ELSEVIER

Information Processing Letters 57 (1996) 197–203

Information Processing Letters

A simple linear-time algorithm for finding path-decompositions of small width

Kevin Cattell *, Michael J. Dinneen, Michael R. Fellows

Department of Computer Science, University of Victoria, Victoria, BC, Canada V8W 3P6

Received 9 January 1995; revised 17 November 1995
Communicated by F. Dehne

Abstract

We described a simple algorithm running in linear time for each fixed constant, k, that either establishes that the pathwidth of a graph G is greater than k, or finds a path-decomposition of G of width at most $O(2^k)$. This provides a simple proof of the result by Bodlaender that many families of graphs of bounded pathwidth can be recognized in linear time.

Keywords: Combinatorial problems; Algorithms; Analysis of algorithms

1. Introduction

The topics of the *pathwidth* and *treewidth* of graphs have proven to be of fundamental interest for two reasons. First of all, they play an important role in the deep results of Robertson and Seymour [20–22,24]. Secondly, and more importantly from a practical point of view, bounded pathwidth and treedwidth have proven to be general "common denominators" for many natural input restrictions of NP-complete problems. For many important problems, we now know that fixing a natural parameter k implies that the yes-instances have bounded treewidth or pathwidth (for examples see [5,15,18]). We also know that many problems can be solved in linear time when the input includes a bounded-width path-decomposition (or tree-decomposition) of the graph (see [1,2,4,12,25] and [6] for many further references).

After several rounds of improvement [23,17,19] the best known algorithm, for finding tree-decompositions is due to Bodlaender [7]. For each fixed k, this algorithm in time $O(2^{k^2}n)$ either determines that the treewidth is greater than k, or produces a tree-decomposition of width at most k. By first running this algorithm and then applying the algorithm of [8,16], a similar result holds for pathwidths. Both of the algorithms involved are quite complicated.

We describe here a very simple algorithm based on "pebbling" the graph using a pool of $O(2^k)$ pebbles, that in linear time (for fixed k), either determines that the pathwidth of a graph is more than k, or finds a path-decomposition of width at most the number of pebbles actually used. The main advantages of this algorithm over previous

* Corresponding author.

results are: (1) the simplicity of the algorithm and (2) the improvement of the hidden constant for a determination that the pathwidth is greater than k. The main disadvantage is in the width of the resulting "approximate" decomposition when the width is less than or equal to k.

2. Preliminaries

All of our discussion concerns finite simple graphs. Some of the graphs have a *boundary* of size k, meaning that they have a distinguished set of vertices labeled $1, 2, \ldots, k$. Two *k-boundaried graphs* (each with a boundary of size k) can be glued with the $\oplus$ operator, which simply identifies vertices with the same boundary label. A boundary size *k factorization* of a graph G is two k-boundaried graphs A and B such that $G = A \oplus B$.

Example. The $\oplus$ operator on two 3-boundaried graphs A and B is illustrated in Fig. 1. Note that common boundary edges (in this case, the edges between boundary vertices 1 and 2) are replaced with a single edge in $G = A \oplus B$.

A *homeomorphic embedding* of a graph $G_1 = (V_1, E_1)$ in a graph $G_2 = (V_2, E_2)$ is an injection from vertices, V_1 to V_2 with the property that the edges E_1 are mapped to disjoint paths of G_2. (These disjoint paths in G_2 represent possible *subdivisions* of the edges of G_1.) The set of homeomorphic embeddings between graphs gives a partial order, called the *topological order*.

A *lower ideal* $\mathcal{J}$ in a partial order $(\mathcal{U}, \geqslant)$ is a subset of $\mathcal{U}$ such that if $X \in \mathcal{J}$ and $X \geqslant Y$ then $Y \in \mathcal{J}$. The *obstruction set* for $\mathcal{J}$ is the set of minimal elements of $\mathcal{U} - \mathcal{J}$.

Definition. A *path-decomposition* of a graph $G = (V, E)$ is a sequence $X_1, X_2, \ldots, X_r$ of subsets of V that satisfy the following three conditions:

(1) $\bigcup_{1 \leqslant i \leqslant r} X_i = V$,

(2) for every edge $(u, v) \in E$, there exists an X_i such that $u \in X_i$ and $v \in X_i$, and

(3) for $1 \leqslant i < j < k \leqslant r$, $X_i \cap X_k \subseteq X_j$.

