Kayles on special classes of graphs — An application of Sprague-Grundy theory

Hans L. Bodlaender

Technical Report RUU-CS-91-49
December 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0924-3275

Kayles on special classes of graphs — An application of Sprague-Grundy theory

Hans L. Bodlaender
Department of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract

Kayles is the game, where two players alternately choose a vertex that has not been chosen before nor is adjacent to an already chosen vertex from a given graph. The last player that choses a vertex wins the game. We show, with help of Sprague-Grundy theory, that the problem to determine which player has a winning strategy for a given graph, can be solved in $O(n^3)$ time on interval graphs, on circular arc graphs, and on permutation graphs, and in $O(n^{1.631})$ time on cographs. For general graphs, the problem is known to be PSPACE-complete, but can be solved in time, polynomial in the number of isolatable sets of vertices of the graph.

1 Introduction

For various reasons, games keep attracting the interest of researchers in mathematics and computer science. Games can provide for models, for instance for human thought processes, economic behavior, fault tolerance in computer systems, and computational complexity of machine models. Also, the analysis of games can provide for entertainment, and/or beautiful theory that is interesting on its own. It may be interesting to note that the very first book written on graph theory [7] already contained a section on the relations between graphs and games.

In this paper we consider a combinatorial game, that is played on graphs, called *Kayles*. In this game, two players alternately choose a vertex from a given graph. Players may not choose a vertex that has been chosen before, and may also not choose a vertex that is adjacent to a vertex that has been chosen before. The last player that is able to choose a vertex wins the game.

The game can also described as follows: when a player chooses a vertex, this vertex and all its neighbors are removed from the graph. The first player that ends his move with the empty graph wins the game.

We consider the problem: given a graph $G = (V, E)$, does there exist a winning strategy for the first player when Kayles is played on G? We denote this problem also

by the name *Kayles*. Kayles has been shown to be PSPACE-complete by Schaefer [8].

Despite its intractability for general graphs, Kayles has some nice characteristics, which together allow for efficient algorithms that solve some special cases. We remark that the game Kayles is:

- a two player game.
- finite. The game always ends after a finite number of moves, and each player can choose each time from a finite number of possible moves.
- full-information. There is no information that is hidden to one or both players, like e.g. in bridge, where cards of other players are unknown.
- deterministic. Every move gives rise to a unique position; no randomization devices (like dice) are used.
- impartial. This means that positions have no preference towards players. In other words, for each position, either the player that must move has a winning strategy, or the other player has — this is regardless of whether player 1 or player 2 must move from the position. For example, chess is not impartial, as there are white and black pieces owned by the players.
- with 'last player that moves wins the game' rule.

These six characteristics of Kayles make that it can be analyzed with help of Sprague-Grundy theory. Some readers may know this theory as the theory of the game Nim. In this theory, one associates to each position a (natural) number, here called nimber after [1]. (The position has nimber i, when it can be represented by a stack of corresponding height in the game Nim.) It is possible to do some calculations with these nimbers, and determine which player has a winning strategy. In many cases, these calculations will be intractable, but — as will be shown in this paper — in some cases, they are not.

Those basic notions and results of Sprague-Grundy theory that are needed for this paper are reviewed in section 2. For more background, we recommend the reader to consult [1] or [3]. Some graph theoretic definitions are also given in section 2.

In section 3, we give a data structure, needed for the algorithm, described in section 4. This algorithm solves Kayles on a graph with n vertices, e edges, and α different isolatable sets of vertices, in time $O(\alpha ne)$. In sections 5, 6 and 7 we give modifications of this algorithm, that solve Kayles on interval graphs, circular arc graphs, cographs, and on permutation graphs in polynomial time. For the result on cographs, we show that the nimber of a cograph with n vertices is of size $O(n^{0.631})$. Some final remarks are made in section 8.

2 Definitions and preliminary results

In this section we give some definitions, and review some results from Sprague-Grundy theory. All graphs in this paper are considered to be finite, undirected and simple. For a graph $G = (V, E)$, and a subset of the vertices $W \subseteq V$, the subgraph of G, induced by W, is denoted by $G[W] = (W, \{(v, w) \in E \mid v, w \in W\})$. We denote $|V|$ by n, and $|E|$ by e. For $v \in V$, denote the set containing v and all neighbors of v by $N(v) = \{v\} \cup \{w \in V \mid (v, w) \in E\}$. For $X \subseteq V$, write $N(X) = \cup_{v \in X} N(v)$.

Definition 2.1 *Let $G = (V, E)$ be a graph. A set of vertices $W \subseteq V$ is called* isolatable, *if both of the following conditions hold:*

1. *$G[W]$ is connected.*
2. *There exists a set $X \subseteq V$, with $W = V - N(X)$.*

In other words, a set $W \subseteq V$ is isolatable, if it induces a connected subgraph of G, and there exists a set of vertices X, such that if we remove X and all neighbors of vertices in X from V, then W remains. The latter condition is equivalent to the following condition:

> There exists a set $X \subseteq V$, such that $G[W]$ is a connected component of $G[V - N(X)]$.

In section 4 we will prove that Kayles can be solved in time, polynomial in the number of different isolatable subsets of vertices of the input graph G. Thus, we are interested in classes of graphs where this number is bounded by a polynomial in the number of vertices of the graph. Examples of such classes are the interval graphs, the circular arc graphs, the cographs and the permutation graphs.

Definition 2.2 *A graph $G = (V, E)$ is an* interval graph, *iff one can associate with each vertex $v \in V$ an interval on the real line $[b_v, e_v] \subseteq R$, such that for all $v, w \in V$, $v \neq w$: $(v, w) \in E \Leftrightarrow [b_v, e_v] \cap [b_w, e_w] \neq \emptyset$.*

Interval graphs can be recognized in $O(n + e)$ time, and in the same order of time, the corresponding interval model can be built [2]. As only the order of the endpoints of the intervals matters, one can assume that all $b_v, e_v \in \{1, 2, \ldots, 2n\}$.

