Post-Processing Elimination Orderings to Reduce Induced Width

Andrew Gelfand

December 11, 2009

Final Project CS 276

Prof. Dechter

1 Introduction

The induced width along an elimination ordering is an important factor in the space and time complexity of many inference algorithms for graphical models. Indeed, slight changes in induced width can sometimes dictate whether a particular problem is feasible (i.e. will fit in memory) using variable elimination methods. For this reason, generating low width elimination orders has received extensive attention in the AI community and many heuristic ordering methods have been proposed (see e.g. [1, 2] for a nice survey). An extensive case study of several different heuristics was performed by Fishelson and Geiger [3]. Their results indicated that overall the Min-Fill and Weighted-Min-Fill heuristics outperformed other greedy ordering methods, including a stochastic greedy algorithm and max cardinality search. In this report, we build off of the results of Fishelson and others and ask a simple question: can post-processing be used to improve upon an already minimal induced width ordering? In particular, can we utilize the greedy edge removal method introduced in [4] to: 1) Find a minimal filled graph given the chordal graph resulting from a particular ordering; and 2) Find a better ordering given the reduced (minimal) filled graph?

Choosing a low width elimination ordering is just one of several graph theory problems that involve creating a chordal supergraph from a given graph [5]. In the minimal ordering problem, the objective is to add edges (fill-in) the graph such that the largest clique in the graph is as small as possible. In a related problem, the goal is simply to add as few edges as possible to the graph en route to making it chordal. This related problem is known as determining minimal fill. Both finding a minim*um* ordering and determining a minim*um* fill are NP-hard problems [6, 7]. Yet, as with the elimination ordering challenge, many different algorithms have been proposed to find good (minim*al*) fill. Perhaps the best known of these algorithms is Lex-M, which is a breadth first search algorithm that uses a lexicographic labeling scheme to identify an elimination ordering that introduces as few fill edges as possible [8].

The remainder of this report is structured as follows. In Section 2, some necessary definitions and background are provided, including a key theorem and upon which Blair et al.'s post-processing algorithm is based. Section 3 provides a brief description of Blair et al.'s algorithm (referred to herein as MinChordal) and illustrative example. Section 4 provides some experimental results from running the post-processing algorithm on a set of UAI benchmarks. Finally, Section 5 contains some concluding remarks.

2 Background

A graph G contains a vertex set $V(G)$ and an edge set $E(G)$. In this report, attention is restricted to undirected graphs. The neighbors of a vertex v in graph G is denoted as $N_G(v)$. An elimination ordering for a graph with n vertices is denoted $\alpha = v_1, ..., v_n$, where v_i denotes the vertex in the i^{th} position - i.e. $\alpha(v) = i$. The elimination of a vertex v from graph G results in creation of a supergraph of G. Associated with α is a sequence of supergraphs $G_0 \subseteq G_1 \cdots \subseteq G_n$. In particular, the graph G_i is obtained by adding edges to graph G_{i-1}, so that an edge exists between all non-adjacent neighbors of v_i - i.e. $N_{G_{i-1}}(v_i) \cap \{v_{i+1}, ..., v_n\}$. Unlike, in traditional variable elimination practices, the vertex v_i is not removed (eliminated) from the graph. Instead, the graph G_n resulting after elimination of the n^{th} vertex contains all n vertices. The set of fill edges added by elimination of v_i is given as $F_i = E(G_i)\backslash E(G_{i-1})$ and the clique induced by elimination of v_i is denoted as C_i (Note that C_i consists of v_i and only its higher numbered neighbors. This distinction is needed because variables are not actually eliminated from G). Finally, the filled graph, or chordal graph resulting from elimination of all vertices is denoted as $(G;\alpha)$. This is done to distinguish between the filled graphs ensuing from alternative orderings, such as $(G;\beta)$.

The elimination order α on graph G induces a sequences of cliques $C_1, ..., C_n$. The induced width along ordering α is denoted $w*(\alpha, G)$ and is formally defined as:

$$w*(\alpha, G) \stackrel{def}{=} \max_{i=1}^{n} |C_i| - 1$$

In finding a minimum elimination ordering, the goal is to find an $\alpha*$ that is a minimum across all possible orderings - i.e. $w*(\alpha*, G) \leq w*(\alpha, G)$ for all possible α. The width from the minim*um* ordering is referred to as the treewidth of the graph. Since finding the minimum ordering is NP-hard, we instead strive to find a minimal ordering.

