Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \rightarrow Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \to Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \rightarrow Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \rightarrow Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \to Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \to Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \rightarrow Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \to Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

(Reachable) Negative cycle, answer is undefined.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \to Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

(Reachable) Negative cycle, answer is undefined.

Find price Function: $\phi : V \to Z$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \rightarrow Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

(Reachable) Negative cycle, answer is undefined.

Find price Function: $\phi : V \rightarrow Z$.
$c_\phi(e = (u, v)) = \phi(u) - \phi(v) + w(e)$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \to Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

(Reachable) Negative cycle, answer is undefined.

Find price Function: $\phi : V \to Z$.
$c_\phi(e = (u, v)) = \phi(u) - \phi(v) + w(e)$.
Note: $d(v) \leq d(u) + w(e)$

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \rightarrow Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

(Reachable) Negative cycle, answer is undefined.

Find price Function: $\phi : V \rightarrow Z$.
$c_\phi(e = (u, v)) = \phi(u) - \phi(v) + w(e)$.
Note: $d(v) \leq d(u) + w(e) \implies d(u) + w(e) - d(v) \geq 0$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \rightarrow Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

(Reachable) Negative cycle, answer is undefined.

Find price Function: $\phi : V \rightarrow Z$.
$c_\phi(e = (u, v)) = \phi(u) - \phi(v) + w(e)$.
Note: $d(v) \leq d(u) + w(e) \implies d(u) + w(e) - d(v) \geq 0$.
$\phi(v) = d(v)$ produces non-negative edge weights.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \to Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

(Reachable) Negative cycle, answer is undefined.

Find price Function: $\phi : V \to Z$.
$c_\phi(e = (u, v)) = \phi(u) - \phi(v) + w(e)$.
Note: $d(v) \leq d(u) + w(e) \implies d(u) + w(e) - d(v) \geq 0$.
$\phi(v) = d(v)$ produces non-negative edge weights.
Shortest path under $c_\phi(e)$ is same as under $w(e)$.

Bellman-Ford. Djikstra. Price Functions.

Given $G = (V, E)$, $w : E \rightarrow Z$, on edges, and $s \in V$, find $d(s, v)$ $\forall v \in V$.

$d(s, v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v) = \infty, d(s) = 0$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u, v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

(Reachable) Negative cycle, answer is undefined.

Find price Function: $\phi : V \rightarrow Z$.
$c_\phi(e = (u, v)) = \phi(u) - \phi(v) + w(e)$.
Note: $d(v) \leq d(u) + w(e) \implies d(u) + w(e) - d(v) \geq 0$.
$\phi(v) = d(v)$ produces non-negative edge weights.
Shortest path under $c_\phi(e)$ is same as under $w(e)$.
Path $p = [(u, v), (v, w)]$, $c_\phi(p) = \phi(u) + w(u, v) - \phi(v) + \phi(v) + w(v, w) - \phi(w) = w(p) + \phi(u) - \phi(w)$. p from s to t, $\sum_{e \in p} c_\phi(e) = \phi(s) - \phi(t) + w(p)$.

Bellman-Ford. Djikstra. Price Functions.

Given $G=(V,E)$, $w:E\rightarrow Z$, on edges, and $s\in V$, find $d(s,v)$ $\forall v\in V$.

$d(s,v)$ - length of shortest path.

Djikstra: Non-Negative edge weights: $d(v)=\infty, d(s)=0$.
$S=\phi$.
Find $u=\text{argmin}_{v\notin S} d(v)$.
update(u): for $e=(u,v), d(v)=\min(d(v), d(u)+w(e))$.
$S=S+u$.

(Reachable) Negative cycle, answer is undefined.

Find price Function: $\phi:V\rightarrow Z$.
$c_\phi(e=(u,v))=\phi(u)-\phi(v)+w(e)$.
Note: $d(v)\leq d(u)+w(e) \implies d(u)+w(e)-d(v)\geq 0$.
$\phi(v)=d(v)$ produces non-negative edge weights.
Shortest path under $c_\phi(e)$ is same as under $w(e)$.
Path $p=[(u,v),(v,w)]$, $c_\phi(p)=\phi(u)+w(u,v)-\phi(v)+\phi(v)+w(v,w)-\phi(w)=w(p)+\phi(u)-\phi(w)$. p from s to t, $\sum_{e\in p} c_\phi(e)=\phi(s)-\phi(t)+w(p)$.

