



МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ М.В. ЛОМОНОСОВА  
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ  
КАФЕДРА МАТЕМАТИЧЕСКИХ МЕТОДОВ ПРОГНОЗИРОВАНИЯ

НИКИШИН ЕВГЕНИЙ СЕРГЕЕВИЧ

# Методы выделения сообществ в социальных графах

КУРСОВАЯ РАБОТА

Научный руководитель:  
д.ф-м.н., профессор  
А.Г. Дьяконов

Образец титульного брал [отсюда](#)

Москва, 2016

version 0.13

# Содержание

<b>1 Введение (неполное)</b>	<b>1</b>
1.1 Модулярность . . . . .	1
<b>2 Разбиение на непересекающиеся сообщества</b>	<b>1</b>
2.1 Edge Betweenness . . . . .	1
2.2 Label Propagation . . . . .	2
2.3 FastGreedy . . . . .	3
2.4 WalkTrap . . . . .	3
2.5 Infomap . . . . .	4
2.6 Leading Eigenvector . . . . .	5
2.7 MultiLevel . . . . .	7
<b>3 Разбиение на пересекающиеся сообщества</b>	<b>8</b>
3.1 k-Clique Perlocation . . . . .	8
3.2 BigCLAM . . . . .	8
3.3 DEMON . . . . .	10
3.4 CONGO . . . . .	10

## 1 Введение (неполное)

### 1.1 Модулярность

$$Q = \frac{1}{2m} \sum_{i,j} \left( A_{ij} - \frac{d_i d_j}{2m} \right) \delta(C_i, C_j)$$

Testtest [2, 5, 3, 4, 1]

## 2 Разбиение на непересекающиеся сообщества

### 2.1 Edge Betweenness

Для каждой пары вершин связного графа можно вычислить кратчайший путь, их соединяющий. Будем считать, что каждый такой путь имеет вес, равный  $1/N$ , где  $N$  — число возможных кратчайших путей между выбранной парой вершин. Если такие веса посчитать для всех пар вершин, то каждому ребру можно поставить в соответствие значение Edge betweenness — сумму весов путей, прошедших через это ребро.

Для ясности приведём следующую иллюстрацию:

В данном графе хочется выделить два сообщества: с вершинами 1-5 и 6-10. Граница же будет проходить через ребро, имеющее максимальный вес, 25. На этой идее и основывается алгоритм: поэтапно удаляем ребра с наибольшим весом, а оставшиеся компоненты связности объявляем сообществами.

Собственно, сам алгоритм:

1. Инициализировать веса

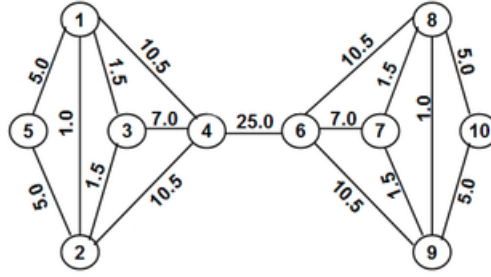


Рис. 1: Граф, для ребёр которого посчитаны значения Edge betweenness

2. Удалить ребро с наибольшим весом
3. Пересчитать веса для ребёр
4. Сообществами считаются все компоненты связности
5. Посчитать функционал модулярности (о нём будет сказано ранее)
6. Повторять с шага 2-6, пока есть рёбра

На каждой итерации процесса получается некое разбиение вершин. Последовательность таких разбиений, имеющая вид дерева, в листьях которого находятся сообщества с одной вершиной, а в корне — большое сообщество, содержащее все вершины, называется дендрограммой. Результатом работы алгоритма является ярус дендрограммы (т.е. разбиение), имеющий максимальную модулярность.

Из необходимости каждый раз пересчитывать веса следует главный минус: вычислительная сложность в худшем случае составляет  $O(m^2n)$ , где  $m$  — количество ребёр,  $n$  — количество вершин. Эксперименты показывают, что пересчитывать обычно приходится только веса для рёбер, которые были в одной компоненте связности, что несколько уменьшает сложность, однако зачастую этого оказывается недостаточно.

