

Moving in a Network under Random Failures: A Complexity Analysis[☆]

Dominik Klein^b, Frank G. Radmacher^{a,*}, Wolfgang Thomas^a

^a*RWTH Aachen University, Lehrstuhl für Informatik 7, 52056 Aachen, Germany*

^b*Japan Advanced Institute of Science and Technology (JAIST), 1-1 Asahidai, Nomi City, Ishikawa 923-1292, Japan*

Abstract

We analyze a model of fault-tolerant systems in a probabilistic setting, exemplified by a simple routing problem in networks. We introduce a randomized variant of a model called the “sabotage game”, where an agent, called “Runner”, and a probabilistic adversary, “Nature”, act in alternation. Runner generates a path from a given start vertex of the network, traversing one edge in each move, while after each move of Runner, Nature deletes some edge of the current network (each edge with the same probability). Runner wins if the generated finite path satisfies a “winning condition”, namely that a vertex of some predefined target set is reached, or – more generally – that the generated path satisfies a given formula of the temporal logic LTL. We determine the complexity of these games by showing that for any probability p and $\varepsilon > 0$, the following problem is PSPACE-complete: Given a network, a start vertex u , and a set F of target vertices (resp. an LTL formula φ), and also a probability $p' \in [p - \varepsilon, p + \varepsilon]$, is there a strategy for Runner to reach F (resp. to satisfy φ) with a probability $\geq p'$? This PSPACE-completeness establishes the same complexity as was known for the non-randomized sabotage games (by the work of Löding and Rohde), and it sharpens the PSPACE-completeness of Papadimitriou’s “dynamic graph reliability” (where probabilities of edge failures may depend on both the edges and positions of Runner). Thus, the “coarse” randomized setting, even with uniform distributions, gives no advantage in terms of complexity over the non-randomized case.

Keywords: game theory, sabotage games, probabilistic systems, fault-tolerant systems

1. Introduction

The subject of this paper is a model of fault-tolerant computation which enables the formal analysis and verification of systems where (due to faults) dynamic changes of the system occur.

In our case, we start from the “sabotage game” suggested by van Benthem in 2002 (see [1]), where the dynamics is generated by the feature of vanishing edges in graphs. More specifically, a sabotage game is played by two players on graphs whose edges may be multi-edges. A player, called “Runner”, moves along edges and wants to reach a vertex in a set F from a given initial vertex u . After each move of Runner, the adversary, called “Blocker”, may delete an edge; and

[☆]This research was supported by the RWTH Aachen Research Cluster UMIC (Ultra High-Speed Mobile Information and Communication) of the German Excellence Initiative, German Research Foundation grant DFG EXC 89.

*Corresponding author

Email addresses: dominik.klein@jaist.ac.jp (Dominik Klein), radmacher@automata.rwth-aachen.de (Frank G. Radmacher), thomas@automata.rwth-aachen.de (Wolfgang Thomas)

in this way Runner and Blocker move in alternation. The algorithmic problem of “solving this game” asks for a winning strategy for Runner that enables him to reach a vertex in F against any choice of Blocker in deleting edges. The theory of these games was developed by Löding and Rohde (see [2, 3, 4, 5, 6]). The basic result states that the problem of solving sabotage games is PSPACE-complete. Gierasimczuk, Kurzen, and Velázquez-Quesada suggested sabotage games for a game-theoretic approach to formal learning theory [7]. Also, other winning conditions than reachability were considered: “Hamilton path” (resp. “complete search”) where Runner has to visit each vertex exactly (resp. at least) once [2, 6, 1], or “safety” where Runner has to avoid visiting a given vertex set [7].

In the present paper we combine the aspect of dynamics with probability assumptions, which lead to a “coarser” or “softer” model. The saboteur Blocker is replaced here by the player Nature who deletes, in each of his moves, one of the existing edges at random (and with equal probability). We extend the known upper and lower bounds of the analysis to this probabilistic framework. On the one hand, we show that the analysis of sabotage games remains in PSPACE, even when extending the reachability objective to a more general specification formalized in *linear temporal logic* (LTL). On the other hand, we show that, even in the softer probabilistic setting, the hardness phenomena of the standard model of sabotage games are still valid.

Our framework of “randomized sabotage games” reflects a more realistic view of the dynamics of “sabotage” than the model of the interplay between Runner and Blocker. There are not many situations in which one has to assume an omniscient adversary that manipulates the world. The faults in natural scenarios are usually better modeled as random events. When replacing “Blocker” by “Nature”, we follow the approach of “games against nature” as suggested by Papadimitriou [8]. In our work, we keep the term “randomized sabotage game” to emphasize our starting point, the sabotage games.

Towards the verification of fault-tolerant systems, we also extend the expressiveness of the specifications (“winning conditions”). Besides reachability, we study more general specifications in *linear temporal logic* (LTL). There, the vertices of the graph are labeled with atomic propositions, and Runner has to traverse the graph in such a way that the generated path of labelings satisfies a given LTL formula.

Our studies extend previous results in several ways. First we provide a surprising result concerning LTL model checking, namely that there exists a polynomial space algorithm for solving randomized sabotage games with LTL specification. The (static) LTL model checking problem of non-deterministic systems is known to be PSPACE-complete (cf. [9, 10]), but determining the winner in a two player game with an LTL winning condition is complete for double exponential time [11]. In the probabilistic setting, LTL model checking is PSPACE-complete for Markov chains (sequential programs), and it is complete for double exponential time for Markov decision processes (concurrent programs) [12]. In this framework, randomized sabotage games are more similar to Markov decision processes, since two kinds of non-determinism are involved: the change of the probabilities after each edge deletion and Runner’s choice for traversing the game arena. In view of these results on two player games [11] and Markov models [12], it is interesting that we obtain a PSPACE upper bound for solving randomized sabotage games with an LTL winning condition. The precise formulation of the problem is the following: Given a game arena with a labeling of the vertices and an LTL formula φ and a probability $p \in [0, 1]$, does Runner have a strategy in the sabotage game such that the labeling sequence of the visited vertices satisfies φ with a probability $\geq p$?

On the other hand, this paper extends the known lower bound results of Löding and Rohde [2, 3] (namely, PSPACE-completeness for sabotage games with reachability winning conditions). We transfer this result to the case of games against nature when nature adopts uniform probability distributions.

Our model is closely related to the problem of “dynamic graph reliability” (DGR) defined by Papadimitriou [8]. There, each edge e fails with a probability $p(e, v)$ in each turn, i.e. the probability depends on both the edge e and the current position v of Runner. So in Papadimitriou’s game model, the probabilities of deletion can be adjusted for each edge after each move; his proof of the PSPACE-hardness of DGR heavily depends on such adjustments. (In fact, all problems that are considered in [8, 13] as “games against nature” allow a precise adjustment of the probability for arbitrary large games, so that a reduction from the PSPACE-complete problem *SSAT* [8, 14] is possible. Here, *SSAT* is a stochastic version of SAT, with randomized quantifiers instead of universal quantifiers). In this paper, we assume a uniform probability distribution (e.g. in each turn, one of the n edges is deleted with probability $p = \frac{1}{n}$).

Our second main result says that, in randomized sabotage games with a uniform distribution of failures where exactly one edge is deleted per turn, for any $p \in [0, 1]$ and $\varepsilon > 0$ the following problem is PSPACE-complete: Given a game arena with initial vertex u , a distinguished set F , and a probability p' in the interval $[p - \varepsilon, p + \varepsilon]$, does Runner have a strategy to reach F from u with a probability $\geq p'$?

In this paper, we assume a uniform probability distribution of edge failures in order to strengthen our PSPACE-hardness result. However, for our positive result, namely solving randomized sabotage games with an LTL winning condition in PSPACE, a more general model which allows an adjustment of the probabilities for edge failures might be more appropriate. Our algorithmic solution can be adapted to this non-uniform setting easily (as explained at the end of Section 3).

The remainder of this paper is structured as follows. In Section 2 we introduce the basic notions of randomized sabotage games. In Section 3 we present algorithms for solving randomized sabotage games with reachability objective and LTL specification in PSPACE. Section 4 is concerned with the PSPACE-hardness of the reachability version of randomized sabotage games. Here we start from the construction of Löding and Rohde [2] showing PSPACE-hardness for the original sabotage game. We introduce a infinite family of probabilities $p_{k,n} \in [0, 1]$ with natural numbers k and n as parameters. For every $p_{k,n}$ we reduce the PSPACE-complete problem of *Quantified Boolean Formulae* (QBF) (see [13], problem “QSAT”) to the question of whether, given a randomized sabotage game with u and F , the target set F is reachable from u with a probability of $p_{k,n}$. In order to complete the proof of our main result, we show in Section 5 that the set of the probabilities $p_{k,n}$ is dense in the interval $[0, 1]$, and that the parameters k and n can be computed efficiently such that $p_{k,n} \in [p - \varepsilon, p + \varepsilon]$. In Section 6 we address perspectives which are treated in ongoing work.

The hardness result (Sections 4 and 5) was originally published in the paper “The Complexity of Reachability in Randomized Sabotage Games” [15].

2. The Randomized Sabotage Game

A sabotage game is played on a graph (V, E) , where V is a set of vertices. The vertices are connected by a set of edges, given by $E \subseteq V \times V$. We will assume undirected graphs from now on, i.e. $(u, v) \in E \Rightarrow (v, u) \in E$; however, the ideas presented here also work for directed graphs in the same way. It should also be noted that, while in the “classical” notion of sabotage games

multi-edges are allowed, we will restrict ourselves to single edges only. Clearly, the hardness result presented here also holds for the case of multi-edges. Also, the methods for solving sabotage games in this paper can be used in the same way for sabotage games with multi-edges, so that the problem still belongs to PSPACE.

