Planarity Testing

Sander Schuckman

October 3, 2007

Outline

1. *st*-Numbering
2. Bush Form and *PQ*-Tree
3. Vertex Addition Algorithm
4. Finding Planar Embedding

Planar Graphs

- Graph is represented by a set of n lists; called adjacency lists
- The adjacency list of a vertex contains all its neighbours
- An embedding of a graph determines the order of the neighbours embedded around a vertex
- A graph is planar if and only if all the biconnected components are planar
- Assume $m \leq 3n$; otherwise the graph is nonplanar

Definition of *st*-Numbering

- An *st*-numbering is numbering $1, \ldots, n$ of the vertices of a graph such that
  - Vertices "1" and "n" are adjacent
  - Every other vertex j is adjacent to two vertices i and k such that $i \leq j \leq k$
- Vertex "1" is the source s and vertex "n" the sink t

Depth-First Search

- Start with arbitrary edge (t, s)
- Compute for each vertex its depth-first number, its parent and its lowpoint

Definition (Lowpoint)

$$\mathrm{LOW}(v) = \min(\{v\} \cup \{w \mid \text{there exists a backedge } (u, w) \text{ such that } u \text{ is descendant of } v \text{ and } w \text{ is an ancestor of } v \text{ in a DFS tree}\})$$

Partition edges into paths

Vertices s, t and edge (s, t) are marked "old"

Case 1 There is a "new" back edge (v, w)

- Mark (v, w) "old"
- Return vw

Case 2 There is a "new" tree edge (v, w)

- Let $ww_1 w_2 \ldots w_k$ be the path to the lowpoint w_k of v
- Mark vertices and edges on the path "old"
- Return $ww_1 w_2 \ldots w_k$

Case 3 There is a "new" back edge (w, v)

- Let $ww_1 w_2 \ldots w_k$ be the path going backward to an old vertex w_k
- Mark vertices and edges on the path "old"
- Return $ww_1 w_2 \ldots w_k$

Case 4 All edges incident to v are "old"

- Return $\emptyset$

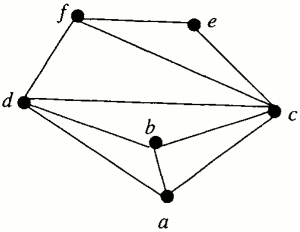
st-Numbering Algorithm

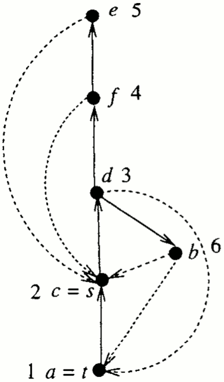
Invariant

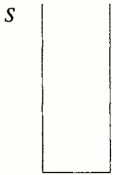
Vertices are pushed into a stack such that for every vertex v one neighbour is stored above v and one neighbour is stored below v; vertex above v will be assigned a lower number and vertex below v a higer number

1. Push vertices t and s onto stack S (s is above t)
2. Pop the top entry v from the stack
3. If PATH$(v) = \emptyset$ then number v
4. Otherwise let PATH$(v) = vu_1 \dots u_k w$; push vertices $v_k, \dots, v_1, v$ onto S (v is top of S)
5. Goto 2