## 2.2 Label Propagation

Допустим, что большинство соседей какой-либо вершины принадлежат одному сообществу. Тогда, с высокой вероятностью, ему также будет принадлежать выбранная вершина. На этом предположении и строится алгоритм Label propagation: каждая вершина в графе определяется в то сообщество, которому принадлежит большинство его соседей. Если же таких сообществ несколько, то выбирается случайно одно из них. Пример:

В начальный момент времени всем вершинам ставится в соответствие отдельное сообщество. Затем происходят перераспределения сообществ. Из-за случайности важно на каждой итерации изменять порядок обхода вершин. Алгоритм заканчивает работу, когда нечего изменять: все вершины относятся к тем сообществам, что и большинство их соседей. Авторы также советуют запускать несколько раз алгоритм и выбирать наилучшее из результирующих разбиений, либо пересекать их. Главное достоинство данного алгоритма, в противовес предыдущему, — почти линейная

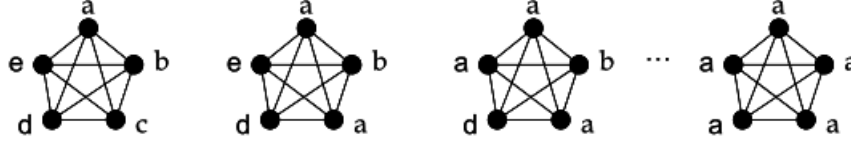


Рис. 2: Демонстрация работы алгоритма для полного графа

сложность. Однако на зашумленных графах зачастую происходит объединение всех вершин в одно сообщество.

## 2.3 FastGreedy

Алгоритм заключается в жадной оптимизации модулярности. Как и в прошлом методе, в каждой вершине графа инициализируется отдельное сообщество, а затем объединяются пары сообществ, приводящие к максимальному увеличению модулярности. При этом объединяются только инцидентные пары вершин, так как, в противном случае, модулярность не может увеличиться [во введении необходимо будет объяснить смысл модулярности, чтобы этот факт не вызывал вопросов].

Результатом работы алгоритма будет ярус дендрограммы, на котором модулярность максимальна.

Метод является вычислительно нетрудоёмким ( $O(m \log n)$ ), легко применим к большим графам и, несмотря на жадность, зачастую неплохо справляется с задачей.

## 2.4 WalkTrap

Допустим, на вершинах графа задана такая метрика, что между двумя вершинами из разных сообществ расстояние велико, а из одного — мало. Тогда выделение сообществ можно рассматривать как задачу кластеризации вершин. Попробуем ввести такую метрику, используя случайные блуждания. Объект может переместиться из вершины  $i$  в вершину  $j$  с вероятностью  $P_{ij} = \frac{A_{ij}}{d_i}$ , где  $A$  — матрица смежности,  $d_i$  — степень  $i$ . То есть на каждом шаге равновероятно выбирается "сосед" вершины  $i$ . Таким образом определяется матрица переходов  $P$  случайного блуждания. Она примечательна тем, что её степени являются вероятностями перехода из одной вершины в другую за соответствующее число шагов: вероятность перехода из  $i$  в  $j$  за  $t$  шагов равна  $(P^t)_{ij}$ . Также следует отметить, что  $P = D^{-1}A$ , где  $D$  — матрица со степенями вершин на диагонали. Используя этот аппарат можно ввести желаемую метрику на вершинах:

$$r_{ij} = \sqrt{\sum_{k=1}^n \frac{(P_{ik}^t - P_{jk}^t)^2}{d(k)}} = \|D^{-\frac{1}{2}}P_{i\bullet}^t - D^{-\frac{1}{2}}P_{j\bullet}^t\|,$$

где  $P_{i\bullet}^t$  — вектор из вероятностей перехода за  $t$  шагов из вершины  $i$  во все другие. Вообще говоря, метрика зависит от  $t$ , авторы советуют брать  $3 \leq t \leq 8$ .

