Dynamic algorithms for the Steiner tree problem

Anna Zych

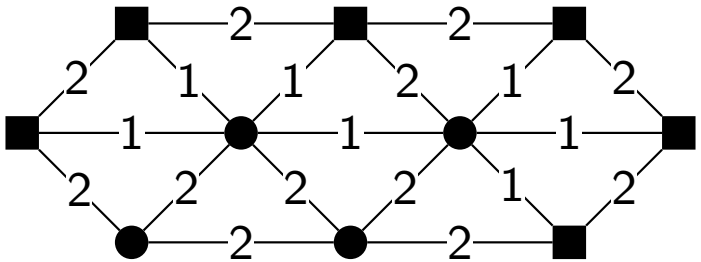
joint work with
Jakub Łącki, Jakub Oćwieja, Marcin Pilipczuk and Piotr Sankowski

Forum Informatyki Teoretycznej, 31 Jan 2015

The Steiner Minimum Tree Problem

Definition (The Steiner Minimum Tree Problem (SMT))

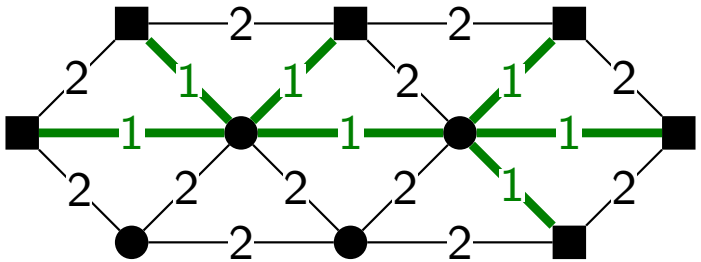
Input: A weighted graph $G = (V, E, w)$, a terminal set $S \subseteq V$.



The Steiner Minimum Tree Problem

Definition (The Steiner Minimum Tree Problem (SMT))

Input: A weighted graph $G = (V, E, w)$, a terminal set $S \subseteq V$.
Output: Minimum-weight tree T with $S \subseteq V(T)$.



The dynamic setting

Setting

- Initially, we get a graph $G = (V, E, w)$ whereas terminal set $S = \emptyset$.
- Then, requests arrive one by one.
- Request: `insert`(v) or `delete`(v).
  Add or delete vertex from terminal set S.
- At each step, maintain a good approximation of $\mathrm{Steiner}(G, S)$.

Efficiency measure.

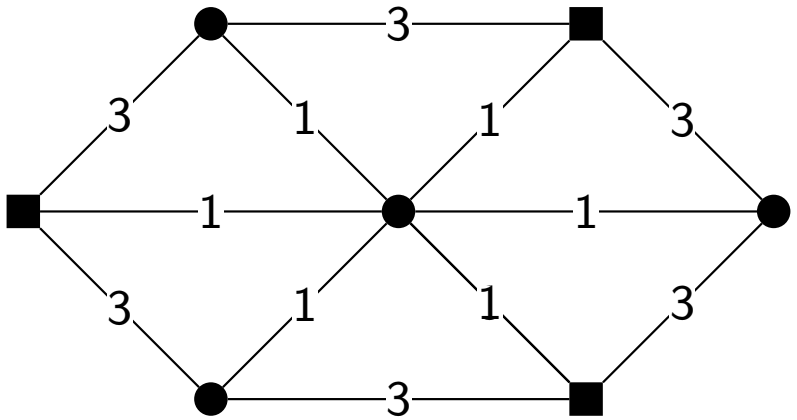
The total time for computation at each step.

Simple approximation

Definition (Metric closure)

For a weighted graph $G = (V, E, w)$,
the *metric closure* of G is a complete graph $\overline{G} = (V, \binom{V}{2}, \delta)$ with costs

$$\delta(uv) = \mathrm{dist}_G(u, v).$$

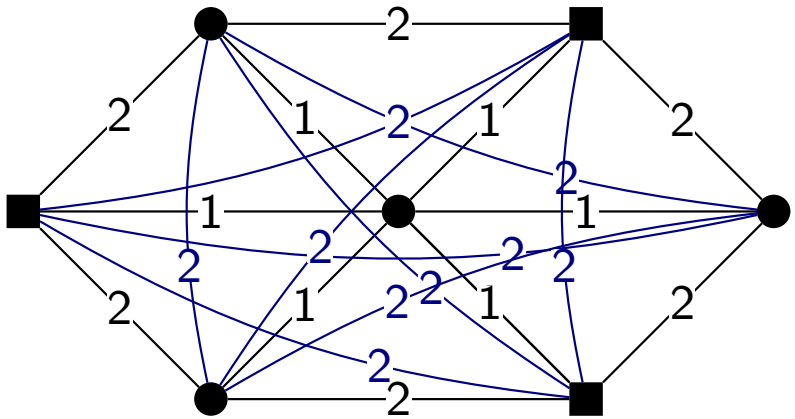


Simple approximation

Definition (Metric closure)

