Exact Algorithms for Kayles

Hans L. Bodlaender

Dieter Kratsch

Technical Report UU-CS-2011-003
February 2011

Department of Information and Computing Sciences
Utrecht University, Utrecht, The Netherlands
www.cs.uu.nl

ISSN: 0924-3275

Department of Information and Computing Sciences
Utrecht University
P.O. Box 80.089
3508 TB Utrecht
The Netherlands

Exact Algorithms for Kayles [*]

Hans L. Bodlaender [†] Dieter Kratsch [‡]

Abstract

In the game of Kayles, two players select alternatingly a vertex from a given graph G, but may never choose a vertex that is adjacent or equal to an already chosen vertex. The last player that can select a vertex wins the game. In this paper, we give an exact algorithm to determine which player has a winning strategy in this game. To analyse the running time of the algorithm, we introduce the notion of K-set: a nonempty set of vertices $W \subseteq V$ is a K-set in a graph $G = (V, E)$, if $G[W]$ is connected and there exists an independent set X such that $W = V - N[X]$. The running time of the algorithm is bounded by a polynomial factor times the number of K-sets in G. We show that the number of K-sets in a graph with n vertices and m edges is bounded by $O(1.6052^n)$, and thus we have an algorithm for Kayles with running time $O(1.6052^n)$. We also show that the number of K-sets in a tree is bounded by $n \cdot 3^{n/3}$ and thus Kayles can be solved on trees in $O(1.4423^n)$ time. We show that apart from a polynomial factor, the number of K-sets in a tree is sharp.

1 Introduction

When a problem is computationally hard, then there still are many situations in which the need can arise to solve it exactly. This motivates the field of exact algorithms, where exact, exponential time algorithms whose running time is as small as possible are sought. Many such exact algorithms have been designed and analysed for problems that are NP-complete or $\#P$-complete, see [6]. Of course, also problems that are complete for a 'harder' complexity class, e.g., PSPACE-complete often ask for exact solutions. Many PSPACE-complete problems arrive from the question which player has a winning strategy for a given position in a combinatorial game. Exact algorithms are of great relevance here, e.g., a program could use a heuristic to find a move, but once a position is simple enough, it switches to an exact algorithm to give optimal play in the endgame.

In this paper, we study exact algorithms for one such PSPACE-complete problem, namely the problem to determine which player has a winning strategy in an given instance of the game Kayles. Kayles is a two-player game that is played on a graph $G = (V, E)$.

[*]The work of the second author was supported by the ANR project AGAPE.

[†]Utrecht University, P.O. Box 80.089, 3508 TB Utrecht, the Netherlands. hansb@cs.uu.nl

[‡]Université Paul Verlaine – Metz, LITA, 57045 Metz Cedex 01, France. kratsch@univ-metz.fr

Alternatingly, the players choose a vertex from the graph, but players are not allowed to choose a vertex that already has been chosen or is adjacent to a vertex that already has been chosen. Thus, the player build together an independent set in G. The last player that chooses a vertex (i.e., turns the independent set into a maximal independent set) wins the game. Alternatively, one can describe the game as follows: the chosen vertex and its neighbors are removed and a player wins when his move empties the graph. The problem to determine the winning player for a given instance of the game is also called KAYLES. This problem was shown to be PSPACE-complete by Schaefer [9]. In an earlier paper [3], we showed that by exploiting Sprague-Grundy theory, KAYLES can be solved in polynomial time on several special graph classes, in particular graphs with a bounded asteroidal number (which includes well known classes of graphs like interval graphs, cocomparability graphs and cographs). Fleischer and Trippen [5] showed that KAYLES can be solved in polynomial time on stars of bounded degree, and also analysed this special case experimentally. For general trees, the complexity of KAYLES is a long standing open problem. Variants of the game on paths were studied and shown to be linear time solvable by Guignard and Sopena [7]. For more background, the reader can consult [1, 2, 4].