Естественным образом расстояние между вершинами обобщается на расстояние

между сообществами:

$$r_{1C_2} = \|D^{-\frac{1}{2}}P_{C_1\bullet}^t - D^{-\frac{1}{2}}P_{C_2\bullet}^t\| = \sqrt{\sum_{k=1}^n \frac{(P_{C_1k}^t - P_{C_2k}^t)^2}{d(k)}},$$

где

$$P_{Cj}^t = \frac{1}{|C|} \sum_{i \in C} P_{ij}^t$$

Теперь, когда задана метрика, можно попытаться выделить кластеры в графе. Начальное разбиение — по одной вершине в каждом кластере  $\mathcal{P}_1 = \{\{v\}, v \in V\}$ . Также для всех пар инцидентных вершин считается расстояние. Далее для каждого  $k$ :

1. Выбрать  $C_1$  и  $C_2$  из  $\mathcal{P}_k$  согласно некоторому метрическому критерию.
2. Объединить два сообщества в новое  $C_3 = C_1 \cup C_2$  и обновить разбиение  $\mathcal{P}_{k+1} = (\mathcal{P}_k \setminus \{C_1, C_2\}) \cup C_3$ .
3. Обновить расстояния между инцидентными сообществами.

После  $n - 1$  шага получается дендрограмма разбиений, а  $\mathcal{P}_n = \{V\}$ . Таким образом, остался неясным только критерий выбора пар сообществ на шаге 1. Будем выбирать пару сообществ, минимизирующих приращение среднего квадратов расстояний между каждой вершиной и их сообществом при объединении сообществ. Т.е.

$$\Delta\sigma(C_1, C_2) = \frac{1}{n} \left( \sum_{i \in C_3} r_{iC_3}^2 - \sum_{i \in C_1} r_{iC_1}^2 - \sum_{i \in C_2} r_{iC_2}^2 \right) \rightarrow \min_{C_1, C_2}$$

Теперь осталось только получить результат, выбрав разбиение, на котором достигается максимума модулярность.

## 2.5 Infomap

В данном методе снова применяется подход, основанный на случайных блужданиях. У каждой вершины есть некоторая вероятность её посещения. С помощью кодов Хаффмана, в соответствии с этими вероятностями, можно закодировать путь блуждателя. Эта последовательность будет иметь некоторую длину. Однако, если использовать иерархическое кодирование (т.е. кодируем сообщество, затем кодируем вершины, попавшие в это сообщество. Коды вершин в разных группах могут совпадать), то можно сократить длину получившейся последовательности.

Как происходит иерархическое кодирование: при входе в сообщество записывается его уникальный код, затем записывается код вершины, в которую попали. Далее при переходах внутри сообщества пишутся только коды вершин. При выходе из сообщества пишется уникальный для него код выхода.

На этом и основывается метод Infomap: жадным способом минимизируется длина кода прогулки блуждателя.

Иллюстрация работы алгоритма (картинка уехала, но сейчас нет смысла править):



и перепишем *cut size* следующим образом:

$$R = \frac{1}{4} \sum_{ij} s_i s_j (d_i \delta_{ij} - A_{ij}) = \frac{1}{4} \mathbf{s}^T \mathbf{L} \mathbf{s}$$

где  $\mathbf{L}$  — матрица Лапласа с элементами

$$L_{ij} = \begin{cases} d_i, & \text{если } i = j \\ -1, & \text{если } i \neq j \text{ и между } i \text{ и } j \text{ есть ребро} \\ 0, & \text{иначе} \end{cases}$$

Далее надо отметить несколько замечательных свойств матрицы Лапласа:

1. Матрица симметричная, а, значит, собственные векторы образуют ортонормированный базис
2. Все собственные значения матрицы неотрицательны
3. Сумма по любой строке или по любому столбцу равна 0
4. Из свойств 2 и 3 следует, что всегда будет нулевое собственное значение и соответствующий ему собственный вектор  $(1, 1, 1, \dots)/\sqrt{n}$

Можно пойти дальше и ещё упростить запись *cut size*: если разложить  $\mathbf{s} = \sum_{i=1}^n a_i \mathbf{v}_i$ , где  $a_i = \mathbf{v}_i^T \mathbf{s}$ , то

$$R = \sum_i a_i \mathbf{v}_i^T \mathbf{L} \sum_j a_j \mathbf{v}_j = \sum_{ij} a_i a_j \lambda_j \delta_{ij} = \sum_i a_i^2 \lambda_i$$

