GRAPH SEPARATION AND SEARCH NUMBER

J.A Ellis[1*], I.H Sudborough[1*], J.S Turner[2]
(1) Electrical Engineering and Computer Science Department, Northwestern University, Evanston, Ill 60201
(2) Computer Science Department
Washington University, St. Louis, Missouri 63130

ABSTRACT

We relate two different concepts in graph theory and algorithmic complexity, namely "searching" a graph [1] [2] and "separators" of undirected graphs [3] [4] [5]. For any graph G, let s(G) denote the "search number" of G and vs(G) denote the vertex separation of G. We show that $vs(G) \leq s(G) \leq vs(G) + 2$ and that if G′ is created from G by placing two added vertices in every edge in G, then $vs(G') = s(G)$.
We discuss algorithms which show that for all fixed $k \geq 1$ and graphs G the following problems are decidable in polynomial time: "Is $vs(G) \leq k$?", "Is $s(G) \leq k$?", "Is the progressive black/white pebble demand of G $\leq k$?". When both G and k are input quantities, these problems are NP complete [6] [7].
We also give an algorithm that for any tree T computes vs(T) in $O(n \log n)$ steps, where n is the number of vertices in the tree, describe the smallest trees with a given vertex separation and characterize trees of a given vertex separation.

1. INTRODUCTION

This paper is an abbreviated version of a paper in preparation which contains detailed proofs of the theorems stated here. Searching a graph is a concept introduced by Parsons [1]. Informally, the search number of a graph G, denoted by s(G), is the minimum number of searchers necessary to guarantee the capture of a fugitive who can move with arbitrary speed about the edges of the graph and who has perfect knowledge of the movements of the searchers. Meggido et al [2] show that determining the search number of a graph is NP-hard and that the search number of a tree can be determined in O(n) steps. It is known that, for any graph G with maximum vertex degree 3, s(G) is identical to the "cutwidth" of G [8]. Hence the search number problem has practical value since finding the cutwidth of a graph is important in some VLSI layout applications [3].

A search sequence is a sequence of the following steps: (a) place a searcher on a vertex, (b) slide a searcher through an edge and (c) remove a searcher from a vertex. We say that an edge $e=\{x,y\}$ is cleared if either (1) there is a searcher on x and a second searcher is moved from x to y or (2) there is a searcher on x, all edges incident to x except e have been cleared, and the searcher on x is shifted through e to y. In the beginning all edges are "contaminated", i.e. not cleared. Searching a graph means to reach a state in which all edges

*Research supported in part by NSF grant #MCS81-09280

are simultaneously cleared. A cleared edge e can become recontaminated by the movement or deletion of a searcher which results in a path without searchers from a contaminated edge to e. LaPaugh [7] has shown that there is always an optimal, so called progressive, search sequence which never allows any edge to be recontaminated. Hence the search number problem is NP complete.

Separation of undirected graphs is an important concern in complexity theory. A separator of an undirected graph is a set of vertices, in the case of a vertex separator, or a set of edges, in the case of an edge separator, whose removal separates the graph into two components. Separators are used to describe good VLSI layouts in [3] and have been used to describe good divide and conquer algorithms in [4]. Separator theorems for planar graphs have been described by Lipton and Tarjan [5]. Lengauer [6] called the above definition of separator static and goes on to define a dynamic "vertex separator game". We consider the same concept as Lengauer but describe it in terms of linear layouts.

Let $G=(V,E)$ be a graph. A linear layout, or simply a layout, of G is a one-one mapping $L:V \to \{1,2,\ldots,|V|\}$.
Define $V_L(i) = \{u \text{ in } V \mid \{u,v\} \text{ in } E \text{ and } L(u) \leq i < L(v)\}$.
Vertices in $V_L(i)$ are called the active vertices at time i. The vertex separation of G with respect to L, denoted by $vs_L(G)$, is defined by

$$vs_L(G) = \max_{1 \leq i < n} (|V_L(i)|)$$

and the vertex separation of G is defined by

$$vs(G) = \min_L (vs_L(G))$$

Examples of layouts which minimize vertex separation are given in Figures 2.1 and 2.2.

2. RELATIONSHIPS BETWEEN VERTEX SEPARATION AND SEARCH NUMBER

Theorem 2.1
Let $G = (V,E)$ be a graph. Then $vs(G) \leq s(G) \leq vs(G) + 2$.

