Lexicographic Breadth First Search – A Survey

Derek G. Corneil

Department of Computer Science, University of Toronto,
Toronto M5S3G4, Ontario, Canada
dgc@cs.utoronto.ca

Abstract. Lexicographic Breadth First Search, introduced by Rose, Tarjan and Lueker for the recognition of chordal graphs is currently the most popular graph algorithmic search paradigm, with applications in recognition of restricted graph families, diameter approximation for restricted families and determining a dominating pair in an AT-free graph. This paper surveys this area and provides new directions for further research in the area of graph searching.

1 Introduction

Graph searching is a fundamental paradigm that pervades graph algorithms. A search of a graph visits all vertices and edges of the graph and will visit a new vertex only if it is adjacent to some previously visited vertex. Such a generic search does not, however, indicate the rules to be followed in choosing the next vertex to be visited. The two fundamental search strategies are *Breadth First Search* (BFS) and *Depth First Search* (DFS). As the names indicate, BFS visits all previously unvisited neighbours of the currently visited vertex before visiting the previously unvisited non-neighbours, whereas DFS follows unvisited edges (if possible) from the most recently visited vertex. Both searches seem to have been "discovered" in the 19th century (and probably earlier) as algorithms for maze traversal. DFS, as popularized by Tarjan [41], has been used for such diverse applications as connectivity, planarity, topological ordering and strongly connected components of digraphs. BFS has been applied to shortest path problems, network flows and the recognition of various graph classes.

In the mid 1970s, Rose, Tarjan and Lueker [42] introduced a variant of BFS called *Lexicographic Breadth First Search* (LBFS). Their application of LBFS was to the recognition of chordal graphs. This algorithm is one of the classic graph algorithms and, given the current interest in LBFS, it is somewhat surprising that little work was done on LBFS until the mid 1990s.

In this paper, we survey many of the applications of LBFS (in Section 4). Before doing so, we provide the graph theoretical background for the paper as well as a description of LBFS and its two most common variants (Section 2) and, in Section 3, present some LBFS structural results. Concluding remarks are made in the final section.

J. Hromkovič, M. Nagl, and B. Westfechtel (Eds.): WG 2004, LNCS 3353, pp. 1–19, 2004.

2 Background

Before presenting LBFS and its various variants, we give some relevant definitions. We start with standard graph theoretical definitions and then define various graph families and indicate some characterizations that will be used in the relevant LBFS algorithms. Further information regarding the definitions and families can be found in [6].

2.1 Definitions and Notation

All graphs will be assumed to be undirected and finite. For a graph $G(V, E)$, we use n to denote $|V|$ and m to denote $|E|$. K_n, C_n and P_n denote the Clique, Cycle and Path respectively on n vertices. A *House, Hole* and *Domino* are respectively: a C_4 sharing an edge with a K_3; an induced $C_k, k > 4$; a pair of C_4s sharing an edge. A subset of vertices M is a *module* if for all vertices $x, y \in M$ and $z \in V \setminus M, xz \in E$ if and only if $yz \in E$. Module M is *trivial* if $M = V, M = \emptyset$ or $|M| = 1$. A *maximal clique module* is a module that is a clique and is maximal with respect to both properties. Subset S of V is a *separator* if the graph induced on $V \setminus S$ is disconnected. A *moplex* is a maximal clique module whose neighbourhood is a minimal separator.

The *distance* between two vertices u and v is the length of a shortest path between u and v and is denoted $d(u, v)$. For vertex $v, ecc(v)$, the *eccentricity* of v is the length of a longest shortest path with v as an endpoint. The *diameter* ($diam(G)$) is the maximum eccentricity of all vertices in G. A vertex is *simplicial* if its neighbourhood is a clique. An ordering $v_1, v_2, \cdots, v_n$ of V is a *perfect elimination ordering* (PEO) if for all $i, 1 < i \leq n, v_i$ is simplicial in the graph induced on $v_1, \cdots, v_i$. A vertex v is *semisimplicial* if v is not the midpoint of any induced P_4. An ordering $v_1, v_2, \cdots, v_n$ of V is a *semiperfect elimination ordering* if for all $i, 1 < i \leq n, v_i$ is semisimplicial in the graph induced on $v_1, \cdots, v_i$. A vertex v is *2-simplicial* if there is no induced P_4 in the graph induced on $\{u : d(u, v) \leq 2\}$. An ordering $v_1, v_2, \cdots, v_n$ of V is a *2-simplicial elimination ordering* if for all $i, 1 < i \leq n, v_i$ is 2-simplicial in the graph induced on $v_1, \cdots, v_i$.

We say that path P *misses* vertex v if $P \cap N(v) = \emptyset$ (i.e., no vertex of P is adjacent to v). A path P is a *dominating path* if no vertex of G is missed by P. A pair of vertices x, y is a *dominating pair* if every path between x and y is a dominating path. Two vertices x, y are *unrelated with respect to vertex* v if there are paths P between x and v and Q between y and v such that P misses y and Q misses x. An independent triple of vertices x, y, z is an *Asteroidal Triple (AT)*, if between every pair of vertices, there is a path that misses the third. A vertex v is *admissible* if there are no unrelated vertices with respect to v. An ordering $v_1, v_2, \cdots, v_n$ of V is an *admissible elimination ordering* (AEO) if for all $i, 1 < i \leq n, v_i$ is admissible in the graph induced on $v_1, \cdots, v_i$.

For $t \geq 1$, an ordering $v_1, v_2, \cdots, v_n$ of V is a *strong t-cocomparability ordering* (strong t-CCPO) if for all $i, j, 1 \leq i < j < k \leq n, d(v_i, v_k) \leq t$ implies $d(v_i, v_j) \leq t$ or $d(v_j, v_k) = 1$. Note that a graph is a *cocomparability* graph (there is a

transitive orientation of the edges of the complement) if and only if it has a strong 1-CCPO [28].

