A UNIFIED VIEW OF GRAPH SEARCHING*

DEREK G. CORNEIL AND RICHARD KRUEGER†

Abstract. Graph searching is perhaps one of the simplest and most widely used tools in graph algorithms. Despite this, few theoretical results are known about the vertex orderings that can be produced by a specific search algorithm. A simple characterizing property, such as is known for LexBFS, can aid greatly in devising algorithms, writing proofs of correctness, and showing impossibility results. This paper unifies our view of graph search algorithms by showing simple, closely related characterizations of various well-known search paradigms, including BFS and DFS. Furthermore, these characterizations naturally lead to other search paradigms, namely Maximal Neighborhood Search and LexDFS.

Key words. graph searching, graph traversal, depth-first search, breadth-first search, LexBFS

AMS subject classifications. 05C85, 68R10, 68W05

1. Introduction.

Graph searching is fundamental. Most graph algorithms employ some mechanism for systematically visiting all vertices and edges in the given graph. After choosing an initial vertex, a search of a connected graph visits each of the vertices and edges of the graph such that a new vertex is visited only if it is adjacent to some previously visited vertex. At any point there may be several vertices that may possibly be visited next; this generic graph search does not specify which vertex to choose. The *breadth-first search* (BFS) and *depth-first search* (DFS) algorithms are the traditional strategies to choose the next vertex to visit (see [7] for a recent overview of these searches). BFS traverses an untraversed edge incident with the least recently visited vertex, whereas DFS traverses an untraversed edge incident with the most recently visited vertex. Both these searches were used in the 19th century to solve maze traversals. More recently, DFS, as popularized by Tarjan [12], has been used for such problems as graph connectivity, planarity, topological ordering and strong connectivity of digraphs. Applications of BFS include shortest path problems, network flows, and recognition of various graph classes.

In the mid 1970s, Rose, Tarjan and Lueker [11] introduced a variant of BFS for determining perfect elimination orderings in chordal graphs. Their *lexicographic breadth-first search* (LexBFS) has garnered much attention recently and has become an important search algorithm used for recognition of various graph families, diameter approximation, and finding key structures in certain graph classes [3]. One of the key points regarding LexBFS, and one of the reasons for its usefulness, is that there is a very simple characterization of LexBFS-orderings. This characterization is critical to the proofs of correctness of various algorithms. In light of the power of such a tool, it is somewhat surprising that the traditional searches of BFS and DFS are not known to have similar characterizations. Other than LexBFS, only the *maximum cardinality search* (MCS) of Tarjan and Yannakakis [13] has a known characterization [1].

To provide a common underpinning for viewing graph searches, we examine what is meant by a "generic graph search." We describe a graph search similarly to Tarjan [12]. Given a possibly disconnected graph G whose vertices are initially unexplored, we start from some vertex s of G and choose one of its edges to follow.

*This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Ontario Graduate Scholarship Program.

†Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G4 ({dgc,krueger}@cs.toronto.edu).

Traversing this edge leads to a new vertex. We continue, at each step selecting an untraversed edge leading from a vertex already reached. Traversing this edge leads either to a new vertex or to one previously reached. Whenever we run out of edges leading from old vertices, we choose some new unreached vertex and begin again from there. Each edge is traversed exactly once, and we eventually reach all vertices in the graph. How we choose amongst all the edges we may next traverse determines the type of search we perform (i.e., whether it is a BFS or a DFS). The order in which we reach the vertices exhibits very different properties depending on this selection rule.

From a characterization point of view, we concentrate on the following situation. Suppose in the ordering of vertices we have three vertices, a, b and c, where a was reached before b, which was reached before c, and we have an edge ac present in our graph, but the edge ab is absent (see Figure 1.1). The question is "how could vertex b have been chosen before vertex c by our search?" With what we know so far about this situation, it seems that c should have been chosen before b. What can we infer about the graph such that the search could have chosen b before c?

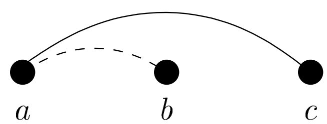


FIG. 1.1. *A curious triple of vertices in a search ordering.*

In the case of LexBFS, the answer is known. There must exist another vertex d chosen earlier than a such that db is an edge, but dc is not an edge of G [8]. In fact, Brandstädt, Dragan and Nicolai [1] showed that this condition is a characterization of a vertex ordering being generated by LexBFS.

Experience with LexBFS has shown that such a characterizing property is instrumental in the applicability of the search to various problems. For example, the LexBFS characterization is fundamental in proving that the reverse of a LexBFS-ordering of a chordal graph is a perfect elimination ordering [11]. Furthermore, many of the recent LexBFS-based algorithms use multiple sweeps of LexBFS to reveal structure of the given graph (see [3] for several examples). Each subsequent search uses information from the previous sweep(s) to break ties of which vertex to visit next. The related proofs of correctness rely heavily on the LexBFS characterization. The characterization can also be used to show that there is some LexBFS-ordering that certifies a graph as being cocomparability, even though we don't know how to find it efficiently [4].

In light of this simple characterization result for LexBFS (and similar results for MCS) and recognizing its usefulness, it is somewhat surprising that it appears that this question has never been asked regarding the more traditional search paradigms of BFS and DFS. In §2 we further explore the question "how could b have been chosen before c by a graph search?", presenting answers to this question for each of the known search paradigms and showing that those answers in fact characterize the vertex orderings realizable by the graph search. We will also introduce two new search paradigms, a depth-first version of LexBFS we call LexDFS, and a generalization we call maximal neighborhood search (MNS). To illustrate the usefulness of these new characterizations, we examine in §3 which searches of a given graph G are valid searches of powers of G. In particular, we would like to obtain a search-ordering of G^k without the necessity of computing G^k (i.e., in time linear in the size of G). We will conclude with some remarks about the impact of these new characterizations, and their extension to other areas.

1.1. Notation and Definitions. We now define some notation to facilitate our discussion. We let $G = (V, E)$ be a graph on vertex set V and edge set E. We will assume, for simplicity, that all graphs in this paper are connected and undirected; extension of our results to disconnected graphs is straightforward. Let $n = |V|$. We define $\sigma = (v_1, \ldots, v_n)$ to be a linear ordering of V, which will typically be the left-to-right order in which we number the vertices in a search. We let $\sigma_i = (v_1, \ldots, v_i)$ be a prefix of the ordering σ for $1 \leq i \leq n$. We use the notation $\sigma(i) = v_i$ and $\sigma^{-1}(v_i) = i$ to refer to a specific vertex in σ. The ordering σ implicitly imposes a total order on V; we say $v_1 <_\sigma v_2 <_\sigma \cdots <_\sigma v_n$, and omit the subscript σ when the ordering is clear. We let $G_i = G[v_1, \ldots, v_i]$ be the subgraph of G induced on a prefix of σ, and let $N_H(v)$ be the neighborhood of v in a subgraph H of G. When refering to searches, $s = v_1$ is the start vertex.

