Адаптивные рандомизированные алгоритмы выделения сообществ в графах

Т. П. Проданов
Санкт-Петербургский Государственный Университет
timofey.prodanov@gmail.com

Последние семнадцать лет развивается изучение сложных сетей и большое применение находит выделение тесно связанных групп узлов, или сообществ, в сложных сетях.
Эффективными оказались рандомизированные алгоритмы выделения сообществ, однако не существует набора параметров, при котором эти алгоритмы давали бы хороший результат на всех сложных сетях. Для решения этой проблемы в работе описывается применение алгоритма одновременно возмущаемой стохастической аппроксимации к рандомизированным алгоритмам для создания приспосабливающихся к входным данным адаптивных модификаций. Целью является создание алгоритмов, дающих хорошие значения на большем количестве сложных сетей.

Ключевые слова: выделение сообществ, сложные сети, стохастическая аппроксимация.

1. Введение

Исторически изучение сетей происходило в рамках теории графов, которая начала свое существование с решения Леонардом Эйлером задачи о кенигсбергских мостах [**?**]. В 1920-х взял свое начало анализ социальных сетей [**?**]. От изучения маленьких сетей внимание переходило к сетям из тысяч или миллионов узлов, развивались методы статистического анализа сетей. К примеру, теория массового обслуживания [**?**], рассматривающая в том числе сети запросов к телефонным станциям, использовала для описания потоков запросов распределение Пуассона.

Лишь последние двадцать лет развивается изучение *сложных сетей*, то есть сетей с неправильной, сложной структурой, в некоторых случаях рассматривают динамически меняющиеся сложные сети. Сложные сети с успехом были применены в таких разных областях, как эпидемиология [**?**], биоинформатика [**?**], поиск преступников [**?**], социология [**?**], изучение структуры и топологии интернета [**?**, **?**] и в многих других.

Типичной характеристикой узла сети является его *степень*, определяемая как количество ребер, выходящих из узла. В процессе изучения сложных сетей, построенных по реальным системам [?, ?, ?, ?, ?, ?], оказалось, что распределение степеней $P(s)$, определенное как доля узлов со степенью s среди всех узлов графа, сильно отличается от распределения Пуассона, которое ожидается для случайных графов. Также сети, построенные по реальным системам характеризуются короткими путями между любыми двумя узлами и большим количеством маленьких циклов [**?**]. Это показывает, что модели, предложенные теорией графов, не всегда будут хорошо работать для графов, построенных по указанным выше реальным системам.

Общее свойство для рассматриваемых в [?, ?, ?, ?, ?, ?] сетей является наличие *сообществ*. Под сообществами можно понимать такие группы узлов графа, что внутри каждой группы соединений между узлами много, а соединений между узлами разных групп мало. Например, тесно связанные группы узлов в социальных сетях представляют людей, принадлежащих социальным сообществам, плотно сплоченные группы узлов в интернете соответствуют страницам, посвященным распространенным темам [?]. Сообщества в сетях, описывающих взаимодействия между генами, связаны с функциональными модулями [**?**]. Способность находить и анализировать подобные группы узлов предоставляет большие возможности в изучении реальных систем, представленных с помощью сложных сетей. Поиск подобных групп узлов называют *выделением сообществ* в графах, или *кластеризацией*.

В [**?**] был предложен рандомизированный жадный алгоритм выделения сообществ на графах, а в [**?**] была представлена схема кластеризации основных групп графа. От выбора параметров этих алгоритмов выделения сообществ критически зависит качество их работы, и остается открытым вопрос об адаптивных версиях алгоритмов, которые были бы работоспособны на большем количестве задач. В выпускной квалификационной работе предлагаются новые версии алгоритмов, который этот вопрос в некоторой степени решают. Так же предложенные алгоритмы лучше решают проблему меняющегося во времени работы оптимального параметра.

Работа устроена следующим образом: в разделе **??** рассмотрена необходимая информация о графах и сложных сетях, существу-

ющие методы выделения сообществ и одновременно возмущаемая стохастическая аппроксимация. Затем в разделе **??** представлен адаптивный рандомизированный жадный алгоритм, его анализ и сравнение с рандомизированным жадным алгоритмом. В разделе **??** предложена адаптивная схема кластеризации основных групп графа, рассмотрены ее параметры и результаты сопоставлены с результатами неадаптивной схемы кластеризации основных групп графа.

2. Предварительные сведения

Определения и обозначения

Формально сложная сеть может быть представлена с помощью графа. В работе будут рассматриваться только невзвешенные неориентированные графы. Неориентированный невзвешенный граф $G = (\mathscr{N}, \mathscr{L})$ состоит из двух множеств — множества $\mathscr{N} \neq \emptyset$, элементы которого называются *узлами* или *вершинами* графа, и множества $\mathscr{L}$ неупорядоченных пар из множества $\mathscr{N}$, элементы которого называются *ребрами* или *связями*. Мощности множеств $\mathscr{N}$ и $\mathscr{L}$ равны N и L соответственно.

Узел обычно обозначают по его порядковому месту i в множестве $\mathscr{N}$, а ребро, соединяющее пару узлов i и j обозначают l_{ij}. Узлы, между которыми есть ребро называются *смежными*. Степенью узла назовают величину s_i, равную количеству ребер, выходящих узла i.

Пусть $G = (\mathscr{N}, \mathscr{L})$ — граф, *разбиением на сообщества* будем называть разбиение множества его вершин $P = \{C_1, \ldots, C_K\}$, то есть $\bigcup_{i=1}^{K} C_i = \mathscr{N}$ и $C_i \cap C_j = \emptyset \ \forall i \neq j \in 1..K$.

