Dynamic Graph Connectivity in polylogarithmic worst case time

Bruce Kapron, Valerie King and Ben Mountjoy
University of Victoria, Victoria, Vancouver Island, BC
Graph with n nodes
Sequence of online updates and queries
Update: Insert \{A,D\}
Update: Delete edge \{E,F\}
QUERY($X,Y$): Is there a path between $X$ and $Y$?
How to avoid $O(m)$ cost of recomputing spanning forest with each update or running $O(m)$ search for each query?

$m=$number of edges
A Simple problem, but lots of interesting ideas....

Early 60’s-70’s: partially dynamic amortized:
- insertions only:
  - Union-find; Tarjan’s $\alpha(m,n)$ analysis
- 1981: edge deletions only Even $O(mn)$

Fully Dynamic (Update times)
- 1983: $O(\sqrt{m})$ worst case Fredrickson
- 1992,7: $O(\sqrt{n})$ Sparsification Eppstein, Galil, Italiano, Nissenzweig
POLYLOG Amortized time updates

Update time / Query time

- 1995 $O(\log^3 n)$ / $O(\log n/\log \log n)$. (expected time) Henzinger, King
- 1998 $O(\log^2 n)$ / $O(\log n/\log \log n)$ Holm, de Lichtenberg, Thorup
- 2000 $O(\log n (\log \log n)^3)$ / $O(\frac{\log n}{\log \log \log \log n})$ Thorup

All with $\Theta(n)$ worst case update time
SODA 2013:

$O(\log^5 n)$ worst case update time
$O(\log n / \log \log n)$ query time

1-sided error:

“Yes” always correct
“No” prob. $1/n^c$ error
All known techniques rely on maintaining a spanning forest
Dynamic Trees (ET-trees, H-K 1995)

Diagram of dynamic trees with nodes labeled A, D, C, E, and F, and a link connecting C to another node.
Dynamic Trees (ET-trees, H-K 1995)
Dynamic Trees (ET-trees, H-K 1995)

weights on nodes

A

D

E

C

25

14

11

20

F
Dynamic Trees (ET-trees, H-K 1995)

Query: Find tree containing node C
Query: Return sum of wts in tree
$O(\log n)$ per update & query
We maintain a spanning forest
When tree edge is deleted, how to find replacement edge?
Here, bitwise XOR method:

\[ V = \{1, 2, \ldots, n\} \]

Form the name of \( \{a, b\} \), \( a < b \):
- \( a \) (as a \( \lg n \) bit number) followed by
- \( b \) (as a \( \lg n \) bit number)

“\( \langle ab \rangle \)”

For each node \( a \), keep a vector of bits \( v(a) \),
\[ v(a) = \text{bitwise XOR of names } \langle ab \rangle \text{ of edges} \]

For any cut \((S, V \setminus S)\), if there is exactly one edge \( \{x, y\} \) in its cutset then
\[ \text{XOR}_{a \in S} \quad v(a) = \langle xy \rangle \]
Example:
v(a)
XOR of $v(a) = 001010$

in $S$

+ 011111

= 010101

= XOR of $v(a)$ in $V-S$
Dealing with larger cutsets

To insert:

- Add $<ab>$ to $v(a,i)$ and $v(b,i)$ with prob. $1/2^i$, for $i=0,2,\ldots,2\lg n$
- Keep record of additions for each $a$ and $i$.

To delete: Add again if it was added before
Dealing with larger cutsets

To insert:
• Add $\langle ab \rangle$ to $v(a,i)$ and $v(b,i)$ with prob. $1/2^i$, for $i=0, 2, \ldots, 2\lg n$
• Keep record of additions for each $a$ and $i$.

