Smallest-Last Ordering and Clustering and Graph Coloring Algorithms

DAVID W. MATULA

Southern Methodist University, Dallas, Texas

AND

LELAND L. BECK

San Diego State University, San Diego, California

Abstract Smallest-last vertex ordering and priority search are utilized to show for any graph $G = (V, E)$ that the set of all connected subgraphs maximal with respect to their minimum degree can be determined in $O(|E| + |V|)$ time and $2|E| + O(|V|)$ space It is further noted that the smallest-last graph coloring algorithm can be implemented in $O(|E| + |V|)$ time, and particularly effective aspects of the resulting coloring are discussed.

Categories and Subject Descriptors· E 2 [**Data Storage Representations**]: *contiguous representations,* F 2 2 [**Analysis of Algorithms and Problem Complexity**]· Nonnumerical Algorithms and Problems; G 2.2 [**Discrete Mathematics**] Graph Theory—*graph algorithms,* H 3 3 [**Information Storage and Retrieval**] Information Search and Retrieval—*clustering*

General Terms Algorithms, Theory

Additional Key Words and Phrases. Adjacency structure, cluster analysis, degree structure, graph coloring, hierarchical cluster analysis, linear-time algorithms, linkage clustering, priority search, smallest-last coloring, smallest-last ordering

1. *Introduction and Summary*

The vertices $v_1, v_2, \ldots, v_n$ of a graph are said to be in smallest-last order whenever v_i has minimum degree in the maximal subgraph on the vertices $v_1, v_2, \ldots, v_i$ for all i. Properties associated with a smallest-last order provide for substantive utilization in a variety of areas, which we illustrate with applications in cluster analysis and graph coloring.

In Section 2 we show that a smallest-last vertex ordering for a graph with vertex set V and edge set E can be determined in time $O(|E| + |V|)$ requiring only $O(|V|)$ additional space, given read-only access to an adjacency structure for the graph; that is, $2|E|$ read-only $+ O(|V|)$ space. We also provide an $O(|E| + |V|)$ time in-place ($O(|V|)$ temporary space) solution to the problem of reordering all adjacency lists of

This research was supported in part by the National Science Foundation under Grants GJ-40487 and DCR 75-10930

Authors' addresses D W Matula, Department of Computer Science, Southern Methodist University, Dallas, TX 75275, L. L Beck, San Diego State University, San Diego, CA 92182.

the adjacency structure in conformity with this (or any other) vertex ordering, should this be desired for subsequent application.

For our main result we utilize smallest-last ordering and priority search in Section 3 to show that the set of all connected subgraphs maximal with respect to their minimum degree can be determined in $O(|E| + |V|)$ time in $2|E|$ read-only + $O(|V|)$ space. The relevance of this result to the various linkage based methods [7, 11, 13, 15, 19–21] of hierarchical cluster analysis is discussed.

The smallest-last coloring algorithm (explicitly or implicitly utilized in [1, 2, 4, 12, 17, 18]) is noted to be implementable in time $O(|E| + |V|)$ in Section 4. For the class of α-uniformly sparse graphs (G is α-uniformly sparse if all its subgraphs have average degree at most α) we in fact obtain an $O(|V|)$ time and space coloring algorithm requiring at most $\lfloor\alpha\rfloor + 1$ colors, which we indicate to be at or near optimal for a variety of interesting classes of graphs. Relations between a smallest-last coloring and the connected subgraphs maximal with respect to their minimum degree are discussed.

2. *Smallest-Last Vertex Ordering*

A graph G is composed of a finite nonvoid vertex set V and an edge set E, where each edge is an unordered pair of vertices of V. Vertices $u, v \in V$ for which uv is an edge are termed adjacent to each other and are the endpoints of the edge. A subgraph H of G, denoted $H \subset G$, is a graph with $V(H) \subset V(G)$ and $E(H) \subset E(G)$. The degree of v in the subgraph H for any $v \in V(H)$, denoted $\deg(v \mid H)$, is the number of vertices of H adjacent to v, and $\delta(H) = \min_{v \in V(H)}\{\deg(v \mid H)\}$ is the minimum degree of H. Relative to a prescribed ordering $v_1, v_2, \ldots, v_n$ of the vertices of G, for each $1 \le i \le n = |V|$ let G_i denote the subgraph on the initial segment $V_i = \{v_1, v_2, \ldots, v_i\}$ of vertices of G, where G_i contains all edges of G both of whose endpoints are in V_i.

Algorithm SL: Smallest-last vertex ordering. Given a graph G on n vertices, the following determines an ordering $v_1, v_2, \ldots, v_n$ of the vertices of G, where $\deg(v_i \mid G_i) = \delta(G_i)$ for $1 \le i \le n$.

SL1. [Initialize.] $j \leftarrow n$, $H \leftarrow G$.
SL2. [Find minimum degree vertex.] Let v_j be a vertex of minimum degree in H.
SL3. [Delete minimum degree vertex.] $H \leftarrow H - v_j$, $j \leftarrow j - 1$.
SL4. [Finished?] If $j \ge 1$, return to step SL2, otherwise terminate with sequence $v_1, v_2, \ldots, v_n$.

Let any ordering $v_1, v_2, \ldots, v_n$ of the vertices of a graph G be a *smallest-last* ordering for G if $\deg(v_i \mid G_i) = \delta(G_i)$ for $1 \le i \le n$.

