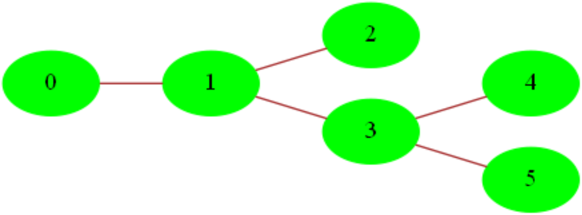
Treewidth

Kai Wang, Zheng Lu, and John Hicks

April 1^{st}, 2015



Outline

- Introduction
  - Definitions
  - Examples
  - History
- Algorithms and Implementations
  - Theorems
  - Pseudocode
  - Heuristics
- Implementations
  - Results
  - Regression
- Applications
- Open Problems
- References
- Homework

Introduction

Treewidth is a graph parameter that measures how "treelike" a graph is. Using tree-decompositions will allow many NP-hard problems to be solved quickly with dynamic programming on graphs of bounded treewidth.

Definitions and Notation

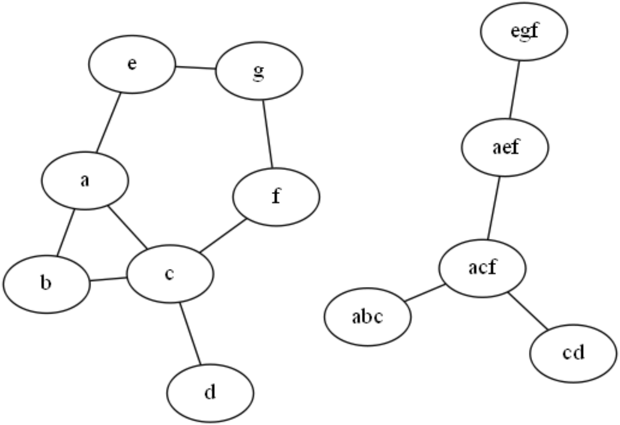
Tree Decomposition

A tree decomposition of a graph G is a pair (T,X), where T is a tree and $X=\{X_t\colon t\in V(T)\}$ is a family of subsets of V(G), often referred to as *bags*, such that:

- For every edge $\{u,v\}$ of G, there exists $t\in V(T)$ with $u,v\in X_t$
- For every pair y,z of vertices of T, if w is any vertex in the path between y and z in T then $X_y\cap X_z \subseteq X_w$

See Reference: 1

Example of Tree Decomposition



Definitions and Notation

Width

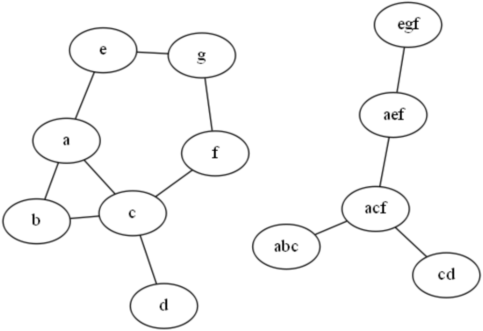
Width of Tree Decomposition is $max\{|X_t| - 1 : t \in V(T)\}$.

Treewidth

Treewidth, denoted **tw**(G), is minimum width of a tree decomposition of G.

Note: This is among all possible tree decompositions of G.

Example of Width and Treewidth



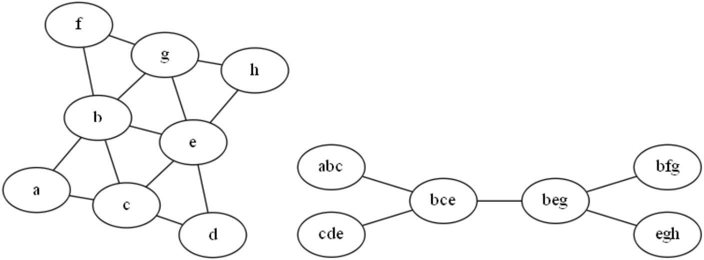
From our previous example, we see that, by definition, this decomposition has a width of 2, G is not a tree, so **tw**(G)=2.