Множество сообществ $P = \{C_1, \ldots, C_{K_1}\}$ будем называеть разбиением на сообщества *на основе* разбиения $\widetilde{P} = \left\{\widetilde{C}_1, \ldots \widetilde{C}_{K_2}\right\}$, если $\forall i \in 1..K_2 \ \exists j \in 1..K_1 : \widetilde{C}_i \subset C_j$.

Нефорально, сообщество — это тесно сплоченное подмножество узлов графа $\mathscr{N}' \subset \mathscr{N}$. Два сообщества называются *смежными*, если существует ребро, направленное из вершины первого сообщества в вершину второго.

Модулярность

В 2004 году в [**?**] была введена целевая функция *модулярность*, оценивающая неслучайность разбиения графа на сообщества.

Допустим, имеется K сообществ, тогда *нормированная матрица смежности сообществ* $\mathbf{e}$ определяется как симметричная матрица размером $K \times K$, где элементы e_{ij} равны отношению количества ребер, которые идут из сообщества i в сообщество j, к полному количеству ребер в графе (ребра l_{mn} и l_{nm} считаются различными, где m, n — узлы). След этой матрицы $\mathrm{Tr}(\mathbf{e}) = \sum_{i \in 1..K} e_{ii}$ показывает отношение ребер в сети, которые соединяют узлы одного и того же сообщества, к полному количеству ребер в графе. Хорошее разбиение на сообщества должно иметь высокое значение следа.

Однако если поместить все вершины в одно сообщество — след примет максимальное возможное значение, притом, что такое разбиение не будет сообщать ничего полезного о графе. Поэтому определяется вектор $\mathbf{a}$ длины K с элементами $a_i = \sum_{j \in 1..K} e_{ij}$. Координата вектора a_i является *нормированной степенью сообщества* i и обозначает долю количества ребер, идущих к узлам, принадлежащим сообществу i, к полному количеству ребер в графе. Если в графе ребра проходят между вершинами независимо от сообществ — e_{ij} будет в среднем равно $a_i a_j$, поэтому модулярность можно определить следующим образом [?]:

$$Q(G, P) = \sum_{i \in 1..K} \left(e_{ii} - a_i^2 \right) = \mathrm{Tr}(\mathbf{e}) - \|\mathbf{e}^2\|, \tag{1}$$

где $\|\mathbf{x}\|$ является суммой элементов матрицы $\mathbf{x}$. Если сообщества распределены не лучше, чем в случайном разбиении — модулярность будет примерно равна 0. Максимальным возможным значением функции будет 1.

Теперь можно поставить задачу выделения сообществ следующим образом: *требуется найти* такое разбиение графа, на котором модулярность принимает максимальное значение. Можно заметить, что такая постановка задачи не использует какого-либо определения сообществ.

Такая задача все еще будет NP-сложной [**?**]. Однако преимущество модулярности состоит в том, что для того, чтобы посчитать,

какой выигрыш будет извлечен из объединения двух сообществ, необходимо произвести только одну операцию: $\Delta Q = 2(e_{ij} - a_i a_j)$, где i и j — потенциально объединяемые сообщества. Для того, чтобы объединить два сообщества необходимо сделать $O(\min\{n_i, n_j\})$ операций, где n_i и n_j обозначают количество смежных к i и j сообществ.

Рандомизированный жадный алгоритм (RG)

Эффективными оказались рандомизированные алгоритмы максимизации модулярности. В 2010 году Овельгенне и Гейер-Шульц в [?] был предложен рандомизированный жадный алгоритм (Randomized Greedy, *RG*), который на каждой итерации рассматривает k случайных сообществ и смежным к ним сообществ, а затем соединяет пару соседей, дающую наибольший выигрыш.

При отсутствии базового разбиения, на основе которого выделяются сообщества — перед первой итерацией граф разбивается на $K = N$ сообществ с одним узлом в каждом сообществом. Алгоритм останавливается в тот момент, когда не остается больше сообществ для объединения, однако применяются объединения сообществ до той итерации, когда объединение последний раз принесло глобальный выигрыш.

Далее в работе рандомизированный жадный алгоритм с параметром k будет обозначаться RG_k.

Схема кластеризации основных групп графа (CGGC)

В 2012 году Овельгенне и Гейер-Шульц выиграли конкурс 10th DIMACS Implementation Challenge в категории *кластеризация графа* со схемой кластеризации основных групп графа (Core Groups Graph Cluster, *CGGC*) [?]. Схема заключается в том, что сначала s *начальных алгоритмов* разбивают граф на сообщества. В тех вершинах, относительно которых начальные алгоритмы разошлись во мнении, выделяет по сообществам *финальный алгоритм*.

Формально это записывается следующим образом:

1. s начальных алгоритмов создают разбиения графа G на сообщества. S — множество *начальных разбиений*, то есть разбиений, полученных начальными алгоритмами
2. Создается *промежуточное разбиение* $\widetilde{P}$, равное максимальному перекрытию начальных разбиений из множества S
3. Финальным алгоритмом создается разбиение P графа G на основе промежуточного разбиения $\widetilde{P}$

Необходимо определить понятие *максимальное перекрытие*. Пусть существует множество разбиений $S = \{P_1, \ldots, P_s\}$, отображение $c_P(v)$ указывает, в каком сообществе находится узел v в разбиении P. Тогда у максимального перекрытия $\widetilde{P}$ множества S будут следующие свойства:

$$v, w \in \mathscr{N}, \forall i \in 1..s : \; c_{P_i}(v) = c_{P_i}(w) \Rightarrow c_{\widetilde{P}}(v) = c_{\widetilde{P}}(w)$$

$$v, w \in \mathscr{N}, \exists i \in 1..s : \; c_{P_i}(v) \neq c_{P_i}(w) \Rightarrow c_{\widetilde{P}}(v) \neq c_{\widetilde{P}}(w)$$

Существует итеративная версия схемы кластеризации основных групп графа, в которой начальные алгоритмы вновь выделяют сообщества на основе промежуточного разбиения до тех пор, пока это будет увеличивать модулярность промежуточного разбиения. Такой алгоритм далее обозначается как *CGGCi*. В качестве начальных и финального алгоритма можно использовать RG_k.

