



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ

КАФЕДРА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ И СИСТЕМНОГО ПРОГРАММИРОВАНИЯ

ЛАБОРАТОРИЯ ITLav



НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО
ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МАТЕМАТИКИ И МЕХАНИКИ
КАФЕДРА ВЫСОКОПРОИЗВОДИТЕЛЬНЫХ ВЫЧИСЛЕНИЙ И СИСТЕМНОГО ПРОГРАММИРОВАНИЯ
ЛАБОРАТОРИЯ ITLav

***ВВЕДЕНИЕ В АНАЛИЗ ПРОИЗВОДИТЕЛЬНОСТИ
И ОПТИМИЗАЦИЮ ПРОГРАММ***

**Степень посредничества вершин.
Матрично-векторная реализация
алгоритма Брандеса**



**LOBACHEVSKY
UNIVERSITY**

Устинов А.В., Пирова А.Ю., Мееров И.Б.

Содержание

- ❑ Введение
- ❑ Степень посредничества вершины. Алгоритм Брандеса
- ❑ Матричная версия алгоритма
- ❑ Оптимизация алгоритма
- ❑ Результаты экспериментов
- ❑ Заключение

Цель лекции

- Рассмотреть проблематику, связанную с высокопроизводительной реализацией алгоритмов на графах
 - Изучить проблемы производительности при реализации алгоритмов на графах
 - Познакомиться с подходом, лежащим в основе библиотеки GraphBLAS
 - На примере алгоритма поиска степени посредничества вершин изучить некоторые типовые приёмы оптимизации производительности, применяемые при реализации алгоритмов на графах

ВВЕДЕНИЕ

1. Введение

1.1. Алгоритмы на графах и их проблемы

- ❑ Алгоритмы на графах применяются для анализа сетей в различных предметных областях (химия, биология, медицина, транспортные и социальные сети)
- ❑ Трудности реализации алгоритмов на графах:
 - Многообразие форматов хранения графов
 - Плотная матрица смежности, список смежности, CRS, специальные форматы
 - Низкая производительность
 - Малая арифметическая интенсивность, нелокальный доступ к памяти
 - Сложность эффективной параллельной реализации
 - Ограничения по скорости работы памяти, синхронизация между потоками
- ❑ Есть ли решения для этих проблем?

1. Введение

1.1. Алгоритмы на графах и их проблемы

- ❑ Алгоритмы на графах применяются для анализа сетей в различных предметных областях (химия, биология, медицина, транспортные и социальные сети)
- ❑ Трудности реализации алгоритмов на графах:
 - Многообразие форматов хранения графов
 - Плотная матрица смежности, список смежности, CRS, специальные форматы
 - Низкая производительность
 - Малая арифметическая интенсивность, нелокальный доступ к памяти
 - Сложность эффективной параллельной реализации
 - Ограничения по скорости работы памяти, синхронизация между потоками
- ❑ Частичное решение – **стандарт GraphBLAS**

1. Введение

1.2. GraphBLAS

- ❑ Стандарт GraphBLAS определяет набор матрично-векторных операций для реализации алгоритмов на графах
- ❑ Граф представляется в виде разреженной матрицы смежности
- ❑ Основные матрично-векторные операции:
 - Применение функции к элементам матрицы
 - Поэлементное сложение и умножение матриц: $A .* B$
 - Умножение матриц: $A \oplus . \otimes B$
- ❑ Достоинства:
 - Единый формат представления графов
 - Высокая производительность достигается за счёт оптимизации небольшого набора основных операций

1. Введение

1.3. Полукольца в GraphBLAS

- Сложение и умножение в GraphBLAS возможны в различных *полукольцах*
- **Полукольцо** $(R, \oplus, \otimes, 0, 1)$ – множество R с заданными на нём бинарными операциями *сложения* $\oplus$ и *умножения* $\otimes$ со следующими свойствами:
 - Сложение – ассоциативность, коммутативность, существование нейтрального элемента 0
 - Умножение – ассоциативность, существование нейтрального элемента 1
 - Дистрибутивность сложения и умножения
- Умножение матриц и векторов в полукольце обозначается $A \oplus \otimes B$ или AB
- Примеры полуколец в GraphBLAS и их применения:
 - $(\mathbb{R}, +, \times, 0, 1)$ – *стандартная арифметика* – подсчёт числа кратчайших путей
 - $(\{0, 1\}, \vee, \wedge, 0, 1)$ – *булева алгебра* – поиск в ширину, проверка связности вершин
 - $(\mathbb{R} \cup \{+\infty\}, \min, +, +\infty, 0)$ – *тропическое полукольцо (min-plus semiring)* – поиск кратчайших путей во взвешенном графе

1. Введение

1.4. Операции с масками

- ❑ Операции GraphBLAS поддерживают использование маски, которая является вектором или матрицей той же размерности, что и результат операции
- ❑ Применение маски M к объекту A обозначается как $M \odot A$
- ❑ В результате применения маски входящие в маску элементы не меняются, а остальные становятся равными нулю
- ❑ Возможно использование инвертированной маски $\neg M$, тогда исключаются входящие в маску элементы, а не входящие не меняются
- ❑ Пример:

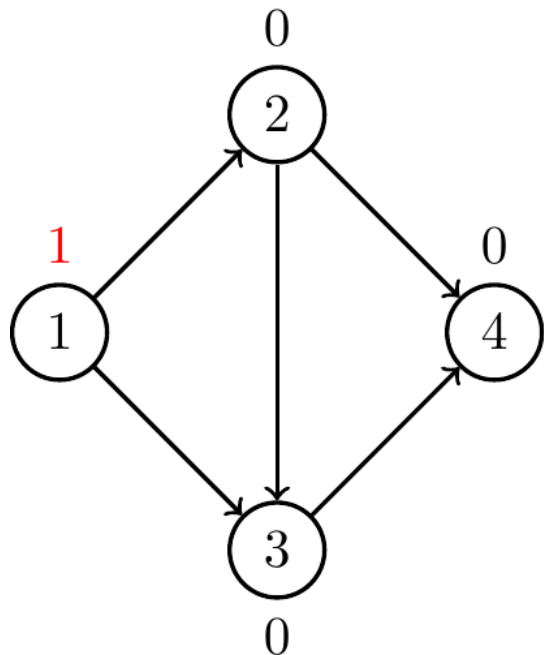
$$- A = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, B = \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}, M = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}$$

$$- A + B = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}, M \odot (A + B) = \begin{pmatrix} 0 & 2 \\ 2 & 0 \end{pmatrix}, \neg M \odot (A + B) = \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix}$$

1. Введение

1.5. Пример. Поиск в ширину (1)

- Подсчёт числа кратчайших путей от вершины 1 до остальных вершин в невзвешенном ориентированном графе – полукольцо $(\mathbb{R}, +, \times, 0, 1)$:



Матрица смежности: $A = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Вектор числа кратчайших путей на шаге 0: $c = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

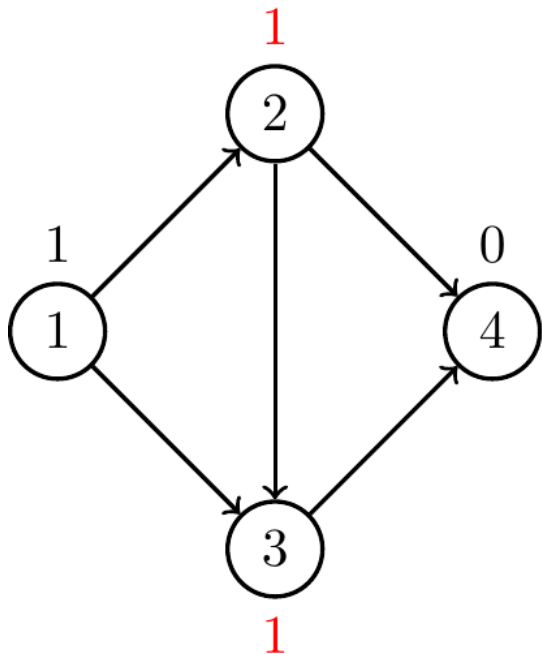
Фронт поиска в ширину на шаге 0: $f = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Шаг i : $f := (\neg c) \odot (A^T f), c := c + f$

1. Введение

1.5. Пример. Поиск в ширину (2)

- Подсчёт числа кратчайших путей от вершины 1 до остальных вершин в невзвешенном ориентированном графе – полукольцо $(\mathbb{R}, +, \times, 0, 1)$:



$$\text{Шаг 1: } c = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, f = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

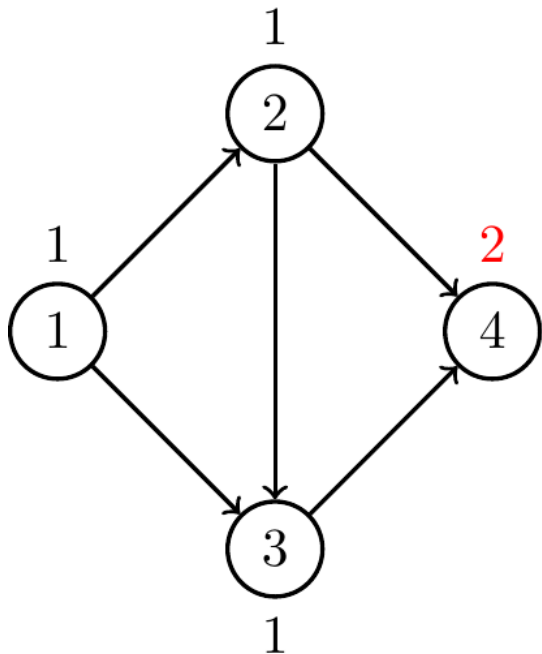
$$A^T f = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

$$f := (\neg c) \odot (A^T f) = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \end{pmatrix} \odot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}, c := \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

1. Введение

1.5. Пример. Поиск в ширину (3)

- Подсчёт числа кратчайших путей от вершины 1 до остальных вершин в невзвешенном ориентированном графе – полукольцо $(\mathbb{R}, +, \times, 0, 1)$:



$$\text{Шаг 2: } c = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix}, f = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix}$$

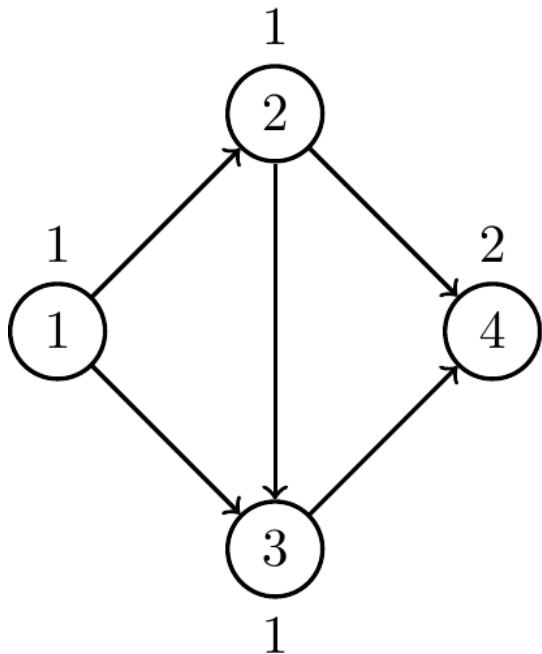
$$A^T f = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \end{pmatrix}$$

$$f := (\neg c) \odot (A^T f) = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \odot \begin{pmatrix} 0 \\ 0 \\ 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \end{pmatrix}, c := \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 0 \\ 2 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \end{pmatrix}$$