Thus: $d(s,v)$ is a price function whose reduced costs make all edge weights positive.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v)=0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e = (u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u) + w(e))$.
$S = \phi$.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e = (u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u) + w(e))$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v)=0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e=(u,v)$, $w(e) \leq 0$, $d(v)=\min(d(v),d(u)+w(e))$.
$S=\phi$.
Find $u=\text{argmin}_{v \notin S} d(v)$.
update(u): for $e=(u,v), d(v)=\min(d(v),d(u)+w(e))$.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e = (u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u) + w(e))$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u,v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e = (u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u) + w(e))$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u,v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

Claim: After k rounds, $d(v) \leq$ path length with $\leq k$ negative edges.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v)=0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e=(u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u)+w(e))$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e=(u,v), d(v) = \min(d(v), d(u)+w(e))$.
$S = S + u$.

Claim: After k rounds, $d(v) \leq$ path length with $\leq k$ negative edges.

Induction.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e = (u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u) + w(e))$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u,v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

Claim: After k rounds, $d(v) \leq$ path length with $\leq k$ negative edges.

Induction.

$O(n)$ iterations of Bellman/Dijkstra is good.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e = (u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u) + w(e))$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u,v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

Claim: After k rounds, $d(v) \leq$ path length with $\leq k$ negative edges.

Induction.

$O(n)$ iterations of Bellman/Dijkstra is good.

$O(n(n + m \log n))$

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e = (u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u) + w(e))$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u,v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

Claim: After k rounds, $d(v) \leq$ path length with $\leq k$ negative edges.

Induction.

$O(n)$ iterations of Bellman/Dijkstra is good.

$O(n(n + m \log n))$ or slightly better.
Quadratic time.

Bellman/Dijkstra.

Approach: Add s, with $w(s,v) = 0$ for all $v \in V$.

Bellman/Djikstra Round: Have $d(v)$.
For all $e = (u,v)$, $w(e) \leq 0$, $d(v) = \min(d(v), d(u) + w(e))$.
$S = \phi$.
Find $u = \text{argmin}_{v \notin S} d(v)$.
update(u): for $e = (u,v), d(v) = \min(d(v), d(u) + w(e))$.
$S = S + u$.

Claim: After k rounds, $d(v) \leq$ path length with $\leq k$ negative edges.

Induction.

$O(n)$ iterations of Bellman/Dijkstra is good.

$O(n(n + m\log n))$ or slightly better.
Quadratic time.

Scaling algorithm: $O(m\sqrt{n}\log nC)$ by Goldberg.

Example(Intuition?): fix one negative vertex.

Negative vertex: v.

Example(Intuition?): fix one negative vertex.

Negative vertex: v.

Has negative arcs from it.

Example(Intuition?): fix one negative vertex.

Negative vertex: v.
 Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.
 Otherwise set $w'(e) = w(e)$

Example(Intuition?): fix one negative vertex.

Negative vertex: v.
  Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.
  Otherwise set $w'(e) = w(e)$

$d(v) = 0$. $S = \{v\}$
Run update (v).
...Djikstra..

Example(Intuition?): fix one negative vertex.

Negative vertex: v.
Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.
Otherwise set $w'(e) = w(e)$

$d(v) = 0$. $S = \{v\}$
Run update (v).
...Djikstra..

Example(Intuition?): fix one negative vertex.

Negative vertex: v.

Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.

Otherwise set $w'(e) = w(e)$

$d(v) = 0$. $S = \{v\}$

Run update (v).

...Djikstra..

Observe: vertices put in S once.

Example(Intuition?): fix one negative vertex.

Negative vertex: v.
  Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.
  Otherwise set $w'(e) = w(e)$

$d(v) = 0$. $S = \{v\}$
Run update (v).
...Djikstra..

  Observe: vertices put in S once.
   Correct distances, w.r.t. $w'(e)$.

Example(Intuition?): fix one negative vertex.

Negative vertex: v.
  Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.
  Otherwise set $w'(e) = w(e)$

$d(v) = 0$. $S = \{v\}$
Run update (v).
...Djikstra..

  Observe: vertices put in S once.
    Correct distances, w.r.t. $w'(e)$.
    If no negative cycle in $w'(e)$.

Example(Intuition?): fix one negative vertex.

