コンピュータ科学特別講義Ⅳ

Parallel Algorithm Design（#8）

Masato Edahiro

July 6, 2018

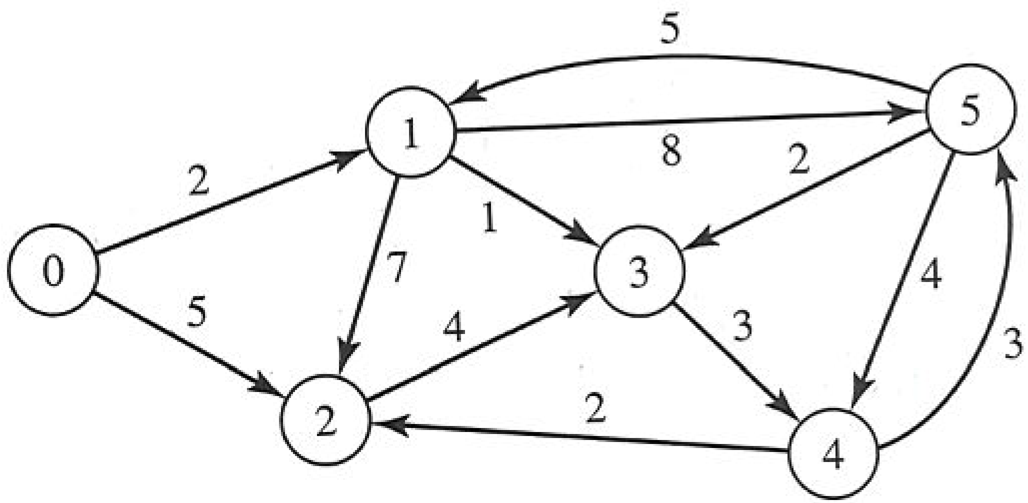
Please download handouts before class from
http://www.pdsl.jp/class/utyo2018/

Contents of This Class

- Our Target
  - Understand Systems and Algorithms on "Multi-Core" processors
- Schedule (Tentative)
  - #1 April 6 (=Today) What's "Multi-Core"?
  - #2 April 13 : Parallel Programming Languages (Ex. 1)
  - April 20, 27, May 4, 11, 18: NO CLASS
  - #3 May 25 : Parallel Algorithm Design
  - #4 June 1 (Fri) : Laws on Multi-Core
  - #5 June 8 : Examples of Parallel Algorithms (1) (Ex. 2)
  - June 15: NO CLASS
  - #6 June 22 : Examples of Parallel Algorithms (2)
  - #7 June 29 : Examples of Parallel Algorithms (3)
  - #8 July 6 : Examples of Parallel Algorithms (4)
  - #9 July 13 : Examples of Parallel Algorithms (5) (Ex. 3)
  - (July 20)
  - If you want to graduate in August, ask Edahiro asap.

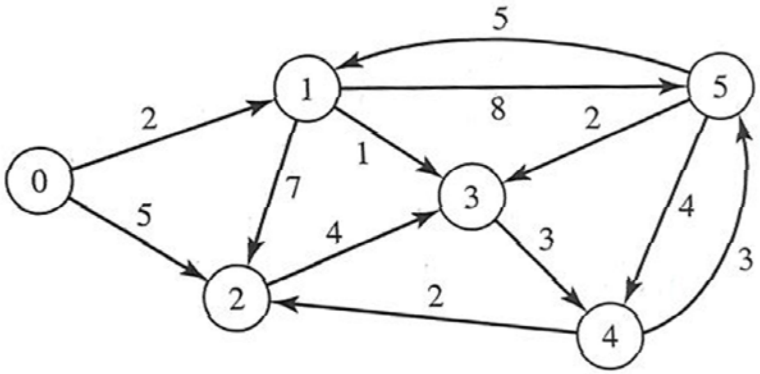
全点間最短経路問題

- 重み付き有向グラフG(V, E)が与えられたとき、すべての点対に対する最短経路長を求める問題を全点間最短経路問題という
- n: 点の数, p: プロセッサの数



Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

隣接行列



(a)

	0	1	2	3	4	5
0	0	2	5	∞	∞	∞
1	∞	0	7	1	∞	8
2	∞	∞	0	4	∞	∞
3	∞	∞	∞	0	3	∞
4	∞	∞	2	∞	0	3
5	∞	5	∞	2	4	0

(b)

	0	1	2	3	4	5
0	0	2	5	3	6	9
1	∞	0	6	1	4	7
2	∞	15	0	4	7	10
3	∞	11	5	0	3	6
4	∞	8	2	5	0	3
5	∞	5	6	2	4	0

(c)

Figure 6.1 (a) A weighted, directed graph. (b) Representation of the graph as an adjacency matrix. Element (i, j) represents the length of the edge from i to j. Nonexistent edges are considered to have infinite length. (c) Solution to the all-pairs shortest path problem. Element (i, j) represents the length of the shortest path from vertex i to vertex j. The infinity symbol represents nonexistent paths.

フロイトのアルゴリズム

Input: n — number of vertices
$a[0..n-1, 0..n-1]$ — adjacency matrix
Output: Transformed a that contains the shortest path lengths

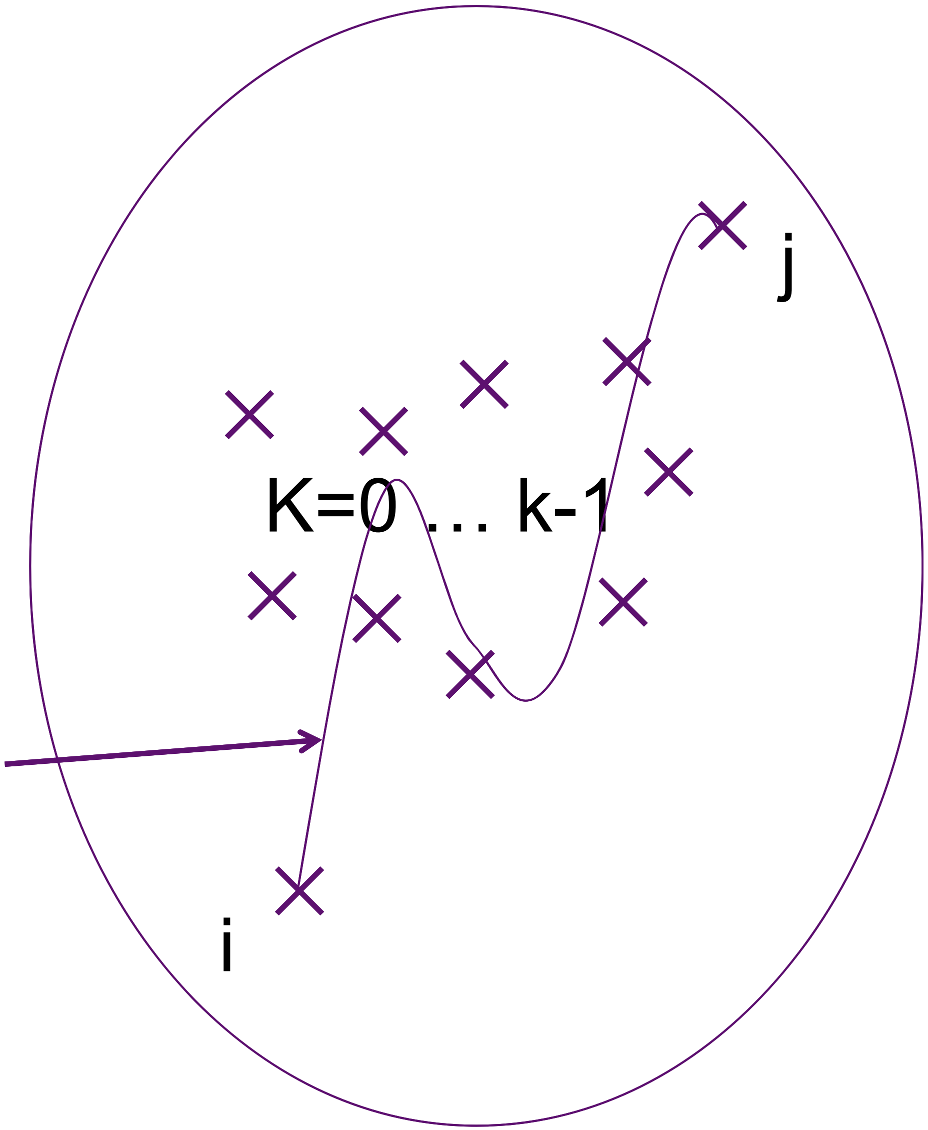
for $k \leftarrow 0$ to $n - 1$
  for $i \leftarrow 0$ to $n - 1$
    for $j \leftarrow 0$ to $n - 1$
      $a[i, j] \leftarrow \min(a[i, j], a[i, k] + a[k, j])$
    endfor
  endfor
endfor

Figure 6.2 Floyd's algorithm is an $\boxed{\Theta(n^3)}$ time algorithm that solves the all-pairs shortest-path problem. It transforms an adjacency matrix into a matrix containing the length of the shortest path between every pair of vertices.

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

フロイトのアルゴリズム

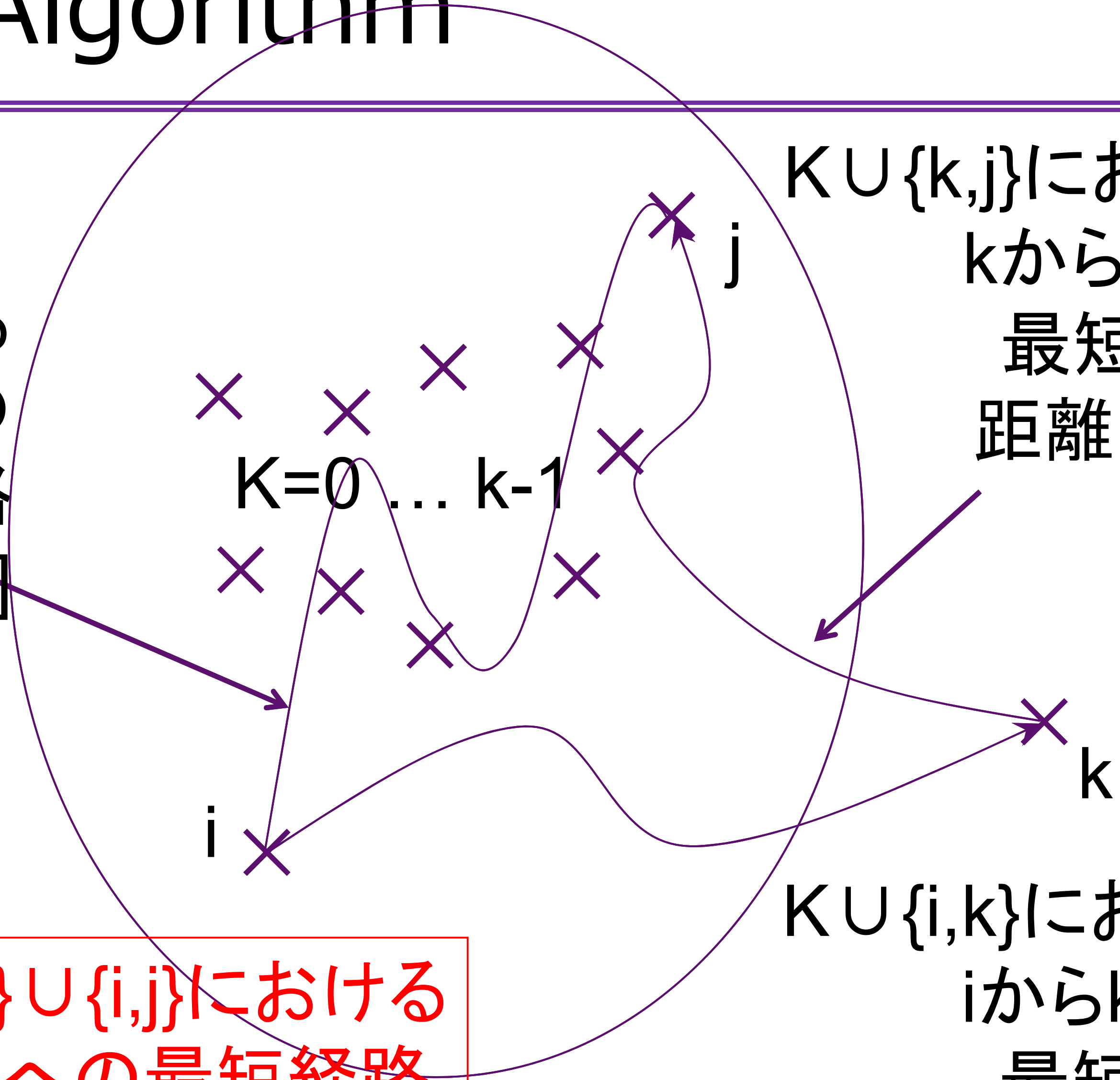
K∪{i,j}における
iからjへの
最短経路
距離 a[i,j]



Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

Floyd’s Algorithm

K∪{i,j}における
iからjへの
最短経路
距離 a[i,j]



