Approximating the Pathwidth of Outerplanar Graphs*

Rajeev Govindan†, Michael A. Langston†and Xudong Yan‡

Abstract

Pathwidth is a well-known $\mathcal{NP}$-Complete graph metric. Only very simple classes of graphs, such as trees, are known to permit practical pathwidth algorithms. We present a technique to approximate the pathwidth of outerplanar graphs. Our algorithm works in linear time, is genuinely practical and produces solutions at most three times the optimum.

Keywords: algorithms, pathwidth, outerplanar graphs, approximation, tree decomposition.

1 Introduction

Pathwidth was defined by Robertson and Seymour in their seminal series of papers on Graph Minors [6]. Since then, this metric has found application in many areas, ranging from circuit layout to natural language processing [4, 5]. Determining pathwidth is $\mathcal{NP}$-Complete. Thus, it is natural to search for fast approximation algorithms. No polynomial-time relative approximation algorithm (one whose solution is within a multiplicative constant of the optimum) is known for the general problem. Moreover, no polynomial-time absolute approximation algorithm (one whose solution is within an additive constant of the optimum) can exist unless $\mathcal{P} = \mathcal{NP}$ [2].

The main result of this paper is a practical relative approximation algorithm for the pathwidth problem on outerplanar graphs. Since outerplanar graphs have treewidth two or less, the methods in [1] can, in principle, be used to compute the pathwidth exactly in polynomial time. This is not a realistic option, however, because of the high degree of the polynomial and its enormous multiplicative constant. In contrast, our algorithm approximates the pathwidth to within a factor of three of the optimum in practical linear time.

*This research is supported in part by the Office of Naval Research under contract N00014–90–J–1855.
†Department of Computer Science, University of Tennessee, TN 37996–1301
‡Prism Solutions Inc, 1000 Hamlin Court, Sunnyvale, CA 94089

2 Our Approach

2.1 Tree and Path Decompositions

We consider only connected graphs without loops or multiple edges.

A *tree decomposition* of a graph G is a pair (T, Y), where T is a tree and $Y = \{Y_i \mid i \in V(T)\}$ is a collection of subsets of $V(G)$ such that (i) for each edge $e \in E(G)$, some Y_i contains both end-points of e, and (ii) for all $i, j, k \in V(T)$, if j is on the path between i and k in T, $Y_i \cap Y_k \subseteq Y_j$. The *width* of a tree decomposition (T, Y) is one less than the size of the largest set in Y. The *treewidth* of G (denoted $tw(G)$) is the smallest width of all its tree decompositions.

A *path decomposition* of G is a sequence $X_1, \ldots, X_r$ of subsets of $V(G)$ such that (i) for each edge $e \in E(G)$, some X_i contains both end-points of e, and (ii) for $1 \leq i \leq j \leq k \leq r$, $X_i \cap X_k \subseteq X_j$. The *width* of a path decomposition $X_1, \ldots, X_r$ is one less than the size of the largest set X_i, $1 \leq i \leq r$. The *pathwidth* of G (denoted $pw(G)$) is the smallest width of all its path decompositions.

2.2 A Conversion Procedure

Path decompositions can be derived from tree decompositions. We employ such a procedure, **td2pd**, and prove its correctness. It requires a routine to construct optimal path decompositions of trees. For this, we use the linear-time method presented in [3]. Since the run-time of **td2pd** is dominated by the time spent in this routine, **td2pd** runs in linear time as well.

Procedure td2pd

Input: A tree decomposition (T, Y) of a graph G.
Output: A path decomposition of G.
begin procedure
  $X_1, \ldots, X_r :=$ an optimal path decomposition of T;
  for $1 \leq i \leq r$ **do**
    $P_i := \bigcup_{j \in X_i} Y_j$;
  output $P_1, \ldots, P_r$;
end procedure

Theorem 1 Let (T, Y) denote a width-t tree decomposition of a graph G. Then **td2pd**$((T, Y))$ returns a path decomposition of G with width no more than $(t + 1)(pw(T) + 1) - 1$.

