Approximate distances in weighted graphs

T. Kavitha
Indian Institute of Science, Bangalore.

(Joint work with Surender Baswana)

The problem

- G : a weighted undirected graph with m edges and n vertices

The problem

- G : a weighted undirected graph with m edges and n vertices

- the APSP problem: compute shortest paths/distances between each pair of vertices.

The problem

- G : a weighted undirected graph with m edges and n vertices

- the APSP problem: compute shortest paths/distances between each pair of vertices.

- build a data structure to answer shortest path/distance queries efficiently.

The problem

- can do it in $\tilde{O}(mn)$ time to produce an $n \times n$ matrix containing distances.

The problem

- can do it in $\tilde{O}(mn)$ time to produce an $n \times n$ matrix containing distances.

- build a *smaller* data structure to answer *approximate* distance queries efficiently.

The problem

- can do it in $\tilde{O}(mn)$ time to produce an $n \times n$ matrix containing distances.

- build a *smaller* data structure to answer *approximate* distance queries efficiently.

- $\hat{\delta}(u,v)$ is a t-stretch estimate of $\delta(u,v)$ if

$$\delta(u,v) \le \hat{\delta}(u,v) \le t\delta(u,v)$$

Data structures for 3-stretch dist.

Data structures for 3-stretch dist.

- of size $O(n^2)$ in time $O(n^2 \log n)$ [Cohen-Zwick]

Data structures for 3-stretch dist.

- of size $O(n^2)$ in time $O(n^2 \log n)$ [Cohen-Zwick]
- of size $O(n^{3/2})$ in expected time $O(m\sqrt{n})$ [Thorup-Zwick]

Data structures for 3-stretch dist.

- of size $O(n^2)$ in time $O(n^2 \log n)$ [Cohen-Zwick]
- of size $O(n^{3/2})$ in expected time $O(m\sqrt{n})$ [Thorup-Zwick]
- *new result:* of size $O(n^{3/2})$ in expected time
  $$O(\min(n^2 \log n, m\sqrt{n}))$$

Data structures for 3-stretch dist.

- of size $O(n^2)$ in time $O(n^2 \log n)$ [Cohen-Zwick]
- of size $O(n^{3/2})$ in expected time $O(m\sqrt{n})$ [Thorup-Zwick]
- *new result:* of size $O(n^{3/2})$ in expected time
$$O(\min(n^2 \log n, m\sqrt{n}))$$
- however, our query answering time is $O(\log n)$.

Approximate Distance Oracles

- a data structure of size $O(kn^{k+1/k})$ constructed in expected time $O(kmn^{1/k})$
  - reports $(2k-1)$-stretch distances in $O(k)$ time.

Approximate Distance Oracles

- a data structure of size $O(kn^{k+1/k})$ constructed in expected time $O(kmn^{1/k})$
  - reports $(2k-1)$-stretch distances in $O(k)$ time.

new result:

- a data structure of size $O(kn^{k+1/k})$ constructed in expected time $O(\min(n^2, kmn^{1/k}))$

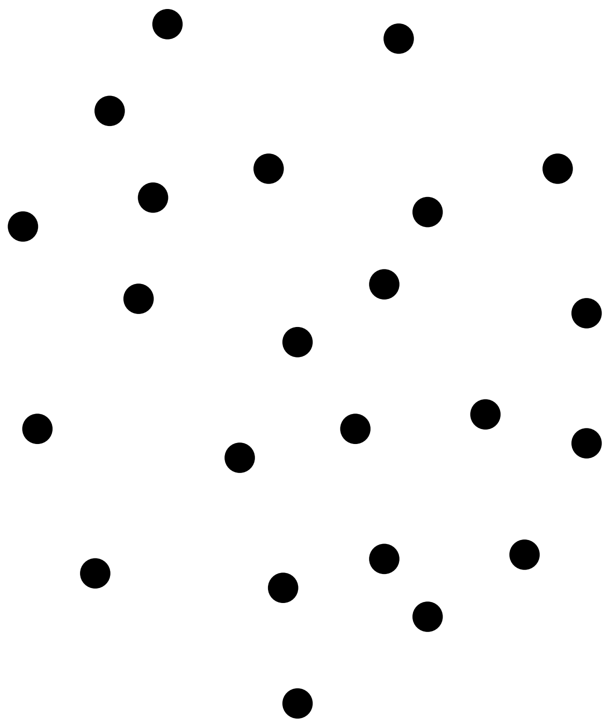
Approximate Distance Oracles

- a data structure of size $O(kn^{k+1/k})$ constructed in expected time $O(kmn^{1/k})$
  - reports $(2k-1)$-stretch distances in $O(k)$ time.