The *pathwidth of a path-decomposition* $X_1, X_2, \ldots, X_r$ is $\max_{1 \leqslant i \leqslant r} |X_i| - 1$. The *pathwidth of a graph G* is the minimum pathwidth over all path-decompositions of G. Determining pathwidth is equivalent to several VLSI layout problems such as *gate matrix layout* and *vertex separation* [18,14].

It is easy to see that the family of graphs of pathwidth at most t is a lower ideal in the topological (and minor) order. It is also known that those graphs with order n have at most $nt - (t^2 + t)/2$ edges.

Let B_h denote the complete binary tree of height h and order $2^h - 1$. Let $h(t)$ be the least value of h such that $B_{h(t)}$ has pathwidth greater than t_1 and let $f(t)$ be the number of vertices of $B_{h(t)}$. To find a bound for $f(t)$, $B_{h(t)}$ needs to contain (above in the topological order) at least one obstruction of pathwidth t. In [14] it shown that all topological tree obstructions of pathwidth t can be recursively generated by the following rules.

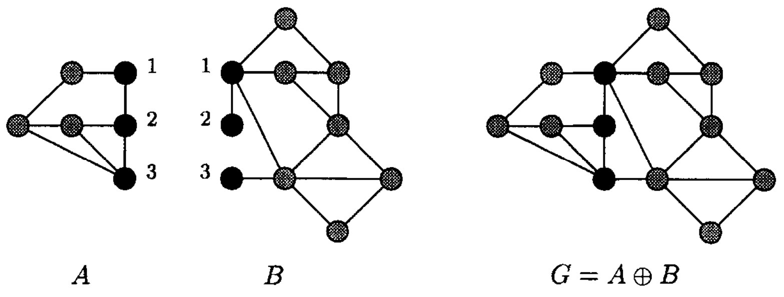


Fig. 1.

(1) The single edge tree K_2 is the only obstruction of pathwidth 0.
(2) If T_1, T_2 and T_3 are any 3 tree obstructions for pathwidth t then the tree T consisting of a new degree 3 vertex attached to any vertex of T_1, T_2 and T_3 is a tree obstruction for pathwidth $t+1$.

From this characterization we see that the orders the tree obstructions of pathwidth t are precisely $(5\cdot 3^t - 1)/2$, (e.g., orders 2, 7, 22 and 57 for pathwidth $t = 0$, 1, 2, and 3). We can easily embed at least one of the tree obstructions for pathwidth t, as shown in Fig. 2, in the complete binary tree of height $2t+2$. Thus, the complete binary tree of order $f(t) = 2^{2t+2} - 1$ has pathwidth greater than t.

3. Pathwidth algorithm

Using the $f(t)$ bound given in the previous section, the main result of the paper now follows:

Theorem 1. *Let H be an arbitrary undirected graph, and let t be a positive integer. One of the following two statements must hold*:
(a) *The pathwidth of H is at most* $f(t) - 1$.
(b) *H can be factored*: $H = A \oplus B$, *where A and B are boundaried graphs with boundary size* $f(t)$, *the pathwidth of A is greater than t and less than* $f(t)$.

Proof. We describe a algorithm that terminates either with a path-decomposition of H of width at most $f(t) - 1$, or with a path-decomposition of a suitable factor A with the last vertex set of the decomposition consisting of the boundary vertices.

If we find an homeomorphic embedding of the *guest tree* $B_{h(t)}$ in the *host graph* H then we know that the pathwidth of H is greater than t. During the search for such an embedding, we work with a partial embedding. We refer to the vertices of $B_{h(t)}$ as *tokens*, and call tokens *placed* or *unplaced* according to whether or not they are mapped to vertices of H in the current partial embedding. A vertex v of H is *tokened* if a token maps to v. At most one token can be placed on a vertex of H at any given time. We recursively label the tokens by the following standard rules:
(1) The root token of $B_{H(t)}$ is labeled by the empty string λ.
(2) The left child token and right child token of a height h parent token $P = b_1 b_2 \cdots b_h$ are labeled $P\cdot 1$ and $P\cdot 0$, respectively.

Let $P[i]$ denote the set of vertices of H that are tokened at time step i. The sequence $P[0], P[1], \ldots, P[s]$ will describe a path-decomposition either of the entirely of H or of a factor A fulfilling the conditions of Theorem 1. In the case of outcome (b) the boundary of the factor A is indicated by $P[s]$.