Proof Let $X_1, \ldots, X_r$ denote the optimal path decomposition of T constructed in **td2pd**, and let $P_1, \ldots, P_r$ denote the output of **td2pd**. Then, for $1 \leq i \leq r$, $|P_i| = |\bigcup_{j \in X_i} Y_j| \leq (t+1)(pw(T)+1)$. Thus the width condition is satisfied, and we only need to check that $P_1, \ldots, P_r$ is a valid path decomposition of G.

It is easy to see that $P_1, \ldots, P_r$ covers all edges in G. We prove by contradiction that $P_1, \ldots, P_r$ has the intersection property. If the intersection property does not hold, then for some $1 \leq i < j < k \leq r$, there is a vertex v in $P_i \cap P_k$ that is not in P_j. Since $v \in P_i \cap P_k$, there must exist $l \in X_i$ and $m \in X_k$, such that v belongs to Y_l and Y_m. Consider the subsets V_1 and V_2 of $V(T)$, where $V_1 = \bigcup_{p<j} X_p - X_j$ and $V_2 = \bigcup_{p>j} X_p - X_j$. The intersection property of $X_1, \ldots, X_r$ implies that V_1 and V_2 are disjoint. Moreover, there is no edge in T connecting V_1 and V_2, because some X_q must contain both end-points of such an edge, contradicting the disjointness of V_1 and V_2. Thus every path between V_1 and V_2 in T contains a vertex from X_j. In particular, the path between l and m must contain a vertex, say h, from X_j. By the intersection property of (T, Y), $v \in Y_h$. Since $h \in X_j$, $Y_h \subseteq P_j$ and $v \in P_j$, a contradiction. ■

3 Path Decompositions of Outerplanar Graphs

A graph is *outerplanar* if it has a planar embedding with all vertices lying on a single face. Outerplanar graphs have treewidth at most two. In this section, we develop an algorithm that, for an outerplanar graph G, constructs an optimal tree decomposition (T, Y) with $pw(T) \leq pw(G)$. By Theorem 1, running **td2pd** on (T, Y) produces a path decomposition with width at most $3 \times pw(G) + 2$.

We say that (T, Y) is *simple* if (T, Y) has width at most two, T is a subgraph of G, and $v \in Y_v$ for all $v \in V(T)$. (Since our algorithms use vertex labels, we insist that the labels of $V(T)$ respect those of $V(G)$.) Because pathwidth cannot be increased by taking a subgraph, if (T, Y) is simple, then $pw(T) \leq pw(G)$. Our algorithm constructs (T, Y) by combining tree decompositions of G's subgraphs. Let (T', Y') and (T'', Y'') denote tree decompositions of subgraphs G' and G'', respectively. Suppose that $V(T')$ and $V(T'')$ are disjoint, and that there are vertices $u \in V(T')$ and $v \in V(T'')$, such that all the vertices in $V(G') \cap V(G'')$ are in both Y'_u and Y''_v. Then we may obtain a tree decomposition of $G' \cup G''$ by adding the edge uv. This decomposition is simple if (T', Y') and (T'', Y'') are simple and if $uv \in E(G') \cup E(G'')$.

3.1 Biconnected Graphs

We concentrate initially on biconnected graphs (those without cut points).

Lemma 1 Let G be biconnected, outerplanar and of order at least three. Let v denote a vertex in G. Then G contains a path P with at least two edges, and with endpoints w and x, such that the following conditions hold :

- $G-(P-\{w,x\})$ is biconnected, outerplanar, and contains v,
- w and x are adjacent in G (and hence, in $G-(P-\{w,x\})$), and
- every edge in G is either in P or in $G-(P-\{w,x\})$.

Proof If G is a cycle, then the lemma is satisfied by setting P to $G-\{uv\}$, where u is a vertex adjacent to v. Otherwise, fix an outerplanar layout of G. Let E_i denote the set of internal edges of G (those not on the external face). Orient the layout so that some edge e' is rightmost and some edge e'' is leftmost in E_i. If v is to the left of e', then setting P to the path consisting of all the external edges to the right of e' satisfies the lemma. Otherwise, set P to the path containing all the external edges to the left of e''. ∎

If a path P contains at least two edges and has endpoints w and x, then it has a width-two tree decomposition (T,Y) such that $T=P-\{w,x\}$ and for $i \in V(T)$, $Y_i=\{x,i,j\}$, where j is the neighbor of i on w's side (the sets Y_i actually form a path decomposition of P). We call (T,Y) a *w-extensible* tree decomposition of P. Figure 1 shows a path and its w-extensible tree decomposition. The sets Y_i are shown inside the ovals.

