Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science 200 (2008) 21–37
Electronic Notes in Theoretical Computer Science
www.elsevier.com/locate/entcs

A Game Theoretic Approach to the Analysis of Dynamic Networks[1]

Frank G. Radmacher and Wolfgang Thomas

RWTH Aachen, Lehrstuhl für Informatik 7, 52056 Aachen, Germany

{radmacher,thomas}@automata.rwth-aachen.de

Abstract

A model of dynamic networks is introduced which incorporates three kinds of network changes: deletion of nodes (by faults or sabotage), restoration of nodes (by actions of "repair"), and creation of nodes (by actions that extend the network). The antagonism between the operations of deletion and restoration resp. creation is modelled by a game between the two agents "Destructor" and "Constructor". In this framework of dynamic model-checking, we consider as specifications ("winning conditions" for Constructor) elementary requirements on connectivity of those networks which are reachable from some initial given network. We show some basic results on the (un-) decidability and hardness of dynamic model-checking problems.

Keywords: adaptive systems, dynamic networks, dynamic graph problems, model-checking, game theory.

1 Introduction and Motivation

The classical methodology of model-checking (see [6]) refers to a fixed system and a specification of its behavior. Usually the states and transitions of the system under consideration are collected in a transition graph (Kripke structure), and the only dynamic aspect is the change of state (via available transitions) which is then captured by the concept of an execution path or (if branching behavior is considered) of a computation tree. Specifications are definitions of desired properties of execution paths or computation trees.

A theory of model-checking over dynamic structures is still very much in its beginnings. Some references known to us are [15,3,13] where the data structures with a dynamic behavior (like programs or UML state machines) are efficiently encoded (e. g. as lists or trees) for the use of well-established model-checking methods. The objective of this paper is to introduce (and prove initial results on) a simple but quite

[1] Research was supported by the RWTH Aachen Research Cluster UMIC (Ultra High Speed Mobile Information and Communication) of the German Excellence Initiative.

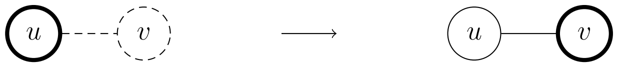

Figure 1. Movement of strong node with restoration.

general model for studying networks which change over time. We consider graphs as models of communication networks. Such a network is assumed to change over time by the deletion and creation of nodes. The different networks arising in this way from an original network G can be collected in a Kripke structure whose elements are networks reachable from G by successive operations of network modification. So we deal with a hierarchical model in which the first layer captures networks and the second (represented by the Kripke structure) the dynamics of networks. Instead of execution paths we consider "network evolvement paths", and specifications typically are addressing properties of networks that should hold throughout (safety properties) or properties that should be achieved (reachability conditions). More complicated specifications are possible (e.g., referring to standard temporal logics like LTL) but are not pursued in the present paper.

Two levels of the dynamics have to be distinguished: The first is concerned only with information flow through a network, or a change of states of the nodes of the network. Nodes and edges as such stay fixed. A natural way to describe this aspect is to assume a labeling of the nodes that may change over time. The classical model-checking problem can be seen as a special case of this dynamic labeling: We consider the Kripke structure as a network and indicate the current state of the Kripke structure by a special label at the corresponding node, or – in other words – as a token on this node. A step consists of a shift of the token to a neighbor node reachable via a transition; this obviously is a special case of re-labeling. In the general case we have labels and change labels simultaneously in several nodes of the network under consideration. We shall capture this idea of "information flow" by re-labeling rules.

The second aspect of the dynamics involves a change in the set of nodes and/or the set of edges. In the present paper we only consider changes in the set of nodes (and induced changes in the set of edges). The more general case that arises when including edges in the dynamics involves heavier notation but does not affect the general results as they appear in this paper. We distinguish three types of changes, with an "original" network G as a reference. We may delete nodes, we may restore nodes that had been in existence before, or we may create completely new nodes. All these steps involve the concept of "strongness" of a node: A strong node cannot be deleted, and it is the prerequisite for performing the restoration and creation of nodes. In the context of communication networks, one can view the strong nodes as stations where maintenance resources are located. These resources may travel through edges to existing nodes, and they may at some node u restore a deleted node v by moving to it if there was an edge (u, v) in the network before the deletion of v (see Fig. 1).

