


Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Нижегородский государственный университет им. Н.И. Лобачевского
Институт информационных технологий, математики и механики

ПАРАЛЛЕЛЬНАЯ ОБРАБОТКА ГРАФОВ

Лекция 8. Некоторые алгоритмы для анализа сетей

Пирова А.Ю.
Кафедра ВВиСП

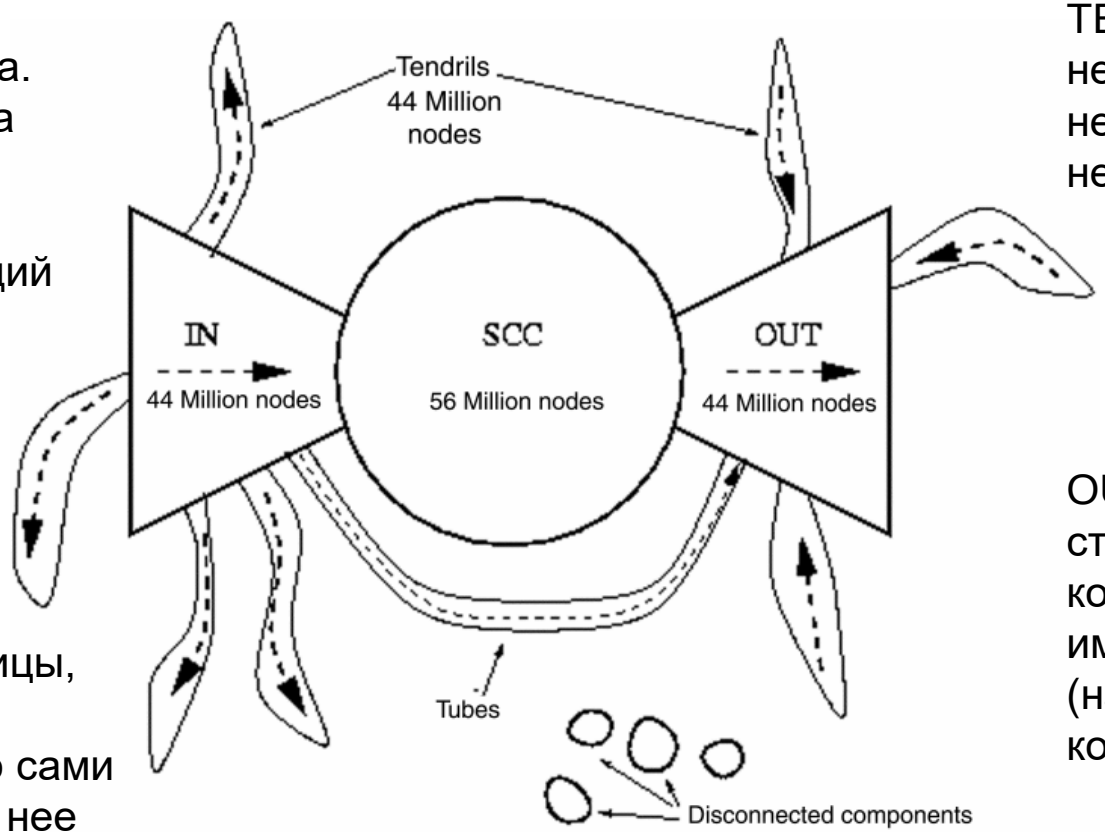
Содержание

- Структура Web
- Виды центральности
- Степень посредничества (betweenness centrality)
- PageRank

Структура Web, Бродер и др. 2000

- Web имеет структуру «галстук-бабочка» с 4 равными частями

SCC – ядро Интернета. Любые страницы ядра достижимы друг для друга по ссылкам. Диаметр SCC 28, общий диаметр – более 500



TENDRILS – страницы, недостижимые из SCC и не имеющие ссылок на нее

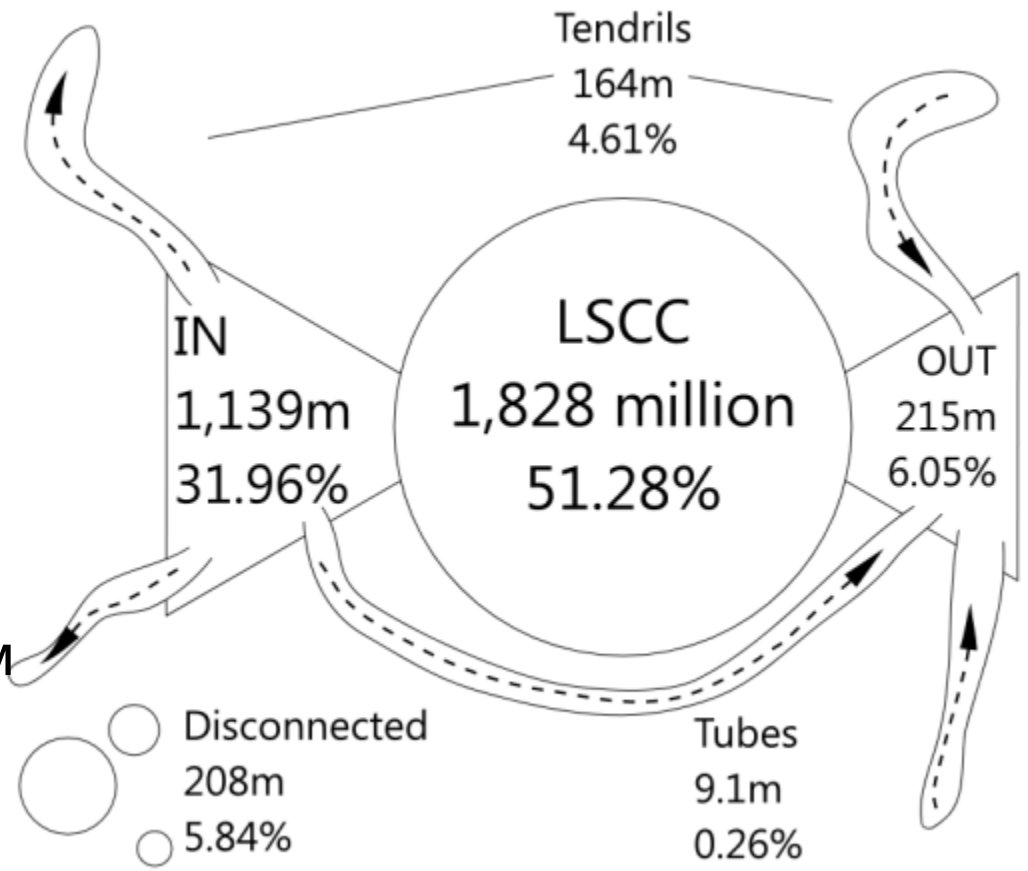
OUT – интернет-страницы, достижимые из компоненты SCC, но не имеющие ссылки вовне (например, копроративный сайт)

IN – интернет-страницы, недостижимые из компоненты SCC, но сами имеющие ссылки на нее (например, новый сайт)

Broder A. et al. Graph structure in the web //Computer networks. – 2000. – Т. 33. – №. 1-6. – С. 309-320.

Структура Web, Мойзел и др. (2014)

- Методология:
  - Использовалось веб-сканирование (web crawl) от CommonCrawlFoundation
  - В основе – поиск в ширину из нескольких начальных страниц с эвристиками для детектирования спама и пустых страниц
  - Извлечен веб-граф, в котором вершина – веб-страница, ребро – наличие ссылки между двумя страницами



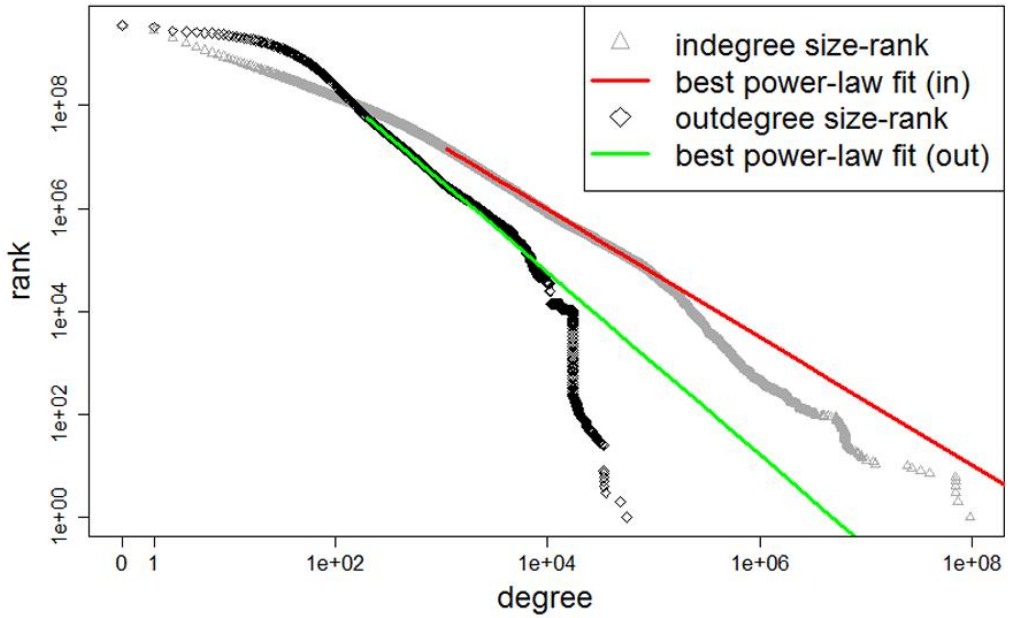
Meusel R. et al. Graph structure in the web – revisited: a trick of the heavy tail // Proceedings of the 23rd international conference on World Wide Web. – 2014. – С. 427-432.