A *randomized sabotage game* is a tuple $\mathcal{G} = (G, v_{\text{in}})$ with a graph $G = (V, E_{\text{in}})$ and the initial vertex $v_{\text{in}} \in V$. A position of the game is a tuple (v_n, E_n) . The initial position is $(v_{\text{in}}, E_{\text{in}})$. In each turn of the game, the player – called Runner – chooses an outgoing edge (v_n, v_{n+1}) in vertex v_n of position (v_n, E_n) and moves to vertex v_{n+1} . Then, “Nature” acts by throwing a dice with $|E_n|$ sides and removing the chosen edge e from E_n . We define $E_{n+1} := E_n \setminus \{e\}$. After this turn, the new position of the game is (v_{n+1}, E_{n+1}) ; we say that Runner has *reached* the vertex v_{n+1} .

A *play* is a sequence of positions $\pi = (v_0, E_0)(v_1, E_1) \cdots (v_m, E_m)$ where $(v_0, E_0) = (v_{\text{in}}, E_{\text{in}})$ and each step from (v_n, E_n) to (v_{n+1}, E_{n+1}) results from a move of Runner followed by an edge deletion. A play ends if Runner wins by satisfying a given winning condition (e.g. reaching a given goal vertex) or if Runner cannot move anymore, i.e. there does not exist any outgoing edge from Runner’s current vertex. (To keep the definitions in this section simple, we assume that formally a play only ends when Runner cannot move anymore; however, the winner is already determined once Runner satisfies the winning condition.) Clearly, the number of positions of a game is finite, and since edges are only deleted and not added, also each play is finite.

In this work, we consider two kinds of winning conditions. The first one is a *reachability winning condition* where Runner’s objective is to reach a given set of final vertices $F \subseteq V$. The other is a more general *LTL winning condition* which is given by an LTL formula φ and a labeling $L: V \rightarrow 2^{AP}$ of the vertices with atomic propositions from a given set AP ; Runner’s aim is to traverse the game arena in such a way that the labeled sequence of visited vertices fulfills the given LTL formula φ .

In the following we recall the syntax and semantics of the LTL [9, 10]. The syntax of LTL formulas over a set AP of atomic propositions is defined as follows:

- **true** is an LTL formula.
- Each atomic proposition in AP is an LTL formula.
- If φ_1 and φ_2 are LTL formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $X\varphi_1$, and $\varphi_1 \cup \varphi_2$ are LTL formulas.

As usual we define **false** := $\neg\mathbf{true}$, $\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, $F\varphi_1 := \mathbf{true} \cup \varphi_1$, and $G\varphi_1 := \neg F\neg\varphi_1$ for LTL formulas φ_1, φ_2 .

We define the semantics of LTL over a path labeled with atomic propositions in AP . For a path $\rho = v_0v_1 \cdots v_{n-1}$, let $\rho[i] = L(v_i)$ and ρ^i be the path $v_iv_{i+1} \cdots v_{n-1}$. Given a path $\rho \in V^*$, a labeling function $L: V \rightarrow 2^{AP}$, and an LTL formula φ , the satisfiability relation \models_L is defined inductively as follows:

$$\begin{aligned}
\rho &\models_L \mathbf{true} \\
\rho &\models_L a && \text{iff } a \in L(\rho[0]) \\
\rho &\models_L \neg\varphi && \text{iff } \rho \not\models_L \varphi \\
\rho &\models_L \varphi_1 \wedge \varphi_2 && \text{iff } \rho \models_L \varphi_1 \text{ and } \rho \models_L \varphi_2 \\
\rho &\models_L X\varphi && \text{iff } \rho^1 \models_L \varphi \\
\rho &\models_L \varphi_1 \cup \varphi_2 && \text{iff } \exists j \geq 0: \rho^j \models_L \varphi_2 \text{ and } \forall 0 \leq k < j: \rho^k \models_L \varphi_1 .
\end{aligned}$$

In order to apply the satisfiability relation to a play of a sabotage game, we define for a play $\pi = (v_0, E_0)(v_1, E_1) \cdots (v_n, E_n)$ the *trace of π* as

$$\text{trace}(\pi) := v_0 v_1 \cdots v_n .$$

For the probabilistic analysis of randomized sabotage games, we introduce the formal definition of a strategy for Runner. A *strategy* is a (partial) function $\sigma: (V \times E)^+ \rightarrow V$ which maps each play prefix $(v_0, E_0)(v_1, E_1) \cdots (v_n, E_n)$ to the vertex v_{n+1} with $(v_n, v_{n+1}) \in E_n$ to which Runner moves next. A strategy is called *positional* (or *memoryless*) if it only depends on the current position, i.e. it is a function $\sigma: V \times E \rightarrow V$ which maps Runner's current position (v_n, E_n) to the vertex v_{n+1} .

Now we build up the *probability tree* $t_{\mathcal{G}, \sigma}$ for a randomized sabotage game \mathcal{G} and a strategy σ . The probability tree contains the positions (where Runner moves next) which may occur when Runner plays according to the strategy σ . The root of the probability tree is $(v_{\text{in}}, E_{\text{in}})$. From a position (v, E) the successor nodes are all positions (v', E') where v' is the vertex $\sigma(v, E)$ and E' results from E by an edge deletion (a position where Runner cannot move, and where thus $\sigma(v, E)$ is not defined, is a leaf).

Given the probability tree $t_{\mathcal{G}, \sigma}$ and the winning condition of \mathcal{G} (which is either a set F of final vertices for a reachability game or an LTL formula φ for an LTL game with a labeling L), we define the *set of won plays*, i.e. the set of paths from the root to a leaf in the probability tree where Runner wins according to the winning condition, as follows:

$$\text{Plays}(t_{\mathcal{G}, \sigma}, F) := \text{set of all paths } \pi \text{ from the root to a leaf in } t_{\mathcal{G}, \sigma} \\ \text{where } \text{trace}(\pi) \text{ contains a vertex in } F ;$$

$$\text{Plays}(t_{\mathcal{G}, \sigma}, (L, \varphi)) := \text{set of all paths } \pi \text{ from the root to a leaf in } t_{\mathcal{G}, \sigma} \\ \text{with } \text{trace}(\pi) \models_L \varphi .$$

For a play $\pi = (v_0, E_0)(v_1, E_1) \cdots (v_n, E_n)$ in $\text{Plays}(t_{\mathcal{G}, \sigma}, F)$ respectively in $\text{Plays}(t_{\mathcal{G}, \sigma}, (L, \varphi))$, let $\text{Prob}_{t_{\mathcal{G}, \sigma}}(\pi)$ be the *probability of π* , i.e.

$$\text{Prob}_{t_{\mathcal{G}, \sigma}}(\pi) := \frac{1}{|E_0|} \cdot \frac{1}{|E_1|} \cdots \frac{1}{|E_{n-1}|} .$$

Given a randomized sabotage game \mathcal{G} with a goal set F resp. with a labeling of the game graph and an LTL formula φ , we say that *Runner wins the reachability game with probability p* resp. *Runner wins the LTL game with probability p* if there exists a strategy σ for Runner such that

$$p = \sum_{\pi \in W} \text{Prob}_{t_{\mathcal{G}, \sigma}}(\pi)$$

where $W = \text{Plays}(t_{\mathcal{G}, \sigma}, F)$ resp. $W = \text{Plays}(t_{\mathcal{G}, \sigma}, (L, \varphi))$. Literally, Runner wins the reachability game resp. the LTL game with probability p if he has a strategy σ such that the probabilities of all plays in $\text{Plays}(t_{\mathcal{G}, \sigma}, F)$ resp. $\text{Plays}(t_{\mathcal{G}, \sigma}, (L, \varphi))$ sum up to p .

Clearly, the LTL winning condition is more general since each reachability objective can be expressed with an LTL formula without changing the game arena, but not conversely (a reachability winning condition can be expressed with the LTL formula $F a$ when exactly the final vertices are labeled with a).

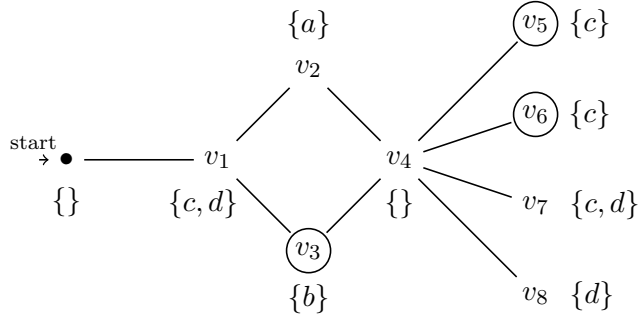


Figure 1: A randomized sabotage game.

Example 1. Consider the game arena in Figure 1. First, we analyze the randomized reachability sabotage game where Runner (starting in the initial vertex) has to reach one of the circled final vertices. We ask whether Runner wins the reachability game with a probability ≥ 0.95 . It is easy to see that there is a strategy for Runner which maximizes his winning probability. In the first turn Runner has to move to v_1 ; in the second turn Runner tries to move to the final vertex v_3 if the edge (v_1, v_3) has not been deleted, and he moves to v_2 otherwise; at v_2 he moves to v_4 if possible; if Runner reaches v_4 , he moves to one of the final vertices v_3, v_5, v_6 via a non-deleted edge.

Runner does not win with probability 1; it may happen with a probability > 0 that after Runner's first move the edge (v_1, v_3) is deleted (for which the probability is $\frac{1}{9}$) and after his second move the edge (v_2, v_4) is deleted (for which the probability is $\frac{1}{8}$). So, Runner loses the game with a probability of $\frac{1}{72}$ and hence he wins with a probability of $\frac{71}{72} > 0.98$, which is higher than the probability of 0.95 we asked for.