A graph is said to be *chordal* if every cycle of length at least 4 has at least one chord. A vertex is *simplicial* in a graph G if its neighborhood is a clique in G. An ordering $(v_1, \ldots, v_n)$ of V is a *perfect elimination ordering* of G if v_i is simplicial in G_i. A graph is chordal if and only if it has a perfect elimination ordering [6]. A graph is *bipartite* if its vertex set can be partitioned into two disjoint sets X, Y such that all edges are of the form xy for some $x \in X$ and $y \in Y$. A graph is *chordal bipartite* if it is bipartite and every cycle of length at least 6 has a chord. The *complement* of a graph $G = (V, E)$, written $\overline{G} = (V, \overline{E})$, is defined by $\overline{E} = \{xy : x, y \in V \text{ with } x \neq y \text{ and } xy \notin E\}$. The kth *power* of a graph G, written $G^k = (V, E^k)$, for some positive integer k, is obtained from G by including an edge between every pair of vertices that are of distance at most k in G (hence G^1 is precisely G itself).

We draw particular attention to the fact that the search algorithms in this paper generate vertex orderings from left to right. In many applications it may be useful to reverse the ordering generated by search, and some of the LexBFS literature considers this reversal as part of the search algorithm.

2. Characterizing Graph Search.

2.1. Generic Search. We begin by formalizing the generic search idea, similar to Tarjan's description in [12], and present it as Algorithm 1. We use a generic "set" data structure S to store the candidate vertices we may next visit during our search. We note that we impose no selection criteria on which member of S we choose in line 3 beyond that it must be in S.

Algorithm 1: Generic search

input : a graph $G = (V, E)$ and start vertex s
output: an ordering σ of V

1 $S \leftarrow \{s\}$
2 **for** $i \leftarrow 1$ *to* n **do**
3   pick and remove an unnumbered vertex v from S
4     $\sigma(i) \leftarrow v$ //This assigns to v the number i
5   **foreach** *unnumbered vertex* w *adjacent to* v **do**
6   add w to S

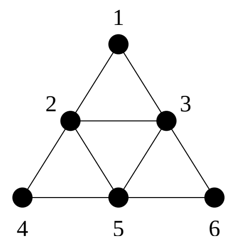
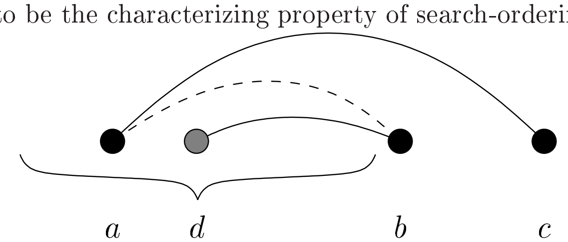


FIG. 2.1. *The 3-sun, used as an example for our graph searches.*

We call any ordering of V that can be generated by Algorithm 1 a *search-ordering* of the graph G. As an example, we notice that (1,2,4,3,5,6) is a valid search-ordering of the 3-sun in Figure 2.1. The reader will notice that this ordering is neither a BFS nor a DFS, which is permissible by the generic selection criterion in line 3. We may opt to use a more restrictive data structure to obtain other properties for our search, such as replacing set S with a queue to obtain BFS. Note also that, as stated, a given vertex may be added to S multiple times as each of its neighbors are numbered. How S deals with this can change the nature of the search. For example, using a stack where the add operation moves a vertex to the top of the stack if that vertex was added previously will produce DFS.

We now proceed to answer the fundamental question raised above about our search-orderings: how could b have been chosen before c? We give a simple answer, which turns out to be the characterizing property of search-orderings.



(S): Given an ordering σ of V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex $d < b$ such that $db \in E$.

We note that (S) makes no statement to the relative order of a and d in σ. It simply says that there must be a vertex adjacent to b visited sometime earlier than b in σ. We will now show that (S) characterizes search-orderings. That is to say that any ordering that the generic search algorithm can produce has this property, and that any vertex ordering with this property can be produced by the generic search algorithm when we break ties suitably.

THEOREM 2.1. *For an arbitrary graph G, an ordering σ of V is a search-ordering of G if and only if σ has property (S).*

Proof. For the forward direction, suppose σ is an ordering generated by Algorithm 1. Let $a < b < c$ be a triple of vertices in σ with $ac \in E$ and $ab \notin E$. At the point when b is chosen, both b and c must be in S. Since b is in S, some neighbor of b must have already been chosen, call this neighbor d. Thus $d < b$ and $db \in E$. As this holds for all triples in σ, property (S) holds on σ.

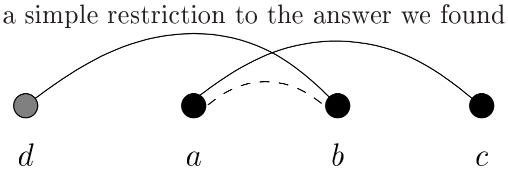
For the converse, suppose $\sigma = (v_1, \ldots, v_n)$ is an ordering respecting property (S). Suppose for contradiction that $(v_1, \ldots, v_{i-1})$ can be obtained by Algorithm 1, but $(v_1, \ldots, v_i)$ cannot, for some i, $1 < i \le n$. That is, v_i is the first vertex in σ that cannot be chosen next by the algorithm. Consider the next iteration of the algorithm after choosing v_{i-1}. Since v_i cannot be chosen next, $v_i \notin S$. Let u be a vertex that the algorithm may choose next. Then u must be in S. Let w be the neighbor of u which caused u to be added to S. Since $v_i \notin S$, we know that $wv_i \notin E$. By applying property (S) to the vertex triple (w, v_i, u), we know there exists a vertex $d < v_i$ with $dv_i \in E$. Hence $v_i \in S$, contradicting our assumption that v_i could not be chosen by Algorithm 1. ◻

2.2. BFS. We now proceed to the familiar breadth-first search algorithm. BFS is a restriction of generic search in that we use a queue to determine which vertex we may choose next at our selection step. BFS does not, however, determine which order to push the neighbors of a numbered vertex onto the queue (see §2.3 LexBFS for a search that does). Algorithm 2 should be familiar to most readers.

Algorithm 2: Breadth-first search (BFS)

```
input  : a graph G = (V, E) and start vertex s
output: an ordering σ of V
initialize queue to {s}
for i ← 1 to n do
    pop v from top of queue
    σ(i) ← v      //This assigns to v the number i
    foreach unnumbered vertex w adjacent to v do
        if w in not already in queue then
            push w on queue
```

We key in on our central question: how could b have been chosen before c in a BFS? The answer is a simple restriction to the answer we found for generic search.



(B): Given an ordering σ of V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex $d < a$ such that $db \in E$.

This property is merely property (S) with the added restriction that $d < a$. Interestingly, we find that this is sufficient to characterize all BFS-orderings.

THEOREM 2.2. *For an arbitrary graph G, an ordering σ of V is a BFS-ordering of G if and only if σ has property (B).*

Proof. For the forward direction, suppose σ is a BFS-ordering. Let $a < b < c$ be a triple of vertices in σ with $ac \in E$ and $ab \notin E$. If b was removed from (and thus placed on) the queue before c, it must have a neighbor d that was numbered as least as early as a. Since a does not have b as a neighbor, d must be earlier than a.