Equivalent Characterizations of Treewidth

- The treewidth of G is one less than the size of the largest clique in the chordal graph containing G with the smallest clique number.
- A graph G has treewidth k iff it has a haven of order $k + 1$, but of no higher order.
- The treewidth of a graph is one less than the maximum order of a bramble.

See Reference: 8

Example of Treewidth

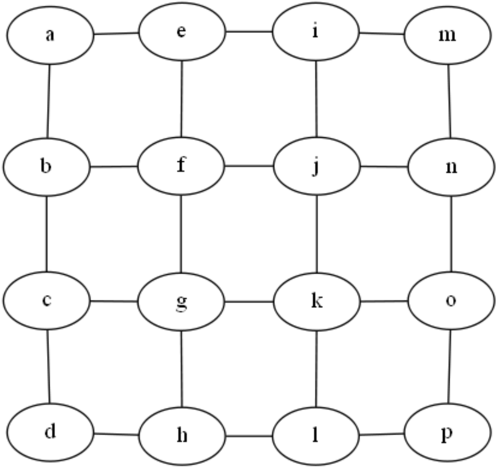


Here is a graph G and its Tree Decomposition.

Question

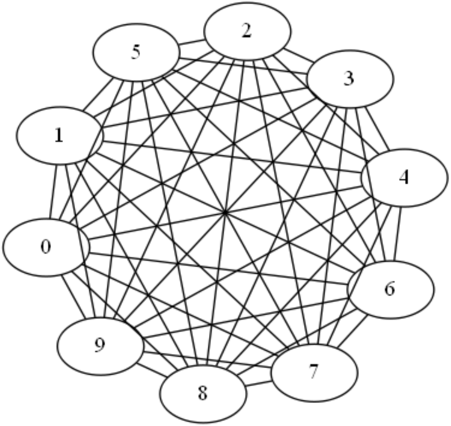
What is the **tw**(G)?

Example: k×k Grids



The example above is the 4×4-grid. This graph has **tw**(G)=4. In general, the k× k-grid has **tw**(G)=k

Example: Complete Graph K_n



In general, the graph K_n has $\textbf{tw}(K_n)=n-1$, and thus the above example of K_{10} has $\textbf{tw}(K_{10})=9$

Some Other Properties of Treewidth

- A connected graph G has **tw**(G)=1 iff G is a tree
- A cycle has treewidth of 2
- Every non-empty graph of treewidth at most w has a vertex of degree at most w
- Graphs with treewidth at most k have been referred to as *partial k-trees*
- Series Parallel graphs have treewidth at most 2
- Graphs of treewidth at most k are closed under taking minors
- A planar graph on n vertices has treewidth $O(\sqrt{n})$

See Reference: 3,9

Propery of Treewidth: Forbidden Minors

By the Graph Minor Theorem there is a finite set of forbidden minors for graphs of treewidth at most k, all of which are not known except for small k

tw(G)≤k	Forbidden Minor
k=0	K_2
k=1	K_3
k=2	K_4
k=3	K_5 and three other graphs
k=4	over 75

See Reference: 9

History

- Introduced by Umberto Bertelé and Francesco Brioschi under the name of **dimension** in 1972
- Rediscovered by Rudolf Halin because of similar properties with **Hadwiger number**, that is the size of the largest complete graph that can be obtained by contracting edges, in 1976
- Again, by Neil Robertson an Paul Seymour in 1984, on their work with the Graph Minors Project
- Since then, it has been extensively studied by many authors including Hans L. Bodlaender

See References: 6,8

Discussion from Paul Seymour

Quote:
"We[Robertson and Seymour] found a structure theorem for the planar graphs that did not contain any fixed planar graph as a minor (this was easy); it was bounded tree-width. We proved that for any fixed planar graph, all the planar graphs that did not contain it as a minor had bounded tree-width."

