Dynamic Bridge-Finding in $\tilde{O}(\log^2 n)$ Amortized Time

Jacob Holm[1] Eva Rotenberg[2,3] Mikkel Thorup[1]

[1]BARC at [2]Department of Computer Science
University of Copenhagen

[3]Department of Applied Mathematics and Computer Science
Technical University of Denmark

Definition

A *bridge* in an undirected (multi-)graph is an edge whose removal increases the number of connected components. A 2-*edge connected component* is a maximal connected subgraph that has no bridge. A graph is 2-edge connected if it consists of a single 2-edge connected component.

Theorem

There exists a deterministic data structure for dynamic multigraphs in the word RAM model with $\Omega(\log n)$ word size, that uses $\mathcal{O}(m+n)$ space, and can handle the following updates, and queries for arbitrary vertices v or arbitrary connected vertices v, u:

- *insert and delete edges in $\mathcal{O}((\log n)^2(\log\log n)^2)$ amortized time,*
- *find a bridge in v's connected component or determine that none exists, or find a bridge that separates u from v or determine that none exists. Both in $\mathcal{O}(\log n/\log\log n)$ worst-case time.*
- *find the size of v's connected component in $\mathcal{O}(\log n/\log\log n)$ worst-case time, or the size of its 2-edge connected component in $\mathcal{O}(\log n(\log\log n)^2)$ worst-case time.*

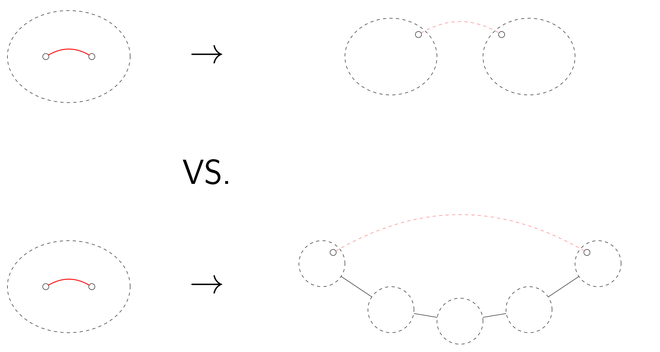

Figure 2 : Deleting an edge from a connected graph splits it into at most 2 connected components. Deleting an edge from a 2-edge connected graph turns it into a string of $\mathcal{O}(n)$ 2-edge connected components, connected by bridges.

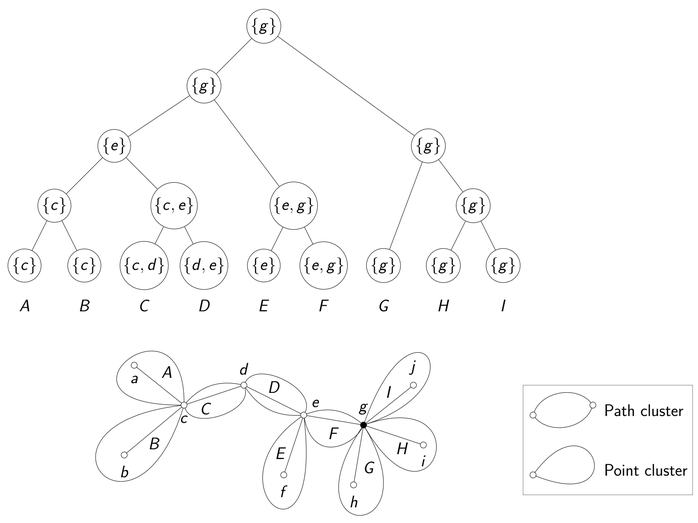

Figure 3 : A top tree and its underlying tree. A *cluster* is a connected subtree with at most two *boundary vertices*. Each node in the top tree corresponds to a cluster with the listed boundary vertices. Vertex g is chosen as *external boundary vertex*.

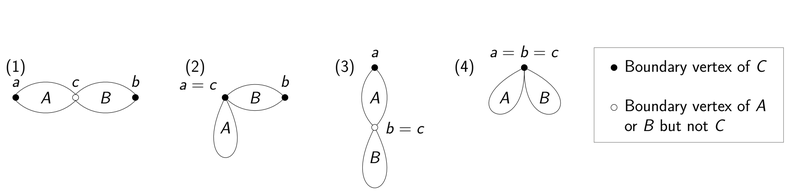

Figure 4 : The 4 different ways a cluster C can be *merged* from, or *split* into two smaller clusters A and B.

