The Complexity of Escaping Labyrinths and Enchanted Forests

Florian D. Schwahn[1]
Department of Mathematics, University of Kaiserslautern, Paul-Ehrlich-Str. 14,
D-67663 Kaiserslautern, Germany
fschwahn@mathematik.uni-kl.de

Clemens Thielen
Department of Mathematics, University of Kaiserslautern, Paul-Ehrlich-Str. 14,
D-67663 Kaiserslautern, Germany
thielen@mathematik.uni-kl.de
https://orcid.org/0000-0003-0897-3571

Abstract

The board games *The aMAZEing Labyrinth* (or simply *Labyrinth* for short) and *Enchanted Forest* published by Ravensburger are seemingly simple family games.

In *Labyrinth*, the players move though a labyrinth in order to collect specific items. To do so, they shift the tiles making up the labyrinth in order to open up new paths (and, at the same time, close paths for their opponents). We show that, even without any opponents, determining a shortest path (i.e., a path using the minimum possible number of turns) to the next desired item in the labyrinth is strongly NP-hard. Moreover, we show that, when competing with another player, deciding whether there exists a strategy that guarantees to reach one's next item faster than one's opponent is PSPACE-hard.

In *Enchanted Forest*, items are hidden under specific trees and the objective of the players is to report their locations to the king in his castle. Movements are performed by rolling two dice, resulting in two numbers of fields one has to move, where each of the two movements must be executed consecutively in one direction (but the player can choose the order in which the two movements are performed). Here, we provide an efficient polynomial-time algorithm for computing a shortest path between two fields on the board for a given sequence of die rolls, which also has implications for the complexity of problems the players face in the game when future die rolls are unknown.

2012 ACM Subject Classification Theory of computation → Shortest paths, Theory of computation → Representations of games and their complexity

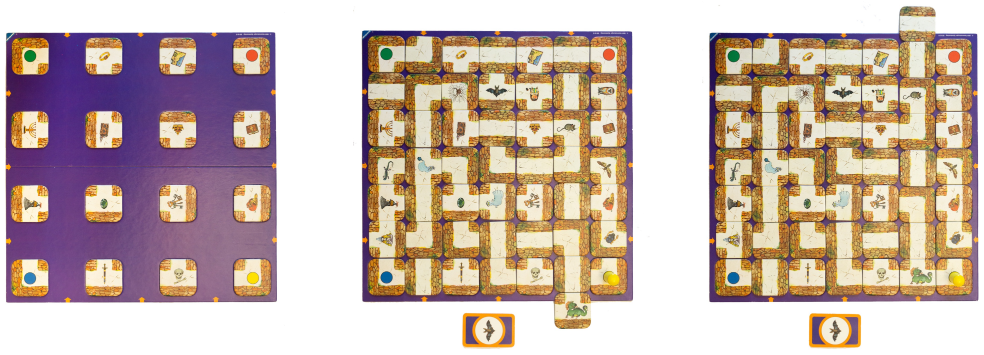
Keywords and phrases board games, combinatorial game theory, computational complexity

Digital Object Identifier 10.4230/LIPIcs.FUN.2018.30

1 Introduction

Computational complexity questions related to games and puzzles have received considerable interest among mathematicians and computer scientists within the last decades. For an introduction to the topic and an overview of known results, we refer to [1, 3, 5]. While many one-player puzzles are NP-complete, two-player games often turn out to be PSPACE-complete or even EXPTIME-complete.

[1] The work of this author was partially supported by Anne M. Schwahn by explaining the two studied board games to him back in the dark winters of 1990 and 1991.



(a) The empty board. **(b)** Before the shift. **(c)** After the shift.

Figure 1 The aMAZEing board.

In this paper, we study the computational complexity of several natural decision problems arising in the two board games *(The aMAZEing) Labyrinth* [6] and *Enchanted Forest* [8]. In both games, the players move on a board subject to specific movement rules in order to find certain items. In Labyrinth, this involves shifting the moving tiles on the board in order to open up paths through the labyrinth. In Enchanted Forest, movement is performed by rolling two dice, resulting in two numbers of fields the player has to move subject to the constraint that each of the two movements must be executed consecutively in one direction.

In the following sections, we first consider Labyrinth and show that – even without any opponents – deciding whether a given tile on the board is reachable within a given number of turns is strongly NP-complete. When competing with another player, the natural extension of this shortest path problem that asks whether a player has a strategy that guarantees to reach a given target tile faster than her opponent reaches their (possibly different) target tile is shown to be PSPACE-hard. For Enchanted Forest, on the other hand, we provide an efficient polynomial-time algorithm for deciding whether a given field on the board can be reached in a given number of turns for a given sequence of die rolls.

2 The aMAZEing Labyrinth

The game *(The aMAZEing) Labyrinth*, developed by Max J. Kobbert, was originally published by Ravensburger in 1986 under the German title "Das verrückte Labyrinth" [6] ("verrückt" is a play on two possible meanings, similar to *disarranged*). The game is a huge success all over the world with about 30 million sold units in over 60 countries so far.

In the game, one to four players[2] try to collect a sequence of items on a board consisting of moving tiles. The board (see Figure 1 (a)) features spots for 7×7 square tiles, with some of those fixed to it. The tiles have three different shapes, which can be rotated in two/four orientations: I-tile (,) , L-tile (, , ,) , T-tile (, , ,) .

