САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра Информатики

Проданов Тимофей Петрович

Адаптивный рандомизированный алгоритм выделения сообществ в графах

Бакалаврская работа

Допущена к защите.
Зав. кафедрой:

Научный руководитель:
д. ф.-м. н., профессор О.Н. Граничин

Рецензент:
В.А. Ерофеева

Санкт-Петербург
2015

SAINT-PETERSBURG STATE UNIVERSITY
Mathematics & Mechanics Faculty

Department of Computer Science

Timofey Prodanov

Adaptive randomised algorithm for community detection in graphs

Bachelor's Thesis

Admitted for defence.
Head of the chair:

Scientific supervisor:
Professor Oleg Granichin

Reviewer:
Victoria Erofeeva

Saint-Petersburg
2015

Оглавление

Введение

Исторически, изучение сетей происходило в рамках теории графов, которая начала своё существование с решения Леонардом Эйлером задачи о кёнигсбергских мостах. В 1920-х взял своё начало анализ социальных сетей и лишь последние двадцать лет развивается изучение *сложных сетей*, то есть сетей с неправильной, сложной структурой, в некоторых случаях рассматривают динамически меняющейся во времени сложные сети. От изучения маленьких сетей внимание переходит к сетям из тысяч или миллионов узлов.

В процессе изучения сложных систем, построенных по реальным системам, оказалось, что распределение степеней $P(s)$, определённое 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 [1]. Это показывает, что модели, предложенные теорией графов, не всегда будут хорошо работать для графов, построенных по реальным системам.

Современное изучение сложных сетей привнесло значительный вклад в понимание реальных систем. Сложные сети с успехом были применены в таких разных областях, как эпидемиология [2], биоинформатика [3], поиск преступников [4], социология [5], изучение структуры и топологии интернета [6, 7] и в многих других.

Свойством, присутствующим почти у любой сети, является структура *сообществ*, разделение узлов сети на разные группы узлов так, чтобы внутри каждой группы соединений между узлами много, а соединений между узлами разных групп мало. Плотно связанные группы узлов в социальных сетях представляют людей, принадлежащих социальным сообществам, плотно сплочённые группы узлов в интернете соответствуют страницам, посвящённым распространённым темам, а сообщества в генетических сетях связаны с функциональными модулями [1]. Способность находить и анализировать подобные группы предоставляет большие возможности в изучении реальных систем, представленных с помощью сложных сетей.

В разделе 1 будут рассмотрены существующие методы выделения сообществ в графах, а так же другая необходимая информация о графах и сложных сетях. Так же в разделе будет кратко описана одновременно возмущаемая стохастическая аппроксимация, а в подразделе 1.6 будет поставлена задача. Затем в разделе 2 будет представлен адаптивный рандомизированный жадный алгоритм, его анализ и сравнение с существующими алгоритмами, возможное применение в подразделе 3.1. В разделе 3 будет представлена адаптивная схема кластеризации основных групп графа, анализ и сравнение с существующими алгоритмами.

1. Предварительные сведения

1.1. Определения и обозначения

Формально, сложная система может быть представлена с помощью графа. В этой 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 *рёбрами* или *связями*. Мощности множеств $\mathscr{N}$ и $\mathscr{L}$ равны N и L соответственно.

Подграфом называется граф $G' = (\mathscr{N}', \mathscr{L}')$, где $\mathscr{N}' \subset \mathscr{N}$ и $\mathscr{L}' \subset \mathscr{L}$.

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 рёбер, выходящих узла i.

Прогулка из узла i в узел j — это последовательность узлов, начинающаяся с узла i и заканчивающаяся узлом j. *Путь* — это прогулка, в которой каждый узел встречается единожды. *Геодезический путь* — это кратчайший путь, а количество узлов в нём на один больше *геодезического расстояния*.

До того, как в работе будет определено понятие *сообщество*, вводится понятие *разбиение* на сообщества. Пусть $G = (\mathscr{N}, \mathscr{L})$ — граф, тогда разбиением на сообщества будет называться 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 $\{C_1, \ldots, C_{K_1}\}$ называется разбиением на сообщества на основе разбиения $\hat{P} = \left\{\hat{C}_1, \ldots \hat{C}_{K_2}\right\}$, если $\forall i \in 1..K_2 \ \exists j \in 1..K_1 : \hat{C}_i \subset C_j$.

Сообщество — это такой подграф, чьи узлы плотно связаны, однако структурную сплочённость узлов можно определить по разному. Одно из определений вводит понятие *клик*. Клик — это максимальный подграф, состоящий из трёх и более вершин, каждая из которых связана с каждой другой вершиной из клика. n-клик — это максимальный подграф, в котором самое большое геодезическое расстояние между любыми двумя вершинами не превосходит n. Другое определение гласит, что подграф G' является сообществом, если сумма всех степеней внутри G' больше суммы всех степеней, направленных в остальную часть графа [8]. Два сообщества называются смежными, если существует ребро, направленное из вершины первого сообщества в вершину второго.

1.2. Модулярность

Однако подобными определениями сообществ пользоваться неудобно и их проверка достаточно долгая. В 2004 году была представлена *модулярность* — целевая

функция, оценивающая неслучайность разбиения графа на сообщества [9]. Допустим, имеется K сообществ, тогда определяется симметричная матрица $\mathbf{e}$ размером $K \times K$. Пусть e_{ij} — отношение количества рёбер, которые идут из сообщества i в сообщество j, к полному количеству рёбер в графе (рёбра l_{mn} и l_{nm} считаются различными, где m, n — узлы). След такой матрицы $\mathrm{Tr}\mathbf{e} = \sum_{i \in 1..K} e_{ii}$ показывает отношение рёбер в сети, которые соединяют узлы одного и того же сообщества, и 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 далее определяется вектор $\mathbf{a}$ длины K, элементы которого $a_i = \sum_{j \in 1..K} e_{ij}$, который обозначает долю количества рёбер, идущих к узлам, принадлежащим сообществу i, к полному количеству рёбер в графе. Если в графе рёбра 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, но на практике единица не достигается.

Было предложено несколько вариаций модулярности [10, 11]. Так, эквивалентным приведённому выше определению будет

$$Q(G, P) = \frac{1}{2L} \sum_{x,y \in 1..N} \left(w_{xy} - \frac{s_x s_y}{2L} \right) \delta(c_P(x), c_P(y)), \tag{2}$$

где L — мощность $\mathscr{L}$, w_{xy} — вес ребра между вершинами x и y, s_x и s_y — степени вершин x и y соответственно, δ — символ Кронекера, а отображение $c_P(\cdot)$ указывает, в каком сообществе разбиения лежит узел графа.

Теперь можно поставить задачу выделения сообществ следующим образом: требуется найти такое разбиение графа, что модулярность примет максимальное значение. Можно заметить, что такая постановка задачи не использует какого-либо определения сообществ, и получившееся разбиение не проверяется на дополнительные свойства, кроме подсчёта модулярности. Однако такая задача всё ещё будет NP-сложной [12].

Преимущество модулярности состоит в том, что для того, чтобы посчитать, какой выигрыш будет извлечён из объединения двух сообществ, необходимо произвести только одну операцию. В рамках определения (1) такой выигрыш будет равен $\Delta Q = 2(e_{ij} - a_i a_j)$, 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. Не умоляя общности, $n_j \leq n_i$, тогда необходимо обновить столбец i-ый столбец и i-ую строку матрицы **e**, а так же i-ый элемент вектора **a**: $e_{ik} = e_{ki} = e_{ki} + e_{kj}$, где k — смежное к j сообщество, и $a_i = a_i + a_j$. При этом сообщество j следует удалить из дальнейшего рассмотрения.

Имея матрицу **e** и вектор **a** не очень важно, как устроен граф и сообщества, что позволяет искать сообщества, основываясь на некотором начальном разбиении, для которого уже построены **e** и **a**.

1.3. Рандомизированный жадный алгоритм

Ньюман в 2004 году предложил алгоритм, максимизирующий модулярность [13]. Алгоритм начинается с разбиения графа на N сообществ из одной вершины, а затем на каждой итерации просматривает все пары сообществ и соединяет ту пару, которая даст наибольший выигрыш модулярности. Такой алгоритм достаточно долго работает и страдает от несбалансированного объединения сообществ — сообщества растут с разной скоростью, большие кластеры соединяются со своими небольшими соседями независимо от того, выгодно это глобально или нет [14].

Поэтому был предложен рандомизированный жадный алгоритм (Randomized Greedy, RG) [15], который на каждой итерации рассматривал k случайных сообществ и смежных к ним сообществ, а затем так же соединял пару, дающую наибольший выигрыш. Трудоёмкость такого алгоритма примерно равна $O(kL \ln N)$. И первый алгоритм, и его рандомизированная вариация соединяют сообщества, записывая только номера соединений, до тех пор, пока не останется только одно сообщество, а затем создают разбиение из списка соединений до того момента, когда достигалась максимальная модулярность (так как в результате лучшего соединения модулярность может уменьшиться).

Входные данные: Невзвешенный неориентированный граф $G = (\mathscr{N}, \mathscr{L})$, параметр k
Выходные данные: Разбиение на сообщества P
for $i \in 1..N$ **do**
  for $j \in 1..N$ **do**
    if i и j смежные **then**
      $e[i, j] = 1/(2 * L)$;
    else
      $e[i, j] = 0$;
    end
  end
  $a[i] = \sum_j e[i, j]$;
end

Алгоритм 1: Этап инициализации рандомизированного жадного алгоритма

```
global_ΔQ ← 0;
max_global_ΔQ ← −∞;
for i ∈ 1..N do
    max_ΔQ ← −∞;
    for j ∈ 1..k do
        c1 ← случайное сообщество;
        forall сообщества c2, смежные с c1 do
            ΔQ ← 2 ∗ (e[i, j] − a[i] ∗ a[j]);
            if ΔQ > max_ΔQ then
                max_ΔQ ← ΔQ;
                next_join ← (c1, c2);
            end
        end
    end
    joins_list.push(next_join);
    global_ΔQ ← global_ΔQ + max_ΔQ;
    if global_ΔQ > max_global_ΔQ then
        max_global_ΔQ ← global_ΔQ;
        best_step ← i;
    end
    (c1, c2) ← next_join;
    if количество соседей(c2) > количество соседей(c1) then
        поменять местами c1 и c2;
    end
    forall соседи c3 сообщества c2, где c3 ≠ c1, c2 do
        e[c3, c1] ← e[c3, c1] + e[c3, c2];
        e[c1, c3] ← e[c3, c1];
    end
    e[c1, c1] ← e[c1, c1] + e[c2, c2] + e[c1, c2] + e[c2, c1];
    a[c1] ← a[c1] + a[c2];
end
P ← создать разбиение из joins_list[1..best_step];
```

Алгоритм 2: Рандомизированный жадный алгоритм

Далее в работе рандомизированный жадный алгоритм с параметром k будет обозначаться RG_k. Можно отметить, что таким алгоритмом можно выделять сообщества на основе некоторого начального разбиения, при этом только немного поменяется начальный этап инициализации матрицы **e** и вектора **a** (как можно увидеть в алгоритме 1).

1.4. Схема кластеризации основных групп графа

Овельгённе и Гейер-Шульц в 2012 году 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, и считается, что те вершины, в которых начальные алгоритмы сошлись во мнении, распределены по сообществам правильно, а те, которые остались, распределяет по сообществам *финальный алгоритм* [16].

Формализовать это можно следующим образом:

1. Создаётся множество S из s разбиений G с помощью начальных алгоритмов
2. Создаётся разбиение $\hat{P}$, равное максимальному перекрытию разбиений из множества S
3. Финальным алгоритмом создаётся разбиение P графа G на основе разбиения $\hat{P}$

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 $\hat{P}$ множества S будут следующие свойства:

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

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

$CGGC$ можно итерировать, заставляя начальные алгоритмы разбивать максимальное перекрытие и получившееся максимальное перекрытие до тех пор, пока это будет увеличивать модулярность, такой алгоритм далее обозначается как $CGGCi$. В таком случае схема будет выглядеть следующим образом:

1. Инициализировать $\hat{P}$ разбиением из сообществ из одного узла
2. Создать множество S из s разбиений графа G на основе разбиения $\hat{P}$ с помощью начальных алгоритмов
3. Записать в $\hat{P}$ максимальное перекрытие множества S
4. Если разбиение P_{best} не существует или оно хуже, чем $\hat{P}$, то присвоить $P_{best} \leftarrow \hat{P}$ и вернуться на второй шаг
5. Финальным алгоритмом создать разбиение P графа G на основе разбиения P_{best}

В качестве начальных и финального алгоритма можно брать RG_k.

1.5. Одновременно возмущаемая стохастическая аппроксимация

Стохастические аппроксимация была введена Роббинсом и Монро в 1951 году [17] и затем была использована для решения оптимизационный задач Кифером и Вольфовицем [18]. В [19] 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). Спалл предложил алгоритм *одновременно возмущаемой стохастической аппроксимации* (*SPSA*) [20], который на каждой итерации использует всего два измерения. Он показал, что *SPSA* алгоритм имеет такую же скорость сходимости, несмотря на то, что в многомерном случае (даже при $m \to \infty$), несмотря на то, что в нём используется заметно меньше измерений [21].

Стохастическая аппроксимация первоначально использовалась как инструмент для статистических вычислений и в дальнейшем разрабатывалась в рамках отдельной ветки теории управления. На сегодняшний день стохастическая аппроксимация имеет большое разнообразие применений в таких областях, как адаптивная обработка сигналов, адаптивное выделение ресурсов, адаптивное управление.

Алгоритмы стохастической аппроксимации показали свою эффективность в решении задач минимизации стационарных функционалов. В [22] 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 начальной центральной точки $\hat{\theta}_0 \in \mathbb{R}^m$, счётчик $n \leftarrow 0$, выбор параметров алгоритма $d \in \mathbb{R} \setminus \{0\}$, $\{\alpha_n\} \subset \mathbb{R}^m$

2. Увеличение счётчика $n \leftarrow n + 1$

3. Выбор вектора возмущения $\Delta_n \in \mathbb{R}^m$, чьи координаты независимо генерируются и в среднем дают ноль. Часто для генерации компонент вектора используют распределение Бернулли, дающее ± 1 с вероятностью $\frac{1}{2}$ для каждого значения

4. Определение новых аргументов функции $\theta_n^- \leftarrow \hat{\theta}_{n-1} - d\Delta_n$ 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 центральной точки

$$\hat{\theta}_n \leftarrow \hat{\theta}_{n-1} - \alpha_n \frac{y_n^+ - y_n^-}{|\theta_n^+ - \theta_n^-|} \tag{3}$$

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

В [23] был представлен метод стохастической аппроксимации с константным размером шага, в таком случае вместо последовательности $\{\alpha_n\}$ используется единственный параметр $\alpha \in \mathbb{R}^m$, и следующая центральная точка вычисляется по следующей формуле: $\hat{\theta}_n \leftarrow \hat{\theta}_{n-1} - \alpha \frac{y_n^+ - y_n^-}{|\theta_n^+ - \theta_n^-|}$

1.6. Постановка задачи

Рандомизированный жадный алгоритм имеет один параметр k, в то время как схема кластеризации основных групп графа один параметр s и различные начальные и финальные алгоритмы. В случае использования RG_k в качестве начального алгоритма — $CGGC$ можно рассматривать как алгоритм с параметрами $s,\ k$ и некоторым финальным алгоритмом.