It is not hard to find an algorithm that solves KAYLES in $O^*(2^n)$ time, by tabulating for each induced subgraph of G which player has a winning strategy from that position. In this paper, we improve upon this trivial algorithm, and give an algorithm that uses $O(1,6052^n)$ time. The algorithm uses ideas from [3], exploiting results from Sprague-Grundy theory (also known as the theory of *nimbers*). To analyze the running time of the algorithm, we introduce the notion of K-set: a set of nonempty vertices $W \subseteq V$ is a K-set in a graph $G = (V, E)$, if $G[W]$ is connected and there exists an independent set X such that $W = V - N[X]$. With a nontrivial analysis we obtain that the number of K-sets of a graph with n vertices is bounded by $O(1.6052^n)$, which yields the bound on the running time of our algorithm. We also show that if G is a tree, then G has at most $n \cdot 3^{n/3}$ K-sets, and thus, KAYLES can be solved in $O^*(3^{n/3}) = O(1.4423^n)$ time on trees (and forests). [1] We also give lower bounds for the number of K-sets. In particular, our bound of $3^{n/3}$ K-sets for trees is sharp except for polynomial terms.

2 Preliminaries

Graph terminology Throughout this paper all graphs $G = (V, E)$ are undirected and simple. Let $S \subseteq V$. Then $N[S] = \cup_{s\in S} N[s]$ is the *closed neighborhood* of S, $N(S) = N[S] \setminus S$ is the *open neighborhood* of S, and $G[S]$ denotes the *subgraph of G induced by S*.

A nonempty set of vertices $W \subseteq V$ of a graph $G = (V, E)$ is called a *K-set* (*Kayles set*) of G, if it fulfills each of the following criteria:

- $G[W]$ is connected
- there exists an independent set $X \subseteq V$ such that $W = V - N(X)$

[1] We use the so called O^* notation: $f(n) = O^*(g(n))$ if $f(n) = O(g(n)p(n))$ for some polynomial $p(n)$. See also [6].

Sprague-Grundy theory Next, we review some notions and results from Sprague-Grundy theory, and give some preliminary results on how this theory can be used for Kayles. For a good introduction to Sprague-Grundy theory, the reader is referred to [1, 4].

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

To each position in a two player game that is finite, deterministic, full-information, impartial, and with 'last player wins rule', 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 1 *[1, 4] 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 $nb(p)$. Given two (finite, deterministic, impartial, ...) games $\mathcal{G}_1$, $\mathcal{G}_2$, the *sum* of $\mathcal{G}_1$ and $\mathcal{G}_2$, denoted $\mathcal{G}_1 + \mathcal{G}_2$ is the game where a move consists of choosing $\mathcal{G}_1$ or $\mathcal{G}_2$ and then making a move in that game. A player that cannot make a move in $\mathcal{G}_1$ nor in $\mathcal{G}_2$ looses the game $\mathcal{G}_1 + \mathcal{G}_2$. 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 $(i = 1, 2)$.

The binary XOR operation is denoted by $\oplus$, i.e., for nimbers i_1, i_2, $i_1 \oplus i_2 = \sum\{2^j \mid (\lfloor i_1/2^j \rfloor$ is odd) $\Leftrightarrow$ $(\lfloor i_2/2^j \rfloor$ is even)$\}$.

Theorem 2 *[1, 4] 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 $nb((p_1, p_2)) = nb(p_1) \oplus nb(p_2)$.*

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 every 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.

An important observation is the following: when $G = G_1 \cup G_2$ for disjoint graphs G_1 and 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, we have the following result.

Lemma 3 *$nb(G_1 \cup G_2) = nb(G_1) \oplus nb(G_2)$.*

Note that G_1 and G_2 might be disconnected graphs.

Our second observation shows how to express the nimber of a graph G in the nimbers of some subgraphs of G. Consider Kayles, played on $G = (V, E)$, and suppose that a vertex $v \in V$ is played. Then, the nimber of the resulting position is the same as the nimber of $G - N[v]$, as the effect of playing on v is the same as the effect of removing v and its neighbors from the graph. As the nimber of a position is the minimum nimber that is not in the set of nimbers of positions that can be reached in one move, we have:

Lemma 4 *(i) If $G = (V, E)$ is the empty graph, then $nb(G) = 0$.*
(ii) If $G = (V, E)$ is not the empty graph, then $nb(G) = mex(nb(\{G - N[v] \mid v \in V\})$.