new result:

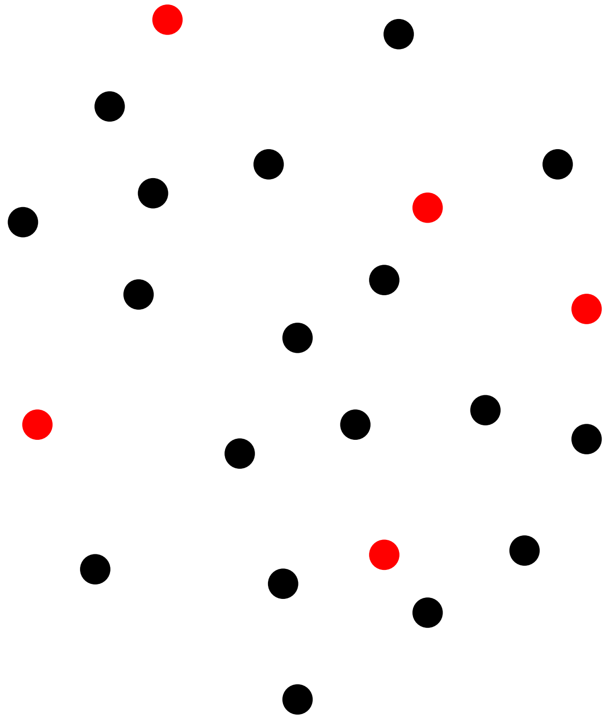
- a data structure of size $O(kn^{k+1/k})$ constructed in expected time $O(\min(n^2, kmn^{1/k}))$
  - reports $(2k-1)$-stretch distances in $O(k)$ time, for any $k > 2$.

The Thorup-Zwick algorithm

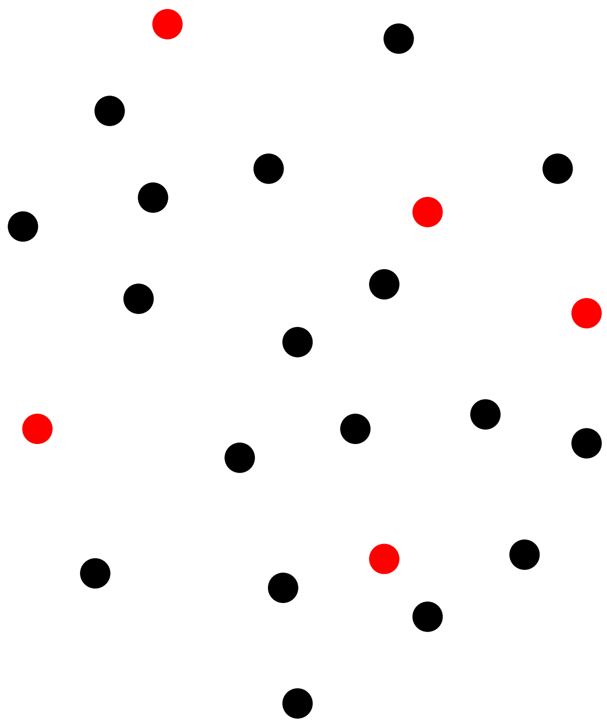


- sample each vertex indep. with prob. $1/\sqrt{n}$.