Одновременно возмущаемая стохастическая аппроксимация (SPSA)

Стохастическая аппроксимация была введена Роббинсом и Монро в 1951 году [**?**] и затем была использована для решения оптимизационный задач Кифером и Вольфовицем [**?**]. В [**?**] алгоритм стохастической аппроксимации был расширен до многомерного случая. В m-мерном пространстве обычная KW-процедура, основанная на конечно-разностной аппроксимации градиента, использовала $2m$ измерений на каждой итерации (по два измерения на каждую координату градиента). Последовательно Граничин [**?**], Поляк и Цыбаков [**?**] и Спалл [**?**] предложили алгоритм *одновременно возмущаемой стохастической аппроксимации* (*SPSA*), который на

каждой итерации использует всего два измерения. Алгоритм *SPSA* имеет такую же скорость сходимости, несмотря на то, что в многомерном случае (даже при $m \to \infty$) в нем используется заметно меньше измерений [**?**].

Алгоритмы стохастической аппроксимации показали свою эффективность в решении задач минимизации стационарных функционалов. В [**?**] для функционалов, меняющихся со временем были применены метод Ньютона и градиентный метод, но они применимы только в случае дважды дифференцируемых функционалов и в случае известных ограничений на Гессиан функционала. Так же оба метода требуют возможности вычисления градиента в произвольной точке.

Общую схему одновременно возмущаемой стохастической аппроксимации можно представить следующим образом:

1. Выбор начального приближения $\hat{\theta}_0 \in \mathbb{R}^m$, счетчик $n \leftarrow 0$, выбор параметров алгоритма $d \in \mathbb{R} \setminus \{0\}$, $\{\alpha_n\} \subset \mathbb{R}^m$
2. Увеличение счетчика $n \leftarrow n + 1$
3. Выбор вектора возмущения $\Delta_n \in \mathbb{R}^m$, чьи координаты независимо генерируются по распределению Бернулли, дающему ± 1 с вероятностью $\frac{1}{2}$ для каждого значения
4. Определение новых аргументов функционала $\theta_n^- \leftarrow \hat{\theta}_{n-1} - d\Delta_n$ и $\theta_n^+ \leftarrow \hat{\theta}_{n-1} + d\Delta_n$
5. Вычисление значений функционала $y_n^- \leftarrow f(\theta_n^-),\ y_n^+ \leftarrow f(\theta_n^+)$
6. Вычисление следующей оценки

$$\hat{\theta}_n \leftarrow \hat{\theta}_{n-1} - \alpha_n \Delta_n \frac{y_n^+ - y_n^-}{2d} \tag{2}$$

7. Далее происходит либо остановка алгоритма, либо переход на второй пункт

В [**?**, **?**, **?**] рассматривается метод стохастической аппроксимации с постоянным размером шага, в таком случае вместо последовательности $\{\alpha_n\}$ используется единственный параметр $\alpha \in \mathbb{R}^m$, и

следующая оценка вычисляется по следующей формуле вместо (2):

$$\hat{\theta}_n \leftarrow \hat{\theta}_{n-1} - \alpha\Delta\frac{y_n^+ - y_n^-}{2d} \tag{3}$$

Это позволяет эффективно решать проблемы очень плохого начального приближения $\hat{\theta}_0$ и дрейфующей оптимальной точки, когда аргумент, при котором функционал принимает лучшие значения, меняется со временем.

Постановка задачи адаптации параметров алгоритма

Рандомизированный жадный алгоритм имеет один параметр k, в то время как на результаты схемы кластеризации основных групп графа влияют параметр s и параметры начальных и финального алгоритма.

Часто алгоритмы на разных входных данных имеют разные оптимальные параметры, то есть нет одного набора параметров, решающих каждую задачу наилучшим образом. При одних и тех же параметрах некоторые графы будут разбивать хорошо, в то время как другие — критически плохо. Алгоритм *SPSA* показал хорошие результаты в создании адаптивных модификаций алгоритмов, то есть модификаций, подстраивающих параметры под входные данные.

В работе рассматривается применение алгоритма *SPSA* к алгоритмам *RG* и *CGGC* для создания алгоритмов, хорошо выделяющих сообщества на большем количестве графов, чем при основных наборах параметров этих рандомизированных алгоритмов.

Оценка качества

Для оценки качества сообщества используется целевая функция модулярность (1), а в качестве тестовых данных — графы, используемые для оценки алгоритмов на конкурсе 10th DIMACS Implementation Challenge, которые можно найти по адресу `http://www.cc.gatech.edu/dimacs10/archive/clustering.shtml`. Граф *celegans_metabolic* в работе обозначается как *celegans*.