For the reverse direction, suppose for contradiction that $\sigma = (v_1, \ldots, v_n)$ is an ordering satisfying property (B1) such that $(v_1, \ldots, v_{i-1})$ can be produced by BFS, but $(v_1, \ldots, v_i)$ cannot, for some i, $1 < i \leq n$. That is, v_i cannot be chosen next by any valid BFS of G. Let u be a vertex that BFS may choose next. Then there must be a vertex $w < v_i$ adjacent to u but not to v_i: choose w to be the leftmost such vertex in σ. Applying (B) to the triple $w < v_i < u$, there must exist a vertex $d < w$ adjacent to v_i. Since w was chosen leftmost, any vertex left of w which is adjacent to

u must also be adjacent to v_i. There is a vertex left of w adjacent to v_i (namely d), thus a BFS could choose v_i before u, contradicting our assumption above. ∎

To illustrate an application of this property, we now prove the correctness of a generalization of a well known technique for extracting a shortest s, w-path (for arbitrary w) given an ordering with property (B) starting with s. For notation, when given a vertex ordering σ of V and a subset $T \subseteq V$, we say σ_T is the restriction of σ to T.

COROLLARY 2.3. *Let σ be an arbitrary ordering of the vertices of a graph G, and let $H = G[T]$ be a connected induced subgraph of G such that σ_T satisfies (B). Suppose s is the first vertex in σ_T, and let $w = w_0$ be an arbitrary vertex in T. For $i > 0$, define w_i to be the leftmost (in σ_T) neighbor of w_{i-1}, until s is reached. Then $(s = w_k, \ldots, w_1, w_0 = w)$ is a shortest s, w-path in H.*

Proof. Suppose, for contradiction, that $(s = w_k, \ldots, w_1, w_0 = w)$ is not a shortest s, w-path in G. Let $(s = x_{k'}, \ldots, x_1, x_0 = w)$ be a shorter s, w-path, with $k' < k$. Then there exists an $i > 0$ with $x_i < w_i$ (setting $i = k'$ suffices). Choose the smallest such i (thus $w_{i-1} \leq x_{i-1}$). Since $x_i x_{i-1} \in E$ and $w_i > x_i$ is the leftmost neighbor of w_{i-1}, we know that $x_i w_{i-1} \notin E$ (and thus $w_{i-1} \neq x_{i-1}$). Consider the triple $x_i < w_{i-1} < x_{i-1}$. Since $x_i x_{i-1} \in E$ but $x_i w_{i-1} \notin E$, property (B) says there is a vertex $d < x_i$ with $d w_{i-1} \in E$. This contradicts w_i being the leftmost neighbor of w_{i-1}, hence our original path was in fact shortest. ∎

2.3. LexBFS. In the studies of LexBFS, the answer to our key question, property (LB) below, has long been exploited as a useful characterization of LexBFS-orderings. For completeness, we present the traditional LexBFS labelling algorithm as Algorithm 3.

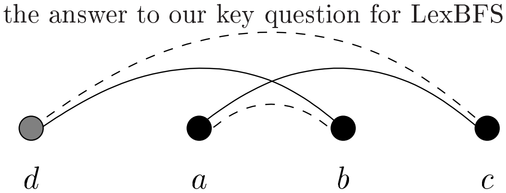
Algorithm 3: Lexicographic breadth-first search (LexBFS)

input : a graph $G = (V, E)$ and start vertex s
output: an ordering σ of V

assign the label $\emptyset$ to all vertices
$label(s) \leftarrow \{n + 1\}$
for $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with lexicographically largest label
  $\sigma(i) \leftarrow v$ //This assigns to v the number i
  foreach *unnumbered vertex w adjacent to v* **do**
    append $(n - i)$ to $label(w)$

As an example, consider Algorithm 3 on the graph in Figure 2.1 from the start vertex 1. The ordering (1,2,3,5,4,6) is a valid LexBFS-ordering of the 3-sun. Note that once 1,2,3 are picked, 5 must be chosen over 4 since $label(5) = \{4, 3\}$ while $label(4) = \{4\}$. This ordering is also a BFS ordering of the 3-sun, but the BFS-ordering (1,2,3,4,5,6) is not a LexBFS-ordering.

We now restate the answer to our key question for LexBFS.



(LB): Given an ordering σ of V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex $d < a$ such that $db \in E$ and $dc \notin E$.

This property says that if we choose b before c, but there is an earlier vertex a that would normally pull c before b, then there must be an even earlier vertex d that caused b to be pulled ahead of c. Though this has long been known to be a property of LexBFS-orderings [11], the reverse direction showing (LB) characterizes LexBFS-orderings has only recently been shown (in fact, independently by several different groups in the late 1990s).

THEOREM 2.4 ([1]). *For an arbitrary graph G, a vertex ordering σ is a LexBFS-ordering of G if and only if σ has property (LB).*

Remarkably, the requirement of dc not being an edge is the only difference between lexicographic and traditional breadth-first search. We obtain the immediate corollary, without any mention of the respective algorithms, that a LexBFS-ordering is a BFS-ordering, since (B) subsumes (LB).

2.4. DFS. We now turn our attention to depth-first search. DFS is very different from BFS in that it progresses forward through the graph as much as possible, backtracking only when necessary, whereas BFS first systematically explores close vertices before venturing out to the far reaches of the graph. One might further reflect that DFS, in addition to the discovery ordering, produces a second ordering of the vertices based on when each vertex's neighborhood is fully explored and incident edges fully traversed. In this paper we concentrate solely on the discovery orderings of depth-first searches.

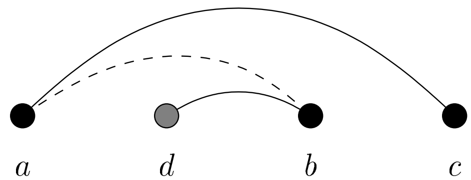
We start by presenting a DFS algorithm exploiting a stack data structure to store candidate vertices to next visit. We present DFS as Algorithm 4 in a style similar to our generic search algorithm in §2.1. We note that line 6 would typically be included in the push operation of a suitable stack-type data structure, but we include the step here in our algorithm for clarity.

Algorithm 4: Depth-first search (DFS)

input : a graph $G = (V, E)$ and start vertex s
output: an ordering σ of V

1 initialize *stack* to $\{s\}$
2 **for** $i \leftarrow 1$ *to* n **do**
3   pop v from top of *stack*
4     $\sigma(i) \leftarrow v$ //This assigns to v the number i
5   **foreach** *unnumbered vertex* w *adjacent to* v **do**
6   **if** w *already in stack* **then** remove w from *stack*
7   push w to top of *stack*

We again concentrate on our key question: if given a DFS-ordering with $a < b < c$ with ac an edge but ab not an edge, how could DFS have chosen b over c? The answer is property (D), which we will show characterizes DFS-orderings.



(D): Given an ordering σ of V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex d with $a < d < b$ such that $db \in E$.

Notice that this property restricts (S) only in that d must also appear to the right of a. We contrast this with BFS property (B) where d was to the left of a. In this sense, DFS is the opposite search paradigm to BFS.

