Dynamic graph algorithms with applications

Mikkel Thorup
AT&T Labs–Research

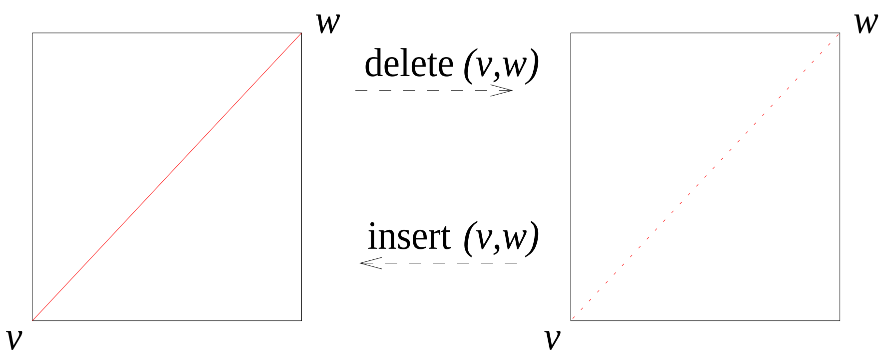
Dynamic data structures in static problems

Standard example: priority queue in greedy algorithm such as Dijkstra's single source shortest path algorithm.

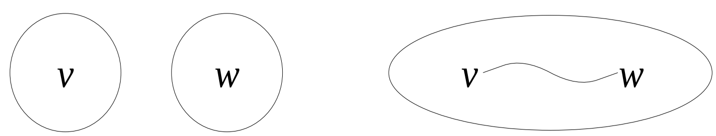
Here we consider dynamic graph algorithms maintaining properties and objects in a changing graph.

Dynamic graph algorithms

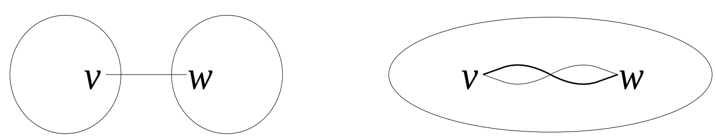
Updates



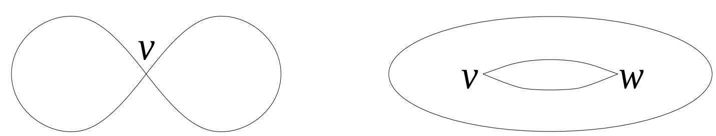
Connectivity | disconnected? | connected *(v,w)*



2-edge-connectivity | bridge? | 2-edge-connected *(v,w)*



Biconnectivity | articulatiion point? | biconnectied *(v,w)*

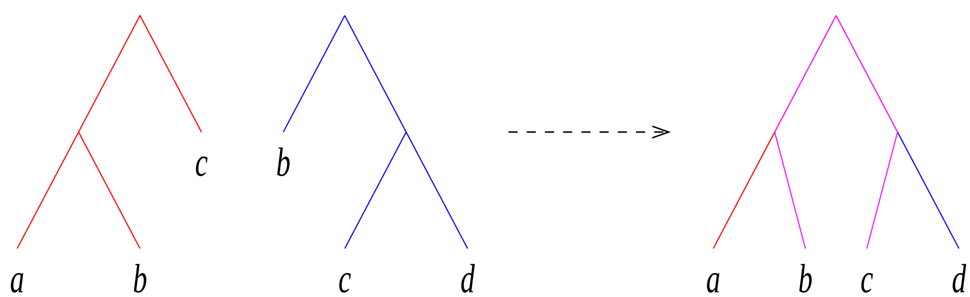


Minimum spanning tree (MST)

Update MST during insertion and deletion

Applications

- Constructing tree from homeomorphic sub-trees



- Unique perfect matching

We shall also talk about dynamic shortest paths and their applications.