A generalization of the interval graphs are the circular arc graphs. They can de defined as follows:

Definition 2.3 *A graph $G = (V, E)$ is a* circular arc graph, *iff one can associate with each vertex $v \in V$ an set of integers $S_v \subseteq \{1, 2, \ldots, 2n\}$, ($n = |V|$), with S_v is either of the form $\{b_v, b_v+1, \ldots, e_v-1, e_v\}$, or of the form $\{b_v, b_v+1, \ldots, 2n-1, 2n\} \cup \{1, 2, \ldots, e_v - 1, e_v\}$, ($b_v, e_v \in \{1, 2, \ldots, 2n - 1, 2n\}$,) such that for all $v, w \in V$, $v \neq w$: $(v, w) \in E \Leftrightarrow S_v \cap S_w \neq \emptyset$.*

Recognition of circular arc graphs, and building the corresponding representation can be done in $O(n^3)$ time [9].

Definition 2.4 *Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two disjoint graphs. The (disjoint) union of G_1 and G_2 is the graph $G_1 \cup G_2 = (V_1 \cup V_2, E_1 \cup E_2)$. The product of G_1 and G_2 is the graph $G_1 \times G_2 = (V_1 \cup V_2, E_1 \cup E_2 \cup \{(v, w) \mid v \in V_1,\ w \in V_2\})$.*

In some cases, we write $G_1 \cup G_2$, or $G_1 \times G_2$, for graphs G_1, G_2 which are not disjoint. As we always want the operations $\cup$ and $\times$ to work on disjoint graphs, we assume implicitly in these cases, that we take the disjoint union or product of two disjoint graphs that are isomorphic with G_1, and G_2, respectively.

Definition 2.5 *A graph is a cograph, if and only if it can be formed by the following rules:*

1. *Every graph with one vertex and no edges is a cograph.*
2. *If $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ are disjoint cographs, then $G_1 \cup G_2$ and $G_1 \times G_2$ are cographs.*

To each cograph G, one can associate a labeled rooted tree T_G, called the *cotree* of G. Each leaf node of T_G corresponds to a (unique) vertex of V. Each internal node is labeled with either a 0 or a 1. Children of nodes, labeled with 1 are labeled with 0, and vice versa. Two vertices are connected, if and only of their lowest common ancestor in the cotree is labeled with a 1. It is possible to associate a cotree with each node of the tree. Leaf nodes correspond to the cotree with the one vertex they represent. Internal nodes labeled with 0 (1) correspond to the disjoint union (product) of the cographs, corresponding to the children of the node. G equals to cograph corresponding with the root of T_G. Cographs can be recognized in $O(n+e)$ time, and in the same time the corresponding cotree can be build [4].

Definition 2.6 *A graph $G = (V, E)$ is a permutation graph, iff there exist a bijection $f : V \to \{1, 2, \ldots, n\}$, and a permutation $\pi : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, n\}$, such that for all $v, w \in V$: $(v, w) \in E \Leftrightarrow (f(v) < f(w)$ and $\pi(f(v)) > \pi(f(w))$) or $(f(v) > f(w)$ and $\pi(f(v)) < \pi(f(w))$).*

Permutation graphs can be recognized in $O(n^3)$ time, and in the same time, the corresponding numbering of the vertices and permutation π can be found [6].

Next, we review some notions and results from Sprague-Grundy theory. For a good introduction to this theory, the reader is referred to [3] or the less formal and very entertaining [1].

A *nimber* is an integer $\in N = \{0, 1, 2, \ldots\}$. For a finite set of nimbers $S \subseteq N$, define the minimum excluded nimber of S as $mex(S) = \min\{i \in N \mid i \notin S\}$.

We now assume that we consider positions in a two-player game, that is finite, deterministic, full-information, impartial, with 'last player wins'-rule. (As in Kayles.)

To each position in such a game, one can associate a nimber in the following way. If no move is possible in the position (and hence the player that must move loses the game), the position gets nimber 0. Otherwise the nimber is the minimum excluded nimber of the set of nimbers of positions that can be reached in one move.

Theorem 2.1 *[1, 3] There is a winning strategy for player 1 from a position, if and only if the nimber of that position is at least 1.*

Denote the nimber of a position p by $nb(p)$. We next define the sum of two games. For (finite, deterministic, impartial, ...) games $\mathcal{G}_1$, $\mathcal{G}_2$, the sum of $\mathcal{G}_1$ and $\mathcal{G}_2$, $\mathcal{G}_1 + \mathcal{G}_2$ is the game, where each player when moving first decides whether he wants to make a move in $\mathcal{G}_1$ or in $\mathcal{G}_2$, and then selects a move in that game. The player that makes the last move (whether it is in $\mathcal{G}_1$ or $\mathcal{G}_2$) wins the game $\mathcal{G}_1 + \mathcal{G}_2$.

Definition 2.7 *Let $i_1, i_2 \in N$ be nimbers. $i_1 \oplus i_2$ is the binary sum of i_1 and i_2 without carry, i.e., $i_1 \oplus i_2 = \sum\{2^j \mid (\lfloor i_1/2^j \rfloor$ is odd$) \Leftrightarrow (\lfloor i_2/2^j \rfloor$ is even$)\}$.*

In other words, write i_1 and i_2 in binary notation. For every digit, take a 1 if either i_1 has a 1 for that digit, and i_2 has a 0 for that digit, or vice versa. For example $10 \oplus 7 = (8+2) \oplus (4+2+1) = 8+4+1 = 13$.

With (p_1, p_2) we denote the position in $\mathcal{G}_1 + \mathcal{G}_2$, where the position in $\mathcal{G}_i$ is p_i $(i = 1, 2)$.