LEMMA 1. *Algorithm SL determines a smallest-last ordering for any graph G and can be implemented to run in time $O(|E| + |V|)$ in space $2|E| + O(|V|)$.*

PROOF. It is immediate that Algorithm SL produces a smallest-last ordering for G. Thus we need only to show appropriate data structures and procedures to realize the complexity bounds.

We shall represent G in the form of an adjacency structure, where the adjacencies for each vertex are stored in a sequential list, thus requiring $2|E| + O(|V|)$ space. During initialization (step SL1) we shall construct a degree structure composed of a doubly linked list of vertices of degree i for each i, with an array of headers pointing to the i-degree lists. This structure implicitly provides a "bucket sort" of the vertices by degree. This initial degree structure can be constructed by a single pass through the adjacency structure, requiring $O(|E| + |V|)$ time and adding only $O(|V|)$ to the space requirement.

Our implementation of Algorithm SL will not alter the adjacency structure; however, the degree structure will be updated with every vertex deletion to provide the degree structure for the current subgraph H. An auxiliary array will be used to record the current value of $\deg(v \mid H)$ for all v remaining in H. Step SL2 is realized by searching the degree structure for the i-degree list of smallest i which is nonempty and selecting the first vertex v_j from this list. This procedure requires $O(\deg(v_j \mid H))$ time. Step SL3 is realized by first extracting v_j from its degree list; the adjacency list of v_j is then scanned—for every vertex u in this list which is still in H, u is extracted from its i-degree list and inserted into the $(i-1)$-degree list corresponding to its new degree $i-1$ in the subgraph $H - v_j$. This can be done in $O(\deg(v_j \mid G))$ time. Since this implementation requires only $O(|V|)$ space in addition to the space for the adjacency and degree structures, the total space requirement is $2|E| + O(|V|)$. Since $\sum_{j=1}^{|V|} \deg(v_j \mid G) = 2|E|$, it follows that this implementation of Algorithm SL runs in time $O(|E| + |V|)$. □

It is important to note the read-only status of the adjacency structure in the preceding implementation of Algorithm SL. If successive updating of the adjacency structure in reasonable time had also been required, then the need for linked adjacency lists and cross pointers to effect vertex deletion in the adjacency structure would have required a much greater storage demand. Furthermore, the read-only status for the adjacency structure means that Algorithm SL can be executed concurrently with other graph algorithms requiring read-only access to the adjacency structure.

A smallest-last vertex ordering for a graph is useful for a variety of applications where the number, $\deg(v_i \mid G_i)$, of adjacencies of v_i to preceding vertices $v_j, j < i$, must not be too large for any i. It has been shown [10, 12, 13] that the smallest-last orderings minimize $\max_i\{\deg(v_i \mid G_i)\}$ over all vertex orderings. To see this let $\hat{\delta}(G)$ denote the maximum over all subgraphs of G of the minimum degree of the subgraph, that is, $\hat{\delta}(G) = \max_{H \subseteq G}\{\delta(H)\}$. Then note for any smallest-last ordering that $\max_i\{\deg(v_i \mid G_i)\} = \max_i\{\delta(G_i)\} \leq \hat{\delta}(G)$. Now let H^* be a subgraph of G with $\delta(H^*) = \hat{\delta}(G)$. Then for any vertex ordering let j be the smallest index such that H^* is a subgraph of G_j. Then v_j is a vertex of H^* with $\deg(v_j \mid G_j) \geq \deg(v_j \mid H^*) \geq \delta(H^*) = \hat{\delta}(G)$. Hence we conclude that $\max_i\{\deg(v_i \mid G_i)\} \geq \hat{\delta}(G)$ for every vertex ordering with equality for any smallest-last ordering. Of course, $\max_i\{\deg(v_i \mid G_i)\}$ can only be small for relatively sparse graphs, since $\sum_i \deg(v_i \mid G_i) = |E|$ for any vertex ordering, so $\max_i\{\deg(v_i \mid G_i)\} \geq |E|/|V|$ must always hold.

For several important classes of sparse graphs a smallest-last ordering $v_1, v_2, \ldots, v_n$ will have $\deg(v_i \mid G_i)$ uniformly small [10]:

(i) for G a forest, $\max_i\{\deg(v_i \mid G_i)\} \leq 1$;
(ii) for G a planar graph, $\max_i\{\deg(v_i \mid G_i)\} \leq 5$;
(iii) for G an outerplanar graph (i.e., G can be embeddded in the plane so all vertices are on a common face), $\max_i\{\deg(v_i \mid G_i)\} \leq 2$.

To achieve further advantages from a smallest-last ordering for a graph G, it may be desirable to rearrange the entries of each adjacency list in the adjacency structure for G in conformity with the smallest-last ordering. The following algorithm provides an $O(|E| + |V|)$ time in-place solution to the general problem of reordering adjacency lists to conform with any specified vertex ordering.

Algorithm RAL: Reorder adjacency lists. Given an adjacency data structure for a graph G where every adjacency list is stored in sequential memory, the following provides an in-place reordering of every adjacency list in conformity with any specified new ordering $v_1, v_2, \ldots, v_n$.

RAL1. [Compact adjacency lists to upper diagonal form.] For each i, $1 \le i \le n$, pack to the rear of the adjacency list for v_i (in any order) all v_j of the list with $j > i$. (Each edge of G is now considered represented by a single arc $v_i v_j$, where $i < j$. The contents of the initial segment of length $\deg(v_i | G_i)$ of the sequential adjacency list for v_i can now be ignored as it will be overwritten.)