A graph is *chordal* if there is no induced cycle of length greater than 3. Fulkerson and Gross [23] showed that a graph is chordal if and only if it has a perfect elimination ordering. G is *weakly chordal* if G and $\overline{G}$ contain no induced cycle $C_k, k \geq 5$. A graph is *strongly chordal* if it is chordal and every cycle of even length at least 6 has an *odd chord*, namely a chord where the distance on the cycle between the endpoints is odd. An *interval graph* is the intersection graph of intervals of a line. If all intervals are of the same length, then G is a *unit interval graph* (equivalently known as *proper interval graphs*, where no interval is allowed to properly contain another interval). A graph G is a *distance hereditary graph* if for every connected subgraph H, $x, y \in H$ implies that $d_H(x, y) = d_G(x, y)$. Nicolai [35] has shown that a graph is distance hereditary if and only if it has a 2-simplicial elimination ordering. A graph is *HHD-free* if it contains no House, Hole or Domino, as defined above. Bipartite graphs with no induced cycles of size greater than 4 are called *chordal bipartite*.

Cographs are the graphs formed by the closure of the disjoint union and complementation operations on individual vertices. There are many equivalent characterizations of cographs including being the graphs that contain no induced P_4, and having a cotree representation. A *cotree* is a rooted tree with the leaves being the vertices of the cograph and the internal vertices alternating between being "0" and "1" nodes. Two vertices x, y of a cograph are adjacent if and only if the lowest common ancestor of x and y in the cograph is a "1" node. See Figure 1 for an example of a cograph and its related cotree. Cographs can be extended in the following ways: a graph where each vertex belongs to at most one P_4 is called a *P_4-Reducible graph*; a graph where every set of five vertices induces at most one P_4 is called a *P_4-Sparse graph*. Both P_4-Reducible and P_4-Sparse graphs (as well as distance hereditary graphs) have a tree structure representation that is an extension of cotrees. G is *AT-free* if it contains no AT. A graph is a *permutation graph* if it is the intersection graph of lines whose endpoints are on two parallel lines. Permutation graphs strictly contain cographs and are cocomparability graphs and thus are also AT-free.

A bipartite graph with bipartition (X, Y) is an *interval bigraph* if each vertex v is assigned an interval I_v and $x \in X, y \in Y$ are adjacent if and only if $I_x \cap I_y \neq \emptyset$; an interval bigraph is *proper* if no interval contains another. These graphs are also known as bipartite AT-free graphs and bipartite permutation graphs.

2.2 LBFS

As mentioned in the Introduction, BFS is one of the fundamental graph searching paradigms and can be found in any standard graph theory text. BFS uses a queue to ensure that whenever a vertex x is visited, its previously unvisited neighbours must be visited before its previously unvisited non-neighbours. A *layer* of a BFS is a set of vertices all of the same distance from the initial vertex of the BFS. LBFS is a restriction of BFS; in the following, we present the details of the generic LBFS algorithm and its implementation. We then describe two popular variants of this generic algorithm.

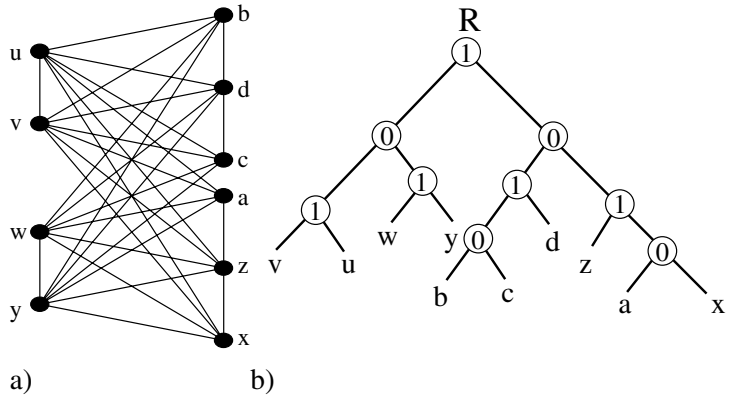

Fig. 1. a) A cograph. b) Its cotree.

Generic LBFS: Note that in the following algorithm, we require the sweep to start at given vertex x; if the algorithm is to start from an arbitrary vertex, then Step 1 is omitted and Step 2 is replaced by assigning Λ to label(y) for all $y \in V$. Note that Step 4 allows the choice of any vertex that has the lexicographically largest label. Later we will present various modifications that explicitly choose the next vertex. We warn the reader that our LBFS ordering of the vertices of the graph may seem "backwards" compared to the ordering produced by other LBFS descriptions.

Procedure LBFS(x)
{Input: Graph $G(V, E)$ and a distinguished vertex x of G;
Output: An ordering σ of the vertices of G.}

1. label(x) $\leftarrow |V|$;
2. **for** each vertex y in $V \setminus \{x\}$ **do** label(y) $\leftarrow \Lambda$;
3. **for** $i \leftarrow |V|$ **downto** 1 **do**
4. pick an unnumbered vertex y with lexicographically the largest label;
5. $\sigma(y) \leftarrow |V| + 1 - i$; {assign to y number $|V| + 1 - i$};
6. **for** each unnumbered vertex z in $N(y)$ **do** append i to label(z).

In an LBFS σ with two arbitrary vertices u and v, if vertex u is visited before v, i.e. $u <_\sigma v$ we say that u *occurs* before v in σ or that u is *visited before* v or that u is *to the left of* v. As mentioned above, this generic LBFS algorithm allows arbitrary choice of a vertex in Step 4. We call the set of tied vertices encountered in Step 4 a *slice* and denote it by S. Note that all vertices of a slice with respect to LBFS σ appear consecutively in σ. Given two vertices u and v of an LBFS σ such that $u <_\sigma v$, $\Gamma^\sigma_{u,v}$ denotes the vertex-minimal slice with respect to σ that contains both u and v. As an example of these concepts consider the graph in Figure 2 where the boxes indicate the slices, including V itself, with respect to the LBFS σ (note that the vertices are numbered as visited by σ). $\Gamma^\sigma_{9,10}$ consists of $\{5, 6, 7, 8, 9, 10\}$.