THEOREM 2.5. *For an arbitrary graph G, an ordering σ of V is a DFS-ordering of G if and only if σ has property (D).*

Proof. For the forward direction, suppose σ is an ordering produced by DFS. Suppose we have a triple of vertices $a < b < c$ in σ with $ac \in E$ and $ab \notin E$. At the point at which b is chosen, b must be on the *stack*. Since $b \neq s$, a neighbor of b must have already been chosen. Let d be the rightmost neighbor of b that precedes b in σ. Thus $d < b$ and $db \in E$. Suppose $d < a$. Since a was chosen after d, c must be placed on the stack above b. As there are no other neighbors of b before b is supposedly chosen, c must be chosen before b as it is higher on the stack, a contradiction. Thus we must have $a < d$, and property (D) holds.

Now for the reverse direction, suppose $\sigma = (v_1, \dots, v_n)$ is an ordering respecting property (D). Suppose $(v_1, \dots, v_{i-1})$ can be obtained by DFS, but $(v_1, \dots, v_i)$ cannot. That is, v_i is the first vertex in σ that cannot be chosen next by DFS. Let u be a vertex that DFS may choose next. Let v_j be the rightmost vertex in σ_{i-1} such that $v_j u \in E$ (that is, v_j is the parent vertex of u in the DFS-tree). Clearly $v_j v_i \notin E$, and, more generally, $v_l v_i \notin E$ for all l, $j \leq l < i$. By applying property (D) to the triple (v_j, v_i, u), we know there exists a vertex d between v_j and v_i such that $dv_i \in E$, a contradiction. □

We can apply (D) repeatedly to obtain the following simple corollary.

COROLLARY 2.6. *Given a graph G and an arbitrary DFS-ordering σ of V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then there exists a path P from a to b whose vertices increase in σ. That is, there exists an ab-path $P = (a = d_0, d_1, \dots, d_k = b)$ in G with $\sigma^{-1}(d_{i-1}) < \sigma^{-1}(d_i)$ for each $d_i \in P \backslash \{a\}$.*

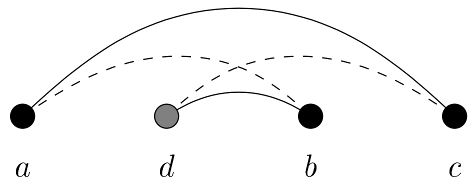
Most previous results regarding DFS rely on properties of the DFS-tree. The *DFS-tree* T is defined on the vertices of G and includes exactly those edges traversed to number a new vertex. Formally, we define $E(T) = \{uv \in E(G) \mid \sigma^{-1}(u) < \sigma^{-1}(v) \text{ and } \sigma^{-1}(u) \text{ minimum}\}$. We say u is the *parent* of v in T if $uv \in E(T)$ and $\sigma^{-1}(u) < \sigma^{-1}(v)$. The *nontree edges* (or *back edges*) are the remaining edges in G. In T, u is an *ancestor* of v if there is a path from u to v all of whose edges go from parent to child. We say two vertices are *related* if one is an ancestor of the other.

Textbooks (such as, for example, [2]) oft mention the following fundamental property of DFS without addressing the issue of characterizing the search.

(D2): In a DFS-tree of an undirected graph, the two endpoints of each back (nontree) edge are related.

This property is a trivial consequence of Corollary 2.6. Let uv be a back edge in a DFS tree. The four-vertex condition (D) is stronger than (D2) in that it says all vertices between u and v (inclusive) are related to u.

2.5. LexDFS. Looking at the relationship between BFS and LexBFS, one naturally asks whether there is a "lexicographic analogue" for DFS. We have noted that LexBFS enforces the requirement of "$dc \notin E$" on top of BFS. It is natural to question whether a similar requirement makes sense for DFS. We state this restriction of property (D) below.



(LD): Given an ordering σ of V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex d with $a < d < b$ such that $db \in E$ and $dc \notin E$.

We turn to the task of devising an algorithm that realizes this property. Considering our 3-sun example in Figure 2.1, once vertices 1 and 2 are chosen, the only valid ordering is (1,2,3,5,6,4). Clearly this is a DFS-ordering, but the DFS-ordering (1,2,4,5,6,3) easily violates (LD). The LexBFS-ordering (1,2,3,5,4,6) is also disallowed.

From this study, we conclude that the algorithm we devise must choose vertices that are adjacent to as many vertices we have most recently numbered as possible. And since it is to be related to LexBFS, our algorithm should be similar in some sense to our LexBFS algorithm (we discuss what property actually relates LexBFS and LexDFS in §2.6). We present Algorithm 5 which satisfies these requirements.

Algorithm 5: Lexicographic depth-first search (LexDFS)

```
input  : a graph G = (V, E) and start vertex s
output: an ordering σ of V
1 assign the label ∅ to all vertices
2 label(s) ← {0}
3 for i ← 1 to n do
4     pick an unnumbered vertex v with lexicographically largest label
5     σ(i) ← v      //This assigns to v the number i
6     foreach unnumbered vertex w adjacent to v do
7         prepend i to label(w)
```

Examining Algorithm 5, our answer to the "what is LexDFS?" question becomes quite convincing. There is just one difference between this algorithm and the LexBFS algorithm. For LexDFS, we prepend the new digit i to *label*(i) at line 7, whereas, for LexBFS, we append the new label digit $n - i$. Studying this elementary change for a few moments should convince the reader that this fact converts the algorithm from a breath-first search into a depth-first search. We now show that our algorithm correctly defines LexDFS.

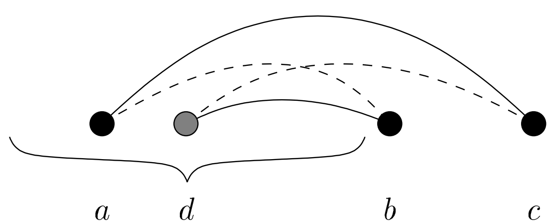
THEOREM 2.7. *For an arbitrary graph G, an ordering σ of V is a LexDFS-ordering of G if and only if σ has property (LD).*

Proof. For the forward direction, suppose σ is an ordering generated by LexDFS. Let $a < b < c$ be a triple of vertices in σ with $ac \in E$ and $ab \notin E$. For b to be chosen over c, *label*$(b) \geq$ *label*(c) when b is chosen. Since $\sigma^{-1}(a)$ is contained in *label*(c) and not contained in *label*(b), *label*(b) must contain some $j > \sigma^{-1}(a)$ which is not contained in $label(c)$. Let $d = \sigma(j)$. Then $db \in E$ and $dc \notin E$. Since $j > \sigma^{-1}(a)$, $a < d$, thus property (LD) is satisfied.