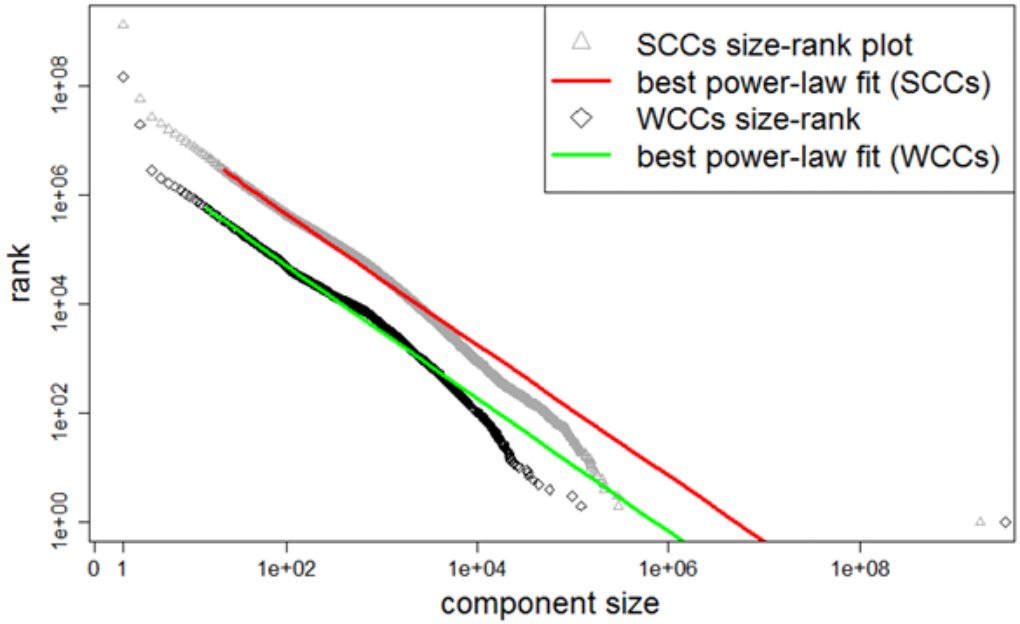
Структура Web, Мойзел и др. (2014)

- Прогнозируемый рост числа страниц, который должен был быть логарифмическим.
- Увеличилась средняя степень (в 5 раз)
- Увеличилась связность графа (процент связанных пар) в 2 раза,
- Уменьшилось среднее расстояние между страницами,

	Common Crawl 2012		Broder *et al.*	
Component	# nodes (in thousands)	% nodes (in %)	# nodes (in thousands)	% nodes (in %)
LSCC	1 827 543	51.28	56 464	27.74
IN	1 138 869	31.96	43 343	21.29
OUT	215 409	6.05	43 166	21.21
TENDRILS	164 465	4.61	43 798	21.52
TUBES	9 099	0.26	-	-
DISC.	208 217	5.84	16 778	8.24

Структура Web, Мойзел и др. (2014)





- Распределение по степеням и распределение по компонентам не соответствует степенному закону. Визуальный осмотр графиков показывает, что их хвосты не толстые (т. е. они убывают быстрее, чем полиномиально)

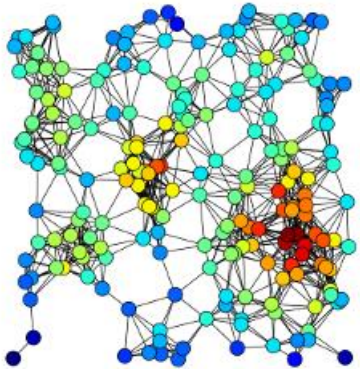
ВИДЫ ЦЕНТРАЛЬНОСТИ

Виды центральности

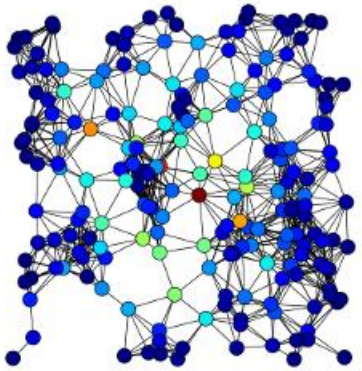
- Как и зачем определять важность вершин в графе?
  - Какие персоны в социальной сети наиболее влиятельны?
  - Какие узлы транспортной сети будут иметь наибольшую нагрузку?
  - Какие роутеры / вышки мобильной связи будут иметь наибольшую нагрузку?
  - Кого надо вакцинировать первым, чтобы предотвратить эпидемию?
- Центральность – мера важности вершины на основе ее связей с другими вершинами.

Виды центральности

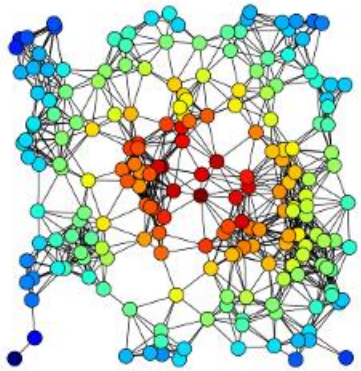
– Степень связности (degree centrality)



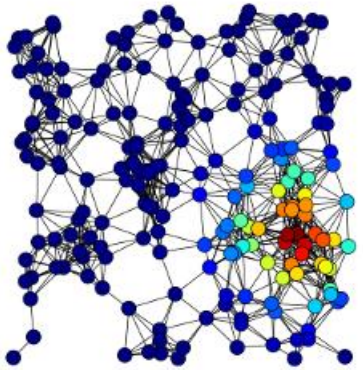
– Степень посредничества (beetweenness centrality)



– Степень близости (closeness centrality)



– Степень влиятельности (eigenvector centrality)



Степень связности

- *Степень связности* (degree centrality) – число связей узла.
  - $C(i) = \deg(i)$
  - Пример: с какой вероятностью через узел пройдет информация / автомобиль проедет по данной дороге?

Степень близости

- *Степень близости* (closeness centrality) – мера близости данного узла к другим узлам. Равна средней длине кратчайшего пути между данной вершиной и всеми остальными вершинами.

$$l_i = \frac{1}{n}\sum_{i \neq j} d_{ij},$$

d_{ij} - кратчайшее расстояние между вершинами i и j.

  - На практике используется обратная величина, $C(i) = \frac{n}{\sum_{i \neq j} d_{ij}}$
  - Недостаток характеристики: значения скученные, мало отличаются у центральных и нецентральных вершин.
  - Для несвязных графов задается с использованием гармонического расстояния
  - Пример: из какого города ближе всего доехать до других городов?

Степень влиятельности

- *Степень влиятельности* (eigenvector centrality) – мера связности центральной вершины с другими центральными вершинами.
  - Пример: сколько дорог между значимыми городами? Сколько ссылок на популярные страницы в сети имеет другая популярная страница?

    $x_i = \frac{1}{\lambda}\sum_J A_{ij}x_j$, решение системы $x = \lambda Ax$, λ – константа.
  - Модификации: PageRank, центральность Каца

Степень посредничества

- *Степень посредничества (betweenness centrality)* – мера центральности графа, равная числу кратчайших путей, проходящих через данную вершину.
  - Пример: сколько городов соединяет дорога через данный город? Какая доля сообщений проходит через данный роутер?
  - Вершины с большой степенью посредничества являются «мостами» между другими вершинами сети. Их удаление нарушает связность сети.
- Фриман (1977): $BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$,

  $\sigma_{st}(v)$ – доля кратчайших путей между s и t, которые включают v.