Theorem 2.2 *[1, 3] Let p_1 be a position in $\mathcal{G}_1$, p_2 a position in $\mathcal{G}_2$. The nimber of position (p_1, p_2) in $\mathcal{G}_1 + \mathcal{G}_2$ equals $nb((p_1, p_2)) = nb(p_1) \oplus nb(p_2)$.*

3 A data structure

In this section we describe a data structure X, which is needed for the algorithm in section 4. For a finite, ordered set V, the data structure can store subsets $W \subseteq V$ with a value $val(W)$, and retrieve these values. The following operations are possible on the data structure:

- store(W, x), for $W \subseteq V$, $x \in N$.
- present(W). Returns true, if an operation store(W, x) has been performed before for any value of x.
- val(W). Returns, if present(W), the value x of the last operation store(W, x). Undefined, if not present(W) holds.

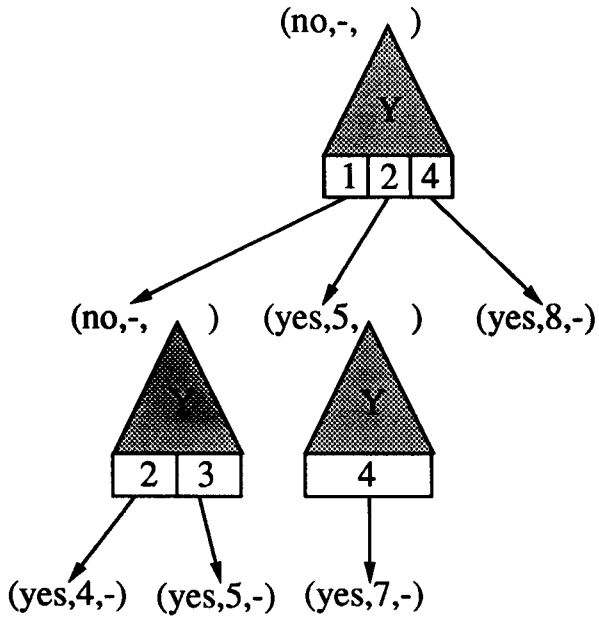

Figure 1: An example of data structure X, storing $\{1,2\}$ with value 4, $\{1,3\}$ with value 5, $\{2\}$ with value 5, $\{2,4\}$ with value 7, and $\{4\}$ with value 8

We assume an ordering $<$ on V, and assume that testing $<$ can be done in constant time. (In our application, V is the set of vertices of G. We just number the vertices of G by $v_1, v_2, \ldots$)

We now give a simple, recursive description of the data structure X. It consists of three parts:

- A boolean variable *empty_present*, which denotes whether present($\emptyset$).
- An integer variable *empty_val*, which denotes val($\emptyset$), if present($\emptyset$), and is undefined otherwise.
- A data structure Y, where for every $v \in V$ such that there exists a $W \subseteq V$ with present(W) and v is the lowest numbered element in W, a pointer is stored to another data structure of type X, that stores all sets $W \subseteq V$ with v the lowest numbered vertex in W.

An example of this construction is given in figure 1.

To search for a set $W \subseteq V$ (assume W is given in sorted order): if $W = \emptyset$, then *empty_present* and *empty_val* give the desired information. Otherwise, let v be the smallest numbered vertex in W, and search for v in the data structure Y. If v does not appear in Y, then not present(W). Otherwise, follow the pointer for v in data

structure Y to the smaller data structure X, and repeat the process with $W - \{v\}$ in this data structure.

To store(W, x), do the following. (Again, we assume that W is given in sorted order.) If the data-structure is not yet initialized, initialize it. (Reserve memory locations for *empty_present* and *empty_val*, and do all what is necessary to initialize data structure Y.) If $W = \emptyset$, set *empty_present* to **true**, and set *empty_val* to x. Otherwise, take the smallest element v from W, and check whether v is present in Y. If not, insert v in Y, and add a pointer to a new, not yet initialized data structure of type X. Otherwise, find the pointer stored for v in Y. In both cases, follow the pointer, and repeat with $W - \{v\}$ on the smaller data structure.

There are several possible ways to implement the data structures Y. Three of these possibilities are:

- as a balanced search tree, e.g. an AVL-tree. As both search and insert operations cost $O(\log n)$ time per access to a data structure of type Y, the total time per store, present or val operation is $O(|W| \log |V|)$. The space needed is $O(\sum_{\text{present}(W)} |W|)$.
- as an van Emde Boas data structure [10]. The operations in data structure Y take now $O(\log \log n)$ time and the total time per store present or val operation becomes $O(|W| \log \log |V|)$. The space needed per data structure Y is $O(|V|)$, hence in total $O(|V|\alpha)$, where α is the number of sets W that are present in the data structure.
- as a dynamic perfect hashing data structure, as described by Dietzfelbinger et al [5]. Search operations cost $O(1)$ time worst case, insert operations cost $O(1)$ expected time. Memory use is linear in the number of stored keys. This method uses randomization. Using this data structure, present and val operations cost $O(|W|)$ time worst case, and store operations cost $O(|W|)$ expected time. The amount of space that is used again is $O(\sum_{\text{present}(W)} |W|)$.

4 Kayles and nimbers

As Kayles is an impartial, deterministic, finite, full-information, two-player game with the rule that the last player that moves wins the game, we can apply Sprague-Grundy theory to Kayles, and we can associate with each graph G the nimber of the start position of the game Kayles, played on G. We denote this nimber $nb(G)$, and call it the *nimber of* G.

Note that when $nb(G)$ is known, then one can directly determine, with theorem 2.2 which player has a winning strategy.

An important observation is the following: when $G = G_1 \cup G_2$ for disjoint graphs G_1, G_2, then the game Kayles, played on G, is the sum of the game Kayles, played on G_1, and the game Kayles, played on G_2. Hence, by theorem 2.2, we have the following result.

Theorem 4.1 $nb(G_1 \cup G_2) = nb(G_1) \oplus nb(G_2)$.

The second important observation is that when some vertices $X \subseteq V$ have been chosen, then the nimber of the resulting position is equal to the nimber of $G[V - N(X)]$. In other words: the game does not change when we remove all vertices that have been chosen and all their neighbors from G. In particular, the nimber of the position arising from playing v in G as first move equals $nb(G[V - N(v)])$.