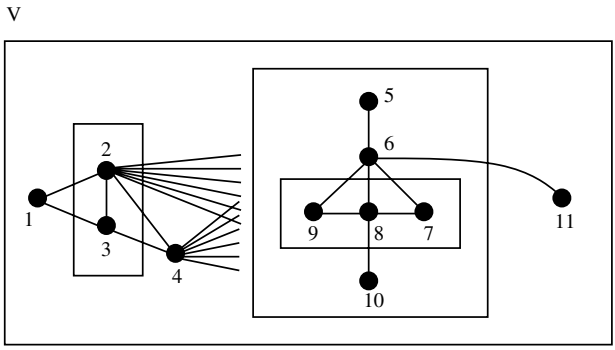

Fig. 2. A graph with its LBFS slices

To implement the generic LBFS algorithm, we use the implementation presented in [24], namely one that follows the paradigm of "partitioning". In this scheme, we start with all vertices in the same cell (i.e., slice) and choose an arbitrary vertex (for reasons that will come clear later, we will choose the first vertex in the cell). When a vertex is chosen, i.e., is chosen as the *pivot*, it is placed in its own cell and invokes a partitioning of all cells that follow it in the ordering. Under this partitioning of a cell, vertices that are adjacent to the pivot form a new cell that precedes the cell containing the vertices not adjacent to the pivot. After this partitioning is complete, a new pivot is chosen from the cell immediately following the old pivot and the process of refinement continues. We refer the reader to Figure 3 for an example of a few steps of partitioning on the graph in Figure 2.

Variants of the Generic LBFS Algorithm: We now describe two variants of the generic LBFS algorithm. In subsequent sections we will reference other variants that have appeared in the literature. In the first case, we break ties in Step 4 by referring to a previous LBFS ordering σ. This variant has been independently investigated by Simon [44] and Ma [32].

Procedure LBFS+ (σ)
{Input: Graph $G(V, E)$ and an LBFS σ of G;
Output: An ordering σ^+ of the vertices of G.}

Do an LBFS of G. When Step 4 is encountered, let S be the set of vertices with the lexicographically largest label. Now y is chosen to be the vertex in S that appears *last* in σ.

As an example, LBFS+ when given the graph in Figure 2 and that LBFS, would produce the order: 11 6 9 8 4 2 7 5 10 3 1.

As pointed out by Lanlignel [30], one of the advantages of using the partitioning implementation of generic LBFS described above, is that we immediately have an implementation of LBFS+. Once σ has been determined, we merely re-

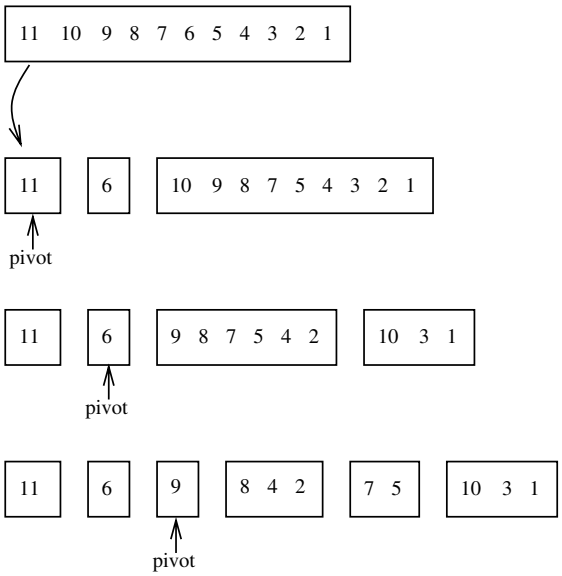

Fig. 3. The first few steps of a partitioning.

verse its ordering of V and run the generic algorithm again. Every time a slice is encountered, the last vertex from σ is automatically the vertex at the front of the list. The example in Figure 3 represents the first few steps of the LBFS+ for the sweep presented in Figure 2.

The second variant produces an LBFS of $\overline{G}$, the complement of graph G. Note that in doing so, we *do not* calculate the complement but rather, as we shall see, slightly modify the generic implementation of LBFS. Furthermore this sweep also requires a previous LBFS as input so that a specific vertex is chosen in Step 4. Note that an overbar placed on an LBFS ordering indicates that the ordering is an LBFS of $\overline{G}$.

Procedure LBFS$^-$ (σ)
{Input: Graph $G(V, E)$ and an LBFS σ of G;
Output: An ordering $\overline{\sigma}^-$ of the vertices of $\overline{G}$.}

Do an LBFS of $\overline{G}$. When Step 4 is encountered, let S be the set of vertices with the lexicographically largest label. Now y is chosen to be the vertex in S that appears *first* in σ.

An example of this algorithm will appear in Section 4, in the example of the Cograph recognition algorithm. As noted in [33, 24], an LBFS of $\overline{G}$ can be done in $O(n + m)$ time by making a slight modification of the implementation of the generic LBFS algorithm. In particular, during the partitioning, the cell containing the vertices adjacent to the pivot is placed *after* the cell containing

the nonadjacent vertices. To make sure that the correct vertex in S is chosen, we merely input σ in its natural order. By choosing the first vertex of every slice as the next pivot, we automatically meet the choice requirement of the algorithm.

3 Structure Results

In this section we present some of the important structure results concerning LBFS. For most families of graphs where LBFS has been used, there are particular results that are unique to that family. The first result is the following characterization of vertex orderings that can be achieved by an LBFS. This characterization is used heavily in the various multi-sweep LBFS algorithms and it is somewhat surprising to note that, except for Maximum Cardinality Search (MCS), similar characterizations for other well known graph searches have only recently been discovered [16].

Theorem 1. *[22] An ordering $\prec$ of the vertices of an arbitrary graph $G(V, E)$ is an LBFS ordering if and only if for all vertices a, b, c of G such that $ac \in E$ and $bc \notin E$, $c \prec b \prec a$ implies the existence of a vertex d in G, adjacent to b but not to a and such that $d \prec c$.*

The following lemma establishes the existence of special paths in $\Gamma^{\sigma}_{u,v}$.

Lemma 1. *[17] (The Prior Path Lemma) Let σ be an arbitrary LBFS of a graph G. Let t be the first vertex of the connected component of $\Gamma^{\sigma}_{u,v}$ containing u. There exists a t, u-path in $\Gamma^{\sigma}_{u,v}$ all of whose vertices, with the possible exception of u, are missed by v. Moreover, all vertices on this path, other than u, occur before u in σ. (Such a path is called a* prior path*).*

As an example of this Lemma, consider the graph in Figure 2 and let $u = 7, v = 10$. Now $\Gamma^{\sigma}_{7,10} = \{5, 6, 7, 8, 9, 10\}$ and path $7-6-5$ is a prior path for this choice of u, v.

