A Forward-Backward Single-Source Shortest Paths Algorithm

"Single-Source Shortest Paths in O(n) time"

David Wilson – Microsoft Research
Uri Zwick – Tel Aviv Univ.

Deuxièmes journées du GT CoA
Complexité et Algorithmes

19-20 novembre
Université Paris Diderot (LIAFA)

Single-Source Shortest Paths in weighted directed graphs with non-negative edge weights

Worst case result

Dijkstra (1959)	$m + n \log n$

Arbitrary graph, random $U[0,1]$ edge weights

Meyer (2003) Goldberg (2008) Hagerup (2006)	$m + n$

All-Pairs Shortest Paths in weighted directed graphs

Worst case results

$n \times$ Dijkstra	$mn + n^2 \log n$
Pettie (2004)	$mn + n^2 \log \log n$
Chan (2007)	$\approx \quad n^3 / \log^2 n$

All-Pairs Shortest Paths in randomly weighted complete directed graphs

Expected running times (some with high probability)

Spira (1973)	$n^2 \log^2 n$
[illegible] (1982)	$n^2 \log n \log^* n$
Demetrescu-Italiano (2004)	$n^2 \log n$
Peres-Sotnikov-Sudakov-Z (2010)	n^2

First three results hold in the *endpoint independent* model

Single-Source Shortest Paths in randomly weighted complete directed graphs with sorted adjacency lists

Using only out-going adjacency lists

Using both out-going and in-coming adjacency lists

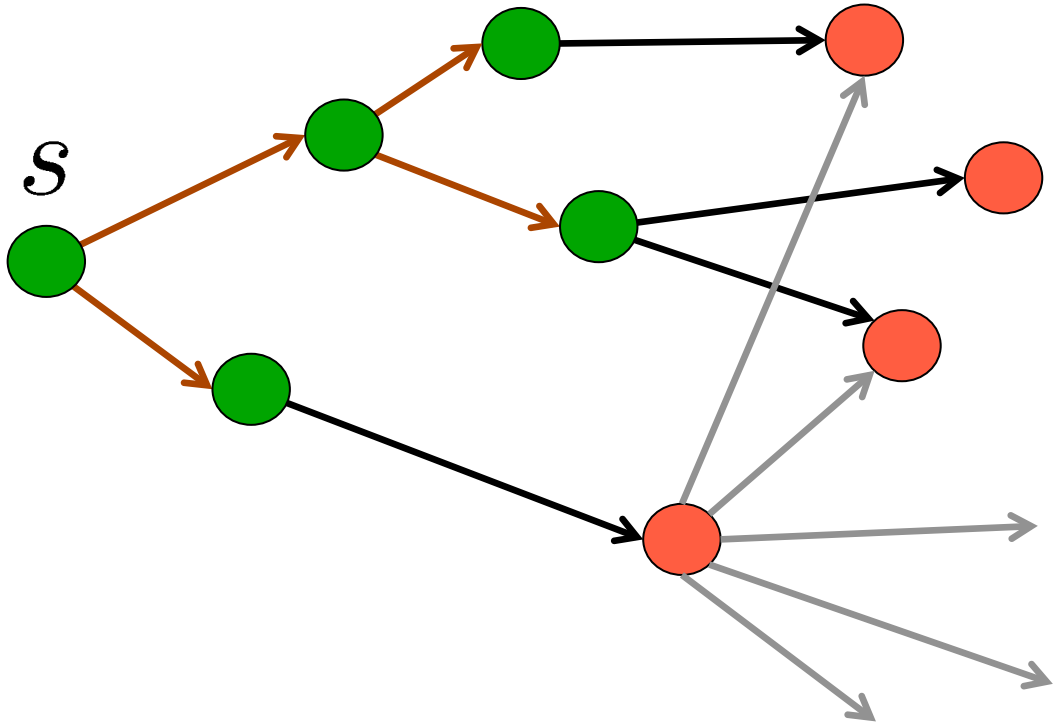
S	$O(n \log^2 n)$
Bloniarz (1983)	$O(n \log n \log^* n)$
[illegible]7)	$O(n \log n)$
[illegible]7)	
[illegible]7)	$\Omega(n \log n)$
Wilson-Z (2013)	$O(n)$

Endpoint independent model
[Spira (1973)]

For each vertex v use an arbitrary process to generate n-1 edge weights

Randomly permute the n-1 edge weights and assign them to the out-going edges of v

Dijkstra's algorithm



Priority-queue
holds vertices

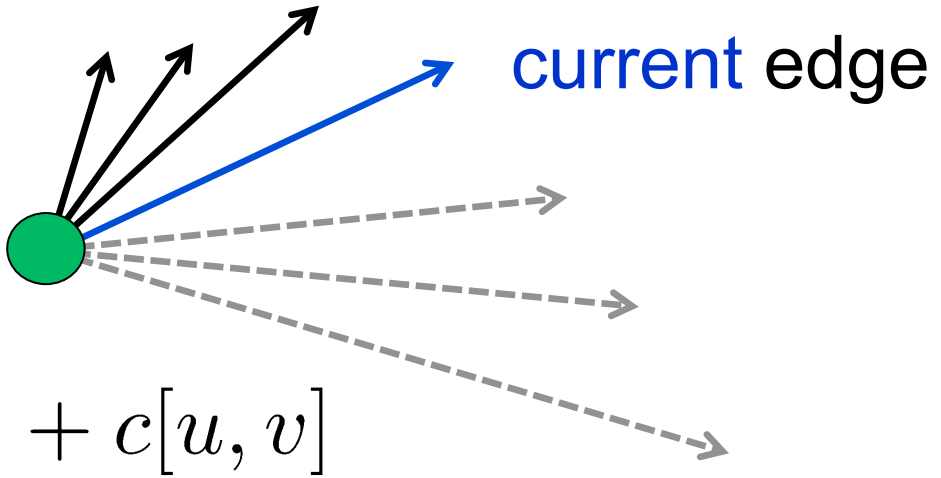
When a vertex is "settled", relax all its out-going edges