It are these two basic observations that make it possible to compute the nimber of graphs with a polynomial number of isolatable sets of vertices in polynomial time.

Theorem 4.2 *There exists algorithms, that given a graph $G = (V, E)$, computes $nb(G)$ in $O(\alpha ne + \alpha n^2 \log\log n)$ time worst case, or in $O(\alpha ne)$ expected time, respectively, and in $O(\alpha n)$ space, where α is the number of different isolatable sets of vertices of G.*

Proof: We use a data structure X, as described in section 3, in which we store isolatable sets $W \subseteq V$ with $nb(G[W])$. Initially, X is empty. The following recursive procedure, when called with a set $W \subseteq V$ returns the value of $nb(G[W])$. It is called by the main algorithm with compute_nimber(W), for every connected component $G[W]$ of G. The nimber of G is the $\oplus$-sum of these nimbers.

```
procedure compute_nimber (W): integer;
{W ⊆ V. The procedure returns the nimber nb(G[W]).}
begin
    if present(W) then return(val(W))
    else
        M := ∅;
        for all v ∈ W do
        begin compute the connected components of G[W − N(v)].
              Suppose these components have sets of vertices
              W_1, ..., W_r ⊆ W.
              { We compute now: nb(G[W − N(v)]) = nb(G[W_1]) ⊕ ...
              ⊕nb(G[W_r]).}
              n := 0;
              for i := 1 to r do
              begin n := n⊕ compute_nimber(W_i);
              end;
              { Now n = nb(G[W − N(v)]).}
        end;
        { M forms the set of the nimbers of positions, reachable in one
        move from G[W].}
        ans := mex(M);
        store(W,ans);
        return(ans)
end.
```

Correctness of the algorithm follows from the earlier made observations. For the running time of the algorithm, observe that the computations in the else-part are carried at most once for a set W. compute_nimber is only called with isolatable sets $W \subseteq V$. It is possible to keep all sets W, W_i sorted in time, linear in W. We count the time, needed for one call of compute_nimber(W), including the calls of present(W_i), and if present(W_i), of val(W_i). We need $O(n^2 + ne) = O(ne)$ time for all computations of connected components. In case the data structures Y are implemented with van Emde Boas data structures, the calls to present(W_i) cost $O(|W_i| \log\log n)$ time each. In total, this is $O(n^2 \log\log n)$ time. If we use dynamic perfect hashing, the expected time becomes $O(n^2)$. The total over all α isolatable sets of vertices gives the bounds stated in the theorem. Clearly, the space needed for the data structure is bounded by $O(\alpha n)$. □

5 Kayles on interval graphs and circular arc graphs

In this section we consider Kayles when played on interval graphs, or on circular arc graphs. We show that nimbers of these graphs, and hence players with a winning strategy, can be determined quickly, as there are only $O(n^2)$ isolatable sets of vertices, and these have an nice and easy structure.

Assume that $G = (V, E)$ is an interval graph, and let with each vertex $v \in V$ an interval $[b_v, e_v]$ be associated, such that $b_v, e_v \in \{1, 2, \ldots, 2n\}$, $b_v \leq e_v$, and for all $v, w \in V$, $v \neq w$: $(v, w) \in E \Leftrightarrow [b_v, e_v] \cap [b_w, e_w] \neq \emptyset$.

Lemma 5.1 *Let $W \subseteq V$ be an isolatable set of vertices. Then there exist numbers $b_W, e_W \in \{1, 2, \ldots\}$, such that for all $v \in V$: $v \in W \Leftrightarrow b_w \leq b_v \leq e_v \leq e_W$.*

Proof: Take $b_W = \min\{b_v \mid v \in W\}$, and $e_W = \max\{e_v \mid v \in W\}$. Suppose $b_{v_1} = b_W$, $e_{v_2} = e_W$, $v_1, v_2 \in W$. Clearly $v \in W \Rightarrow b_W \leq b_v \leq e_v \leq e_W$.

Suppose there exists a vertex $w \notin W$ with $b_W \leq b_w \leq e_w \leq e_W$. Either $w \in X$, or w is adjacent to a vertex $x \in X$. In both cases, we have a vertex $x \in X$ with $[b_x, e_x] \cap [b_W, e_W] \neq \emptyset$. As x may not be adjacent to v_1 or v_2, we have $b_W < b_x \leq e_x < e_W$. As vertices in W are not adjacent to X, we can write W as the disjoint union of $A = \{v \in W \mid e_v < b_x\}$ and $B = \{v \in W \mid e_x < b_v\}$. A and B are non-empty, as they contain v_1 and v_2, respectively, and there cannot be an edge between a vertex in A and a vertex in B. Hence $G[W]$ is not connected, contradiction. □

From this result and theorem 4.2, it follows that Kayles can be solved in polynomial time on interval graphs. We give a modification of this general result, which uses $O(n^3)$ time.

Let G_{ij} $(1 \leq i \leq j \leq n)$ denote the subgraph of G, induced by all vertices $v \in V$ with $i \leq b_v \leq e_v \leq j$: $G_{ij} = G[\{v \in V \mid i \leq b_v \leq e_v \leq j\}]$. The algorithm uses a two-dimensional array A, where $A(i,j)$ will store the value of $nb(G_{ij})$.

for $i := 0$ **to** $2n-1$ **do**
begin for $j := 1$ **to** $2n-i$ **do**
  begin $S := \emptyset$;
    for all $v \in V$ **do**
    begin if $j < b_v \leq e_v < j+i$
      then $S := S \cup \{A(j, b_v - 1) \oplus A(e_v + 1, j+i)$
      else if $j = b_v \leq e_v < j+i$
      then $S := S \cup \{A(e_v + 1, j+i)\}$
      else if $j < b_v \leq e_v = j+i$
      then $S := S \cup \{A(j, b_v - 1)\}$
      else if $j = b_v \leq e_v = j+i$
      then $S := S \cup \{0\}$
    end;
    $A(j, j+i) := mex(S)$
  end
end;
output $A(1, 2n)$.