In the fundamental paper by Rose, Tarjan and Lueker [42], their chordal graph recognition algorithm was based on the following theorem.

Theorem 2. *[42] Let σ be an LBFS of a chordal graph G and let v be an arbitrary vertex of G. Let W denote the set of vertices w that occur before v in σ. Then v is simplicial in the subgraph of G induced by $W \cup \{v\}$.*

From this theorem, we immediately see that the reverse ordering of an LBFS of a chordal graph G yields a PEO of G. Berry and Bordat [2] have generalized this theorem as follows:

Theorem 3. *[2] Let σ be an LBFS of a chordal graph G and let v be an arbitrary vertex of G. Let W denote the set of vertices w that occur before v in σ. Then v belongs to a moplex in the subgraph of G induced by $W \cup \{v\}$.*

Furthermore they showed that the vertices in the moplex containing v are consecutive vertices in σ up to and including v.

Interestingly, a very similar result to Theorem 2 holds for an arbitrary LBFS in an AT-free graph. In particular,

Theorem 4. *[18] Let σ be an LBFS of an AT-free graph G and let v be an arbitrary vertex of G. Let W denote the set of vertices w that occur before v in σ. Then v is admissible in the subgraph of G induced by $W \cup \{v\}$.*

Again, this theorem yields the result that the reverse ordering of an LBFS of an AT-free graph G yields an AEO of G. Unfortunately, the existence of an AEO for a graph G does not imply that G is AT-free. More will be said about this issue later in this section. Given any subset of vertices X in either a chordal or an AT-free graph, these two theorems show the importance of x, the last vertex of X, in any LBFS. In particular, such a vertex is guaranteed to be simplicial (respectively, admissible) in the subset of vertices that have occurred up to and including x and thus also in X itself. In many multi-sweep LBFS algorithms, we want to "break ties" by choosing a vertex with a particular property. LBFS+, the algorithm that starts a slice S with the last S vertex in the previous sweep, was developed for precisely this reason and, as we shall see in Subsection 4.1, is currently the most popular restricted version of LBFS in multi-sweep LBFS algorithms.

To formalize the notion of "last vertex" mentioned above, we define a vertex x to be an *end-vertex* of graph G if there is an LBFS of G that ends at x. Is it possible that end-vertices of a graph can be characterized? For interval graphs, the answer is affirmative as shown in the following Lemma.

Lemma 2. *[17] A vertex in an interval graph is an end-vertex if and only if it is simplicial and admissible.*

This result can be extended to arbitrary graphs in the following way.

Lemma 3. *[15] Let G be an arbitrary graph. If x is a simplicial and admissible vertex of G, then x is an end-vertex.*

Unfortunately, it seems unlikely that there is a nice characterization of end-vertices for arbitrary graphs, as shown by the following complexity result.

Theorem 5. *[15] Given a graph G and vertex x, it is NP-complete to determine whether x is an end-vertex of G.*

Furthermore, the problem remains NP-complete for weakly chordal graphs, is linearly time solvable for interval graphs (using an LBFS followed by an LBFS+) and remains unresolved for both chordal and AT-free graphs [15].

As we shall see, most LBFS based algorithms involve a number of LBFS sweeps and thus require some knowledge of the behaviour of parts of the graph in previous sweeps. Typically such arguments are based on either the behaviour of paths (where the Prior Path Lemma is fundamental) or the behaviour of slices, which we now discuss. In particular, we look at the restriction of an LBFS to a slice from some other LBFS. The strongest result of this type is for chordal graphs.

Lemma 4. *[17](The LBFS Lemma) Let G be a chordal graph and let S be a slice of an arbitrary LBFS ordering τ of G. Further let σ be another arbitrary LBFS ordering of G. Then the restriction of σ to S is an LBFS ordering of the graph induced by the vertices of S.*

As stated in [17], "to put this lemma in perspective, it is important to note that the desired property does not hold for arbitrary subsets of chordal (or even interval) graphs. For example, consider the interval graph shown in Figure 4. The numbering of the vertices indicates a legitimate LBFS ordering; however when vertex 1 is removed, the restriction of this ordering to the remaining subset is not a legitimate LBFS ordering of the subset. Also, as shown in Figure 5, the lemma does not hold for AT-free graphs. $S = \{2, 3, 4\}$ is a slice of the LBFS: 1 2 3 4 5. Now consider an arbitrary LBFS starting at 5. Vertex 3 occurs after 2 and 4, which cannot occur in an LBFS of S."

It is somewhat surprising and disappointing that the LBFS Lemma does not generalize to AT-free graphs. Nevertheless, there is something that can be said in this regard for AT-free graphs. First we note the following obvious Corollary of Theorem 2 and Lemma 4.

Corollary 1. *Let G be a chordal graph with S a slice with respect to LBFS σ. Then for every LBFS τ of G, x, the last vertex of τ_S, is an end-vertex of S.*

As shown in [15], there is a similar result for AT-free graphs.

Lemma 5. *[15] Let G be an AT-free graph with S a slice with respect to LBFS σ. Then for every LBFS τ of G, x, the last vertex of τ_S, is either an end-vertex of S or is adjacent to an end-vertex of S.*

Finally, to end this section, we mention a pair of graph families that are characterized by properties of *every* LBFS. The reader is cautioned that a statement of the form: "G is an X-graph if and only if every LBFS has property P" can be quite misleading in the sense that "every" can either be interpreted as "an arbitrary" (for example, chordal and distance hereditary graphs, as discussed in the next Section) or "all", (for example, HHD-free and AT-free graphs) as we now present.

Lemma 6. *[27] G is an HHD-free graph if and only if all LBFSs are semiperfect elimination orderings.*

Lemma 7. *[15] G is an AT-free graph if and only if all LBFSs are admissible elimination orderings.*

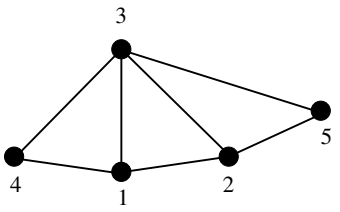

Fig. 4. The LBFS Lemma does not hold for arbitrary subsets of interval graphs.