For a weighted graph $G = (V, E, w)$,
the *metric closure* of G is a complete graph $\overline{G} = (V, \binom{V}{2}, \delta)$ with costs

$$\delta(uv) = \mathrm{dist}_G(u, v).$$

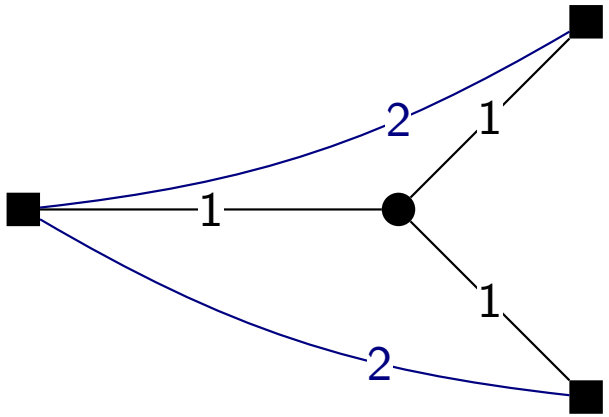


Simple approximation

Definition (Metric closure)

For a weighted graph $G = (V, E, w)$,
the *metric closure* of G is a complete graph $\overline{G} = (V, \binom{V}{2}, \delta)$ with costs

$$\delta(uv) = \mathrm{dist}_G(u, v).$$



Lemma

$$\mathrm{MST}\left(\overline{G}[S]\right) \leq 2\mathrm{Steiner}(G, S).$$

Straightforward approach

Theorem (Holm et. al.)

There exists a fully dynamic MSF algorithm, that for a graph on n vertices supports m edge additions and removals in $O(m\log^4 n)$ total time.

Straightforward approach

Theorem (Holm et. al.)

There exists a fully dynamic MSF algorithm, that for a graph on n vertices supports m edge additions and removals in $O(mlog^4 n)$ total time.

Approach:

- Maintain $\overline{G}[S]$ under required updates:
- `insert`(s):: add all the edges adjacent to s in $\overline{G}[S \cup s]$
- `delete`(s):: remove all the edges adjacent to s
- Use the algorithm of Holm et. al. to maintain MST on $\overline{G}[S]$
- This gives a 2-approximation with $O(n \log^4 n)$ update time

Straightforward approach

Theorem (Holm et. al.)

There exists a fully dynamic MSF algorithm, that for a graph on n vertices supports m edge additions and removals in $O(m\log^4 n)$ total time.

Approach:

- Maintain $\overline{G}[S]$ under required updates:
- `insert`(s):: add all the edges adjacent to s in $\overline{G}[S \cup s]$
- `delete`(s):: remove all the edges adjacent to s
- Use the algorithm of Holm et. al. to maintain MST on $\overline{G}[S]$
- This gives a 2-approximation with $O(n \log^4 n)$ update time

Issues:

- Minor issue: the weights of edges in the metric closure are needed
- Bigger issue: linear time per update is not satisfactory

Our results

Setting	apx	update time	preprocessing time
general, full	$8k-4$	$\tilde{O}(kn^{1/k})$	$O(kmn^{1/k})$
general, full	$6+\varepsilon$	$\tilde{O}(\varepsilon^{-5}\sqrt{n}\log D)$	$\tilde{O}(\varepsilon^{-3}n\sqrt{n}\log D)$
general, incremental	$6+\varepsilon$	$\tilde{O}(\varepsilon^{-1}\sqrt{n})$	$\tilde{O}(\varepsilon^{-1}\sqrt{n}m)$
general, decremental	$6+\varepsilon$	$O(\varepsilon^{-1}\sqrt{n}\log n)$	$\tilde{O}(\sqrt{n}m)$
planar, full	$4+\varepsilon$	$O(\varepsilon^{-1}\log^6 n)$	$O(\varepsilon^{-1}n\log^2 n)$
planar, full	$2+\varepsilon$	$\tilde{O}(\varepsilon^{-5.5}\sqrt{n}\log D)$	$\tilde{O}(\varepsilon^{-5}n\log D)$
planar, incremental	$2+\varepsilon$	$O(\varepsilon^{-1}\log^3 n\log D)$	$\tilde{O}(\varepsilon^{-1}n\log D)$

Our results

Setting	apx	update time	preprocessing time
general, full	$8k-4$	$\tilde{O}(kn^{1/k})$	$O(kmn^{1/k})$
general, full	$6+\varepsilon$	$\tilde{O}(\varepsilon^{-5}\sqrt{n}\log D)$	$\tilde{O}(\varepsilon^{-3}n\sqrt{n}\log D)$
general, incremental	$6+\varepsilon$	$\tilde{O}(\varepsilon^{-1}\sqrt{n})$	$\tilde{O}(\varepsilon^{-1}\sqrt{n}m)$
general, decremental	$6+\varepsilon$	$O(\varepsilon^{-1}\sqrt{n}\log n)$	$\tilde{O}(\sqrt{n}m)$
planar, full	$4+\varepsilon$	$O(\varepsilon^{-1}\log^6 n)$	$O(\varepsilon^{-1}n\log^2 n)$
planar, full	$2+\varepsilon$	$\tilde{O}(\varepsilon^{-5.5}\sqrt{n}\log D)$	$\tilde{O}(\varepsilon^{-5}n\log D)$
planar, incremental	$2+\varepsilon$	$O(\varepsilon^{-1}\log^3 n\log D)$	$\tilde{O}(\varepsilon^{-1}n\log D)$

A more careful approach

Before:

dynamic MST algorithm applied to dynamically maintained **metric closure**

Now:

dynamic MST algorithm applied to dynamically maintained **bipartite emulator**

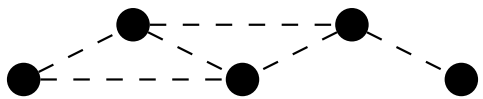
Idea:

The bipartite emulator approximates the distances in the metric closure. It is, however, sparser and hence can be modified with fewer edge modifications.