To delete: Add again if it was added before

Observe: $C$ cutset of $(S, V-S)$. For $i \sim \lg |C|$, 
$\Pr[\text{Adding an edge } \{a, b\} \text{ in } C \text{ to } v(a,i)] \sim = 1/|C|$ and 
$\Pr[\text{Exactly one edge in } C \text{ was added to some } v(a,i)] = \Pr[\text{bitwiseXOR}_{a \in S} v(a,i) = \text{name of edge in } C] = \text{a const.}$
Dealing with larger cutsets

To insert:
• Add \( <ab> \) to \( v(a,i) \) and \( v(b,i) \) with prob. \( 1/2^i \)
  for \( i=0,2,\ldots,2\log n \)
• Keep record of additions for each \( a \) and \( i \).

To delete: Add again if it was added before.

Observe:
  \( C \) cutset of \((S,V-S)\). For \( i \approx \log |C|\),
  \( \Pr[\text{bitwiseXOR}_{a \in S} v(a,i) = \text{edge in } C] = \) a const.

Repeat for \( \log n \) versions. Then for some version, the name of exactly one edge in \( C \) appears
with prob \( 1-1/n^c \).
Over a sequence of updates:

Union bound gives small error over polynomial length sequence, *provided the choice of updates are independent of the random bits*

**Record** enables incremental rebuilding and periodic correction of data structure to maintain prob. of error.
Solution to dynamic connectivity?? (not quite)

Problems:
A. Can’t let adversary know the spanning tree edges
B. Adversary sees answers to queries
   --Update sequence is independent of random bits while all queries correctly answered, as they are then determined by the graph itself.
C. Choice of cut searched depends on random bits!
XOR method solves easier problem:

“CUTSET” DataStructure (DS)

Maintain a forest $F$ of dynamic disjoint trees in graph $G$:

Updates: insert-edge, delete-edge, insert-tree-edge, delete-tree-edge.

Query ($S$) returns an edge in the cutset ($S, V\setminus S$)

Updates are independent of random bits.
Maintain spanning forest using Cutset DS\(_i\), \(i=0\ldots\lg n = \text{TOP}\)

Random bits from Cutset DS\(_i\) used to pick edges in \(F_{i+1}\) joining trees from \(F_i\)

“Tier \(i+1\) edge”

Query(T,k) returns a \(k+1\) edge if it exists
INVARIANTS:

- Structure of $F_i$ is independent of random bits from tiers $i$ and higher.

- Every tree on tier $i$ is matched (linked) to another tree on tier $i$ by a tier $i+1$ edge unless it’s maximal in $G$.
  $\rightarrow$ spanning forest by TOP tier.
Initially, all $F_i$ are singleton nodes
Insert edge: insert into all Cutset DS\(_i\)

If edge joins unconnected trees in F\(_{\text{top}}\)
insert edge as tree edge into all F\(_i\)
Delete edge: delete from all Cutset DS$_i$

Restore Invariants using Cutset DS$_i$
Example: $F_0$
Example: $F_1$
$F_2$
$F_3$
Deletion of a tier 1 edge:
Deletion: If unmatched tree $T$ in tier $i$, find new edge in Cut $(T, V \setminus T)$ and insert into all $F_{i'}$, $i' > i$
But new tree edge may cause an unmatched tree on a higher tier
Unmatched tree in $F_2$
Delete \((x,y)\)

Delete\((x, y)\)

remove \(\{x,y\}\) from all \(\text{CutSet}_i\) containing it.

\[
\text{for } u \text{ in } \{x,y\} \text{ do}
\]

\[
\text{while } u \text{ has an unmatched ancestor in the Boruvka tree do}
\]

\[
A \leftarrow \text{the lowest unmatched ancestor of } u
\]

\[
k \leftarrow \text{ (tier of } A)\]

\[
\text{Reconnect}(A, k)
\]
Reconnect(A, k)

\[ e = \{v, w\} \leftarrow \text{Query}(A, k) \] (assume that \( v \) is the endpoint of \( e \) in \( A \))

if \( e = \text{null} \) then mark \( A \) as maximal

else \{remove higher edge from \( F \) to break cycle\}

\textbf{if} there is a path from \( v \) to \( w \) in \( F_{\text{top}} \) then do
\n\[ e' \leftarrow \text{maximum tier edge on the path between } v \text{ and } w. \]
Remove \( e' \) from all \( F_i \) that contain it
Add \( e \) to \( F_{k'} \) for all \( k' > k \)
To implement:
“if there is a path from v to w in $F_{\text{top}}$ then do $e' \leftarrow \text{maximum tier edge on the path between v and w}.”