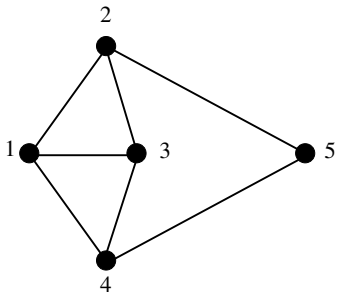

Fig. 5. The LBFS Lemma does not hold for AT-free graphs.

4 Applications of LBFS

In this section, we survey many of the applications of LBFS. The most notable application, presented in Subsection 4.1, is the recognition of various restricted graph classes. For many graph classes, the current "best" recognition algorithm is based on LBFS, usually in the form of a multi-sweep algorithm. In Subsection 4.2, we will show other diverse applications of LBFS including diameter approximation for various graph classes and the determination of a dominating pair in an AT-free graph.

4.1 Recognition of Various Graph Classes

LBFS was discovered in the development of a simple, linear time chordal graph recognition algorithm [42]. The algorithm is based on the following fundamental result:

Theorem 6. *[42] A graph is chordal if and only if an arbitrary LBFS yields a perfect elimination ordering.*

Thus the associated chordal graph recognition algorithm is as follows:

The *Chordal graph* Recognition Algorithm[42]
{Input: Graph $G(V, E)$;
Output: A statement declaring whether or not G is a *chordal graph.*}

1. Do an arbitrary LBFS σ.
2. If the reverse of σ is a perfect elimination ordering, then conclude that G is a chordal graph; else, conclude that G is not a chordal graph.

Since determining whether a particular ordering is a perfect elimination ordering can be accomplished in linear time, the algorithm has a straightforward linear time implementation. In a subsequent paper, Tarjan and Yannakakis [43] extended this algorithm to be certifying by showing how to find, in linear time, an induced cycle of size greater than three if the reverse of σ is not a perfect elimination ordering.

Interestingly there is another family of graphs that has the same single LBFS recognition algorithm. In particular, Dragan and Nicolai [21] proved the following theorem:

Theorem 7. *[21] A graph is distance hereditary if and only if an arbitrary LBFS yields a 2-simplicial elimination ordering.*

As with chordal graph recognition, there is an associated distance hereditary graph recognition algorithm.

Although this is a very simple algorithm, it does not seem to have a linear time implementation since it is not clear how one can determine in linear time whether the reverse of σ is a 2-simplicial elimination ordering. Later in this

subsection, we will discuss a linear time multi-sweep LBFS distance hereditary graph recognition algorithm.

We now turn our attention to LBFS recognition algorithms that require at least two LBFS sweeps. In presenting such multi-sweep recognition algorithms, we will take two "basic" algorithms for the recognition of unit interval graphs and cographs, respectively, and show how modifications of these algorithms can lead to recognition algorithms for related graph families. All of these algorithms are easily implementable in linear time. Typically, they are not the first linear time algorithm for the particular recognition problem but they are simpler than the previous non-LBFS algorithms. References to these other algorithms are contained in the appropriate reference describing the LBFS algorithm.

Unit Interval Graphs: The LBFS based unit interval graph recognition algorithm [11] is the following:

The *Unit Interval graph* Recognition Algorithm[11]
{Input: Graph $G(V, E)$;
Output: A statement declaring whether or not G is a *unit interval graph.*}

1. Do an arbitrary LBFS σ.
2. LBFS+ (σ) yielding sweep σ^+.
3. LBFS+ (σ^+) yielding sweep σ^{++}.
4. If σ^{++} satisfies a "particular condition", then conclude that G is a unit interval graph; else, conclude that G is not a unit interval graph.

In the case of unit interval graphs, the "particular condition" to be tested is the "Neighbourhood Condition", namely that $G(V, E)$ is a unit interval graph if and only if there is an ordering of V such that for all $v \in V$, $N[v]$ (the closed neighbourhood of v) is consecutive. For this and other characterizations of unit interval graphs, see [39], [40] and [31].

This algorithm is not "certifying" in the sense that if the input graph fails the Neighbourhood Condition and the algorithm concludes that the input graph is not a unit interval graph, then there is no immediate "proof" that the graph is in fact not a unit interval graph. Note, that the algorithm does certify the conclusion that the graph is a unit interval graph since it is easy to build a unit interval model if the Neighbourhood Condition is satisfied. Recently, two algorithms have been developed to provide a certificate of nonmembership. The first, by Meister [34], is similar to the above algorithm in that it uses three LBFS sweeps with the second and third sweeps using "min-LexBFS" which requires a special implementation rather than LBFS+ which, as pointed out in Subsection 2.2, has an immediate partitioning implementation. The certificate that Meister's algorithm produces is either an induced cycle of size greater than 3 (thereby showing that the graph is not chordal, and thus not interval) or an AT (showing that the graph is not AT-free, and thus not interval) or a claw (i.e., $K_{1,3}$). The second certifying algorithm, by Hell and Huang [25], augments the algorithm presented above and uses Wegner's characterization of unit interval graphs [45],

namely that a graph is a unit interval graph if and only if it does not contain an induced cycle of size greater than 3, a claw, or a "3-sun" or its complement, the "net" which consists of a triangle each of whose vertices is adjacent to a unique vertex of degree 1. One of the pretty aspects of the Hell and Huang algorithm is that it incorporates the certificate steps throughout the algorithm, in the sense that it does some testing after each of the three LBFS steps, and only proceeds to the next sweep if the test has been satisfied. Furthermore, they also show that the algorithm presented above can be augmented to provide a linear time certifying recognition algorithm for proper interval bigraphs. Again the certification is distributed throughout the algorithm. Chang, Ho and Ko [9] have presented a linear time 2-sweep LBFS based algorithm for recognizing bipartite permutation graphs (equivalent to proper interval bigraphs). Their algorithm modifies the second LBFS sweep to break ties according to the value of a degree related function.