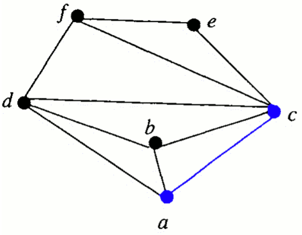
Example of an *st*-Numbering

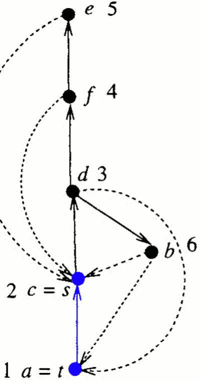


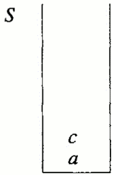




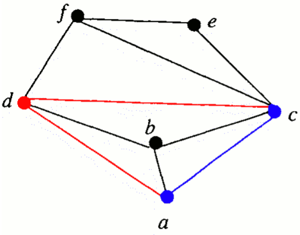
Example of an *st*-Numbering

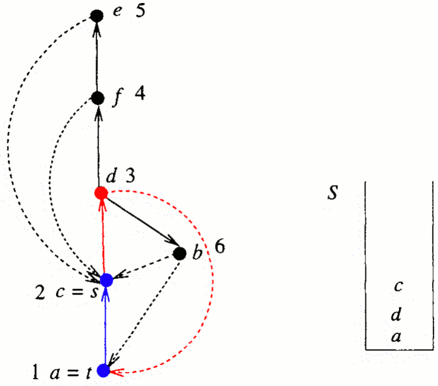




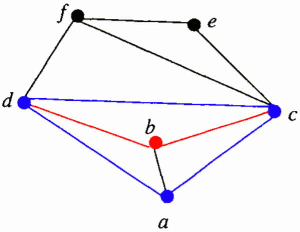


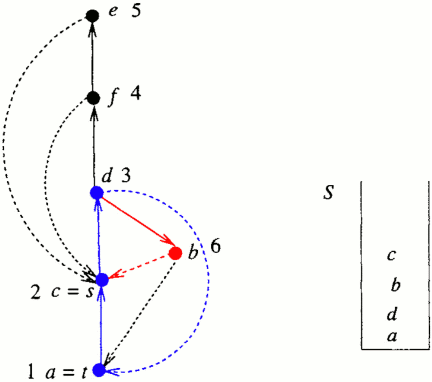
Example of an *st*-Numbering



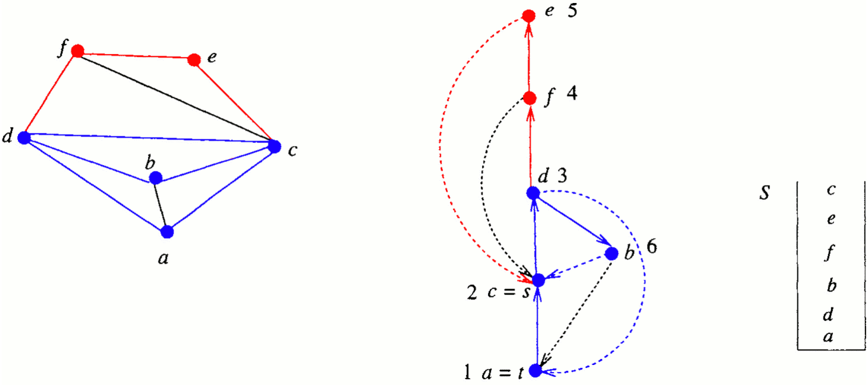


Example of an *st*-Numbering

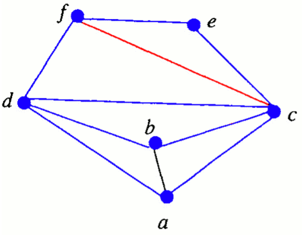


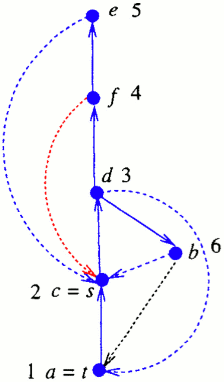


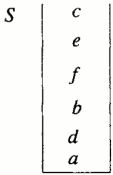