K∪{k,j}における
kからjへの
最短経路
距離 a[k,j]

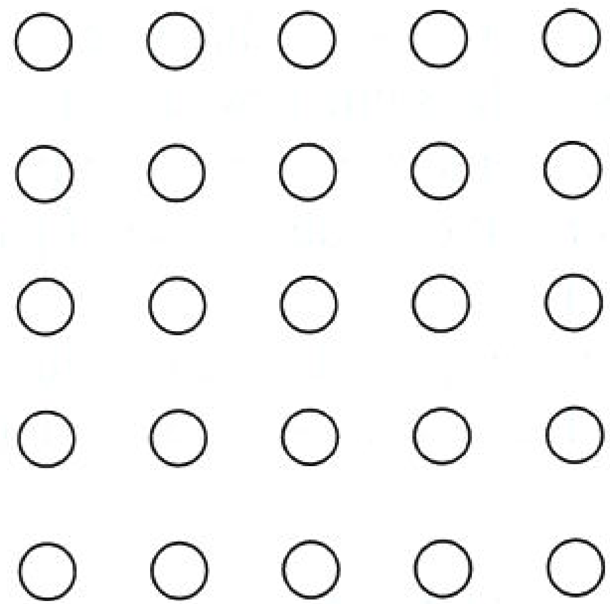
K∪{i,k}における
iからkへの
最短経路
距離 a[i,k]

K∪{k}∪{i,j}における
iからjへの最短経路
a[i,j] ← min(a[i,j], a[i,k]+a[k,j])

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

Partitioning

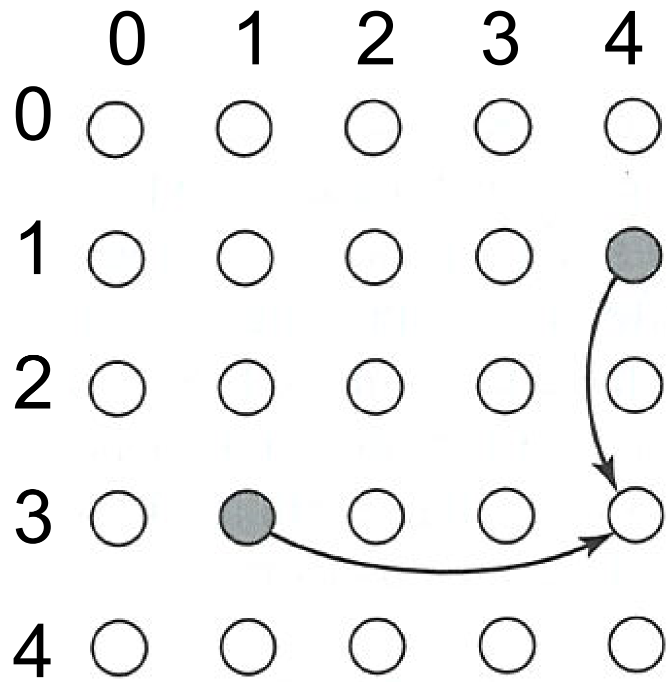
- Partitioning
  - n^2 for (i,j)



Communication for Calculation

- a[i,j]←min(a[i,j],a[i,k]+a[k,j])

for $k \leftarrow 0$ to $n-1$ k: fix
 for $i \leftarrow 0$ to $n-1$ i,j: parallel Calc.
  for $j \leftarrow 0$ to $n-1$
   $a[i,j] \leftarrow \min(a[i,j], a[i,k] + a[k,j])$
  endfor
 endfor
endfor

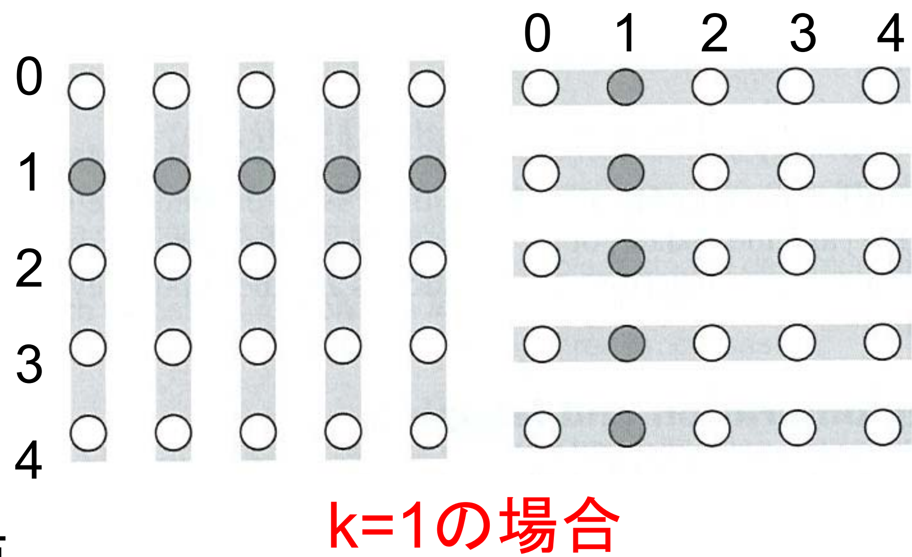


i=3, j=4, k=1

a[3,4] ← min(a[3,4], a[3,1]+a[1,4])

Communication

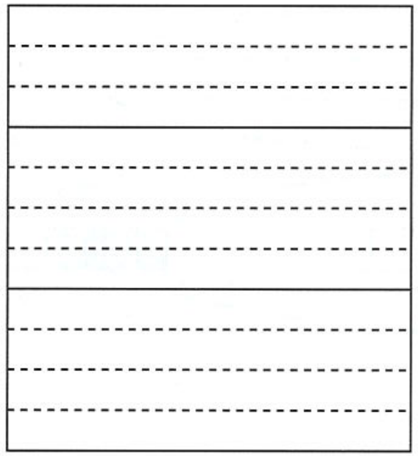
- Communication
  - 各kに対し、n個のデータに関する列方向通信とn個のデータに関する行方向通信が必要




```
for k ← 0 to n − 1
  for i ← 0 to n − 1
    for j ← 0 to n − 1
      a[i, j] ← min(a[i, j], a[i, k] + a[k, j])
    endfor
  endfor
endfor
```

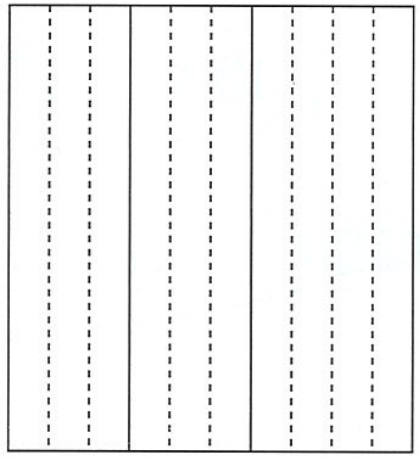
Agglomeration & Mapping

- 行方向分割
  - 行方向broadcastに対してオーバーヘッド無
- 列方向分割
  - 列方向broadcastに対してオーバーヘッド無
- どちらがいいのか？
  - C言語を使っているならば、配列は行方向優先で実現されるため、行方向ブロック分割の方が性能が出やすい



(a)

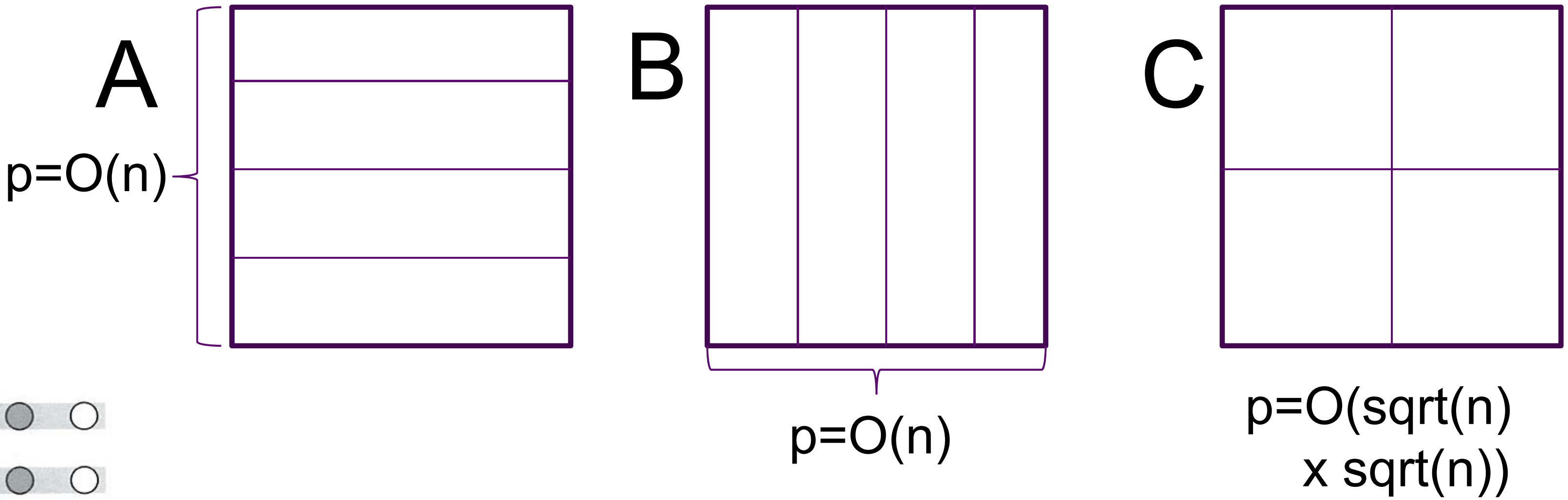
3プロセッサで11行を分割

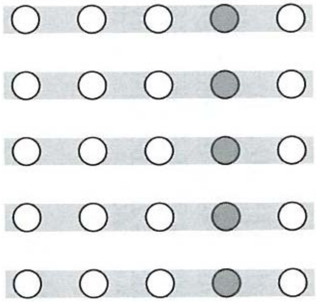


(b)

3プロセッサで10列を分割

Mapping & Communication

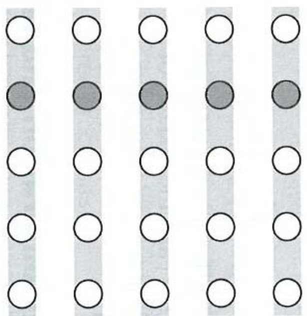




オーバーヘッド
なし

n個のデータを
pプロセッサに
通信

?



n個のデータを
pプロセッサに
通信

オーバーヘッド
なし

?

解析（ケースAとBの場合）

- n: 点の数
- p: プロセッサの数
- χ: 一つの配列要素を更新するための平均時間
- λ: 通信路初期化に必要な時間
- β: バンド幅

(一回のbroadcastには(log p)steps. 各stepで4nバイト通信する時間 = λ+4n/β（n個の4バイトデータ）)

- 最外ループの各kにおいて
  - 計算時間: $\chi\, n^2/p$
  - 通信時間: $(\lambda+4n/\beta)\ (\log p)$
- 全体の時間:

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

実験結果

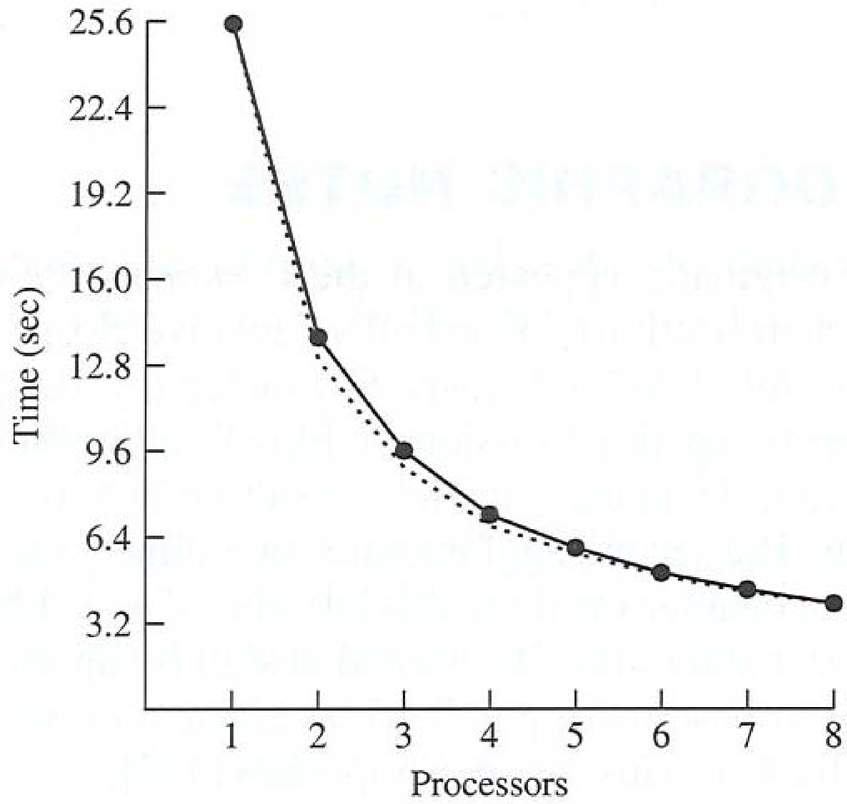
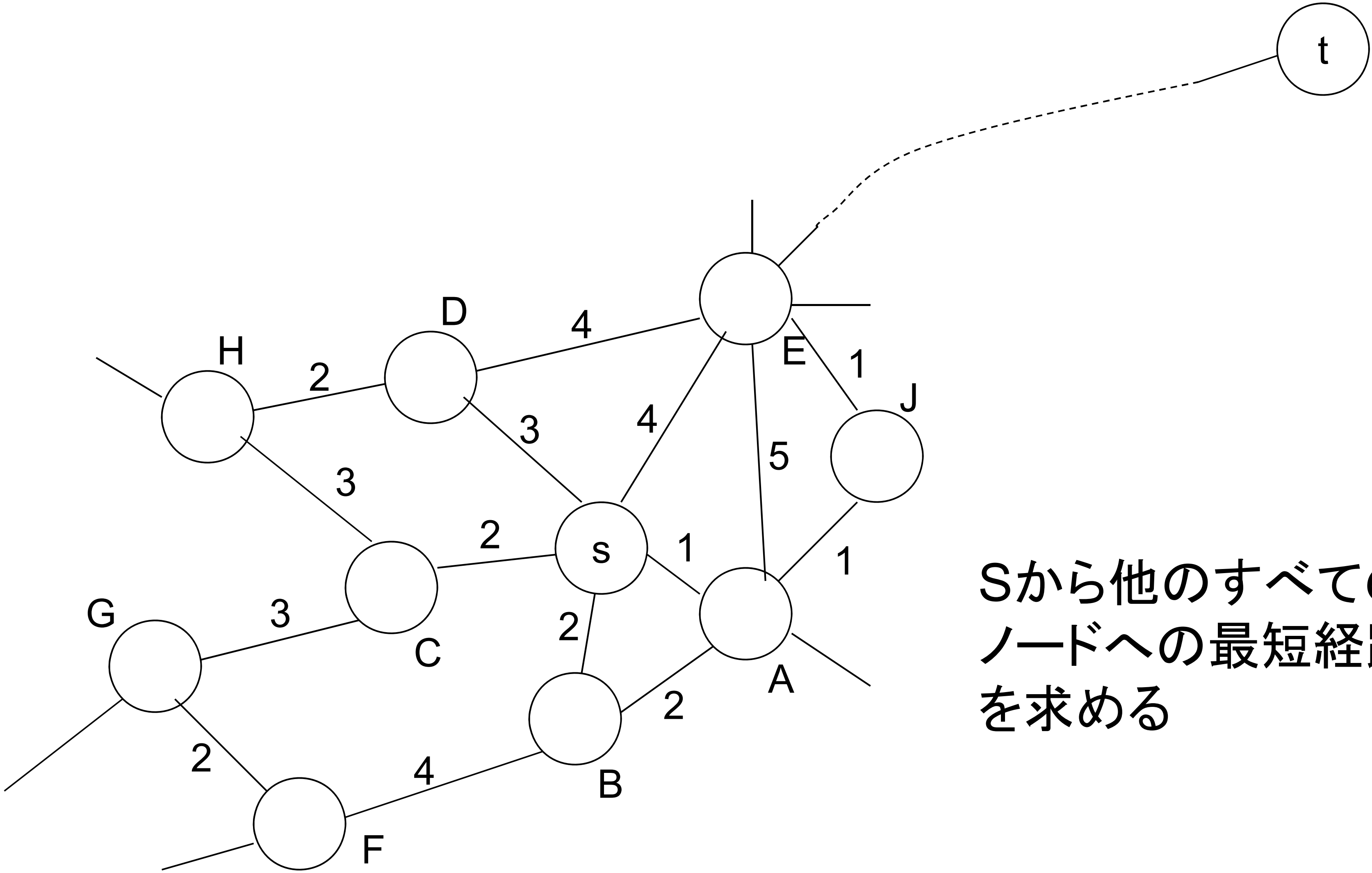


Figure 6.11 Predicted (dotted line) and actual (solid line) execution times of parallel implementation of Floyd's algorithm on a commodity cluster, solving a problem of size 1,000.

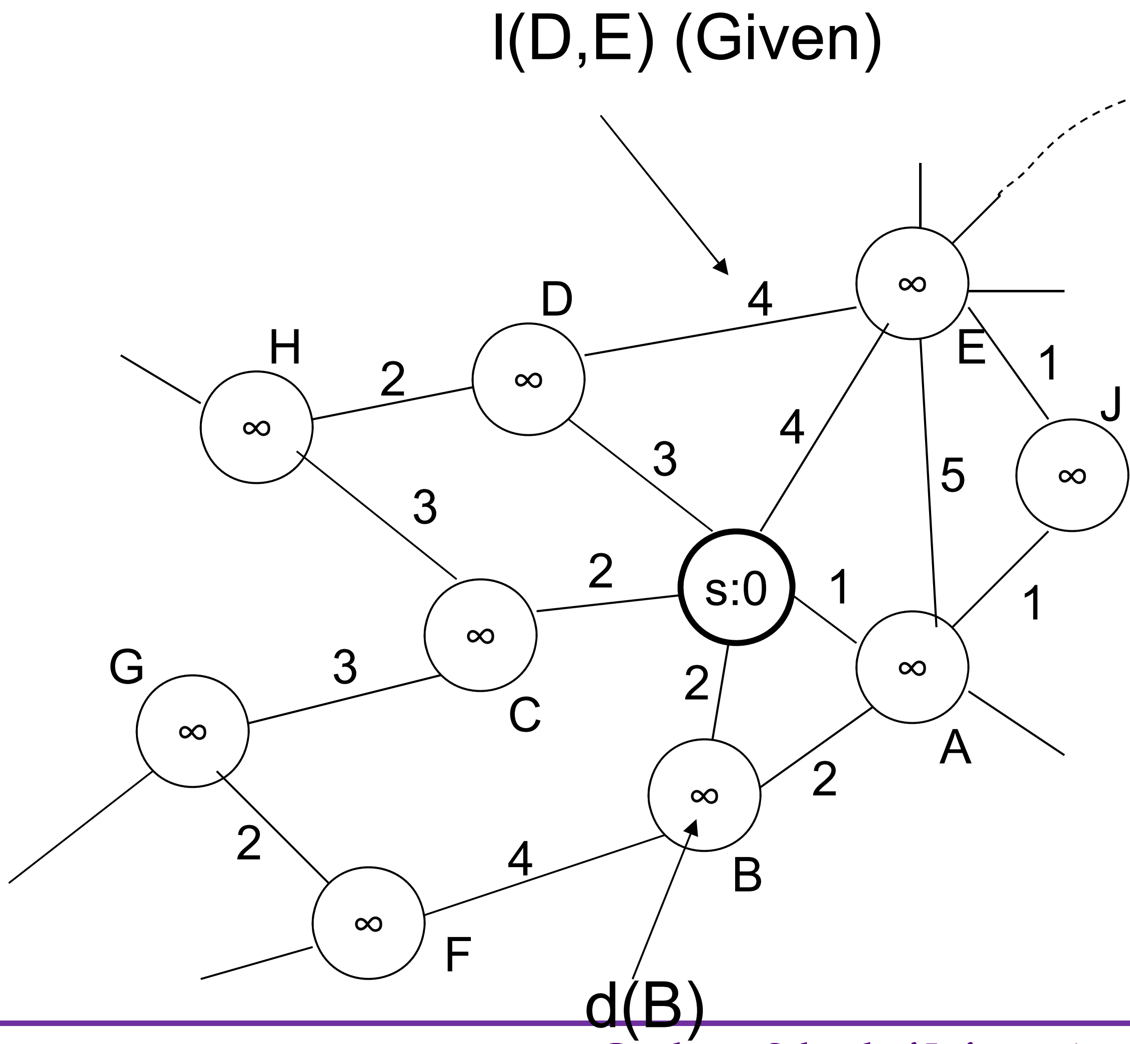
Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

最短経路問題



Sから他のすべてのノードへの最短経路を求める

ダイクストラ法

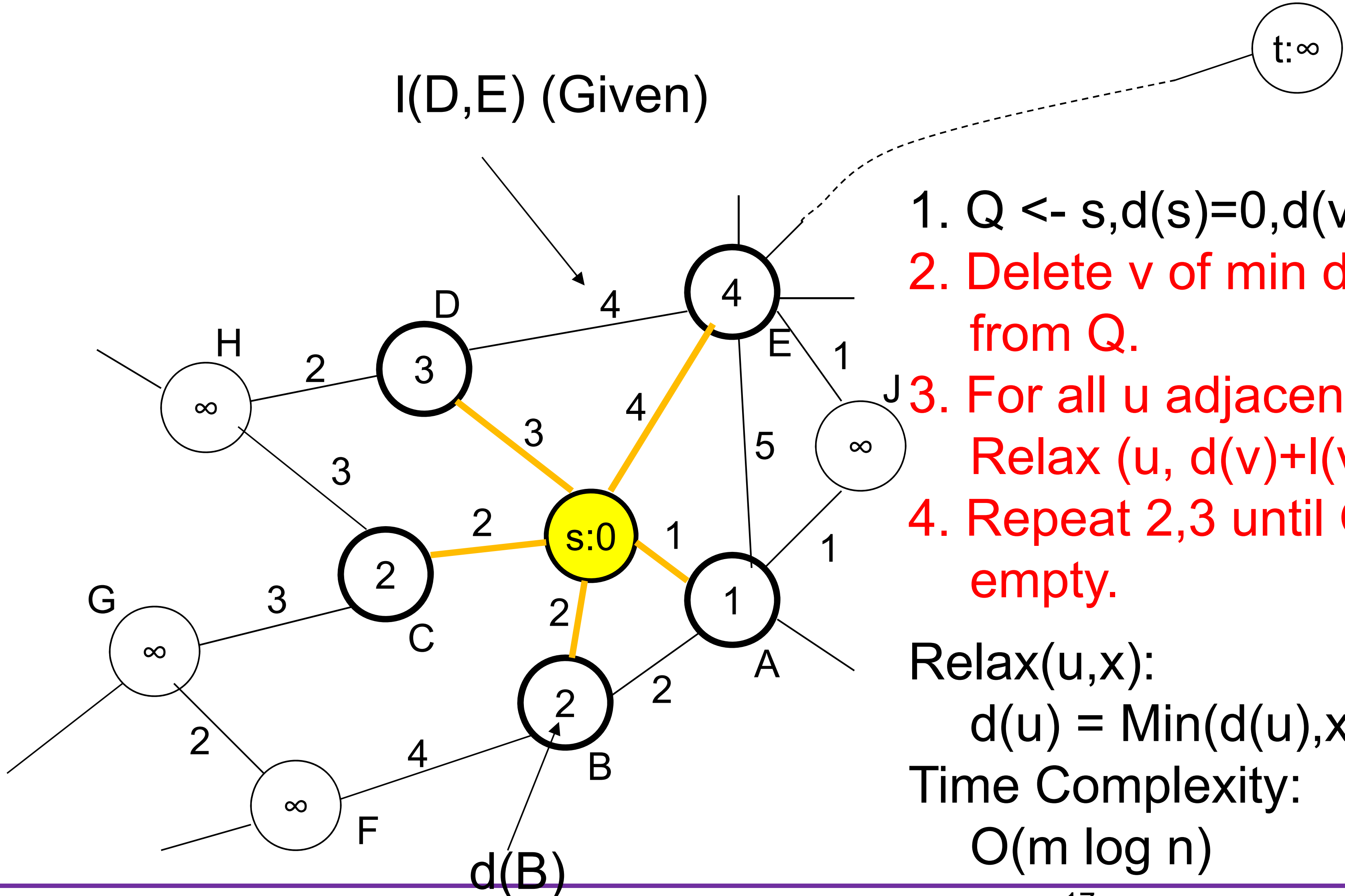


1. Q <- s,d(s)=0,d(v)=∞
2. Delete v of min d(v) from Q.
3. For all u adjacent to v, Relax (u, d(v)+l(v,u))
4. Repeat 2,3 until Q is empty.