The Thorup-Zwick algorithm

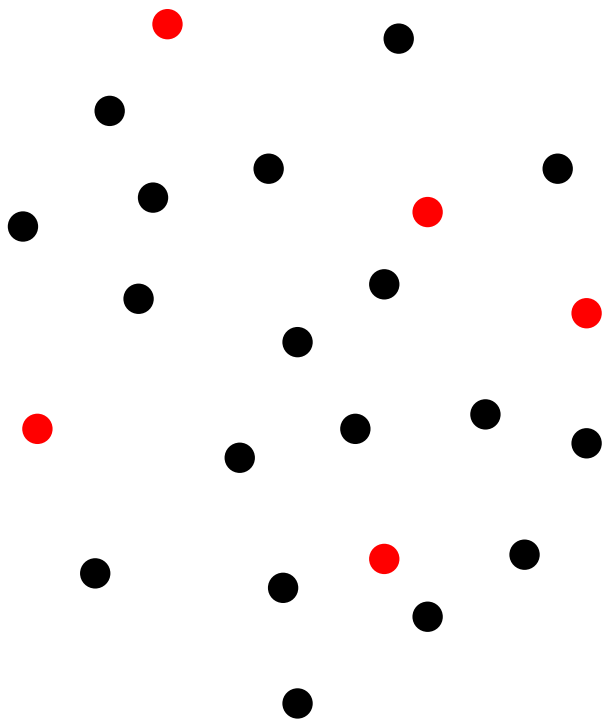


The Thorup-Zwick algorithm



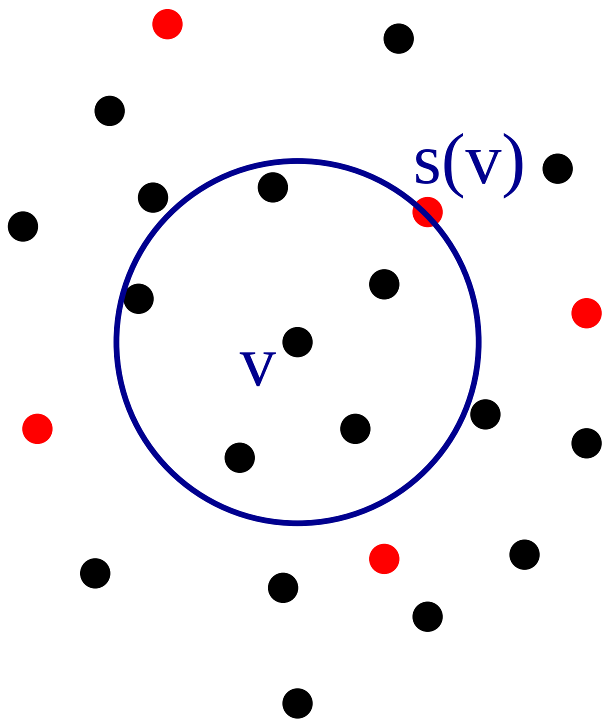
- let S be the set of sampled vertices.

The Thorup-Zwick algorithm



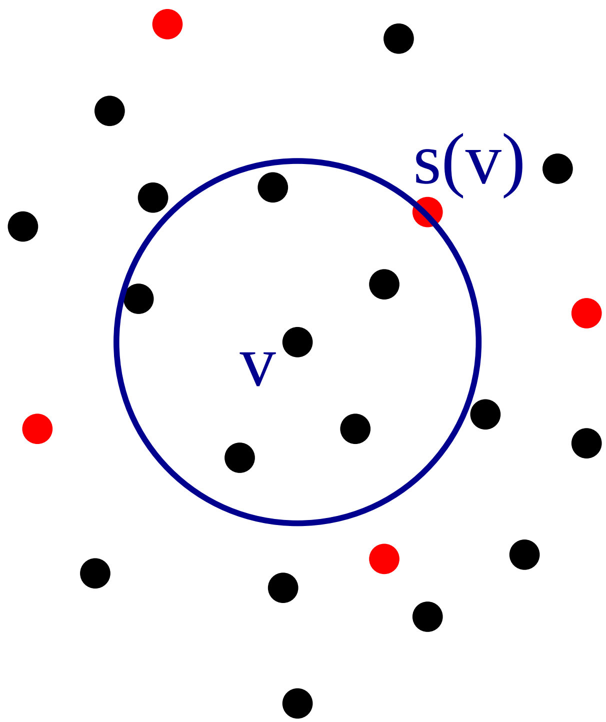
- let S be the set of sampled vertices.
- $E[|S|] = \sqrt{n}$.