BETWEENNESS CENTRALITY

Применение

- Широко используется для анализа сетей в разных предметных областях
- Примеры:
  - Поиск сообществ в социальных сетях
  - Определение генных заболеваний в графе взаимодействия генов
  - Пересылка сообщений в мобильных сетях
  - Анализ чрезвычайных ситуаций в электрических сетях
  - Анализ транспортных сетей (выявление ключевых элементов сети)

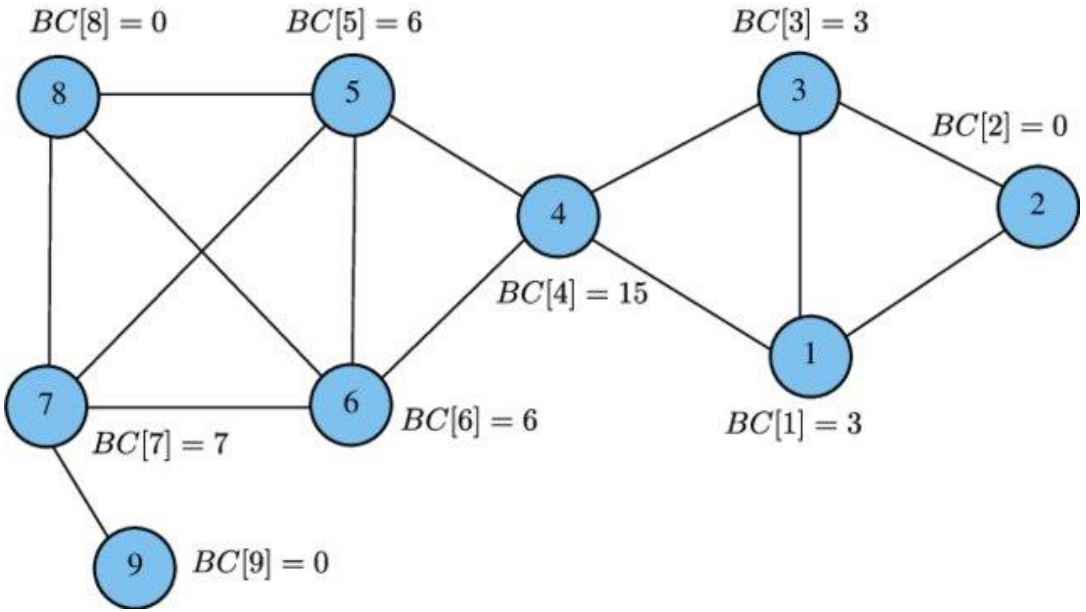
Варианты постановки задачи

- Фриман (1977): $BC(v) = \sum_{s \neq t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}}$,

  $\sigma_{st}(v)$ – доля кратчайших путей между s и t, которые включают v.

- ε-аппроксимация BC: набор $\tilde{B}: \{\widetilde{BC}(v), v \in V\}$
  $\left|\widetilde{BC}(v) - BC(v)\right| < \varepsilon$ для всех $v \in V$
  - Для сокращения накладных расходов используют batched BC: кратчайшие пути считают из заданного числа начальных вершин, а не из всех.
  - Для динамически изменяющегося графа поддержание приближенного значения BC значительно сокращают временную сложность алгоритма в сравнении с пересчетом BC с нуля.

Пример



McLaughlin A., Bader D. A. Scalable and high performance betweenness centrality on the GPU //SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. – IEEE, 2014. – С. 572-583.

Алгоритм Брандеса

- Brandes U. A faster algorithm for betweenness centrality //J. of math. sociology. – 2001. – Т. 25. – № 2. – С. 163-177. [pdf]
- Рекуррентное соотношение:
  - $P(s,w)$ – список предшественников w в кратчайшем пути от s к w.
  - *зависимость* источника s от вершины v:

$$\delta_{s,-}(v) = \sum_{t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

$$\Longrightarrow \quad \delta_{s,-}(v) = \sum_{w:\, v \in P(s,w)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_{s,-}(w))$$

$$BC(v) = \sum_{s \neq v \in V} \delta_{s,-}(v)$$

Алгоритм Брандеса

- Вход: граф $G(V, E)$, Выход: массив значений $BC(v)$

1. Для каждой вершины $s \in V$
   1. *Построение дерева*

      Найти дерево кратчайших путей D из s до каждой вершины $v \in V$
      1. Вычислить σ_{sv}
      2. Найти предшественника $P(s, v)$
   2. *Вычисления*

      Обход дерева D снизу вверх: Для каждой вершины $v \in D$
      1. Вычислить $\delta_s(v)$
      2. Обновить $BC(v) \mathrel{+}= \delta_s(v)$

- Вычислительная сложность алгоритма для взвешенного графа $O(nm + n^2 \log n)$, для невзвешенного - $O(nm)$. Требуемый объем памяти $O(n + m)$.

Алгоритм Брандеса

Algorithm 1: Betweenness centrality in unweighted graphs

```
C_B[v] ← 0, v ∈ V;
for s ∈ V do
    S ← empty stack;
    P[w] ← empty list, w ∈ V;
    σ[t] ← 0, t ∈ V;   σ[s] ← 1;
    d[t] ← −1, t ∈ V;   d[s] ← 0;
    Q ← empty queue;
    enqueue s → Q;
    while Q not empty do
        dequeue v ← Q;
        push v → S;
        foreach neighbor w of v do
            // w found for the first time?
            if d[w] < 0 then
                enqueue w → Q;
                d[w] ← d[v] + 1;
            end
```

Алгоритм Брандеса

```
            // shortest path to w via v?
            if d[w] = d[v] + 1 then
                σ[w] ← σ[w] + σ[v];
                append v → P[w];
            end
        end
    end
    δ[v] ← 0, v ∈ V;
    // S returns vertices in order of non-increasing distance from s
    while S not empty do
        pop w ← S;
        for v ∈ P[w] do δ[v] ← δ[v] + (σ[v]/σ[w]) · (1 + δ[w]);
        if w ≠ s then C_B[w] ← C_B[w] + δ[w];
    end
end
```

Параллельный алгоритм для общей памяти

Вход: граф $G(V, E)$
Выход: BC – массив значений $BC[v]$
Переменные: P – список предшественников в дереве обхода графа,
S – стек дерева обхода,

1. **parallel for** Для всех $v \in V$ $BC[v] = 0$
2. **for** Для всех $s \in V$
3. *I. Инициализация*
4. **for** Для всех $t \in V$ $P[t] = \emptyset, \sigma[t] = 0, d[t] = -1$
5. $\sigma[s] = 1, d[s] = 0;$
6. $phase = 0, S[phase] = \emptyset$
7. $S[phase].push(s)$
8. $count = 1$

Параллельный алгоритм для общей памяти

II. Обход графа

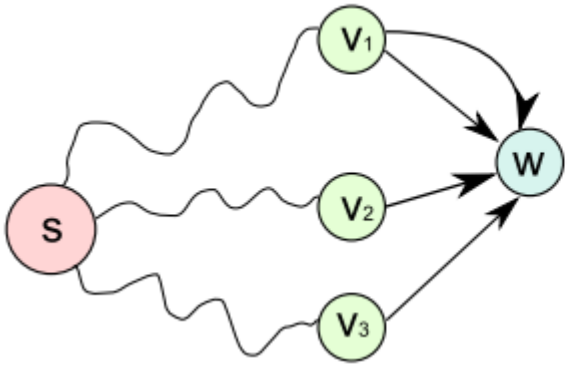
```
9.    while (count > 0) do
10.     count = 0;
11.     parallel for для всех v ∈ S[phase] do
12.       parallel for для каждого w ∈ Adj(v) do
13.         dw = atomic_compare_and_swap(&dw, -1, phase + 1)
14.         if (dw == −1) then
15.           p = atomic_fetch_add(&count, 1)
16.           S[phase + 1].push(w) на позицию p
17.           dw = phase + 1
18.         if (dw == phase + 1) then
19.           p = atomic_fetch_add(&P_count[w], 1)
20.           P[w].push(v) на позицию p
21.           atomic_fetch_add(&σ[w], σ[v])
22.       phase = phase + 1
23.   phase = phase – 1
```

Параллельный алгоритм для общей памяти

24. *III. Подсчет зависимостей*
25. **for** Для всех $t \in V$ $\delta[t] = 0$
26. ***while*** (phase > 0) **do**
27. **parallel for** для всех $w \in S[phase]$ **do**
28. **for** для всех $v \in P[w]$ **do**
29. **lock**(v)
30. $\delta[v] \mathrel{+}= \frac{\sigma[v]}{\sigma[w]} * (1 + \delta[w])$
31. **unlock**(v)
32. $BC[w] = BC[w] + \delta_w$
33. $phase = phase - 1$

Параллельный алгоритм для общей памяти

- Проблема: если вершина w имеет много предшественников, то объединение $\delta_s(v_i)$ станет «узким местом»
  - Требует большое число синхронизаций
  - случайный доступ к памяти при обходе предшественников



Predecessor updates in Algorithm 1	*New approach*
append $v_1 \to P[w]$	append $w \to Succ[v_1]$
append $v_1 \to P[w]$	append $w \to Succ[v_1]$
append $v_2 \to P[w]$	append $w \to Succ[v_2]$
append $v_3 \to P[w]$	append $w \to Succ[v_2]$

- Решение: построим массив последователей Succ
- Pull-подход: обход графа кратчайших путей также можно выполнить «снизу вверх».