The board is randomly filled with the movable tiles and one additional tile is placed aside. In a player's turn, she has to execute a *shift* and a *move* action (see Figures 1 (b) and (c)).

[2] This refers to the original release of the game, where playing alone was actually allowed. Now, it is sold as a game for *two* to four players.

Shift: The (mandatory) shift action is performed by pushing in the surplus tile - in any orientation - from any border, such that all tiles in this row/column are shifted by one place and the border tile on the other end of the row/column is pushed out of the board (obviously, the rows and columns containing fixed tiles cannot be shifted). This tile is left in the place in order to mark the last shift action for the next player, who is not allowed to directly reverse the previous shift. If some player is standing on the tile that is pushed out of the board, the player is instead placed on the tile that has just been pushed in (*wrap-around rule*).

Move: After shifting, a player may either move to any tile reachable in the labyrinth (e.g. the requested bat tile in Figure 1 (c)), or choose not to move at all. An adjacent tile is (directly) reachable if both tiles feature open space on their common edge.

Goal: Some tiles feature the symbol of an item, animal, or mythical creature. At the beginning of the game, each player gets a stack of cards requesting to collect some of those objects. At each point in time, each player only knows about the object she is requested to collect next, but not about the further objects she has to collect. The currently requested object is collected if the move action of the player ends on the tile featuring this object. She may then look at her next card, showing the next object to collect. Once a player collects the last object requested from her, the last goal is to return to her starting tile. Whoever is the first to accomplish this is the winner of the game.

2.1 Formal Problem Definition

In order to analyze the game mathematically, we consider a board of arbitrary size. Hence, the rectangle board contains $a \times b$ *spots* for $(a \cdot b) + 1$ *tiles*, some of which may be fixed. The kinds of tiles considered and the rules of the game are as described above.

At each point in time during the game, a player only knows about the next object she is requested to collect. Hence, the fundamental problem faced at each point in time when a player plays alone is reaching the next object (or her starting tile in case that she has already collected the last object) in the minimum number of turns. The decision version of this single-player problem is formally defined as follows:

▶ **Definition 1** (SP-Labyrinth).

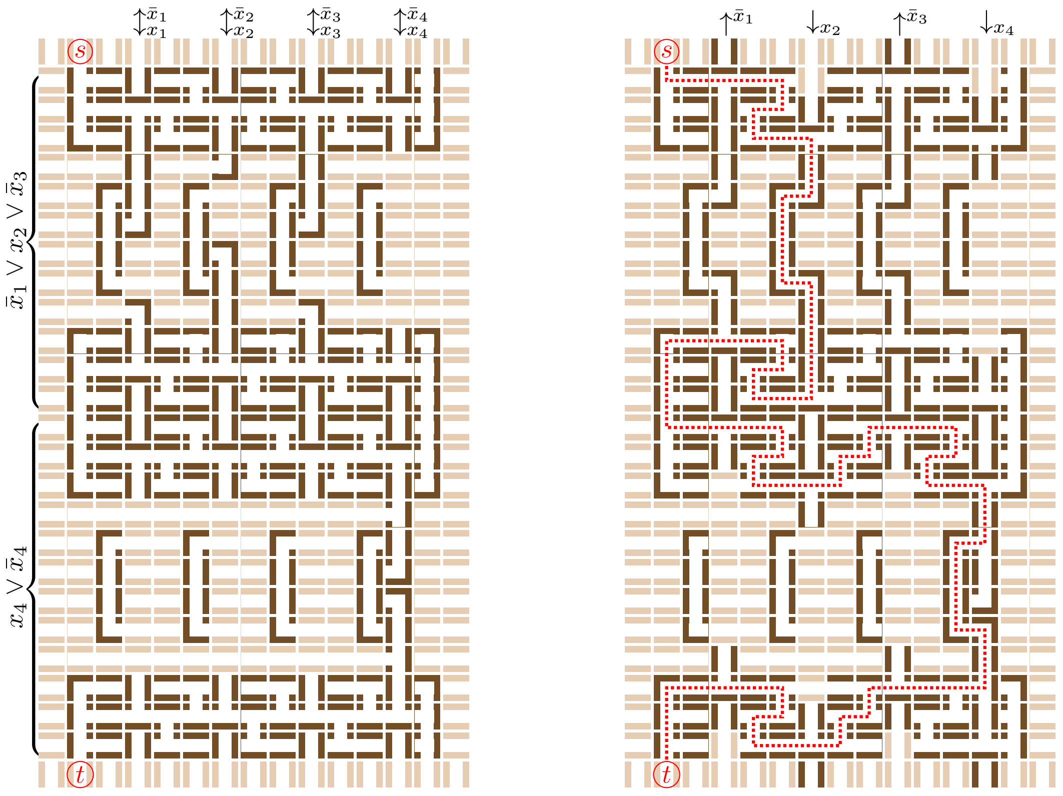
INSTANCE:	The initial board setting, the shape of the current surplus tile, two distinct tiles s and t on the board, and an allowed number of turns $k \in \mathbb{N}$.
QUESTION:	Can a single player starting at tile s reach tile t in at most k turns?

When two opposing players play alternately, the fundamental problem each player faces amounts to reaching the object she is requested to collect next (or her starting tile if she has already collected all required objects) faster than her opponent reaches their next object (or starting tile). The decision version of the two-player problem is, thus, defined as follows:

▶ **Definition 2** (SP-Versus-Labyrinth).