In a similar fashion, the minimum fill problem can be formulated as follows. Letting $E(G)$ denote the set of edges in the original graph and $E(G;\alpha)$ denote the edges in the graph filled in along the ordering α, the goal is to find an ordering $\alpha*$ such that $|E(G;\alpha*) - E(G)| \leq |E(G;\alpha) - E(G)|$ for all possible orderings. The following theorem from [8] provides a useful characterization of minimal fill orderings:

> α is a minimal elimination ordering (in the minimal fill sense) if and only if each fill edge is the unique chord of a 4-cycle in $(G;\alpha)$.

In other words, if each edge added to G in creating the filled graph $(G;\alpha)$ is a unique chord, then there is no strict subgraph of $(G;\alpha)$ that is a chordal graph of G. For a chordal supergraph $(G;\alpha)$ of G, a fill edge $(E((G;\alpha))\backslash E(G))$ is a *candidate* for removal if it is not the unique chord of any 4-cycle in $(G;\alpha)$. So, if $(G;\alpha)$ is not minimal, it contains candidate edges and these edges can be removed without destroying the chordality of the graph. This idea forms the basis of the algorithm presented in the next section - namely, greedily removing candidate fill edges from the graph to arrive at a reduced graph and then finding a (perfect) elimination ordering on the reduced graph.

3 The MinChordal Algorithm

The MinChordal algorithm proposed by [4] takes as input: 1) A graph G; and 2) An elimination ordering α. After computing the filled graph $(G;\alpha)$ along $\alpha$and recording the cliques and fill edges (C_i's and F_i's) associated with the elimination of each variable, the algorithm moves backwards along the ordering α. Starting with the final vertex in the elimination ordering v_n and moving down to the first vertex in the ordering, the algorithm looks for candidate edges among the fill edges F_i introduced by the elimination of vertex v_i. If any of the edges in F_i are candidates, the algorithm identifies a graph W_i which is a subgraph of C_i in which all candidate and non-incident edges have been removed. The original Lex-M algorithm of [8] is then run to determine the minimal fill needed to make W_i chordal and any non-unique edges in W_i are removed from the graph $(G;\alpha)$. A detailed description of these algorithms can be found in the Appendix.

To better illustrate the algorithm, consider the simple example played out in the following figures.

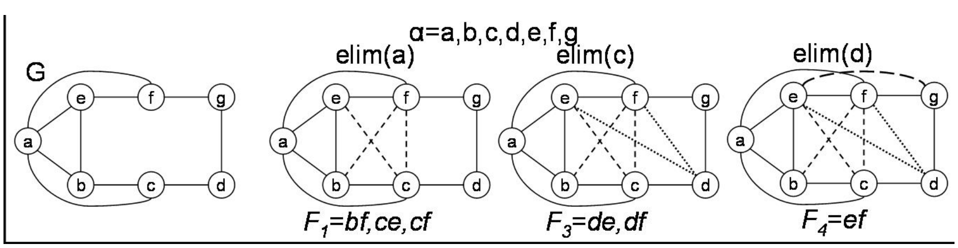

Figure 1: Filling in G along α

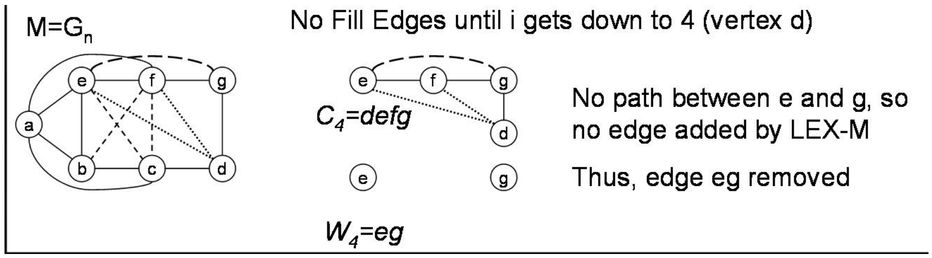

Figure 2: Checking v_4 - the first node encountered with fill edges

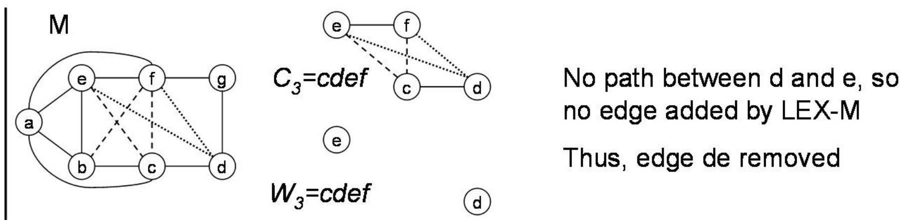

Figure 3: Checking v_3 - Lex-M returns no edge, so candidate de removed.