Relax(u,x):
  d(u) = Min(d(u),x)

Time Complexity:
  O(m log n)

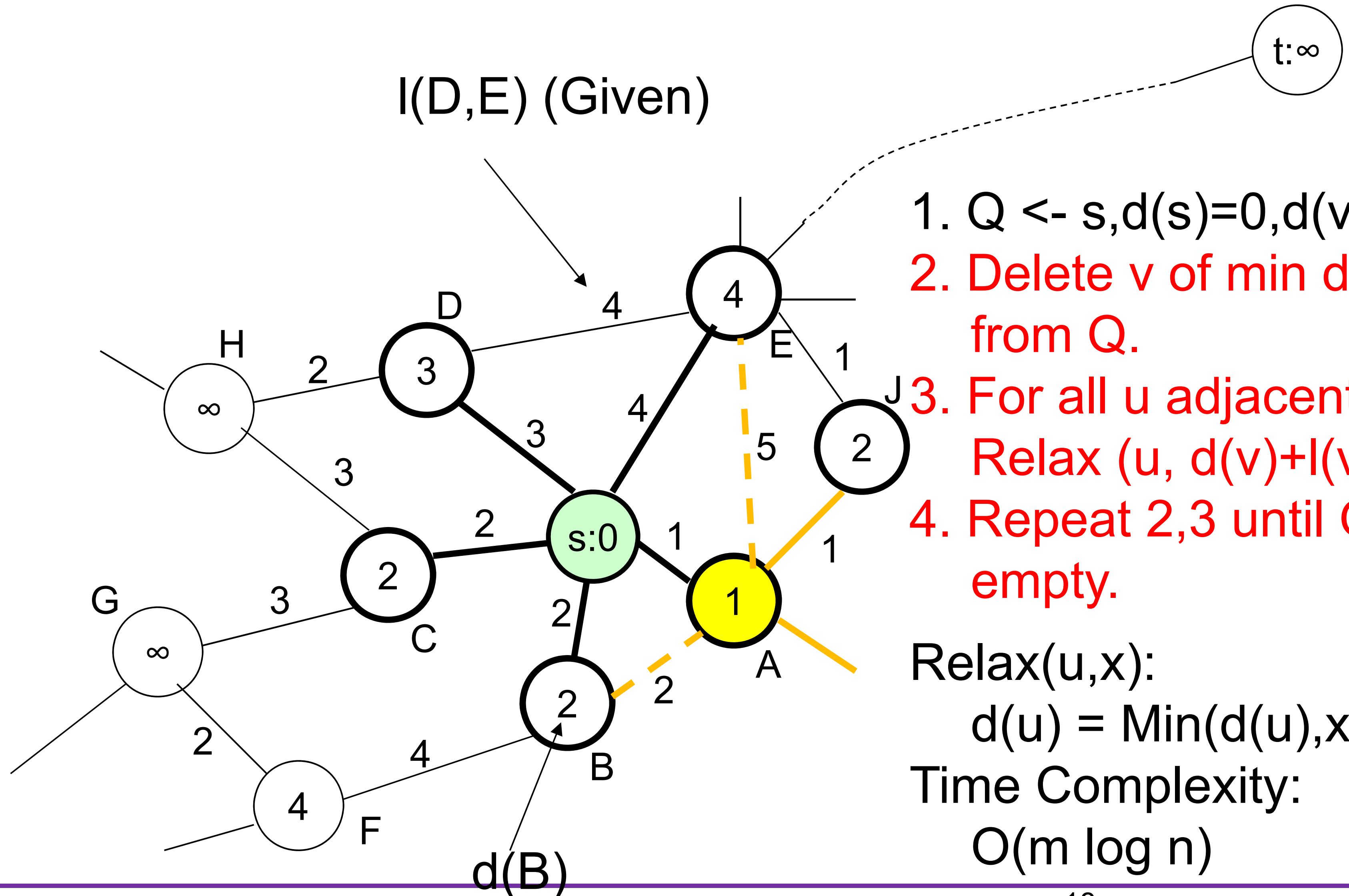
ダイクストラ法(Loop 1)



1. Q <- s,d(s)=0,d(v)=∞
2. Delete v of min d(v) from Q.
3. For all u adjacent to v, Relax (u, d(v)+l(v,u))
4. Repeat 2,3 until Q is empty.

Relax(u,x):
d(u) = Min(d(u),x)
Time Complexity:
O(m log n)

ダイクストラ法(Loop 2)



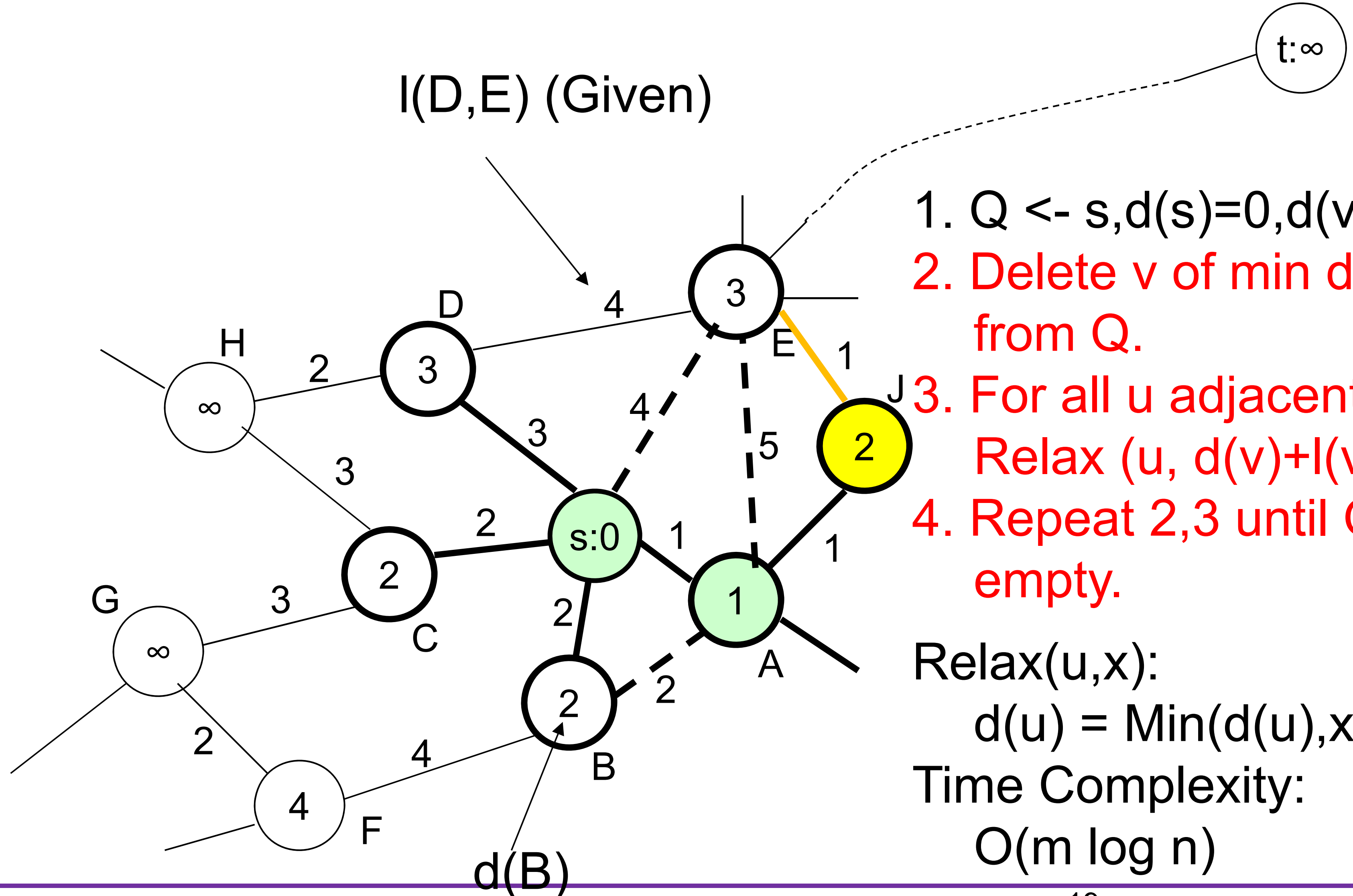
1. Q <- s,d(s)=0,d(v)=∞
2. Delete v of min d(v) from Q.
3. For all u adjacent to v, Relax (u, d(v)+l(v,u))
4. Repeat 2,3 until Q is empty.

Relax(u,x):
d(u) = Min(d(u),x)

Time Complexity:
O(m log n)

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

ダイクストラ法(Loop 3)

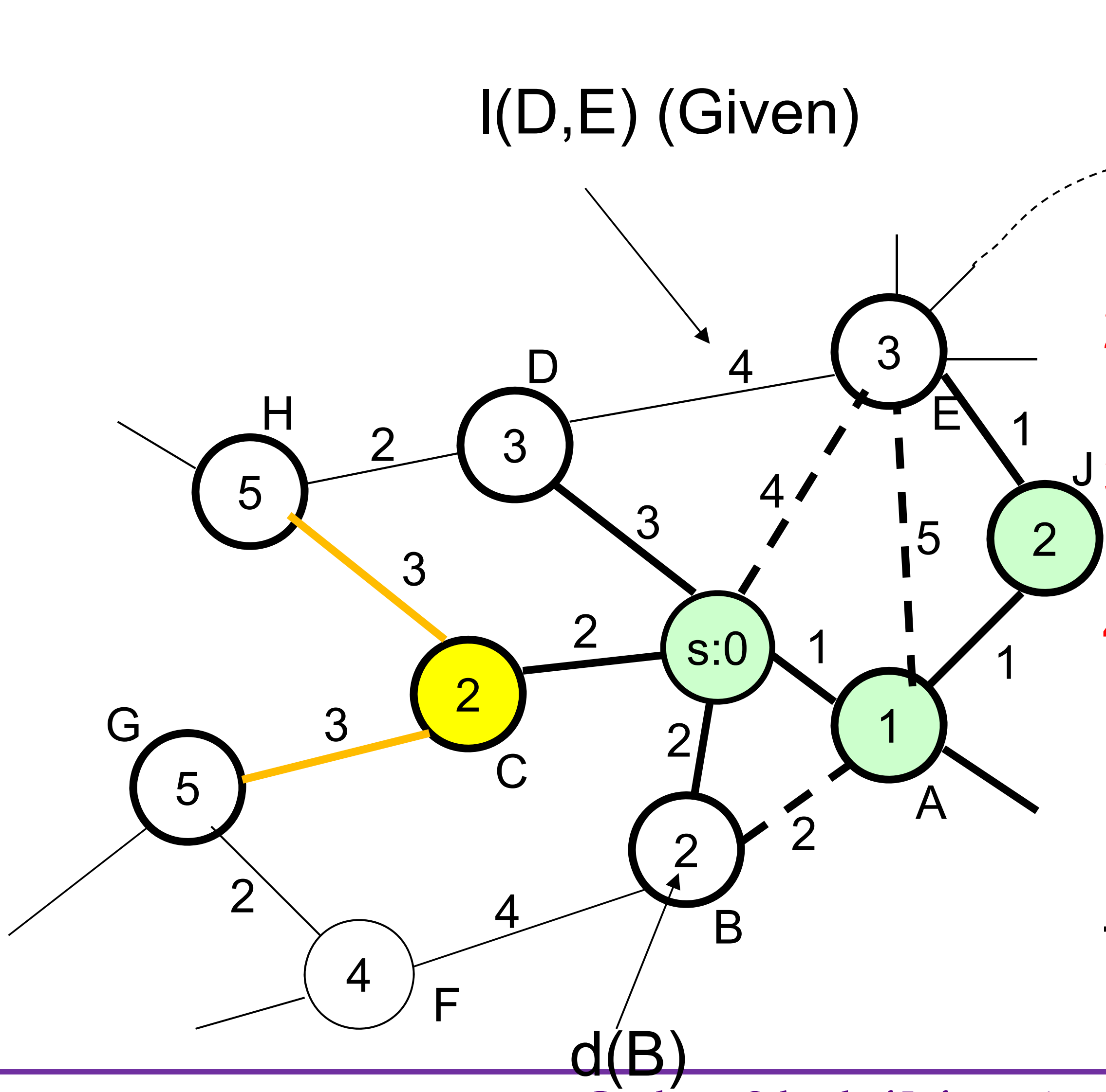


1. Q <- s,d(s)=0,d(v)=∞
2. Delete v of min d(v) from Q.
3. For all u adjacent to v, Relax (u, d(v)+l(v,u))
4. Repeat 2,3 until Q is empty.

