Algorithmica (2022) 84:2642–2666
https://doi.org/10.1007/s00453-022-00981-5

Graph Searches and Their End Vertices

Guozhen Rong[1] · Yixin Cao[1,2] · Jianxin Wang[1] · Zhifeng Wang[1]

Received: 28 December 2020 / Accepted: 28 April 2022 / Published online: 21 May 2022

Abstract
For a graph search algorithm, the end vertex problem is concerned with which vertices of a graph can be the last visited by this algorithm. We characterize all maximum cardinality searches on chordal graphs and derive from this characterization a polynomial-time algorithm for the end vertex problem of maximum cardinality searches on chordal graphs. 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.

Keywords End vertex of a graph search · Maximum cardinality search · Chordal graph · Weakly chordal graph

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. The use of BFS and DFS can be found, sometimes implicitly, in most graph algorithms. In general, a graph search algorithm is a systematic exploration of a graph, and its core lies in the strategy of choosing the next vertex to visit. Mostly greedy, graph search algorithms are elementary but sometimes have magical powers. For example, DFS has played a

A preliminary version of this paper appears in the Proceedings of the 30th International Symposium on Algorithms and Computation (ISAAC 2019).

G. Rong, Y. Cao, J. Wang, Z. Wang: Supported by National Natural Science Foundation of China (61828205, 61672536), Hunan Provincial Key Lab on Bioinformatics, and Hunan Provincial Science and Technology Program (2018WK4001).
Y. Cao: Supported in part by the Hong Kong Research Grants Council (RGC) under Grant 15201317 and the National Natural Science Foundation of China (NSFC) under Grant 61972330.

✉ Yixin Cao
yixin.cao@polyu.edu.hk

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

2 Department of Computing, Hong Kong Polytechnic University, Hong Kong, China

significant role in Tarjan's award-winning work, testing planarity [19] and finding strongly connected components [29].

Two other search algorithms, lexicographic breadth-first search (LBFS) [25] and maximum cardinality search (MCS) [30], 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), which exist if and only if the graph is chordal. LBFS and MCS have other important applications. LBFS is used in scheduling [26] and is the base of the recent linear-time algorithm for computing modular decompositions of graphs [31]. Tarjan and Yannakakis [30] also used MCS in testing acyclic hypergraphs. Nagamochi and Ibaraki [23] rediscovered MCS and applied it to compute minimum cuts of a graph and to find forest decompositions; see also [24].

Simon [28] proposed an interesting way of using LBFS. It conducts LBFS more than once, and each new run uses previous runs to break ties; in particular, except for 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 surprisingly simple and elegant recognition algorithms for (unit) interval graphs [6, 9, 14, 22]. See the survey of Corneil [8] for more algorithms using multiple runs of LBFS. Some of these results have a flavor of "ad-hoc" because we have not fully understood the execution process of LBFS, and this is precisely the motivation of this line of research.

The outputs of BFS and DFS are usually rooted spanning forests of the graph, while LBFS and MCS produce orderings of the vertex set. To have a unified view of them, Corneil et al. [12] focused on ordering the vertices being visited and conducted a systematic study. Motivated by this study, they proposed the lexicographic version of DFS, lexicographic depth-first search (LDFS), which has proved to be very useful [10], and a very general search paradigm, maximal 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 give us the answer to the ostensibly naïve question: which vertex can be the last of such an ordering? Corneil et al. [11] called it the *end vertex problem* and studied it from combinatorial and algorithmic perspectives. Apart from being a natural starting point for 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 [4, 25, 27, 30] (a vertex v is simplicial if $N[v]$ is a clique). 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 [9, 14]. Important properties and use of end vertices of other graph searches can be found in [10, 13, 18].

One may find it surprising, but the end vertex problem is NP-hard for all the six mentioned graph search algorithms [1, 7, 12]. The study has thus been focused on chordal graphs and their 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 disappointing.) Moreover, most applications of LBFS and LDFS are on related graph classes. The most natural superclass of chordal graphs is arguably

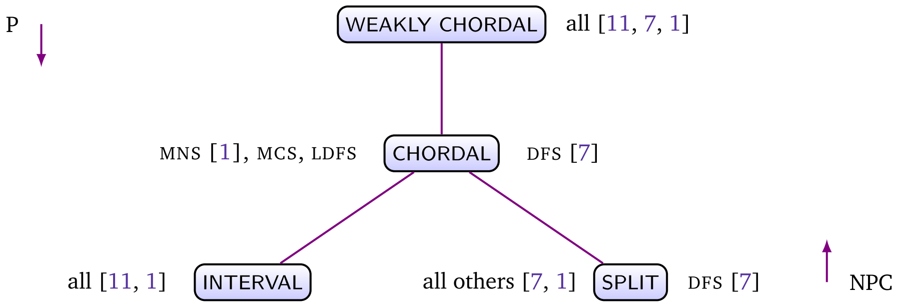


Fig. 1 A summary of the known complexity of the end vertex problem for the six graph search algorithms. For each graph class, the end vertex problem of graph searches listed to its left can be solved in polynomial time on this class, while those to the right are NP-hard. The complexity of the BFS end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

the weakly chordal graphs, and two important subclasses are interval graphs and split graphs.[1] It has been known that on weakly chordal graphs, the end vertex problems for all but MCS are NP-complete, while on chordal graphs, the problem is NP-complete only for DFS [1, 7, 12]. 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 search algorithms, 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. [17] have shown that MCS orderings 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 completely visited maximal clique. Therefore, for a minimum separator S of G (i.e., a minimum vertex set such that $G - S$ is not connected), 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), we can make a similar conclusion on the subgraph induced by it. In other words, this property on minimum separators holds recursively. For an MCS end vertex z, which is necessarily simplicial, we can find a sequence of increasing separators. The first is a minimum separator of G and the last comprises all the non-simplicial vertices in $N(z)$. An MCS ending with z has to "cross" these separators *in order*, and for each S of these separators, visit the component of $G - S$ containing z as the last one. We have thus a full understanding of all MCS orderings of a chordal graph. As it turns out, our result is easier to be presented in the so-called clique graph of G [5, 17]. It enables us to show that if we run MCS twice, the first starting from some z and the second using the output of the first run to break ties, then the second run ends with z if and only if z is an MCS end vertex.

[1] A graph is a *split graph* if its vertex set can be partitioned into a clique and an independent set. Definitions of weakly chordal graphs and interval graphs are deferred to Sects. 4 and 6, respectively.

Theorem 1.1 *Let G be a chordal graph. A vertex z is an* MCS *end vertex of G if and only if for any* MCS *ordering σ of G with $\sigma(z) = 1$, the ordering* $\text{MCS}^+(G, \sigma)$ *ends with z.*

As usual, we use n to denote the order of the input graph. Theorem 1.1 readily implies a simple $O(n^2)$-time algorithm for the MCS end vertex problem on chordal graphs. We complement this result by showing that the MCS end vertex problem becomes NP-complete already on weakly chordal graphs; the proof is inspired by and adapted from Beisegel et al. [1].

Theorem 1.2 *The* MCS *end vertex problem is NP-complete on weakly chordal graphs.*

We then turn to LDFS on chordal graphs and BFS on interval graphs. Note that every LDFS ordering of a graph G is also an MNS ordering of G [12]. Hence, every LDFS end vertex of G is also an MNS end vertex of G. Berry et al. [3] have presented a characterization of MNS end vertices of chordal graphs. It is surprising that the very same characterization is also true for LDFS end vertices. In other words, 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. Corneil et al. [11] have presented an elegant algorithm for solving the LBFS end vertex problem on interval graphs. However, the situation for BFS is more complicated: a BFS end vertex of an interval graph may not be simplicial. The intuition behind our algorithm is as follows. An interval graph G can be represented by a clique path $\mathcal{K}$ [16]. In each of its two ends, we can find at least one simplicial vertex; let them be u and w. For any vertex v in G, at least one of u and w has the largest distance to v. This observation helps us demarcate the range of the starting vertices, if we want a specific vertex z to be the last. Roughly speaking, if z is close to the right end of $\mathcal{K}$, then we can start the search from the left end. However, z might be "in the middle" of the clique path, which can only happen when G has a small diameter. In this case, we try both directions.