RAL2. [Initialize for reordering.] Establish a pointer p_i to the initial position of the sequential adjacency list for v_i for each i. Then let $i \leftarrow 0$.

RAL3. [Finished?] If $i = n$, stop, otherwise $i \leftarrow i + 1$.

RAL4. [Read adjacency list of v_i, and place v_i in its appropriate new position in all adjacency lists to which it belongs.] For each entry v_{i_j} in the (current) adjacency list of v_i, insert v_i in the position pointed at by p_{i_j} in the adjacency list of v_{i_j}, and let $p_{i_j} \leftarrow p_{i_j} + 1$. Then go to step RAL3. (It is straightforward to show the following by induction on i for $i = 1, 2, \ldots, n$. At the time the adjacency list of v_i is to be read, all adjacency lists will currently contain their adjacencies to v_j for all $j < i$ in their appropriate new order, with their pointers indicating the next position after these initial ordered entries in their respective lists. p_i will then be pointing to the $\deg(v_i | G_i) + 1$ position of the adjacency list of v_i, where we note the $\deg(v_i | G_i) + 1$ through $\deg(v_i | G)$ positions of that list contain all v_j adjacent to v_i with $j > i$ (in some order) by our previous packing in step RAL1, so the reading of the adjacency list v_i identifies each adjacency of v_i in G.)

The correctness of Algorithm RAL follows from the inductive argument in the comment for step RAL4. Implementation in $O(|E| + |V|)$ time is straightforward with temporary additional space $|V| + O(1)$ sufficient for the adjacency list pointers and other needs.

3. *Stratified-Linkage Cluster Analysis*

There are a variety of "linkage"-based cluster analysis methods that are readily interpreted as degree- and connectivity-based procedures on graphs [15]. Several influential formalizations of cluster analysis [7, 21] start with the assumption that the input data is available in the form of a pairwise (dichotomous, ordinal, or possibly real-valued) measure of proximity on a finite collection of objects. A proximity relation is determined by those pairs of objects exceeding a particular threshold level, with the objects and the proximity relation then yielding a "threshold" graph corresponding to that level. The extensively utilized single-linkage clustering method corresponds simply to determining as clusters the components in the sequence of threshold graphs as the threshold level varies over its range, and the alternative complete-linkage method corresponds to identifying a hierarchy of cliques over the threshold parameter range. Both single-linkage and complete-linkage methods are efficiently computable even by hand, which explains their early popularity. However, single-linkage can yield weakly related clusters, and complete-linkage is not always well defined.

To avoid these pitfalls, Sneath [20] proposed k-linkage clustering, which requires that each object of a cluster be related to at least k other objects of the cluster. This has been further refined [15] to "strong" k-linkage in correspondence to the determination of maximal k-edge connected subgraphs of the threshold graphs [13, 16], and standard (weak) k-linkage in correspondence to the more tractable determination of maximal connected subgraphs of minimum degree k, investigated in [11, 13, 15, 19]. For dichotomous proximity data we obtain a single graph rather than a family of threshold graphs. In this case a hierarchy of clusters may be obtained [15] by determining the maximal connected subgraphs of minimum degree k for all k, with successive levels of k giving the stratification in the hierarchy. An efficient algorithm for this stratified-linkage clustering is now developed, employing a smallest-last vertex ordering in the associated graph.

In view of these cluster analysis applications we term a subgraph L of the graph G that is connected and maximal with respect to its minimum degree a *linkage* of G. A linkage where every vertex has degree at least k is a *k-linkage*. The linkage level $l(v)$ for the vertex v of G is the largest value of k for which some k-linkage contains v, so equivalently,

$$l(v) = \max_{\substack{H \subset G \\ v \in V(H)}} \{\delta(H)\}. \tag{1}$$

The following facts about the linkages and linkage level function of a graph and their relation to any smallest-last vertex ordering for the graph are readily proved from the definitions.

Assume $v_1, v_2, \ldots, v_n$ is a smallest-last vertex ordering for the graph G.

(F1) $l(v_i) = \max_{j \geq i}\{\deg(v_j \mid G_j)\}$.
(F2) A k-linkage is a connected subgraph maximal with respect to the property $l(v) \geq k$ for all vertices v of the subgraph.
(F3) If $\deg(v_i \mid G_i) = k$ and $\deg(v_j \mid G_j) < k$ for $j > i$, then the components of G_i are the k-linkages of G.
(F4) The linkages form a nested hierarchical structure in that any two distinct linkages of the same minimum degree are disjoint, and any two linkages having a nonvoid intersection must have one linkage as a subgraph of the other.