INSTANCE:	The initial board setting, the shape of the current surplus tile, and two pairs (s_1, t_1), (s_2, t_2) of tiles on the board.
QUESTION:	Is there a strategy for player 1 (who has the first turn and starts at tile s_1) that allows her to reach tile t_1 before player 2 (starting at tile s_2) can reach tile t_2?

In the following subsections, we first show that already the single-player problem SP-Labyrinth in strongly NP-complete (this problem has already been used as a benchmark problem for testing ASP solvers, cf. [2]). Afterwards, we consider the two-player problem SP-Versus-Labyrinth and show that this problem is PSPACE-hard.



$\uparrow\bar{x}_1$ $\downarrow x_1$
$\uparrow\bar{x}_2$ $\downarrow x_2$
$\uparrow\bar{x}_3$ $\downarrow x_3$
$\uparrow\bar{x}_4$ $\downarrow x_4$
s
t
$\bar{x}_1 \vee x_2 \vee \bar{x}_3$
$x_4 \vee \bar{x}_4$
$\uparrow\bar{x}_1$
$\downarrow x_2$
$\uparrow\bar{x}_3$
$\downarrow x_4$
s
t

Figure 2 Example of a clause and a variable gadget (initial and final board setting).

2.2 Escaping the Labyrinth is hard ...

Poor little Alice got lost in the aMAZEing Labyrinth and has to find her way to the exit before the batteries of her flashlight run out. Unfortunately (for her) we now show that deciding whether she can get out in time is strongly NP-complete:

▶ **Theorem 3.** *SP-Labyrinth is strongly NP-complete.*

Proof. Since it is straightforward to check that the player reaches t from s with a given sequence of shifts and moves, the problem is clearly contained in NP.

To show NP-hardness, we use a reduction from 3SAT (a problem that Alice has probably encountered when reading [4]). Given an instance $\mathcal{I}$ of 3SAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$, we construct an instance of SP-Labyrinth in which the player can reach t in at most $k := n$ turns if and only if $\mathcal{I}$ is satisfiable.[3]

Figure 2 shows an example of a clause and a variable gadget we use in this reduction. In total, there is one gadget for each variable and one gadget for each clause, i.e., there are $n+m$ gadgets with twelve rows each. Together with the top row containing s and the bottom row containing t, this makes $12 \cdot (n+m)+2$ rows. To reach t from s, the player needs to move through or get shifted through the $12 \cdot (n+m)$ rows between s and t. As each shift of the avatar itself can only contribute a single row, she has to move along vertical paths to reach t fast enough. To avoid problems with the wrap-around rule, we augment the board on

[3] This also shows that it is sometimes helpful to have a SAT solver with you when entering a labyrinth.

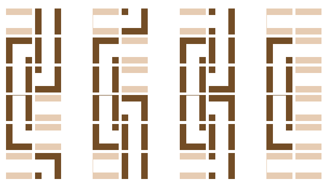


Figure 3 The four hook gadgets (negated, unnegated, either-or, empty).

each side by $2n$ columns / rows that are filled with ┃┣-tiles and ═-tiles in alternating order. Thus, none of the central tiles shown in Figure 2 can be shifted out of the board and the player cannot reach any boundary of the augmented board in n turns. Moreover, even with those additional tiles, the resulting board has size polynomial in n and m, so the instance can be constructed in polynomial time.

The complete board setting includes one gadget for each clause and one gadget for each variable. The gadgets vary only in the special variable columns (marked with $\updownarrow^{\bar{x}_i}_{x_i}$). Here, shifting the variable column corresponding to a variable x_i downwards will correspond to setting x_i to `true`, while shifting the column upwards will correspond to setting x_i to `false`.[4] The connection to the clauses is made by using the four different kinds of *hook gadgets* shown in Figure 3. Each clause gadget contains the corresponding hook gadget for each variable (depending on whether the variable is contained in the clause negated, unnegated, both negated and unnegated, or not at all). For example, the clause gadget for $\bar{x}_1 \vee x_2 \vee \bar{x}_3$ presented in Figure 2 contains a negated hook gadget for x_1 and x_3, and unnegated hook gadget for x_2, and an empty hook gadget for x_4 (since x_4 is not contained in the clause). As one can see, the clause gadget can be crossed if and only if the variable column corresponding to either x_1 or x_3 is shifted upwards, or the variable column corresponding to x_2 is shifted downwards. Since x_4 is not contained in the clause, shifting the corresponding variable column cannot make the gadget crossable.

The first and last three rows of every clause gadget (above and below the hook gadgets) ensure that, starting from the top left of the gadget, we can choose to cross by any variable column (given that the corresponding variable was set to satisfy the clause) and reach the bottom left of the gadget in order to enter the next gadget.

Below the clause gadgets, there is an analogously constructed gadget for each variable (i.e., the gadget for variable x_i corresponds to a clause gadget for the clause $x_i \vee \bar{x}_i$, see Figure 2). Thus, the variable gadget corresponding to variable x_i will be crossable if and only if the column of variable x_i was shifted either upwards or downwards. Hence, all variable gadgets will be crossable if and only if each variable was set to either `true` or `false`.

We now show that the constructed instance is equivalent to the given instance $\mathcal{I}$ of 3SAT, i.e., that the player can reach t in at most n turns if and only if $\mathcal{I}$ is satisfiable.

First assume that $\mathcal{I}$ is satisfiable. Then, shifting the variable columns according to some satisfying variable assignment yields a path from s to t through which the player can move after the n-th shift (in the previous $n-1$ turns, the player does not move at all). Hence, the player can reach t in at most n turns.

