Dynamic Graphs: Dynamic edge orientation

Eva Rotenberg

Dynamic.

Algorithms:

Algorithmic problem $\rightarrow$ Algorithm $\rightarrow$ Solution.

Dynamic algorithms:

Update to problem $\rightarrow$ Dynamic algorithm $\rightarrow$ Update to solution.

- Add/delete element in datastructure
- Add/delete edge in graph, $\leftarrow$ Note, $O(\log n)$ bits.
- Add/delete/change character in string, or point in curve, ...

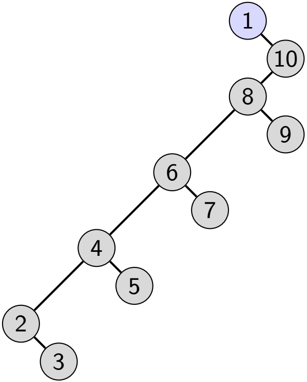
Motivation:

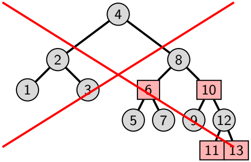
- Useful:
  - Efficiently maintain information in large, changing datasets,
  - Applications in (static) algorithms, sabotage logic, other models ...
- Revisit fundamental problems and properties
  - graph connectivity, planarity, distance, min-cut, colouring, clustering, ...

Toolbox:

- Maintain/update some data structure,
- Amortised algorithms

Last time: Splay trees





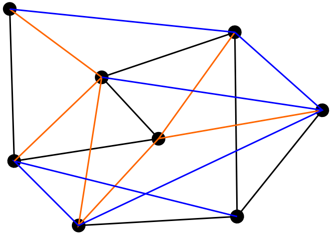
Insert 1, 2, 3, 4, . . ., 10. Splay 1.
Be lazy
Balance things out when needed.

not eager.

Analysis: Potential function.
Benefit: amortisation allows us to approach the problem from a new angle, apply different ideas.

Dynamic bounded out-degree orientations

Sparse graphs



A graph is sparse if it has few edges per vertex. A measure of sparsity is arboricity, the treeishness of the graph.

- Arboricity of G is $\leq c$,

  $\Updownarrow$
- Union of c forests: $G = F_1 \cup F_2 \cup \ldots \cup F_c$

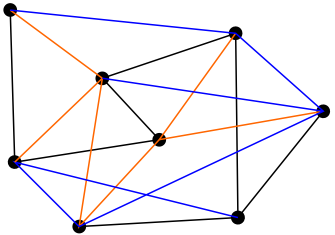
  $\Updownarrow$
- Any subgraph J on n_J vertices has $\leq c(n_J - 1)$ edges.

Note: Given $G = F_1 \cup F_2 \cup \ldots \cup F_c$ possible to orient F_i towards root. Thus, outdegree$(v) \leq c$.

Problem: Dynamic graph whose arboricity never exceeds c. Orient edges, so that outdegree(v) is $O(c)$.

Motivation: adjacency(u, v) in $O(c)$ time. (I.e: are u and v neighbours?)

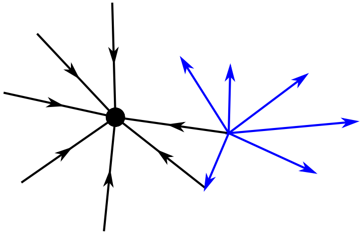
Dynamic bounded out-degree orientations



Setup:
Arboricity $\leq c$
$G = F_1 \cup F_2 \cup \ldots \cup F_c$

The problem:
Dynamic graph, arboricity always $\leq c$.
out-degree(v) $\leq \Delta = 6 \cdot c$?

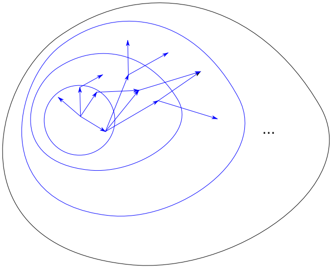
The algorithm (amortised)
Deletion? Easy.
Insertion, safe case? Easy.
Insertion, overflow? Flip all edges.
This may cause some neighbours to overflow
(ie. more than Δ out-neighbours.)
recursively repeat on the overflowing vertices,
until the stack of overflowing vertices is empty.



Correctness? If terminates, then correct Δ out-orientation
Running time? Amortised analysis.

Dynamic bounded out-degree orientations – the omniscient algorithm

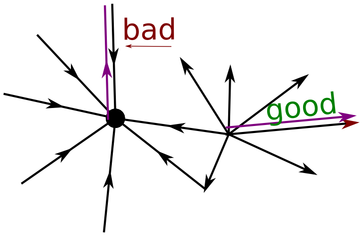
Algorithm: 6α-overflow $\Rightarrow$ flip-in all edges, keep flipping until no overflow.
Analysis: Consider maintaining a 2α-orientation.
If out-degree$(u) \geq 2\alpha$, there is a path of length $\log n$ to some v of out-degree $< 2\alpha$.
Why? (1) such a vertex must exist (arb. $\leq \alpha$).
(2) Consider V_i; i'th out-neighbourhood of u.
If one of them contains such a v – done.
Otherwise, $|V_i > 2^i|$. Why? Induction.
Assume $V_j > 2^j$, and we are not done.
Consider the 2α out-edges of V_j: $2\alpha \cdot 2^j$ edges.
Arboricity α: those $2\alpha \cdot 2^j$ edges must take up $2\alpha \cdot 2^j/\alpha$ vertices. I.e. 2^{j+1}. $\leftarrow V_{j+1}$. :)
So after $\log n$ steps, $V_{\log n} = G$.



Conclusion: There is an omniscient 2α-orientation algorithm that performs only $\log n$ flips per dynamic operation.

Dynamic bounded out-degree orientations – putting it together

Algorithm: When overflow $\Rightarrow$ flip-in all edges,
keep flipping until no overflow.
Overflow: $> 6c$ out-edges on a vertex.
Recall: omniscient $2c$-algorithm, $\log n$ flips.
Say an edge is *good* if it agrees with the omniscient algorithm.
What happens when 'overflow'$\Rightarrow$ flip-all?
At least $4c$ bad edges become good.
At most $2c$ good edges become bad.
Potential = number of bad edges
Greedy algorithm does amortized $O(\log n)$ flips per edge update.



Take-home message:
Amortised analysis is a potent tool for analysing very simple algorithmic ideas.
Recourse analysis can be an important tool for amortised analysis of greedy algorithms.
(Exercises.)

Lower bound

Assume c is a constant, and is an upper bound on the arboricity of the dynamic graph.
Then if we force out-degree $\leq c$, we may have to perform $\Omega(n)$ edge-reorientations per insert/delete.
Idea: Think of a path. If we cut and link, we may force $\Omega(n)$ reorientations.
For larger c, the construction is a union of paths. Still, a cut and link in one of these paths is what shows the lower bound.
(For details, see Thm. 4.)