A Note on Exact Algorithms for Vertex Ordering Problems on Graphs

Hans L. Bodlaender

Fedor V. Fomin

Arie M.C.A. Koster

Dieter Kratsch

Dimitrios M. Thilikos

Technical Report UU-CS-2009-023
November 2009

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

A Note on Exact Algorithms for Vertex Ordering Problems on Graphs*

Hans L. Bodlaender† Fedor V. Fomin‡ Arie M.C.A. Koster§
Dieter Kratsch¶ Dimitrios M. Thilikos‖

Abstract

In this note, we give a proof that several vertex ordering problems can be solved in $O^*(2^n)$ time and $O^*(2^n)$ space, or in $O^*(4^n)$ time and polynomial space. The algorithms generalize algorithms for the Travelling Salesman Problem by Held and Karp [12] and Gurevich and Shelah [11]. We survey a number of vertex ordering problems to which the results apply.

1 Introduction

In this note, we look at exact algorithms with 'moderately exponential time' for graph problems. We show that with relatively simple adaptations of the existing algorithms for the Travelling Salesman Problem, a large collection of vertex ordering problems can be solved in $O^*(2^n)$ time and $O^*(2^n)$ space or in $O^*(4^n)$ time and polynomial space. (Here, the O^*-notation suppresses factors that are polynomial in n.) The algorithms that use $O^*(2^n)$ time and $O^*(2^n)$ space employ dynamic programming and have the same structure as the classical algorithm for TSP by Held and Karp [12]. The algorithms with $O^*(4^n)$

*This research was partially supported by the project *Treewidth and Combinatorial Optimization* with a grant from the Netherlands Organization for Scientific Research NWO and by the Research Council of Norway and by the DFG research group "Algorithms, Structure, Randomness" (Grant number GR 883/9-3, GR 883/9-4). The last author was supported by the project "Kapodistrias" (ΑΠ 02839/28.07.2008) of the National and Kapodistrian University of Athens (project code: 70/4/8757). Parts of this paper appeared earlier in the conclusions section of [2].

†Institute of Information and Computing Sciences, Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. hansb@cs.uu.nl

‡Department of Informatics, University of Bergen, N-5020 Bergen, Norway. fomin@ii.uib.no

§Lehrstuhl II für Mathematik, RWTH Aachen University, Wüllnerstr. zwischen 5 und 7, D-52062 Aachen, Germany. koster@math2.rwth-aachen.de

¶LITA, Université de Metz, F-507045 Metz Cedex 01, France. kratsch@sciences.univ-metz.fr

‖Department of Mathematics, National and Kapodistrian University of Athens, Panepistimioupolis, GR-15784, Athens, Greece. sedthilk@math.uoa.gr.

time and polynomial space are of a recursive nature and follow a technique first used for TSP by Gurevich and Shelah [11].

This paper is organized as follows. In Section 2, we give some preliminary definitions. A general theorem that gives for all problems of a specific form an algorithm of the Held-Karp type is given and proved in Section 3. A similar theorem with proof for Gurevich-Shelah type algorithms (i.e., with polynomial space) is given in Section 4. Then, in Section 5, we discuss a number of well known vertex ordering problems on graphs to which these theorems can be applied. A few final remarks are made in Section 6.

2 Definitions

We assume the reader to be familiar with standard notions from graph theory. Throughout this paper, $n = |V|$ denotes the number of vertices of graph $G = (V, E)$. For a graph $G = (V, E)$ and a set of vertices $W \subseteq V$, the subgraph of G induced by W is the graph $G[W] = (W, \{\{v, w\} \in E \mid v, w \in W\})$.

A *linear ordering* of a graph $G = (V, E)$ is a bijection $\pi : V \to \{1, 2, \ldots, |V|\}$. For a linear ordering π and $v \in V$, we denote by $\pi_{<,v}$ the set of vertices that appear before v in the ordering: $\pi_{<,v} = \{w \in V \mid \pi(w) < \pi(v)\}$. Likewise, we define $\pi_{\leq,v}$, $\pi_{>,v}$, and $\pi_{\geq,v}$.