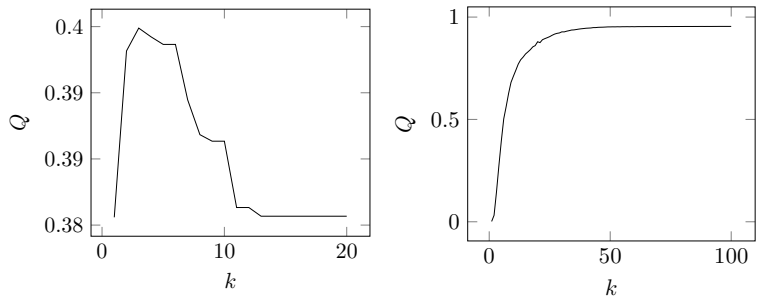
Так же в качестве тестового графа использовался синтетический граф *auto40* из $N = 40,000$ вершин, $K = 40$ сообществ, и с вероятностями $p_1 = 0.1$ и $p_2 = 10^{-4}$ обнаружить связь между вершинами одного сообщества или вершинами разных сообществ, соответственно.

3. Адаптивный рандомизированный жадный алгоритм (ARG)

Применимость алгоритма SPSA. Функция качества

Применимость *SPSA* обоснована теоретически для выпуклой усредненной функции качества.

В зависимости от графа усредненная модулярность результатов работы RG_k либо принимает максимум при небольшом k, как на рисунке 1a, либо постепенно увеличивается при росте k, как на рисунке 1b. Значение k, при которым RG_k будет на графе принимать наилучшее значение, далее в работе называется *оптимальным* k или k_{opt}.



(a) Зависимость модулярности от k на графе *karate*

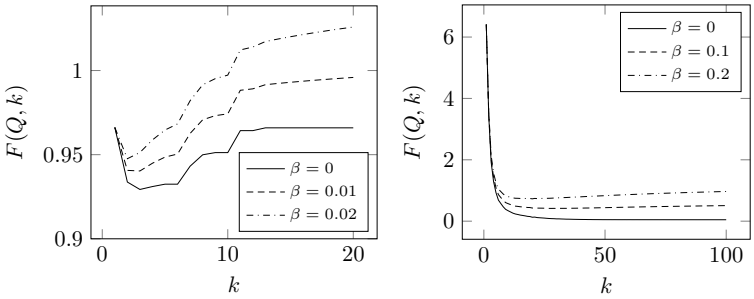
(b) Зависимость модулярности от k на графе *netscience*

Рис. 1: Зависимость модулярности от параметра k на двух графах

Время работы RG_k линейно растет при росте k, поэтому предлагается следующая функция качества:

$$f(Q, k) = -\ln Q + \beta \ln k. \tag{4}$$

Коэффициент $\beta \geq 0$ можно рассматривать в виде $\beta = \frac{\ln \gamma}{\ln 2}$, где γ указывает, во сколько раз можно позволить увеличиться модулярности, чтобы покрыть увеличение времени (то есть k) вдвое.



(a) Функция качества на графе *karate*

(b) Функция качества на графе *netscience*

Рис. 2: Функция качества при разных значения значениях β на двух графах

Как видно из рисунка 2b, иногда для того, чтобы функция качества имела экстремум — надо задавать довольно большое значение коэффициента β. Однако это логично, если время работы алгоритма удовлетворительно — выгодно все время увеличивать значение параметра k. Однако, как будет показано дальше, есть небольшие значения β дают хорошие результаты на всех тестовых графах.

Адаптивный рандомизированный жадный алгоритм

Для использования алгоритма *SPSA* в рандомизированном жадном алгоритме предлагается разбить действие алгоритма на шаги длиной в σ итераций. В течении каждого шага используется одно значение k, а после каждых двух шагов подбираются следующие два значения k, которые будут ближе к тому из предыдущих значений k, которое дало меньшую функцию качества.

В функции качества используется медиана прироста модулярности за σ шагов. Это связано с тем, что алгоритм рандомизированный, время от времени будут появляться очень хорошие соединения

сообществ, которые будут портить функцию качества, такой большой прирост может появиться даже при очень плохом k. В таком случае схема алгоритма будет выглядеть следующим образом:

1. Выбирается начальная оценка $\hat{k}_0 \in \mathbb{N}$, размер возбуждения $d \in \mathbb{N}$, чувствительность к перепадам функций качества $\alpha \in \mathbb{R},\ \alpha > 0$, значимость времени $\beta \in \mathbb{R},\ \beta \geq 0$, и количество $\sigma \in \mathbb{N}$ итераций в одном шаге. Устанавливается счетчик $n \leftarrow 0$

2. Увеличивается счетчик $n \leftarrow n + 1$

3. Определяются следующие два параметра $k_n^- \leftarrow \max\{\hat{k}_{n-1} - d, 1\}$ и $k_n^+ \leftarrow \hat{k}_{n-1} + d$

4. σ итераций рассматриваются k_n^- случайных сообществ и все их соседей, соединяется пара соседей, дающая лучший прирост модулярности. Медиана прироста модулярности за σ итераций обозначается μ_n^-

5. σ итераций рассматриваются k_n^+ случайных сообществ и все их соседей, соединяется пара соседей, дающая лучший прирост модулярности. Медиана прироста модулярности за σ итераций обозначается μ_n^+

6. Вычисляется функции качества $y_n^- \leftarrow -\ln \mu_n^- + \beta \ln k_n^-$ и $y_n^+ \leftarrow -\ln \mu_n^+ + \beta \ln k_n^+$

7. Вычисляется следующая оценка

$$\hat{k}_n \leftarrow \max\left\{1, \left[\hat{k}_{n-1} - \alpha \frac{y_n^+ - y_n^-}{k_n^+ - k_n^-}\right]\right\} \tag{5}$$