Theorem 1.3 *There are linear-time algorithms for solving the* LDFS *end vertex problem on chordal graphs and 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 problems can be solved in linear time on split graphs, we conjecture that they can also 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.

Before concluding this section, let us have a quick remark on the end vertex problems on general graphs. By enumerating all possible orderings, a trivial algorithm can find all end vertices of any graph search algorithm 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 SAT or 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. Therefore, it is straightforward to use dynamic programming to solve the MCS end vertex problem in $2^n \cdot n^{O(1)}$ time. Kratsch et al. [21] have shown that a similar approach actually works for the BFS and DFS end vertex problems. Also studied are the end vertex problems on bipartite graphs; see [32] for the latest results.

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 $X \subseteq V(G)$, we denote by $G[X]$ the subgraph of G induced by X, and by $G - X$ the subgraph $G[V(G)\backslash X]$. The two ends of an edge are *neighbors* of each other, and they are *adjacent*. The *neighborhood* of a vertex $v \in V(G)$, denoted by $N(v)$, comprises all the neighbors of v. The *closed neighborhood* of v is $N[v] = N(v) \cup \{v\}$. A *clique* is a set of pairwise adjacent vertices, and a clique is maximal if it is not a proper subset of any other clique. A vertex v is *simplicial* if $N[v]$ is a clique. 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 set S of vertices is a u–v separator if u and v are not in S and there is no u–v path in $G - S$, and a u–v separator is *minimal* if no proper subset of S is a u–v separator. We say that S is 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 $\sigma(z) = n$. When applying a graph search algorithm S to a graph G, the order of visiting the vertices is called an S-*ordering* of G. Given a graph G and a vertex $z \in V(G)$, the *end vertex problem* for a graph search algorithm 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* [16]. As shown in Fig. 2, the greedy strategy of MCS is

```
1.    for each $v \in V(G)$ do
1.1.      label($v$) $\leftarrow 0$;
2.    for $i \leftarrow 1$ to $n$ do
2.1.      $S \leftarrow$ the set of unvisited vertices with the most visited neighbors;
2.2.      $v \leftarrow$ any vertex in $S$;
2.3.      $\sigma(v) \leftarrow i$;
2.4.      for each unvisited vertex $x \in N(v)$ do
              label($x$) $\leftarrow$ label($x$) + 1;
3.    return $\sigma$.
```

Fig. 2 The MCS algorithm

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 [30].

To avoid unnecessary digressions, all the input graphs in this paper are assumed to be connected. One can easily lifted all the results to graphs with more than one component.

3 Maximum Cardinality Search on Chordal Graphs

Another important characterization of chordal graphs is through their maximal cliques. A graph G is chordal if and only if there exists a tree T whose nodes are the maximal cliques of G such that for each vertex $v \in V(G)$, the maximal cliques containing v induce a subtree of T; such a tree is called a *clique tree* of G [15]. A chordal graph G has at most n maximal cliques [15], and for any pair of adjacent K_i and K_j on a clique tree of G, the intersection $K_i \cap K_j$ is a minimal separator of G.

Out of a chordal graph G, we can define a *clique graph* $C(G)$ as follows. The clique graph $C(G)$ has ℓ vertices, where ℓ is the number of maximal cliques of G, and each vertex of $C(G)$ is labeled by a distinct maximal clique of G. To simplify the presentation, we will refer to vertices of $C(G)$ as cliques (of G); 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, $1 \leqslant i, j \leqslant \ell$, 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 $|K_i \cap K_j|$. See Fig. 3 for an example. It is known that a tree on the maximal cliques G is a clique tree of G if and only if it is a maximum spanning tree of $C(G)$ [2, 5, 17], i.e., a spanning tree of $C(G)$ with the maximum total edge weights.

Proposition 3.1 [17] *Let G be a chordal graph and $C(G)$ the clique graph of G. A vertex set $S \subseteq V(G)$ is a minimal separator of G if and only if it is the label of some edge of $C(G)$.*

One can use Prim's algorithm to find a maximum spanning tree of $C(G)$. (Although proposed for the purpose of 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

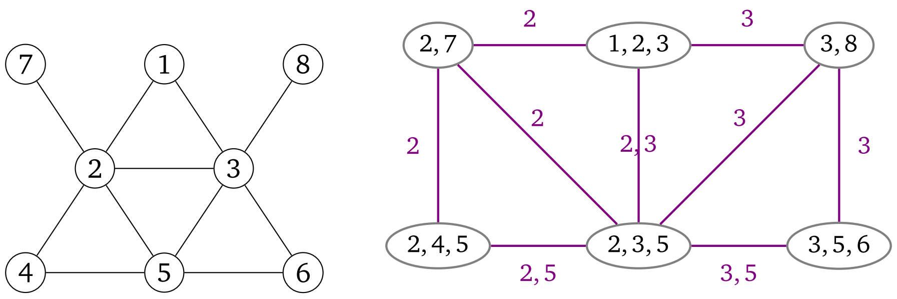


Fig. 3 A chordal graph and its clique graph

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* as the order in which maximal cliques of G are visited by Prim's algorithm, applied to $C(G)$.

Our first observation is that at any moment, the set of maximal cliques visited by Prim's algorithm is the clique graph of the subgraph G induced by those visited maximal cliques. (This is not always true for a connected induced subgraph of $C(G)$; e.g., consider the five maximal cliques except $\{2, 3, 5\}$ in Fig. 3.) Recall that Prim's algorithm always maintains a tree of visited cliques, and this tree is a subtree of the final output. Another simple fact we need for the following proof is that between any two vertices in a tree, there is a unique path connecting them.

Lemma 3.2 *Let G be a connected chordal graph with ℓ maximal cliques. For any Prim ordering π of $C(G)$ and any t with $1 \leqslant t \leqslant \ell$, the subgraph C' of $C(G)$ induced by the first t cliques in π is the clique graph of the subgraph of G induced by all vertices in those cliques.*

Proof Without loss of generality, let $\pi = \langle K_1, \ldots, K_\ell \rangle$, and let $U = \bigcup_{i=1}^{t} K_i$. Note that Prim's algorithm maintains a subtree of $C(G)$. We use T to denote the clique tree of G created by the Prim ordering π, and T' the subtree of T on the first t cliques in π.

We argue that for every clique K of $G[U]$, there must be $1 \leqslant j \leqslant t$ such that $K \subseteq K_j$. Suppose for contradiction that $K \nsubseteq K_j$ for all $j = 1, \ldots, t$. We can find K_p such that $K \cap K_p$ is maximal among $\{K \cap K_j \mid 1 \leqslant j \leqslant t\}$. There are an $x \in K \backslash K_p$ and a K_q with $1 \leqslant q \leqslant t$ such that $x \in K_q$. Since T is a clique tree, there exists K_r such that $K \subseteq K_r$. Between any two nodes in a tree there is a unique path. We consider the common clique K' on the three paths of T connecting the three cliques K_p, K_q, and K_r; note that K' can be K_p or K_q. By the definition of clique trees, K' contains all the vertices in $K_p \cap K_r$ and all the vertices in $K_q \cap K_r$. From $K \subseteq K_r$ it can be inferred that

$$K \cap K_p = K \cap K_r \cap K_p \subseteq K \cap K'.$$

For the same reason, $K \cap K_q \subseteq K \cap K'$. Thus, we have $K \cap K_p \subset K \cap K'$ because $x \in K \cap K_q \subseteq K \cap K'$ but $x \notin K_p$. Since K' is on the path of T' connecting K_p and K_q, it is one of $K_1, \ldots, K_t$ (note that since T' is a subtree of T, the path connecting K_p and K_q in T' is also the unique path connecting them in T). We have thus a contradiction: $K \cap K_p$ is not maximal among $\{K \cap K_j \mid 1 \leqslant j \leqslant t\}$.