1. Введение

1.5. Пример. Поиск в ширину (4)

- Подсчёт числа кратчайших путей от вершины 1 до остальных вершин в невзвешенном ориентированном графе – полукольцо $(\mathbb{R}, +, \times, 0, 1)$:



Вектор числа кратчайших путей: $c = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 2 \end{pmatrix}$

СТЕПЕНЬ ПОСРЕДНИЧЕСТВА ВЕРШИНЫ. АЛГОРИТМ БРАНДЕСА

2. Степень посредничества вершины. Алгоритм Брандеса

2.1. Степень посредничества вершины

□ Пусть $G = (V, E)$ – сильно связный граф, n – число вершин, m – число рёбер

□ **Степень посредничества вершины** $v \in V$ есть

$$C_B(v) = \sum_{s,t \in V \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}}, \quad (1)$$

где σ_{st} - число всех кратчайших путей из s в t ,

$\sigma_{st}(v)$ - число кратчайших путей из s в t , проходящих через v

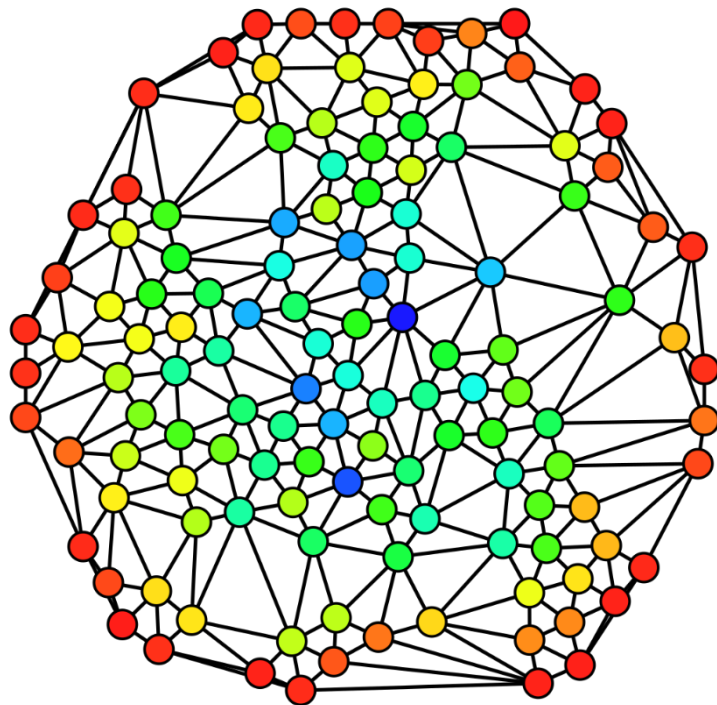
□ Введена Дж. Антониссе в 1971 г. и Л. Фриманом в 1977 г.

□ Используется в биоинформатике и в анализе социальных и транспортных сетей для определения важности вершин: чем больше степень посредничества, тем важнее вершина

2. Степень посредничества вершины. Алгоритм Брандеса

2.1. Степень посредничества вершины

□ Пример:



Красный – меньше степень посредничества
Синий – больше степень посредничества

Как вычислить?

2. Степень посредничества вершины. Алгоритм Брандеса

2.2. Простой алгоритм

- Как определить, что v лежит на кратчайшем пути из s в t ?
 - Пусть d_{xy} - длина кратчайшего пути из $x \in V$ в $y \in V$
 - Тогда должно выполняться $d_{st} = d_{sv} + d_{vt}$
- Как посчитать число кратчайших путей из s в t , проходящих через v ?
 - Вспомним, что σ_{xy} - число кратчайших путей из $x \in V$ в $y \in V$
 - Любой такой путь состоит из кратчайшего пути из s в v и кратчайшего пути из v в t , каждый из них выбирается независимо от другого
 - Поэтому $\sigma_{st}(v) = \sigma_{sv} \cdot \sigma_{vt}$
- Как посчитать d_{xy} и σ_{xy} ?
 - d_{xy} - алгоритм Флойда-Уоршелла ($\Theta(n^3)$) или алгоритм Дейкстры ($O(mn + n^2 \log n)$)
 - σ_{xy} - поиск в ширину ($O(mn + n^2)$)
- Затем $C_B(v)$ считается для всех $v \in V$ за $\Theta(n^3)$

2. Степень посредничества вершины. Алгоритм Брандеса

2.3. Проблемы простого алгоритма

- Длина и число кратчайших путей хранятся для всех пар вершин
 - $\Theta(n^2)$ памяти – неприемлемо для достаточно больших графов
- Общее время работы составляет $\Theta(n^3)$
 - Как минимум, вычисление $C_B(v) = \sum_{s \in V \setminus \{v\}} \sum_{t \in V \setminus \{v\}} \frac{\sigma_{sv} \cdot \sigma_{vt}}{\sigma_{st}}$ для всех вершин требует столько времени
 - Даже для небольших графов ($n \approx 10^5$) время работы слишком большое
- Следовательно, для больших задач (n порядка $10^6 - 10^9$) алгоритм не подходит

- Какие возможны улучшения?

2. Степень посредничества вершины. Алгоритм Брандеса

2.3. Проблемы простого алгоритма

- Длина и число кратчайших путей хранятся для всех пар вершин
 - $\Theta(n^2)$ памяти – неприемлемо для достаточно больших графов
- Общее время работы составляет $O(n^3)$
 - Как минимум, вычисление $C_B(v) = \sum_{s \in V \setminus \{v\}} \sum_{t \in V \setminus \{v\}} \frac{\sigma_{sv} \cdot \sigma_{vt}}{\sigma_{st}}$ для всех вершин требует столько времени
 - Даже для небольших графов ($n \approx 10^4$) время работы слишком большое
- Следовательно, для больших задач (n порядка $10^6 - 10^9$) алгоритм не подходит

- Какие возможны улучшения? [Алгоритм Брандеса](#)

2. Степень посредничества вершины. Алгоритм Брандеса

2.3. Алгоритм Брандеса

- Алгоритм предложен в 2001 г. У. Брандесом
- Время работы составляет $O(nm)$ на невзвешенных графах и $O(nm + n^2 \log n)$ на взвешенных графах
- Потребление памяти составляет $O(n + m)$
- Введём понятие зависимости v от s :

$$\delta_s(v) = \sum_{t \in V \setminus \{v\}} \frac{\sigma_{st}(v)}{\sigma_{st}} \quad (2)$$

Это вклад, который вносит начальная вершина s в $C_B(v)$.

- На основе $\delta_s(v)$ степень посредничества v вычисляется как

$$C_B(v) = \sum_{s \in V \setminus \{v\}} \delta_s(v) \quad (3)$$

2. Степень посредничества вершины. Алгоритм Брандеса

2.3. Алгоритм Брандеса

- Алгоритм состоит из **прямого хода** и **обратного хода**, которые выполняются для каждой $s \in V$
- В **прямом ходе** с помощью поиска в ширину вычисляются σ_{st} для всех $t \in V$
- В **обратном ходе** вычисляется $\delta_s(v)$ для всех $v \in V$ по формуле:

$$\delta_s(v) = \sum_{w \in S(s,v)} \frac{\sigma_{sv}}{\sigma_{sw}} (1 + \delta_s(w)), \quad (4)$$

где $S(s, v)$ – множество вершин, следующих за v на кратчайших путях из s

- Время работы прямого хода: для невзвешенного графа - $O(n + m)$, для взвешенного графа - $O(m + n \log n)$ (алгоритм Дейкстры с фибоначчиевой кучей)
- Время работы обратного хода: $O(n + m)$
- Потребление памяти: $O(n + m)$ – храним данные только для текущей s

МАТРИЧНАЯ ВЕРСИЯ АЛГОРИТМА

3. Матричная версия алгоритма

3.1. Матрично-векторный алгоритм Брандеса

- Сфокусируемся на невзвешенных графах
- Прямой ход и обратный ход можно описать с помощью матричных операций

- $A \in \mathbb{R}^{n \times n}$ - матрица смежности графа
- Σ_k - множество вершин на расстоянии k от вершины s
- $\mathbf{f}^{(k)} \in \mathbb{R}^n$ - k -й фронт поиска в ширину: $\mathbf{f}_i^{(k)} = \begin{cases} 0, i \notin \Sigma_k \\ \sigma_{si}, i \in \Sigma_k \end{cases}$
- $\mathbf{p}^{(k)} \in \mathbb{R}^n$ - вектор, где отмечены посещённые за первые k шагов вершины: $\mathbf{p}_i^{(k)} = \begin{cases} 0, i \notin \bigcup_{j=0}^k \Sigma_j \\ \sigma_{si}, i \in \bigcup_{j=0}^k \Sigma_j \end{cases}$
- $\mathbf{b}^{(k)} \in \mathbb{R}^n$ - зависимости от вершины s на шаге k

Прямой ход:

$$\begin{aligned} \mathbf{f}^{(0)} &= \begin{cases} 1, i = s \\ 0, i \neq s \end{cases}, \mathbf{p}^{(0)} = \mathbf{f}^{(0)} \\ \mathbf{f}^{(k+1)} &= (\neg \mathbf{p}^{(k)}) \odot (A^T \mathbf{f}^{(k)}) \\ \mathbf{p}^{(k+1)} &= \mathbf{p}^{(k)} + \mathbf{f}^{(k+1)} \end{aligned}$$

Шаг d - последний, если $\mathbf{f}^{(d+1)} = 0$

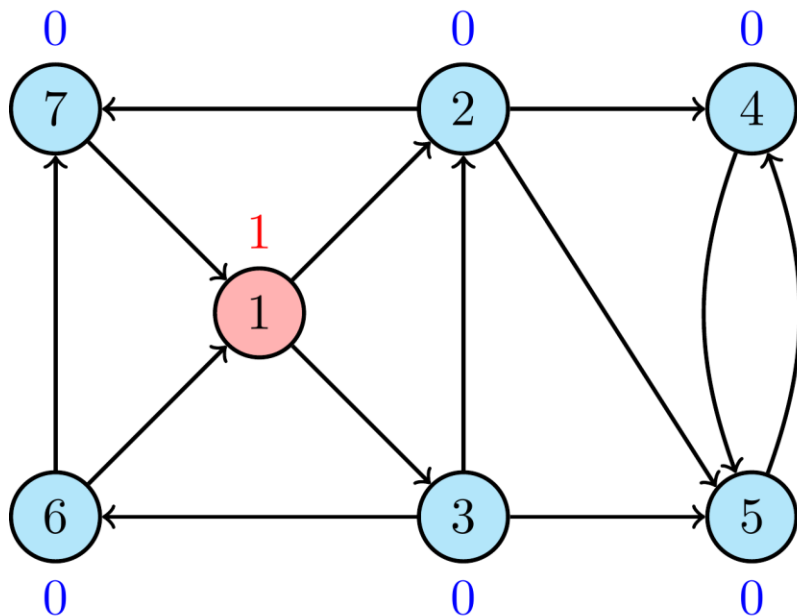
Обратный ход:

$$\begin{aligned} \mathbf{b}^{(d)} &= \mathbf{1}_n = (1, 1, \dots, 1)^T \\ \mathbf{w}^{(k)} &= \mathbf{f}^{(k)} \odot (\mathbf{b}^{(k)} ./ \mathbf{p}) \\ \mathbf{b}^{(k-1)} &= \mathbf{b}^{(k)} + (\mathbf{f}^{(k-1)} \odot A \mathbf{w}^{(k)}) ./ \mathbf{p} \end{aligned}$$