The algorithm computes $nb(G_{j(j+i)})$ successively for i ranging from 0 to $2n-1$ and j from 1 to $2n-i$. If $j \leq b_v \leq e_v \leq j+i$, then v belongs to $G_{j(j+i)}$. Then, removing v and its neighbors from $G_{j(j+i)}$ splits $G_{j(j+i)}$ in two parts $G_{j(b_v-1)}$ and $G_{(e_v+1)(j+i)}$. If either $j = b_v$, or $e_v = j+i$, then one of this parts is empty. If both $j = b_v$ and $e_v = j+i$, then both parts are empty and the resulting nimber is 0. It follows that for each v in $G_{j(j+i)}$, $nb(G_{j(j+i)} - N(v))$ is computed correctly, and put into S. Hence $mex(S) = nb(G_{j(j+i)})$. This shows correctness of the algorithm.

Clearly, the algorithm uses $O(n^3)$ time. With theorem 2.1, we can directly determine which player has a winning strategy after execution of the algorithm.

Theorem 5.2 *There exists an $O(n^3)$ algorithm for Kayles on interval graphs.*

The same result holds for circular arc graphs. We use basically the same algorithm. Define V_{ij} to be the set $\{v \in V \mid S_v \subseteq \{i, i+1, \ldots, j-1, j\}\}$, if $i \leq j$, and $\{v \in V \mid S_v \subseteq \{i, i+1, \ldots, 2n-1, 2n\} \cup \{1, 2, \ldots, j-1, j\}$, if $i > j$. Similar as before, we can compute all $nb(G_{ij})$, now for all pairs i, j, with $i, j \in \{1, 2, \ldots, 2n\}$, using dynamic programming, in $O(n^3)$ time in total.

Theorem 5.3 *There exists an $O(n^3)$ algorithm for Kayles on circular arc graphs.*

6 Kayles on cographs

Cographs also have the property that the number of isolatable sets is bounded by a polynomial.

Theorem 6.1 *Let $G = (V, E)$ be a cograph, and let T_G be its corresponding cotree. Let $W \subseteq V$ be an isolatable set of size at least two. Then there exists a 1-labeled internal node i in T_G such that W equals all vertices, represented by a leaf-descendant of i in T_G.*

Proof: Let i be the lowest common ancestor in T_G of all vertices in W. i is an internal node with label 1. (If i has label 0, then $G[W]$ is not connected.) i cannot have a leaf-descendant x with $x \in X$, because then i will be lowest common ancestor of a vertex $w \in W$ and x, hence $(w, x) \in E$, contradiction.

Suppose a leaf-descendant v of i does not belong to W. v must be adjacent to a vertex $x \in X$, and the lowest common ancestor j of v and x must be a 1-labeled internal node that is an ancestor of i. But now j is also the lowest common ancestor of an arbitrary vertex $w \in W$ and x, hence $(w, x) \in E$, contradiction. □

It follows that there are $O(n)$ isolatable sets in a cograph with n vertices. From theorem 4.2, it follows that Kayles can be solved on cographs in $O(n^2e)$ expected time, or $O(n^2e + n^3 \log\log n)$ worst case time. A better algorithm can be obtained with a more careful analysis. Hereto, we compute for each cograph, associated with a node in the cotree T_G, the set of nimbers of the positions, reachable in one move.

Definition 6.1 *Let $G = (V, E)$ be a graph. The nimberset of G is the set of nimbers $nbs(G) = \{nb(G[V - N(v)] \mid v \in V\}$.*

Recall that $nb(G) = mex(nbs(G))$. We use the following notation: for a set of nimbers $S \subseteq N$, and a nimber α, we denote $\alpha \oplus S = \{\alpha \oplus \beta \mid \beta \in S\}$.

Lemma 6.2 *Let $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$ be two disjoint graphs.*
(i) $nbs(G_1 \cup G_2) = nb(G_2) \oplus nbs(G_1) \cup nb(G_1) \oplus nbs(G_2)$.
(ii) $nbs(G_1 \times G_2) = nbs(G_1) \cup nbs(G_2)$.

Proof: (i) $nbs(G_1 \cup G_2) = \{nb(G_1[V_1 - N(v)] \cup G_2) \mid v \in V_1\} \cup \{nb(G_2[V_2 - N(v)] \cup G_1) \mid v \in V_2\} = nb(G_2) \oplus \{nb(G_1[V_1 - N(v)] \mid v \in V_1\} \cup nb(G_1) \oplus \{nb(G_2[V_2 - N(v)] \mid v \in V_2\} = nb(G_2) \oplus nbs(G_1) \cup nb(G_1) \oplus nbs(G_2)$.

(ii) Write $G = G_1 \times G_2$. $nbs(G) = \{nb(G[V_1 \cup V_2 - N(v)]) \mid v \in V_1\} \cup \{nb(G[V_1 \cup V_2 - N(v)]) \mid v \in V_2\} = \{nb(G_1[V_1 - N(v)]) \mid v \in V_1\} \cup \{nb(G_2[V_2 - N(v)]) \mid v \in V_2\} = nbs(G_1) \cup nbs(G_2)$. □

The lemma can be generalized as follows.

Lemma 6.3 *Let* $G_1 = (V_1, E_1)$, $G_2 = (V_2, E_2)$, ..., $G_r = (V_r, E_r)$ *be* r *disjoint graphs.*
(i) $nbs(G_1 \cup G_2 \cup \ldots \cup G_r) = \bigcup_{1 \leq i \leq r} nb(G_1) \oplus nb(G_2) \oplus \ldots \oplus nb(G_{i-1}) \oplus nb(G_{i+1}) \oplus \ldots \oplus nb(G_r) \oplus nbs(G_i)$.
(ii) $nbs(G_1 \times G_2 \times \ldots \times G_r) = nbs(G_1) \cup nbs(G_2) \cup \ldots \cup nbs(G_r)$.