It remains to show that the edges of the clique subgraph of $G[U]$ are precisely $E(C')$. First, suppose that $K_i K_j$ is an edge in C'. For any $x \in K_i \backslash K_j$ and $y \in K_j \backslash K_i$, the set $K_i \cap K_j$ is an x–y separator in G, and hence also an x–y separator in $G[U]$. This separator is obviously minimal because $K_i, K_j \subseteq U$ and every vertex in $K_i \cap K_j$ is a common neighbor of x and y. By definition, $K_i K_j$ is an edge of the clique graph of $G[U]$. For the other direction, let $K_i K_j$ be an edge of the clique graph of $G[U]$. We have seen that T' is a clique tree of $G[U]$, and thus there is a path P in T' connecting K_i and K_j. It suffices to prove that one of the separators on the path P is $K_i \cap K_j$, which means that $K_i \cap K_j$ is a minimal x–y separator for $x \in K_i \backslash K_j$ and $y \in K_j \backslash K_i$

in $G[U]$. Accordingly, $K_i K_j$ is an edge of C'. By the definition of clique trees, the label of every edge on P contains $K_i \cap K_j$. Suppose for contradiction that the label of every edge on P is a proper superset of $K_i \cap K_j$. For each edge on P, we can pick a common vertex of its two ends that is not in $K_i \cap K_j$. Note that any consecutive two of these chosen vertices are either equal or adjacent, and thus they form an x–y path with x and y in $G[U] - K_i \cap K_j$. We have thus a contradiction to that $K_i \cap K_j$ is an x–y separator in $G[U]$. This concludes the proof. □

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 v, respectively. 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 $1 \leqslant i \leqslant \ell$, 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 one direction explicitly. For the sake of completeness, we give a proof here.

Lemma 3.3 *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 $1 \leqslant i \leqslant \ell$; note that $\{K'_1, K'_2, \ldots, K'_\ell\}$ is a partition of $V(G)$. We show by induction that for each $1 \leqslant i \leqslant n$, 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. By Lemma 3.2, there is a j with $1 \leqslant j < t$ such that $N(v_q) \cap A \subseteq K_j$. 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 $\langle K_1, \ldots, K_\ell \rangle$ is a Prim ordering of $C(G)$, when K_t is visited, the edge that was chosen by Prim's algorithm is an edge of a clique tree, and hence has label $K_t \cap A = N(v_{p+1}) \cap A$. Therefore, we have $|N(v_{p+1}) \cap A| \geqslant |N(v_q) \cap A|$ for all $q > p$. Noting that v_{p+1} is adjacent to all other vertices in K_t, we can conclude that v_{p+1} has the maximum number of neighbors in $\{v_1, \ldots, v_p\}$. This completes the proof. □

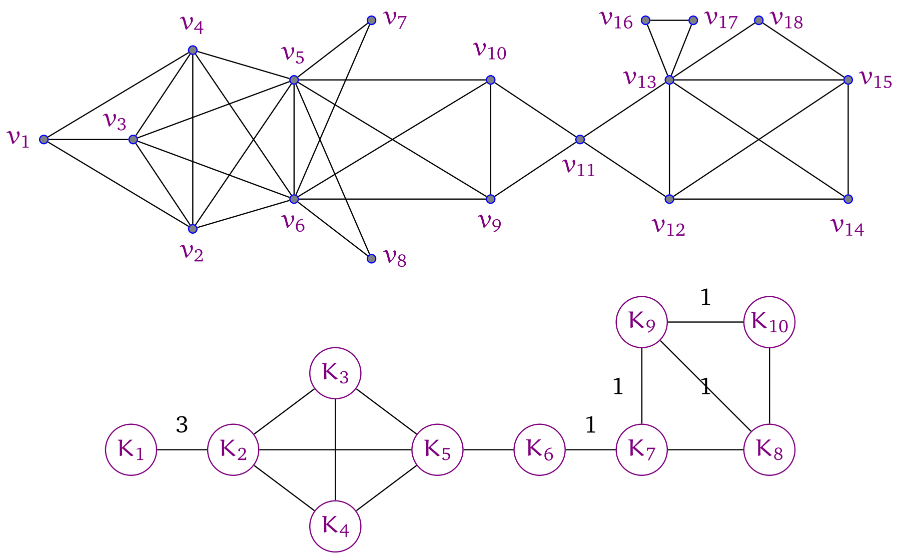


Fig. 4 On the top is a chordal graph G on 18 vertices, and below the clique graph of G, where all the omitted edge weights are 2. There are 10 maximal cliques $K_1 = \{v_1, v_2, v_3, v_4\}$, $K_2 = \{v_2, \ldots, v_6\}$, $K_3 = \{v_5, v_6, v_7\}$, $K_4 = \{v_5, v_6, v_8\}$, $K_5 = \{v_5, v_6, v_9, v_{10}\}$, $K_6 = \{v_9, v_{10}, v_{11}\}$, $K_7 = \{v_{11}, v_{12}, v_{13}\}$, $K_8 = \{v_{12}, \ldots, v_{15}\}$, $K_9 = \{v_{13}, v_{16}, v_{17}\}$, $K_{10} = \{v_{13}, v_{15}, v_{18}\}$. 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

By Lemma 3.3, MCS orderings of a chordal graph G can be fully characterized by Prim orderings of its 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 3.4 *Let z be a simplicial vertex in a chordal graph G. There exists an* MCS *ordering of G ending with z if and only if there exists a Prim ordering of $C(G)$ ending 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, including S itself. In Fig. 4, 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, and K_9K_{10} are removed in $C(G) - \{v_{12}, v_{13}\}$. The component of $C(G) - S$ containing $N[z]$ is called the *z-component* of $C(G) - S$. It is worth mentioning that $C(G) - S$ does not have a natural correspondence to any subgraph of G.

Proposition 3.5 *Let S be a separator of a chordal graph G.*

(i) *For any vertex $v \in V(G)\backslash S$, the maximal cliques containing v remain connected in $C(G) - S$.*
(ii) *For any two distinct vertices $u, v \in V(G)\backslash S$, the maximal cliques containing u are not connected to the maximal cliques containing v in $C(G) - S$ if and only if 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 the label is 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 can observe that both sides are trivially false when $uv \in E(G)$. Thus, we may assume that $uv \notin E(G)$. Suppose for the contradiction to 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 $1 \leqslant i \leqslant p$, 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 $1 \leqslant i < p$. We have thus a u–v path in G avoiding S, which contradicts 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. Let T be a clique tree of $C(G)$. Recall that T is also a (maximum) spanning tree of $C(G)$. For each $0 \leqslant i \leqslant p$, the maximal cliques containing x_i induce a subtree of T, in which each edge contains x_i. While for each $1 \leqslant j \leqslant p$, there is a maximal clique containing both x_{j-1} and x_j, thus the maximal cliques containing a vertex in $\{x_0, x_1, \ldots, x_p\}$ also induce a subtree T' of T. Note that T' is the union of all subtrees induced by the maximal cliques containing x_i. We can find a path in T', which is also a path in $C(G)$, whose one end contains u and the other contains v. Obviously, for each edge on this path, its label contains one of x_i, $0 < i < p$. Since the maximal cliques containing u are not connected to the maximal cliques containing v 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. □

Let e be a minimum-weight edge of $C(G)$. By Proposition 3.1, the label of e is a minimum separator of G. We say that e is a *critical edge* for the 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 greater than e. In Fig. 4, for example, 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.