Now for the reverse direction, suppose $\sigma = (v_1, \ldots, v_n)$ is an ordering respecting property (LD). Suppose $(v_1, \ldots, v_{i-1})$ can be obtained by LexDFS, but $(v_1, \ldots, v_i)$ cannot. That is, v_i is the first vertex in σ that cannot be chosen next by LexDFS. Let u be a vertex that LexDFS may choose next. Since u must be chosen over v_i, $label(u)$ must be lexicographically larger than $label(v_i)$: let j be the largest digit in $label(u)$ that is not in $label(v_i)$ (i.e., v_j is the rightmost vertex in σ_{i-1} with $v_j u \in E$ and $v_j v_i \notin E$). By applying property (LD) to the vertex triple (v_j, v_i, u), we know there exists a vertex d between v_j and v_i such that $dv_i \in E$ and $du \notin E$. Since $v_j < d$, $label(v_i) > label(u)$, so v_i should have been chosen before u, a contradiction. ☐

As a new search algorithm, we should concern ourselves with its implementation. Algorithm 5 does not immediately have a linear-time implementation using those ideas used for LexBFS. The partitioning idea exploited for LexBFS cannot efficiently be applied to LexDFS, as its depth-first properties can pull in an arbitrary vertex to be the next to be numbered. We refer to [10] for details regarding implementation of LexDFS and its running time.

2.6. Maximal Neighborhood Search. We now consider the question of "what fact relates LexBFS and LexDFS?" What useful property does this nonedge dc give us? Equivalently, we ask what type of search we get when adding the restriction $dc \notin E$ to the generic search property (S). We formalize our discussion around this property.



(M): Given an ordering σ of V, if $a < b < c$ and $ac \in E$ and $ab \notin E$, then there exists a vertex d with $d < b$ and $db \in E$ and $dc \notin E$.

Surprisingly, this property has already been used in the literature. Tarjan and Yannakakis [13] mentioned that both maximum cardinality search (where at each step we choose a vertex that is adjacent to the maximum number of vertices previously numbered) and LexBFS respect property (M), and showed that any ordering with property (M) is the reverse of a perfect elimination ordering of a chordal graph. But no thought was given to whether there was a search algorithm that is characterized by this property. We answer that question here by introducing *maximal neighborhood search* (MNS) as Algorithm 6.

The idea behind MNS is simple: at each step, pick a vertex whose neighborhood in the part of the graph already explored is maximal. This is a straightforward generalization of LexBFS, LexDFS and MCS. We obtain each of these searches from Algorithm 6 by specifying which maximal neighborhood is most interesting. For LexBFS, we choose a vertex adjacent to as many *earliest* chosen vertices as possible. For LexDFS, we choose a vertex adjacent to as many *most recently* chosen vertices as possible. For MCS, we choose a vertex whose neighborhood has *maximum* cardinality, which must also clearly be maximal.

We now show that the vertex orderings produced by Algorithm 6 are characterized by property (M).

Algorithm 6: Maximal neighborhood search (MNS)

input : a graph $G = (V, E)$ and start vertex s
output: an ordering σ of V

assign the label $\emptyset$ to all vertices
label$(s) \leftarrow \{n + 1\}$
for $i \leftarrow 1$ *to* n **do**
  pick an unnumbered vertex v with maximal label
  $\sigma(i) \leftarrow v$ //This assigns to v the number i
  foreach *unnumbered vertex* w *adjacent to* v **do**
    add i to *label*(w)

THEOREM 2.8. *For an arbitrary graph G, an ordering σ of V is a MNS-ordering if and only if σ has property (M).*

Proof. For the forward direction, suppose σ is an MNS-ordering. Suppose $a < b < c$ is a triple in σ that violates (M). That is, $ac \in E$ and $ab \notin E$, yet there is no vertex $d < b$ with $db \in E$ and $dc \notin E$. Consider the point the in execution when $b = v_i$ is chosen. Then $N_{G_{i-1}}(b) \subset N_{G_{i-1}}(c)$, so b could not have been chosen next by MNS, a contradiction. Hence (M) must hold for σ.

For the converse, suppose $\sigma = (v_1, \ldots, v_n)$ is an ordering respecting propery (M). Suppose $(v_1, \ldots, v_{i-1})$ can be obtained by MNS, but $(v_1, \ldots, v_i)$ cannot; that is, v_i is the first vertex in σ that cannot be chosen next by MNS. Then there must be another vertex u with $N_{G_{i-1}}(u) \supsetneq N_{G_{i-1}}(v_i)$. Let $x \in N_{G_{i-1}}(u) - N_{G_{i-1}}(v_i)$ (such an x must exist). We apply (M) to the triple $x < v_i < u$, giving a vertex $d < v_i$ with $dv_i \in E$ and $d_u \notin E$. This contradicts $N_{G_{i-1}}(u) \supsetneq N_{G_{i-1}}(v_i)$, hence our assumption that v_i could not be chosen by MNS was incorrect. □

3. Application to powers of graphs. All searches give a certain amount of insight into a graph's structure, including the insight gained from our four-vertex characterizations. At times we wish to use this insight to gain information about some graph arising from a given graph via some function. Examples of such functions include the graph's complement, some power of the graph, the line graph, or the product or join of the graph. Often, we want to obtain insight into the structure of a derived graph without incurring the possibly large cost of computing the function.

Search is often a convenient and efficient way to do this. For example, it has been previously noted that a LexBFS-ordering (and thus a BFS-ordering) of the complement of a graph can be found without computing the complement, given a slight modification of the standard LexBFS algorithm [9]. Dahlhaus, Gustedt and McConnell [5] have shown how to find a DFS-ordering of the complement of a graph in time linear in the original graph. We can use such results to obtain information about the derived graph (such as whether the graph's complement is chordal) more efficiently than actually computing the derived graph.

In this section, we consider the problem of computing some type of search ordering of some power of the graph, but without resorting to computing that power. One possibility is to compute a search ordering of the original graph, and ask whether it is a search ordering of the power. We will show that the aforementioned search characterizations are invaluable in obtaining straightforward proofs of both positive and negative results for this problem.

We start with the simple question of whether a BFS-ordering of a graph remains a BFS-ordering for all powers of a graph.

THEOREM 3.1. *For any graph G, every BFS-ordering of G is a BFS-ordering of G^k, for all $k \geq 1$.*

Proof. Let σ be a BFS-ordering of G. The theorem is trivially satisfied when $k = 1$. For $k > 1$, we assume for induction that σ is a BFS-ordering for all powers of G strictly less than k and consider σ on G^k. Suppose $a < b < c$ is a triple in σ on G^k with $ac \in E^k$ and $ab \notin E^k$.

If (a, b, c) is such a triple in σ on G^{k-1}, then by (B) and our induction hypothesis, there exists a vertex $d < a$ with $db \in E^{k-1} \subseteq E^k$, hence satisfying (B) on G^k.

Suppose instead that (a, b, c) is not such a triple in σ on G^{k-1}. Then $ac \notin E^{k-1}$. But then there must be a vertex e with $ae \in E^{k-1}$ and $ec \in E$. Since $ab \notin E^k$ but $ae \in E^{k-1}$, we must have $eb \notin E$. We have 3 cases for the position of e in σ relative to a and b.

Case 1: $e < a$. In σ on G, since $ec \in E$ and $eb \notin E$, we apply (B) to the triple $e < b < c$, giving us a vertex $d < e < a$ such that $db \in E \subseteq E^k$, satisfying (B) on G^k.