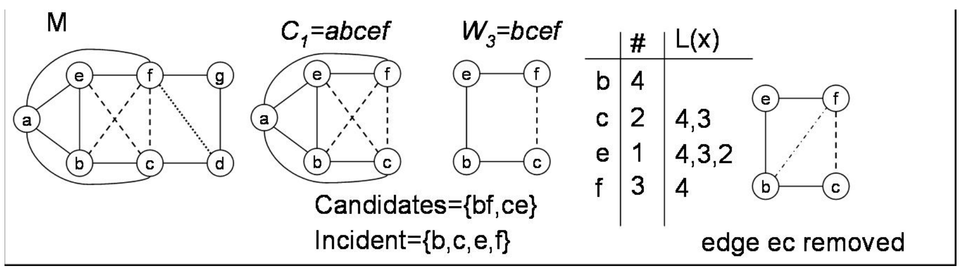

Figure 4: Checking v_1- Lex-M returns only one of edges bf, ce. The other is redundant and can be removed.

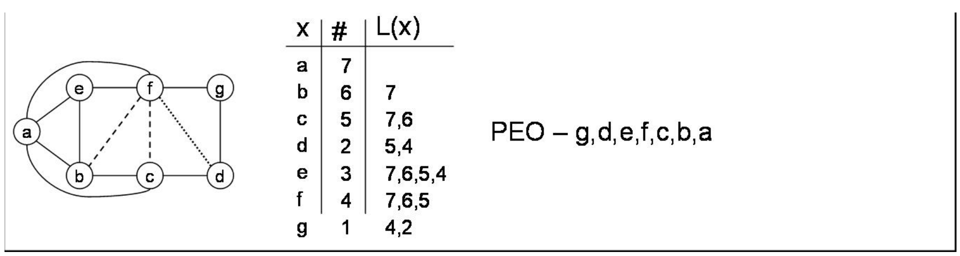

Figure 5: Finding the Perfect Elimination Ordering (PEO) given the minimal chordal graph

4 Results

The MinChordal algorithm was implemented in C as an extension to the toolbar package (see [10]). The toolbar package includes implementations of several ordering heuristics including: 1) Max Cardinality; 2) Min-Fill; 3) Min-Width; and 4) Min-Induced-Width. The MinChordal algorithm was developed in two variants - Min-Chordal-RAND and Min-Chordal-MF. Min-Chordal-RAND uses an arbitrary (random) elimination ordering as its starting point, while the Min-Chordal-MF variant uses the elimination ordering returned by Min-Fill as its initial ordering.

A set of experiments were conducted on several of the UAI 2008 benchmarks (see [9] for details). Each benchmark contains several network instances and each of the aforementioned ordering algorithms were run 10 times on each network in each benchmark. The minimum, maximum and average induced widths from the orderings produced by each algorithm were tabulated over the 10 trials. In order to get an accurate assessment of the improvement between Min-Fill and Min-Chordal-MF, in each trial on a network instance, the ordering produced by Min-Fill was written to file and read in as the initial ordering for the Min-Chordal-MF algorithm. Configuring the experiment in this manner removes the effects of any randomness in the ordering process, ensuring a fair comparison.

The results from several of the benchmarks can be seen in the tables below. The rows in each table correspond to specific network instances in each benchmark. Each column contains the min, max and average induced widths produced by the five ordering algorithms. Somewhat surprisingly, post-processing seems to have no effect on the induced widths yielded by the Min-Fill algorithm. As can be seen in the results from the bn20 benchmark in Figure 6, the Min-Chordal-MF algorithm did not offer a reduction in any of the widths of the Min-Fill heuristic. Similar results can be found for the Grids benchmark and the UCI Linkage benchmark. To get some measure of the utility of post processing (and to ensure that the algorithm was properly coded) the Min-Chordal algorithm was fed the ordering of the Min-Width heuristic on the Linkage benchmark. This set of runs, denoted as Min-Chordal-MW can be seen in Figure 7. In this case, the Min-Chordal algorithm is able to find an ordering that significantly reduces induced width, yielding widths that are comparable to Min-Fill.