Proof Sketch To show that $vs(G) \leq s(G)$ we show that given a search sequence we can use it to derive a layout. Let S be a progressive search sequence for G. We define the layout L for G by assigning the vertex x to the integer i if it is the i-th vertex to be first visited by a searcher in the sequence S. It can be shown that, since S is a progressive strategy and does not allow recontamination, there must be at least a searcher on every active vertex at each point in the layout. Hence $vs(G) \leq s(G)$.

To show that $s(G) \leq vs(G) + 2$ we show that given a layout we can use it to derive a search strategy. Let L be a layout for G. We clear all the edges of G by the sequence of steps given in the procedure search described below. It is easily seen that the procedure is effective and that it never uses more than (the number of currently active vertices + 2) searchers .

Hence $s(G) \leq vs(G) + 2$.

```
procedure search(G,L);
    for i := 1 to |V| do
        begin x := $L^{-1}(i)$; Place a searcher on x;
        for each left neighbor y of x do
            begin add a new searcher to y;
            slide a searcher from y to x;
            remove a searcher from x
            end;
        Remove searchers from inactive vertices in $L_i$
        end;
```

The bound in Theorem 2.1 is the best possible because the graph $K_{3,3}$ shown in Figure 2.1 has vertex separation three and search number five.

Let the 2-expansion of a graph G be the graph formed by replacing each edge $\{x,y\}$ of G with the two new vertices, say u,v, and the edges $\{x,u\}$, $\{u,v\}$, and $\{v,y\}$. For example, Figures 2.2 and 2.3 show a graph G and its 2-expansion.

Theorem 2.2. For any graph G, s(G) is identical to the vertex separation of the 2-expansion of G.

Proof Sketch In Theorem 2.1 we have seen that, for any graph G, $vs(G) \leq s(G)$. Let G′ be the 2-expansion of G. Clearly, subdividing edges does not change search number, so $s(G) = s(G')$. Thus, it follows that $vs(G') \leq s(G)$.

To show that $s(G) \leq vs(G')$ we argue similarly as in the proof of Theorem 2.1. Let L be a layout of G′ such that $vs_L(G')=k$. It can be shown that there is a layout of G′ with vertex separation k such that, for any edge $e = \{x,y\}$ in G, the vertices added to e are given consecutive positions and are laid out either to the right of x or to the right of y. We construct a searching algorithm based on a layout L which does satisfy these properties. Because of the special property there is now no need to use more searchers than the number of active vertices at each point. Figure 2.2 shows the complete graph on four vertices which has search number 4 and vertex separation 3. Figure 2.3 shows the 2-expansion and a layout satisfying the properties just described and with a vertex separation of 4, which is optimal. The search number is unchanged.

3. THE VERTEX SEPARATION OF TREES

Trees allow the possibility of computing certain properties of the root by recursively computing the property for its subtrees and then combining the results. For example, Meggido et al [2] give an algorithm for computing the search number of a tree, Chung et al [9] give an algorithm for computing the cutwidth of a tree, and Yannakis [10] describes an improved cutwidth algorithm for trees which is also effective for computing their black/white pebble demand.

Our algorithm also is based on a recursive characterization of the vertex separation of a tree in terms of the vertex separation of the subtrees induced by its root. The algorithm is easily developed to give an optimal layout for vertex separation for the given tree.

3.1 A Recursive Characterization of Trees with Vertex Separation k

The subtrees induced by a vertex x are all of the trees in the forest obtained by deleting x from the given tree. A tree with one vertex has vertex separation 0 and we shall adopt the convention that the empty tree has vertex separation -1.

Theorem 3.1 Let T be a tree. For any integer $k \geq 1$, let P(k) denote the following property of T:
For all vertices x at most two subtrees induced by x have vertex separation k and all remaining subtrees induced by x have vertex separation at most k-1.
T has vertex separation at most k if and only if P(k) is true.