Negative vertex: v.
  Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.
  Otherwise set $w'(e) = w(e)$

$d(v) = 0$. $S = \{v\}$
Run update (v).
...Djikstra..

  Observe: vertices put in S once.
    Correct distances, w.r.t. $w'(e)$.
    If no negative cycle in $w'(e)$.

Example(Intuition?): fix one negative vertex.

Negative vertex: v.
Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.
Otherwise set $w'(e) = w(e)$

$d(v) = 0$. $S = \{v\}$
Run update (v).
...Djikstra..

Observe: vertices put in S once.
Correct distances, w.r.t. $w'(e)$.
If no negative cycle in $w'(e)$.

Reduced costs of $w'(e)$ w.r.t $d(\cdot)$ are positive w.r.t.

Example(Intuition?): fix one negative vertex.

Negative vertex: v.

Has negative arcs from it.

Set all $\forall e' \in E, w(e') \leq 0$ set $w'(e) = 0$.

Otherwise set $w'(e) = w(e)$

$d(v) = 0$. $S = \{v\}$

Run update (v).

...Djikstra..

Observe: vertices put in S once.

Correct distances, w.r.t. $w'(e)$.

If no negative cycle in $w'(e)$.

Reduced costs of $w'(e)$ w.r.t $d(\cdot)$ are positive w.r.t.

Reduce number of negative vertices by one.

Hop Distance

$d^h(u, v)$ shortest distance using at most h negative edges.

Hop Distance

$d^h(u, v)$ shortest distance using at most h negative edges.

u and v are h-hop connected: $d^h(u, v) < 0$ or $d^h(v, u) < 0$.

Hop Distance

$d^h(u, v)$ shortest distance using at most h negative edges.

u and v are h-hop connected: $d^h(u, v) < 0$ or $d^h(v, u) < 0$.

True/False: If u and v are not h-hop connected they are not $h + 1$ connected.

True/False: If u and v are not h-hop connected they are not $h - 1$ connected.

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Idea: Running “Dijsktra” makes them all not-negative.

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Idea: Running "Dijsktra" makes them all not-negative.

S is independent if all pairs $u, v \in S$ are independent.

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Idea: Running "Dijsktra" makes them all not-negative.

S is independent if all pairs $u, v \in S$ are independent.

For $e = (u, v)$, $u \notin S$, set $w'(u, v) = 0$ if $w(u, v) < 0$.

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Idea: Running "Dijsktra" makes them all not-negative.

S is independent if all pairs $u, v \in S$ are independent.

For $e = (u, v)$, $u \notin S$, set $w'(u, v) = 0$ if $w(u, v) < 0$.
else $w'(u, v) = w(u, v)$.

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Idea: Running "Dijsktra" makes them all not-negative.

S is independent if all pairs $u, v \in S$ are independent.

For $e = (u, v)$, $u \notin S$, set $w'(u, v) = 0$ if $w(u, v) < 0$.
else $w'(u, v) = w(u, v)$.

update vertices in S, run Dijkstra.

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Idea: Running "Dijsktra" makes them all not-negative.

S is independent if all pairs $u, v \in S$ are independent.

For $e = (u, v)$, $u \notin S$, set $w'(u, v) = 0$ if $w(u, v) < 0$.
else $w'(u, v) = w(u, v)$.

update vertices in S, run Dijkstra.

Reduced costs of $w'(e)$ w.r.t. $d(v)$ are non-negative.

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Idea: Running "Dijsktra" makes them all not-negative.

S is independent if all pairs $u, v \in S$ are independent.

For $e = (u, v)$, $u \notin S$, set $w'(u, v) = 0$ if $w(u, v) < 0$.
else $w'(u, v) = w(u, v)$.

update vertices in S, run Dijkstra.

Reduced costs of $w'(e)$ w.r.t. $d(v)$ are non-negative.

For reduced costs of $w(e)$ w.r.t. $d(v)$?

Eliminating many negative vertices.

Negative vertices are independent if they are not 1-hop connected.

Idea: Running “Dijsktra” makes them all not-negative.

S is independent if all pairs $u, v \in S$ are independent.

For $e = (u, v)$, $u \notin S$, set $w'(u, v) = 0$ if $w(u, v) < 0$.
else $w'(u, v) = w(u, v)$.

update vertices in S, run Dijkstra.

Reduced costs of $w'(e)$ w.r.t. $d(v)$ are non-negative.

