Algorithms for graph visualization

Data Structures for Planar Graph Embeddings

Patrizio Angelini
A drawing of a graph
Two drawings of a graph
An embedding of a graph

- Different geometry in the two drawings, but the ordering of the edges around each vertex is the same.
Two embeddings of a graph

- Different “topology” in the two drawings
Properties of embeddings

- Fàry’s Theorem (1946):
  If a graph admits a planar drawing where edges are curves, than it also admits a straight-line planar drawing

- Planarity is a “topological” problem!
  - In order to say that a graph is planar, we need to test whether it admits a planar embedding
  - Forget about drawings
  - Well, it depends on the problem...
How many drawings/embeddings?

- Infinite number of drawings (Continuous space)

![Graphs representing infinite number of drawings](image1)

- Finite number of embeddings (Discrete space)
  - Planar embeddings are equivalence classes of planar drawings
  - So, how many planar embeddings?
Connectivity

- A graph is **connected** if for every pair of vertices there exists a path connecting them.
- A graph is **k-connected** if for every pair of vertices there exist *k* disjoint paths connecting them.
Connectivity

- $k = 1$: (simply) connected graph

- cut-vertex

- 2-connected component
Connectivity

- $k = 2$: biconnected graph

Separation pair

3-connected component
Connectivity

- \( k = 3 \): triconnected graph

Separating triplet (triangle)

4-connected component
Question time

How connected can a planar graph be?

More formally, what is the largest value of $k$ such that there exists a planar graph that is $k$-connected?

5

at most $3n-6$ edges
Why did we stop at $k = 3$?

Even more, why are we speaking about connectivity?

We’ll answer both questions in a while.
Connectivity – Embeddings

How connected is this graph?
Connectivity – Embeddings

How connected is this graph? 3
How connected is this graph? 3
How many planar embeddings?
Connectivity – Embeddings

How connected is this graph? 3
How many planar embeddings? 2
Connectivity – Embeddings

- **Theorem (Whitney, 1932):**

A 3-connected planar graph admits only two planar embeddings, which differ by a flip.
Connectivity – Embeddings

How connected is this graph?
How connected is this graph? 2
Connectivity – Embeddings

How connected is this graph? 2
How many planar embeddings?
Connectivity – Embeddings

How connected is this graph? 2
How many planar embeddings?
Connectivity – Embeddings

How connected is this graph? 2
How many planar embedding?
Connectivity – Embeddings

- Permutations of parallel subgraphs
- Flips of triconnected subgraphs
Connectivity – Embeddings

So, how many embeddings?

- Permutations of $k$ parallel subgraphs
  - $k!$
- Flips of $k$ triconnected subgraphs
  - $2^k$
Connectivity – Embeddings

How connected is this graph?
Connectivity – Embeddings

How connected is this graph? 1
Connectivity – Embeddings

How connected is this graph? 1
How many planar embeddings?
Connectivity – Embeddings
Connectivity – Embeddings

- All possible nesting configurations
- Combined with all possible embeddings of the biconnected components
Connectivity – Embeddings

So, how many embeddings?

- All possible nesting configurations
- Combined with all possible embeddings of the biconnected components

Quite a lot!
Connected graphs: data structure

- Block Cut-vertex tree (BC-tree)
  - A B-node for each block
  - A C-node for each cut-vertex
Biconnected: data structure
There exist many separation pairs
Biconnected: data structure

- There exist many separation pairs
Biconnected: data structure

- There exist many separation pairs
Biconnected: data structure

- There exist many separation pairs
We need a step-by-step decomposition.

At each step, we look at the graph from the point of view of a particular separation pair.
SPQR-tree decomposition

- A split pair \( \{u,v\} \) is a pair of vertices such that:
  - \( \{u,v\} \) is a separation pair, or
  - \((u,v)\) is an edge

- An SPQR-tree is a rooted tree whose nodes are of 4 types:
  - Series-nodes
  - Parallel-nodes
  - Q-nodes
  - Rigid-nodes
We first select any edge as the root.

At each step we consider a split pair \( \{u,v\} \) and add a node whose type depends on how the graph looks like from the point of view of \( \{u,v\} \).

Each node of the SPQR-tree is associated with a multigraph, called skeleton, describing how the children of the node are arranged.

- Each child corresponds to an edge of the skeleton, called virtual edge.
Q-node

- If the graph between the split pair \{u,v\} is an edge, we add a Q-node.
- The skeleton of the Q-node is just an edge.
If the graph between the split pair \( \{u,v\} \) is a chain (series) of \( k \) components separated by cut-vertices, we add an S-node with \( k \) children.
If the graph between the split pair \(\{u,v\}\) is a chain (series) of \(k\) components separated by cut-vertices, we add an S-node with \(k\) children.

The skeleton of the S-node is a path between \(u\) and \(v\) whose internal vertices are the cut-vertices.
If the graph between the split pair \{u,v\} is a chain (series) of k components separated by cut-vertices, we add an S-node with k children.