[4] Instead, one could equivalently shift the column to the left of the variable column in the opposite direction, which has the same effect (but could be prevented by putting a fixed tile on top of this column). In the following, we will assume that the variable column itself is always shifted instead of the column to its left.

In order to prove the other direction, we observe that, in every solution to the constructed SP-Labyrinth instance, each variable column must be shifted exactly once. This follows since the corresponding variable gadget is not crossable in its initial form and the only possible way to make it crossable by using only a single (row or column) shift is to shift the corresponding variable column exactly once (as there are n variable gadgets in total and only n shifts are available, we cannot use more than one shift to make a single variable gadget crossable). Hence, each solution to the constructed SP-Labyrinth instance corresponds to a truth-assignment for the variables in the given 3SAT-instance $\mathcal{I}$ by setting x_i to `true` (`false`) if and only if the variable column corresponding to x_i is shifted downwards (upwards). Moreover, since all clause gadgets are crossable in the solution to the SP-Labyrinth instance, all clauses in $\mathcal{I}$ must be satisfied by this truth-assignment. ◀

2.3 ...but doing it faster than someone else is even harder.[5]

The famous archaeologists Lara and Henry Jr. play a game of the aMAZEing Labyrinth and each of them only needs to return to their starting tile. It is Lara's turn – but will she manage to arrive first and, thus, win the game? Who will resort to a destructive strategy? Does it pay off?

▶ **Theorem 4.** *SP-Versus-Labyrinth is PSPACE-hard.*

Proof. We use a reduction from *Quantified Satisfiability* (QSAT) (also known as *Quantified Boolean Formula*, cf. [9]). Given an instance $\mathcal{I}$ of QSAT with variables $x_1, \ldots, x_n$ and clauses $C_1, \ldots, C_m$, we construct an instance of SP-Versus-Labyrinth in which player 1 (Lara) has a winning strategy if and only if $\mathcal{I}$ is a yes-instance.[6] Without loss of generality, we assume that the number n of variables in $\mathcal{I}$ is odd.

This time, we translate "setting variables" into "shifting variable *rows*" by using similar clause gadgets as in the proof of Theorem 3 (rotated clockwise by 90 degrees). However, we now use only a single variable gadget corresponding to variable x_n - the other variables have no corresponding variable gadgets. Instead, the board contains a distinct order preserving gadget for each player, which is used to make sure that the player has to set her corresponding variables x_i (the ones with odd i for Lara and the ones with even i for Henry) in the correct order. In between each pair of adjacent gadgets and at the sides of the board, we need some buffer columns to avoid any unintentional interaction of the gadgets as well as using the wrap-around rule. Since we will show that at least one of the players will always reach their goal after at most $n+1$ turns, $n+2$ buffer columns consisting of alternating ‖ and =-tiles are sufficient between each pair of adjacent gadgets and at the left and the right of the board.[7]

In order to reach her goal, Lara has to cross her order gadget as well as the clause gadgets, whereas Henry only has to cross his order gadget and the variable gadget of variable x_n (but *not* the clause gadgets). An overview of the structure of the board is given in Figure 4. The pink row at the top in Figure 4 is used in order to connect the different gadgets and will be referred to as the *wiring row* throughout the rest of the proof. The blue row at the bottom is used as an *escape path* that makes a player reach her goal directly in case that the other

[5] Assuming NP $\neq$ PSPACE.
[6] This shows that, as expected, beating an archaeologist as intelligent as Henry Jr. is very challenging.
[7] As will become clear later in the proof, two buffer columns (instead of $n+2$) are actually sufficient at each of these places since no row will ever be shifted more than twice before one of the players can reach their goal.

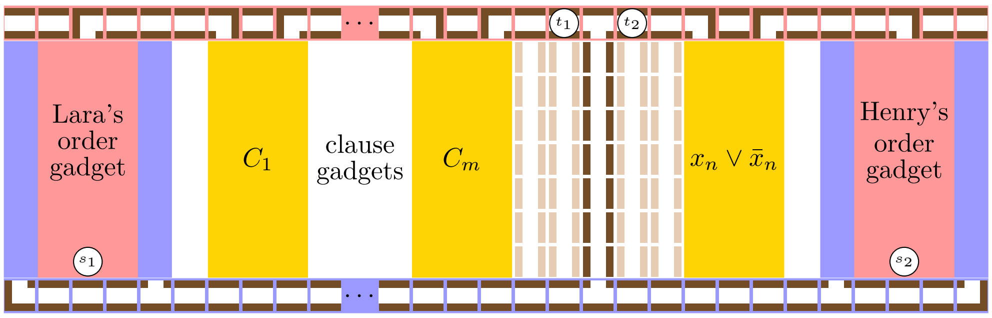


Figure 4 Overview of the board used in the proof of Theorem 4.