	Max-Cardinality			Min-Width			Min-Induced-Width			Min-Fill			Min-Chordal-MF		
	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg
bn2o-30-15-150-1a	23	24	23	24	24	24	23	23	23	23	23	23	23	23	23
bn2o-30-15-150-1b	23	25	23	24	24	24	23	23	23	23	23	23	23	23	23
bn2o-30-15-150-2a	23	24	23	24	24	24	23	23	23	23	23	23	23	23	23
bn2o-30-15-150-2b	23	25	23	24	24	24	23	23	23	23	23	23	23	23	23
bn2o-30-15-150-3a	23	24	23	24	24	24	23	23	23	23	23	23	23	23	23
bn2o-30-15-150-3b	23	24	23	24	24	24	23	23	23	23	23	23	23	23	23
bn2o-30-20-200-1a	26	27	26	26	27	26	26	26	26	26	26	26	26	26	26
bn2o-30-20-200-1b	26	27	26	27	27	27	26	26	26	26	26	26	26	26	26
bn2o-30-20-200-2a	26	28	26	26	27	26	26	26	26	26	26	26	26	26	26
bn2o-30-20-200-2b	26	27	26	26	27	26	26	26	26	26	26	26	26	26	26
bn2o-30-20-200-3a	26	27	26	26	27	26	26	26	26	26	26	26	26	26	26
bn2o-30-20-200-3b	26	27	26	26	27	26	26	26	26	26	26	26	26	26	26
bn2o-30-25-250-1a	25	26	25	25	25	25	25	25	25	25	25	25	25	25	25
bn2o-30-25-250-1b	25	26	25	25	25	25	25	25	25	25	25	25	25	25	25
bn2o-30-25-250-2a	25	26	25	25	25	25	25	25	25	25	25	25	25	25	25
bn2o-30-25-250-2b	25	26	25	25	25	25	25	25	25	25	25	25	25	25	25
bn2o-30-25-250-3a	25	26	25	25	25	25	25	25	25	25	25	25	25	25	25
bn2o-30-25-250-3b	25	26	25	25	25	25	25	25	25	25	25	25	25	25	25

Figure 6: Results from the bn20 benchmark. A collection of 18 two-layer noisy-or Bayesian networks.

	Max-Cardinality			Min-Width			Min-Chordal-MW			Min-Induced-Width			Min-Fill			Min-Chordal-MF		
	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg
pedigree1	17	38	23	46	48	47	20	30	25	18	27	22	16	16	16	16	16	16
pedigree13	41	58	46	107	109	107	92	92	92	40	53	47	37	46	40	37	46	40
pedigree18	27	47	34	72	83	77	58	60	58	25	31	27	21	25	23	21	25	23
pedigree19	20	49	30	156	163	159	107	107	107	33	47	37	27	31	29	27	31	29
pedigree20	25	33	28	37	41	38	31	32	31	25	32	29	23	25	23	23	25	23
pedigree23	20	30	24	51	51	51	39	39	39	24	33	29	25	31	28	25	31	28
pedigree25	32	45	37	77	81	78	47	54	50	29	38	33	32	35	33	32	35	33
pedigree30	32	40	35	142	157	147	137	146	138	26	30	28	22	26	24	22	26	24
pedigree31	33	62	51	96	99	96	61	61	61	36	51	42	34	38	36	34	38	36
pedigree33	26	42	31	55	58	56	46	47	46	42	64	53	32	35	33	32	35	33
pedigree34	34	49	38	126	133	130	112	113	112	44	62	51	34	41	36	34	41	36
pedigree37	45	60	49	55	58	56	22	23	22	24	27	25	22	23	22	22	23	22
pedigree38	39	72	48	90	95	93	52	56	54	18	20	18	16	18	17	16	18	17
pedigree39	32	38	34	61	65	64	44	44	44	25	32	27	22	26	23	22	26	23
pedigree40	24	41	30	164	169	167	111	111	111	37	54	45	29	37	34	29	37	34
pedigree41	30	53	39	181	200	191	108	109	108	41	59	49	35	42	38	35	42	38
pedigree42	25	39	29	79	83	81	66	68	67	25	30	27	24	26	24	24	26	24
pedigree44	28	41	34	130	132	131	89	95	91	32	48	40	28	31	29	28	31	29
pedigree50	33	42	35	65	71	67	52	56	54	18	21	19	17	18	17	17	18	17
pedigree51	42	75	54	157	157	157	91	101	97	43	55	48	42	47	43	42	47	43
pedigree7	41	64	52	120	127	123	69	71	70	43	56	47	34	42	37	34	42	37
pedigree9	32	39	35	102	103	102	97	98	97	36	49	43	28	34	30	28	34	30

Figure 7: Results from the Linkage2 benchmark. While Min-Chordal post-processing yielded no reduction in width given orderings from Min-Fill, it did significantly reduce width given orderings from Min-Width.