For the creation of a node we can pick some set S of strong nodes, create a new node v (that is or is not strong) and connect it with each of the nodes in S by a transition (see Fig. 2). This corresponds to the assumption that creation is "more

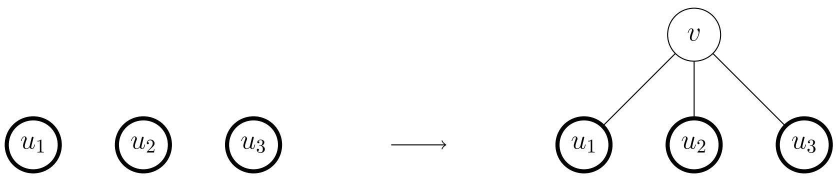

Figure 2. Creation of a new node by a set $U = \{u_1, u_2, u_3\}$ of strong nodes.

expensive" than restoration.

Both kinds of rules can be applied either generally or subject to constraints which refer to the current labeling of nodes.

We model the natural antagonism between the deletion and restoration resp. creation of nodes by a two-player game between the two agents "Destructor" (performing deletion actions anywhere in the network) and "Constructor" (performing the actions of restoration and creation). Note that the actions of Destructor occur globally in the network, whereas the actions of Constructor occur locally (they are based on shifting strongness of nodes to neighbors). This asymmetry between deletion and restoration or creation seems adequate for modelling phenomena of fault-tolerance: Faults may happen anywhere, but actions of repair need coordination of resources. For simplicity, we shall consider here only turn-based games where Destructor and Constructor act in alternation (but both have the option to skip). Another simplification is the assumption that Constructor's (and Destructor's) moves are decided taking into account full information about the current network. (This simplifying assumption is to be dropped in future work, using the mechanism of information flow.) The task of Constructor is to enforce that properties given in a specification will be reached (or preserved) against any sequence of moves by Destructor.

Specifications for dynamic networks are typically concerned with guaranteeing desirable properties of networks. In the present paper we only consider the properties of connectivity and biconnectivity. (A graph is biconnected if deletion of any node still leads to a connected graph.) There are many similar properties, like k-connectivity, having a small diameter, having a certain minimal degree for all nodes, etc.

The *dynamic model-checking problem* now takes the form of a problem of game solution, where a network G and a set R of rules for its modification are given. We associate deletion steps with Destructor and all other moves with Constructor.

For example we can ask for a solution of a safety game, given G and R:

Starting from G, does Constructor have a winning strategy to guarantee that the network always stays connected?

A reachability game is given by the following problem:

Starting from G which is connected, does Constructor have a winning strategy that will lead to a biconnected graph?

These are first examples of a much wider class of problems that are studied in the algorithmic theory of infinite games (see [9]). The perspective of our work is to link this theory to the analysis of dynamic networks.

The paper is structured as follows. In the next section we briefly discuss related work. Then we present our model in formal detail. After that we list some natural network properties that are relevant in the present context and show some initial results on the solution of dynamic model-checking problems. Not surprisingly, it turns out that in the general framework the reachability problem is undecidable. In the restricted case of "non-expansive" dynamic networks we obtain decidability but establish some hardness results. In the concluding section we address perspectives of the game theoretic analysis of dynamic networks which are treated in ongoing work.

2 Related Work

The game theoretic framework developed here is an extension of the theory of "sabotage games" suggested by van Benthem (cf. [22]) and developed by Ph. Rohde and C. Löding at RWTH Aachen; see [17,18,16,19,20]. The networks in these papers are directed graphs with possibly multiple edges between nodes, on which a player "Runner" wants to reach a node v from a given start node u. After each move of Runner, the adversary, called "Blocker", may delete an edge; and 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 node v against any choice of Blocker in deleting edges. Note that this game involves only re-labelings (shift of position of Runner) and (edge-) deletions. As in the general game above, the moves of the "good" player are local (i. e., transitions through available edges), whereas the "bad" player (the Blocker) can act globally and access any existing edge in any move.

Another interesting approach arises from the studies of dynamic versions of the *Dynamic Logic of Permission* (DLP) which is in turn an extension of the *Propositional Dynamic Logic* (PDL). In DLP, "computations" in a Kripke structure from one state to another are considered which are subject to "permissions". The logic $\mathrm{DLP}^{+}_{\mathrm{dyn}}$ (see [5,8]) extends DLP with formulas which allow updates of the permission set and thus can be seen as a dynamically changing Kripke structure. Nevertheless, all the dynamics have to be specified in the formula; an adversarial player is not considered.