Таким образом, минимизацию  $R$  можно рассматривать как выбор  $a_i^2$ , минимизирующих сумму. Как было отмечено, всегда существует собственный вектор из единиц. Если положить  $\mathbf{s} = (1, 1, 1, \dots)$ , то  $R$  становится равным нулю, что соответствует объединению всех вершин в одно сообщество. Такой тривиальный случай нас не интересует, поэтому рассматривается собственный вектор, соответствующий второму минимальному собственному значению. То есть мы будем подбирать вектор  $\mathbf{s}$  наиболее близким к  $\mathbf{v}^{(2)}$ . Учитывая ограничение, что значения  $\mathbf{s}$  могут быть только  $\pm 1$ , искомое  $\mathbf{s}$  принимает вид

$$s_i = \begin{cases} +1, & v_i^{(2)} \geq 0 \\ -1, & v_i^{(2)} < 0 \end{cases}$$

Это и есть спектральный метод в простейшем виде.

Однако авторы отмечают, что хорошее разделение — не совсем то, через которое проходит наименьшее число вершин. Поэтому они предлагают разделять те места, где количество рёбер меньше, чем ожидалось, или, наоборот, объединять те вершины, у которых количество рёбер больше, чем ожидалось.

$Q = (\text{количество вершин внутри сообщества}) - (\text{ожидаемое количество вершин})$

$$= \frac{1}{2m} \sum_{i,j} \left( A_{ij} - \frac{d_i d_j}{2m} \right) \delta(C_i, C_j) \rightarrow \max$$

Вот откуда берётся модулярность  
Используя  $\delta(C_i, C_j) = \frac{1}{2}(s_i s_j + 1)$  перепишем

$$Q = \frac{1}{4m} \sum_{i,j} \left( A_{ij} - \frac{d_i d_j}{2m} \right) (s_i s_j + 1) = \frac{1}{4m} \mathbf{s}^T \mathbf{B} \mathbf{s},$$

где  $\mathbf{B}$ , называемая матрицей модулярности, во многих смыслах похожа на матрицу Лапласа

$$B_{ij} = A_{ij} - \frac{d_i d_j}{2m}$$

И именно настраивая вектор  $\mathbf{s}$  на собственный вектор, соответствующий максимальному собственному значению, получается метод Leading Eigenvector.

$$s_i = \begin{cases} +1, & u_i^{(1)} \geq 0 \\ -1, & u_i^{(1)} < 0 \end{cases}$$

Напомним, что будет относить к сообществу 1 те вершины, у которых соответствующее значение вектора  $\mathbf{s}$  равно плюс единице, и к сообществу 2 иначе. Подобным образом граф разбивается на сообщества, пока увеличивается значение модулярности.

## 2.7 MultiLevel

Алгоритм основан на оптимизации модулярности. Как и в многих предыдущих методах, каждой вершине сначала ставится в соответствие по сообществу. Далее чередуются следующие этапы:

### 1. Первый этап

- Для каждой вершины перебираем её соседей
- Перемещаем в сообщество соседа, при котором модулярность увеличивается максимально
- Если перемещение в любое другое сообщество может только уменьшить модулярность, то вершина остаётся в своём сообществе
- Последовательно повторяем, пока какое-либо улучшение возможно

### 2. Второй этап

- Создать метаграф из сообществ-вершин. При этом рёбра будут иметь веса, равные сумме весов всех рёбер из одного сообщества в другое или внутри сообщества (т.е. будет взвешенная петля)
- Перейти на первый этап для нового графа

Алгоритм прекращает работу, когда на обоих этапах модулярность не поддаётся улучшению. Все исходные вершины, которые входят в финальную метавершину, принадлежат одному сообществу.

