

**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 4. Поиск кратчайшего пути в графе**

Пирова А.Ю.  
Кафедра МОСТ

# Содержание

---

- ❑ Предметные области
- ❑ Постановка задачи
- ❑ Алгоритм Дейкстры
- ❑ Алгоритм Беллмана – Форда
- ❑ Параллельный алгоритм Дейкстры
- ❑ Алгоритм дельта-шага
- ❑ Параллельный алгоритм дельта-шага
- ❑ Оптимизация алгоритма дельта-шага
- ❑ Результаты экспериментов
- ❑ Заключение

# Предметные области

---

- ❑ Поиск кратчайшего пути в различных сетях: транспортных, электрических, VLSI, графах социальных сетей
- ❑ Транспортные сети: навигация, логистическое планирование, планирование транспортных развязок и расположения объектов
- ❑ Графы социальных сетей: Betweenness centrality
- ❑ Маршрутизация в телекоммуникационных сетях

# Постановка задачи

- Пусть дан ненаправленный граф  $G = (V, E, w(e))$ ,  $|V| = n$ ,  $|E| = m$ , с неотрицательными весами ребер  $w: E \rightarrow \mathbf{R}_{\geq 0}$ .
- Вес пути  $P(v_1, v_2, \dots, v_k)$  между вершинами равен сумме весов входящих в него ребер:  $w(P) = \sum_{i=1}^{k-1} w(v_i, v_{i+1})$
- *Вес кратчайшего пути* из вершины  $s$  в вершину  $t$  определяется как

$$\delta(s, t) = \begin{cases} \min w(P), & P \text{ — путь из } s \text{ в } t \\ \infty & \end{cases}$$

- *Исток* — вершина, из которой осуществляется поиск путей.

# Постановка задачи

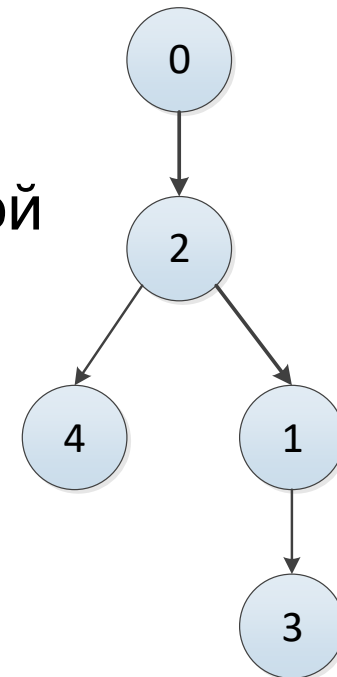
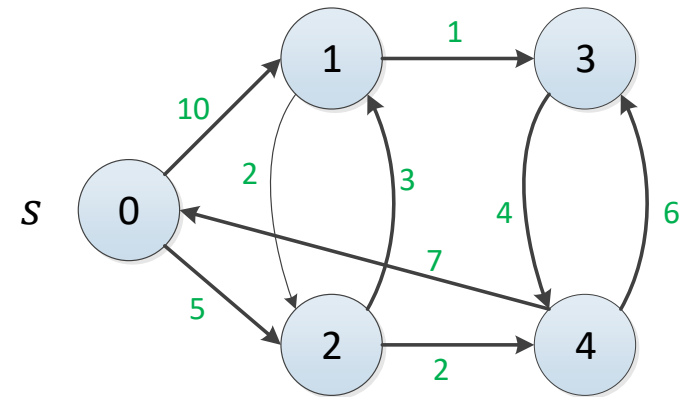
## □ Варианты постановки задачи:

- Найти кратчайший путь между двумя вершинами,  $s$  и  $t$
- Найти кратчайшие пути из вершины  $s$  во все остальные (single source shortest problem, SSSP)
- Найти кратчайшие пути между всеми вершинами (all pairs shortest paths problem, APSP)

## □ Мы рассмотрим задачу SSSP

# Постановка задачи

- Информацию обо всех кратчайших путях из одной вершины можно представить в виде дерева.
- *Дерево кратчайших путей* с корнем в вершине  $s$  — это ориентированный подграф графа  $G$ , который является корневым деревом и содержит простой кратчайший путь из истока  $s$  к каждой достижимой из нее вершине.



# Методы поиска кратчайших путей

- ❑ Подробный обзор алгоритмов
  - презентация Р. Вернека про алгоритм Дейкстры и его модификации [[pdf](#)]
  - Задача поиска пути между двумя вершинами  
Bast H. et al. Route planning in transportation networks //Algorithm engineering. – Springer, Cham, 2016. – С. 19-80. [[pdf](#)]
  
- ❑ Алгоритмы можно разделить на две группы:
  - **label setting** – после посещения вершины расстояние до нее вычислено и не будет меняться
  - **label correcting** – расстояния до вершин пересчитываются во время работы алгоритма



# Методы поиска кратчайших путей из одной вершины

Алгоритм	Сложность	Ограничения	Распараллеливаемость	Год
Алгоритм Дейкстры	$O(n \log n + m)$ при использовании фибоначчиевых куч	Неотрицатель ные веса ребер	Плохо масштабируется, требуется параллельной обработки очереди	1959
Алгоритм Беллмана-Форда	$O(nm)$	Допускает ребра с отрицательны м весом, без отриц. циклов	Просто распараллеливается, но требуется большого объема памяти	1958
Алгоритм дельта-шага (Мейерс, Сандерс)	$O(n + m + dL)$ , $d$ – максимальная степень вершины, $L$ – вес максимального кратчайшего пути	Неотрицатель ные веса ребер	распараллеливается	1998
Алгоритм Торупа	$O(n + m)$	Неотрицатель ные веса ребер	Есть параллельная версия, но требует сложных структур данных	2004

# Некоторые реализации SSSP

	Дейкстра	Беллман-Форд	дельта-шаг
igraph	+	+	--
GAP			+
			OpenMP
NetworKit	+	+	--
	seqv	algebra	
Graph500			+
			MPI
Boost			+
			MPI
GBBS	--	+	+
		omp	omp
Galois	+	+ модиф, наз. Toro	+
	seqv, omp		seqv, omp

# Алгоритм Дейкстры

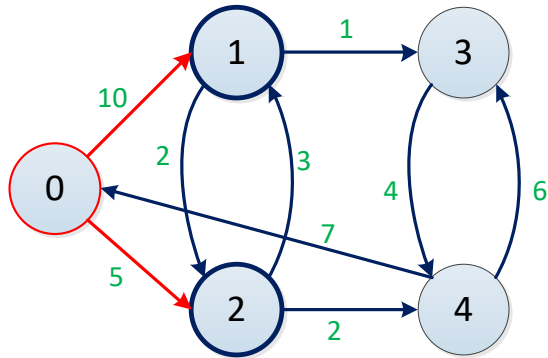
- ❑ Идея: обходим граф в ширину, на каждом шаге выбирая вершину с наименьшим расстоянием, и от нее пересчитываем оценку расстояния от источника до других вершин.  
(жадный принцип)
- ❑ Вершины разделены на 3 группы:
  - $S$  - вершины, для которых расстояние уже вычислено,
  - $Q$  – *активные вершины*, до которых расстояние конечно и еще вычисляется. Хранятся в приоритетной очереди
  - $V \setminus S \setminus Q$  – еще не достигнутые вершины (расстояние  $\infty$ )
- ❑ Обозначим  $d$  – вектор расстояний
- ❑ Основная операция – *Релаксация* – пересчет оценки расстояния до вершины