For reduced costs of $w(e)$ w.r.t. $d(v)$?
S are no longer negative vertices.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Do DAG computation. $O((m + n \log n)r)$ time.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Do DAG computation. $O((m + n\log n)r)$ time.

Only negative edges back to first level.

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Do DAG computation. $O((m+n\log n)r)$ time.

Only negative edges back to first level.

Paths in this graph have only n/r negative arcs in their path!

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Do DAG computation. $O((m + n\log n)r)$ time.

Only negative edges back to first level.

Paths in this graph have only n/r negative arcs in their path!

So n/r iterations of Bellman/Ford is enough!

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Do DAG computation. $O((m + n\log n)r)$ time.

Only negative edges back to first level.

Paths in this graph have only n/r negative arcs in their path!

So n/r iterations of Bellman/Ford is enough!

So $n/r \times O(r(m + n\log n))$

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Do DAG computation. $O((m + n \log n)r)$ time.

Only negative edges back to first level.

Paths in this graph have only n/r negative arcs in their path!

So n/r iterations of Bellman/Ford is enough!

So $n/r \times O(r(m + n \log n)) = O(n(m + n \log n))$!!!

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Do DAG computation. $O((m+n\log n)r)$ time.

Only negative edges back to first level.

Paths in this graph have only n/r negative arcs in their path!

So n/r iterations of Bellman/Ford is enough!

So $n/r \times O(r(m+n\log n)) = O(n(m+n\log n))$!!! Doh!!!!

Remoteness

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of graph,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

"Computation dag" of r iterations of Dijkstra/Bellman. (except for edge back.)

Do DAG computation. $O((m + n\log n)r)$ time.

Only negative edges back to first level.

Paths in this graph have only n/r negative arcs in their path!

So n/r iterations of Bellman/Ford is enough!

So $n/r \times O(r(m + n\log n)) = O(n(m + n\log n))$!!! Doh!!!!

No improvement.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.
thus the n/r iterations is enough to see all negative paths.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.
thus the n/r iterations is enough to see all negative paths.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.
thus the n/r iterations is enough to see all negative paths.

$O(n/r) \times O((r(n/r + m/r \log n)) + (n + m \log n))$

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.
thus the n/r iterations is enough to see all negative paths.

$O(n/r) \times O((r(n/r + m/r \log n)) + (n + m \log n)) = O(n/r(n + m \log n))$.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.
thus the n/r iterations is enough to see all negative paths.

$O(n/r) \times O((r(n/r + m/r \log n)) + (n + m \log n)) = O(n/r(n + m \log n))$.

n/r iterations of Bellman/Dijkstra to get rid of $|S|$ negative vertices.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.
thus the n/r iterations is enough to see all negative paths.

$O(n/r) \times O((r(n/r + m/r \log n)) + (n + m \log n)) = O(n/r(n + m \log n))$.

n/r iterations of Bellman/Dijkstra to get rid of $|S|$ negative vertices.

To get rid of all of them: $(n/|S|)(n/r)$ versus n.

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.
thus the n/r iterations is enough to see all negative paths.

$O(n/r) \times O((r(n/r + m/r \log n)) + (n + m \log n)) = O(n/r(n + m \log n))$.

n/r iterations of Bellman/Dijkstra to get rid of $|S|$ negative vertices.

To get rid of all of them: $(n/|S|)(n/r)$ versus n.

Thus, if $|S|r \geq n$, it is win!

This time use remoteness.

Set S is *r-remote* if S can reach $\leq n/r$ vertices with r-hop paths.

Elimination Algorithm.

Make r copies of reachable vertices,
connect successive levels by (directed) negative edges.
And connect copies of vertices by 0-weight edges.
and edge back from last to first
Copies of positive edges are internal to level.
Zero out weights on negative $v \notin S$.

The vertices $v \notin S$ are r-hop far.
Thus, must use $\geq r$ negative nodes in S.
thus the n/r iterations is enough to see all negative paths.

$O(n/r) \times O((r(n/r + m/r \log n)) + (n + m \log n)) = O(n/r(n + m \log n))$.

n/r iterations of Bellman/Dijkstra to get rid of $|S|$ negative vertices.

To get rid of all of them: $(n/|S|)(n/r)$ versus n.

Thus, if $|S|r \geq n$, it is win!

Find remote set with big r and big S.