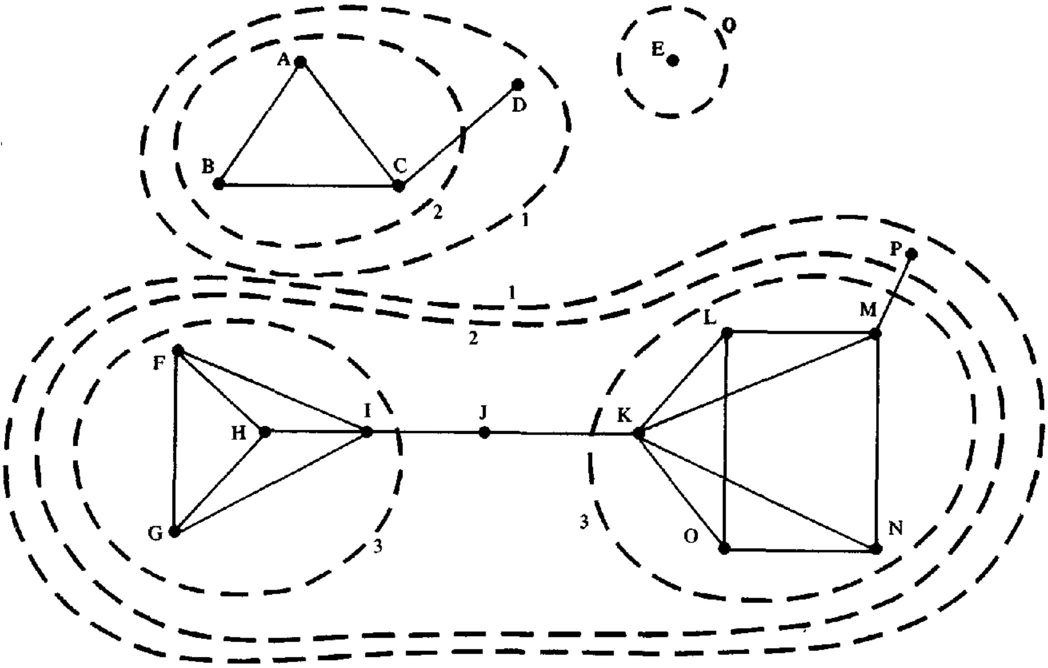
The linkages of the graph in Figure 1 are enclosed in nested dashed lines illustrating the hierarchical structure. A table of values of $\deg(v_i \mid G_i)$ and $l(v_i)$ relevant to a smallest-last ordering of G is also shown in Figure 1 for illustration of the properties (F1)–(F4).

From property (F1) it is evident that adding the computation "max$l \leftarrow 0$" to step SL1 and the computations "max$l \leftarrow \max\{\max l, \deg(v_j \mid H)\}$" and "$l_j \leftarrow \max l$" to step SL2 in Algorithm SL allows Algorithm SL to also compute the linkage level of each vertex within the same time and space complexity order bounds noted in Lemma 1. We now show that this linkage level function can be utilized to determine the priority for a search of the graph which will determine the full linkage hierarchy.

A *search* of the graph G is an ordering $v_1, v_2, \ldots, v_n$ of the vertices of G where v_i is adjacent to v_j for some $j < i$ whenever G_{i-1} is not the union of components of G. The search is said to *visit* the vertex v_i at stage i, and $R_i = \{w \mid vw \in E(G), v \in V_i, w \notin V_i\}$ is the set *reached* (but not visited) by stage i. Let a *priority* at stage i, denoted by $\mathrm{pri}_i(w)$, be associated with each reached vertex $w \in R_i$, where without loss of generality we assume the priority is integer valued and $0 \leq \mathrm{pri}_i(w) \leq |V(G)| - 1$ for all $w \in R_i$ for all i. A *priority search* then has v_{i+1} chosen from the maximum priority members of R_i whenever R_i is nonvoid.

LEMMA 2. *Let H_k be a connected subgraph of G maximal with respect to the property $f(v) \geq k$ for all $v \in V(H_k)$ for some integer-valued function f having $0 \leq f(v) \leq |V(G)| - 1$ for all $v \in V(G)$. Let $v_1, v_2, \ldots, v_n$ be a priority search of G with priority for each stage i given by*

$$pri_i(w) = \max_{\substack{j \leq i \\ v_j w \in E(G)}} \{min\{f(v_j), f(w)\}\} \qquad for\ all \quad w \in R_i. \tag{2}$$



	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}	v_{12}	v_{13}	v_{14}	v_{15}	v_{16}
smallest-last ordering	K	L	M	N	O	F	G	H	I	J	A	B	C	P	D	E
deg $(v_i \mid G_i)$	0	1	2	2	3	0	1	2	3	2	0	1	2	1	1	0
$\ell(v_i)$	3	3	3	3	3	3	3	3	3	2	2	2	2	1	1	0

FIG. 1. Illustration of the connected subgraphs, termed linkages, that are maximal with respect to their minimum degree. The illustrated smallest-last ordering and linkage level values are readily verified by a right-to-left computation.

Then for some $q \geq 0$ the segment $v_{q+1}, v_{q+2}, \ldots, v_{q+|V(H_k)|}$ is a priority search of H_k with respect to the same priority function.