Previous studies of dynamic networks addressing the aspect of adversarial agents have been concerned with routing problems. In such studies it was either assumed that the network under consideration stays connected or the connectivity of the network is at least given infinitely often over time (cf. [6,1,2]).

The idea of changing networks is of course studied in considerable depth in the theory of graph grammars, graph rewriting, and graph transformations (see [4,21]). While there the class of generable graphs (networks) is the focus of study, we deal here with a more refined view when considering "evolvement paths" and the properties of graphs occurring in them. In the "one-player" framework of model-checking, we mention the work [7] where *graph-interpreted temporal logic* is introduced as a rule-based specification. It extends LTL by interpreting formulas on transition systems which nodes are graphs. A technique is developed to map the "graph transition systems" to finite Kripke structures, so that classical LTL-model-checking can be applied.

In finite model theory, S. Patnaik, N. Immerman, and W. Hesse developed a theory of "dynamic complexity" (cf. [11,10]). For example, the decision problem DynREACH is the question whether, given two vertices u and v and a sequence of edge insertions and deletions (as well as changes of u and v), there is a path from u to v. This problem corresponds to the restriction of our framework to one-player games, where Constructor can add and delete edges from networks. For this problem there are motivations in data base theory, but the asymmetry between Destructor and Constructor (as in settings where fault tolerance is the guiding motivation) is missing.

In the theory of *fully dynamic algorithms*, dynamic graphs in which vertices and edges are inserted or deleted are considered, but the focus of investigation is the computational complexity of static graph properties with respect to a given update step resp. a given sequence of update steps (cf. [14,12]).

3 Dynamic Networks via Games

3.1 The Basic Model

We present *networks* in the form $G = (V, E, A, S, (P_a)_{a\in\Sigma})$ with

- a finite set V of vertices (also called nodes),
- an undirected edge relation E,
- a set $A \subseteq V$ of "active nodes",
- a set $S \subseteq A$ of "strong nodes",
- a partition of V into sets P_a for some label alphabet Σ (A node belongs to P_a if it carries label a. For technical simplicity we include a blank symbol $\sqcup$ in Σ (as a default for "unlabeled" nodes).

The dynamics of a network arises from an initial network G by the possible moves from Destructor and Constructor (Constructor's moves are subject of a set of rules) which are changing the respective current network. A *game arena* will be presented as a pair $\mathcal{G} = (G, R)$, with an *initial network* G as above and a finite set R of rules for Constructor. (In this work we will always have $A = V$ for the initial graph G; no nodes are deleted at the beginning of the play.)

The two players *Destructor* and *Constructor* play turn by turn (Destructor starts). In our model the players are allowed to skip as well. Plays are infinite in general, but may be considered finite in the cases where neither of the players can move, or a given winning condition is satisfied.

Let us describe the rules that define the moves. When it is Destructor's turn, he can do a *deletion step* by deleting some node $v \in (A \setminus S)$; A is changed to $A \setminus \{v\}$.

When it is Constructor's turn, she can make in accordance with his rule set R a move of one of the following types:

Labeling rule: $\langle\, a, b \longrightarrow c, d \,\rangle$ means to change the labels a and b of two edge-connected nodes of A into c and d, respectively. In the case of "pure information flow" we have $b, c = \sqcup$ and $a = d$. Formally, for a pair $(u, v) \in E$ with $u \in P_a$ and $v \in P_b$ the sets P_a, P_b, P_c, and P_d are updated to $P_a \setminus \{u\}$, $P_b \setminus \{v\}$, $P_c \cup \{u\}$, and $P_d \cup \{v\}$.

For labeling rules we will also consider rules with multiple re-labelings in one turn. This corresponds to our intuition that there can be a lot of information flow in the network at the same time. For example for two labeling rules in one turn we use the notation $\langle\, a, b \longrightarrow c, d\,;\, e, f \longrightarrow g, h \,\rangle$. The rules are applied one after the other but in the same turn.

Move of strong node resp. restoration of node: For an arbitrary $u \in S$ with an edge $(u, v) \in E$ and a node $v \notin S$, update S to $(S \setminus \{u\}) \cup \{v\}$ and A to $A \cup \{v\}$. In case $v \in A$ this means to simply shift strongness to v; in case $v \in V \setminus A$ this means restoration of v. The ability of a strong node to move may be subject to a certain labeling (a, b) of the pair (u, v). We use the Notation $\langle\, a \xrightarrow{\text{move}} b \,\rangle$.