Выполнять, пока $k \geq 2$

3. Матричная версия алгоритма

3.2. Матрично-векторный алгоритм Брандеса. Пример

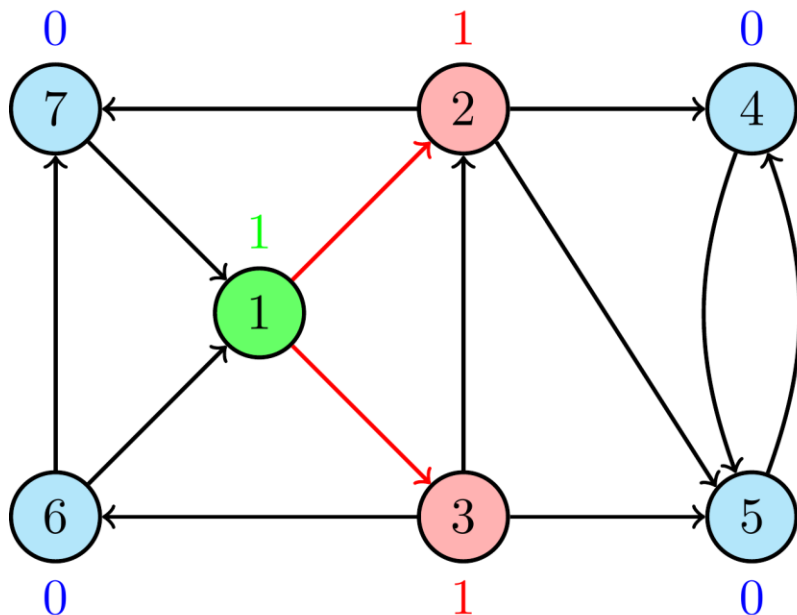


$$A = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\mathbf{f}^{(0)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \mathbf{p}^{(0)} = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \neg \mathbf{p}^{(0)} = \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

3. Матричная версия алгоритма

3.2. Матрично-векторный алгоритм Брандеса. Пример

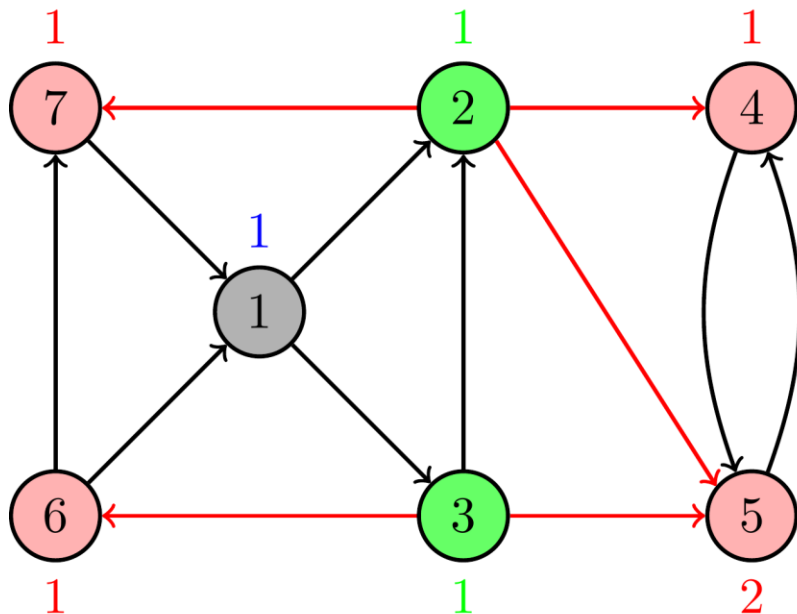


$$\mathbf{f}^{(1)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \odot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} =$$

$$= \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \mathbf{p}^{(1)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \neg \mathbf{p}^{(1)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}$$

3. Матричная версия алгоритма

3.2. Матрично-векторный алгоритм Брандеса. Пример

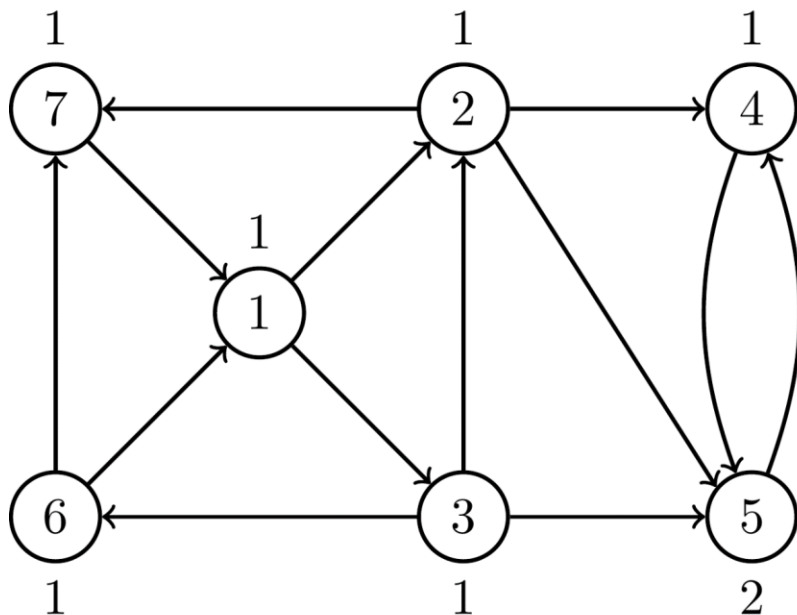


$$\mathbf{f}^{(2)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \odot \begin{pmatrix} 0 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 1 \\ 1 \end{pmatrix}, \mathbf{p}^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \end{pmatrix}, \neg \mathbf{p}^{(2)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

3. Матричная версия алгоритма

3.2. Матрично-векторный алгоритм Брандеса. Пример

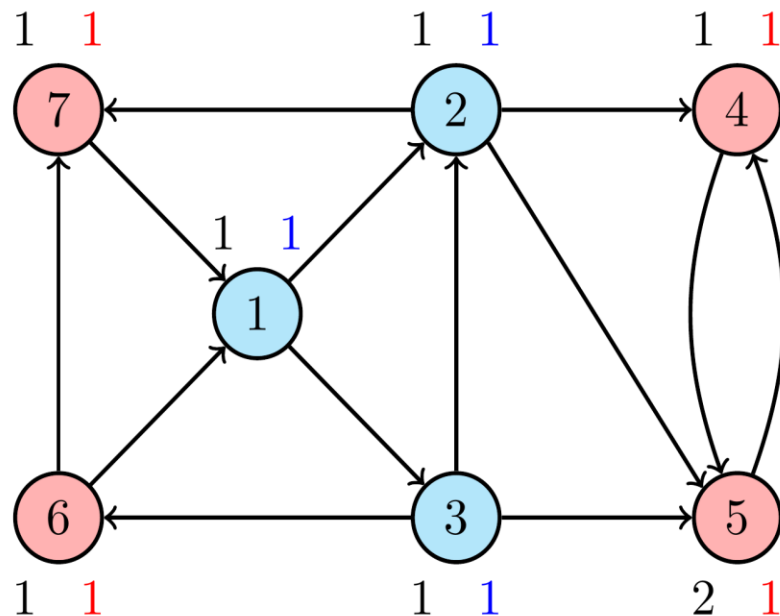


$$\mathbf{f}^{(3)} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 1 \\ 1 \end{pmatrix} \odot \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} =$$

$$= \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \text{ прямой ход завершён}$$

3. Матричная версия алгоритма

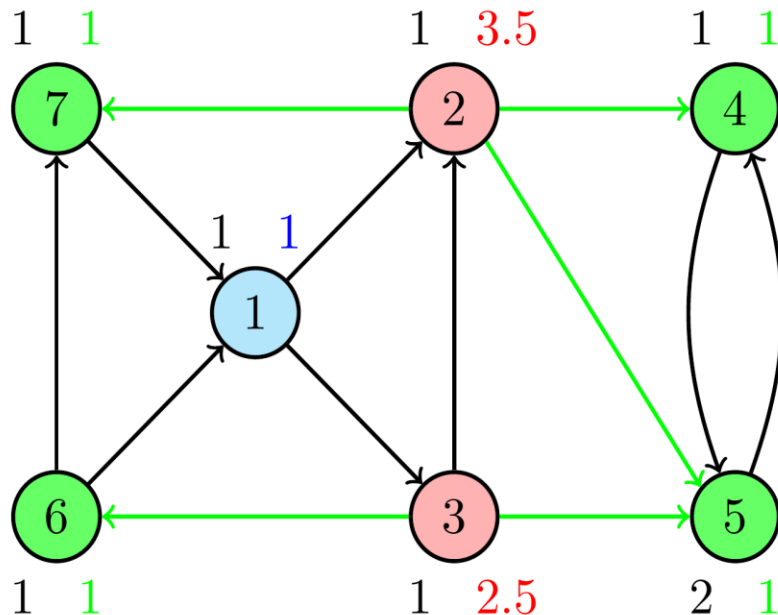
3.2. Матрично-векторный алгоритм Брандеса. Пример



$$\mathbf{b}^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \mathbf{f}^{(2)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 1 \\ 1 \end{pmatrix}, \mathbf{p} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \end{pmatrix}$$

3. Матричная версия алгоритма

3.2. Матрично-векторный алгоритм Брандеса. Пример



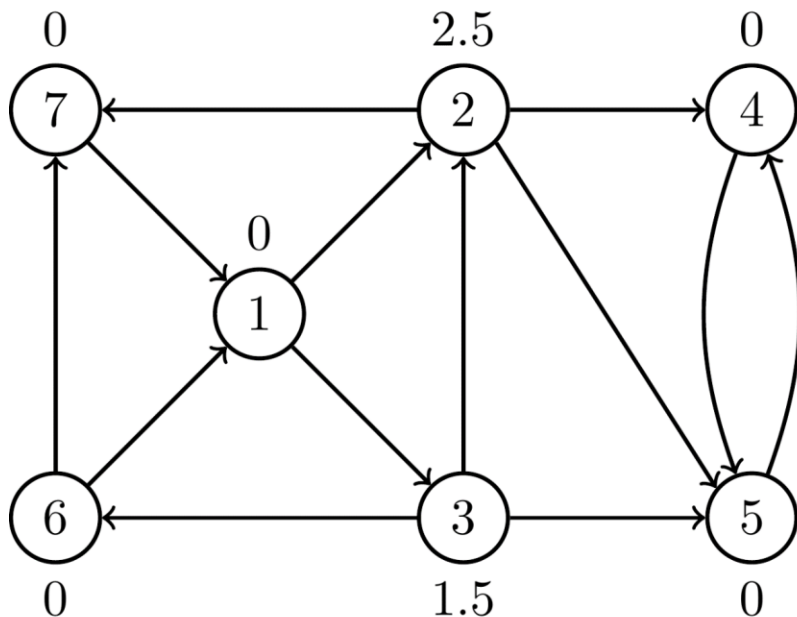
$$\mathbf{b}^{(2)} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \mathbf{f}^{(2)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 2 \\ 1 \\ 1 \end{pmatrix}, \mathbf{p} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 1 \\ 1 \end{pmatrix}$$

$$\begin{aligned} \mathbf{w}^{(2)} &= (\mathbf{b}^{(2)} ./ \mathbf{p}) \odot \mathbf{f}^{(2)} \\ &= (1 \quad 1 \quad 1 \quad 1 \quad 0.5 \quad 1 \quad 1)^T \\ &\odot (0 \quad 0 \quad 0 \quad 1 \quad 2 \quad 1 \quad 1)^T \\ &= (0 \quad 0 \quad 0 \quad 1 \quad 0.5 \quad 1 \quad 1)^T \end{aligned}$$

$$\begin{aligned} \mathbf{b}^{(1)} &= \mathbf{b}^{(2)} + (\mathbf{A}\mathbf{w}^{(2)} \odot \mathbf{f}^{(1)}) . \mathbf{p} = \\ &= (1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1)^T \\ &+ (0 \quad 2.5 \quad 1.5 \quad 0 \quad 0 \quad 0 \quad 0)^T = \\ &= (1 \quad 3.5 \quad 2.5 \quad 1 \quad 1 \quad 1 \quad 1)^T \end{aligned}$$