Now, consider the LTL formula $F(a \wedge F c) \vee F(b \wedge F d)$ and the LTL sabotage game over the same game arena with the labeling depicted in Figure 1 with atomic propositions over the set $AP = \{a, b, c, d\}$. We ask whether Runner can guarantee to win the LTL game surely, i.e. with probability 1. In fact, Runner has such a strategy with which he wins against every possible sequence of edge deletions. In the first turn Runner has to move to v_1 ; in the second turn Runner moves to v_3 if edge (v_1, v_2) or edge (v_2, v_4) has been deleted, and he moves to v_2 otherwise; at v_2 (resp. v_3) Runner moves to v_1 if the edge (v_1, v_2) (resp. (v_1, v_3)) is available and to v_4 otherwise; at v_4 Runner moves to v_5, v_6 , or v_7 if he has visited v_2 , and he moves to v_7 or v_8 otherwise.

Note that Runner does not have a *positional* strategy to win this LTL game with probability 1. Assume the following two sequences of edge deletions: $(v_1, v_3), (v_1, v_2), (v_4, v_7)$ and $(v_1, v_2), (v_1, v_3), (v_4, v_7)$. Both sequences lead to the same position, where Runner is at vertex v_4 ; but the first sequence forces Runner to move via v_2 while the second forces Runner to move via v_3 . Then, at v_4 , Runner has to move either to one of the vertices v_5, v_6 or to the vertex v_8 depending on the sequence of previous edge deletions.

Let p be an arbitrary number in $[0, 1]$. The problem *randomized sabotage reachability game for probability p* is the following:

Given a randomized sabotage game $\mathcal{G} = ((V, E_{\text{in}}), v_{\text{in}})$ and a designated set $F \subseteq V$, does Runner win the reachability game with a probability $\geq p$?

The problem *randomized sabotage LTL game for probability p* can be formulated analogously:

Given a randomized sabotage game $\mathcal{G} = ((V, E_{\text{in}}), v_{\text{in}})$, a labeling $L: V \rightarrow 2^{AP}$ of the vertices, and an LTL formula φ , does Runner win the LTL game with a probability $\geq p$?

Löding and Rohde have already shown that solving classical sabotage games with a reachability winning condition is PSPACE-complete [2]. So, solving randomized sabotage games with a reachability or LTL winning condition for probability $p = 1$ is also PSPACE-hard. On the other hand, the problem of whether Runner wins a randomized sabotage reachability game with a probability $p > 0$ is decidable in linear time, because Runner wins with a probability > 0 iff there is a path from the initial to a final vertex.

In this article, we will show that on the one hand randomized sabotage games can still be solved in PSPACE even with LTL winning conditions and on the other hand the probabilistic setting does not make the game problem easier in general.

The first result is the following:

Theorem 2. *Given a randomized sabotage game \mathcal{G} with a reachability (resp. an LTL) winning condition and $p \in [0, 1]$, it is decidable in PSPACE whether Runner wins \mathcal{G} with a probability $\geq p$. Moreover, given a randomized sabotage game \mathcal{G} with a reachability (resp. an LTL) winning condition, Runner’s maximal winning probability for \mathcal{G} can be computed in polynomial space.*

The proof is easy, using standard techniques from the analysis of finite games. The idea is to generate the game tree in a depth-first manner, with a storage for paths (and some auxiliary information). We will provide such algorithms, which require only polynomial space, in Section 3.

The second result says that solving randomized reachability sabotage games remains PSPACE-hard even if we restrict the probability to any interval. For any fixed $p \in [0, 1]$ and $\varepsilon > 0$, the randomized reachability game problem for a probability p' which may vary in the interval $[p - \varepsilon, p + \varepsilon]$ is PSPACE-hard (and thus PSPACE-complete due to Theorem 2). More precisely:

Theorem 3. *For each fixed $p \in [0, 1]$ and $\varepsilon > 0$, the following is PSPACE-complete: Given a randomized sabotage game \mathcal{G} with goal set F and a probability $p' \in [p - \varepsilon, p + \varepsilon]$, does Runner win \mathcal{G} with a probability $\geq p'$?*

For the hardness proof we use a parametrized reduction of the problem *Quantified Boolean Formulae* (QBF). For each QBF-sentence φ , we construct a family of instances $\mathcal{G}_{\varphi, k, n}$ and $p_{k, n}$ (with natural numbers k and n as parameters) such that, for each k and n , the sentence φ is true iff, over $\mathcal{G}_{\varphi, k, n}$ with goal set F , Runner wins with a probability $\geq p_{k, n}$. Furthermore, we guarantee that, given p and $\varepsilon > 0$, the probability $p_{k, n}$ can be chosen in $[p - \varepsilon, p + \varepsilon]$ for suitable k and n , and that this choice can be made in polynomial time. Section 4 provides the indicated reduction, and in the subsequent section, we address the question of the distribution and efficient computation of the probabilities $p_{k, n}$.

3. Solving Randomized Sabotage Games in PSPACE

Solving classical sabotage games, where Runner plays against a malicious saboteur, have been shown to be PSPACE-complete for various winning conditions [2, 6]: reachability, Hamilton path, and complete search. The membership of these problems to PSPACE is shown in [6] by providing an alternating polynomial time algorithm. Such an algorithm guesses Runner’s moves, and each edge that is blocked next is chosen universally, i.e. the algorithm has to accept for all possible choices of the edge deletion. However, we consider edge deletions as random events; so, we cannot choose

the edge universally. Nevertheless, we still have to inspect each possible edge deletion to compute Runner’s winning probability. So, our algorithm can be seen as a deterministic variant of the alternating algorithm in [6] where the game tree is generated “on the fly” in a depth-first manner. This allows us to memorize Runner’s winning probability in the currently inspected subgame for each possible edge deletion, which is not achievable with the alternating algorithm in [6].

Also, it turns out that solutions for reachability games are essentially simpler than for LTL games in general. If Runner has a strategy to win a reachability game with a certain probability, he also has a *positional* strategy to win with at least this probability. For the more general LTL games, we have already seen in Example 1 that positional strategies might not be sufficient for Runner to win a game with a given probability.

Therefore, it is convenient that we first give a polynomial space algorithm for solving randomized sabotage reachability games. Thereafter, we treat the generalization for solving randomized sabotage games with an LTL winning condition. At the end of this section, we mention some possible extensions of the presented algorithms to solve sabotage games in more general settings.

3.1. Solving Randomized Sabotage Reachability Games

In order to show that the randomized sabotage reachability game problem is decidable in polynomial space, we present the algorithm `solve-reachability-game` (Algorithm 1), which builds up and traverses the game tree in a depth-first manner.

More precisely, for a reachability sabotage game $\mathcal{G} = (G_{\text{in}}, v_{\text{in}})$ with a set F of final vertices, the *game tree* consists of both all reachable positions where it is Runner’s turn and all reachable intermediate positions where Nature acts next. Each tree node has the form $(G, v, 0/1)$ consisting of the current game arena G , Runner’s current position v , and a bit which is 1 iff Runner moves next. The root of the game tree is $(G_{\text{in}}, v_{\text{in}}, 1)$, where Runner starts to move. From a position $(G, v, 1)$ with $G = (V, E)$ and $v \notin F$ where Runner moves, the successor nodes are all positions $(G, v', 0)$ with $(v, v') \in E$ (a position (v, E) , with $v \in F$ or $(v, v') \notin E$ for all v' , is a leaf). Now, the successors of $(G, v', 0)$ are the positions $(G', v', 1)$ with $G' = (V, E')$ where E' results from E by an edge deletion.

To each node of Runner our algorithm assigns the probability for Runner to win the subgame starting in this node. These probabilities are computed inductively in the obvious way, starting with 1 and 0 at a leaf $(G, v, 1)$ depending on whether $v \in F$ or not. For an inner node s in the game tree with successors s_1, \dots, s_k with winning probabilities p_i ($1 \leq i \leq k$), the algorithm assigns to s the winning probability p_{out} as follows. If s is a tree node of Nature, then $p_{\text{out}} := \frac{1}{k} \sum_i p_i$ (each edge is hit with the same probability). If s is a tree node of Runner, $p_{\text{out}} := \max_i p_i$ (Runner moves to the vertex with the maximal winning probability). Then, Runner wins \mathcal{G} with the probability p_{out} assigned to the root node.

So, the call `solve-reachability-game`($G, v_{\text{in}}, F, 1$) returns Runner’s maximal winning probability p_{out} for the reachability game. Since the algorithm builds up and traverses the entire game tree, the computation may require exponential time in general. However, the required space is still polynomial. The memory that is needed for each recursive call is linear in $|G| := |V| + |E|$. Since the depth of the game tree and hence the recursion depth of `solve-reachability-game` is bounded by the number of edges $|E|$, the algorithm runs in $O(|G|^2)$ space.

Hence, given a randomized sabotage reachability game and p , one can decide in PSPACE whether Runner wins with a probability $\geq p$. For that, one can call `solve-reachability-game` and compare the output value p_{out} with p . So, we showed Theorem 2 for reachability games.

