Graph Searches and Their End Vertices

Yixin Cao
Department of Computing, Hong Kong Polytechnic University, Hong Kong, China
yixin.cao@polyu.edu.hk

Zhifeng Wang
School of Computer Science and Engineering, Central South University, Changsha, China

Guozhen Rong
School of Computer Science and Engineering, Central South University, Changsha, China

Jianxin Wang
School of Computer Science and Engineering, Central South University, Changsha, China

Abstract

Graph search, the process of visiting vertices in a graph in a specific order, has demonstrated magical powers in many important algorithms. But a systematic study was only initiated by Corneil et al. a decade ago, and only by then we started to realize how little we understand it. Even the apparently naïve question "which vertex can be the last visited by a graph search algorithm," known as the end vertex problem, turns out to be quite elusive. We give a full picture of all maximum cardinality searches on chordal graphs, which implies a polynomial-time algorithm for the end vertex problem of maximum cardinality search. It is complemented by a proof of NP-completeness of the same problem on weakly chordal graphs. We also show linear-time algorithms for deciding end vertices of breadth-first searches on interval graphs, and end vertices of lexicographic depth-first searches on chordal graphs. Finally, we present $2^n \cdot n^{O(1)}$-time algorithms for deciding the end vertices of breadth-first searches, depth-first searches, and maximum cardinality searches on general graphs.

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases maximum cardinality search, (lexicographic) breadth-first search, (lexicographic) depth-first search, chordal graph, weighted clique graph, end vertex

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2019.1

Related Version A full version of the paper is available at `https://arxiv.org/abs/1905.09505`.

Funding Supported by the RGC grants 15201317 and 15226116, and the NSFC grant 61972330.

1 Introduction

Breadth-first search (BFS) and depth-first search (DFS) are the most fundamental graph algorithms, and the standard opening of a course on this subject. Their use can be found, sometimes implicitly, in most graph algorithms. In general, a graph search is a systematic exploration of a graph, and its core lies on the strategy of how to choose the next vertex to visit. Mostly greedy, graph searches are very simple but sometimes have magical powers. DFS has played a significant role in Tarjan's award-winning work, in testing planarity [19] and in finding strongly connected components [25].

Two other search algorithms, lexicographic breadth-first search (LBFS) [21] and maximum cardinality search (MCS) [26], were invented for the purpose of recognizing chordal graphs, i.e., graphs not containing any induced cycle on four or more vertices. On a chordal graph, both LBFS and MCS produce perfect elimination orderings (see definition in the next section) of the graph, which exist if and only if the graph is chordal. Albeit relatively less well known compared to BFS and DFS, LBFS and MCS did find important applications. LBFS is used in scheduling [22], and is the base of the recent linear-time algorithm for computing modular decomposition of a graph [27]. MCS is used in testing acyclic hypergraphs and in computing minimum cuts of a graph and find forest decompositions.

Simon [24] proposed an interesting way of using LBFS. It conducts LBFS more than once, and each new run uses previous runs in breaking ties; in particular, except the first, each run starts from the last vertex of the previous run. This generic approach turns out to be very useful, e.g., the extremely simple recognition algorithm for unit interval graphs [7]. See the survey of Corneil [8] for more algorithms using multiple runs of LBFS. Some of these results have a flavor of "ad-hoc": We do not fully understand the execution process of LBFS.

The outputs of BFS and DFS are usually rooted spanning forests of the graph, while LBFS and MCS produce orderings of its vertices. To have a unified view of them, Corneil et al. [11] focused on the ordering of the vertices being first visited and conducted a systematic study of them. This study motivates them to propose the lexicographic version of DFS, lexicographic depth-first search (LDFS), another very powerful graph search [9], and a very general search paradigm, maximum neighborhood search (MNS). They showed that all the aforementioned graph searches can be characterized by variants of the so-called four-vertex condition. These nice characterizations are however not sufficient to allow us to answer the ostentatiously naïve question: Which vertex can be the last of such an ordering? Corneil et al. [10] defined the end vertex problem and studied it from both combinatorial and algorithmic perspectives. Apart from a natural starting point of understanding the graph searches in general, end vertices of graph searches are of their own interest. Behind the original use of LBFS and MCS, in the recognition of chordal graphs, is nothing but the properties of their end vertices, which are always simplicial on a chordal graph [21, 26, 23, 4]. Moreover, the success of multiple-run LBFS crucially hinges on the end vertices; e.g., an end vertex of a (unit) interval graph can always be assigned an extreme (i.e., leftmost or rightmost) interval [7, 13]. Important properties and use of end vertices of other graph searches can be found in [9, 17, 12].

One may find it surprising, but the end vertex problem is NP-hard for all the six mentioned graph search algorithms [11, 6, 1]. The study has thus been focused on chordal graphs and its closely related superclasses and subclasses. After all, LBFS and MCS were invented for recognition of chordal graphs, and their properties on chordal graphs have been intensively studied. (This renders the stagnation on chordal graphs a little more embarrassing.) Moreover, most applications of LBFS and LDFS are on related graph classes. The most natural superclass of chordal graphs is arguably the weakly chordal graphs, and two important subclasses are interval graphs and split graphs. It has been known that on weakly chordal graphs, the end vertex problems for all but MCS are NP-complete, while only DFS end vertex is NP-complete on chordal graphs [11, 6, 1]. There are other polynomial-time algorithms for interval graphs and split graphs, most of which actually run in linear time. We complete the pictures for, in terms of graph searches, MCS and LDFS, and, in terms of graph classes, weakly chordal graphs and interval graphs. A summary of known results is given in Fig. 1.

Blair and Peyton [5] and Galinier et al. [16] have shown that MCS of a chordal graph are closely related to its maximal cliques. Let G be a chordal graph. An MCS visits all vertices in a maximal clique of G before proceeding to another, and the next maximal clique is always chosen to have the largest intersection with a visited one. Therefore, for a minimum separator S of G, there is an MCS visiting the components of $G - S$ one by one, with S visited together with the first component. If we turn to any component C of $G - S$, and consider its closed neighborhood, (which contains C and S,) then we have a similar statement. In other words, this property on minimum separators holds in a recursive way. For an MCS end vertex z, which is necessarily simplicial, we can find a sequence of increasing separators such that the first is a minimum separator of G and the last comprises all the non-simplicial vertices in $N(z)$. An MCS ended with z has to "cross" these separators in order, and for each of them, visit the component containing z in the last. We have thus a full understanding of all MCS orderings of a chordal graph. As it turns out, this result is easier to be presented in the

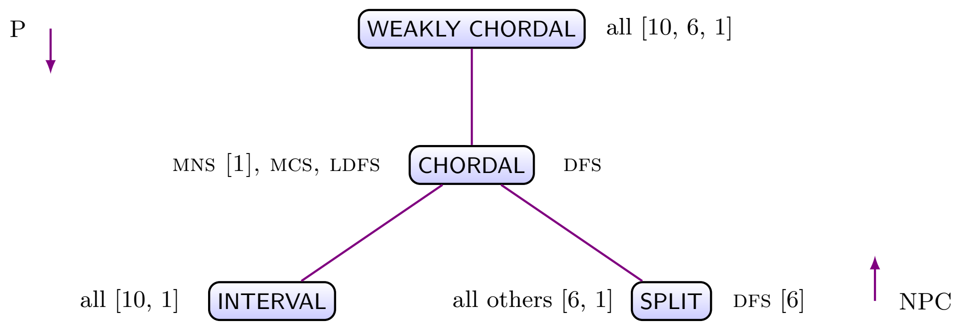

Figure 1 A summary of the known complexity of the end vertex problem for the six graph search algorithms. For each graph class, the end vertex problem of graph searches listed to the left of it can be solved in polynomial time on this class, while those to the right are NP-hard. The complexity of the BFS end vertex and LBFS end vertex problems on chordal graphs are still open.

so-called weighted clique graph of G [5, 16]. It enables us to show that if we run MCS twice, first starting from z, and the second starting from the end vertex of the first run and using the first ordering to break ties, then the second run ends with z if and only if z is an MCS end vertex. As usual, n denotes the number of vertices in the input graph.

▶ **Theorem 1.** *The* MCS *end vertex problem can be solved in $O(n^2)$ time on chordal graphs.*

We complement this result by showing that the MCS end vertex problem becomes NP-complete on weakly chordal graphs; the proof is inspired by and adapted from [1].

▶ **Theorem 2.** *The* MCS *end vertex problem is NP-complete on weakly chordal graphs.*

We then turn to LDFS on chordal graphs. Surprisingly, the characterization of Berry et al. [3] for end vertices of MNS on chordal graphs is also true for LDFS: A simplicial vertex z of a chordal graph G is an LDFS end vertex if and only if the minimal separators of G in $N(z)$ are totally ordered by inclusion. We also show a simple algorithm for solving the BFS end vertex problem on interval graphs.

▶ **Theorem 3.** *There are linear-time algorithms for solving the* LDFS *end vertex problem on chordal graphs and for solving the* BFS *end vertex problem on interval graphs.*

We have to, nevertheless, leave open the BFS and LBFS end vertex problems on chordal graphs. Since both can be solved in linear time on split graphs, we conjecture that they can be solved in polynomial time on chordal graphs. It is extremely rare that a problem is hard on chordal graphs but easy on split graphs.

