Branchwidth and b-parses

Michael J. Dinneen, mjd@cs.auckland.ac.nz

Semester 1, 2013

These slides based on Andrew Probert's MSc Thesis (2013)

THE UNIVERSITY OF AUCKLAND

What is branchwidth?

- Branchwidth and treewidth are closely related but have slightly different views.

$$bw(G) \leq tw(G) + 1 \leq \frac{3}{2} bw(G)$$

- Treewidth $t = tw(G)$ focuses on how tree-like a graph G is by decomposing its vertices into cliques or separators of size at most $t + 1$.
- Branchwidth $b = bw(G)$ is about how the edges of a graph G are interconnected in some (unrooted) binary tree hierarchy with leaves being the edges of G. (goal: minimize the shared/separated vertices when cutting the tree's edges)
- Both are NP-hard to determine the width for general graphs.
- However, for planar graphs the branchwidth can be computed in polynomial time. (still open problem for treewidth)

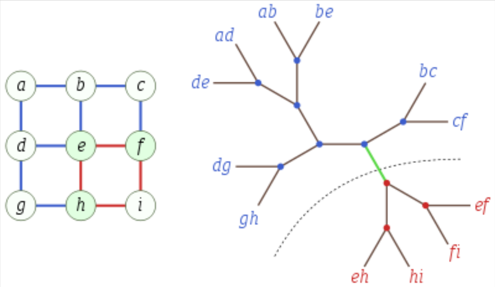
What is branchwidth?

- Branchwidth and treewidth are closely related but have slightly different views.

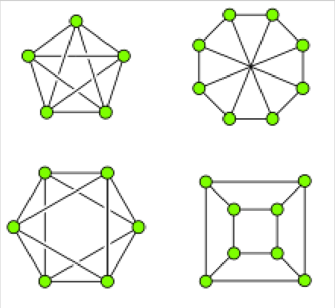
$$bw(G) \leq tw(G) + 1 \leq \frac{3}{2} bw(G)$$

- Treewidth $t = tw(G)$ focuses on how tree-like a graph G is by decomposing its vertices into cliques or separators of size at most $t + 1$.
- Branchwidth $b = bw(G)$ is about how the edges of a graph G are interconnected in some (unrooted) binary tree hierarchy with leaves being the edges of G. (goal: minimize the shared/separated vertices when cutting the tree's edges)
- Both are NP-hard to determine the width for general graphs.
- However, for planar graphs the branchwidth can be computed in polynomial time. (still open problem for treewidth)

en.wikipedia.org/wiki/Branch-decomposition



Branch decomposition of a grid graph, showing an e-separation. The separation, the decomposition, and the graph all have width three.

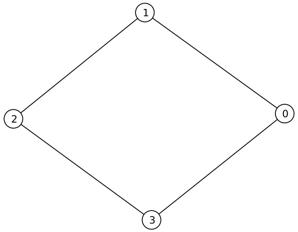


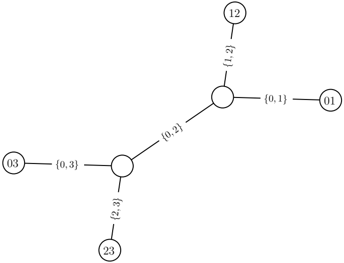
The four forbidden minors for graphs of branchwidth three.

Definition

Given a graph $G = (V, E)$ a **branch decomposition** is a pair (T, m) where T is a tree with every internal node ternary and m is a bijection from E to the leaves of T.
For a branch decomposition (T, m) of a graph G if we cut any edge of T we separate $E(G)$ into two disjoint subsets. The set of vertices of $V(G)$ that belong to edges in both of these subsets is called the **middleset**.
The size of the maximum middleset for a branch decomposition is called its **width** and the **branchwidth** of G is the minimum of these widths across all the branch decompositions of G.