Proof Sketch We first show that if T satisfies P(k), then there is a layout L of T such that $vs_L(T) \leq k$. Let V_k be the set of vertices in T which induce exactly two subtrees each with vertex separation k. It can be shown that there exists a path containing all the members of V_k. Let $x_1, x_2, .. , x_p$, for some $p \geq 1$, be a chain of vertices such that, for all i ($1 \leq i \leq p$), x_i induces subtrees which, if they do not contain other chain vertices, have vertex separation at most k-1. Given such a chain we can construct a layout with vertex separation k by laying out all the subtrees with vertex separation $\leq$ k-1 in sequence immediately to the right of the chain vertex that induces them. See Figure 3.1. In this figure a, b, c and d are members of the chain.

Now we show that if there is a layout L such that $vs_L(T)$ is at most k, then P(k) is true. Let vertices a and b be the first and last vertices in such a layout. Let x be any vertex in T. There must be paths from both a and b to x. Remove from the layout the vertex x, all edges incident to x, and the one or two subtrees induced by x and containing the vertices a and b. It can be seen that for all remaining subtrees T', $vs(T') \leq$ k-1 because the paths from a and b to x are removed. No more than two subtrees were removed and these had vertex separation $\leq$ k.

Corollary 3.1 $vs(T) > k$ iff there exists a vertex which induces 3 subtrees T' such that $vs(T') \geq k$.

For example, the tree shown in Figure 3.2 has vertex separation 3 and not 2, since the indicated vertex x induces three subtrees with vertex separation 2.

3.2 k-Criticality

In the following we shall consider directed trees for convenience. The vertex separation of a directed tree is the vertex separation of the underlying undirected tree. Let T<x> denote the subtree of the directed tree T with root vertex x.

Definition 3.1. A vertex x is k-critical in a directed tree T if (1) vs(T<x>) = k and (2) there are two children y and z of x such that vs(T<y>) = k and vs(T<z>) = k.

Let $T<x,v_1, v_2, \ldots, v_i>$ denote the subtree of T with the root vertex x from which the subtrees $T<v_1>, \ldots, T<v_i>$ have been deleted. It follows from Theorem 3.1 that, if T^1 is a directed tree with root x and vertex separation k, and there is a k-critical vertex v in T, then $vs(T<x,v>) \le k-1$. We shall assign a set of integers as a label to all vertices in a tree. These labeling sets have the following interpretation. If a vertex x is assigned the label $(a_1, a_2, \ldots, a_d)$ in the directed tree T, then:

1. $vs(T<x>) = a_1$,
2. for all i ($1 \le i < d$), there exists an a_i-critical vertex v_i such that $vs(T<x,v_1,v_2,\ldots,v_{i-1},v_i>) = a_{i+1}$,
3. $vs(T<x,v_1,v_2,\ldots,v_{d-1}>) = a_d$ and $T<x,v_1,v_2,\ldots,v_{d-1}>$ does not have an a_d-critical vertex.

For example the label {2,0} on vertex x means that T<x> has vertex separation 2, that there is a 2-critical vertex v and that T<x,v> has vertex separation 0. The label {2,-1} on the vertex x means that the vertex separation of T(x) is 2 and the vertex v=x is a 2-critical vertex so that T<x,v> is empty.

We now describe a procedure for combining the labels of the d subtrees induced by some vertex x to compute the label of x.

```
procedure compute-label($S_1,S_2,\ldots,S_d$)
k <--- max( $S_1 \cup S_2 \cup \ldots \cup S_d$ );
if k = 0 then return {1};
if d = 1
then begin
     if min $S_1$ > 0 then return $S_1$;
     if min $S_1$ = -1 then return {0} U {i > 0 | i is in $S_1$}
     i <--- min { j | j > 0 and j is not in $S_1$ };
     return {i} U { j > i | j is in $S_1$ }
     end;
m <--- | { $S_i$ | k is in $S_i$ } |;
if m $\ge$ 3 then return {k+1};
if m=2
then begin (* let k be in the sets $S_x$ and $S_y$ *)
      if $S_x$ > {k} or $S_y$ > {k} then return {k+1};
      return {k,-1}
       end;
(* In this case the element k is in one set, say $S_x$ *)
if k = min $S_x$ then return {k}
H <--- compute-label($S_1,S_2,\ldots,S_x-\{k\},\ldots,S_d$);
if H $\ge$ {k} then return {k+1} else return {k} U H
```

An outer procedure, called initially on the root, will return {0} as the label all leaves. On other than leaves it recursively calls itself on each of the children and passes the resulting labels to the procedure compute-label which computes the label of x. The vertex separation of a tree is then the largest number in the label of its root.