We add a (virtual) edge between u and v that represents the “rest of the graph”
If the graph between the split pair \( \{u,v\} \) is a composition of \( k \) parallel components, we add a P-node with \( k \) children.
If the graph between the split pair \{u,v\} is a composition of k parallel components, we add a P-node with k children.

The skeleton is composed of k+1 edges between u and v (one is for the rest of the graph).
In all the other cases, we add an R-node whose skeleton (including the edge between \( u \) and \( v \)) is a triconnected graph, and add a child for each edge of the skeleton (except for one).
SPQR-tree: an example
SPQR-tree: an example

Reference (root) edge
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
SPQR-tree: an example
To describe an embedding:

- Flip of R-nodes skeletons
- Ordering of multi-edges of P-nodes skeletons

An SPQR-tree rooted at a reference edge $e$ represents all the embeddings of the graph in which $e$ is on the outer face.

If you choose a different reference edge, the resulting SPQR-tree is the same (up to a re-rooting)
SPQR-trees


Applications:

- (Dynamic) Planarity testing
- Navigating the graph (recursively) to compute embeddings, drawings, colorings, ...
- Computing an embedding that has some property or that is optimal with respect to some measure
An application

Input:
- A biconnected planar graph $G$;
- A simple cycle $C$ of $G$;
- A partition of the vertices of $G/C$ in two sets $V_1$ and $V_2$

Output:
- An embedding of $G$ such that the vertices of $V_1$ and those of $V_2$ are separated by $C$
  * vertices of $V_1$ are inside $C$ and those of $V_2$ are outside, or vice versa
An application
An application

- Given a split pair and a component with respect to it, there exist 3 possibilities:
  1. No vertex of C belongs to the component
An application

- Given a split pair and a component with respect to it, there exist 3 possibilities:
  1. No vertex of C belongs to the component
     - All the vertices of the component must belong to the same set
Given a split pair and a component with respect to it, there exist 3 possibilities:

1. No vertex of C belongs to the component
   - All the vertices of the component must belong to the same set
   - The node is *1-colored*
An application

2. All the vertices of C belong to the component
   • The node contains C
2. All the vertices of $C$ belong to the component
   
   - The node *contains* $C$
   - Vertices must be “correctly placed” inside/outside $C$
An application

3. Some (but not all) of the vertices of C belong to the component
   - The node is *traversed*
3. Some (but not all) of the vertices of C belong to the component

- The node is *traversed*
- Vertices of the component that are separated by the path of C between u and v must belong to different sets
  - The node is *well-separated*
Algorithm

- Compute the SPQR-tree T of G rooted at any reference edge
- Perform a bottom-up visit of T
  - At each step, consider a node of T and test whether there exists an embedding of the skeleton of the node that satisfies the properties with respect to the cycle
  - The test is based on the fact that all the children of the node in T have already been tested (and embedded)
  - Depending on the type of the node, the test and the embedding algorithm is different
If one of the children of the node is traversed by the cycle, then all the children are traversed (and the node itself is traversed)

- Since all the children are well-separated by induction, the S-node can be made well-separated by flipping the children in such a way that elements of the same set are on the same side
Algorithm: S-node

- If none of the children is traversed, then all of them (possibly except one) are 1-colored
  - Just check that the color is the same!
  - If one of the children contains C, then check if the color outside C is the same as the color of the others
Algorithm: R-node

- If none of the children is traversed, then all of them (possibly except one) are 1-colored
  - Just check that the color is the same!
  - If one of the children contains C, then check if the color outside C is the same as the color of the others

![Diagram of R-node with nodes u and v]
Algorithm: R-node

If one of the children is traversed, then some of the others are traversed. Two cases:

- If the node contains the whole cycle
  - All the not-traversed children are 1-colored
    - just check whether the color is the correct one
  - All the traversed children are well-separated
    - choose the correct flip
Algorithm: R-node

- If the node contains part of the cycle (the node is traversed)
  - All the not-traversed children are 1-colored
    - just check whether the color is the correct one to make the node well-separated
  - All the traversed children are well-separated
    - choose the correct flip

![Diagram of a graph with nodes U and V connected by edges, illustrating the R-node algorithm.](diagram.png)
Algorithm: P-node

- If none of the children is traversed, then all of them (possibly except one) are 1-colored
  - Just check that the color is the same!
  - If one of the children contains C, then check if the color outside C is the same as the color of the others
Algorithm: P-node

- At most 2 children are traversed
- If they are 2, the node contains the cycle
  - Order (permute) the children so that the 1-colored children are correctly placed inside/outside C
    - To choose the inside/outside, look at any vertex in the rest of the graph
  - Flip the 2 traversed children correctly
Algoritmo: P-node

- If there is 1 traversed child, the node is traversed
  - Order (permute) the children so that the 1-colored children are on different sides of the traversed child
    - Any left/right subdivision is good, we can flip the whole component later, if needed
  - Flip the traversed child correctly
Algorithm

- If the conditions are satisfied for every node, and in particular for the unique child of the root (that is considered at the last step of the bottom-up visit), the test is positive.
References