3 An Upper Bound on the Number of K-sets

In this section, we will show an upper bound on the number of K-sets in a graph. This bound is needed for the analysis of our algorithm, see Section 5. Our main result is the following.

Theorem 5 *Let G be a graph with n vertices. Then G has $O(1,6052^n)$ K-sets.*

The proof of Theorem 5 is algorithmic: we give a branching procedure that generates all K-sets. By distinguishing different types of vertices, assigning these different weights, and considering the different branching vectors, we obtain a set of recurrences, whose solution gives us the desired bound. For information on branching algorithms and their analysis, in particular branching vectors and the corresponding recurrences we refer to [6].

We say that a K-set is *nontrivial*, if it has at least three vertices; otherwise we call it trivial. As each trivial set either consists of a single vertex or the two endpoints of an edge, the number of trivial K-sets is at most $n+m$, where m is the number of edges of the graph.

During our branching process, we decide at some points to put some vertices in an independent set X and forbid for some vertices to put them in the independent set. When placing a vertex in X, we say we *select* the vertex. The vertices in G are of four types:

- **White** or free vertices. Originally all vertices in G are white. We have not made any decision yet for a white vertex. All white vertices have weight one.
- **Red** vertices. Red vertices may not be placed in the independent set X: i.e., we already decided this during the branching. It still is possible that a red vertex becomes deleted later, however. Red vertices have a weight $\alpha = 0,5685$.
- **Green** vertices. A green vertex is 'safe': it never will be removed. I.e., we cannot place the green vertex nor any of its neighbors in the independent set X. Green vertices have weight zero.
- **Removed vertices**: these are either placed in the independent set or are a neighbor of a vertex in the independent set. All removed vertices have weight zero. Removed vertices are considered not existing, i.e., when discussing the neighbors of a vertex, these neighbors will be white, red, or green.

The *measure* of an instance G is the total weight of all vertices, and the difference in the measure from an instance to one of a subproblem often called *gain* is used to analyse the branching algorithm via branching vectors. Our branching process may be overcounting the number of K-sets (in particular, in some cases, we will not detect that a generated set is not connected), but the obtained bound nevertheless is valid as an upper bound.

The semantics of the colors imply that we can always perform the following actions:

- **Rule 1**: If a red vertex v has no white neighbors, we can color it green. This is valid, as we can no longer place a neighbor of v in X.

- **Rule 2**: If a green vertex v has a white neighbor w, we can color w red. This is valid, as placing w in X would remove v, which we are not allowed by the green color of v.

Rules 1 and 2 will always decrease the measure. They ensure that each red vertex will have a white neighbor, and that white vertices have no green neighbors.

The following action also can always be performed; the removed vertices can no longer be part of a nontrivial K-set.

- **Rule 3**: If $W \subseteq V$ is a set of white vertices that are not incident to nondeleted vertices not in W, and $|W| \leq 2$, then remove all vertices of W.

Before starting the main recursive branching, we first fix one vertex $v_0 \in V$, of which we will assume that it is an element of the K-set. In terms of colors, this means that we color v_0 green and all neighbors of v_0 red. Clearly, the total number of K-sets will be at most n times the bound on the number of K-sets that contain a specific vertex.

We obtain a fourth rule.

- **Rule 4**: If G has more than one connected component, then remove all vertices from components that do not contain v_0.

As a consequence, we have that when no rule can be applied and there is at least one white vertex, then there exists a white vertex that is adjacent to a red vertex.

We consider two main types of branching. The first type of branching is a *vertex branch*. Let $v \in V$ be a white vertex. We consider two cases: v is placed in X, and v is not placed in X. In the former case, we remove and decrease the measure by the total weight of all white and red vertices in the closed neighborhood of v. In the latter case, we color v red and have a measure decrease of $1 - \alpha$. In some cases, we gain more by applications of Rules 1, 2, and 3.