PROOF. The result is immediate for $|V(H_k)| = 1$, so assume $|V(H_k)| \geq 2$. Let v_{q+1} be the first vertex of H_k visited by the priority search $v_1, v_2, \ldots, v_n$ of G. By induction assume $v_j \in V(H_k)$ for $q + 1 \leq j \leq m$ for a given m, where $q + 1 \leq m \leq q + |V(H_k)| - 1$. Now $R_m \cap V(H_k)$ must contain some vertex w for which $\mathrm{pri}_m(w) \geq k$; so then $f(v_{m+1}) \geq \mathrm{pri}_m(v_{m+1}) \geq \mathrm{pri}_m(w) \geq k$. If $v_{m+1} \notin R_q$, then v_{m+1} is adjacent to some v_i, $q + 1 \leq i \leq m$, hence $v_{m+1} \in H_k$. Suppose $v_{m+1} \in R_q$. Now $\mathrm{pri}_q(v_{q+1}) < k$, since otherwise $v_i v_{q+1} \in E(G)$ and $f(v_i), f(v_{q+1}) \geq k$ for some $i \leq q$ would follow, and also then $v_i \in H_k$, contradicting the choice of index $q + 1$. Then $\mathrm{pri}_q(v_{m+1}) \leq \mathrm{pri}_q(v_{q+1}) < k$, hence $\mathrm{pri}_q(v_{m+1}) < \mathrm{pri}_m(v_{m+1})$. So v_{m+1} is adjacent to some v_i, $q + 1 \leq i \leq m$, and again $v_{m+1} \in H_k$. By the induction assumption we then obtain $v_j \in H_k$ for $q + 1 \leq j \leq q + |V(H_k)|$. Since $\mathrm{pri}_q(v) < k$ for all $v \in R_q$, each vertex v_j for $q + 2 \leq j \leq q + |V(H_k)|$ is chosen consistent with its priority restricted to the subgraph $V(H_k)$, so $v_{q+1}, v_{q+2}, \ldots, v_{q+|V(H_k)|}$ is a priority search of H_k. □

Let any connected subgraph H_k of G maximal with respect to the property $f(v) \geq k$ for all $v \in H_k$ (as in Lemma 2) be termed a *level component* of G. The relation

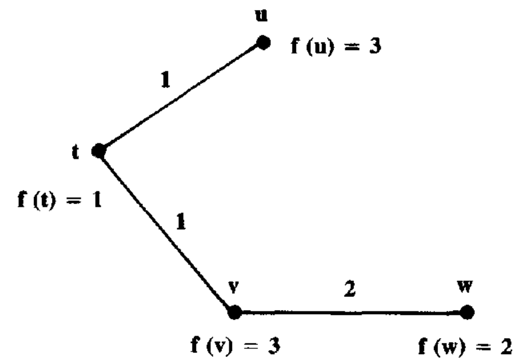


FIG. 2. A graph with edge labels $f(xy) = \min\{f(x), f(y)\}$ for each edge xy, with priority search t, v, w, u.

between the level components and the priority search in Lemma 2 becomes clearer if we extend the function f to the edges of G by defining $f(vu) = \min\{f(v), f(u)\}$ for $vu \in E(G)$. Then $f(vu) \geq k$ for all $vu \in E(H_k)$; however, an edge vu with $v \notin H_k$, $u \in H_k$ has $f(vu) < k$. Thus the priority criterion of (2) simply causes the search to traverse edges of maximum f value to visit adjacent vertices and thereby always keeps the search within a level component until all of its vertices are visited.

The graph of Figure 2 illustrates why it is not sufficient simply to use $\mathrm{pri}_i(w) = f(w)$ in Lemma 2. Starting the priority search with t, v yields $R_2 = \{u, w\}$. Although $f(u) > f(w)$, the vertex sequence t, v, u, w would not have the level component on v, w correspond to consecutive vertices. The f values on the edges properly guide the search yielding t, v, w, u, where all level components then correspond to vertex segments. Furthermore, although $f(v), f(w), f(u) \geq 2$ for the consecutive subsequence v, w, u, the fact that u is selected for visitation with priority level 1 (i.e., $f(tu) = 1$) indicates that u is not in the level component H_2 containing v, w.

Let $v_1, v_2, \ldots, v_n$ be a priority search of G, as determined in Lemma 2. It is then evident that all segments $v_{j+1}, v_{j+2}, \ldots, v_{j+m}$ corresponding to level components of G can be determined by applying the following rules as we traverse the search $v_1, v_2, \ldots, v_n$:

(i) When $f(v_i) > \mathrm{pri}_{i-1}(v_i)$, then the search is entering, at v_i, level components H_k for all k in the range $\mathrm{pri}_{i-1}(v_i) < k \leq f(v_i)$.
(ii) When $f(v_i) > \mathrm{pri}_i(v_{i+1})$, then the search is exiting, at v_i, level components H_k for all k in the range $\mathrm{pri}_i(v_{i+1}) < k \leq f(v_i)$.
(iii) When the reached set R_i is void, then the search is exiting, at v_i, all level components currently being traversed by the search and is entering, at v_{i+1}, level components H_k for all $k \leq f(v_{i+1})$.

Level components within a search resulting from Lemma 2 are conveniently delimited by brackets, so that $k + 1$ levels of nested brackets then denote a level component H_k. For the graph of Figure 2 we obtain [[*t*, [[*v*], *w*], [[*u*]]]]. For the graph of Figure 1 (f given by the linkage level) we obtain [[[*A*, *B*, *C*], *D*]], [*E*], [[[[*F*, *G*, *H*, *I*], *J*, [*K*, *L*, *M*, *N*, *O*]], *P*]] which exhibits, by property (F2), the nested hierarchy of all linkages of the graph. These observations are summarized in the following algorithm for determining all level components of a graph.

Algorithm LCPS: *Level component priority search*. Given a graph G on n vertices and an integer-valued function f over the vertices with range $[0, n - 1]$, the following determines a priority search with interspersed paired brackets such that the vertices $v_{q+1}, v_{q+2}, \ldots, v_{q+m}$ enclosed by paired brackets at depth $k + 1$ are the vertices of a level component H_k of G, with each level component of G so determined.

