Undirected Single Source Shortest Paths in Linear Time

Mikkel Thorup
AT&T Labs—Research

Based on:
Mikkel Thorup.Undirected single source shortest paths with positive integer weights in linear time. *Journal of the ACM*, 46(3):362–394, 1999. See also FOCS'97.

SSSP

Weighted graph $G = (V, E)$, $s \in V$, $n = |V|$, $m = |E|$

Find $dist(s, v)$ $\forall v \in V$

This talk: undirected SSSP in deterministic linear time and linear space.

Previously linear time only for planar graphs [Klein, Rao, Rauch, Subramanian, STOC'94]

Since 1959 all theoretical developments for general directed and undirected graphs based on Dijkstra's algorithm

Dijkstra

Super distance $D(v) \geq d(v) = dist(s, v)$

$v \in S \Rightarrow D(v) = d(v)$
$v \notin S \Rightarrow D(v) = \min_{u \in S}\{d(u) + \ell(u, v)\}$

Dijkstra's SSSP algorithm

$S \leftarrow \{s\}$
$D(s) \leftarrow 0$, $\forall v \neq s : D(v) \leftarrow \ell(s, v)$
while $S \neq V$
  pick $v \in V \setminus S$ minimizing $D(v)$
  $\triangleright$ $D(v) = d(v)$
  $S \leftarrow S \cup \{v\}$
  for all $(v, w) \in E$
    $D(w) \leftarrow \min\{D(w), D(v) + \ell(v, w)\}$

Implementations of Dijkstra

$O(m+n^2)$	Dijkstra'59
$O(m \log n)$	William'64
$O(m+n\log n)$	Fredman and Tarjan'87
$O(m\sqrt{\log n})$	Fredman and Willard'93
$O(m+n\frac{\log n}{\log\log n})$	Fredman and Willard'94
$O(m\log\log n)$	Thorup'96
$O(m+n\sqrt{\log n}^{1+\varepsilon})$	Thorup'96
$O(m+n\sqrt[3]{\log n}^{1+\varepsilon})$	Raman'97
$O(m+n\sqrt[3]{\log n}^{1+\varepsilon})$	Raman'97
$O(m\sqrt{\log\log n})$	Han and Thorup'02
$O(m+n\log\log n)$	Thorup'03

$O(m\log\log C)$	van Emde Boas'77
$O(m+n\sqrt{\log C})$	Ahuja et.al.'90
$O(m+n\sqrt[3]{\log C\log\log C})$	Cherkassky et.al.'97
$O(m+n\sqrt[4]{\log C}^{1+\varepsilon})$	Raman'97
$O(m+n\log\log C)$	Thorup'03

Linear Dijkstra $\Longleftrightarrow$ linear sorting, Thorup'96

Still use S, D:

$$v \in S \Rightarrow D(v) = d(v)$$
$$v \notin S \Rightarrow D(v) = \min_{u \in S}\{d(u) + \ell(u, v)\}$$

"visit v" $\equiv$ moving v to S

New: flexible visit sequence, **not** order of $d(v)$

Identify **many other** vertices $v \notin S$ with $D(v) = d(v)$

Note: Dinitz (1978) buckets according to

$$\lfloor D(v) / \min_{e \in E} \ell(e) \rfloor$$

We use hierarchical bucketting structure.

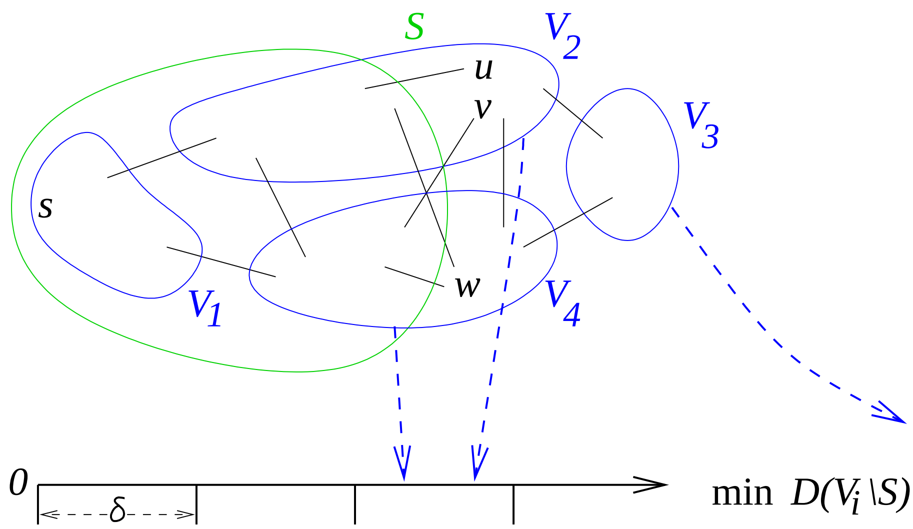
Suppose

- V partitions into $V_1, ..., V_k$
- Edges between different V_i have weight $\geq \delta$
- For some $v \in V_i \setminus S$,