Algorithm 1: solve-reachability-game

Input: game arena G , current position u of Runner, set F of final vertices,
boolean variable $runners_turn$

Output: winning probability p_{out} for Runner in the game (G, u) for reaching a final vertex
in F

```
1 if  $u \in F$  then
2    $p_{out} := 1$ ;
3 else
4   if  $u$  has no outgoing edges then
5      $p_{out} := 0$ ;
6   else (*  $u$  has at least one outgoing edge *)
7     if  $runners\_turn$  then (* Runner moves next *)
8        $p_{out} := 0$ ;
9       let  $m$  be the number of outgoing edges from  $u$ ;
10      for  $i := 1$  to  $m$  do
11        let  $(u, v_i)$  be the  $i$ -th outgoing edge from  $u$ ;
12         $p_{out} := \max\{p_{out}, \text{solve-reachability-game}(G, v_i, F, 0)\}$ ;
13      end
14    else (* Nature acts next *)
15       $p_{out} := 0$ ;
16      let  $n$  be the total number of edges in  $G$ ;
17      for  $j := 1$  to  $n$  do
18        let  $G_j$  be the game graph resulting from  $G$  by deletion of the  $j$ -th edge  $e_j$ ;
19         $p_{out} := p_{out} + \frac{1}{n} \cdot \text{solve-reachability-game}(G_j, u, F, 1)$ ;
20      end
21    end
22  end
23 end
24 return  $p_{out}$ ;
```

Remark 4. We can adapt Algorithm 1 to compute an optimal strategy for Runner (i.e. the strategy maximizes Runner’s winning probability) that is *positional*. For that, in the for loop for Runner’s move (lines 10–13), where the current game arena is $G = (V, E)$, we have to memorize the v_i for which the recursive call $\text{solve-reachability-game}(G, v_i, F, 0)$ maximizes the probability p_{out} . Then, an optimal positional strategy for Runner is the following: when the play is in position (E, u) , Runner moves to vertex v_i . So, in a randomized reachability sabotage game, positional strategies are sufficient, i.e. if Runner has a strategy to win the reachability game with probability p , he also has a positional strategy to win with at least probability p .

3.2. Solving Randomized Sabotage LTL Games

As mentioned in the introduction, solving the LTL game problem in PSPACE is a non-trivial problem. In the static case (without sabotage) this problem is double exponential time complete for two-player games and Markov decision processes [11, 12]. Nevertheless, solving randomized sabotage LTL games in PSPACE is still possible due to the finiteness of our plays (in contrast to

the models in [11, 12]).

The most obvious approach for solving the randomized LTL game might be traversing the game tree in a depth-first manner and computing the probabilities of each subformula at each node. However, this approach does not work, since one choice may maximize the probability for fulfilling one subformula while another choice may maximize the probability for fulfilling another subformula. So, at a node v , one has to store the probabilities for all paths in the game tree rooted at v in the worst case.

Nevertheless, since the height of the game tree is bounded by the number of edges in the game arena, we can compute at each leaf in polynomial time whether this sequence satisfies the given LTL formula. Therefore, our algorithm `solve-LTL-game` (Algorithm 2) stores the *history* of Runner's moves, i.e. the sequence of vertices which Runner has visited in a play. The algorithm traverses the game tree in a depth-first manner and, at each backtracking step, computes the probability for which Runner wins the probabilistic LTL subgame starting from this node. The winner of a play is computed at each leaf by the function `play-satisfies-formula`, which returns true iff the sequence of labelings of the play satisfies the LTL formula, i.e.

$$\text{play-satisfies-formula}(\rho, L, \varphi) := \begin{cases} \text{true} & \text{if } \rho \models_L \varphi \\ \text{false} & \text{otherwise} . \end{cases}$$

Then again, for each node where Nature acts next, we take the arithmetic mean over the successors; for a node where Runner moves next we take the maximal probability of the successors. Runner wins the LTL game with the probability p_{out} that is assigned to the root node.

So, given a randomized sabotage game $\mathcal{G} = (G, v_{\text{in}})$ over an arena $G = (V, E)$, a labeling function $L: V \rightarrow 2^{AP}$, and an LTL formula φ , the call `solve-LTL-game`($G, v_{\text{in}}, L, \varphi, 1$) returns Runner's winning probability p_{out} for the LTL game \mathcal{G} . For the complexity analysis of the algorithm, we first note that the additional memory that is needed per call is linear in $|V| + |E|$ if the last node of the current play prefix is an inner node of the game tree (either one node is added to the history ρ or one edge in $|E|$ is removed). For a leaf the function `play-satisfies-formula` is called, which requires at most $O(|E| \cdot |\text{cl}(\varphi)|)$ space, where $\text{cl}(\varphi)$ denotes the set of all subformulas of φ . Since the depth of the game tree and hence the recursion depth of `solve-LTL-game` is bounded by the number of edges $|E|$, the algorithm requires the space

$$O(|E| \cdot (|V| + |E|) + |E| \cdot |\text{cl}(\varphi)|) \subseteq O(|G|^2 + |E| \cdot |\text{cl}(\varphi)|) .$$

It remains to be shown that the function `play-satisfies-formula` requires at most $O(|E| \cdot |\text{cl}(\varphi)|)$ space. More precisely, for a history $\rho = u_1 \cdots u_n$, the function `play-satisfies-formula`(ρ, L, φ) runs in $O(n \cdot |\text{cl}(\varphi)| \cdot (n + |L|))$ time and in $O(n \cdot |\text{cl}(\varphi)|)$ space. This can be done by filling a matrix with $|\text{cl}(\varphi)|$ rows and n columns; each (i, j) -entry of this matrix is a bit which is set to 1 iff ρ^j satisfies the i -th subformula. The computation of each matrix entry is in $O(|L| + n)$ time and constant space: for an atomic proposition, we have to inspect at most a bit vector of size $|L|$; for all other entries, we have to inspect at most two other rows of the matrix (this is the case for an until-formula).

Hence, given a randomized sabotage LTL game and p , one can decide in PSPACE whether Runner wins with a probability $\geq p$. For that, one can call `solve-LTL-game` and compare the output value p_{out} with p . So, we showed that Theorem 2 also holds for LTL games.

Algorithm 2: solve-LTL-game

Input: game arena G , the history ρ of Runner's moves including his current position, LTL formula φ , boolean variable *runners_turn*

Output: winning probability p_{out} for Runner for satisfying φ in the original game under the assumption that he played ρ and reaches G

```
1 let  $u$  be Runner's current position after playing  $\rho$ , i.e.  $\rho = \rho'u$  for some  $\rho'$ ;  
2 if  $u$  has no outgoing edges then  
3   if play-satisfies-formula( $\rho, L, \varphi$ ) then  
4      $p_{\text{out}} := 1$ ;  
5   else  
6      $p_{\text{out}} := 0$ ;  
7   end  
8 else (*  $u$  has at least one outgoing edge *)  
9   if runners_turn then (* Runner moves next *)  
10     $p_{\text{out}} := 0$ ;  
11    let  $m$  be the number of outgoing edges from  $u$ ;  
12    for  $i := 1$  to  $m$  do  
13      let  $(u, v_i)$  be the  $i$ -th outgoing edge from  $u$ ;  
14       $p_{\text{out}} := \max\{p_{\text{out}}, \text{solve-LTL-game}(G, \rho v_i, L, \varphi, 0)\}$ ;  
15    end  
16  else (* Nature acts next *)  
17     $p_{\text{out}} := 0$ ;  
18    let  $n$  be the total number of edges in  $G$ ;  
19    for  $j := 1$  to  $n$  do  
20      let  $G_j$  be the game graph resulting from  $G$  by deletion of the  $j$ -th edge  $e_j$ ;  
21       $p_{\text{out}} := p_{\text{out}} + \frac{1}{n} \cdot \text{solve-LTL-game}(G_j, \rho, L, \varphi, 1)$ ;  
22    end  
23  end  
24 end  
25 return  $p_{\text{out}}$ ;
```

Remark 5. We have already seen in Example 1 that positional strategies might not be sufficient to win LTL games with a given probability. Nevertheless, we can adapt Algorithm 2 to compute an optimal strategy for Runner (i.e. the strategy maximizes Runner's winning probability) that only needs *finite memory*. For that, in the for loop for Runner's move (lines 12–15), where the current game arena is $G = (V, E)$ and the history of the play is $\rho = u_1 \cdots u_n$, we have to memorize the v_i for which the recursive call $\text{solve-LTL-game}(G, \rho v_i, L, \varphi, 0)$ maximizes the probability p_{out} . We can store the strategy by assigning to the pair $((E, u_n), \rho)$ of position and history the vertex v_i (for which p_{out} is maximal). Then, an optimal finite memory strategy for Runner is the following: when the play is in position (E, u_n) after Runner moved according to the history ρ , Runner moves to vertex v_i . Since the length of a sequence ρ , on which the strategy depends, is at most $|E|$, finite memory strategies are sufficient to maximize the winning probability in LTL sabotage games.

3.3. Solving More General Randomized Sabotage Games

To get a complete picture, we briefly mention some possible extensions of the presented algorithms to solve sabotage games in more general settings. On the one hand, we remark that Algorithm 2 for solving sabotage games with an LTL winning condition only checks at each leaf of the game tree whether the generated path satisfies the given LTL formula. By replacing the function `play-satisfies-formula`, the algorithm can be adapted to check more general specifications of linear time properties (e.g. regular expressions). Indeed, for every linear time property whose membership problem is decidable in PSPACE, the corresponding randomized sabotage game problem can also be solved in PSPACE. For example, the complete search and Hamilton path winning conditions (see [2]) can be checked in this way (one has to check whether the generated path contains each vertex of the game arena at least once resp. exactly once).

On the other hand, we note that, for convenience, we presented our algorithms for game graphs with single edges only (rather than multi-edges). Also, we confined ourselves to a uniform probability distribution of edge failures and the assumption that Nature deletes exactly one edge per turn. Both algorithms (`solve-reachability-game` and `solve-LTL-game`) can be adapted easily to generalized settings such that solvability in PSPACE is preserved.