# Алгоритм Дейкстры

1. Инициализация. Для всех  $v \in V$  расстояние  $d[v] = \infty$ .
2. Положить  $d[s] = 0$ ,  $S = \emptyset$ . Вставить  $s$  в очередь  $Q$
3. До тех пор, пока очередь  $Q$  не пуста:
  1. Извлечь из  $Q$  вершину с минимальным приоритетом  $u$ .  $S = S \cup \{u\}$
  2. *Релаксация*. Для каждой вершины, смежной с  $u$ ,  $(u, z) \in E$ 
    1. Обновить оценку расстояния:  
если  $d[z] > d[u] + w(u, z)$ , то  $d[z] = d[u] + w(u, z)$
    2. Если  $u \notin S$ , вставить  $u$  в очередь  $Q$  или уменьшить ключ

- Сложность алгоритма зависит от реализации приоритетной очереди. Пусть в графе  $n$  вершин,  $m$  ребер, тогда:
- Список или массив  $O(n^2)$
  - Сбалансированное бинарное дерево поиска  $O((n + m) \log n)$
  - Фибоначчиева куча  $O(n + m \log n)$

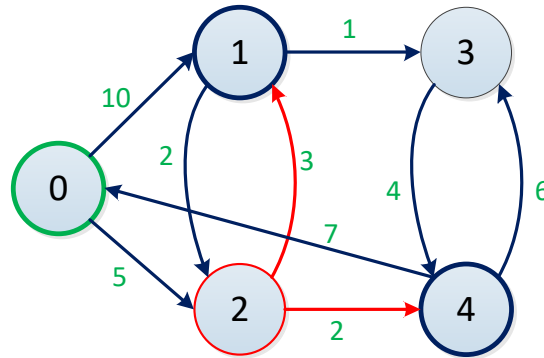
# Алгоритм Дейкстры. Пример



$d$ 

0	$\infty$	$\infty$	$\infty$	$\infty$
---	----------	----------	----------	----------

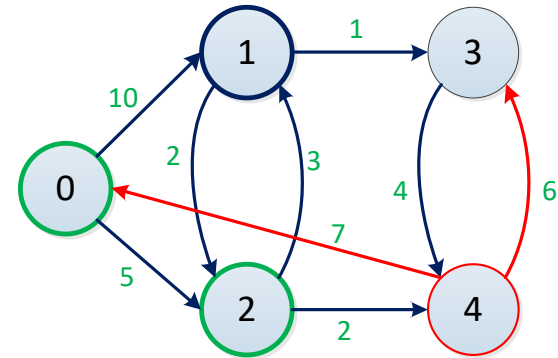
$S = \emptyset$   $Q = \{0\}$



$d$ 

0	10	5	$\infty$	$\infty$
---	----	---	----------	----------

$S = \{0\}$   $Q = \{(2, 5), (1, 10)\}$

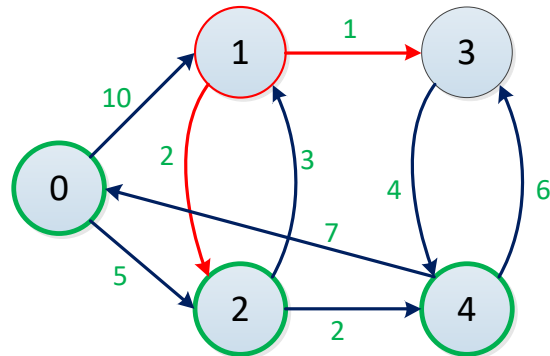


$d$ 

0	8	5	$\infty$	7
---	---	---	----------	---

$S = \{0, 2\}$   $Q = \{(4, 7), (1, 8)\}$

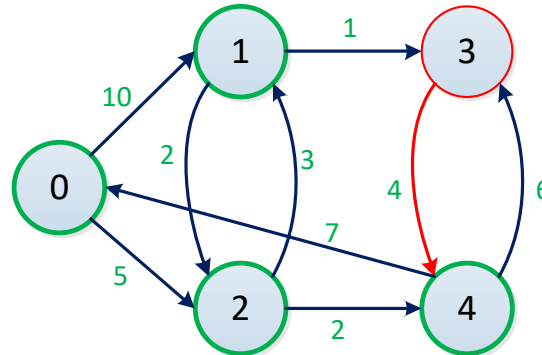
ИЗМЕНИЛСЯ КЛЮЧ



$d$ 

0	8	5	13	7
---	---	---	----	---

$S = \{0, 2, 4\}$   $Q = \{(1, 8), (3, 13)\}$



$d$ 

0	8	5	9	7
---	---	---	---	---

$S = \{0, 1, 2, 4\}$   $Q = \{(3, 9)\}$

# Параллельный алгоритм Дейкстры

- Рассмотрим алгоритм для направленного графа авторов Краузер, Мельхорн, Мейер, Сандерс, 1998 г.

Сложность  $O(n^{\frac{1}{3}} \log n)$

- Обозначения. :

- $i$  – индекс процесса (потока).  $P$  – число процессов (потоков)
- $V_i$  - множество вершин, локальных для процесса (потока)
- Очередь  $Q_i^d$ . Приоритет вершины  $v$  – оценка расстояния  $d[v]$ .
- Очередь  $Q_i^{in}$ . Приоритет вершины  $v$  – оценка расстояния по входящим ребрам,  $in[v] = d[v] - \min\{w(u, v) : (v, u) \in E, v \in V_i\}$
- Очередь  $Q_i^{out}$ . Приоритет вершины  $v$  – оценка расстояния по исходящим ребрам,  $out[v] = d[v] + \min\{w(u, v) : (v, u) \in E, v \in V_i\}$
- $S$  – Множество обработанных вершин, для которых расстояние уже вычислено