Proposition 3.6 *Let z be a simplicial vertex of a connected chordal graph G. If all the critical edges for $N[z]$ have the same label S, then all the cliques in the z-component of $C(G) - S$*

(i) *appear consecutively in any Prim ordering of $C(G)$; and*
(ii) *can be visited last by Prim's algorithm.*

Moreover, the cliques not in the z-component of $C(G) - S$ induce a connected subgraph of $C(G)$.

Proof Since G is connected, $C(G)$ is connected as well. Let Z denote the z-component of $C(G) - S$. We first argue that the weight of every edge in Z is strictly greater than $|S|$. By definition, no edge has a smaller weight than $|S|$. Suppose that there is an edge K_iK_j in Z such that the weight of K_iK_j is $|S|$. We can find a path from $N[z]$ to K_i

or K_j in the component. Without loss of generality, we can assume that the weights of other edges on this path are strictly greater than $|S|$. Let K_i be the end of $K_i K_j$ closer to $N[z]$ on this path, and let $S' = K_i \cap K_j$. Note that $S' \neq S$ but $|S'| = |S|$. By definition, S' is a u-v separator for any pair of vertices $u \in K_i \setminus K_j$ and $v \in K_j \setminus K_i$. By Proposition 3.5(ii), K_i and K_j are not connected in $C(G) - S'$. Hence, K_i and K_j are not connected after all minimum-weight edges are removed from $C(G)$. But then $K_i K_j$ is a critical edge for $N[z]$ with label S', which is different from S, hence a contradicton.

Let π be any Prim ordering of $C(G)$. We consider the first maximal clique K in Z visited by π. Note that $|S|$ is the minimum among the weights of all edges in $C(G)$, and the weight of any edge between Z and other components of $C(G) - S$ is precisely $|S|$. No matter how many maximal cliques outside of Z have been visited, when K is going to be visited, the weight of every edge between a visited clique and an unvisited clique is precisely $|S|$. Since every edge in Z has weight strictly greater than $|S|$, after visiting K, we have to visit all the cliques in Z before visiting one outside of Z. This concludes assertion (i).

By the definition of $C(G)$, in each component of $C(G) - S$, there is a maximal clique containing S. We argue that there is an edge in $C(G)$ with label S between any two components of $C(G) - S$. Let K_p and K_q be two maximal cliques containing S in two different components of $C(G) - S$. Clearly, K_p and K_q are not connected in $C(G) - S$. By Proposition 3.5(ii), S is a u-v separator for all $u \in K_p \setminus K_q$, $v \in K_q \setminus K_p$. Since every vertex $w \in S$ is adjacent to both u and v, the set S is a minimal u-v separator. As a result, $K_p K_q$ is an edge with label S in $C(G)$. Therefore, the cliques not in Z induce a connected subgraph of $C(G)$. For assertion (ii), we can start the Prim's algorithm from a clique not in Z, and then finish all the cliques outside of Z before visiting an edge crossing Z, which has the minimum weight among all edges. □

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 for MCS end vertices. For example, this condition can be used to exclude v_{14} and v_{18} in Fig. 4. We leave it to the reader to verify that v_{14} and v_{18} cannot be MCS end vertices of the graph.

Lemma 3.7 *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 component $U_1 \in \mathcal{C}_1$ and any component $U_2 \in \mathcal{C}_2$, it holds that U_1 and U_2 have no common maximal clique and there is no edge between U_1 and U_2 in $C(G)$.

For $i = 1, 2$, by the definition of critical edges, there is a path P_i from $N[z]$ to a clique incident to e_i within the z-component; let K_i denote the other end of the path. Note that K_i is one end of e_i, and the other end is in some component of $\mathcal{C}_i$, which may not be U_i. To make U_i be a component in $C(G) - S_i$, at least one edge containing S_i is deleted, of which one end is in U_i. Therefore, there must be some clique K_i' in U_i

containing S_i. 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, $K_i K_i'$ is also a critical edge with label S_i for $N[z]$. Appending the edge $K_2 K_2'$ to P_1 we obtain a path from $N[z]$ to K_2' 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 3.5, 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 $C(G) - S_1$. Recall that K_2' and $N[z]$ are connected in $C(G) - S_1$, and U_1 is a component of $C(G) - S_1$ not containing $N[z]$. Thus, all maximal cliques of U_2 are in the z-component of $C(G) - S_1$, and hence U_1 and U_2 have no common maximal clique. 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 between them.

Let π be a Prim ordering of $C(G)$ ending with $N[z]$. Assume without loss of generality that the first visited clique in $\mathcal{C}_1 \cup \mathcal{C}_2$ 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 K from the z-component of $C(G) - S_1$. By the same argument as above, there is a path from K to $N[z]$ on which no edge is critical: otherwise we have an edge between a component of $C(G) - S_1$ and a component of $C(G) - S$ for some other critical edge with label S. After visiting K, however, π will not visit any edge of label S_2 before finishing the z-component. Therefore, $N[z]$ cannot be the end clique, a contradiction. □

In other words, if z is an MCS end vertex, then there is a unique minimum separator of G that is "the closest to z" in a sense. Although this condition itself is not sufficient, it 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 3.6 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 C'. In this juncture we may view C' as a clique graph of an induced subgraph of G and find all critical edges for $N[z]$ in C'. They also need to have the same label; suppose it is S', which is strictly greater than S. One more subtle point is that we need to make sure that when S is crossed, it can reach a maximal clique that is not in the z-component of $C' - S'$. In Fig. 4, if we delete vertices v_{16} and v_{17}, (hence K_9,) then $K_6 K_7$ is the only critical edge for K_8. The condition of Lemma 3.7 is vacuously satisfied, but v_{14} is still not an MCS end vertex. (Now v_{18} is.)

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 14 of Beisegel et al. [1]. For example, the sequence of critical edges for $N[v_1]$ in Fig. 4 are $K_6 K_7$, $K_2 K_5$, and K_1K_2, which correspond to minimal separators $\{v_{11}\}$, $\{v_5, v_6\}$, and $\{v_2, v_3, v_4\}$, respectively.