The Thorup-Zwick algorithm



$b(v)$: set of all vertices closer to v than $s(v)$.

The Thorup-Zwick algorithm



$b(v)$: set of all vertices closer to v than $s(v)$.

$E[|b(v)|] = \sqrt{n}$.

The Thorup-Zwick algorithm

1. each v stores distances to all vertices in $b(v)$.

The Thorup-Zwick algorithm

1. each v stores distances to all vertices in $b(v)$.

2. each $s \in S$ stores distances to all vertices.

The Thorup-Zwick algorithm

1. each v stores distances to all vertices in $b(v)$.

2. each $s \in S$ stores distances to all vertices.

– if *query* $= (v, u)$,

The Thorup-Zwick algorithm

1. each v stores distances to all vertices in $b(v)$.

2. each $s \in S$ stores distances to all vertices.

– if *query* $= (v, u)$,

- if $u \in b(v)$ return $\delta(v, u)$.

The Thorup-Zwick algorithm

1. each v stores distances to all vertices in $b(v)$.

2. each $s \in S$ stores distances to all vertices.

– if *query* $= (v, u)$,

- if $u \in b(v)$ return $\delta(v, u)$.
- else return $\delta(v, s(v)) + \delta(s(v), u)$.

The Thorup-Zwick algorithm

1. each v stores distances to all vertices in $b(v)$.

2. each $s \in S$ stores distances to all vertices.

– if *query* $= (v, u)$,

- if $u \in b(v)$ return $\delta(v, u)$.
- else return $\delta(v, s(v)) + \delta(s(v), u)$.

$$\leq \delta(v, u) + \delta(s(v), v) + \delta(v, u)$$

The Thorup-Zwick algorithm

1. each v stores distances to all vertices in $b(v)$.

2. each $s \in S$ stores distances to all vertices.

– if *query* $= (v, u)$,

- if $u \in b(v)$ return $\delta(v, u)$.
- else return $\delta(v, s(v)) + \delta(s(v), u)$.

$$\leq \delta(v, u) + \delta(s(v), v) + \delta(v, u)$$

$$\leq 3\delta(v, u).$$

The Thorup-Zwick algorithm

The Thorup-Zwick algorithm

1. each v computes distances to vertices in $b(v)$ by performing a truncated Dijkstra.

The Thorup-Zwick algorithm

1. each v computes distances to vertices in $b(v)$ by performing a truncated Dijkstra.

– the expected running time is $O(m\sqrt{n})$.

The Thorup-Zwick algorithm

1. each v computes distances to vertices in $b(v)$ by performing a truncated Dijkstra.

– the expected running time is $O(m\sqrt{n})$.

2. each $s \in S$ stores distances to all vertices by performing a Dijkstra in the entire graph.

The Thorup-Zwick algorithm

1. each v computes distances to vertices in $b(v)$ by performing a truncated Dijkstra.

– the expected running time is $O(m\sqrt{n})$.

2. each $s \in S$ stores distances to all vertices by performing a Dijkstra in the entire graph.

– the expected running time is $O(m\sqrt{n})$.

The Thorup-Zwick algorithm

- the size of the data structure is $O(n^{3/2})$.