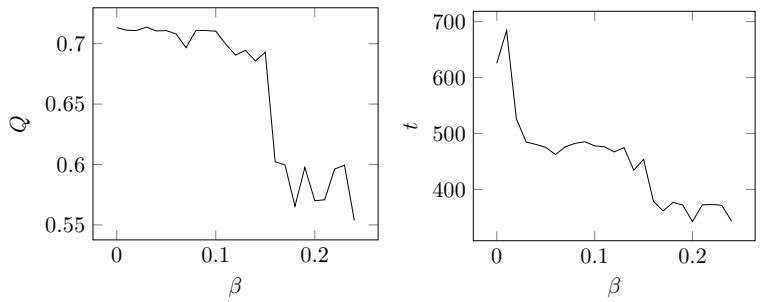
8. Далее происходит переход на второй пункт

Алгоритм заканчивает работу в тот момент, когда для рассмотрения осталось ровно одно сообщество. Далее в работе этот алгоритм называется *адаптивным рандомизированным жадным алгоритмом* или *ARG* (Adaptive Randomized Greedy).

Исследование работоспособности при разных параметрах

В отличии от *RG*, имевшего один параметр — *ARG* имеет пять независимых параметров. Однако, как показали эмпирические исследования на графах *cond-mat-2003*, *caidaRouterLevel* и *cnr-2000*, чувствительность к перепадам функций качества α и количество σ итераций в шаге слабо влияют на результаты вычислений. Размер возмущения d и начальная оценка $\hat{k}_0$ оказывают большее влияние на результаты, однако для них существуют значения, дающие хорошие результаты на каждом тестовом графе. Например, хорошие результаты будет давать набор параметров $\alpha = 10,\ \sigma = 1000,\ d = 5,\ \hat{k}_0 = 8$.

При увеличении значения коэффициента значимости времени β время работы алгоритма уменьшается за счет уменьшения работы, однако часто при небольших значениях β алгоритм дает более хорошие разбиения, чем при нулевом β.



(a) Зависимость модулярности от β в работе *ARG*

(b) Зависимость времени работы t алгоритма *ARG* от β

Рис. 3: Зависимости модулярности и времени работы *ARG* от коэфициента значимости времени β на графе *cond-mat-2003*

Сравнение RG и ARG

Для сравнения алгоритмов *RG* и *ARG* сопостовлялась медианная модулярность разбиений тестовых графов. В таблицах 1 и 2 указаны медианные модулярности разбиений, полученных с помощью *RG* с разными значениям параметра k:

- $k = 1$ — минимально возможное значение
- $k = 3$ — значение, чаще остальных дающее лучшие значения
- $k = 10$ — значение, дающее стабильные результаты
- $k = 50$ — значение, показывающее поведение RG при больших значениях параметра k

Алгоритм ARG запускался со следующими параметрами: $\alpha = 10,\ \sigma = 1000,\ d = 5,\ \hat{k}_0 = 8,\ \beta = 0.05$.

Таблица 1: Медианные модулярности разбиений, полученных в результате работы RG_k с разными k и ARG на тестовых графах

	RG_1	RG_3	RG_{10}	RG_{50}	ARG
as-22july06	0.65281	0.64658	0.64024	0.63479	0.64264
cond-mat-2003	0.00012	0.19727	0.70738	0.69403	0.71193
auto40	0.78944	0.79988	0.80417	0.80273	0.80174
caidaRouterLevel	0.01938	0.81101	0.79883	0.79300	0.80216
cnr-2000	0.90237	0.91192	0.91144	0.90997	0.91039
eu-2005	0.92765	0.92559	0.91780	0.90416	0.91048
in-2004	0.00026	0.97836	0.97185	0.97596	0.97616

Таблица 2: Время работы (в миллисекундах) RG_k с разными k и ARG на тестовых графах

	RG_1	RG_3	RG_{10}	RG_{50}	ARG
as-22july06	177	189	231	464	238
cond-mat-2003	58	184	463	931	474
auto40	4,652	4,591	6,017	12,558	6,479
caidaRouterLevel	852	9,114	10,244	15,217	11,514
cnr-2000	26,083	26,056	27,137	33,592	29,054
eu-2005	202,188	200,686	207,689	246,170	225,748
in-2004	9,208	487,953	553,196	607,408	617,345

В большинстве случаев один или несколько параметров k дают RG_k лучшие результаты, чем результат ARG, однако адаптивный алгоритм дает более стабильные результаты. ARG можно сравнить по стабильности с RG_{10}, однако в среднем ARG дает бóльшую модулярность.

Применение ARG в качестве начального алгоритма CGGC

Время работы схемы основных групп графа ($CGGC$) складывается из времени работы s начальных алгоритмов и времени работы финального алгоритма. Поэтому имеет смысл в качестве начальных алгоритмов брать алгоритмы, работающие быстро. Однако в случае, если начальные алгоритмы очень плохие, то есть дают разбиения не лучше случайного — модулярность разбиений, полученных $CGGC$ будет приблизительно равна модулярности разбиений, полученных финальным алгоритмом.

Если граф достаточно большой (количество узлов как минимум в несколько раз больше 2σ) — на нем в качестве начального алгоритма имеет смысл использовать ARG, чтобы получать стабильные результаты.