LCPS1. [Initialize.] $i \leftarrow 0$, $k \leftarrow -1$, $R \leftarrow \varnothing$.

LCPS2. [Finished?] If $i = n$, insert $k + 1$ close brackets after v_n and stop, otherwise let $i \leftarrow i + 1$.

LCPS3. [Passage to another component of G.] If $R = \varnothing$, choose v_i to be any member of $V - \{v_1, v_2, \ldots, v_{i-1}\}$. Then insert $k + 1$ close brackets followed by $f(v_i) + 1$ open brackets before v_i, let $k \leftarrow f(v_i)$, $R \leftarrow \{w \mid v_i w \in E\}$, $p \leftarrow \max\{\min\{k, f(w)\} \mid w \in R\}$, and go to step LCPS2. (All level components are bounded within a component of G.)

LCPS4. [Maximum priority selection from the reached set R.] Let v_i be chosen from R with priority p, the maximum priority. (Note that the priority of w in R is always $\max\{\min\{f(v_j), f(w)\} \mid j < i, v_j w \in E\}$. Implicitly the search traverses an edge $v_j v_i$, $j < i$, with $f(v_j v_i) = \min\{f(v_j), f(v_i)\} = p$, and thus continues in level components H_l for all $l \leq p$.)

LCPS5. [Adjust component depth level of search.] Insert $k - p$ close brackets followed by $f(v_i) - p$ open brackets before v_i, and let $k \leftarrow f(v_i)$.

LCPS6. [Update reached set R.] $R \leftarrow R - v_i$, $R \leftarrow R \cup \{w \mid v_i w \in E, w \neq v_j \text{ for } j < i\}$, compute priorities in R by formula (2), assigning the new maximum priority to p if $R \neq \varnothing$, and go to step LCPS2.

It is immediate from Lemma 2 and the arguments preceding the statement of Algorithm LCPS that the algorithm determines all level components for any graph G. An implementation achieving time complexity $O(|E| + |V|)$ may not always be possible, owing to the difficulty of maintaining an appropriate priority queue for R. We now show that with the function f given by the linkage level of equation (1), an $O(|E| + |V|)$ implementation is possible.

THEOREM 3. *For a graph G with f the linkage level function $f(v) = \max\{\delta(H) \mid v \in V(H), H \subset G\}$ for $v \in V(G)$, Algorithm LCPS determines all linkages (connected subgraphs maximal with respect to their minimum degree) and can be implemented in time $O(|E| + |V|)$ in space $2|E| + O(|V|)$.*

PROOF. Algorithm LCPS determines all level components with regard to the linkage level function, and by property (F2) these are precisely the linkages of G. Our implementation of Algorithm LCPS will have an adjacency structure for G with the adjacencies of v stored in a sequential list for each $v \in V(G)$. The priority queue for R will have vertices of priority j in a doubly linked list for each j, $0 \leq j \leq |V| - 1$, with an array of headers and a variable p giving the largest j for which the j-priority list is nonvoid (or a flag value when $R = \varnothing$). We also maintain a list of current priorities for all $w \in R$.

Since a k-linkage must contain at least $k + 1$ vertices, the jth close bracket can not occur before v_j. Thus there are at most $2|V|$ brackets inserted into the search. It follows that the total time in steps LCPS2, LCPS3, and LCPS5 is $O(|V|)$. Step LCPS4 requires only constant time on each entry, since the computation updating the priority queue for R and the value of p occurs in steps LCPS3 and LCPS6.

To implement step LCPS6, v_i is deleted from its p-priority list, and priority lists for $p, p - 1, p - 2, \ldots$ are inspected, reassigning p the first value corresponding to a nonvoid list. Then for each w in the adjacency list of v_i, w is inserted into the $(j = \min\{f(v_i), f(w)\})$-priority list if w was not already in a list of this or higher priority, p is reassigned the value $\max\{p, j\}$, and w is deleted from its previous priority list if it had been present at lower priority. The priority at which v_i was chosen was at most $\deg(v_i \mid G)$, so this step requires time $O(\deg(v_i \mid G))$. Thus this implementation requires time $O(|E| + |V|)$ and space $2|E| + O(|V|)$. □

The preceding implementation required read-only access to the adjacency structure of G. Thus our implementations of both Algorithms SL and LCPS allow for the determination of all connected subgraphs maximal with respect to their minimum degree in time $O(|E| + |V|)$ with additional space $O(|V|)$, given read-only access to the adjacency structure for G.

4. *Graph Coloring*

A k-coloring of the graph G is an assignment of an integer color value $1 \leq c(v) \leq k$ to each vertex $v \in V$ such that adjacent vertices receive different color values. The chromatic number, $\chi(G)$, is the minimum k for which G has a k-coloring. An independent set is a set of mutually nonadjacent vertices of G. The set of vertices of each particular color value determined by a k-coloring of G is then an independent set, and an appropriate choice of $\chi(G)$ independent sets is sufficient to cover all vertices of G.

The determination of a $\chi(G)$-coloring for an arbitrary graph G is an NP-complete problem [8]. Garey and Johnson [3] have shown that even the determination of a $(2 - \epsilon)\chi(G)$-coloring of an arbitrary graph G for any $\epsilon > 0$ is an NP-complete problem, which casts doubt on the likelihood of any efficient coloring algorithm guaranteeing near optimal behavior in the worst case. Regarding average-case analysis, a simple algorithm is shown to yield a $(2 + \epsilon)\chi(G)$-coloring of G for almost all graphs G.