Spira's algorithm
[Spira (1973)]

Lazy version of Dijkstra's algorithm

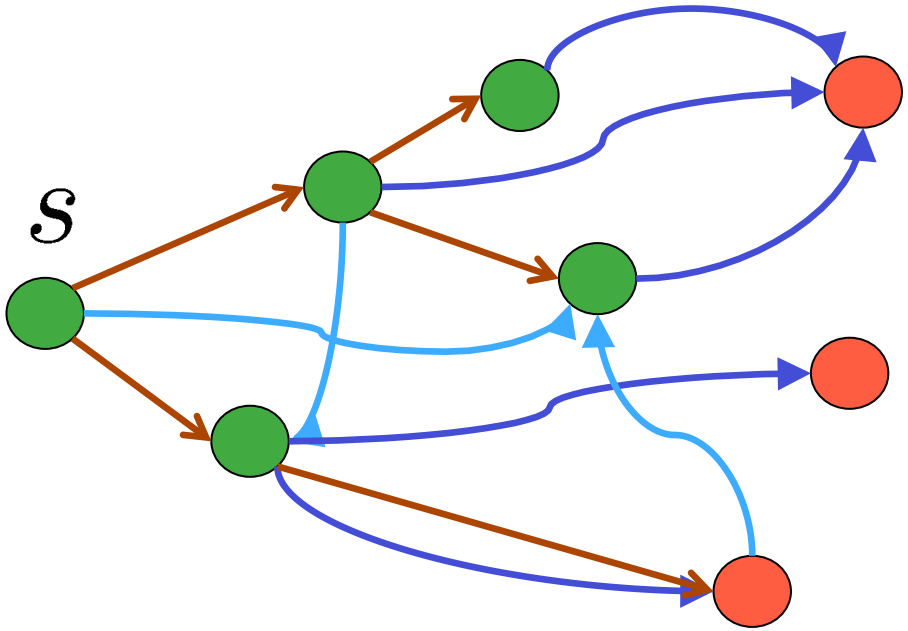
Uses sorted out-going adjacency lists

Edges are examined one at a time



$$key[u,v] \;=\; d[u] + c[u,v]$$

Spira's algorithm
[Spira (1973)]



Priority-queue holds current edges

After extracting an edge, examine the next out-going edge

Fewer edges examined

More extractions from priority queue

Algorithm `Spira`$(G = (V, Out, c), s)$

$S \leftarrow \{s\}$; $P \leftarrow \varnothing$

foreach $v \in V$ **do**
    $d[v] \leftarrow \infty$
    $p[v] \leftarrow \perp$
    `reset`$(Out[v])$

$d[s] \leftarrow 0$
`forward`(s)

while $S \neq V$ `and` $P \neq \varnothing$ **do**
    $(u, v) \leftarrow$ `extract-min`(P)
    `forward`(u)
    if $v \notin S$ **then**
        // New distance found
        $d[v] \leftarrow d[u] + c[u, v]$
        $p[v] \leftarrow u$
        $S \leftarrow S \cup \{v\}$
        `forward`(v)

Function `forward`(u)

$v \leftarrow$ `next`$(Out[u])$
if $v \neq \perp$ **then**
    `insert`$(P, (u, v), d[u] + c[u, v])$

Analysis of Spira's algorithm in the endpoint independent model

Suppose that k vertices were found

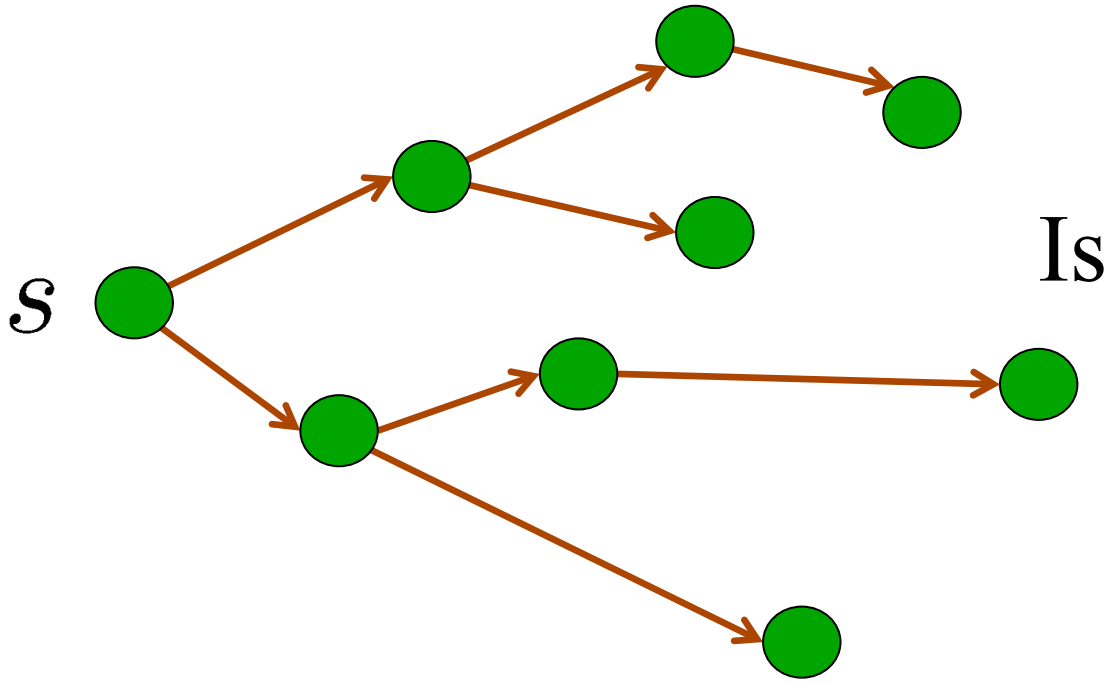
Each edge extracted from PQ has prob. $> (n\text{-}k)/n$ of leading to a new vertex

$$\sum_{k=1}^{n-1} \frac{n}{n-k} = nH_{n-1} \sim n \ln n$$

Sub-linear algorithm

Can it be further improved?