Часто алгоритмы на каждых входных данных имеют разные оптимальные параметры, то есть нет одного набора параметров, решающих каждую задачу наилучшим образом. И алгоритм *SPSA* показал хорошие результаты в адаптировании параметров подобных алгоритмов, когда в ходе работы рассматриваемого алгоритма его параметры варьируются, довольно быстро достигая оптимальной точки.

В данной работе будет рассматриваться применение алгоритма *SPSA* к двум алгоритмам выделения сообществ в графах, а так же сравнение модулярностей этих алгоритмов и полученных с помощью *SPSA* модификаций.

1.7. Тестовые графы

В качестве тестовых графов были взяты графы, используемые для оценки алгоритмов на конкурсе 10th DIMACS Implementation Challenge. Далее представлены используемые графы с кратким описанием, в порядке увеличения количества узлов.

- **karate**, $N = 34,\ L = 78$: социальная сеть между 34 членами карате клуба с 1970 по 1972 год [24]
- **chesapeake**, $N = 39,\ L = 170$: сеть мезогалинных вод Чесапикского залива [25]
- **dolphins**, $N = 62,\ L = 159$: социальная сеть частых общений между 62 дельфинами [26]

- **polbooks**, $N = 105,\ L = 441$: сеть книг о политике США, изданных во время президентских выборов 2004 года. Рёбра между книгами означают частые покупки одними и теми же покупателями. Сеть скомпилирована Валдисом Кребсом, однако не была опубликована
- **adjnoun**, $N = 112,\ L = 425$: сеть смежности частых прилагательных и существительных в романе «Давид Копперфильд» Чарльза Диккенса [27]
- **football**, $N = 115,\ L = 613$: сеть игр в американский футбол между колледжами из дивизиона IA во время регулярного сезона осенью 2000 года [28]
- **jazz**, $N = 198,\ L = 2742$: сеть джазовых музыкантов [29]
- **celegans_metabolic**, $N = 453,\ L = 2025$: метаболическая сеть Caenorhabditis elegans [30]
- **email**, $N = 1133,\ L = 5451$: сеть связей по электронной почте между членами Университета Ровира и Вирхилий [31]
- **polblogs**, $N = 1490,\ L = 16715$: сеть ссылок между интернет блогами о политике США в 2005 году [32]
- **netscience**, $N = 1589,\ L = 2742$: сеть соавторства среди учёных, работающих над сложными сетями [27]
- **PGPgiantcompo**, $N = 10680,\ L = 24316$: список узлов гигантской компоненты сети пользователей алгоритма Pretty-Good-Privacy для защищенного обмена информацией [33]
- **as-22july06**, $N = 22963,\ L = 48436$: структура интернета на уровне автономных систем на 22 июля 2006 года. Сеть создана Марком Ньюманом и не опубликована
- **cond-mat-2003**, $N = 31163,\ L = 120029$: сеть соавторства среди учёных, публиковавших препринты в определённые архивы между 1995 и 2003 годами [34]
- **caidaRouterLevel**, $N = 192244,\ L = 609066$: граф структуры интернета на уровне роутера, собранный ассоциацией CAIDA в апреле и мае 2003 года
- **cnr-2000**, $N = 325557,\ L = 2738969$: небольшая часть обхода итальянского домена .cnr [35, 36, 37]
- **eu-2005**, $N = 862664,\ L = 16138468$: небольшая часть обхода домена Европейского союза .eu [35, 36, 37]
- **in-2004**, $N = 1382908,\ L = 13591473$: небольшая часть обхода индийского домена .in [35, 36, 37]

Кроме того, некоторые тесты используют автоматически сгенерированные графы с заранее известным количеством сообществ. Такой граф имеет четыре параметра — количество узлов N, количество сообществ K (все сообщества одинаковых размеров), вероятность появления ребра между узлами одного сообщества p_1 и вероятность появления ребра между вершинами разных сообществ p_2. Преимущество автоматически сгенерированных графов заключается в проверке работы алгоритма на разных по размеру графах с разными по плотности сообществами. Так же из построения известны реальные сообщества. Однако графы, построенные по реальным системам могут сильно отличаться от подобных графов.

Так, далее в работе используется автоматически сгенерированный граф под названием *auto40*, со следующими параметрами: $N = 40,000,\ K = 40,\ p_1 = 0.1,\ p_2 = 5 \cdot 10^{-4}$.

2. Адаптивный рандомизированный жадный алгоритм

2.1. Применимость алгоритма SPSA

Для того, чтобы алгоритм SPSA был применим — необходимо иметь выпуклую функцию качества, которую необходимо минимизировать. В большинстве случаев модулярность результатов работы RG на разных графах с разными значениями параметра k будет выглядеть следующим образом:

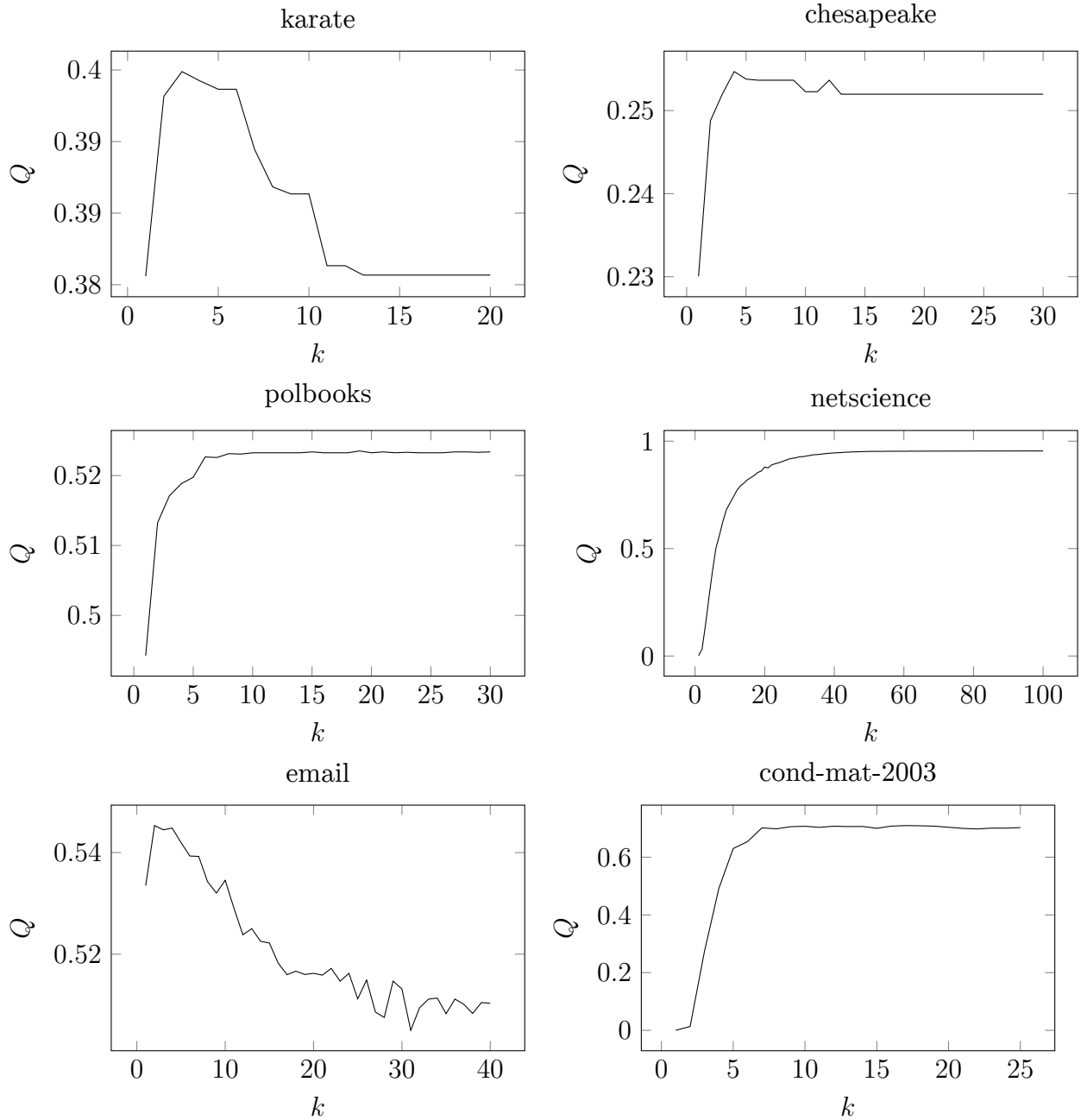

Рис. 1: Зависимость модулярности от k в результатах работы RG_k на шести тестовых графах

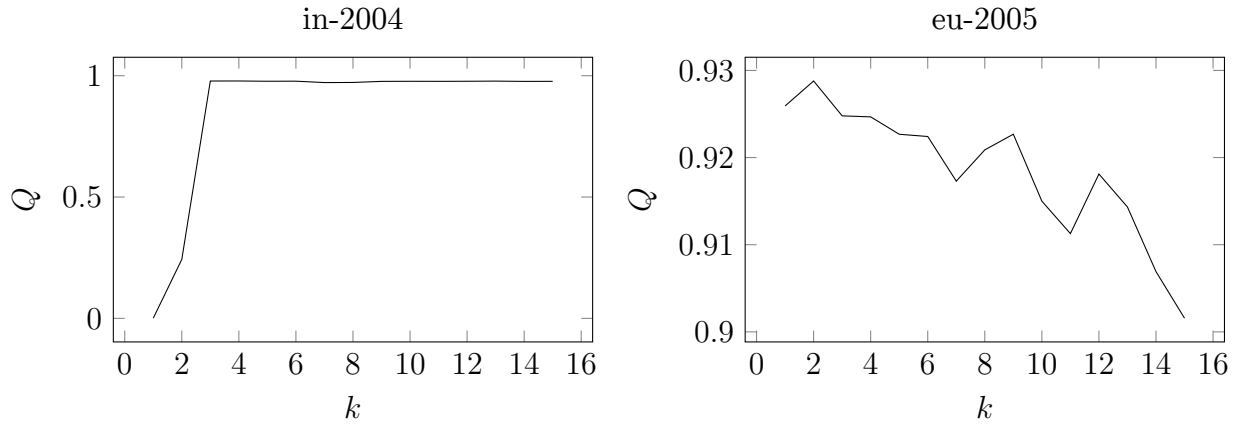

Рис. 2: Продолжение рисунка 1. Зависимость модулярности от k в результатах работы RG_k на графах *in-2004* и *eu-2005*

Значение k, при которым RG_k будет на графе принимать наилучшее значение, далее в работе называется *оптимальным* k или k_{opt}.

Рисунки 1 и 2 показывают разделение поведения RG на разных графах на два возможных случая: в первом алгоритм принимает наилучший результат при некотором небольшом, но разным для разных графов k_{opt} (например работа алгоритма на графе *karate* и на графе *eu-2005*). Во втором результаты алгоритма постепенно возрастают, приближаясь к некоторой асимптоте (хорошим примером будет работа алгоритма на графе *netscience*). К первому случаю так же относится такое поведение, в котором алгоритм быстро (с ростом k) достигает своего лучшего значения, и затем его результаты очень несильно падают, и в дальнейшем держатся того же значения (такое выполняется, например, на графе *in-2004*).

Похожее поведение алгоритм показывает на автоматически сгенерированных графах, но в таких графах можно получить менее предсказуемые значения k_{opt} и можно предположить, что будет происходить с модулярностью при дальнейшем росте k.

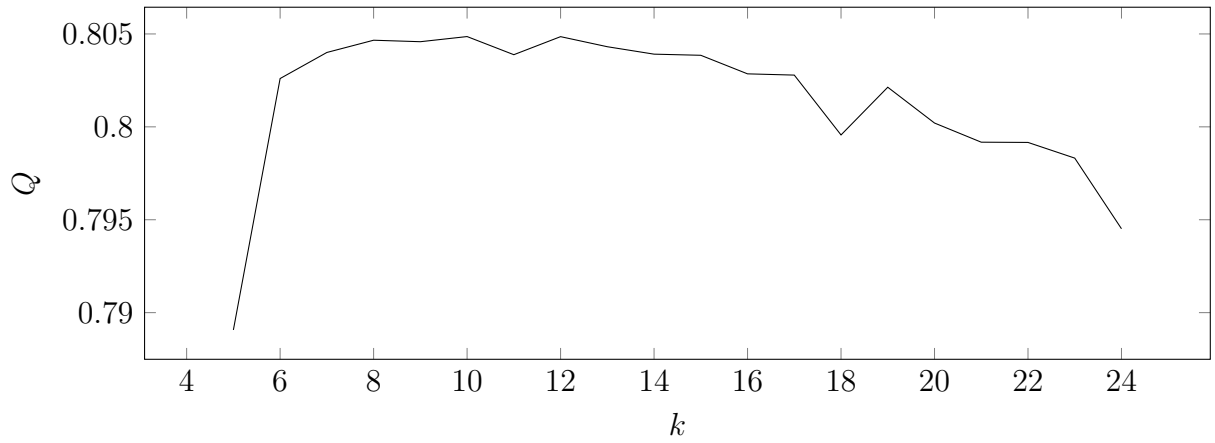

Рис. 3: Зависимость модулярности от k для разбиения на сообщества автоматически сгенерированного графа *auto40* с параметрами $N = 40.000,\ K = 40,\ p_1 = 0.1,\ p_2 = 5{\cdot}10^{-4}$ алгоритмом RG_k

2.2. Функция качества

Таким образом, имеет смысл использовать в функции качества не только модулярность, но и время. Подсчёт времени сам по себе занимает время и в разных случаях может давать сильно отличающиеся результаты. Теоретическая трудоёмкость алгоритма линейно зависит от параметра k, на реальных графах зависимость тоже близка к линейной:

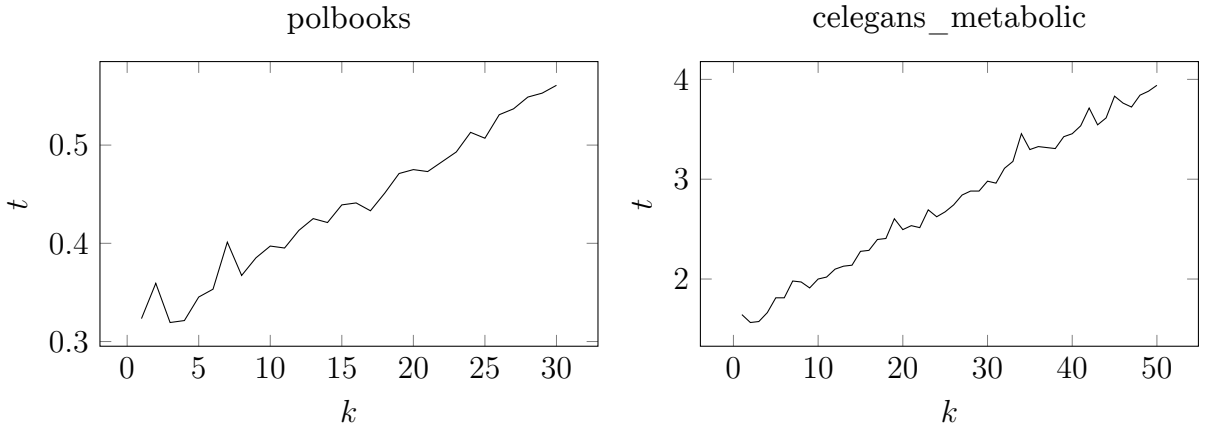

Рис. 4: Зависимость времени t в миллисекундах от k для результатов работы RG_k на графах *polbooks* и *celegans_metabolic*

Основываясь на этих данных вместо времени можно использовать значение k. Максимальное значение модулярности, которое может быть достигнуто на графе очень сильно отличается, поэтому нет смысла использовать абсолютное значение модулярности, но имеет смысл использовать относительные значения. При вычислении центральной точки (3) в алгоритме одновременно возмущаемой стохастической аппроксимации используется только разность функций качества. Поэтому если использовать в функции качества логарифмы от модулярности — разность функций укажет, во сколько раз модулярность изменилась.

