Dynamic bounded out-degree orientations.

- Dynamic bounded out-degree orientations
  - Problem definition
  - Motivation
- Algorithm. Analysis:
  - Augmenting path lemma
  - All-knowing algorithm
  - Potential argument
- Lower bound

Dynamic d-orientations

A sparse graph is one where there are few edges per vertex.
A way to measure sparseness is arboricity, the treeishness of the graph, which is both the fewest forests it takes to cover its edges, and also a bound on the number of edges per vertex in any induced subgraph:

- Arboricity $\leq c \Leftrightarrow$
- The edge set is the union of $\leq c$ forests $\Leftrightarrow$
- Any subgraph J on n_J vertices has at most $c(n_J - 1)$ edges

Problem: Given a dynamic graph whose arboricity never exceeds c, orient the edges so that each vertex has $O(c)$ out-edges.
Motivation: If each edge stores its out-neighbours, we can then check adjacency in $O(c)$ time.

Algorithm

Algorithm for Δ-orientation.
Delete(e) – just delete it.
Insert((u, v)) – Two cases.
Safe: if out-degree of u less than Δ – just insert it.
Overflow: otherwise, take all out-edges of u, and flip their direction.
This may cause some u's out-neighbours to overflow, so

- recursively repeat on the overflowing vertices,
- until the stack of overflowing vertices is empty.

Correctness? If the algorithm terminates, correct Δ-orientation.
Today: Amortised analysis.

Augmenting path lemma.

Let G have arboricity c. Meaning: everytime you have a subset J, there are at most ca. c edges per vertex in J.

$$\frac{|E(J)|}{|V(J)|-1} \leq c \quad \Leftrightarrow \quad |E(J)| \leq c \cdot (|V(J)|-1) \quad \Leftrightarrow \quad \frac{|E(J)|}{c} < |V(J)|$$

Note: $E(J)$ are the edges of G where both endpoints are in J.
Lemma: Let $\delta > c$. Given u with δ out-edges. Then there exists a path from u of length $\log_{\delta/c}(n)$ to some v with less than δ out-edges.
Idea: u has δ out-neighbours. If one of them has small out-degree, we are done. Otherwise, they each have δ out-neighbours. Keep going.
Formally: Let V_i be the vertices reachable by a path of length i from u. By induction, $|V_i| > (\delta/c)^i$. Start: $|V_1| = \delta > 1$.
Step: Assume $|V_i| > (\delta/c)^i$. Consider V_i and its neighbours. How many edges? $\delta|V_i|$ out-edges. So at least $\delta(\delta/c)^i$ edges. How many vertices? Arboricity c: $E(J)/c < V(J)$. That makes at least $\delta \cdot (\delta/c)^i/c$ vertices. $V_{i+1} \geq (\delta/c)^{i+1}$.
Now: When is $V_j \geq n$? When j is $\log_{\delta/c} n$. So by $\log_{\delta/c} n$ there must be a vertex of out-degree less than δ.

The omniscient algorithm

For the sake of analysis, consider the case where the whole sequence of updates is known in advance.
Note that the role of deletions and insertions can be swapped by looking forward or backward in time.
Consider maintaining a δ-orientation. $(\delta > c)$.
Every time a vertex u has out-degree $\delta + 1$, we just saw some vertex v exists, of out-degree $< \delta$, such that we may augment the path $u \longrightarrow v$ of length at most $\log_{\delta/c} n$. Augmenting this path causes $\rceil \log_{\delta/c} n \lceil$ edges to change direction.

So: Deletion 0 changes, insertion $\lceil \log_{\delta/c} n \rceil$ changes.
Or, reversing the time-line, insertion 0 changes, deletion $\lceil \log_{\delta/c} n \rceil$ changes.

Putting it all together

The algorithm: Do nothing until some vertex u's outdegree exceeds Δ, and then revert edge directions, possibly causing other vertices to overflow, and keep handling overflow-vertices until the stack is empty.
Now, let's set $\Delta = 3\delta$, and $\delta = 2c$.
For the sake of analysis, consider the orientations made by the omniscient algorithm.

- Say an edge is *good* if its orientation is consistent with the omniscient orientation, and *bad* otherwise.
- For amortised analysis, we will use a potential that is simply counting the bad edges.

If we can show that handeling an overflowing vertex is payed for by the potential, then our amortised analysis works.
When a vertex overflows, it has out-degree $\Delta + 1 = 3\delta + 1$. At most δ of those are good, so there are $2\delta + 1$ bad edges.
So, by flipping the orientation of all these edges, our potential will decrease by at least δ. Thus paying for all the $O(\delta)$ incurring edge-reorientations.
Thus, amortised times: $O(1)$ for insertion and $O(c + \log n)$ for deletion.
Adjacent in $O(c)$ worst-case time.

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.
(More on blackboard, see also Thm. 4.)