Theorem 3.8 *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 $e_1, e_2, \ldots, e_k$ in $C(G)$, where the label of e_i is S_i, on a path ending with $N[z]$ such that*

(i) *S_1 is the label of all critical edges for $N[z]$ and S_k is the set of non-simplicial vertices in $N[z]$; and*
(ii) *for $1 \leqslant i < k$, in the z-component of $C(G) - S_i$, all the critical edges for $N[z]$ have the same label, 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 $1 \leqslant i < k$.) For each $1 \leqslant i \leqslant k$, 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 3.6. The resulting ordering is 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 $e_1, e_2, \ldots, e_k$ in order, and show that for each $1 \leqslant i \leqslant k$, 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 ending with $N[z]$ because $C(G)$ is connected. Now suppose that the first i edges, namely, $e_1, \ldots, e_i$, have been selected, then we find e_{i+1} as follows. For each $1 \leqslant j \leqslant i$, let Z_j denote the z-component of $Z_{j-1} - S_j$, where $Z_0 = C(G)$. If we run Prim's algorithm from $N[z]$, then by Proposition 3.6, cliques in Z_i are the first visited. Therefore, by Lemma 3.2, we can consider Z_i as the clique graph of a graph by itself. By Lemma 3.7, all the critical edges for $N[z]$ in Z_i have the same label. Let S_{i+1} be this label, and let Z_{i+1} be the z-component of $Z_i - S_{i+1}$. We argue that there must be a maximal clique K in $Z_i - Z_{i+1}$ containing S_i; otherwise, the first component visited in $Z_i - S_{i+1}$ would be the z-component, and then $N[z]$ cannot be the last visited clique. By Proposition 3.5(i), K and K_i has no common vertex except S_i. It is easy to see that $K \cap K_i = S_i$ is a minimal x-y separator for all $x \in K_i \backslash K$ and $y \in K \backslash K_i$. By the definition of clique graph, KK_i is an edge of Z_{i-1}, and hence also an edge of $C(G)$. We can use edge KK_i to replace $e_i = K_iK'_i$—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. □

Theorem 3.8 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 has a high time complexity because the size of $C(G)$ is not linear on 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 3.8, 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 starting from $N[z]$ will cross the separators in the reversed order, and before

```
Algorithm MCS$^+$(G, σ).
Input:   A connected graph G and an MCS ordering σ of G.
Output:  An MCS$^+$ ordering $\sigma^+$ of G.

1.    for each $v \in V(G)$ do
1.1.      label($v$) ← 0;
2.    for $i \leftarrow 1$ to $n$ do
2.1.      $S$ ← the set of unvisited vertices with the maximum label;
2.2.      $v$ ← the vertex in $S$ that last appears in the ordering σ;
2.3.      $\sigma^+(v) \leftarrow i$;
2.4.      for each unvisited vertex $x \in N(v)$ do
              label($x$) ← label($x$) + 1;
3.    return $\sigma^+$.
```

Fig. 5 The MCS$^+$ algorithm

crossing the ith separator S_i, it has to exhaust the whole z-component $C(G)-S_i$. We are now ready to describe the MCS$^+$ algorithm in Fig. 5 and use it to prove Theorem 1.1.[2] The only difference lies in step 2.2: when there are more than one vertex of the largest number of visited neighbors, we use the ordering σ to break ties. Therefore, any MCS$^+$ ordering is also an MCS ordering. Note that MCS$^+$ is deterministic: MCS$^+(G, \sigma)$ is unique for any graph G and any MCS ordering σ of G.

Proof of Theorem 1.1 The if direction is correct because the ordering produced by MCS$^+$ is an MCS ordering, and hence we focus on the only if direction.

Suppose that z is an MCS end vertex of G, and let σ be an MCS ordering of G starting from z. We argue that $\sigma^+ = $ MCS$^+(G, \sigma)$ visits z at the very end.

Let $S_1, \ldots, S_k$ be the sequence of separators specified in Theorem 3.8. For $i = 1, \ldots, k$, let V_i be the set of vertices from the same component of $G - S_i$ as z, and let C_i be the z-component of $C(G) - S_i$. We also use C_0 to denote $C(G)$, and let $V_0 = V(G)$. Note that $|S_1| < |S_2| < \cdots < |S_k|$, and C_{i-1} contains C_i for $i = 1, \ldots, k$. The ith z-component C_i is an induced subgraph of $C(G)$ by Lemma 3.5, and the weight of every edge in C_i is strictly greater than $|S_i|$ by Proposition 3.6.

Thus, any Prim ordering of $C(G)$ starting from $N[z]$ visits the cliques in C_{i+1} before the others in C_i.

By Lemma 3.2, C_i is the clique graph of $G[V_i \cup S_i]$; hence, $K \subseteq V_i \cup S_i$ for every K in C_i. By Lemma 3.3, σ visits the vertices in $V_i \cup S_i$ before $(V_{i-1} \cup S_{i-1}) \setminus (V_i \cup S_i)$, and the vertices in $V(G) \setminus (V_1 \cup S_1)$ are visited in the end.

By definition, σ^+ starts from the last visited vertex of σ, which is in $V(G) \setminus (V_1 \cup S_1)$. We show by contradiction that for $i = 1, \ldots, k$, vertices in $V(G) \setminus V_i$ are visited before vertices in V_i in σ^+, i.e.,

$$\sigma^+ = V_0 \setminus V_1, V_1 \setminus V_2, \ldots, V_{k-1} \setminus V_k, V_k, \tag{1}$$

[2] The algorithm MCS$^+$ is defined in the same spirit of LBFS$^+$, and it has not been explicitly mentioned in literature because no application of this algorithm has been discovered.

while ordering of vertices in each set does not concern us. Suppose for contradiction that (1) does not hold. Let v be the first vertex in σ^+ that violates (1); suppose that $v \in V_i$ and it is visited before some vertex in $V_{i-1} \backslash V_i$. By Lemmas 3.2 and 3.3, for at least one maximal clique in C_{i-1}, all the vertices have been visited before v. Thus, before vertices in V_{i-1} have been full visited, at least one unvisited vertex x in V_{i-1} has $|S_i|$ visited neighbors. On the other hand, at the moment of visiting, all the visited neighbors of v are in S_i. Hence, the label of v cannot be greater than that of x. But since $v <_\sigma x$, the MCS$^+(G, \sigma)$ should visit x instead of v, a contradiction.

Note that C_k consists of the only clique $N[z]$ and V_k comprises true twins of z. Since $\sigma(z) = 1$, it has to be the last visited vertex of V_k, hence also the last of the whole graph. This concludes the proof of the correctness. □

The MCS$^+$ algorithm is named in a similar fashion to the famous algorithm LBFS$^+$ [8]. Unlike LBFS$^+$, however, it is not immediately clear how to carry MCS$^+$ out in linear time.

Corollary 3.9 *The* MCS *end vertex problem can be solved in $O(n^2)$ time on chordal graphs.*

Proof We apply MCS from z to produce an MCS ordering σ, and then apply MCS$^+(G, \sigma)$ to produce another ordering σ^+. We return yes if and only if the last vertex of σ^+ is z. The correctness follows from Theorem 1.1. For the running time, note that the only difference between the algorithm and the original MCS algorithm is step 2.2. 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 $O(n^2 + m) = O(n^2)$. □

Let us briefly sketch a linear-time algorithm for the MCS end vertex problem. We refrain from a formal presentation because it needs the details of the implementation of MCS, which will blur the focus of this paper. Since the algorithm MCS$^+$ will not stop at z if z is not an end-vertex, it suffices to show that it stops at z when z is an MCS end vertex. The observation is that we do not need to strictly follow the MCS$^+$. Instead, it suffices to ensure that the vertices in V_i are the last selected when crossing S_i for all $i = 1, \ldots, k$. The standard implementation of MCS is by partition refinement, and to make the correct selection in step 2.2, we need to keep vertices in each part in the reversed order of σ. This is challenging for MCS$^+$ because we may need to merge two parts during the process, and it is not clear how to maintain the order in time proportional to the number of neighbors of the current vertex.[3] By assumption, z is an MCS end vertex, and thus the separators and vertex sets are defined as above. In the execution of the first MCS, we can mark the vertices in $V_1, \ldots, V_k$ as follows. After finishing $N(z)$, we mark all the neighbors of the next selected vertex as V_k, whose label is $|S_k|$. (We do not know the value of k at this moment, so we can mark the set and rename it after all the markings are done.) Starting from $i = k$ and inductively, when the maximum label is decreased (which happens if and only if we cross S_{i-1}) we mark all the visited vertices that are not a neighbor of the next vertex as V_i. In the

[3] This is not a problem for LBFS$^+$, because it never merges two parts.

execution of MCS$^+$, note that for $i = 1, \ldots, k$, the labels of vertices in $V_i \cap N(S_i)$ increase simultaneously. Thus, they are always in the same part of the partition before the first of them is visited. In case that this part is merged with another part, it suffice to make sure that these vertices are at the end of the part.