We also consider algorithms for solving the end vertex problems on general graphs. By enumerating all possible orderings, a trivial algorithm can find all end vertices of any graph search in $n! \cdot n^{O(1)}$ time. On the other hand, with the only exception of BFS, the reductions used in proving NP-hardness of the end vertex problems are linear reductions from (3-)SAT. As a result, these problems cannot be solved in subexponential time, unless the exponential time hypothesis fails [20]. A natural question is thus which of them can be solved in $2^{O(n)}$ time. If we put them under closer scrutiny, we will see that these graph searches are somewhat different: When selecting the next vertex, MCS only needs to know which vertices have been visited, while the order of visiting them is immaterial. In contrast, the other graph searches are not *oblivious* and need to keep track of the whole visiting history. It is straightforward to use dynamic programming to solve the MCS end vertex problem in $2^n \cdot n^2$ time. Interestingly, a similar approach actually works for the BFS and DFS end vertex problems.

▶ **Theorem 4.** *There are $2^n \cdot n^{O(1)}$-time algorithms that solve the end vertex problems of the following graph searches:* MCS, BFS, *and* DFS.

2 Preliminaries

All graphs discussed in this paper are undirected and simple. The vertex set and edge set of a graph G are denoted by, respectively, $V(G)$ and $E(G)$, and we use $n = |V(G)|$ and $m = |E(G)|$ to denote their cardinalities. For a subset $X \subseteq V(G)$, denote by $G[X]$ the subgraph of G induced by X, and by $G - X$ the subgraph $G[V(G) \setminus X]$. The *degree* of a vertex v is the number of neighbors it has, i.e., $d(v) = |N(v)|$. A vertex v is *simplicial* if $N[v]$ induces a complete graph. Two distinct vertices u and v are *true twins* if $N[u] = N[v]$, and *false twins* if $N(u) = N(v)$; note that true twins are adjacent while false twins are not.

A vertex set S is a *u-v separator* if u and v are not in S and they are not connected in $G - S$, and a *u-v separator* is *minimal* if no proper subset of S is a *u-v separator*. We call S a (minimal) separator if it is a (minimal) u-v separator for some pair of u and v, and it is a *minimum separator* of G if it has the smallest cardinality among all separators of G.

An *ordering* σ of the vertices of G is a bijection from $V(G) \to \{1, \dots, n\}$. For two vertices u and v, we use $u <_\sigma v$ to denote $\sigma(u) < \sigma(v)$. The *end vertex* of σ is the vertex z with $\sigma(z) = n$. Given a graph G and a vertex $z \in V(G)$, the *end vertex problem* for graph search S is to determine whether there is an S-ordering of G of which z is the end vertex.

A graph is *chordal* if it contains no induced cycle on four or more vertices. A graph is chordal if and only if it can be made empty by removing simplicial vertices from the remaining graph one by one; the order of the vertices removed is called a *perfect elimination ordering* [15]. The greedy strategy of MCS is to choose an unvisited vertex with the maximum number of visited neighbors. On a chordal graph G, the last vertex of any MCS is simplicial, and thus the reversal of an MCS ordering is always a perfect elimination ordering [26].

To avoid unnecessary digressions, we consider only connected graphs.

3 Maximum cardinality search on chordal graphs

Another important characterization of chordal graphs is through its maximal cliques. A graph G is chordal if and only if we can arrange its maximal cliques as a tree such that for each vertex $v \in V(G)$, maximal cliques containing v induce a subtree; such a tree is called a *clique tree* of G [14]. A chordal graph G has at most n maximal cliques [14], and for any pair of adjacent K_i and K_j on the clique tree, intersection $K_i \cap K_j$ is a minimal separator of G.

Out of a chordal graph G, we can define a *weighted clique graph* $C(G)$ as follows. It has ℓ vertices, where ℓ is the number of maximal cliques of G, and each vertex is labeled by a distinct maximal clique of G. To simplify the presentation, we will refer to vertices of $C(G)$ as cliques; note that we are not going to use cliques of the graph $C(G)$ in this paper. There is an edge between maximal cliques K_i and K_j, $1 \leq i, j \leq \ell$, if and only if $K_i \cap K_j$ is a minimal x-y separator for all $x \in K_i \setminus K_j$ and $y \in K_j \setminus K_i$. We label this edge with $K_i \cap K_j$, and set its weight to be $|K_i \cap K_j|$. It is known that a tree on the maximal cliques of G is a clique tree of G if and only if it is a maximum spanning tree of $C(G)$ [2, 5, 16], i.e., a spanning tree of $C(G)$ with the maximum total edge weights.

▶ **Proposition 5.** *Let G be a chordal graph and $C(G)$ the weighted clique graph of G. A set $S \subseteq V(G)$ is a minimal separator of G if and only if it is the label for some edge of $C(G)$.*

One can use Prim's algorithm to find a maximum spanning tree of G. (Although proposed for finding a minimum spanning tree, Prim's algorithm can be easily modified to find a maximum one.) Starting from an arbitrary clique, it grows the tree by including one edge and one clique at a time, while the edge is chosen to have the largest weight among those crossing the partial tree that has been built, i.e., with one end in the current tree and the other not. In the same spirit of graph search orderings, we can define a *Prim ordering* to be the order maximal cliques of G being included by Prim's algorithm, applied to $C(G)$.

Let π be an ordering of the maximal cliques of G. We say that an ordering σ of $V(G)$ is *generated by* π if $K_u <_\pi K_v$ implies $u <_\sigma v$, where K_u and K_v are the first maximal cliques in π containing u, and respectively, v. If $\pi = \langle K_1, K_2, \ldots, K_\ell\rangle$ and $c_i = |K_i \setminus \bigcup_{j=1}^{i-1} K_j|$ for $1 \le i \le \ell$, then σ can be represented as

$$\underbrace{\sigma^{-1}(1), \ldots, \sigma^{-1}(c_1)}_{K_1}, \underbrace{\sigma^{-1}(c_1+1), \ldots, \sigma^{-1}(c_1+c_2)}_{K_2 \setminus K_1}, \ldots, \underbrace{\sigma^{-1}(n-c_\ell+1), \ldots, \sigma^{-1}(n)}_{K_\ell \setminus \bigcup_{j=1}^{\ell-1} K_j}.$$

The following has been essentially observed by Blair and Peyton [5], who however only stated explicitly one direction. For the sake of completeness, we give a proof here.

▶ **Lemma 6.** *Let G be a chordal graph. An ordering σ of $V(G)$ is an* MCS *ordering of G if and only if it is generated by some Prim ordering π of $C(G)$.*

Proof. The only if direction has been proved by Blair and Peyton [5, Lemma 4.8 and Theorem 4.10]. Here we show the if direction. Suppose that σ is generated by π. We may renumber the vertices in G such that $\sigma = \langle v_1, v_2, \ldots, v_n\rangle$, and renumber the maximal cliques such that $\pi = \langle K_1, K_2, \ldots, K_\ell\rangle$. Let $K'_i = K_i \setminus \bigcup_{j=1}^{i-1} K_j$ for $1 \le i \le \ell$; note that $\{K'_1, K'_2, \ldots, K'_\ell\}$ is a partition of $V(G)$. We show by induction that for each $1 \le i \le n$, there is an MCS ordering of G of which the first i vertices are $v_1, \ldots, v_i$; in other words, among vertices $v_i, \ldots, v_n$, vertex v_i has the maximum number of neighbors in the first $i-1$ vertices. It is vacuously true for $i = 1$. Now suppose that it is true for v_p, we show that it is also true for v_{p+1}.

When $v_{p+1} \in K'_1 = K_1$, it is adjacent to all previous vertices and we are done. In the rest $v_{p+1} \in K'_t$ for some $t > 1$. Let $A = \bigcup_{j=1}^{t-1} K_j$; note that $v_{p+1} \notin A$. For any $q > p$, let G_q denote the subgraph of G induced by $v_1, v_2, \ldots, v_p$, and v_q. By the induction hypothesis, $\langle v_1, v_2, \ldots, v_p, v_q\rangle$ is an MCS ordering of G_q. Since G_q is chordal, v_q is simplicial in it. Therefore, $N(v_q) \cap A$ is a clique for all $q > p$; denote it by X_q. We argue by contradiction that there must be $1 \le s < t$ such that $X_q \subseteq K_s$. We find an i with $1 \le i < t$ such that $K_i \cap X_q$ is maximal. If $X_q \not\subseteq K_i$, then there is a vertex $x \in X_q \setminus K_i$; let K_j, where $1 \le j < t$, contain x. By the maximality of $K_i \cap X_q$, there exists $y \in (X_q \cap K_i) \setminus K_j$. Of the first $t-1$ maximal cliques, those containing $K_i \cap X_q$ and those containing $X_q \setminus K_i$ are disjoint. Prim's algorithm always maintains a tree of visited cliques, and this tree is a subtree of a clique tree of G. Therefore, there is an x-y separator. But this is impossible because x and y are both in X_q, hence adjacent.