# Параллельный алгоритм Дейкстры

1. Инициализация. Для всех  $v \in V_i$ : расстояние от истока  $d[v] = \infty$ .
2. Положить  $d[s] = 0$ ,  $S = \emptyset$ . Добавить  $s$  в очереди  $Q_i^d$ ,  $Q_i^{in}$ ,  $Q_i^{out}$ .
3. До тех пор, пока очереди  $Q_i^d$  не пусты:
  1. Извлечь из  $Q_i^{out}$  вершину с минимальным приоритетом, равным  $L_i$ . Выполнить **редукцию**: найти  $L = \min\{L_i, i = \overline{1, P}\}$ .
  2. Извлечь из  $Q_i^d$  вершину с минимальным приоритетом, равным  $M_i$ . Выполнить **редукцию**: найти  $M = \min\{M_i, i = \overline{1, P}\}$ .
  3. На каждом процессе: найти множество вершин  $R_i$ , для которых  $d[v] \leq L$  или  $in[v] \leq M$ . Для этих вершин расстояние уже найдено и равно  $d[v]$ .  $S = S \cup \bigcup_{i=1}^P R_i$
  4. Удалить вершины из множества  $R_i$  из очередей  $Q_i^d$ ,  $Q_i^{in}$ ,  $Q_i^{out}$ .
  5. На каждом процессе: сформировать множество пар  $\langle Z_i, X_i \rangle$  необработанных вершин, смежных с вершинами из  $R_i$ , и весов ведущих к ним ребер.
  6. Для каждой пары  $\langle z, x \rangle \in \langle Z_i, X_i \rangle$  обновить приоритет вершины  $z$  в очередях своего или чужого процесса (потока)  $Q_i^d$ ,  $Q_i^{in}$ ,  $Q_i^{out}$  по правилу: если  $d[z] > d[z] + x$ , то  $d[z] = d[z] + x$ .
7. Синхронизация: проверить критерий останова

# Алгоритм Беллмана–Форда

- ❑ Алгоритм типа label correcting, основан на принципе динамического программирования.
- ❑ На каждой итерации выполняется релаксация по всем ребрам.

1. Инициализация. Для всех  $v \in V$  расстояние  $d[v] = \infty$ .
2. Повторить  $n - 1$  раз:
  1. Для всех ребер  $(v, u) \in E$ 
$$d(v) = \min\{d(v), d(u) + w(u, v)\}$$
3. Проверка на наличие цикла с отрицательным весом:
  1. Для всех ребер  $(v, u) \in E$ :

Если  $d(u) + w(u, v) < d(v)$ , то ЕСТЬ ЦИКЛ

❑ Вычислительная сложность  $O(nt)$



# Алгоритм Беллмана–Форда

□ Модификация: добавим проверку сходимости алгоритма.

1. Инициализация. Для всех  $v \in V$  расстояние  $d[v] = \infty$ .  $d[s] = 0$
2. Повторить  $n - 1$  раз:
  1. Для всех ребер  $(v, u) \in E$   
 $d(v) = \min\{d(v), d(u) + w(u, v)\}$
  2. Проверить, изменялся ли массив  $d$ . Если нет, то BREAK.
3. Проверка на наличие цикла с отрицательным весом:
  1. Для всех ребер  $(v, u) \in E$ :  
Если  $d(u) + w(u, v) < d(v)$ , то ЕСТЬ ЦИКЛ

□ Вычислительная сложность  $O(nt)$

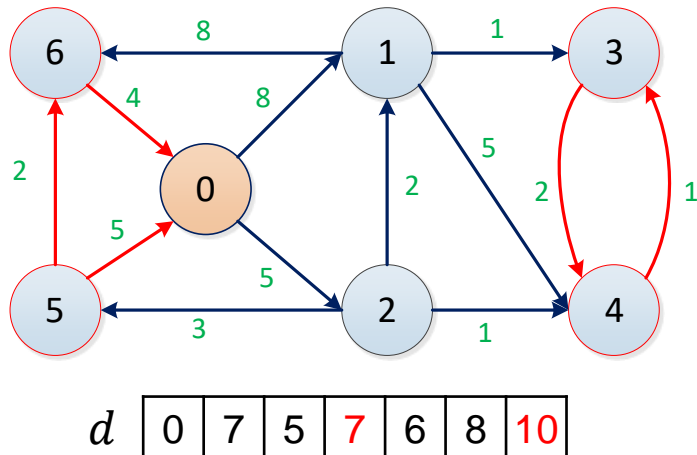
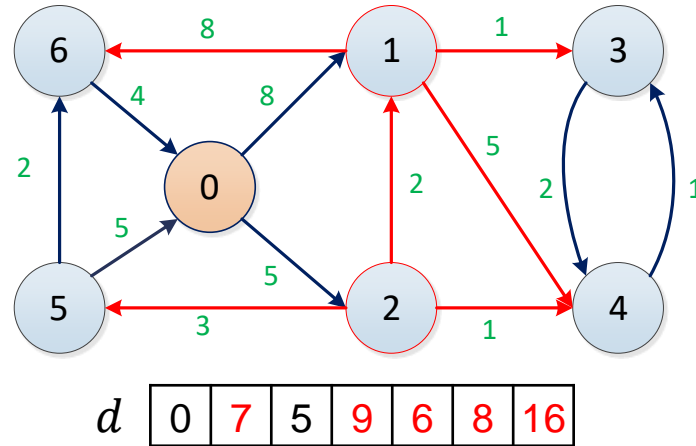
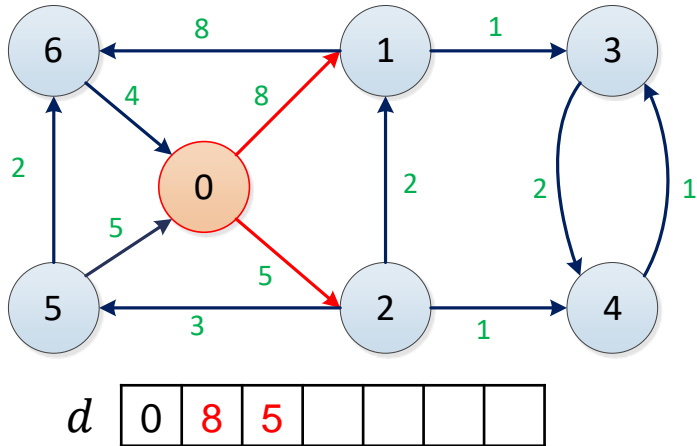
# Алгоритм Беллмана–Форда–Мура (или Shortest Path Faster Algorithm, SPFA)

- Мур, 1959.
- Модификация: используем очередь вершин, для которых расстояние изменялось на предыдущей итерации.

1. Инициализация. Для всех  $v \in V$  расстояние  $d[v] = \infty$ .
2. Положить  $d[s] = 0$ . Множество активных вершин  $Q = \{s\}$
3. До тех пор, пока множество  $Q$  не пусто:
  1. Для каждой вершины  $u$  из  $Q$ :
    1. Для каждого ребра  $(u, z) \in E$ :
      1. Вычислить  $d_{new}[z] = d[u] + w(u, z)$ ;
      2. *Релаксация*. Если  $d_{new}[z] < d[z]$ , то  $d[z] = d_{new}[z]$ ,  $Q = Q \cup \{z\}$ .
    2.  $Q = Q \setminus \{u\}$ .