Несколько замечаний:



- На первом этапе вершина может рассматриваться несколько раз
- Порядок перебора не сильно влияет на точность, однако может существенно влиять на время работы алгоритма
- На практике оказывается достаточно 3-4 итераций

Для ясности приведём иллюстрацию общей схемы работы алгоритма

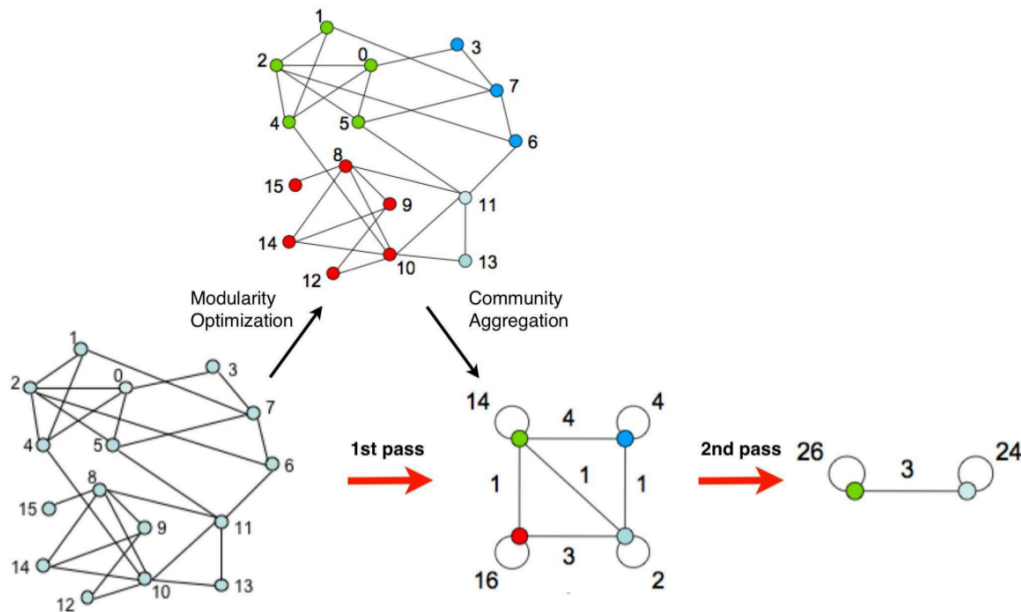


Рис. 4: Два прохода алгоритма. Для первого показаны оба этапа

## 3 Разбиение на пересекающиеся сообщества

### 3.1 k-Clique Perlocation

Clique perlocation method (CPM) основан на предположении, что сообщества состоят из пересекающихся полных подграфов. Алгоритм начинает работу с поиска всех клик размера  $k$ , после чего строится новый граф, вершинами которого являются найденные клики. Ребро образуется в случае, если пересечение вершин-клик состоит из  $k - 1$  вершины исходного графа. Компоненты связности нового графа и будут определять найденные сообщества. Эксперименты показывают, что  $k$  хорошо брать в пределах от 3 до 6. Метод хорош своей интуитивностью, однако неприменим на графах с очень большим количеством вершин.

### 3.2 BigCLAM

Cluster Affiliation Model for Big Networks — вероятностная генеративная модель, сводящая задачу выделения сообществ к задаче неотрицательной матричной факторизации. Для начала немного изменим исходную постановку: теперь у нас будет

двудольный граф, в одной доле которого находятся сообщества, а в другой — вершины, причём каждая вершина  $u \in V$  не просто принадлежит сообществу  $c \in C$ , а принадлежит ему с каким-то неотрицательным весом  $F_{uc}$  (если не принадлежит, то вес равен нулю):

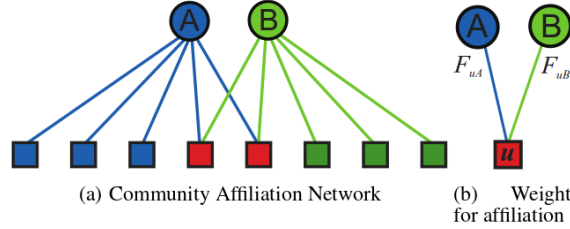


Рис. 5: Двудольный граф принадлежности. Рёбра с нулевыми весами не отображаются

Тогда, для заданной матрицы весов  $F$  предполагается, что каждое сообщество соединяет вершины  $u$  и  $v$  с вероятностью  $1 - \exp(-F_{uc} \cdot F_{vc})$ . Для всех сообществ вероятность ребра между  $u$  и  $v$  равна  $1 - \exp(-\sum_c F_{uc} \cdot F_{vc})$  или, в краткой форме,

