A General Label Search to Investigate Classical Graph Search Algorithms

Geneviève Simonet, Anne Berry, Richard Krueger

Discrete Applied Mathematics 159 (2011) 128–142

Contents lists available at ScienceDirect

Discrete Applied Mathematics

journal homepage: www.elsevier.com/locate/dam

A General Label Search to investigate classical graph search algorithms

R. Krueger[a], G. Simonet[b,*], A. Berry[c]

[a] *Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G4*
[b] *LIRMM, Univ. Montpellier 2, CNRS, 161, Rue Ada, F-34392 Montpellier, France*
[c] *LIMOS, Ensemble scientifique des Cézeaux, F-63177 Aubière, France*

ARTICLE INFO

Article history:
Received 26 June 2008
Received in revised form 17 February 2010
Accepted 27 February 2010
Available online 18 November 2010

Keywords:
Graph search algorithm
Graph labeling algorithm
Breadth First Search
Depth First Search
LexBFS

ABSTRACT

Many graph search algorithms use a labeling of the vertices to compute an ordering of the vertices. We generalize this idea by devising a general vertex labeling algorithmic process called General Label Search (GLS), which uses a labeling structure which, when specified, defines specific algorithms.

We characterize the vertex orderings computable by the basic types of searches in terms of properties of their associated labeling structures. We then consider performing graph searches in the complement without computing it, and provide characterizations for some searches, but show that for some searches such as the basic Depth-First Search, no algorithm of the GLS family can exactly find all the orderings of the complement. Finally, we present some implementations and complexity results of GLS on a graph and on its complement.

1. Introduction

Graph search algorithms are fundamental in computer science. The well-known Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms are commonly used by algorithms as a simple method for systematically visiting all the vertices or all the edges of a graph. More specialized searches, including Lexicographic Breadth-First Search (LexBFS), are widely used for tasks such as the recognition of various graph classes.

Graph Search computes an ordering of the vertices of a graph G in the following manner, as described by Tarjan [11]. We pick an arbitrary vertex and select one of its incident edges to follow. 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. The way in which we choose the next edge to traverse determines what kind of search is performed (such as whether it is a BFS or DFS).

Corneil and Krueger [5] unified the basic paradigms of graph search by deriving characterizations of the vertex orderings each algorithm may produce. The previously known characterization of LexBFS orderings [8,2,3] has become extremely important in devising and proving LexBFS-based algorithms in a variety of areas [4]. Such a characterization has proven valuable for devising multi-sweep search algorithms, where several graph searches are performed in sequence, with each sweep possibly using information gathered in the previous sweep(s). The new characterizations of [5] permit the same

* Corresponding author. Fax: +33 467144176.
E-mail addresses: krueger@cs.toronto.edu (R. Krueger), simonet@iutmontp.univ-montp2.fr, simonet@lirmm.fr (G. Simonet), berry@isima.fr (A. Berry).

0166-218X/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.dam.2010.02.011

techniques currently exploited for LexBFS to be expanded to all other basic searches, allowing the full power of graph search to be applied to new areas.

Recently, Berry, Krueger and Simonet [1] studied graph search in the context of computing minimal triangulations, and introduced an algorithm, called MLS, to capture all graph searches that compute a perfect elimination ordering (peo) when run on a chordal graph.

In this paper, we present a more general search paradigm, which we call GLS for General Label Search. We extend the results of [1] by deriving a more general form of MLS where there is no condition on the function which is used to modify the labels. GLS thus captures a much wider variety of graph algorithms, since we do not require that a peo be computed when the algorithms are applied to chordal graphs. We thus present labeling structure properties and use them in new characterizations of the vertex orderings computable by the different types of search. We find that GLS can also express some algorithms other than a search of the input graph. We characterize labeling structures that yield certain types of graph search orderings of the complement of the graph, which are found without computing the complement. We also show that no algorithm of the GLS family can find precisely the general graph search orderings nor the depth-first search orderings of the complement of a graph. Finally, we discuss some efficient implementations of GLS.

The paper is organized as follows: after this introduction, we give some preliminary notions in Section 2. Section 3 presents algorithmic paradigm GLS, Section 4 shows that seven classical graph search algorithms are instances of GLS. Section 5 studies the relationship between some properties of labeling structures and the type of vertex ordering computed by GLS. Section 6 shows how to use GLS to perform different types of search in the complement of a graph, and Section 7 presents some implementations of GLS and complexity results.

2. Preliminaries

We consider simple undirected graphs $G = (V, E)$ with $n = |V|$ and $m = |E|$. The *neighborhood* of a vertex v in G is denoted by $N_G(v)$, or simply $N(v)$ if G is understood. An *ordering* of V (or of G) is a bijection σ from $\{1, \dots, n\}$ to V. We use shorthand $\sigma = (v_1, \dots, v_n)$ to mean $\sigma(i) = v_i$ for $1 \leq i \leq n$. When speaking of orderings computed by an algorithm, v_1 is the first vertex which is visited (i.e. the first vertex to receive its number) and v_n is the last vertex to be visited; some other papers in the literature generate orderings in the reverse order, i.e. the first vertex visited receives number n and the last vertex visited receives number 1 [1,9,12]. This is the case in particular for algorithms which compute perfect elimination orderings, as the last visited vertex will be the first to be eliminated in the corresponding elimination ordering. We denote by $\mathbb{N}^+$ the set of positive integers $\{1, 2, 3, \dots\}$ and for any integer $i \geq 0$, we denote by $\mathbb{N}_i^+$ the set of positive integers which are smaller or equal to i, i.e. $\{1, 2, 3, \dots, i\}$. The *power set* of a set X is the set of subsets of X. A *chordal* graph is a graph with no chordless cycle of length greater or equal to 4. A *partial order*, or simply *order*, on a set X is a reflexive, antisymmetric and transitive binary relation on X. An order $\preceq$ on X is *total* if $\forall x, y \in X, x \preceq y$ or $y \preceq x$. A *linear extension* of an order is a total super-order of this order. A *strict order* on X is an irreflexive, antisymmetric and transitive binary relation on X. With each order $\preceq$ on X is associated a strict order $\prec$ on X defined by: $\forall x, y \in X, x \prec y$ iff ($x \preceq y$ and $x \neq y$). The *dual* (or *reverse*) of an order $\preceq$ on X is the order $\preceq_d$ on X defined by: $\forall x, y \in X, x \preceq_d y$ iff $y \preceq x$.

3. GLS, a generic graph search algorithm

All basic graph search algorithms proceed in a standard manner: iteratively visiting a new vertex that is adjacent to a previously visited vertex whenever possible. Once a vertex is visited, any of its unvisited neighbors becomes eligible for being visited next, though the exact selection criteria varies according to the type of search which is used.

When performing a search of some graph, we consider that each unvisited vertex is associated with some label indicating its "priority" for being visited next. For example, a vertex with no visited neighbors would have lower priority than a vertex with a visited neighbor, and hence have a lower-valued label. As additional neighbors of a vertex are visited, its priority may be increased (or even decreased) for some search paradigms. When the next vertex is chosen to be visited, there is no other unvisited vertex with higher priority.

We proceed to formalize this concept as a generic algorithm. First we precisely specify the "label" of a vertex, $l(v)$, as being a member of some set L. The label of a vertex may change many times during the execution of the search. The "priority" of a vertex for being visited next is expressed mathematically as a partial order on L. If this order is not total, there may be many different labels which may be chosen at any point in the search.

Once a vertex is chosen, the unexplored vertices in its neighborhood may become more desirable to choose next. Algorithmically, we define a function $UPLAB(l, i)$ that will update the label $l = l(v)$ of an unvisited vertex v when one of its neighbors has been chosen as the ith vertex. The $UPLAB(l, i)$ function will be used to update $l(v)$ to a new element in L (which may depend on i) to reflect the new priority of choosing v in our search.

Each vertex must initially have some starting label associated with it. We choose some element of L for this purpose, and call this initial label l_0.

We combine these four concepts into a labeling structure that, once specified, can be used to develop an algorithm for various types of graph search.

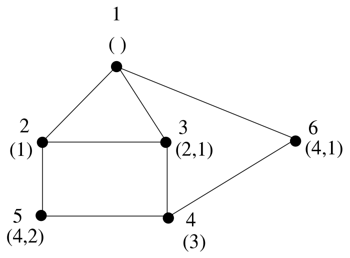

Fig. 1. An example of execution of GLS.

Definition 3.1. A *labeling structure* is a four-tuple $(L, \preceq, l_0, UPLAB)$ where L is a set of labels (which are used to label the vertices), $\preceq$ is a partial order on L which is used to choose a maximal label, l_0 is an element of L (which is the initial label given to all vertices), and *UPLAB* is a function from $L \times \mathbb{N}^+$ to L (which is used to update the labels). $\prec$ denotes the strict order associated with $\preceq$.

This definition of a labeling structure is more general than the definition given in [1], as here there is no constraint on function *UPLAB*, whereas to compute peos an 'inclusion condition' is required, ensuring that if the set of numbered neighbors of a vertex x is included in the set of numbered neighbors of a vertex y, then the label of x is smaller than the label of y.

```
Algorithm GLS (General Label Search)
input  : A graph $G = (V, E)$ and a labeling structure $(L, \preceq, l_0, UPLAB)$.
output : An ordering $\sigma$ of $V$.

Initialize all labels to $l_0$;
for $i \leftarrow 1$ to $n$ do
    Pick an unnumbered vertex $v$ whose label is maximal under $\preceq$ ;
    $\sigma(i) \leftarrow v$ (assigns to $v$ the number $i$);
    foreach unnumbered vertex $w$ adjacent to $v$ do
        $l(w) \leftarrow UPLAB(l(w), i)$;
```

Notation 3.2. *When we shall restrict our attention to a particular labeling structure S, we shall call the algorithm S-GLS. If σ is an ordering of G found by GLS using a labeling structure S, i.e. found by S-GLS, we say that σ is an S-GLS ordering of G.*

Iterating i of the **for** *$i \leftarrow 1$* **to** *n loop in an execution of GLS or S-GLS will be referred to as iteration i of this execution. In the subsequent notations, subscript i means: at the end of iteration i of the considered execution. Thus we will use subscript $i-1$ to refer to the state of variables at the beginning of iteration i, and subscript 0 to refer to the state of variables just before entering the loop for the first time.*

Example 3.3. Let us consider for example labeling structure $S = (L, \preceq, l_0, UPLAB)$, which will be associated with Algorithm LexDFS: L is the set of strings over the alphabet $\mathbb{N}^+$, $\preceq$ is lexicographic order $\leq_L$, l_0 is the empty string ϵ, $UPLAB(l, i)$ is digit i prepended to string l (i.e. i is added at the beginning of string l). An execution of S-GLS on a graph G is shown in Fig. 1. In this figure, two informations are given on each vertex x of G: the number i received by x in this execution, i.e. such that $\sigma(i) = x$, and the label of x at the beginning of iteration i (which will no longer be updated in this execution). For instance, vertex $\sigma(3)$ has label (2, 1) at the beginning of iteration 3, which is maximal at that time since the labels of the other unnumbered vertices $\sigma(4)$, $\sigma(5)$ and $\sigma(6)$ are respectively the empty list, (2) and (1) at that time.