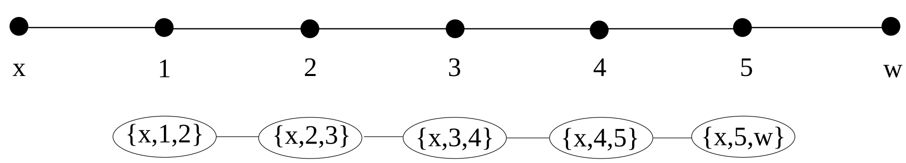

Figure 1: A path and its w-extensible tree decomposition.

Note that for every edge $ij \in E(P)$, either $\{i,j\} \subseteq Y_i$ or $\{i,j\} \subseteq Y_j$. We use the notion of extensibility to derive **bc-op-td**, our algorithm to construct simple tree decompositions of biconnected outerplanar graphs.

Procedure bc-op-td

```
Input:    A biconnected outerplanar graph G of order two or more, and a vertex v in G.
Output:   A simple tree decomposition (T, Y) of G, with T spanning G − {v}.
begin procedure
    if |V(G)| = 2
        then begin
            u := the vertex adjacent to v;
            T := {u}  and  Y := {Y_u}, where Y_u := {u, v};
            end
        else begin
            P := a path, between some two vertices w and x, that satisfies Lemma 1;
            (T', Y') := bc-op-td(G − (P − {w, x}), v);
            if {w, x} ⊆ Y'_w
                then begin
                    e := the edge incident on w in P;
                    (T'', Y'') := the w-extensible tree decomposition of P;
                    end
                else begin
                    e := the edge incident on x in P;
                    (T'', Y'') := the x-extensible tree decomposition of P;
                    end
            T := T' ∪ T'' ∪ {e}  and  Y := Y' ∪ Y'';
            end
    output (T, Y);
end procedure
```

At this point, we may as well assume that v is chosen at random. A specific choice of v is necessary when G is a biconnected component of a larger graph (see Section 3.3).

Lemma 2 Let G be biconnected and outerplanar, and let v denote a vertex in G. Let (T, Y) denote the result of the call to **bc-op-td**(G, v). Then (T, Y) is a simple tree decomposition of G, and T is a spanning tree of $G - \{v\}$.

Proof We prove, using induction on $|E(G)|$, a somewhat stronger result. We show that (T, Y) is simple, that T spans $G - \{v\}$, and that for each edge ij in G, either $i \in V(T)$ with $\{i, j\} \subseteq Y_i$ or $j \in V(T)$ with $\{i, j\} \subseteq Y_j$. The lemma holds for the basis case, in which G contains just one edge. If $|E(G)| > 1$, let P, with endpoints w and x, denote a path that satisfies Lemma 1. Let G' denote $G - (P - \{w, x\})$. Thus v is in G'. By the induction hypothesis, **bc-op-td**(G', v) returns a simple tree decomposition (T', Y'), with T' spanning $G' - \{v\}$, and with $\{w, x\} \subseteq Y'_w$ or $\{w, x\} \subseteq Y'_x$. Assume, without loss of generality, that $\{w, x\} \subseteq Y'_w$. Let (T'', Y'') denote the w-extensible tree decomposition of P. Then $T'' = P - \{w, x\}$ and $\{w, x\} \subseteq Y''_a$, where a is w's neighbor in P. T is formed by adding an edge between vertex w in T' and vertex a in