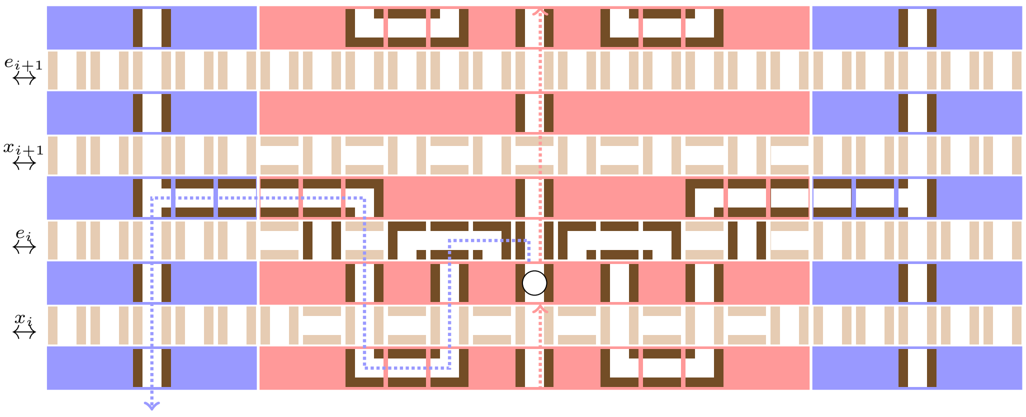


Figure 5 A segment of a player's order gadget (before turn i of the other player).

player does not shift their corresponding variable rows in the correct order. All tiles of the wiring row and the escape path row are fixed, which implies that only rows can be shifted and the wrap-around rule cannot be used on the top or the bottom of the board (where no buffer was added).

In Figure 5, we illustrate a segment of a player's order gadget. Here, exactly the variable rows of Henry's variables (i.e., the ones with even index) are crossable via the pink path at the beginning of the game in Lara's order gadget, while the variable rows of Lara's variables (i.e., the ones with odd index) are crossable via the pink path in Henry's order gadget at the beginning. Moreover, Lara starts in the middle of the row *below* the variable row corresponding to x_1 in her order gadget, while Henry starts in the middle of the row *above* the variable row of x_1. The lowermost shiftable row corresponds to x_1 and the uppermost shiftable row corresponds to x_n (i.e., there is no row e_n). We note that, compared to the (rotated) structure of the clause gadgets and the variable gadget of x_n shown in Figure 2, the row e_i represents an additional row (column in the figure) between the variables x_i and x_{i+1} for each $i \in \{1, \dots, n-1\}$. Within the clause gadgets and the variable gadget of x_n, this additional row is simply filled by ❙❙-tiles (which would correspond to 〓-tiles in Figure 2 due to the rotation) so that it can always be crossed and does not change the structure of the gadgets.

In total, the board contains $4n$ rows (2 for each x_i for $i \in \{1, \dots, n\}$, 2 for each e_i for $i \in \{1, \dots, n-1\}$, the wiring row, and the escape path row). Since each of the two order gadgets, each of the m clause gadgets, and the variable gadget corresponding to x_n only requires a constant number of columns and $n+2$ buffer columns are inserted in between each pair of adjacent gadgets as outlined above, the constructed board is of size poly(n, m), so the described instance can be constructed from the given instance $\mathcal{I}$ of QSAT in polynomial time.

We now show that the constructed instance is equivalent to the given instance $\mathcal{I}$ of QSAT, i.e., that Lara has a winning strategy if and only if $\mathcal{I}$ is a yes-instance. To do so, we first assume that both players only shift their corresponding variable rows in the correct order, i.e., that Lara always sets the next odd variable available in each of her turns and that Henry always sets the next even variable available in each of his turns. As we show afterwards, this behavior of the players will be enforced since each player can reach the escape path at the bottom of the board (and, thus, win) if the other player deviates from this rule.

Lara starts the game and, according to the above assumption, shifts the variable row corresponding to x_1 in turn one, thereby setting x_1 to either `true` or `false`. Moreover, this closes the blue path in Henry's order gadget (Figure 5 and, thus, prevents Henry from using this path in order to reach the escape path at the bottom of the board. After her shift, Lara can move past the variable row of x_1 and also the untouched variable row of x_2 in her order gadget. However, she cannot yet cross the variable row corresponding to x_3 in her order gadget. Thus, in order to still have the blue path after the variable row of x_2 available in case that Henry should not set x_2 in the next turn, Lara will stop in the row between x_2 and e_2.

Again according to the above assumption, Henry will now shift the variable row corresponding to x_2 in turn two (which is Henry's first turn), thereby setting x_2 to either `true` or `false`. Moreover, this closes the blue path in Lara's order gadget and prevents her from entering the escape path at the bottom of the board in her next turn. After his shift, Henry can move past the variable row of x_2 and also the untouched variable row of x_3 in his order gadget. However, he cannot yet cross the variable row corresponding to x_4 in his order gadget. Thus, in order to still have the blue path after the variable row of x_3 available in case that Lara should not set x_3 in her second turn, Henry will stop in the row between x_3 and e_3.

It is then Lara's turn again and she will now set variable x_3 and so on. At the end, in turn $n-1$, Henry will set variable x_{n-1} due to the assumption that n is odd. Thus, Henry reaches the entry to the wiring row at the top of his order gadget in turn $n-1$. However, he cannot yet reach his goal t_2 since he cannot yet cross the variable gadget of the still unset variable x_n. Turn n is then Lara's turn and she only has to set variable x_n in order to reach the entry to the wiring row at the top of her order gadget. Since she has to cross the clause gadgets in order to reach her goal t_1, she will, thus, arrive at t_1 in turn n (i.e., before Henry reaches t_2) if and only if all the clauses are satisfied by the variable assignment determined by both players in the n turns. Since, in any case, Henry will reach t_2 in turn $n+1$, this shows that Lara has a winning strategy (i.e., a strategy of setting the odd variables in her corresponding turns so that all clauses will be satisfied no matter how Henry sets the even variables in his turns) if and only if the given instance $\mathcal{I}$ of QSAT is a yes-instance.