The correctness of the procedure can be established by considering each of the various cases that it handles and applying Theorem 3.1. An analogous procedure for computing the cutwidth of trees was described in [9].

Assume that suitable data structures are used and note that there are O(log n) entries in each label, because (section 3.4) the maximum vertex separation of a tree with n vertices is O(log n). Recursion depth is limited by the number of items in a label so the time complexity of the procedure combine-label is $O(d_i * \log n)$, where d_i is the degree of the ith vertex. The procedure is invoked for all vertices in the tree. Since the sum of the vertex degrees over all vertices is O(n) we have that the time complexity of the entire process is O(n log n).

3.3 Computing an Optimal Layout

Once labels have been computed for all vertices, an optimal layout can be computed. This is accomplished by the following procedure in which label(x) means the label of x, pos is the position assigned to a vertex and L(x) is the layout function.

```
procedure layout (x);
    k <-- max label(x); c <-- x;
    while c is not a k-critical vertex do
    begin delete k from label(c);
        c <-- the child of c with k in label(child)
    end;
    Let ($v_1$, $v_2$, ..... $v_s$) be the path of all
    vertices in T<c> with k in the label;
    for i := 1 to s do
        begin L($v_i$) <-- pos;
        pos <-- pos + 1; delete $v_i$ from T;
        for all children y of $v_i$ do layout(y);
        if $v_i$ has a parent not in ($v_1$ ... $v_s$)
        then layout(x)
        end
```

The correctness of this procedure can be demonstrated by noting that it is finding the chain of vertices that induce subtrees of vertex separation k-1 or less, as discussed in the proof of Theorem 3.1, and laying out these subtrees to the right of their inducing chain vertex.

The time complexity of the procedure is O(n log n) since no vertex is visited more than k times and k is O(log n) as shown in section 3.4.

3.4 Smallest Trees with a Given Vertex separation

Let T(k) denote the collection of trees with the smallest number of of vertices and vertex separation k. It is easy to see that T(1) and T(2) contain just one tree. There are several trees in the set T(3). One of is shown in Figure 3.2. To construct a tree in the set T(k+1), for all k, one takes three trees from T(k), say T_1, T_2, and T_3, which need not be distinct, and a new vertex x, and joins the three subtrees by adding an edge from an arbitrary vertex of T_1, T_2, and T_3 to the vertex x. It follows from Theorem 3.1 that the constructed tree has vertex separation k+1. Furthermore, from Theorem 3.1, any tree with vertex separation k+1 must have a vertex x such that x induces at least three subtrees with vertex separation at least k. Consequently, the constructed tree must be among the smallest trees with vertex separation k+1.

One can also deduce that the number of vertices in a smallest tree with vertex separation k is $|5/6*3^k|$ for all $k \geq 0$ so that vs(T) is O(log |V|).

3.5 Characterization of Trees with Vertex separation k

The operation of replacing an edge {x,y} with a new vertex z and the two edges {x,z} and {z,y} is called edge subdivision. A graph G′ is a homeomorphic image of a graph G when G′ can be obtained from G by a finite number of edge subdivisions.

Let T be a tree. Let S(T) denote the set of all trees that can be obtained from T by a single edge subdivision operation. Similarly, if F is a family of trees, then S(F) denotes the family of trees { S(T) | T is in F }. For all $i \geq 1$, let F(i) be the family of trees defined by:
F(1) = T(1); F(i+1) = the collection of all trees that can be formed by taking three trees in F(i) U S(F(i)) and a new vertex x and joining x by an edge to an arbitrary vertex in each tree, for all $i \geq 1$.

Theorem 3.2. For all $k \geq 1$, a tree has vertex separation at least k if and only if it contains a subtree which is a homeomorphic image of a tree in F(k).

A proof by induction on the size of the tree is straightforward. A corollary follows immediately.

Corollary 3.1. For all $k \geq 1$, a tree has vertex separation k if and only if it contains a subtree that is a homeomorphic image of a tree in F(k) and does not contain a subtree which is a homeomorphic image of a tree in F(k+1).