At iteration i of the algorithm, in order to choose a new vertex with maximal label, which can be numbered next, we need to know the labels of each unnumbered vertex v; this label depends solely on the set of neighbors of v which are already numbered, or rather on the set I of *numbers* of the visited neighbors of v. At each iteration i, the label of each neighbor v of the vertex $\sigma(i)$ currently being visited changes from l to $UPLAB(l, i)$. During the execution of the search, the label of each vertex is thus the result of successive applications of the *UPLAB* function from l_0 with increasing values of i.

We will use the following notations throughout this paper:

Notation 3.4. *For any labeling structure $S = (L, \preceq, l_0, UPLAB)$ and any subset I of $\mathbb{N}^+$, let $lab_S(I)$, (or $lab(I)$ if S is clear from the context), be the label of L defined by:*

$lab_S(I)$ (or $lab(I)$) $= UPLAB(\ldots(UPLAB(UPLAB(l_0, i_1), i_2), \ldots), i_k)$, where $I = \{i_1, i_2, \ldots, i_k\}$ with $i_1 < i_2 < \cdots < i_k$.

For any integer $i \geq 0$, the set $L_{S,i}$ (or simply L_i), is defined by:

$L_{S,i} = \{lab_S(I),\ I \subseteq \mathbb{N}_i^+\}$.

For instance, if S is the labeling structure given in Example 3.3, $lab_S(I)$ is the string of the integers in I in decreasing order.

Thus the label of an unnumbered vertex at any point of an execution of S-GLS is equal to $lab_S(I)$ where I is the set of numbers of its numbered neighbors, and $L_{S,i}$ is the set of possible labels at the end of iteration i of an execution of S-GLS.

For each vertex, we need to know the set of numbers assigned to its numbered neighbors.

Notation 3.5. *For any graph G, any ordering σ of V, any integer i from 0 to n and any vertex v of G, $NumSet^{\sigma}_{G,i}(v)$ denotes the set of integers $k \le i$ such that $\sigma(k)$ is adjacent to v in G.*

Thus the set of numbers of the numbered neighbors of a vertex v of a graph G at the end of iteration i of an execution of S-GLS computing ordering σ is $NumSet^{\sigma}_{G,i}(v)$, and the label of v at this point is $lab_S(NumSet^{\sigma}_{G,i}(v))$.

For instance, in Example 3.3, the label of $\sigma(3)$ at the beginning of iteration 3, i.e. at the end of iteration 2, is $lab_S(NumSet^{\sigma}_{G,2}(\sigma(3))) = lab_S(\{1, 2\}) = (2, 1)$.

Now, an ordering σ of a graph G can be computed by S-GLS if and only if for each integer i from 1 to n, the label of $\sigma(i)$ is maximal among the labels of unnumbered vertices in an execution of S-GLS having numbered successively $\sigma(1), \sigma(2), \ldots, \sigma(i-1)$ so far. We immediately obtain the following characteristic property of an S-GLS ordering of a graph, which will be useful in subsequent proofs, as it separates what depends on S (lab_S) from what depends on σ ($NumSet^{\sigma}_{G,i-1}$).

Remark 3.6. For any labeling structure S, any graph G and any ordering σ of V, σ is an S-GLS ordering of G if and only if for any integers i, j such that $1 \le i < j \le n$, $lab_S(NumSet^{\sigma}_{G,i-1}(\sigma(i))) \not\prec lab_S(NumSet^{\sigma}_{G,i-1}(\sigma(j)))$.

4. Seven classical graph search algorithms

In this paper we consider a number of graph search algorithms. Every such algorithm computes an ordering of the vertices of the input graph by successively visiting (and numbering with the next number) an unnumbered vertex until all vertices are visited.

We will present seven classical or basic search algorithms, namely the general graph search (GGS), Breadth-First Search (BFS), Depth-First Search (DFS), Lexicographic Breadth-First Search (LexBFS), Maximum Cardinality Search (MCS), Lexicographic Depth-First Search (LexDFS), and Maximal Neighborhood search (MNS). In order to make notations easier, we will call *BasicAlg* the set of these seven algorithms, and often refer to an arbitrary one of these algorithms as X. For any algorithm X in *BasicAlg*, the X orderings of a graph are the orderings of this graph that can be computed by algorithm X.

To illustrate how Algorithm GLS can describe various graph searches, we exhibit labeling structures that correspond to each of these well-known graph search paradigms. More accurately, we define for each algorithm X in *BasicAlg* a labeling structure $S_0(X)$ such that X and $S_0(X)$-GLS compute the same orderings on every graph. Note that $S_0(X)$ is not the only labeling structure S such that S-GLS computes the same orderings as X. We will study later in this paper some conditions on S for S-GLS to compute only X orderings, or exactly the X orderings of every graph.

Graph Search, called GGS in this paper, as discussed in Section 1, chooses at each step whenever possible a vertex having a visited neighbor.

```
Algorithm GGS (General Graph Search)

input  : A graph $G = (V, E)$.
output : An ordering $\sigma$ of $V$.

Initialize set Reached as the empty set;
for $i \leftarrow 1$ to $n$ do
    if Reached is empty then
        Pick any unnumbered vertex $v$;
    else
        Pick and remove a vertex $v$ from Reached;
    $\sigma(i) \leftarrow v$ (assigns to $v$ the number $i$);
    foreach unnumbered vertex $w$ adjacent to $v$ do
        add $w$ to the set Reached;
```

In this search, the only constraint is that any unnumbered vertex belonging to the set *Reached*, i.e. having at least one numbered neighbor, has priority on any unnumbered vertex having no numbered neighbor. This can be done by giving label 1 to the vertices having at least one numbered neighbor and label 0 to the other vertices; all labels are initialized as 0, and each time a vertex is numbered, its unnumbered neighbors receive label 1.

Thus GGS computes the same orderings as $S_0(GGS)$-GLS, where $S_0(GGS)$ is the labeling structure defined by:

$L = \{0, 1\}$, $\preceq$ is the natural order $\le$ on integers, $l_0 = 0$, $UPLAB(l, i) = 1$.

It follows that $lab(I) = 0$ if $I = \emptyset$ and 1 otherwise. Thus the label of each vertex v is a Boolean indicating whether the vertex is adjacent to an already visited vertex.

Breadth-First Search (BFS) chooses at each step as a priority a vertex having a *least* recently visited neighbor. It uses a queue to store the vertices to be visited next in appropriate order.

Algorithm BFS (Breadth-First Search)

```
input  : A graph $G = (V, E)$.
output : An ordering $\sigma$ of $V$.

Initialize queue $Q$ as the empty queue;
for $i \leftarrow 1$ to $n$ do
    if $Q$ is empty then
        Pick any unnumbered vertex $v$;
    else
        Pick and remove the first vertex $v$ from $Q$;
    $\sigma(i) \leftarrow v$ (assigns to $v$ the number $i$);
    foreach unnumbered vertex $w$ adjacent to $v$ do
        if $w$ is not already in $Q$ then
            add $w$ to the end of $Q$;
```

At each iteration i of BFS, the priority relation among unnumbered vertices is modified as follows. The priority of the unnumbered vertices having $\sigma(i)$ as first numbered neighbor is increased: it becomes higher than the priority of unnumbered vertices having no numbered neighbor, but remains lower than the priorities of unnumbered vertices having a neighbor numbered at some previous iteration. The priority relation among other unnumbered vertices remains unchanged.

Thus BFS computes the same orderings as $S_0(BFS)$-GLS, where $S_0(BFS)$ is the labeling structure defined by:

$L = \mathbb{N}^+ \cup \{\infty\}$, $\preceq$ is the natural order $\geq$ on integers, $l_0 = \infty$, $UPLAB(l, i)$ equals i if $l = \infty$ and equals l otherwise.

It follows that $lab(I) = \infty$ if $I = \emptyset$ and $min(I)$ otherwise (min for usual order $\leq$ on integers). Thus the label is the number assigned to the first (least recently) visited neighbor if it exists.

Depth-First Search (DFS) chooses at each step as a priority a vertex having a *most* recently visited neighbor. It uses a stack to store the vertices to be visited next in appropriate order.

Algorithm DFS (Depth-First Search)

```
input  : A graph $G = (V, E)$.
output : An ordering $\sigma$ of $V$.

Initialize stack $S$ as the empty stack;
for $i \leftarrow 1$ to $n$ do
    while there is a numbered vertex at the top of $S$ do
        Pop the vertex from top of $S$;
    if $S$ is empty then
        Pick any unnumbered vertex $v$;
    else
        Pop the vertex $v$ from top of $S$;
    $\sigma(i) \leftarrow v$ (assigns to $v$ the number $i$);
    foreach unnumbered vertex $w$ adjacent to $v$ do
        push $w$ on top of $S$;
```

At each iteration i of DFS, the priority relation among unnumbered vertices is modified as follows. The priority of every unnumbered neighbor of $\sigma(i)$ is increased: it becomes the highest at that time. The priority relation among other unnumbered vertices remains unchanged.

Thus DFS computes the same orderings as $S_0(DFS)$-GLS, where $S_0(DFS)$ is the labeling structure defined as follows:

$L = \{0\} \cup \mathbb{N}^+$, $\preceq$ is the natural order $\leq$ on integers, $l_0 = 0$, $UPLAB(l, i) = i$.

It follows that $lab(I) = 0$ if $I = \emptyset$ and $max(I)$ otherwise. Thus the label is the number assigned to the last (most recently) visited neighbor if it exists.

Lexicographic Breadth-First Search (LexBFS) is a restricted version of BFS introduced by Rose, Tarjan and Lueker [9] to compute a peo of a chordal graph and thereby recognize chordal graphs. It assigns labels to vertices and uses a lexicographic order to choose the next vertex to visit. Algorithm LexBFS as defined in [9] numbers vertices from n down to 1 in order to compute a peo of a chordal graph, and compares labels under the usual lexicographic order on integer strings. As our search algorithms numbers the vertices from 1 to n, labels are compared under the lexicographic order on integer strings where the order on integers is reversed : i is considered larger than j when $i < j$.

