A Fast Parallel Branch and Bound Algorithm for Treewidth

Yang Yuan

Peking University

November 8, 2011

What is treewidth?

- A **tree decomposition** is a mapping of a graph into a tree. The **treewidth** measures the number of vertices mapped onto any tree node in an *optimal* tree decomposition. Generally, treewidth describes how much the graph *resembles* a tree.
- To **eliminate** a vertex v is to make the its neighbors to form a clique after removing it.
- Treewith closely relates to **elimination ordering**, which is the ordering of vertices for elimination.
- The **width** of an elimination ordering is the *maximum* degree of the vertices when they are eliminated from the graph.
- Treewidth equals to the *minimum* width over all possible elimination orderings.

Why study treewidth

- Treewidth is used widely in many fields, including Constraint Satisfaction Problem, Probabilistic Inference, Bucket Elimination, Recursive Conditioning, etc.
- The complexity of the algorithm is exponential in the treewidth of the corresponding graphs. The smaller the treewidth is, the faster the algorithm will be.
- Treewidth measures the size of a "core" in the graph. It indicates how well the graph is globally connected.
- According to (Lokshtanov et al., 2010), if Strong Exponential Time Hypothesis is true, current best known algorithms for a number of well-studied problems on graphs of bounded treewidth are essentially the best possible. Examples: INDEPENDENT SET, DOMINATING SET, MAX CUT, etc.

Previous techniques

- QuickBB (Gogate and Dechter, 2004)
  - is a branch and bound algorithm.
  - has the search space of size $\Theta(n!)$.
- BestTW (Dow and Korf, 2007)
  - improves QuickBB as it searches only $\Theta(2^n)$ nodes, and utilizes best-first search.
  - uses memory-efficient representations for intermediate graphs, which incur the overhead of intermediate graph generation.
- COMB (Zhou and Hansen, 2009)
  - is the state-of-the-art algorithm.
  - uses breadth-first search to save memory.
  - introduces an efficient depth-first search approach to reduce the average generating time of intermediate graphs.

Search Space

FPBB is based on depth-first search. It is proved that (Bodlaender et al., 2006), the same set of vertices eliminated from the original graph in any order will produce the same intermediate graph. So the search space will be $\Theta(2^n)$. We only need to store the eliminated vertex set to represent current search node.