	Max-Cardinality			Min-Width			Min-Induced-Width			Min-Fill			Min-Chordal-MF		
	min	max	avg	min	max	avg	min	max	avg	min	max	avg	min	max	avg
50-12-1	15	27	22	16	25	19	17	20	18	16	18	17	16	18	17
50-12-10	15	26	21	18	25	20	18	20	18	16	17	16	16	17	16
50-12-2.	14	27	22	16	25	20	17	21	18	16	17	16	16	17	16
50-12-3	16	25	22	18	25	21	17	20	18	16	18	16	16	18	16
50-12-4	16	26	22	18	21	19	17	20	18	16	18	17	16	18	17
50-12-5	14	25	20	19	26	20	17	20	18	16	18	17	16	18	17
50-12-6	15	26	21	16	24	20	18	20	18	16	18	17	16	18	17
50-12-7	15	26	20	18	24	20	17	21	19	16	19	17	16	19	17
50-12-8	16	27	22	17	25	20	17	19	18	16	18	16	16	18	16
50-12-9.	14	27	19	15	24	19	17	21	18	16	18	16	16	18	16
50-14-1.	17	31	25	24	29	27	20	26	21	20	22	20	20	22	20
50-14-10	17	31	24	26	31	28	20	23	21	20	23	20	20	23	20
50-14-2.	17	29	23	25	29	27	22	25	23	19	22	20	19	22	20
50-14-3	19	31	25	21	29	26	22	25	23	19	21	20	19	21	20
50-14-4	20	29	25	22	30	26	22	24	22	20	22	20	20	22	20
50-14-5	16	33	23	23	30	27	21	25	22	20	23	21	20	23	21
50-14-6.	16	30	24	23	30	27	21	26	22	19	22	20	19	22	20
50-14-7	16	29	20	26	32	28	21	24	22	19	23	21	19	23	21
50-14-8	17	31	26	26	32	28	19	25	22	19	22	20	19	22	20
50-14-9	16	32	25	25	29	27	21	27	23	20	22	20	20	22	20
50-15-1	18	35	26	26	36	29	22	29	24	21	24	22	21	24	22
50-15-10	18	33	26	26	37	32	22	28	24	21	25	22	21	25	22
50-15-2	17	35	27	26	36	29	22	27	24	21	25	22	21	25	22
50-15-3	18	34	25	26	36	31	24	26	24	21	24	22	21	24	22
50-15-4	17	35	24	25	36	29	23	29	24	21	24	22	21	24	22
50-15-5	19	33	26	27	36	30	23	26	24	21	23	22	21	23	22
50-15-6	17	32	22	25	36	30	22	25	23	21	24	22	21	24	22
50-15-7	18	32	26	26	37	30	23	30	25	21	24	22	21	24	22
50-15-8	17	32	25	23	35	30	23	26	24	21	24	22	21	24	22
50-15-9	17	35	24	27	37	31	22	27	25	21	24	21	21	24	21

Figure 8: Results from the Grids benchmark. A set of grid networks (12x12 to 50x50) with between 144 and 2,500 binary variables.

5 Conclusion

The issue of finding elimination orderings that yield minimal induced widths was examined in this report. It is well-believed in the AI community that Min-Fill is a superior ordering heuristic and indeed the experimental results in this report confirm that finding. However, in this report it was desired to extend the case study presented in [3] and determine whether some amount of post-processing can be used to improve upon an already low width ordering. In particular, the greedy edge removal method introduced in [4] was proposed as such a post-processing algorithm because it finds a minimal filled graph from a potentially non-minimal graph and also determines a new (perfect) elimination ordering given the reduced filled graph.

The results from using the Min-Chordal algorithm indicate that post-processing is able to reduce the induced width when given a low width ordering from the Min-Width heuristic (i.e. Min-Chordal-MW). However, the Min-Chordal algorithm was unable to yield any improvement in the orderings produced by Min-Fill. Since post-processing incurs additional computational overhead, this means that there is no gain in using the Min-Chordal algorithm for post-processing. It would be interesting an interesting follow on to determine if perhaps other post processing algorithms, such as those presented in [11] can be used to improve upon Min-Fill.