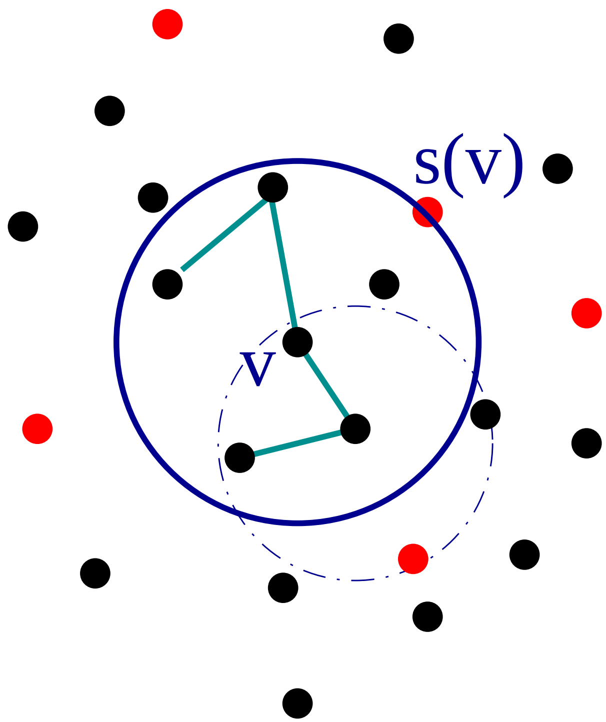
The Thorup-Zwick algorithm

- the size of the data structure is $O(n^{3/2})$.

* the query answering time is $O(1)$.

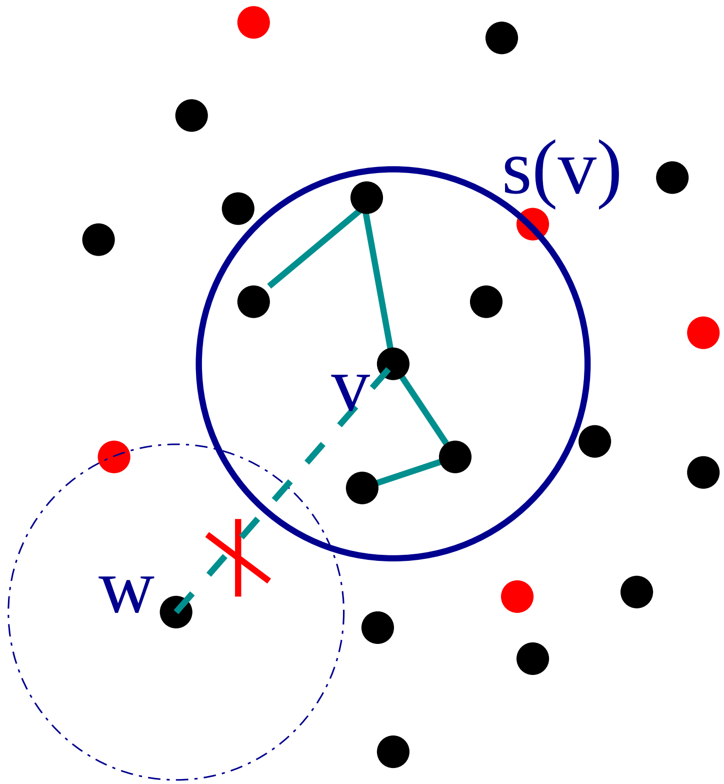
A faster algorithm

A faster algorithm



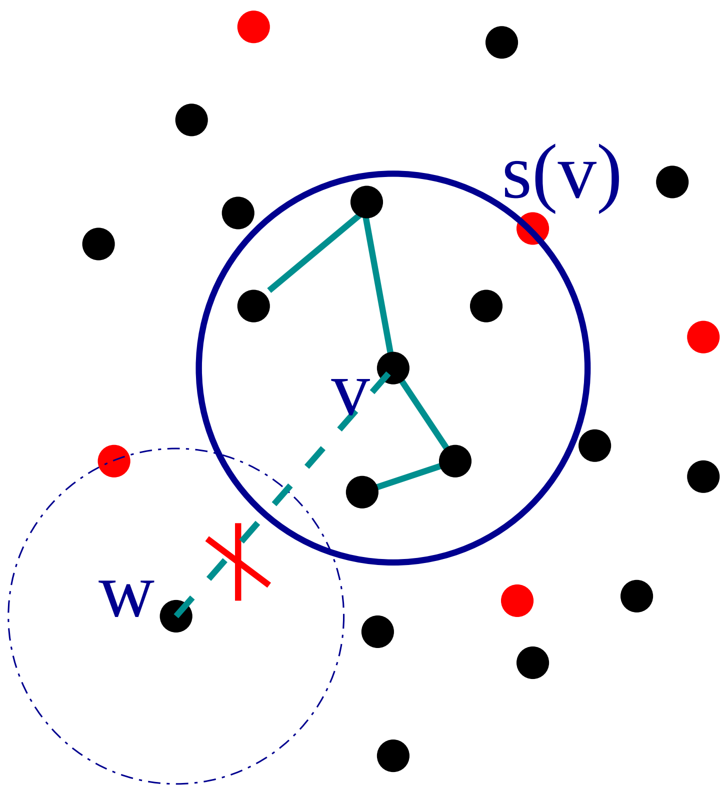
– *improving step 1*: each v performs truncated Dijkstra in a subgraph G' of G.