Таблица 3: Модулярность разбиений, полученных в результате работы $CGGC$ с начальным алгоритмом A_{init} и финальным алгоритмом A_{final} на трех графах

A_{init}	RG_3		RG_{10}		ARG	
A_{final}	RG_3	RG_{10}	RG_3	RG_{10}	RG_3	RG_{10}
cond-mat-2003	0.16840	0.71155	0.44934	0.74794	0.42708	0.74872
auto40	0.80628	0.80645	0.80633	0.80645	0.80628	0.80647
caidaRouterLevel	0.84078	0.85372	0.84031	0.84448	0.83671	0.85279

Таблица 4: Время работы $CGGC$ с начальным алгоритмом A_{init} и финальным алгоритмом A_{final} на трех графах

A_{init}	RG_3		RG_{10}		ARG	
A_{final}	RG_3	RG_{10}	RG_3	RG_{10}	RG_3	RG_{10}
cond-mat-2003	2.0	2.3	4.7	4.8	4.8	4.9
auto40	47.0	46.6	57.7	57.2	65.7	65.8
caidaRouterLevel	94.0	93.7	104.2	104.3	115.1	118.5

На таблицах 3 и 4 сравниваются результаты работы $CGGC$ с разными начальными и финальными алгоритмами. В качестве начального алгоритма используеся RG_k с параметрами $k = 3$ и $k = 10$, и ARG с параметрами $\alpha = 10,\ \sigma = 1000,\ d = 5,\ \hat{k}_0 = 8,\ \beta = 0.02$. В качестве финального алгоритма используются только RG_3 и RG_{10}, так как на рассматриваемых графах промежуточное разбиение со-

стоит из недостаточно большого количества сообществ, чтобы использовать ARG.

Граф *cond-mat-2003* плохо разбивается случайными жадными алгоритмами с маленьким k, и если обратить внимание на таблицу 3 — в случаях, когда RG_3 используется на этом графе в качестве начального алгоритма — модулярность $CGGC$ сравнима с модулярностью финального алгоритма на этом графе (модулярности RG_3 и RG_{10} на разных графах можно увидеть в таблице 1). В случаях, когда RG_3 используется в качестве финального алгоритма — модулярность получается сравнительно плохой.

Заранее неизвестно, при каких k алгоритм будет хорошо работать, а при каких плохо, кроме того неизвестно по результату, хорошее ли это разбиение для конкретного графа или нет. $CGGC$ работает достаточно долго, поэтому запускать $CGGC$ с разными начальными алгоритмами несколько раз для определения хороших параметров не выгодно. Таким образом, выгодно использовать ARG и в качестве начального алгоритма, и в качестве финального, если промежуточное разбиение подходит по размерам.

4. Адаптивная схема кластеризации основных групп графа (CGGC)

Для создания адаптивного алгоритма, работающего на графах любого размера, предлагается использовать алгоритм $SPSA$ при выборе начальных алгоритмов. Так же при построении промежуточного разбиения предлагается использовать не все разбиения, а только некоторое количество лучших. Более точно адаптивная схема описывается следующим образом:

1. На вход подается граф G. Выбираются размер возмущения $d \in \mathbb{N}$, чувствительность к перепадам функций качества $\alpha > 0 \in \mathbb{R}$, количество шагов $l \in \mathbb{N}$, начальное приближение $\hat{k}_0 \in \mathbb{N}$, максимальная оценка $k_{max} > 1 \in \mathbb{N}$ и $r \in (0, 1]$ — параметр, отвечающий за количество лучших начальных разбиений, учавствующих в создании промежуточного разбиения. Инициализируется счетчик $n \leftarrow 0$ и множество начальных разбиений $S \leftarrow \emptyset$

2. Увеличивается счетчик $n \leftarrow n + 1$

3. Вычисляются следующие два параметра RG:
$k_n^- \leftarrow \max\{1,\ \hat{k}_{n-1} - d\}$ и $k_n^+ \leftarrow \min\{k_{max},\ \hat{k}_{n-1} + d\}$

4. Выделяются сообщества в графе G алгоритмом RG_k с параметром $k = k_n^-$, получившееся разбиение P_n^- записывается в список S, его модулярность обозначается Q_n^-

5. Выделяются сообщества в графе G алгоритмом RG_k с параметром $k = k_n^+$, получившееся разбиение P_n^+ записывается в список S, его модулярность обозначается Q_n^+

6. Вычисляются функции качества:
$y_n^- \leftarrow -\ln Q_n^-$ и $y_n^+ \leftarrow -\ln Q_n^+$

7. Вычисляется следующая оценка

$$\hat{k}_n = \max\left\{1,\ \min\left\{k_{max},\ \left[\hat{k}_{n-1} - \alpha\frac{y_n^+ - y_n^-}{k_n^+ - k_n^-}\right]\right\}\right\} \qquad (6)$$

8. Если $n \neq l$ — осуществляется переход на второй пункт, иначе — на следующий

9. Из множества S исключаются все разбиения на сообщества с модулярностью $Q \leq (1 - r)Q_{best}$, где Q_{best} — модулярность лучшего разбиения в S.

10. Создается промежуточное разбиение — максимальное перекрытие разбиений из множества S

11. Финальным алгоритмом выделяются сообщества на основе промежуточного разбиения

Далее в работе описанная схема будет называться адаптивной схемой кластеризации основных групп графа и обозначаться $ACGGC$ (Adaptive Core Groups Graph Clustering).

Исследование работоспособности при разных параметрах

ACGGC имеет достаточно много параметров, однако подобранные на четырех графах значения параметров показали хорошие результаты на всех остальных тестовых графах. Однако даже при плохих параметрах результат будет хуже лишь на несколько процентов. Хорошим набором параметров будет, например, $d = 2,\ \alpha = 1000,\ l = 6,\ \hat{k}_0 = 5,\ k_{max} = 50,\ r = 0.05$.