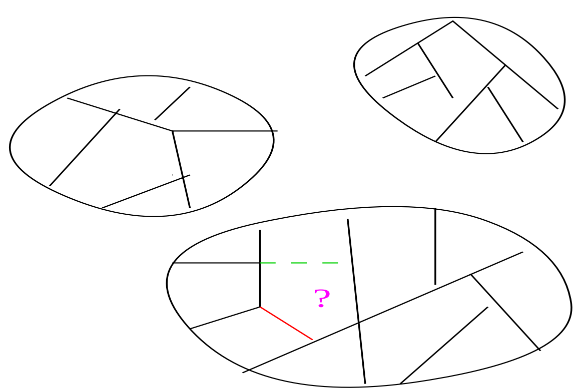
Connectivity of $G = (V, E)$, $|V| = n$,
during m updates, starting $E = \emptyset$.

Maintain spanning forest F

(for dynamic forest F "everything" takes $O(\log n)$ time)

insert$((v, w))$ if v and w disconnected in F,
$F := F \cup \{(v, w)\}$

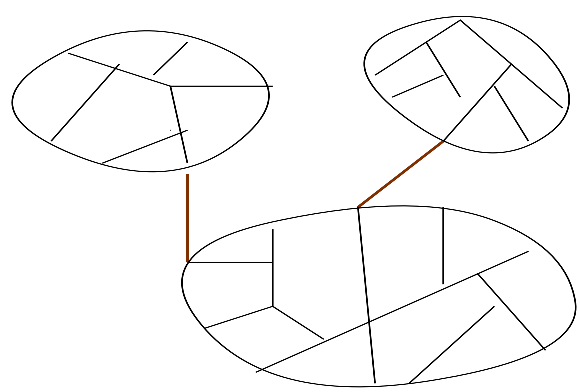
delete$((v, w))$ if $(v, w) \in F$, seek replacement edge from E reconnecting $F \setminus \{(v, w)\}$



Introduce levels $\ell : E \to \{0, ..., \lfloor \log_2 n \rfloor\}$

$G_i = (V, \{e \in E : \ell(e) \geq i\})$

(i) F ℓ-maximal spanning forest
$\Rightarrow F_i = F \cap G_i$ spanning forest of G_i



(ii) Components of G_i contain $\leq n/2^i$ vertices.

Idea: amortize over level increases

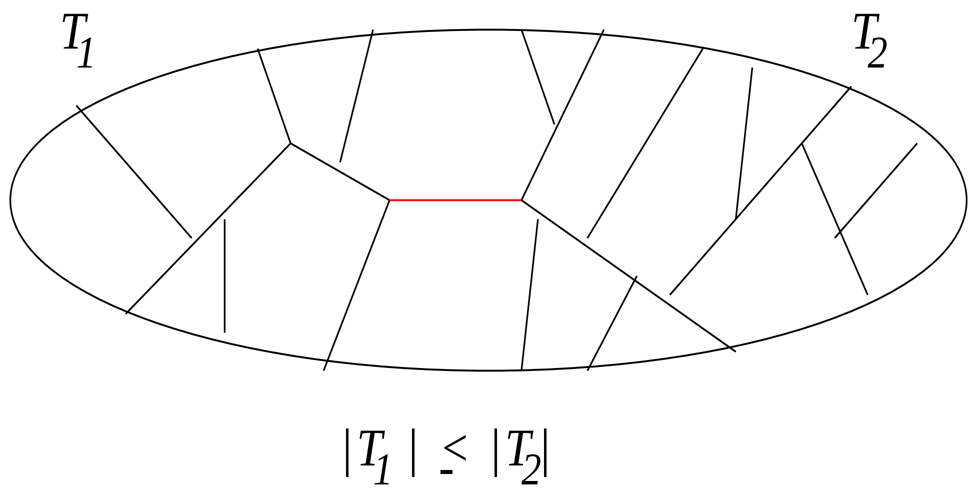
Insert((v, w)): $\ell((v, w)) := 0$.
If v and w disconnected in F, $F \cup \{(v, w)\}$

Delete(e): If $e \in F$, $F := (F \setminus \{e\}) \cup$ Replace(e)

Replace(e)

For $i := \ell(e)$ downto 0 do

$\triangleright$ no replacement edge on level $> i$



$\triangleright$ level i replacement connect T_1 and T_2

$\triangleright$ $|T_1| \leq n/2^{i+1}$

For all level i edges $f \in T_1$: $\ell(f) := i+1$.

Consider level i edges (v, w), $v \in T_1$, one by one:

If $w \notin T_1$, return $\{(v, w)\}$.