Параллельный алгоритм для общей памяти - 2

Вход: граф $G(V, E)$
Выход: BC – массив значений $BC[v]$
Переменные: P – список предшественников в дереве обхода графа,
S – стек дерева обхода, $Succ$ – множество последующих вершин в дереве обхода

1. **parallel for** Для всех $v \in V$ $BC[v] = 0$
2. **for** Для всех $s \in V$
3. *I. Инициализация*
4. **for** Для всех $t \in V$ $P[t] = \emptyset, \sigma[t] = 0, d[t] = -1$
5. $\sigma[s] = 1, d[s] = 0;$
6. $phase = 0, S[phase] = \emptyset$
7. $S[phase].push(s)$
8. $count = 1$

Параллельный алгоритм для общей памяти - 2

II. Обход графа

```
9.    while (count > 0) do
10.     count = 0;
11.     parallel for для всех v ∈ S[phase] do
12.       parallel for для каждого w ∈ Adj(v) do
13.         atomic_compare_and_swap(dw, -1, phase + 1)
14.         if (dw == −1) then
15.           p = atomic_fetch_add(count, 1)
16.           S[phase + 1].push(w) на позицию p
17.           dw = phase + 1
18.         if (dw == phase + 1) then
19.           p = atomic_fetch_add(Succ_count[v], 1)
20.           Succ[v].push(w) на позицию p
21.           atomic_fetch_add(σ[w], σ[v])
22.       phase = phase + 1
23.   phase = phase – 1
```

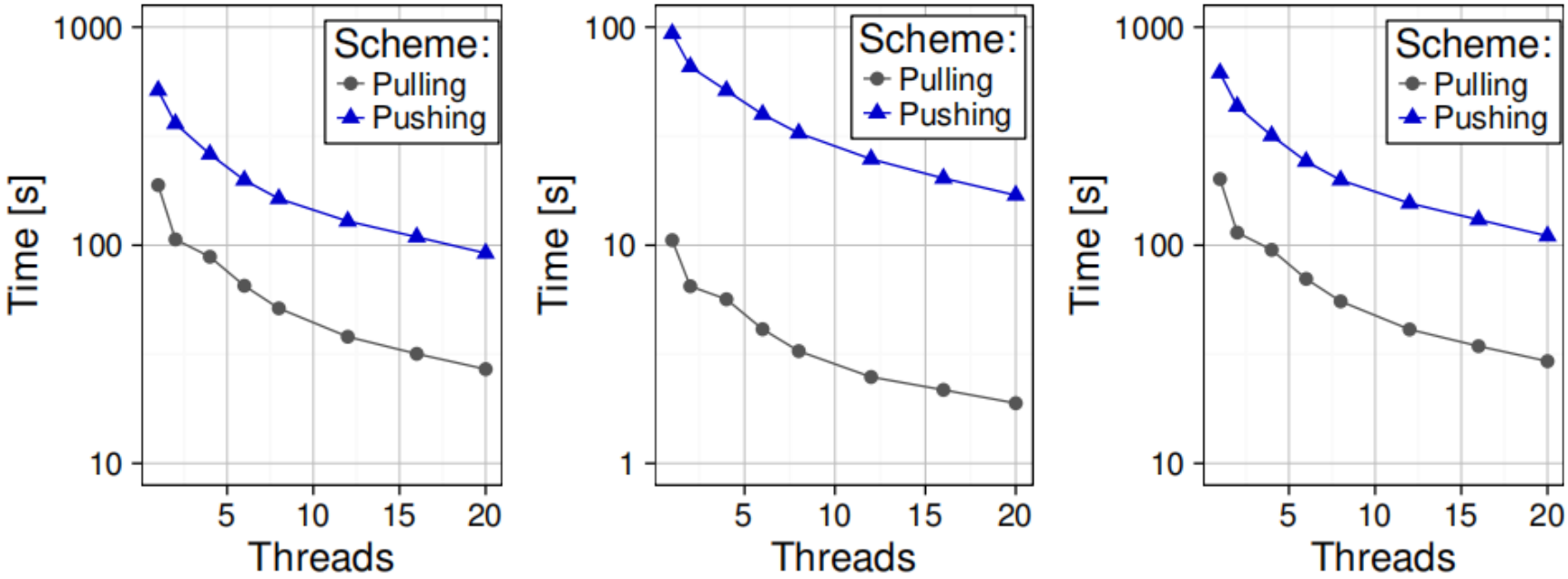
Параллельный алгоритм для общей памяти - 2

24. *III. Подсчет зависимостей*
25. **for** Для всех $t \in V$ $\delta[t] = 0$
26. $phase = phase - 1$
27. **parallel for** для всех $w \in S[phase]$ **do**
28. $\delta_{sum} = 0;$
29. **for** для всех $v \in Succ[w]$ **do**
30. $\delta_{sum} \mathrel{+}= \frac{\sigma[w]}{\sigma[v]} * (1 + \delta[v])$
31. $\delta[w] = \delta_{sum}$
32. $BC[w] \mathrel{+}= \delta_{sum}$

Вычислительные эксперименты

- Источник:
  Besta M. et al. To push or to pull: On reducing communication and synchronization in graph computations //Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing. – 2017. – С. 93-104.
- Тестовое окружение:
  Cray XC nodes from the CSCS supercomputing systems. We use XC50 and XC40 nodes from the Piz Daint machine. An XC50 node contains a 12-core Intel Xeon E5-2690 CPU with 64 GiB RAM. Each XC40 node contains an 18-core Intel Xeon E5-2695 CPU with 64 GiB RAM.
- **Тестовые матрицы:**
  orc, 3.07M вершин, 117M ребер, степени вершин от 9 до 39.

Вычислительные эксперименты



(a) First BFS runtime (b) Second BFS runtime (c) Total runtime.

- Основное время выполнения занимает первый проход поиском в ширину.
- Push подход медленнее, чем pull, так как содержит больше конфликтов записи и операций синхронизации между потоками.
- Ускорение около 5 раз для pull подхода, около 9 раз – для push подхода.

Студенческая работа (Орлов Максим, 2022 г.)

- Реализован гибридный алгоритм Брандеса

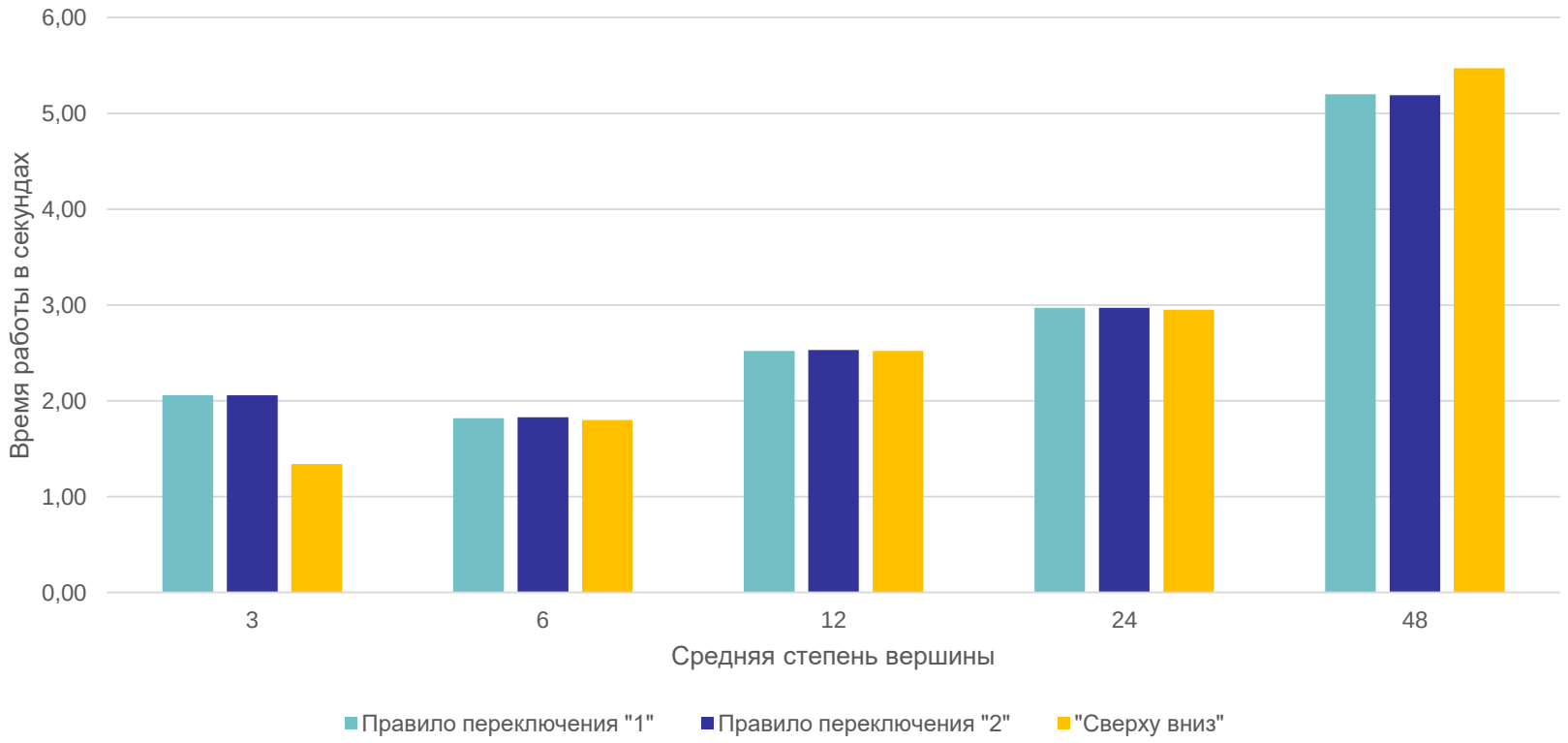
- Тестовые графы: сгенерированы случайным образом, параметры - число вершин, средняя степень вершины.
- Тестовая система:
  - узел суперкомпьютера «Лобачевский» (два процессора Intel Sandy Bridge E5-2660 с частотой 2.20 ГГц, 8 ядер, оперативная память 64.00 Гб)

Студенческая работа (Орлов Максим, 2022 г.)