$$p(u, v) = 1 - \exp(-F_u \cdot F_v^T).$$

Теперь можно воспользоваться методом максимизации правдоподобия: для заданного графа  $G(V, E)$  будем стараться найти  $K$  сообществ, при которых  $\hat{F} \in \mathbb{R}^{N \times K}$  доставляет максимум правдоподобия:

$$\hat{F} = \arg \max_{F: F_{uc} \geq 0} l(F)$$

$$l(F) = \log P(G|F) = \sum_{(u,v) \in E} \log(1 - \exp(-F_u \cdot F_v^T)) - \sum_{(u,v) \notin E} F_u \cdot F_v^T$$

Эта задача оптимизации и сводится к неотрицательной факторизации матрицы смежности  $A$  графа  $G$ . Определять  $K$  предлагается по качеству на отложенной выборке.

В итоге получается матрица  $F$  степеней принадлежности вершин сообществам. Результирующая принадлежность (принадлежит/не принадлежит) определяется отсечением по порогу.

Осталось лишь описать поиск  $F$ . Будем использовать блочно-координатный градиентный метод. Предлагается обновлять каждое  $F_u$  при фиксированных остальных  $F_v$ , то есть обновлять принадлежности конкретной вершины при фиксированных принадлежностях других. Главная причина фиксации: задача становится выпуклой. То есть, для каждой вершины  $u$  решается вспомогательная задача:

$$\hat{F}_u = \arg \max_{F_{uc} \geq 0} l(F_u),$$

где

$$l(F_u) = \sum_{v \in \mathcal{N}(u)} \log(1 - \exp(-F_u \cdot F_v^T)) - \sum_{v \notin \mathcal{N}(u)} F_u \cdot F_v^T,$$

где  $\mathcal{N}(u)$  — множество соседей вершины  $u$ . Именно благодаря суммированию только по соседям метод получается очень масштабируемым: реальные социальные графы больших размеров очень разреженные, то есть вершины имеют малое по сравнению с количеством вершин в графе количество соседей, вследствие чего каждое обновление имеет близкую к константе вычислительную сложность.

### 3.3 DEMON

Алгоритм Democratic Estimate of the Modular Organization of a Network является обобщением метода Label Propagation, описанного ранее. Сначала для каждой вершины  $v$  строится эго-сеть: выбирается подграф, вершинами которого являются соседи  $v$ , а рёбрами — все рёбра между всеми соседями  $v$ . Далее, для данной эго-сети запускается Label Propagation, в результате работы которого получается некоторое разбиение  $\mathcal{C}(v)$  на сообщества соседей  $v$ . После этого его необходимо объединить с итоговым покрытием  $\mathcal{C}$ , которое инициализируется пустым. Опишем, как происходит объединение. Два сообщества  $I$  и  $J$  объединяются в том и только в том случае, если не более  $\varepsilon$  процентов меньшего из них не содержится в большем из них. Например, для  $\varepsilon = 0$  объединение будет происходить только, когда одно из сообществ полностью содержится в другом, а для  $\varepsilon = 1$  объединение будет происходить всегда. Теперь можем описать сам алгоритм:

1. Инициализировать  $\mathcal{C} = \emptyset$
2. Для вершины  $v \in V$  построить эго-сеть и получить её разбиение  $\mathcal{C}(v)$  с помощью Label Propagation
3. Для каждого из сообществ в  $\mathcal{C}(v)$  и для каждого из сообществ в  $\mathcal{C}$  произвести объединение с заданным порогом.
4. Повторять шаги 2-4, пока есть нерассмотренные вершины.