For the ordering $v_1, v_2, \ldots, v_n$ of the vertices of G, *a sequential coloring* [17] is a k-coloring of G, where

$$c(v_i) = \min\{m \mid 1 \leq m \neq c(v_j) \text{ for } v_j \text{ adjacent to } v_i, j < i\}, \tag{3}$$

and $k = \max\{c(v_i) \mid 1 \leq i \leq n\}$. A sequential coloring is readily determined in time $O(|E| + |V|)$ by assigning colors to the vertices in the order $v_1, v_2, \ldots, v_n$ so as to satisfy (3). Results on the coloring of random graphs [5, 9] show that a procedure determining a sequential coloring for a random ordering of the vertices provides an $O(|E| + |V|)$ coloring algorithm which, for any $\epsilon > 0$, will yield a $(2 + \epsilon)\chi(G)$-coloring of G for all but a vanishingly small fraction of graphs as $|V| \to \infty$.

Algorithm SLC shall denote the procedure for determining a sequential coloring corresponding to a smallest-last vertex ordering. This procedure was utilized as a proof technique by Matula [12] and independently by Finck and Sachs [2], and was formulated as the smallest-last coloring algorithm by Matula et al. [17]. The following is immediate from Lemma 1 and property (F1) of the linkage level function given by (1).

LEMMA 4. *Algorithm SLC (smallest-last coloring) can be implemented to generate a k-coloring of any graph G in time* $O(|E| + |V|)$, *where*

$$c(v) \leq 1 + \max\{\delta(H) \mid H \subset G, v \in V(H)\} \quad \text{for} \quad v \in V,$$
$$k = \max\{c(v) \mid v \in V\} \leq 1 + \hat{\delta}(G).$$

Consider the bracketed priority search given by Algorithm LCPS of the previous section with f the linkage level function. The implication of Lemma 4 is that, locally, the color value assigned any vertex is never greater than the depth level of the innermost brackets enclosing the vertex, and, at the same time, the maximum color value satisfies the known chromatic number bound $1 + \hat{\delta}(G)$ [22]. Applying Algorithm SLC to the smallest-last ordering for the graph in Figure 1 and appending the resulting color values as subscripts of the vertices reordered to priority search order, we obtain

$$[[[A_1, B_2, C_3], D_1]], [E_1], [[[[F_1, G_2, H_3, I_4], J_2, [K_1, L_2, M_3, N_2, O_3]], P_1]].$$

Empirical results [17] suggest that the average-case behavior of Algorithm SLC is only slightly better than that of an arbitrary sequential coloring in terms of the number of colors required. In common with other efficient heuristic coloring algorithms proposed in the literature, Algorithm SLC also can take arbitrarily more than $\chi(G)$ colors for certain contrived graphs. A primary virtue of Algorithm SLC is in its provably good performance on certain classes of sparse graphs. From our observations regarding smallest-last orderings for certain sparse graphs in Section 2, it follows that Algorithm SLC must always generate a $\chi(G)$-coloring whenever G is a forest or outerplanar graph. Furthermore, Algorithm SLC will color any planar graph in at most six colors (and can be enhanced [18] to provide a linear time 5-coloring of any planar graph).

Let the graph G be termed *uniformly α-sparse* if every subgraph of G has average degree at most α, that is,

$$\frac{|E(H)|}{|V(H)|} \leq \frac{\alpha}{2} \qquad \text{for all} \quad H \subset G.$$

LEMMA 5. *For any fixed α, Algorithm SLC determines a k-coloring of any α-sparse graph G in time and space $O(|V|)$ for $k = \lfloor \alpha \rfloor + 1$.*

PROOF. Since α is fixed, $|E| = O(|V|)$, so that time and space bounds linear in the number of vertices follow from Lemma 1. Since the minimum degree is at most equal to the average degree, $\hat{\delta}(G) \leq \alpha$ for any uniformly α-sparse graph, and the result follows from Lemma 4. □

The class of graphs representable as the union of at most θ planar graphs has been studied in the literature and has appeared in applications to printed circuit testing, for example, [4]. Any graph G_θ representable as the union of θ planar graphs is uniformly $(6\theta - \epsilon)$-sparse for suitably small $\epsilon > 0$ and so by Lemma 5 will be colored in at most 6θ colors in $O(|V|)$ time by Algorithm SLC. This result is nearly optimal for the class G_θ of graphs representable as the union of θ planar graphs, since it is known [6, p. 120] that $\chi(G) \geq 6\theta - 3$ for some graph $G \in G_\theta$.

The graphs embeddable on a surface of genus γ have an asymptotically sparse property from which we obtain the following.