Else $\ell((v, w)) := i + 1$.

Return $\emptyset$

Each statement iterated $\leq m \log_2 n$ times
$\wedge$ each statement supported in $O(\log n)$ time
$\Rightarrow O(m \log^2 n)$ total time.

Decremental MST of $G = (V, E)$, $|V| = n$, $|E| = m$.

Maintain minimum spanning forest F

delete$((v, w))$ if $(v, w) \in F$, seek lightest replacement from E reconnecting $F \setminus \{(v, w)\}$

Introduce levels $\ell : E \to \{0, ..., \lfloor \log_2 n \rfloor\}$

$G_i = (V, \{e \in E : \ell(e) \geq i\})$

(i) F ℓ-maximal spanning forest
$\Rightarrow F_i = F \cap G_i$ spanning forest of G_i

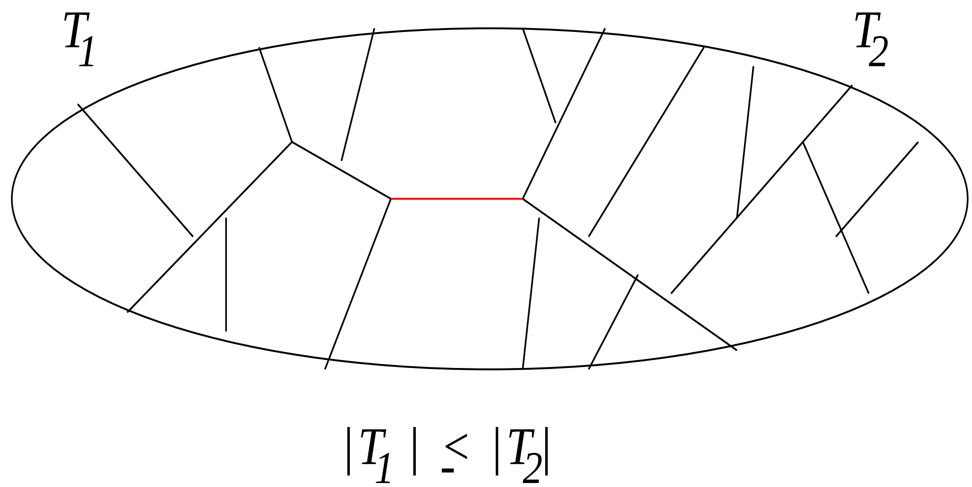
(ii) Components of G_i contain $\leq n/2^i$ vertices.

Initially F minimum spanning forest
and $\forall e \in E : \ell(e) = 0$

Delete(e): if $e \in F$, $F := (F \setminus \{e\}) \cup \text{Replace}(e)$

Replace(e)

For $i := \ell(e)$ downto 0 do



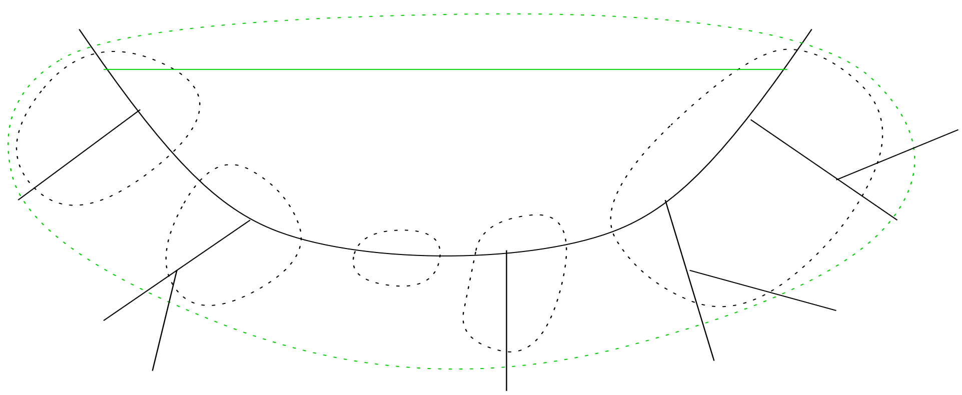
For all level i edges $f \in T_1$: $\ell(f) := i+1$.
Consider level i edges (v, w), $v \in T_1$, one by one, in order of increasing weight:
If $w \notin T_1$, return $\{(v, w)\}$.
Else $\ell((v, w)) := i + 1$.