It remains to show that whoever violates the order first loses the game. So assume that both players always set their corresponding variables as desired during the first $i-1$ turns, but in turn i, the corresponding player (say Lara) decides not to shift the variable row corresponding to x_i. For Lara, this means that the variable row corresponding to x_i is still blocked in her order gadget and she cannot proceed. For $i = n$, Lara loses in this case since Henry will reach t_2 in turn $n+1$ as seen above. For $i < n$, Henry finds his order gadget as in Figure 5 before his next turn (standing in the circled spot in the row between x_i and e_i) and, with a left or right shift of row e_i, he can move onto the blue path (which is not blocked since Lara did not shift the variable row corresponding to x_i). Note that, even though Lara may have shifted row e_i in turn i, Henry can always enter the blue path either to the left or to the right since any single shift in row e_i opens the blue path for him. Consequently, Henry can enter the escape path row, which directly takes him to his goal t_2, and Lara loses the game. Similarly, Lara can enter the escape path row if Henry deviates first, which finishes the proof. ◀

Note that our proof of PSPACE-hardness did not rely on the rule that prevents a player from directly reversing the previous shift – the hardness result in Theorem 4 holds both with and without this rule.

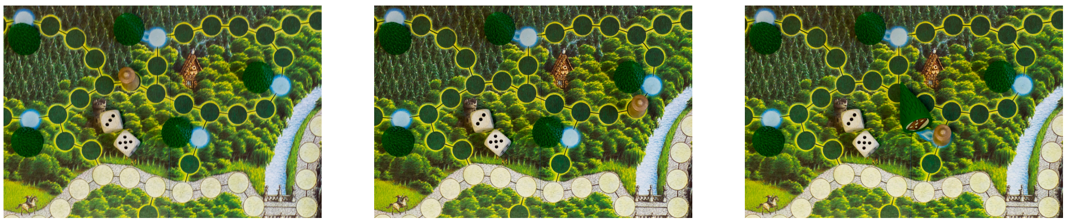
Concerning upper bounds on the computational complexity of SP-Versus-Labyrinth, note that the number of possible positions for each of the avatars and goals of the two players is bounded by the number of spots on the board, there are only twenty different tiles possible at each spot (counting different orientations and whether the tile is fixed or not), and only three possible shapes exit for the current surplus tile. Consequently, the number of possible game states in SP-Versus-Labyrinth is bounded from above by an exponential function of the board size $a \cdot b$ and it follows by standard arguments that SP-Versus-Labyrinth $\in$ EXPTIME (see, for example, [12]). Furthermore, it can easily be seen that SP-Versus-Labyrinth $\in$ PSPACE when upper bounding the number of turns by some value k polynomial in the encoding length of the game and then rating which player has achieved the better situation after k turns, i.e., has fewer turns left to reach her goal when continuing to play alone (similar to the turn-restricted version of Go analyzed in [9]). Since our proof of PSPACE-hardness works also for this turn-restricted version of SP-Versus-Labyrinth, it follows that the turn-restricted version of the problem is PSPACE-complete. Whether the actual (not turn-restricted) version of SP-Versus-Labyrinth belongs to PSPACE, however, remains an interesting open question.

3 Enchanted Forest

The game *Enchanted Forest* developed by Alex Randolph and Michel Matschoss was originally published by Ravensburger in 1981 under the German title “Sagaland” [8]. In 1982, the game earned the prestigious award “Spiel des Jahres” (engl. *Game of the Year*) [10].

In the game, two to six players move around in an enchanted forest in order to find items from popular fairy tales that are hidden under special trees. At the beginning of the game, all players start in the village next to the enchanted forest. A player gets to know which item is hidden under one of the trees if she ends her move on the blue field next to the tree. The king in the castle requests the location of the different items and whoever moves up to the castle and reports the correct location of the currently requested item earns a point. The player standing in the castle may then continue to name (guess if necessary) the locations of the next requested items to earn additional points. When she names a wrong location for the first time, she must return to the starting point in the village. The first player to earn three points wins the game.

A player’s turn consists of rolling two dice, which results in two numbers of fields she has to move. Each of the two movements must be executed consecutively in one direction, but the player can choose the order in which the two movements are performed (see Figure 6 for an example). If the roll is a double, the player may alternatively choose to either move to any blue field adjacent to a tree or to the castle, or to shuffle the cards determining the order in which the king requests the items. If a player moves to a field already occupied by another player, the other player is moved back to the starting point in the village.



(a) Initial position of the player and the result of rolling the two dice.

(b) The player's position after her first movement (five fields).

(c) After the second movement (three fields), the player reaches the tree and observes the item hidden underneath it.

Figure 6 A player's turn in Enchanted Forest.

3.1 Formal Problem Definition

In order to analyze the game mathematically, we model the board as an undirected, connected, simple graph $G = (V, E)$, where each vertex corresponds to a field on the board and there is a unit-length edge between each pair of adjacent fields.[8] As usual, we let $n := |V|$ and $m := |E|$. The special fields to which a player can move instantaneously when rolling a double (the blue fields adjacent to the trees and the castle) are given as a subset $V' \subseteq V$. Moreover, we consider two arbitrary d-sided dice for the moves (where $d > n$ is possible since the graph may contain cycles). If a player rolls $(x, \bar{x})$ with $x, \bar{x} \in \{1, \dots, d\}$ and starts at $s \in V$, she can decide on two (not necessarily simple) paths to follow, where the first one starts at s and the second one starts at the end vertex of the first one. One of these two paths must have length x and the other one length $\bar{x}$. Moreover, the requirement that each movement has to be executed consecutively in one direction means that the two paths are not allowed to contain cycles of length 2 as subpaths. The special rules used when rolling a double mean that, if $x = \bar{x}$, the player may alternatively choose to move to any vertex in the subset V'.