Так же функция качества должна принимать минимум, а не максимум, поэтому первой версией подобной функции может быть $f(Q) = -\alpha \ln Q,\ \alpha > 0$. Для того, чтобы принимать во внимание время работы, разумно добавить логарифм от k:

$$f(Q,k) = -\alpha(\ln Q - \beta \ln k),\ \alpha > 0, \beta \geq 0 \tag{4}$$

Коэффициент β в таком случае можно рассматривать в следующем виде $\beta = \frac{\ln \gamma}{\ln 2}$, где коэффициент γ указывает, во сколько раз необходимо увеличиться модулярности для того, чтобы покрыть увеличение времени (то есть k) вдвое. Если время не имеет значение коэффициент γ принимает значение 1, а следовательно коэффициент β принимает значение 0. Коэффициент α же играет роль размера шага при выборе следующей центральной точки.

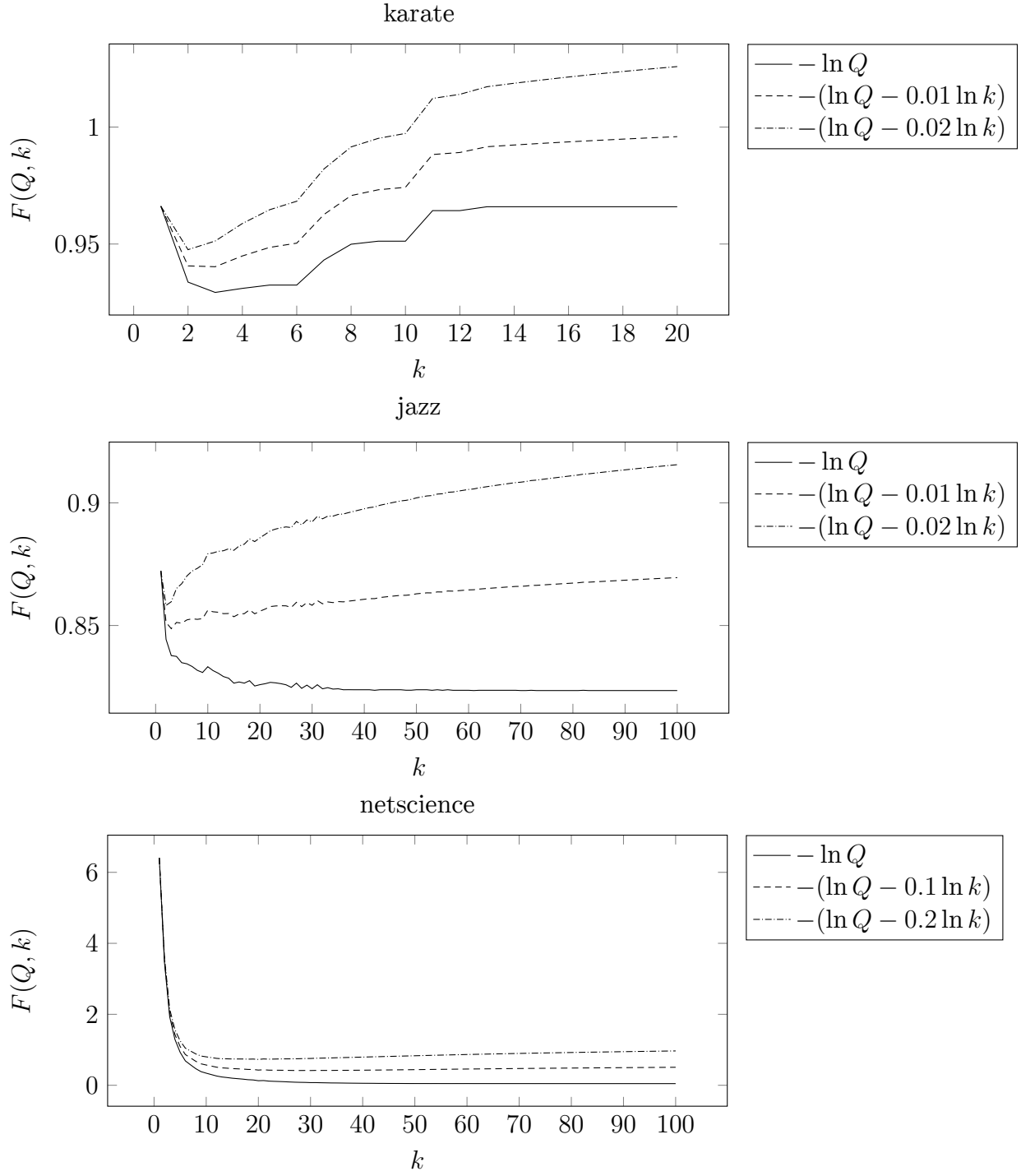

Рис. 5: Функции качества с разными коэффициентами β для графов *karate*, *jazz* и *netscience*

Как видно из функций качества с разными коэффициентами β для графа *netscience* на рисунке 5, иногда для того, чтобы функция качества имела минимум на небольших k, надо задавать довольно большое значение коэффициента β. Однако это логично, на данном графе если нас устраивает время работы — выгоднее всё время увеличивать значение k.

Так же заметно, что при хорошо подобранных β функция качества выпукла и имеет минимум.

2.3. Адаптивный 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. После каждого шага можно считать прирост модулярности, но вместо этого имеет смысл считать медиану прироста модулярности за σ итераций — так как 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. Выбор начальной центральной точки $\hat{k}_0 \in \mathbb{N}$, счётчик $n \leftarrow 0$, выбор размера возбуждения $d \in \mathbb{N}$, коэффициентов функции качества $\alpha, \beta \in \mathbb{R},\ \alpha > 0,\ \beta \geq 0$, и $\sigma \in \mathbb{N}$ — количество итераций в одном шаге

2. Увеличение счётчика $n \leftarrow n + 1$

3. Определение новых аргументов функции $k_n^- \leftarrow \max\{\hat{k}_{n-1} - d, 1\}$ и $k_n^+ \leftarrow \hat{k}_{n-1} + d$

4. Выполнение σ итераций с параметром k_n^-, вычисление медианы прироста модулярности μ_n^-

5. Выполнение σ итераций с параметром k_n^+, вычисление медианы прироста модулярности μ_n^+

6. Вычисление функций качества $y_n^- \leftarrow -\alpha(\ln \mu_n^- - \beta \ln k_n^-)$, $y_n^+ \leftarrow -\alpha(\ln \mu_n^+ - \beta \ln k_n^+)$

7. Вычисление следующей центральной точки

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

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).

2.4. Чувствительность к перепадам функции качества

Коэффициент α отвечает за то, насколько чувствителен алгоритм будет к перепадам функций качества — чем больше α, тем сильнее будет отличаться новая центральная точка от предыдущей в одной и той же ситуации. На трёх графах были измерены зависимости модулярности от параметра α при $d = 2,\ \sigma = 500,\ \hat{k}_0 = 10,\ \beta = 0$:

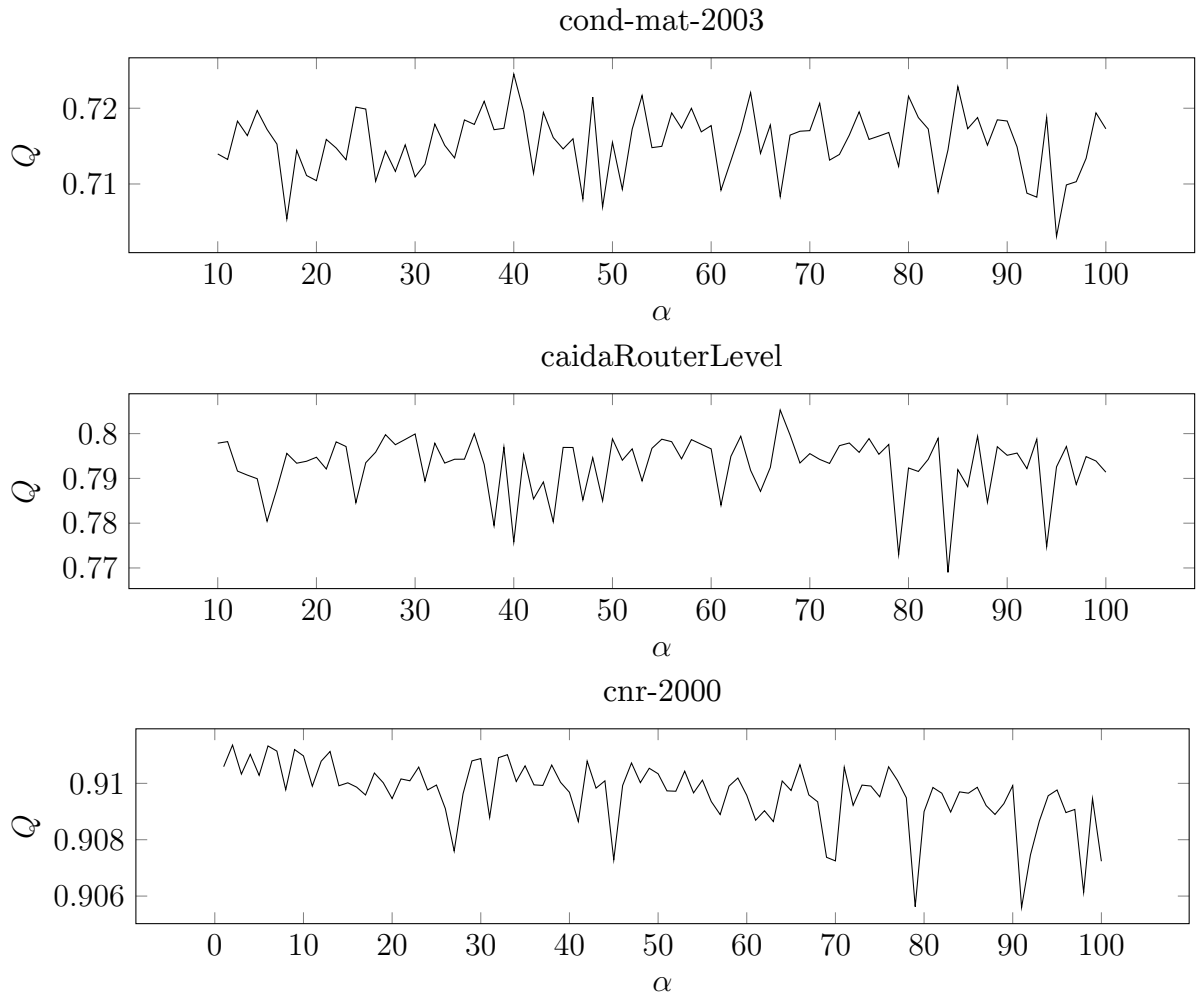

Рис. 6: Зависимость модулярности от параметра α в работе *ARG* на графах *cond-mat-2003*, *caidaRouterLevel*, *cnr-2000*

Из рисунка 6 видно, что результат не очень стабильный, хотя значение модулярности и колеблется на небольшом промежутке. Так же на графе *cnr-2000* заметно, что увеличение параметра α влечёт за собой уменьшение модулярности в среднем.

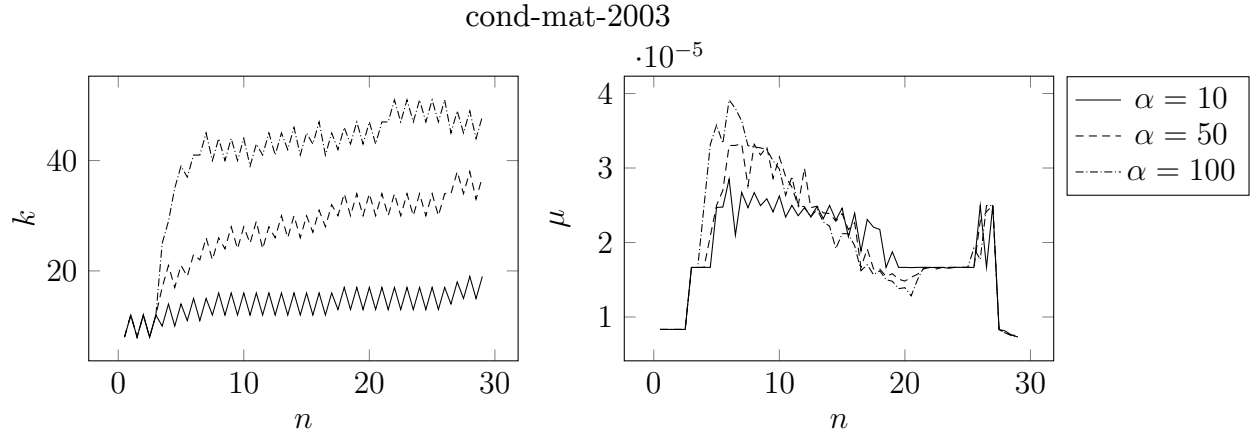

Рис. 7: Изменения параметра k и медианы приросты модулярности со временем для разных параметров α на графе *cond-mat-2003*

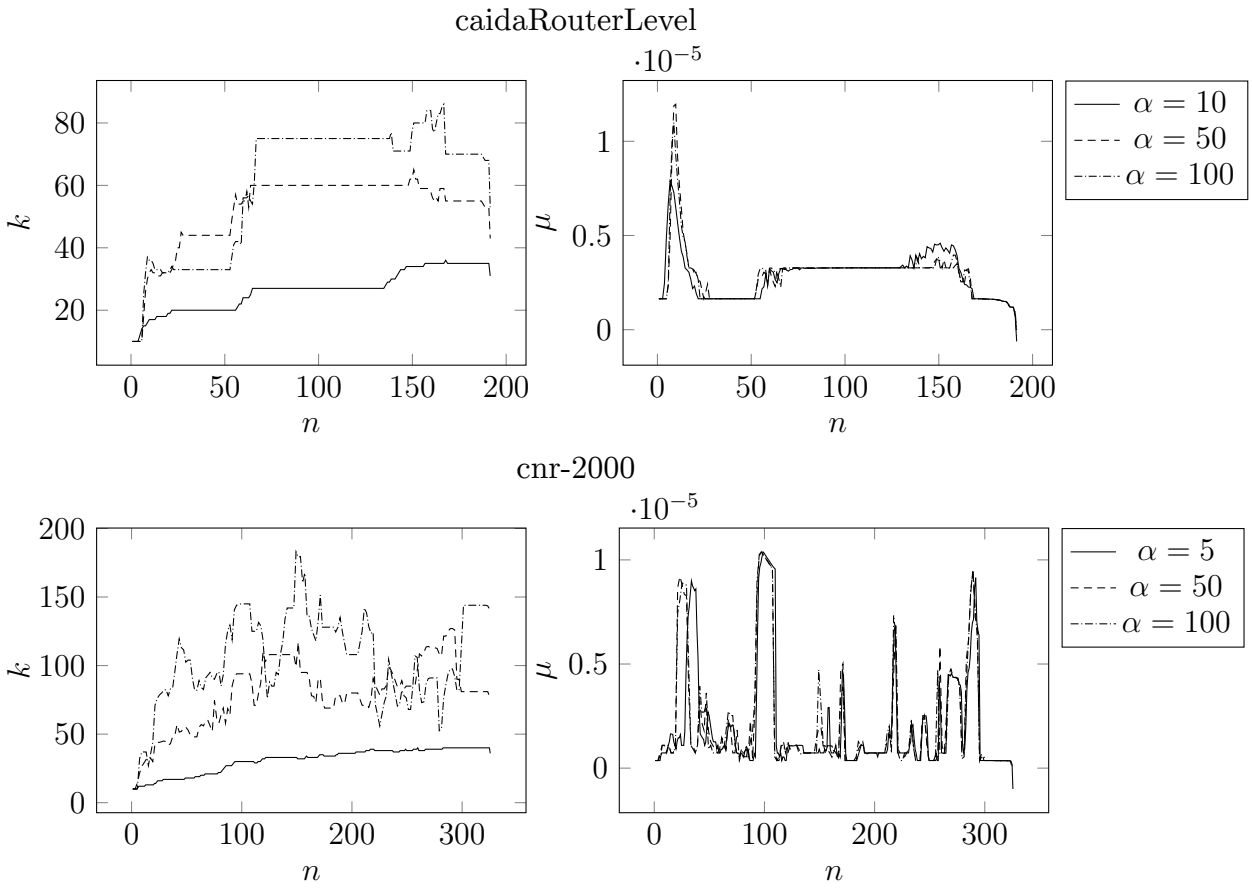

Рис. 8: Продолжение рисунка 7. Изменения параметра k и медианы приросты модулярности со временем для разных параметров α на графах *caidaRouterLevel* и *cnr-2000*. Графики смазаны для повышения читаемости

Из рисунков 7 и 8 видно, что в зависимости от α параметр k, с которым прогоняются шаги по σ шагов, и который поочередно принимает значения k_n^- и k_n^+, меняется с разной интенсивностью, но прирост модулярности при этом изменяется примерно одинаково.