4 Maximum Cardinality Search on Weakly Chordal Graphs

The complement $\overline{G}$ of a graph G is defined on the same vertex set as G and two distinct vertices of $\overline{G}$ are adjacent if and only if they are not adjacent in G. A graph G is *weakly chordal* if neither G nor its complement contains an induced cycle on five or more vertices. Since the complement of an induced cycle on five vertices is also an induced cycle on five vertices, and the complement of each induced cycle on six or more vertices contains an induced cycle on four vertices, all chordal graphs are weakly chordal. To prove the 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 clauses, we construct a graph G as follows (see Fig. 6 for an example). Let the variables and clauses of $\mathcal{I}$ be denoted by $x_1, x_2, \ldots, x_p$ and $c_1, c_2, \ldots, c_q$, 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 form 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 $\{a_2, u_1, u_2, y\} \times (L \cup C)$.

Proposition 4.1 *The graph G constructed above is a weakly chordal graph.*

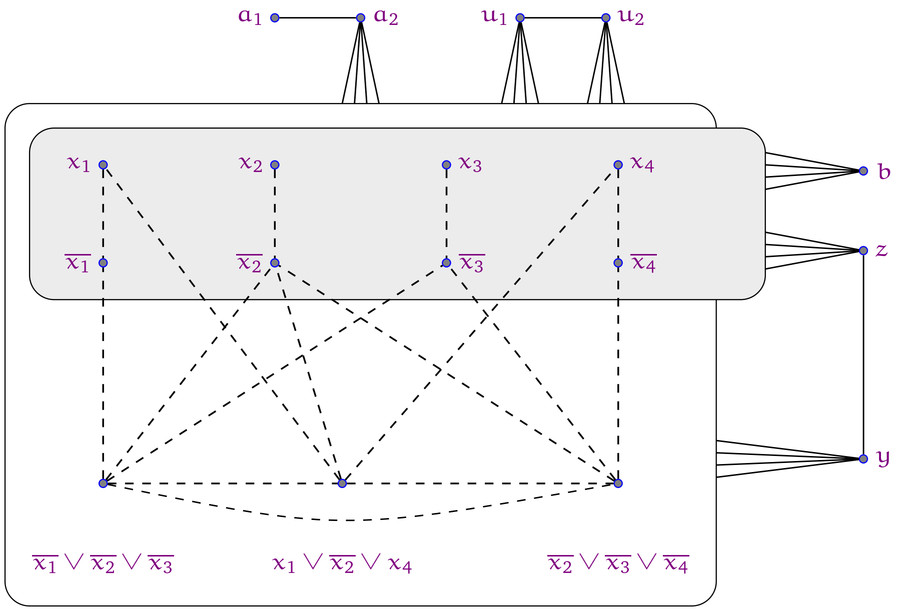
Proof We need to show that neither G nor $\overline{G}$ contains an induced cycle on five or more vertices.

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 at least two neighbors.
(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 a_1 has a single neighbor 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



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
$\overline{x_1} \vee \overline{x_2} \vee \overline{x_3}$
$x_1 \vee \overline{x_2} \vee x_4$
$\overline{x_2} \vee \overline{x_3} \vee \overline{x_4}$

Fig. 6 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

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 $\geqslant 2$ 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 a_2 has only one neighbor 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 three vertices on H are from L, but this is impossible because each vertex in L has only one neighbor in L.

We can thus conclude that G is a weakly chordal graph. □

We are now ready to prove the NP-completeness of the MCS end vertex problem on weakly chordal graphs.

Proof of Theorem 1.2 It is clear that the MCS end vertex problem is in NP, and we now show that it is NP-hard on weakly chordal graphs. 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 of G 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.

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

The remaining vertices are u_1, u_2, y, z, and the 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 can be visited in any order. We have thus obtained an MCS ordering of G ending 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 a_1 has a single neighbor, $\{\sigma(a_1), \sigma(a_2)\} = \{1, 2\}$; 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, $X = \{\ell \mid 3 \leqslant \sigma(\ell) \leqslant p + 2\} \subset L$: as for each variable, one literal vertex has more visited neighbors than b, z, y, u_1, u_2, and 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

```
1.   for each $v \in V(G)$ do label$(v) \leftarrow \emptyset$;
2.   for $i \leftarrow 1$ to $n$ do
2.1.     $S \leftarrow$ unvisited vertices with the lexicographically largest label;
2.2.     $v \leftarrow$ any vertex in $S$;
2.3.     $\sigma(v) \leftarrow i$;
2.4.     for each unvisited neighbor $x$ of $v$ do
             add $i$ to the beginning of label$(v)$;
3.   return $\sigma$.
```

Fig. 7 The LDFS algorithm

neighbors, ($\{a_2\} \cup X$,) while any other unvisited vertex in $V(G)\backslash C$ has at most p visited neighbors. But then $\sigma(c) = p + 3$, 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] gave a full characterization of MNS end vertices on chordal graphs: a vertex z of a chordal graph G is an MNS end vertex of G if and only if it is simplicial and the minimal separators of G in $N(z)$ are totally ordered by inclusion. Since LDFS, described in Fig. 7, 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.

To see that any LDFS is a DFS, note that the latest visited vertices have the largest number among all visited vertices. In particular, after visiting v in the ith iteration, if it has unvisited neighbors, then one of them has the largest label and is to be visited next. Similar to DFS, LDFS visits a neighbor of the most recent vertex, or backtracks if all its neighbors have been visited. The difference between DFS and LDFS lies in the choice when the current vertex has more than one unvisited neighbor. When there are ties, LDFS chooses a vertex with the lexicographically largest label. The following is actually a simple property of DFS.

Proposition 5.1 *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.*

Proof Let i be the first step a vertex in X is visited. By assumption, all the vertices in $N(X)$ have been visited at this moment. Thus, if a vertex in $V(G)\backslash X$ is still unvisited and has a visited neighbor, then its label starts with a number smaller than i. On the other hand, since $G[X]$ is connected, after step i and before step $i + |X| - 1$, there always exists an unvisited vertex x in X that has a visited neighbor in X. The label of x starts with a number at least i, hence lexicographically larger than the label of any unvisited vertex in $V(G)\backslash X$. □

Assume the first vertex of X is visited in step i. Then all non-empty labels of unvisited vertices in $V(G) - X$ start with a number smaller than i. After step i, it holds that there is an unvisited vertex w in X with a visited neighbor in X as long as there is still an unvisited vertex in X (since $G[X]$ is connected). The label of w starts with a number $\geqslant i$ and, thus, it is lexicographically larger than the label of any vertex in $V(G) - X$.

The following is a well-known property of a minimal separator in a chordal graph.