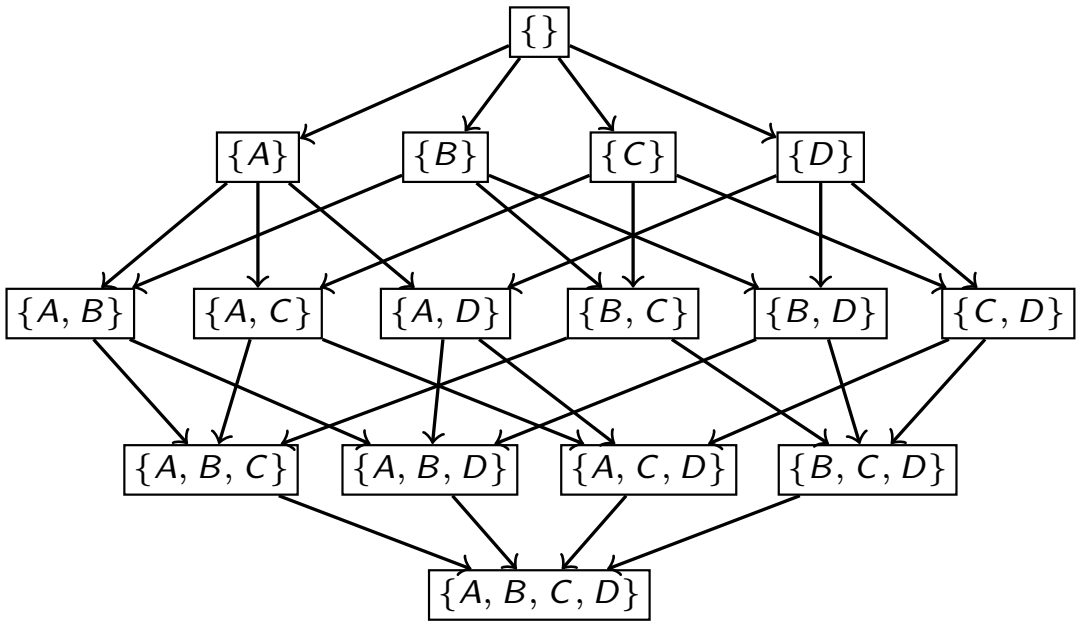


Figure: 2^n Intermediate Graphs in Search Space

Multithreading Techniques

- We use multithreads to improve the running speed. There are two kinds of threads: working threads and free threads. Each time one working thread wants to expand new search nodes, it will try to allocate some of these nodes to the free threads.
- A shared hash table is maintained for duplicated nodes detection. It is a reader-writer problem, but we use a different solution. Readers can always access the table, while only one writer can access the table at one time.

Multithreading Techniques

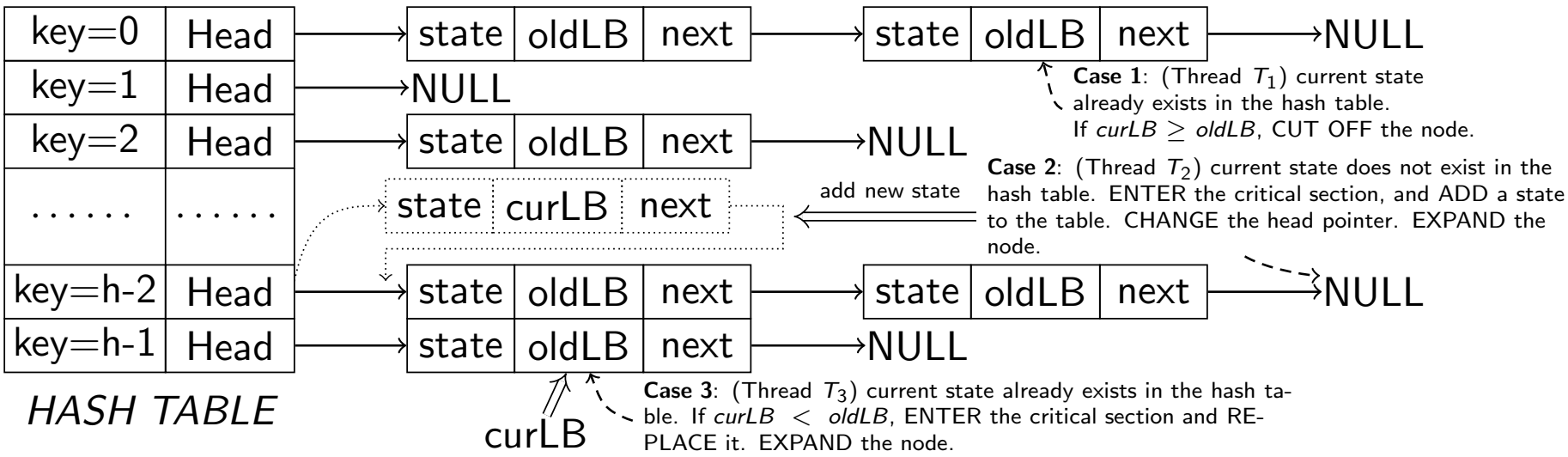


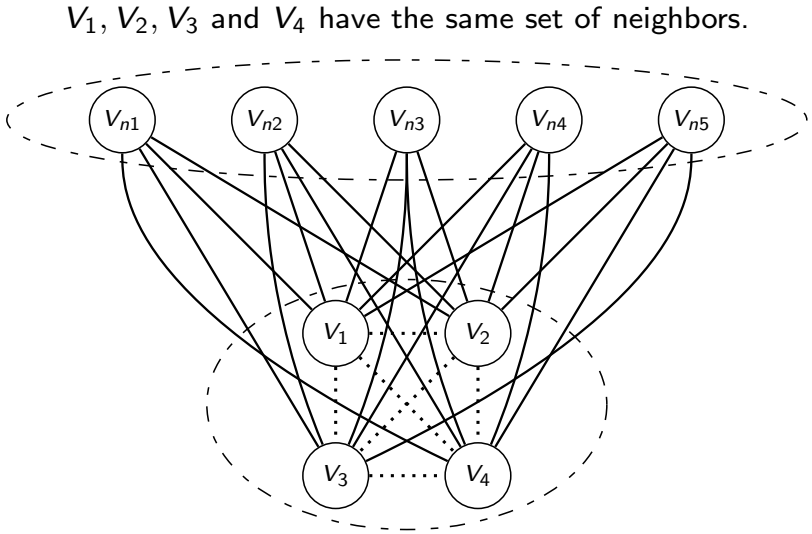
Figure: Different Cases for Expanding a New Node