Bipartite emulators

Definition

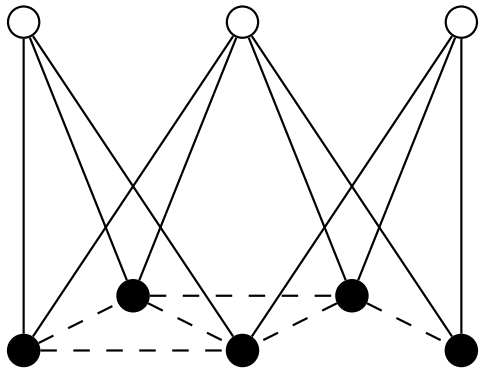
An α-bipartite emulator of a graph $G = (V, E, w)$



Bipartite emulators

Definition

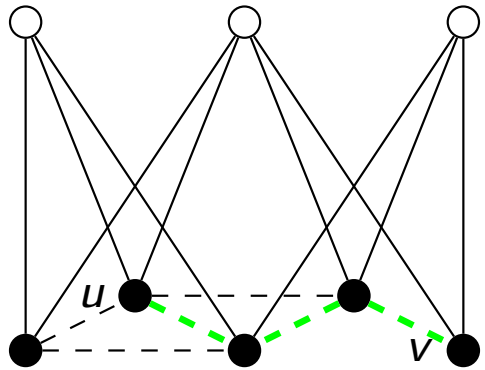
An α-bipartite emulator of a graph $G = (V, E, w)$ is a bipartite graph $B = (V \cup N, E', w')$ such that



Bipartite emulators

Definition

An α-bipartite emulator of a graph $G = (V, E, w)$ is a bipartite graph $B = (V \cup N, E', w')$ such that for every $u, v \in V$

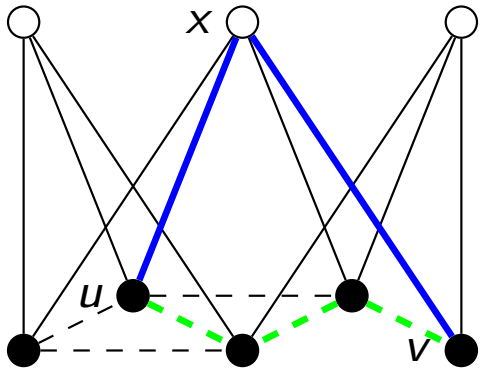


Bipartite emulators

Definition

An α-bipartite emulator of a graph $G = (V, E, w)$ is a bipartite graph $B = (V \cup N, E', w')$ such that for every $u, v \in V$ there exists a vertex $x \in N$ such that

$$w'(ux) + w'(vx) \leq \alpha \cdot \mathrm{dist}_G(u, v)$$

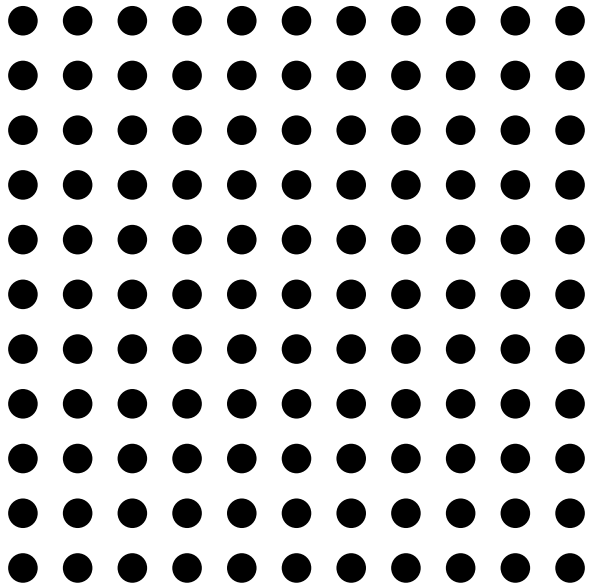


Thorup and Zwick distance oracle

Theorem (Thorup and Zwick)