Таблица 5: Модулярности разбиений, полученных *ACGGC* с разными k_{max} на тестовых графах

k_{max}	$+\infty$	50	20	10	6
polbooks	0.527237	0.527237	0.527237	0.527082	0.526985
adjnoun	0.299720	0.299690	0.299859	0.300141	0.299676
football	0.603324	0.603324	0.604184	0.604266	0.604266
jazz	0.444739	0.444739	0.444739	0.444739	0.444739
celegans	0.439770	0.439368	0.439750	0.439460	0.439431
email	0.573470	0.573416	0.573652	0.573756	0.573513
netscience	0.953033	0.908130	0.842085	0.793289	0.768572
cond-mat-2003	0.737611	0.743572	0.749595	0.749894	0.739200

Таблица 6: Время работы *ACGGC* при разных k_{max} на тестовых графах

k_{max}	$+\infty$	50	20	10	6
polbooks	5.029	4.976	4.615	4.207	4.087
adjnoun	6.115	6.099	5.481	4.952	4.744
football	7.179	7.155	6.377	5.820	5.584
jazz	23.66	23.25	20.92	19.12	18.59
celegans	23.85	23.49	22.48	20.92	20.01
email	70.06	72.89	68.34	63.85	62.97
netscience	477.97	85.40	46.08	38.26	30.89
cond-mat-2003	41,950	9,596	6,075	5,092	4,166

Уменьшение максимальной оценки k_{max} играет роль снижения времени работы, более того, часто при уменьшении k_{max} модулярность растет. Сравнение модулярности разбиений и времени работы *ACGGC* с разными значениями k_{max} можно увидеть на таблицах 5 и 6.

Сравнение CGGC и ACGGC

Для сравнения $CGGC$ и $ACGGC$ сопоставлялась медианная модулярность разбиений тестовых графов. На таблице 7 обозначены результаты пять алгоритмов:

- $ACGGC$ с параметрами $d = 2,\ \alpha = 1000,\ l = 6,\ \hat{k}_0 = 5,\ k_{max} = 50,\ r = 0.05$, финальным алгоритмом RG_{10}, обозначается $ACGGC^I$
- $ACGGC$ с параметрами $d = 2,\ \alpha = 1000,\ l = 8,\ \hat{k}_0 = 5,\ k_{max} = 20,\ r = 0.05$, финальным алгоритмом RG_{10}, обозначается $ACGGC^{II}$
- $CGGC$ с начальным алгоритмом RG_{10}, финальным алгоритмом RG_{10} и $s = 16$, обозначается $CGGC_{10}^{10}$
- $CGGC$ с начальным алгоритмом RG_3, финальным алгоритмом RG_{10} и $s = 16$, обозначается $CGGC_3^{10}$
- $CGGC$ с начальным алгоритмом RG_{10}, финальным алгоритмом RG_3 и $s = 16$, обозначается $CGGC_{10}^3$

Измерения для $CGGC_3^{10}$ и $CGGC_{10}^3$ на графе *in-2004* не производились из-за большого времени работы.

В таблице 7 лучшая модулярность на графе обозначалась сплошным подчеркиванием, а вторая по величине — пунктирным подчеркиванием (если несколько алгоритмов принимали лучшее значение — все отмечались пукнтирной линией и второе по величине значение никак не отмечалось).

При этом $ACGGC$ в основном дают меньшее время, чем $CGGC$ с указанными параметрами, однако в качестве параметра s алгоритма $CGGC$ было выбрано значение $s = 16$, так как в [?] отмечалось, что при росте s модулярности приближаются к асимптоте максимального принимаемого алгоритмом значения или держатся на одном уровне. Такое значение будет гарантировать хорошие для $CGGC$ модулярности, но не будет гарантировать небольшого времени работы.

Таблица 7: Модулярность разбиений, полученных $ACGGC$ и $CGGC$ на тестовых графах

	$ACGGC^{I}$	$ACGGC^{II}$	$CGGC_{10}^{10}$	$CGGC_{3}^{10}$	$CGGC_{10}^{3}$
karate	0.417242	0.417406	0.415598	0.396532	0.405243
dolphins	0.524109	0.523338	0.521399	0.523338	0.522428
chesapeake	0.262439	0.262439	0.262439	0.262439	0.262370
adjnoun	0.299704	0.299197	0.295015	0.292703	0.290638
polbooks	0.527237	0.527237	0.527237	0.526938	0.526784
football	0.603324	0.604266	0.604266	0.599537	0.599026
celegans	0.439604	0.438584	0.435819	0.436066	0.432261
jazz	0.444739	0.444848	0.444871	0.444206	0.444206
netscience	0.907229	0.835267	0.724015	0.708812	0.331957
email	0.573333	0.573409	0.571018	0.572667	0.567423
polblogs	0.424107	0.423208	0.422901	0.421361	0.390395
pgpGiantCompo	0.883115	0.883085	0.882237	0.882532	0.880340
as-22july06	0.671249	0.670677	0.666766	0.669847	0.665260
cond-mat-2003	0.744533	0.750367	0.751109	0.708775	0.413719
caidaRouterLevel	0.846312	0.855651	0.851622	0.858955	0.843835
cnr-2000	0.912762	0.912783	0.912500	0.912777	0.912496
eu-2005	0.938292	0.936984	0.935510	0.936515	0.936420
in-2004	0.979844	0.979771	0.979883		

Итеративная схема

По аналогии с итеративной схемой $CGGCi$ представляется адаптивная итеративная схема $ACGGCi$. В такой схеме после создания промежуточного разбиения — на его основе вновь выделяют сообщества начальные алгоритмы, это повторяется, пока модулярность промежуточного разбиения увеличивается. И после этого получившееся на основе промежуточного разбиения выделяет сообщества финальный алгоритм. Такой подход не сильно повышает время работы, так как с каждой итерации количество узлов (сообществ) для разбиения уменьшается, и соответственно, выделение сообществ на каждой итерации занимает меньше времени.