3. Матричная версия алгоритма

3.2. Матрично-векторный алгоритм Брандеса. Пример



$$\mathbf{b}^{(1)} = \begin{pmatrix} 1 \\ 3.5 \\ 2.5 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}, \text{ обратный ход завершён}$$

Вектор зависимостей вершин от $s = 1$:

$$\mathbf{b} = \mathbf{b}^{(1)} - \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 3.5 \\ 2.5 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 2.5 \\ 1.5 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

3. Матричная версия алгоритма

3.3. Переход к матричному алгоритму

- ❑ Приведённая матрично-векторная версия вычисляет зависимости от одной начальной вершины s
- ❑ Зависимости для различных начальных вершин не зависят друг от друга
- ❑ Поэтому можно выполнять операции сразу для нескольких начальных вершин – достаточно заменить векторы матрицами, для этого выполним конкатенацию векторов по горизонтали:

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 0 \\ 1 \\ 0 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}.$$

Такой алгоритм называется *пакетным* (*batched*)

- ❑ Появляется потенциал для эффективной реализации за счёт использования матричных операций

3. Матричная версия алгоритма

3.4. Параллельный матричный алгоритм

- ❑ В алгоритме Брандеса можно выделить два способа распараллеливания:
 1. Разным процессорам даются различные начальные вершины – подходит для систем с распределённой памятью (MPI)
 2. Параллельная реализация матричных операций – подходит для систем с общей памятью (OpenMP, TBV)
- ❑ Оба подхода независимы и могут использоваться одновременно
 - Можно распределять по узлам наборы вершин и запускать на каждом узле пакетный алгоритм с распараллеленными матричными операциями
- ❑ Количество вершин в одном наборе следует выбирать так, чтобы все матрицы помещались в оперативную память

3. Матричная версия алгоритма

3.5. Базовая реализация, псевдокод

s_j ($j = \overline{1, k}$) – набор начальных вершин **F** – текущий фронт поиска в ширину, **P** – матрица с числами кратчайших путей, S_i ($i = 1, 2, \dots$) – все фронты поиска в ширину, **Pinv** – матрица с обратными числами кратчайших путей, **B** – матрица зависимостей,

W – вспомогательная матрица; **F, P, S_i** – разреженные, **Pinv, B, W** – плотные

Прямой ход:

F := **O_{nm}**, **P** := **O_{nm}**, $d := 0$

for $j = \overline{1, k}$:

$F_{s_j s_j} := 1$

while **F** ≠ **O_{nm}**: // **O_{nm}** - нулевая матрица $n \times m$

$S_d := F$

P := SparseAdd(**P, F**) // **P** := **P** + **F**

F := MaskedSparseGEMM(**A^T, F, $\neg P$**) // **F** := $\neg P \odot (A^T F)$

$d := d + 1$

Обратный ход:

Pinv := **1_{nm}** ./ **P**, **B** := **1_{nm}** // **1_{nm}** - матрица $n \times m$, заполненная единицами

for $i = \overline{d-1, 2}$:

W := MaskedElementWiseMult(**B, Pinv, S_i**) // **W** := $S_i \odot (B \cdot Pinv)$

W := MaskedSparseDenseMM(**A, W, S_{i-1}**) // **W** := $S_{i-1} \odot AW$

ElementWiseMultAdd(**W, P, B**) // **B** := **B** + **W** · **P**

3. Матричная версия алгоритма

3.5. Базовая реализация, недостатки

s_j ($j = \overline{1, k}$) – набор начальных вершин $\mathbf{F}$ – текущий фронт поиска в ширину, $\mathbf{P}$ – матрица с числами кратчайших путей, $\mathbf{S}_i$ ($i = 1, 2, \dots$) – все фронты поиска в ширину, $\mathbf{P}^{-1}$ – матрица с обратными числами кратчайших путей, $\mathbf{B}$ – матрица зависимостей,

$\mathbf{W}$ – вспомогательная матрица; $\mathbf{F}, \mathbf{P}, \mathbf{S}_i$ – разреженные, $\mathbf{P}^{-1}, \mathbf{B}, \mathbf{W}$ – плотные

Прямой ход:

$\mathbf{F} := \mathbf{O}_{nm}, \mathbf{P} := \mathbf{O}_{nm}, d := 0$

for $j = \overline{1, k}$:

$\mathbf{F}_{s_j s_j} := 1$

while $\mathbf{F} \neq \mathbf{O}_{nm}$:

$\mathbf{S}_d := \mathbf{F}$

$\mathbf{P} := \text{SparseAdd}(\mathbf{P}, \mathbf{F}) \leftarrow$ неэффективно, когда процент заполнения $\mathbf{P}$ высокий

$\mathbf{F} := \text{MaskedSparseGEMM}(\mathbf{A}^T, \mathbf{F}, \neg \mathbf{P})$

$d := d + 1$

Обратный ход:

$\mathbf{P}^{-1} := \mathbf{1}_{nm} ./ \mathbf{P}, \mathbf{B} := \mathbf{1}_{nm}$

for $i = \overline{d-1, 2}$:

$\mathbf{W} := \text{MaskedElementWiseMult}(\mathbf{B}, \mathbf{P}^{-1}, \mathbf{S}_i)$

$\mathbf{W} := \text{MaskedSparseDenseMM}(\mathbf{A}, \mathbf{W}, \mathbf{S}_{i-1})$

$\text{ElementWiseMultAdd}(\mathbf{W}, \mathbf{P}, \mathbf{B})$

} накладные расходы при работе с $\mathbf{W}$

ОПТИМИЗАЦИЯ АЛГОРИТМА

4. Оптимизация алгоритма

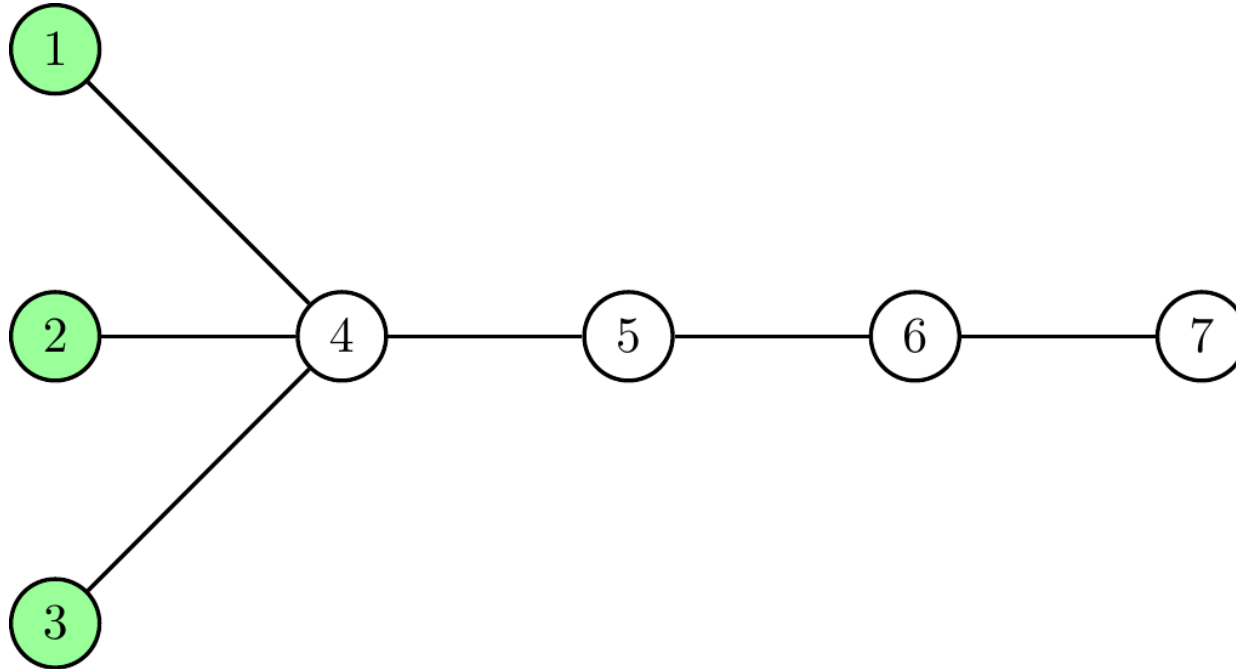
4.1. Проблема 1

- ❑ В строке $\mathbf{P} := \text{SparseAdd}(\mathbf{P}, \mathbf{F})$ выполняется сложение разреженных матриц
- ❑ Разреженная матрица хранится в формате CRS (Compressed Row Sparse), поэтому сложение напоминает слияние отсортированных массивов
- ❑ Асимптотика – $O(nz(\mathbf{P}) + nz(\mathbf{F}))$ времени и $O(nz(\mathbf{P}) + nz(\mathbf{F}))$ памяти – необходимо выделять память под сумму матриц ($nz(A)$ – число ненулевых элементов матрицы A)
- ❑ В начале прямого хода $\mathbf{P}$ пустая, в конце $\mathbf{P}$ полностью заполняется
- ❑ Сложение выполняется в цикле и суммарно может занимать много времени, когда $\mathbf{P}$ оказывается заполнена достаточно плотно в течение многих итераций