2.5. Размер возмущения

Коэффициент d отвечает за то, насколько сильно будет возмущаться центральная точка для получения следующих измерений. То есть, насколько k_n^+ и k_n^- будут отличаться от $\hat{k}_{n-1}$. На рисунке 14 можно увидеть зависимость модулярности от размера возмущения при $f(\mu, k) = -10 \ln \mu,\ \sigma = 500,\ \hat{k}_0 = 10$.

На графах *cnr-2000* и *eu-2005* значения модулярности не очень сильно менялись в зависимости от параметра d, хотя некоторые значения d и давали более большие значения. Однако на графах *cond-mat-2003* и *caidaRouterLevel* после некоторого порогового значения возмущения модулярность показывала, что получившееся разбиение не лучше случайного.

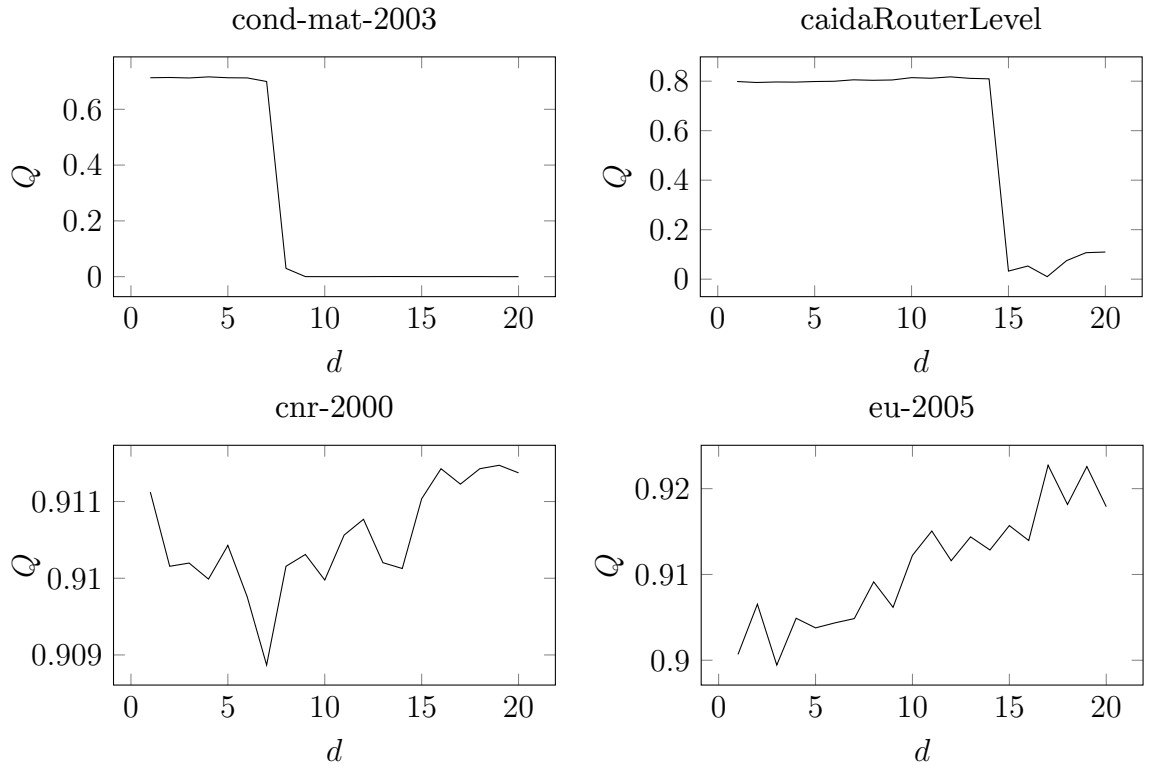

Рис. 14: Зависимость модулярности от размера возмущения на четырёх графах

2.6. Количество итераций в одном шаге

Параметр σ указывает, как часто меняется k в ходе работы ARG. Так как в функции качества используется медиана прироста модулярности, а не прирост модулярности за все σ итераций — при изменении σ нет необходимости менять функцию качества, модулярность прироста будет оставаться приблизительно такой же по величине, в то время как прирост модулярности линейно зависит от σ. Зависимость модулярности от количества итераций в одном шаге при $d = 5,\ f(\mu, k) = -10 \ln \mu,\ \hat{k}_0 = 10$ принимает такой вид:

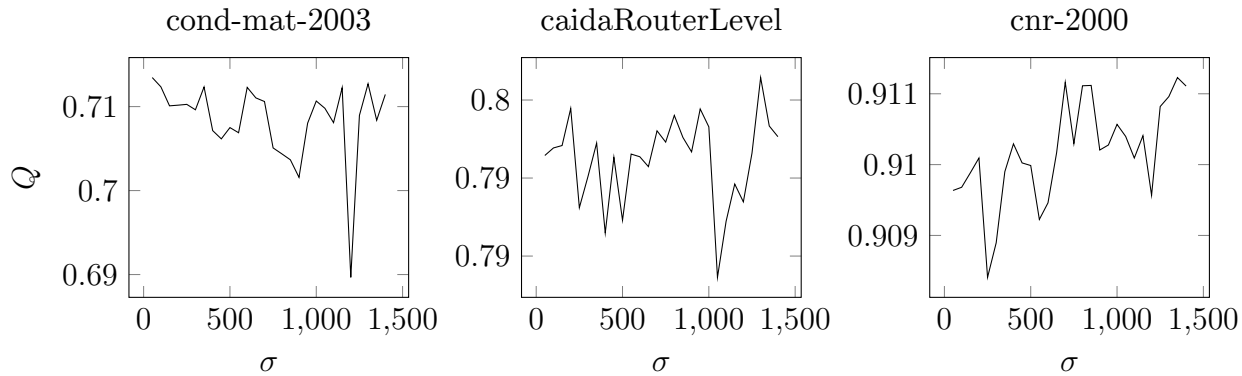

Рис. 15: Зависимость модулярности от значения σ на трёх графах. σ принимает значения от 50 до 1400

На всех трёх графах модулярность несильно, но непредсказуемо меняется при

изменении σ. Во всех случаях есть хорошие и плохие значения σ, но в целом значения параметра достаточно равнозначны.

2.7. Начальная центральная точка

Начальная центральная точка указывает, от какой точки будут строиться k_1^- и k_1^+, с которыми будут проходить первые и вторые σ итераций, соответственно. Зависимость модулярности от начальной центральной точки $\hat{k}_0$ при $d = 5,\ f(\mu, k) = -10 \ln \mu,\ \sigma = 1000$ будет выглядеть следующим образом:

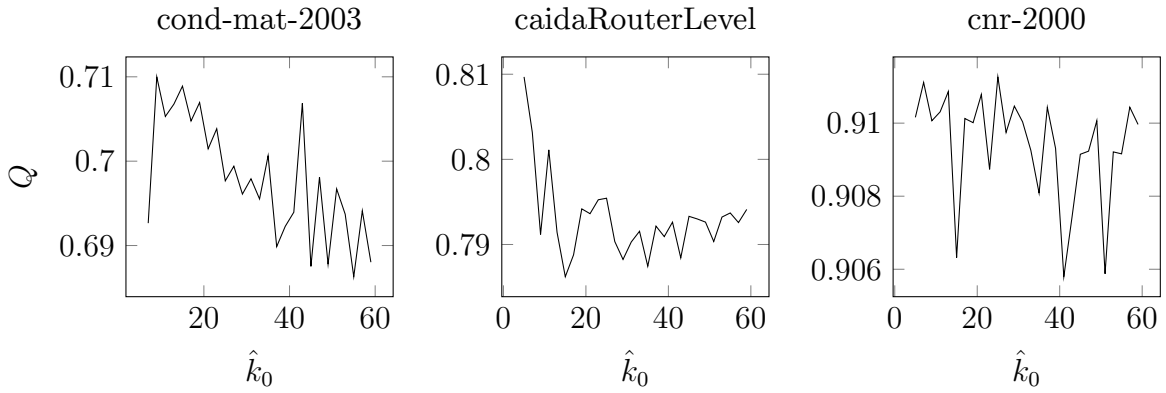

Рис. 16: Зависимость модулярности от значения $\hat{k}_0$ на трёх графах

Как можно заметить, на разных графах модулярность ведёт себя по разному в зависимости от $\hat{k}_0$. График для графа *cond-mat-2003* на рисунке 16 начинается от первого вхождения $\hat{k}_0 = 7$ (остальные от $\hat{k}_0 = 5$), так как при меньших значениях модулярность очень сильно падает. Это связано с тем, что $d = 5$ и следовательно $k_1^- = 1$, в то время как RG_1 часто работает сильно хуже RG_k с параметром $k > 1$. Однако, как видно на рисунке, выгоднее использовать небольшое значение $\hat{k}_0$, хоть и большее $d + 1$.

2.8. Коэффициент β функции качества

Коэффициент β функции качества (4) указывает на то, насколько много внимания алгоритм обращает на величину точек k_n^- и k_n^+ при выборе следующей центральной точки. Предполагалось, что чем больше β, тем быстрее будет работать алгоритм, однако возможно будет находить более плохие разбиения. На рисунке 17 изображены результаты работы ARG с параметрами $d = 5,\ \sigma = 1000,\ \hat{k}_0 = 8$, функция качества $f(\mu, k) = -10(\ln \mu - \beta \ln k)$ с разными значениями β на графе *cond-mat-2003*.

Время работы сначала увеличивается на небольших значениях параметра β, а затем постепенно падает, в то время как модулярность при уменьшении β сначала постепенно падает, но после некоторого переломного значения β сильно падает.

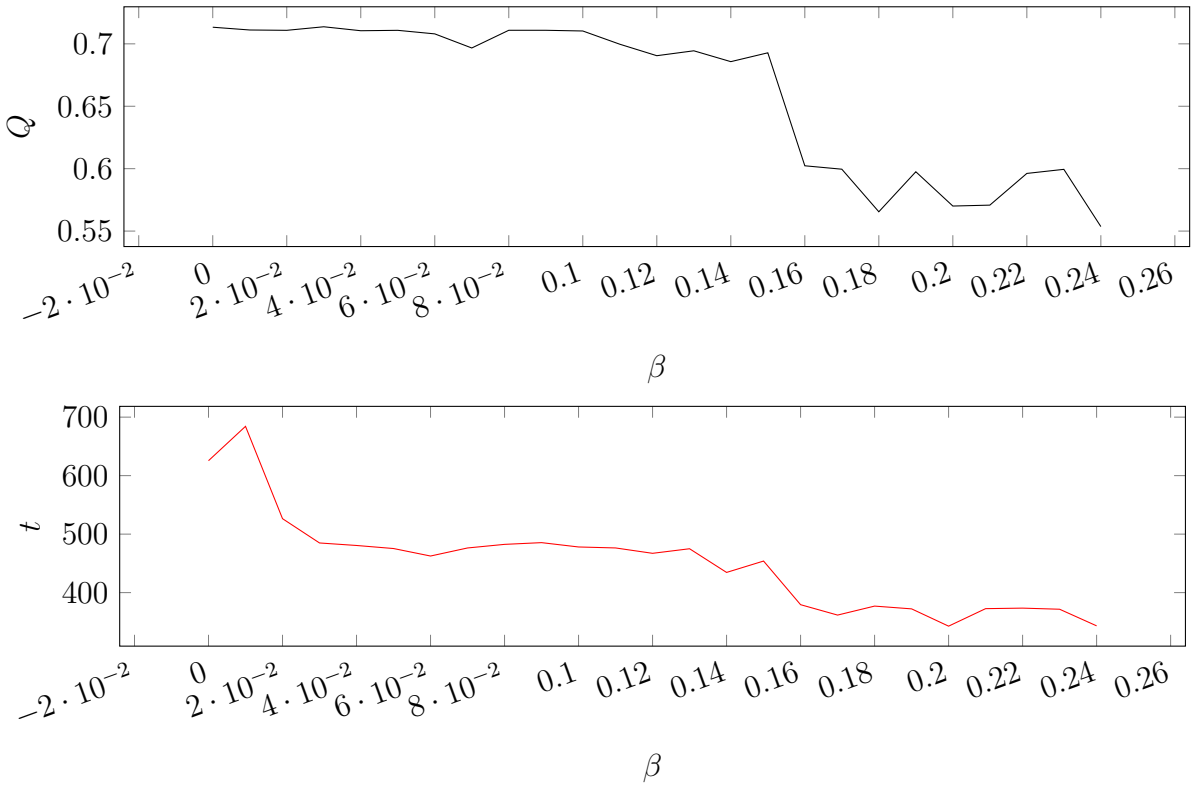

Рис. 17: Зависимость модулярности и времени от параметра функции качества β на графе *cond-mat-2003*

2.9. Сравнение

На таблицах 1 и 2 показано сравнение результатов работы рандомизированного жадного алгоритма и адаптивного рандомизированного жадного алгоритма. В качестве входных данных для рассмотрения выбраны 7 тестовых графов.

Рандомизированный жадный алгоритм в таблицах обозначается как RG_k. Адаптивный рандомизированный жадный алгоритм обозначается как ARG_β с параметром функции качества β в индексе, остальные параметры при этом равны $d = 5$, $f(\mu, k) = -10(\ln \mu - \beta \ln k)$, $\sigma = 1000$, $\hat{k}_0 = 8$. Такой набор параметров не гарантирует наилучший результат на каждом графе или даже наилучший в среднем по всем графам результат для адаптивного алгоритма, однако показывает неплохие результаты.

Оба алгоритма — рандомизированные, поэтому на каждом отдельном запуске результат может оказаться очень хорошим или сравнительно плохим, однако даст мало сведений о качестве алгоритма. Поэтому для каждого графа производилось определённое количество запусков, и в качестве модулярности бралась медиана значений, а в качестве времени — среднее, для повышения точности измерения времени.

Для графа *as-22july06* производился 101 запуск, для *cond-mat-2003* — 51 запуск, для *caidaRouterLevel* и *cnr-2000* — 11 запусков, для *in-2004* — пять запусков, и для *eu-2005* — три запуска.