Как видно из таблицы 8, модулярность итеративной схемы каждый раз немного выше, хотя и время работы каждый раз заметно выростало, на некоторых графах в полтора раза, а на других — более, чем в три.

В качестве финального алгоритма $ACGGCi$ можно использовать $CGGCi$, ровно как и наоборот, в качестве финального алгоритма $CGGCi$ использовать $ACGGCi$. Такой подход обусловлен тем,

Таблица 8: Модулярность и время работы $ACGGC$ и $ACGGCi$ на пяти графах

	$ACGGC$		$ACGGCi$	
	Q	t	Q	t
jazz	0.444739	23.68	0.444871	31.51
celegans	0.439724	23.92	0.446973	77.25
netscience	0.907922	86.38	0.909400	96.55
as-22july06	0.671205	2,329	0.674992	5,801
cond-mat-2003	0.743594	9,371	0.746731	11,654

что не умоляя общности сначала $ACGGCi$ итерирует создание промежуточных разбиений, останавливаясь, когда это перестает приносить выигрыш. Однако другая схема может оказаться эффективной на получившемся промежуточном разбиении. Назовем такой подход *комбинированной схемой кластеризации*.

На графе *celegans* провелось сравнение модулярности и времени работы итерационных схем с разными финальными алгоритмами, результаты изображены на таблице 9. Стрелка $\rightarrow$ указывает на то, какой алгоритм принимает начинает следующим выделять сообщества в промежуточном множестве. Например, $ACGGCi \rightarrow CGGCi \rightarrow RG_{10}$ обозначает $ACGGCi$ с финальным алгоритмом $CGGCi$ с финальным алгоритмом RG_{10}. Начальным алгоритмом $CGGCi$ используется также RG_{10}.

Таблица 9: Модулярность и время работы итерационных схем с разными финальными алгоритмами

			Q	t
$ACGGCi$	$\rightarrow RG_{10}$		0.446973	77.25
$CGGCi$	$\rightarrow RG_{10}$		0.445008	55.29
$ACGGCi$	$\rightarrow CGGCi$	$\rightarrow RG_{10}$	0.447324	89.96
$CGGCi$	$\rightarrow ACGGCi$	$\rightarrow RG_{10}$	0.445660	112.49

На графе *celegans* комбинированные схемы с $CGGCi$ и $ACGGCi$ на первом этапе дали лучшие результаты, чем обычные итерационные схемы $CGGCi$ и $ACGGCi$, соответственно.

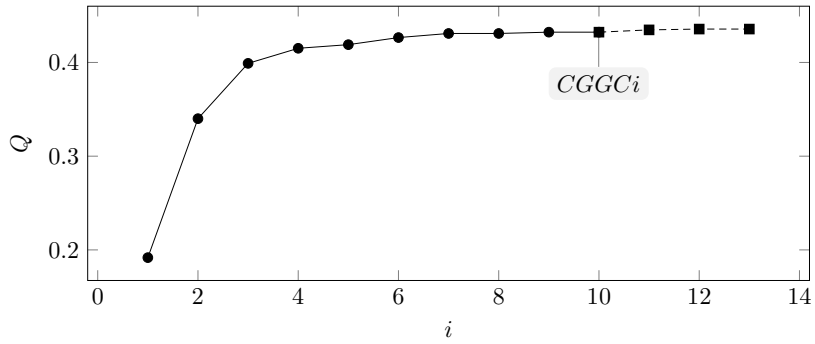


Рис. 4: Модулярность промежуточных разбиений в работе $ACGGCi \to CGGCi \to RG_{10}$ на графе *celegans*, i — номер итерации. Круглые точки указывают промежуточные разбиения $ACGGCi$, а квадратные — $CGGCi$

Сравнение итерационных схем

Таблица 10: Модулярность работы четырех итеративных алгоритмов на небольших тестовых графах

	$ACGGCi^I$	$ACGGCi^{II}$	$CGGCi$	combined
karate	0.417242	0.417406	0.417242	0.417242
dolphins	0.525869	0.525869	0.525869	0.525869
chesapeake	0.262439	0.262439	0.262439	0.262439
adjnoun	0.303731	0.303504	0.303571	0.303970
polbooks	0.527237	0.527237	0.527237	0.527237
football	0.604266	0.604407	0.604429	0.604407
celegans	0.446964	0.446836	0.445442	0.447234
jazz	0.444871	0.444871	0.444871	0.444871
netscience	0.908845	0.888422	0.725781	0.907443
email	0.576778	0.577000	0.576749	0.577110
polblogs	0.424025	0.422920	0.423281	0.423996

На таблице 10 представлено сравнение модулярностей четырех алгоритмов, $ACGGCi^I$, $ACGGCi^{II}$ и $CGGCi$ представляют итеративные версии $ACGGC^I$, $ACGGC^{II}$ и $CGGC_{10}^{10}$, соответственно. Схема $ACGGCi \to CGGCi \to RG_{10}$ изображена в таблице под на-

званием *combined*. Сплошной линией отмечены лучшие разбиения.

Итеративная схема *CGGCi* на тестовых графах всего единожды приняла лучшее значение и, в отличии от остальных итерационных схем, один раз (на графе *netscience*) приняла очень плохое значение.