As in *Labyrinth*, we consider the fundamental problem of reaching a certain location on the board (e.g., the castle or a specific tree) in a minimum number of turns. Here, we assume that the player has complete knowledge of the sequence of die rolls for her future turns. The decision version of this problem is formally defined as follows:

▶ **Definition 5** (SP-EnchantedForest).

INSTANCE:	The simple graph $G = (V, E)$, the subset $V' \subseteq V$, the maximum die value $d \in \mathbb{N}$, the die rolls $(x_1, \bar{x}_1), \dots, (x_k, \bar{x}_k)$ with $x_i, \bar{x}_i \in \{1, \dots, d\}$, and two vertices $s, t \in V$.
QUESTION:	Can a player starting at vertex s reach vertex t in at most k turns using the given rolls of the dice?

The encoding length of an instance of SP-EnchantedForest (when storing the graph G in adjacency list representation) is $\mathcal{O}(n + m + k \cdot \log_2 d)$.

In the following subsection, we show that SP-EnchantedForest can be solved efficiently in polynomial time. This result also has implications for the complexity of problems the players face in Enchanted Forest when the outcomes of future die rolls are unknown. For example, the polynomial-time solvability of SP-EnchantedForest directly implies that, when

[8] Note that, even though G models an enchanted *forest*, the graph may contain cycles (as does the original board).

the outcomes of future die rolls are unknown, the problem of choosing two movements for the current turn that maximize the probability of reaching a desired location in (at most) k turns for a given constant k can be solved in polynomial time.

3.2 Finding one's way in the Enchanted Forest is easy

After living a long, rich, and joyful life, Gretel wants to relive her childhood memories and eat some gingerbread from the gingerbread house in the Enchanted Forest. However, the ancient enchantments do not allow her to simply follow the path of pebbles laid out. Before making the next step, she has to roll two dice and move accordingly. At least, with all the lucky charms she obtained from the witch's heritage, she can predict the rolls. Can she find the delicious gingerbread or will the journey be too long to bear the appetite?

Gretel rolls (or rather predicts to roll) $(x_1, \bar{x}_1), \ldots, (x_k, \bar{x}_k)$ and decides that it is probably a good idea to first compute which vertices in G she can reach from a given position in the enchanted forest by using a single die roll. Thus, for any $x \in \{1, \ldots, d\}$, she defines the (symmetric) $(n \times n)$-matrix D_x with entries in $\{0, 1\}$ such that, for each pair $(u, v) \in V \times V$, the matrix has an entry 1 at the position corresponding to u and v if and only if there exists a (not necessarily simple) path of length x from u to v in G that does not contain any cycles of length 2 as subpaths. However, in order to compute D_x efficiently in polynomial time, Gretel cannot explore the graph G step-by-step by always moving to an adjacent vertex since this would lead to a time requirement polynomial in d, but *not* in $\log_2 d$. Instead, she uses the following procedure provided by two friendly scholars:

Computing $\mathbf{D_x}$ efficiently: In order to make sure that only paths that do not contain cycles of length 2 as subpaths are considered, Gretel constructs a directed graph $H = (N, A)$ from G by setting

$$N := \{v_u : \{u, v\} \in E\}, \quad \text{and} \quad A := \{(v_u, w_v) : v_u, w_v \in N, \{v, w\} \in E, w \neq u\}.$$

Here, the vertex set N of H contains a copy v_u of each vertex $v \in V$ for every neighbor u of v in G. The arcs in H are constructed such that the copy v_u of v corresponding to u is connected by a directed arc (of unit length) to all copies w_v of the neighbors w of v that are *different from* u. Thus, there exists a path of length x without cycles of length 2 as subpaths from a node $\tilde{u}$ to another node $\tilde{v}$ in G if and only if there exists a directed path of the same length from *some copy* of $\tilde{u}$ to *some copy* of $\tilde{v}$ in H. Hence, the problem of determining whether a node $\tilde{v}$ in G can be reached from a given node $\tilde{u}$ in G by a path of length x without cycles of length 2 as subpaths reduces to determining whether *some copy* $\tilde{v}_w$ of $\tilde{v}$ can be reached from *some copy* $\tilde{u}_z$ of $\tilde{u}$ by a directed path of length x in H.

To compute which vertices in H are reachable from which other vertices, Gretel computes the x-th power of the adjacency matrix M of H (cf. [11]). This can be done in $\mathcal{O}(|N|^{2.373} \log_2 x) = \mathcal{O}(m^{2.373} \log_2 d)$ time by computing M^{2^α} for $\alpha = 1, \ldots, \lfloor \log_2 x \rfloor$ via the square matrix multiplication algorithm from [7] (since $|N| = 2 \cdot |E| = 2m$ and $x \in \{1, \ldots, d\}$). Then, D_x has a 1 at the position corresponding to $u \in V$ and $v \in V$ exactly if M^x has a positive entry at some position corresponding to a copy of u and a copy of v.