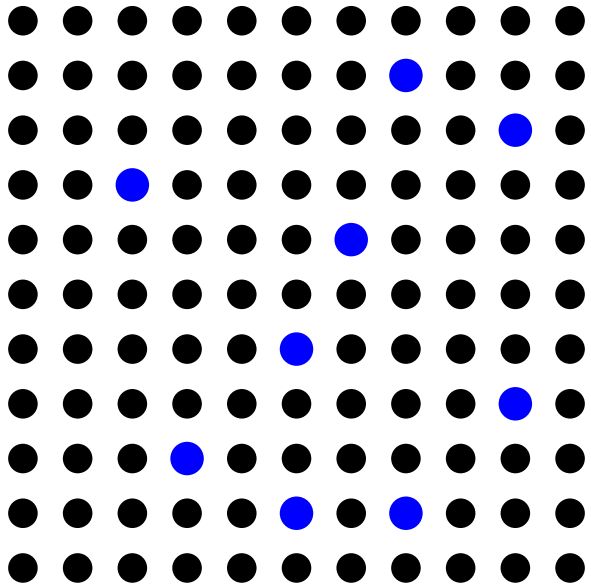
There is a data structure, which
takes $O(kmn^{1/k})$ *time to* ***preprocess***,
occupies $O(kn^{1+1/k})$ ***space***, *and*
allows to estimate distances in ***time*** $O(k)$.
The ***stretch*** *of the produced estimates is bounded from above by* $2k-1$
and no distance is underestimated.

Thorup and Zwick distance oracle for $k = 2$



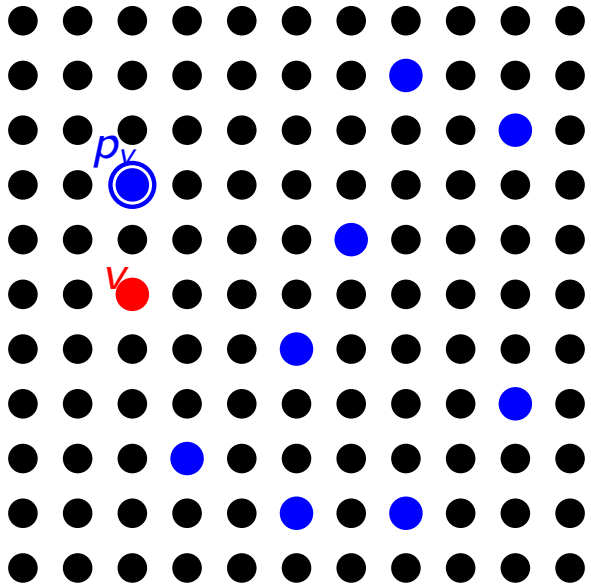
Thorup and Zwick distance oracle for $k = 2$

- Sample $\widetilde{\mathcal{O}}(\sqrt{n})$ *portals*: $P \leftarrow \mathit{Sample}(1/\sqrt{n}, V)$.



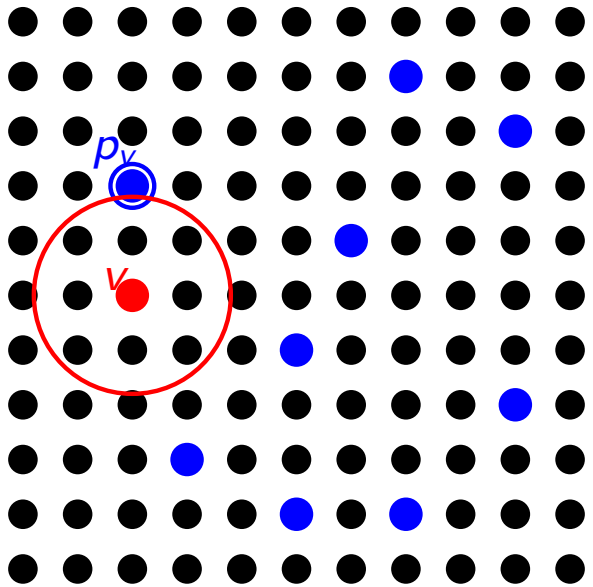
Thorup and Zwick distance oracle for $k = 2$

- Sample $\widetilde{\mathcal{O}}(\sqrt{n})$ *portals*: $P \leftarrow \mathit{Sample}(1/\sqrt{n}, V)$.
- Choose p_v to be the closest portal to v.



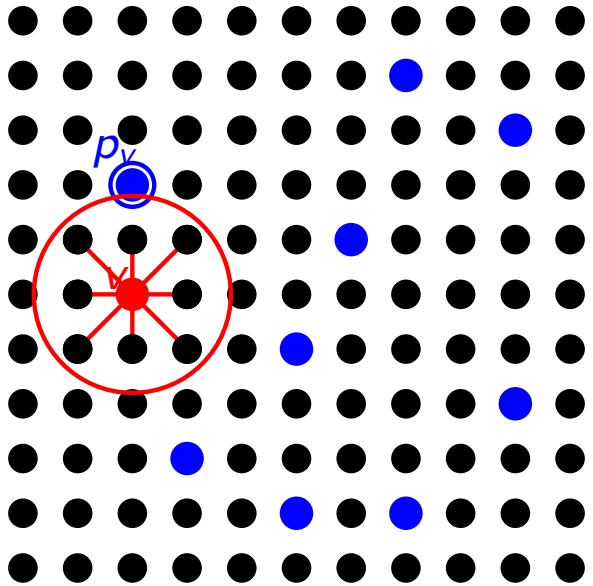
Thorup and Zwick distance oracle for $k = 2$

- Sample $\widetilde{\mathcal{O}}(\sqrt{n})$ *portals*: $P \leftarrow \mathit{Sample}(1/\sqrt{n}, V)$.
- Choose p_v to be the closest portal to v.
- $B(v) = \{w \in V : \mathrm{dist}(v, w) < \mathrm{dist}(v, p_v)\}$. Then $|B(v)| \in \widetilde{\mathcal{O}}(\sqrt{n})$.