Example of an *st*-Numbering



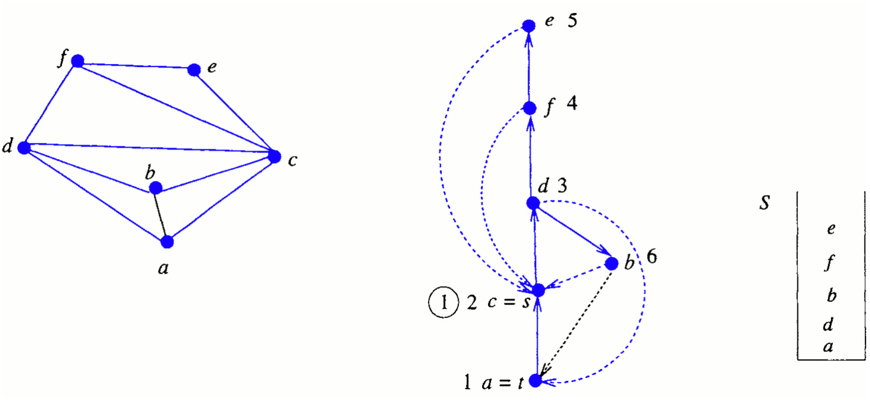
Example of an *st*-Numbering



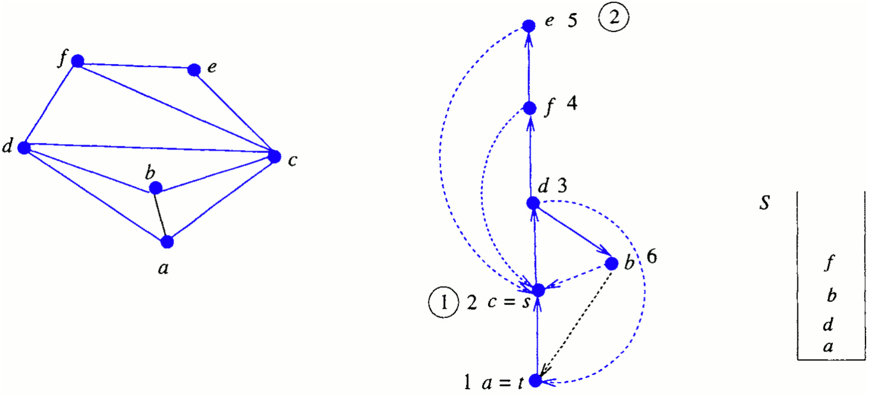




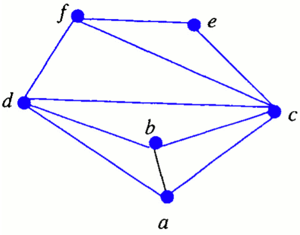
Example of an *st*-Numbering

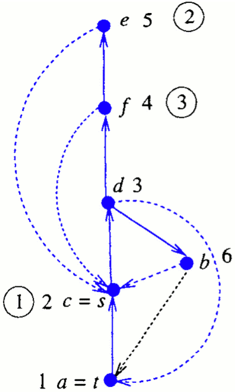


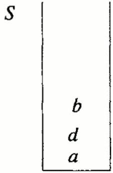
Example of an *st*-Numbering



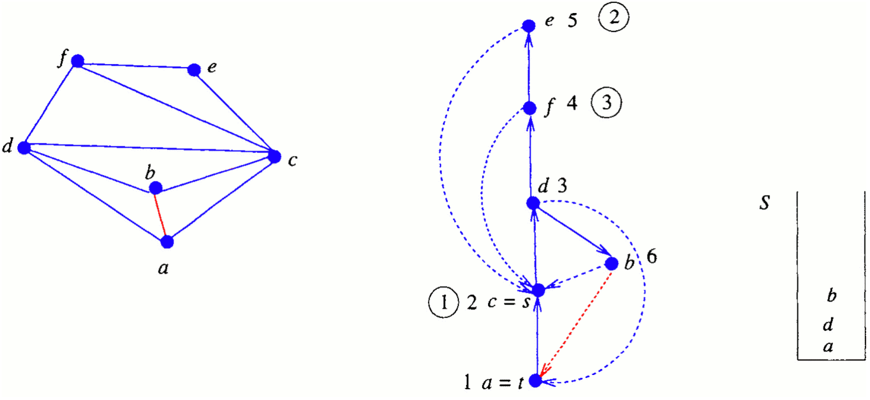
Example of an *st*-Numbering



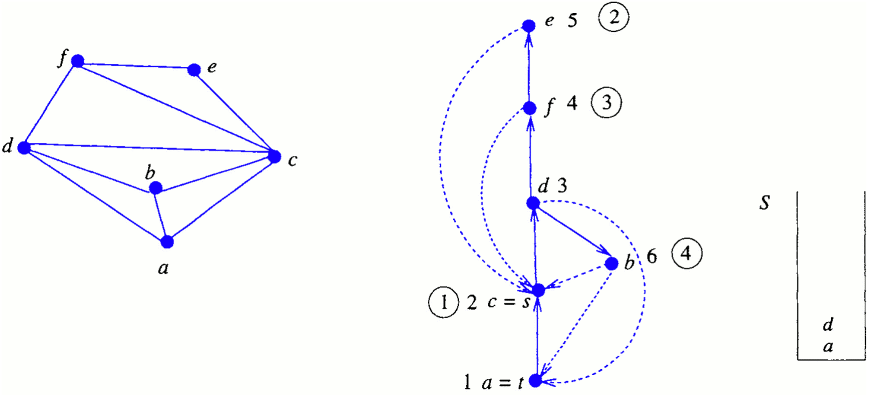




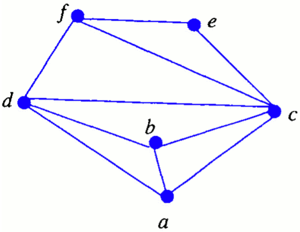
Example of an *st*-Numbering

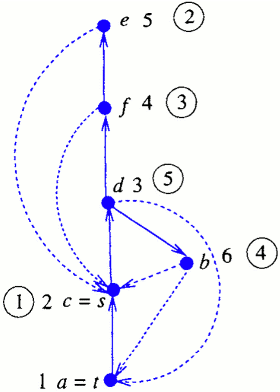


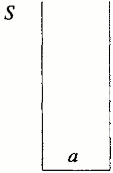