Verification of Shortest Paths Trees

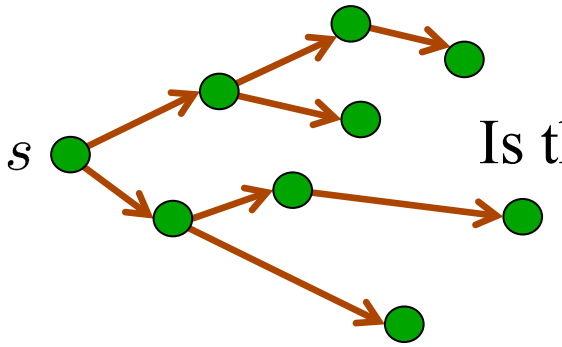


Is this a SPT?

$$d[u] + c[u,v] \;\geq\; d[v] \quad , \quad (u,v) \in E$$

$$c[u,v] \;\geq\; d[v] - d[u] \quad , \quad (u,v) \in E$$

Verification of Shortest Paths Trees



Is this a SPT?

$$D = \max_{v \in V} d[v]$$

For each vertex u, examine
out-going edges, in sorted order, until

$$d[u] + c[u, v] \geq D$$

Essentially optimal!

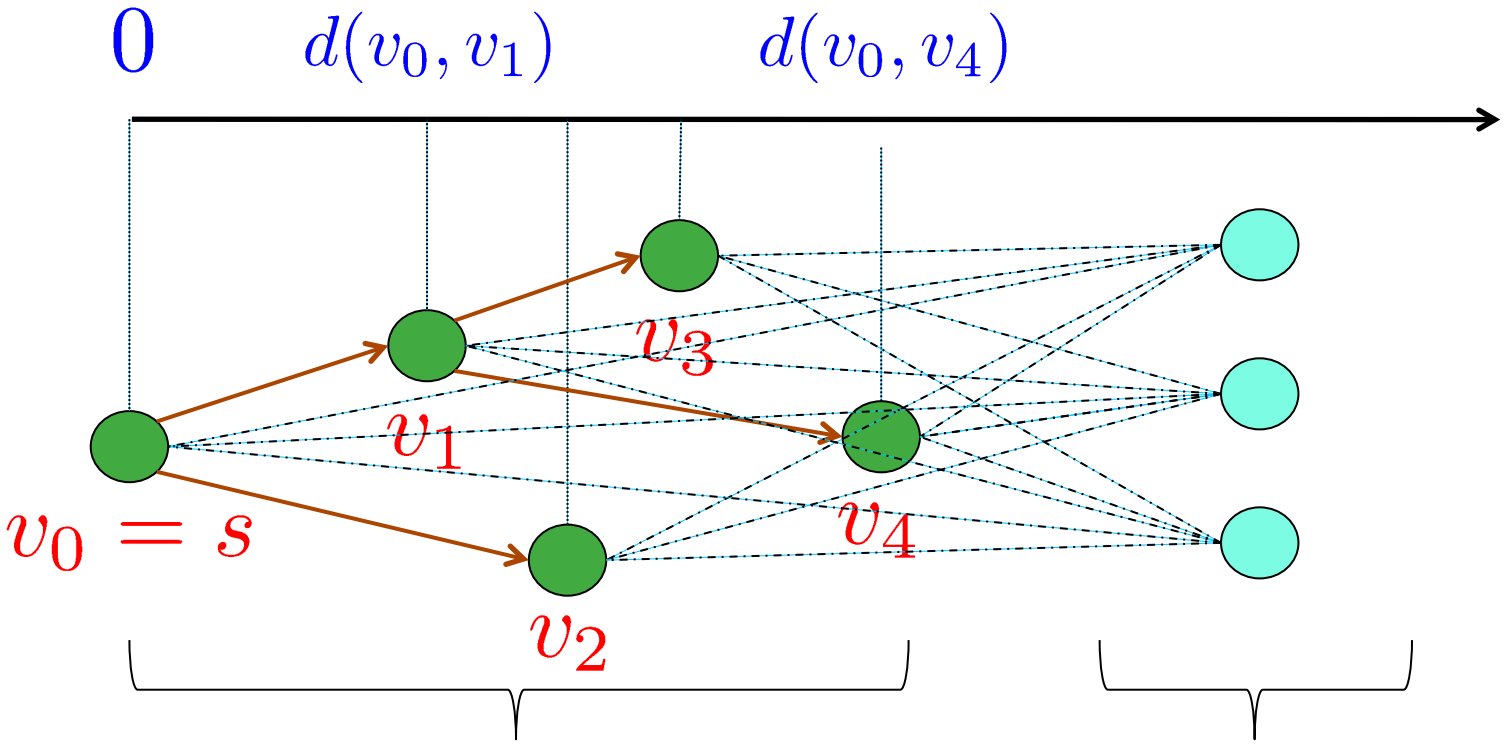
Assuming only out-going adjacency lists available

$\Omega(n \log n)$ for independent edge weights

Could in-coming adjacency lists help?