- При реализации гибридного обхода в ширину использовались следующие **правила переключения**:
  - Правило «1»:
    - если TD и $E_f > \alpha E_u$, E_f растет, переключение на BU;
    - если BU и $V_f > \beta n$, V_f падает, переключение на TD,
    - где E_f – суммарная степень вершин фронта, E_u – суммарная степень непосещенных вершин, V_f – число вершин во фронте.
  - Правило «2»:
    - если TD и $E_f > \alpha E$, переключение на BU;
    - если BN и $E_f < \alpha E$, переключение на TD,
    - где E_f – суммарная степень вершин фронта, E – суммарная степень вершин графа.

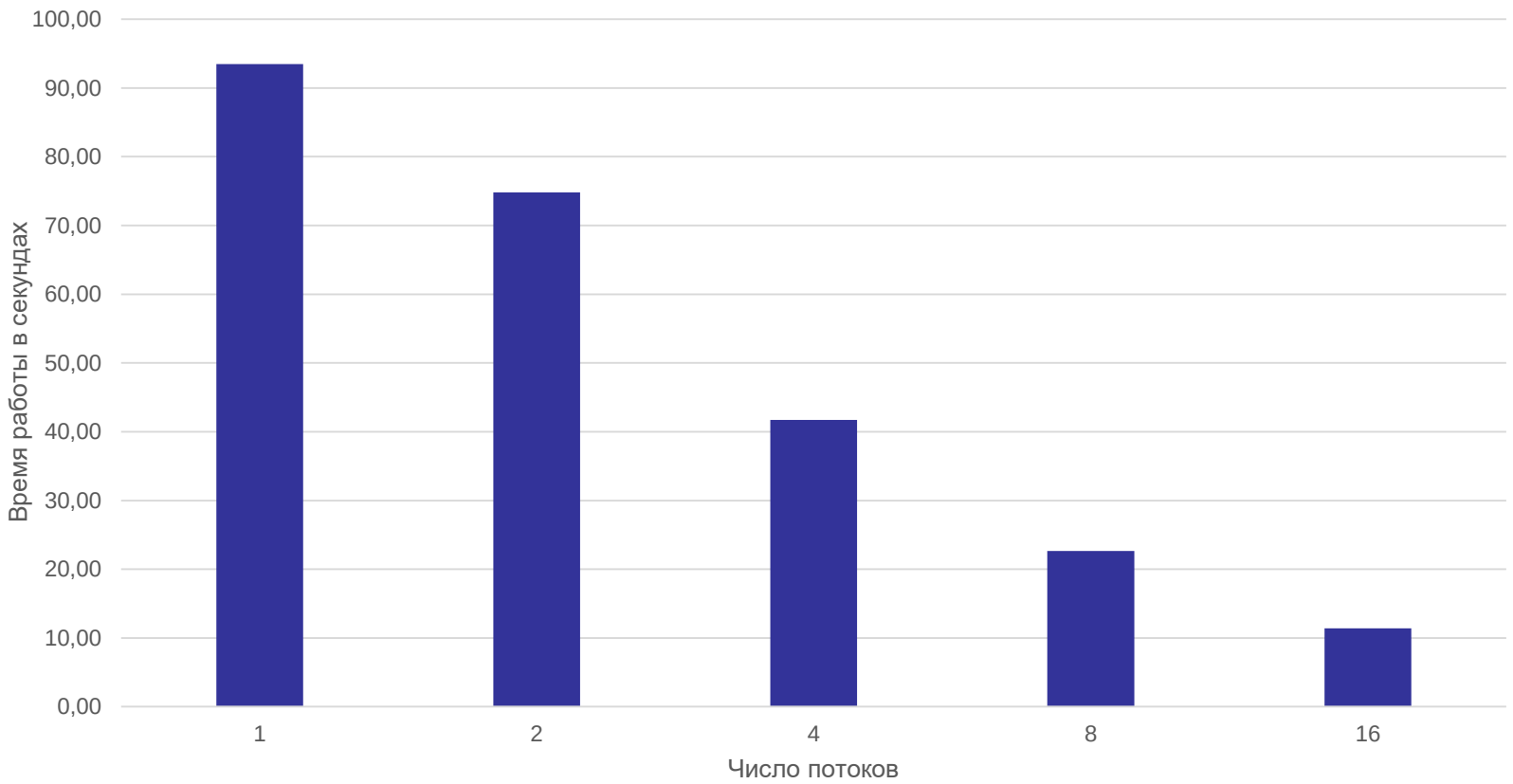
Студенческая работа (Орлов Максим, 2022 г.)

Время работы последовательных алгоритмов Брандеса с гибридным и негибридным обходом в зависимости от плотности графа. Число вершин – 3000.

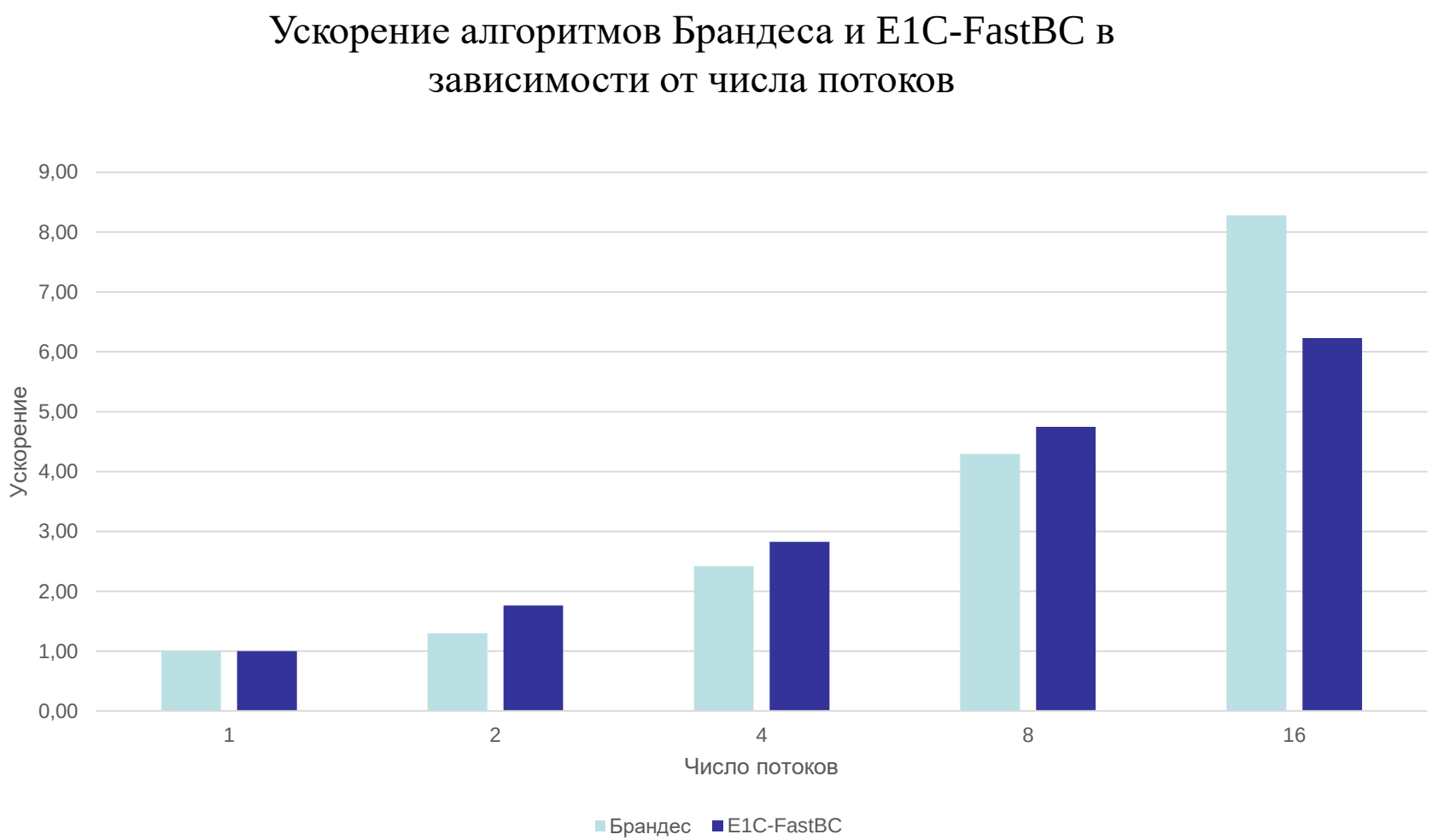


Студенческая работа (Орлов Максим, 2022 г.)

Время работы гибридного алгоритма Брандеса при числе вершин, равном 8000, в зависимости от числа потоков



Студенческая работа (Орлов Максим, 2022 г.)