Example of an *st*-Numbering



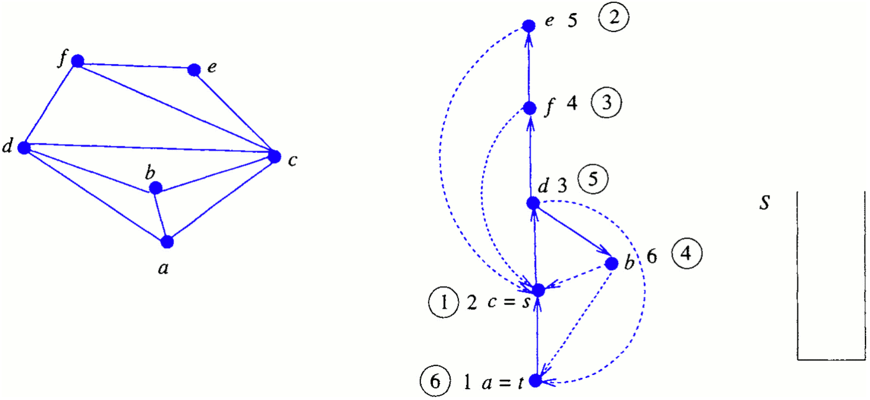
Example of an *st*-Numbering







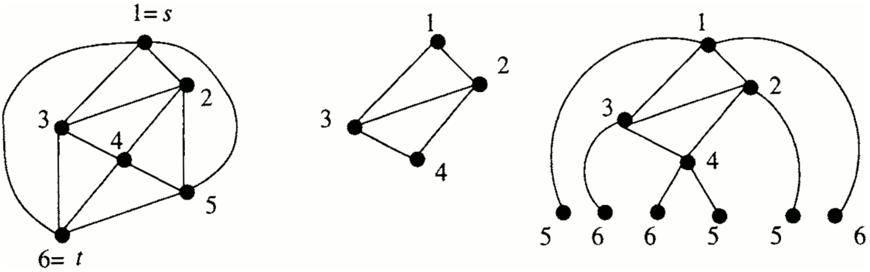
Example of an *st*-Numbering



Bush Form

- Let $G_k = (V_k, E_k)$ be the subgraph induced by the vertices $V_k = \{1, \ldots, k\}$
- Let G'_k be the graph formed by adding all edges with ends in $V - V_k$, where the ends of the edges are kept seperate
- These edges are called virtual edges and their ends virtual vertices
- A bush form of G'_k is an embedding of G'_k such that the virtual vertices are on the outer face