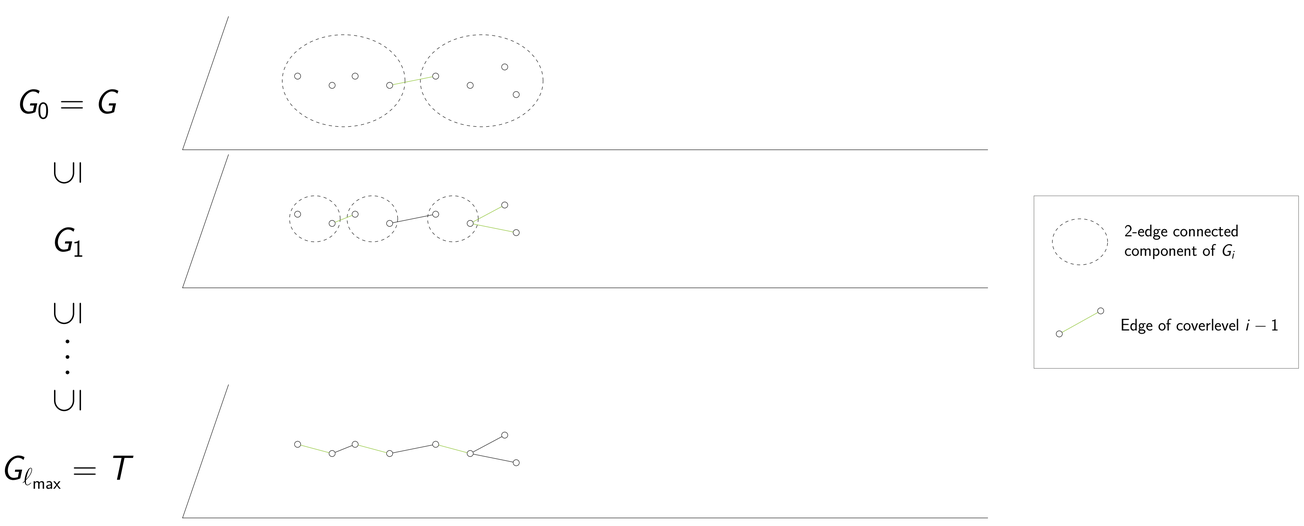

Figure 1 : Let $\mathcal{K}_2(G)$ be the set of all the 2-edge connected components of G. The algorithm considers a nested family of subgraphs G_i with $\max_{C\in\mathcal{K}_2(G_i)}|V(C)| \le \lfloor \frac{1}{2^i}|V(G)|\rfloor$. For every edge e, $\ell(e) = \max\{i \in [\ell_{\max}] \mid e \in G_i\}$ is its *level*, and for $e \in T$ let $c(e) = \min\{i \in [\ell_{\max}]-1 \mid e \text{ is a bridge in } G_{i+1}\}$ be its *cover level*.

Main Ideas

- Reduce to maintaining a spanning tree with *cover levels*. A bridge is an edge whose cover level is -1 (Figure 1).
- Whenever an edge is deleted, pay for the cost of updating the cover levels by increasing edge levels. This is similar to the best known amortized connectivity algorithm, but harder (Figure 2).
- Use a single *top tree* (Figure 3) for each connected component in G to keep track of the cover levels and 2-edge connected components in all G_i. Then, most computations happen during *merge* and *split* (Figure 4).
- Whenever we inspect an edge, we want to increase its level to pay for the inspection, unless this would violate our size constraints. To determine this, we need to be able to compute the sizes of certain subtrees. (Figure 5 and 6).
- The previous best algorithm maintained a matrix of $\Theta(\log^2 n)$ sizes for each path cluster. New version keeps a binary tree of prefix sums of vectors instead, saving an $\Omega(\frac{\log n}{\log\log n})$ factor in time per merge/split (Figure 7).
- For linear size, use *fat-bottomed* top trees. Essentially recompute instead of store when the tree is small enough.