This structure theorem helped prove Wagner's conjecture for all planar graph, and this lead to studying more of treewidth and a discovery that it worked well for algorithms.

See Reference: 5

Algorithms

Theorems

- **[Arnborg, Corneil, Proskurowski '87]**
  Deciding whether the treewidth of a given graph is at most k is NP-complete
- **[Bodlaender '96]**
  For any $k \in \mathbb{N}$, $\exists$ a linear time algorithm to test whether a given graph has treewidth at most k and finds a tree decomposition of width at most k, with runtime being exponential in k^3

See Reference: 3

Alg1: Dynamic Programming for Treewidth Pseudocode

ALGORITHM 1: Dynamic-Programming-Treewidth(Graph $G = (V, E)$)

Set $TW(\emptyset) = -\infty$.
for i = 1 to n **do**
  for all sets $S \subset V$ with $|S| = i$ **do**
    Set $TW(S) = \min_{v \in S} \max\left\{TW(S - \{v\}), |Q(S - \{v\}, v)|\right\}$
  end for
end for
return $TW(V)$

This algorithm uses $O^*(2^n)$ time.

See Reference: 4

Alg4: Algorithm TWDP

ALGORITHM 4: Algorithm TWDP (Graph $G = (V, E)$, clique $C \subseteq V$)

```
n = |V|.
Compute some initial upper bound up on the treewidth of G. (For example, set up = n − 1.)
Let TW_0 be the set, containing the pair (∅, −∞).
for i = 1 to n − |C| do
  Set TW_i to be an empty set.
  for each pair (S, r) in TW_{i−1} do
    for each vertex x ∈ V − S do
      Compute q = |Q(S, v)|.
      Set r' = min{r, q}.
      if r' < up then
        up = min{up, n − |S| − 1}
        if There is a pair (S ∪ {x}, t) in TW_i for some t then
          Replace the pair (S ∪ {x}, t) in TW_i by (S ∪ {x}, min(t, r')).
        else
          Insert the pair (S ∪ {x}, r') in TW_i.
        end if
      end if
    end for
  end for
end for
if TW_{n−|C|} contains a pair (V − C, r) for some r then
  return r
else
  return up
end if
```

See Reference: 4

Possible Heuristics on Upper Bounds

- **Minimum Degree** : In tree decompositions, can be used to obtain an elimination ordering.
  - Take a vertex of minimum degree
  - Make neighbors of chosen vertex a clique
  - Remove chosen vertex and repeat on rest of graph G
  - Add chosen vertex with neighbors to tree decomposition
- **Minimum Fill-In**: This heuristic is similar to minimum degree but, chooses a vertex v, where the number of edges added when turning the chosen vertex's neighborhood into a clique is as small as possible.
- **Minimum Separating Vertex Sets**: Starts with a trivial tree decomposition and refines it by a stepwise process using minimum separators.

See Reference: 2,3

Possible Heuristics on Lower Bounds

- Minimum Degree is a lower bound on the treewidth, since if G had treewidth k, then it has a vertex of degree at most k
- The treewidth of a sub-graph of G is at most the the treewidth of G. This can help compute a lower bound by:
  - Set $k = 0$
  - While G is nonempty, select a vertex v of minimum degree
  - Set $k{=}max\{k, deg(v)\}$ and remove v and incident edges from G
  - Repeat
- By contracting an edge of a vertex to a neighbor of minimum degree or to a neighbor that has few common neighbors, a lower bound can be placed on treewidth. Since the contraction of an edge will not increase the treewidth.