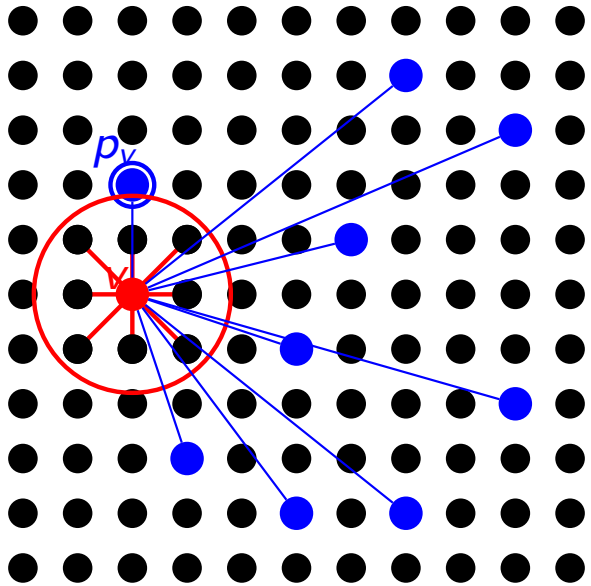
Thorup and Zwick distance oracle for $k = 2$

- Sample $\widetilde{\mathcal{O}}(\sqrt{n})$ *portals*: $P \leftarrow \mathit{Sample}(1/\sqrt{n}, V)$.
- Choose p_v to be the closest portal to v.
- $B(v) = \{w \in V : \mathrm{dist}(v, w) < \mathrm{dist}(v, p_v)\}$. Then $|B(v)| \in \widetilde{\mathcal{O}}(\sqrt{n})$.
- Every $v \in V$ remembers distances to $B(v)$



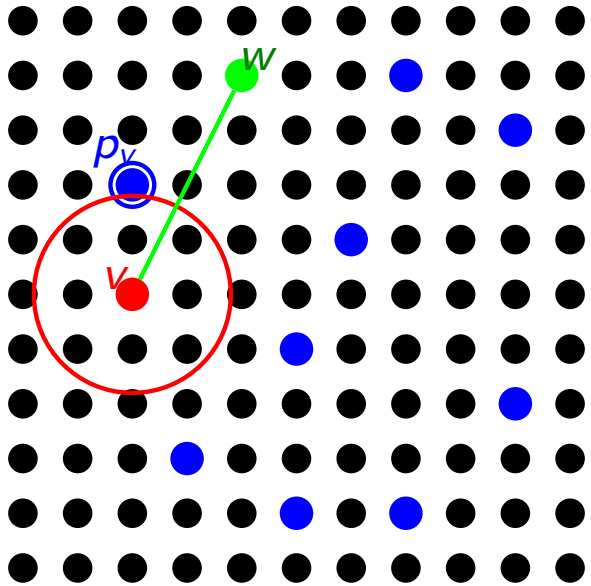
Thorup and Zwick distance oracle for $k = 2$

- Sample $\widetilde{\mathcal{O}}(\sqrt{n})$ *portals*: $P \leftarrow \mathit{Sample}(1/\sqrt{n}, V)$.
- Choose p_v to be the closest portal to v.
- $B(v) = \{w \in V : \mathrm{dist}(v, w) < \mathrm{dist}(v, p_v)\}$. Then $|B(v)| \in \widetilde{\mathcal{O}}(\sqrt{n})$.
- Every $v \in V$ remembers distances to $B(v) \cup P$.



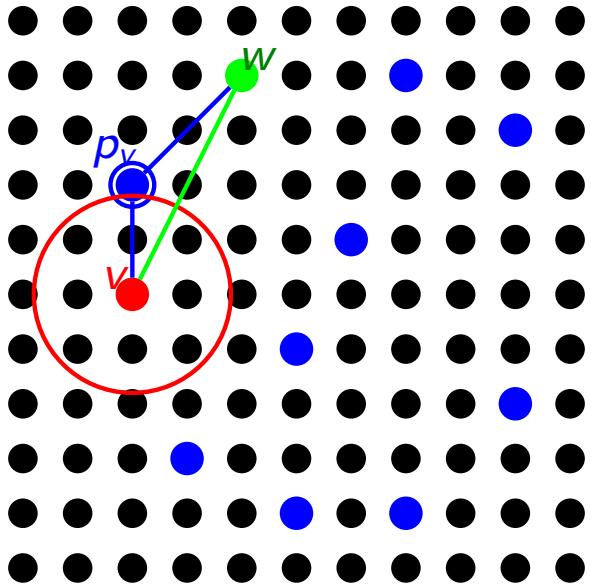
Thorup and Zwick distance oracle for $k = 2$

- Sample $\widetilde{\mathcal{O}}(\sqrt{n})$ *portals*: $P \leftarrow \mathit{Sample}(1/\sqrt{n}, V)$.
- Choose p_v to be the closest portal to v.
- $B(v) = \{w \in V : \mathrm{dist}(v, w) < \mathrm{dist}(v, p_v)\}$. Then $|B(v)| \in \widetilde{\mathcal{O}}(\sqrt{n})$.
- Every $v \in V$ remembers distances to $B(v) \cup P$.