For each $q > p$, there is some maximal clique K of G that contains $(N(v_q) \cap A) \cup \{v_q\}$. It cannot be one of $K_1, \ldots, K_{t-1}$ because $v_q \notin A$. Since $K_1, \ldots, K_\ell$ is a Prim ordering of $C(G)$, we have $|N(v_{p+1}) \cap A| \ge |N(v_q) \cap A|$ for all $q > p$. On the other hand, v_{p+1} is adjacent to all vertices in K_t. We can thus conclude that v_{p+1} has the maximum number of neighbors in $\{v_1, \ldots, v_p\}$, and this completes the proof. ◀

By Lemma 6, MCS orderings of a chordal graph G can be fully characterized by Prim orderings of its weighted clique graph $C(G)$. In particular, the MCS end vertices are the private vertices of the cliques last visited by Prim's algorithm. Note that a vertex v is simplicial if and only if it belongs to precisely one maximal clique, namely, $N[v]$, and a set of true twins can be visited in any order.

▶ **Corollary 7.** *Let z be a simplicial vertex in a chordal graph G. There is an* MCS *ordering of G ended with z if and only if there exists a Prim ordering of $C(G)$ ended with $N[z]$.*

Let S be a separator of G. We abuse notation to use $C(G) - S$ to denote the subgraph of $C(G)$ obtained by deleting all edges whose labels are subsets of S. The component of $C(G) - S$ containing $N[z]$ is called the z-component of $C(G) - S$. It is worth noting that $C(G) - S$ cannot be mapped back to G. In Fig. 2, for example, $C(G) - \{v_5, v_6\}$ does not have edges among $K_2, \ldots, K_5$, while edges $K_7K_8, K_7K_9, K_8K_9, K_9K_{10}$ will be removed in $C(G) - \{v_{12}, v_{13}\}$.

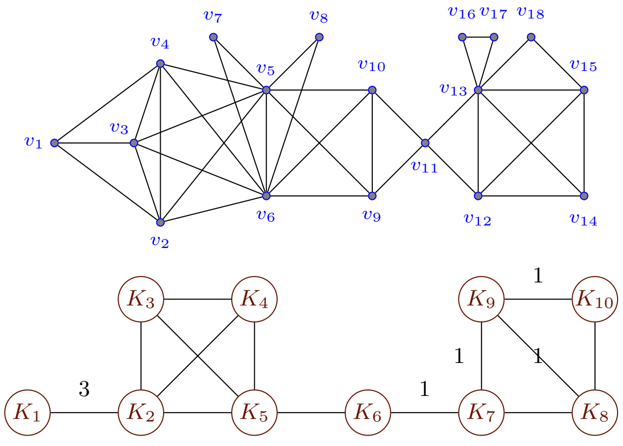

Figure 2 On the top is a chordal graph G on 18 vertices, and below the weighted clique graph of G, where all the omitted edge weights are 2. There are 10 maximal cliques $K_1 = \{v_1, v_2, v_3, v_4\}, K_2 = \{v_2, \ldots, v_6\}, K_3 = \{v_5, v_6, v_7\}, K_4 = \{v_5, v_6, v_8\}, K_5 = \{v_5, v_6, v_9, v_{10}\}, K_6 = \{v_9, v_{10}, v_{11}\}, K_7 = \{v_{11}, v_{12}, v_{13}\}, K_8 = \{v_{12}, \ldots, v_{15}\}, K_9 = \{v_{13}, v_{16}, v_{17}\}, K_{10} = \{v_{13}, v_{15}, v_{18}\}$. There are 7 simplicial vertices v_1, v_7, v_8, v_{14}, v_{16}, v_{17}, v_{18}, of which v_{14} and v_{18} are not MCS end vertices.

▶ **Proposition 8.** *Let S be a separator of a chordal graph G. For any vertex $v \notin S$, maximal cliques containing v remain connected in $C(G) - S$. For any two distinct vertices $u, v \notin S$, maximal cliques containing u and v are not connected in $C(G) - S$ iff S is a u-v separator.*

Proof. By definition, the maximal cliques containing v are connected in any clique tree of G. Since a clique tree of G is a subgraph of $C(G)$, these cliques also induce a connected subgraph in $C(G)$. For any edge in this subgraph, its label contains v, hence not a subset of S. Therefore, these cliques induce the same connected subgraph in $C(G) - S$ as in $C(G)$.

For the second assertion, we may assume $uv \notin E(G)$: Both sides are trivially false when $uv \in E(G)$. Suppose to the contradiction of the if direction that there is a path $K_0, \ldots, K_p$ in $C(G) - S$ such that $u \in K_0$ and $v \in K_p$ while $u, v \notin K_i$ for $0 < i < p$. For each $1 \leq i \leq p$, we can find a vertex $x_i \in (K_{i-1} \cap K_i) \setminus S$. (These p vertices may or may not be distinct.) Then $ux_1, x_pv \in E(G)$, while x_i and x_{i+1} are either the same or adjacent for all $1 \leq i < p$. We have thus a u-v path in G avoiding S, contradiction that S is a u-v separator.

We now consider the only if direction. Let $u = x_0, x_1, \ldots, x_p = v$ be any u-v path in G. Note that for each $0 \leq i \leq p$, maximal cliques containing x_i induce a connected subgraph, while for each $1 \leq j \leq p$, there is a maximal clique containing both x_{j-1} and x_j. We can find a path in $C(G)$ of which one end contains u and the other contains v. For each edge on this path, its label contains one of x_i, $0 < i < p$. Since maximal cliques containing u and v are not connected in $C(G) - S$, the label of at least one edge on this path is a subset of S. By the first assertion, at least one of $x_1, \ldots x_{p-1}$ is in S. In other words, every u-v path intersects S. Therefore, S is a u-v separator. This concludes the proof. ◀

We say that a minimum-weight edge e of $C(G)$ – by Proposition 5, its label is a minimum separator of G, – is a *critical edge* for maximal clique K if one end of e is in the same component as K after all minimum-weight edges, including e, are removed from $C(G)$. In other words, there is a path connecting K and e on which every edge has weight larger than e. In Fig. 2, e.g., K_6K_7 is a critical edge for all cliques but K_9, while K_8K_9 and $K_{10}K_9$ are critical edges for K_8 and K_{10} respectively. The following fact explains "critical" in the name.

▶ **Proposition 9.** *Let z be a simplicial vertex of a connected chordal graph G, and let S_1, …, S_k be the labels of all critical edges for $N[z]$. In any Prim ordering of $C(G)$, cliques in the z-component of $C(G) - S_1 - \cdots - S_k$ appear consecutively. Moreover, if $S_1 = \cdots = S_k$, then the z-component of $C(G) - S_1$ can be visited in the end.*

Proof. Note that $C(G)$ is connected since G is connected. Let T denote the z-component of $C(G) - S_1 - \cdots - S_k$. Being minimum separators of G, all of S_1, …, S_k have the same size; let it be t. Note that the weight of every edge in T is strictly larger than t; otherwise, we can find a path from $N[z]$ to such an edge in T, and identify another critical edge for $N[z]$ on this path.

Let π be any Prim ordering of $C(G)$. We consider the first maximal clique K in T visited by π. If $\pi(K) \neq 1$, the edge leading to K has weight t. By Prim's algorithm, when K is visited, for each clique K' with $K' <_\pi K$, all the edges between K' and its unvisited neighbors have weight t. All edges between T and other components have weight t as well, while all edges inside T have weight $> t$. Therefore, the maximal cliques in T must be finished before a clique out of T is visited. This concludes the first assertion.

For the second assertion, suppose that $S = S_1 = \cdots = S_k$. We give a Prim ordering that visits cliques in T in the end. It starts from a clique not in T, and it suffices to show that all cliques out of T have been visited before the first in T. By the definition of $C(G)$, in each component of $C(G) - S$, there is a maximal clique containing S. Therefore, by Proposition 8, there is an edge with label S between any two components of $C(G) - S$. In other words, the cliques not in T are connected in $C(G)$. Since the edges connecting T and other components of $C(G) - S$ have weight t, the minimum in $C(G)$, Prim's algorithm can always choose another edge. Therefore, we can finish them before entering T. ◀

Whether a simplicial vertex z can be an MCS end vertex turns out to be closely related to the critical edges for $N[z]$. We first present a necessary condition, which is not satisfied by v_{14} and v_{18} in Fig. 2; we leave it to the reader to verify that they cannot be MCS end vertices.

▶ **Lemma 10.** *Let z be a simplicial vertex of a connected chordal graph G. If $N[z]$ is the end clique of a Prim ordering of $C(G)$, then all critical edges for $N[z]$ have the same label.*

Proof. Suppose for contradiction that there are two critical edges e_1 and e_2 for $N[z]$ with different labels. For $i = 1, 2$, let S_i be the label of e_i, and let $\mathcal{C}_i$ denote the set of components of $C(G) - S_i$ not containing $N[z]$. We argue that for any $U_1 \in \mathcal{C}_1$ and $U_2 \in \mathcal{C}_2$, they are different and there is no edge between them.