Параллельный алгоритм для распределенной памяти

- Фазу 1 алгоритма (обход графа) заменим на алгоритм дельта-шага (label-correcting). Такой алгоритм имеет больший потенциал параллелизма, чем label-setting алгоритмы.
- Для каждой вершины храним множество предшественников P и множество последователей $Succ$ в дереве обхода графа → можно вычислить только P, $Succ$ получить транспонированием

Параллельный алгоритм для распределенной памяти

- Хранение:
  - Вершины графа случайно распределены между процессами. Каждый процесс хранит списки смежности его локальных вершин
  - Каждый процесс хранит свою очередь обхода графа для его локальных вершин.
  - Список предшественников и последователей хранится в динамических таблицах. Для случайных графов число предшественников и последователей известно

Параллельный алгоритм для распределенной памяти

Input: Graph $\mathcal{G}$

1 $\forall v \in V$: $C_B(v) = 0$;

2 **foreach** $s \in V$ **do**

```
    // Step 1: Compute shortest path
       predecessors
```

3 $\mathcal{P}_s = shortestpaths(\mathcal{G}, s)$;

```
    // Step 2: Compute shortest path
       successors from predecessors
```

4 $\mathcal{S}_s = transpose(\mathcal{P}_s)$;

```
    // Step 3: Compute path counts in
       DAG of shortest paths
```

5 $(\sigma_s, Q) = pathcounts(\mathcal{S}_s, s)$;

```
    // Step 4: Update betweenness
       centrality
```

6 $C_B = updatecentrality(\mathcal{P}_s, Q, \sigma_s, C_B)$;

- Вычислительная сложность $O(n \log^2 n)$ для невзвешенных графов, $O(\frac{n \log^3 n}{\log \log n})$ – для взвешенных графов

Параллельный алгоритм для распределенной памяти

Algorithm 3: $shortestpaths(G, s)$ – find shortest paths predecessor map – Δ-stepping [27]

Input: Weighted graph $\mathcal{G} = (V, E)$, vertex s
Output: $\forall v \in V$: predecessors $\mathcal{P}_s = \{p_i\}$ on all i shortest paths $(s, \ldots, p_i, v)$

```
1  ∀ v ∈ V : tent[v] = ∞; P_s[v] = ∅;
2  i = 0; B[0] = s; tent[s] = 0;
3  while B not empty do
4      D = ∅;
5      while B[i] ≠ ∅ do
6          R = {(v, w) | ∀ v ∈ B[i] ∧ c(v, w) ≤ Δ};
7          D = D ∪ B[i]; B[i] = ∅;
8          foreach (v, w) ∈ R do relax(v, w);
9      R = {(v, w) | ∀ v ∈ D ∧ c(v, w) > Δ};
10     foreach (v, w) ∈ R do relax(v, w);
11     i = i + 1;
```

Параллельный алгоритм для распределенной памяти

Algorithm 5: $pathcounts(\mathcal{S}_s, s)$ – accumulate shortest path counts

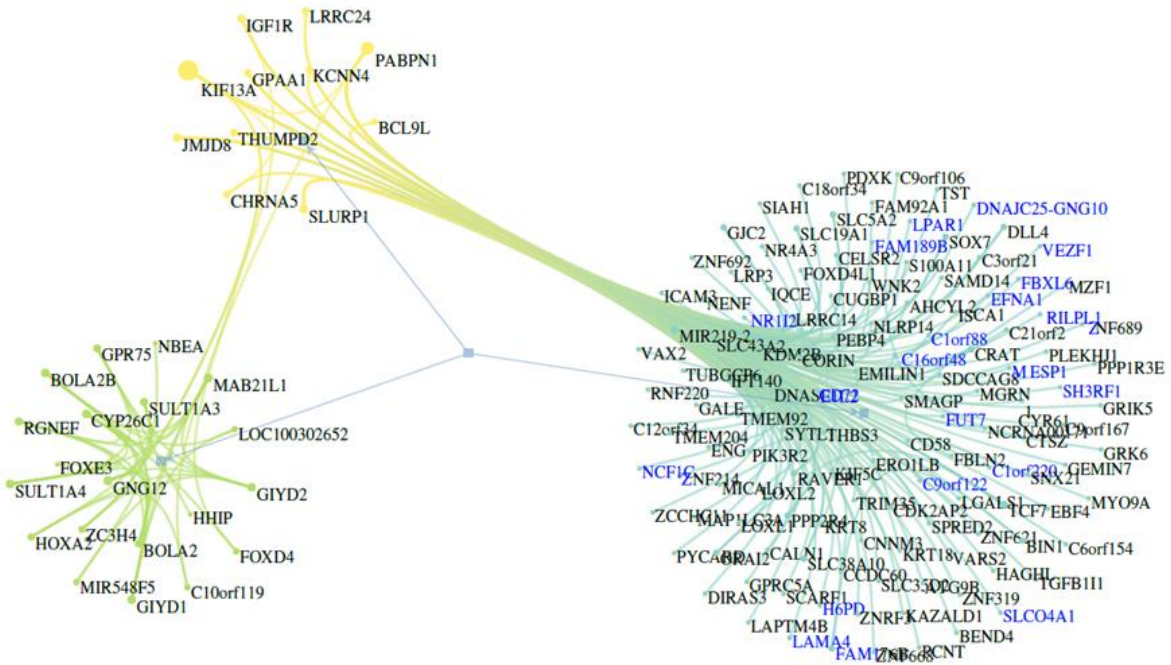
Input: Successor set $\mathcal{S}_s[v], \forall v \in V$, starting vertex s
Output: Number of shortest paths $\sigma[v]$, $\forall v \in V$, Q, a queue consisting of all vertices with no successors

```
1  localQ ← empty queue;
2  σ[t] = 0, ∀t ∈ V; σ[s] = 1;
3  enqueue s → localQ;
4  while localQ not empty do
5      dequeue v ← localQ;
6      foreach w ∈ S[v] do
7          σ[w] = σ[w] + σ[v];
8          enqueue w → localQ;
9      if S_s[v] = empty list then
10         enqueue v → Q;
```

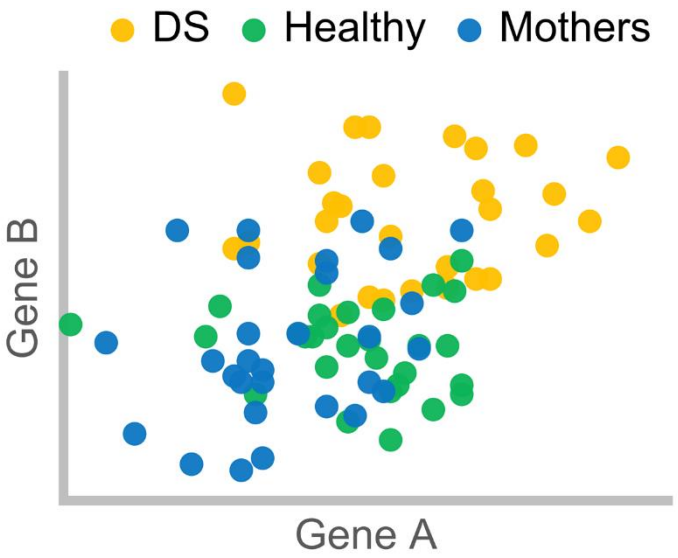
Пример. Исследование структуры ДНК

- Задача: детектирование изменений структуры метилированной ДНК, связанной с синдромом Дауна, в зависимости от возраста человека.
- Krivonosov M. et al. DNA methylation changes with age as a complex system: a parenclitic network approach to a family-based cohort of patients with Down Syndrome //bioRxiv. – 2020. [pdf]
- О проекте: https://github.com/mike-live/parenclitic
- Исходные данные: метилированное ДНК 29 семей (мать, здоровые братья, больной ребенок)
- Для каждого человека построен граф изменения ДНК: вершины – гены, ребро соединяет ген А и ген В, если есть отличие в этой паре генов от пациентов с синдромом Дауна (СД)

Пример. Исследование структуры ДНК



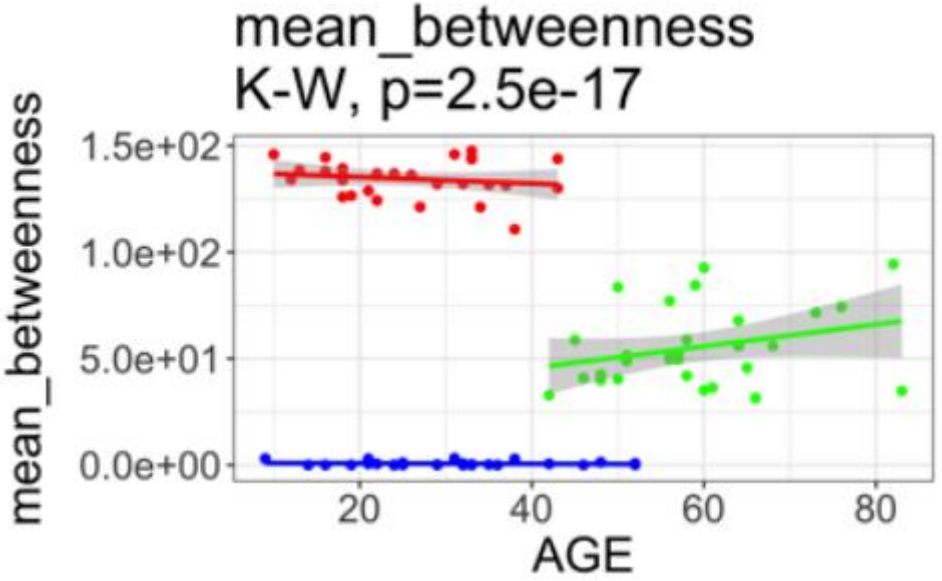
Граф ДНК пациента



Связь генов

Пример. Исследование структуры ДНК

- Зависимость среднего значения Betweenness Centrality графа ДНК от возраста человека