Distances in a complete graph with independent EXP(1) edge weights

[Davis-Prieditis (1993)] [Janson (1999)]



$$d(v_0, v_k) - d(v_0, v_{k-1}) = \frac{X}{k(n-k)}$$

Distances in a complete graph with independent EXP(1) edge weights

[Davis-Prieditis (1993)] [Janson (1999)]

$$d(v_0, v_k) = \sum_{i=1}^{k} \frac{X_i}{i(n-i)} \quad , \quad X_i \sim \mathrm{EXP}(1)$$

$$\mathbb{E}[d(v_0, u)] = \frac{1}{n-1} \sum_{k=1}^{n-1} \sum_{i=1}^{k} \frac{1}{i(n-i)} = \frac{H_{n-1}}{n-1}$$

$$\mathbb{E}[d(v_0, v_{n-1})] = \sum_{i=1}^{n-1} \frac{1}{i(n-i)} = \frac{1}{n} \sum_{i=1}^{n-1} \left(\frac{1}{i} + \frac{1}{n-i} \right) = \frac{2H_{n-1}}{n-1}$$

Distances in a complete graph with independent EXP(1) edge weights

[Janson (1999)]

$$\mathbb{E}[d(v_0, u)] = \frac{H_{n-1}}{n-1} \sim \frac{\ln n}{n}$$

$$\mathbb{E}[d(v_0, v_{n-1})] = \frac{2H_{n-1}}{n-1} \sim \frac{2\ln n}{n}$$

$$n\, d(v_0, u) - \ln n \xrightarrow{d} W_1 + W_2 - W_3$$

$$n\, d(v_0, v_{n-1}) - 2\ln n \xrightarrow{d} W_1 + W_2$$

$$\mathbb{P}[W_i \le x] = e^{-e^{-x}} \quad \text{(Gumbel)}$$

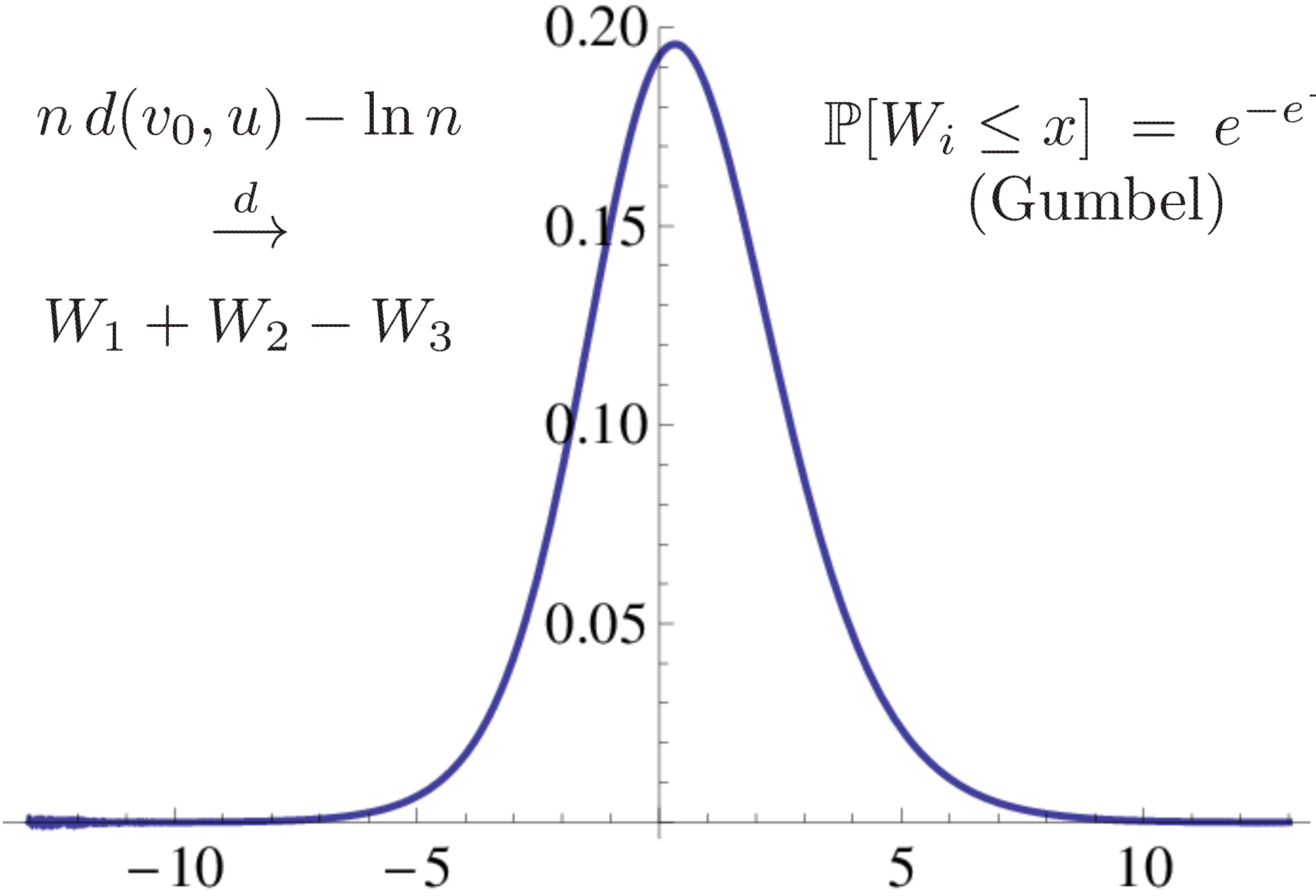
$$n\,d(v_0, u) - \ln n$$

$$\xrightarrow{d}$$

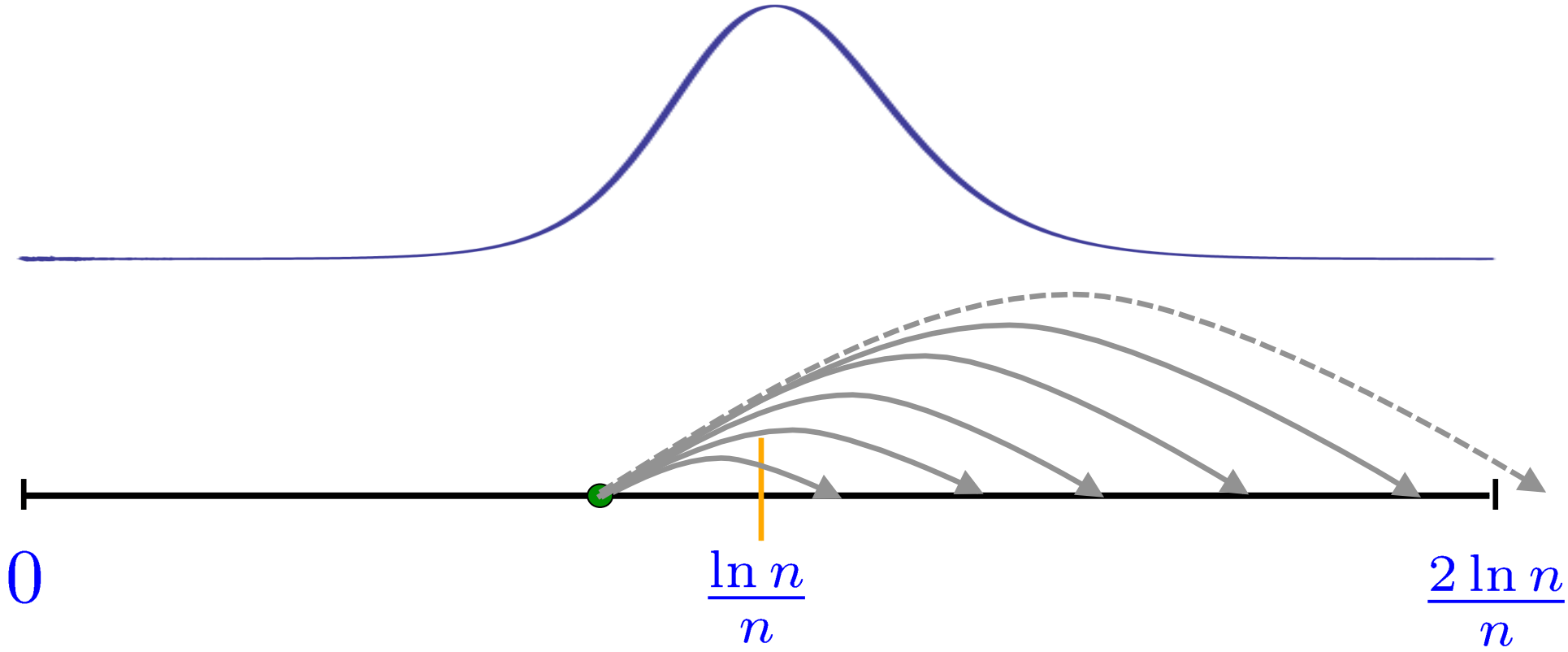
$$W_1 + W_2 - W_3$$

$$\mathbb{P}[W_i \leq x] = e^{-e^{-x}}$$

(Gumbel)