A faster algorithm



– an edge like (v, w) cannot lie in $b(u)$, for any u.

A faster algorithm



- an edge like (v, w) cannot lie in $b(u)$, for any u.
- G' contains no such edge: $E[|G'|] = n^{3/2}$.

A faster algorithm

1. each v computes distances to vertices in $b(v)$ by performing a truncated Dijkstra *in G'*.

A faster algorithm

1. each v computes distances to vertices in $b(v)$ by performing a truncated Dijkstra *in* G'.

– the expected running time is $O(m_{G'}\sqrt{n})$.

A faster algorithm

1. each v computes distances to vertices in $b(v)$ by performing a truncated Dijkstra *in* G'.

– the expected running time is $O(m_{G'}\sqrt{n})$.

– the exp. time for this step now is $O(n^2)$.

A faster algorithm

1. each v computes distances to vertices in $b(v)$ by performing a truncated Dijkstra *in G'*.

– the expected running time is $O(m_{G'}\sqrt{n})$.

– the exp. time for this step now is $O(n^2)$.

– *we now need to improve step 2:*

each $s \in S$ computes distances to all vertices in G.

A faster algorithm

A faster algorithm

- sample vertices from S to form S_1.

A faster algorithm

- sample vertices from S to form S_1.
- we have

$$S_0 = S \supseteq S_1 \supseteq S_2 \supseteq \cdots S_{(\log n)/2} \supseteq \emptyset.$$

A faster algorithm

- sample vertices from S to form S_1.
- we have

$$S_0 = S \supseteq S_1 \supseteq S_2 \supseteq \cdots S_{(\log n)/2} \supseteq \emptyset.$$

- S_{i+1} is obtained by sampling each vertex in S_i with prob. $1/2$.

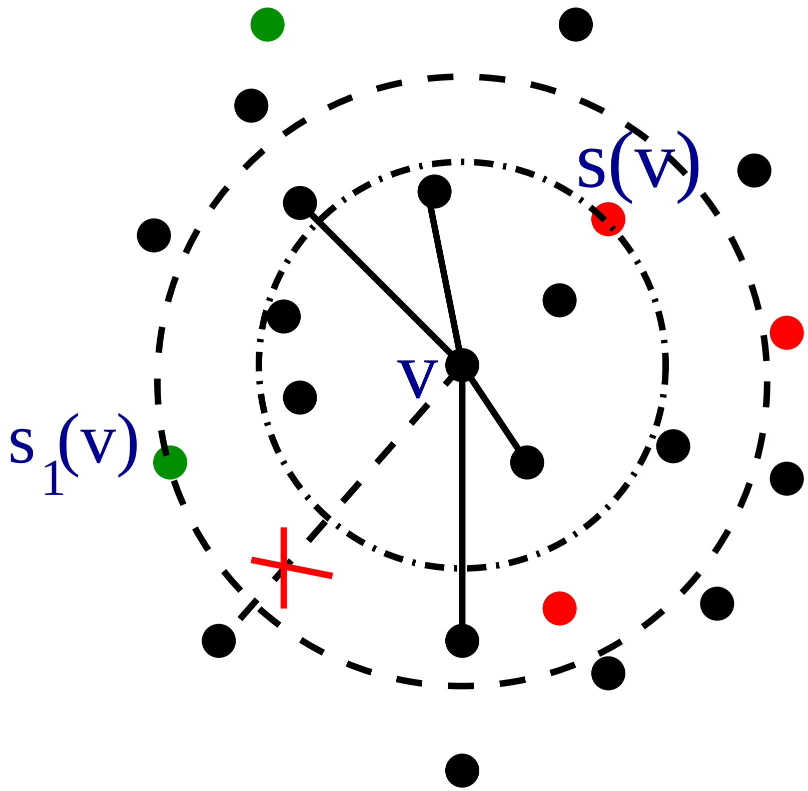
A faster algorithm

- sample vertices from S to form S_1.
- we have

$$S_0 = S \supseteq S_1 \supseteq S_2 \supseteq \cdots S_{(\log n)/2} \supseteq \emptyset.$$

- S_{i+1} is obtained by sampling each vertex in S_i with prob. $1/2$.
- each $s_i \in S_i$ does a Dijkstra in the subgraph G_{i+1}.