The idea is to use these lemmas to compute for all internal nodes in T_G, the nimber and nimberset of the corresponding cograph. It is helpful for decreasing the running time of this computation, when we know what the maximum nimber is that a cograph with n vertices can attain.

Let $s(K)$ denote the minimum number of vertices of a cograph G with nimber at least 2^K. We will show that $s(K) = 3^K$.

Lemma 6.4 *For all* $K \geq 0$, $s(K) \leq 3^K$.

Proof: We give a series of cographs, H_0, H_1, H_2, ..., with H_K containing exactly 3^K vertices, and $nbs(H_K) = \{0, 1, 2, \ldots, 2^K - 1\}$, and hence $nb(H_K) = 2^K$.

For H_0, take a graph with one vertex and no edges. For $K \geq 1$, take $H_K = (H_{K-1} \cup H_{K-1}) \times H_{K-1}$. With lemma 6.2, one easily verifies with induction that H_K fulfills the conditions mentioned above. □

Theorem 6.5 *For all* $K \geq 0$, $s(K) = 3^K$.

The proof of theorem 6.5 is given in Appendix A.

Corollary 6.6 *For every cograph* $G = (V, E)$ *with* n *vertices,* $nb(G) < 2n^{1/\log 3}$.

Proof: Take the largest K with $2^K \leq nb(G)$. By theorem 6.5, $n \geq 3^K$, hence $nb(G) < 2 \cdot 2^K = 2 \cdot 3^{(1/\log 3)K} \leq 2 \cdot n^{1/\log 3}$. □

Note that $1/\log 3 \approx 0.63093$.

We now give an algorithm that solves Kayles on cographs. We suppose that cograph G is given together with its cotree T_G. For each node i in T_G, let G_i denote the cograph corresponding with this node, and write $nb(i) = nb(G_i)$, and $nbs(i) = nbs(G_i)$. Let $z = \lfloor 2n^{1/\log 3} \rfloor$. To store $nbs(i)$ for each node i in T_G, associate with each node i in T_G a boolean array with entries for all nimbers 0, 1, ..., z. The algorithm computes $nbs(i)$ and $nb(i)$ for all nodes i in T_G. For easier presentation, we state the algorithm as a recursive procedure, which is called with compute_nimbersets(r), with r the root of T_G.

```
procedure compute_nimbersets(node i):
begin if i is a leaf of T_G
      then begin nbs(i) := {0};
```

```
            nb(i) := 1;
        end;
    else begin Suppose the children of i are j_1, ..., j_r;
            helpnbs := nbs(j_1);
            helpnb := nb(j_1);
            if label(i) = 0
            then for k := 2 to r do
                begin helpnbs := nb(j_k) ⊕ helpnbs ∪ helpnb ⊕ nbs(j_k);
                      helpnb := mex(helpnbs);
                end;
            else begin for k := 2 to r
                      do helpnbs := helpnbs ∪ nbs(j_k);
                      helpnb := mex(helpnbs);
                 end;
            nbs(i) := helpnbs;
            nb(i) := helpnb;
        end;
end.
```

Correctness follows from lemmas 6.2 and 6.3. Taking the union of two sets of nimbers, and taking the $\oplus$-sum of a nimber and a set of nimbers can be done in $O(z)$ time. As T_G has n leaves, and hence $\leq n-1$ edges, a linear number of these operations is done. Hence, the total time of the algorithm is bounded by $O(nz) = O(n^{1+1/\log 3}) = O(n^{1.631})$.

Theorem 6.7 *Kayles can be solved on cographs in $O(n^{1+1/\log 3}) = O(n^{1.631})$ time.*

7 Kayles on permutation graphs

In this section we show that Kayles can be solved in $O(n^3)$ time on permutation graphs. The method is more or less similar to the method used for interval graphs.

One can show that the number of isolatable subsets of a permutation graph is bounded by $O(n^4)$. We do not need this fact, and do not prove it here.

We suppose that we have a permutation graph $G = (V, E)$, with $V = \{1, 2, \ldots, n\}$, and a permutation $\pi : \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\}$, such that $E = \{(v, w) \mid v, w \in V,\ (v < w \wedge \pi(v) > \pi(w)) \vee (v > w \wedge \pi(v) < \pi(w))\}$.

For $1 \leq i \leq j \leq n$, denote $V_{ij} = \{v \in V \mid i < v < j \text{ and } \pi(i) < \pi(v) < \pi(j)\}$. Note that $V_{0(n+1)} = V$, and $V_{ii} = V_{i(i+1)} = \emptyset$. Write $G_{ij} = G[V_{ij}]$.

The algorithm in this section is based on computing the nimbers $nb(G_{ij}$, for all i, j, $0 \leq i \leq j \leq n+1$, with dynamic programming.

Lemma 7.1 *Let $v \in V_{ij}$, $0 \leq i \leq j \leq n+1$. The connected components of $G[V_{ij} - N(v)]$ are G_{iv} and G_{vj}.*

Proof: From the definitions it follows that $V_{ij} - N(v) = V_{iv} \cup V_{vj}$. Further note that $V_{iv} \cap V_{vj} = \emptyset$, and no vertex in V_{iv} can be adjacent to a vertex in V_{vj}. □

From this lemma, and theorem 4.1, the following lemma can be derived directly:

Lemma 7.2 *Let* $0 \leq i \leq j \leq n+1$.
(i) For all $v \in V_{ij}$: $nb(G[V_{ij} - N(v)]) = nb(G_{iv}) \oplus nb(G_{vj})$.
(ii) $nb(G_{ij}) = mex\{nb(G_{iv}) \oplus nb(G_{vj}) \mid v \in V_{ij}\}$.

This lemma suggests the following dynamic programming algorithm.

for $i := 0$ **to** $n+1$ **do** $nb(G_{ii}) := 0$;
for $i := 0$ **to** n **do** $nb(G_{i(i+1)}) := 0$;
for $i := 0$ **to** $2n-1$ **do**
begin for $j := 1$ **to** $2n - i$ **do**
    begin $S := \emptyset$;
        for all $v \in V$ with $j < v < j+i$ and $\pi(j) < \pi(v) < \pi(j+i)$
        do $S := S \cup \{nb(G_{jv}) \oplus nb(G_{v(j+i)})\}$;
        $nb(G_{j(j+i)}) := mex(S)$;
    end
end;
output $nb(G_{0(n+1)})$.