Thorup and Zwick distance oracle for $k = 2$

- Sample $\widetilde{\mathcal{O}}(\sqrt{n})$ *portals*: $P \leftarrow \mathit{Sample}(1/\sqrt{n}, V)$.
- Choose p_v to be the closest portal to v.
- $B(v) = \{w \in V : \mathrm{dist}(v, w) < \mathrm{dist}(v, p_v)\}$. Then $|B(v)| \in \widetilde{\mathcal{O}}(\sqrt{n})$.
- Every $v \in V$ remembers distances to $B(v) \cup P$.

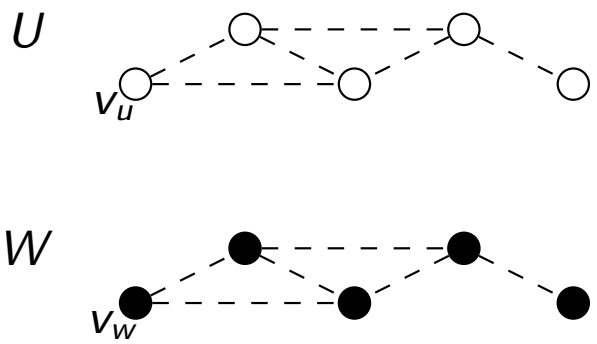


Lemma

If $w \notin B(v)$ *then* $\mathrm{dist}(v, p_v) + \mathrm{dist}(p_v, w) \leq 3\mathrm{dist}(v, w)$.

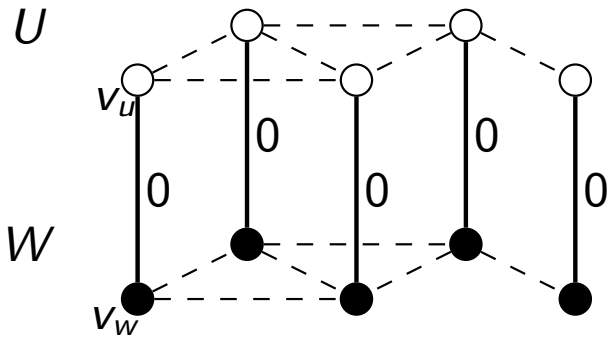
From TZ-oracle to a bipartite emulator

- make two copies of V: U and W. Now every vertex occurs in two copies: v_u and v_w.



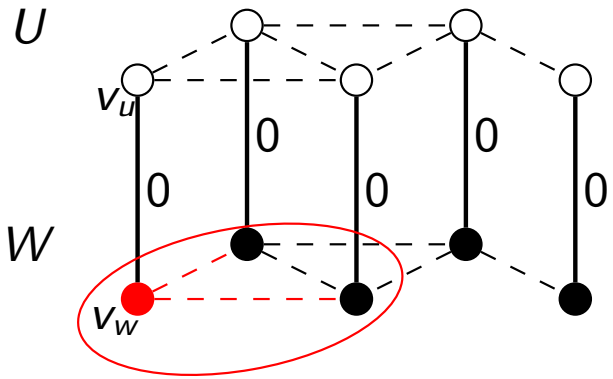
From TZ-oracle to a bipartite emulator

- make two copies of V: U and W. Now every vertex occurs in two copies: v_u and v_w.
- join the corresponding vertices via edges with 0 weight



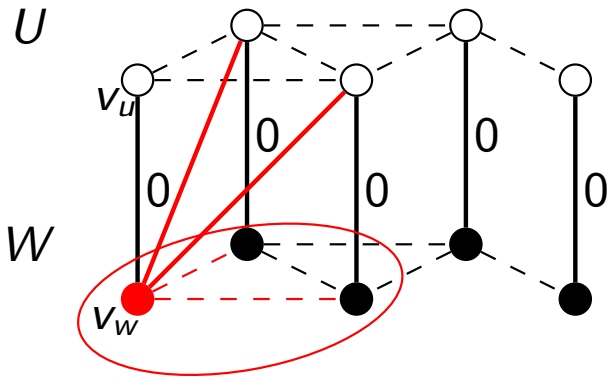
From TZ-oracle to a bipartite emulator

- make two copies of V: U and W. Now every vertex occurs in two copies: v_u and v_w.
- join the corresponding vertices via edges with 0 weight
- for every vertex $v \in V$
  - for every $\overline{v} \in B(v)$



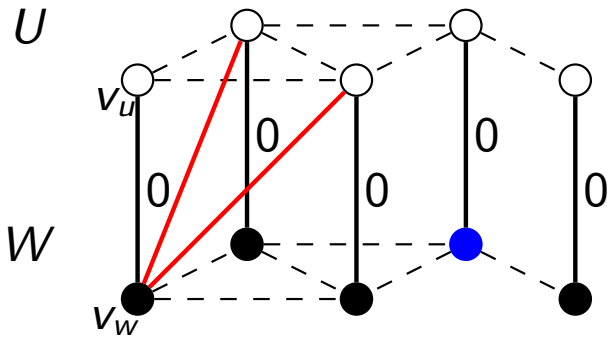
From TZ-oracle to a bipartite emulator

- make two copies of V: U and W. Now every vertex occurs in two copies: v_u and v_w.
- join the corresponding vertices via edges with 0 weight
- for every vertex $v \in V$
  - for every $\overline{v} \in B(v)$ add the distances between v_w and $\overline{v}_u$