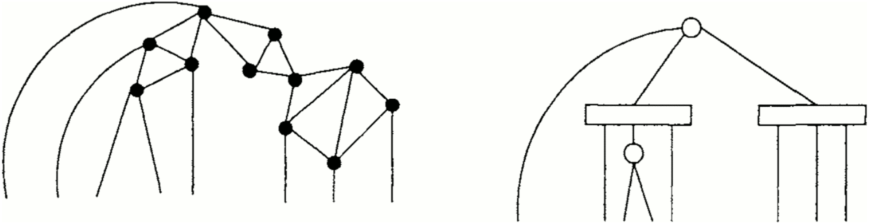
Example of Bush Form



PQ-Tree Data Structure

- Use *PQ*-tree to represent bush form B_k
- *PQ*-tree consists of
  - *P*-nodes: Represents a cut vertex of B_k, and its children can be permuted arbitrarily
  - *Q*-nodes: Represents a biconnected component of B_k, and its children are only allowed to reverse
  - leaves: Represents a virtual vertex of B_k
- *PQ*-tree represents all the permutations and reversions possible in a bush form B_k

Example of *PQ*-Tree



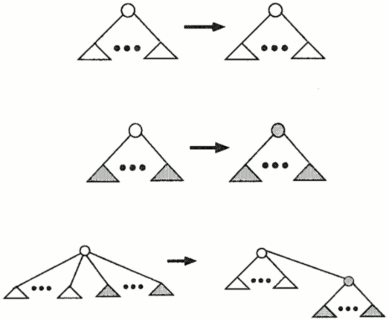
Vertex Addition Algorithm

Lemma

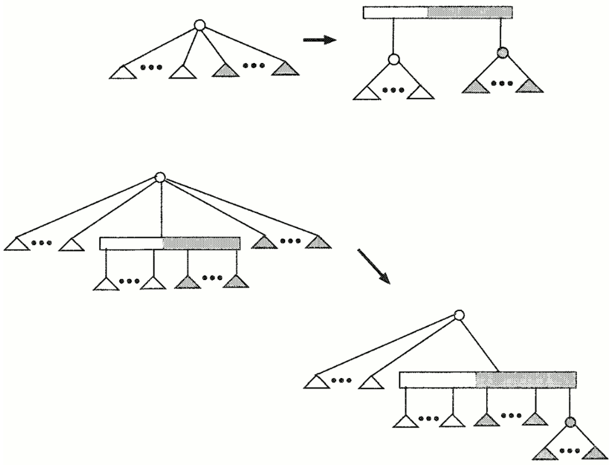
If we have a bush form B_k of a subgraph G_k of a planar graph G, then there exists a sequence of permutations and reversions to make all virtual vertices labeled "$k+1$" occupy consecutive positions

- Idea of the algorithm is to test planarity of G_{k+1} by finding these permutations and reversions
- The permutations and reversions can be found by applying nine transformation rules to the PQ-tree
- A leaf labeled "$k+1$" is pertinent and a pertinent subtree is a minimal subtree of a PQ-tree containing all the pertinent leaves
- A node of a PQ-tree is full if all the leaves of its descendents are pertinent