Again, by theorem 2.1, player 1 has a winning strategy, if and only if $nb(G_{0(n+1)}) > 0$. The algorithm clearly takes $O(n^3)$ time.

Theorem 7.3 *Kayles can be solved in* $O(n^3)$ *time on permutation graphs.*

8 Final remarks

In this paper, we obtained polynomial time algorithms for Kayles, when restricted to either interval graphs, circular arc graphs, permutation graphs, or cographs. For several other interesting classes of graphs, the complexity of Kayles is still open. Probably the most notable of these classes is the class of trees. Already in 1978, Schaefer mentioned as an open problem the complexity of Kayles, when restricted to trees where only one vertex has degree at least three [8]. To the authors best knowledge, this problem is still unresolved.

Acknowledgements

I like to thank Mark de Berg, Goos Kant, Ton Kloks, and Marc van Kreveld for several very helpful ideas and comments.

References

[1] E. Berlekamp, J. Conway, and R. Guy. *Winning Ways for your mathematical plays, Volume 1: Games in General.* Academic Press, New York, 1982.

[2] K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using *pq*-tree algorithms. *J. Comp. Syst. Sc.*, 13:335–379, 1976.

[3] J. Conway. *On Numbers and Games.* Academic Press, London, 1976.

[4] D. G. Corneil, Y. Perl, and L. K. Stewart. A linear recognition algorithm for cographs. *SIAM J. Comput.*, 4:926–934, 1985.

[5] M. Dietzfelbinger, A. Karlin, K. Mehlhorn, F. Meyer auf der Heide, H. Rohnert, and R. E. Tarjan. Dynamic perfect hashing: Upper and lower bounds. In *Proceedings of the 29th Annual Symposium on Foundations of Computer Science*, pages 524–531, 1988.

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

[7] D. König. *Theorie der endlichen und unendlichen Graphen.* Akademische Verlagsgesellschaft, Leipzig, 1936.

[8] T. J. Schaefer. On the complexity of some two-person perfect-information games. *J. Comp. Syst. Sc.*, 16:185–225, 1978.

[9] A. Tucker. An efficient test for circular-arc graphs. *SIAM J. Comput.*, 9:1–24, 1980.

[10] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1984.

A Proof of theorem 6.5

Here we show that for all $K \geq 0$, $s(K) \geq 3^K$. As it is shown in lemma 6.4 that $s(K) \leq 3^K$, theorem 6.5 follows.

First observe that $s(0) = 1$, as the empty graph has nimber 0, and a graph with one vertex has nimber 1. The graph with two vertices and no edges has nimber 0, and the graph with two vertices and one edge has nimber 1. Hence $s(1) \geq 3$. We will now show that for all $K \geq 1$, $s(K+1) \geq 3 \cdot s(K)$. With induction, the result then follows.

Suppose $G = (V, E)$ is a cograph with nimber $\geq 2^{K+1}$, with minimum size of $n = |V|$. Let T_G be the corresponding cotree. We write G_i for the cograph, corresponding with node i in T_G.

First observe, that if for any node i in T_G, $2^{K+1} \in nbs(G_i)$, then G_i contains a subgraph with nimber 2^{K+1}. The size of this subgraph is smaller than the size of G, hence G was not of minimum size. Thus we may assume that $nbs(G) = \{0, 1, 2, \ldots, 2^{K+1} - 1\}$, and for all $i \in I$: $nbs(G_i) \subseteq \{0, 1, 2, \ldots, 2^{K+1} - 1\}$.

We say that a node $i \in T_G$ is *K-heavy*, if $nb(G_i) \geq 2^K$ or $nbs(G_i) \cap \{2^K, 2^K + 1, \ldots, 2^{K+1} - 1\} \neq \emptyset$. A node $i \in T_G$ is *K-precise*, if $nbs(G_i) = \{0, 1, 2, \ldots, 2^K - 1$. Note that for a K-precise node i, $nb(G_i) = 2^K$.

Claim A.1 *If i_0 is K-heavy, then i_0 is K-precise, or i_0 has a descendant that is K-precise.*

Proof: From lemma 6.3 it easily follows that every K-heavy node either is K-precise, or has a K-heavy child. □

As the root r of T_G is K-heavy, it follows that there must be at least one K-precise node in T_G. Note that if a K-precise node i has a descendant $j \neq i$ that is also a K-precise node, then G is not minimal: use the cograph corresponding to the cotree, obtained by replacing the subtree rooted at i in T_G by the subtree rooted at j. So assume no K-precise node has a descendant which is also K-precise.

Claim A.2 *There are at least two K-precise nodes in T_G.*

Proof: Suppose that i is the only K-precise node in T_G. Then the only K-heavy nodes in T_G are the nodes on the path from T_G to root r. With induction, one can prove that for each node j on this path, $\{0, 1, 2, \ldots, 2^K - 1\} \subseteq nbs(G_j)$. (Use lemma 6.3, and note that only one term contains the binary factor 2^K.) Hence, if a predecessor j_0 of i is 1-labeled, it follows that for the unique K-heavy child j_1 of j_0, $nbs(G_{j_1}) = nbs(G_{j_0})$. Hence G was not of minimum size, contradiction.