Кроме того, в таблицу вошли результаты работы над автоматически сгенериро-

ванным графом *auto40*, который уже появлялся в работе, к примеру в рисунке 3. Для этого графа для повышения точности производилось 11 запусков.

Таблица 1: Модулярности разбиений, полученных в результате работы рандомизированного жадного алгоритма и адаптивного рандомизированного жадного алгоритма с разными параметрами на тестовых графах

	RG_1	RG_3	RG_{10}	RG_{50}	ARG_0	$ARG_{0.01}$	$ARG_{0.05}$	$ARG_{0.1}$	$ARG_{0.2}$
as-22july06	0.65281	0.64658	0.64024	0.63479	0.64262	0.64041	0.64264	0.64134	0.64192
cond-mat-2003	0.00012	0.19727	0.70738	0.69403	0.71129	0.71232	0.71193	0.69749	0.56631
auto40	0.78944	0.79988	0.80417	0.80273	0.80136	0.80145	0.80174	0.80152	0.80102
caidaRouterLevel	0.01938	0.81101	0.79883	0.79300	0.79970	0.80114	0.80216	0.80059	0.80176
cnr-2000	0.90237	0.91192	0.91144	0.90997	0.91028	0.91041	0.91039	0.91108	0.91075
eu-2005	0.92765	0.92559	0.91780	0.90416	0.91062	0.91047	0.91048	0.91199	0.91242
in-2004	0.00026	0.97836	0.97185	0.97596	0.97614	0.97615	0.97616	0.97618	0.97588

Таблица 2: Время работы (в миллисекундах) рандомизированного жадного алгоритма и адаптивного рандомизированного жадного алгоритма с разными параметрами на тестовых графах

	RG_1	RG_3	RG_{10}	RG_{50}	ARG_0	$ARG_{0.01}$	$ARG_{0.05}$	$ARG_{0.1}$	$ARG_{0.2}$
as-22july06	177	189	231	464	238	241	238	233	222
cond-mat-2003	58	184	463	931	477	477	474	476	351
auto40	4,652	4,591	6,017	12,558	6,526	6,807	6,479	6,428	6,105
caidaRouterLevel	852	9,114	10,244	15,217	11,573	11,607	11,514	11,509	11,220
cnr-2000	26,083	26,056	27,137	33,592	30,033	30,465	29,054	29,971	29,784
eu-2005	202,188	200,686	207,689	246,170	233,761	226,869	225,748	226,427	266,038
in-2004	9,208	487,953	553,196	607,408	622,813	625,124	617,345	640,454	616,187

В большинстве случаев один или несколько параметров k дают рандомизированному жадному алгоритму лучшие результаты, чем результаты адаптивного рандомизированного алгоритма, однако адаптивный вариант даёт более стабильные результаты. Можно заметить, что небольшие значения параметра β дают лучшие результаты, чем нулевое значение, а при более бóльших значениях параметра время работы алгоритма уменьшается, но незначительно. В таблицу не попали результаты алгоритмов со значениями β больше 0.2, где время работы действительно сильно снижалось, однако и значения модулярности получались слишком маленькими.

ARG можно сравнить с RG_{10}, так как он тоже даёт стабильные результаты, в отличии от $RG_1,\ RG_3$. Однако заметно, что ARG в большинстве случаев даёт бóльшую модулярность.

3. Адаптивная схема кластеризации основных групп графа

3.1. ARG в схеме кластеризации основных групп графа

Схема кластеризации основных групп графа ($CGGC$) была более подробно описана в подразделе 1.4, однако общая схема работы заключается в том, что сначала s начальных алгоритмов выделяют сообщества, а те узлы, в которых они разошлись по мнению, затем разбивает на сообщества финальный алгоритм. Трудоёмкость таким образом складывается из s трудоёмкостей начальных алгоритмов и трудоёмкости 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.

Далее разбиение, полученное начальными алгоритмами называется *начальным* разбиением, полученное финальным алгоритмом — *финальным* разбиением. Также разбиение, равное максимальному перекрытию начальных разбиений далее называется *промежуточным*.

Если промежуточное разбиение состоит из достаточно большого количества сообществ (как минимум в несколько раз больше 2σ) — на нём разумно использовать в качестве финального алгоритма ARG, чтобы получать стабильные результаты. Однако на небольшом количестве узлов или сообществ ARG не имеет смысла.

Так же, если граф, подающийся на вход, достаточно большой — в качестве начального алгоритма на нём имеет смысл использовать адаптивный рандомизированный жадный алгоритм.

В таблице 3 сравниваются результаты работы $CGGC$ с разными начальными и финальными алгоритмами. В качестве начального алгоритма используюся два рандомизированных жадных алгоритма с параметрами $k = 3$ и $k = 10$ (обозначаются как и выше в работе RG_k, со значением параметра k в индексе), и адаптивный рандомизированный жадный алгоритм с параметрами $d = 5,\ f(\mu, k) = -10(\ln \mu - 0.02 \ln k),\ \sigma = 1000,\ \hat{k}_0 = 8$ (обозначается ARG). А в качестве финального алгоритма используются только RG_3 и RG_{10}, так как на рассматриваемых графах промежуточное разбиение состоит из недостаточно большого количества сообществ, чтобы использовать ARG.

$CGGC$ используется с параметром $s = 10$, начальный алгоритм обозначается как A_{init}, а финальный как A_{final}. Для повышения точности $CGGC$ с разными начальными и финальными алгоритмами запускались на графе *cond-mat-2003* запускались 21 раз, на *caidaRouterLevel* и *auto40* — три раза.

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

Граф *cond-mat-2003* плохо разбивается случайными жадными алгоритмами с маленьким k, 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.

3.2. Адаптивное построение промежуточного разбиения

Так как адаптивный рандомизированный жадный алгоритм не может работать на графах с маленьким количеством узлов или на разбиениях с маленьким количеством сообществ — была предложена другая схема адаптивной кластеризации основных групп графа:

1. На вход подаётся граф G. Выбор параметров алгоритма: $d \in \mathbb{N}$ — размер возмущения, $f(Q, k) : \mathbb{R} \times \mathbb{N} \to \mathbb{R}$ — функция качества, $l \in \mathbb{N}$ — количество шагов, $r \in (0, 1]$ — параметр, отвечающий за количество лучших начальных разбиений,

учавствующих в создании промежуточного разбиения. И наконец $\hat{k}_0 \in \mathbb{N}$ — начальная центральная точка. Инициализация счётчика $n \leftarrow 0$и списка начальных разбиений $S \leftarrow \emptyset$

2. Увеличение счётчика $n \leftarrow n + 1$

3. Вычисление следующих параметров рандомизированного жадного алгоритма $k_n^- \leftarrow \max\{1, \hat{k}_{n-1} - d\}$ и $k_n^+ \leftarrow \hat{k}_{n-1} + d$

4. Выделение сообществ в графе G рандомизированным жадным алгоритмом с параметром $k = k_n^-$, результирующее разбиение P_n^- записывается в список S

5. Выделение сообществ в графе G рандомизированным жадным алгоритмом с параметром $k = k_n^+$, результирующее разбиение P_n^+ записывается в список S

6. Вычисление функций качества $y_n^- \leftarrow f(Q_n^-, k_n^-),\ y_n^+ \leftarrow f(Q_n^+, k_n^+)$

7. Вычисление следующей центральной точка аналогично (5)

$$\hat{k}_n \leftarrow \max\left\{1, \left[\hat{k}_{n-1} - \frac{y_n^+ - y_n^-}{k_n^+ - k_n^-}\right]\right\} \tag{6}$$

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

9. Существует два способа выбора лучшие разбиения: *пропорциональный выбор*, по которому выбираются $\lceil r|S| \rceil = \lceil 2rl \rceil$ лучших разбиений (с наибольшей модулярностью).

   Второй, *непропорциональный*, выбор заключается в отбрасывании всех разбиений на сообщества с модулярностью $Q \leq (1 - r)Q_{best}$, где Q_{best} — модулярность лучшего разбиения в S. Из выбранных разбиений создается максимальное разбиение, которое затем передаётся в финальный алгоритм

При этом граф G может быть иметь сколь угодно маленькое количество узлов.

Далее в работе схема кластеризации основных групп графа с адаптивным построением промежуточного разбиения будет называться *адаптивной схемой кластеризации основных групп графа* ($ACGGC$).

3.3. Размер возмущения и чувствительность к перепадам функции качества

Функцией качества для построения промежуточного разбиения выбрана $f(Q, k) = -\alpha(\ln Q - \beta \ln k)$. Мотивация этого выбора аналогична выбору функции качества для

ARG, описанная в подразделе 2.2. Аналогично, параметр α указывает на чувствительность алгоритма к разностям значений функции качества, а параметр β указывает на значимость времени работы алгоритма.

На рисунке 18 изображены результаты адаптивной стратегии с параметром $\beta = 0$ и остальными параметрами равными $f(Q, k) = -\alpha \ln Q$, $\hat{k}_0 = 5$, $l = 10$, $r = 1$. В качестве финального алгоритма используется RG_{10}.

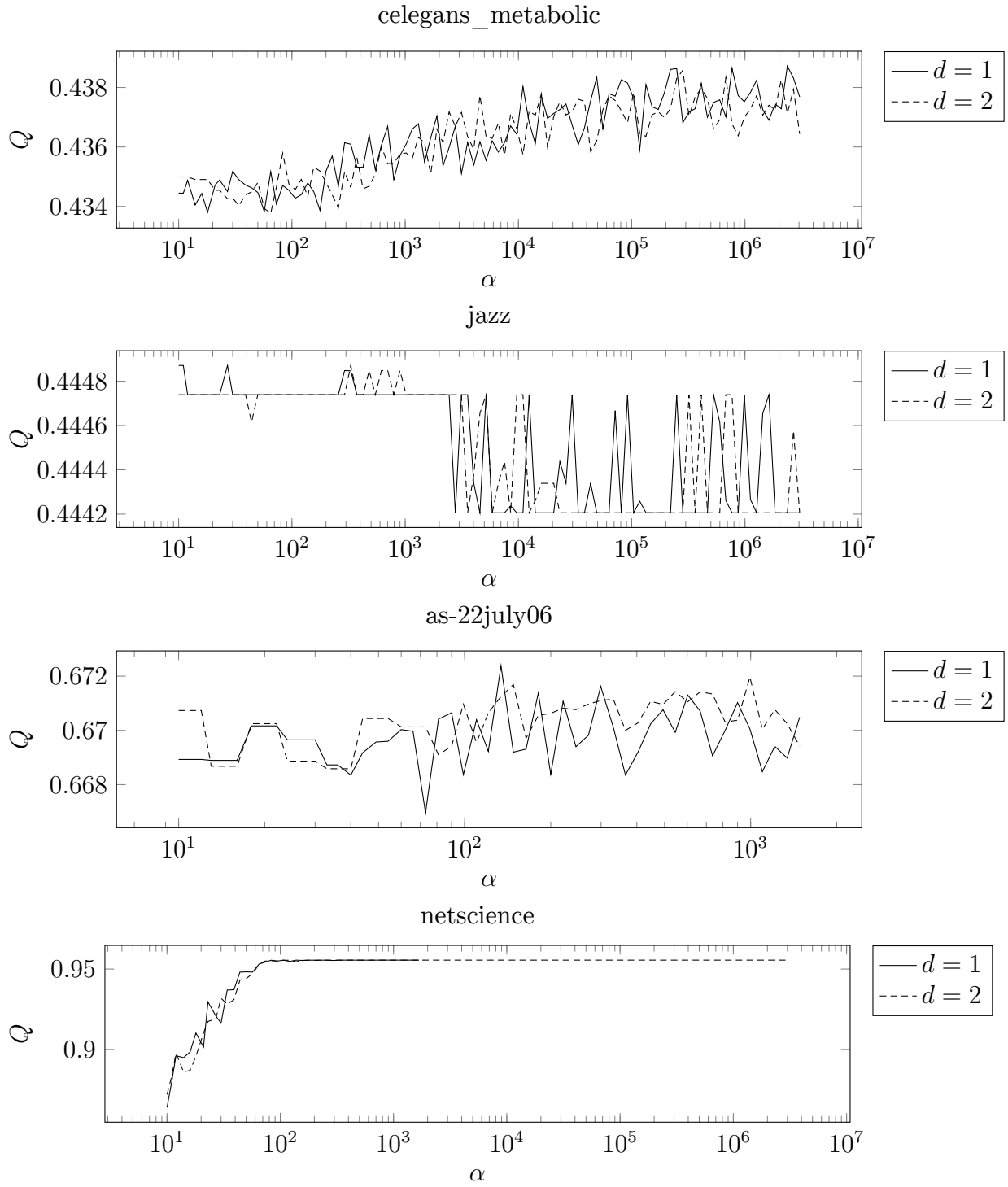

Рис. 18: Зависимость модулярности от параметра α при $d = 1$ и $d = 2$ в работе $ACGGC$ на четырёх графах

Значения модулярности колеблются в очень небольшом промежутке и на вышеприведённых четырёх графах значение α, примерно равное 10^3 даёт в среднем лучшие результаты. При этом на графах *celegans_ metabolic*, *jazz* и *netscience* зависимости модулярности от α выглядят одинакого для $d = 1$ и $d = 2$, а на графе *as-22july06* зависимость при параметре $d = 2$ лежит в среднем выше, чем при параметре $d = 1$. Бóльшие значения параметра d чреваты тем, что алгоритм будет слишком сильно отклоняться от оптимального значения параметра k_{opt} (то есть при котором RG в среднем будет получать лучшие результаты).

На рисунке 19 представлено изменение k_n^- и k_n^+ в ходе построения промежуточного разбиения от номера шага при $\alpha = 994$ на графе *as-22july06* и $\alpha = 1020$ на графе *netscience*, в обоих случаях $d = 2$, а остальные параметры те же, что и на рисунке 18. Отдельно измерено, что наилучшее в среднем значение рандомизированный жадный алгоритм даст при $k = 1$ на графе *as-22july06* и при очень больших k на графе *netscience*.

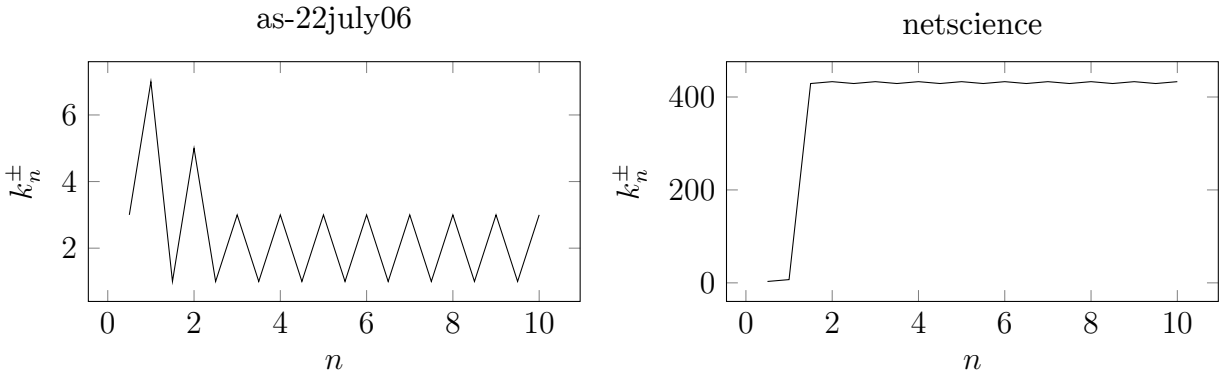

Рис. 19: Изменение k_n^- и k_n^+ с изменением номера шага в работе $ACGGC$ на графах *as–22july06* и *netscience*

Заметно, как быстро алгоритмы находят близкие к k_{opt} центральные точки $\hat{k}_n$.