In the second type of branching, we consider a number of cases, of which one must apply. Again, in some cases, we can gain more by applications of Rules 1, 2, and 3.

In the sequel we present all branching rules in a preference order. Hence when Case i branching is applied to an instance all earlier cases do not apply.

Case 1: There is a white vertex with at least three white neighbors If v has three white neighbors, we can perform a vertex branch on v. The branching vector in this case will be $(4, 1 - \alpha)$, i.e., in one case, we decrease the measure by at least four, and in the other case, we decrease the measure by $1 - \alpha$.

Case 2: There is a white vertex with two white neighbors and at least one red neighbor If v has two white neighbors and at least one red neighbor, then a vertex branch on v gives a branching vector of $(3 + \alpha, 1 - \alpha)$.

Suppose Cases 1 and 2 cannot be applied anymore. Then all white vertices have at most two white neighbors. Moreover, there cannot be a cycle of white vertices, as such a cycle would either be removed by Rule 4 or contains a vertex to which Case 2 applies. Similar for white vertices forming paths. Only the endpoints of such a path can be adjacent to a red vertex, and at least one endpoint is adjacent to a red vertex.

Case 3: The subgraph induced by white vertices contains a path of length at least two, with both endpoints incident to at least one red vertex. Suppose now we have a path of white vertices $v_1, \ldots, v_r$, $r \geq 2$, with v_1 and v_r incident to a red vertex. As Case 2 no longer applies, we can assume that $v_2, \ldots, v_{r-1}$ have no nondeleted neighbors outside the path.

Let R be the set of red vertices that are adjacent to v_1 and/or v_r.

Case 3.1: $r = 2$ We must either select v_1, or select v_2, or select neither v_1 nor v_2. In the latter case, both v_1 and v_2 can be colored green, so the measure is decrease by two in this case. Hence, we have a branching vector $(2 + \alpha, 2 + \alpha, 2)$.

Case 3.2: $r = 3$ **and** $|R| \geq 2$ We consider all cases of placing vertices from $\{v_1, v_2, v_3\}$ in X:

- Select v_1 and v_3: we decrease the measure by $3 + 2 \cdot \alpha$.
- Select v_1: we decrease the measure by $3 + \alpha$.
- Select v_2: we decrease the measure by 3.
- Select v_3: we decrease the measure by $3 + \alpha$.
- Choose none: we decrease the measure by 3. (All three vertices can be colored green.)

So, in this case, we obtain a branching vector $(3 + 2 \cdot \alpha, 3 + \alpha, 3, 3 + \alpha, 3)$.

Case 3.3: $r = 3$ **and** $|R| = 1$ The vertices v_1 and v_3 have a common red neighbor. Now, we can perform a vertex branch on v_1. If we select v_1, then v_3 becomes an isolated vertex, and thus we have a branching vector of $(3 + \alpha, 1 - \alpha)$.

Case 3.4: $r = 4$ **and** $|R| \geq 2$ Like in Case 3.2, we consider all cases of placing vertices from $\{v_1, v_2, v_3, v_4\}$ in X, and obtain a somewhat tedious case analysis. In each case, each vertex in $\{v_1, v_2, v_3, v_4\}$ either is removed or is green. If v_1 or v_4 is placed in X, we gain an additional α for the removal of the red neighbor of this vertex. In case we select both v_1 and v_4, we gain $2 \cdot \alpha$; here we use that $|R| \geq 2$. This gives a branching vector of $(4 + \alpha, 4 + 2 \cdot \alpha, 4 + \alpha, 4 + \alpha, 4, 4, 4 + \alpha, 4)$, corresponding to selecting $\{v_1, v_3\}$, $\{v_1, v_4\}$, $\{v_2, v_4\}$, $\{v_1\}$, $\{v_2\}$, $\{v_3\}$, $\{v_4\}$ or no vertex from this path for inclusion in X.