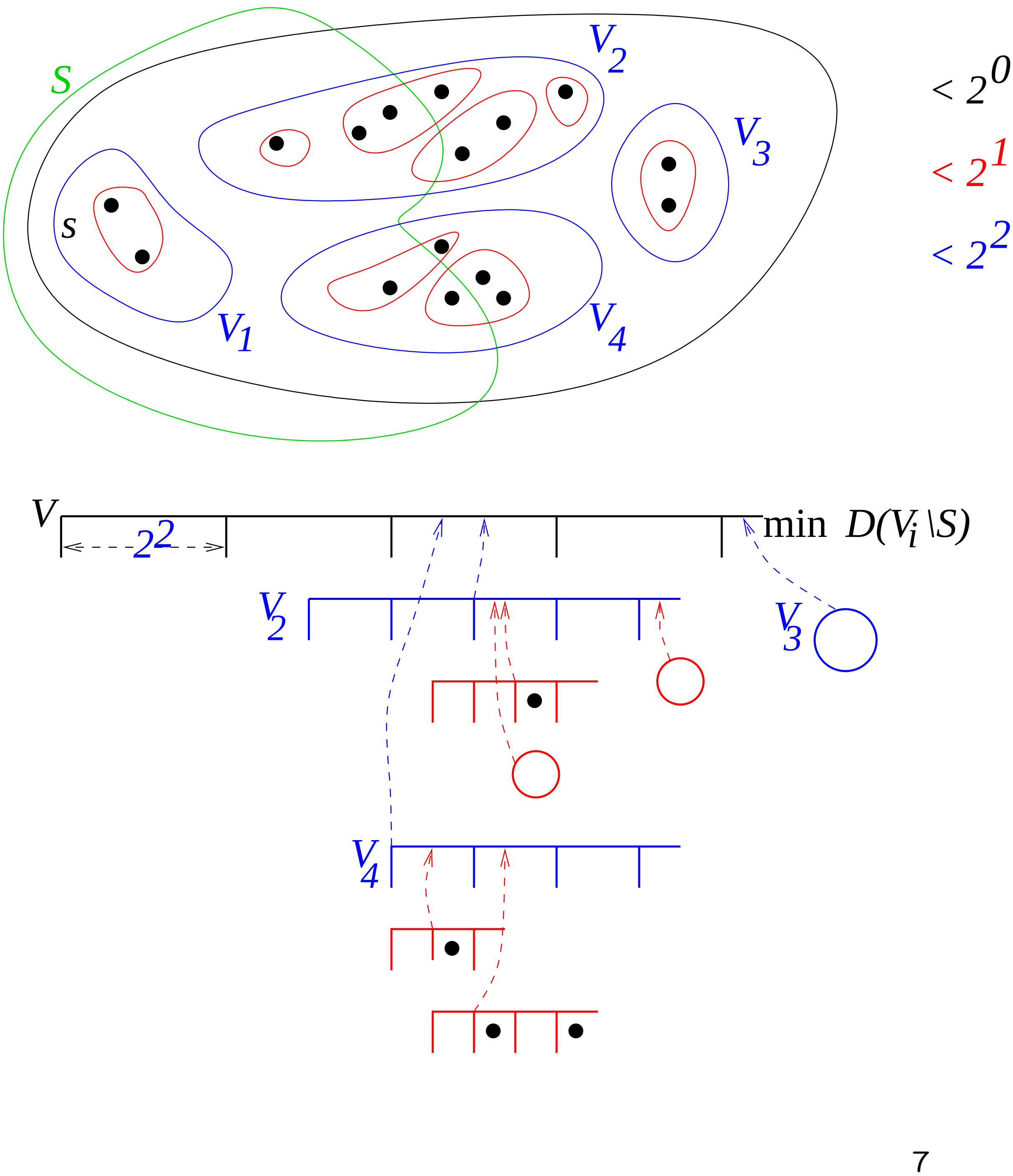
$$D(v) = \min D(V_i \setminus S) \leq \min_j D(V_j \setminus S) + \delta$$