Proposition 5.2 [3] *Let G be a connected chordal graph, S a minimal separator of G, and C a component of $G - S$. If $N(C) = S$, then C contains a vertex adjacent to every vertex in S.*

Lemma 5.3 *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 [12] 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 $1 \leqslant i \leqslant k$, each component of $G - S_i$ not containing z is a component of $G - S_k$; let $\mathcal{C}$ denote these components. We construct an LDFS ordering σ of G ending in z as follows. It starts by 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 vertices are z and its true twins, of which it chooses z as 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 5.1, 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 vertex that has the largest label is a vertex adjacent to all visited vertices in $N(z)$, i.e., the most recently visited separator. Hence, it is always correct to select a vertex from $N(z)$. By Proposition 5.2, from each component $C \in \mathcal{C}$ with $N(C) = S_i$, we can find a vertex v such that $S_i \subseteq N(v)$, and hence v does have the largest label. □

Lemma 5.4 *The* LDFS *end vertex problem can be solved in $O(n + m)$ time on chordal graphs.*

Proof Note that the characterization of LDFS end vertices on chordal graph in Lemma 5.4 is the same to the characterization of MNS end vertices on chordal graph in [3, Characterization 8.1]. Hence, in a chordal graph, a vertex is an LDFS end vertex if and only if it is an MNS end vertex. Since the MNS end vertex problem can be solved in linear time on chordal graph [1, Corollary 16], this also holds for the LDFS end vertex problem on chordal graphs. □

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 [16]. Corneil et al. [11] 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 our algorithm in Fig. 5. They conducted an LBFS starting 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

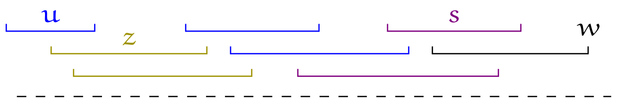
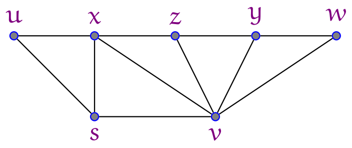


Fig. 8 A BFS starting from z may end with s or w, but a BFS starting from w has to end with u. (Note that a BFS starting from s may end with z)

Fig. 9 s, v, u, x, w, y, z is a BFS ordering ending with z



second run. As shown in Fig. 8, however, this algorithm cannot be directly adapted to the BFS end vertex problem.

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

Proposition 6.1 [14] *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 [14]. However, a BFS end vertex may satisfy neither of the two conditions. In Fig. 8, 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. However, a BFS end vertex might be at distance two to both u and w, as shown in Fig. 9.

For a fixed clique path $K_1, \ldots, K_p$ of an interval graph G, we let $\texttt{lp}(v)$ and $\texttt{rp}(v)$ denote, respectively, the smallest and the largest number i such that $v \in K_i$.[4] We use $\operatorname{dist}(u, v)$ to denote the distance between u and v. Dependent on which of the simplicial vertices in K_1 and the simplicial vertices in K_p are visited first, we can informally assign a direction to a BFS ordering σ of this graph. We say that σ goes from the left to the right if it visits simplicial vertices in K_1 earlier than those in K_p. As we cannot be sure that there is a BFS ordering ending with z from one direction, we consider both directions separately. For each of them, we try to find the first vertex and the second vertex satisfying certain conditions, and show that there exists a BFS ordering of G ending with z if and only if this pair of vertices exist.

Lemma 6.2 *The* BFS *end vertex problem can be solved in $O(n + m)$ time on interval graphs.*

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. [14] to build a clique path for G,

[4] One may note that $\{[\texttt{lp}(v), \texttt{rp}(v)] \mid v \in V(G)\}$ gives an interval representation for G.

```
INPUT: A connected interval graph $G$, a clique path $K_1, \ldots, K_p$ 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. $X \leftarrow \{x \in V(G) \mid \text{dist}(x, z) = \text{dist}(x, w) \geqslant \text{dist}(x, u)\}$;
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 $\text{dist}(s, u) - 1$ to $u$ do
       if $\text{dist}(v, z) > \text{dist}(v, u)$ then return "yes";
9. return "no."
```

Fig. 10 Main procedure for BFS end vertex on interval graphs

and take simplicial vertices v_1, v_2 from the first and last cliques of the clique path. We call the procedure described in Fig. 10 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 $\text{dist}(s, u) \leqslant \text{dist}(s, w) \leqslant \text{dist}(s, z)$. On the other hand, Proposition 6.1 implies $\text{dist}(s, z) \leqslant \max\{\text{dist}(s, u), \text{dist}(s, w)\} = \text{dist}(s, w)$. Therefore, a desired BFS ordering σ, if it exists, must start from a vertex s satisfying

$$\text{dist}(s, z) = \text{dist}(s, w) \geqslant \text{dist}(s, u). \qquad (\dagger)$$

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

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

Let P_u be any shortest s–u path and P_w any shortest 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 one of the last two; otherwise $\text{dist}(s, z) < \text{dist}(s, u)$. 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 6.1. For step 2, note that if $v \neq z$ is a universal vertex, then $\langle v, u, w, \ldots, z\rangle$ 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 $q = \text{dist}(s, z) = \text{dist}(s, w)$. Note that $q \geqslant 2$ because s is not universal. Hence, $z, w \notin N(s)$.

We show the correctness of step 6 by contradiction. Suppose that $\texttt{rp}(z) < \texttt{lp}(s)$ 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 $\texttt{lp}(s) \leqslant \texttt{lp}(s')$. Then $\texttt{rp}(u) = 1 \leqslant \texttt{rp}(z) < \texttt{lp}(s) \leqslant \texttt{lp}(s')$, therefore, $\text{dist}(s', u) \geqslant 2$. In this case, on any shortest s'–u path, z is adjacent to the second to last vertex but no vertex before it. Hence, $\text{dist}(s', z) = \text{dist}(s', u) = \text{dist}(s', w)$; let it be q'. Since $u <_\sigma w$, there must be some neighbor u'' of u at distance $q' - 1$ to s' visited before all neighbors of w. The vertex u'' cannot be universal, hence it is not adjacent 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, $1 \leqslant i \leqslant q$, the first visited vertex is w_i. As $w_1 \in K_1$ and $w_{q-1} \in K_p$, every vertex of the graph is adjacent to at least one vertex in $\{w_1, \ldots, w_{q-1}\}$. Therefore, 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 $\text{dist}(w_1, z) = \text{dist}(w_1, w) = q - 1$. By the selection of v in step 8, we know that $\texttt{lp}(v) < \texttt{lp}(s)$ and $\text{dist}(v, u) = \text{dist}(s, u) - 1 \leqslant q - 1$. It must hold that $v \neq w_1$; otherwise step 5 should have chosen v. For $1 \leqslant i \leqslant q - 1$, 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 greater 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 $\texttt{lp}(s') \geqslant \texttt{lp}(s)$, which implies $\text{dist}(s', u) \geqslant \text{dist}(s, u)$. Note that s' is adjacent to any s–w path, and hence its distance to w is at most $q + 1$. In summary,

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

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 $v \in Y \cap Z$ would be adjacent to s, and would 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 $\mathrm{dist}(s', u) = \mathrm{dist}(s', z) = \mathrm{dist}(s', w) = q$, then to have $u <_\sigma w$, one vertex in $Y \cap N(s)$ must be visited before all vertices of Z. But this would force z to be visited before w, because z is at distance $q - 1$ to all vertices in $Y \cap N(s)$. Thus, it must hold that $\mathrm{dist}(s', w) = q + 1$. If $\mathrm{dist}(s', u) = q$, then at least one vertex $v \in Y$ is adjacent to s'; it is in $N(s)$ because $\mathtt{lp}(s) \leqslant \mathtt{lp}(s')$. But then $\mathrm{dist}(s', z) \leqslant 1 + \mathrm{dist}(v, z) = 1 + q - 1 = q < \mathrm{dist}(s', w)$. Therefore, $\mathrm{dist}(s', u) = q + 1$ 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 time 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 making three BFS starting from z, w, and u, respectively. 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 any v. Therefore, the total running time is $O(n + m)$. □

Lemmas 5.4 and 6.2 conclude Theorem 1.3.

Acknowledgements Y.C. would like to thank Jing Huang for bringing the end vertex problems to his attention.