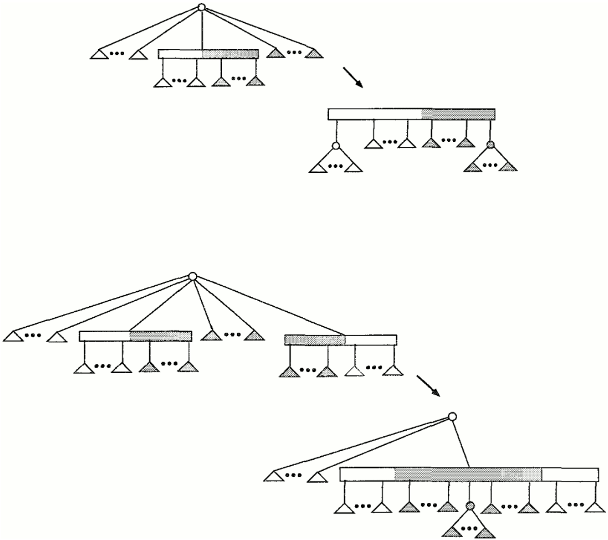
Template matchings



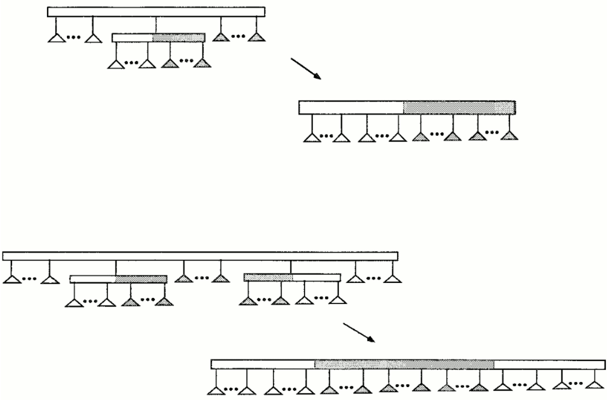
Template matchings



Template matchings



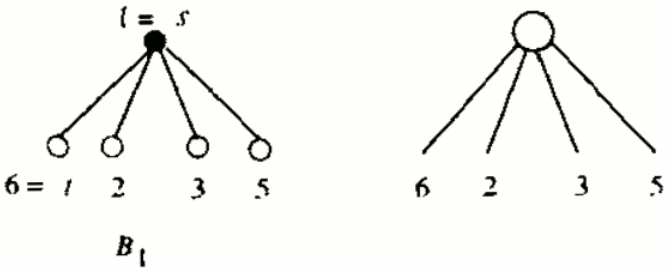
Template matchings



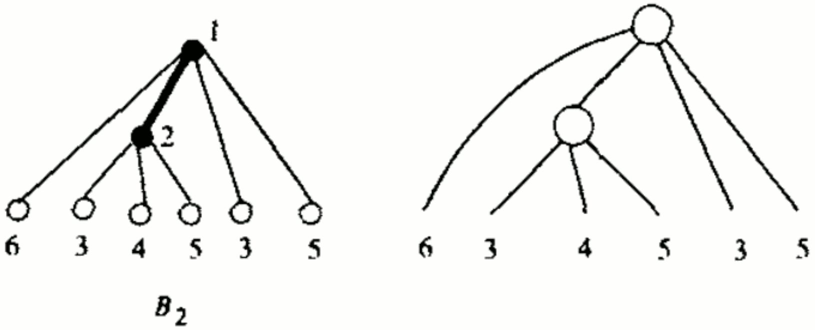
Planarity Testing Algorithm

1. Assign *st*-numbers to the vertices of G
2. Construct PQ-tree corresponding to G'_1
3. Gather pertinent leaves by applying the template matchings
4. If the reduction fails then G is nonplanar
5. Replace full nodes of the PQ-tree by a new P-node
6. Goto 3