Case 2: $a < e < b$. We first consider σ on G. We know that $ec \in E$ and $eb \notin E$, so by applying (B) to the triple $e < b < c$, we get a vertex $d < e$ such that $db \in E$. If $d < a$, we're done; otherwise, we have $a < d$. Since $db \in E$ but $ab \notin E^k$, we must have $ad \notin E^{k-1}$. We apply (B) and our inductive hypothesis to the triple $a < d < e$ in σ on G^{k-1}, giving us a vertex $d_2 < a$ such that $d_2 d \in E^{k-1}$. Since $d_2 d \in E^{k-1}$ and $db \in E$, d_2 and b are of distance at most k in G. Thus $d_2 b \in E^k$, satisfying (B) on G^k.

Case 3: $b < e$. In σ on G^{k-1}, since $ae \in E^{k-1}$ and $ab \notin E^{k-1}$, we apply (B) and our inductive hypothesis to the triple $a < b < e$, giving us a vertex $d < a$ with $db \in E^{k-1} \subseteq E^k$, satisfying (B) on G^k.

In each case, we show the existence of a vertex $d < a$ with $db \in E^k$, hence (B) is satisfied by σ on G^k. Thus σ is a BFS-ordering of G^k. $\square$

Whereas a BFS-ordering survives through the powers of a graph, and clearly generic search-orderings also survive, one naturally wonders about DFS. We now show that a DFS-ordering of G is not necessarily a DFS-ordering in a given power of G. Again, the characterizing property of DFS will play an crucial role in our proof.

THEOREM 3.2. *For every $k \geq 2$, there exists a graph G such that no DFS-ordering of G is a DFS-ordering of G^k.*

Proof. Fix $k \geq 2$. We first show the existence of a graph G such that no DFS-ordering starting from a distinguished start vertex s is a DFS-ordering of G^k.

Let G be the graph consisting of two vertices s and r connected by three paths of length $k + 1$. This graph is diagrammed in Figure 3.1.

Let σ be an arbitrary DFS-ordering of G starting from s. The DFS will proceed from s along one of the three paths to reach r. From r, the DFS will follow a second path towards s to reach a adjacent to s. As s is already discovered, the DFS will recurse back to r, and travel the third path towards s. Let b be the first vertex visited on this third path, and let c be the next vertex on the path.

Thus, in σ, we have $s < r < a < b < c$. Moreover, b appears in σ immediately following a (when a is reached, all neighbors of a are already numbered, so DFS recurses to r, where b must be the next vertex to be numbered).

Now consider σ in G^k. One easily verifies that a and c are of distance k in G, and a and b are of distance $k + 1$ in G. Hence $ac \in E^k$ and $ab \notin E^k$. But since a and b are next to each other in σ, there can be no vertex between a and b. Therefore, the

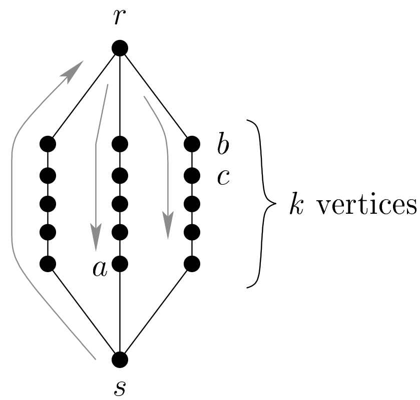


FIG. 3.1. *Given k, a graph G where no DFS from s is a DFS of G^k. The possible DFS path through G is shown in grey.*

triple $a < b < c$ violates the condition (D). Hence σ cannot be a DFS-ordering of G^k.

To prove the theorem, we consider two copies of G joined at s by an edge: call this graph G'. Consider any DFS-ordering σ' of G': it may start in either copy of G, possibly at s itself. We observe that if σ' starts in one copy of G, σ' restricted to the other copy of G appears as a DFS-ordering of G starting at s. Following the argument above, we can identify three vertices $a < b < c$ in the second copy of G.

We note that no vertex from the other copy of G is within distance k of b, thus there can be no edge ab in $(G')^k$. The triple $a < b < c$ violates (D) as above, so σ' is not a DFS of $(G')^k$. Since σ' was chosen arbitrarily, no DFS-ordering of G' can be a DFS-ordering of $(G')^k$. ☐

Failing for DFS, one may wonder about LexBFS. Brandstädt, Dragan and Nicolai [1] have shown that the reverse of a LexBFS-ordering of a chordal graph G is a common perfect elimination ordering of all odd powers of G. One may be tempted to think that a LexBFS-ordering of a chordal graph is a LexBFS-ordering of the odd powers, but we show this is not the case.

THEOREM 3.3. *For each $k \geq 2$, there exists a chordal graph G such that no LexBFS-ordering of G is a LexBFS-ordering of G^k.*

Proof. Let $k \geq 2$ be fixed. If k is even, we consider the graph G_k in Figure 3.2(a); otherwise, if k is odd, we consider the graph H_k in Figure 3.2(b). We note that each of these graphs are chordal. We will show that no LexBFS-ordering of the graph starting from vertex 1 will be a LexBFS-ordering of the kth power, then join two copies of the graph at vertex 1 to prove the theorem for a LexBFS-ordering starting from an arbitrary vertex.

Suppose k is even. We note that any LexBFS-ordering σ of the graph G_k which starts at 1 must have $7 < a' < b' < c'$. Then $a < b < c$ in σ. We examine the triple $a < b < c$ in ${G_k}^k$. We note that $ac \in E({G_k}^k)$ since $d_{G_k}(a, c) = 2(k/2 - 1) + 2 = k$, and $ab \notin E({G_k}^k)$ since $d_{G_k}(a, b) = 2(k/2 - 1) + 3 = k + 1$. It is easy to check that there is no vertex $d < a$ with $db \in E({G_k}^k)$ and $dc \notin E({G_k}^k)$: any vertex left of a within distance k of b in G_k must be within distance k of c. Thus property (LB) is violated in ${G_k}^k$, showing σ is not a LexBFS-ordering of ${G_k}^k$.