Case 3.5: $r = 4$ **and** $|R| = 1$ We do a vertex branch on v_1: if we select v_1, then Rule 3 will remove v_3 and v_4. So the branching vector is $(4 + \alpha, 1 - \alpha)$.

Case 3.6: $r \geq 5$ We branch as follows:

- v_1 is placed in X: we decrease the measure by $2 + \alpha$.
- v_2 is placed in X: we decrease the measure by 3.
- v_3 is placed in X and v_1 is not placed in X. v_1 can be colored green, and thus we decrease the measure by 4.
- v_4 is placed in X and v_1 and v_2 are not placed in X. v_1 and v_2 can be colored green, and thus we decrease the measure by 5.
- None of v_1, v_2, v_3, v_4 is placed in X. v_1, v_2, v_3 become green, and v_4 becomes red: a measure decrease of $4 - \alpha$.

Thus, the branching vector is $(2 + \alpha, 3, 4, 5, 4 - \alpha)$.

Case 4: The subgraph induced by white vertices contains a path of length at least two, with exactly one endpoint incident to a red vertex Suppose $v_1, \ldots, v_r$ is a path of white vertices, and suppose $r \geq 2$ is maximal. Assume without loss of generality that v_1 has a red neighbor, say w.

Case 4.1: $r \geq 3$ We do a vertex branch on v_{r-2}. If we select v_{r-2} then we gain at least $3 + \alpha$: if $r \geq 4$, then v_{r-2} has two white neighbors, and if $r = 3$, then v_{r-2} has a white neighbor (v_{r-1}) and a red neighbor (w). Moreover, v_r becomes an isolated vertex after v_{r-2} is placed in the independent set, and thus is removed by Rule 3. If we do not select v_{r-2}, we gain $1 - \alpha$, and thus we have a branching vector of $(3 + \alpha, 1 - \alpha)$.

Case 4.2: $r = 2$ **and** w **has a white neighbor** $x \neq v_1$ We can now perform a vertex branch on x. If we place x in the independent set, then w and x are removed, but also v_1, v_2 are removed as Rule 3 can be applied: they form a connected component of at most two white vertices. So, the measure is decreased by at least $3 + \alpha$. If we do not select x, we color x red so obtain a measure decrease of $1 - \alpha$. So, this case gives a $(3 + \alpha, 1 - \alpha)$ branching vector.

Case 4.3: $r = 2$ **and** w **has no white neighbor** We either must select v_1, or we select v_2, or we select neither v_1 or v_2. If we select v_2, then w can be colored green, as its only white neighbor v_1 is removed. If we select neither v_1 nor v_2, then w, v_1 and v_2 can be colored green, so we decrease the measure $2 + \alpha$ in this case. So we obtain a branching vector of $(2 + \alpha, 2 + \alpha, 2 + \alpha)$.

If Cases 1 – 4 cannot be applied, then there are no adjacent white vertices. The remaining cases thus deal with white vertices that have no white neighbors. If a white vertex has no red neighbors, then it is removed by Rule 3, so we assume that each white vertex has at least one red neighbor but no other neighbors.

Case 5: v_1 is a white vertex with no white but at least two red neighbors We do a vertex branch on v_1. If we do not select v_1, it can be colored green, by Rule 1. So we obtain a branching vector $(1 + 2 \cdot \alpha, 1)$.

Case 6: v_1 is a white vertex with exactly one neighbor, which is red Let w be the red neighbor of v_1.

Case 6.1: w has a white neighbor $x \neq v_1$ If a white neighbor of w has at least two red neighbors, then we can deal with it as in Case 5, and obtain a branching vector of $(1 + 2 \cdot \alpha, 1)$. So suppose all white neighbors of w have degree one, and thus w is their unique neighbor. We now have the following branch:

- w is a vertex in the K-set. In this case, w and all white neighbors of w are colored green. So, the measure decreases by at least $2 + \alpha$.
- w is not a vertex in the K-set. In this case, we must place *all* white neighbors of w in the independent set X. Again, the measure decreases by at least $2 + \alpha$.

So, we obtain a $(2 + \alpha, 2 + \alpha)$ branching vector.