So we may assume that i has exactly one predecessor, namely r, which is labeled with a 0. Hence we can write $G = G_i \cup H$. (H is the union of all cographs, corresponding to the other childs of r.) $nbs(G_i) = \{0, 1, 2, \ldots, 2^K - 1\}$, $nbs(H) \subseteq \{0, 1, 2, \ldots, 2^K - 1\}$. If $nb(H) < 2^K$, then $2^K + nb(H) \notin nbs(G_i \cup H)$, hence $nb(G) < 2^{K+1}$. So $nb(H) = 2^K$. Applying lemma 6.2 it follows that $nbs(G_i \cup H) = \{2^K, 2^K + 1, 2^K + 2, \ldots, 2^{K+1} - 1\}$, contradiction. □

If there are at least three K-precise nodes, then note that each of these must have at least $s(K)$ leaf-descendants. Hence $s(K+1) \geq 3 \cdot s(K)$. Assume now there are exactly two K-precise nodes, i_0 and i_1. Let i_2 be the lowest common ancestor of i_0 and i_1. Similar as above, we can argue that G is not of minimum size, if a node between i_0 and i_2 or a node between i_1 and i_2 has a label 1, and if i_2 has label 1, then it has exactly two children, which are both K-heavy. (Each subtree rooted at one of these two children contains exactly one of i_0 and i_1, in this case.)

We consider now two cases, namely that i_2 has label 0, and that i_2 has label 1.

Case 1. i_2 has label 0. Then G_{i_2} can be written as $G_{i_2} = G_{i_0} \cup G_{i_1} \cup G_{j_1} \cup \ldots G_{j_r}$. Write $H = G_{j_1} \cup \ldots \cup G_{j_r}$. Note that $nbs(H) \subseteq \{0, 1, 2, \ldots, 2^K - 1\}$, as none of

the nodes $j_1, \ldots, j_r$ is K-heavy. If $nb(H) = 2^K$, then $nbs(G_{i_2}) = \{0, 1, \ldots, 2^K - 1\}$ and i_2 is K-precise, contradiction. So $nb(H) < 2^K$, and it follows that $nbs(G_{i_2}) = \{2^K, 2^K + 1, \ldots, 2^{K+1} - 1\} \cup nbs(H)$.

Claim A.3 *Let j be a node in T_G on the path from i_2 to r. Let H_j be the subgraph of G_j, obtained by removing all leaf-descendants of i_0 and of i_1 from G_j. Then $nbs(G_j) = nbs(H_j) \cup \{2^K, 2^K + 1, \ldots, 2^{K+1} - 1\}$.*

Proof: With induction. For $j = i_2$, the claim holds, as is argued above. Suppose the claim holds for the K-heavy child j' of j. Note that none of the other children of j is K-heavy. We must have that $nb(G_{j'}) = nb(H_{j'}) < 2^K$, otherwise $nb(G_{j'}) = 2^{K+1}$, which contradicts the minimality of G.

Write $G_j = G_{j'} \cup H$, or $G_j = G_{j'} \times H$. If j is labeled with a 0, then $H_j = H_{j'} \cup H$, and $nbs(G_j) = nb(G_{j'}) \oplus nbs(H) \cup nb(H) \oplus nbs(H_{j'}) \cup nb(H) \oplus \{2^K, 2^K + 1, \ldots, 2^{K+1}\} = nb(H_{j'}) \oplus nbs(H) \cup nb(H) \oplus nbs(H_{j'}) \cup \{2^K, 2^K + 1, \ldots, 2^{K+1}\} = nb(H_{j'} \cup H)\{2^K, 2^K + 1, \ldots, 2^{K+1}\}$. If j is labeled with a 1, then $H_j = H_{j'} \times H$ and $nbs(G_j) = nbs(G_{j'}) \cup nbs(H) = nbs(H_{j'}) \cup \{2^K, 2^K + 1, \ldots, 2^{K+1} - 1\} \cup nbs(H) = nb(H_{j'} \times H) \cup \{2^K, 2^K + 1, \ldots, 2^{K+1}\}$. □

In particular, we have for the root r of T_G that $\{0, 1, 2, \ldots, 2^{K+1} - 1\} = nbs(G_r) = nbs(H_r) \cup \{2^K, 2^K + 1, \ldots, 2^{K+1} - 1\}$. Hence $nb(H_r) \geq 2^K$. So, if we remove all leaf-descendants of i_0 and i_1 from G, we remain with a graph with nimber at least 2^K, hence with a graph with at least $s(K)$ vertices. As both G_{i_0} and G_{i_1} contain at least $s(K)$ vertices, G contains at least $3 \cdot s(K)$ vertices. This ends the analysis of case 1.

Case 2. i_2 has label 1. Then G_{i_2} can be written as $G_{i_2} = (G_{i_0} \cup G_{j_1} \cup \ldots G_{j_r}) \times (G_{i_1} \cup G_{k_1} \cup \ldots G_{k_s})$. Write $H = G_{j_1} \cup \ldots G_{j_r}$, and $K = G_{k_1} \cup \ldots G_{k_r}$. Calculation shows that $nbs(G_{i_2}) = \{0, 1, 2, \ldots, 2^K - 1\} \cup 2^K \oplus nbs(H) \cup 2^K \oplus nbs(K)$.

Consider the graph $G' = G_{i_0} \cup (H \times K)$. There are two cases:

Case 2.1. $nb(H \times K) = 2^K$. Then G contains at least $3 \cdot s(K)$ vertices: H and K together contain at least $s(K)$ vertices and are disjoint from G_{i_0} and G_{i_1}, which both also contain at least $s(K)$ vertices.

Case 2.2. $nb(H \times K) < 2^K$. As none of the nodes $j_1, \ldots, j_r$, $k_1, \ldots, k_s$ is K-heavy, it follows that $nbs(G') = nb(H \times K) \oplus nbs(G_{i_0}) \cup nb(G_{i_0}) \oplus nbs(H \times K) = nb(H \times K) \oplus \{0, 1, 2, \ldots, 2^K - 1\} \cup 2^K \oplus nbs(H) \cup 2^K \oplus nbs(K) = nbs(G_{i_2})$. Now let G'' be the cograph, that corresponds to the cotree that is obtained by replacing in T_G the subtree rooted at i_2 by the cotree $T_{G'}$ of G'. The nimberset for i_2 does not change under this replacement operation, and hence $nb(G'') = nb(G)$. But G'' has fewer vertices than G, contradiction. This ends the proof of case 2, and of theorem 6.5.