Now suppose k is odd. Considering the graph in Figure 3.2(b), we note that any LexBFS-ordering σ of H_k that starts at 1 must have $a' < b' < c'$. Then $a < b < c$ in σ. We note that $d_{H_k}(b', 3) = 2$ and $d_{H_k}(c', 3) = 3$, and $d_{H_k}(b', 6) = 1$ and $d_{H_k}(c', 6) = 3$,

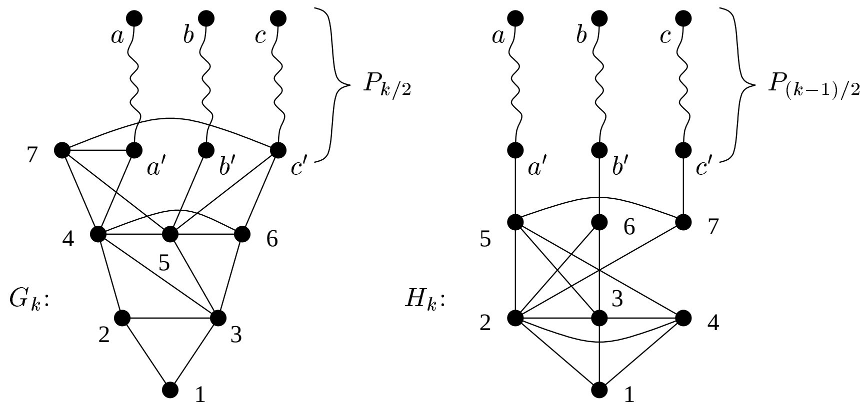


FIG. 3.2. *Constructions for (a) k even and (b) k odd, where no LexBFS-ordering starting from 1 is a LexBFS-ordering of G^k.*

but for all other vertices $x < a'$ in σ, $d_{H_k}(b', x) \geq d_{H_k}(c', x)$. We examine the triple $a < b < c$ in ${H_k}^k$. We have $ac \in E({H_k}^k)$, since $d_{H_k}(a, c) = 2(\frac{k-1}{2} - 1) + 3 = k$, and $ab \notin E({H_k}^k)$, since $d_{H_k}(a, b) = 2(\frac{k-1}{2} - 1) + 4 = k + 1$. But since all vertices $d < a$ in σ within distance k of b are also within distance k of c, there is no candidate vertex d to satisfy property (LB) for a, b, c. Thus σ is not a LexBFS-ordering of ${H_k}^k$.

To complete the proof, we construct a graph G consisting of two copies of G_k, if k is even (respectively, two copies of H_k, if k is odd), with the vertices labelled 1 in each copy joined by an edge. Any LexBFS-ordering of G when restricted to at least one of the copies of G_k (or H_k) will appear as a LexBFS-ordering with start vertex 1. Applying the argument above will identify vertices $a < b < c$ that violate property (LB) in G^k, completing our proof. ∎

We note that the graph in Figure 3.2(b) is in fact strongly chordal. A graph is *strongly chordal* if it is chordal and every cycle $C = (x_1, \ldots, x_{2k})$ of even length has a chord $x_i x_j$ where the distance between the endpoints x_i and x_j in C is odd. An immediate corollary of Theorem 3.4 is that for each $k \geq 1$, there exists a strongly chordal graph G such that no LexBFS-ordering of G is a LexBFS-ordering of G^{2k+1}.

LexBFS does, however, work for a particular class of graphs, as shown by the next theorem.

THEOREM 3.4. *Let G be a chordal bipartite graph. If σ is a LexBFS-ordering of G, then σ is a LexBFS-ordering of the graph G^2.*

Proof. Let σ be a LexBFS-ordering of a chordal bipartite graph $G = B(X, Y, E)$. Let $l(x)$ denote the BFS layer of x in σ; $l(x) = 0$ if x is the first vertex of σ. We first state a few simple facts.

FACT 1. *If $x \in X$ and $xy \in E$, then $y \in Y$.*

FACT 2. *If $x \in X$ and $xy \in E^2 - E$, then $y \in X$.*

FACT 3. *If $x \in X$, $y \in Y$ and $z \in X$, then $l(x) = l(y) + 2i - 1 = l(z) + 2j$ for some (possibly negative) integers i, j.*

FACT 4. *If there is some induced P_5 a, b, c, d, e with $l(a) = l(e) = l(b) - 1 = l(d) - 1 = l(c) - 2$, then there exists an induced C_{2k}, $k \geq 3$, in G.*

To prove our theorem, suppose we have $a < b < c$ with $ac \in E^2$ and $ab \notin E^2$. We will show the existence of a vertex $z < a$ with $zb \in E^2$ and $zc \notin E^2$.

Case 1: $ac \in E$. By (LB) on (a, b, c), there is a $d < a$ with $db \in E$ and $dc \notin E$. If $dc \notin E^2$, d satisfies the theorem. So assume $dc \in E^2 - E$. Since $ab \notin E^2$, we must have

$da \notin E$. By Facts 2 and 3, and since $l(d) \leq l(a) = l(c) - 1$, we have $l(d) = l(c) - 2$. By (LB) on (d, a, b), there is an $e < d$ with $ea \in E$ and $eb \notin E$. Since $l(e) = l(d)$, $ed \notin E$. By (LB) on (e, d, a), there must be some $f < e$ with $fd \in E$, $fa \notin E$ and $l(f) = l(d) - 1 = l(c) - 3$. Thus $fb \in E^2$ and $fc \notin E^2$, satisfying our requirement.

Case 2: $ac \notin E$. Then there is some d with $da, dc \in E$; choose the leftmost (in σ) such d. Since $ab \notin E^2$, $db \notin E$. We have 4 cases for the position of d relative to a, b and c.

Case 2.1: $c < d$. By (LB) on (a, c, d), there is an $e < a$ with $ec \in E$ and $ed \notin E$. Since d is leftmost, $ea \notin E$. By (LB) on (e, a, c), there is an $f < e$ with $fa \in E$ and $fc \notin E$. Then f, a, d, c, e is an induced P_5 such that Fact 4 gives us an induced cycle of length at least 6, contradicting G being chordal bipartite.

Case 2.2: $b < d < c$. By (LB) on (a, b, d), there is an $e < a$ with $eb \in E$ and $ed \notin E$. Since $ab \notin E^2$, $ea \notin E$. For generality, we discard our assumptions that $b < d$ and $ed \notin E$. Now, by (LB) on (e, a, b), there is some $f < e$ with $fa \in E$ and $fb \notin E$, which means $l(f) = l(a) - 1 = l(c) - 3$. If $fe \in E$, then $fb \in E^2$ and since $fc \notin E^2$, f satisfies the theorem. Otherwise $fe \notin E$, so applying (LB) to (f, e, a) gives us a $g < f$ with $ge \in E$, $ga \notin E$. Since $l(g) \leq l(f) = l(c) - 3$, we know $gc \notin E^2$, so g fulfills the theorem since $gb \in E^2$.

Case 2.3: $a < d < b$. By (LB) on (d, b, c), there is some $e < d$ with $eb \in E$ and $ec \notin E$. Since $ab \notin E^2$, $ea \notin E$. If $e < a$, we are in the generalized Case 2.2 above, and can find a vertex g to satisfy the theorem. When $a < e$, we can apply (LB) to (a, e, d) to give an $f < a$ with $fe \in E$, $fd \notin E$. Thus $fb \in E^2$ and f satisfies the theorem unless there exists some g with $fg, gc \in E$. Suppose such a g exists. Then $l(f) = l(g) - 1 = l(c) - 2$ and $dg \notin E$. If $ag \notin E$, then a, d, c, g, f forms a P_5 and Fact 4 gives us an induced cycle of length at least 6, contradicting G being chordal bipartite. Thus $ag \in E$ and $d < g$ (since d is leftmost). By (LB) on (f, d, g), there is an $h < f$ with $hd \in E$ and $hg \notin E$. Then $l(h) = l(f)$ and $hf \notin E$. But then f, g, c, d, h is a P_5 such that Fact 4 gives us an induced cycle of length at least 6, contradicting G being chordal bipartite. Thus no such g can exist, and f satisfies the theorem.