Then

$$d(v) = D(v)$$



A recursive version



Component Hierarchy

$G_i = (V, \{e \in E | \ell(e) < 2^i\})$

$[v]_i$: component of v in G_i
$\equiv$ "level i component of v"

(notation: $x \downarrow i \equiv \lfloor x/2^i \rfloor \equiv$ "x drop i")

Observation $u \notin [v]_i, dist(u,v) \geq 2^i$

(notation: $[v]_i^- = [v]_i \setminus S$)

$[v]_i$ min-child $[v]_{i+1}$ if

$$\min D([v]_i^-) \downarrow i = \min D([v]_{i+1}^-) \downarrow i$$

$[v]_i$ minimal if $\forall j \geq i : [v]_j$ min-child $[v]_{j+1}$

Lemma $[v]_i$ minimal $\Rightarrow$ $\min D([v]_i^-) = \min d([v]_i^-)$

Corollary $[v]_0$ minimal $\Rightarrow$ $D(v) = d(v)$

Component hierachy only stores components with multiple children

—don't store $[v]_i$ if $[v]_{i-1} = [v]_i$.

—at most $2n - 1$ nodes in hierachy.

Component hierachy computed in linear time via minimum spanning tree

Some clusters $[v]_i$ are **expanded**:

- Children clusters stored in buckets $B\langle [v]_i\ ,\ \cdot \rangle$.

- Child $[v]_h$ stored in

$$B\Big\langle [v]_i\ ,\ \min D([v]_h^-) \downarrow (i-1) \Big\rangle$$

  unless $[v]_h^- = \emptyset$.

- Maintain index

$$ix\langle [v]_i \rangle = \min D([v]_i^-) \downarrow (i-1)$$

  of first non-empty bucket.

- min-children in $B\langle [v]_i\ ,\ ix\langle [v]_i \rangle \rangle$.

$[v]_i$ expandable if minimal and parent expanded

No vertex in $[v]_i$ visited yet so $[v]_i^- = [v]_i$

Expanding $[v]_i$

$ix\langle [v]_i \rangle \leftarrow \min D([v]_i) \downarrow i-1$
for all children $[w]_h$ of $[v]_i$,
put $[w]_h$ in $B\langle [v]_i\, , \min D([w]_h) \downarrow (i-1) \rangle$

We shall later see...

A data structure maintains $\min D([w]_h)$ for all unexpanded roots, i.e., unexpanded children of expanded clusters.

The total number of buckets needed is linear.

Visiting a vertex

v visitable if $[v]_0$ minimal and parent expanded

Visiting v

$\triangleright$ $D(v) = d(v)$
for all $(v, w) \in E$
  $D(w) \leftarrow \min\{D(w), D(v) + \ell(v, w)\}$
  update bucket of unexpanded root of w
$S \leftarrow S \cup \{v\}$
$\triangleright$ updating expanded bucket structure
let i be maximal level such that $[v]_i^- = \emptyset$
let $[v]_j$ be parent of $[v]_i$
remove $[v]_i$ from $B\Big\langle [v]_j \text{ , } ix\Big\langle v_j \Big\rangle\Big\rangle$
loop
  exit if $B\Big\langle [v]_j \text{ , } ix\Big\langle [v]_j \Big\rangle\Big\rangle \neq \emptyset$
  $ix\Big\langle [v]_j \Big\rangle \leftarrow ix\Big\langle [v]_j \Big\rangle + 1.$
  let $[v]_k$ be parent of $[v]_j$
  exit if $ix\Big\langle [v]_j \Big\rangle \downarrow (k - j) = ix\langle [v]_k \rangle$
  move $[v]_j$ to $B\langle [v]_k \text{ , } ix\langle [v]_k \rangle + 1\rangle$
  $j \leftarrow k$

Work in bucket structure proportional to number of buckets.