□ Вычислительная сложность  $O(nm)$

# Алгоритм Беллмана–Форда. Пример



# Алгоритм дельта-шага (delta stepping)

- ❑ Идея: модифицируем алгоритм Дейкстры так, чтобы выделить группы вершин, для которых расстояние можно пересчитать одновременно.
- ❑ Выберем порог  $\Delta > 0$ . Все ребра по весу можно разделить на «легкие» ( $\{(v, u) \in E : w(v, u) < \Delta\}$ ) и «тяжелые»  $\{(v, u) \in E : w(v, u) \leq \Delta\}$ .
- ❑ Для хранения оценки расстояния используем массив карманов (buckets), в которых вершины сгруппированы по текущей оценке расстояния.
- ❑ В кармане  $B[i]$  хранятся вершины  $v$  из очереди, для которых оценка расстояния  $d[v] \in [i\Delta, (i + 1)\Delta)$ .  
Величина  $\Delta$  называется *шириной кармана*.

# Алгоритм дельта-шага (delta stepping)

- ❑ Если вершины связаны «легким» ребром, то при пересчете расстояния они окажутся в одном кармане. Если вершины связаны «тяжелым» ребром, то они окажутся в разных карманах.
- ❑ Одновременно можно пересчитать расстояние по всем «легким» ребрам, исходящим из вершин одного кармана. В результате этой операции в текущий карман могут попасть новые вершины или вершины, которые уже были в нем, но с улучшенной оценкой расстояния.

# Алгоритм дельта-шага (delta stepping)

1. *Предобработка.* Для всех  $v \in V$ : разделить список смежности вершины по весу ребер:  $L[v]$  – все ребра с весом меньше  $\Delta$ ,  $H[v]$  – все ребра с весом больше  $\Delta$ .
2. *Инициализация.* Для всех  $v \in V$ :  $d[v] = \infty$ .  $B[0] = \{s\}$ .  $i = 0$ .
3. Пока все карманы  $B$  не пусты:
  1. Множество рассмотренных вершин  $S = \emptyset$
  2. Для каждого кармана  $B[i]$  выполнять, пока карман не пуст:
    1. Сформировать множество пар  $Req$ , состоящее из вершин, соединенных с вершинами из  $B[i]$  **легким ребром**, и новой оценкой расстояния для них:  
 $Req = \{(u, x): v \in B[i], (v, u) \in L(v), x = d[v] + w(v, u)\}$ .
    2.  $S = S \cup B[i]$ ;  $B[i] = \emptyset$
    3. *Релаксация.* Для каждого  $(u, x) \in Req$  выполнить **Relax(u, x)**
  3. Сформировать множество пар вершин, соединенных с вершинами из  $S$  **тяжелым ребром**, и новой оценки расстояния для них:  
 $Req = \{(u, x): v \in B[i], (v, u) \in H(v), x = d[v] + w(v, u)\}$
  4. *Релаксация.* Для каждого  $(u, x) \in Req$  выполнить **Relax(u, x)**

# Алгоритм дельта-шага (delta stepping)

Релаксация **Relax**( $u$ ,  $x$ ):

Если  $d[u] > x$ , то

1.  $d[u] = x$ ,
2. исключить  $u$  из кармана  $B[d[u] / \Delta]$ ,
3. вставить  $u$  в карман  $B[x / \Delta]$ .

# Алгоритм дельта-шага (delta stepping)

- Эффективность алгоритма зависит от выбора  $\Delta$  и реализации системы карманов.
  - Каждый карман – список или динамический массив, поддерживающий изменение размера. Должны поддерживаться вставка и удаление элемента за  $O(1)$
- Мейер и Сандерс доказали, что для случайных графов с весами, равномерно распределенными в  $[0, 1]$ , при  $\Delta = \Theta(\frac{1}{d})$  время работы в среднем  $O(n + m + dL)$ , где  $L$  – длина максимального пути,  $d$  – максимальная степень вершины



# Алгоритм дельта-шага. Пример

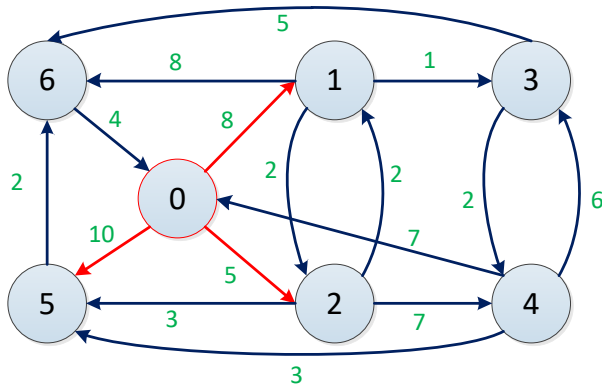
$$\Delta = 4$$

$$B[0] = \{v: d[v] \in [0,4)\}$$

$$B[1] = \{v: d[v] \in [4,8)\}$$

$$B[2] = \{v: d[v] \in [8,12)\}$$

Текущий карман –  $B[0]$

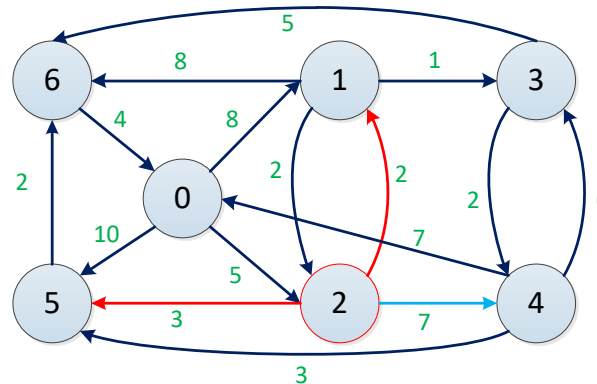


Шаг 1.  $B[0] = \{0\}$ ,  $B[1] = B[2] = \{\}$   
Тяжелые ребра из кармана  $B[0]$

$$d \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & \infty & \infty & \infty & \infty & \infty & \infty \\ \hline \end{array}$$

$$Req = \{(1, 8), (2, 5), (5, 10)\}$$

Текущий карман –  $B[1]$

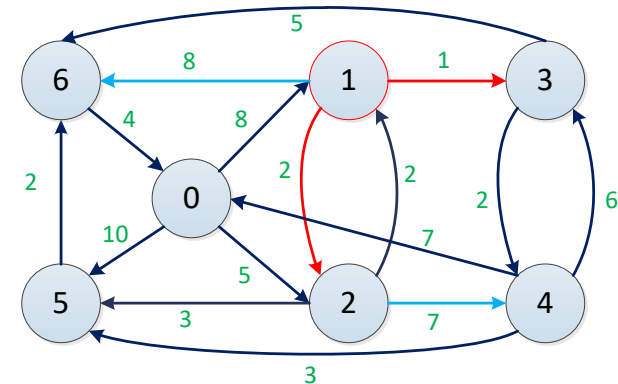


Шаг 2.1.  $B[0] = \{\}$ ,  $B[1] = 2$ ,  $B[2] = \{1, 5\}$   
Легкие ребра из кармана  $B[1]$

$$d \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 8 & 5 & \infty & \infty & 10 & \infty \\ \hline \end{array}$$