Return $\emptyset$

$\rightarrow$ fully-dynamic polylogarimic MST using general reduction of Henzinger and King (ICALP'97).

2-edge-connectivity of $G = (V, E)$, $|V| = n$, during m updates, starting with $E = \emptyset$.

Maintain spanning forest F



$(v, w) \in E$ covers path $v \cdots w$ from v to w in F

Lem x and y 2-edge connected $\iff$ $x \cdots y$ covered.

Introduce levels $\ell : E \setminus F \to \{0, ..., \lfloor \log_2 n \rfloor\}$

$G_i = (V, F \cup \{e \in E : \ell(e) \geq i\})$

(i) 2-edge connected components of G_i contain $\leq n/2^i$ vertices.

For each $f \in F$ maintain highest level of covering edge, denoted $c(f)$. If f bridge, $c(f) = -1$.

Connected(x, y): $\forall e \in x \cdots y : c(e) \geq 0$.

Insert$((v, w))$
    If v and w disconnected in F,
        $F \cup \{(v, w)\}$.
        $c((v, w)) := -1$.
    Else
        $\ell((v, w)) := 0$
        call Cover$\ell((v, w))$

Cover$((v, w))$
    For all $f \in v \cdots w$ with $c(f) < \ell((v, w))$,
        $c(f) := (v, w)$.

Delete(e)