1. For solving randomized sabotage games with *multi-edges*, we can replace the edge relation by a multiplicity function $e: V \times V \rightarrow \mathbb{N}$ where $e(u, v)$ denotes the multiplicity of the edge (u, v) . Note that $e(u, v) = e(v, u)$ in the case of undirected edges. Instead of G_j being the game graph resulting from G by deleting the j -th edge e_j , it results from G by decreasing the multiplicity function for the edge e_j by one (see [2]).
2. For solving randomized sabotage games with *non-uniform distributions of edge failures*, let us assume that for each position (v, E) of the game a distribution of edge failures $p_v(e) \mapsto [0, 1]$ with $\sum_{e \in E} p_v(e) = 1$ is given. Then, in line 19 of Algorithm 1 (resp. line 21 of Algorithm 2) we just have to replace the factor $\frac{1}{n}$ by $p_u(e_j)$.
3. For solving randomized sabotage games in which *k edges fail in each turn*, it suffices to replace the boolean variable `runners_turn` by a modulo- $(k+1)$ counter (so that Nature deletes k edges in succession). With some work our algorithms can also be adapted to the case that in each turn the number of edge deletions is chosen randomly (in $\{1, \dots, k\}$). For that, the algorithm has to calculate the probability for all k cases recursively and take the arithmetic mean of these probabilities. However, we have to require that at least one edge is deleted per turn; otherwise we cannot guarantee the termination of our algorithms.

4. PSPACE-Hardness of the Reachability Game

In order to prove the PSPACE-hardness of the Reachability Game, we use a parametrized reduction from the problem Quantified Boolean Formulae (QBF), which is known to be PSPACE-complete (cf. [13], problem “QSAT”). The reduction uses parts of the construction of Löding and Rohde [2]. The basic strategy is to construct an arena in such a way that, in a first part of the game, Runner can select the assignments for existentially quantified variables of the formula, and that he is forced to choose certain assignments for the universally quantified variables. Then, this assignment is verified in a second part.

Formally, an instance of the problem QBF is a special quantified boolean formula; more precisely: given a boolean expression ϑ in conjunctive normal form over boolean variables x_1, \dots, x_m , is $\exists x_1 \forall x_2 \dots Q_m x_m \vartheta$ true? Without loss of generality, one requires the formula to start with an

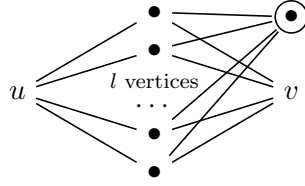


Figure 2: An l -edge from u to v .

existential quantifier. If m is even, $Q_m = \forall$; otherwise $Q_m = \exists$. For each instance φ of QBF, we construct a game arena $\mathcal{G}_{\varphi,k,n}$ and a rational number $p_{k,n}$ such that φ is true iff Runner has a strategy to reach a final vertex in $\mathcal{G}_{\varphi,k,n}$ exactly with probability $p_{k,n}$. Thereby the parameter k is an arbitrary natural number ≥ 2 , and the parameter $n \in \mathbb{N}$ essentially has to be greater than the size of φ , i.e. $n \geq c \cdot |\varphi|$ for some constant c . If Runner plays suboptimally or if the formula is false, the maximal probability of winning is strictly lower than $p_{k,n}$; so the reduction meets the formulation of our game problem.

The arena consists of four types of subparts or “gadgets”: the parametrization, existential, universal, and verification gadgets. In the parametrization gadget, one can, by adding or removing edges, adjust the probability $p_{k,n}$.

The outline of the proof is as follows. We first introduce a construction to simulate a kind of multi-edge; this is convenient for a feasible analysis of the probabilistic outcome in a framework where only single edges are allowed. Then, we briefly recall the construction for the existential, universal, and verification gadgets [2], and adapt the argument to meet our model of a play against nature. In a second step, we introduce the parametrization gadget to adjust the probability $p_{k,n}$. Finally, we use this construction to prove our main result (Theorem 3).

4.1. The l -Edge Construction

In the following proofs, it will be necessary to link two vertices u and v in such a way that the connection is rather strong, i.e. there needs to be a number of several subsequent deletions until Runner is no longer able to move from u to v or vice versa. This is achieved by the construction shown in Figure 2, which we will call an “ l -edge” from now on ($l \geq 1$). The circled vertex is a goal belonging to the set F of final vertices. Here, if after l deletions all of the l edges between u and the middle vertices are deleted, Runner is disconnected from v . If $l - 1$ or less edges have been deleted anywhere in the game graph, there is at least one path from u over some middle vertex to v and an edge between that middle vertex and the final vertex. Then, Runner can move from u (resp. v) to that middle vertex and is able to reach v (resp. u) in the next step, or he wins immediately by moving to the (circled) final vertex.

Lemma 6. *Given a randomized sabotage game with an l -edge between two nodes u and v , Runner can guarantee to reach v via this l -edge iff he can guarantee to reach u within $l - 1$ moves.*

For a sufficiently high l , depending on the structure of the game arena, it is clear that Runner cannot be hindered from winning by edge deletions in an l -edge; such l -edges will be called “ ∞ -edges” to indicate that destruction of the connection can be considered impossible. (For classical sabotage games, l can be bounded by the total number of vertices in the game arena, because if Runner has a winning strategy in a classical sabotage game, he has also a winning strategy without

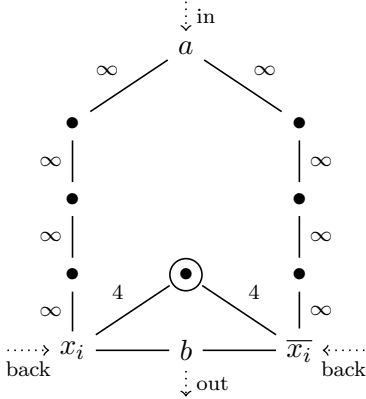


Figure 3: The \exists -gadget for x_i with i odd.

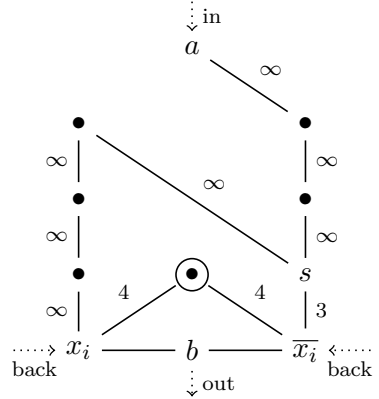


Figure 4: The \forall -gadget for x_i with i even.

visiting any vertex twice [2]. For the constructions in this paper where ∞ -edges are used, it will be sufficient to consider ∞ -edges as 4-edges.)

Remark 7. In order to sharpen the hardness result of this paper to randomized sabotage games with a *unique* goal, one may intend to merge final vertices to one vertex. But this is not always possible. Consider an l -edge to a final vertex v . Since we do not consider graphs with multi-edges, v cannot be merged with the other final vertex from the l -edge construction without breaking Lemma 6. For this reason, in this paper, PSPACE-hardness is shown only for randomized sabotage games with at least two final vertices.

4.2. Existential, Universal, and Verification Gadgets

In this section, we briefly recall constructions from [2] to have a self-contained exposition. We introduce the existential and universal gadgets (applied according to the quantifier structure of the given formula), and the verification gadget (corresponding to its quantifier-free kernel).

The Existential Gadget. Intuitively, the existential component allows Runner to set an existential-quantified variable to true or false. The gadget is depicted in Figure 3. The node a is the input vertex for this gadget, and x_i (resp. \bar{x}_i) is the variable vertex of this component, which Runner intends to visit if he sets x_i to false (resp. true). The vertex b is the exit node of this gadget; it coincides with the in-vertex of the next gadget (i.e. the universal gadget for x_{i+1} , or the verification gadget if x_i is the last quantified variable). The “back”-edges from x_i and \bar{x}_i lead directly to the last gadget of the construction, the verification gadget. Later, Runner possibly move back via these edges to verify his assignment of the variable x_i . (We will see later that taking these edges as a shortcut, starting from the existential gadget, directly to the verification gadget, is useless for Runner.)

Of course, Runner has a very high probability of winning the game within the existential gadget (especially in an l -edge construction for an ∞ -edge). But we are only interested in the worst-case scenario, where edges are deleted in the following precise manner.

When it is Runner’s turn and he is currently residing in vertex a , he will move either left or right and can reach x_i (resp. \bar{x}_i) in four turns. When Runner moves towards x_i (resp. \bar{x}_i), the 4-edge from x_i (resp. \bar{x}_i) to the (circled) final vertex may be subsequently subject to deletion so

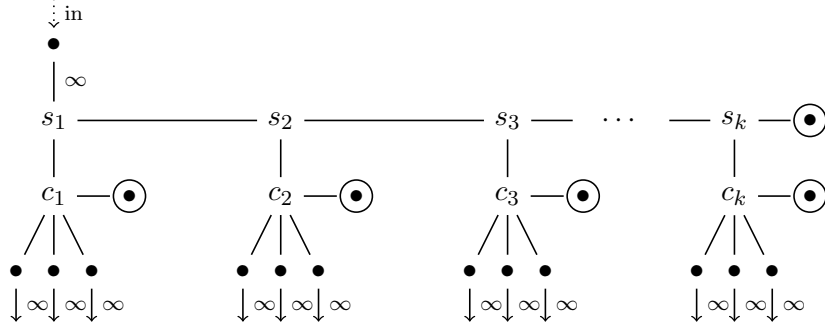


Figure 5: The verification gadget for a formula with k clauses.

that Runner ends up at node x_i (resp. \bar{x}_i) with no connection to the final vertex left. If Runner then moves towards \bar{x}_i (resp. x_i) and the edge between b and \bar{x}_i (resp. x_i) is deleted, he is forced to exit the gadget via b and move onwards. The 4-edge from \bar{x}_i (resp. x_i) to the final vertex remains untouched so far. If Runner is later forced to move back to x_i or \bar{x}_i from the verification gadget, he can only guarantee a win in one of these vertices.

The Universal Gadget. In the universal component a truth value for the all-quantified variables is chosen arbitrarily, but this choice can be considered to Runner's disadvantage in the worst case. Runner can be forced to move in one or the other direction and has to set x_i to true or false, respectively. The gadget is depicted in Figure 4. A path through this gadget starts in node a and is intended to exit via node b , which coincides with the in-vertex of the next gadget (i.e. the existential gadget for x_{i+1} , or the verification gadget if x_i is the last quantified variable). Again, only the worst cases are important for now; Runner is able to win the game immediately in all other cases. Clearly, Runner is going to move in the direction of vertex s . There are two interesting scenarios, which may happen with a probability > 0 .