Case 6.2: v_1 is the unique white neighbor of w In this case, we do a vertex branch on v_1. If we do not place v_1 in the independent set, then both v_1 and w can be colored green. So, the branching vector is $(1 + \alpha, 1 + \alpha)$.

If no case applies, then there are no white, and hence also no red vertices left, so we found one (or zero, in case the green vertices are not connected) K-set. Our choice of $\alpha = 0,5685$ gives the best value for the base of the exponent for the given branching vectors, namely the claimed $1,6052$. Thus, it follows that there are $O(1.6052^n)$ nontrivial K-sets that contain v_0. As the value 1.6052 is obtained by rounding, and there are at most $n + m$ trivial K-sets, the result follows.

4 A Bound on the Number of K-sets in Trees

In this section, we establish an upper bound on the number of K-sets in a tree. This bound is used in Section 5 to show a bound on the running time of our algorithm, when the input graph is a tree or a forest.

Theorem 6 *Let $T(n)$ be the maximum number of K-sets in a tree on n nodes. Then $T(n) \leq n \cdot 3^{n/3}$.*

Proof. We denote as a *rooted K-set* of a rooted tree T any K-set of T containing r, where r denotes the root of T. Let $R(n)$ be the maximum number of rooted K-sets in any rooted tree on n nodes. We claim that $R(n) \leq 3^{n/3} - 1$ for all $n \geq 2$.

We are going to prove this claim by induction. To see that the claim is true for the base case $n = 2$, note that the only K-set containing r is the one containing both nodes of the tree, and that $3^{2/3} - 1 > 1.08$.

As induction hypothesis let us assume that the claim is true for all $n' < n$ and consider any rooted tree T on $n > 2$ nodes. Let r be the root of the tree and $u_1, u_2, \dots, u_p$ be the children of v. For every $i = 1, 2, \dots, p$, let T_i be the subtree of T rooted at u_i. Furthermore for all $i = 1, 2, \dots, p$, we denote by n_i the number of nodes of T_i.

Let W be any K-set of T containing its root r. Then for every i, the intersection of W with T_i is either empty or a K-set of T_i containing its root u_i. Note that $n_i = 1$ implies that W also contains u_i since $r \in W$ (and thus r cannot be taken into the independent set X generating W). Using the induction hypothesis and $\sum_{i=1}^{p} n_i = n - 1$, we establish the following upper bound for the number of rooted K-sets of a rooted tree on n nodes

$$\begin{aligned} R(n) &\le \prod_{i:n_i \ge 2} (R(n_i) + 1) \le \prod_{i:n_i \ge 2} 3^{n_i/3} \\ &\le \prod 3^{(n-1)/3} \le 3^{n/3} - 1. \end{aligned}$$

This completes the proof of our claim.

To complete the proof of the theorem simply note that any K-set is counted at least once as a rooted K-set for some vertex v chosen to be the root, and thus $T(n) \le n \cdot R(n)$. □

The above proof can be used to obtain an algorithm to enumerate all K-sets of a tree in time $O^*(3^{n/3})$. This algorithm chooses any vertex r of maximum degree and branches into two subproblems: in one r is taken into W and in the other one r is discarded from W and thus all neighbors of r are discarded from S.

5 The Exact Algorithm

In this section we present our exact exponential time algorithm solving KAYLES. The algorithm starts with a call to the procedure `compute_nimber` shown in Figure 1, with input $G = (V, E)$. If it returns a nimber that is at least one, then Player 1 has a winning strategy on G; if it otherwise returns nimber zero, then Player 2 has a winning strategy. Correctness of the procedure directly follows from the discussion in Section 2.