- У больных СД характеристика существенно больше, чем у их здоровых братьев и сестер
- У матерей с увеличением возраста характеристика увеличивается
- Изменения в геноме пациентов СД, связанные со старением, произошли раньше, чем у их матерей

Литература

1. Brandes U. A faster algorithm for betweenness centrality //J. of math. sociology. – 2001. – Т. 25. – № 2. – С. 163-177. [pdf]
2. Madduri K. et al. A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets //2009 IEEE International Symposium on Parallel & Distributed Processing. – IEEE, 2009. – С. 1-8. [pdf]
3. Edmonds N., Hoefler T., Lumsdaine A. A space-efficient parallel algorithm for computing betweenness centrality in distributed memory //2010 International Conference on High Performance Computing. – IEEE, 2010. – С. 1-10. [pdf]
4. Solomonik E. et al. Scaling betweenness centrality using communication-efficient sparse matrix multiplication //Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. – 2017. – С. 1-14. [pdf]

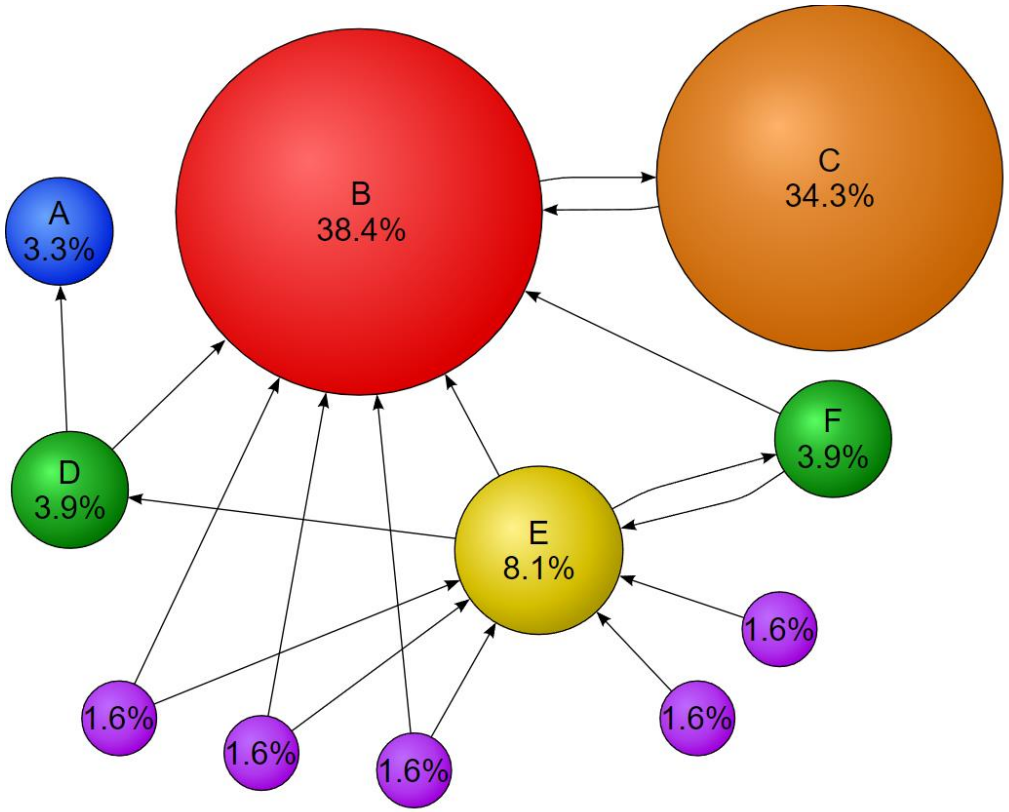
PAGE RANK

Введение

- PageRank – алгоритм ссылочного ранжирования веб-страниц, который вычисляет важность страницы в Web графе на основе числа ссылок на нее.
- Предложен Л. Пэйджем и С. Брином в 1998 г. Использовался в поисковой системе Google с 1998 по 2016 г.
- Применение в веб-графах:
  - Обработка запросов в поисковых сетях
  - Ранжирование сайтов
  - Исследование влияния сайта или сообщества на Интернет в целом
  - Детектирование спама
  - Конфигурирование программы-сборщика (web crawler)
  - Персональные рекомендации в социальной сети

Введение

- рейтинг вебстраницы (PageRank) для простой сети, выраженный в процентах (Google использует логарифмическую шкалу).



https://ru.wikipedia.org/wiki/PageRank

Введение

- Применение в других областях:
  - Оценка научного влияния исследователей (pagerank-index, Pi) в научных базах
  - Анализ белковых сетей (ProteinBank), выявление значимых генов в сети связей между генами (GeneBank)
  - Прогнозирование перемещения людей и транспорта в транспортной сети

- Подробнее: Gleich D. F. PageRank beyond the Web //Siam Review. – 2015. – Т. 57. – №. 3. – С. 321-363. [pdf]

Определение

- Модель случайного блуждания (*random surfer*): пользователь случайно перемещается с текущей страницы веб-графа
  - (1) по ссылке, расположенной на этой странице, с вероятностью c,
  - (2) телепортируется на любую страницу веб-графа с вероятностью $1 - c$, согласно заданному вектору v (*teleportation distribution vector*).
- В общем случае рассматривается перемещение по случайной матрице P

Определение

- Пусть $\boldsymbol{A} = (a_{ij})$ – матрица смежности графа,
  $\deg(i) = \sum_j a_{ij}$ – степень вершины i
- $\boldsymbol{v}$ – случайный вектор $(\boldsymbol{e}^T\boldsymbol{v} = 1)$, $\boldsymbol{e}$ – вектор из единиц
- Нормализуем матрицу $\boldsymbol{A}$, учтем страницы без исходящих ссылок (*dangling pages*) вставкой индикатора такой страницы:
  - $\boldsymbol{P} = (p_{ij}) = (\frac{a_{ij}}{\deg(i)}), \deg(i) > 0$
  - $\boldsymbol{P}' = \boldsymbol{P} + \boldsymbol{d}\boldsymbol{v}^T$,
  - $d = \delta(\deg(i), 0)$ – индикатор
- PageRank – стационарная точка решения уравнения

$$p = \boldsymbol{P}''^T p \qquad (1)$$
$$\boldsymbol{P}'' = c\boldsymbol{P}' + (1-c)(ev^T)$$

Параметры

- Матрица $\boldsymbol{P''}$ разреженная, несимметричная
- Виды распределения для вектора телепортации v:
  - Равномерное в классической постановке задачи, $v_i = \frac{1}{n}$ → равномерное перемещение по всему графу
  - Неравномерное для персонализированного PageRank → перемещение по некоторой части графа

- $0.85 < c < 0.99$

Методы вычисления PageRank

Метод степенной итерации

- Вектор PageRank можно получить, итерационно решая уравнение $p = \boldsymbol{P''}^T p$

1. $p^{(0)} = v, k = 0$
2. **Пока** решение изменяется, $(\left\| p^{(k+1)} - p^{(k)} \right\| > \delta)$, найти $p^{(k+1)} = \boldsymbol{P''}^T p^{(k)}$

- $y = \boldsymbol{P''}^T x$ можно вычислить как

$$y = c\boldsymbol{P}x$$
$$\gamma = \|x\| - \|y\|$$
$$y = y + \gamma v$$

Метод степенной итерации

Algorithm 1. (Poor man's PageRank algorithm.)

Input: Given a transition matrix P, a teleportation vector v, and a coefficient c

Output: Compute PageRank p

begin

Initialize $p^{(0)} = v$, $k = 0$

repeat

$p^{(k+1)} = cP^T p^{(k)}$

$\gamma = \|p^{(k)}\| - \|p^{(k+1)}\|$

$p^{(k+1)} = p^{(k+1)} + \gamma v$

$\delta = \|p^{(k+1)} - p^{(k)}\|$

$k = k + 1$

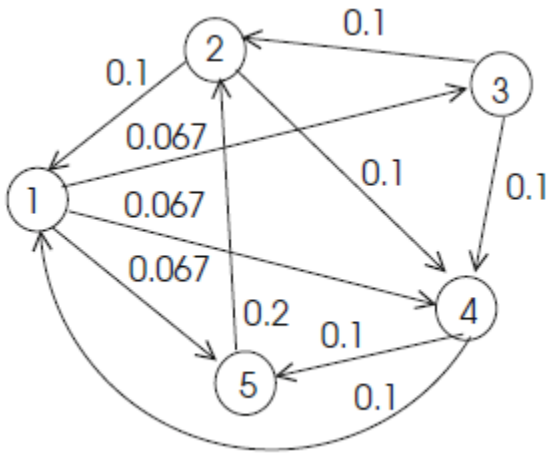
until $\delta < \epsilon$

end

return $p^{(k)}$

Berkhin P. A survey on PageRank computing //Internet mathematics. – 2005. – Т. 2. – №. 1. – С. 73-120.