4. Оптимизация алгоритма

4.2. Проблема 1. Пример

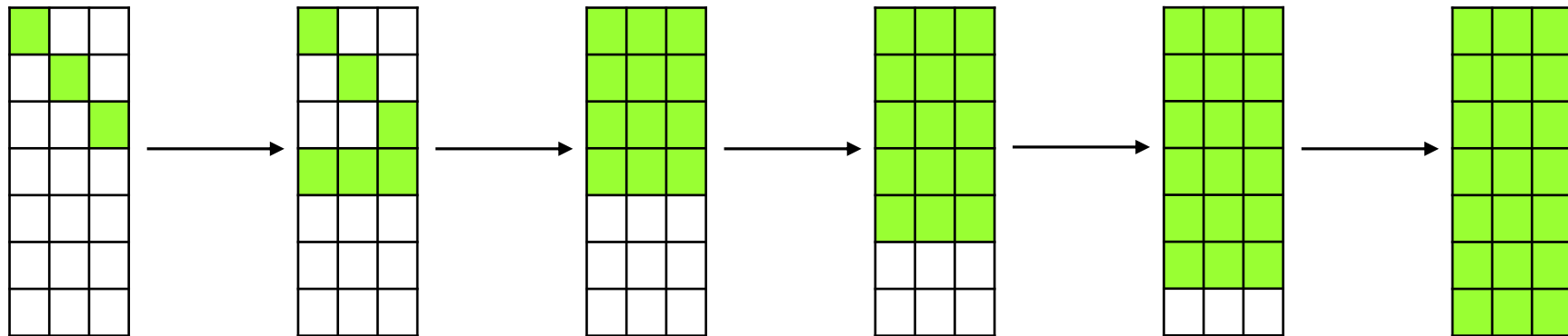
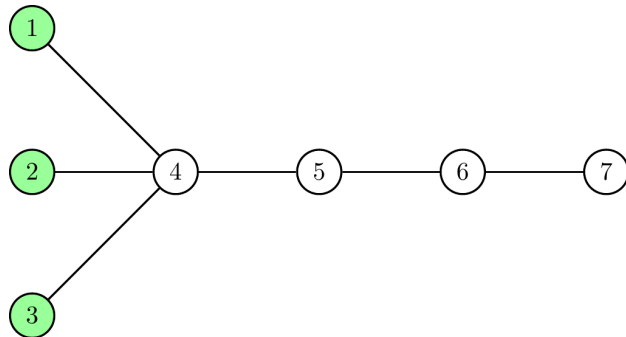
□ Рассмотрим граф и проследим, как меняется заполнение P со временем:



4. Оптимизация алгоритма

4.2. Проблема 1. Пример

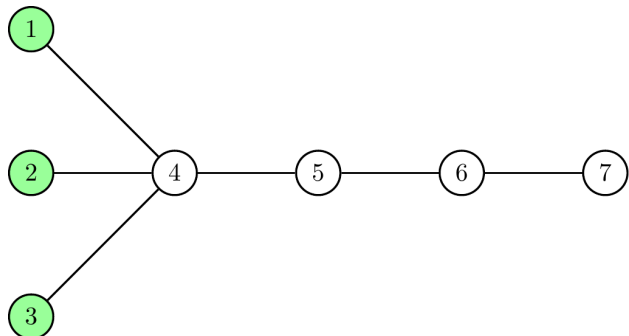
□ Рассмотрим граф и проследим, как меняется заполнение P со временем:



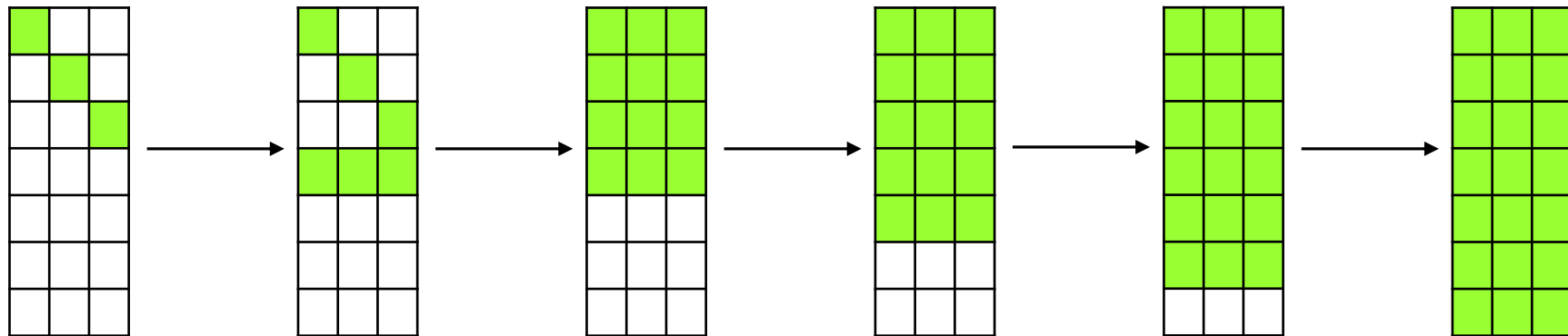
4. Оптимизация алгоритма

4.2. Проблема 1. Пример

□ Рассмотрим граф и проследим, как меняется заполнение P со временем:



Много выделений памяти и копирований!



4. Оптимизация алгоритма

4.3. Проблема 1. Решение

□ *Оптимизация 1*: сделаем матрицу $\mathbf{P}$ плотной

- Суммарное время выполнения $\mathbf{P} := \text{SparseAdd}(\mathbf{P}, \mathbf{F})$ за весь прямой ход будет составлять $O(nk)$, k – число начальных вершин, потому что каждый элемент $\mathbf{P}$ обновляется не более одного раза
- Память будет выделяться лишь один раз – перед прямым ходом

□ Заметим, что $\mathbf{P}$ является маской в $\mathbf{F} := \text{MaskedSparseGEMM}(\mathbf{A}^T, \mathbf{F}, \neg \mathbf{P})$

- Использование плотной маски вместо разреженной может замедлить умножение, поскольку нужно просмотреть больше элементов
- В случае инвертированной маски выводы неочевидны, поскольку и в разреженном случае может потребоваться просмотреть все элементы матрицы
- Эксперименты показывают, что оптимизация не ухудшает время работы прямого хода

4. Оптимизация алгоритма

4.4. Проблема 2

- В обратном ходе отдельные матричные операции влекут накладные расходы

```
MaskedElementWiseMult(A,B,M):
```

```
    DenseMatrix C;
```

```
    // заполняем C нулями
```

```
    // для всех (i,j), входящих в M: C[i][j] = A[i][j] * B[i][j]
```

```
    return C;
```

```
MaskedSparseDenseMM(A,B,M):
```

```
    DenseMatrix C;
```

```
    for (int row = 0; row < n; ++row) {
```

```
        // заполняем строку C нулями
```

```
        // считаем строку маскированного произведения A*B
```

```
        // записываем в строку C
```

```
    }
```

```
    return C;
```

```
ElementWiseMultAdd(A,B,C):
```

```
    // for (i = 0..n-1) for (j = 0..k-1)
```

```
    //     C[i][j] += A[i][j] * B[i][j]
```

4. Оптимизация алгоритма

4.4. Проблема 2

- В обратном ходе отдельные матричные операции влекут накладные расходы

```
MaskedElementWiseMult(A,B,M):
```

```
    DenseMatrix C;
```

```
    // заполняем C нулями ← каждый раз заполняется плотная матрица
```

```
    // для всех (i,j), входящих в M: C[i][j] = A[i][j] * B[i][j]
```

```
    return C;
```

```
MaskedSparseDenseMM(A,B,M):
```

```
    DenseMatrix C;
```

```
    for (int row = 0; row < n; ++row) {
```

```
        // заполняем строку C нулями
```

```
        // считаем строку маскированного произведения A*B
```

```
        // записываем в строку C
```

```
    }
```

```
    return C;
```

нулями заполняется вся плотная матрица, а в итоге ненулевых элементов достаточно мало

```
ElementWiseMultAdd(A,B,C):
```

```
    // for (i = 0..n-1) for (j = 0..k-1)
```

```
    //     C[i][j] += A[i][j] * B[i][j]
```

цикл по всем элементам, а ненулевых элементов в A достаточно мало

4. Оптимизация алгоритма

4.5. Проблема 2. Решение

- ❑ В базовой версии матричные операции реализованы как отдельные функции в отрыве от контекста их использования. Реализация корректная, но накладные расходы из-за необходимости обрабатывать матрицы целиком оказываются значительны
- ❑ *Оптимизация 2*: объединим матричные операции обратного хода в одну функцию и учтём разреженность результата на некоторых стадиях
- ❑ Оптимизация исключает накладные расходы и позволяет использовать данные более локально
- ❑ Для единообразия также объединим матричные операции прямого хода в отдельную функцию

4. Оптимизация алгоритма

4.5. Проблема 2. Решение

□ Псевдокод реализации прямого и обратного ходов:

```
ForwardStep(AT,P,F):
```

```
    // прибавляем разреженную матрицу F к плотной матрице P
    // считаем произведение AT*F с инвертированной маской P
```

```
BackwardStep(A,P,Pinv,Front,Next,B):
```

```
    // для всех (row, col), входящих в Front:
    //     Front[row][col] = B[row][col] * Pinv[row][col]
    for (int row = 0; row < n; ++row) {
        // считаем строку prod произведения A*Front с маской Next
        // для всех col, входящих в prod:
        //     B[row][col] += prod[col] * P[row][col]
    }
```

4. Оптимизация алгоритма

4.6. Оптимизированный алгоритм. Псевдокод

s_j ($j = \overline{1, k}$) – набор начальных вершин $\mathbf{F}$ – текущий фронт поиска в ширину, $\mathbf{P}$ – матрица с числами кратчайших путей, $\mathbf{S}_i$ ($i = 1, 2, \dots$) – все фронты поиска в ширину, $\mathbf{P}_{\text{inv}}$ – матрица с обратными числами кратчайших путей, $\mathbf{B}$ – матрица зависимостей; $\mathbf{F}, \mathbf{S}_i$ – разреженные, $\mathbf{P}, \mathbf{P}_{\text{inv}}, \mathbf{B}$ – плотные

Прямой ход:

$\mathbf{F} := \mathbf{O}_{nm}, \mathbf{P} := \mathbf{O}_{nm}, d := 0$

for $j = \overline{1, k}$:

$\mathbf{F}_{s_j s_j} := 1$

while $\mathbf{F} \neq \mathbf{O}_{nm}$:

$\mathbf{S}_d := \mathbf{F}$

$\text{ForwardStep}(\mathbf{A}^T, \mathbf{P}, \mathbf{F})$ // $\mathbf{P} := \mathbf{P} + \mathbf{F}; \mathbf{F} := (\neg \mathbf{P}) \odot (\mathbf{A}^T \mathbf{F})$

$d := d + 1$

Обратный ход:

$\mathbf{P}_{\text{inv}} := \mathbf{1}_{nm} \cdot / \mathbf{P}, \mathbf{B} := \mathbf{1}_{nm}$

for $i = \overline{d-1, 2}$:

$\text{BackwardStep}(\mathbf{A}, \mathbf{P}, \mathbf{P}_{\text{inv}}, \mathbf{S}_i, \mathbf{S}_{i-1}, \mathbf{B})$ // $\begin{cases} \mathbf{S}_i := \mathbf{S}_i \odot (\mathbf{B} \cdot \mathbf{P}_{\text{inv}}) \\ \mathbf{B} := \mathbf{B} + (\mathbf{S}_{i-1} \odot \mathbf{A} \mathbf{S}_i) \cdot \mathbf{P} \end{cases}$

РЕЗУЛЬТАТЫ ЭКСПЕРИМЕНТОВ

5. Результаты экспериментов

5.1. Методика проведения экспериментов

- Параметры тестового окружения (узел кластера Лобачевский):