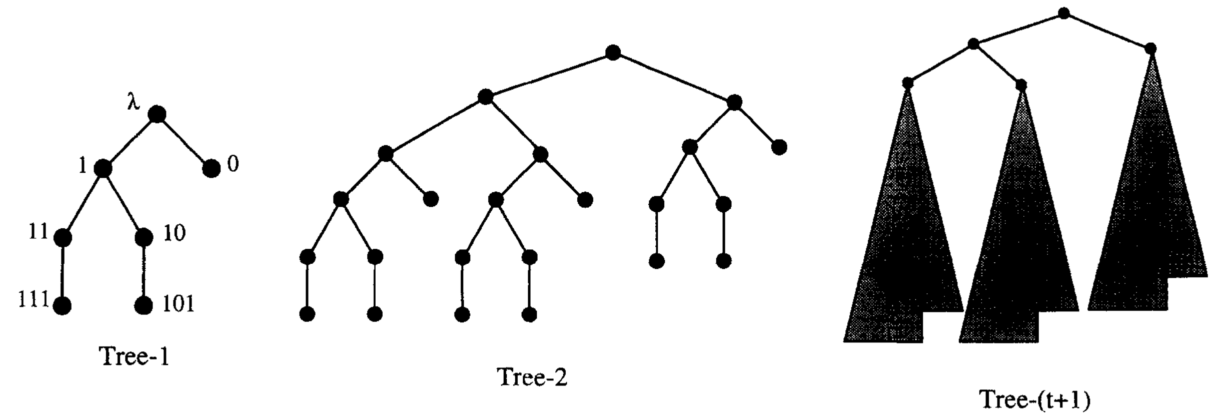


Fig. 2. Embedding pathwidth t tree obstructions Tree-t in binary trees.

The placement algorithm is described as follows. Initially consider that every vertex of H is colored *blue*. In the course of the algorithm a vertex of H has its color changed to *red* when a token is placed on it, and stays red if the token is removed. Only blue vertices can be tokened, and so a vertex can only receive a token once.

```
  **function** GrowTokenTree
1 **if** root token $\lambda$ is not placed of $H$ **then**
     arbitrary place $\lambda$ on a blue vertex of $H$
  **endif**
2 **while** there is a vertex $u \in H$ with token $T$ and
     blue neighbor $v$, and token $T$ has an unplaced child $T \cdot b$ **do**
  2.1 place token $T \cdot b$ on $v$
  **endwhile**
3 **return** {tokened vertices of $H$}
```

```
  **program** PathDecompositionOrSmallFatFactor
1 $i \leftarrow 0$
2 $P[i] \leftarrow$ **call** GrowTokenTree
3 **until** $|P[i]| = f(t)$ or $H$ has no blue vertices
     **repeat**
  3.1 pick a token $T$ with an unplaced child token
  3.2 remove $T$ from $H$
  3.3 **if** $T$ had one tokened child **then**
        replace all tokens $T \cdot b \cdot S$ with $T \cdot S$
      **endif**
  3.4 $i \leftarrow i + 1$
  3.5 $P[i] \leftarrow$ **call** GrowTokenTree
  **enduntil**
  **done**
```

Before we prove the correctness of the algorithm, we note some properties: (1) the root token will need to be placed (step 1 of the GrowTokenTree) at most once for each component of H; (2) the GrowTokenTree function only returns when $B_{h(t)}$ has been embedded in H or all parent tokens with unplaced children have no blue neighbors in the underlying host H; (3) the algorithm will terminate since during each iteration of step 3.2 a tokened red vertex becomes untokened, and this can happen at most n times, where n denotes the order of the host H.

Since tokens are placed only on blue vertices and are removed only from red vertices, it follows that the interpolation property of a path-decomposition is satisfied. Suppose the algorithm terminates at times s with all of the vertices colored red. To see that the sequence of vertex sets $P[0], \dots, P[s]$ represents a path-decomposition of H, it remains only to verify that for each edge (u, v) of H there is a time i is with both vertices u and v in $P[i]$. Suppose vertex u is tokened first and untokened before v is tokened. But vertex u can be untokened only if all neighbors, including vertex v, are colored red (see step 3.1 and comment (2) above).