See Reference: 3

Implementations

What follows are the results
of the implementations of Alg1 and Alg4

Graphs Tested with Respective Treewidths

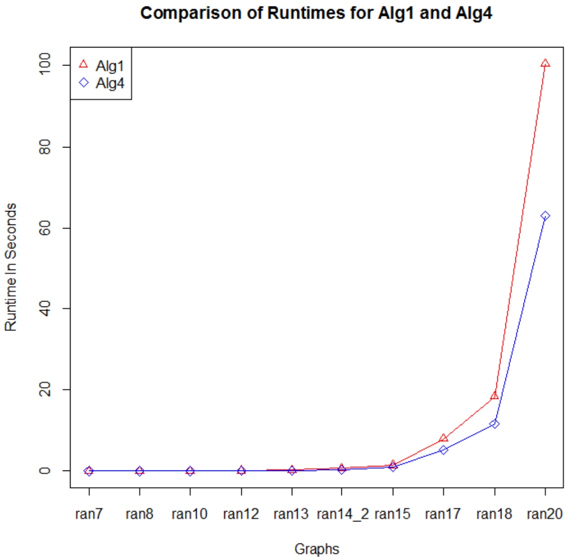
Graph	Number of Vertices	Number of Edges	**tw**(G)
ran7	7	14	3
ran8	8	13	3
ran10	10	31	6
ran12	12	31	5
ran13	13	23	3
ran14_2	14	45	7
ran15	15	39	6
ran17	17	49	7
ran18	18	50	6
ran20	20	67	8

Results of Implemented Algorithm's Runtime

graph	Alg1	Alg4
ran7	0	0
ran8	0	0
ran10	20	10
ran12	100	70
ran13	220	140
ran14_2	680	410
ran15	1450	920
ran17	7950	5220
ran18	18350	11710
ran20	100430	63010

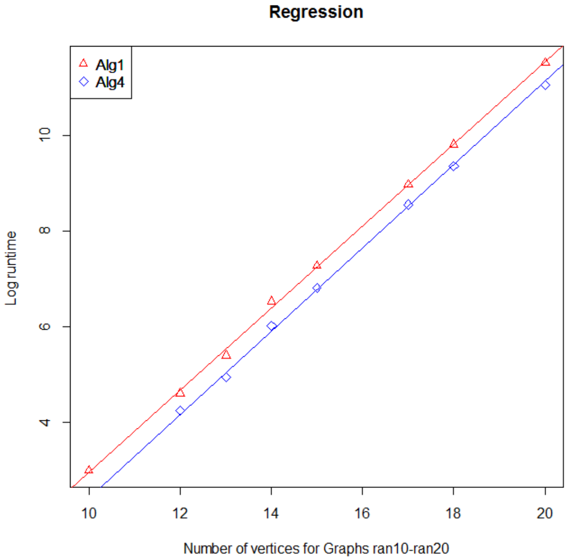
The runtime is measured in ms.

Plot of Runtime Shown in Seconds



Plot of log(Runtime) versus Number of Vertices

Least Squares Regression was performed.



Results of Regression For Alg1 and Alg4

The Least Square Regression Equations are as follows (Alg1,Alg4 refers to runtime in ms):

- $log(Alg1) = -5.639 + .859 * (number\ of\ vertices)$
  Pearson Correlation $= .9996$
- $log(Alg4) = -6.2975 + .871 * (number\ of\ vertices)$
  Pearson Correlation $= .9995$

Comparison of Predicted vs Observed Runtimes in ms for Alg1

Number of Vertices	Observed runtime	Predicted runtime
10	20	19.13112
12	100	106.62449
13	220	251.71857
14	680	594.25595
15	1450	1402.91648
17	7950	7818.94882
18	18350	18458.93530
20	100430	102878.16322

Comparison of Predicted vs Observed Runtimes in ms for Alg4

Number of Vertices	Observed runtime	Predicted runtime
10	10	11.20281
12	70	64.00139
13	140	152.97508
14	410	365.63854
15	920	873.94326
17	5220	4992.81865
18	11710	11933.75365
20	63010	68177.27222

Applications

Some NP-hard problems on arbitrary graphs become linear or polynomial time solvable on graphs of bounded treewidth. Some of these include:

- Hamiltonian Circuit
- Independent Set
- Vertex Cover

Note: The algorithms usually require dynamic programming

See Reference: 3

Applications: Probabilistic Network

Inference, concerning taking observed variables and trying to compute a probability distribution for the other variables, is a common problem of interest. This problem is $\#P - \mathit{hard}$, but can be solved in linear time when the moralized graph has small treewidth with the Lauritzen-Spiegelhalter algorithm.