One of the early uses of LBFS appeared in the Korte - Möhring interval graph recognition algorithm [29]. By using LBFS, they were able to streamlne the first linear time interval graph recognition algorithm by Booth and Lueker [3]. We now present a second extension of the unit interval graph recognition algorithm that provides an easily implementable, linear time recognition algorithm for interval graphs. This algorithm is as follows:

The *Interval graph* Recognition Algorithm[17]
{Input: Graph $G(V, E)$;
Output: A statement declaring whether or not G is an *interval graph.*}

1. Do an arbitrary LBFS π.
2. LBFS+ (π) yielding sweep σ.
3. LBFS+ (σ) yielding sweep σ^+.
4. LBFS+ (σ^+) yielding sweep σ^{++}.
5. LBFS* (σ^+, σ^{++}) yielding sweep σ^*.
6. If σ^* satisfies the "Interval Graph Ordering Condition" then conclude that G is an interval graph; else, conclude that G is not an interval graph.

The "interval Graph Ordering Condition" states that a graph $G(V, E)$ is an interval graph if and only if there is a linear ordering $\prec$ on V such that for every choice of vertices $u,\ v,\ w$, with $u \prec v$ and $v \prec w$, $uw \in E \implies uv \in E$ [26, 36, 38, 37]. LBFS* requires *two* previous sweeps and breaks ties for a slice S by examining the last vertices of S in each of these two sweeps. Since LBFS* and the "Interval Graph Ordering Condition" can easily be implemented in linear time, the entire algorithm is easily implementable in linear time. (See [17] for further details.) Recently Choi and Farach-Colton [10] have used this algorithm to develop a new interval graph based algorithm for the sequence assembly problem that is significantly superior to existing algorithms. It is interesting to note that Simon ([44]) incorrectly claimed that terminating the algorithm after the third LBFS+ would suffice to recognize interval graphs. Ma [32], however, showed that Simon's algorithm is flawed and that for any constant c, there is an interval

graph, and an initial LBFS ordering such that after c applications of LBFS+, the linear ordering of vertices is still not apparent! It is known, however, that using n applications of LBFS+ will work, but of course not in linear time [14].

Cographs: Our second basic algorithm is the one for cographs. The generic algorithm is as follows:

The *Cograph* Recognition Algorithm[8]
{Input: Graph $G(V, E)$;
Output: A cotree if G is a *cograph*, or an induced P_4 otherwise.}

1. Do an arbitrary LBFS σ.
2. LBFS$^-(\sigma)$ yielding sweep $\overline{\sigma}^-$ (of $\overline{G}$).
3. LBFS$^-(\overline{\sigma}^-)$ yielding sweep σ^- (of G).
4. If $\overline{\sigma}^-, \sigma^-$ satisfy a "particular condition", then CONSTRUCT_COTREE; else REPORT_P_4.

In the case of cograph recognition, the "particular condition" is the "Neighbourhood Subset Property", a property that can easily be checked in linear time, yielding an easily implementable linear time algorithm. See [7, 8] for further details. As an example of the algorithm, consider the cograph in Figure 6. The two LBFS$^-$ sweeps satisfy the "Neighbourhood Subset Property" and the algorithm produces the cotree.

The first extension of this cograph recognition algorithm is to P_4-Reducible graphs, namely those graphs where each vertex belongs to at most one P_4. The algorithm is as follows:

The P_4-*Reducible graph* Recognition Algorithm[7]
{Input: Graph $G(V, E)$;
Output: A P_4-R_tree, if G is a P_4-*Reducible graph*, or two P_4s containing one vertex otherwise.}

1. Do an arbitrary LBFS σ.
2. LBFS+ (σ) yielding sweep σ^+.
3. LBFS$^-(\sigma^+)$ yielding sweep $\overline{\sigma}^-$ (of $\overline{G}$).
4. LBFS$^-(\overline{\sigma}^-)$ yielding sweep σ^- (of G).
5. If $\overline{\sigma}^-, \sigma^-$ satisfy a "particular condition", then CONSTRUCT_P_4-R_TREE; else REPORT_MULTIPLE_P_4.

The appropriate "particular condition" for this algorithm is the "P_4-Reducible Neighbourhood Property" described in [7]. This condition is easily tested in linear time.

P_4-Sparse graphs, namely graphs for which no set of five vertices induces more than one P_4, generalize P_4-Reducible graphs and have a very similar recognition algorithm.

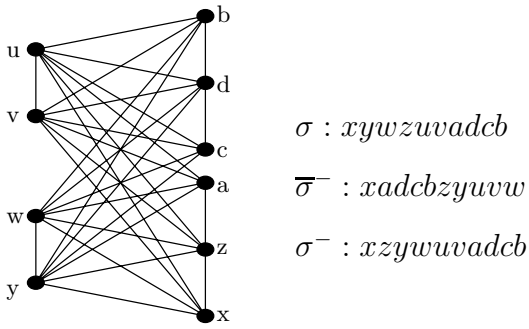

Fig. 6. The execution of the algorithm on the cograph in Figure 1.

The P_4-*Sparse graph* Recognition Algorithm[7]
{Input: Graph $G(V, E)$;
Output: A P_4-S_tree, if G is a P_4-*Sparse graph*, or a set of five vertices inducing two P_4s otherwise.}

1. Do an arbitrary LBFS σ.
2. LBFS+ (σ) yielding sweep σ^{+}.
3. LBFS^{-}(σ^{+}) yielding sweep $\overline{\sigma}^{-}$ (of $\overline{G}$).
4. LBFS^{-}($\overline{\sigma}^{-}$) yielding sweep σ^{-} (of G).
5. If $\overline{\sigma}^{-}, \sigma^{-}$ satisfy a "particular condition", then CONSTRUCT_P_4-S_TREE; else REPORT_5_SET.

The appropriate "particular condition" for this algorithm is the "P_4-Sparse Neighbourhood Property" described in [7]. Again an easily implementable linear time algorithm is obtained.

Finally we turn to a new LBFS based recognition algorithm for distance hereditary graphs. Given Theorem 7, it is not surprising that cographs play a critical role in characterizing distance hereditary graphs. In particular, Bandelt and Mulder [1] showed that every layer of any BFS of a distance hereditary graph is a cograph and that there are specific neighbourhood intersection conditions inside and between layers. Bretscher [7] has shown that these conditions can be expressed by neighbourhood conditions on LBFS slices in BFS layers to produce an LBFS characterization of distance hereditary graphs. By using this characterization and the new LBFS cograph recognition algorithm, she has produced a new simpler linear time LBFS based distance hereditary graph recognition algorithm [7].

4.2 Other Applications

These applications vary from diameter approximation for various families of graphs to the determination of a dominating pair in an AT-free graph to common properties of powers of graphs.