3.4. Начальная центральная точка

Начальная центральная точка $\hat{k}_0$ указывает, рядом с какой точкой будут находиться параметры первых двух запусков рандомизированного жадного алгоритма. Очень большим делать начальную центральную точку не имеет смысла, так как довольно много графов имеет небольшой оптимальный параметр.

Однако, как показывает рисунок 20, на самом деле модулярность очень слабо зависит от начальной центральной точки. Для получения рисунка $ACGGC$ запускалась с параметрами $d = 2,\ f(Q,k) = -1000 \ln Q,\ l = 10,\ r = 1$, в качестве финального алгоритма использовался RG_{10}.

На графе *as-22july06* при увеличении начальной центральной точки модулярность понижалась, однако изменения были достаточно небольшими, а так же на других графах такой зависимости выявлено не было.

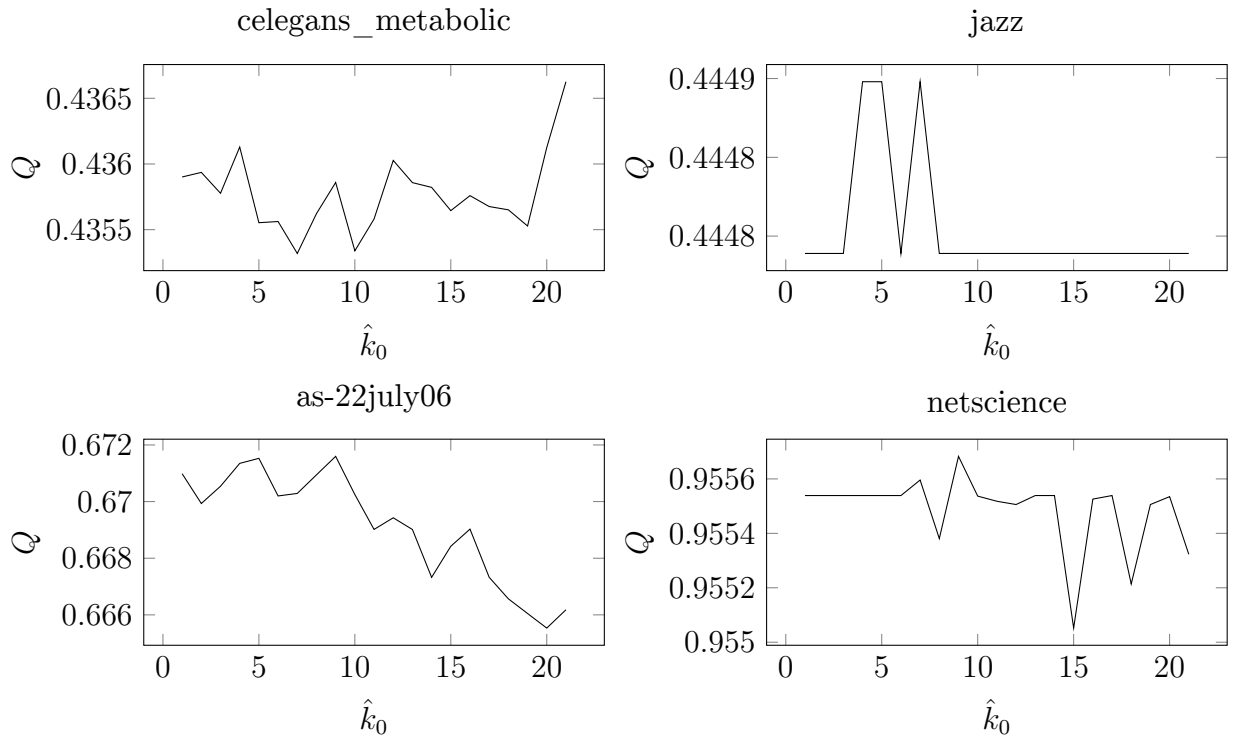

Рис. 20: Зависимость модулярности от начальной центральной точки $\hat{k}_0$ в работе $ACGGC$ на четырёх графах

3.5. Количество шагов и доля хороших разбиений

Множество S состоит из $2l$ разбиений, из которых затем выбирается l_2 лучших разбиений двумя способами в зависимости от параметра r. Пропорциональный выбор заключается в отборе ровно $\lceil 2rl \rceil$ лучших разбиений, а непропорциональный выбор заключается в отборе всех разбиений, у которых модулярность строго больше $(1 - r)Q_{best}$, где Q_{best} — модулярность лучшего разбиения в множестве S.

Затем l_2 лучших разбиений учавствуют в создании промежуточного разбиения. Предполагается, что при слишком маленьких l алгоритм не успеет найти оптимальную точку k, в то время как при $r \to 1$ в создании промежуточного разбиения примут участие много плохих разбиений, полученных на первых шагах. Зависимость модулярности от доли хороших разбиений при $d = 2,\ f(Q, k) = -1000 \ln Q,\ \hat{k}_0 = 5,\ l = 10$ и финальном алгоритме RG_{10} изображена на рисунке 21.

Из рисунка видно, что маленькие r в пропорциональном выборе дают плохие разбиения, что логично, так как в этом случае выбирается очень мало разбиений. В то время как в непропорциональном выборе даже при маленьких r (на рисунке 21 наименьшее значение $r = 0.05$) значения достаточно хорошие. Это обусловлено тем, что есть достаточно много разбиений, близких к лучшему. Затем на разных графах алгоритмы ведут себя по разному, хотя и разница между модулярностями при разных значениях r небольшия. Однако непропорциональный выбор ведёт себя более предсказумо на разных графах, поэтому такой выбор более рекомендуем и далее в работе

будет рассматриваться только непропорциональный выбор лучших разбиений.

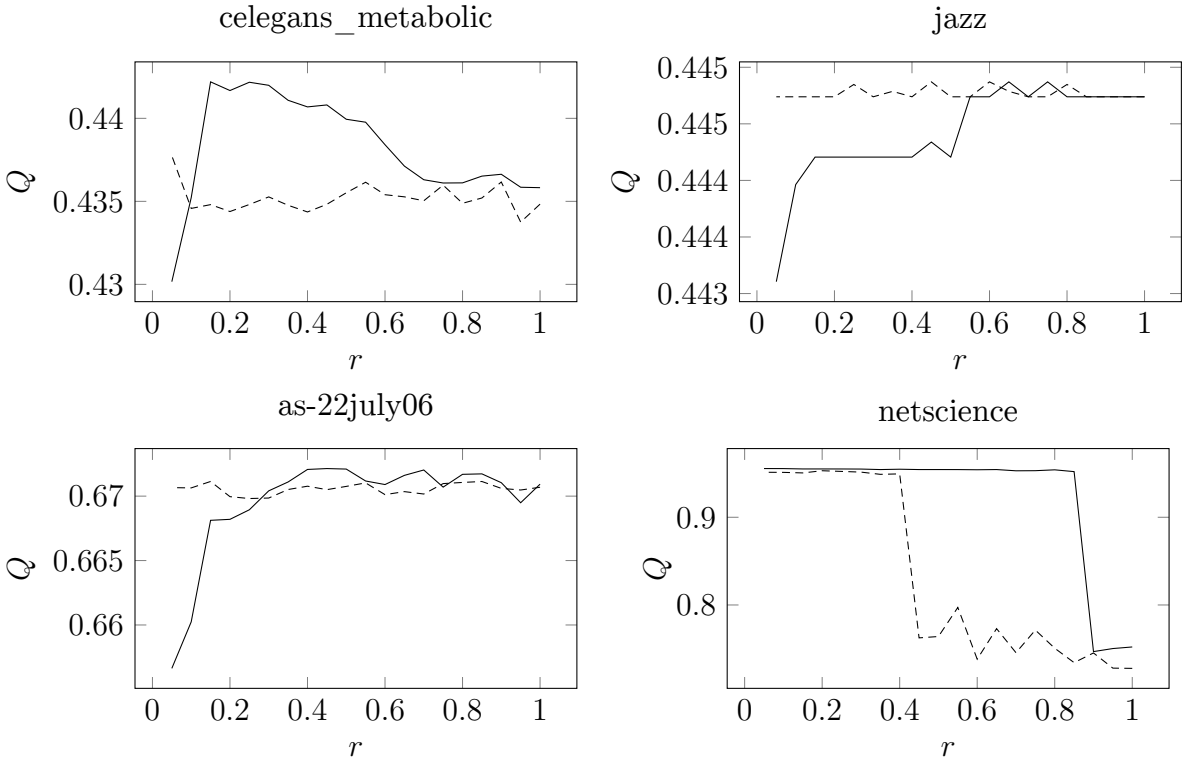

Рис. 21: Зависимость модулярности от доли хороших разбиений, учавствующих в создании промежуточного разбиения, в работе $ACGGC$ на четырёх графах. Сплошной линией отмечен пропорциональный выбор, а прерывистой — непропорциональный

На рисунках 22 и 23 изображены зависимость времени работы и модулярности от количества шагов l при $r = 0.05$ и $r = 1$. Остальные параметры при этом были равны $d = 2,\ f(Q,k) = -1000 \ln Q,\ \hat{k}_0 = 5$. В качестве финального алгоритма, как и в предыдущих примерах, использовался RG_{10}.

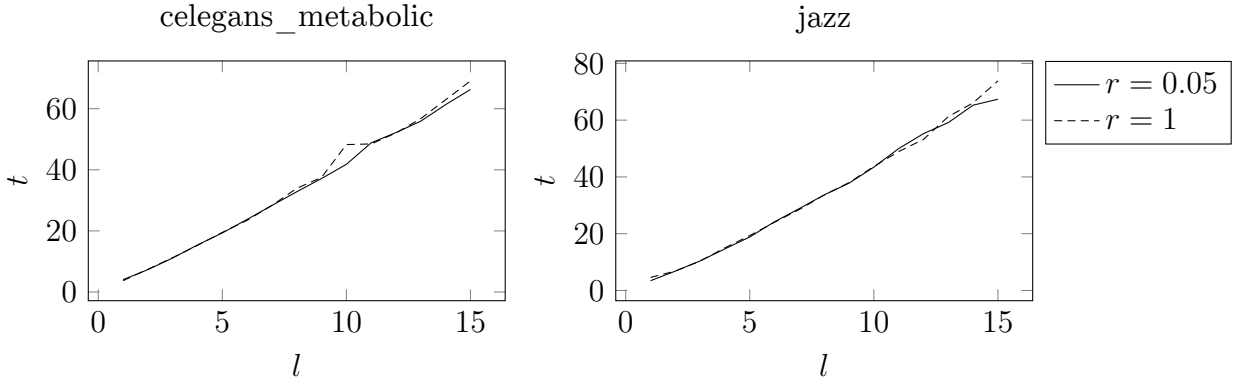

Рис. 22: Время работы от количества шагов l в работе $ACGGC$ на четырёх графах при $r = 0.05$ и $r = 1$

Стоит отметить, что на графах *jazz*, *as-22july06* модулярности при разных r практически не отличались, а также начиная с некоторого l на этих двух графах и на графе *netscience* модулярность не росла. На графе *celegans_metabolic* модулярность

достигала некоторого оптимального l и затем падала. При этом время работы растёт практически линейно от l, как можно увидеть на рисунке 22.

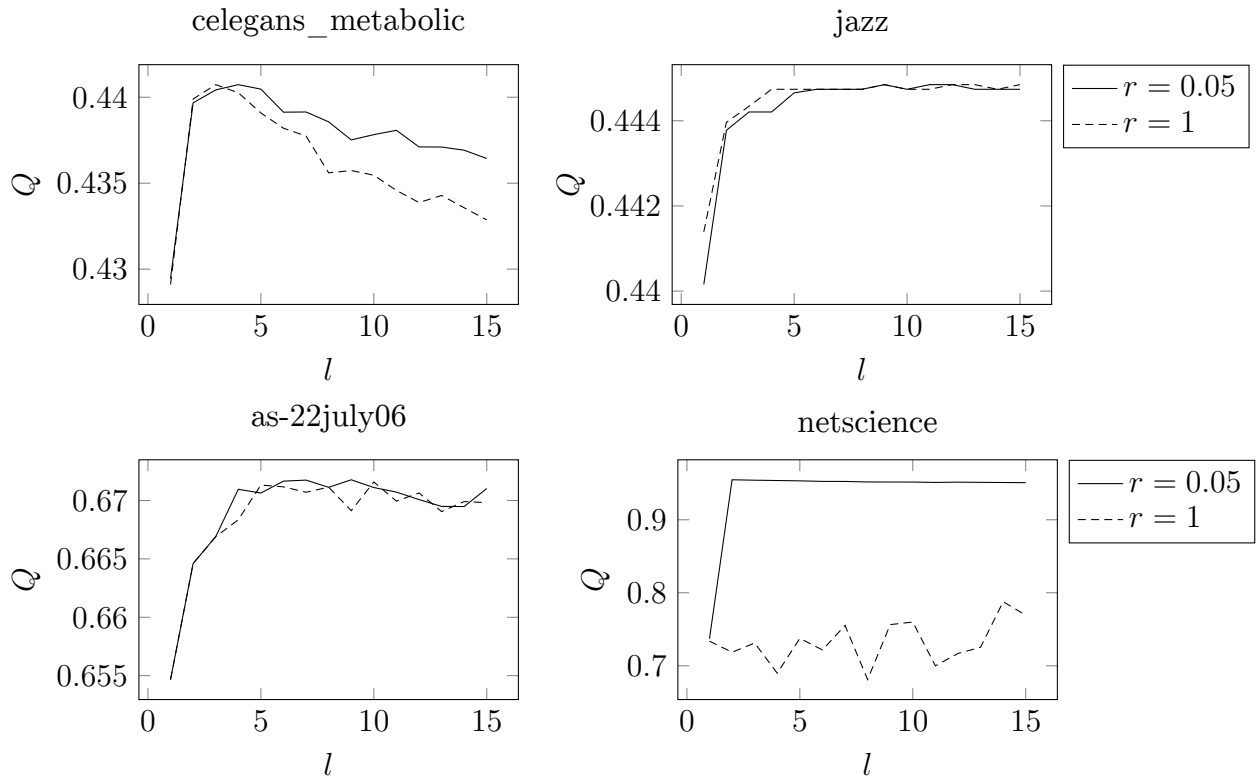

Рис. 23: Зависимость модулярности от количества шагов l в работе $ACGGC$ на четырёх графах при $r = 0.05$ и $r = 1$

Таким образом, нет смысла брать большое l, повышая время работы, но не гарантируя улучшения получившихся разбиений.

3.6. Время работы

Одним из способов влиять на время работы является увеличение коэффициента β в функции качества $f(Q, k) = -\alpha(\ln Q - \beta \ln k)$. Осмысленно сокращать время работы, так как при наличии хорошего финального алгоритма, или при использовании итеративной схемы кластеризации основных групп графа, время работы начальных алгоритмом очень критично, в то время как увеличение модулярности начальных алгоритмов на небольшие значения может не принести большой выгоды глобально. Стоит отметить, что на графах, у которых существует ярко выраженная оптимальная точка k, увеличение β не сильно повлияет на модулярность, но зато и не будет влиять на время работы. На графах, у которых с увеличением k растёт модулярность, увеличение β снижает модулярность и время работы.

Рисунок 27 был получен при применении $ACGGC$ с параметрами $d = 2,\ f(Q, k) = -1000(\ln Q - \beta \ln k),\ \hat{k}_0 = 5,\ l = 6,\ r = 0.05$.