PageRank. Пример



Vertices	1	2	3	4	5
n_{out}	3	2	2	2	1
$k = 1$: *weight/edge*	0.067	0.1	0.1	0.1	0.2
rank	0.2	0.3	0.067	0.267	0.167
$k = 2$: *weight/edge*	0.067	0.15	0.034	0.134	0.167
rank	0.284	0.201	0.034	0.251	0.201
$k = 3$: *weight/edge*	0.095	0.101	0.017	0.067	0.167
rank	0.168	0.184	0.095	0.213	0.162

Erciyes K. Guide to Graph Algorithms. – Springer International Publishing, 2018.

Параллельный метод степенной итерации для систем с общей памятью

- Besta M. et al. «To push or to pull…»
- Push(v) – суммирование вклада v к соседним с ней вершинам
- Pull(v) – суммирование вклада соседей v к ней

```
1 /* Input: a graph G, a number of steps L, the damp parameter f
2   Output: An array of ranks pr[1..n] */
3
4 function PR(G,L,f) {
5   pr[1..v] = [f..f]; //Initialize PR values.
6   for(l = 1; l < L; ++l) {
7     new_pr[1..n] = [0..0];
8     for v ∈ V do in par {
9       update_pr(); new_pr[v] += (1−f)/n; pr[v] = new_pr[v];
10 } } }
11
12 function update_pr() {
13   for u ∈ N(v) do [in par] {
14     {new_pr[u] += (f·pr[v])/d(v) (W) (f);}        PUSHING
15
16     {new_pr[v] += (f·pr[u])/d(u) (R);}            PULLING
17 } }
```

Параллельный метод степенной итерации для систем с общей памятью

- Оптимизация «push»: Разделить списки смежности каждой вершины на локальную для потока и удаленную части. Это позволяет сократить число обращений к общей памяти

```
1 //The code below corresponds to lines 19-10 in Algorithm 1.
2 //V_L is a set of vertices owned by a local executing tread.
3 //V_G is a set of vertices owned by a tread different from the
4 //local one. V_L ∪ V_G = V; V_L ∩ V_G = ∅.
5
6 for v ∈ V_L do in par                                  PART 1: LOCAL UPDATES
7   for u ∈ N(v) do [in par]
8     new_pr[u] += (f·pr[v])/d(v)
9
10 barrier(); //A lightweight barrier to synchronize all threads.
11
12 for v ∈ V_G do in par                                 PART 2: REMOTE UPDATES
13   for u ∈ N(v) do [in par]
14     new_pr[u] += (f·pr[v])/d(v)  W  i
```

Параллельный метод степенной итерации. Вычислительные эксперименты

- Источник:
  Besta M. et al. To push or to pull: On reducing communication and synchronization in graph computations //Proceedings of the 26th International Symposium on High-Performance Parallel and Distributed Computing. – 2017. – С. 93-104.
- Тестовые матрицы:

Type	ID	n	m	$\bar{d}$	$\bar{D}$
R-MAT graphs	rmat	33M-268M	66M-4.28B	2-16	19-33
Social networks	orc	3.07M	117M	39	9
	pok	1.63M	22.3M	18.75	11
Ground-truth [53] community	ljn	3.99M	34.6M	8.67	17
Purchase network	am	262k	900k	3.43	32
Road network	rca	1.96M	2.76M	1.4	849

Параллельный метод степенной итерации. Вычислительные эксперименты

- Время работы

G	PageRank [ms] orc	pok	ljn	am	rca
Pushing	572	129	264	4.62	6.68
Pulling	557	103	240	2.46	5.42

pull быстрее push
от 1,03 до 1,88 раз

- Профилировка
  - В pull больше кэш-промахов,
  - меньше операций чтения, записи,
  - Меньше ветвлений при доступе к соседям

Event	orc (PR) Push	Push+PA	Pull	rca (PR) Push	Push+PA	Pull
L1 misses	335M	382M	572M	2,062M	10,560M	2,857M
L2 misses	234M	289M	446M	640k	7,037M	1,508M
L3 misses	64,75M	53,49M	181M	348M	537k	866k
TLB misses (data)	130M	142M	129M	12,21k	274k	21628
TLB misses (inst)	1188	336	1161	220	250	218
atomics	234M	219M	0	5,533M	5,374M	0
locks	0	0	0	0	0	0
reads	1,196B	1,183B	1,187B	43,39M	62,59M	37,49M
writes	474M	460M	237M	14.99M	14,86M	7,499M
branches (uncond)	234M	222M	1971	5,533M	7,340M	533
branches (cond)	474M	466M	240M	15M	18,79M	9.467M

Экстраполяционный метод

- Свойства системы (1):
  - Граф с матрицей смежности P'' *сильно связный* → λ_1 - простое число, собственный вектор u_1 с положительными компонентами
  - Граф с матрицей смежности P'' *апериодичный*, т.е. для любых i и j существует путь любой длины l (с повторениями) за исключением конечного числа длин → итерационное решение системы (1) сойдется.
- Собственные числа: $\lambda_1(\boldsymbol{P}) = \lambda_1(\boldsymbol{P''}) = 1, \lambda_2(\boldsymbol{P''}) = c.$
- Решение системы (1) можно представить в виде

$$p^{(k)} = u_1 + \lambda_2^k u_2 + \cdots + \lambda_m^k u_m + \cdots$$

- Haveliwala, Kamvar (2003):

$$p^{(k-d)} = u_1 + u_2 + \cdots + u_{d+1} + \cdots$$
$$p^{(k)} = u_1 + \lambda_2^d u_2 + \cdots + \lambda_{d+1}^d u_d + \cdots$$

Блочный метод

- Идея: представим всю сеть как совокупность доменов (блоков). Связей между страницами внутри домена больше, чем между разными доменами. → PageRank всей сети получим как сумму «локальных» PageRank для доменов.
- Преимущества метода:
  - Локальные PageRank можно вычислять приближенно
  - Вычисления отдельных блоков можно назначить разным потокам / процессам
  - В динамике связи изменяются внутри части блоков, можно пересчитывать только соответствующие данные

Блочный метод

Algorithm 2. (Block structure method (*blockRank*).)

Input: Given a graph W over nodes $G = \bigcup G_I$, $I = 1 : N$, a teleportation vector v, and a coefficient c

Output: Compute PageRank p

begin

  for $I = 1 : N$ **do**

    let P_{II} be a transition matrix over block G_I

    for $i \in G_I$ set $v_{I,i} = v_i / \sum_{j \in G_I} v_j$

    $l_I = pageRank(P_{II}, v_I, v_I)$ ← локальный PR

  end

  for $I = 1 : N$ **do**

    $\tilde{v}_I = \sum_{i \in G_I} v_i$

    for $J = 1 : N$ **do**

      $\tilde{L}_{IJ} = \sum_{i \in I, j \in J} P_{ij} l_i$ ← $\tilde{L}$ - матрица связей между блоками

    end

  end

  let $\tilde{P}$ be a transition matrix corresponding to weighted block structure $\tilde{L}$

  $b = pageRank(\tilde{P}, \tilde{v}, \tilde{v})$

  set $s = (b_1 l_1, b_2 l_2, \ldots, b_N l_n)$ ← b – вес локального PR в общем PR

  $p = pageRank(P, s, v)$ s - начальное приближение общего PR

end

return p

Применение линейной алгебры

- Вычисление PageRank можно свести к решению СЛАУ с разреженной матрицей:

$$p = \boldsymbol{P''}^T p \ , \boldsymbol{P''} = c\boldsymbol{P'} + (1-c)(ev^T)$$
$$(cP^T + c(vd^T) + (1-c)(ve^T))x = x$$
$$(I - cP^T)x = k\,v \qquad (2)$$
$$k = k(x) = (1-c)\|x\| + d^T x = 1 - c + d^T x$$

- (2) – СЛАУ с разреженной матрицей, если k – фиксированное.
- СЛАУ (2) решается итерационными методами (методы Якоби, GMRES, BiCG, BiCGStab). Для улучшения сходимости методов применяется предобуславливание системы (2).

Литература. PageRank

1. Brin S., Page L. The anatomy of a large-scale hypertextual web search engine. – 1998. [pdf]
2. Berkhin P. A survey on PageRank computing //Internet mathematics. – 2005. – Т. 2. – №. 1. – С. 73-120. [pdf]
3. Gleich D. F. PageRank beyond the Web //Siam Review. – 2015. – Т. 57. – №. 3. – С. 321-363. [pdf]
4. Gleich D., Zhukov L., Berkhin P. Fast parallel PageRank: A linear system approach //Yahoo! Research Technical Report YRL-2004-038 – 2004. – Т. 13. – С. 22. [pdf]
5. Beamer S., Asanović K., Patterson D. Reducing pagerank communication via propagation blocking //2017 IEEE International Parallel and Distributed Processing Symposium (IPDPS). – IEEE, 2017. – С. 820-831. [pdf]
6. Gleich D. F. et al. An inner-outer iteration for computing PageRank //SIAM Journal on Scientific Computing. – 2010. – Т. 32. – №. 1. – С. 349-371. [pdf]

Литература общая

1. Newman M. Networks: an Introduction. – Oxford university press, 2018.
2. Easley D. et al. Networks, crowds, and markets. – Cambridge: Cambridge university press, 2010. – Т. 8. [URL]
3. Costa L. F. et al. Characterization of complex networks: A survey of measurements //Advances in physics. – 2007. – Т. 56. – №. 1. – С. 167-242. [pdf]