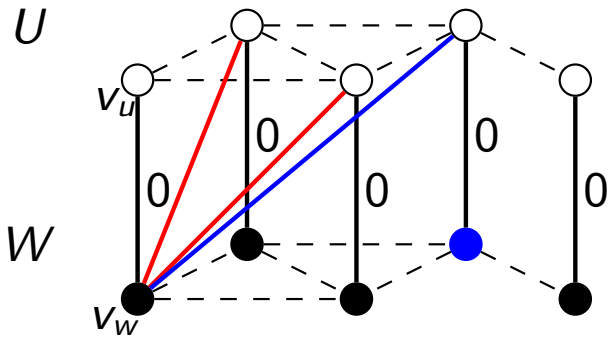
From TZ-oracle to a bipartite emulator

- make two copies of V: U and W. Now every vertex occurs in two copies: v_u and v_w.
- join the corresponding vertices via edges with 0 weight
- for every vertex $v \in V$
  - for every $\overline{v} \in B(v)$ add the distances between v_w and $\overline{v}_u$
  - for every portal $p \in P$



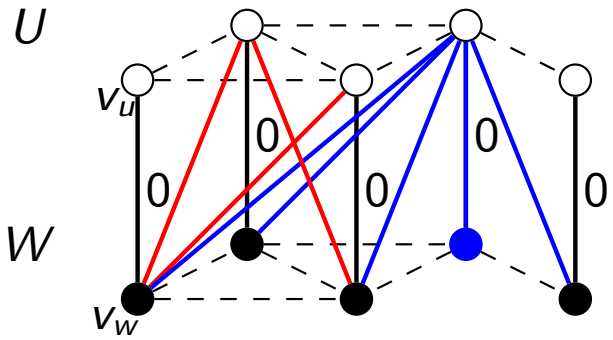
From TZ-oracle to a bipartite emulator

- make two copies of V: U and W. Now every vertex occurs in two copies: v_u and v_w.
- join the corresponding vertices via edges with 0 weight
- for every vertex $v \in V$
  - for every $\overline{v} \in B(v)$ add the distances between v_w and $\overline{v}_u$
  - for every portal $p \in P$ add distances between v_w and p_u



From TZ-oracle to a bipartite emulator

- make two copies of V: U and W. Now every vertex occurs in two copies: v_u and v_w.
- join the corresponding vertices via edges with 0 weight
- for every vertex $v \in V$
  - for every $\overline{v} \in B(v)$ add the distances between v_w and $\overline{v}_u$
  - for every portal $p \in P$ add distances between v_w and p_u

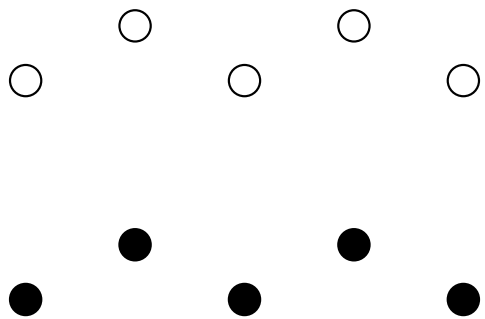


The algorithm

- Build $B(G) = (U \cup W, E, w')$-the bipartite emulator for G.
- Maintain dynamically a subgraph $H(G) \subseteq B(G)$:

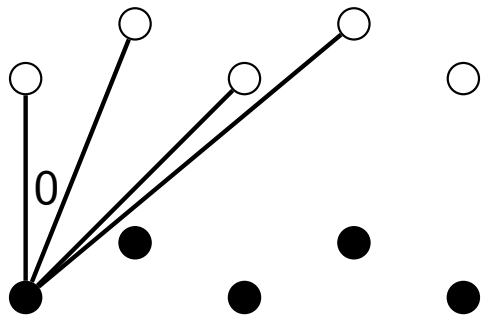
The algorithm

- Build $B(G) = (U \cup W, E, w')$-the bipartite emulator for G.
- Maintain dynamically a subgraph $H(G) \subseteq B(G)$:
  - Set $H(G) := (U \cup W, \emptyset)$



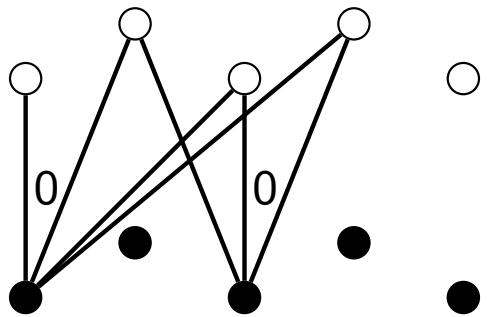
The algorithm

- Build $B(G) = (U \cup W, E, w')$-the bipartite emulator for G.
- Maintain dynamically a subgraph $H(G) \subseteq B(G)$:
  - Set $H(G) := (U \cup W, \emptyset)$
  - `insert`(s) : add all the edges adjacent to s_w to $H(G)$