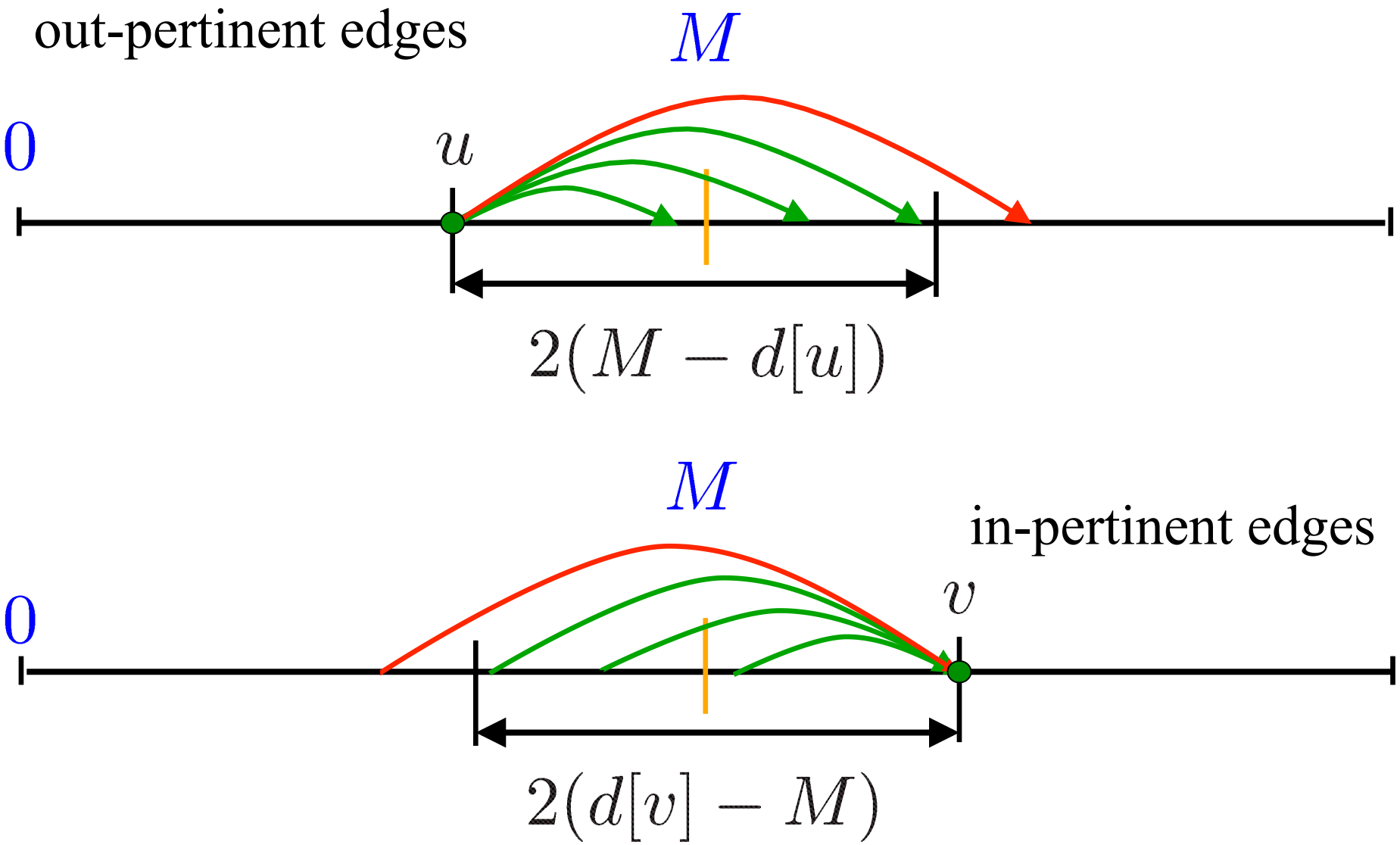


Forward-only verification



$\Omega(n \log n)$ edges need to be examined

Pertinent edges



Forward-backward verification

Check only pertinent edges!

Pertinent edges:

$$c[u,v] \;\leq\; 2(M - d[u]) \;\text{ or }\; c[u,v] \;<\; 2(d[v] - M)$$

For non-pertinent edges:

$$\begin{aligned} c[u,v] &> \tfrac{1}{2}\Big(2(M - d[u]) + 2(d[v] - M)\Big) \\ &= d[v] - d[u] \end{aligned}$$

!

Number of pertinent edges

Expected number of pertinent edges is $\Theta(n)$

Probability that number of pertinent edges is more than $\Theta(n)$ is exponentially small

$$\mathbb{E}[|E_{per}|] \leq 4n$$

$$\mathbb{P}[|E_{per}| \geq 8n + \Delta] \leq (4/5)^{\Delta/2}$$

Number of pertinent edges

Condition on a SPT and distances

$$c[v_i, v_j] = \begin{cases} d[v_j] - d[v_i] & \text{if } (v_i, v_j) \in SPT \\ d[v_j] - d[v_i] + \text{EXP}(1) & \text{if } (v_i, v_j) \notin SPT \text{ and } i < j \\ \text{EXP}(1) & \text{if } (v_i, v_j) \notin SPT \text{ and } i > j \end{cases}$$

$$\text{Out-pertinent: } c[v_i, v_j] \leq 2(M - d[v_i])$$

$$X_{i,j} = [\,(v_i, v_j) \text{ is out-pertinent}\,]$$

$$Y_{i,j} = [\text{EXP}(1) \leq \underbrace{2(M - d[v_i])}_{\lambda_{i,j}}]$$

Number of out-pertinent edges

Comparison with a Poisson process

$$X_{i,j} = [\,(v_i, v_j) \text{ is out-pertinent}\,]$$

$$Y_{i,j} = [\mathrm{EXP}(1) \leq \underbrace{2(M - d[v_i])}_{\lambda_{i,j}}\,]$$

$$\sum X_{i,j} \prec \sum Y_{i,j} \prec \mathrm{Poisson}(\sum \lambda_{i,j})$$

$$\Lambda_{out} = \sum \lambda_{i,j} = 2(n-1) \sum_{k=\lfloor \frac{n}{2} \rfloor + 1}^{n-1} \frac{\mathrm{EXP}(1)}{k} \prec 4 \sum_{i=1}^{n/2} \mathrm{EXP}(1)$$

Forward-backward SSSP algorithm

Goal: Run Spira's algorithm on the subgraph composed of pertinent edges

Run Spira's algorithm until median distance M is found

Out-pertinent edges are easy to identify

How do we identify the in-pertinent edges?

Backward scans used to identify in-pertinent edges

When an in-pertinent edge is found, a request is issued for its forward scan

Algorithm sssp$(G = (V, In, Out, c), s)$

$S \leftarrow \{s\}$; $M \leftarrow \infty$; $n \leftarrow |V|$
$P \leftarrow \varnothing$; $Q \leftarrow \varnothing$

foreach $v \in V$ **do**
  $d[v] \leftarrow \infty$; $p[v] \leftarrow$ `nil`
  $out[v] \leftarrow$ `true` ; $Req[v] \leftarrow \varnothing$
  `reset`$(Out[v])$; `reset`$(Req[v])$
  `reset`$(In[v])$

$d[s] \leftarrow 0$
`forward`(s)

while $S \neq V$ and $P \neq \varnothing$ **do**
  $(u, v) \leftarrow$ `extract-min`(P)
  `forward`(u)
  if $v \notin S$ **then**
    // New distance found
    $d[v] \leftarrow d[u] + c[u, v]$
    $p[v] \leftarrow u$
    $S \leftarrow S \cup \{v\}$
    `forward`(v)
    if $|S| = \lceil n/2 \rceil$ **then**
      $M \leftarrow d[v]$
      foreach $w \notin S$ **do**
        `backward`(w)
  // Find more *in-pertinent* edges
  while $Q \neq \varnothing$ and
  `min`$(Q) < 2\,($`min`$(P) - M)$ **do**
    $(u, v) \leftarrow$ `extract-min`(Q)
    if $v \notin S$ **then**
      `backward`(v)
      `request`(u, v)

Function `forward`(u)

if $out[u]$ **then**
  // find next *out-pertinent* edge
  $v \leftarrow$ `next`$(Out[u])$
  if $v =$ `nil` or
  $c[u, v] > 2\,(M - d[u])$ **then**
    $out[u] \leftarrow$ `false`

if not $out[u]$ **then**
  // find next *in-pertinent* edge
  $v \leftarrow$ `next`$(Req[u])$

$active[u] \leftarrow v \neq$ `nil`
if $active[u]$ **then**
  `insert`$(P, (u, v), d[u] + c[u, v])$