A faster algorithm



$G_1 = \cup_v$ edges of the form above.

A faster algorithm

- $E[|S_i|] = \sqrt{n}/2^i$ and $E[|G_{i+1}|] = n^{3/2}.2^i$

A faster algorithm

- $E[|S_i|] = \sqrt{n}/2^i$ and $E[|G_{i+1}|] = n^{3/2}.2^i$
  $\rightsquigarrow$ exp. time for step 2 now is $O(n^2 \log n)$.

A faster algorithm

- $E[|S_i|] = \sqrt{n}/2^i$ and $E[|G_{i+1}|] = n^{3/2}.2^i$
  $\rightsquigarrow$ exp. time for step 2 now is $O(n^2 \log n)$.

- each v stores distances to all vertices in $b(v)$ and vertices $s(v), s_1(v), s_2(v), \ldots$.

A faster algorithm

- $E[|S_i|] = \sqrt{n}/2^i$ and $E[|G_{i+1}|] = n^{3/2}.2^i$
  $\rightsquigarrow$ exp. time for step 2 now is $O(n^2 \log n)$.

- each v stores distances to all vertices in $b(v)$ and vertices $s(v), s_1(v), s_2(v), \ldots$.

- each $s \in S$ stores all the distances that it computes.

Answering queries

– if *query* $= (v, u)$,

Answering queries

– if *query* $= (v, u)$,

- if $u \in b(v)$ return $\delta(v, u)$.

Answering queries

- if *query* $= (v, u)$,
  - if $u \in b(v)$ return $\delta(v, u)$.
  - else return

$$\min_{0 \le i \le (\log n)/2} \delta(v, s_i(v)) + \delta(s_i(v), u).$$

Answering queries

– if *query* $= (v, u)$,

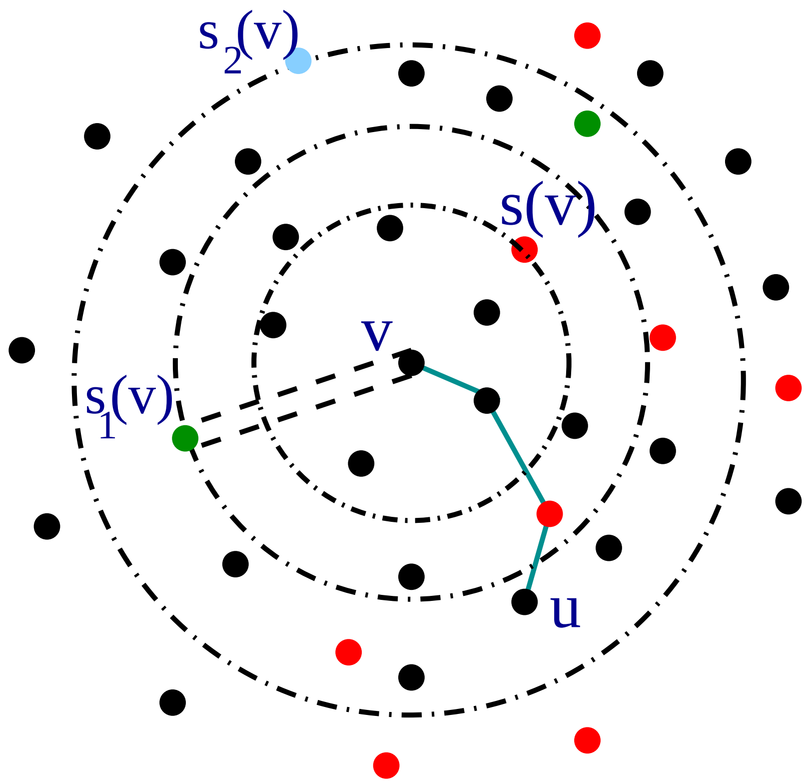
- if $u \in b(v)$ return $\delta(v, u)$.
- else return