For $i = 1, 2$, by the definition of critical edges, there is a path from $N[z]$ to e_i; let K_i denote the end of e_i that is closer to $N[z]$ on this path. There must be some clique K'_i in U_i containing S_1. Note that $K_i \cap K'_i = S_i$ because K'_i and K_i are in different components of $C(G) - S_i$. Hence, $K_iK'_i$ is also a critical edge with label S_i for $N[z]$. There is a $N[z]$-K'_2 path in $C(G) - S_1$, and hence K'_2 and $N[z]$ are connected in $C(G) - S_1$. Likewise, K'_1 and $N[z]$ are connected in $C(G) - S_2$.

Since $S_1 \neq S_2$ and they have the same cardinality, we can find $v_2 \in S_2 \setminus S_1 \subset K'_2$. By Proposition 8, S_1 is not a z-v_2 separator. Thus, no maximal clique in U_1 contains v_2. It follows that U_1 remains connected in $C(G) - S_2$ (note that S_2 is a minimum separator). For the same reason, U_2 remains connected in $C(G) - S_1$. If there exists an edge between U_1 and U_2, then this edge remains in at least one of $C(G) - S_1$ and $C(G) - S_2$: It cannot have both labels S_1 and S_2. But then U_1 and U_2 are connected in $C(G) - S_1$ or $C(G) - S_2$, neither of which is possible. We can thus conclude that components in $\mathcal{C}_1 \cup \mathcal{C}_2$ are disjoint and there is no edge among them.

Let π be a Prim ordering of $C(G)$ ended with $N[z]$. Assume without loss of generality that the first visited clique in these components is from $U_1 \in \mathcal{C}_1$, then we show that $N[z]$ is visited before all components $U_2 \in \mathcal{C}_2$. Since there is no edge between U_1 and U_2, before visiting U_2, it must visit a clique from the z-component of $C(G) - S_1$. After that, however, it will not visit any edge of label S_2 before finishing this component. Therefore $N[z]$ cannot be the end clique, a contradiction. This concludes the proof. ◀

In other words, if z is an MCS end vertex, then there is a unique minimum separator of G that is "closest to z" in a sense. This, although not sufficient, can be extended to a sufficient condition for MCS end vertices as follows. To decide whether a simplicial vertex z is an MCS end vertex, we can find the minimum separator S in Proposition 9 and focus on how the z-component of $C(G) - S$ is explored. We have to start from a maximal clique not in it, and after that visit all maximal cliques in other components of $C(G) - S$ before the z-component. In this juncture we may view the z-component as a separate graph and find all critical edges for $N[z]$ with respect to this component. They also need to have the same label; suppose it is S', which is strictly larger than S. But this is not sufficient because we need to make sure that when S is crossed, it can reach a maximal clique not in the z-component of $C(G) - S'$. In Fig. 2, if we delete vertices v_{16} and v_{17}, (hence K_9,) then K_6K_7 is the only critical edge for K_8. The condition of Lemma 10 is satisfied, but v_{14} is still not an MCS end vertex.

Repeating this step recursively, we should obtain a sequence of separators with increasing cardinalities. Note that we only need to keep track of how these separators are crossed, while the ordering in each layer is irrelevant. This observation leads us to the following characterization, which subsumes Theorem 13 of Beisegel et al. [1]. For example, the sequence of critical edges for $N[v_1]$ in Fig. 2 are K_6K_7, K_2K_5, and K_1K_2, which correspond to minimal separators $\{v_{11}\}$, $\{v_5, v_6\}$, and $\{v_2, v_3, v_4\}$, respectively.

▶ **Theorem 11.** *Let z be a simplicial vertex of a connected chordal graph G. The clique $N[z]$ is a Prim end clique if and only if there is a sequence of edges e_1, e_2, …, e_k in $C(G)$, where the label of e_i is S_i, on a path ended with $N[z]$ such that*

(i) *S_1 is the label of critical edges for $N[z]$ and S_k is the set of non-simplicial vertices in $N[z]$; and*

(ii) *for $1 \le i < k$, in the z-component of $C(G) - S_i$, all the critical edges for $N[z]$ have the same lable, which is S_{i+1}.*

Moreover, every clique not in the z-component of $C(G) - S_1$ can be the start clique.

Proof. We first show the if direction. We may denote the two ends of e_i by K_i and K_i', where K_i' is in the z-component of $C(G) - S_i$. (It is possible that $K_i' = K_{i+1}$ for some $1 \le i < k$.) For each $1 \le i \le k$, we visit all the other components of $C(G) - S_i$ before using the edge K_iK_i' to enter the z-component, visiting K_i'. This is possible because of Proposition 9, and as such we produce a Prim ordering of $C(G)$ that ends with $N[z]$.

Now consider the only if direction, for which we construct the stated path by induction: We find the edges e_1, e_2, …, e_k in order, and show that for each $1 \le i \le k$, the first i edges can be extended to a path that ends with $N[z]$ and satisfies both conditions. The first edge e_1 can be any critical edge for $N[z]$, and it is on a path ended with $N[z]$ because $C(G)$ is connected. Now suppose that the first i edges, namely, e_1, …, e_i, have been selected, and we find e_{i+1} as follows. For each $1 \le j \le i$, let T_j denote the z-component of $T_{j-1} - S_j$, where $T_0 = C(G)$. If T_i comprises the only maximal clique $N[z]$, we are done.

Containing $N[z]$, cliques in T_i are last visited by Proposition 9. It is also a Prim ordering of the component itself. Therefore, Lemma 10 applies, and all the critical edges for $N[z]$ in T_i have the same label. Let S_{i+1} be this label, and let T_{i+1} be the z-component of $T_i - S_{i+1}$. We argue that there must be a maximal clique K in $T_i - T_{i+1}$ containing S_i; otherwise, the first component visited in $T_i - S_{i+1}$ would be the z-component, and then $N[z]$ cannot be the last visited clique. We can use edge K_iK to replace e_i, – note that they have the same label, – and choose any edge between K and $N[z]$ with label S_{i+1} as e_{i+1}. This concludes the inductive step and the proof. ◀

The proof of the only if direction of Theorem 11 can be directly translated into an algorithm to decide Prim end cliques, implying a polynomial-time algorithm for the MCS end vertex problem on chordal graphs. This algorithm however has to take $\Omega(n^2)$ time because the size of $C(G)$. We show a very simple algorithm below, which itself best reveals the spirit of graph searches. As long as we cross the separators in the order specified in Theorem 11, and make sure we finish other components before visiting the z-component, then it is the Prim ordering we need. On the other hand, a run of Prim's algorithm started from $N[z]$ will cross the separators in the reversed order, and before crossing the ith separator S_i, it has to exhaust the whole z-component $C(G) - S_i$.

Algorithm 1 Algorithm for deciding whether a vertex z is an MCS end vertex of a chordal graph.

```
INPUT: A graph $G$ and an MCS ordering $\sigma$ of $G$ started with $z$.
OUTPUT: Whether $z$ can be an MCS end vertex of $G$.

1.     for $i \leftarrow 1$ to $n$ do
1.1.       $D \leftarrow$ the set of unvisited vertices with the maximum number of visited neighbors;
1.2.       visit the vertex $\arg\max_{v \in D} \sigma(v)$;
2.     if the last visited vertex is $z$ then return "yes";
       else return "no."
```

Proof of Theorem 1. Let G be a connected chordal graph. We find an MCS ordering σ of G started with z, and then use Algorithm 1. We first show its correctness: Vertex z is an MCS end-vertex of G if and only if z is the last visited vertex. The if direction is correct because the algorithm conducts MCS, and Hence we focus on the only if direction. Let $S_1, \ldots, S_k$ be the set of separators specified in Theorem 11, and let σ^+ denote the ordering returned by Algorithm 1. We show by induction that for each $1 \leq i \leq k$, vertices in all the other components of $G - S_i$ are visited before those in the same component with z.

Let T'_1 be the component of $G - S_1$ containing z. By Proposition 9 and Corollary 7, vertices in T_1 are at the beginning of σ. In each component of $G - S_1$, there is a vertex adjacent to all vertices in S_1. When the first vertex in T'_1 is being visited, it has precisely $|S_1|$ visited neighbors, i.e., S_1. By the selection of vertices in step 1, all other components have been finished. Thus, T'_1 is the last visited component of $G - S_1$.

For the inductive step, suppose that the induction hypothesis is true for all p with $1 \leq i \leq p < k$, we show it is also true for $p + 1$. For $1 < i \leq k$, let T'_i be the component of $T_{i-1} - S_i$ containing z, and let T_i be the subgraph induced by $V(T'_i) \cup S_i$. Let $v \in T_{p+1}$ be the vertex satisfying $v <_{\sigma^+} u$ for all $u \in T_{p+1} \setminus \{v\}$. Then $S_{p+1} \subseteq N(v)$ and $x <_{\sigma^+} v$ for all $x \in S_{p+1}$. Since S_{p+1} is a minimum separator of T_p, any other component of $T_p - S_{p+1}$ has a vertex adjacent to all of S_{p+1}. Such a vertex x would satisfy $v <_\sigma x$ because of Proposition 9 and Corollary 7, and then be chosen by step 1 before v. Now that all the vertices in $G - N[z]$ and the non-simplicial vertices in $N[z]$ have been visited, the only remaining vertices are true twins of z. Since $\sigma(z) = 1$, it has to be the last visited. We have proved the correctness.