Suppose the algorithm terminates with all tokens placed. The argument above establishes that the subgraph A of H induced by the red vertices, with boundary set $P[s]$ has pathwidth at most $f(t)$. To complete the proof we argue that in this case the sequence of token placements established that A contains a subdivision of $B_{h(t)}$, and hence must have pathwidth greater than t. Since the GrowTokenTree function only attaches pendant tokens to parent tokens we need to only to observe that the operation in step 3.3 subdivides the edge between T and its parent. □

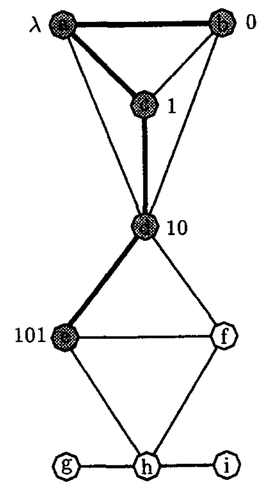
Corollary 2. *Given a graph H of order n and an integer t, there exists a linear-time algorithm that gives evidence that the pathwidth of H is greater than t or finds a path-decomposition of width at most* $O(2^t)$.

Proof. We show that program PathDecompositionOrSmallFatFactor runs in linear time. First, if H has more than $t \cdot n$ edges, then the pathwidth of H is greater than t. By the proof of Theorem 1, the program terminates with either the embedded binary tree as evidence, or a path-decomposition of width at most $f(t)$.

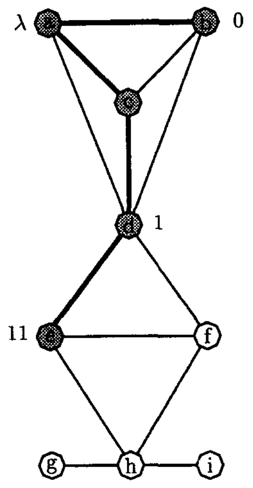
Note that the guest tree $B_{h(t)}$ has constant order $f(t)$, and so token operations that do not involve scanning H are constant time. In function GrowTokenTree, the only non-constant time operation is the check for blue neighbors in step 2. While scanning the adjacent edges of vertex u any edge to a red vertex can be removed, in constant time. Edge (u, v) is also removed when step 2.1 is executed. Therefore, across all calls to GrowTokenTree, each edge of H needs to be considered at most once, for a total of $O(n)$ steps. In program PathDecompositionOrSmallFatFactor, all steps except for GrowTokenTree are constant time. The total number of iterations through the loop is bounded by n, by the termination argument following the program. □

The next result shows that we can improve the pathwidth algorithm by restricting the guest tree. This allows us to use the subdivided tree obstructions given in Fig. 2.

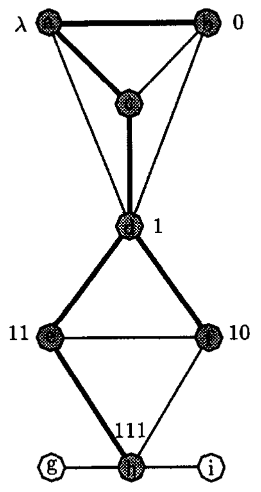
Corollary 3. *Any subtree of the binary tree $B_{h(t)}$ that has pathwidth greater than t may be used in the algorithm for Theorem* 1.



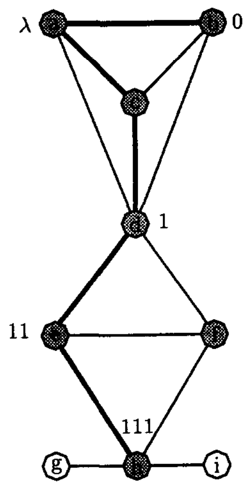
After step 2,
$P[0] = \{a, b, c, d, e\}$



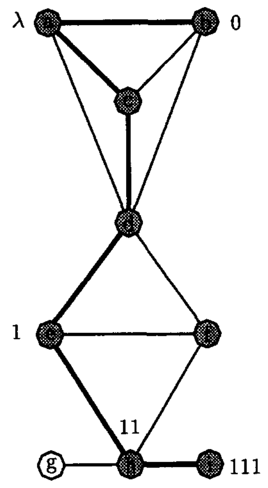
After 3.3, $T = 1$



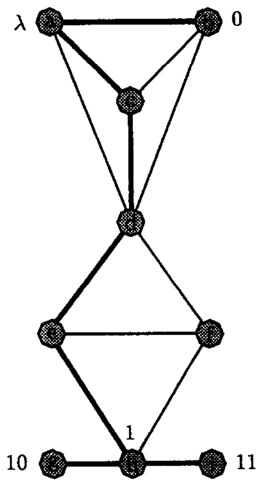
After 3.5
$P[1] = \{a, b, d, e, f, h\}$



$T = 10$, after 3.5
$P[2] = \{a, b, d, e, h\}$



$T = 1$, after 3.5
$P[3] = \{a, b, e, h, i\}$



$T = 1$, after 3.5
$P[4] = \{a, b, g, h, i\}$

Proof. The following simple modifications allow the algorithm to operate with a subtree. The subtree is specified by a set of *flagged* tokens in $B_{h(t)}$. At worst, the algorithm can potentially embed all of $B_{h(t)}$.