Algorithm LexBFS (Lexicographic Breadth-First Search)

```
input  : A graph $G = (V, E)$.
output : An ordering $\sigma$ of $V$.
Initialize all labels as the empty string;
for $i \leftarrow 1$ to $n$ do
    Pick an unnumbered vertex $v$ whose label is maximal under lexicographic order (where integers are compared in reverse order of the usual one);
    $\sigma(i) \leftarrow v$ (assigns to $v$ the number $i$);
    foreach unnumbered vertex $w$ adjacent to $v$ do
        append $i$ to $l(w)$;
```

Thus LexBFS is exactly algorithm $S_0(LexBFS)$-GLS, where $S_0(LexBFS)$ is the labeling structure defined as follows:

L is the set of strings over the alphabet $\mathbb{N}^+$, $\preceq$ is the lexicographic order on L in which digit i is considered larger than digit j when $i < j$, l_0 is the empty string ϵ, $UPLAB(l, i)$ is digit i appended to string l.

It follows that $lab(I)$ is the string of integers in I in increasing order (for the usual order $\leq$).

Maximum Cardinality Search (MCS) was introduced by Tarjan and Yannakakis [12] as a simplification of LexBFS. MCS iteratively chooses a vertex that is adjacent to the maximum number of previously chosen vertices. As is the case for LexBFS, MCS as defined in [12] numbers vertices from n down to 1 to compute a peo of a chordal graph. Here again we number form 1 to n.

Algorithm MCS (Maximal Cardinality Search)

```
input  : A graph $G = (V, E)$.
output : An ordering $\sigma$ of $V$.
Initialize all labels as 0;
for $i \leftarrow 1$ to $n$ do
    Pick an unnumbered vertex $v$ whose label is maximal under $\leq$;
    $\sigma(i) \leftarrow v$ (assigns to $v$ the number $i$);
    foreach unnumbered vertex $w$ adjacent to $v$ do
        $l(w) \leftarrow l(w) + 1$;
```

Thus MCS is exactly algorithm $S_0(MCS)$-GLS, where $S_0(MCS)$ is the labeling structure defined as follows:

$L = \{0\} \cup \mathbb{N}^+$, $\preceq$ is the natural order $\leq$ on integers, $l_0 = 0$, $UPLAB(l, i) = l + 1$.

It follows that $lab(I) = |I|$. The label is the number of visited neighbors.

Lexicographic Depth-First Search (LexDFS) is a depth-first analog of LexBFS introduced by Corneil and Krueger [5].

Algorithm LexDFS (Lexicographic Depth-First Search)

```
input  : A graph $G = (V, E)$.
output : An ordering $\sigma$ of $V$.
Initialize all labels as the empty string;
for $i \leftarrow 1$ to $n$ do
    Pick an unnumbered vertex $v$ whose label is maximal under the usual lexicographic order;
    $\sigma(i) \leftarrow v$ (assigns to $v$ the number $i$);
    foreach unnumbered vertex $w$ adjacent to $v$ do
        prepend $i$ to $l(w)$;
```

Thus LexDFS is exactly algorithm $S_0(LexDFS)$-GLS, where $S_0(LexDFS)$ is the labeling structure defined as follows:

L is the set of strings over the alphabet $\mathbb{N}^+$, $\preceq$ is the usual lexicographic order $\leq_L$, l_0 is the empty string ϵ, $UPLAB(l, i)$ is digit i prepended to string l.

Therefore $lab(I)$ is the string of integers of I in decreasing order.

Maximal Neighborhood Search (MNS) is a generalization of both LexBFS and MCS, and was also introduced by Corneil and Krueger [5].

```
Algorithm MNS (Maximal Neighborhood Search)
input  : A graph $G = (V, E)$.
output : An ordering $\sigma$ of $V$.
Initialize all labels as the empty set;
for $i \leftarrow 1$ to $n$ do
    Pick an unnumbered vertex $v$ whose label is maximal under inclusion;
    $\sigma(i) \leftarrow v$ (assigns number $i$ to $v$);
    foreach unnumbered vertex $w$ adjacent to $v$ do
        $l(w) \leftarrow l(w) \cup \{i\}$;
```

Thus MNS is exactly algorithm $S_0(MNS)$-GLS, where $S_0(MNS)$ is the labeling structure defined as follows:

L is the power set of $\mathbb{N}^+$, $\preceq$ is set inclusion $\subseteq$, l_0 is the empty set, $UPLAB(l, i) = l \cup \{i\}$.

It follows that $lab(I) = I$. The label is the set of numbers of visited neighbors.

Let us also mention algorithms MEC and MCC, as defined by Shier [10]. MCC is a more general variant of MCS and MEC is a more general variant of MNS: the condition of maximality to choose a vertex is required in the connected component of the remaining graph containing this vertex instead of the whole remaining graph. Contrary to the seven basic algorithms, there is no labeling structure S such that S-GLS computes the same orderings as MCC or MEC on every graph: as GLS chooses the next vertex to visit only from the sets of numbers of the visited neighbors of the remaining vertices, it cannot take the structure of the residual graph, here its connected components, into account.

5. Characterizing structures for the 7 classical graph searches

Let us now consider a labeling structure S along with one of our classical search algorithms, which we will call X. We ask the following questions:

Q_1: Does S-GLS compute only X orderings?
Q_2: Is the set of S-GLS orderings *equal* to the set of X orderings?

As the X orderings of G are exactly the $S_0(X)$-GLS orderings of G as given in the previous section, questions Q_1 and Q_2 are special cases of the more general question Q_3:

Q_3: Given two labeling structures S and S', is every S-GLS ordering an S'-GLS ordering for every graph G?

We prove a necessary and sufficient condition for Q_3 to be answered as positive with the help of the following lemma.

Lemma 5.1. *For any labeling structure S, any integer $p \geq 1$ and any subsets I and J of $\mathbb{N}_p^+$, if $lab(I) \nprec lab(J)$ then there exist a graph G and an S-GLS ordering σ of G such that $NumSet_{G,p}^{\sigma}(\sigma(p+1)) = I$ and $NumSet_{G,p}^{\sigma}(\sigma(p+2)) = J$.*

Proof. Let $G = (V, E)$, with $V = \{z_1, z_2, \dots, z_{p+2}\}$ and $E = \{z_i z_k \mid 1 \leq k < i \leq p$ and if $lab(I \cap \mathbb{N}_{i-1}^+) \prec lab(J \cap \mathbb{N}_{i-1}^+)$ then $k \in J$ else $k \in I\} \cup \{z_{p+1} z_k \mid k \in I\} \cup \{z_{p+2} z_k \mid k \in J\}$. Let $\sigma = (z_1, z_2, \dots, z_{p+2})$. By definition of E and σ, for any integers i, j such that $1 \leq i \leq j \leq p+2$, $NumSet_{G,i-1}^{\sigma}(\sigma(j))$ is either equal to $I \cap \mathbb{N}_{i-1}^+$ or to $J \cap \mathbb{N}_{i-1}^+$, with $lab(NumSet_{G,i-1}^{\sigma}(\sigma(i))) \nprec lab(NumSet_{G,i-1}^{\sigma}(\sigma(j)))$. By Remark 3.6 σ is an S-GLS ordering, and we have $NumSet_{G,i-1}^{\sigma}(\sigma(p+1)) = I$ and $NumSet_{G,i-1}^{\sigma}(\sigma(jp+2)) = J$. □

Theorem 5.2. *For any labeling structures S and S' with partial orders $\preceq_S$ and $\preceq_{S'}$ resp., every S-GLS ordering is an S'-GLS ordering for every graph G if and only if the following property $Sub(S, S')$ holds,*

$$Sub(S, S') : \textit{for any subsetsI and J of } \mathbb{N}^+, \textit{ if } lab_{S'}(I) \prec_{S'} lab_{S'}(J) \textit{ then } lab_S(I) \prec_S lab_S(J).$$

Proof. $Sub(S, S')$ is a sufficient condition as a direct consequence of Remark 3.6 by the following argument. Let G be a graph and σ be an S-GLS ordering of G. By Remark 3.6, for any integers i, j such that $1 \leq i < j \leq n$, $lab_S(NumSet_{G,i-1}^{\sigma}(\sigma(i))) \nprec_S lab_S(NumSet_{G,i-1}^{\sigma}(\sigma(j)))$, so by $Sub(S, S') lab_{S'}(NumSet_{G,i-1}^{\sigma}(\sigma(i))) \nprec_{S'} lab_{S'}(NumSet_{G,i-1}^{\sigma}(\sigma(j)))$. By Remark 3.6 again σ is an S'-GLS ordering of G.

Let us now assume that $Sub(S, S')$ does not hold. Let us show that there are a graph G and an S-GLS ordering that is not an S'-GLS ordering. Let I and J be two subsets of $\mathbb{N}^+$ such that $lab_{S'}(I) \prec_{S'} lab_{S'}(J)$, but $lab_S(I) \nprec_S lab_S(J)$, and let $p = max(I \cup J)$. By Lemma 5.1, there are a graph G and an S-GLS ordering σ such that $NumSet_{G,p}^{\sigma}(\sigma(p+1)) = I$ and $NumSet_{G,p}^{\sigma}(\sigma(p+2)) = J$. It follows that $lab_{S'}(NumSet_{G,p}^{\sigma}(\sigma(p+1))) = lab_{S'}(I) \prec_{S'} lab_{S'}(J) = lab_{S'}(NumSet_{G,p}^{\sigma}(\sigma(p+2)))$, and by Remark 3.6 σ is not an S'-GLS ordering. □

Let us now go back to answering questions Q_1 and Q_2:

Theorem 5.3. *For any labeling structure S and any X in BasicAlg,*

(A1) *S-GLS computes only X orderings of G for every graph G if and only if $Sub(S, S_0(X))$ holds,*
(A2) *the set of S-GLS orderings of G is equal to the set of X orderings of G for every graph G if and only if both $Sub(S, S_0(X))$ and $Sub(S_0(X), S)$ hold.*

It turns out that for X in $\{LexBFS, LexDFS\}$, questions Q_1 and Q_2 are equivalent. This will come as a corollary of the following proposition.

Proposition 5.4. *Let S' be a labeling structure with partial order $\preceq_{S'}$. Every labeling structure S satisfying $Sub(S, S')$ also satisfies $Sub(S', S)$ if and only if $\preceq_{S'}$ is a total order on the set $\{lab_{S'}(I),\ I \subseteq \mathbb{N}^+\}$ and for any distinct subsets I and J of $\mathbb{N}^+$, $lab_{S'}(I) \neq lab_{S'}(J)$.*

Proof. We suppose that $\preceq_{S'}$ is a total order on the set $\{lab_{S'}(I),\ I \subseteq \mathbb{N}^+\}$ and for any distinct subsets I and J of $\mathbb{N}^+$, $lab_{S'}(I) \neq lab_{S'}(J)$. Let S be a labeling structure satisfying $Sub(S, S')$. Let us show that $Sub(S', S)$ holds. Let I and J be subsets of $\mathbb{N}^+$ such that $lab_S(I) \prec_S lab_S(J)$. It follows that I and J are distinct, so $lab_{S'}(I) \neq lab_{S'}(J)$, and that $lab_S(J) \not\preceq_S lab_S(I)$, so $lab_{S'}(J) \not\preceq_{S'} lab_{S'}(I)$ since $Sub(S, S')$. As $\preceq_{S'}$ is a total order on $\{lab_{S'}(I),\ I \subseteq \mathbb{N}^+\}$, $lab_{S'}(I) \preceq_{S'} lab_{S'}(J)$ with $lab_{S'}(I) \neq lab_{S'}(J)$, so $lab_{S'}(I) \prec_{S'} lab_{S'}(J)$.