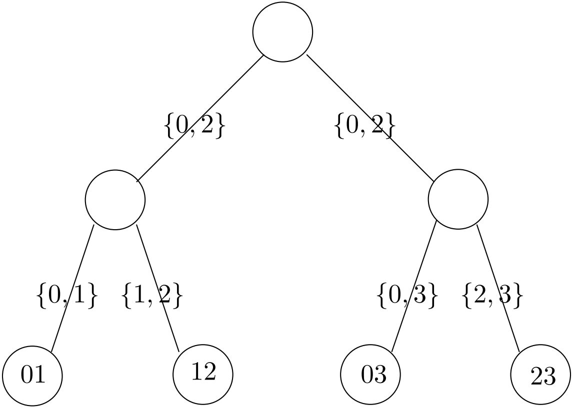
We can **root** the tree T at an internal edge $\{a, b\}$ by adding a new node r to T by subdividing the edge. The edges $\{a, r\}$ and $\{b, r\}$ incident to the root r will have the same middleset.





A graph with branchwidth 2.

A branch decomposition of width 2.



A possible rooted branch decomposition for the graph C_4 with the middlesets given using the vertices of the edge labels.

The *b*-parse (informal) part 1/2

- Recall the *t*-parse data structure for graphs of treewidth $\leq t$ is algebraically represented with two unary operators for adding new boundary vertices, (i), or boundary edges, $\boxed{i\ j}$, and a binary operator, $\oplus$, to combine two boundaried graphs.
- To simplify for use/input in programs, the *t*-parse $\boxed{0\ 1}\ \boxed{1\ 2}$ $\oplus\ \boxed{0\ 2}\ \boxed{0\ 1}$ is represented as 2 ((10 21) (20 10))
- Similarly for *b*-parse trees (branchwidth *b*) can be represented as list-of-list trees. (ignoring middlesets for the moment) The previous C_4 as ((01 + 12) + (03 + 23)) or ((01 12) (03 23))
- We can alternately eliminate the need for the brackets by using reverse polish notation giving the example as 01 12 + 03 23 + +.

THE UNIVERSITY OF AUCKLAND

The b-parse (informal) part 1/2

- Recall the t-parse data structure for graphs of treewidth $\leq t$ is algebraically represented with two unary operators for adding new boundary vertices, i, or boundary edges, $\boxed{i\ j}$, and a binary operator, $\oplus$, to combine two boundaried graphs.
- To simplify for use/input in programs, the t-parse $\boxed{0\ 1}$ $\boxed{1\ 2}$ $\oplus$ $\boxed{0\ 2}$ $\boxed{0\ 1}$ is represented as 2 ((10 21) (20 10))
- Similarly for b-parse trees (branchwidth b) can be represented as list-of-list trees. (ignoring middlesets for the moment) The previous C_4 as ((01 $+$ 12) $+$ (03 $+$ 23)) or ((01 12) (03 23))
- We can alternately eliminate the need for the brackets by using reverse polish notation giving the example as 01 12 $+$ 03 23 $+$ $+$.

The b-parse (informal) part 2/2

- For a rooted version of a branch decomposition we can distinguish the edges (and their middlesets) incident to a vertex as outer (e.g. on path to root)
  or as inner (e.g. away from root to subtree).
- For b-parse we need to reuse labels for edges in the build process so need to indicate how each of the two inner edges are joined to create the middleset for the outer edge.
  (The labels not in the outer middleset can be recycled.)
- We annotate the b-parse operator $+$ with a subscript indicating the outer middleset. For example the graph C_4 is now represented as 2 01 12 $+_{02}$ 01 12 $+_{02}$ $+$
  (The first element denotes $b = 2$ and the old '3' reuses '1'.)

The b-parse (formalized) part 1/2

Definition

For the b-**parse operator set** Σ_b is composed of the following operators: edge operators L and composition operators $+_{ij\ldots k}$ for $0 \leq i < j < \ldots < k < \left\lfloor \frac{3}{2} b \right\rfloor$.
Let G and H be b-terminal binary decomposition trees (or subtrees) then the operators, read left to right, are defined as:
$G\ L$ Add a new edge to the decomposition where L is one of three possible values depending on the size of the middleset M for this edge:

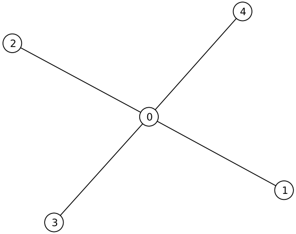
- If $|M| = 0$ then $L = K_2$ where this is a disconnected edge.
- If $|M| = 1$ then $L = P_i$ $(0 \leq i \leq b+1)$ where this is a pendant edge with only the vertex labelled i in the outer middleset
- If $|M| = 2$ then $L = ij$ $(0 \leq i < j \leq b+1)$ where both of the vertices i and j of this edge are in the outer middleset.