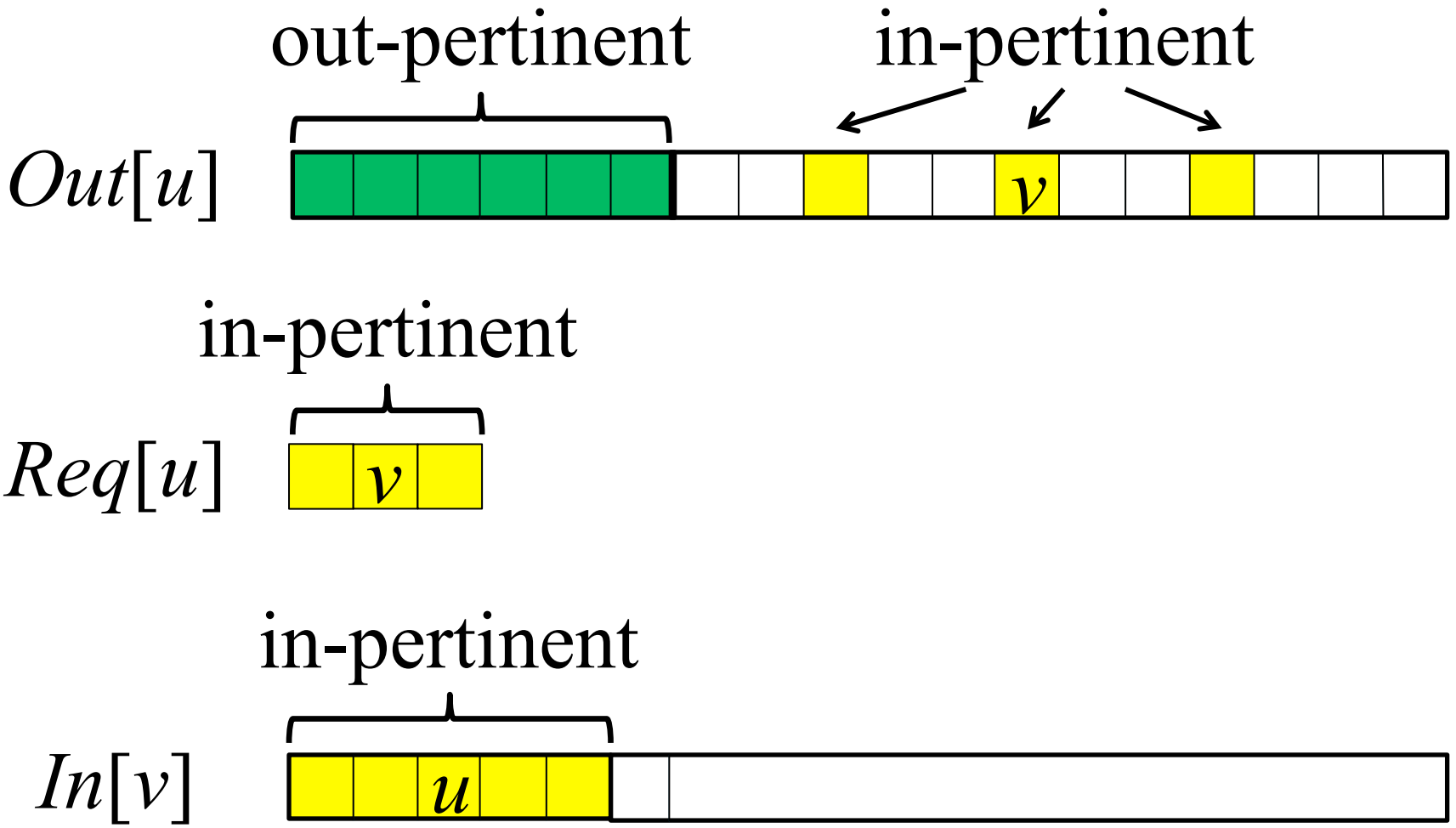
Function `backward`(v)

$u \leftarrow$ `next`$(In[v])$
if $u \neq$ `nil` **then**
  `insert`$(Q, (u, v), c[u, v])$

Function `request`(u, v)

`append`$(Req[u], v)$
if $u \in S$ and not $active[u]$ **then**
  // urgent request
  `forward`(u)

Adjacency lists



Identifying in-pertinent edges

Priority Queue P:

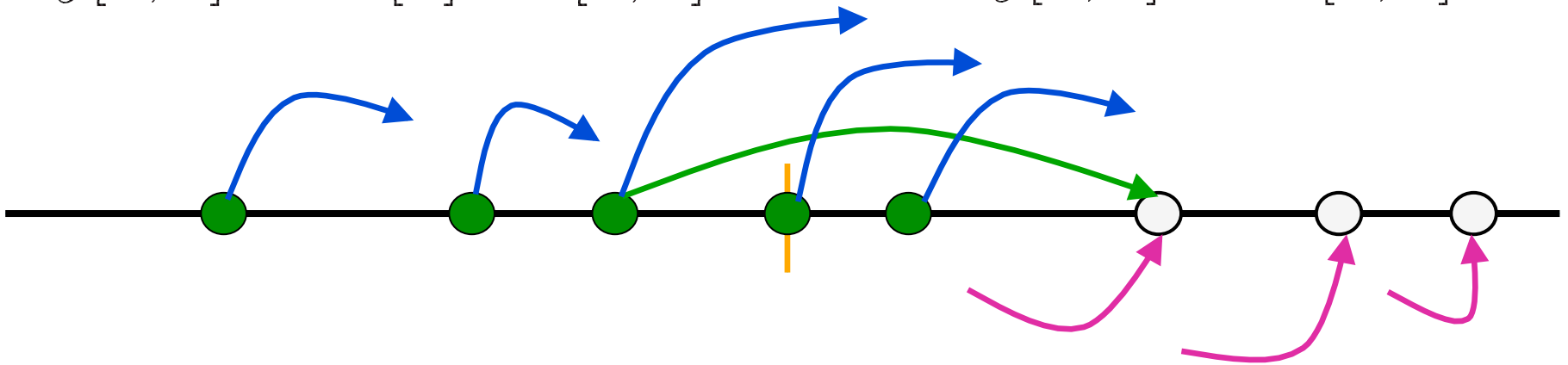
current out-going edges

$key[u,v] \;=\; d[u] + c[u,v]$

Priority Queue Q:

current in-going edges

$key[u,v] \;=\; c[u,v]$



while $Q \neq \varnothing$ and $\texttt{minkey}(Q) < 2\,(\texttt{minkey}(P) - M)$ **do**