Relax(u,x):
  d(u) = Min(d(u),x)

Time Complexity:
  O(m log n)

ダイクストラ法(Loop 4)



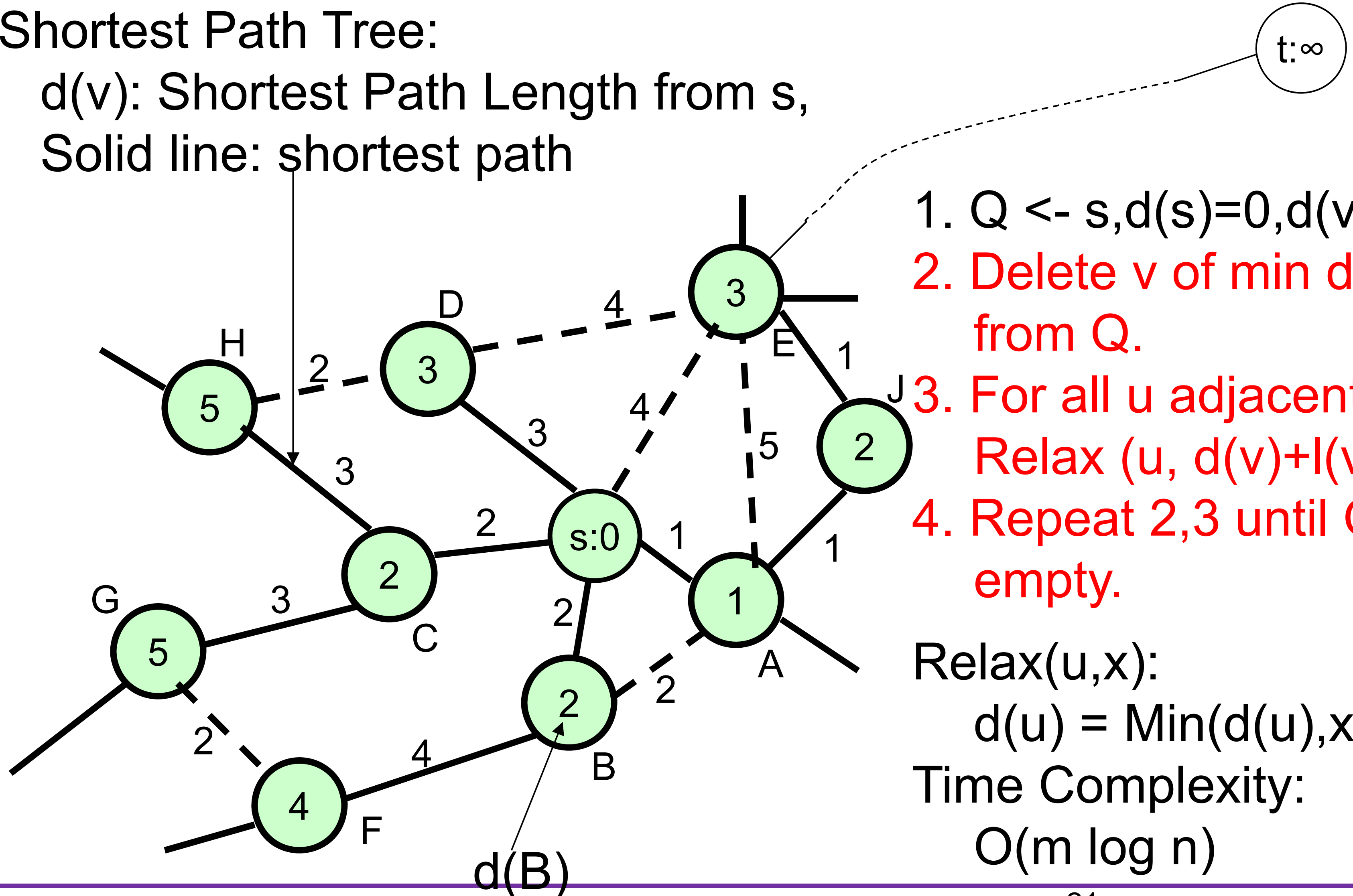
1. Q <- s,d(s)=0,d(v)=∞
2. Delete v of min d(v) from Q.
3. For all u adjacent to v, Relax (u, d(v)+l(v,u))
4. Repeat 2,3 until Q is empty.

Relax(u,x):
    d(u) = Min(d(u),x)

Time Complexity:
    O(m log n)

ダイクストラ法(Loop 10)

Shortest Path Tree:
  d(v): Shortest Path Length from s,
  Solid line: shortest path

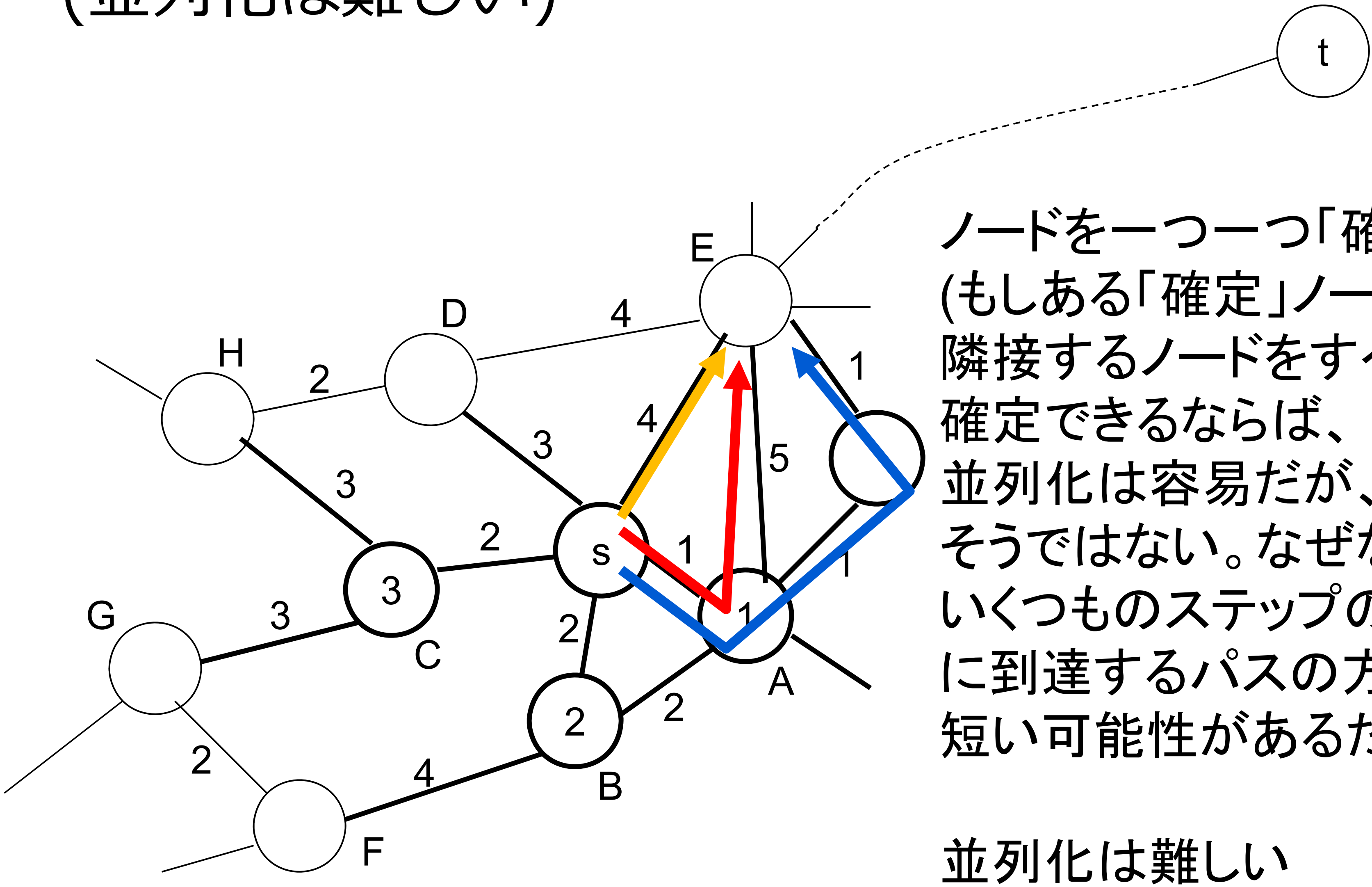


1. Q <- s,d(s)=0,d(v)=∞
2. Delete v of min d(v) from Q.
3. For all u adjacent to v, Relax (u, d(v)+l(v,u))
4. Repeat 2,3 until Q is empty.

Relax(u,x):
  d(u) = Min(d(u),x)

Time Complexity:
  O(m log n)

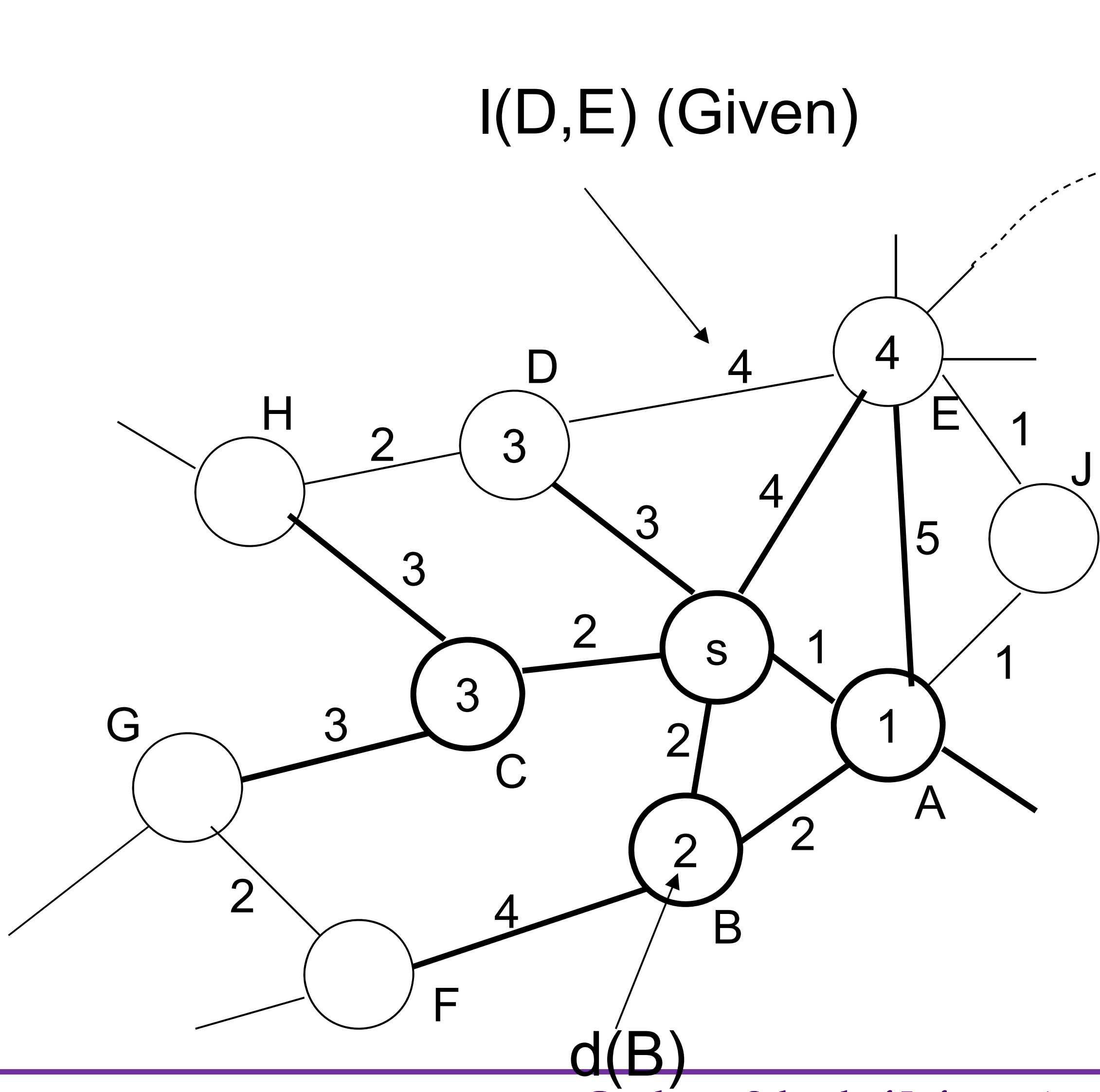
ダイクストラ法
(並列化は難しい)



ノードを一つ一つ「確定」
(もしある「確定」ノードに隣接するノードをすべて確定できるならば、並列化は容易だが、そうではない。なぜならいくつものステップの後に到達するパスの方が短い可能性があるため。

並列化は難しい

Bellman-Ford法
(並列化容易だがオーバーヘッド大)