$G\ H\ +_{ij\ldots k}$ $(0 \leq i < j < \ldots < k < \left\lfloor \frac{3}{2} b \right\rfloor)$ Join two subtrees with an outer middleset consisting of vertices with labels $\{i, j, \ldots, k\}$ such that $|\{i, j, \ldots, k\}| \leq b$.

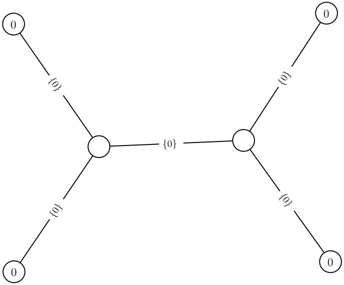
The *b*-parse (formalized) part 2/2

- The middleset $\{i, j, ..., k\}$ must always be a subset of the union of the middlesets of the two preceding entries and the size of this outer middleset must be at most b.
- The labels can range from 0 to $\left\lfloor \frac{3}{2} b \right\rfloor - 1$ inclusive.
- If there is no $+$ operator then there must be at most one K_2 edge operator.
- For computer representation, the first element of the *b*-parse must always be the width b.
- The final join operation must have an empty middleset.
- Finally, in the decimal representation only *b*-parses of width $b \leq 7$ are allowed.

Example star S_4: $1\ P_0\ P_0\ +_0\ P_0\ P_0\ +_0\ +$



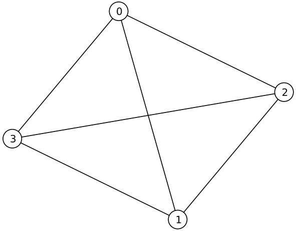
A star S_4.



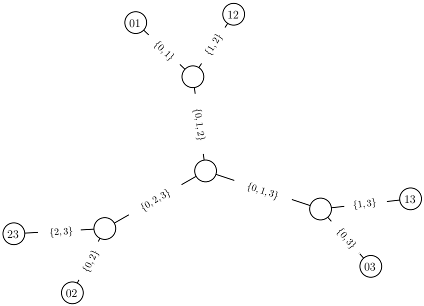
A *b*-parse of width 1.

Example clique K_4:

$3\ 01\ 12\ +_{012}\ 02\ 23\ +_{023}\ +_{013}\ 03\ 13\ +_{013}\ +$



The clique K_4.



A *b*-parse of width 3.

Graph algorithms using b-parses

- We can build algorithms for b-parses (like for t-parses) by defining a subsolution state for each of the subgraphs induced from the subtrees of the b-parse.
- In most cases this dynamic programming method uses all possible subsets of the middlesets as properties/indices into the "state".
- In what follows we call the collection of substates, denoted by V, as solution (or value) sets.
- For the join operation, the boundary B (for outer middleset) will be defined by the relevant inner middlesets.
  Moreover, for any two value sets V' and V'' the associated boundaries B' and B'' need not be the same.

algorithm `processBParse`
    Input: width b, parse $[g_1, ..., g_n]$ and a stack
begin
    initialize solution set V for the operator g_1 and push V on stack
    while there are more operators g_i **do**
    begin
        if g_i is a leaf operator L **then**
        begin
            initialize solution set V for g_i
            push V onto the stack
        end
        if g_i is a join operator $+_B$ where B is the boundary for the join **then**
        begin
            pop solution sets V' and V'' from the stack
            join V' and V'' and produce V given the boundary B
            push V onto the stack
        end
    end
    pop the last element V from the stack
    return V
end

Vertex Cover   part 1/2

Definition (Vertex-Cover)

Input: Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$
Question: Does there exist $W \subseteq V$ such that every edge in E has a vertex in W and $|W| \leq k$?

- Let B be the boundary vertices of the subgraph induced by a subtree of a *b*-parse.
- The solution set will be the set of $V[S]$, for $S \subseteq B$, where $V[S] = min\{|W| : W$ is a valid vertex cover and $S = W \cap B\}$
- For leaf nodes of a *b*-parse, it is easy to see the initial $V[S] = |S|$ if $|S| = 1$ or 2; and $V[\emptyset] = 1$ or ∞.