Case 2.4: $d < a$. By (LB) on (d, b, c), there is an $e < d$ with $eb \in E$, $ec \notin E$. Then $l(e) = l(b) - 1 = l(c) - 1$, so $ec \notin E^2$, thus completing the proof. $\square$

Note that this theorem cannot be strengthened to bipartite graphs, as shown by the graph in Figure 3.3. Call this graph B. The ordering $\sigma = (a, b, c, d, e, f, g)$ is

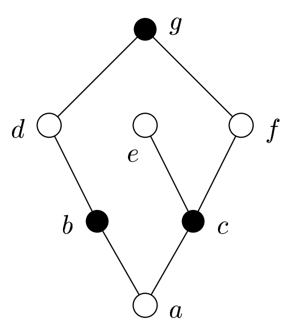


FIG. 3.3. *A bipartite graph with a LexBFS-ordering that is not a LexBFS-ordering of the square.*

clearly a LexBFS-ordering of B; however, the triple of vertices $d < e < f$ violates property (LB) in B^2. We have $df \in E^2$ and $de \notin E^2$, but every vertex in σ left of d that is adjacent to e in B^2 is also adjacent to f in B^2. Attaching a second copy of

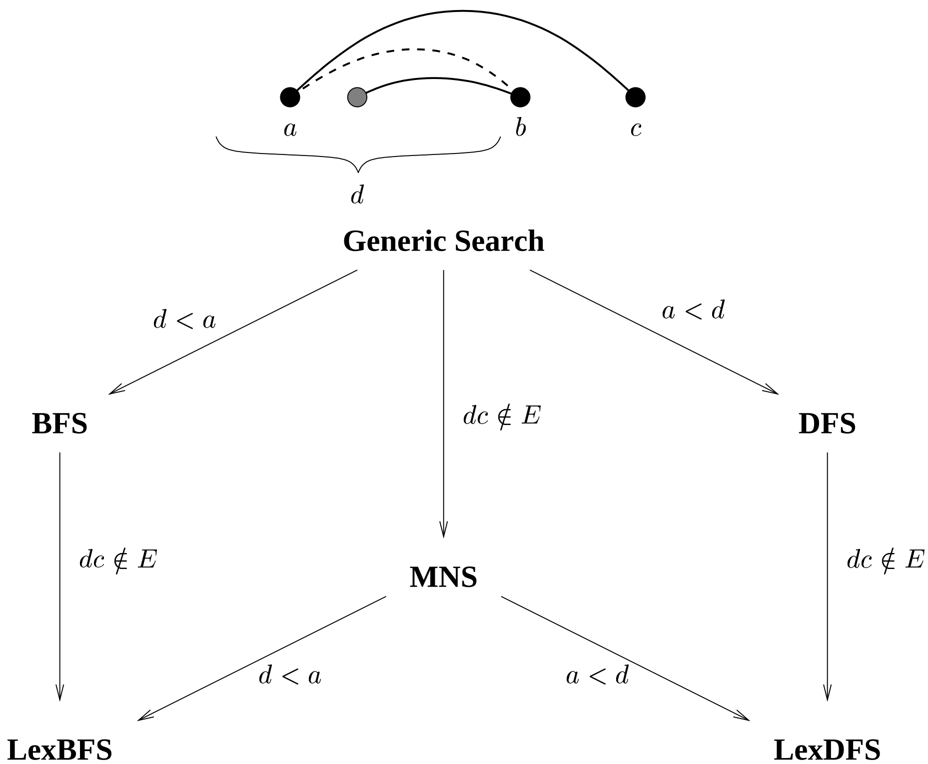


Fig. 4.1. *Summary of search characterizations. For example, vertex ordering σ is a LexDFS-ordering iff σ satisfies the condition for generic search together with $dc \notin E$ and $a < d$.*

B via an edge at a ensures no LexBFS-ordering of the graph is a LexBFS-ordering of its square.

4. Concluding Remarks. We have examined six different search algorithms from the perspective of our key question: if a search visits a vertex a before b before c, and there is an edge ac but no edge ab, how could the search have visited b before c? In this examination, we have discovered characterizing properties for the classic breadth-first and depth-first searches, and discovered a new class of maximal neighborhood searches. We summarize the characterizations of the six search paradigms in Figure 4.1.

Though we have concentrated on simple graphs in this paper, these ideas can easily be extended to more general environments. Our characterizations make no mention of multiple edges, but are equally applicable to searches on multigraphs. A search ordering implicitly imposes an orientation on the edges of a simple graph: it is straightforward to apply the same ideas to directed graphs to obtain similar characterizations. We can also extend such characterizations beyond graphs: these ideas also hold for searches on hypergraphs. We refer to [10] for details.

Such characterizing properties can be a very powerful and useful tool for studying graphs. Beyond giving us a better understanding of how a search reveals the structure of a graph, it allows us to employ (and analyze) multiple sweeps of a search to gather additional information about a graph. Multi-sweep LexBFS has recently been applied to a variety of problems [3], resulting in very simple, and in many cases intuitive, algorithms. Will the results presented in this paper lead to multi-sweep algorithms involving other searches or combinations of different searches?

REFERENCES

[1] A. Brandstädt, F. F. Dragan, and F. Nicolai, *LexBFS-orderings and powers of chordal graphs*, Discrete Math., 171 (1997), pp. 27–42.

[2] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*, McGraw-Hill Higher Education, 2nd ed., 2001.

[3] D. G. Corneil, *Lexicographic breadth first search — a survey*, in Proceedings of WG2004, LNCS, to appear.

[4] D. G. Corneil, S. Olariu, and L. Stewart, *LBFS orderings and cocomparability graphs*, in Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms, Society for Industrial and Applied Mathematics, 1999, pp. 883–884.

[5] E. Dahlhaus, J. Gustedt, and R. M. McConnell, *Partially complemented representations of digraphs*, Discrete Math. Theor. Comput. Sci., 5 (2002), pp. 147–168.

[6] D. R. Fulkerson and O. A. Gross, *Incidence matrices and interval graphs*, Pacific J. Math., 15 (1965), pp. 835–855.

[7] H. Gabow, *Searching*, in Handbook of Graph Theory, J. Gross and J. Yellen, eds., 2003, pp. 953–984.

[8] M. C. Golumbic, *Algorithmic Graph Theory and Perfect Graphs*, Academic Press, New York, 1980.

[9] M. Habib, R. McConnell, C. Paul, and L. Viennot, *Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing*, Theoret. Comput. Sci., 234 (2000), pp. 59–84.

[10] R. Krueger, *Graph Searching*, PhD thesis, University of Toronto, in preparation.

[11] D. J. Rose, R. E. Tarjan, and G. S. Lueker, *Algorithmic aspects of vertex elimination on graphs*, SIAM J. Comput., 5 (1976), pp. 266–283.

[12] R. E. Tarjan, *Depth first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[13] R. E. Tarjan and M. Yannakakis, *Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs*, SIAM J. Comput., 13 (1984), pp. 566–579.