Note that the procedure `compute_nimber`$(G[W])$ is only called for K-sets, and thus $G[W]$ is always connected, with one possible exception: if G is not connected, then the first call to the procedure is for $G[V]$ with V not a K-set. As the overhead per recursive call is polynomial, the running time is a polynomial factor times the number of K-sets in G. The procedure computes the nimber $nb(W)$ of $G[W]$ for all K-sets W of G and stores the value in a table using Memorisation, i.e., computed values are stored in a table, and by look-up no value $nb(W)$ is computed more than once. It follows that the running time of the algorithm is $O^*(|\mathcal{K}(G)|)$ where $\mathcal{K}(G)$ is the set of K-sets of G.

```
Procedure compute_nimber(G[W]).
  if $nb(W)$ already computed then
    return $nb(W)$
  else
    $M := \emptyset$;
    for all $v \in W$ do
      let $Z_1, Z_2, \ldots, Z_r$ $(r \geq 1)$ be the components of $G - N[w]$;
      $nim := 0$;
      for $i \leftarrow 1$ to $r$ do
        $nim := nim \oplus$ compute_nimber($G[Z_i]$);
        $M := M \cup \{nim\}$
      $answer :=$ mex($M$);
    $nb(W) := answer$;
    return $answer$
```

Figure 1: Procedure compute_nimber

Combining the bounds of the previous sections on the number of K-sets with the algorithm of this section, we establish the following result.

Theorem 7 KAYLES *can be solved in time $O^*(1.6052^n)$ for graphs on n vertices.* KAYLES *can be solved in time $O^*(1.4423^n)$ for trees on n nodes.*

6 Lower Bounds

In this section we present graphs on n vertices having $\Theta(3^{n/3})$ different K-sets. This implies a lower bound on the maximum number of K-sets of any graph on n vertices as well as a lower bound on the running time of any exact algorithm solving KAYLES by using all K-sets of the input graph.

Theorem 8 *There are graphs on n vertices with $3^{n/3} + 2n/3$ different K-sets.*

Proof. Consider the following family of (chordal) graphs G_n for all positive integers n on the vertex set $\{1, 2, \ldots, 3n\}$. The edge set of G_n is constructed as follows:

- $\{3i \ : \ i = 1, 2, \ldots, n\}$ is a clique of G_n, and
- for all $i = 1, 2, \ldots, n$, the vertex set $\{3i - 2, 3i - 1, 3i\}$ induces a path.

Let us count the K-sets W of G_n.
Case 1: $W \cap \{3i \ : \ i = 1, 2, \ldots, n\} = \emptyset$, which implies $|S \cap \{3i \ : \ i = 1, 2, \ldots, n\}| = 1$. Say $S \cap \{3i \ : \ i = 1, 2, \ldots, n\} = \{3i_0\}$. Hence $W \subseteq \{3i - 1, 3i - 2\}$ for some i. Thus if $i \neq i_0$ then $W = \{3i - 1, 3i - 2\}$; and if $i = i_0$ then $W = \{3i - 2\}$. thus there are $2n$ different K-sets in this case.

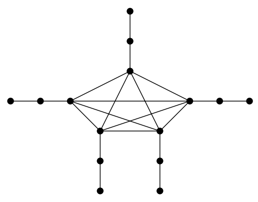

Figure 2: Example of the construction of Theorem 8, with $n = 5$

Case 2: $W \cap \{3i \ : \ i = 1, 2, \ldots, n\} \neq \emptyset$, which implies $S \cap \{3i \ : \ i = 1, 2, \ldots, n\} = \emptyset$. Then $W \subseteq \{3i-2, 3i-1, 3i\}$ may be any of the following sets $\{3i-2, 3i-1, 3i\}$, $\{3i\}$, $\emptyset$. Thus there are $3^n - 1$ different K-sets W in this case.

In total the graph G_n has at least $3^n + 2n$ K-sets. □

Theorem 9 *There are trees on n nodes with $3^{(n-1)/3} + 4(n-1)/3$ different K-sets.*

Proof. Consider the following family of trees T_n for all positive integers n. The node set of T_n is the set $\{0, 1, 2, \ldots, 3n+1\}$. The edgeset is constructed as follows:

- For all $i = 1, 2, \ldots, n$, the vertex set $\{3i-2, 3i-1, 3i\}$ induces a path, and
- the node 0 is adjacent to all nodes in the set $\{3i \ : \ i = 1, 2, \ldots, n\}$ and no others.