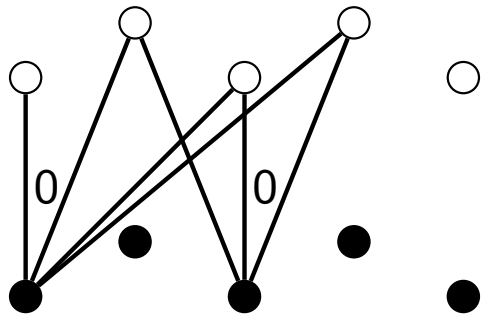
The algorithm

- Build $B(G) = (U \cup W, E, w')$-the bipartite emulator for G.
- Maintain dynamically a subgraph $H(G) \subseteq B(G)$:
  - Set $H(G) := (U \cup W, \emptyset)$
  - `insert`(s) : add all the edges adjacent to s_w to $H(G)$



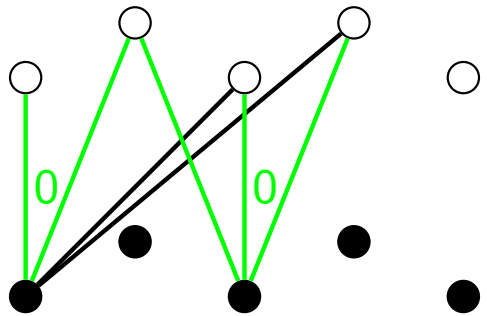
The algorithm

- Build $B(G) = (U \cup W, E, w')$-the bipartite emulator for G.
- Maintain dynamically a subgraph $H(G) \subseteq B(G)$:
  - Set $H(G) := (U \cup W, \emptyset)$
  - `insert`(s) : add all the edges adjacent to s_w to $H(G)$
  - `delete`(s) : remove all the edges adjacent to s_w from $H(G)$



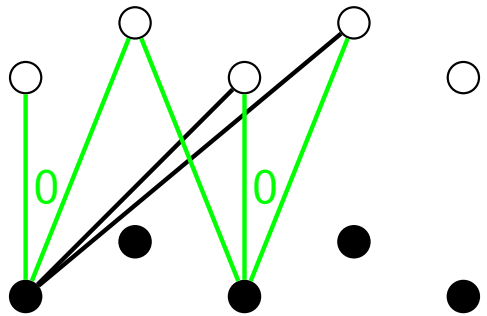
The algorithm

- Build $B(G) = (U \cup W, E, w')$-the bipartite emulator for G.
- Maintain dynamically a subgraph $H(G) \subseteq B(G)$:
  - Set $H(G) := (U \cup W, \emptyset)$
  - `insert`(s) : add all the edges adjacent to s_w to $H(G)$
  - `delete`(s) : remove all the edges adjacent to s_w from $H(G)$
  - Maintain MST on $H(G)$ using the algorithm of Holm et. al.



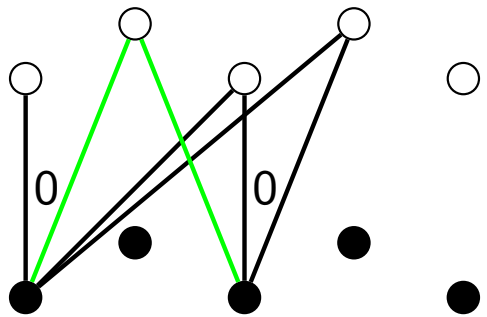
The algorithm

- Build $B(G) = (U \cup W, E, w')$-the bipartite emulator for G.
- Maintain dynamically a subgraph $H(G) \subseteq B(G)$:
  - Set $H(G) := (U \cup W, \emptyset)$
  - `insert`(s) : add all the edges adjacent to s_w to $H(G)$
  - `delete`(s) : remove all the edges adjacent to s_w from $H(G)$
  - Maintain MST on $H(G)$ using the algorithm of Holm et. al.
- Since $|B(s) \cup P| \in O(\sqrt{n})$, updates take $O(\sqrt{n}\log^4 n)$ time



The algorithm

- Build $B(G) = (U \cup W, E, w')$-the bipartite emulator for G.
- Maintain dynamically a subgraph $H(G) \subseteq B(G)$:
  - Set $H(G) := (U \cup W, \emptyset)$
  - `insert`(s) : add all the edges adjacent to s_w to $H(G)$
  - `delete`(s) : remove all the edges adjacent to s_w from $H(G)$
  - Maintain MST on $H(G)$ using the algorithm of Holm et. al.
- Since $|B(s) \cup P| \in O(\sqrt{n})$, updates take $O(\sqrt{n}\log^4 n)$ time



Lemma

If we cut off the leaves in U from the maintained tree, we get a 2α-approximation of $MST(\overline{G}[S])$

Open ends

- In all our algorithms, for simplicity, we approximate MST rather than SMT. Can one use better approximations in the dynamic setting, and still guarantee a reasonable update time?
- For what other hard problems can one develop efficient dynamic approximation algorithms?