$$Req = \{(1, 7), (5, 8)\}$$

Текущий карман –  $B[1]$



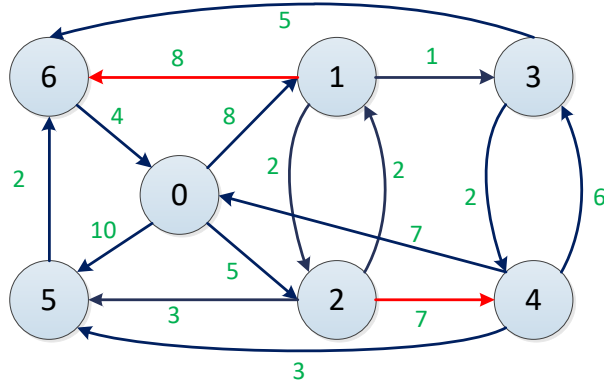
Шаг 2.2.  $B[0] = \{\}$ ,  $B[1] = \{1\}$ ,  $B[2] = \{5\}$   
Легкие ребра из кармана  $B[1]$

$$d \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 7 & 5 & \infty & \infty & 8 & \infty \\ \hline \end{array}$$

карман 1 не пуст  $\rightarrow$  продолжаем  
 $Req = \{(2, 9), (3, 8)\}$

# Алгоритм дельта-шага. Пример

Текущий карман – B[1]

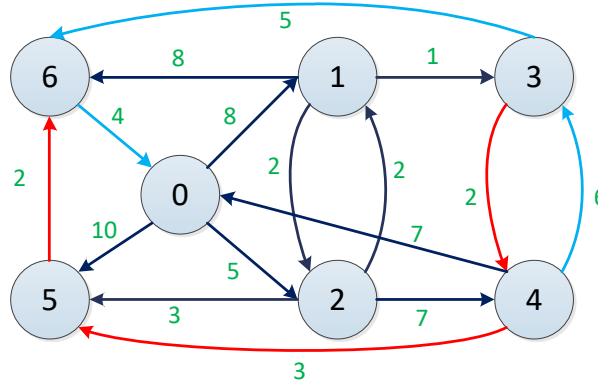


Шаг 2.3. Тяжелые ребра из кармана B[1]

$$d \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 7 & 5 & 8 & \infty & 8 & \infty \\ \hline \end{array}$$

Req = {(4, 12), (6, 15)}

Текущий карман – B[2]



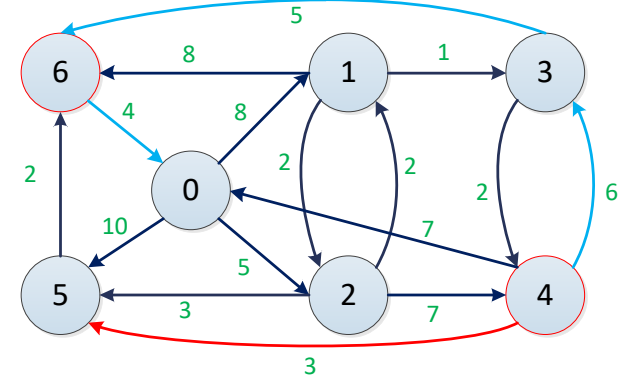
Шаг 2.4. B[0] = {}, B[1] = {}, B[2] = {3, 4, 5, 6}.

Легкие ребра из кармана B[2]

$$d \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 7 & 5 & 8 & 12 & 8 & 15 \\ \hline \end{array}$$

Req = {(4, 10), (5, 15), (6, 10)}

Текущий карман – B[2]



Шаг 2.4. B[0] = {}, B[1] = {}, B[2] = {4, 6}

Легкие ребра из кармана B[2]

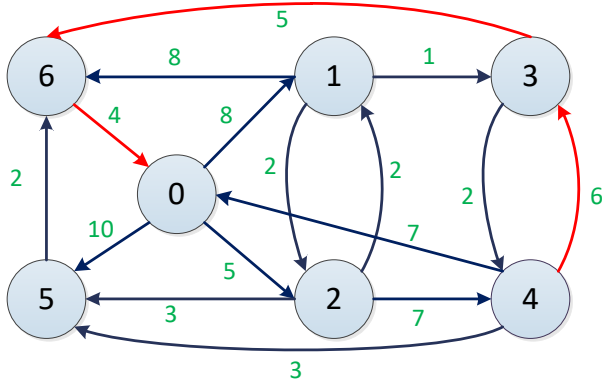
$$d \begin{array}{|c|c|c|c|c|c|c|} \hline 0 & 7 & 5 & 8 & 10 & 8 & 10 \\ \hline \end{array}$$

Req = {(5, 13)}

до вершин 4 и 6 изменялось расстояние, поэтому они активные

# Алгоритм дельта-шага. Пример

Текущий карман –  $B[2]$



Шаг 2.5.  $B[0] = \{\}$ ,  $B[1] = \{\}$ ,  $B[2] = \{\}$

Тяжелые ребра из кармана  $B[2]$

$d$	0	7	5	8	10	8	10
-----	---	---	---	---	----	---	----

$Req = \{(3, 16), (6, 13), (0, 14)\}$

# Параллельный алгоритм дельта-шага

- Параллельно выполняем обработку ребер из одного кармана (шаги 3.2, 3.4).
  - Подход эффективен, если число карманов мало, а множество запросов велико.
  - Синхронизация: процессы (потoki) обмениваются подмножествами запросов Req так, что каждый обрабатывает запросы для своих локальных вершин. В Req удаляются дубли на этапе формирования множества
  - Дополнительная синхронизация: определить следующий непустой карман, проверить критерий останова.