T''. The only vertices common to G' and P are w and x, which are contained in both Y'_w and Y''_a. Therefore (T,Y) is a valid tree decomposition of G. (T,Y) is simple because (T',Y') and (T'',Y'') are simple, with the edge wa existing in G. T' spans $G-\{v\}$ because T' spans $G'-\{v\}$ and T'' spans $P-\{w,x\}$. To complete the induction, observe that for each edge $ij \in E(G)$, $\{i,j\} \subseteq Y_i$ or $\{i,j\} \subseteq Y_j$, because either $\{i,j\} = \{w,a\} \subseteq Y''_a$ or $\{i,j\}$ is contained in one of Y'_i, Y'_j, Y''_i and Y''_j. ∎

3.2 Efficiency

We store the input graph in doubly-linked adjacency list format. This is space-efficient, because outerplanar graphs have a linear number of edges (if $tw(G) \leq 2$, then $|E(G)| < 2|V(G)|$). We also employ a few additional links. To facilitate the removal of an edge ab, links are maintained between the copy of b in a's adjacency list and the copy of a in b's adjacency list. The only steps in **bc-op-td** that take more than constant time are (i) finding a path P that satisfies Lemma 1, (ii) deleting the edges and internal vertices of P from the input graph, and (iii) constructing an extensible tree decomposition of P. Of these, steps (ii) and (iii) take at most linear time over all calls to **bc-op-td**. Thus the question of efficiency reduces to the implementation of step (i). One fast method is described below.

Some preprocessing is required. We first construct an outerplanar layout of G. We scan the layout in a clockwise direction, starting at v, and number vertices in the order in which they are encountered. Then we rearrange the adjacency list of each vertex, a, so that neighbors numbered lower than a occur before neighbors numbered higher than a. Each of these tasks takes only linear time.

Once preprocessing is completed, paths to play the role of P are found during a second clockwise scan. It follows from Lemma 1 that, until G is reduced to a cycle, a pair of vertices may be the endpoints of P if and only if they are adjacent by an internal edge and all vertices with numbers between them have degree two. Vertices of degree three or more are maintained on a stack. As a new vertex is scanned, we check whether it is adjacent by an internal edge to the vertex on top of the stack. If it is, then we have found P's endpoints. If not, we push the new vertex on the stack and continue the scan.

It turns out that no vertex will be pushed on the stack as long as an internal edge makes it adjacent it to a lower-numbered vertex. This, in turn, implies that G is reduced to a cycle before the scan returns to v. Thus the scan terminates after a linear number of steps, and we only need argue that each step can be accomplished in constant time. Let k denote the vertex being scanned, and j the vertex on top of the stack. We need to check whether j and k are adjacent by an internal edge. Since G is outerplanar, and since j cannot be adjacent to a lower-numbered vertex by an internal edge, either j is adjacent by an internal edge only to k, or k is adjacent by an internal edge to no lower-numbered vertex other than j. In the first case, j has degree at most three. In the second, j can only be one of the first two elements in k's adjacency list. Therefore, we need to scan at most five elements in the adjacency lists of j and k. Thus step (i) requires only linear time, and so does **bc-op-td**.

3.3 Tackling Non-Biconnected Graphs

We now generalize our algorithm to handle all outerplanar graphs.

Procedure op-td

Input: An outerplanar graph G of order two or more, and sets B and C of its biconnected components and cut points.
Output: A simple tree decomposition (T, Y) of G, with T spanning G.
begin procedure
    if G is biconnected
        then begin
            $u, v :=$ any two adjacent vertices in G;
            $(T', Y') :=$ **bc-op-td**(G, v);
            $T := T' \cup \{v\} \cup \{uv\}$ and $Y := Y' \cup \{Y_v\}$, where $Y_v = \{v\}$;
            end
        else begin
            $B_i :=$ an element of B that contains exactly one vertex v from C;
            if v is not a cut point in $G - (B_i - \{v\})$ **then** $C := C - \{v\}$;
            $(T', Y') :=$ **bc-op-td**(B_i, v);
            $(T'', Y'') :=$ **op-td**$(G - (B_i - \{v\}), B - \{B_i\}, C)$;
            $u :=$ an arbitrary neighbor of v in B_i;
            $T := T' \cup T'' \cup \{uv\}$, and $Y := Y' \cup Y''$;
            end
    output (T, Y);
end procedure

Lemma 3 Let G be outerplanar. Let (T, Y) denote the result of the call to **op-td**(G). Then (T, Y) is a simple spanning tree decomposition of G.