[disconnected edge (1), pendant edge (1), boundary edge (1, 2 or ∞)]

THE UNIVERSITY OF AUCKLAND

Vertex Cover part 1/2

Definition (Vertex-Cover)

Input: Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$
Question: Does there exist $W \subseteq V$ such that every edge in E has a vertex in W and $|W| \leq k$?

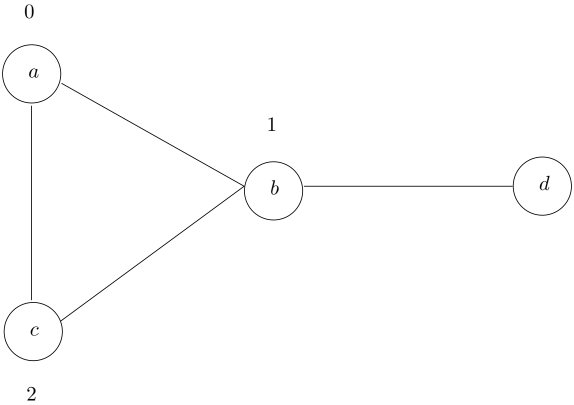
- Let B be the boundary vertices of the subgraph induced by a subtree of a b-parse.
- The solution set will be the set of $V[S]$, for $S \subseteq B$, where $V[S] = min\{|W| : W$ is a valid vertex cover and $S = W \cap B\}$
- For leaf nodes of a b-parse, it is easy to see the initial $V[S] = |S|$ if $|S| = 1$ or 2; and $V[\emptyset] = 1$ or ∞.

[disconnected edge (1), pendant edge (1), boundary edge (1, 2 or ∞)]

Vertex Cover part 2/2

- Each join will consist of two specified input solution sets and an outer middleset for the join.
- The root solution set will always have an empty outer middleset.
- If B' and B'' are the middlesets for the two input solution sets we calculate the matching values of $S \subseteq B$ given B' and B'' as: $S = (S' \cup S'') \cap B$ for some $S' \subseteq B'$ and $S'' \subseteq B''$
- The solution sets for join is given by: $V[S] = min\{V'[S'] + V''[S''] - |S' \cap S''| : \forall S' \subseteq B' \ \forall S'' \subseteq B''\}$ such that $S = (S' \cup S'') \cap B$

Example *b*-parse for Vertex Cover and 3-Coloring Algs



This graph of branchwidth 2 has the following possible *b*-parse:

$2\ 01\ P_1\ +_{01}\ 02\ 12\ +_{01}\ +$

Sequence of solution sets for Vertex Cover for the example b-parse.

S	Value
$\emptyset$	2

$+$

S	Value
$\emptyset$	∞
0	2
1	1
0, 1	2

$+_{01}$

S	Value
$\emptyset$	1
0	2
1	2
0, 1	2

$+_{01}$

S	Value
$\emptyset$	∞
0	1
1	1
0, 1	2

01

S	Value
$\emptyset$	1
1	1

P_1

S	Value
$\emptyset$	∞
0	1
2	1
0, 2	2

02

S	Value
$\emptyset$	∞
1	1
2	1
1, 2	2

12

THE UNIVERSITY OF AUCKLAND

Graph 3-Coloring part 1/2

Definition (3-Vertex Coloring)

Input: Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$
Question: Do there exist sets of vertices X_1, X_2, X_3 such that $X_1 \cup X_2 \cup X_3 = V(G)$, $X_i \cap X_j = \emptyset$ for $i \neq j$, and the two vertices of any edge do not belong both to X_i for any i?

- A solution set is indexed by the members of the relevant middleset and consists of multiple possible colorings of the members of the middleset.

[So for a solution set, $V[(0, 1)] = \{(1, 2), (2, 1)\}$ means that the middleset $\{0, 1\}$ with vertices 0 and 1 colored 1 and 2, or swapped, is a valid coloring.]

- The solution sets for the three base cases of an edge or leaf operator are defined as follows.
  disconnected edge: true; **pendant edge**: true for any boundary coloring; **boundary edge**: true for all ways of using two distinct colors.

Graph 3-Coloring part 1/2

Definition (3-Vertex Coloring)

Input: Let $G = (V, E)$ be a graph and $k \in \mathbb{N}$
Question: Do there exist sets of vertices X_1, X_2, X_3 such that $X_1 \cup X_2 \cup X_3 = V(G)$, $X_i \cap X_j = \emptyset$ for $i \neq j$, and the two vertices of any edge do not belong both to X_i for any i?