In the first scenario, the 3-edge to \bar{x}_i is deleted completely. Then, Runner can only guarantee to leave the gadget at b via x_i , but no visit of \bar{x}_i or the (circled) final vertex, because the 4-edge from x_i to the final vertex and the edge between b and \bar{x}_i may be deleted successively. In this case, the 4-edge between \bar{x}_i and the final vertex remains untouched.

In the second case, only the 4-edge from \bar{x}_i to the final vertex is subject to deletion. At s , Runner is intended to move downward to \bar{x}_i and leave the gadget at b . Thereby the 4-edge between \bar{x}_i and the final vertex (which was already reduced to a 1-edge) is deleted completely, and after this, the edge between b and x_i is deleted. Consequently, the 4-edge from x_i to the final vertex is untouched. If Runner "misbehaves" in the sense that he moves from s to the left, it may happen that the final vertex becomes completely disconnected from both x_i and \bar{x}_i ; in this case, Runner cannot win in this vertex if he is forced to move back to x_i or \bar{x}_i from the verification gadget.

The Verification Gadget. The verification gadget is constructed in such a way that, when Runner arrives here, he can only force a win if the assignment for the variables which has been chosen beforehand satisfies the quantifier-free kernel of the formula.

The verification gadget for a QBF-formula with k clauses C_1, \dots, C_k is depicted in Figure 5. Its in-vertex coincides with exit vertex b of the last existential/universal gadget. For a clause $C_i = (\neg)x_{i_1} \vee (\neg)x_{i_2} \vee (\neg)x_{i_3}$, there are three paths, each from c_i via a single edge and an ∞ -edge

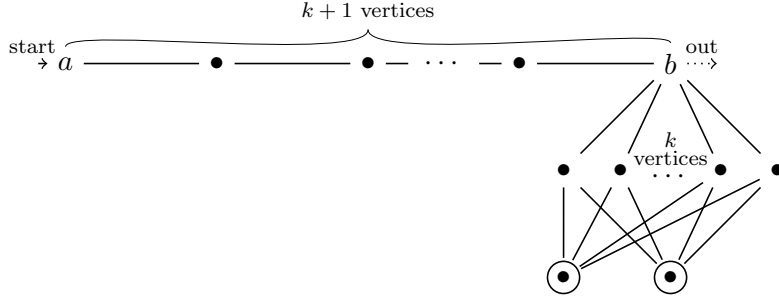


Figure 6: The parametrization gadget \mathcal{H}_k .

(the literal edge L_{ij}) back to the variable vertex x_{i_j} (resp. $\overline{x_{i_j}}$) in the corresponding gadgets. Again, looking at the interesting scenarios is important.

Assume that Runner has chosen the appropriate assignments of the variables for a satisfiable formula. He reaches the first selection vertex s_1 via the ∞ -edge from the last existential/universal gadget. If Runner is in s_i and the edge to c_i is deleted, he has to proceed to s_{i+1} . Now, assume that the edge between s_i and s_{i+1} is removed. Then, Runner is forced to move towards c_i . If the quantifier-free kernel of the QBF-formula is satisfied with the chosen assignment of Runner, then there is at least one literal that satisfies the clause C_i . Runner chooses to move alongside the corresponding literal edge L_{ij} back to x_{i_j} (resp. $\overline{x_{i_j}}$), into the gadget where he has chosen the assignment (and wins there by moving to the final vertex). In any other case Runner is able to win immediately by moving via s_k or via s_i, c_i to the (circled) final vertex. Note that he only has a chance of *always* winning if his chosen assignment actually fulfills the quantifier-free kernel of the formula.

If he did not choose a correct assignment or if the formula is not satisfiable, there is at least one clause that falsifies the QBF-formula, say clause c_i . If he is forced to go to c_i , the path that leads him back to the final vertex of an existential/universal gadget can be cut off (and hence Runner may lose the game).

As a side remark, one should note that it is never optimal for Runner to take a “back”-edge (i.e. a literal edge L_{ij}) as a shortcut, moving directly from some x_i (resp. $\overline{x_i}$) of an existential/universal gadget to the verification gadget. In this case, the edge connecting c_i and the ∞ -edge from x_i (resp. $\overline{x_i}$) to the verification gadget may be destroyed. Then, Runner has to move back and loses his chance of always winning the game.

We introduced so far the construction from [2], which suffices to show PSPACE-hardness for $p = 1$; namely, using the gadgets above, Runner has a winning strategy iff the given formula is true. For a QBF-formula φ , we call the game arena of this construction \mathcal{G}_φ .

4.3. The Parametrization Gadget

The parametrization gadget is the initial part of the game arena that is constructed; it is used to “adjust” the overall winning probability of Runner. Runner starts from the initial vertex in this gadget. For each $k \geq 1$, we define the parametrization gadget \mathcal{H}_k ; it is depicted in Figure 6.

We reduce the question of whether the QBF-formula φ is true to the reachability problem over certain game arenas that result from combining \mathcal{H}_k with \mathcal{G}_φ as a graph $\mathcal{H}_k \circ \mathcal{G}_\varphi$. For that, the out-vertex b in \mathcal{H}_k is identified with the in-vertex a of the first existential gadget of \mathcal{G}_φ . Assume \mathcal{G}_φ has n_0 edges. We get modifications of \mathcal{G}_φ with any number $n \geq n_0$ of edges by adding artificial

extra edges (without changing the behavior in the discussed worst case); for instance, this can be achieved by adding a path with a dead end. We call this game arena \mathcal{G}_φ^n . In the following, we work with

$$\mathcal{G}_{\varphi,k,n} := \mathcal{H}_k \circ \mathcal{G}_\varphi^n .$$

Since \mathcal{H}_k has $4k$ edges, the overall number of edges in our game arena $\mathcal{G}_{\varphi,k,n}$ is $4k + n$. Let

$$p_{k,n} := \text{probability for Runner to traverse the parametrization gadget } \mathcal{H}_k \text{ of } \mathcal{G}_{\varphi,k,n} .$$

Lemma 8. *Runner wins the randomized reachability game over $\mathcal{G}_{\varphi,k,n}$ for probability $p_{k,n}$ iff the QBF-formula φ is true.*

PROOF. First note the following. If Runner resides at vertex b , and in \mathcal{H}_k exactly the k edges below vertex b have been deleted so far, then Runner wins with probability 1 iff φ is true (this follows immediately from the classical construction by Löding and Rohde [2]).

Now assume that φ is true. In the parametrization gadget Runner starts at node a and obviously moves towards b . Clearly, if any edge between his current position and b is deleted, he loses the game immediately. However, if he succeeds in getting to b , he will always win the game. To show this, we first assume the case that, in Runner's k steps towards b , only edges in \mathcal{H}_k were subject to deletion. Then, Runner always wins by moving in the first existential gadget and traversing \mathcal{G}_φ^n , as mentioned before. If we assume the other case that at least one of the k deletions took place outside of \mathcal{H}_k , there is at least one edge leading from b downward to some middle node, say b' , and there are at least two edges leading from b' to the two final vertices in the parametrization gadget. Then, Runner wins by moving from b downward to b' and then, depending on the next deletion, by moving to one of the two final vertices. In both cases, Runner wins over $\mathcal{G}_{\varphi,k,n}$ exactly with the probability $p_{k,n}$ of traversing the parametrization gadget \mathcal{H}_k from node a to node b .

Conversely, assume now that φ is false, and that only the k edges below vertex b in \mathcal{H}_k are subject to deletion while Runner moves towards b (this may happen with a probability > 0). Then, Runner's only chance to win from b is by moving towards some final vertex in \mathcal{G}_φ^n . Since φ is false, his winning probability in b is strictly smaller than 1, and hence his overall winning probability for $\mathcal{G}_{\varphi,k,n}$ is strictly smaller than $p_{k,n}$. \square

The computation of the probabilities $p_{k,n}$ only depends on the parametrization gadget \mathcal{H}_k and n . Clearly $p_{1,n} = 1$. For $k \geq 2$, the winning probability is obtained from the probability of not failing in the first turn, multiplied by the probability of not failing in the second turn, etc., until the probability of not failing in the $(k-1)$ -th turn. If after $k-1$ turns no edge between Runner's current position and vertex b has been deleted, Runner can move to b (so, he cannot fail in the k -th turn). After Runner's first move, he has still to cross $k-1$ edges; neither of them may be deleted in Nature's first turn. Overall, there are $4k+n$ edges in $\mathcal{G}_{\varphi,k,n}$; so Runner does not lose in the first turn if any of the other $4k+n-(k-1)$ edges is deleted. After Runner's second move, he has still to cross $k-2$ edges. There are $4k+n-1$ edges left in the game arena; so Runner does not lose if any of the other $4k+n-1-(k-2)$ edges is deleted. After $k-1$ moves of Runner, $k-2$ edges have been deleted already; so $4k+n-(k-2)$ edges are left in the game arena. If any edge except the one between Runner's current position and b is deleted, Runner is able to reach b . Generally, in the i -th turn there are $4k+n-(i-1)$ edges left; and in order for Runner to still be able to reach b , one of the $3k+n+1$ edges that are not between Runner's current position and b

has to be deleted. Altogether,

$$p_{k,n} = \frac{3k+n+1}{4k+n} \cdot \frac{3k+n+1}{4k+n-1} \cdots \frac{3k+n+1}{4k+n-(k-2)} = \prod_{i=0}^{k-2} \frac{3k+n+1}{4k+n-i}.$$

We can summarize these observations in the following theorem:

Theorem 9. *Given a QBF-formula φ so that \mathcal{G}_φ has n_0 edges, for all $k, n \in \mathbb{N}$ with $k \geq 2$ and $n \geq n_0$ the following holds: Runner wins the randomized reachability game over $\mathcal{G}_{\varphi,k,n}$ for probability $p_{k,n} = \prod_{i=0}^{k-2} \frac{3k+n+1}{4k+n-i}$ iff the QBF-formula φ is true.*

In order to use this theorem for a reduction to the randomized sabotage game, we need to show that the game arena can be constructed in polynomial time. In the following Lemma, we show that the size of the constructed game $\mathcal{G}_{\varphi,k,n}$ is linear in the size of the inputs φ , k , and n :

Lemma 10. *The size of \mathcal{G}_φ is linear in $|\varphi|$, and the size of \mathcal{H}_k is linear in k .*

PROOF. For the first part, it is sufficient to realize that the size of each gadget can be bounded by a constant. Since the number of gadgets is linear in the size of φ , the number of vertices and edges of \mathcal{G}_φ is linear in $|\varphi|$. The only problem might be the ∞ -edges; but by detailed observation, we see that each ∞ -edge can be replaced by a 4-edge, and the construction still works in the same way.