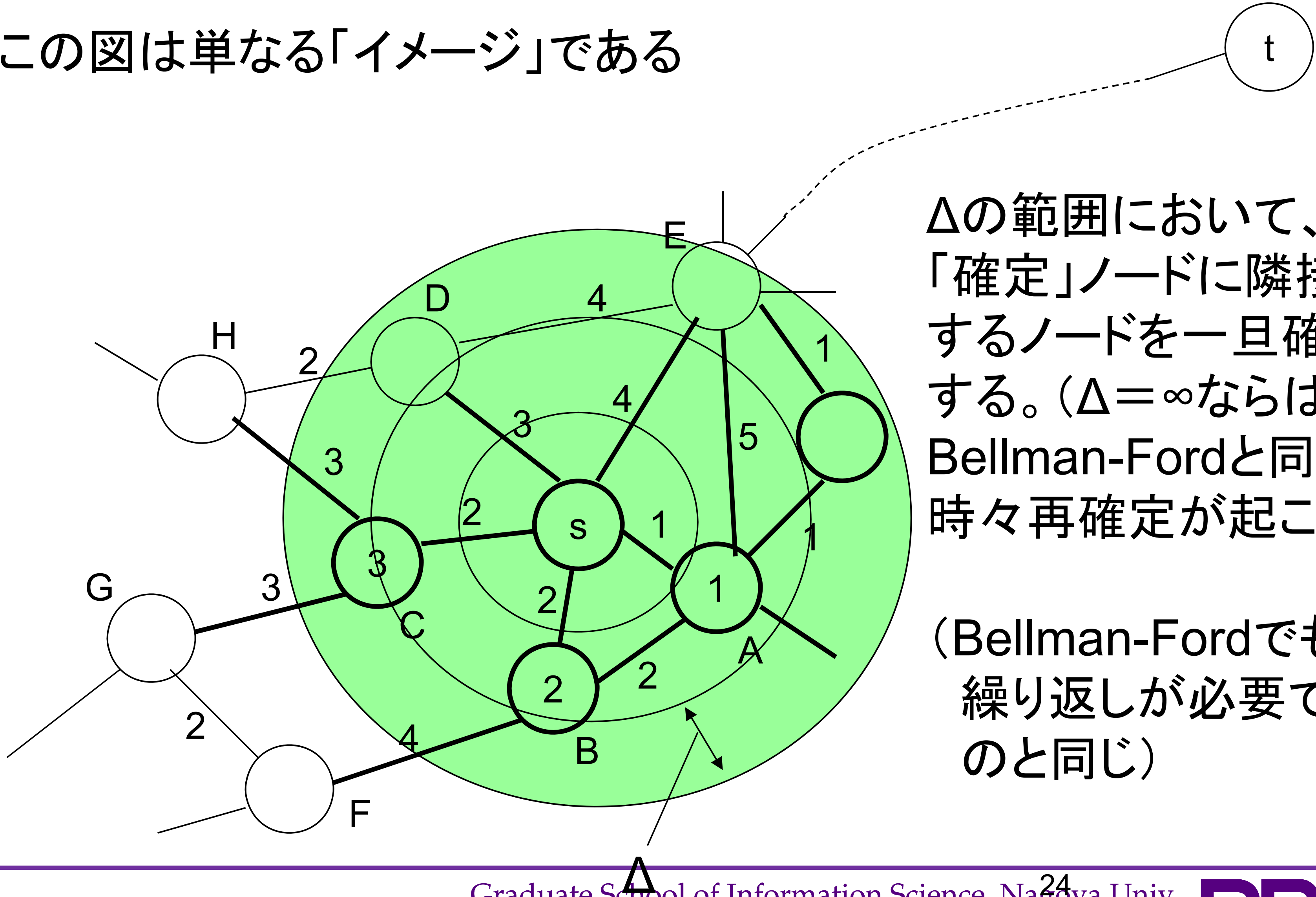
1. d(s)=0,d(v)=∞
2. For all v,
   For all u adjacent to v,
   Relax (v, d(u)+l(v,u))
3. Repeat 2 n times.

Relax(u,x):
  d(u) = Min(d(u),x)
Time Complexity:
  O(mn)