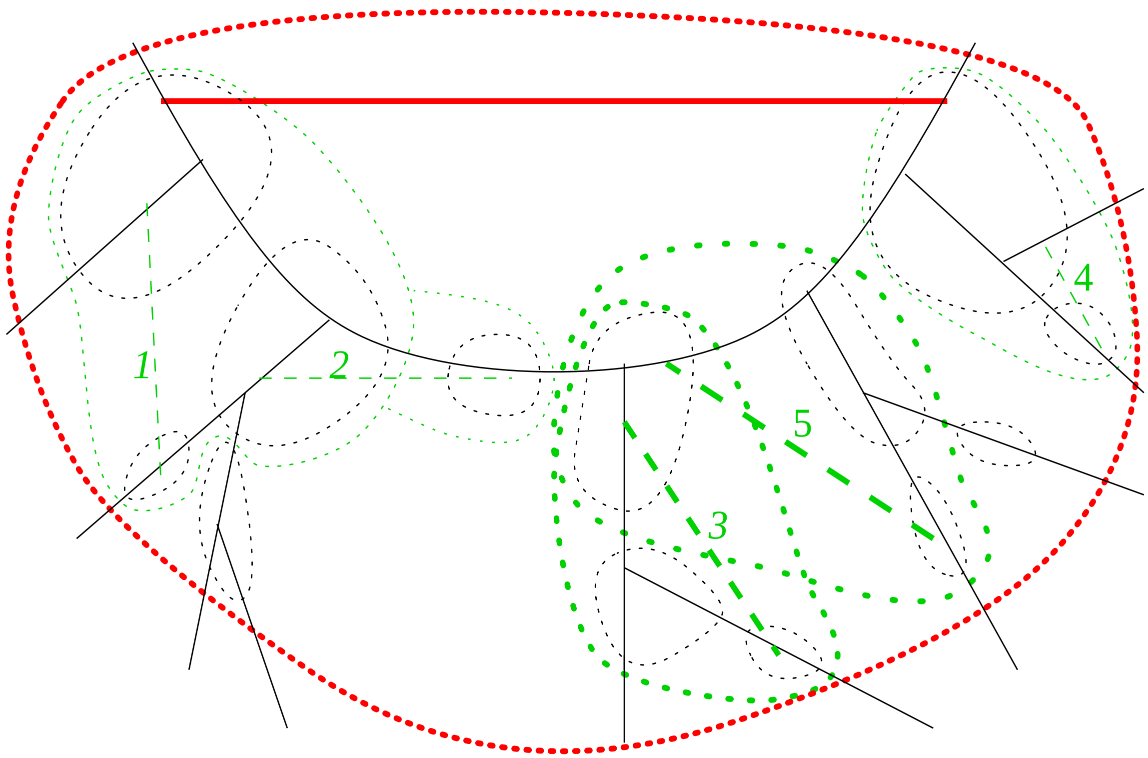
if $e \in F$,

swap e in F with covering edge f on highest level

$(c(f), \ell(e)) := (\ell(f), c(e))$

$e := f$

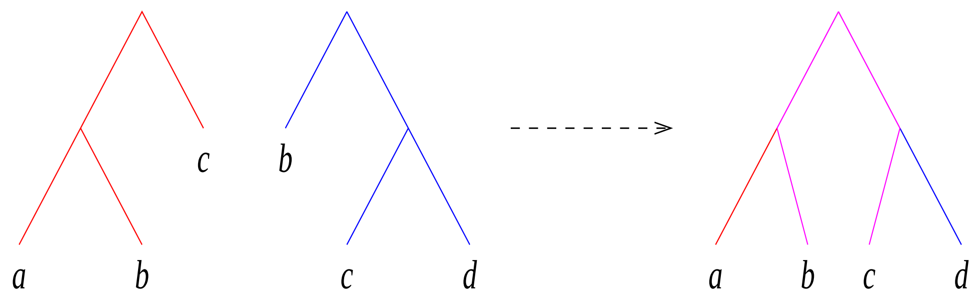
Recover(e)



Cover only called $O(m \log n)$ times, each at polylogarithmic cost using data structures for dynamic forests.

Applications

Constructing tree from homeomorphic subtrees



Reduction to decremental connectivity by Henzinger, King, and Warnow (SODA'96)

Small trees represented as triples $((a, b), c) \in T$ with $a, b, c \in A$.

Obs If $((a, b), c) \in T$, a and b must descend from same child of root.

Make child for each component of $G = (A, \{(a, b) : ((a, b), c) \in T\})$

This resolves all triples $((a, b), c)$ with c disconnected from b (and a) in G.

Grandchildren found by removing edge (a, b) for each resolved triple $((a, b), c)$.

Unique perfect matchings

Reduction to decremental 2-edge connectivity by Gabow, Kaplan, and Tarjan (STOC'99)

Lem (Kotzig 1959) A unique perfect matching has a bridge.

Constructing unique perfect matching, if any

$M := \emptyset$
While component C of G has bridge (v, w)
  If components of $C \setminus \{(v, w)\}$ both have odd number of vertices,
    $M := M \cup \{(v, w)\}$.
    Delete all edges incident to v and w from G.
  Elseif components of $C \setminus \{(v, w)\}$ both have even number of vertices,
    Delete (v, w) from G.
  Else G has no perfect matching. EXIT.
If G empty, return M;
Else G has no perfect matching.

...another application

Thm (Petersen 1891) Every bridgeless 3-regular graph has a perfect matching.

Biedl, Bose, Demaine, and Lubiw (SODA'99) have used dynamic 2-edge connectivity to construct such a perfect matching in $\tilde{O}(n)$ time, improving over the bound the $\tilde{O}(n^{3/2})$ obtained using the general time bound for matching when $m = O(n)$.

Shortest paths: some techniques

Ramalingam and Reps suggested lazy Dijkstra for single source shortest paths.

- Running time proportional to # edges incident to vertices changing distance from source.
- Works great in practice.

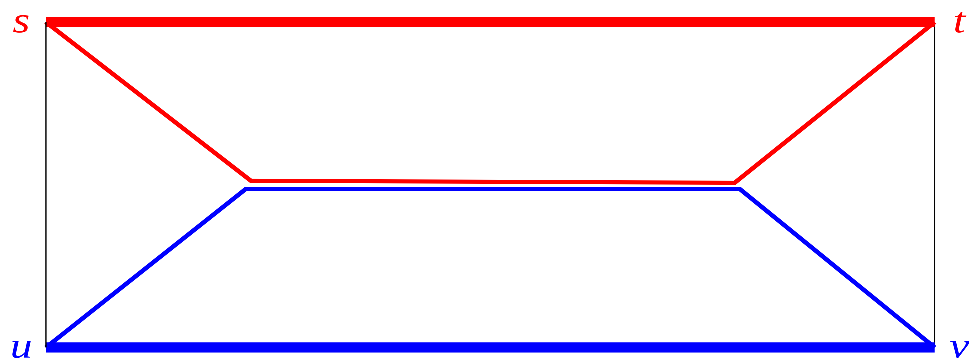
Recent break-through by Demetrescu and Italiano on all pairs-shortest path:

- Each vertex update supported in $\tilde{O}(n^2)$ time.
- Works even better in practice.
- Current best has update time $O(n^2(\log n + \log^2(m/n))$ and works for arbitrary weights [Thorup].