Similar Group

We introduce a new heuristic called Similar Group. Two vertices v_0 and v_1 are similar if they share the same set of vertices that are adjacent to them.

Similar property is an equivalence relation in the graph. We can use disjoint sets to find all the similar groups in a graph.

The key idea of this heuristic is that all the vertices in the same similar group are interchangeable in the elimination ordering. If a similar group has s vertices, then ideally this property might reduce the search space to $1/s!$ of the original size.



Similar Group

- Similar groups can be found very fast.
  1. enumerate every pair of vertices
  2. if they are similar, merge them using disjoint sets
  3. all the similar groups can be found in $\Theta(n^3)$ before the search
- Similar groups can be used conveniently during the search.
  1. for all the vertices $v_1, v_2, \cdots, v_t$, we sort them arbitrarily.
  2. These vertices should be eliminated in that order.

Other Heuristics

We also choose other useful heuristics for FPBB, based on experimental results.

- **Simplicial Vertex Rule**: A vertex v is simplicial if its neighbourhood induces a clique. If v is simplicial, we can eliminate it first.
- **Almost Simplicial Vertex Rule**: A vertex v is almost simplicial if the set of all but one of its neighbours induces a clique. If v of degree d is almost simplicial, and the lower bound on treewidth is at least d, we can eliminate it first.

Other heuristics

- **Backtrack Condition**: We maintain LB, the lower bound of the width of current elimination ordering during the search. If current intermediate graph has no more than $LB + 1$ vertices left, we will backtrack.
- **Maximum Clique**: For all cliques in the graph, there exists an elimination ordering with the vertices of that clique as the last vertices of the ordering, and the ordering has the width equals to the exact treewidth of the graph. So we find the maximum clique before the search, and fix them to be last vertices in the ordering.

Branch and Bound

Branch and bound search maintains a lower bound and an upper bound. If lower bound is *lb*, treewidth is at least *lb*. If upper bound is *ub*, treewidth is at most *ub*.
In the search space, when searching one particular node, FPBB will compute current upper bound and lower bound of Treewidth, and use it to cut off searching branches.
Benefits:

1. FPBB is an anytime algorithm. It will continue to give better upper bound before finding the exact answer.
2. Good bounds may greatly improve the running speed.
3. We may manually set the upper bound to be t_0, which is a comparatively small number, and begin the search. If the upper bound is updated after the search, we get the exact treewidth. Otherwise, we get a lower bound.

MFPBB

We use a modified version of FPBB (MFPBB) to find a good upper bound.

1. MFPBB expands only three most promising child nodes each time, which greatly reduces the search space.
2. We use minor-min-width heuristic [2] to compute a lower bound for each child node, and those have smaller lower bound are considered promising.
3. We pick the one with smaller degree to break the tie.
4. The total number of expanded nodes is limited.