We now analyze the running time. The only difference between the algorithm and the original MCS algorithm is step 1.2. We need to compare the σ-numbers of vertices in D. It needs to be done n times, and each time takes $O(n)$ time, and hence the extra time is $O(n^2)$. Together with the time for MCS itself, the total running time is $O(n^2 + m) = O(n^2)$. ◀

4 Maximum cardinality search on weakly chordal graphs

A graph G is *weakly chordal* if neither G nor its complement contains an induced cycle on five or more vertices. It is well known that all chordal graphs are weakly chordal. To prove NP-completeness of the MCS end vertex problem on weakly chordal graphs, we use a reduction from the 3-satisfiability problem (3-SAT), in which each clause comprises precisely three literals.

Given an instance $\mathcal{I}$ of 3-SAT with p variables and q literals, we construct a graph G as follows (see Fig. 3 for an example). Let the variables and clauses of $\mathcal{I}$ be denoted by x_1, x_2, …, x_p and c_1, c_2, …, c_q, respectively. For each literal, (including those that do not occur in any clause,) we introduce a vertex; let L denote this set of $2p$ literal vertices. For each literal vertex, we add edges between it and other vertices in L, with the only exception of its negation. We also introduce a set C of q clause vertices, each for a different clause; they forms an independent set. For each $\ell \in L$ and $c \in C$, we add an edge ℓc if the literal ℓ does not occur in the clause c. Therefore, each clause vertex has $2p - 3$ neighbors in L. Finally, we add seven extra vertices $a_1, a_2, u_1, u_2, b, y, z$ and edges a_1a_2, u_1u_2, yz, $\{b, z\} \times L$ and $\{a_2, u_1, u_2, y\} \times (L \cup C)$.

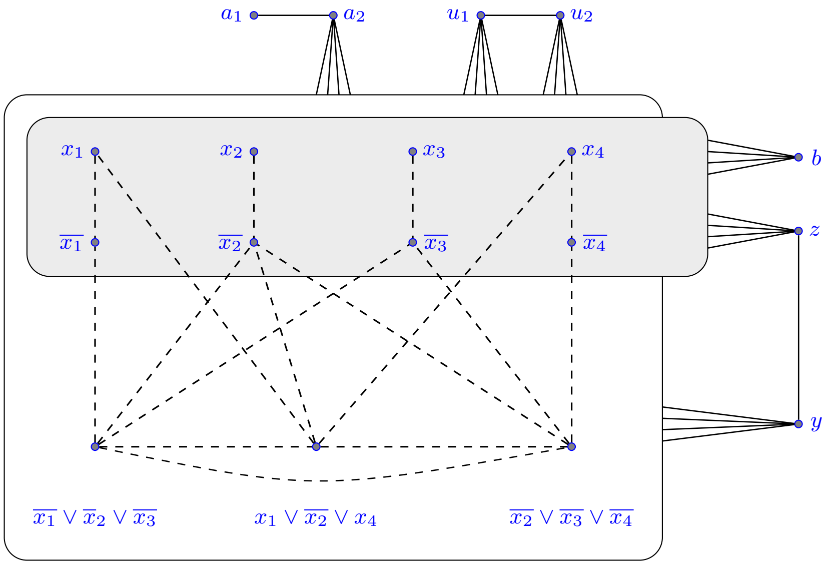
a_1
a_2
u_1
u_2
x_1
x_2
x_3
x_4
$\overline{x_1}$
$\overline{x_2}$
$\overline{x_3}$
$\overline{x_4}$
b
z
y
$\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$
$x_1 \vee \overline{x_2} \vee x_4$
$\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}$

Figure 3 Construction for NP-completeness proof of the MCS end vertex problem on weakly chordal graphs. The 3-SAT instance has four variables and three clauses, $(\overline{x_1} \vee \overline{x_2} \vee \overline{x_3})$, $(x_1 \vee \overline{x_2} \vee x_4)$, $(\overline{x_2} \vee \overline{x_3} \vee \overline{x_4})$, i.e., $p = 4$ and $q = 3$. The $2p$ literal vertices are shown in the small gray box, and the q clause vertices are in the big box. In the boxes, two vertices are nonadjacent if there is a dashed line between them, and adjacent otherwise. Vertices b and z are adjacent to all literal vertices, while vertices a_2, u_1, u_2, and y are adjacent to all literal vertices and all clause vertices. The MCS ordering $\langle a_1, a_2, x_1, \overline{x_2}, x_3, x_4, b, \overline{x_1}, x_2, \overline{x_3}, \overline{x_4}, u_1, u_2, y, c_1, c_2, c_3, z\rangle$ of G corresponds to the satisfying assignment in which all variables but x_2 are set to be true.

▶ **Proposition 12.** *The graph G constructed above is a weakly chordal graph.*

Proof. We need to show that neither G nor $\overline{G}$ contains an induced cycle on five or more vertices. We proceed as follows: We identify a vertex $v \in V(G)$ such that G contains an induced cycle on five or more vertices if and only if $G - v$ contains an induced cycle on five or more vertices, and then consider $G - v$. The following properties are straightforward:

(i) A vertex on any induced cycle on five or more vertices has degree at least two.
(ii) A simplicial vertex is not on any induced cycle on five or more vertices.
(iii) An induced cycle on five or more vertices cannot contain a pair of true twins or false twins, and when it contains one of them, this vertex can be replaced by the other.
(iv) If a vertex is on an induced cycle on five or more vertices, then it has at least two non-neighbors, and there is at least one edge among these non-neighbors.

We can reduce G to $G - \{a_1\}$ because $d(a_1) = 1$ and (i); then to $G - \{a_1, u_2\}$ because u_1 and u_2 are true twins and (iii); to $G - \{a_1, u_1, u_2\}$ because u_1 and a_2 are false twins in $G - \{a_1, u_2\}$ and (iii); to $G - \{a_1, u_1, u_2, y\}$ because the only two remaining non-neighbors of y, namely, a_2 and b, are not adjacent to each other and (iv); to $G - \{a_1, u_1, u_2, y, a_2\}$ for the same reason; to $G - \{a_1, u_1, u_2, y, a_2, b\}$ because z and b are false twins in $G - \{a_1, u_1, u_2, y, a_2\}$ and (iii); and finally to $G - \{a_1, u_1, u_2, y, a_2, b, z\}$ because the only non-neighbors of z, namely, C, are independent and (iv). The remaining graph is $G[L \cup C]$. Suppose that there is an induced cycle H on five or more vertices. It must intersect both L and C, since each vertex in L has only one non-neighbor in it, and since C is independent. Let $v \in C$ be a vertex on this cycle. Its two neighbors on H have to be from L; and since they are nonadjacent to each other, they have to be x and $\bar{x}$ for some variable x. Since both x and $\bar{x}$ are adjacent to all other vertices in L, the other ≥ 2 vertices on H have to be from C. But this is impossible because C is independent.

Now we consider $\overline{G}$. It can be reduced to $\overline{G} - \{a_1\}$ because a_1 has only one non-neighbor and (iv); then to $\overline{G} - \{a_1, u_2\}$ because u_1 and u_2 are false twins and (iii); to $\overline{G} - \{a_1, u_1, u_2\}$ because u_1 and a_2 are true twins in $\overline{G} - \{a_1, u_2\}$ and (iii); to $\overline{G} - \{a_1, u_1, u_2, y\}$ because y is simplicial in $\overline{G} - \{a_1, u_1, u_2\}$ and (ii); to $\overline{G} - \{a_1, u_1, u_2, y, b\}$ because z and b are true twins in $\overline{G} - \{a_1, u_1, u_2, y\}$ and (iii); to $\overline{G} - \{a_1, u_1, u_2, y, b, a_2\}$ because the degree of a_2 is one in $\overline{G} - \{a_1, u_1, u_2, y, b\}$ and (i); and finally to $\overline{G} - \{a_1, u_1, u_2, y, a_2, b, z\}$ because z is simplicial in $\overline{G} - \{a_1, u_1, u_2, y, b, a_2\}$ and (ii). The remaining graph is $\overline{G}[L \cup C]$. Suppose that there is an induced cycle H on five or more vertices. Since C is a clique, H contains at most two vertices from C. In other words, at least 3 vertices on H are from L, but this is impossible because each vertex in L has only one neighbor in L. Thus, G is weakly chordal. ◀

Proof of Theorem 2. It is clear that the MCS end vertex problem is in NP, and we now show that it is NP-hard. Let $\mathcal{I}$ be an instance of 3-SAT, and let G be the graph constructed from $\mathcal{I}$. We show that z is an MCS end-vertex of G if and only if $\mathcal{I}$ has a satisfying assignment.