- $(u,v) \leftarrow \texttt{extract-min}(Q)$
- **if** $v \notin S$ **then**
  - $\texttt{backward}(v)$
  - $\texttt{request}(u,v)$

`request`(u, v)

((u,v) is an in-pertinent edge)

Append (u,v) to $Req[u]$

If u has no current edge, the request is urgent,
and (u,v) is immediately inserted into P

Identifying in-pertinent edges

Priority Queue P:

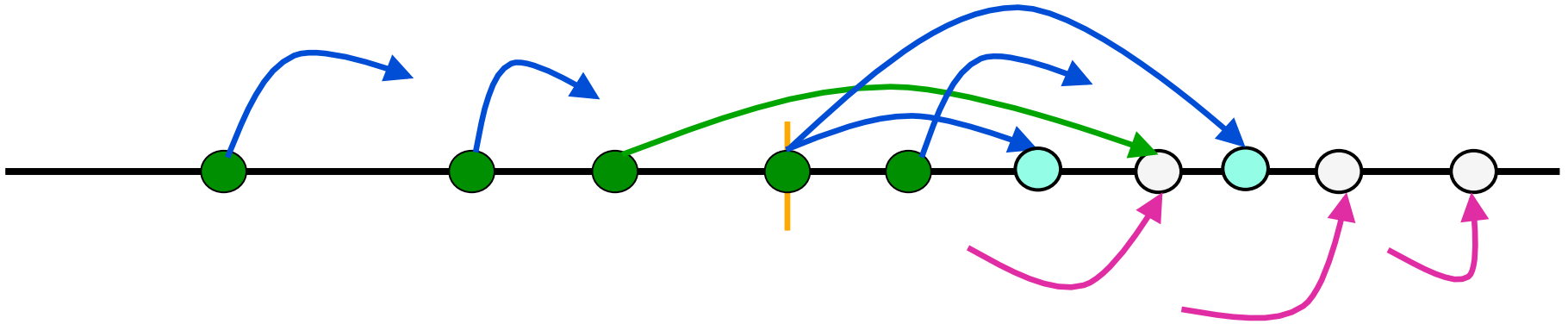
current out-going edges

$key[u,v] \;=\; d[u] + c[u,v]$

Priority Queue Q:

current in-going edges

$key[u,v] \;=\; c[u,v]$



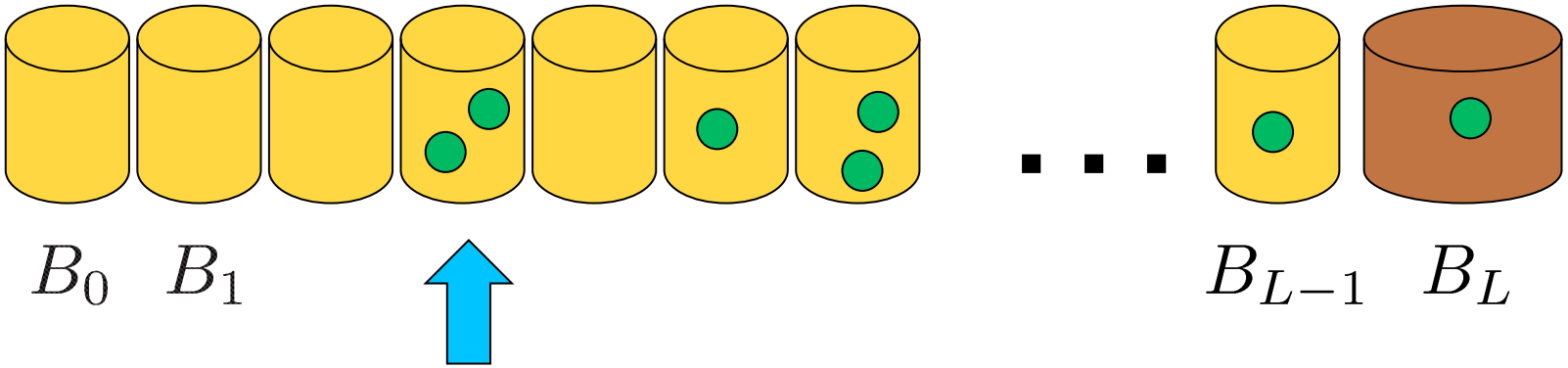
while $Q \neq \varnothing$ and $\texttt{minkey}(Q) < 2\,(\texttt{minkey}(P) - M)$ **do**

- $(u,v) \leftarrow \texttt{extract-min}(Q)$
- **if** $v \notin S$ **then**
  - $\texttt{backward}(v)$
  - $\texttt{request}(u,v)$

Bucket-based priority queues

B_i contains distances in $[iW, (i+1)W)$

$$L = \Theta(n)$$



Two-level Bucket-based priority queues

If a bucket contains k items when it becomes active, split it into k sub-buckets

Store the items in each sub-bucket in a binary heap

Probability of more than $\Theta(n)$ time is

$$\exp(\Theta(-n/\log n))$$

Open problems

More edge weight distributions?
(We already have some extensions)

$n+o(m)$ for arbitrary graphs with random edge weights?

End-point independent model?

Could the new forward-backward algorithm be useful in practical applications?