Overview

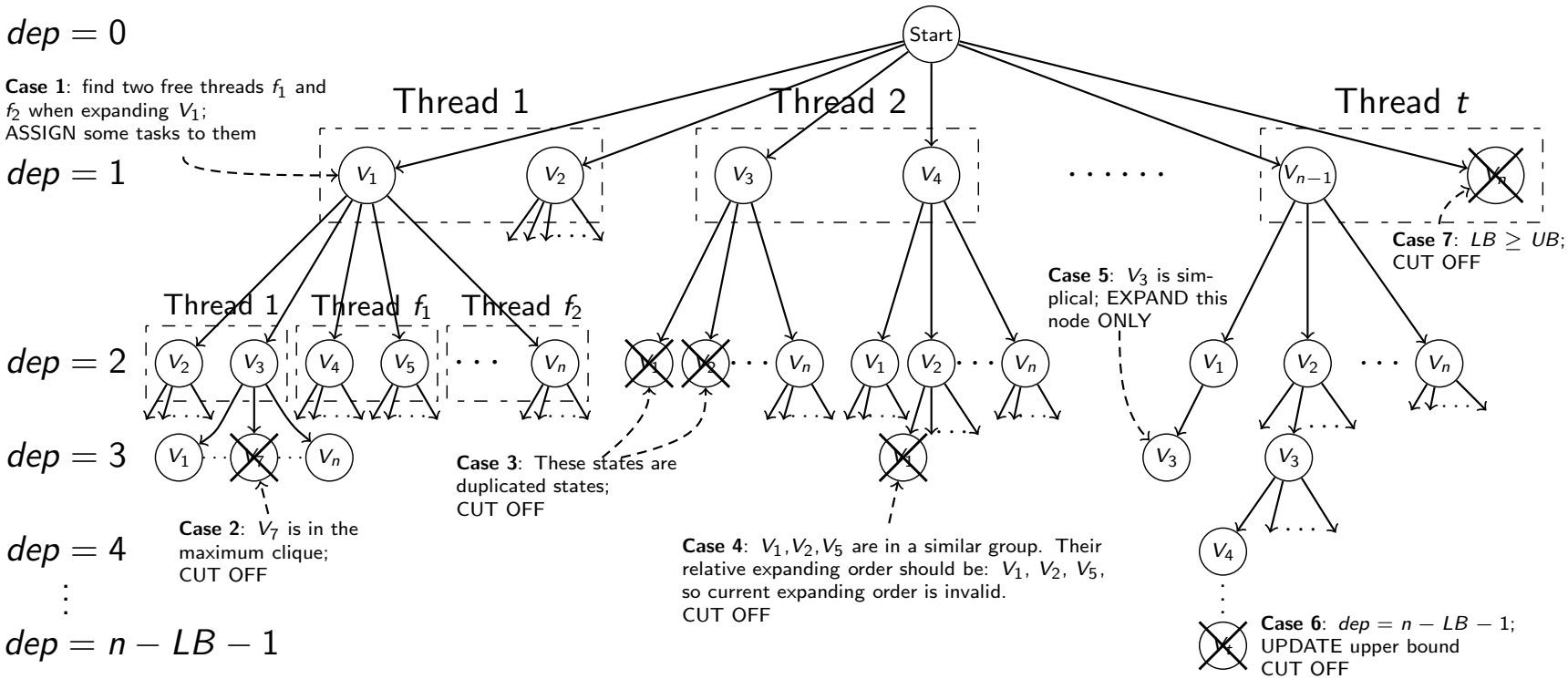


Figure: An Overview of FPBB

Settings

- Implemented in C++
- Intel Core i7-870 Processor
- 8 GB RAM
- Windows 7
- Four threads used

Benchmark Graphs

In this experiment, we use benchmark graphs from TreewidthLIB. FPBB has solved 17 benchmark graphs whose exact treewidths were previously unknown, and improved 12 known bounds of the others. Some results:

Name	N	UB	LB	Exp	Sec
1dj7	73	**26**	26	1255	0.02
1c9o	66	**28**	28	907015	3.84
miles750	128	36	**36**	5703212	22.59
1qtn	86	**23**	23	10181450	72.67
queen9_9*	81	58	**53**	250602432	758.95
1cc8	70	32	**29**	350271892	2854.10
1fse	67	**26**	25	282549	1.56
		26	**26**	108955514	706.78

Table: Improved Bounds of Benchmark Graphs

Random Graphs

In this experiment, we run FPBB on random graphs generated in G(n,m) Model.