Proof The proof proceeds by induction on the number of biconnected components of G. The basis case, when G is biconnected, follows from Lemma 2 and the modifications made to (T, Y)

after the call to **bc-op-td**(G, v). So let $B_i, v, (T', Y'), (T'', Y'')$ and u be as defined in **op-td**. Let $\hat{G}$ denote $G - (B_i - \{v\})$. From the proof of Lemma 2, we know that (T', Y') is simple, that it spans $B_i - \{v\}$, and that $\{u, v\} \subseteq Y'_u$ (there is no Y'_v). By the induction hypothesis, (T'', Y'') is a simple spanning tree decomposition of $\hat{G}$. Thus, by construction, (T, Y) is a simple spanning tree decomposition of G. ∎

3.4 Main Result

Biconnected components and cut points can be found using a depth-first search. Procedure **op-td** builds an optimal tree-decomposition using **bc-op-td**. This tree decomposition is converted into a path decomposition using **td2pd**. Recalling Theorem 1, and noting that each of the aforementioned steps requires only linear time, we achieve the following result.

Theorem 2 If G is outerplanar, a path decomposition of G with width at most $3 \times pw(G) + 2$ can be constructed in linear time.

4 Concluding Remarks

We have implemented our algorithm in the C programming language. Tests on a SPARC ULTRA indicate that the implementation is fast in practice, taking, for instance, less than two seconds to compute the path decomposition of a graph with ten thousand vertices. It is difficult to gauge the quality of the solutions produced, because there is no practical way to obtain optimal path decompositions for comparison. As a compromise, we tested the program on pseudo-random outerplanar graphs of known pathwidth. These tests indicate that the approximate decompositions tend to have much smaller width than the worst case guarantee.

Our work has exploited the fact that if the width of a tree decomposition (T, Y) of G is bounded, and if $pw(T)$ is within some constant multiple of $pw(G)$, then we can construct a path decomposition of G whose width is at most a constant times $pw(G)$. Series-parallel graphs also have treewidth at most two. Optimal tree decompositions for them can be constructed quickly. We believe that, for these graphs, it is possible to ensure $pw(T) \leq 2pw(G)$, yielding a factor-of-six relative approximation algorithm.

On a more general note, we conjecture that any graph G has an optimal tree decomposition (T, Y) such that $pw(T) \leq pw(G)$. If true, a constructive proof of this would provide a relative approximation algorithm for any class of graphs whose bounded-width tree decompositions can be found efficiently. Currently, this class includes all graphs of treewidth four or less and, for any fixed k, k-chordal graphs, k-outerplanar graphs and graphs with disk dimension k, to name just a few.

References

[1] H. L. Bodlaender and T. Kloks, "Efficient and Constructive Algorithms for the Pathwidth and Treewidth of Graphs," *Journal of Algorithms* 25 (1996), 358–402.

[2] N. Deo, M. S. Krishnamoorthy and M. A. Langston, "Exact and Approximate Solutions for the Gate Matrix Layout Problem," *IEEE Transactions on Computer-Aided Design* 6 (1987), 79–84.

[3] J. A. Ellis, I. H. Sudborough and J. S. Turner, "The Vertex Separation and Search Number of a Graph," *Information and Computation* 113, August 1994, 50–79.

[4] M. R. Fellows and M. A. Langston, "On Well-Partial-Order Theory and Its Application to Combinatorial Problems of VLSI Design," *SIAM Journal on Discrete Mathematics* 5:1 (1992), 117–126.

[5] A. Kornai and Z. Tuza, "Narrowness, pathwidth, and their application in natural language processing," *Discrete Applied Mathematics* 36 (1992), 87–92.

[6] N. Robertson and P. D. Seymour, "Graph Minors II. Algorithmic Aspects of Treewidth," *Journal of Algorithms* 7 (1986), 309–322.

[7] X. Yan, "A Relative Approximation Algorithm for Computing Pathwidth," Master's Thesis, Department of Computer Science, Washington State University, Pullman, WA 99164 (1989).