For the if direction, suppose that $\mathcal{I}$ is satisfiable, and we give an MCS ordering σ as follows. Let us fix a satisfying assignment of $\mathcal{I}$, and let T be the set of variables that are set to be true. The starting vertex is a_1, which is followed by a_2; visited after them are $\{x \mid x \in T\} \cup \{\bar{x} \mid x \notin T\}$, (i.e., the literal vertices corresponding to true literals,) in any order. After these $p + 2$ vertices, each of y, z, u_1, u_2, b, and each of the unvisited literal vertices has p visited neighbors. On the other hand, each clause vertex has at most p visited neighbors: Each clause contains a true literal, and hence each clause vertex has at least one non-neighbor in the visited literal vertices.

Then $\sigma(b) = (p+3)$. Since b is adjacent to only literal vertices, the next vertex is one of them. On the other hand, since vertices $L \setminus T$ form a clique, they have to be visited between $p+4$ and $2p+3$, i.e., before others.

The remaining vertices are u_1, u_2, y, z, and clause vertices. Each of u_1, u_2, y, and z has $2p$ visited neighbors, while each clause vertex has only $2p-2$, because each clause is nonadjacent to three literal vertices. Let u_1, u_2, and y be visited next. After that, all the remaining vertices (z and all clause vertices) have the same number of visited neighbors, $2p+1$. There is no edge among these vertices, so they an be visited in any order. We have thus obtained an MCS ordering of G ended with z.

We now prove the only if direction. Suppose that σ is an MCS ordering of G with $\sigma(z) = n$. Since $N(z) = N(b) \cup \{y\}$, visiting y before b would force z to be visited before b; therefore, $b <_\sigma y <_\sigma z$. Likewise, $N(b) = L \subset L \cup C \subset N(y)$ and $b <_\sigma y$ demand

$$b <_\sigma c \text{ for all } c \in C. \qquad (\star)$$

Since $d(a_1) = 1$, it is easy to verify that $\{\sigma(a_1), \sigma(a_2)\} = \{1, 2\}$; otherwise, σ must end with a_1. The third vertex of σ has to be from $N(a_2)$, i.e., $L \cup C$. It cannot be from C because of $(\star)$. Therefore, $X = \{\ell \mid 3 \leq \sigma(\ell) \leq p+2\} \subset L$: (1) For each variable, one literal vertex has more visited neighbors than b, z, y, u_1, u_2; (2) clause vertices cannot be visited before b. There cannot be any variable x such that both $x, \bar{x} \in X$, because $x\bar{x} \notin E(G)$. We claim that assigning a variable x to be true if and only if $x \in X$ is a satisfying assignment for $\mathcal{I}$. Suppose for contradiction that some clause c is not satisfied by this assignment. By the construction of G, the clause vertex c is adjacent to all vertices of X. After visiting the first $p+2$ vertices, c has $p+1$ visited neighbors, ($\{a_2\} \cup X$,) while any other unvisited vertex in $V(G) \setminus C$ has at most p visited neighbors. But then $\sigma(c) = k+3$, contradicting $(\star)$. Therefore, all clauses are satisfied, and this completes the proof. ◀

5 Lexicographic depth-first search on chordal graphs

Berry et al. [3, Characterization 8.1] have given a full characterization of MNS end vertices on chordal graphs: A vertex z is an MNS end vertex if and only if it is simplicial and the minimal separators of G in $N(z)$ are totally ordered by inclusion. Since LDFS is a special case of MNS, its end vertices also have this property. We show that this condition is also sufficient for a vertex to be an LDFS end vertex.

Similar as DFS, LDFS visits a neighbor of the most recent vertex, or backtracks if all its neighbors have been visited. The difference lies on the choice when the vertex has more than one unvisited neighbors. Each unvisited vertex has a label, which is all its visited neighbors. When there are ties, it chooses a vertex with the lexicographically largest label. The following is actually a simple property of DFS.

▶ **Proposition 13.** *Let $X \subseteq V(G)$ such that $G[X]$ is connected. If an LDFS visits all vertices in $N(X)$ before the first vertex in X, then it visits vertices in X consecutively.*

▶ **Lemma 14.** *A vertex z of a chordal graph G is an LDFS end vertex if and only if it is simplicial and the minimal separators of G in $N(z)$ are totally ordered by inclusion.*

Proof. The only if direction follows from that all LDFS orderings are MNS orderings [11] and the result of Berry et al. [3]. For the if direction, suppose that $S_1, \ldots, S_k$ are the minimal separators in $N(z)$ and $S_1 \subset \cdots \subset S_k$. It is easy to see that for all $1 \leq i \leq k$, each component of $G - S_i$ not containing z is a component of $G - S_k$; let $\mathcal{C}$ denote these components. We show an LDFS ordering σ of G as follows. It starts from visiting all vertices in S_1, followed by components $C \in \mathcal{C}$ with $N(C) = S_1$, visited one by one. In the same manner, it deals with $S_2 \ldots . S_k$ in order. After that the only unvisited vertex are z and its true twins, of which it chooses z the last. We now verify that this is indeed a valid LDFS ordering. It is clear for S_1. Since vertices in each component $C \in \mathcal{C}$ are visited after $N(C)$, By Proposition 13, it suffices to show the correctness when it visits a vertex in $N(z)$ and when it visits the first vertex of a new component $C \in \mathcal{C}$. When such a decision is made, the label of an unvisited vertex is either $\emptyset$ or all visited vertices in $N(z)$, i.e., the most recently visited separator. So it is always correct to select a vertex from $N(z)$. When a vertex v in a component C is selected, the visited vertices in $N(z)$ are precisely $N(C)$, hence v does have the largest label. ◀

6 Breadth-first search on interval graphs

Interval graphs are intersection graphs of intervals on the real line. An interval graph is always chordal, and in particular, it has a clique tree that is a path [15]. Corneil et al. [10] gave a very simple linear-time algorithm for deciding whether a vertex z is an LBFS end vertex of an interval graph, which is very similar to Algorithm 1. They conducted an LBFS started from z, and then another LBFS that uses the first run to break ties. They proved that z is an LBFS end vertex if and only if it is the last of the second run. As shown in Fig. 4, however, this algorithm cannot be directly adapted to the BFS end vertex problem.

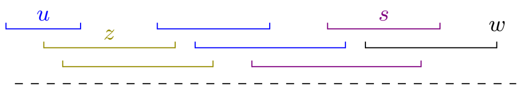

Figure 4 A BFS started from z may end with s or w, but a BFS started from w has to end with u. (Note that a BFS started from s may end with z.)

If a graph has one and only one universal vertex, then each of the other vertices is a BFS end-vertex, but not itself. If it has two or more universal vertices, then every vertex can be a BFS end-vertex. Therefore, we may focus on graphs with no universal vertex. Such an interval graph has at least three maximal cliques.

▶ **Proposition 15** ([13]). *Let G be a connected interval graph, and let $K_1, \ldots, K_p$ be a clique path of G. Let $u \in K_1$ and $w \in K_p$ be two simplicial vertices.*
(i) *Both u and w are LBFS end vertices.*
(ii) *For any vertex $v \in V(G)$, one of u and w has the largest distance to v.*

It is known that a vertex z of an interval graph G can be an LBFS end vertex if and only if it is simplicial and $N[z]$ can be one of the two ends of a clique path of G [13]. However, a BFS may satisfy neither of the two conditions. In Fig. 4, for example, vertex z is not simplicial but can be a BFS end vertex. When z is not in an end clique, it should be close to one. Actually, it should be at distance at most two to one of the u and w as specified in Proposition 15. However, a BFS end vertex might be at distance two to both u and w.

For a fixed clique path $K_1, \ldots, K_p$ of an interval graph G, we let $\mathtt{lp}(v)$ and $\mathtt{rp}(v)$ denote, respectively, the smallest and the largest number i such that $v \in K_i$. We use $\text{dist}(u, v)$ to denote the distance between u and v.

▶ **Lemma 16.** *The BFS end vertex problem can be solved in $O(n + m)$ time on interval graphs.*

Algorithm 2 Main procedure for BFS end vertex on interval graphs.

```
INPUT: A connected interval graph $G$, a clique path $K_1, \dots, K_p$ of $G$,
       simplicial vertices $u \in K_1$ and $w \in K_p$, and $z \in V(G)$.
OUTPUT: Whether there exists a BFS ordering $\sigma$ of $G$ with $\sigma(z) = n$ and $u <_\sigma w$.

1.  if $z = w$ then return "yes";
2.  if there exists a universal vertex in $V(G) \setminus \{z\}$ then return "yes";
3.  $X \leftarrow \{x \in V(G) : \mathrm{dist}(x, z) = \mathrm{dist}(x, w) \geq \mathrm{dist}(x, u)\}$;
4.  if $X = \emptyset$ then return "no";
5.  $s \leftarrow$ any vertex in $\arg\min_{v \in X} \mathtt{lp}(v)$;
6.  if $\mathtt{rp}(z) < \mathtt{lp}(s)$ then return "no";
7.  if $s = u$ then return "yes";
8.  for each vertex $v \in N(s)$ at distance $\mathrm{dist}(s, u) - 1$ to $u$ do
        if $\mathrm{dist}(v, z) > \mathrm{dist}(v, u)$ then return "yes";
9.  return "no."
```

Proof. Let G be an interval graph; we may assume without loss of generality that G is connected. We use the algorithm of Corneil et al. [13] to build a clique path for G, and take simplicial vertices v_1, v_2 from the first and last cliques of the clique path. We call the procedure described in Algorithm 2 twice, first with $u = v_1, w = v_2$; in the second call, we reverse the clique path, and use $u = v_2, w = v_1$. Suppose that the procedure is correct, then vertex z is a BFS end vertex if and only if at least one of the two calls returns yes. In the rest we prove the correctness of the procedure and analyze its running time.