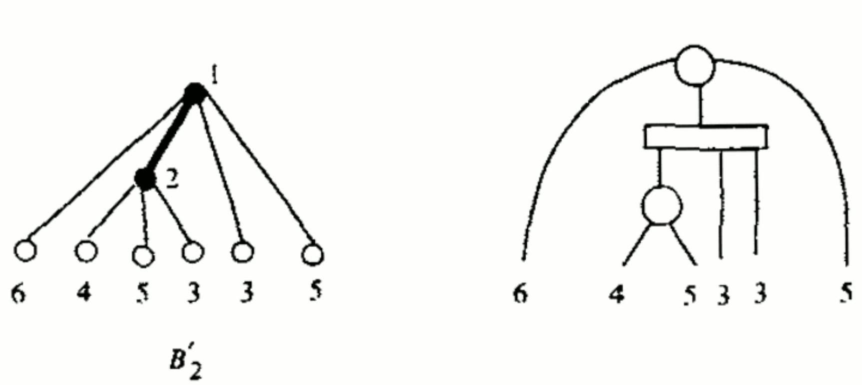
Example of Vertex Addition Algorithm



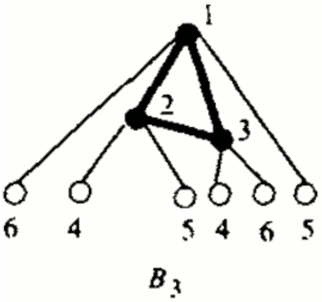
Example of Vertex Addition Algorithm

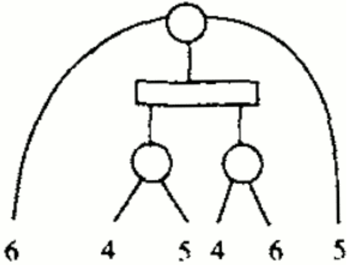


Example of Vertex Addition Algorithm

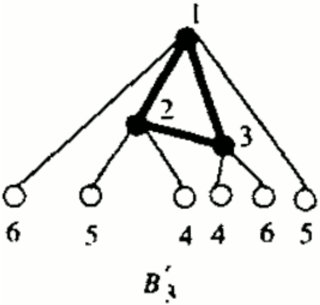


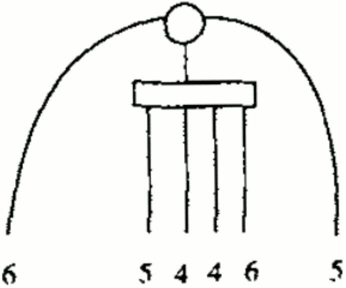
Example of Vertex Addition Algorithm



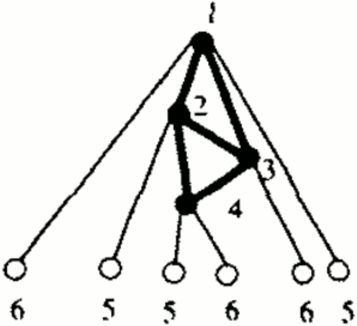


Example of Vertex Addition Algorithm

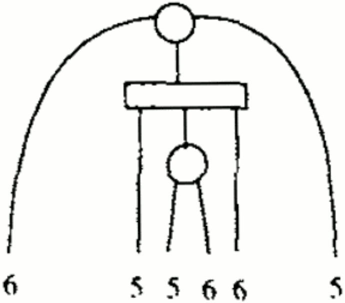




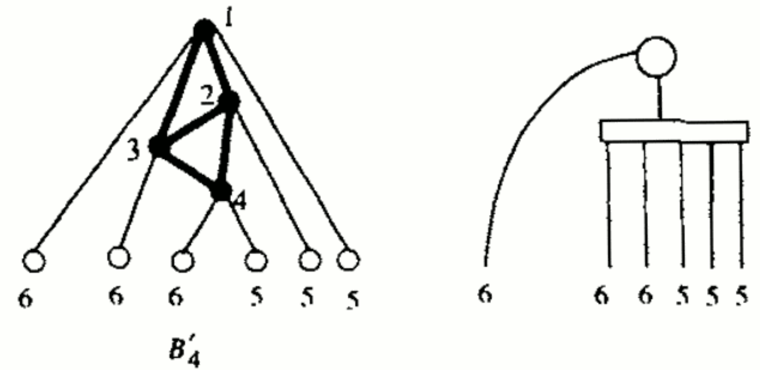
Example of Vertex Addition Algorithm



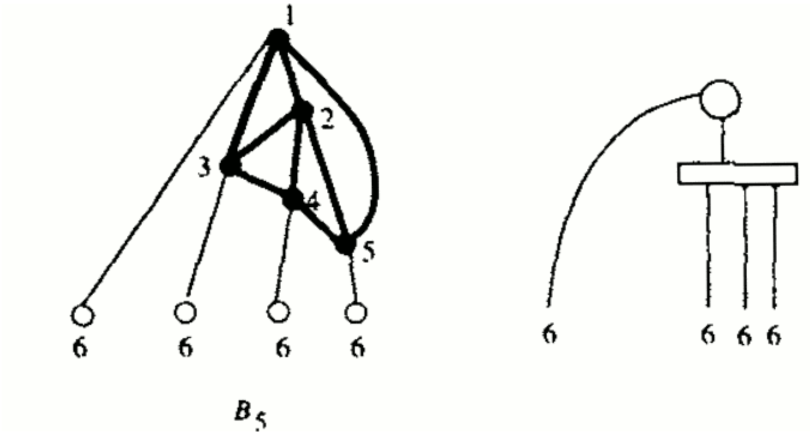
B_4



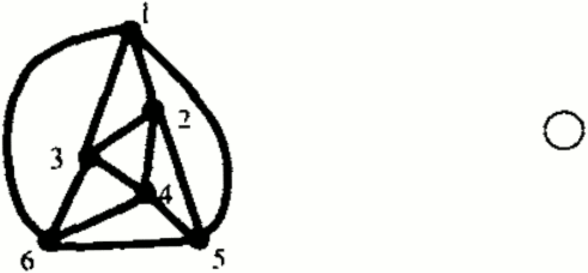
Example of Vertex Addition Algorithm



Example of Vertex Addition Algorithm



Example of Vertex Addition Algorithm



$$G = G_6 = B_6$$

Naive Embedding Algorithm

- Rewrite the adjacency lists of the bush form with each reduction of the PQ-tree
- Updating adjacency lists take time $O(n)$ per reduction step
- Algorithm spends time $O(n^2)$

Upward Embedding

- An upward digraph is a digraph obtained from G by assigning a direction to every edge from the larger vertex to the smaller.
- An upward embedding A_u is an embedding of an upward digraph.
- First determine an upward embedding; second construct entire embedding from upward embedding

Constructing Upward Embedding