Betweenness

v is h-hop between s and t if $d^h(s, v) < 0$ and $d^h(v, t) < 0$.

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large?

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

For $b \log n$ vertices, compute h-hop in-distances and out-distances.

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

For $b \log n$ vertices, compute h-hop in-distances and out-distances.
  Compute reduced costs using these potential functions.

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

For $b \log n$ vertices, compute h-hop in-distances and out-distances.
  Compute reduced costs using these potential functions.
 Time: $O(hb \log n(m + n \log n))$.

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

For $b \log n$ vertices, compute h-hop in-distances and out-distances.
Compute reduced costs using these potential functions.
Time: $O(hb \log n(m + n \log n)$.

Lemma: W.h.p. $|B^h(s,t)| \leq n/b$.

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

For $b\log n$ vertices, compute h-hop in-distances and out-distances.
Compute reduced costs using these potential functions.
Time: $O(hb\log n(m+n\log n)$.

Lemma: W.h.p. $|B^h(s,t)| \leq n/b$.

Proof sketch: Consider s,t, sort $B^h(s,t)$ vertices by $d^h(s,u)+d^h(u,t)$.
W.h.p. vertex is in smallest n/b vertices of $B^h(s,t)$.
Observe:

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

For $b \log n$ vertices, compute h-hop in-distances and out-distances.
Compute reduced costs using these potential functions.
Time: $O(hb \log n(m + n \log n)$.

Lemma: W.h.p. $|B^h(s,t)| \leq n/b$.

Proof sketch: Consider s,t, sort $B^h(s,t)$ vertices by $d^h(s,u) + d^h(u,t)$.
W.h.p. vertex is in smallest n/b vertices of $B^h(s,t)$.
Observe:
Price function adjustments, make paths positive!

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

For $b \log n$ vertices, compute h-hop in-distances and out-distances.
Compute reduced costs using these potential functions.
Time: $O(hb \log n(m + n \log n)$.

Lemma: W.h.p. $|B^h(s,t)| \leq n/b$.

Proof sketch: Consider s,t, sort $B^h(s,t)$ vertices by $d^h(s,u) + d^h(u,t)$.
W.h.p. vertex is in smallest n/b vertices of $B^h(s,t)$.
Observe:
Price function adjustments, make paths positive!
And keep shortest paths ordered. So other nodes are positive.

Betweenness

v is h-hop between s and t if $d^h(s,v) < 0$ and $d^h(v,t) < 0$.

For pair, s,t, $B^h(s,t)$ is set of vertices between s and t.

How large? Could be a lot.

For $b\log n$ vertices, compute h-hop in-distances and out-distances.
Compute reduced costs using these potential functions.
Time: $O(hb\log n(m+n\log n)$.

Lemma: W.h.p. $|B^h(s,t)| \leq n/b$.

Proof sketch: Consider s,t, sort $B^h(s,t)$ vertices by $d^h(s,u)+d^h(u,t)$.
W.h.p. vertex is in smallest n/b vertices of $B^h(s,t)$.
Observe:
Price function adjustments, make paths positive!
And keep shortest paths ordered. So other nodes are positive.
Thus, the $|(B')^h(s,t)| \leq n/b$ $\square$.

Negative Sandwich, Betweeness, Remoteness

(s, t, S) is a h-hop negative sandwich if $\forall u \in S, d^h(s, u) + d^h(u, t) < 0$ for negative vertices S.

Negative Sandwich, Betweeness, Remoteness

(s, t, S) is a h-hop negative sandwich if $\forall u \in S, d^h(s, u) + d^h(u, t) < 0$ for negative vertices S.

negative between vertices..

Negative Sandwich, Betweeness, Remoteness

(s,t,S) is a h-hop negative sandwich if $\forall u \in S, d^h(s,u) + d^h(u,t) < 0$ for negative vertices S.

negative between vertices..

Negative Sandwich, Betweeness, Remoteness

(s, t, S) is a h-hop negative sandwich if $\forall u \in S, d^h(s, u) + d^h(u, t) < 0$ for negative vertices S.

negative between vertices..

Price function: $\phi(u) = \min(0, \max(d^{h+r}(s, u), -d^{h+r}(u, t))$.

Negative Sandwich, Betweeness, Remoteness

(s, t, S) is a h-hop negative sandwich if $\forall u \in S, d^h(s, u) + d^h(u, t) < 0$ for negative vertices S.

negative between vertices..