For the second part, it suffices to note that \mathcal{H}_k has $2k+3$ vertices and $4k$ edges. \square

Now, a preliminary result can be formulated in the following form:

Corollary 11. *For all $k \geq 2$, the following problem is PSPACE-hard: Given a randomized sabotage game with $4k+n$ edges, does Runner win with a probability $\geq p_{k,n}$?*

4.4. Towards the PSPACE-Hardness for Arbitrary Probabilities

We already have a reduction of QBF to randomized sabotage games with a varying probability $p_{k,n}$ (which depends on the given game graph). By a closer look at the term $p_{k,n}$ we see that the probability $p_{k,n}$ can be adjusted arbitrarily close to 0 and arbitrarily close to 1. More precisely, for a fixed k , we have $\lim_{n \rightarrow \infty} p_{k,n} = 1$; and for a fixed n , we have $\lim_{k \rightarrow \infty} p_{k,n} = 0$. We will show a stronger result, namely that the probabilities $p_{k,n}$ form a dense set in the interval $[0, 1]$, and that k and n can be computed efficiently such that $p_{k,n}$ is in a given interval. More precisely, we shall show the following:

Theorem 12. *The set of probabilities $\{p_{k,n} \mid k, n \in \mathbb{N}, k \geq 2\}$ is dense in the interval $[0, 1]$. Moreover, given $n_0 \in \mathbb{N}$, $p \in [0, 1]$, and $\varepsilon > 0$, there exist $k, n \in \mathbb{N}$ with $k \geq 2$ and $n \geq n_0$ such that $p_{k,n} \in [p - \varepsilon, p + \varepsilon]$; the computation of such k , n , and $p_{k,n}$ is polynomial in the numerical values of n_0 , $\frac{1}{p}$, and $\frac{1}{\varepsilon}$.*

The proof of this theorem is the subject of Section 5.

Note that Theorem 12 provides a pseudo-polynomial algorithm, since the computation is only polynomial in the *numerical values* of n_0 , $\frac{1}{p}$, and $\frac{1}{\varepsilon}$ (and not in their *lengths*, which are logarithmic in the numerical values). For our needs – i.e. a polynomial time reduction to prove Theorem 3 – this is no restriction; the parameter n_0 corresponds to the number of edges in the input game (which has already a polynomial representation), and p and ε are fixed values (i.e. formally they do not belong to the problem instance).

Now, we prove our main result:

PROOF (OF THEOREM 3). For arbitrary p and ε , we give a reduction from QBF to the randomized sabotage game problem where only probabilities in the interval $[p - \varepsilon, p + \varepsilon]$ are allowed. Note that p and ε do not belong to the problem instance; so they are considered constant in the following.

Given a QBF-formula φ , we need to compute a randomized sabotage game $\mathcal{G}_{\varphi,k,n}$ and a $p_{k,n} \in [p - \varepsilon, p + \varepsilon]$ such that Runner wins $\mathcal{G}_{\varphi,k,n}$ with a probability $\geq p_{k,n}$ iff φ is true. Given φ , we first apply the construction of Section 4.2 to construct an equivalent sabotage game \mathcal{G}_{φ} . Let n_0 be the number of edges of \mathcal{G}_{φ} , which is linear in the size of φ according to Lemma 10. Then, we can compute $k \geq 2$, $n \geq n_0$, and $p_{k,n}$ according to Theorem 12. For fixed p and ε , the computations are polynomial in n_0 and hence polynomial in $|\varphi|$. Now, we extend \mathcal{G}_{φ} to an equivalent sabotage game with n edges, denoted \mathcal{G}_{φ}^n . This can be achieved by adding $n - n_0$ dummy-edges (e.g. we can add a path with a dead end). Thereafter, we construct the randomized sabotage game $\mathcal{G}_{\varphi,k,n}$ by combining the parametrization gadget \mathcal{H}_k with the game arena \mathcal{G}_{φ}^n .

The claimed equivalence of φ to the stated randomized reachability game problem for probability $p_{k,n}$ holds due to Theorem 9. The requirement that $p_{k,n}$ is in the interval $[p - \varepsilon, p + \varepsilon]$ follows from Theorem 12. \square

5. On the Distribution and Computation of the Probabilities $p_{k,n}$

This section deals with the proof of Theorem 12: Given $n_0 \in \mathbb{N}$, $p \in [0, 1]$, and an $\varepsilon > 0$, we can construct $k > 2$ and $n \geq n_0$ in polynomial time with respect to the numerical values of n_0 , $\frac{1}{p}$, and $\frac{1}{\varepsilon}$, such that $p_{k,n}$ is in the interval $[p - \varepsilon, p + \varepsilon]$.

The idea is to first adjust the probability $p_{k,n}$ arbitrarily close to 1, and then go with steps of length below any given $\varepsilon > 0$ arbitrarily close to 0; so, we hit every ε -neighborhood in the interval $[0, 1]$.

In order to adjust the probability $p_{k,n}$ arbitrarily close to 1, we first choose $k = 2$ and a sufficiently high $n \geq n_0$. (We can artificially increase n by adding a path with a dead end.) We will show that it suffices to choose $n := \max\{n_0, \lceil \frac{1}{\varepsilon} \rceil\}$. For this choice we obtain $p_{2,n} \geq 1 - \varepsilon$, which is at least as large as $p - \varepsilon$. Then, we decrease the probability by stepwise incrementing k by 1 (changing \mathcal{H}_k to \mathcal{H}_{k+1} and keeping n constant). It will turn out that the probability decreases by a value that is lower than $\frac{1}{4k+n+4}$, which is, with the choice of n as above, lower than ε . Iterating this, the values converge to 0, and we hit the interval $[p - \varepsilon, p + \varepsilon]$. Hence, the set of probabilities $\{p_{k,n} \mid k, n \in \mathbb{N}, k \geq 2\}$ is dense in the interval $[0, 1]$. Furthermore, we will show that it will be sufficient to increase k at most up to $8n$. For this choice, we obtain $p_{k,n} \leq \varepsilon$, which is at least as small as $p + \varepsilon$.

For the complexity analysis, note the following. After each step, the algorithm has to check efficiently whether $p_{k,n} \in [p - \varepsilon, p + \varepsilon]$. The computation of the term $p_{k,n}$ is pseudo-polynomial in k , n , and the test for $p_{k,n} \leq p + \varepsilon$ is in addition polynomial in $\frac{1}{p}$ and $\frac{1}{\varepsilon}$. Since k and n are pseudo-linear in n_0 and $\frac{1}{\varepsilon}$, the whole procedure is pseudo-polynomial in n_0 , $\frac{1}{p}$, and $\frac{1}{\varepsilon}$.

Four claims remain to be proved:

- The adjustment of $p_{k,n}$ arbitrarily close to 1 with the proposed choice of n , i.e. given $\varepsilon > 0$, for $n \geq \frac{1}{\varepsilon}$ it holds $p_{2,n} \geq 1 - \varepsilon$.
- The adjustment of $p_{k,n}$ arbitrarily close to 0 with the proposed choice of k , i.e. given $\varepsilon > 0$ and $n \geq \frac{1}{\varepsilon}$, for $k \geq 8n$ it holds $p_{k,n} \leq \varepsilon$.
- The estimation $p_{k,n} - p_{k+1,n} < \frac{1}{4k+n+4}$.

- The test for $p_{k,n} \in [p - \varepsilon, p + \varepsilon]$ is pseudo-polynomial in k , n , $\frac{1}{p}$ and $\frac{1}{\varepsilon}$.

These claims are shown in the rest of this section.