Conversely, suppose that $\preceq_{S'}$ is not a total order on the set $\{lab_{S'}(I),\ I \subseteq \mathbb{N}^+\}$ or that there are distinct subsets I and J of $\mathbb{N}^+$ such that $lab_{S'}(I) = lab_{S'}(J)$. Let us show that there is a labeling structure S satisfying $Sub(S, S')$, but not $Sub(S', S)$. In the first case it is sufficient to take S obtained from S' by replacing $\preceq_{S'}$ by one of its linear extensions. In the second case let I_1 and I_2 be distinct subsets of $\mathbb{N}^+$ such that $lab_{S'}(I_1) = lab_{S'}(I_2)$ with the smallest possible $|I_1|$. Let $l_1 = lab_{S'}(I_1) = lab_{S'}(I_2)$, $i_2 = max(I_2)$ and $l'_2 = lab_{S'}(I_2 - \{i_2\})$. We define S from S' such that $lab_S(I_1) \prec_S lab_S(I_2)$. S is obtained from S' by adding a new label l_2 such that $l_1 \prec_S l_2$, redefining $UPLAB(l'_2, i_2)$ as l_2 (instead of l_1) and making l_2 behave as l_1 with respect to $UPLAB$ ($UPLAB(l_1, i) = UPLAB(l_2, i)$) and to $\preceq_S$ when compared to the other labels. Thus for any subset I of $\mathbb{N}^+$, if $max(I) = i_2$ and $lab_{S'}(I - \{i_2\}) = l'_2$ then $lab_{S'}(I) = l_1$ and $lab_S(I) = l_2$, otherwise $lab_{S'}(I) = lab_S(I)$. It follows that $Sub(S, S')$ holds. It is impossible that $max(I_1) = i_2$ and $lab_{S'}(I_1 - \{i_2\}) = l'_2$ (otherwise $lab_{S'}(I_1 - \{i_2\}) = lab_{S'}(I_2 - \{i_2\})$ which would contradict the choice of I_1 and I_2 with the smallest possible $|I_1|$). Therefore $lab_S(I_1) = l_1 \prec_S l_2 = lab_S(I_2)$, so $Sub(S', S)$ does not hold. □

It follows that among our seven classical graph search algorithms, LexBFS and LexDFS are the only ones for which questions Q_1 and Q_2 are equivalent: $\preceq_{S(MNS)}$ is not a total order on the set $\{lab_S(I),\ I \subseteq \mathbb{N}^+\}$ and for any X in $\{GGS, BFS, DFS, MCS\}$, there are distinct subsets I and J of $\mathbb{N}^+$ such that $lab_{S_0(X)}(I) = lab_{S_0(X)}(J)$.

Corollary 5.5. *For any labeling structure S, if S-GLS only computes LexBFS (resp. LexDFS) orderings of G for every graph G then the set of S-GLS orderings of G is equal to the set of LexBFS (resp. LexDFS) orderings for every graph G.*

We will use characteristic property $Sub(S, S_0(X))$ to define some sufficient (and necessary in some cases) conditions on S for S-GLS to compute only X orderings for every graph G, i.e. for a positive answer to question Q_1 (and to question Q_2 for *LexBFS* and *LexDFS* by Corollary 5.5).

We first define some properties on labeling structures. We recall that L_{i-1} is the set $\{lab(I),\ I \subseteq \mathbb{N}^+_{i-1}\}$, i.e. the set of possible labels of vertices at the beginning of iteration i of an execution of GLS.

Definition 5.6. For a labeling structure $S = (L, \preceq, l_0, UPLAB)$, we define the following properties:

(L_1) minimum initial label: $l_0 \prec UPLAB(l, i)$ for all $i \geq 1$ and $l \in L_{i-1}$.
(L_2) monotonic non-decreasing: $l \preceq UPLAB(l, i)$, for all $i \geq 1$ and $l \in L_{i-1}$.
(L_3) monotonic strictly increasing: $l \prec UPLAB(l, i)$, for all $i \geq 1$ and $l \in L_{i-1}$.
(L_4) stability: if $l \prec l'$ then $UPLAB(l, i) \prec UPLAB(l', i)$, for all $i \geq 1$ and $l, l' \in L_{i-1}$.
(L_5) breadth-firstness: if $l \prec l'$ then $UPLAB(l, i) \prec l'$ for all $i \geq 1$ and $l, l' \in L_{i-1}$.
(L_6) depth-firstness: $l \prec UPLAB(l', i)$ for all $i \geq 1$ and $l, l' \in L_{i-1}$.
(L_7) independence: $UPLAB(l, i) = UPLAB(l, i')$ for all i, i' with $1 \leq i < i'$ and all $l \in L_{i-1}$.

Definition 5.7. For any X in *BasicAlg* and any labeling structure S, $Prop_X(S)$ is the property on S defined as the conjunction of the properties from L_1 to L_7 marked with a check mark on line X of Table 1; asterisks on line X indicate properties implied by this conjunction; That is to say:

1. $Prop_{GGS}(S) = L_1$,
2. $Prop_{BFS}(S) = L_1 \wedge L_2 \wedge L_5$, which implies L_4,
3. $Prop_{DFS}(S) = L_6$, which implies L_1, L_2 and L_3,
4. $Prop_{MNS}(S) = L_3 \wedge L_4$, which implies L_1 and L_2,
5. $Prop_{LexBFS}(S) = L_3 \wedge L_5$, which implies L_1, L_2 and L_4,
6. $Prop_{LexDFS}(S) = L_4 \wedge L_6$, which implies L_1, L_2 and L_3,
7. $Prop_{MCS}(S) = L_3 \wedge L_7$, which implies L_1, L_2 and L_4.

The following lemma is easy to verify.

Lemma 5.8. *For any X in BasicAlg, property $Prop_X(S_0(X))$ holds.*

We now come to our sufficient (and necessary) conditions for a positive answer to question Q_1. As we see in the following theorem, we have a sufficient condition for an algorithm to compute only X-orderings, but this condition is not always necessary.

Table 1
Labeling structure properties for graph searches.

X	$Prop_X(S)$						
	L_1	L_2	L_3	L_4	L_5	L_6	L_7
GGS	✓						
BFS	✓	✓		*	✓		
DFS	*	*	*			✓	
MNS	*	*	✓	✓			
LexBFS	*	*	✓	*	✓		
LexDFS	*	*	*	✓		✓	
MCS	*	*	✓	*			✓

Theorem 5.9. *For any X in BasicAlg, $Prop_X(S)$ is a sufficient condition on a labeling structure S for S-GLS to compute only X orderings of G for every graph G.*
It is also a necessary condition if X is in $\{GGS, DFS, LexBFS, LexDFS\}$, but not otherwise, i.e. if X is in $\{BFS, MNS, MCS\}$.

Proof. We will begin this proof with a general discussion.
Let $X \in BasicAlg$ and $S = (L, \preceq, l_0, UPLAB)$ be a labeling structure. In the following, $lab(I)$ stands for $lab_S(I)$. By Theorem 5.3(A1), $Prop_X(S)$ is a sufficient condition iff

$Prop_X(S) \Rightarrow Sub(S, S_0(X))$, and $Prop_X(S)$ is a necessary condition if and only if
$Sub(S, S_0(X)) \Rightarrow Prop_X(S)$.

Let us show that $Prop_X(S)$ is a sufficient condition. We assume that $Prop_X(S)$ holds. Let I and J be subsets of $\mathbb{N}^+$ such that $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$. Let us show that $lab(I) \prec lab(J)$. For any integer $i \geq 0$, let $l_i = lab(I \cap \mathbb{N}_i^+)$ and $l'_i = lab(J \cap \mathbb{N}_i^+)$. Note that for $i = 0$, l_i corresponds to the label l_0 defined in S and that for any $i \geq 0$, l_i and l'_i belong to L_i. It is sufficient to show that $l_i \prec l'_i$ for some $i \geq max(I \cup J)$. In the following, we show that $lab(I) \prec lab(J)$ for each $X \in BasicAlg$.
In each case, we will also prove in the following discussion whether or not $Prop_X(S)$ is or is not a necessary condition.

$X = GGS$
$Prop_X(S) = L_1$, '$\prec_{S_0(X)}$' = '$<$' and for any subset I of $\mathbb{N}^+$, $lab_{S_0(X)}(I) = 0$ if $I = \emptyset$ and 1 otherwise.
As $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$, $I = \emptyset$ and $J \neq \emptyset$. Let $i = max(J)$. By $L_1 l_i = l_0 \prec UPLAB(l'_{i-1}, i) = l'_i$, with $i = max(I \cup J)$. Thus $Prop_X(S)$ is a sufficient condition.
Let us show that $Prop_X(S)$ is also a necessary condition. We assume that $Sub(S, S_0(X))$ holds. For any $i \geq 1$ and $l \in L_{i-1}$, $l_0 = lab(I)$ with $I = \emptyset$ and $UPLAB(l, i) = lab(J)$ with $J \neq \emptyset$, so $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$ and by $Sub(S, S_0(X)) l_0 = lab(I) \prec lab(J) = UPLAB(l, i)$.

$X = BFS$
$Prop_X(S) = L_1 \wedge L_2 \wedge L_5$, '$\prec_{S_0(X)}$' = '$>$' and for any subset I of $\mathbb{N}^+$, $lab_{S_0(X)}(I) = \infty$ if $I = \emptyset$ and $min(I)$ otherwise (min is for the usual order $\leq$ on integers).
As $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$, $J \neq \emptyset$ and either $I = \emptyset$ or $min(I) > min(J)$. It is sufficient to show that $l_i \prec l'_i$ for every $i \geq min(J)$. Let us prove it by induction on i. For $i = min(J)$, by L_1, $l_i = l_0 \prec UPLAB(l_0, i) = l'_i$. We assume that $l_{i-1} \prec l'_{i-1}$. As l_i is either equal to l_{i-1} or to $UPLAB(l_{i-1}, i)$, by induction hypothesis and L_5, we have $l_i \prec l'_{i-1}$. As by $L_2 l'_{i-1} \preceq l'_i$, we obtain $l_i \prec l'_{i-1} \preceq l'_i$, which completes the proof by induction. Thus $Prop_X(S)$ is a sufficient condition.
Let us show that $Prop_X(S)$ is not a necessary condition. Let $L = \{1, 1.5, 2, 2.5, 3, \ldots\} \cup \{\infty\}$, '$\preceq$' = '$\geq$', $l_0 = \infty$, and let $UPLAB(l, i)$ equal i if $l = \infty$, $l + 0.5$ if l is an integer and l otherwise. For any subset I of $\mathbb{N}^+$, $lab(I) = \infty$ if $I = \emptyset$, $min(I)$ if $|I| = 1$ and $min(I) + 0.5$ otherwise. Thus $Sub(S, S_0(X))$ holds, but not $Prop_X(S)$ since L_2 is violated: $UPLAB(l, i) = l + 0.5 \prec l$ if l is an integer.