Price function: $\phi(u) = \min(0, \max(d^{h+r}(s, u), -d^{h+r}(u, t))$.

Any vertex that is not r-between s and t is r-remote from S.

Negative Sandwich, Betweeness, Remoteness

(s, t, S) is a h-hop negative sandwich if $\forall u \in S, d^h(s,u) + d^h(u,t) < 0$ for negative vertices S.

negative between vertices..

Price function: $\phi(u) = \min(0, \max(d^{h+r}(s,u), -d^{h+r}(u,t))$.

Any vertex that is not r-between s and t is r-remote from S.
Recall: only n/r in between.

Negative Sandwich, Betweeness, Remoteness

(s,t,S) is a h-hop negative sandwich if $\forall u \in S, d^h(s,u) + d^h(u,t) < 0$ for negative vertices S.

negative between vertices..

Price function: $\phi(u) = \min(0, \max(d^{h+r}(s,u), -d^{h+r}(u,t))$.

Any vertex that is not r-between s and t is r-remote from S.
Recall: only n/r in between.

S is r-remote set.

Negative Sandwich, Betweeness, Remoteness

(s, t, S) is a h-hop negative sandwich if $\forall u \in S, d^h(s,u) + d^h(u,t) < 0$ for negative vertices S.

negative between vertices..

Price function: $\phi(u) = \min(0, \max(d^{h+r}(s,u), -d^{h+r}(u,t))$.

Any vertex that is not r-between s and t is r-remote from S.
Recall: only n/r in between.

S is r-remote set.

Find large sandwich.

Negative Sandwich, Betweeness, Remoteness

(s, t, S) is a h-hop negative sandwich if $\forall u \in S, d^h(s, u) + d^h(u, t) < 0$ for negative vertices S.

negative between vertices..

Price function: $\phi(u) = \min(0, \max(d^{h+r}(s, u), -d^{h+r}(u, t))$.

Any vertex that is not r-between s and t is r-remote from S.
Recall: only n/r in between.

S is r-remote set.

Find large sandwich. That is, $|S|r \geq n$.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty h* negative hops.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty h* negative hops.

Lemma: $O(h(m + n\log n))$ time for set S of negative vertices,

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty* h negative hops.

Lemma: $O(h(m+n\log n))$ time for set S of negative vertices,
(i) finds pair $s,t \in S$ with proper h-hop distance $\leq h$ or

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty* h negative hops.

Lemma: $O(h(m + n \log n))$ time for set S of negative vertices,
(i) finds pair $s, t \in S$ with proper h-hop distance $\leq h$ or

(ii) distance $d_S(t, V)$ for all V in G_S.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty* h negative hops.

Lemma: $O(h(m + n \log n))$ time for set S of negative vertices,
(i) finds pair $s, t \in S$ with proper h-hop distance $\leq h$ or

(ii) distance $d_S(t, V)$ for all V in G_S.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty* h negative hops.

Lemma: $O(h(m + n \log n))$ time for set S of negative vertices,
(i) finds pair $s, t \in S$ with proper h-hop distance $\leq h$ or

(ii) distance $d_S(t, V)$ for all V in G_S.

G_S is G where negative weights are zero'd outside of S.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty* h negative hops.

Lemma: $O(h(m+n\log n))$ time for set S of negative vertices,
(i) finds pair $s,t \in S$ with proper h-hop distance $\leq h$ or

(ii) distance $d_S(t,V)$ for all V in G_S.

G_S is G where negative weights are zero'd outside of S.

Proof: Consider G_S. Compute $d^h(S,t)$ and $d^{h+1}(S,t)$ for all t.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty* h negative hops.

Lemma: $O(h(m+n\log n))$ time for set S of negative vertices,
(i) finds pair $s,t \in S$ with proper h-hop distance $\leq h$ or

(ii) distance $d_S(t,V)$ for all V in G_S.

G_S is G where negative weights are zero'd outside of S.

Proof: Consider G_S. Compute $d^h(S,t)$ and $d^{h+1}(S,t)$ for all t.
If change, then *exactly* $h+1$ negative vertices.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty* h negative hops.

