ - время, которое требуется на поиск минимального разреза в графе с $n$ вершинами и $m$ ребрами.
На рис. показан пример работы алгоритма.




**Замечание:** В дальнейшем, мы не будем считать одиночные вершины кластерами и будем складывать их в отдельное множество.
Доказательства следующих свойств HCS кластеризации можно найти в [6]

1. *Диаметр* графа - наибольшее из расстояний между двумя произ-

вольными вершинами. Диаметр произвольного хорошо связанного графа не превосходит 2.

2. Пусть $S$ - минимальный разрез в графе $G$, $|S| \leq |V|/2$; $H, \bar{H}$ - как и раньше кампоненты связности после удаления разреза, для определенности $|V(\bar{H})| \leq |V(H)|$. Тогда если диаметр $G \leq 2$, то (1) каждая вершина $\bar{H}$ инцидентна некоторому ребру из $S$ и более того, (2) $\bar{H}$ является кликой и, если $V(\bar{H}) > 1$, то $V(\bar{H}) = |S|$.

3. ($a$) Число ребер хорошо связанного графа квадратично. ($b$) количество ребер, удаленных на каждой итерации линейно.

Свойства 1 и 3 являются свидетельством гомогенности (internal homogenity) получающихся в результате работы алгоритма кластеров. Свойства 2 и $3b$ в свою очередь свидетельствуют о том, что результирующие кластеры будут, скорее всего, хорошо отделены.
В качестве алгоритма поиска минимального разреза автором использовался алгоритм Штор-Вагнера ([2]) в реализации, работающей за $\Theta(V^3)$. Существуют и более быстрые алгоритмы для решения этой задачи , но с использованием описанных ниже эвристик, время работы алгоритма и так оказалось более чем приемлимым.
Итак, эвристики:

1. Так как нас интересуют лишь достаточно сильносвязные участки графа, то выберем некторое натуральное число $d$, после уберем из рассмотриния все вершины, степени меньше $d$. Затем запустим алгоритм, после чего выбросим из данных качественные кластера и повторим процесс с меньшим значением $d$.

2. Каждый раз, когда нам удается найти нетривиальный кластер, пребираем каждую оставшуюся, если она имеет достаточно большое количество соседей в кластере (скажем, треть от его размера), то присоединяем ее к кластеру.

# Список литературы


[1] Rui Xu Donald C.Wunsch. *Clustering*. IEEE Press, 2008.

[2] e maxx. Stoer-wagner. http://e-maxx.ru/...

[3] Priebe et al. Iterative denoising for cross-corpus discovery. In *COMPSTAT*, 2004.

[4] S. Lafon A.B. Lee. Diffusion maps and coarse-graining: A unified framework for dimensionality reduction, graph partitioning and data set parameterization. *Unknown*, todo:todo, 2006.

[5] Carnegie Mellon. Data mining lecture notes. *todo*, todo:todo, 2009.

[6] E. Hartuv R. Shamir. A clustering algorithm based on graph connectivity. *todo*, todo:todo, 1999.