Lemma 13. *Given $\varepsilon > 0$, for $n \geq \lceil \frac{1}{\varepsilon} \rceil$ we have $p_{2,n} \geq 1 - \varepsilon$.*

PROOF. Since $n \geq \lceil \frac{1}{\varepsilon} \rceil \geq \frac{1}{\varepsilon} - 8$ for $\varepsilon > 0$, the result follows from

$$p_{2,n} = \frac{n+7}{n+8} \geq 1 - \varepsilon \iff n \geq \frac{1}{\varepsilon} - 8 .$$

□

Lemma 14. *Given $\varepsilon > 0$ and $n \in \mathbb{N}$ with $n \geq \frac{1}{\varepsilon}$ and $n \geq 4$, for $k \geq 8n$ we have $p_{k,n} < \varepsilon$.*

PROOF. First note that we have at least $n \geq 1$ and $k \geq 8$. Then

$$\begin{aligned} p_{k,n} &= \prod_{i=0}^{k-2} \frac{3k+n+1}{4k+n-i} \leq \left(\frac{3k+n+1}{3.5k+n} \right)^{\lceil \frac{k}{2} \rceil} \leq \left(\frac{3k+n+1}{3.5k+n} \right)^{\frac{k}{2}} \leq \left(\frac{4k+1}{4.5k} \right)^{\frac{k}{2}} \\ &\leq \left(\frac{4.125k}{4.5k} \right)^{\frac{k}{2}} = \left(\frac{11k}{12k} \right)^{\frac{k}{2}} = \left(\frac{11}{12} \right)^{\frac{k}{2}} \leq \left(\frac{11}{12} \right)^{4n} < \varepsilon . \end{aligned}$$

The inequality $\left(\frac{11}{12} \right)^{4n} < \varepsilon$ remains to be shown. Since $\frac{1}{n} \leq \varepsilon$, it is sufficient to show that $\left(\frac{11}{12} \right)^{4n} < \frac{1}{n}$:

$$\left(\frac{11}{12} \right)^{4n} < \frac{1}{n} \iff n^{\frac{1}{4n}} < \frac{12}{11} \iff \sqrt[n]{n^{\frac{1}{4}}} \leq \sqrt{2^{\frac{1}{4}}} < \frac{12}{11} .$$

The inequality $\sqrt[n]{n^{\frac{1}{4}}} \leq \sqrt{2^{\frac{1}{4}}}$ is equivalent to $n^2 \leq 2^n$ and holds for all $n \geq 4$. □

Lemma 15. *For $k, n \in \mathbb{N}$ with $k \geq 2$, we have $p_{k,n} - p_{k+1,n} < \frac{1}{4k+n+4}$.*

PROOF. In this proof we use the substitution $m := 4k + n + 4$.

$$\begin{aligned} p_{k,n} - p_{k+1,n} &= \prod_{i=0}^{k-2} \frac{3k+n+1}{4k+n-i} - \prod_{i=0}^{k-1} \frac{3k+n+4}{4k+n+4-i} \\ &\leq \prod_{i=0}^{k-2} \frac{3k+n+4+1}{4k+n+4-i} - \prod_{i=0}^{k-1} \frac{3k+n+4}{4k+n+4-i} = \prod_{i=0}^{k-2} \frac{m-k+1}{m-i} - \prod_{i=0}^{k-1} \frac{m-k}{m-i} \\ &= \prod_{i=0}^{k-1} \frac{m-k+1}{m-i} - \prod_{i=0}^{k-1} \frac{m-k}{m-i} = \frac{(m-k+1)^{k-1} - (m-k)^{k-1}}{\prod_{i=0}^{k-1} m-i} . \end{aligned}$$

Now we can use the equation $a^l - b^l = (a-b)(a^{l-1} + a^{l-2}b + \dots + ab^{l-2} + b^{l-1})$ for the estimation $(d+1)^{k-1} - d^{k-1} = (d+1)^{k-2} + (d+1)^{k-3}d + \dots + (d+1)d^{k-3} + d^{k-2} \leq (k-1)(d+1)^{k-2}$. We obtain

$$p_{k,n} - p_{k+1,n} \leq \frac{(k-1)(m-k+1)^{k-2}}{\prod_{i=0}^{k-1} m-i} = \frac{k-1}{m(m-1)} \prod_{i=2}^{k-1} \frac{(m-k+1)}{m-i} \leq \frac{k-1}{m(m-1)} .$$

Since $m > k$ for all $k, n \in \mathbb{N}$, we obtain $p_{k,n} - p_{k+1,n} < \frac{1}{m} = \frac{1}{4k+n+4}$. □

Lemma 16. *The computation of the term $p_{k,n}$ is pseudo-polynomial in k and n . The test for $p_{k,n} \leq p + \varepsilon$ is pseudo-polynomial in k and n , and polynomial in $\frac{1}{p}$ and $\frac{1}{\varepsilon}$.*

PROOF. First, we rewrite $p_{k,n}$ in the form

$$\frac{(3k + n + 1)^{k-1}}{\prod_{i=0}^{k-2} 4k + n - i}.$$

Now, we compute the numerator and the denominator separately. For the computation, we can switch to binary encoding. Each multiplication can be performed in polynomial time in the length of its binary encoding [16]. We need $k - 2$ multiplications (for this reason, the algorithm is only pseudo-polynomial). The division and comparison of two rational numbers can be done in polynomial time with respect to the length of their binary representations [16]. So, the quotient $p_{k,n}$ can be computed in pseudo-polynomial time with respect to k and n , and the test to check whether $p_{k,n} \leq p + \varepsilon$ is in addition polynomial in $\frac{1}{p}$ and $\frac{1}{\varepsilon}$. \square

6. Perspectives

We have introduced randomized sabotage games, and showed that the reachability game problem (resp. LTL game problem) for a probability which may vary in a fixed interval $[p - \varepsilon, p + \varepsilon]$ is PSPACE-complete. This is a small contribution to the emerging research on the analysis of dynamical networks with aspects of randomness. As concrete open issues, we mention the following problems:

1. In our proof of the PSPACE-hardness, it seems difficult to adjust the probability *exactly* to a given probability p (in our formulation this is the case if $\varepsilon = 0$). It remains open whether this can be achieved by a refinement of the construction.
2. In our hardness proof, we used the reachability problem with a target set F containing at least two vertices (see Remark 7). One task is to extend the result to cover also the case of a singleton as target set (note that in the non-randomized case, the singleton reachability problem is PSPACE-hard only if one allows multi-edges [2]).
3. The proof for our hardness result depends on the restriction that exactly one edge per turn is deleted (rather than possibly multiple edge deletion occurring subject to given probabilities). Sharpening the mentioned problem of “dynamic graph reliability” [8] to probabilities that are independent of Runner’s position, we can study the model where in every turn each edge fails with a probability $p(e)$, or even with probability $\frac{1}{n}$ in the uniform case.
4. An interesting direction of research are winning conditions in branching time logics where the probabilities are incorporated into the logic, e.g. PCTL (cf. [10]) It is an open issue to find such an appropriate logic for which the sabotage game problem can still be decided in PSPACE [17].
5. Extending the model with a mechanism of restoration is a challenging task (for instance, *reactive Kripke models* [18] and *backup parity games* [6] address this issue). In [19] we developed a theory of dynamic networks where Runner and Blocker are replaced by two players, *Constructor* and *Destructor*, that add resp. delete vertices/edges, and the problem of guaranteeing certain network properties (like connectivity) is addressed. We are presently integrating probabilistic features into this model, starting from the present paper.

Acknowledgments. We thank Łukasz Kaiser for his help regarding Lemma 15. We also thank the anonymous referees for their valuable comments and suggestions in improving this paper.

References

- [1] J. van Benthem, An essay on sabotage and obstruction, in: Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday, volume 2605 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 268–276.
- [2] C. Löding, P. Rohde, Solving the Sabotage Game is PSPACE-hard, Technical Report AIB-05-2003, RWTH Aachen, 2003.
- [3] C. Löding, P. Rohde, Solving the sabotage game is PSPACE-hard, in: Proceedings of MFCS, volume 2747 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 531–540.
- [4] C. Löding, P. Rohde, Model checking and satisfiability for sabotage modal logic, in: Proceedings of FSTTCS, volume 2914 of *Lecture Notes in Computer Science*, Springer, 2003, pp. 302–313.
- [5] P. Rohde, Moving in a crumbling network: The balanced case, in: Proceedings of CSL, volume 3210 of *Lecture Notes in Computer Science*, Springer, 2004, pp. 310–324.
- [6] P. Rohde, On Games and Logics over Dynamically Changing Structures, Ph.D. thesis, RWTH Aachen, 2005.
- [7] N. Gierasimczuk, L. Kurzen, F. R. Velázquez-Quesada, Learning and teaching as a game: A sabotage approach, in: Proceedings of LORI, volume 5834 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 119–132.
- [8] C. H. Papadimitriou, Games against nature, *Journal of Computer and System Sciences* 31 (1985) 288–301.
- [9] E. M. Clarke, Jr., O. Grumberg, D. A. Peled, *Model Checking*, The MIT Press, Cambridge, MA, USA, 2000.
- [10] C. Baier, J.-P. Katoen, *Principles of Model Checking*, The MIT Press, Cambridge, MA, USA, 2008.
- [11] A. Pnueli, R. Rosner, On the synthesis of an asynchronous reactive module, in: ICALP, volume 372 of *Lecture Notes in Computer Science*, Springer, 1989, pp. 652–671.
- [12] C. Courcoubetis, M. Yannakakis, The complexity of probabilistic verification, *Journal of the ACM* 42 (1995) 857–907.
- [13] C. H. Papadimitriou, *Computational Complexity*, Addison Wesley, 1994.
- [14] M. L. Littman, S. M. Majercik, T. Pitassi, Stochastic boolean satisfiability, *Journal of Automated Reasoning* 27 (2001) 251–296.
- [15] D. Klein, F. G. Radmacher, W. Thomas, The complexity of reachability in randomized sabotage games, in: Proceedings of FSEN, volume 5961 of *Lecture Notes in Computer Science*, Springer, 2009, pp. 162–177.
- [16] J. von zur Gathen, J. Gerhard, *Modern Computer Algebra*, Cambridge University Press, Cambridge, UK, second edition, 2003.
- [17] D. Klein, Solving Randomized Sabotage Games for Navigation in Networks, Diploma thesis, RWTH Aachen, 2008.
- [18] D. M. Gabbay, Introducing reactive Kripke semantics and arc accessibility, in: Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday, volume 4800 of *Lecture Notes in Computer Science*, Springer, 2008, pp. 292–341.
- [19] F. G. Radmacher, W. Thomas, A game theoretic approach to the analysis of dynamic networks, in: Proceedings of VerAS, volume 200 (2) of *Electronic Notes in Theoretical Computer Science*, Elsevier, 2008, pp. 21–37.