Creation of node: Take any set $U \subseteq S$ of strong nodes, create a new node w and connect it to the nodes of U. Formally, the sets V and A are updated to $V \cup \{w\}$, resp. $A \cup \{w\}$, and also E is updated by adding edges between w and each node of U. The creation may be subject to the node labels $a_1, \ldots, a_n$ $(n = |U|)$ of the nodes in U. Also the labels of the nodes in U may change after creation (to $a'_1, \ldots, a'_n$). We use the notation $\langle\, a_1, \ldots, a_n \xrightarrow{\text{create}(c)} a'_1, \ldots, a'_n \,\rangle$, when the new node w is labeled with c. By simply writing $\langle\, a_1, \ldots, a_n \xrightarrow{\text{create}} a'_1, \ldots, a'_n \,\rangle$ we assume w gets the label $\sqcup$. For the *creation of a strong node* we use the notation $\langle\, a_1, \ldots, a_n \xrightarrow{\text{s-create}(c)} a'_1, \ldots, a'_n \,\rangle$. In this case also S is updated to $S \cup \{w\}$. Note that these moves are the only ones that change V and E. The new node w is created only once; it can henceforth only be deleted and restored.

For a game arena as given here and a *network property* P, we distinguish two types of games referring to P:

- *Safety Game with respect to P:* Does Constructor have a winning strategy to preserve, starting from G and proceeding by the rules of R, the property P for all networks reached in a play?
- *Reachability Game with respect to P:* Does Constructor have a winning strategy to guarantee, starting from G and proceeding by the rules of R, that some network with property P is reached?

The *Dynamic Model-Checking Problem* (in the safety, respectively reachability format) consists of the question, given G and R and network property P, whether Constructor has a winning strategy in the safety game, respectively reachability game for P.

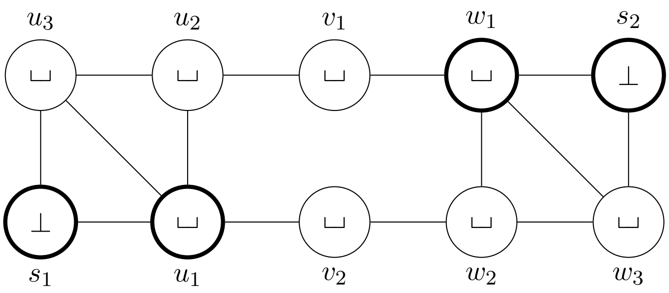

Figure 3. An example initial game graph (bold nodes are strong).

More generally, one can introduce, referring to network properties $P_1, \dots P_n$, the temporal logic LTL over $P_1, \dots, P_n$ and ask for winning strategies that guarantee satisfaction of an LTL-formula for an infinite play between Destructor and Constructor.

Example 3.1 Consider the game arena $\mathcal{G} = (G, R)$ with labels $\perp$, $\sqcup$ in Σ. The initial game graph $G = (V, E, A, S, (P_a)_{a \in \Sigma})$ with the set $S = \{s_1, s_2, u_1, w_1\}$ of strong nodes is depicted in Fig. 3.

Consider as rules R only moves of strong nodes (if applicable with restoration) labeled with a blank: $\langle \sqcup \xrightarrow{\text{move}} \sqcup \rangle$. The strong nodes s_1 and s_2 are not able to move, because their label does not match the move rule. As a scenario for this game arena we could imagine two clients s_1, s_2 which communicate over a network with unreliable intermediate nodes but two mobile maintenance ressources.

We consider a safety game with connectivity of s_1 and s_2 as property P. So, we are interested in the question whether Constructor can always guarantee that there is a path from s_1 to s_2 in the network. If we take a closer look at this example we see that Destructor has a winning strategy. He can delete w_3 in his first move. Then we distinguish between two cases: If Constructor restores w_3 then Destructor deletes v_1 in his next move and finally u_1 or v_2. If Constructor does not move the upper movable strong node to w_3 this node has to remain at w_1, otherwise Constructor loses by deletion of w_1. In this case it is easy to see that Destructor wins by a suitable deletion of nodes in $\{u_1, u_2, v_1, v_2\}$.