$X = DFS$
$Prop_X(S) = L_6$, '$\prec_{S_0(X)}$' = '$<$' and for any subset I of $\mathbb{N}^+$, $lab_{S_0(X)}(I) = 0$ if $I = \emptyset$ and $max(I)$ otherwise.
As $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$, $J \neq \emptyset$ and either $I = \emptyset$ or $max(I) < max(J)$. Let $i = max(J)$, by $L_6 l_i = l_{i-1} \prec UPLAB(l'_{i-1}, i) = l'_i$ with $i = max(I \cup J)$. Thus $Prop_X(S)$ is a sufficient condition.
Let us show that $Prop_X(S)$ is also a necessary condition. We assume that $Sub(S, S_0(X))$ holds. Let $i \geq 1$ and $l, l' \in L_{i-1}$. Let us show that $l \prec UPLAB(l', i)$. Let I and J be subsets of $\mathbb{N}_{i-1}^+$ such that $l = lab(I)$ and $l' = lab(J)$. Hence $UPLAB(l', i) = lab(J')$ with $J' = J \cup \{i\}$. $J' \neq \emptyset$ and either $I = \emptyset$ or $max(I) \leq i - 1 < i = max(J')$, so $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J')$ and by $Sub(S, S_0(X)) l = lab(I) \prec lab(J') = UPLAB(l', i)$.

$X = MNS$
$Prop_X(S) = L_3 \wedge L_4$, '$\prec_{S_0(X)}$' is strict set inclusion '$\subset$' and for any subset I of $\mathbb{N}^+$, $lab_{S_0(X)}(I) = I$. As $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$, $I \subset J$. It is sufficient to show that $l_i \prec l'_i$ for every $i \geq min(J - I)$. Let us prove it by induction on i. For $i = min(J - I)$, $I \cap \mathbb{N}_{i-1}^+ = J \cap \mathbb{N}_{i-1}^+$, so by $L_3 l_i = l_{i-1} = l'_{i-1} \prec UPLAB(l'_{i-1}, i) = l'_i$. We assume that $l_{i-1} \prec l'_{i-1}$. By $L_3 l'_{i-1} \preceq l'_i$. If $i \notin I$ then $l_i = l_{i-1} \prec l'_{i-1} \preceq l'_i$, otherwise $i \in I \cap J$ and by L_4 and induction hypothesis $l_i = UPLAB(l_{i-1}, i) \prec UPLAB(l'_{i-1}, i) = l'_i$, which completes the proof by induction. Thus $Prop_X(S)$ is a sufficient condition.

Let us show that $Prop_X(S)$ is not a necessary condition. Let S be the labeling structure obtained from $S_0(X)$ by replacing $\preceq_{S_0(X)}$ by the partial order $\preceq$ on L defined by: for any $l, l' \in L$, $l \preceq l'$ iff ($l \subseteq l'$ or ($l = \{2\}$ and $3 \in l'$)) (the verification that $\preceq$ is a partial order on L is left to the reader). $Sub(S, S_0(X))$ holds since $\preceq_{S_0(X)}$ is a sub-order of $\preceq$, but $Prop_X(S)$ does not hold since L_4 is violated: $\{2\} \prec \{3\}$ but $UPLAB(\{2\}, 4) = \{2, 4\} \not\prec \{3, 4\} = UPLAB(\{3\}, 4)$.

$X = LexBFS$

$Prop_X(S) = L_3 \wedge L_5$, $\prec_{S_0(X)}$ is the strict lexicographic order with order $\geq$ on integers, and for any subset I of $\mathbb{N}^+$, $lab_{S_0(X)}(I)$ is the string of the integers in I in increasing order (for usual order $\leq$).

As $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$, $J - I \neq \emptyset$ and for $i = min(J - I)$, $I \cap \mathbb{N}^+_{i-1} = J \cap \mathbb{N}^+_{i-1}$. It is sufficient to show that $l_i \prec l'_i$ for every $i \geq min(J - I)$. Let us prove it by induction on i. For $i = min(J - I)$, in the same way as for $X = MNS$, by $L_3 l_i = l_{i-1} = l'_{i-1} \prec UPLAB(l'_{i-1}, i) = l'_i$. We assume that $l_{i-1} \prec l'_{i-1}$. In the same way as for $X = BFS$, by induction hypothesis and L_5, $l_i \prec l'_{i-1}$, and as by $L_2 l'_{i-1} \preceq l'_i$, we obtain $l_i \prec l'_{i-1} \preceq l'_i$, which completes the proof by induction.

Thus $Prop_X(S)$ is a sufficient condition.

Let us show that $Prop_X(S)$ is also a necessary condition. We assume that $Sub(S, S_0(X))$ holds. By Proposition 5.4 with $S' = S(LexBFS)$, $Sub(S_0(X), S)$ also holds. Thus for any subsets I and J of $\mathbb{N}^+$, $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$ iff $lab(I) \prec lab(J)$. Properties L_3 and L_5 can be rewritten into:

L_3: $lab(I) \prec lab(I \cup \{i\})$, for all $i \geq 1$ and $I \subseteq \mathbb{N}^+_{i-1}$,

L_5: if $lab(I) \prec lab(J)'$ then $lab(I \cup \{i\}) \prec lab(J)$ for all $i \geq 1$ and $I, J \subseteq \mathbb{N}^+_{i-1}$.

Hence as by Lemma 5.8 $Prop_X(S_0(X))$ holds, $Prop_X(S)$ also holds.

$X = LexDFS$

$Prop_X(S) = L_4 \wedge L_6$, $\prec_{S_0(X)}$ is the strict lexicographical order with order $\leq$ on integers, and for any subset I of $\mathbb{N}^+$, $lab_{S_0(X)}(I)$ is the string of the integers in I in decreasing order.

As $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$, $J - I \neq \emptyset$ and for any $i > max(J - I)$, $i \in I$ iff $i \in J$. It is sufficient to show that $l_i \prec l'_i$ for every $i \geq max(J - I)$. Let us prove it by induction on i. For $i = max(J - I)$, in the same way as for $X = DFS$, by $L_6 l_i = l_{i-1} \prec UPLAB(l'_{i-1}, i) = l'_i$. We assume that $l_{i-1} \prec l'_{i-1}$. If $i \notin I$ then $i \notin J$ and $l_i = l_{i-1} \prec l'_{i-1} = l'_i$, otherwise $i \in I \cap J$ and by L_4 and induction hypothesis $l_i = UPLAB(l_{i-1}, i) \prec UPLAB(l'_{i-1}, i) = l'_i$, which completes the proof by induction. Thus $Prop_X(S)$ is a sufficient condition.

We show that $Prop_X(S)$ is also a necessary condition in the same way as for $X = LexBFS$.

$X = MCS$

$Prop_X(S) = L_3 \wedge L_7$, '$\prec_{S_0(X)}$' = '$<$' and for any subset I of $\mathbb{N}^+$, $lab_{S_0(X)}(I) = |I|$.

As $lab_{S_0(X)}(I) \prec_{S_0(X)} lab_{S_0(X)}(J)$, $|I| < |J|$. For any $i \geq 1$ and any $l \in L_{i-1}$, let $f(l) = UPLAB(l, i)$ (which is independent from i by L_7) and for any $p \geq 0$, let $f^p(l)$ be the label obtained from l by p applications of f. $lab(I) = f^{|I|}(l_0)$ and $lab(J) = f^{|J|}(l_0)$. As $|I| < |J|$, by $L_3 lab(I) = f^{|I|}(l_0) \prec f^{|J|}(l_0) = lab(J)$. Thus $Prop_X(S)$ is a sufficient condition.

Let us show that $Prop_X(S)$ is not a necessary condition. Let $L = \{0, 1, 1.5, 2, 2.5, 3, \ldots\}$, '$\preceq$' = '$\leq$', $l_0 = 0$, and let $UPLAB(l, i)$ equal $l + 1.5$ if $i = 1$ and $l + 1$ otherwise. For any subset I of $\mathbb{N}^+$, $lab(I) = |I| + 0.5$ if $1 \in I$ and $|I|$ otherwise. Thus $Sub(S, S_0(X))$ holds, but not $Prop_X(S)$ since L_7 is violated: $UPLAB(0, 1) = 1.5$ and $UPLAB(0, 2) = 1$. □

For $X = BFS$, we can build from the previous proof a property P on a labeling structure S that is weaker than $Prop_{BFS}(S)$ and is also a sufficient condition for S-GLS to compute only BFS orderings: $P = (L1') \wedge (L2') \wedge L_4 \wedge L_5$, where $(L1')$ and $(L2')$ are the weaker variants of L_1 and L_2 respectively defined by

$(L1')$ $l_0 \prec UPLAB(l_0, i)$ for all $i \geq 1$,
$(L2')$ if $l \prec l'$ then $l \prec UPLAB(l', i)$ for all $i \geq 1$ and $l, l' \in L_{i-1}$.

Unfortunately, P is also not a necessary condition since the counterexample given in the previous proof for $X = BFS$ violates $(L2')$: if l is an integer then $l + 0.5 \prec l$ but $l + 0.5 = UPLAB(l, i)$.

We obtain the following characterization of X orderings for every type X of search in $BasicAlg$.

Characterization 5.10. *Let G be a graph, σ be an ordering of V and $X \in BasicAlg$. Then σ is an X ordering of G if and only if there is a labeling structure S such that $Prop_X(S)$ holds and σ is an S-GLS ordering of G.*

Proof. If σ is an X ordering of G then it is a $S_0(X)$-GLS ordering of G, with $Prop_X(S_0(X))$ by Lemma 5.8. The converse follows from Theorem 5.9. □

To illustrate Characterization 5.10, Fig. 2 (inspired from [5]) shows the hierarchy between the sets of orderings yielded by the different graph searches we studied: an arrow from X to X' means that every X ordering of a graph is an X' ordering of this graph.

