Planning for Fast Connectivity Updates

Mihai Pătraşcu

...until June 2008
Jobs, anyone?

Mikkel Thorup
Connectivity in *Changing Graphs*

- **insert**(edge)
- **delete**(edge)
- **connected**(u, v) = are u and v in the same component?

Well understood: \( t_q = \Theta \left( \frac{\log n}{\log \frac{t_u}{\log n}} \right) \)

Amortized:
- randomized \( t_u = O(\log n \ (\log \log n)^3) \) [T’05]
- deterministic \( t_u = O(\log^2 n) \) [HLT’98]

Worst case: deterministic \( t_u = O(\sqrt{n}) \) [F’83]
What’s wrong with amortized?

• mathematician:
  “Nothing, but deamortization is a big challenge. It’s hard therefore it’s interesting.”

• CS theorist:
  “May spend $O(n)$ per update! Bad for practice! And practice is always our main motivation.”

• CS practitioner:
  “Does spend $O(n)$ /update at worst possible times... But I don’t really care anyway.”

starting with empty graph
Emergency Planning

Preprocess graph during good times
   ... when emergency comes, understand what happened quickly

   more???

If one edge goes down, what happens to:

• connectivity:  graph bridges
• reachability:  [King-Sagert STOC’99]
• shortest path:  [Hershberger-Suri FOCS’01, Roditty-Zwick ICALP’05]
• APSP:  [Chowdhury-Ramachandran’02, Demetrescu-Thorup SODA’02]

=> Nice way to understand graph structure (algorithmically)
Planning for Connectivity

1. Preprocessing: graph with \( m \) edges
   - time \( \text{poly}(m) \)
   - space \( O(m) \)

2. Batched updates: \( d \) edge deletions, insertions
   - “understand connectivity” in time \( O(d \ lg^2 m \ lg lg m) \)
   - \( \Rightarrow \) # connected components
   - \( \Rightarrow \) size of each connected component
   - \( \Rightarrow \) oracle ~~~

3. Oracle query: \( \text{root}(v) \) = ID of connected component
   - time \( O(lg lg m) \) per query

optimal, actually
Idea 1: Don’t worry, be happy

Any respectable graph is an expander...

• preprocessing: I’m feeling lucky

• batched deletions: $O(d/\Phi)$ time

• oracle query: $O(1)$
Exploring Expanders

Grow components around deleted edges:
• **isolated:** no adjacent edges
  => found connected component
• **active:** \#deleted edges > \( \Phi \) \#original edges
  => keep growing component
• **passive:** otherwise
  not enough deleted edges to destroy expansion
  => eventually, all passive comps will unite into one giant comp
  => no need to explore further

Can only explore \( O(d/\Phi) \) edges before everything becomes passive.
Idea 2: Worry later, be happy

Remove cuts sparser than $\Phi$ for “later”

$\Rightarrow$ partition into expanders

“Later”? [Henzinger-King STOC’95]

- set $\Phi = 1/(2 \lg m)$
  $\Rightarrow$ update time $O(d/\Phi)$ still ok

- for every cut, charge $\Phi$ to each edge on smaller side
  $\Rightarrow$ # edges cut $\leq \$ charged

- each edge charged at most $\lg m$ times
  $\Rightarrow$ total $\$ \leq \Phi \lg m \leq m/2$
Hierarchical Decomposition

• Level 1: original graph promote sparse cuts to level 2

• Level 2: at most $\frac{m}{2}$ edges contract level-1 expanders promote sparse cuts to level 3

• Level 3: at most $\frac{m}{4}$ edges

... up to $\leq \lg m$ levels
Handling Deletions

In each expander, run expander algorithm.
If an expander is split:

let \( k = \) #edges on smaller side

=> at least \( \Phi_k \) edges deleted in expander

=> can afford to inspect edges on smaller side @ next level
Trouble with Hierarchies in Paradise

**Cascading!**

- delete 1 edge at level 1
- separates 2 edges at level 2
- separates 4 edges at level 3
  ...

But don’t try this at home...
Idea 3: Cut to the Bone

Trouble: cuts that don’t look too sparse on level $i$
    but are very sparse viewed from level $i' \gg i$

Fix: consider sparsity of cuts that violate levels

Let $E_i = \text{edges on level } \geq i$

Before: $\phi_i = \min_s \frac{|E_i \cap (S \times S)|}{|E_i \cap (S \times S)|}$

Now: $\phi_i = \min_s \frac{|E_i \cap (S \times S)|}{|E_i \cap (S \times S)|}$

Thus, we never contract components on higher levels
Levels = reweighting of the graph
Nota Bene

Level promotions not **monotone**

This changes nothing

... but makes ever step of the reasoning a bit trickier

**Profile:**
e was in a sparse cut on lev 1, 2
not on level 3, 4, 5
but again on level 6
Updating the Analysis of Updates

At level $i$: \(\text{Vertices} = \text{components united by remaining edges on levels } \leq i\)

\[\text{Edges} = \begin{cases} \\ \\
\text{original edges on level } i \\
\text{deleted edges between components}
\end{cases}\]

expander!
Unfortunately...

• constructing the hierarchy takes $\text{poly}(m)$ time
  😞 need $O(m)$ construction for fully dynamic

• [Spielman-Teng STOC’05]
  construct the original hierarchy in $O(m)$
  local approximation to weighted sparsest cut?

• need better random walks for volume in weighted graphs?
Oracle Queries

**Level-i component**
- = comp induced by edges on levels ≤ i

**Hierarchy tree**
- = parent relations between components

Isolated component => break parent pointer

Query = find lowest broken pointer
  - Binary search on level
  - => $O(lglg m)$ time per query
The End

Anarchists question hierarchies