Now we consider the same game, but additionally with the rule $\langle \sqcup, \sqcup \xrightarrow{\text{create}} \sqcup, \sqcup \rangle$. We claim that now Constructor has a winning strategy. If Destructor deletes the vertex v_1 or v_2 Constructor creates a new vertex v_3 with the creation rule which establishes a new connection between the two strong nodes u_1 and w_1. If Destructor deletes the new vertex v_3 Constructor creates a new vertex again, and so on. Note that in this way the set V of vertices of the graph can increase to an unbounded number.

Even in these very small examples the exact formalization of the winning strategies is quite complex. So, this example should already suffice as a motivation for formal model-checking methods.

3.2 Remarks on Special Cases and Variants

The model introduced above is quite general, and many special cases and variants can be studied. We only mention three of them.

(i) *Classical Model-Checking:* As mentioned in the Introduction, the classical model-checking problem for LTL can be captured as a special case of the dynamic model-checking problem (in the framework of LTL). In this case, the graph is directed and the properties $P_1, \ldots, P_n$ refer to pointed graphs (G, v) (so P_i may be considered as a vertex rather than a graph). Destructor stays silent – so we have a solitaire game –, and Constructor realizes the change of the "current state" by shifting a token through the given graph (using only re-labeling rules).

(ii) *Solitaire Games:* In this case, only Constructor moves and keeps control of the evolvement of networks. Deletion and hence restoration moves are not admitted, and the moves that remain are either the creation or the re-labeling of nodes. A variant suggested by the solitaire model is the situation where Constructor also carries out the deletion moves. This corresponds to a situation where the evolvement of a structure (here: a network) is controlled completely by one agent.

(iii) *Non-expansive Games:* For proper games between Destructor and Constructor, we can consider the special case that Constructor does not use creation moves. Then we call the rules *non-expanding.* This situation corresponds to a network evolvement where the original set V of nodes is never extended. Obviously, the set of networks reachable from a given initial graph – i. e., the Kripke structure built from G – is finite in this case. As a consequence, the "non-expanding dynamic model-checking problem" (in the safety-, reachability-, or even LTL-format) is decidable. Below we show that dropping the condition of non-expansion causes undecidability.

There are many more variants that might have good motivations from different applications. A natural variant is to consider deletion and insertion of edges rather than nodes (or even of both, nodes and edges). This would require corresponding conventions regarding the rules and their reference to labels. Note that the sabotage games were based on the idea of edge deletion. For the general results shown in this paper, this aspect is not essential; so we do not pursue it. However, it will play a major role for refined studies, e. g. concerning information flow.

A possible criticism about our framework may be the radical difference between Destructor and Constructor, which puts the former into an advantage. This can be remedied by natural conventions, for example by the assumption that Destructor keeps silent (i. e., skips) for certain periods of time until a deletion is carried out. It depends on the respective area of application how a balance between Constructor and Destructor should be tuned. In the context of fault tolerance (which is the motivation of our work), it is natural to assume that that the deletion actions of Destructor are undesirable and the creation and restortation actions of Constructor are desirable. In other domains, the converse is conceivable. The purpose of the present paper is to exhibit a framework where such modification are easy to incorporate.

4 Basic Results on Dynamic Model-Checking

Let us list some basic network properties that arise naturally in our framework.

We refer to the restriction of networks in the simple format $G = (V, E)$ of plain undirected graphs. As a preliminary remark we mention that in most practical applications the degree of networks should be bounded by a constant d (meaning that each node is connected to at most d neighbors).

A graph $G = (V, E)$ is connected if for each $u, v \in V$ there is a path from u to v. It is biconnected (resp. k-connected for $k > 2$) if for each node v (resp. for each $k-1$-tuple $(v_1, \dots, v_{k-1})$ of nodes) the induced graph over $V \setminus \{v\}$ (resp. $V \setminus \{v_1, \dots, v_{k-1}\}$) is connected. A graph $G = (V, E)$ is of diameter d if for each pair $u, v \in V$ there is a path from u to v of length $\leq d$.

In our first result we show that the dynamic model-checking problem even in the solitaire version (where Destructor always decides to skip) with respect to connectivity properties is undecidable. The essential feature of the undecidability proof is the availability of creation moves.