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

Δ-Stepping Algorithm

この図は単なる「イメージ」である

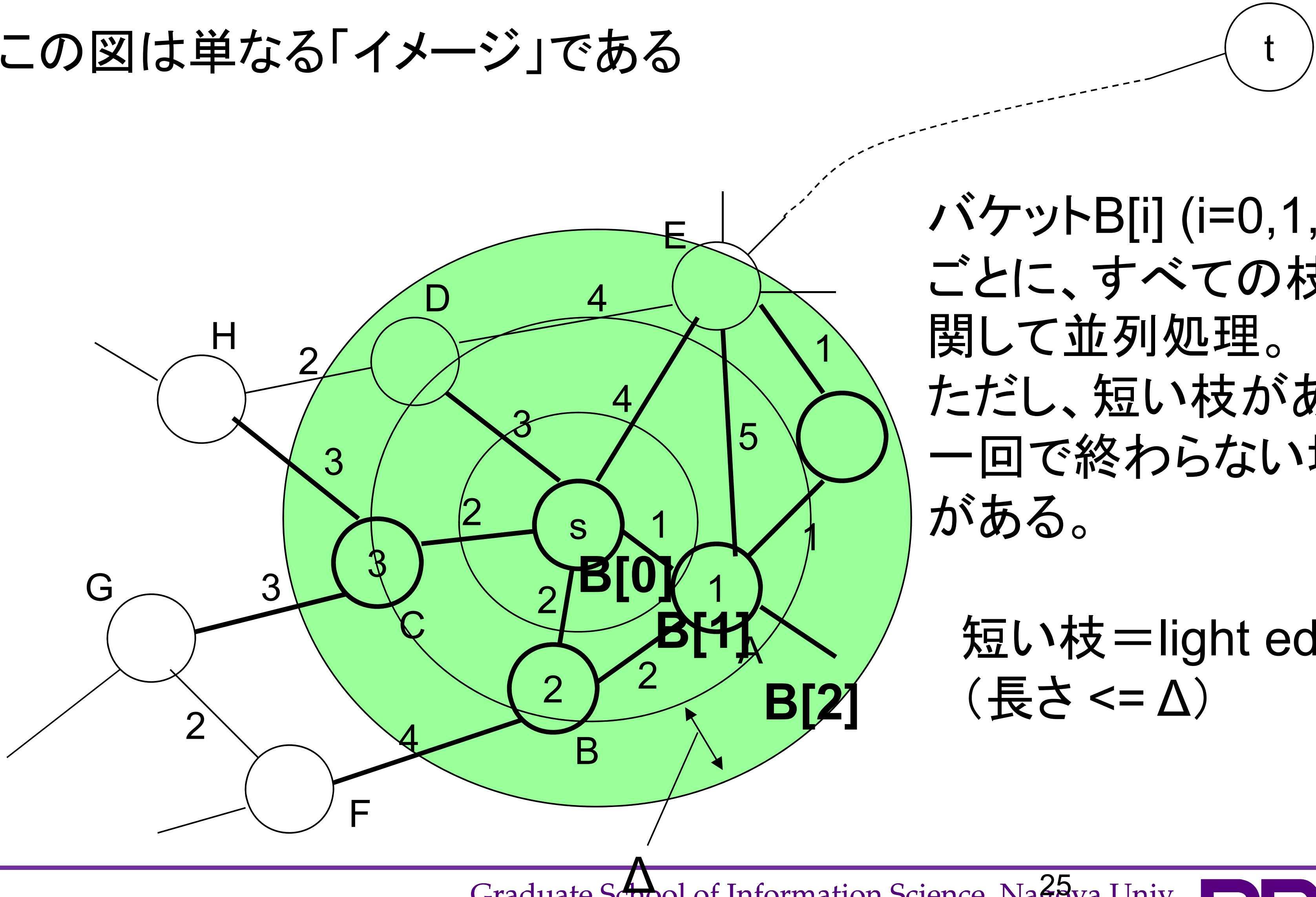


Δの範囲において、「確定」ノードに隣接するノードを一旦確定する。（Δ＝∞ならばBellman-Fordと同じ）時々再確定が起こる。

（Bellman-Fordでも繰り返しが必要であるのと同じ）

Δ-Stepping Algorithm

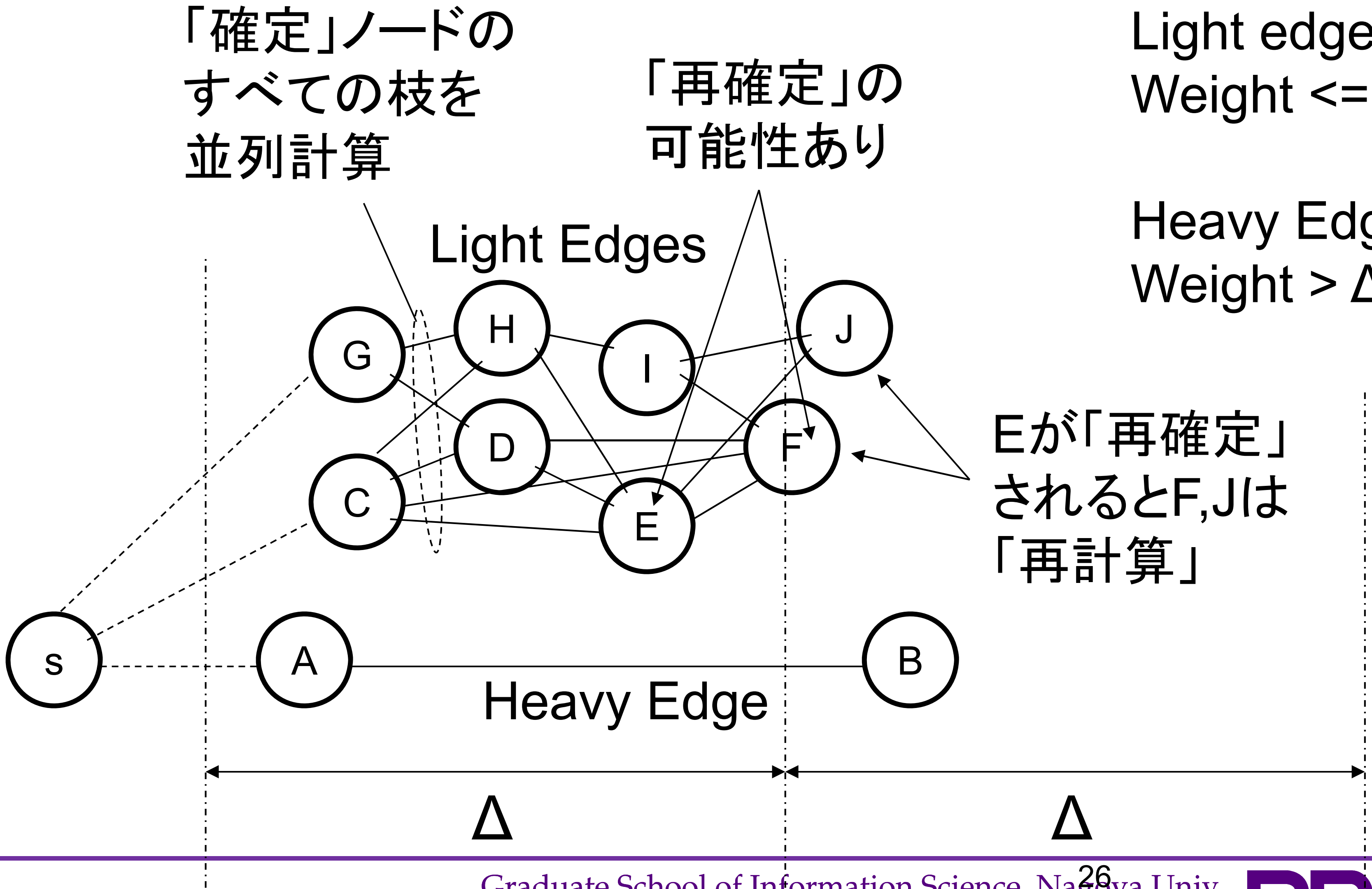
この図は単なる「イメージ」である



バケットB[i] (i=0,1,2…)ごとに、すべての枝に関して並列処理。
ただし、短い枝があると、一回で終わらない場合がある。

短い枝＝light edge
（長さ <= Δ）

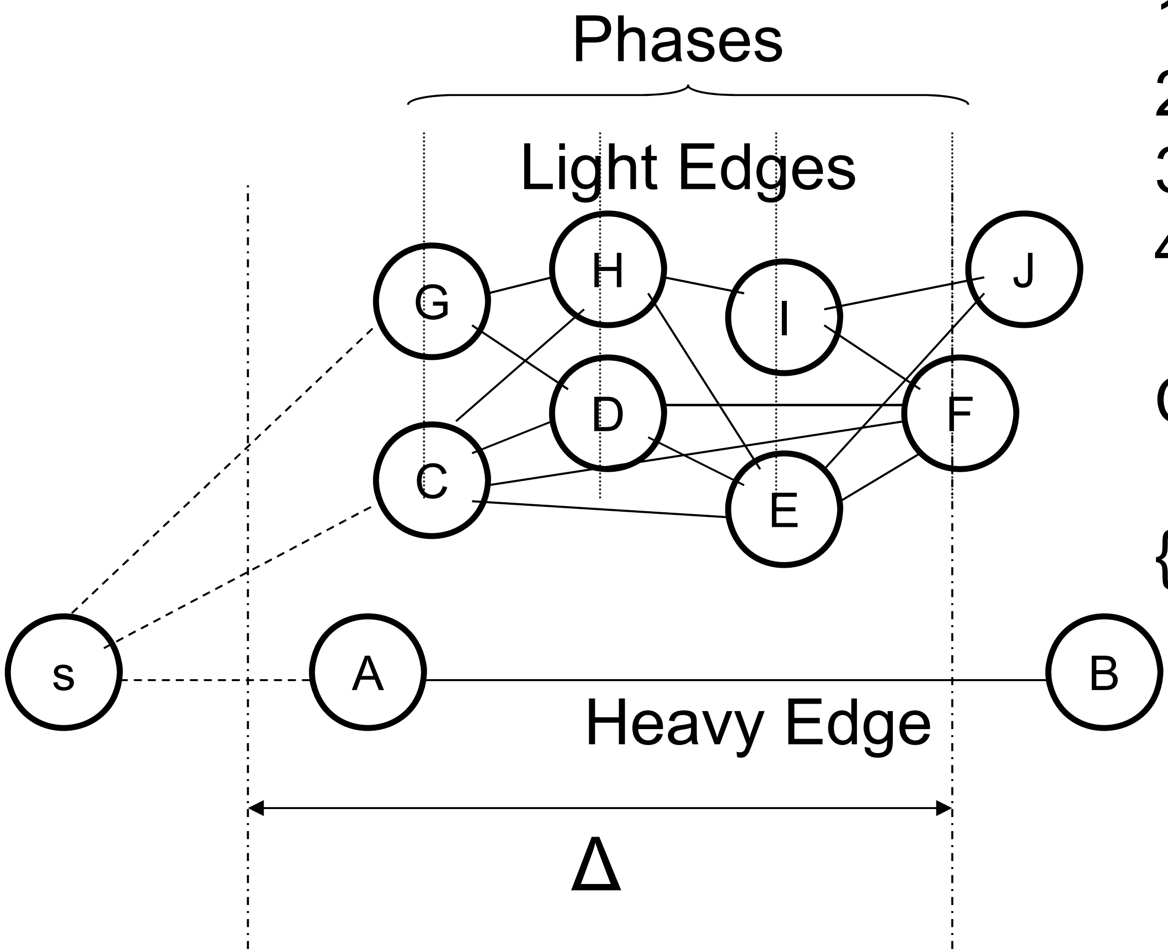
Δ-Stepping Algorithm



Light edge
Weight <= Δ

Heavy Edge
Weight > Δ

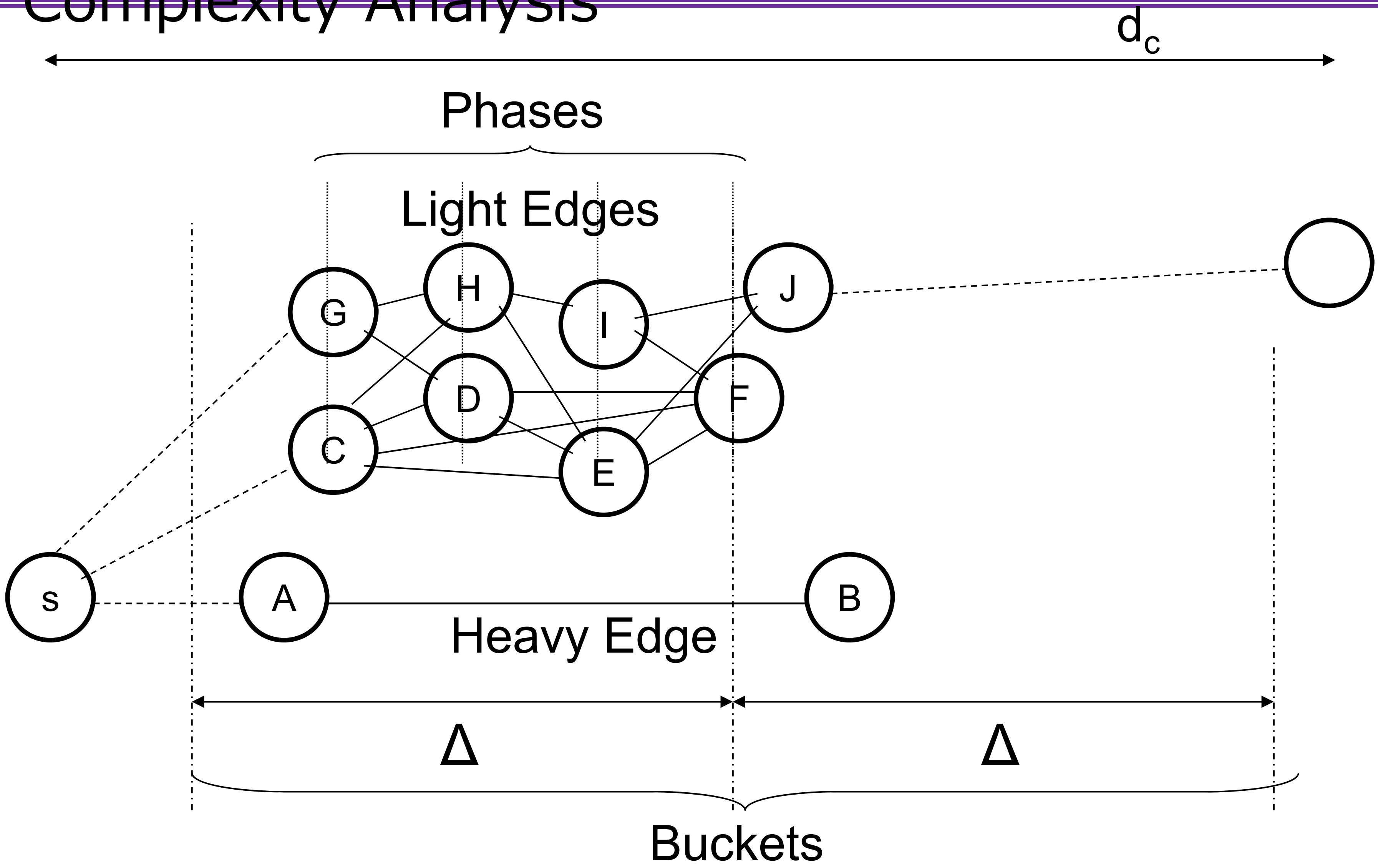
Δ-Stepping Algorithm



Contents of B[i] (while line 9-14 of Algorithm)
1st Phase: {A, C, G}
2nd Phase: {D, E, F, H}
3rd Phase: {E, F, I}
4th Phase] {F, J}

Contents of S (at line 15 of Algorithm)
{A, C, D, E, F, G, H, I}

Δ-Stepping Algorithm
Complexity Analysis



Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

Algorithm (Δ-Steppnig)

Algorithm 1: Δ-stepping algorithm

Input: $G(V, E)$, source vertex s, length function $l : E \rightarrow \mathbb{R}$

Output: $\delta(v), v \in V$, the weight of the shortest path from s to v

foreach $v \in V$ **do**
- $heavy(v) \longleftarrow \{\langle v, w\rangle \in E : l(v, w) > \Delta\}$;
- $light(v) \longleftarrow \{\langle v, w\rangle \in E : l(v, w) \leq \Delta\}$;
- $d(v) \longleftarrow \infty$;

relax(s, 0);
$i \longleftarrow 0$;
while *B is not empty* **do**
- $S \longleftarrow \phi$;
- **while** $B[i] \neq \phi$ **do**
  - $Req \longleftarrow \{(w, d(v) + l(v, w)) : v \in B[i] \wedge \langle v, w\rangle \in light(v)\}$;
  - $S \longleftarrow S \cup B[i]$;
  - $B[i] \longleftarrow \phi$;
  - **foreach** $(v, x) \in Req$ **do**
    - relax(v, x);
- $Req \longleftarrow \{(w, d(v) + l(v, w)) : v \in S \wedge \langle v, w\rangle \in heavy(v)\}$;
- **foreach** $(v, x) \in Req$ **do**
  - relax(v, x);
- $i \longleftarrow i + 1$;

foreach $v \in V$ **do**
- $\delta(v) \longleftarrow d(v)$;

S: all nodes evaluated in the i-th bucket

B[i]: nodes to be evaluated next in the i-th bucket

Req: Set of pairs (node, value)

Execute in Parallel for light edges

Repeat until no reinsertion

Execute in Parallel for heavy edges

Algorithm (*relax(v,x)*)

Algorithm 2: The *relax* routine in the Δ-stepping algorithm

Input: v, weight request x

Output: Assignment of v to appropriate bucket

if $x < d(v)$ **then**

$\quad B\left[\lfloor d(v)/\Delta \rfloor\right] \leftarrow B\left[\lfloor d(v)/\Delta \rfloor\right] \setminus \{v\};$

$\quad B\left[\lfloor x/\Delta \rfloor\right] \leftarrow B\left[\lfloor x/\Delta \rfloor\right] \cup \{v\};$

$\quad d(v) \leftarrow x;$

Δ-Stepping Algorithm
Definitions and Properties

- Definitions
  - Reinsertion（再確定）：すでに削除されたノードの再追加
  - Rerelaxation（再計算）：再追加ノードによる再Relax
  - d_c: 最短経路の最大値
  - P_Δ: 重みが高々 Δ である経路の集合
  - l_{max}: P_Δ に含まれる経路の中で枝数の最大値
- Properties
  - Bucket数は $\lceil d_c/\Delta \rceil$
  - 再確定数は $|P_\Delta|$以下
  - Phases数は$(d_c/\Delta)l_{max}$以下
  - Randomization解析によるアルゴリズムの性質は Section 3 直前の数行に記載されている(p.4)

Graduate School of Information Science, Nagoya Univ.
Parallel & Distributed Systems Lab.

Δ-Stepping Algorithm

- Fig. 1: 単体プロセッサで最適化されたアルゴリズムとの比較
  – オーバーヘッドがある
- Fig. 2: Light Edgeの計算
  – 並列度が高い for 2^{20} Nodes
- Fig. 3: 性能に関する指標 (Δ=0.25)
  – Shortest Path Weight
  – Phases
  – Buckets
  – Insertions （実験的には20%のオーバーヘッド）
- Fig. 4: Execution Time
  – 40CPUで約30倍

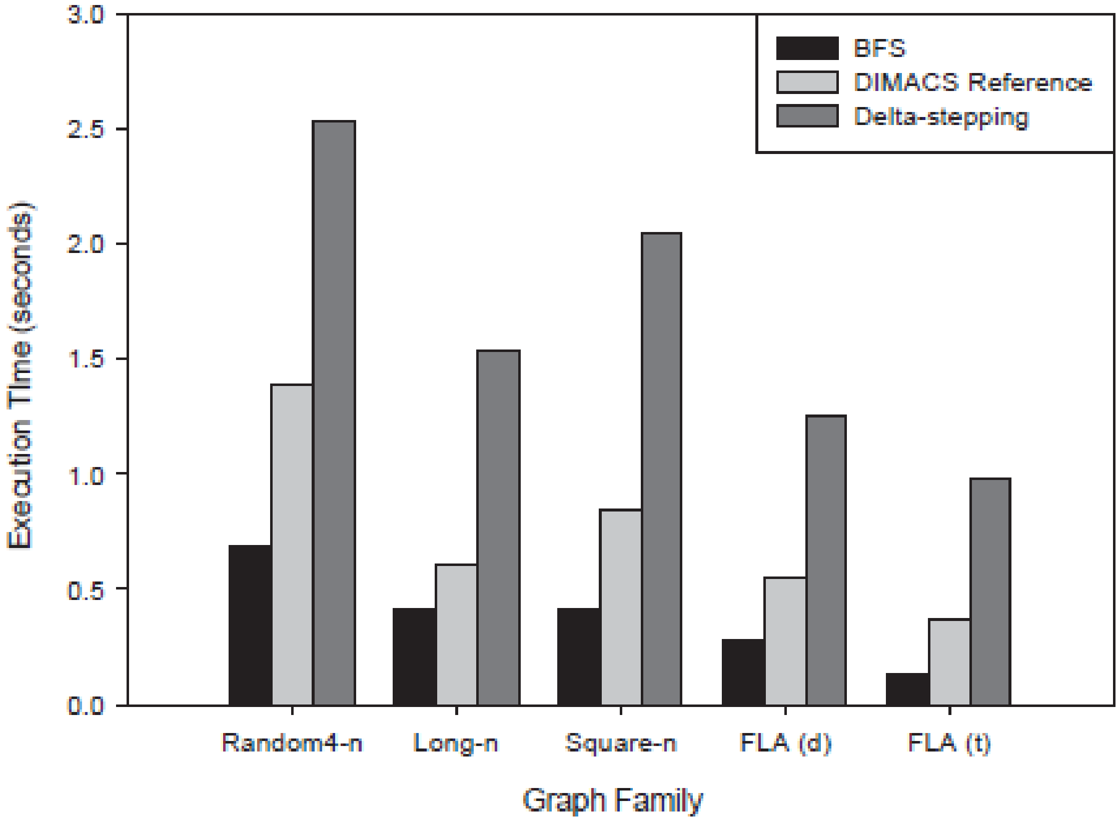
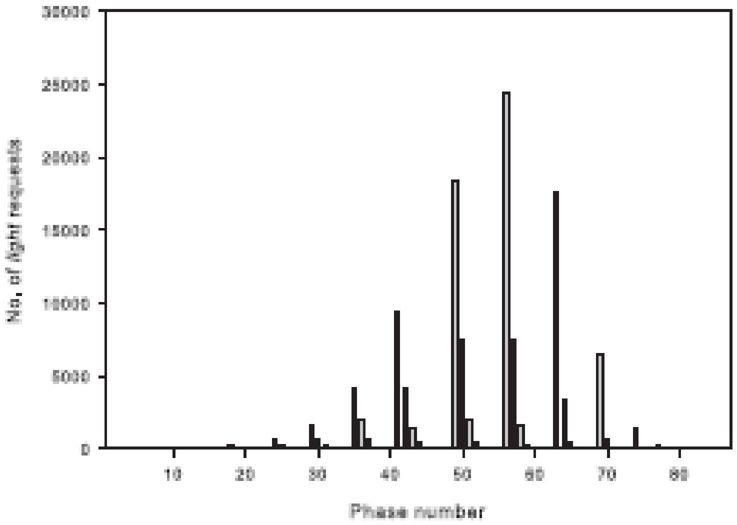
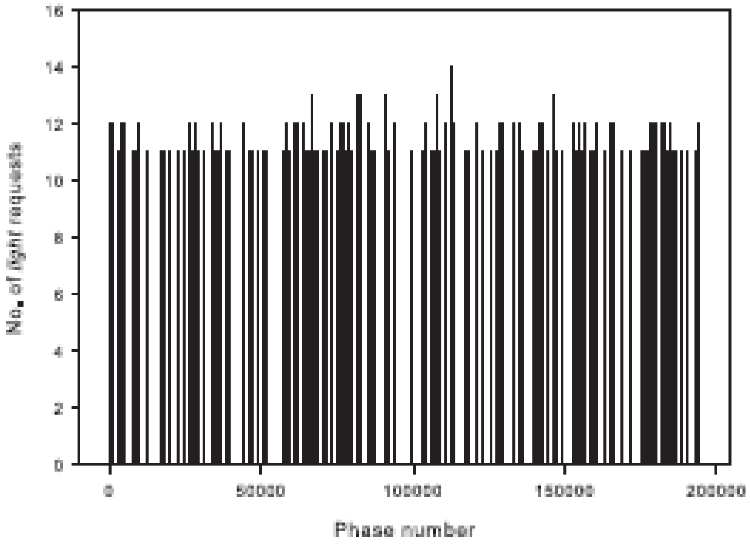