Lemma: $O(h(m+n\log n))$ time for set S of negative vertices,
(i) finds pair $s,t \in S$ with proper h-hop distance $\leq h$ or

(ii) distance $d_S(t,V)$ for all V in G_S.

G_S is G where negative weights are zero'd outside of S.

Proof: Consider G_S. Compute $d^h(S,t)$ and $d^{h+1}(S,t)$ for all t.
If change, then *exactly* $h+1$ negative vertices.
Pick subpath starting and ending in S.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty h* negative hops.

Lemma: $O(h(m + n \log n))$ time for set S of negative vertices,
(i) finds pair $s, t \in S$ with proper h-hop distance $\leq h$ or

(ii) distance $d_S(t, V)$ for all V in G_S.

G_S is G where negative weights are zero'd outside of S.

Proof: Consider G_S. Compute $d^h(S, t)$ and $d^{h+1}(S, t)$ for all t.
If change, then *exactly* $h + 1$ negative vertices.
Pick subpath starting and ending in S.

Note: Case (ii), gives price function to make S non-negative.

Proper hop distance.

Hop distance is distance with $\leq h$ negative hops.

Proper is with *exacty* h negative hops.

Lemma: $O(h(m+n\log n))$ time for set S of negative vertices,
(i) finds pair $s,t \in S$ with proper h-hop distance $\leq h$ or

(ii) distance $d_S(t,V)$ for all V in G_S.

G_S is G where negative weights are zero'd outside of S.

Proof: Consider G_S. Compute $d^h(S,t)$ and $d^{h+1}(S,t)$ for all t.
If change, then *exactly* $h+1$ negative vertices.
Pick subpath starting and ending in S.

Note: Case (ii), gives price function to make S non-negative.

Case (i): Can be used to make sandwich.

Large sandwich

Sample a set U with probability q.

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
Either fix $|U|$ negative vertices.

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
Either fix $|U|$ negative vertices.

Or: find pair s, t in S with negative proper h hop distance using S.
At least h negative vertices sampled in (s, t, S).

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
Either fix $|U|$ negative vertices.

Or: find pair s, t in S with negative proper h hop distance using S.
At least h negative vertices sampled in (s, t, S).
Expected size of S is h/q.

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
Either fix $|U|$ negative vertices.

Or: find pair s, t in S with negative proper h hop distance using S.
At least h negative vertices sampled in (s, t, S).
Expected size of S is h/q.

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
Either fix $|U|$ negative vertices.

Or: find pair s, t in S with negative proper h hop distance using S.
At least h negative vertices sampled in (s, t, S).
Expected size of S is h/q.

Let k be number of negative vertices.

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
Either fix $|U|$ negative vertices.

Or: find pair s, t in S with negative proper h hop distance using S.
At least h negative vertices sampled in (s, t, S).
Expected size of S is h/q.

Let k be number of negative vertices.
Set $q = 2\sqrt{h/k}$.

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
Either fix $|U|$ negative vertices.

Or: find pair s, t in S with negative proper h hop distance using S.
At least h negative vertices sampled in (s, t, S).
Expected size of S is h/q.

Let k be number of negative vertices.
Set $q = 2\sqrt{h/k}$.
Expected size of U is $qk = \Omega(\sqrt{kh})$.

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
Either fix $|U|$ negative vertices.

Or: find pair s,t in S with negative proper h hop distance using S.
At least h negative vertices sampled in (s,t,S).
Expected size of S is h/q.

Let k be number of negative vertices.
Set $q = 2\sqrt{h/k}$.
Expected size of U is $qk = \Omega(\sqrt{kh})$.
Expected size of S is $h/q = \Omega(\sqrt{kh})$.

Large sandwich

Sample a set U with probability q.

Use proper hop lemma on U.
  Either fix $|U|$ negative vertices.

  Or: find pair s, t in S with negative proper h hop distance using S.
   At least h negative vertices sampled in (s, t, S).
   Expected size of S is h/q.

Let k be number of negative vertices.
  Set $q = 2\sqrt{h/k}$.
   Expected size of U is $qk = \Omega(\sqrt{kh})$.
   Expected size of S is $h/q = \Omega(\sqrt{kh})$.

Putting it together.

Steps:

Putting it together.

Steps:

Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.

Putting it together.

Steps:
Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.
Proper hops: $\tilde{O}(hm)$ time
to get remote $\Omega(\sqrt{nh})$ vertices

Putting it together.

Steps:

Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.