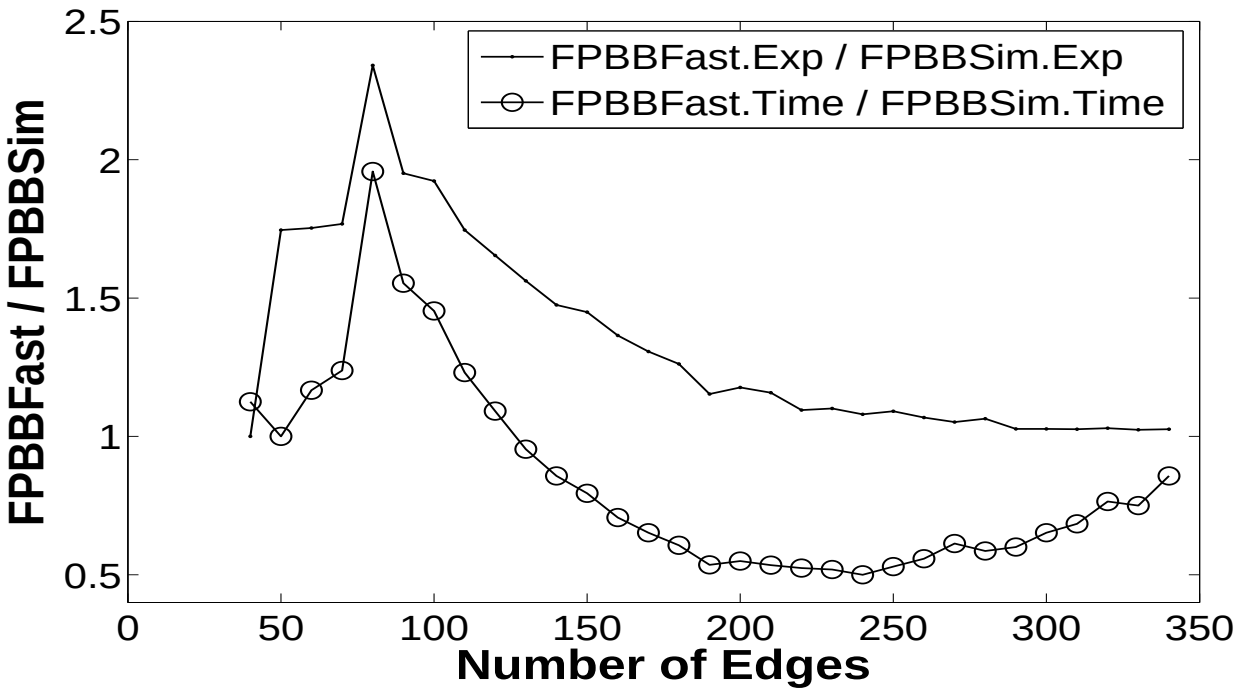


Figure: A Comparison of Expanded Nodes and Runtimes Between FPBBSim and FPBBFast. $n = 35$.

Random Graphs

In this experiment, we compare COMB, FPBB and FPBB with single thread implementation on 300 random graphs.