Процессор	Два 8-ядерных процессора Intel Xeon CPU E5-2660 2.2 GHz
Память	64 GB
Операционная система	Linux CentOS 6.4
Компилятор	Intel C++ Compiler 2023

- Проводилось сравнение базовой и оптимизированной реализаций алгоритма Брандеса с эталонной реализацией от PASSIONLab (<https://github.com/PASSIONLab/MaskedSpGEMM>)

5. Результаты экспериментов

5.1. Методика проведения экспериментов

□ Тестовые матрицы

– Матрицы общего вида из коллекции SuiteSparse

Граф	Вершин	Рёбер
human_gene1	22283	24647360
bmwcra_1	148770	10641602
af_shell2	504855	17084020
packing-500x100x100-b050	2145852	34976486
Freescale1	3428755	17052626

– Матрицы, полученные генератором R-MAT, и графы Кронекера из SuiteSparse

Граф	Вершин	Рёбер
rmat18	262144	5284330
rmat19	524288	10783504
rmat20	1048576	20537868
kron_g500-logn18	262144	21165372
kron_g500-logn19	524288	43561574
kron_g500-logn20	1048576	89238804

5. Результаты экспериментов

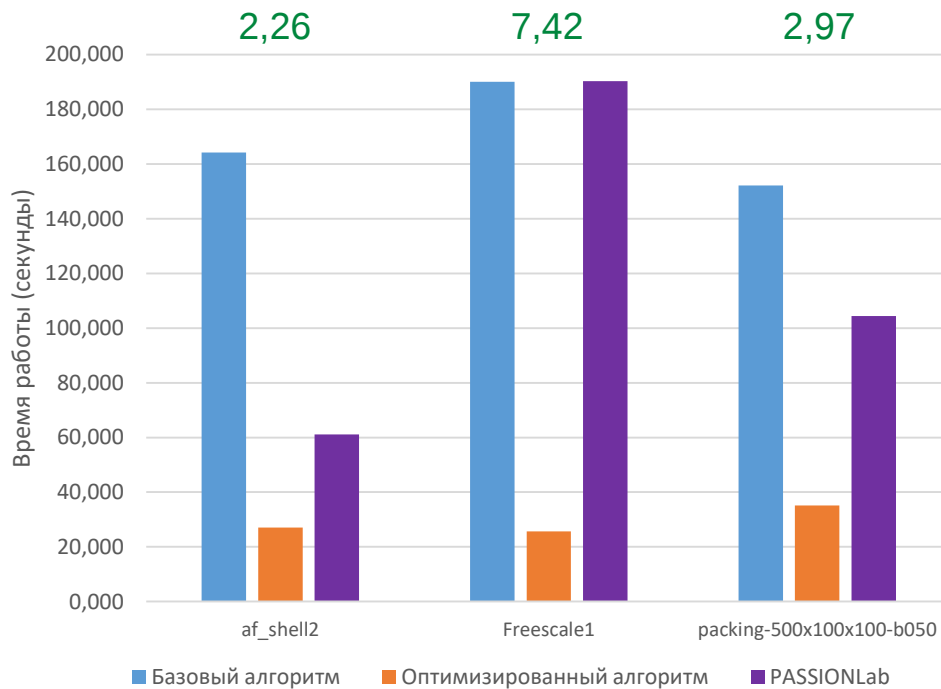
5.1. Методика проведения экспериментов

- ❑ Количество стартовых вершин в наборе составляло 128 или 256
- ❑ Сравниваемые характеристики:
 - Время работы всего алгоритма (с указанием, во сколько раз оптимизированная реализация быстрее эталонной реализации)
 - Ускорение многопоточной версии относительно однопоточной версии
 - Время работы прямого и обратного ходов

5. Результаты экспериментов

5.2. Время работы на 16 потоках

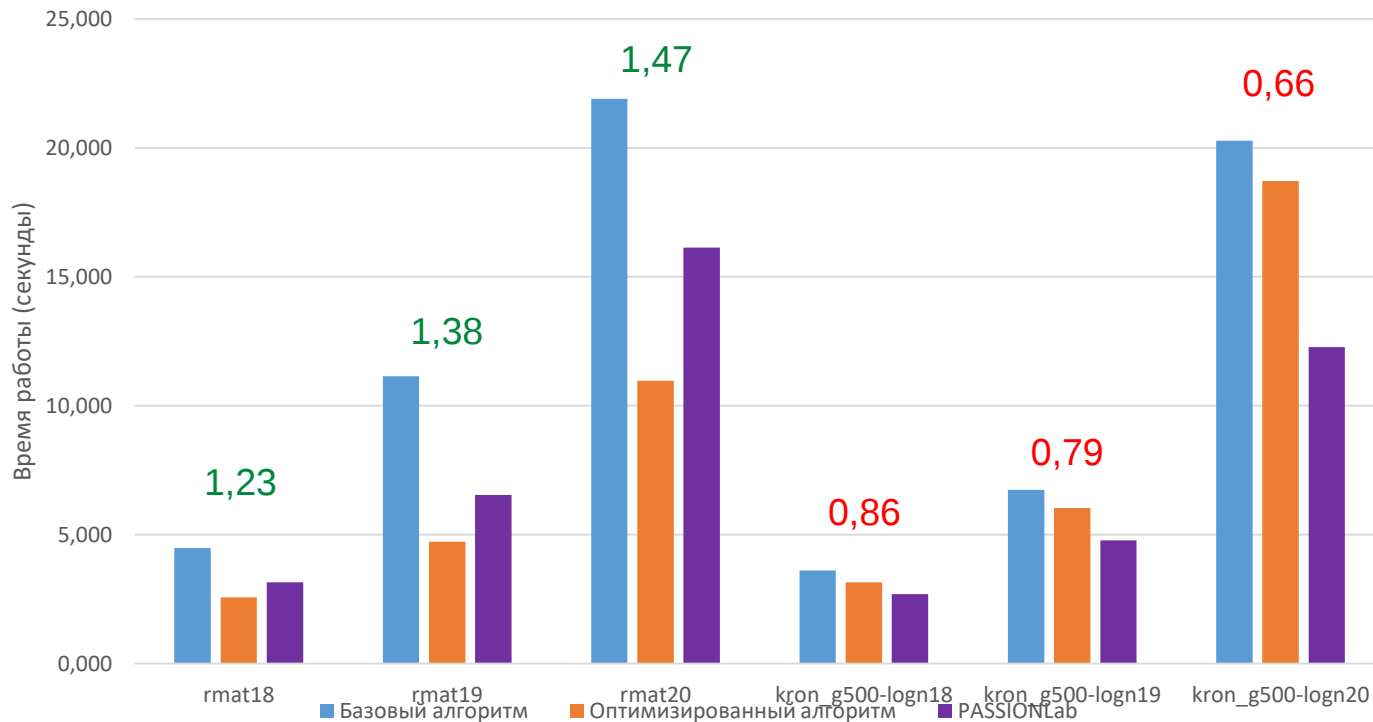
□ Матрицы общего вида:



5. Результаты экспериментов

5.2. Время работы на 16 потоках

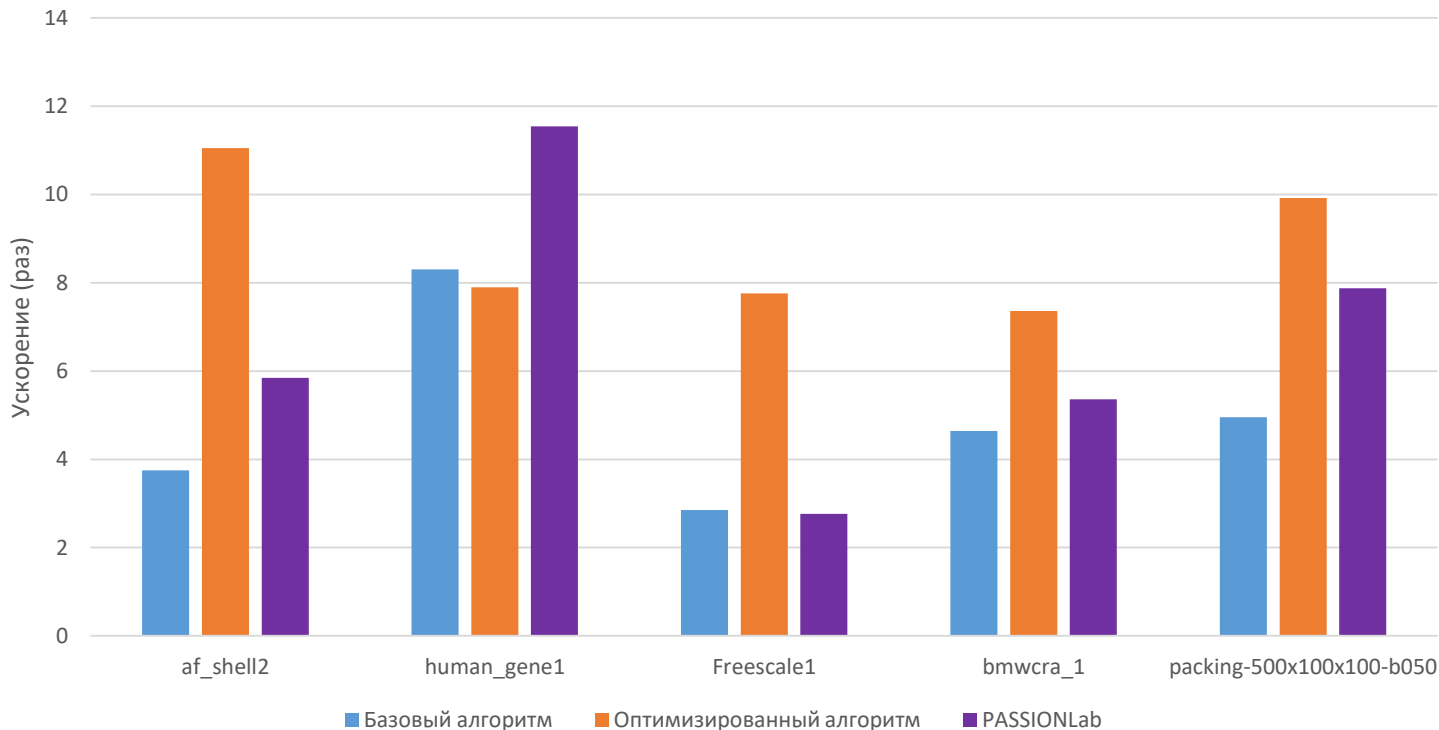
□ R-MAT и графы Кронекера:



5. Результаты экспериментов

5.3. Ускорение на 16 потоках

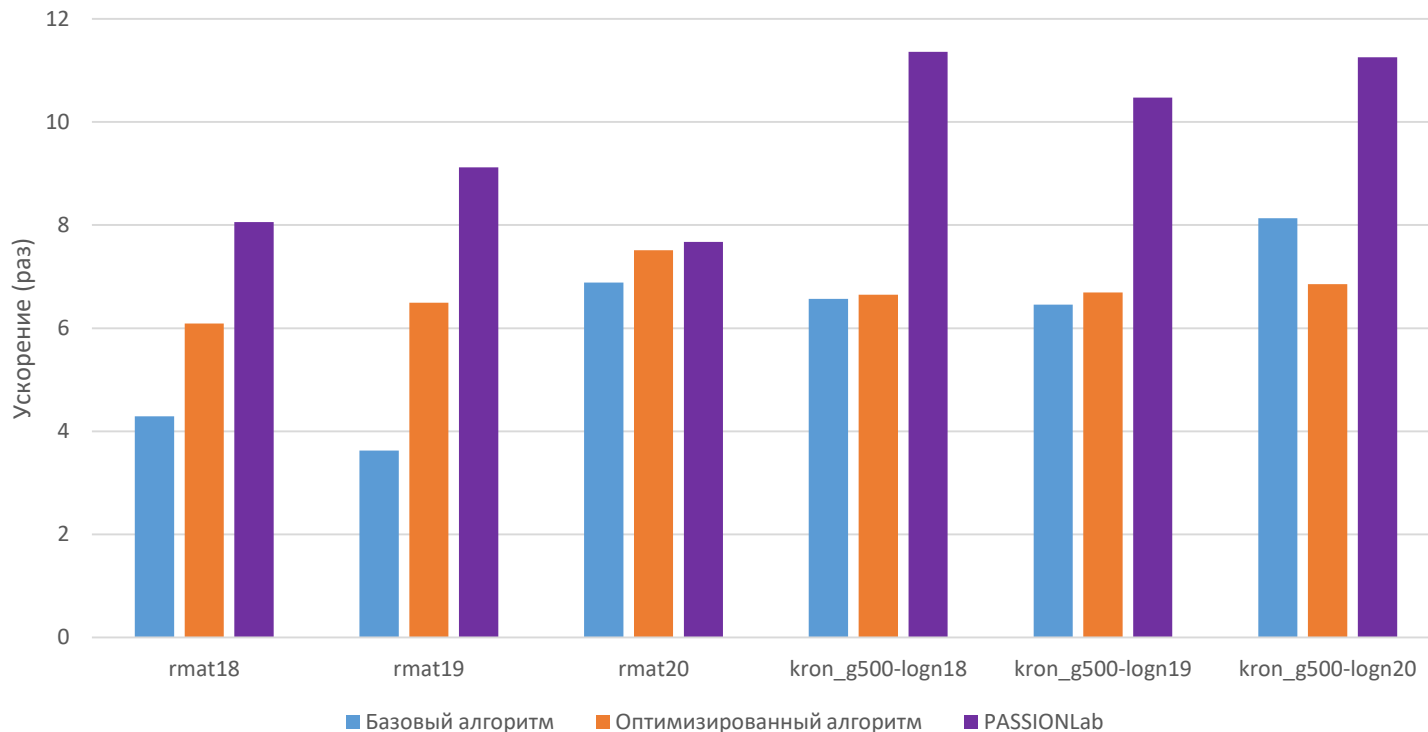
□ Матрицы общего вида:



5. Результаты экспериментов

5.3. Ускорение на 16 потоках

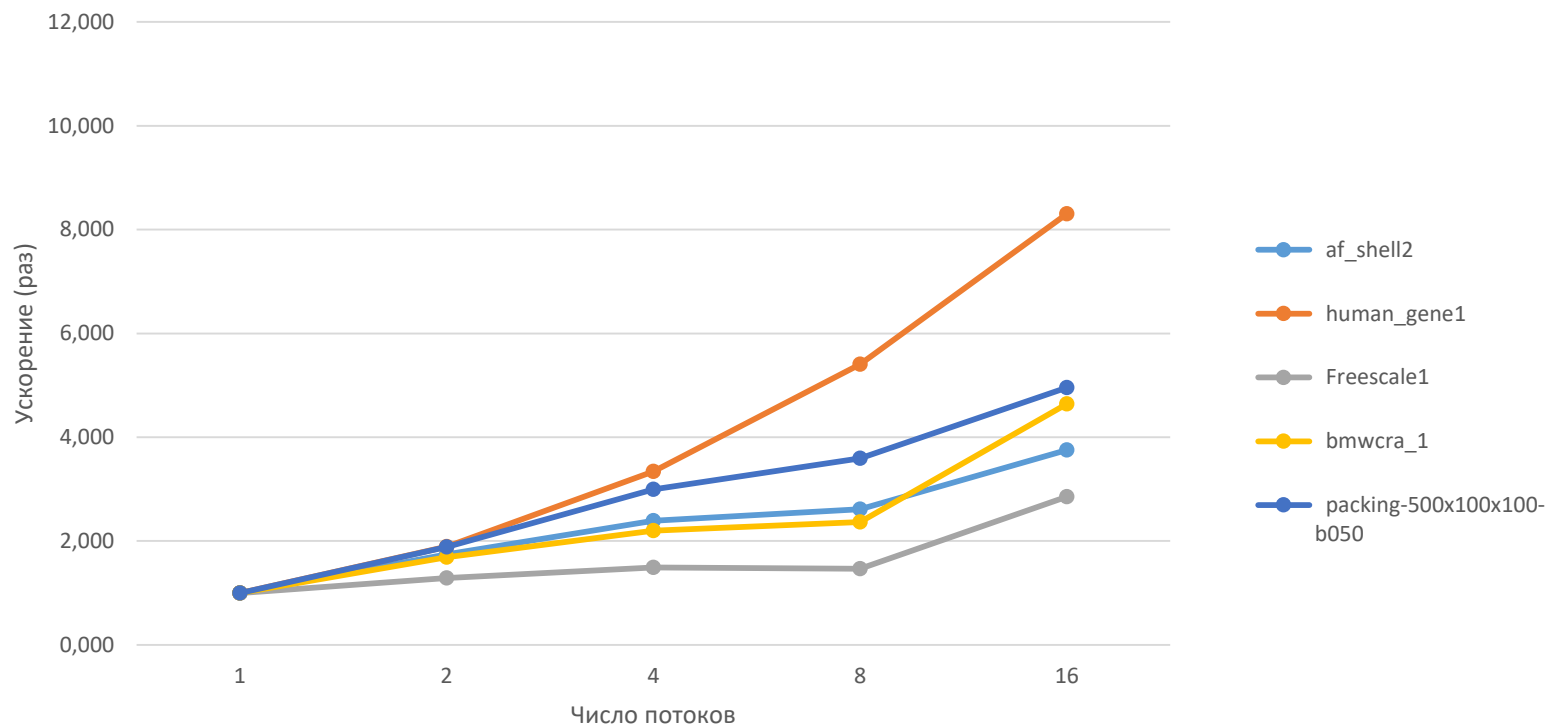
□ R-MAT и графы Кронекера:



5. Результаты экспериментов

5.3. Ускорение в зависимости от числа потоков

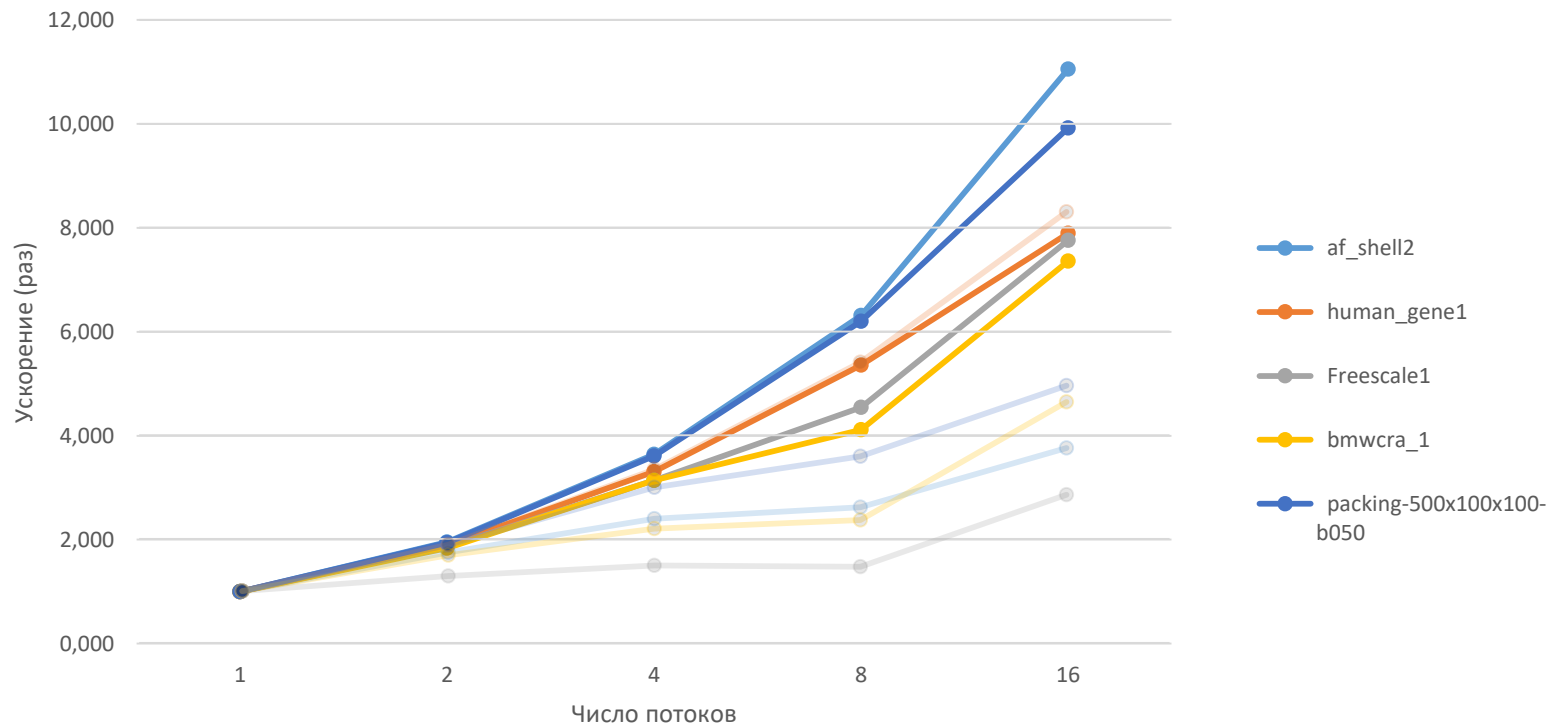
□ Матрицы общего вида, базовая версия:



5. Результаты экспериментов

5.3. Ускорение в зависимости от числа потоков

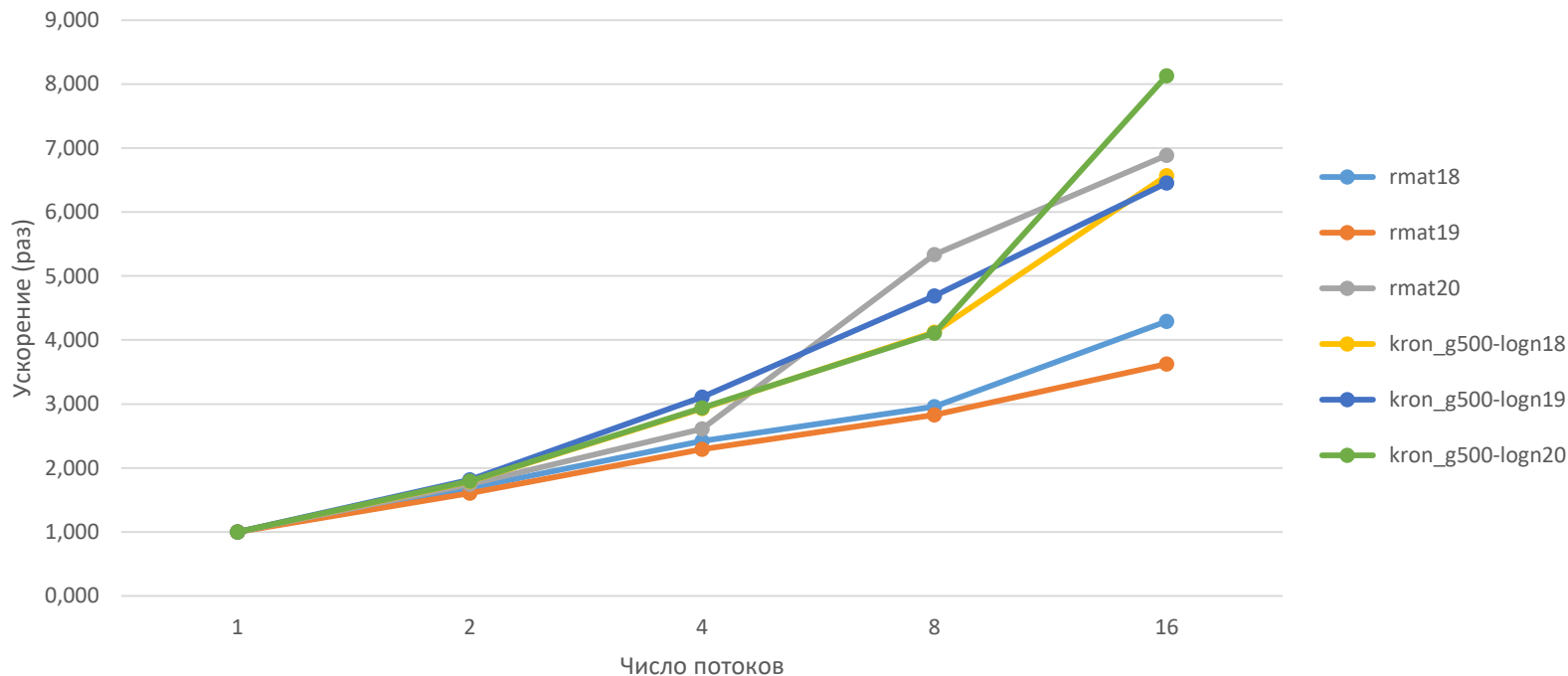
□ Матрицы общего вида, оптимизированная версия против базовой:



5. Результаты экспериментов

5.3. Ускорение в зависимости от числа потоков

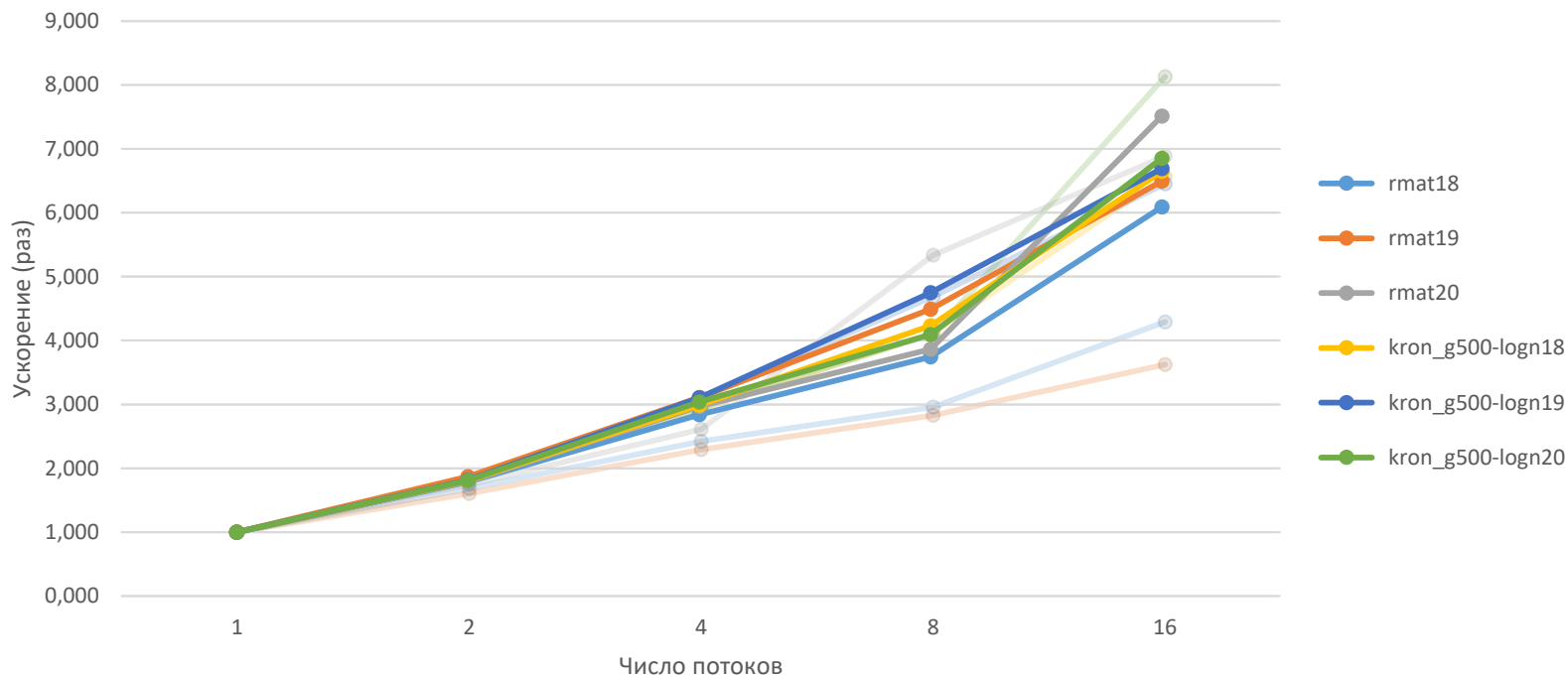
□ R-MAT и графы Кронекера, базовая версия:



5. Результаты экспериментов

5.3. Ускорение в зависимости от числа потоков

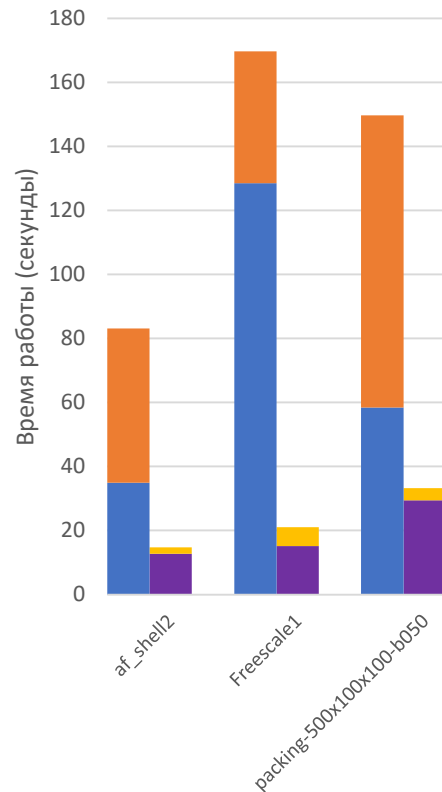
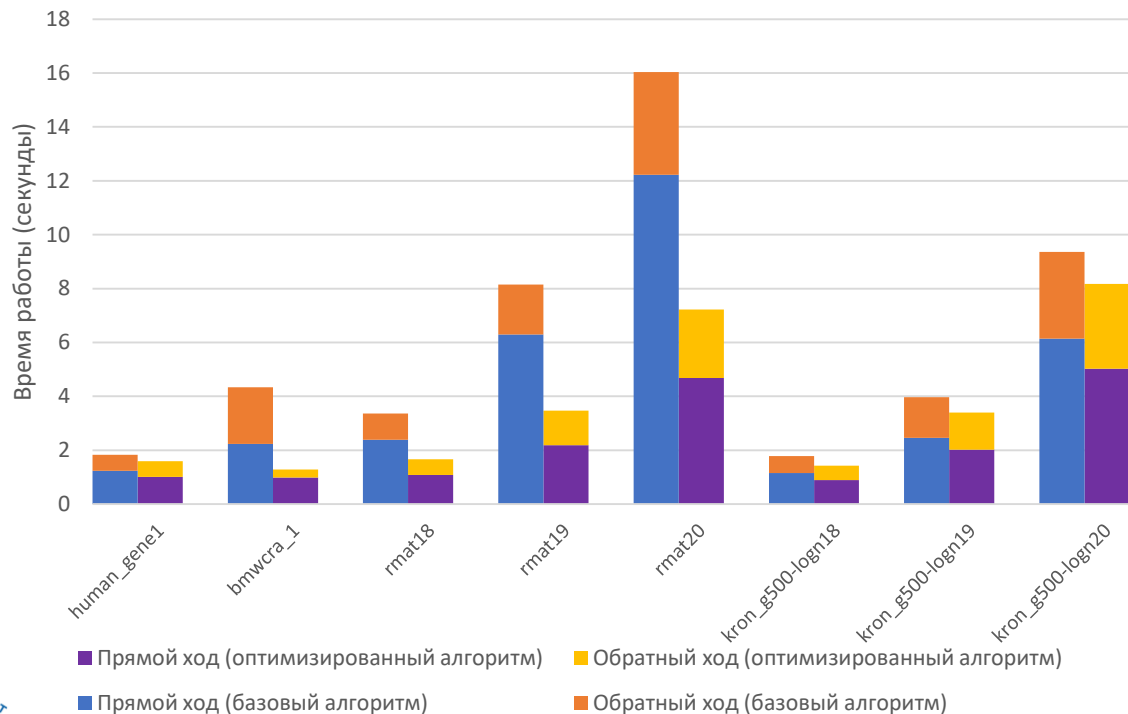
□ R-MAT и графы Кронекера, оптимизированная версия против базовой:



5. Результаты экспериментов

5.4. Прямой и обратный ходы

□ Время работы на 16 потоках:



ЗАКЛЮЧЕНИЕ

6. Заключение

- ❑ GraphBLAS является перспективной технологией для масштабируемых высокопроизводительных вычислений с графами и разреженными матрицами
- ❑ На примере алгоритма Брандеса вычисления степени посредничества вершин можно увидеть проблемы, возникающие в реализации алгоритмов на графах, и способы оптимизации вычислений
- ❑ Оптимизации, направленные на более эффективное использование данных, позволили существенно сократить время работы алгоритма и улучшить масштабируемость на многих тестовых графах

Литература

1. Kepner J. et al. Graphs, matrices, and the GraphBLAS: Seven good reasons. – Procedia Computer Science, 2015. – Т. 51.
2. Buluç A. et al. Design of the GraphBLAS API for C. – IPDPSW, 2017.
3. Kepner, Jeremy, and John Gilbert, eds. Graph algorithms in the language of linear algebra. – SIAM, 2011.
4. Brandes, Ulrik. A faster algorithm for betweenness centrality. – Journal of mathematical sociology 25.2, 2001.

Контакты

Нижегородский государственный университет

<http://www.unn.ru>

Институт информационных технологий, математики и механики

<http://www.itmm.unn.ru>

Кафедра высокопроизводительных вычислений и системного программирования