In step 2 of GrowTokenTree, the algorithm only looks for a flagged untokened child $T \cdot b$ to place, since unflagged tokens need not be place. The stopping condition in step 3 of PathDecompositionOrSmallFatFactor is changed to "all flagged tokens of $B_{h(t)}$ are placed or ...," so that termination occurs as soon as the subtree has been embedded. The relabeling in step 3.3. can place unflagged tokens of $B_{h(t)}$ on vertices of H since all the rooted subtress of a fixed height are not isomorphic. If that happens, we expand our guest tree (by adding flags) with those new tokens. It is easy to see that the new guest is still a tree. These newly flagged tokens may be relabeled by future edge subdivisions that occur above the token in the host tree. Duplication of token labels will not happen if unflagging is not allowed. Thus, the $f(t)$ width bound is preserved. □

In the proof of Corollary 3 one may wish to not expand the guest tree by flagging new tokens. This can be done and, in fact, is what we would do in practice. Without loss of generality, suppose token $T \cdot 1$ is on vertex $u \in H$ and has children that can not be placed on H, nd $T \cdot 1$ has one unflagged sibling token $T \cdot 0$ on $v \in H$. If we ignore the flagging of new vertices in the current algorithm, the token $T \cdot 1$ would be removed and the parent T (which has only one legitimate child) would be placed on vertex u. What happens to any blue vertices that are adjacent to only vertex v (or its unflagged subtree)? The answer is that they are lost and the algorithm would not terminate unless it could embed the guest tree in the remaining portion of H. We can fix this problem by checking for unflagged siblings before step 3.3 and then shift the token $T \cdot 1$ from u to v. See step 3.3′ below.

```
3.3′ **if** $T$ had one tokened child **then**
        replace all tokens $T \cdot b \cdot S$
          with $T \cdot S$
     **else if** $T = P \cdot b$ had an unflagged sibling
     **then**
        replace all tokens $P \cdot b \cdot S$
        with $P \cdot \text{not}(b) \cdot S$
     **endif**
```

Example. Using Tree-1 from Fig. 2 (subdivided $K_{1,3}$) as the guest tree of pathwidth 2 the program trace in Fig. 3 terminates with all vertices colored red (gray) yielding a path-decomposition of width 5.

Observe that our pathwidth algorithm provides an easy proof of basically the main result of [3] (or its earlier variant [20]) that for any forest F, there is a constant c, such that any graph not containing F as a minor has pathwidth at most c.

Corollary 4. *Every graph with no minor isomorphic to forest F, where F is a minor of a complete binary tree B, has pathwidth at most $c = |B| - 2$.*

Proof. Without loss of generality, we can run our pathwidth algorithm using (as the guest) any subtree T of B that contains F as a minor. Since at most $|B| - 1$ pebbles are used when we do *not* find an embedding of T (in any host graph), the resulting path decomposition has width at most $|B| - 2$. □

Our constant c is identical to the one given in [3] when $F = B$. They point out that their constant $c = |F| - 2$ is the best possible since the complete graph K_{c-1}, with pathwidth $c - 2$, does not contain any forest with c vertices.

4. Further directions

In the case that the pathwidth of an input graph G is at most k, our algorithm yields a path-decomposition that can have a width exponential in k, but that is equal in any case to the maximum number of tokens placed on the graph at any given time, minus 1. It would be interesting to know if this exponential bad behavior is "normal" or whether the algorithm tends to use a smaller number of tokens in practice. Since the pebbling proceeds according to a greedy strategy with much flexibility, there may be placement heuristics that can improve its performance on "typical" instances.

For applications in the theory of graph minors (in particular, in the theory of obstruction sets [9-11,13]) it would be interesting to know whether an analog of Theorem 1 can be proved for treewidth. For these applications the proof need not be given, as it is here, in the form of an efficient algorithm - a purely structural argument would suffice.

References

[1] S. Arnborg, Efficient algorithms for combinatorial problems on graphs with bounded decomposability - A survey, *BIT* **25** (1985) 2-23.