Let $\Pi(S)$ be the set of all permutations of a set S. So, $\Pi(V)$ is the set of all linear orderings of G, and let for disjoint sets S and R, $\Pi(S, R)$ be the set of all permutations of $S \cup R$ which start with a permutation of S and end with a permutation of R.

A graph $G = (V, E)$ is *chordal*, if every cycle in G of length at least four has a chord, i.e., there is an edge connecting two non-consecutive vertices in the cycle. A *triangulation* of a graph $G = (V, E)$ is a graph $H = (V, F)$ that contains G as subgraph ($F \subseteq E$) and is chordal.

3 Exact Algorithms with Exponential Space

In this section, we show that a large collection of vertex ordering problems on graphs can be solved in $O^*(2^n)$ time and $O^*(2^n)$ space. The technique exploited here is dynamic programming in the style of the Held-Karp algorithms for the Travelling Salesman Problem [12].

Theorem 1 *Let f be a polynomial time computable function, mapping each 3-tuple, consisting of a graph $G = (V, E)$, a vertex set $S \subseteq V$, and a vertex $v \in V$ to an integer. Then we can compute in $O^*(2^n)$ time and $O^*(2^n)$ space the following values for a given graph $G = (V, E)$:*

$$\min_{\pi \in \Pi(V)} \max_{v \in V} f(G, \pi_{<,v}, v)$$

or

$$\min_{\pi \in \Pi(V)} \sum_{v \in V} f(G, \pi_{<,v}, v).$$

Note that values $f(G, S, v)$ do not depend on the ordering of S. The proof of the theorem follows the arguments of Held and Karp in [12] and an algorithm of this type for TREEWIDTH from [3].

Let f be as in the statement of Theorem 1. We first give the algorithm that uses $O^*(2^n)$ time and space to compute $\min_{\pi \in \Pi(V)} \max_{v \in V} f(G, \pi_{<,v}, v)$. Define

$$A_G(S) = \min_{\pi \in \Pi(S)} \max_{v \in S} f(G, \pi_{<,v}, v).$$

We set $A_G(\emptyset) = -\infty$. Note that $A_G(V)$ is the value to compute.

Lemma 2 *Let $G = (V, E)$ be a graph, and $S \subseteq V$. If $S \neq \emptyset$, then*

$$A_G(S) = \min_{w \in S} \max\{f(G, S, w), A_G(S - \{w\})\}$$

Proof: Suppose $A_G(S) = \max_{v \in S} f(G, \pi_{<,v}, v)$ for $\pi \in \Pi(S)$, then let w be the vertex on the last position of π. Now

$$\begin{aligned} A_G(S) &= \max_{v \in S} f(G, \pi_{<,v}, v) \\ &= \max\{f(G, \pi_{<w,w}, w), \max_{v \in S - \{w\}} f(G, \pi_{<,v}, v)\} \\ &= \max\{f(G, S, w), A_G(S - \{w\})\} \end{aligned}$$

This shows that

$$A_G(S) \leq \min_{w \in S} \max\{f(G, S, w), A_G(S - \{w\})\}$$

Suppose $\max\{f(G, S, w), A_G(S - \{w\})\}$ is minimal for $w \in S$, and

$$A_G(S - \{w\}) = \max_{v \in S - \{w\}} f(G, \pi'_{<,v}, v)$$

for a permutation $\pi' \in \Pi(S - \{w\})$. Let π be the permutation in $\Pi(S)$, that starts with π' and ends with w. Now,

$$\begin{aligned} A_G(S) &\geq \max_{v \in S} f(G, \pi_{<,v}, v) \\ &= \max\{f(G, \pi_{<,w}, w), \max_{v \in S - \{w\}} f(G, \pi_{<,v}, v)\} \\ &= \max\{f(G, S, w), \max_{v \in S - \{w\}} f(G, \pi'_{<,v}, v)\} \\ &= \max\{f(G, S, w), A_G(S - \{w\})\} \end{aligned}$$

This shows that

$$A_G(S) \geq \min_{w \in S} \max\{f(G, S, w), A_G(S - \{w\})\}$$

and thus completes the proof of this lemma. □

Algorithm 1 Dynamic-Programming-Algorithm(Graph $G = (V, E)$)

```
Set A(∅) = −∞.
for i = 1 to n do
  for all vertex sets S ⊂ V with |S| = i do
    Set A(S) = min_{w∈S} max{f(G, S, w), A(S − {w})}
  end for
end for
return A(V)
```

Lemma 2 directly gives us a method to compute $A_G(V)$ by dynamic programming: we compute all values $A_G(S)$ in order of increasing number of elements in S, using the formulas given by Lemma 2. We then output $A_G(V)$. Each single value can be computed in polynomial time; we need to store and compute 2^n values, thus the running time and the space are $O^*(2^n)$. See Algorithm 1.