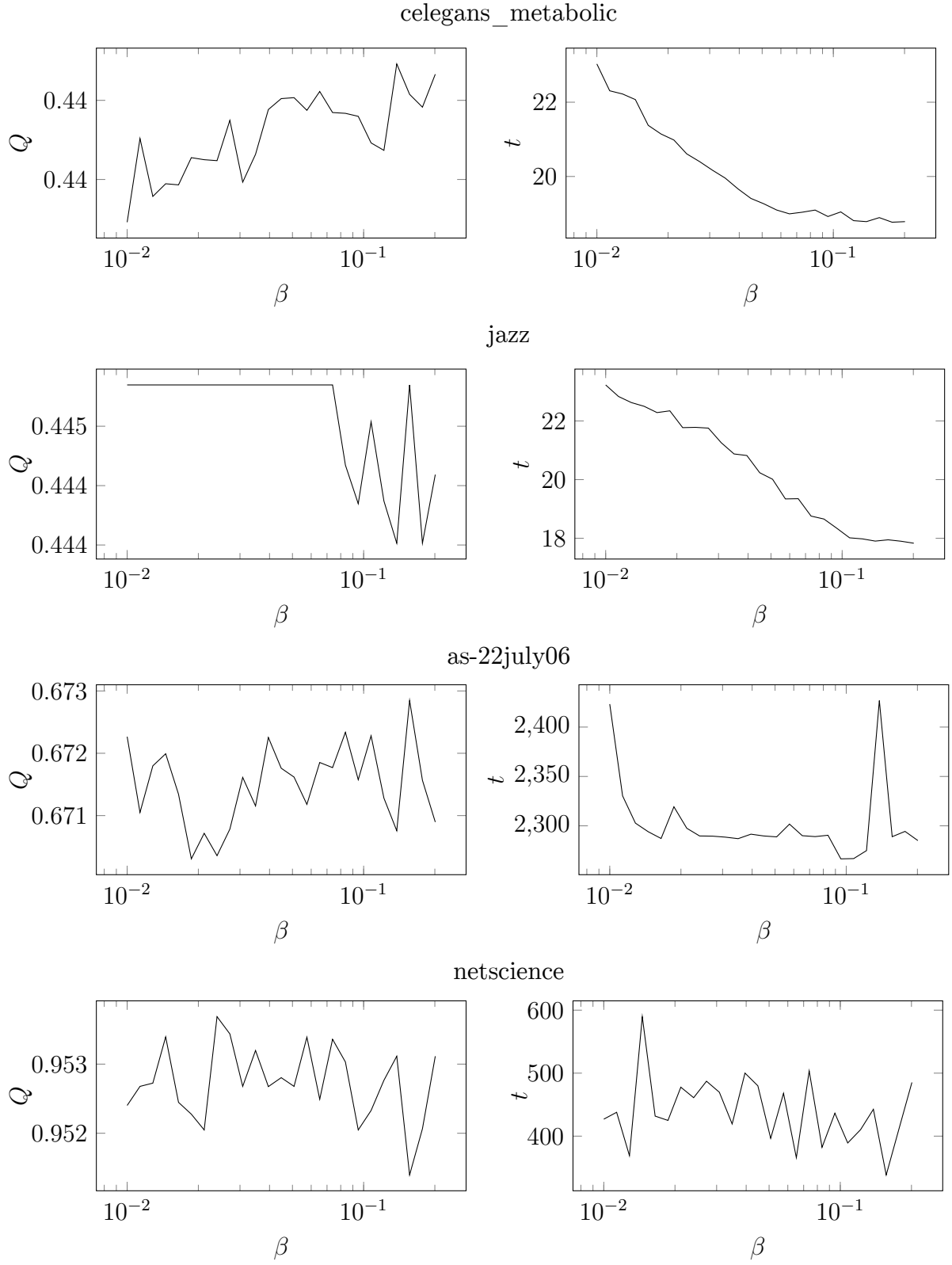

Рис. 27: Зависимость модулярности и времени работы алгоритма от параметра β в работе $ACGGC$ на четырёх графах

Видно, что на графах *celegans_ metabolic*, *jazz* и *as-22july06* увеличение β довольно сильно влияло на время работы, при этом на графе *celegans_ metabolic* модулярность росла, на графе *jazz* оставалась неизменной и лишь после некоторого переломного β начинала падать в среднем, но сравнительно слабо. На графах *as-22july06* и *netscience*

модулярность вела себя непредсказуемо, но принимала небольшой промежуток значений.

Другим подходом к ограничению времени является установка максимального возможного значения k_n^+ и $\hat{k}_n$ (как k_n^- и $\hat{k}_n$ ограничены снизу единицей). Далее в работе ограничение сверху обозначается $k_{max} > 1 \in \mathbb{N}$.

Тогда шаг 3 в описании схемы $ACGGC$ в подразделе 3.2 преобразуется следующим образом: $k_n^- = \max\{1, \hat{k}_{n-1} - d\}$, $k_n^+ = \min\{k_{max}, \hat{k}_{n+1} + d\}$, а формула вычисления следующей центральной точки (6) заменится следующей формулой:

$$\hat{k}_n = \max\left\{1,\ \min\left\{k_{max},\ \left[\hat{k}_{n-1} - \frac{y_n^+ - y_n^-}{k_n^+ - k_n^-}\right]\right\}\right\} \tag{7}$$

Предполагается, что при таком подходе адаптивная стратегия будет давать несильно увеличивающиеся модулярности при уменьшении k_{max} на графах с небольшой оптимальной точкой k, так как в таком случае будут отсекаться неправдоподобные большие k, при этом время будет так же несильно падать. На графах, у которых модулярность несильно растёт при увеличении k время будет сильно снижаться, но при снижении модулярности.

В таблицах 5 и 6 показаны результаты применения адаптивной стратегии с параметрами $d = 2,\ f(Q,k) = -1000 \ln Q,\ \hat{k}_0 = 5,\ l = 6,\ r = 0.02$ и финальным алгоритмом RG_{10} к четырём графам, поведение модулярности которых при разных k можно увидеть на рисунке 28.

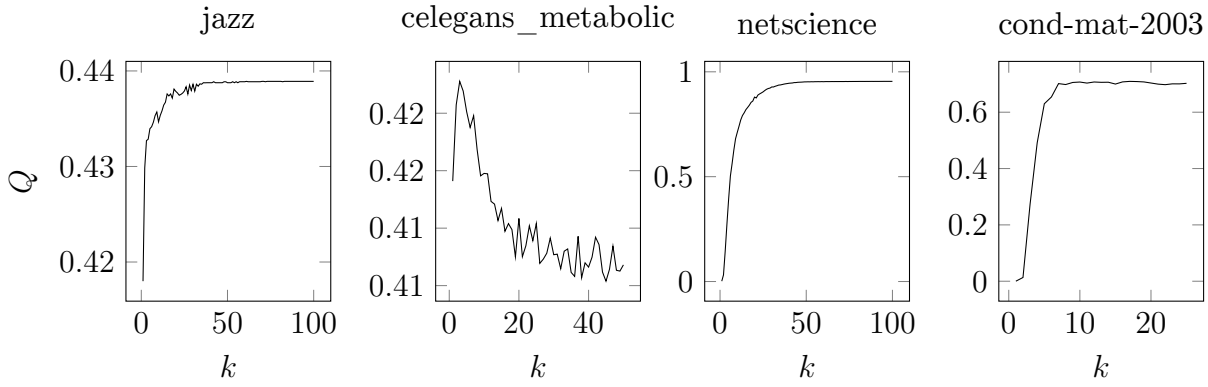

Рис. 28: Зависимость модулярности от параметра k в работе RG_k на четырёх графах

Из таблицы 5 можно увидеть, что время работы действительно заметно падает при уменьшении k_{max}, в то время как модулярность сильно падает только на графе *netscience*. Более того, на некоторых графах (например на графах *football*, *cond-mat-2003*) модулярность при уменьшении k_{max} даже растёт. Такое поведение может объясняться существованием на этих графах небольших параметров рандомизированного жадного алгоритма k, лучших или неотличимых по модулярности от больших k, но отличающихся по времени работы. И при установке на k ограничения сверху такие небольшие оптимальные параметры k будут чаще обнаруживаться.

Таблица 5: Модулярность $ACGGC$ при разных k_{max} на четырёх 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_metabolic	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 четырёх 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_metabolic	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

3.7. Итеративная схема

В подразделе 1.4 описана итеративная схема кластеризации основных групп графа. Адаптивную версию алгоритма также можно итерировать, в таком случае после адаптивного создания промежуточного разбиения на его основе запускается новое адаптивное создание промежуточного множества. Это продолжается до тех пор, пока новое промежуточное разбиение не будет не лучше предыдущего, после чего промежуточное множество передаётся в финальный алгоритм. Такая схема далее называется *итеративная адаптивная схема кластеризации основных групп графа* ($ACGGCi$).

Такой подход не сильно повышает время работы, так как с каждой итерации количество узлов (сообществ) для разбиения уменьшается. Количество узлов или сообществ не может увеличиться или остаться прежним, так как разбиение P_1 на основе разбиения P_2 имеет не больше сообществ, чем разбиение P_2, и в случае, если P_1 и P_2 имеют одинаковое количество сообществ — они равны, то есть и их модулярности равны.

Таблица 7: Модулярность адаптивной схемы кластеризации основных групп графа и её итеративной версии

	ACGGC	ACGGCi
jazz	0.444739	0.444871
celegans_metabolic	0.439724	0.446973
netscience	0.907922	0.909400
as-22july06	0.671205	0.674992
cond-mat-2003	0.743594	0.746731

Таблица 8: Время работы адаптивной схемы кластеризации основных групп графа и её итеративной версии

	ACGGC	ACGGCi
jazz	23.68	31.51
celegans_metabolic	23.92	77.25
netscience	86.38	96.55
as-22july06	2,329	5,801
cond-mat-2003	9,371	11,654

Как видно из таблиц 7 и 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.

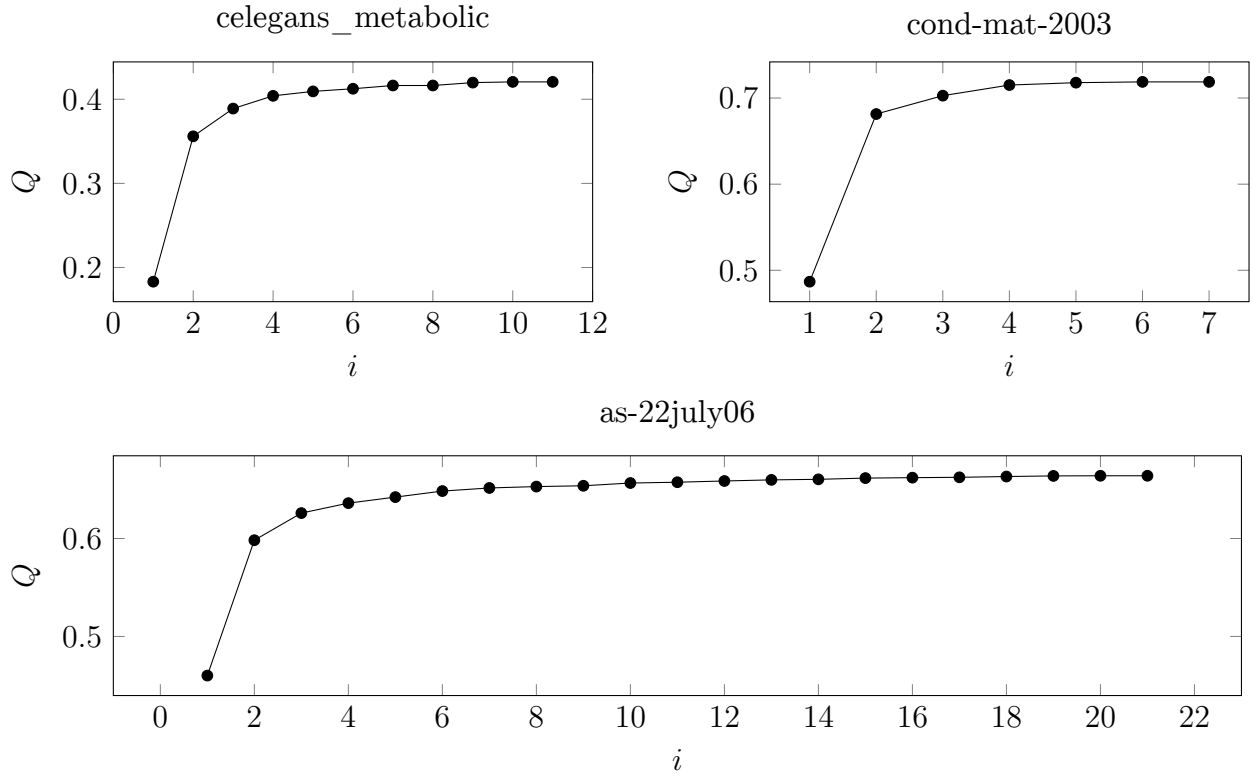

Рис. 29: Модулярности промежуточных разбиений в работе *ACGGCi* на трёх графах, i — номер итерации

Как можно увидеть на рисунке 29, первое промежуточное разбиение имеет очень небольшую модулярность, а начиная с некоторой итерации модулярность следующего промежуточного разбиения увеличивается очень несильно. Стоит так же заметить, что при $l = 1$, к примеру, модулярность первого промежуточного разбиения была бы выше, так как было бы меньше разбиений, которые участвовали в создании максимального перекрытия разбиений. Но это бы не принесло глобального выигрыша, так как на таком разбиении сложно выделять новые множества. Поэтому небольшая мо-

дулярность первого промежуточного разбиения не означает, что оно неудобно для последующей обработки.

Для создания таблиц 7 и 8 и рисунка 29 использовались следующие параметры $ACGGC$ и $ACGGCi$: $d = 2,\ f(Q,k) = -1000 \ln Q,\ \hat{k}_0 = 5,\ l = 6,\ r = 0.05,\ k_{max} = 50$. В качестве финального алгоритма использовался RG_{10}.

3.8. CGGC и ACGGC в качестве финального алгоритма

Одно из преимуществ схемы кластеризации основных групп графа заключается в том, что граф с большим количеством вершин преобразуется в небольшой граф, на котором можно выделять сообщества более точными, но и более долгими алгоритмами, используемыми в качестве финальных алгоритмов. Ранее в работе в качестве финальных алгоритмом рассматривался только RG. Однако существует вариант использования $ACGGC$ с финальным алгоритмом $CGGC$ или $ACGGC$ с другими параметрами, ровно как и наоборот, использовать $CGGC$ с $ACGGC$ в качестве финального алгоритма. Это даёт наибольший выигрыш в итеративной схеме кластеризации основных групп графа и её адаптивного аналога, в таком случае одна из версий алгоритма выделяет сообщества в промежуточных разбиениях, пока это возможно. В некоторый момент создание новых промежуточных множеств оказывается неэффективным, однако новые параметры другой версии схемы могут всё ещё быть эффективны.

Так, на графе *celegans_metabolic* $ACGGCi$ с RG_{10} в качестве финального алгоритма имела модулярность 0.446973 как медиану по 3000 запусков, и работала со средним временем 77.25 миллисекунды. С тем же количеством запусков $CGGCi$ с тем же финальным алгоритмом имела модулярность 0.445008 со временем 55.29 миллисекунд. Однако использование $ACGGCi$ с $CGGCi$ в качестве финального алгоритма дало модулярность 0.447324 со временем 89.96 миллисекунд (в качестве финального алгоритма $CGGCi$ при этом использовалось RG_{10}). Таким образом, прирост времени по отношению к использованию в качестве финального алгоритма RG_{10} оказался не очень большим, однако такой подход дал ненулевое увеличение модулярности. Использование же $CGGCi$ стратегии с $ACGGCi$ в качестве финального алгоритма не дало результата на этом графе: модулярность 0.44566 при времени 112.49 миллисекунд.