Diameter Approximation: Determining the diameter of a graph is a fundamental graph property whose current best algorithm (i.e. $O(nm)$) is too slow for very large input graphs. This naive algorithm performs a BFS from each vertex x, thereby calculating the eccentricity, $ecc(x)$ of x. The diameter is then determined by finding the maximum eccentricity of any vertex in G. Since any BFS from a vertex of maximum eccentricity immediately produces the diameter of the graph, one approach to approximating a graph's diameter is to search for a vertex of high eccentricity. The most common way of finding such a vertex has been to take the end-vertex of a specific search from an arbitrary vertex. The searches that have been considered are BFS, LBFS, LL and LL+ where LL chooses an arbitrary vertex in the last BFS layer and LL+ chooses an arbitrary vertex in the last BFS layer that has minimum degree into the second last BFS layer. For the restricted graph families considered in [13], none of BFS, LL or LL+ beat LBFS. In particular, the eccentricity of an LBFS end-vertex is guaranteed: to be $diam(G)$ for interval graphs [22] and {AT,claw}– free graphs [4]; to be at least $diam(G) - 1$ for chordal [22] and AT-free [12] graphs; and to be at least $diam(G) - 2$ for graphs that contain no induced cycles of size greater than 4 [13]. Dragan [19] presented similar LBFS results on other restricted families of graphs. Corneil et al [12] also looked at the end-vertex of a "double-sweep" LBFS algorithm (see the following Dominating Pair Algorithm) on chordal and AT-free graphs. They established a forbidden subgraph structure on chordal or AT-free graphs where $diam(G) - 1$ is the eccentricity of the end-vertex of the second sweep. They also showed examples of chordal and AT-free graphs where for no c, the "c-sweep" LBFS algorithm is guaranteed to find a vertex of maximum eccentricity. Furthermore, for any c there is a graph G (albeit with large induced cycles whose size depends on c) where the eccentricity of the chosen vertex is at least c away from the diameter of G.

In a related approach, Dragan [20] showed how particular vertex orderings (including LBFS) can be used to appoximate the All Pairs Shortest Path problem to within a small additive constant for various restricted families of graphs.

Dominating Pairs in AT-free Graphs: One of the first indications that LBFS has far-reaching applications beyond families of graphs related to chordal graphs came in the 2-sweep algorithm for finding a dominating pair in a connected AT-free graph. The algorithm is as follows:

The *Dominating Pair* Algorithm[18]
{Input: A connected AT-free graph $G(V, E)$;
Output: A pair of vertices x, y that form a dominating pair of G.}

1. Do an arbitrary LBFS σ where x is the end-vertex of σ.
2. Do an LBFS(x) τ where y is the end-vertex of τ.
3. Return x, y.

As noted in [18], this algorithm can be modified to return, in linear time, a succinct representation of *all* dominating pairs in any connected AT-free graph of

diameter greater than 3. In particular, if $diam(G) > 3$, then there are nonempty, disjoint sets X and Y of vertices of G, such that x, y is a dominating pair if and only if $x \in X$ and $y \in Y$. The algorithm returns the sets X and Y.

Properties of Powers of Graphs: In families of graphs related to chordal graphs, considerable attention has been given to the problem of determining the graph class membership of various powers of a given graph. For example, it is known that every odd power of a chordal graph is chordal. This result is an immediate corollary of the following theorem:

Theorem 8. *[5] The reversal of every LBFS ordering of a chordal graph G is a common perfect elimination ordering of all odd powers of G.*

A similar result is captured in the following theorem:

Theorem 9. *[21] The reversal of every LBFS ordering of a distance hereditary graph G is a perfect elimination ordering of every even power $G^{2k}, k \geq 1$.*

Note that the result in Theorem 8 does not imply that the reversal of an LBFS ordering of G is also a reversal of an LBFS ordering of odd powers of G [16]. In fact, there are chordal graphs where no LBFS ordering is also an LBFS ordering of any powers of the graph. On the other hand, every LBFS ordering of a chordal bipartite graph is also an LBFS ordering of its square, a property that does not hold for bipartite graphs [16].

In a similar vein, Chang, Ho and Ko [9] with respect to LBFS orderings and strong 2-CCPOs, considered τ, the ordering produced by the second LBFS in the Dominating Pair Algorithm and proved:

Theorem 10. *[9] Given an AT-free graph G, τ is a strong 2-CCPO of G.*

5 Concluding Remarks

One of the surprising observations in the development of LBFS based algorithms is that LBFS works so well on both chordal and AT-free related families of graphs, yet these two families have very little structural similarity (other than the absence of large induced cycles). Is there some generalization of these two families that explains the success of LBFS?

Although there is now a very impressive list of graph families whose recognition is best achieved using an LBFS approach, there have been a number of graph families, especially strongly chordal, chordal bipartite and permutation graphs, that so far have resisted this approach. Currently no linear time recognition algorithm is known for the family of strongly chordal graphs. Chordal bipartite graphs are a closely related family (see [6]) that has also resisted linear time recognition. Although there are linear time recognition algorithms for permutation graphs, it is possible that there is a simpler LBFS based one.

Given the number and power of multi-sweep LBFS algorithms, it is somewhat surprising that other graph searches have seldom been used in a multi-sweep fashion. One reason for this may be that, until recently [16], most of them do

not have a vertex ordering characterization similar to that for LBFS presented in Theorem 1. This also raises the possibility of multi-sweep hybrid algorithms where BFS and DFS variants are combined. Finally, given the power of the "lexicographic" extension of BFS, it is natural to wonder whether DFS would benefit from a similar extension; a candidate algorithm for Lexicographic Depth First Search is presented in [16].

Acknowledgements

Financial assistance from the Natural Sciences and Engineering Research Council of Canada was gratefully received. Many thanks to Anna Bretscher, Feodor Dragan, Ekki Koehler and Lorna Stewart for their improvements of an earlier draft of this paper.