$$\min_{0 \le i \le (\log n)/2} \delta(v, s_i(v)) + \delta(s_i(v), u).$$

$$\le \delta(v, u) + \delta(s_i(v), v) + \delta(v, u) \le 3\delta(v, u)$$

for some $s_i(v)$ which knows $\delta(s_i(v), u)$.

A faster algorithm



$$\delta(v, s_1(v)) + \delta(s_1(v), u) \leq 3\delta(u, v).$$

Analysis

Analysis

- computing the minimum of $\delta(v, s_i(v)) + \delta(s_i(v), u)$, for $0 \le i \le (\log n)/2$ takes $O(\log n)$ time.

Analysis

- computing the minimum of $\delta(v, s_i(v)) + \delta(s_i(v), u)$, for $0 \leq i \leq (\log n)/2$ takes $O(\log n)$ time.

- time to construct the data structure: $O(n^2 \log n)$.

Analysis

- computing the minimum of $\delta(v, s_i(v)) + \delta(s_i(v), u)$, for $0 \leq i \leq (\log n)/2$ takes $O(\log n)$ time.

- time to construct the data structure: $O(n^2 \log n)$.

- space requirement: $O(n^{3/2})$

An Extension

An Extension

- sample vertices from V to form S_1.

An Extension

- sample vertices from V to form S_1.

- we have

$$S_0 = V \supseteq S_1 \supseteq S_2 \supseteq \cdots S_{\log n} \supseteq \emptyset.$$

An Extension

- sample vertices from V to form S_1.
- we have

$$S_0 = V \supseteq S_1 \supseteq S_2 \supseteq \cdots S_{\log n} \supseteq \emptyset.$$

- S_{i+1} is obtained by sampling each vertex in S_i with prob. $1/2$.

An Extension

- each $s_i \in S_i$ does a Dijkstra in the subgraph G_{i+1}.
- G_{i+1}: consists of all edges incident on each v whose weight is less than $\delta(v, s_{i+1}(v))$.
- For each v, u in G

$$d[v,u] \leftarrow \min_{0 \le i \le \log n}(\delta(v, s_i(v)) + \hat{\delta}(u, s_i(v)), \delta(u, s_i(u)) + \hat{\delta}(v, s_i(u)))$$

An Extension

An Extension

- easy to show that

$$\delta(u,v) \le d[u,v] \le 2\delta(u,v) + w_{uv}$$

An Extension

- easy to show that

$$\delta(u,v) \leq d[u,v] \leq 2\delta(u,v) + w_{uv}$$

- $E[|S_i|] = n/2^i$ and $E[|G_{i+1}|] = n.2^i$

An Extension

- easy to show that

$$\delta(u,v) \leq d[u,v] \leq 2\delta(u,v) + w_{uv}$$

- $E[|S_i|] = n/2^i$ and $E[|G_{i+1}|] = n.2^i$
  $\rightsquigarrow$ exp. time for building d is $O(n^2 \log n)$.

An Extension

- easy to show that

$$\delta(u,v) \leq d[u,v] \leq 2\delta(u,v) + w_{uv}$$

- $E[|S_i|] = n/2^i$ and $E[|G_{i+1}|] = n.2^i$
  $\leadsto$ exp. time for building d is $O(n^2 \log n)$.

- all-pairs-$(2,w)$-approx. distances in exp. $O(n^2 \log n)$ time.

An Extension

- easy to show that

$$\delta(u,v) \leq d[u,v] \leq 2\delta(u,v) + w_{uv}$$

- $E[|S_i|] = n/2^i$ and $E[|G_{i+1}|] = n.2^i$
  $\rightsquigarrow$ exp. time for building d is $O(n^2 \log n)$.

- all-pairs-$(2, w)$-approx. distances in exp. $O(n^2 \log n)$ time.

Thank you.