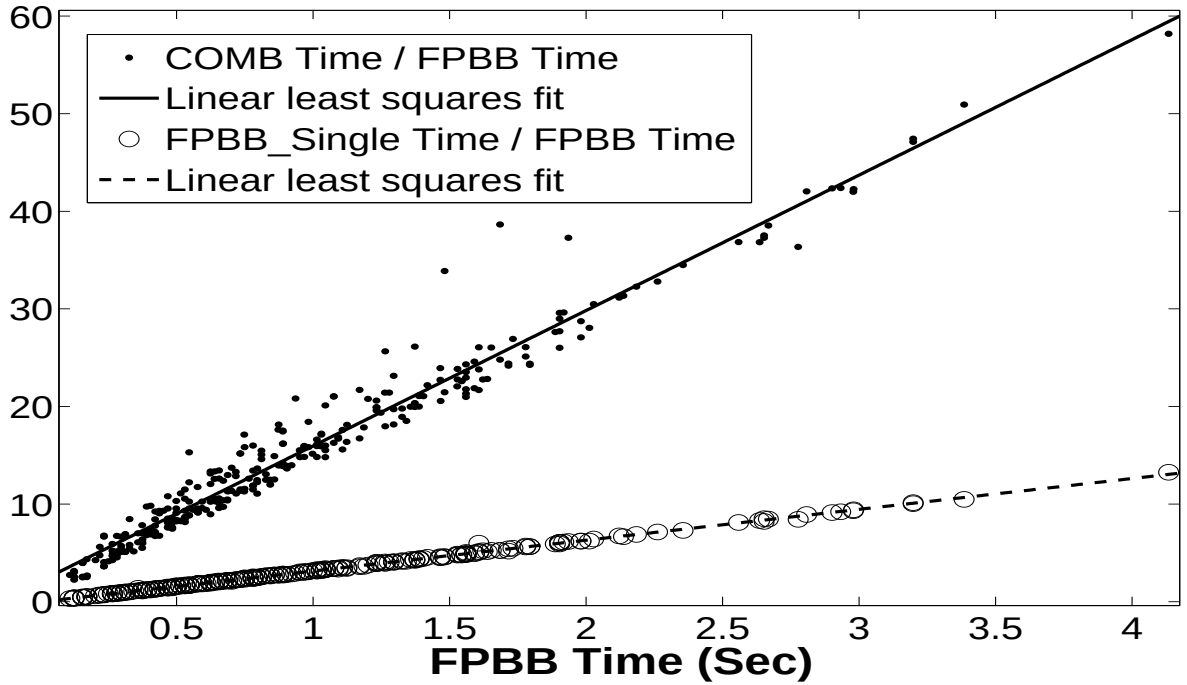


Figure: Ratio of COMB and FPBB Single Runtime to FPBB Runtime on 300 Random Graphs. $n = 35, m = 140$.

DIMACS Graphs

In this experiment, we compare FPBB with COMB using DIMACS vertex coloring graphs.

Name	TW	UB	FPBB Alo	FPBB LUT	FPBB Exp	FPBB Sec	COMB Exp	COMB Sec
queen6_6*	25	25	7	0.03	11187	0.03	11185	0.39
miles500	22	22	0	0.03	2	0.03	2	0.02
inithx.i.1	56	56	3	0.09	1392	0.09	1305	15.07
queen7_7*	35	35	18	0.02	533999	0.67	529242	26.96
myciel5	19	19	18	0.03	3482899	4.71	3351675	73.20
miles750	**36**	38	24	1.39	5703212	22.59	-	-
queen8_8*	45	46	24	16.10	18345352	33.73	-	-

Table: A Comparison Between FPBB and COMB

Conclusion

Treewidth is an important problem. We designed FPBB for computing exact Treewidth. FPBB has the following features:

1. It utilizes Branch and Bound search, which makes both lower bounds and upper bounds easier and faster to be computed.
2. It uses multithreading techniques. The more cores you have, the faster you will be.
3. It chooses many time-efficient heuristics, including Similar Group, Simplicial Vertex Rule, etc.
4. It is fast (about 40 times faster than COMB), and has solved many benchmarks.

Get FPBB from `www.callowbird.com`. Feel free to use it! :-)

References

1. D. Lokshtanov, D. Marx, and S. Saurabh. Known algorithms on graphs of bounded treewidth are probably optimal. CoRR, 2010.
2. V. Gogate and R. Dechter. A complete anytime algorithm for treewidth. In Proceedings of the 20th Conference in Uncertainty in Artificial Intelligence, UAI 2004, pages 201C208, 2004.
3. P. A. Dow and R. E. Korf. Best-first search for treewidth. In AAAI 2007, pages 1146C1151, 2007.
4. R. Zhou and E. A. Hansen. Combining breadth-first and depth-first strategies in searching for treewidth. In Proceedings of the 21st International Joint Conference on Artificial Intelligence, IJCAI 2009, pages 640C645, 2009.
5. H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. On exact algorithms for treewidth. In Algorithms - ESA 2006, 14th Annual European Symposium, Proceedings, volume 4168 of Lecture Notes in Computer Science, pages 672C683. Springer, 2006.

That's all!

Any questions?