See Reference: 2,3

Applications: Electrical Networks

There are three laws that allow us to transform networks to smaller, equivalent networks.These are:

- Series Rule
- Parallel Rule
- Star - Triangle Rule

The order in which we apply the laws to the vertices makes a difference, and treewidth can be used to determine for which networks there exists a series of applications that yield a single edge. This is seen by a **Proposition** from Bodlaender:

Let G' =(V,E $\cup \{s,t\}$) be a biconnected graph. There exists a series of rule applications that reduces G to the single edge $\{s,t\}$ iff G has treewidth at most three.

See Reference: 3

Applications: Hosoya Index

The Hosoya index, or Z index, in a graph is the total number of matchings in the graph. The Hosoya index is the number of non-empty matchings plus one in a graph. This is $\#P - complete$ to compute, but is fixed-parameter tractable for graphs of bounded treewidth.

See Reference:6

Future Discussions Open Problems

- Is there a (polynomial-time) constant-factor approximation algorithm for treewidth?
- Can Treewidth be computed in polynomial time on planar graphs?

See Reference: 9

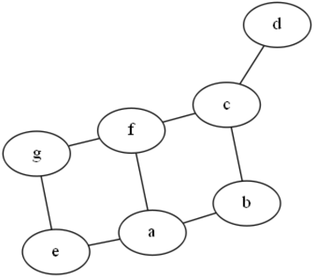
References

1. Bienstock, Daniel and Langston, Michael A., "Algorithmic Implications of the Graph Minor Theorem", *Handbook of Operations Research and Management Science: Volume on Networks and Distribution* Chapter
2. Bodlaender, Hans L., *Treewidth: Algorithms and Networks*, www.cs.uu.nl/docs/vakken/an/an-treewidth.ppt
3. Bodlaender, Hans L., "Treewidth: Characterizations, Applications, and Computations" ,2006
4. Bodlaender, H. L., Fomin, F. V., Kratsch, D., Koster, A. M. C. A., and Thilikos, D. M. 2012. "On exact algorithms for treewidth". ACM Trans. Algor. 9, 1, Article 12
5. http://cstheory.stackexchange.com/questions/5018/the-origin-of-the-notion-of-treewidth
6. http://en.wikipedia.org/wiki/Hadwiger_number
7. http://en.wikipedia.org/wiki/Hosoya_index
8. http://en.wikipedia.org/wiki/Treewidth
9. Wagner, Uli, *Graphs & Algorithms: Advanced Topics Treewidth*,http://www.ti.inf.ethz.ch/ew/lehre/GA10/lec-treewidth-new.pdf

Homework

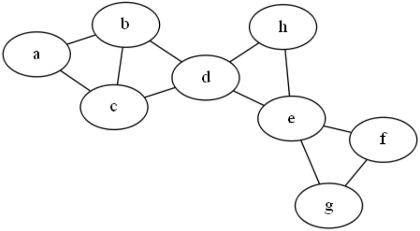
Please send solutions to zlu12@vols.utk.edu

- 1. For the following graph, find the tree decompostion with a width of 2.



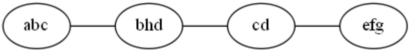
Homework

- 2. Multiple choice: On the next slide you will find 3 choices for a tree decomposition of the following graph. Choose the correct tree decomposition and state the treewidth.

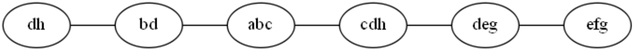


Homework

A



B



c