Fig. 1. Sequential performance of our Δ-stepping implementation on the core graph families. All the synthetic graphs are directed, with 2^{20} vertices and $\frac{m}{n} \approx 4$. FLA(d) and FLA(t) are road networks corresponding to Florida, with 1070376 vertices and 2712768 edges

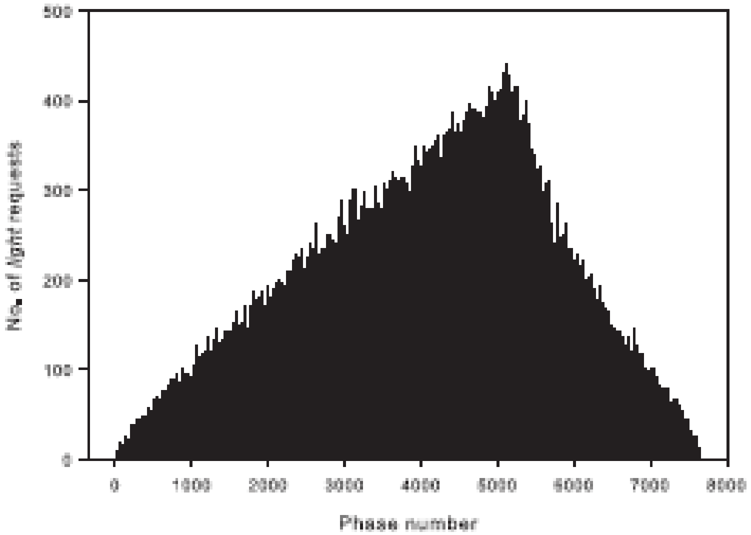
BFS: Breadth First Search (to show ideal parallel performance)



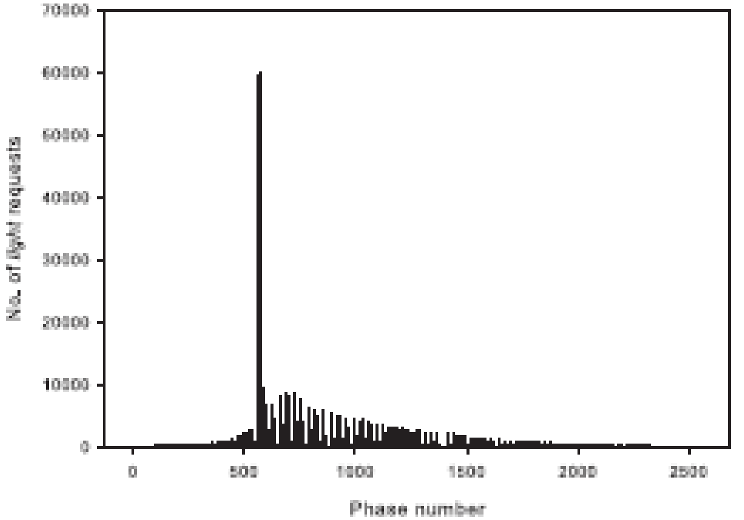
(a) Random4-n family, $n = 2^{20}$.



(b) Long-n family, $n = 2^{20}$.

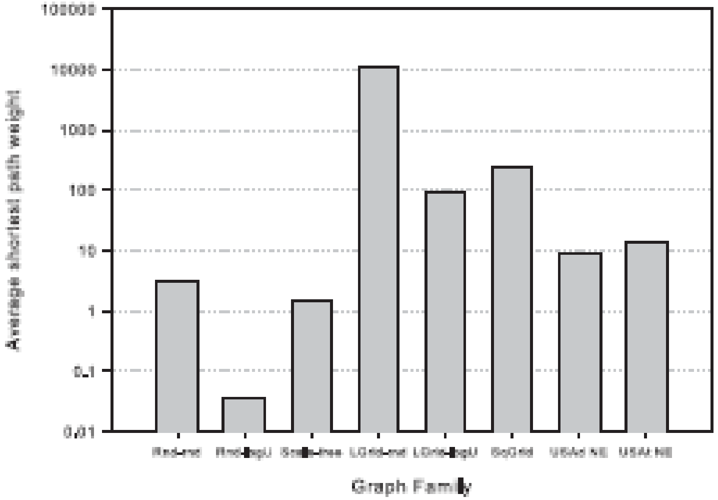


(c) Square-n family, $n = 2^{20}$.

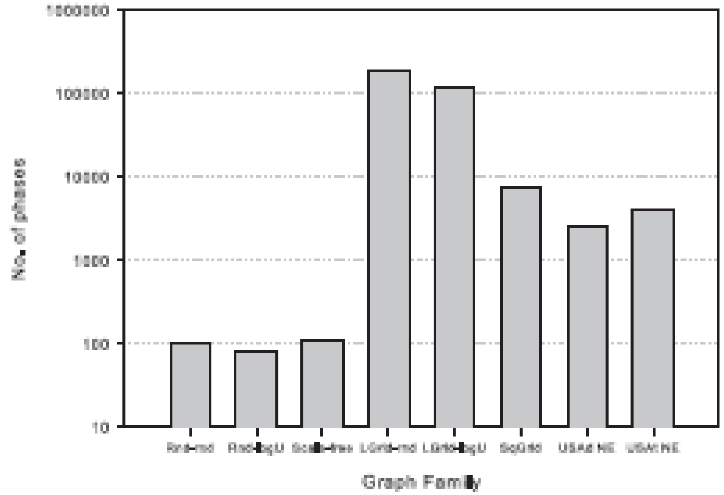


(d) USA-road-d family, Northeast USA (NE). $n = 1524452$, $m = 3897634$.

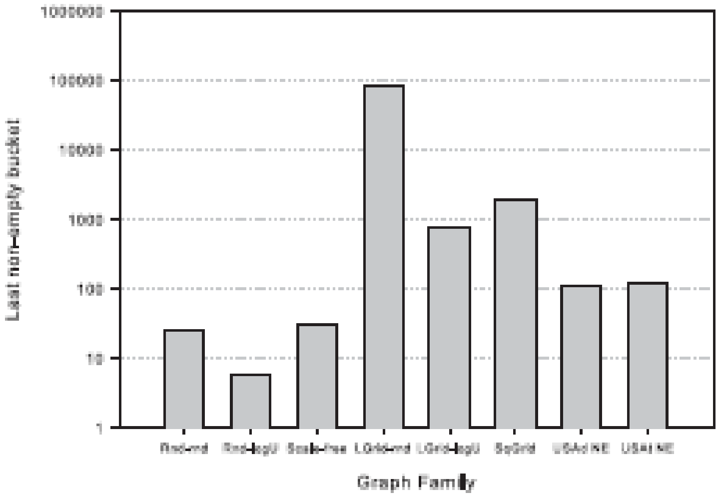
Fig. 2. Δ-stepping algorithm: Size of the light request set at the end of each phase, for the core graph families. Request set sizes less than 10 are not plotted.



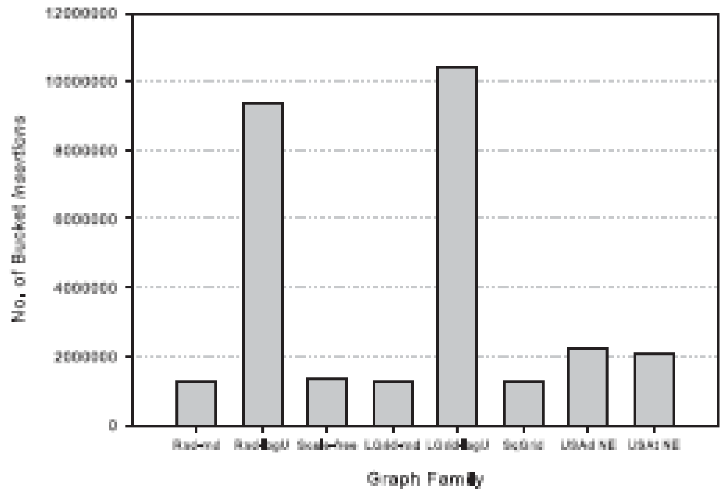
(a) Average shortest path weight $(\frac{1}{n} * \sum_{v \in V} \delta(v))$



(b) No. of phases

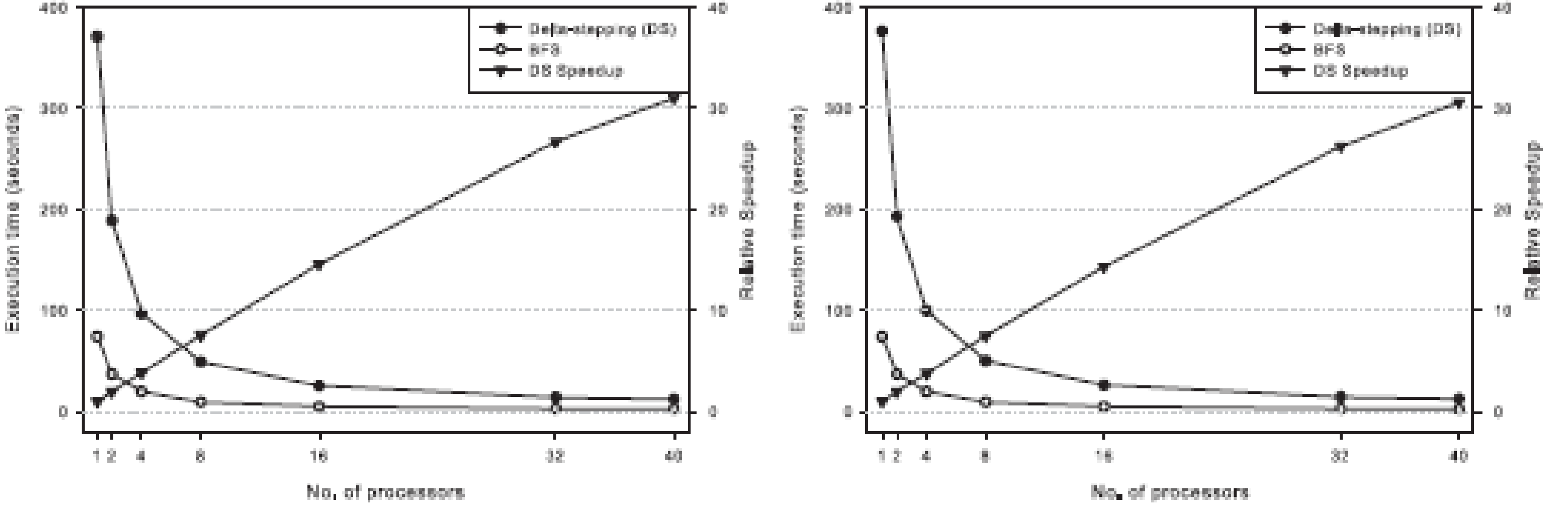


(c) Last non-empty bucket



(d) Number of relax requests

Fig. 3. Δ-stepping algorithm performance statistics for various graph classes. All synthetic graph instances have n set to 2^{20} and $m \approx 4n$. Rand-rnd: Random graph with random edge weights, Rand-logU: Random graphs with log-uniform edge weights, Scale-free: Scale-free graph with random edge weights, Lgrid: Long grid, SqGrid: Square grid, USA NE: 1524452 vertices, 3897634 edges. Plots (a), (b), (c) are on a log scale, while (d) uses a linear scale.



(a) (b)

Fig. 4. Δ-stepping execution time and relative speedup on the MTA-2 for Random4-n (left) and ScaleFree4-n (right) graph instances (directed graph, n=2^{28} vertices and $m = 4n$ edges, random edge weights).