Proper hops: $\tilde{O}(hm)$ time

to get remote $\Omega(\sqrt{nh})$ vertices

Fix remote vertices $O(\sqrt{nh})$ in time $\tilde{O}(\sqrt{nh}rm)$

Total time $\tilde{O}((h^2 + \sqrt{nh}/h)m)$ to fix $\sqrt{nh}$ remote vertices.

Putting it together.

Steps:

Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.

Proper hops: $\tilde{O}(hm)$ time

to get remote $\Omega(\sqrt{nh})$ vertices

Fix remote vertices $O(\sqrt{nh})$ in time $\tilde{O}(\sqrt{nh}rm)$

Total time $\tilde{O}((h^2 + \sqrt{nh}/h)m)$ to fix $\sqrt{nh}$ remote vertices.

Putting it together.

Steps:

Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.

Proper hops: $\tilde{O}(hm)$ time

to get remote $\Omega(\sqrt{nh})$ vertices

Fix remote vertices $O(\sqrt{nh})$ in time $\tilde{O}(\sqrt{nh}rm)$

Total time $\tilde{O}((h^2 + \sqrt{nh}/h)m)$ to fix $\sqrt{nh}$ remote vertices.

Roughly $n/\sqrt{nh}$ iterations.

Putting it together.

Steps:

Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.

Proper hops: $\tilde{O}(hm)$ time

to get remote $\Omega(\sqrt{nh})$ vertices

Fix remote vertices $O(\sqrt{nh})$ in time $\tilde{O}(\sqrt{nh}rm)$

Total time $\tilde{O}((h^2 + \sqrt{nh}/h)m)$ to fix $\sqrt{nh}$ remote vertices.

Roughly $n/\sqrt{nh}$ iterations.

Time: $\tilde{O}(n/\sqrt{nh}(h^2 + \sqrt{nh}h)m) = \tilde{O}(\sqrt{n}(h^{3/2} + \sqrt{n}h^{1/2})m)$.

Putting it together.

Steps:

Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.

Proper hops: $\tilde{O}(hm)$ time

to get remote $\Omega(\sqrt{nh})$ vertices

Fix remote vertices $O(\sqrt{nh})$ in time $\tilde{O}(\sqrt{nh}rm)$

Total time $\tilde{O}((h^2 + \sqrt{nh}/h)m)$ to fix $\sqrt{nh}$ remote vertices.

Roughly $n/\sqrt{nh}$ iterations.

Time: $\tilde{O}(n/\sqrt{nh}(h^2 + \sqrt{nh}h)m) = \tilde{O}(\sqrt{n}(h^{3/2} + \sqrt{n}h^{1/2})m)$.

$h = n^{1/5}$

Putting it together.

Steps:
Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.
Proper hops: $\tilde{O}(hm)$ time
to get remote $\Omega(\sqrt{nh})$ vertices
Fix remote vertices $O(\sqrt{nh})$ in time $\tilde{O}(\sqrt{nh}rm)$
Total time $\tilde{O}((h^2 + \sqrt{nh}/h)m)$ to fix $\sqrt{nh}$ remote vertices.

Roughly $n/\sqrt{nh}$ iterations.
Time: $\tilde{O}(n/\sqrt{nh}(h^2 + \sqrt{nh}h)m) = \tilde{O}(\sqrt{n}(h^{3/2} + \sqrt{n}h^{1/2})m)$.

$h = n^{1/5} \implies \tilde{O}(n^{4/5}m)$.

Putting it together.

Steps:

Betweeness: $\tilde{O}(h^2 m)$ to get h-betweenness down to n/h.

Proper hops: $\tilde{O}(hm)$ time

to get remote $\Omega(\sqrt{nh})$ vertices

Fix remote vertices $O(\sqrt{nh})$ in time $\tilde{O}(\sqrt{nh}rm)$

Total time $\tilde{O}((h^2 + \sqrt{nh}/h)m)$ to fix $\sqrt{nh}$ remote vertices.

Roughly $n/\sqrt{nh}$ iterations.

Time: $\tilde{O}(n/\sqrt{nh}(h^2 + \sqrt{nh}h)m) = \tilde{O}(\sqrt{n}(h^{3/2} + \sqrt{n}h^{1/2})m)$.

$h = n^{1/5} \implies \tilde{O}(n^{4/5}m)$.

Oh my.