The computation of $\min_{\pi \in \Pi(V)} \sum_{v \in V} f(G, \pi_{<,v}, v)$ is similar. We define

$$B_G(S) = \min_{\pi \in \Pi(S)} \sum_{v \in S} f(G, \pi_{<,v}, v)$$

Now, $B_G(\emptyset) = 0$, and, similar to Lemma 2, we have

$$B_G(S) = \min_{w \in S} f(G, S, w) + B_G(S - \{w\})$$

The remaining details are similar to the maximization case and left to the reader.

In a practical implementation, several improvements to the scheme of Algorithm 1 can be made; an algorithmic engineering study for Treewidth has been carried out, see [3].

4 Exact Algorithms with Polynomial Space

In this section, we give a variant of Theorem 1. This variant applies to the same collection of problems. In contrast with Theorem 1, Theorem 3 uses polynomial space but more (i.e. $O^*(4^n)$) time. It employs recursion instead of dynamic programming, and has the same structure as the algorithm for TSP by Gurevich and Shelah [11]. An algorithm of this type for Treewidth appears in [3].

Theorem 3 *Let f be a polynomial time computable function, mapping each 3-tuple, consisting of a graph $G = (V, E)$, a vertex set $S \subseteq V$, and a vertex $v \in V$ to an integer. Then we can compute in $O^*(4^n)$ time and polynomial space the following values for a given graph $G = (V, E)$:*

$$\min_{\pi \in \Pi(V)} \max_{v \in V} f(G, \pi_{<,v}, v)$$

or

$$\min_{\pi \in \Pi(V)} \sum_{v \in V} f(G, \pi_{<,v}, v)$$

Again, we concentrate on the computation of $\min_{\pi\in\Pi(V)} \max_{v\in V} f(G,\pi_{<,v},v)$, and leave the variant where we take instead the sum to the reader.

Define, for a graph $G=(V,E)$, sets of vertices $L,S\subseteq V$, $L\cap S=\emptyset$, $S\neq\emptyset$:

$$C_G(L,S)=\min_{\pi\in\Pi(L,S)} \max_{v\in S} f(G,\pi_{<,v},v)$$

Note that we want to compute the value $C_G(\emptyset,V)$.

Lemma 4 *Let $G=(V,E)$ be a graph, and $S\subseteq V$, $L\subseteq V$, $L\cap S=\emptyset$.*

1. *If $S=\{x\}$, then $C_G(S)=f(G,L,x)$.*
2. *Suppose $|S|\geq 2$ and $1\leq k<|S|$. Then*

$$C_G(L,S)=\min_{S'\subseteq S,|S'|=k} \max\{C_G(L,S'),C_G(L\cup S',S-S')\}$$

Proof: If $S=\{x\}$, then each $\pi\in\Pi(L,S)$ first has the vertices in L in some ordering and then x. So $\pi_{<,x}=L$, and hence $\max_{v\in S} f(G,\pi_{<,v},v)=f(G,L,x)$. Part (1) now directly follows.

Suppose now that $|S|\geq 2$. Consider $S'\subseteq S$ with $S'\neq\emptyset$. Let $\pi'\in\Pi(L,S')$ fulfill

$$C_G(L,S')=\max_{v\in S'} f(G,\pi'_{<,v},v)$$

and let $\pi''\in\Pi(L\cup S',S-S')$ fulfill

$$C_G(L\cup S',S-S')=\max_{v\in S-S'} f(G,\pi'_{<,v},v)$$

By definition, π' and π'' exist. Define now vertex ordering $\pi\in\Pi(L,S)$ as follows: first we start with the vertices in $L\cup S'$ in the same order as they appear in π', and then take the vertices in $S-S'$ in the same order as they appear in π''. I.e., we first have the vertices in L, then the vertices in S', and then the vertices in $S-S'$. For $v\in S'$, $L\subseteq\pi_{<,v}=\pi'_{<,v}$, and for $v\in S-S'$, $L\cup S'\subseteq\pi_{<,v}=\pi''_{<,v}$.