Computing which vertices are reachable in a single turn: With the knowledge of the next k rolls $(x_1, \bar{x}_1), \ldots, (x_k, \bar{x}_k)$ and the above method for computing the sets D_x, Gretel can now compute efficiently which vertices v she can reach from any given vertex u in the graph G with any single pair $(x_i, \bar{x}_i)$ of die rolls. To do so, she defines the $(n \times n)$-matrix $D_{(x_i, \bar{x}_i)}$

with entries in $\{0,1\}$ such that, for each pair $(u,v) \in V \times V$, the matrix has an entry 1 at the position corresponding to u and v if and only if vertex v can be reached from vertex u with the pair $(x_i, \bar{x}_i)$ of die rolls. Gretel now wants to compute $D_{(x_i,\bar{x}_i)}$ efficiently. If $(x_i, \bar{x}_i)$ is not a double (i.e., if $x_i \neq \bar{x}_i$), she notes that, by definition of the matrices D_{x_i} and $D_{\bar{x}_i}$, the matrix $D_{(x_i,\bar{x}_i)}$ has a 1 at the position corresponding to u and v if and only if at least one of the matrices $D_{x_i} \cdot D_{\bar{x}_i}$ and $D_{\bar{x}_i} \cdot D_{x_i}$ has a positive entry at this position. If $(x_i, \bar{x}_i)$ is a double (i.e., if $x_i = \bar{x}_i$), Gretel has to take into account that she can also decide to move instantaneously to any vertex in the subset $V' \subseteq V$. Hence, in this case, the matrix $D_{(x_i,\bar{x}_i)}$ also has a 1 at the position corresponding to u and v whenever $v \in V'$. If Gretel again uses the square matrix multiplication algorithm from [7] to compute $D_{x_i} \cdot D_{\bar{x}_i}$ and $D_{\bar{x}_i} \cdot D_{x_i}$, this shows that she can obtain $D_{(x_i,\bar{x}_i)}$ from the matrices D_x and $D_{\bar{x}_i}$ in time $\mathcal{O}(n^{2.373})$.

Turn-expanded network: Similar to a time-expanded network, Gretel constructs the (directed) *turn-expanded network* $F = (N_F, A_F)$ with

$$N_F := \{v^i : \; v \in V, i = 0, \dots, k\} \cup \{t^*\}, \text{ and}$$

$$A_F := \bigcup_{i=1}^{k} \{(u^{i-1}, v^i) : \; D_{(x_i,\bar{x}_i)} \text{ has entry 1 at position } (u,v)\} \cup \{(t^1, t^*), \dots, (t^k, t^*)\}.$$

She can now compute a shortest path from s_0 to t^* by breadth-first search in $\mathcal{O}(|N_F|+|A_F|) = \mathcal{O}(k \cdot n^2)$ time and decide whether she can reach her favorite dish in k turns (the required number of turns equals the length of a shortest s_0-t^*-path minus one). As computing the at most $2k$ matrices D_x is the most time consuming step, the overall running time of Gretel's procedure is $\mathcal{O}(k \cdot m^{2.373} \cdot \log_2 d)$, which is polynomial in the input length. This shows:

▶ **Theorem 6.** *SP-EnchantedForest can be solved in polynomial time $\mathcal{O}(k \cdot m^{2.373} \cdot \log_2 d)$.* ◀

As noted before, the result of Theorem 6 also has implications for the complexity of problems that Gretel faces when she cannot use her lucky charms in order to predict the outcomes of future die rolls. For example, the polynomial-time solvability of SP-EnchantedForest directly implies that, when the outcomes of future die rolls are unknown, Gretel only needs polynomial time in order to compute two movements for her current turn that maximize the probability of reaching the gingerbread house in (at most) k turns for a given constant k.

References

1 E. D. Demaine and R. A. Hearn. Playing games with algorithms: Algorithmic combinatorial game theory, 2001. `http://arxiv.org/abs/cs.CC/0106019`.

2 C. Dodaro, M. Alviano, W. Faber, N. Leone, F. Ricca, and M. Sirianni. The birth of a WASP: Preliminary report on a new ASP solver. In *Proceedings of the 26th Italian Conference on Computational Logic (CILC)*, pages 99–113, 2011.

3 D. Eppstein. Computational complexity of games and puzzles. `https://www.ics.uci.edu/~eppstein/cgt/hard.html`. Accessed: 2018-04-16.

4 M. R. Garey and D. S. Johnson. *Computers and Intractability (A Guide to the Theory of NP-Completeness).* W.H. Freeman and Company, New York, 1979.

5 G. Kendall, A. J. Parkes, and K. Spoerer. A survey of NP-complete puzzles. *ICGA Journal*, 31(1):13–34, 2008.

6 M. J. Kobbert. Das verrückte Labyrinth. Ravensburger, 1986.

7 F. Le Gall. Powers of tensors and fast matrix multiplication. In *Proceedings of the 39th International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 296–303, 2014.

8 M. Matschoss and A. Randolph. Sagaland. Ravensburger, 1981.

9 C. Papadimitriou. *Computational Complexity*. Addison Wesley, 1993.

10 Spiel des Jahres e.V. Spiel des Jahres. `http://www.spiel-des-jahres.com/en`.

11 R. P. Stanley. *Enumerative Combinatorics*. Wadsworth Publ. Co., 1986.

12 L. J. Stockmeyer and A. K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, 1979.