This hierarchy is deduced in [5] from some other characterizations of the algorithms of $BasicAlg$, except for MCS which is not studied in [5]. It is also clear from Characterization 5.10. For any X, X' in $BasicAlg$, if $Prop_X(S)$ implies $Prop_{X'}(S)$, i.e. if every property marked in Table 1 (with a check mark) on line X' is also marked (with a check mark or an asterisk) on line X, then every X ordering of a graph is also an X' ordering of this graph.

Berry, Krueger and Simonet [1] defined a labeling structure with the restriction that it has to satisfy two conditions (ls1) and (ls2), which are exactly L_3 and L_4. Thus a labeling structure as defined in [1] corresponds in this paper to a labeling

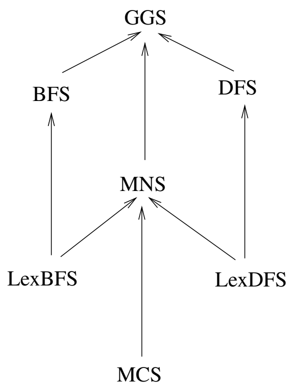

Fig. 2. Hierarchy of types of search.

structure S such that $Prop_{MNS}(S)$. It is shown in [1] that every S-GLS ordering defined with this restriction is an MNS ordering of G, which immediately follows from Theorem 5.9. A labeling structure S is defined in [1] with the restriction that it satisfies condition IC, which is exactly $Sub(S, S(MNS))$. Thus the fact that every S-GLS ordering defined with this restriction is an MNS ordering of G immediately follows from Theorem 5.3(A1), and by Theorem 5.9 condition IC is strictly more general than the conjunction of (ls1) and (ls2). It is proved in [1] that IC, i.e. $Sub(S, S(MNS))$, is a necessary and sufficient condition on a labeling structure S for S-GLS to compute only peos of every chordal graph.

6. Searching the complement graph

Not all search-like algorithms can be expressed using a labeling structure. GLS does not "look ahead" into the unnumbered portion of the graph. Thus, as mentioned at the end of Section 4, the MEC and MCC algorithms of Shier [10], which make decisions based on the connected components of the subgraph induced by the unnumbered vertices, require information that GLS itself cannot obtain. GLS does, however, express some interesting non-graph search algorithms. We will define labeling structures that allow GLS to compute search orderings of the complement $\overline{G}$ of a graph G without computing $\overline{G}$.

Theorem 6.1. *For any labeling structures S and S' with partial orders $\preceq_S$ and $\preceq_{S'}$ resp., the set of S-GLS orderings of G is equal to the set of S'-GLS orderings of $\overline{G}$ for every graph G if and only if the following property $Comp(S, S')$ holds:*

$Comp(S, S')$: for any integer $n \geq 1$ and any subsets I and J of $\mathbb{N}_n^+$, $labs_S(I) \prec_S labs_S(J)$ if and only if $labs_{S'}(\mathbb{N}_n^+ - I) \prec_{S'} labs_{S'}(\mathbb{N}_n^+ - J)$.

Proof. We assume that $Comp(S, S')$ holds. Let G be a graph and σ be an ordering of V. By Remark 3.6, σ is an S'-GLS ordering of $\overline{G}$ if and only if for any integers i, j such that $1 \leq i < j \leq n$, $labs_{S'}(NumSet^{\sigma}_{\overline{G},i-1}(\sigma(i))) \not\prec_{S'} labs_{S'}(NumSet^{\sigma}_{\overline{G},i-1}(\sigma(j)))$, i.e. $labs_{S'}(\mathbb{N}^+_{i-1} - NumSet^{\sigma}_{G,i-1}(\sigma(i))) \not\prec_{S'} labs_{S'}(\mathbb{N}^+_{i-1} - NumSet^{\sigma}_{G,i-1}(\sigma(j)))$, or by $Comp(S, S') labs_S(NumSet^{\sigma}_{G,i-1}(\sigma(i))) \not\prec_S labs_S (NumSet^{\sigma}_{G,i-1}(\sigma(j)))$. We conclude by Remark 3.6 again that σ is an S'-GLS ordering of $\overline{G}$ iff it is an S-GLS ordering of G.

We assume now that $Comp(S, S')$ does not hold. Let us show that there are a graph G and an S-GLS ordering σ of G that is not an S'-GLS ordering of $\overline{G}$ or vice-versa. Let p be an integer ≥ 1 and I, J be subsets of $\mathbb{N}_p^+$ such that exactly one of '$labs_S(I) \prec_S labs_S(J)$' and '$labs_{S'}(\mathbb{N}_p^+ - I) \prec_{S'} labs_{S'}(\mathbb{N}_p^+ - J)$' holds. We suppose that $labs_S(I) \not\prec_S labs_S(J)$ and $labs_{S'}(\mathbb{N}_p^+ - I) \prec_{S'} labs_{S'}(\mathbb{N}_p^+ - J)$ (the proof in the other case is similar). By Lemma 5.1, there are a graph G and an S-GLS ordering σ such that $NumSet^{\sigma}_{G,p}(\sigma(p+1)) = I$ and $NumSet^{\sigma}_{G,p}(\sigma(p+2)) = J$. It follows that $labs_{S'}(NumSet^{\sigma}_{\overline{G},p}(\sigma(p+1))) = labs_{S'}(\mathbb{N}_p^+ - I) \prec_{S'} labs_{S'}(\mathbb{N}_p^+ - J) = labs_{S'}(NumSet^{\sigma}_{\overline{G},p}(\sigma(p+2)))$, and by Remark 3.6 σ is not an S'-GLS ordering of $\overline{G}$. □

Definition 6.2. For any labeling structure S with partial order $\preceq$, $Rev(S)$ is the labeling structure obtained from S by replacing $\preceq$ by its dual order.

Lemma 6.3. *For any X in $\{MNS, LexBFS, LexDFS, MCS\}$, property $Comp(Rev(S_0(X)), S_0(X))$ holds.*

Proof. Let $X \in \{MNS, LexBFS, LexDFS, MCS\}$, n be an integer ≥ 1, I and J be subsets of $\mathbb{N}_n^+$, $\bar{I} = \mathbb{N}_n^+ - I$ and $\bar{J} = \mathbb{N}_n^+ - J$. Let us show that $lab_{S_0(X)}(J) \prec_{S_0(X)} lab_{S_0(X)}(I)$ iff $lab_{S_0(X)}(\bar{I}) \prec_{S_0(X)} lab_{S_0(X)}(\bar{J})$.

It is evident for $X = MNS$ since $J \subset I$ iff $\bar{I} \subset \bar{J}$, and for $X = MCS$ since $|J| < |I|$ iff $|\bar{I}| < |\bar{J}|$.

For $X = LexBFS$, $lab_{S_0(X)}(J) \prec_{S_0(X)} lab_{S_0(X)}(I)$ iff $I \neq J$ and $min((I - J) \cup (J - I)) \in I - J$, or equivalently $\bar{I} \neq \bar{J}$ and $min((\bar{J} - \bar{I}) \cup (\bar{I} - \bar{J})) \in \bar{J} - \bar{I}$, i.e. $lab_{S_0(X)}(\bar{I}) \prec_{S_0(X)} lab_{S_0(X)}(\bar{J})$.

The proof for $X = LexDFS$ is obtained from the proof for $X = LexBFS$ by replacing min by max. □

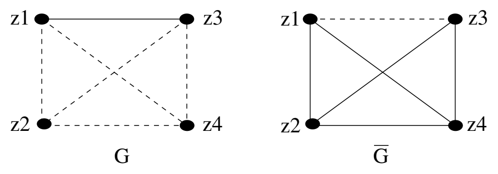

Fig. 3. A graph for which GLS cannot find exactly the GGS nor the DFS orderings of the complement. Dashed lines indicate non-edges.

Lemma 6.4. *Let $S_{\overline{BFS}} = (L, \preceq, l_0, UPLAB)$, with $L = \mathbb{N}^+$, '$\preceq$' = '$\geq$', $l_0 = 1$ and $UPLAB(l, i) = l + 1$ if $l = i$ and l otherwise. Then property $Comp(S_{\overline{BFS}}, S(BFS))$ holds.*

Proof. Let $S = S_{\overline{BFS}}$. Let us first show that for any integer $n \geq 1$ and any subset I of $\mathbb{N}_n^+$, $labs_S(I) = n + 1$ if $I = \mathbb{N}_n^+$ and $min(\mathbb{N}_n^+ - I)$ otherwise. Let n be an integer ≥ 1, I be a subset of $\mathbb{N}_n^+$ and for any integer $i \geq 0$, let $l_i = labs_S(I \cap \mathbb{N}_i^+)$. If $I = \mathbb{N}_n^+$ then for any i from 0 to n $l_i = i + 1$ (by induction on i, as $l_0 = 1$ and for any i from 1 to n $l_i = UPLAB(l_{i-1}, i) = UPLAB(i, i) = i + 1$) and therefore $labs_S(I) = l_n = n + 1$. We suppose now that $I \neq \mathbb{N}_n^+$. Let $p = min(\mathbb{N}_n^+ - I)$. $l_{p-1} = labs_S(\mathbb{N}_{p-1}^+) = p$ by the previous result, and for any $i \geq p$ $l_i = p$ (by induction on i, as $l_p = l_{p-1} = p$ and for any $i > p$ l_i is either equal to $l_{i-1} = p$ or to $UPLAB(l_{i-1}, i) = UPLAB(p, i) = p$). Thus $labs_S(I) = l_n = p = min(\mathbb{N}_n^+ - I)$.

Let us show now that $Comp(S, S(BFS))$. Let n be an integer ≥ 1 and I, J be subsets of $\mathbb{N}_n^+$. $labs_{S(BFS)}(\mathbb{N}_n^+ - I) \prec_{S(BFS)} labs_{S(BFS)}(\mathbb{N}_n^+ - J)$ iff $\mathbb{N}_n^+ - J \neq \emptyset$ and (either $\mathbb{N}_n^+ - I = \emptyset$ or $min(\mathbb{N}_n^+ - I) > min(\mathbb{N}_n^+ - J)$), i.e. $J \neq \mathbb{N}_n^+$ and (either $I = \mathbb{N}_n^+$ or $min(\mathbb{N}_n^+ - I) > min(\mathbb{N}_n^+ - J)$), which is exactly $labs_S(I) \prec_S labs_S(J)$. □

Theorem 6.5. *For any $X \in \{BFS, MNS, LexBFS, LexDFS, MCS\}$, let $S = S_{\overline{BFS}}$ if $X = BFS$ and $Rev(S_0(X))$ otherwise. Then the set of S-GLS orderings of G is equal to the set of X orderings of $\overline{G}$ for every graph G.*

For any $X \in \{GGS, DFS\}$, there is no labeling structure S such that the set of S-GLS orderings of G is equal to the set of X orderings of $\overline{G}$ for every graph G.