Now

$$\begin{aligned} C_G(L,S) &\leq \max_{v\in S} f(G,\pi_{<,v},v) \\ &= \max\{\max_{v\in S'} f(G,\pi'_{<,v},v), \max_{v\in S-S'} f(G,\pi''_{<,v},v)\} \\ &= \max\{C_G(L,S'),C_G(L\cup S',S-S')\} \end{aligned}$$

As this holds for each $S'\subseteq S$ with $S'\neq\emptyset$, we have

$$C_G(L,S)\leq\min_{S'\subseteq S,|S'|=k} \max\{C_G(L,S'),C_G(L\cup S',S-S')\}$$

For the other direction, let $\pi \in \Pi(L, S)$ fulfill

$$C_G(L, S) = \max_{v \in S} f(G, \pi_{<,v}, v)$$

Let S' be the set consisting of the k elements in S with minimum index in π, i.e., $|S'| = k$ and all elements in S' appear before all elements in $S - S'$ in π. We have that $\pi \in \Pi(L \cup S', S - S')$. Let $\pi' \in \Pi(L, S')$ be obtained from π by restricting π to $L \cup S'$. Now

$$\begin{aligned} C_G(L, S) &= \max_{v \in S} f(G, \pi_{<,v}, v) \\ &= \max\{\max_{v \in S'} f(G, \pi_{<,v}, v), \max_{v \in S - S'} f(G, \pi_{<,v}, v)\} \\ &= \max\{\max_{v \in S'} f(G, \pi'_{<,v}, v), \max_{v \in S - S'} f(G, \pi_{<,v}, v)\} \\ &\leq \max\{C_G(L, S'), C_G(L \cup S', S - S')\} \end{aligned}$$

This shows the result. □

Our algorithm uses recursion, each time employing Lemma 4 with $k = \lfloor |S|/2 \rfloor$. The algorithm is given in pseudo-code in Algorithm 2.

Algorithm 2 Recursive(Graph G, vertex set L, vertex set S)

```
if |S|=1 then
  Suppose S = {v}.
  return f(G, L, v)
end if
Set opt = ∞.
for all vertex sets S' ⊆ S, |S'| = ⌊|S|/2⌋ do
  Compute v1 = Recursive(G, L, S');
  Compute v2 = Recursive(G, L ∪ S', S − S');
  Set opt = min {opt, max {v1, v2}};
end for
return opt
```

Correctness of Algorithm 2 follows directly from Lemma 4. The running time can be estimated as follows. Let $T(k)$ be the number of recursive calls made when Recursive is called with the third argument S with $|S| = k$. Clearly, $T(1) = 1$. If $k > 1$, then for each of the $\binom{k}{\lfloor k/2 \rfloor}$ subsets of S of size $\lfloor k/2 \rfloor$, we have a recursive call with third parameter of size $\lfloor |S|/2 \rfloor$ and a recursive call with third parameter of size $\lceil |S|/2 \rceil$; and thus we use per subset S' two calls at this level of the recursion, and $T(\lfloor k/2 \rfloor) + T(\lceil k/2 \rceil)$ calls deeper in the recursion tree. So

$$T(k) \leq \binom{k}{\lfloor k/2 \rfloor} (T(\lfloor k/2 \rfloor) + T(\lceil k/2 \rceil) + 2)$$

It follows that $T(k) < 4^k$. As the time per recursive call is bounded by a polynomial in n, the total time is bounded by $O^*(4^n)$. In most cases, the dynamic programming algorithm

from Section 3 is more practical than the recursive algorithm, as already the enumeration over all subsets of size $n/2$ is very time consuming, except for very small values of n, but for such values, the space requirements for the $O^*(2^n)$ algorithm can be expected to be small enough for modern computers.