References

1. Beisegel, J., Denkert, C., Köhler, E., Krnc, M., Pivac, N., Scheffler, R., Strehler, M.: On the end-vertex problem of graph searches. Discrete Math. Theor. Comput. Sci. **21**(1), (2019). https://doi.org/10.23638/DMTCS-21-1-13
2. Bernstein, P.A., Goodman, N.: Power of natural semijoins. SIAM J. Comput. **10**(4), 751–771 (1981). https://doi.org/10.1137/0210059
3. Berry, A., Blair, J.R.S., Bordat, J.P., Simonet, G.: Graph extremities defined by search algorithms. Algorithms **3**(2), 100–124 (2010). https://doi.org/10.3390/a3020100
4. Berry, A.: Separability generalizes Dirac's theorem. Discrete Appl. Math. **84**(1–3), 43–53 (1998). https://doi.org/10.1016/S0166-218X(98)00005-5
5. Blair, J.R.S., Peyton, B.W.: An introduction to chordal graphs and clique trees. In: George, J.A., Gilbert, J.R., Liu, J.W.-H. (eds.) Graph Theory and Sparse Matrix Computation, Volume 56 of IMA, pp. 1–29. Springer, Berlin (1993)
6. Cao, Y.: Recognizing (unit) interval graphs by zigzag graph searches. In: Le Viet, H., King, V. (eds) Proceedings of the 4th SIAM Symposium on Simplicity in Algorithms (SOSA), pp. 92–106. SIAM (2021). https://doi.org/10.1137/1.9781611976496.11
7. Charbit, P., Habib, M., Mamcarz, A.: Influence of the tie-break rule on the end-vertex problem. Discrete Math. Theor. Comput. Sci. **16**(2), 57–72 (2014). https://doi.org/10.46298/dmtcs.2081
8. Corneil, D.G.: Lexicographic breadth first search: a survey. In: Volume 3353 of LNCS, pp. 1–19. Springer (2004). https://doi.org/10.1007/978-3-540-30559-0_1
9. Corneil, D.G.: A simple 3-sweep LBFS algorithm for the recognition of unit interval graphs. Discrete Appl. Math. **138**(3), 371–379 (2004). https://doi.org/10.1016/j.dam.2003.07.001
10. Corneil, D.G., Dalton, B., Habib, M.: LDFS-based certifying algorithm for the minimum path cover problem on cocomparability graphs. SIAM J. Comput. **42**(3), 792–807 (2013). https://doi.org/10.1137/11083856X
11. Corneil, D.G., Köhler, E., Lanlignel, J.-M.: On end-vertices of lexicographic breadth first searches. Discrete Appl. Math. **158**(5), 434–443 (2010). https://doi.org/10.1016/j.dam.2009.10.001
12. Corneil, D.G.: A unified view of graph searching. SIAM J. Discrete Math. **22**(4), 1259–1276 (2008). https://doi.org/10.1137/050623498

13. Corneil, D.G., Olariu, S., Stewart, L.: Linear time algorithms for dominating pairs in asteroidal triple-free graphs. SIAM J. Comput. **28**(4), 1284–1297 (1999). https://doi.org/10.1137/S0097539795282377. (**A preliminary version appeared in ICALP 1995**)
14. Corneil, D.G., Olariu, S., Stewart, L.: The LBFS structure and recognition of interval graphs. SIAM J. Discrete Math. **23**(4), 1905–1953 (2009). https://doi.org/10.1137/S0895480100373455. (**A preliminary version appeared in SODA 1998**)
15. Dirac, G.A.: On rigid circuit graphs. Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg **25**(1), 71–76 (1961). https://doi.org/10.1007/BF02992776
16. Fulkerson, D.R., Gross, O.A.: Incidence matrices and interval graphs. Pac. J. Math. **15**(3), 835–855 (1965). https://doi.org/10.2140/pjm.1965.15.835
17. Galinier, P., Habib, M., Paul, C.: Chordal graphs and their clique graphs. In: Nagl, M. (eds) Proceedings of the 21st International Workshop on Graph-Theoretic Concepts in Computer Science (WG), volume 1017 of LNCS, pp. 358–371. Springer (1995). https://doi.org/10.1007/3-540-60618-1_88
18. Habib, M., McConnell, R.M., Paul, C., Viennot, L.: Lex-BFS and partition refinement, with applications to transitive orientation, interval graph recognition and consecutive ones testing. Theoret. Comput. Sci. **234**(1–2), 59–84 (2000). https://doi.org/10.1016/S0304-3975(97)00241-7
19. Hopcroft, J.E., Tarjan, R.E.: Efficient planarity testing. J. ACM **21**(4), 549–568 (1974). https://doi.org/10.1145/321850.321852
20. Impagliazzo, R., Paturi, R.: On the complexity of k-SAT. J. Comput. Syst. Sci. **62**(2), 367–375 (2001). https://doi.org/10.1006/jcss.2000.1727. (**A preliminary version appeared in CCC 1999**)
21. Kratsch, D., Liedloff, M., Meister, D.: End-vertices of graph search algorithms. In: Paschos, V.T., Widmayer, P. (eds) Proceedings of the 12th International Conference on Algorithms and Complexity (CIAC), volume 9079 of Lecture Notes in Computer Science, pp. 300–312. Springer (2015). https://doi.org/10.1007/978-3-319-18173-8_22
22. Li, P., Yaokun, W.: A four-sweep LBFS recognition algorithm for interval graphs. Discrete Math. Theor. Comput. Sci. **16**(3), 23–50 (2014). https://doi.org/10.46298/dmtcs.2094
23. Nagamochi, H.: Computing edge-connectivity in multigraphs and capacitated graphs. SIAM J. Discrete Math. **5**(1), 54–66 (1992). https://doi.org/10.1137/0405004
24. Nagamochi, H., Ibaraki, T.: Algorithmic aspects of graph connectivity. In: Encyclopedia of Mathematics and its Applications. Cambridge University Press (2008). https://doi.org/10.1017/CBO9780511721649
25. Rose, D.J., Tarjan, R.E., Lueker, G.S.: Algorithmic aspects of vertex elimination on graphs. SIAM J. Comput. **5**(2), 266–283 (1976). https://doi.org/10.1137/0205021. (**A preliminary version appeared in STOC 1975**)
26. Sethi, R.: Scheduling graphs on two processors. SIAM J. Comput. **5**(1), 73–82 (1976). https://doi.org/10.1137/0205005
27. Shier, D.R.: Some aspects of perfect elimination orderings in chordal graphs. Discrete Appl. Math. **7**(3), 325–331 (1984). https://doi.org/10.1016/0166-218X(84)90008-8
28. Simon, K.: 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, pp. 289–308 (1991). https://doi.org/10.1007/3-540-54891-2_22
29. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J. Comput. **1**(2), 146–160 (1972). https://doi.org/10.1137/0201010. (**A preliminary version appeared in SWAT (FOCS) 1971**)
30. Tarjan, R.E., Yannakakis, M.: Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. SIAM J. Comput. **13**(3), 566–579 (1984). With Addendum in the same Journal **14**(1):254–255 (1985.) https://doi.org/10.1137/0213035
31. Tedder, M., Corneil, D.G., Habib, M., Paul, C.: Simpler linear-time modular decomposition via recursive factorizing permutations. In: Automata, Languages and Programming (ICALP), Volume 5125 of LNCS, pp. 634–645. Berlin, Heidelberg (2008). https://doi.org/10.1007/978-3-540-70575-8_52
32. Zou, M., Wang, Z., Wang, J., Cao, Y.: End vertices of graph searches on bipartite graphs. Inf. Process. Lett. **173**, 106176 (2022). https://doi.org/10.1016/j.ipl.2021.106176