We start from characterizing the first vertex s of a BFS ordering σ with $\sigma(z) = n$ and $u <_\sigma w$, if one exists. Since $u <_\sigma w <_\sigma z$, we must have $\mathrm{dist}(s, u) \leq \mathrm{dist}(s, w) \leq \mathrm{dist}(s, z)$. On the other hand, Proposition 15 implies $\mathrm{dist}(s, z) \leq \max\{\mathrm{dist}(s, u), \mathrm{dist}(s, w)\} = \mathrm{dist}(s, w)$. Therefore, a desired BFS ordering σ, if it exists, must start from a vertex s satisfying

$$\mathrm{dist}(s, z) = \mathrm{dist}(s, w) \geq \mathrm{dist}(s, u). \tag{†}$$

We argue that at least one of the following is true for z:

- on any shortest s-u path, z is adjacent to the 2nd to last vertex but no vertex before it.
- on any shortest s-w path, z is adjacent to the 2nd to last vertex but no vertex before it.

Let P_u be any s-u path and P_w any s-w path. Since they together form a u-w path that visits all the maximal cliques of G, vertex z is adjacent to at least one of these two paths. If z is adjacent to a vertex on P_u, then it has to be the last two; otherwise $\mathrm{dist}(s, z) < \mathrm{dist}(s, u)$. Since u is simplicial, z is adjacent to its neighbor on the path if $zu \in E(G)$. Therefore, z is always adjacent to the second to last vertex on this path. The same argument applies if z is adjacent to P_w.

The correctness of step 1 follows from Proposition 15. For step 2, note that if $v \neq z$ is a universal vertex, then $\langle v, u, w, \dots, z \rangle$ is such a BFS ordering. Steps 3 and 4 are justified by (†). When the algorithm reaches step 5, X is not empty, and hence s is well defined. Let $q = \mathrm{dist}(s, z) = \mathrm{dist}(s, w)$. Note that $q \geq 2$ because s is not universal. Hence, $z, w \notin N(s)$.

We show the correctness of step 6 by contradiction. Suppose that $\mathtt{rp}(z) < \mathtt{lp}(s)$ but there exists a BFS ordering σ with $\sigma(z) = n$ and $u <_\sigma w$. Let s' be the first vertex of σ. Since $s' \in X$, the selection of s implies $\mathtt{lp}(s) \leq \mathtt{lp}(s')$. Then $\mathtt{rp}(u) = 1 \leq \mathtt{rp}(z) < \mathtt{lp}(s) \leq \mathtt{lp}(s')$, therefore, $\mathrm{dist}(s', u) \geq 2$. In this case, on any shortest s'-u path, z is adjacent to the second to last vertex but no vertex before it. Hence, $\mathrm{dist}(s', z) = \mathrm{dist}(s', u) = \mathrm{dist}(s', w)$; let it be q'. Since $u <_\sigma w$, there must be some neighbor u'' of u at distance $q' - 1$ to s' visited before neighbors of w. The vertex u'' cannot be universal, hence nonadjacent to w. But u''

is adjacent to z, which implies $z <_\sigma w$, a contradiction. Therefore, step 6 is correct, which means $\texttt{rp}(s) < \texttt{lp}(z)$ because s and z are not adjacent. Let $s = w_0, w_1, \ldots, w_{q-1}, w_q = w$ be a shortest s-w path. Note that $w_{q-1} \in N(z)$.

For step 7, it suffices to give the following BFS ordering, which starts with $s = u$. Of all vertices at distance i to s, $1 \leq i \leq q$, the first visited vertex is w_i. Note that every vertex is adjacent to $w_1, \ldots, w_{q-1}$. From $\texttt{rp}(w_{q-1}) = p$ it can be inferred that all vertices at distance q to s are adjacent to w_{q-1}. Since w_{q-1} is the first visited vertex at level $q-1$, vertices at distance q to s can be visited in any order. Therefore, we can have a BFS ordering σ of G with $u <_\sigma w$ and $\sigma(z) = n$.

We now consider step 8, for which we show that there exists a BFS ordering σ with $\sigma(s) = 1$, $\sigma(v) = 2$, $\sigma(z) = n$, and $u <_\sigma w$. Note that $\text{dist}(w_1, z) = \text{dist}(w_1, w) = q - 1$. Therefore $v \neq w_1$; otherwise step 5 should have chosen v because $\texttt{lp}(v) < \texttt{lp}(s)$. For $1 \leq i \leq q-1$, vertex w_i is always visited in the earliest possible time; in particular, $\sigma(w_1) = 3$. Since v is on a shortest s-u path, u is a descendant of v in the BFS tree generated by σ. On the other hand, since both $\text{dist}(v, z)$ and $\text{dist}(v, w)$ are larger than $\text{dist}(v, u)$, either vertices z and w are not descendants of v, or they are at a lower level than u. In either case, we have $u <_\sigma w$. When w_q is visited, all the unvisited vertices are at distance q to s and adjacent to w_{q-1}. Thus, we can have $\sigma(z) = n$.

We are now at the last step. Note that the algorithm can reach here only when $\text{dist}(s, z) = \text{dist}(s, w) = \text{dist}(s, u)$: The condition of step 8 must be true if $\text{dist}(s, u) < q$. Suppose for contradiction that there exists a BFS ordering σ with $\sigma(z) = n$ and $u <_\sigma w$ but no vertex satisfies the condition in step 8. Let s' be the starting vertex of σ. Since $s' \in X$ and by the selection of s, we have $\texttt{lp}(s') \geq \texttt{lp}(s)$, which implies $\text{dist}(s', u) \geq \text{dist}(s, u)$. Note that s' is adjacent to any s-w path, and hence its distance to w is at most $q + 1$. In summary,

$$q = \text{dist}(s, u) \leq \text{dist}(s', u) \leq \text{dist}(s', w) \leq q + 1.$$

Let Y denote all vertices at distance $q - 1$ to u, and let Z denote all vertices at distance $q - 1$ to w. Note that Y is disjoint from Z: A vertex in $v \in Y \cap Z$ would be adjacent to s, and have the same distance to u, w, and z, but then it contradicts the selection of s because $\texttt{lp}(v) < \texttt{lp}(s)$. Since no vertex in Y satisfies the condition of step 8, $\text{dist}(v, z) = \text{dist}(v, u)$ for all $v \in Y \cap N(s)$.

If $\text{dist}(s', u) = \text{dist}(s', z) = \text{dist}(s', w) = q$, then to have $u <_\sigma w$, one vertex in $Y \cap N(s)$ must be visited before Z. But this would force z to be visited before w, because z is at distance $q - 1$ to all vertices in $Y \cap N(s)$. Now that $\text{dist}(s', w) = q + 1$, if $\text{dist}(s', u) = q$, then at least one vertex $v \in Y$ is adjacent to s'; it is in $N(s)$ because $\texttt{lp}(s) \leq \texttt{lp}(s')$. But then $\text{dist}(s', z) \leq 1 + \text{dist}(v, z) = 1 + q - 1 = q < \text{dist}(s', w)$. Therefore, $\text{dist}(s', u) = q + 1$ as well. Each vertex in $Y \cup Z$ has distance at least two to s'. Of vertices at distance two to s', one vertex in $Y \cap N(s)$ must be visited before Z, but then we have the same contradiction as in the first case of this paragraph. Therefore, step 9 is also correct and this concludes the proof of correctness.

We now analyze the running of the algorithm. Steps 1 and 2 can be easily checked in $O(n + m)$ time. For step 3, it suffices to calculate the distances between z, w, u and all other vertices; this can be done by visiting the maximal cliques one by one. Steps 4–7 can be done in $O(n)$ time. Step 8 can be checked in $O(n)$ time: We have already calculated the distance between z and v. Therefore, the total running time is $O(n + m)$. ◀

7 Graph searches on general graphs

We now describe an algorithm for deciding whether a vertex z of a general graph is an MCS end vertex. For each subset $X \subseteq V(G) \setminus \{z\}$, we define $f(X)$ to be true if there exists an MCS visiting X before others, and false otherwise. The question whether z can be an end vertex is then simply the value of $f(V(G) \setminus \{z\})$. For a set X with $f(X)$ is true and $v \notin X$, let $g(X, v)$ indicate whether there exists a search ordering that visits v after X and before others. We have $f(X) = \bigvee_{v \in X} \big(f(X \setminus \{v\}) \wedge g(X \setminus \{v\}, v)\big)$. For MCS, $g(X, v)$ can be calculated in linear time, and thus we have a simple $O(2^n n^{O(1)})$-time algorithm similar to the classic Held–Karp algorithm [18].