5 Linear Ordering Problems

There are several problems to which Theorems 1 and 3 can be applied. Several of these will be discussed below. A good overview paper, discussion several linear ordering problems is [5]. Relations between treewidth, pathwidth, and other parameters can be found in [1].

5.1 Treewidth

Treewidth is a well studied graph parameter. While treewidth is usually defined in terms of tree decompositions, it also has a characterization as a vertex ordering problem (see e.g., [1, 4]. Using this characterization, in [3] explicit proofs of algorithms as in Theorems 1 and 3 are given for Treewidth. Several improvements on these algorithms were made: using balanced separators and potential maximal cliques, a polynomial space algorithm using $O((2.9512^n)$ time was given in [3]. This was improved further with a clever method to list and count the number of potential maximal cliques to $O(2.6151^n)$ time by Fomin and Villanger [8]. Several papers give improved algorithms for Treewidth, if we allow exponential space. An algorithm with $O(1.9601^n)$ time was given in 2004 by Fomin et al. [6]. This was improved further in [16, 7, 8, 9]; the best running time is given by a recent paper by Fomin and Villanger, who solve Treewidth in $O(1.7347^n)$ time.

5.2 Minimum Fill-in

A problem, related to treewidth, is the Minimum Fill-In problem. Exact algorithms with exponential space for Minimum Fill-In were obtained by Fomin et al. [6], and later improved [16, 7, 8, 9]; the currently fastest algorithm uses $O(1.7347^n)$ time and space [9]. These algorithms use the same techniques as for Treewidth. The Minimum Fill-In problem has important applications in Gaussian elimination.

The *minimum fill-in* of a graph $G = (V, E)$ is the minimum over all triangulations $H = (V, E_H)$ of G of $|E_H - E|$, i.e., the minimum number of edges that, when added to G, make G chordal.

For a graph $G = (V, E)$, a linear ordering of its vertices $\pi \in \Pi(V)$, and a vertex $v \in V$, let

$$R_\pi(v) = |\{w \in V \mid \pi(w) > \pi(v) \wedge \text{ there is a path from } v \text{ to } w \text{ in } G[\pi_{\leq,v} \cup \{w\}]\}|$$

The following proposition can be shown in the same way as a similar result for Treewidth in [3].

Proposition 5 *Let $G = (V, E)$ be a graph, and k a non-negative integer. The minimum fill-in of G is at most k if and only if there is a linear ordering π of G, such that*

$$\sum_{v \in V} R_\pi(v) \leq k + |E|$$

While for Treewidth there are polynomial space algorithms that are faster than the $O^*(4^n)$ bound implied by Theorem 3, this remains open for Minimum Fill-In.

5.3 Pathwidth

The pathwidth of a graph is usually defined in terms of path decompositions, but it has several equivalent characterizations, see e.g., [1] for an overview. Kinnersley [13] showed that pathwidth can be defined as a vertex ordering problem. We use this characterization to obtain the exact algorithms.

Definition 6 *The* vertex separation number *of a linear ordering π of $G = (V, E)$ is*

$$\max_{v \in V} |\{w \in V \mid \exists x \in V : \{w, x\} \in E \wedge \pi(w) < \pi(v) \leq \pi(x)\}|$$

The vertex separation number *of a graph G is the minimum vertex separation number over all linear orderings of G.*

Theorem 7 (Kinnersley [13]) *The vertex separation number of a graph equals its pathwidth.*

We thus see that the Vertex Separation Number has the shape for which we can apply Theorems 1 and 3. Very recently, Suchan and Villanger [15] obtained a faster exact algorithm for Pathwidth, i.e., using $O(1.9657^n)$ time and exponential space. It is open if this can be used for a faster algorithm with polynomial space.