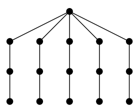

Figure 3: Example of the construction of Theorem 9, with $n = 5$

To count the K-sets W of T_n we distinguish two cases.

Case 1: $0 \notin W$. Then $S \cap \{0, 3, 6, \ldots, 3n\} \neq \emptyset$. Hence $W \subseteq \{3i, 3i-1, 3i-2\}$ for some i. Thus $W = \{3i, 3i-1, 3i-2\}$, $W = \{3i, 3i-2\}$, $W = \{3i\}$, $W = \{3i-2\}$. thus there are $4n$ different K-sets.

Case 2: $0 \in W$. Then $S \cap \{0, 3, 6, \ldots, 3n\} = \emptyset$. For every i, consider $W \cap \{3i-2, 3i-1, 3i\}$. By connectedness of $G[W]$ and $0 \in W$, we obtain that $W \cap \{3i-2, 3i-1, 3i\}$ is any of the following sets $\{3i-2, 3i-1, 3i\}$, $\{3i\}$, $\emptyset$. Thus there are $3^n - 1$ different K-sets W in this case.

Summarizing, the tree T_n has at least $3^n + 4n$ K-sets. □

7 Conclusions

In this paper, we gave an algorithm to determine which player has a winning strategy for the game KAYLES. To analyse the running time, we introduced the notion of K-sets, and obtained upper and lower bounds on the maximum number of K-sets that a graph can have. We also obtained such bounds for trees; up to a polynomial factor, the bounds are sharp for trees.

A number of interesting directions for further research remain. The complexity of KAYLES on trees remains a long standing open problem. But one can also ask if there exists a subexponential time algorithm for KAYLES on trees, e.g., with running time of the form $O(c^{\sqrt{n}})$.

Our algorithm uses exponential memory. It also is open if there exists a polynomial space algorithm with a running time of $O^*(2^n)$, and this may well be hard to obtain.

Our paper is a first example of exact algorithms for problems that are PSPACE-complete. It would be interesting to study such algorithms for other PSPACE-complete problems, e.g., for other combinatorial games, or for a problem like QUANTIFIED 3-SATISFIABILITY [8]. An algorithm that solves QUANTIFIED (3-)SATISFIABILITY in $O^*(2^n)$ time is not hard to find, but it seems very hard (or impossible) to find an algorithm with a running time $O^*(c^n)$ with $c < 2$ for this problem.

References

[1] BERLEKAMP, E. R., CONWAY, J. H., AND GUY, R. K. *Winning Ways for your mathematical plays, Volume 1: Games in General.* Academic Press, New York, 1982.

[2] BERLEKAMP, E. R., CONWAY, J. H., AND GUY, R. K. *Winning Ways for your mathematical plays, Volume 2: Games in Particular.* Academic Press, New York, 1982.

[3] BODLAENDER, H. L., AND KRATSCH, D. Kayles and nimbers. *Journal of Algorithms 43* (2002), 106–119.

[4] CONWAY, J. H. *On Numbers and Games.* Academic Press, London, 1976.

[5] FLEISCHER, R., AND TRIPPEN, G. Kayles on the way to the stars. In *Proceedings of the 4th International Conference on Computers and Games, CG 2004* (2006), H. J. van den Herik, Y. Björnsson, and N. S. Netanyahu, Eds., vol. 3846 of *Lecture Notes in Computer Science*, Springer Verlag, pp. 232–245.

[6] FOMIN, F. V., AND KRATSCH, D. *Exact Exponential Algorithms.* Springer, 2010.

[7] GUIGNARD, A., AND ÉRIC SOPENA. Compound Node-Kayles on paths. *Theoretical Computer Science 410* (2009), 2033–2044.

[8] SCHAEFER, T. J. The complexity of satisfiability problems. In *Proceedings of the 10th Annual Symposium on Theory of Computing, STOC'78* (1978), pp. 216–226.

[9] Schaefer, T. J. On the complexity of some two-person perfect-information games. *Journal of Computer and System Sciences 16* (1978), 185–225.