4. RECOGNIZING GRAPHS WIH FIXED VERTEX SEPARATION

In [11] [12] and [8] dynamic programming algorithms have been described which recognize graphs with fixed bandwidth or cutwidth in polynomial time. However,the application of these earlier algorithms to the problem of vertex separation does not produce a polynomial time algorithm because no vertex degree constraint follows from the fact that a graph has a small vertex separation. For example, the star graph S_n, which is the tree with n leaves and a single internal vertex with degree n, has vertex separation 1 for all n. Consequently, if one defines an equivalence relation on the class of partial layouts as was done in the algorithms just cited, then the number of equivalence classes is not bounded by any polynomial in the number of vertices. A dynamic programming algorithm has been devised for computing vertex separation which uses an equivalence relation for vertex separation k that divides the class of all plausible partial layouts into $O(n^m)$ equivalence classes, where $m = (k^2+5k+2)/2$. Since the method requires that each class be considered O(n) times, the following Theorems can be derived.

Theorem 4.1
For each $k \geq 1$ there is an $O(n^m)$ step algorithm, where $m = (k^2+5k+4)/2$, to decide if a graph with n vertices has vertex separation k.

Since we can expand any graph and the vertex separation of the result is equal to the search number of the original graph, we have the following, which solves an open problem in Meggido et al [2]. The expansion was illustratd in Figure 2.3.

Corollary 4.1
For each $k \geq 1$ there is an $O(n^m)$ step algorithm, where $m = k^2+5k+4$, to decide if a graph G with n vertices has search number k.

The transformation described by Lengauer [6] from the black-white pebble game to the vertex separation problem justifies the following corollary.

Corollary 4.2
For each $k \geq 1$ there is an $O(n^m)$ step algorithm, where $m = (k^2+5k+4)/2$, to decide if a graph G with n vertices has pebble demand k in the progressive black-white pebble game.

REFERENCES

[1] Parsons, T. D, "Pursuit-Evasion in a Graph", in Theory and Application of Graphs, Y. Alavi and D. R. Lick,(eds) Springer-Verlag, Berlin, 1976, pp. 426-441

[2] Meggido, N, Hakimi,S. L, Garey M. R, Johnson, D. S and Papadimitriou, C. H, "The Complexity of Searching a Graph (Preliminary Version)", Proc. IEEE Foundations of Computer Science Symp. (1981), pp. 376-385

[3] Leiserson, C. E, "Area-Efficient Graph Layouts, for VLSI", 21st Annual IEEE Symposium on FOCS, (1980)

[4] Lipton, R. J and Tarjan, R. E, "Applications of a Planar Separator Theorem", SIAM J. Comput. 9,3 (1980), pp. 615-627

[5] Lipton, R. J and Tarjan, R. E, "A Separator Theorem for Planar Graphs", SIAM J. Appl. Math. 36,2 (1979), pp. 177-189

[6] Lengauer, T, "Black-White Pebbles and Graph Separation", SIAM J. Algebraic and Discrete Methods, 1982.

[7] LaPaugh, A. S., "Recontamination does not Help to Search a Graph", Technical Report, Electrical Engineering and Computer Science Department, Princeton University (1983)

[8] Makedon, F. and Sudborough, I. H, "Minimizing Width in Linear Layouts", Proc. 10th International Colloquium on Automata, Languages, and Programming, vol. 154, Lecture Notes in Computer Science, Springer Verlag (1983), pp. 478-490.

[9] Chung, M. J, Makedon, F, Sudborough, I. H and Turner, J, "Polynomial Algorithms for the Min-Cut Linear Arrangement Problem on Degree Restricted Trees", Proc. 23rd Annual IEEE Foundations of Computer Science Symp. (1982), pp. 262-271.

[10] Yannakakis, M., "A Polynomial Algorithm for the Min Cut Linear Arrangement of Trees", to appear in Proc. 24th Annual IEEE Foundations of Computer Science Symposium (1983)

[11] Saxe, J. B, "Dynamic-Programming Algorithms for Recognizing Small Bandwidth Graphs in Polynomial Time", SIAM J. on Algebraic and Discrete Methods, December, 1980

[12] Gurari, E. M and Sudborough, I. H., "Improved Dynamic Programming Algorithms for Bandwidth Minimization and the Min-Cut Linear Arrangement Problem", J. Algorithms, to appear.