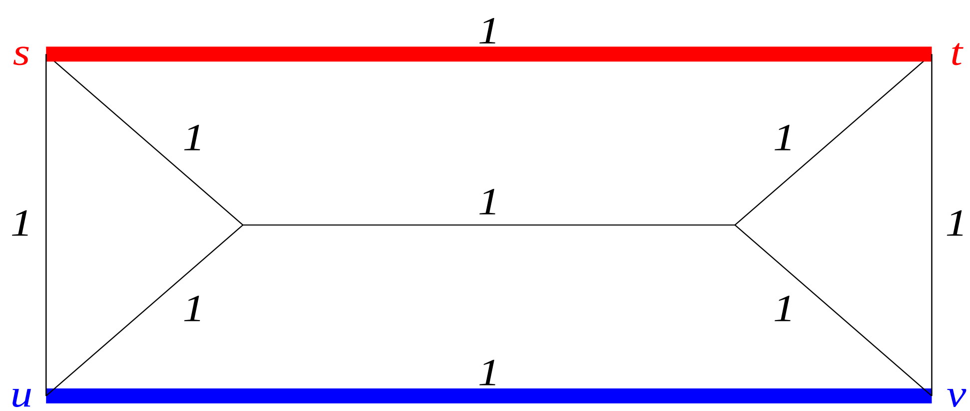
Internet traffic engineering

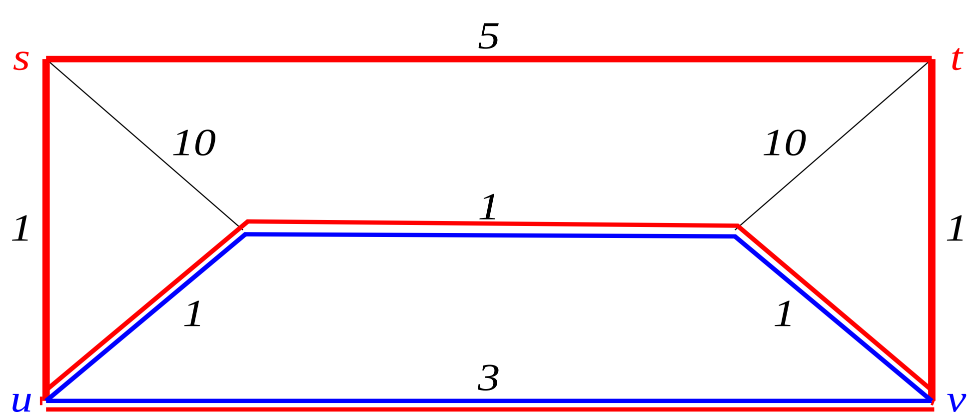
Demand of 1 for (s,t) and (u,v)

General routing: max load 2/3



Shortest path routing: max load 1 or 3/4





Optimizing shortest path routing with dynamic shortest paths [Fortz Thorup]

Finding weights minimizing max utilization (load/capacity) within factor 3/2 is NP-hard.

Cisco default: link weight inverse of capacity.

Local search heuristics

Iteratively change a weight that reduces max-utilization.

When inner loop tries a weight change, new shortest path routes are found and evaluated.

Ramalingam and Reps gave speed-up by factor 15 with 100 nodes and 300 edges.

Gained 50% over Cisco default on AT&T IP backbone.

Got within few percent of optimal general routing.

Concluding remarks

Talked about dynamic graph algorithms and their applications in solving static problems

Similar to priority queues in greedy algorithms

Challenge: dynamic reachability between fixed s and t for sparse graphs
$\rightarrow$ better augmenting paths max-flow algorithms.