На рисунке 30 изображены модулярности последовательных промежуточных разбиений в $ACGGCi$ с $CGGCi$ в качестве финального алгоритма. К десятой итерации $ACGGCi$ достигло придельного значения, однако после использования на получившемся промежуточном множестве одной итерации $CGGCi$ произошёл небольшой глобально, но большой относительно предыдущих изменений скачок модулярности.

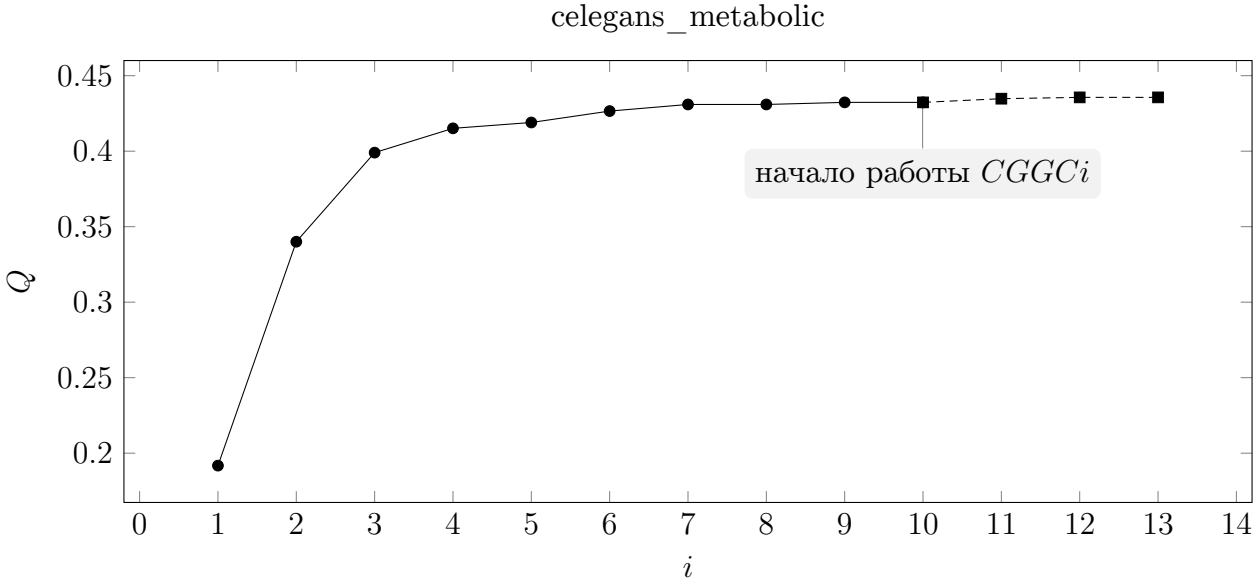

Рис. 30: Модулярности промежуточных разбиений в работе $ACGGCi$ с $CGGCi$ в качестве финального алгоритма на графе *celegans_ metabolic*, i — номер итерации, синяя линия — промежутночные разбиения адаптивной версии, красная обозначает промежуточные разбиения неадаптивной версии

В измерениях и на рисунке 30 использовались $ACGGCi$ с параметрами $d = 2,\ f(Q,k) = -1000 \ln Q,\ \hat{k}_0 = 5,\ l = 6,\ r = 0.05,\ k_{max} = 50$ и $CGGCi$ с начальным алгоритмом RG_{10} и параметром $s = 16$.

3.9. Сравнение

На таблицах 9 и 10 обозначены результаты работы пяти алгоритмов:

- $ACGGC$ с параметрами $d = 2,\ f(Q,k) = -1000 \ln Q,\ \hat{k}_0 = 5,\ l = 6,\ r = 0.05,\ k_{max} = 50$, финальным алгоритмом RG_{10}, обозначается $ACGGC^{I}$
- $ACGGC$ с параметрами $d = 2,\ f(Q,k) = -1000 \ln Q,\ \hat{k}_0 = 5,\ l = 8,\ r = 0.05,\ k_{max} = 20$, 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 9 лучшая модулярность на графе обозначалась сплошным подчёркиванием, а вторая по величине — пунктирным подчёркиванием (если несколько алгоритмов принимали лучшее значение — все отмечались пукнтирной линией и второе по величине значение никак не отмечалось).

Таблица 9: 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_metabolic	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		

В качестве параметра s алгоритма $CGGC$ было выбрано значение $s = 16$, так как в [16] 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.

В 11 случаях из 18 $ACGGC$ с разными параметрами приняло лучшее значение модулярности, чем $CGGC$ с разными параметрами, в то же время $CGGC$ приняло лучшее значения модулярности, чем $ACGGC$ только 4 раза. Алгоритм $ACGGC^{II}$ принимал лучшее или второе по величине значение 17 раз из 18.

Измерения на каждом графе, кроме больших графов *caidaRouterLevel*, *cnr-2000*, *eu-2005*, *in-2004*, запускались более 100 раз для повышения точности.

Таблица 10: Время работы $ACGGC$ и $CGGC$ на тестовых графах

	$ACGGC^{I}$	$ACGGC^{II}$	$CGGC_{10}^{10}$	$CGGC_{3}^{10}$	$CGGC_{10}^{3}$
karate	1.045	1.358	1.381	1.159	1.383
dolphins	2.453	2.982	3.012	2.471	2.998
chesapeake	1.792	2.344	2.244	1.914	2.214
adjnoun	6.193	7.627	7.411	6.342	7.571
polbooks	5.064	6.310	6.136	5.166	6.135
football	7.157	8.441	8.638	7.439	8.547
celegans_metabolic	23.438	29.847	31.842	25.991	31.561
jazz	23.306	27.546	27.580	24.268	27.508
netscience	84.985	61.307	60.831	40.893	59.673
email	71.719	91.856	94.421	80.227	94.101
polblogs	173.03	174.07	177.14	123.35	178.01
pgpGiantCompo	635.50	864.96	1,033.56	832.90	1,030.35
as-22july06	2,330.91	3,152.38	3,714.06	3,071.44	3,716.53
cond-mat-2003	9,165.87	8,172.36	7,485.29	3,443.61	7,422.58
caidaRouterLevel	153,378	154,456	162,787	146,626	162,802
cnr-2000	306,539	413,869	419,965	404,296	420,604
eu-2005	$2.35 \cdot 10^6$	$3.16 \cdot 10^6$	$3.26 \cdot 10^6$	$3.16 \cdot 10^6$	$3.23 \cdot 10^6$
in-2004	$6.59 \cdot 10^6$	$8.32 \cdot 10^6$	$8.91 \cdot 10^6$		

В подразделе 3.7 была описана итеративная адаптивная схема кластеризации основных групп графа, предполагается, что она даёт лучшие резульаты, чем неитеративная версия за счёт того, что выделяя сообщества в промежуточном разбиении для получения нового промежуточного разбиения, алгоритм повышает качество промежуточного множества. На каждой итерации остаётся всё меньше узлов, относительно которых начальные алгоритмы разошлись во мнении. Необходимо проверить, не исчезает ли в ходе работы итерационной стратегии разница между модулярностями схем $CGGC$ и $ACGGC$.

Итеративные схемы $CGGCi$ и $ACGGCi$ также следует сравнить с использованием $CGGCi$ в качестве финального алгоритма для $ACGGCi$, более подробно описанным в подразделе 3.8.

В таблице 11 представлены модулярности разбиений, полученных итеративной версией $ACGGC^{I}$, $ACGGC^{II}$ и $CGGC_{10}^{10}$, обозначаемые $ACGGCi^{I}$, $ACGGCi^{II}$ и $CGGCi$, соответственно. Также в столбце *combined* представлены модулярности $ACGGCi^{I}$ с $CGGCi$ в качестве финального алгоритма. Часто все алгоритмы принимали одно и то же, предположительно максимально возможное, значение. В остальных случаях сплошной линией отмечены лучшая среди четырёх алгоритмов модулярность, достигнутая на графе.

Таблица 11: Модулярность работы четырёх 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_metabolic	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

Заметно, что в итеративные версии схемы примерно равнозначны по результатам, хотя схемы $ACGGC^I$ и *combined* дают лучшие результаты чуть чаще, а также их результаты всегда достаточно близко к лучшему результату, в отличии от $CGGCi$.

Заключение

Были проанализированы современные методы выделения сообществ в графах и создано две модификации рандомизированных алгоритмов выделения сообществ в графах с помощью одновременно возмущаемой стохастической аппроксимации: адаптивный рандомизированный жадный алгоритм, описанный в разделе 2, а также адаптивная схема кластеризации основных групп графа, описанная в разделе 3. Для каждого алгоритма приведён анализ его параметров и результатов.

При этом адаптивный рандомизированный жадный алгоритм показал более стабильные результаты, чем его неадаптивный вариант (подраздел 2.9), что может в том числе иметь применение, описанное в подразделе 3.1.

Адаптивная схема кластеризации основных групп графа показала в среднем лучшие и более стабильные результаты, чем неадаптивная схема (подраздел 3.9). Также адаптивная схема имеет эффективный механизм снижения времени работы, описанный в подразделе 3.6.

В дальнейшних исследованиях области имеет смысл рассмотреть поведение параметров алгоритмов на больших графах, а так же использовать изменяющиеся со временем параметры, к примеру в итеративной адаптивной схеме кластеризации основных групп графа.

Список литературы

[1] Stefano Boccaletti, Vito Latora, Yamir Moreno, Martin Chavez, and D-U Hwang. Complex networks: Structure and dynamics. *Physics reports*, 424(4):175–308, 2006.

[2] Cristopher Moore and Mark EJ Newman. Epidemics and percolation in small-world networks. *Physical Review E*, 61(5):5678, 2000.

[3] Jing Zhao, Hong Yu, Jianhua Luo, ZW Cao, and Yixue Li. Complex networks theory for analyzing metabolic networks. *Chinese Science Bulletin*, 51(13):1529–1537, 2006.

[4] Wang Hong, Wang Zhao-wen, Li Jian-bo, and Qiu-hong Wei. Criminal behavior analysis based on complex networks theory. In *IT in Medicine & Education, 2009. ITIME'09. IEEE International Symposium on*, volume 1, pages 951–955. IEEE, 2009.

[5] John Scott. *Social network analysis*. Sage, 2012.

[6] Michalis Faloutsos, Petros Faloutsos, and Christos Faloutsos. On power-law relationships of the internet topology. In *ACM SIGCOMM Computer Communication Review*, volume 29, pages 251–262. ACM, 1999.

[7] Andrei Broder, Ravi Kumar, Farzin Maghoul, Prabhakar Raghavan, Sridhar Rajagopalan, Raymie Stata, Andrew Tomkins, and Janet Wiener. Graph structure in the web. *Computer networks*, 33(1):309–320, 2000.

[8] Stanley Wasserman. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

[9] Mark E. J. Newman and Michelle Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, Feb 2004.

[10] Stefanie Muff, Francesco Rao, and Amedeo Caflisch. Local modularity measure for network clusterizations. *arXiv preprint cond-mat/0503252*, 2005.

[11] Santo Fortunato and Marc Barthélemy. Resolution limit in community detection. *Proceedings of the National Academy of Sciences*, 104(1):36–41, 2007.

[12] Ulrik Brandes, Daniel Delling, Marco Gaertler, Robert Gorke, Martin Hoefer, Zoran Nikoloski, and Dorothea Wagner. On modularity clustering. *Knowledge and Data Engineering, IEEE Transactions on*, 20(2):172–188, 2008.

[13] Mark E. J. Newman. Fast algorithm for detecting community structure in networks. *Physical Review E*, 69:066133, Jun 2004.

[14] Michael Ovelgönne and Andreas Geyer-Schulz. A comparison of agglomerative hierarchical algorithms for modularity clustering. In *Challenges at the Interface of Data Analysis, Computer Science, and Optimization*, pages 225–232. Springer, 2012.

[15] Michael Ovelgönne and Andreas Geyer-Schulz. Cluster cores and modularity maximization. In *Data Mining Workshops (ICDMW), 2010 IEEE International Conference on*, pages 1204–1213. IEEE, 2010.

[16] Michael Ovelgönne and Andreas Geyer-Schulz. An ensemble learning strategy for graph clustering. *Graph Partitioning and Graph Clustering*, 588:187, 2012.

[17] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[18] Jack Kiefer and Jacob Wolfowitz. Stochastic estimation of the maximum of a regression function. *The Annals of Mathematical Statistics*, 23(3):462–466, 1952.

[19] Julius R Blum. Multidimensional stochastic approximation methods. *The Annals of Mathematical Statistics*, pages 737–744, 1954.

[20] James C Spall. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. *IEEE Transactions on Automatic Control*, 37(3):332–341, 1992.

[21] James C. Spall. *Introduction to stochastic search and optimization: estimation, simulation, and control*, volume 65. John Wiley & Sons, 2005.

[22] Boris T. Polyak. *Introduction to optimization*. Optimization Software New York, 1987.

[23] Oleg Granichin and Natalia Amelina. Simultaneous perturbation stochastic approximation for tracking under unknown but bounded disturbances. *IEEE Transactions on Automatic Control*, 60(5), 2015.

[24] Wayne W Zachary. An information flow model for conflict and fission in small groups. *Journal of anthropological research*, pages 452–473, 1977.

[25] Daniel Baird and Robert E Ulanowicz. The seasonal dynamics of the Chesapeake Bay ecosystem. *Ecological Monographs*, pages 329–364, 1989.

[26] David Lusseau, Karsten Schneider, Oliver J Boisseau, Patti Haase, Elisabeth Slooten, and Steve M Dawson. The bottlenose dolphin community of doubtful sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54(4):396–405, 2003.

[27] Mark EJ Newman. Finding community structure in networks using the eigenvectors of matrices. *Physical review E*, 74(3):036104, 2006.

[28] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, 99(12):7821–7826, 2002.

[29] Pablo M. Gleiser and Leon Danon. Community structure in jazz. *Advances in complex systems*, 6(04):565–573, 2003.

[30] Jordi Duch and Alex Arenas. Community detection in complex networks using extremal optimization. *Physical review E*, 72(2):027104, 2005.

[31] Roger Guimerà Manrique, L Danon, Albert Díaz Guilera, Francesc Giralt, and Àlex Arenas. Self-similar community structure in a network of human interactions. *Physical Review E, 2003, vol. 68, núm. 6, p. 065103-1-065103-4*, 2003.

[32] Lada A Adamic and Natalie Glance. The political blogosphere and the 2004 US election: divided they blog. In *Proceedings of the 3rd international workshop on Link discovery*, pages 36–43. ACM, 2005.

[33] Marián Boguñá, Romualdo Pastor-Satorras, Albert Díaz-Guilera, and Alex Arenas. Models of social networks based on social distance attachment. *Phys. Rev. E*, 70:056122, Nov 2004.

[34] Mark EJ Newman. The structure of scientific collaboration networks. *Proceedings of the National Academy of Sciences*, 98(2):404–409, 2001.

[35] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[36] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th international conference on World Wide Web*. ACM Press, 2011.

[37] Paolo Boldi, Bruno Codenotti, Massimo Santini, and Sebastiano Vigna. Ubicrawler: A scalable fully distributed web crawler. *Software: Practice & Experience*, 34(8):711–726, 2004.