5.4 Minimum Interval Graph Completion

Another problem, related to Pathwidth, which can be solved with Theorems 1 and 3 is the Minimum Interval Graph Completion problem. The Minimum Interval Graph Completion problem is the following: given a graph $G = (V, E)$, what is the minimum size of a set of edges, that, when added to G, yield an interval graph. The problem is known to be equivalent to the Sum Cut problem and the Profile problem, see for example [5]. In the Sum Cut problem, we look for a linear ordering π which minimizes

$$\sum_{v \in V} |\{w \in V \mid \exists x \in V : \{w, x\} \in E \wedge \pi(w) < \pi(v) \leq \pi(x)\}|$$

5.5 Cutwidth and Variants

The *cutwidth* of a linear ordering π of a graph $G = (V, E)$ is

$$\max_{v \in V} |\{w, x\} \in E \wedge \pi(w) \leq \pi(v) < \pi(x)\}|$$

The *modified cutwidth* of a linear ordering π of a graph $G = (V, E)$ is

$$\max_{v \in V} |\{w, x\} \in E \wedge \pi(w) < \pi(v) < \pi(x)\}|$$

The cutwidth (modified cutwidth) of a graph is the minimum cutwidth (modified cutwidth) of a linear ordering of it. The parameters have variants for directed acyclic graphs. The cutwidth (modified cutwidth) of a directed acyclic graph $G = (V, A)$ is the minimum cutwidth (modified cutwidth) of a topological ordering of G; the latter are defined similar to the undirected counterparts. By setting $f(G, S, v)$ to a very high value when there is an arc $(v, w) \in A$ with $w \in S$, we can force that the minimum is taken at a topological sort, and thus fit the problem into the form of Theorems 1 and 3.

5.6 Optimal Linear Arrangement

The Optimal Linear Arrangement problem, of which the decision variant was proved NP-complete in [10], asks, given a graph $G = (V, E)$, for the minimum over all linear orderings π of $\sum_{\{v,w\} \in E} |\pi(v) - \pi(w)|$. The following simple lemma shows that we can write the problem again in the form where we can apply Theorems 1 and 3.

Lemma 8 *For each graph $G = (V, E)$, and for each linear ordering π of G,*

$$\sum_{\{v,w\} \in E} |\pi(v) - \pi(w)| = \sum_{v \in V} |\{\{x, y\} \in E \mid \pi(x) \leq \pi(v) < \pi(w)\}|$$

The directed variant, where we look for topological orderings π of $G = (V, A)$ with $\sum_{(v,w) \in A} (f(w) - f(v))$ can be handled in a similar way.

5.7 Directed Feedback Arc Set

The Directed Feedback Arc Set is the following: given a directed graph $G = (V, A)$, find a set of arcs $F \subseteq A$ with $|F|$ as small as possible, such that $(V, A - F)$ is acyclic, i.e., each cycle in G contains at least one arc in F. It is a variant of the well known Feedback Vertex Set and Directed Feedback Vertex Set problems (which look for a set of vertices that break all cycles). (The problem to find in an undirected graph a minimum size set of edges that breaks all cycles is trivial; its weighted variant is a reformulation of the polynomial time solvable Minimum Spanning Tree problem. The (Directed) Feedback Vertex Set problems are trivially solvable in $O^*(2^n)$ time with linear space, and thus we have to focus only to Directed Feedback Arc Set.) One can also look

to a weighted variant: each arc has a weight, and we look for a set of arcs that break all cycles of minimum total weight.

The following lemma shows that we can formulate (Weighted) Directed Feedback Arc Set in a shape such that Theorems 1 and 3 can be applied. Recall that a graph is acyclic, if and only if it has a topological ordering.

Lemma 9 *Let $G = (V, A)$ be a directed graph, and let $w : A \to \mathbf{N}$ be a function that assigns each arc a non-negative integer weight. Let $K \in \mathbf{N}$ be an integer. There exists a set of arcs $F \subseteq A$ with $(V, A - F)$ acyclic and $\sum_{a \in F} w(a) \leq K$, if and only if there is a linear ordering π of G, such that $\sum_{(x,y) \in A,\ \pi(x) > \pi(y)} w((x, y)) \leq K$.*

5.8 Summary

The following result summarizes the discussion in the paragraphs above.

Theorem 10 *Each of the following problems:* Treewidth, Minimum Fill-In, Pathwidth, Sum Cut, Minimum Interval Graph Completion, Cutwidth, Directed Cutwidth, Modified Cutwidth, Directed Modified Cutwidth, Optimal Linear Arrangement, Directed Optimal Linear Arrangement *and* Directed Feedback Arc Set

1. *can be solved in $O^*(2^n)$ time and $O^*(2^n)$ space.*
2. *can be solved in $O^*(4^n)$ time and polynomial space.*

In each case, the $O^*(2^n)$ algorithm resembles the classic Held-Karp algorithm for TSP [12], and the $O^*(4^n)$ its variant by Gurevich and Shelah [11]. Note that for Treewidth, Minimum Fill-In and Pathwidth faster algorithms with exponential space are known [9, 15], and for Treewidth a faster algorithm with polynomial space is known [6, 7].