- A solution set is indexed by the members of the relevant middleset and consists of multiple possible colorings of the members of the middleset.

[So for a solution set, $V[(0,1)] = \{(1,2),(2,1)\}$ means that the middleset $\{0,1\}$ with vertices 0 and 1 colored 1 and 2, or swapped, is a valid coloring.]

- The solution sets for the three base cases of an edge or leaf operator are defined as follows.
  disconnected edge: true; **pendant edge**: true for any boundary coloring; **boundary edge**: true for all ways of using two distinct colors.

Graph 3-Coloring part 2/2

algorithm `joinSolutions`
  Input: solution sets V', V''
begin
  for each coloring $s_j \in V'$
    for each coloring $s_k \in V''$
      if s_j is compatible with s_k
      begin
        unite s_j and s_k and return a solution for V_i
      end
end

where $s_j \in V'$ and $s_k \in V''$ are compatible if $\forall c \in B' \cap B''$ $[s_j(c) = s_k(c)]$ where B' and B'' are the inner middlesets and B is the outer middleset for the join.
We unite s_j and s_k by forming $s \in V$ as follows:
$s = \bigcup_{c' \in (B' \cap B) - (B' \cap B'')} [s_j(c')] \ \cup \ \bigcup_{c' \in (B' \cap B'') \cap B} [s_j(c')] \ \cup$
$\bigcup_{c'' \in (B'' \cap B) - (B' \cap B'')} [s_k(c'')] \ \forall c' \in B' \ \forall c'' \in B''$ s.t. $c', c'' \in (B' \cup B'') \cap B$

Further comments on 3-Coloring

Some special cases occur where a middleset is empty.
In these cases we apply the logic:

algorithm `processEmptySet`
  Input: solution sets V', V'', middlesets B, B' and B''
begin
  if $B = \emptyset$ and $B' = B''$ **then**
  begin
    if $V' \cap V'' \neq \emptyset$ **then** $V(\emptyset) = 1$
    else leave V empty
  end
  else if $B' \cap B''$ is empty **then** $V = V' \times V''$
end

3-Coloring algorithm runs in time $O(3^{2b}n)$, where $n = |V(G)|$.
3-Coloring algorithm uses space $O(b3^{b}m)$, where $m = |E(G)|$.

Sequence of solution sets for 3-Coloring for the example b-parse.

c	Solutions
$\emptyset$	$\emptyset$

$+$

c	Solutions					
0	1	2	1	3	2	3
1	2	1	3	1	3	2

$+_{01}$

c	Solutions								
0	1	1	2	2	1	3	3	2	3
1	1	3	2	3	2	2	3	1	1

$+_{01}$

c	Solutions					
0	1	2	1	3	2	3
1	2	1	3	1	3	2

01

c	Solutions		
1	1	2	3

P_1

c	Solutions					
0	1	2	1	3	2	3
2	2	1	3	1	3	2

02

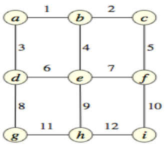
c	Solutions					
1	1	2	1	3	2	3
2	2	1	3	1	3	2

12

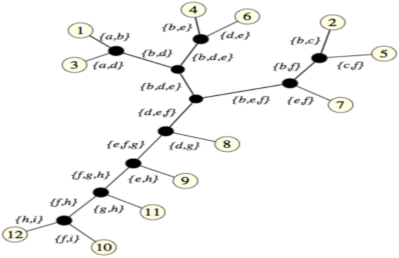
THE UNIVERSITY OF AUCKLAND

Dorn and Telle's "two birds": `http://www.ii.uib.no/~frederic/DT.pdf`

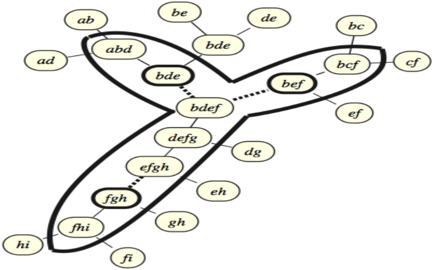
G



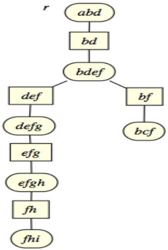
BD(G)



TD(G)



semi-nice TD(G)



THE UNIVERSITY OF AUCKLAND