6 Appendix

```
Algorithm MinimalChordal (G; α);
Input: A graph G and an elimination ordering α = v1, ..., vn of G.
Output: 1. A chordal graph M which is both a minimal chordal supergraph of G
           and a subgraph of the filled graph (G; α).
        2. A minimal elimination order β of G s.t. M is the filled graph (G; β).
begin
    Find (G; α) and Ci, Fi for i = 1, 2, ..., n;
    M = (G; α);
    for i = n downto 1 do
        Candidate(i) = ∅;
        Incident(i) = ∅;
        for all edges uv ∈ Fi do
            if CandidateEdge(uv, i, M) then
                Candidate(i) = Candidate(i) ∪ {uv};
                Incident(i) = Incident(i) ∪ {u, v};
            end-if
        end-for
        if Candidate(i) ≠ ∅ then
            Wi = Ci[Incident(i)] \ Candidate(i);
            KeepFill(i) = LEX-M(Wi);
            M = M \ (Candidate(i) \ KeepFill(i));
        end-if
    end-for
    return M and β = LEX-P(M);
end

Function CandidateEdge(uv, i, M): boolean;
Input: An edge uv ∈ Fi and the graph M.
Output: Returns true if uv is a candidate to be removed from M, false o.w.
begin
    cand = true;
    for each neighbor x of u do
        if α(x) > i and xv ∈ E(M) and xvi ∉ E(M) then
            cand = false;
    end-for
    return cand;
end
```

Figure 9: The MinChordal Algorithm ([4])

Lex M (Rose, Tarjan, and Lueker [12])

input : $G = (V, E)$.

output : A minimal elimination ordering α and G_α^+.

begin

  $G_\alpha^+ = G$;

  for *all vertices* u *in* G **do**

    $L(u) = \emptyset$;

  for $i = n$ *to* 1 **do**

    Let v be one of the unnumbered vertices with largest label;

    $\alpha^{-1}(v) = i$;

    for *each unnumbered vertex* u *such that there exists a path* $u = x_0, x_1, ..., x_k = v$ *in* G, *where* x_j *is unnumbered and* $L(x_j) < L(u)$ *for* $0 < j < k$ **do**

      add i to $L(u)$;

      add fill edge (v, u) to G_α^+;

end

Figure 10: The Lex-M Algorithm ([8])

References

[1] A. Darwiche, Modeling and Reasoning with Bayesian Networks. Cambridge Press, New York, NY. 2009

[2] D. Koller & N. Friedman, Probabilistic Graphical Models: Principles and Techniques. MIT Press, Cambridge MA. 2009.

[3] M. Fishelson & D. Geiger, "Optimizing exact genetic linkage computations." In Proc. of 7th intl. conf. on research in computational molecular biology (RECOMB), 2003. pp:114-121.

[4] J. R. S. Blair, P. Heggernes, J.A. Telle, "A practical algorithm for making filled graphs minimal," Theoretical Comp. Sci., Vol. 250, Issues 1-2, 6 January 2001.

[5] Y. Villanger, "LEX M versus MCS-M," Discrete Mathematics 2006 306(3): 393-400.

[6] S. Arnborg, D. G. Corneil, A. Proskurowski, "Complexity of finding embeddings in a k-tree," SIAM J. Alg. Disc. Meth., 8:277–284, 1987.

[7] M. Yannakakis, "Computing the minimum fill-in is NP-complete," SIAM J. Alg. Disc. Meth., 2:77–79, 1981.

[8] D.J. Rose, R.E. Trajan, G.S. Lueker, "Algorithmic aspects of vertex elimination on graphs," SIAM J. Comp. 5 (1976) 266-283.

[9] http://graphmod.ics.uci.edu/uai08/Evaluation/Report/Benchmarks

[10] de Givry, S., Heras, F., Larrosa, J. & Schiex, T. Toolbar - A Constraint Optimization Toolbox. INRA, Biometry and AI Lab. Toulouse, France & UPC, Language and Computer Sciences Dpt. Barcelona, Spain.

[11] Fomin, F.V., Kratsch, D., Todinca, I. & Villanger, Y. "Exact Algorithms for Treewidth and Minimum Fill-In," SIAM J. Comp. (2008) Vol. 38, Issue 3, 1058-1079.