6 Concluding Remarks

This note discusses simple exponential time algorithms for a collection of vertex layout problems. Recently, Koivisto and Parviainen [14] have exploited the ideas further, and show that a tradeoff between time and space can be made, i.e., they give a range of algorithms, running in $O(c^n)$ time and $O(s^n)$ space, for various values of c and s.

Computational experiments in [3] show that the $O^*(2^n)$ time algorithm for Treewidth is practical for small graphs, especially when one applies a few optimizations to the algorithm. A similar algorithm engineering study for other problems that we listed in Section 5 would be very interesting.

References

[1] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theor. Comp. Sc.*, 209:1–45, 1998.

[2] H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. On exact algorithms for treewidth. Technical Report UU-CS-2006-032, Department of Information and Computing Sciences, Utrecht University, Utrecht, the Netherlands, 2006.

[3] H. L. Bodlaender, F. V. Fomin, A. M. C. A. Koster, D. Kratsch, and D. M. Thilikos. On exact algorithms for treewidth. In Y. Azar and T. Erlebach, editors, *Proceedings of the 14th Annual European Symposium on Algorithms, ESA 2006*, pages 672–683. Springer Verlag, Lecture Notes in Computer Science, vol. 4168, 2006.

[4] F. Clautiaux, A. Moukrim, S. Négre, and J. Carlier. Heuristic and meta-heuristic methods for computing graph treewidth. *RAIRO Operations Research*, 38:13–26, 2004.

[5] J. Díaz, J. Petit, and M. Serna. A survey of graph layout problems. *ACM Computing Surveys*, 34:313–356, 2002.

[6] F. V. Fomin, D. Kratsch, and I. Todinca. Exact (exponential) algorithms for treewidth and minimum fill-in. In J. Díaz, J. Karhumäki, A. Lepistö, and D. Sanella, editors, *Proceedings of the 31st International Colloquium on Automata, Languages and Programming, ICALP 2004*, pages 568–580. Springer Verlag, Lecture Notes in Computer Science, vol. 3142, 2004.

[7] F. V. Fomin, D. Kratsch, I. Todinca, and Y. Villanger. Exact algorithms for treewidth and minimum fill-in. *SIAM J. Comput.*, 38:1058–1079, 2008.

[8] F. V. Fomin and Y. Villanger. Treewidth computation and extremal combinatorics. In L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukuewics, editors, *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, ICALP 2008, Part I*, pages 210–221. Springer Verlag, Lecture Notes in Computer Science, vol. 5125, 2008.

[9] F. V. Fomin and Y. Villanger. Finding induced subgraphs via minimal triangulations. Report, published on http://arxiv.org/abs/0909.5278, 2009.

[10] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theor. Comp. Sc.*, 1:237–267, 1976.

[11] Y. Gurevich and S. Shelah. Expected computation time for Hamiltonian path problem. *SIAM J. Comput.*, 16:486–502, 1987.

[12] M. Held and R. Karp. A dynamic programming approach to sequencing problems. *J. SIAM*, 10:196–210, 1962.

[13] N. G. Kinnersley. The vertex separation number of a graph equals its path width. *Information Processing Letters*, 42:345–350, 1992.

[14] M. Koivisto and P. Parviainen. A space-time tradeoff for permutation problems. To appear in Proceedings SODA 2010, 2010.

[15] K. Suchan and Y. Villanger. Computing pathwidth faster than 2^n. To appear in Proceedings IWPEC 2009, 2009.

[16] Y. Villanger. Improved exponential-time algorithms for treewidth and minimum fill-in. In J. R. Correa, A. Hevia, and M. A. Kiwi, editors, *Proceedings of the 7th Latin American Symposium on Theoretical Informatics, LATIN 2006*, pages 800–811. Springer Verlag, Lecture Notes in Computer Science, vol. 3887, 2006.