FIGURES

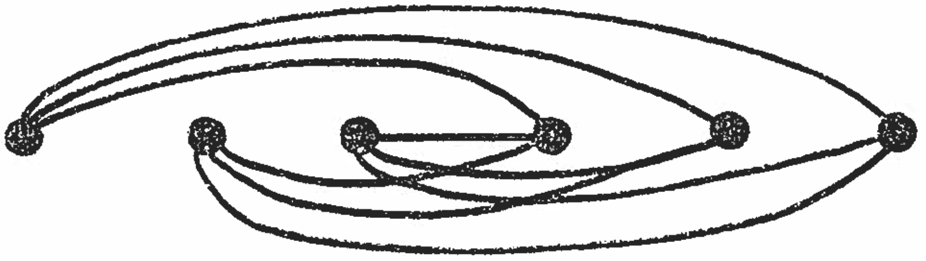
Figure 2.1 The Graph $K_{3,3}$ with vs(G)=3 and s(G)=5

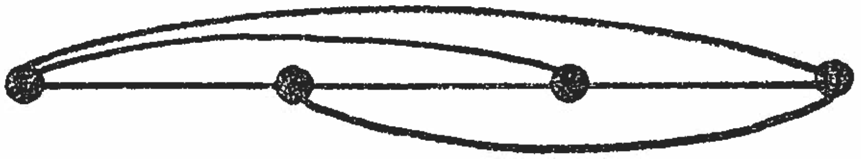
Figure 2.2 A Graph with vs(G)=3 and s(G)=4

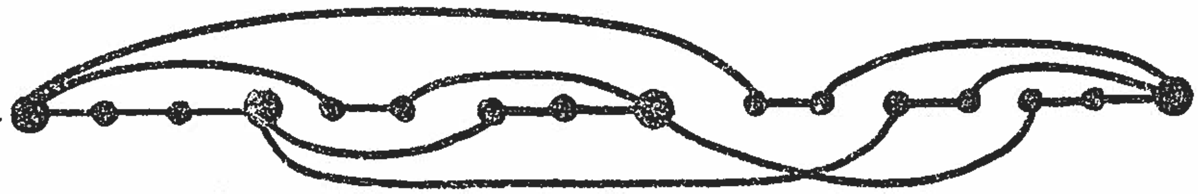
Figure 2.3 The Expanded Graph from 2.2 with vs(G´)=s(G)=4

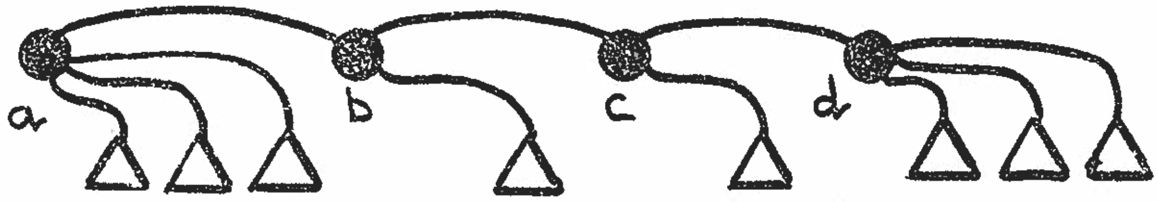

Figure 3.1 Inserting Subtrees in the Chain so that vs(T)=k

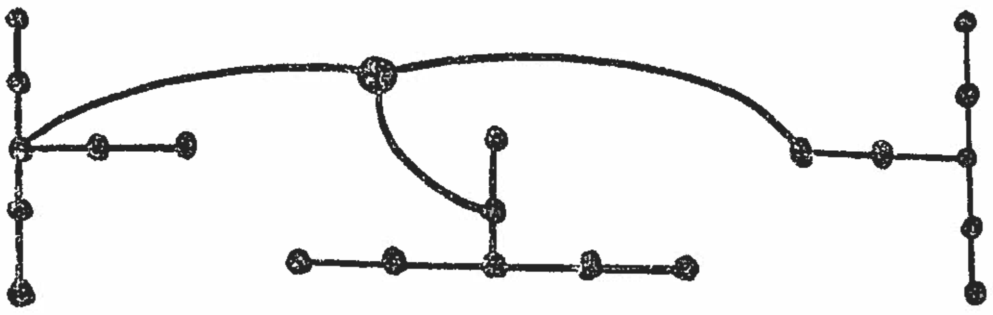
Figure 3.2 One of the Smallest Trees with vs(T)=3