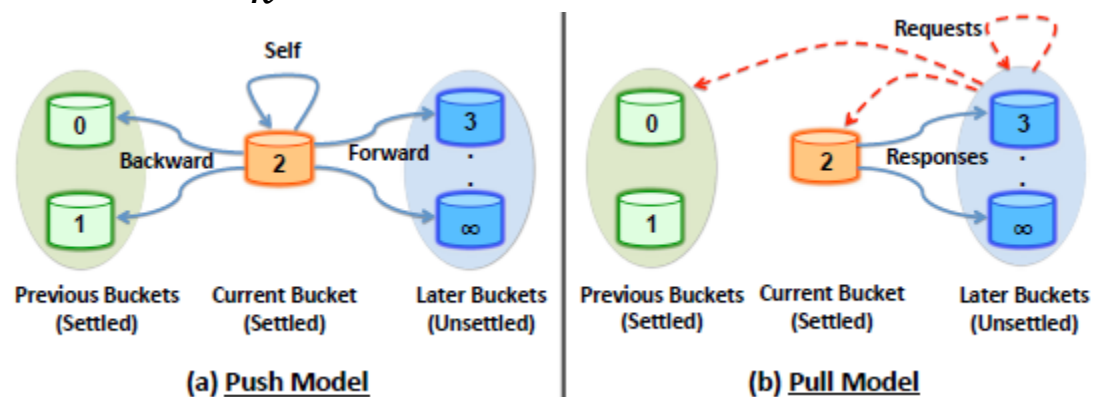
# Параллельный алгоритм дельта-шага

1. *Предобработка.* Параллельно для всех  $v \in V$ : сформировать  $L[v]$ ,  $H[v]$ .
2. *Инициализация.* Параллельно для всех  $v \in V$ :  $d[v] = \infty$ .  $B[0] = \{s\}$ .
3. Пока все карманы  $B$  не пусты:
  1. На каждом процессе (потоке)  $k$  множество рассмотренных вершин  $S[k] = \emptyset$
  2. Для каждого кармана  $B[i]$  выполнять, пока карман не пуст:
    1. Параллельно На каждом процессе (потоке)  $k$  для локальных вершин из  $B[i]$  сформировать множество пар  $Req$  без дублей:  $Req[k]$ .
    2.  $S[k] = S[k] \cup B[i]$ ;  $B[i] = \emptyset$
    3. *Релаксация.* Обменяться множествами  $Req[k]$  так, чтобы на каждом процессе (потоке) обработать  $Req$  для локальных вершин. Выполнить релаксацию
  3. Параллельно На каждом процессе (потоке)  $k$  сформировать множество запросов  $Req[k]$  из вершин, соединенных с вершинами из  $S[k]$  тяжелым ребром
  4. *Релаксация.* Обменяться множествами  $Req[k]$  так, чтобы на каждом процессе (потоке) обработать  $Req$  для локальных вершин. Выполнить релаксацию
  5. *Синхронизация:* найти следующий непустой карман

# Оптимизация алгоритма дельта-шага (Chakaravarthy и др., 2014)

## □ Оптимизация 1 (Prune):

- Цель – сократить число релаксаций по тяжелым ребрам
- Исключим релаксации по ребрам, которые ведут в карманы с меньшим номером (для вершин из таких карманов расстояние уже найдено).
- На распределенной памяти релаксация включает отправку запроса и отправку ответа. Для ребра  $e = (u, v)$  запрос отправляется, если  $d[v] > k\Delta + w(e)$ . Процесс-владелец  $u$  отправляет ответ только если  $u \in B_k$ .



# Оптимизация алгоритма дельта-шага (Chakaravarthy и др., 2014)

## □ Оптимизация 2:

- Цель – сократить число релаксаций по легким ребрам
- Для вершины  $u$  из текущего кармана  $B_k$  релаксация ребра  $e = (u, v)$  выполняется, только если  $v$  попадет в тот же карман:  $d_{new}(v) = d(u) + w(e) < (k + 1)\Delta$ . Иначе релаксация ребра выполняется вместе с тяжелыми ребрами (шаг 3.3)

## □ Оптимизация 3: гибридизация

- Цель – уменьшить число внешних итераций алгоритма
- Алгоритмом дельта-шага выполняется обработка первых  $l$  карманов. Остальные карманы объединяются в один и обрабатываются алгоритмом Беллмана–Форда.
- Показано, что переключение целесообразно выполнить после обработки 40% вершин

# Результаты экспериментов.

## Сравнение разных алгоритмов

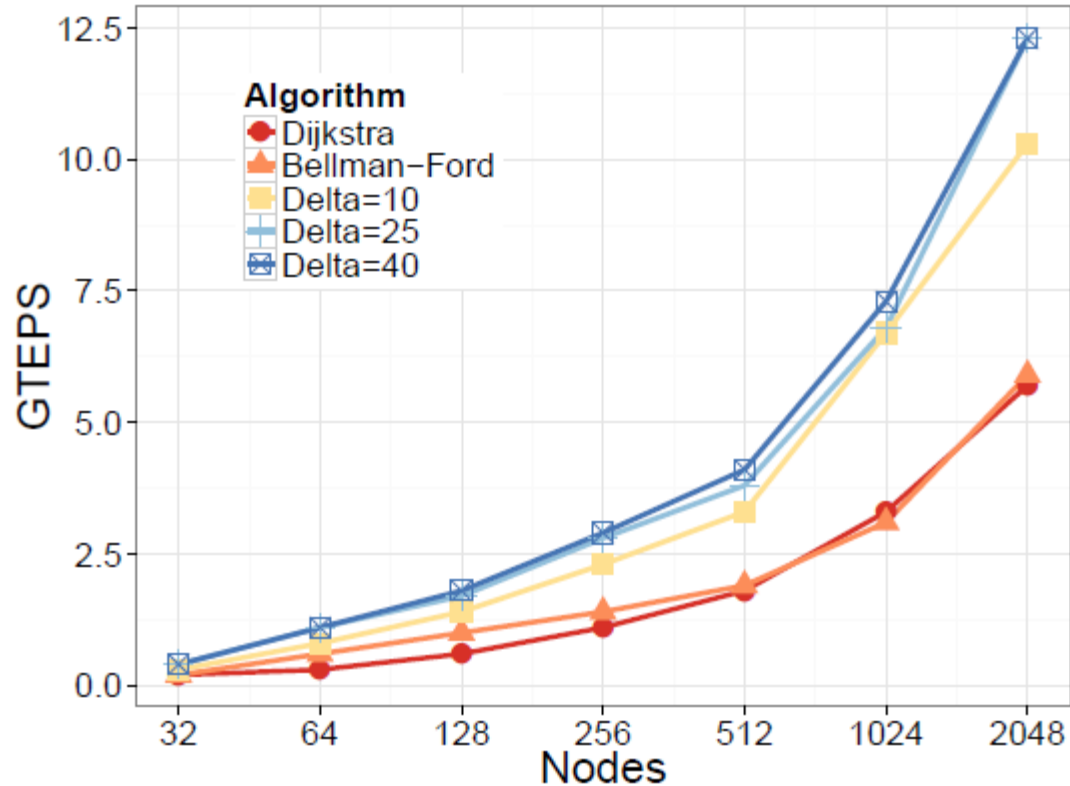
- ❑ Источник: Chakaravarthy V. T. et al. Scalable single source shortest path algorithms for massively parallel systems //IEEE Transactions on Parallel and Distributed Systems. – 2016. – Т. 28. – №. 7. – С. 2031-2045.
- ❑ Инфраструктура: СК Blue Gene/Q, 16-ядерные узлы с поддержкой SMT, 64 потока на узел.
- ❑ Реализация на C, с поддержкой Pthreads и SPI. Компилятор GCC 4.4.6.
- ❑ Тестовый граф: RMAT с параметрами  $A = 0.57$ ,  $B = C = 0.19$ ,  $D = 1 - A - 2B = 0.05$ . Средняя степень вершины – 32. Максимальная степень вершины:

Scale	28	29	30	31	32
RMAT – 1	2.4 M	3.8 M	5.9 M	9.4 M	14.4 M



# Результаты экспериментов.

## Сравнение разных алгоритмов



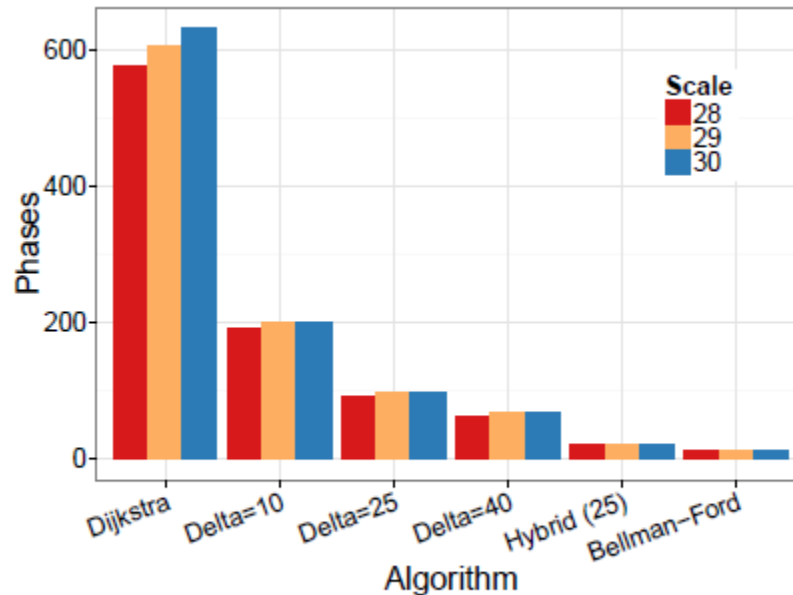
- Лучшая производительность – при Delta ~ средней степени вершины
- Производительность алгоритма дельта-шага в 2 раза выше, чем Дейкстры и Беллмана-Форда

RMAT – 1: Performance of  $\Delta$ -stepping algorithm

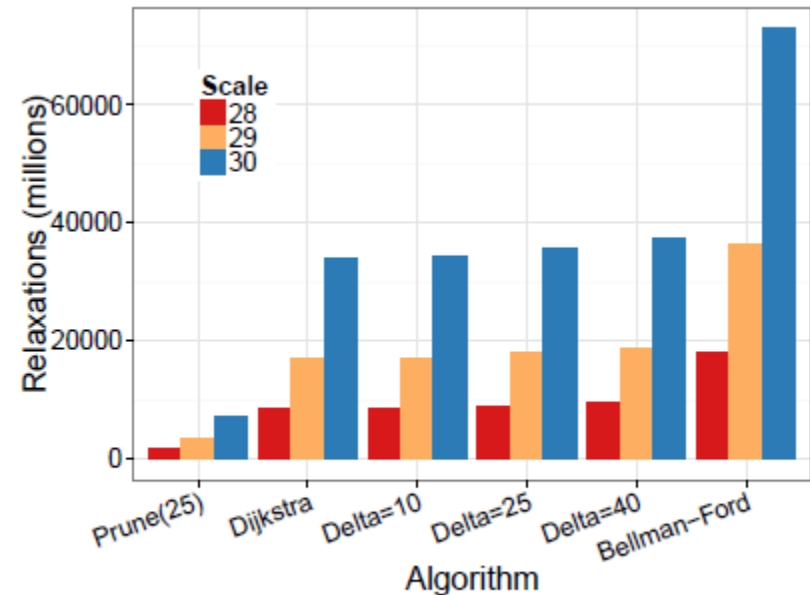
# Результаты экспериментов.

## Сравнение разных алгоритмов

□ Сравнение числа итераций и числа релаксаций



(a) Number of phases



(b) Number of relaxations

# Результаты экспериментов.

## Сравнение разных алгоритмов

---

### □ Выводы:

- Число итераций в алгоритме Дейкстры значительно больше, чем в алгоритмах дельта-шага и Беллмана-Форда
- Число релаксаций алгоритмов Дейкстры и дельта-шага близко, оно меньше, чем в алгоритме Беллмана-Форда в  $\sim 2$  раза.
- Оптимизация Prune сокращает число релаксаций в  $\sim 5$  раз
- Гибридизация алгоритмов дельта-шага и Беллмана-Форда позволяет сократить число внешних итераций в  $\sim 2.5$  раза. При этом производительность увеличивается на  $\sim 30\%$ .

# Результаты экспериментов.

## Графы из приложений

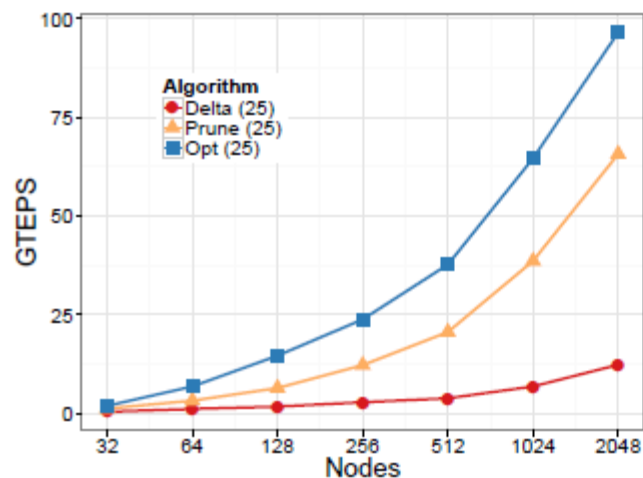
- ❑ Сравнение базовой и оптимизированной реализаций алгоритма дельта-шага ( $\Delta = 40$ ) на графах из реальных приложений:

GTEPS	Vertices	Edges	Del-40	Opt-40
Friendster	63 million	1.8 billion	1.8	4.3
Orkut	3 million	117 million	2.1	4.6
Live Journal	4.8 million	68 million	1.1	2.2

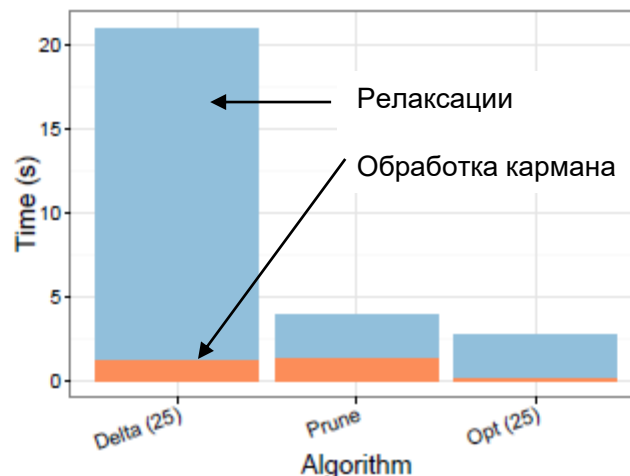
- ❑ Оптимизация позволила улучшить производительность в 2 раза.