Let us consider then BFS. We may fix the starting vertex s, which can be found by enumerating all the other $n - 1$ vertices. Let $\ell = \max_{v \in V(G)} \operatorname{dist}(s, v)$, and for $1 \leq i \leq \ell$, let L_i denote the set of vertices at distance i to s. Suppose that there is a BFS ordering σ started with s and ended with z, then $z \in L_\ell$. Clearly, vertices in $L_{\ell-1}$ are visited after those in $L_{\ell-2}$ and before L_ℓ. Let u be the first visited vertex in $L_{\ell-1}$ that is adjacent to z, and let X be those vertices in $L_{\ell-1}$ visited before u. Since z is the last vertex, all vertices in $L_\ell \setminus N(X)$ must be adjacent to u. We do not need any constraint on the order of vertices in $L_{\ell-1} \setminus (X \cup \{u\})$ being visited. Therefore, the information we need at level $\ell - 1$ are the set X and the vertex u. We can generalize this observation to give a recursive formula for the BFS end vertex problem. For lack of space, the proofs in this section are left for the full version of the paper.

▶ **Lemma 17.** *There is a $2^n \cdot n^{O(1)}$-time algorithm for solving the* BFS *end vertex problem.*

In the last we consider DFS. Recall that a DFS sets two timestamps for a vertex v, first when it is visited, and second when it is *finished*, i.e., when all its neighbors have been examined and the search backtracks to the vertex that discovered v (or terminates when v is the source vertex). Note that when a vertex is finished, all its neighbors have been visited, and all but one of them have been finished. In particular, when the last vertex is visited, no vertex in its neighborhood has been finished. At any moment, the set of vertices that have been visited but not finished form a path in the depth-first tree. Suppose that z is the end vertex of a DFS ordering σ of G. If v is the earliest visited neighbor of z, then all the vertices after v are descendants of v in the depth-first tree.

The following simple property of DFS is stronger than Proposition 13. In a DFS ordering σ, if the set of vertices after v, i.e., $\{u : v <_\sigma u\}$, and v induce a connected subgraph, then their visiting order is irrelevant to vertices visited before v.

▶ **Proposition 18.** *Let σ be a* DFS *ordering of a graph G, and let X be the set of last visited $|X|$ vertices in σ. The sub-ordering $\sigma|_X$ is a* DFS *ordering of $G[X]$. Moreover, if $G[X]$ is connected, then σ remains a* DFS *ordering of G after replacing $\sigma|_X$ with any* DFS *ordering of $G[X]$ that starts with $\arg\min_{v \in X} \sigma(v)$.*

▶ **Lemma 19.** *There is a $2^n \cdot n^{O(1)}$-time algorithm for solving the* DFS *end vertex problem.*

References

1 Jesse Beisegel, Carolin Denkert, Ekkehard Köhler, Matjaz Krnc, Nevena Pivac, Robert Scheffler, and Martin Strehler. On the End-Vertex Problem of Graph Searches. *Discrete Mathematics & Theoretical Computer Science*, 21(1), 2019. URL: `http://dmtcs.episciences.org/5572`.

2 Philip A. Bernstein and Nathan Goodman. Power of Natural Semijoins. *SIAM Journal on Computing*, 10(4):751–771, 1981. `doi:10.1137/0210059`.

3 Anne Berry, Jean R. S. Blair, Jean Paul Bordat, and Geneviève Simonet. Graph Extremities Defined by Search Algorithms. *Algorithms*, 3(2):100–124, 2010. `doi:10.3390/a3020100`.

4 Anne Berry and Jean Paul Bordat. Separability Generalizes Dirac's Theorem. *Discrete Applied Mathematics*, 84(1-3):43–53, 1998. `doi:10.1016/S0166-218X(98)00005-5`.

5 Jean R. S. Blair and Barry W. Peyton. An introduction to chordal graphs and clique trees. In J. A. George, J. R. Gilbert, and J. W.-H. Liu, editors, *Graph Theory and Sparse Matrix Computation*, volume 56 of *IMA*, pages 1–29. Springer-Verlag, 1993.

6 Pierre Charbit, Michel Habib, and Antoine Mamcarz. Influence of the tie-break rule on the end-vertex problem. *Discrete Mathematics & Theoretical Computer Science*, 16(2):57–72, 2014. URL: `http://dmtcs.episciences.org/2081`.

7 Derek G. Corneil. A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. *Discrete Applied Mathematics*, 138(3):371–379, 2004. `doi:10.1016/j.dam.2003.07.001`.

8 Derek G. Corneil. Lexicographic Breadth First Search - A Survey. In Juraj Hromkovic, Manfred Nagl, and Bernhard Westfechtel, editors, *Graph-Theoretic Concepts in Computer Science (WG)*, volume 3353 of *LNCS*, pages 1–19. Springer, 2004. `doi:10.1007/978-3-540-30559-0_1`.

9 Derek G. Corneil, Barnaby Dalton, and Michel Habib. LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. *SIAM Journal on Computing*, 42(3):792–807, 2013. `doi:10.1137/11083856X`.

10 Derek G. Corneil, Ekkehard Köhler, and Jean-Marc Lanlignel. On end-vertices of Lexicographic Breadth First Searches. *Discrete Applied Mathematics*, 158(5):434–443, 2010. `doi:10.1016/j.dam.2009.10.001`.

11 Derek G. Corneil and Richard Krueger. A Unified View of Graph Searching. *SIAM Journal on Discrete Mathematics*, 22(4):1259–1276, 2008. `doi:10.1137/050623498`.

12 Derek G. Corneil, Stephan Olariu, and Lorna Stewart. Linear Time Algorithms for Dominating Pairs in Asteroidal Triple-free Graphs. *SIAM Journal on Computing*, 28(4):1284–1297, 1999. A preliminary version appeared in ICALP 1995. `doi:10.1137/S0097539795282377`.

13 Derek G. Corneil, Stephan Olariu, and Lorna Stewart. The LBFS Structure and Recognition of Interval Graphs. *SIAM Journal on Discrete Mathematics*, 23(4):1905–1953, 2009. `doi:10.1137/S0895480100373455`.

14 Gabriel A. Dirac. On rigid circuit graphs. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 25(1):71–76, 1961. `doi:10.1007/BF02992776`.

15 Delbert R. Fulkerson and Oliver A. Gross. Incidence matrices and interval graphs. *Pacific Journal of Mathematics*, 15(3):835–855, 1965. `doi:10.2140/pjm.1965.15.835`.

16 Philippe Galinier, Michel Habib, and Christophe Paul. Chordal Graphs and Their Clique Graphs. In Manfred Nagl, editor, *Graph-Theoretic Concepts in Computer Science (WG)*, volume 1017 of *LNCS*, pages 358–371. Springer, 1995. `doi:10.1007/3-540-60618-1_88`.

17 Michel Habib, Ross M. McConnell, Christophe Paul, and Laurent Viennot. Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. *Theoretical Computer Science*, 234(1-2):59–84, 2000. `doi:10.1016/S0304-3975(97)00241-7`.

18 Michael Held and Richard M. Karp. A Dynamic Programming Approach to Sequencing Problems. *Journal of the Society for Industrial and Applied Mathematics*, 10(1):196–210, 1962. `doi:10.1137/0110015`.

19 John E. Hopcroft and Robert Endre Tarjan. Efficient Planarity Testing. *Journal of the ACM*, 21(4):549–568, 1974. `doi:10.1145/321850.321852`.

20 Russell Impagliazzo and Ramamohan Paturi. On the Complexity of k-SAT. *Journal of Computer and System Sciences*, 62(2):367–375, 2001. A preliminary version appeared in CCC 1999. `doi:10.1006/jcss.2000.1727`.

21 Donald J. Rose, Robert Endre Tarjan, and George S. Lueker. Algorithmic Aspects of Vertex Elimination on Graphs. *SIAM Journal on Computing*, 5(2):266–283, 1976. A preliminary version appeared in STOC 1975. `doi:10.1137/0205021`.

22 Ravi Sethi. Scheduling Graphs on Two Processors. *SIAM Journal on Computing*, 5(1):73–82, 1976. doi:10.1137/0205005.

23 Douglas R. Shier. Some aspects of perfect elimination orderings in chordal graphs. *Discrete Applied Mathematics*, 7(3):325–331, 1984. doi:10.1016/0166-218X(84)90008-8.

24 Klaus Simon. A New Simple Linear Algorithm to Recognize Interval Graphs. In *Computational Geometry - Methods, Algorithms and Applications, International Workshop on Computational Geometry CG'91, Bern, Switzerland, March 21-22, 1991*, pages 289–308, 1991. doi:10.1007/3-540-54891-2_22.

25 Robert Endre Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. A preliminary version appeared in SWAT (FOCS) 1971. doi:10.1137/0201010.

26 Robert Endre Tarjan and Mihalis Yannakakis. Simple Linear-Time Algorithms to Test Chordality of Graphs, Test Acyclicity of Hypergraphs, and Selectively Reduce Acyclic Hypergraphs. *SIAM Journal on Computing*, 13(3):566–579, 1984. With Addendum in the same journal, 14(1):254-255, 1985. doi:10.1137/0213035.

27 Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Automata, Languages and Programming (ICALP)*, volume 5125 of *LNCS*, pages 634–645. Springer-Verlag, 2008. doi:10.1007/978-3-540-70575-8_52.