Use S-T dynamic trees:
Maintain $F_{\text{TOP}}$ with edges labeled by their tier number.
Find maximum weighted edge in path from v to w, $O(\log n)$ per operation.
Other Implementation details:

Use ET-Trees to maintain XOR sums:

- $O(\log^2 n)$ size vectors, $\Rightarrow O(\log^3 n)$ cost to change a tree edge
- 2 tree edges per tier inserted per deletion
- Each edge insertion affects forests in up to $\lg n$ tiers
- $\Rightarrow O((\log^3 n)(2 \log n)(\log n))$

--> $O(\log^5 n)$ overall cost per deletion
Space

Record of insertions requires $\tilde{O}(m)$. Omit by using hash function for randomness, but then can only be run for poly time.

See Graph Sketches paper, Ahn, Guha, McGregor, SODA 2012, which uses similar ideas to ours, but for a somewhat different problem.
Open Problems

Reduce update cost: lots of possibilities, or modify goal to reduced worst case expected cost.

Is there a Las Vegas or deterministic alg with polylog worst case time?

Is there a polylog worst case alg. for dynamic MST?
Come visit us in Victoria

Questions?
1995,98 ET trees used
Euler Tour Tree
(from Erik Demaine.’s class notes)
Euler Tour Tree
Euler Tour Tree: augmented balanced search tree

- findroot, cut, link, sum of node weights in tree
Lower Bounds for Dynamic Connectivity

\(\Omega(\log n)\) time per operation (Patrascu, Demaine 2004) in the Cell probe model = \#memory accesses (where each word contains \(\log n\) bits)

Also lower bounds on tradeoffs between query time and update time, e.g.:
query time \(\times \log(\text{update time}/\text{query time}) = \Omega(\log n)\)
I would like to take a moment to remember Mihai Patrascu a very talented young colleague in this area whom I will miss.

July 17, 1982 -
June 5, 2012
Lower Bound for Connectivity

Random distribution of BATCH updates and queries:

Prob. 1\/2: replace a randomly chosen $\Pi_k$ by a random $\Pi$
Lower Bound for Connectivity

Random distribution of BATCH updates and queries:

\[
\text{Prob. } 1/2 \text{ do update } (k) : \text{ replace a randomly chosen } \pi_k \text{ by a random } \pi
\]
Lower Bound for Connectivity

Random distribution of BATCH updates and queries:

Prob. 1/2: update (k): replace a randomly chosen \( \pi_k \) by a random \( \pi \)

Prob. 1/2: query (k): \( \forall \) rows \( i \), random column \( k \), test \( \pi_k(...) (\pi_2(\pi_1(i))) \)
Sequence of batch operations
Split into two time intervals

\[ U_1 < U_2 < ... < U_{k-1} \]

\[ Q_1 < Q_2 < ... < Q_k \]

Note: High expected number \( L \) of interleaves:
\[ U_1 < Q_1, < U_2 < U_3 < Q_2 < ... < U_{k-1} \]

To answer \( Q_2 \) need to know \( U_2, U_3 \)
---Need to know a different \( U \) for each interleaving
Lower Bound for Connectivity

Sequence of batch operations

Split into two time intervals

$U_1 < U_2 < ... < U_{k-1}$

$Q_1 < Q_2 < ... < Q_k$

Number of READS of these WRITES must be sufficient to provide enough bits to encode $L U'$s.

Paper shows method for concise encoding of info from READS from which $U'$s can be reconstructed.
Lower Bound for Connectivity

Sum up expected costs over intervals given by binary tree,

Parent interval = union of children intervals.

Adds log n factor

Note: Each read is counted once, by the lowest common Ancestor of the read and most recent preceding write time.