- In the vertex addition step for vertex v we can easily construct an upward adjacency list $A_u(v)$ for v
- If v is reversed during the reduction step, then correct $A_u(v)$ by reversing it
- Simple counting algorithm takes time $O(n^2)$

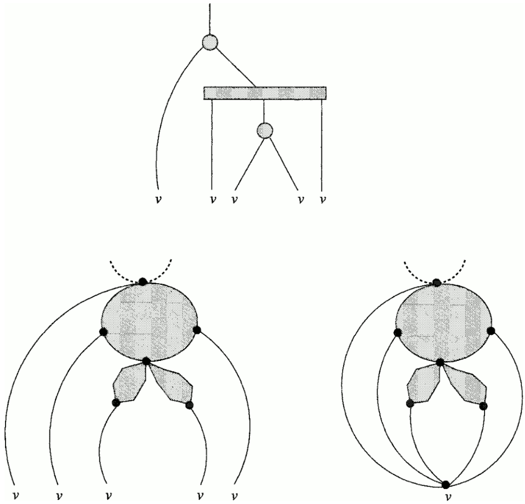
Direction Indicators

- At the vertex addition step for v we add a special "direction indicator" node to the PQ-tree as one of v's siblings
- Indicator is used to track the reversions of v
- Indicator gives the direction of v relative to its brothers, when clockwise ordering of one its brothers is known $A_u(v)$ can be corrected

Upward Embedding Algorithm

- At the vertex addition step for v add to $A_u(v)$ the direction indicators between leaves of v
- If the root of the pertinent subtree is full
  - The pertinent subtree corresponds to a reversible component
  - Assume vertices in $A_u(v)$ are in clockwise order
  - For each direction indication w in $A_u(v)$ which is in opposite direction correct $A_u(w)$ recursively
- Otherwise add direction indicator v as child of the pertinent subtree

Reversible component



Extending A_u into entire embedding

Lemma

In an embedding of a planar graph all neighbours smaller than a vertex v are embedded consecutively around v

Do a depth-first search starting at the sink t on the upward digraph and add vertex y_k to the front of the list $A_u(v)$ when the directed edge (y_k, v) is searched