Not too many buckets

$\max d([v]_i) - \min d([v]_i) \leq \sum_{e \in [v]_i} \ell(e)$

so allocate

$$
\begin{aligned}
&|B\langle [v]_i, \cdot\rangle\,| \\
&= |\{\min d([v]_i) \downarrow i-1, \ldots, \max d([v]_i) \downarrow i-1\}| \\
&\leq 2 + \textstyle\sum_{e \in [v]_i} \ell(e)/2^{i-1}
\end{aligned}
$$

Thus

$$
\begin{aligned}
&|B(\cdot,\cdot)| \\
&\leq \textstyle\sum_{[v]_i} (2 + \sum_{e \in [v]_i} \ell(e)/2^{i-1}) \\
&< 4n + \textstyle\sum_e \sum_{[v]_i \ni e} \ell(e)/2^{i-1} \\
&< 4n + \textstyle\sum_e \sum_{i \geq h} \ell(e)/2^{i-1}, \text{ where } 2^h > \ell(e) \\
&< 4n + \textstyle\sum_e \sum_{j \geq 0} 2^{1-j} \\
&< 4n + \textstyle\sum_e 4 \\
&= 4n + 4m \\
&= O(m)
\end{aligned}
$$

For each unexpanded root $[v]_i$, maintain min $D[v]_i$.

Formulated as independent data structure:

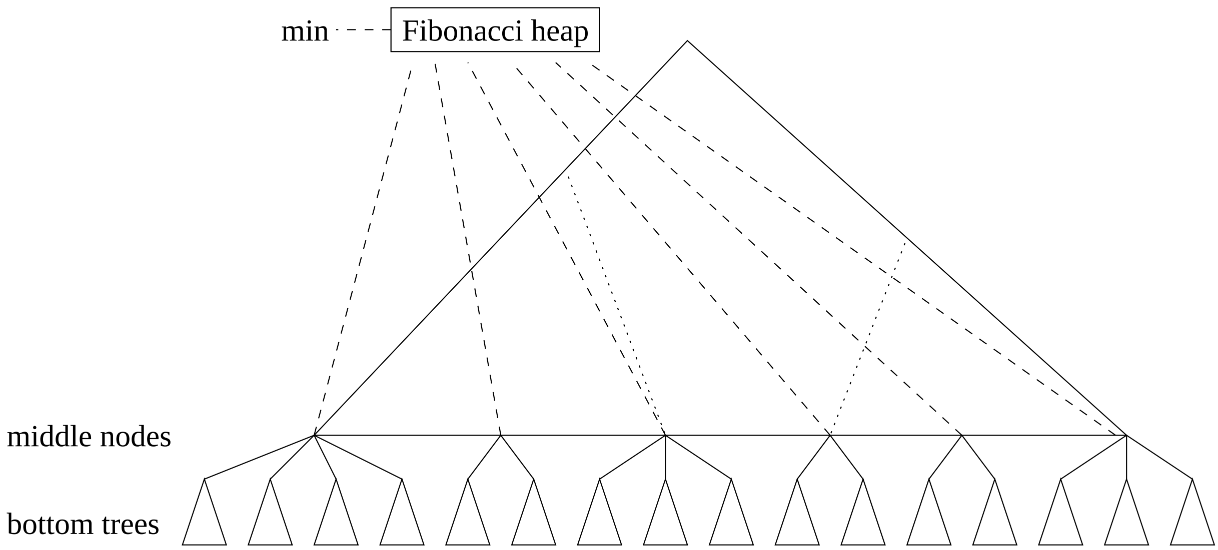
- We have a forest of rooted trees.
- Each leaf w has a key $D(w)$.
- The root has min key of descending leaves.
- The key of a leaf may decrease.
- A root may be deleted.

bottom trees are maximal with $< \log^2 n$ leaves.

bottom trees are handled recursively

above bottom are $\leq n/\log^2 n$ middle nodes.

decrease→bottom root→middle→Fibonacci heap



When root deleted, bigger subtree inherits Fibonacci heap

After two recursions: size $O(\log\log^2 n)$.
Then atomic heaps with tabulation.

Now all updates in constant time.

Summing up

- Computing the component hierachy takes linear time.

- The data structure allows us in constant time to move unexpanded roots when a key is decreased.

- The bucketting of expanded components is maintained in constant time per bucket and the number of buckets is linear.

Thus undirected SSSP solved in linear time.

Concluding remarks

- People have implemented simpler variants. If the component hierachy has been constructed once for the whole graph, subsequent USSSP compuations are fast in practice.

- Basid ideas reused for the best external memory USSSP.

- **Main open problem** do directed SSSP in linear time... Hagerup has done some nice generalizations for directed graph, but lost the linear time.

Exercises for undirected SSSP

- How quickly can you construct component hierachy?

- Solve independent data structures problem for trees of size $O(\log\log^2 n)$ using tables and atomic heaps (free rank queries within set of size $O(\log\log^2 n)$ while items decreased).

- Why doesn't this work for immediately directed graphs?

- Discuss simpler implementation, e.g., not using atomic heaps, and what happens to the asymptotic running time.