# Результаты экспериментов.

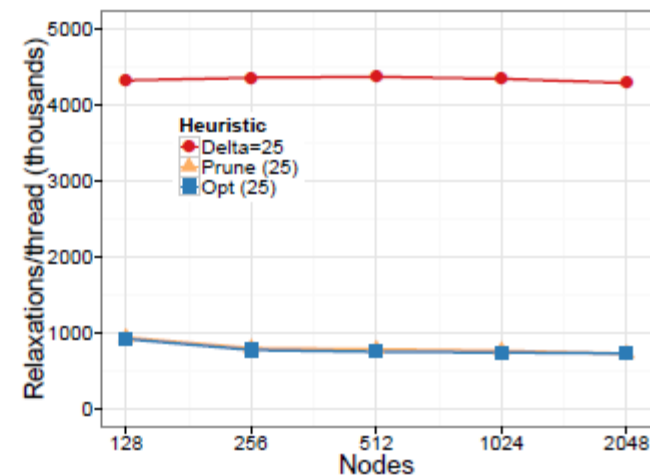
## Алгоритм дельта-шага



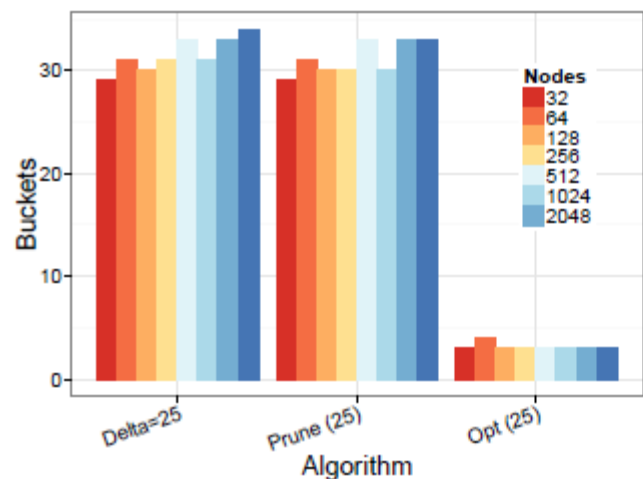
(a) GTEPS of different algorithms



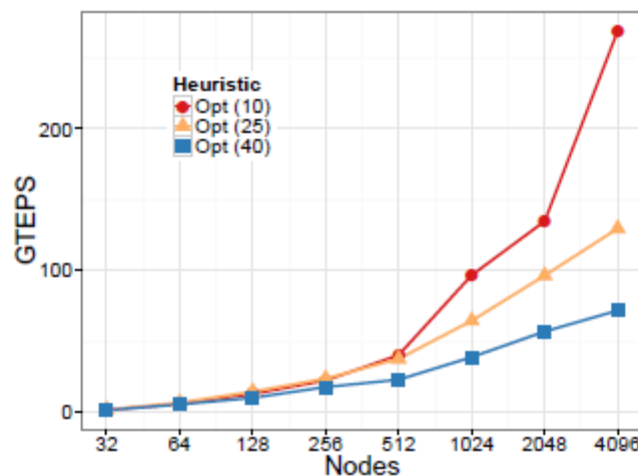
(b) Time breakdown



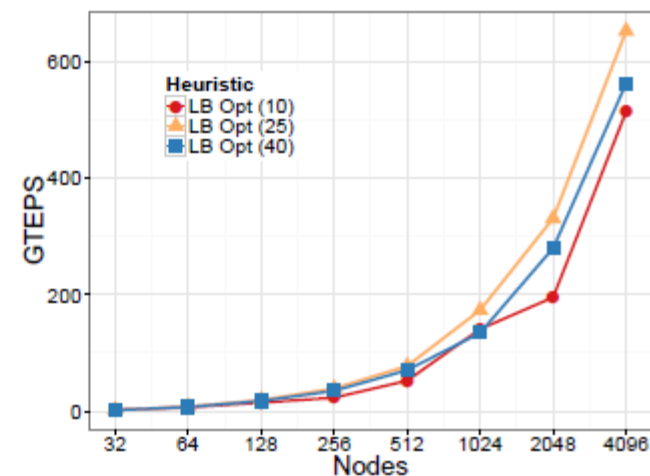
(c) Relaxations



(d) Number of buckets



(e) Effect of  $\Delta$  on OPT



(f) Effect of load balancing on OPT

# Результаты экспериментов. Выводы

- ❑ Оптимизированный алгоритм позволил получить производительность до 8 раз больше, чем исходный
- ❑ Оптимизация Prune сократила время релаксаций в  $\sim 7$  раз
- ❑ При  $\Delta = 25$  использовалось 30 карманов. В гибридной версии использовалось 5 карманов.
- ❑ Масштабируемость реализации ограничена дисбалансом вычислений, который возникает из-за разрыва в распределении степеней вершин тестового графа. Для балансировки использовалась параллельная обработка потоками в рамках одного вычислительного узла. Это позволило увеличить производительность реализации от 2 до 8 раз.

# Заключение

---

- ❑ Задача поиска кратчайших путей – одна из основных задач теории графов, имеющая широкое применение. Она применяется в комбинаторных задачах оптимизации и при анализе сложных сетей.
- ❑ Разработано большое число методов поиска кратчайших путей. Одни из широко используемых алгоритмов – алгоритм Дейкстры, алгоритм Беллмана–Форда, алгоритм дельта-шага.
- ❑ При построении параллельных алгоритмов Дейкстры и дельта-шага распределяются вычисления внутри итерации. При пересчете расстояния требуется коммуникация между процессами.

# Заключение

---

- ❑ Число итераций в алгоритме Дейкстры значительно больше, чем в алгоритме дельта-шага. При этом число релаксаций алгоритмов близко.
- ❑ Производительность алгоритма дельта-шага около 2 раз больше, чем алгоритма Дейкстры.
- ❑ Алгоритмическая оптимизация позволяет значительно улучшить производительность параллельной реализации алгоритма дельта-шага.



# Литература

1. Т. Кормен, Ч. Лейзерсон, Р. Ривест, К. Штайн. Алгоритмы: построение и анализ, 3-е издание. –М.: «Вильямс», 2013. –1328 с.
2. Crauser A. et al. A parallelization of Dijkstra's shortest path algorithm // Mathematical Foundations of Computer Science 1998. – 1998. – С. 722–731.
3. Meyer U., Sanders P.  $\Delta$ -stepping: a parallelizable shortest path algorithm //J. of Algorithms. – 2003. – Т. 49. – №. 1. – С. 114–152.
4. Madduri K. et al. Parallel shortest path algorithms for solving large-scale instances. – Georgia Institute of Technology, 2006.
5. Chakaravarthy V. T. et al. Scalable single source shortest path algorithms for massively parallel systems //IEEE Transactions on Parallel and Distributed Systems. – 2016. – Т. 28. – №. 7. – С. 2031-2045.

# Контакты

---

Нижегородский государственный университет

<http://www.unn.ru>

Институт информационных технологий, математики и механики

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

Пирова А.Ю.

[anna.pirova@itmm.unn.ru](mailto:anna.pirova@itmm.unn.ru)