This gives a purely combinatorial data structure using linear space and $O((\log n)^3 \log\log n)$ amortized update time. To get the rest of the way we need the word RAM with $\Omega(\log n)$ bit words.

- For faster queries, maintain an additional $\Theta(\frac{\log n}{\log\log n})$-ary top tree. This requires some word RAM tricks.
- Instead of maintaining exact sizes, maintain an $(1+\mathcal{O}(\frac{1}{\log n}))$-approximation of each size. This can be represented as an $\mathcal{O}(\log\log n)$ bit floating point value. Vectors of $\mathcal{O}(\log n)$ of these values can be packed into $\mathcal{O}(\log\log n)$ words of size $\Omega(\log n)$, and addition of such vectors takes only $\mathcal{O}(\log\log n)$ time, saving another factor of $\mathcal{O}(\frac{\log n}{\log\log n})$.

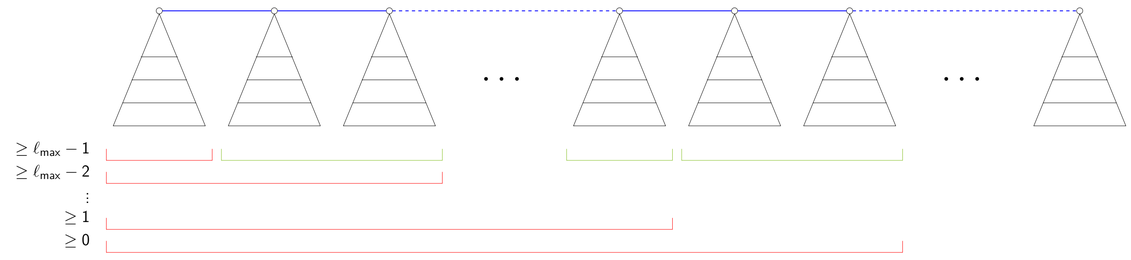

Figure 5 : The trees along the *cluster path* connecting the two boundary vertices of a path cluster. The i'th red group contains the trees where all edges up to that point are covered at level $\ge \ell_{\max} - i$. The green groups are the differences between the reds.

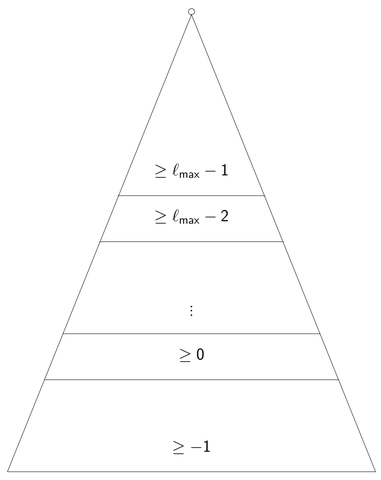

Figure 6 : Each subtree hanging off the path has for each i a set of vertices such that the path from the root is covered at level $\ge \ell_{\max} - i$.

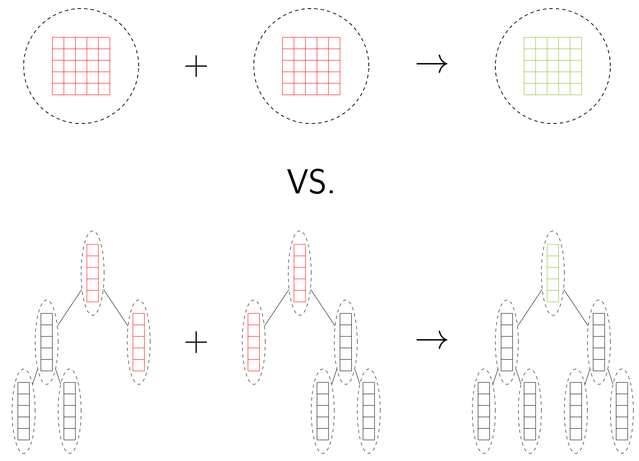

Figure 7 : In each merge, the old algorithm had to recompute the whole matrix. The new one inherits most of the vectors from the previous nodes.

References

[1] Jacob Holm, Eva Rotenberg, and Mikkel Thorup.
Dynamic bridge-finding in $\tilde{O}(\log^2 n)$ amortized time.
In *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 35–52, 2018.