References

1. H-K. Bandelt and H.M. Mulder: Distance-hereditary graphs, J. Combin. Theory B 41 (1986) 182–208.
2. A. Berry and J-P. Bordat: Separability generalizes Dirac's theorem, Disc. Appl. Math. 84 (1998) 43–53.
3. K.S. Booth and G.S. Lueker: Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms, J. Comput. System Sci. 13 (1976) 335–379.
4. A. Brandstädt and F.F. Dragan: On linear and circular structure of a (claw,net)-free graph, Disc. Appl. Math. 129 (2003) 285–303.
5. A. Brandstädt, F.F. Dragan and F. Nicolai: LexBFS-orderings and powers of chordal graphs, Disc. Math. 171 (1997) 27–42.
6. A. Brandstädt, V.B. Le and J.P. Spinrad: Graph Classes: A Survey, SIAM Monographs on Disc. Math. and Applic., SIAM 1999.
7. A. Bretscher: LexBFS based recognition algorithms for cographs and related families, Ph.D. thesis in preparation, Dept. of Computer Science, University of Toronto, Toronto, Canada.
8. A. Bretscher, D.G. Corneil, M. Habib and C. Paul: A simple linear time LexBFS cograph recognition algorithm (extended abstract), LNCS 2880 (2003) 119–130.
9. J-M. Chang, C-W. Ho and M-T. Ko: LexBFS-ordering in Asteroidal Triple-free graphs, LNCS 1741 (1999) 163–172.
10. V. Choi and M. Farach-Colton: Barnacle: an assembly algorithm for clone-based sequences of whole genomes, Gene 320 (2003) 165–176.
11. D.G. Corneil, A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs: Disc. Appl. Math. 138 (2004) 371–379.
12. D.G. Corneil, F.F. Dragan, M. Habib and C. Paul: Diameter determination on restricted graph families, Disc. Appl. Math. 113 (2001) 143–166.
13. D.G. Corneil, F.F. Dragan and E. Koehler: On the power of BFS to determine a graph's diameter, Networks 42 (2003) 209–222.
14. D.G. Corneil and E. Koehler: unpublished manuscript.
15. D.G. Corneil, E. Koehler and J-M. Lanlignel: On LBFS end-vertices, in preparation.

16. D.G. Corneil and R. Krueger: A unified view of graph searching, in preparation.
17. D.G. Corneil, S. Olariu, and L. Stewart: The LBFS structure and recognition of interval graphs, under revision; extended abstract appeared as The ultimate interval graph recognition algorithm? (extended abstract) in Proc. SODA 98, Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (1998) 175–180.
18. D.G. Corneil, S. Olariu, and L. Stewart: Linear time algorithms for dominating pairs in asteroidal triple-free graphs, SIAM J. Comput. 28 (1999) 1284–1297.
19. F.F. Dragan: Almost diameter of a house-hole-free graph in linear time via LexBFS, Disc. Appl. Math. 95 (1999) 223–239.
20. F.F. Dragan: Estimating all pairs shortest paths in restricted graph families: a unified approach (extended abstract), LNCS 2204 (2001) 103–116.
21. F.F. Dragan and F. Nicolai: Lex-BFS-orderings of distance-hereditary graphs, Schriftenreihe des Fachbereichs Mathematik der Universität Duisburg, Duisburg, Germany, SM-DU-303 (1995).
22. F.F. Dragan, F. Nicolai, and A. Brandstädt: LexBFS-orderings and powers of graphs, LNCS 1197 (1997) 166–180.
23. D.R. Fulkerson and O.A. Gross: Incidence matrices and interval graphs, Pacific J. Math. 15 (1965) 835–855.
24. 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) 59–84.
25. P. Hell and J. Huang: Certifying LexBFS recognition algorithms for proper interval graphs and proper interval bigraphs, to appear SIAM J. Disc. Math. (2004).
26. M.S. Jacobson, F.R. McMorris, and H.M. Mulder: Tolerance intersection graphs, in 1988 International Kalamazoo Graph Theory Conference, Y. Alavi et al, eds., Wiley 1991 705–724.
27. B. Jamison, S. Olariu: On the semi-perfect elimination, Advances in Appl. Math. 9 (1988) 364–376.
28. D. Kratsch and L. Stewart: Domination on cocomparability graphs, SIAM J. Disc. Math. 6 (1993) 400–417.
29. N. Korte and R.H. Möhring: An incremental linear-time algorithm for recognizing interval graphs, SIAM J. Comput. 18 (1989) 68–81.
30. J-M. Lanlignel: Private communications, 1999.
31. P.J. Looges and S. Olariu: Optimal greedy algorithms for indifference graphs, Computers Math. Applic. 25 (1993) 15–25.
32. T. Ma: unpublished manuscript.
33. R.M. McConnell and J. Spinrad: Linear-time modular decomposition and efficient transitive orientation of comparability graphs, Proc. SODA 94, Fifth Annual ACM-SIAM Symposium on Discrete Algorithms (1994) 536–545.
34. D. Meister: Recognizing and computing minimal triangulations efficiently, Technical Report 302, Fakultät für Mathematik und Informatik, Universität Würzburg, 2002.
35. F. Nicolai: A hypertree characterization of distance-hereditary graphs, manuscript, Gerhard-Mercator-Universität Duisburg (1996).
36. S. Olariu: An optimal greedy heuristic to color interval graphs, Inform. Process. Lett. 37 (1991) 65–80.
37. G. Ramalingam and C. Pandu Rangan: A uniform approach to domination problems on interval graphs, Inform. Process. Lett., 27 (1988) 271–274.
38. A. Raychaudhuri: On powers of interval and unit interval graphs, Congr. Numer. 59 (1987) 235–242.

39. F.S. Roberts: Indifference graphs, in: F. Harary, ed., Proof Techniques in Graph Theory, Academic Press, New York, 1969 139–146.
40. F.S. Roberts: On the compatibility between a graph and a simple order, J. Combin. Theory Ser. B 11 (1971) 28–38.
41. R.E. Tarjan: Depth first search and linear graph algorithms, SIAM J. Comput. 1 (1972) 146–160.
42. D.J. Rose, R.E. Tarjan, and G.S. Lueker: Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput. 5 (1976) 266–283.
43. 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) 566–579.
44. K. Simon: A new simple linear algorithm to recognize interval graphs, LNCS 553 (1992) 289–308.
45. G. Wegner: Eigenschaften der Nerven homologisch-einfacher Familien in R^n, Ph.D. thesis, Universität Göttigen, Germany, 1967.