Proof. The first result follows from Theorem 6.1 and Lemmas 6.3 and 6.4.

Let $X \in \{GGS, DFS\}$. Let us assume for contradiction that there is a labeling structure S such that the set of S-GLS orderings of G is equal to the set of X orderings of $\overline{G}$ for every graph G. Let G be the graph shown in Fig. 3. Let $l_1 = UPLAB(l_0, 1)$. Consider ordering $\sigma = (z_1, z_2, z_3, z_4)$. Since σ is an X ordering of $\overline{G}$, σ is an S-GLS ordering of G. As z_3 is chosen at iteration 3 with label l_1 whereas z_4 has label l_0, $l_1 \not\prec l_0$. We now consider ordering $\tau = (z_1, z_3, z_2, z_4)$. Since $l_1 \not\prec l_0$ τ is an S-GLS ordering of G. But τ is not an X ordering of $\overline{G}$. □

For $X \in \{GGS, DFS\}$, though there is no labeling structure S such that S-GLS computes exactly the X orderings of the complement of the input graph, there is a labeling structure S such that S-GLS only computes X orderings of the complement. For instance, $Rev(S(LexDFS))$-GLS only computes X orderings of $\overline{G}$ for every graph G since every $LexDFS$ ordering of $\overline{G}$ is also an X ordering of $\overline{G}$.

We can deduce from the previous result some sufficient (and necessary) conditions for S-GLS to compute only X orderings of $\overline{G}$ for every graph G.

Theorem 6.6. *For any X in $\{MNS, LexBFS, LexDFS, MCS\}$, $Prop_X(Rev(S))$ (which is the conjunction of the properties check-marked in Table 1 as applied to S, endowed with its dual order) is a sufficient condition on a labeling structure S for S-GLS on input graph G to compute only X orderings of $\overline{G}$ for every graph G.*

It is also a necessary condition if X is in $\{LexBFS, LexDFS\}$, but not if X is in $\{MNS, MCS\}$.

Proof. Let $X \in \{MNS, LexBFS, LexDFS, MCS\}$. By Theorem 6.5, S-GLS on input graph G only computes X orderings of $\overline{G}$ for every graph G iff it computes only $Rev(S_0(X))$-GLS orderings of G for every graph G. By Theorem 5.2 it is still equivalent to $Sub(S, Rev(S_0(X)))$, i.e. $Sub(Rev(S), S_0(X))$. Thus by Theorem 5.3(A1) we are looking for a sufficient (and necessary) condition on a labeling structure S for $Rev(S)$-GLS to compute only X orderings of G for every graph G. The result follows from Theorem 5.9. □

Characterization 6.7. *Let G be a graph, σ be an ordering of V and $X \in \{MNS, LexBFS, LexDFS, MCS\}$. Then σ is an X ordering of $\overline{G}$ if and only if there is a labeling structure S such that $Prop_X(Rev(S))$ holds and σ is an S-GLS ordering of G.*

Proof. If σ is an X ordering of $\overline{G}$ then by Theorem 6.5 it is a $Rev(S_0(X))$-GLS ordering of G, and $Prop_X(Rev(Rev(S_0(X)))) = Prop_X(S_0(X))$ holds by Lemma 5.8. The converse follows from Theorem 6.6. □

7. Implementing graph searches

In this section we complete the results given in [1] about the peo-computing instances of GLS and Test-GLS, which are recalled in Theorem 7.1. Algorithm Test-GLS takes as input a graph $G = (V, E)$, a labeling structure S and an ordering σ of V and determines whether σ is an S-GLS ordering or not. It is obtained from GLS by replacing the choice of an unnumbered vertex with maximal label at iteration i by a test of maximality of the label of $\sigma(i)$ among unnumbered vertices. If the test is negative then the algorithm stops with the answer no, otherwise it goes on and will answer yes if all tests have been positive. Test-S-GLS denotes the instance of Test-GLS having labeling structure S as input. For simplicity, for any $X \in BasicAlg$, algorithms $S_0(X)$-GLS and Test-$S_0(X)$-GLS are denoted X and Test-X respectively. We will present implementations and complexity bounds of X and Test-X for each $X \in BasicAlg$, first on a graph G and then on the complement $\overline{G}$ of a graph G without computing this complement. For this we will extensively use the implementation with a list of lists, which generalizes the implementation of LexBFS given in [9].

An implementation of S-GLS computes only S-GLS orderings, but is not necessarily able to compute every S-GLS ordering of a graph. It follows that S-GLS can be implemented by any implementation of S'-GLS as soon as any S'-GLS ordering is also an S-GLS ordering for every graph G. For instance, the implementation of LexBFS given in [9] and the implementation of MCS given in [12] are also implementations of MNS. An implementation of MNS able to compute every MNS ordering of a graph can be derived from the implementation of Test-MNS with the same complexity [1]. We recall these results on LexBFS and MCS from [9,12] and the complexity bounds given in [1] for the four peo-computing instances of GLS.

Theorem 7.1 *([9,12,1]).* *LexBFS, Test-LexBFS, MCS, Test-MCS and MNS can be implemented in $O(n+m)$ time and space. LexDFS and Test-LexDFS can be implemented in $O(n^2)$ time and $O(n+m)$ space. Test-MNS can be implemented in $O(n(n+m))$ time and $O(n^2)$ space.*

By Theorem 5.2 if $Sub(S, S')$ holds then any implementation of S-GLS is also an implementation of S'-GLS. In particular, S-GLS can be implemented by replacing the partial order on labels by anyone of its linear extensions. This property does not hold for Test-S-GLS, which has to keep closer to the labeling structure S.

The implementation of GLS with a list of lists defined for LexBFS in [9] and adapted to LexDFS in [1] can be generalized to S-GLS.

Implementation of S-GLS with a list of lists

We assume that $\preceq$ is a total order on labels (if it is not the case then replace $\preceq$ by one of its linear extensions). As in the implementation of LexBFS, the current state of labels is represented by a list L of non-empty lists l. Each list l contains the unnumbered vertices having a given label, and the list L is ordered in decreasing order of the labels associated with the lists l (see [9] for a full description of the data structure). It is initialized with a unique list l containing all vertices of the graph. At iteration i, $\sigma(i)$ is picked in the first list in L and removed from this list. Then list L is updated: each unnumbered neighbor of $\sigma(i)$ whose label is modified at iteration i is removed from its list l and inserted into the list corresponding to its new label, which may be created at iteration i or already existing. For some labeling structures, this can be done without knowing the vertex labels. In that case the labels are not explicitly stored and the implementation is said to be *label-free*.

For Test-S-GLS, order $\preceq$ has to be a total order (it cannot be replaced by one of its linear extensions). At iteration i, we test whether $\sigma(i)$ is in the first list or not.

We clearly have the following lemma.

Lemma 7.2. *Let S be a labeling structure with a total order $\preceq$ such that in an implementation of S-GLS with a list L of lists, the list L (and vertex labels if needed) can be updated at each iteration i from 1 to n in $O(t_{update}(i, n))$ time and $O(n+m)$ space.*

Then S-GLS and Test-S-GLS can be implemented in $O(n + \sum_{i=1}^{n} t_{update}(i, n))$ time and $O(n+m)$ space.

For instance, for LexBFS, as described in [9], list L is updated at iteration i by transferring each unnumbered neighbor of $\sigma(i)$ from its list l into a new list l_1 inserted into L just before l. Thus we get $t_{update}(i, n) = 1 + |N(\sigma(i))|$ and $O(n + \sum_{i=1}^{n} t_{update}(i, n)) = O(n+m)$. LexDFS, as described in [1], proceeds as LexBFS, but it requires at the end of each iteration i an additional scan of L to form a new list L_1 with all new lists l_1 in the same order, which requires $O(n)$ time. The new list L is obtained by concatenation of L_1 and the remaining list L, which requires $O(1)$ time. Thus we get $t_{update}(i, n) = n$ and $O(n + \sum_{i=1}^{n} t_{update}(i, n)) = O(n^2)$. In both cases the implementation is label-free. MCS can also be implemented with a list of lists (instead of an array of lists as described in [12]). But it is not label-free: the label associated with each list l must explicitly be stored to make it possible to decide whether an unnumbered neighbor of $\sigma(i)$ removed from its list l must be inserted into the list l' preceding l in L (if the labels associated with l and l' are consecutive integers) or into a new list l_1 inserted into L just before l. As for LexBFS we get $t_{update}(i, n) = 1 + |N(\sigma(i))|$, and therefore linear time and space bounds for MCS.

It is well-known that BFS and DFS can be implemented in linear time and space with a queue and a stack respectively, so that GGS is linear in time and space too. To extend these results to their Test-GLS variants, we can combine an implementation with a queue or a stack and labels to make it possible to decide if $\sigma(i)$ has a maximal label or not. We can also use a label-free implementation with a list of lists, which will be useful when implementing X and Test-X on the complement of a graph.

Proposition 7.3. *For any $X \in \{GGS, BFS, DFS\}$, X and Test-X can be implemented in $O(n+m)$ time and space.*

Proof. We use a label-free implementation with a list of lists.

For *GGS* and *BFS*, list L may be active or inactive. It is initialized as active and remains so as long as at least one unnumbered vertex has no visited neighbor. At iteration i if L is active then it is updated as follows. For *GGS*, every neighbor of $\sigma(i)$ in the last list l of L is transferred into the list preceding l in L if it exists, otherwise into a new list l_1 inserted into L just before l. For *BFS*, every neighbor of $\sigma(i)$ in the last list l of L is transferred into a new list l_1 inserted into L just before l. In both cases, if the last list l has become empty by this process then L becomes inactive and will no longer be updated (except for the removal of $\sigma(j)$ at further iterations j).

For *DFS*, at iteration i every unnumbered neighbor of $\sigma(i)$ is transferred from its list l into a unique new list l_1 inserted into L as its first element.

We conclude with Lemma 7.2 since in each case we have $t_{update}(i, n) = 1 + |N(\sigma(i))|$. □

GLS can in some cases be implemented with the data structure of a stratified tree defined by Van Emde Boas [13,14] to manipulate priority queues. In particular, if labels are positive integers ordered by $\leq$ and for any integer $n \geq 1$, L_n is a subset of an interval $[1, r(n)]$ with $r(n)$ in $O(n)$, then S-GLS and Test-S- GLS can be implemented in $O((n+m)\ \log\log n+m\, t_{UPLAB}(n))$ time and $O(n + m)$ space, where $t_{UPLAB}(n)$ is the time required to update a label of L_n with function *UPLAB* [1].

We now turn to implementations of X and Test-X on the complement of a graph without computing this complement. Habib et al. [7] showed that a LexBFS search of $\overline{G}$ can be performed in $O(n + m)$ time and space by a slight modification of the standard LexBFS algorithm. We extend this result in the following theorem.