LEMMA 6. *For any fixed integer $\gamma \geq 1$, application of Algorithm SLC to any graph G embeddable on the surface of genus γ will determine a set V^* of at most $12(\gamma - 1)$ vertices and a coloring of $G - V^*$ in at most seven colors in time $O(|V|)$.*

PROOF. From Euler's formula the average degree of any j-vertex graph embeddable on a surface of genus γ is at most $6 + 12(\gamma - 1)/j$. Let $v_1, v_2, \ldots, v_n$ be a smallest-last ordering of a graph G which can be embedded on a surface of genus γ. Then G_i is also embeddable on the surface of genus γ for each i, so

$$\deg(v_i \mid G_i) = \delta(G_i) \leq 6 + \frac{12(\gamma - 1)}{i} \qquad \text{for every} \quad i.$$

Hence, $\deg(v_i \mid G_i) \leq 6$ for all $i > 12(\gamma - 1)$, and the coloring yields $c(v_i) \leq 7$ for $i > 12(\gamma - 1)$. For the class of graphs embeddable on a surface of genus γ we have $|E| = O(|V|)$, so the time bound follows from Lemma 1. □

Denote by I_1 the set of vertices colored with color value 1 upon an application of Algorithm SLC to an arbitrary graph G. Every vertex not in I_1 is adjacent to some vertex in I_1 (otherwise it would have been colored 1), so every vertex of $G - I_1$ has smaller degree in $G - I_1$ than in G. It may also be shown by extension of an argument in [14] that every vertex in $G - I_1$ has a smaller linkage level in $G - I_1$ than in G.

Thus Algorithm SLC identifies a vertex set I_1 whose removal uniformly weakens the linkage structure on all of G, and not simply on an isolated subgraph H for which $\delta(H)$ might be relatively high.

From the lemmas and observations of this section it is evident that the coloring obtained by application of the smallest-last coloring algorithm has considerable structure inherited from the graph theoretic properties of a smallest-last ordering. Since this coloring is obtainable in time $O(|E| + |V|)$, Algorithm SLC is both tractable and informative for a wide range of applications of graph coloring.

ACKNOWLEDGMENT. We would like to thank J. F. Feld for the development of some programs and numerical tests of Algorithms SL and LCPS.

REFERENCES

1. BRÉLAZ, D. New methods to color the vertices of a graph *Commun. ACM 22*, 4 (Apr. 1979), 251–256.
2. FINCK, H.J, AND SACHS, H. Uber eine von H. S Wilf angegebene Schrenke fur de chromatische Zahl endlicher Graphen. *Math Nachr. 39* (1969), 373–386
3. GAREY, M.R., AND JOHNSON, D S. The complexity of near-optimal graph coloring. *J ACM 23*, 1 (Jan. 1976), 43–49.
4. GAREY, M R., JOHNSON, D.S, AND SO, H C. An application of graph coloring to printed circuit testing *IEEE Circ. Syst 23* (1976), 591–599
5 GRIMMETT, G R., AND MCDIARMID, C.J.H. On coloring random graphs. *Math. Proc. Camb. Phil Soc. 77* (1975), 313–324.
6 HARARY, F. *Graph Theory*. Addison-Wesley, Reading, Mass, 1969.
7. JARDINE, N., AND SIBSON, R. *Mathematical Taxonomy*. Wiley, London, 1971.
8 KARP, R.M. Reducability among combinatorial problems. In *Complexity of Computer Computations*, R E. Miller and J.W. Thatcher, Eds., Plenum, New York, 1972, pp. 85–103.
9. KARP, R M. The fast approximate solution of hard combinatorial problems. Proc. 6th Southeastern Conf. on Combinatorics, Graph Theory and Computing, Utilities Mathematica, Winnipeg, 1975, pp 15–31
10 LICK, D.R, AND WHITE, A T. *k*-degenerate graphs *Canad J. Math. 22* (1970), 1082–1096.
11. LING, R.F. On the theory and construction of *k*-clusters. *Comput. J. 15* (1972), 326–332.
12. MATULA, D.W A min-max theorem for graphs with application to graph coloring. *SIAM Rev. 10* (1968), 481–482.
13 MATULA, D.W *k*-components, clusters and slicings in graphs. *SIAM J. Appl. Math. 22* (1972), 459–480.
14 MATULA, D.W Bounded color functions on graphs. *Networks 2* (1972), 29–44.
15 MATULA, D.W Graph theoretic techniques for cluster analysis algorithms. In *Classification and Clustering*, J. Van Ryzin, Ed., Academic Press, New York, 1977, pp. 95–129.
16. MATULA, D W. Subgraph connectivity numbers of a graph In *Theory and Applications of Graphs*, Lecture Notes in Mathematics 642, Y Alavi and D R. Lick, Eds, Springer-Verlag, New York, 1978, pp 371–383.
17. MATULA, D W., MARBLE, G, AND ISAACSON, J D. Graph coloring algorithms. In *Graph Theory and Computing*, R C Read, Ed., Academic Press, New York, 1972, pp. 109–122.
18 MATULA, D.W., SHILOACH, Y., AND TARJAN, R.E. Two linear-time algorithms for five-coloring a planar graph. Tech Rep STAN-CS-80-830, Computer Science Dep., Stanford Univ., Stanford, Calif., 1980
19 SLATER, P B. Graph-theoretic clustering of transaction flows. An application to the 1967 United States Interindustrial Transactions Table. Regional Research Institute Report, Dep. of Statistics and Computer Science, West Virginia Univ, 1974.
20 Sneath, P H.A. A comparison of different clustering methods as applied to randomly-spaced points. *Classification Soc. Bull. 1* (1966), 2–18.
21 SNEATH, P.H A., AND SOKAL, R.R *Numerical Taxonomy*. Freeman, San Francisco, 1973.
22 SZEKERES, G., AND WILF, H.S. An inequality for the chromatic number of a graph. *J Combinatorial Theory 4* (1968), 1–3.

RECEIVED JULY 1981, REVISED AUGUST 1982; ACCEPTED SEPTEMBER 1982