### 3.4 CONGO

Cluster-Overlap Newman Girvan Optimized algorithm также является обобщением ранее описанного метода, а именно Edge Betweenness. Автор вводит дополнительную операцию разбиения вершины для того, чтобы результатом было разбиение на пересекающиеся сообщества. Раньше каждому ребру ставилось в соответствие значение edge betweenness, с помощью которого происходило последовательное удаление рёбер (удалялось ребро с максимальным edge betweenness, после чего значения пересчитывались). Теперь дополнительно каждой вершине будем ставить в соответствие величину split betweenness. Представим, что вершину  $v$  заменили на  $v1$  и  $v2$ . Тогда split betweenness вершины  $v$  будет равен количеству кратчайших путей, проходящих через виртуальное ребро между  $v1$  и  $v2$ . При этом смежные с  $v$  рёбра делятся между  $v1$  и  $v2$  таким образом, чтобы величина split betweenness была максимальной для  $v$ . Пример:

Далее последовательно повторяются удаления рёбер и разбиения вершин. Однако автор отмечает, что после каждой такой операции пересчитывать значения split и edge betweenness очень дорого, так как приходится "проходить" по всему графу. Поэтому предлагается пересчитывать значения betweenness только для путей,

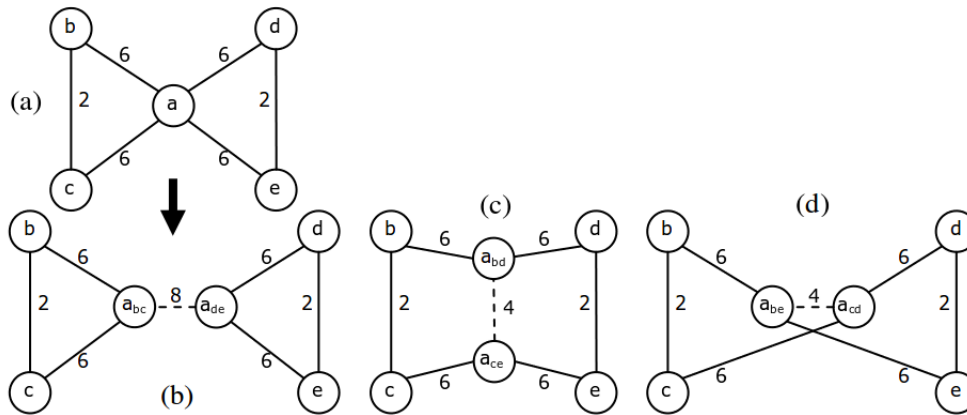


Рис. 6: (a) Граф. (b) Лучшее разбиение вершины a. (c), (d) Другие разбиения.

длина которых не превосходит некоторого  $h$ , являющегося параметром алгоритма.  
 Итоговый алгоритм:

1. Посчитать все edge betweenness для рёбер и split betweenness для вершин
2. Найти ребро или вершину с максимальным значением betweenness
3. Удалить выбранное ребро или разбить выбранную вершину
4. Пересчитать величины betweenness в  $h$ -окрестности данного ребра или данной вершины
5. Повторять шаги 2-5, пока есть рёбра

2do: разобраться с русской кодировкой и правильным оформлением списка литературы  
 2do: замечания в блокноте

## Список литературы

- [1] Clauset, A. Finding community structure in very large networks / Aaron Clauset, M. E. J. Newman, Cristopher Moore // Physical Review E. — 2004. — <http://arxiv.org/abs/cond-mat/0408187>.
- [2] Girvan, M. Community structure in social and biological networks / Michelle Girvan, M. E. J. Newman // Proceedings of the National Academy of Sciences. — 2001. — <http://arxiv.org/abs/cond-mat/0112110>.
- [3] igraph library. — 2016. — <http://igraph.org/python/>.

- [4] Raghavan, U. N. Near linear time algorithm to detect community structures in large-scale networks / Usha Nandini Raghavan, Reka Albert, Soundar Kumara // Physical Review E. — 2007. — <http://arxiv.org/abs/0709.2938>.
- [5] Slavnov, K. A. Social graph analysis. — 2015. — [http://www.machinelearning.ru/wiki/images/6/60/2015\\_417\\_SlavnovKA.pdf](http://www.machinelearning.ru/wiki/images/6/60/2015_417_SlavnovKA.pdf).