[2] S. Arnborg and A. Proskurowski, Linear algorithms for NP-hard problems restricted to partial k-trees, *Discrete Appl. Math.* **23** (1989), 11-24.

[3] D. Bienstock, N. Robertson, P.D. Seymour and R. Thomas, Quickly excluding a forest, *J. Combin. Theory Ser. B* **52** (1991) 274-283.

[4] H.L. Bodlaender, Dynamic programming algorithms on graphs with bounded tree-width, in: *Proc. 15th Internat. Coll. on Automata, Languages and Programming*, Lecture Notes in Computer Science **317** (Springer, Berlin, 1988) 105-119.

[5] H.L. Bodlaender, Some classes of graphs with bounded treewidth, *Bull. EATCS* **36** (1988) 116-126.

[6] H.L. Bodlaender, A tourist guide through treewidth, *Acta Cybernet.* **11** (1993) 1-23.

[7] H.L. Bodlaender, A linear time algorithm for finding tree-decompositions of small treewidth, in: *Proc. 25th ACM Symp. on Theory of Computing* (1993) 226-234.

[8] H.L. Bodlaender and T. Kloks, Better algorithms for the pathwidth and treewidth of graphs, in: *Proc. 18th Internat. Coll. on Automata, Languages and Programming* Lecture Notes in Computer Science **510** (Springer, Berlin, 1991) 544-555.

[9] K. Cattell and M.J. Dinneen, A characterization of graphs with vertex cover up to five, in: *Orders, Algorithms and Applications, ORDAL'94*, Lecture Notes in Computer Science **831** (Springer, Berlin, 1994) 86-99.

[10] K. Cattell, M.J. Dinneen and M.R. Fellows, Obstructions to within a few vertices or edges of acyclic, in: *Proc. 4th Workshop on Algorithms and Data Structures, WADS'95*, Lecture Notes in Computer Science (Springer, Berlin, 1995).

[11] K. Cattell, M.J. Dinneen, R.G. Downey and M.R. Fellows, Computational aspects of the graph minor theorem: Obstructions for unions and intertwines, University of Victoria draft, 1995.

[12] B. Courcelle and M. Mosbah, Monadic second-order evaluations on tree-decomposable graphs, *Theoret. Comput. Sci.* **109** (1993) 49-82.

[13] M.J. Dinneen, Bounded combinatorial width and forbidden substructures, Ph.D. Dissertation, University of Victoria, Victoria, B.C., Canada, 1995.

[14] J. Ellis, I.H. Sudborough and J. Turner, The vertex separation and the search number of a graph, *Inform. and Comput.* **113** (1994) 50-79.

[15] M.R. Fellows and M.A. Langston, On well-partial-order theory and its application to combinatorial problems of VLSI design, *SIAM J. Discrete Math.* **5** (1992) 117-126.

[16] T. Kloks, Treewidth, Ph.D. Thesis, Utrecht University, Utrecht, the Netherlands, 1993.

[17] J. Lagergren, Efficient parallel algorithms for tree-decomposition and related problems, in: *Proc. 31st Symp. on Foundations of Computer Science* (1990) 173-182.

[18] R.H. Möhring, Graph problems related to gate matrix layout and PLA folding, in: G. Tinhofer, E. Mayr, H. Noltemeier and M. Syslo, eds., *Computational Graph Theory, Computing Supplementum* **7** (Springer, Berlin, 1990) 17-51.

[19] B. Reed, Finding approximate separators and computing tree-width quickly, in: *Proc. 24th Ann. Symp. on Theory of Computing* (1992) 221-228.

[20] N. Robertson and P.D. Seymour, Graph Minors. I. Excluding a forest, *J. Combin. Theory Ser. B* **35** (1983) 39-61.

[21] N. Robertson and P.D. Seymour, Graph minors II: Algorithmic aspects of treewidth, *J. Algorithms* **7** (1986) 309-322.

[22] N. Robertson and P.D. Seymour, Graph minors X: Obstructions to tree-decomposition, *J. Combin. Theory Ser. B* **52** (1991) 153-190.

[23] N. Robertson and P.D. Seymour, Graph minors XIII: The disjoint paths problem, Manuscript, 1986.

[24] N. Robertson and P.D. Seymour, Graph minors XVI: Wagner's conjecture, to appear.

[25] T.V. Wimer, S.T. Hedetniemi and R. Laskar, A methodology for constructing linear graph algorithms, *Congr. Numer.* **50** (1985) 43-60.