Theorem 7.4. *For any $X \in \{BFS, MNS, LexBFS, LexDFS, MCS\}$, X and Test-X on $\overline{G}$ can be implemented with the same time and space bounds (in function of the numbers n of vertices and m of edges of G) as X and Test-X on G respectively.*

Proof. Let $X \in \{BFS, MNS, LexBFS, LexDFS, MCS\}$, and let $S = S_{\overline{BFS}}$ if $X = BFS$ and $Rev(S_0(X))$ otherwise. By Theorem 6.5, X on $\overline{G}$ and Test-X on $\overline{G}$ can be implemented by implementations of S-GLS on G and Test-S-GLS on G respectively. It is sufficient to show that the time and space bounds of these implementations are the same as the bounds of X on G and Test-X on G respectively.

If $X = BFS$ then $S = S_{\overline{BFS}}$. We can use a label-free implementation of S-GLS on G and Test-S-GLS on G with a list of lists L, which may be active or inactive. It is initialized as active and remains so as long as at least one unnumbered vertex is a neighbor in G of every visited vertex. It is updated at iteration i as follows. If L is active then every neighbor of $\sigma(i)$ in the last list l of L is transferred into a new list l_2 inserted into L just after l. If there was no neighbor of $\sigma(i)$ in the last list l to be transferred then L becomes inactive and will no longer be updated (except for the removal of $\sigma(j)$ at further iterations j). Thus $t_{update}(i, n) = 1 + |N(\sigma(i))|$ and by Lemma 7.2 S-GLS on G and Test-S-GLS on G are linear in time and space, as BFS on G and Test-BFS on G.

Otherwise, $S = Rev(S_0(X))$. We check that the complexity bounds of the implementation used for $S_0(X)$ also hold for $Rev(S_0(X))$, i.e. that replacing $\preceq$ by its dual does not affect the complexity bounds. For an implementation with a list of lists L, just reverse the total order on L. For the implementation of Test-MNS given in [1] with a Boolean matrix *Preceq* such that for any unnumbered vertices v and w, $Preceq(v, w) = True$ iff $label(v) \preceq label(w)$, just transpose matrix *Preceq*. In each case replacing $\preceq$ by its dual does not affect the complexity bounds. □

The implementations of X on $\overline{G}$ and Test-X on $\overline{G}$ with a list of lists deduced from Theorem 6.5 can be directly derived from the implementations of X on G and Test-X on G with a list of lists, which yields a new proof of the corresponding results in Theorem 6.5. The idea is that in these cases, the list L of lists can be updated at each iteration i by moving the non-neighbors of $\sigma(i)$ instead of its neighbors. For LexBFS, instead of transferring each unnumbered *neighbor* of $\sigma(i)$ from its list l into a new list l_1 inserted into L just *before* l, we transfer each unnumbered *non-neighbor* of $\sigma(i)$ from its list l into a new list l_2 inserted into L just *after* l. Thus we obtain the same orderings by simultaneously replacing G by $\overline{G}$ and the order on labels by its dual. It follows that the set of $Rev(S(LexBFS))$-GLS orderings of G is equal to the set of *LexBFS* orderings of $\overline{G}$ for every graph G, which is a result of Theorem 6.5. Similarly, for LexDFS we can form a new list L_2 with all new lists l_2 in the same order and concatenate the remaining list L and L_2 in this order instead of concatenating L_1 and the remaining list L in this order. Once again we have replaced G by $\overline{G}$ and the order on labels by its dual. The same technique can be used for MCS, by transferring each unnumbered neighbor of $\sigma(i)$ from its list l either into the list l' following l in L (if the labels associated with l and l' are consecutive integers) or into a new list l_2 inserted into L just after l. For BFS, if L is active then every non-neighbor of $\sigma(i)$ in the last list l of L is transferred into a new list l_2 inserted into L just after l. If there was no non-neighbor of $\sigma(i)$ in the last list l to be transferred then L becomes inactive. This is exactly the implementation of $S_{\overline{BFS}}$-GLS on $\overline{G}$ described in the proof of Theorem 7.4. Note that $S_{\overline{BFS}}$ is different from $Rev(S(BFS))$ because according to $Rev(S(BFS))$, the first list of L would be modified when updating L instead of its last list (this dissymmetry does not occur for LexBFS, LexDFS and MCS since every list of L is modified when updating L).

As expected from Theorem 6.5, in an implementation of GGS or DFS by a list of lists, the list L cannot be updated at iteration i by moving the non-neighbors of $\sigma(i)$ instead of its neighbors. Dahlhaus et al. [6] gave an implementation of a DFS search in the complement of a graph in $O(n + m)$ time and space.

Theorem 7.5 ([6]). *DFS on $\overline{G}$ can be implemented in $O(n + m)$ time and space.*

The implementation of DFS on $\overline{G}$ is given in [6] uses several stacks instead of the usual DFS stack. It can be described as a label-free implementation with a list L of lists, which will make it easy to extend these linear bounds to Test-DFS on $\overline{G}$. The list L is updated at iteration i as follows. Every neighbor of $\sigma(i)$ in the first list l of L is transferred into a new list l_2 inserted just after l in L. The other lists of L are scanned and the non-neighbors of $\sigma(i)$ are transferred into the first list l.

These scans globally cost $O(n+m)$ because (1) testing whether a vertex is a neighbor of $\sigma(i)$ or not takes constant time by previously marking the neighbors of $\sigma(i)$, (2) scanning the neighbors of $\sigma(i)$ globally costs $O(n+m)$ time and (3) scanning and transferring the non-neighbors of $\sigma(i)$ also costs $O(n+m)$ time since scanning and transferring them for the first time costs $O(n)$ time and a vertex can only be transferred again if it has previously been transferred from the first list l of L into a new list l_2, which globally costs $O(n+m)$ time. Thus $O(n+\sum_{i=1}^{n} t_{update}(i,n)) = O(n+m)$ and by Lemma 7.2, these linear bounds also hold for Test-DFS on $\overline{G}$.

Corollary 7.6. *Test-DFS on $\overline{G}$ can be implemented in $O(n+m)$ time and space.*

As any BFS or DFS ordering is a GGS one, GGS on $\overline{G}$ can be implemented in linear time and space by an implementation of BFS on $\overline{G}$ or DFS on $\overline{G}$. However, to prove that Test-GGS on $\overline{G}$ is also linear in time and space, we define a label-free implementation of GGS on $\overline{G}$ with a list of lists inspired from those of GGS on G and DFS on $\overline{G}$.

Theorem 7.7. *GGS and Test-GGS on $\overline{G}$ can be implemented in $O(n+m)$ time and space.*

Proof. We use the label-free implementation of GGS with a list of lists described in the proof of Proposition 7.3 with input graph $\overline{G}$ instead of G, with the following modification. In the implementation of GGS on G, it is by scanning the list of neighbors of $\sigma(i)$ that the neighbors of $\sigma(i)$ in the last list l of L are transferred into the list preceding l in L or into a new list l_1 inserted into L just before l. In the implementation of GGS on $\overline{G}$, this transfer of the non-neighbors of $\sigma(i)$ from the last list l of L is performed by scanning this last list l. These scans globally cost $O(n+m)$ because (1) testing whether a vertex is a neighbor of $\sigma(i)$ or not takes constant time by previously marking the neighbors of $\sigma(i)$, (2) scanning the neighbors of $\sigma(i)$ globally costs $O(n+m)$ time and (3) scanning and transferring the non-neighbors of $\sigma(i)$ only costs $O(n)$ time since a vertex can only be transferred once. We conclude with Lemma 7.2. □

8. Concluding remarks

In this paper we have described a general labeling algorithm for computing vertex orderings of a graph, and showed how conditions imposed on the labeling structures allow this algorithm to perform various graph searches.

Our GLS algorithm captures vertex labeling algorithms beyond just graph search. As an example, we characterized labeling structures that yield algorithms that compute some types of search orderings in the complement of a graph while avoiding computing the complement, and also showed the limitations on GLS algorithm in such a context. The benefit of using a single general algorithm like GLS is that one can conveniently prove results for a wide variety of specific algorithms simply by using the properties of the labeling structures.

Finally, we note that GLS can easily be extended to capture some important variants of graph search. Several algorithms use multiple sweeps of a graph search to gather information about a graph, where each subsequent sweep uses a previous ordering to compute a new ordering with additional properties [4]: one could define an extended labeling structure that allows various different initial labels to be assigned to vertices based on a given input ordering.

References

[1] A. Berry, R. Krueger, G. Simonet, Geneviève Simonet, Maximal label search algorithms to compute perfect and minimal elimination orderings, SIAM J. Discrete Math. 23 (1) (2009) 428–446.
[2] A. Brandstädt, F.F. Dragan, F. Nicolai, LexBFS orderings and powers of chordal graphs, Discrete Math. 171 (1–3) (1997) 27–42.
[3] A. Bretscher, D. Corneil, M. Habib, C. Paul, A simple linear time lexbfs cograph recognition algorithm, SIAM J. Discrete Math. 22 (4) (2008) 1277–1296.
[4] D.G. Corneil, Lexicographic breadth first search—a survey, in: Proceedings of WG2004, in: Lecture Notes in Computer Science, vol. 3353, 2004, pp. 1–19.
[5] D. Corneil, R. Krueger, A unified view of graph searching, SIAM J. Discrete Math. 22 (2008) 1259–1276.
[6] E. Dahlhaus, J. Gustedt, R.M. McConnell, Partially complemented representations of digraphs, Discrete Math. Theoret. Comput. Sci. 5 (2002) 147–168.
[7] M. Habib, R. McConnell, C. Paul, L. Viennot, Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing, Theoret. Comput. Sci. 234 (1–2) (2000) 59–84.
[8] B. Jamison, S. Olariu, On the semi-perfect elimination, Adv. Appl. Math. 9 (3) (1988) 364–376.
[9] D.J. Rose, R.E. Tarjan, G.S. Lueker, Algorithmic aspects of vertex elimination on graphs, SIAM J. Comput. 5 (2) (1976) 266–283.
[10] D.R. Shier, Some aspects of perfect elimination orderings in chordal graphs, Discrete Appl. Math. 7 (1984) 325–331.
[11] R.E. Tarjan, Depth first search and linear graph algorithms, SIAM J. Comput. 1 (2) (1972) 146–160.
[12] R.E. Tarjan, M. Yannakakis, Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs, SIAM J. Comput. 13 (3) (1984) 566–579.
[13] P. van Emde Boas, Preserving order in a forest in less than logarithmic time and linear space, Inform. Process. Lett. 6 (3) (1977) 80–82.
[14] P. van Emde Boas, R. Kaas, E. Zijlstra, Design and implementation of an efficient priority queue, Math. Syst. Theory 10 (1977) 99–127.