Theorem 4.1 *The dynamic model-checking for solitaire games in reachability format is undecidable in the following version: Given a connected network G and a finite set R of rules for Constructor, is there a strategy for Constructor to reach a biconnected network?*

Proof We use a reduction to the halting problem for Turing machines. We present Turing machines in the format

$$M = (Q, \Gamma, \delta, q_0, q_{\text{stop}})$$

with a state set Q, tape alphabet Γ (we assume $Q \cap \Gamma = \emptyset$), transition function $\delta : Q \setminus \{q_{\text{stop}}\} \times \Gamma \to Q \times \Gamma \times \{L, R\}$, initial state q_0, and stop state q_{stop}.

For each such Turing machine M we construct a solitaire game (i. e. an initial connected network G and a set R of rules) such that M halts when started on the empty tape iff Constructor can reach a biconnected network from G by applying the rules of R.

The idea of the construction is to represent a Turing machine configuration $[a_1 \dots a_n q b_1 \dots b_m]$ (here, [and] are left and right marker, respectively) by a sequence of $n + m + 3$ nodes labeled with these symbols in succession. The label alphabet thus contains the tape alphabet of M, the states of M, and the end markers. For later use we allow also the letters $[_s$, $]_s$, p_+ for each state $p \in Q$, and a_s for each letter a of the tape alphabet.

As initial graph we take the three node graph with nodes labeled [, q_0,]. We define these three nodes as strong.

It is easy to supply a set of re-labeling rules which allow to change this line graph only in a way that the computation of M starting with the empty tape is simulated. Formally, for each pair (q, a, p, b, X) with $\delta(q, a) = (p, b, X)$ we add the game rules $\langle\, q, a \longrightarrow p_+, b\,\rangle$ and $\langle\, y, p_+ \longrightarrow p, y\,\rangle$ if $X = L$, and $\langle\, q, a \longrightarrow b, p\,\rangle$ if $X = R$ (here, y denotes an arbitrary label of $\Gamma \cup \{\sqcup\}$).

For the case that more space is needed (beyond the current end markers), we introduce creation steps subject to labelings, making the endmarker nodes strong again: $\langle\, [\xrightarrow{\text{s-create}(\,[\,)} \sqcup\,\rangle$ and $\langle\,] \xrightarrow{\text{s-create}(\,]\,)} \sqcup\,\rangle$.

Finally, we allow special re-labeling rules that can be applied when the final state q_{stop} is reached. The extra index "s" is now propagated through the existing line graph to the right and to the left, using the extra letters mentioned above. Given two strong nodes with labels $[_s,]_s$, we allow to create a new node w connected with both of them, thus generating a circle graph (which of course is biconnected). □

The result also holds for the two-player game. Since in our simulation we have defined all nodes as strong, the presence of a destructive player does not change the plays of the game.

Note that the proof relies on the availability of creation moves; if these are cancelled, the problem becomes trivially decidable:

Remark 4.2 The dynamic model-checking problem in the reachability and the safety version is decidable for non-expanding networks.

For the proof it suffices to observe that from the initial network only finitely many distinct networks (over the unchanged original vertex set V) can be generated. If there are N such networks, each play will yield a repetition after at most N steps. Hence, for deciding the winner (and providing a winning strategy) it suffices to build up the game tree up to depth N.

Remark 4.3 In order to keep the model simple we do not consider scenarios where deletion and insertion of edges are allowed. Since the number of possible edges in the non-expansive case is bounded by $\binom{|V|}{2}$, the absence of edge insertion and deletion is insignificant for the decidability of the model-checking problem.

In the following we will show that solvability of non-expansive games (i. e., without creation rules) is PSPACE-hard. We use a reduction to the solution of *sabotage games* as mentioned in the introduction. Löding and Rohde showed that the sabotage game problem is complete for the PSPACE [17,18,20]. Let us recall the precise formulation.

A *sabotage game* $\mathcal{G}_s = (G_s, v_{\text{in}})$ is played on a game graph $G_s = (V, E)$ which changes over time; also a designated set $F \subseteq V$ is given. The two players, called *Runner* and *Blocker*, play as follows: Runner starts the play in a vertex v_{in} and moves to a node v' with $(v_{\text{in}}, v') \in E$. Then player Blocker erases an edge $(w, w') \in E$ resulting in a graph G'. Runner continues the play on G', and so on in alternation. Runner wins iff she can can reach a vertex in the given set $F \subseteq V$ of final vertices.

Löding and Rohde introduced sabotage games with multi-graphs (with possibly multiple edges), but showed that single-graphs suffice (see Lemma 1 in [18]).

Theorem 4.4 *In the non-expansive case, the dynamic model-checking problem in*

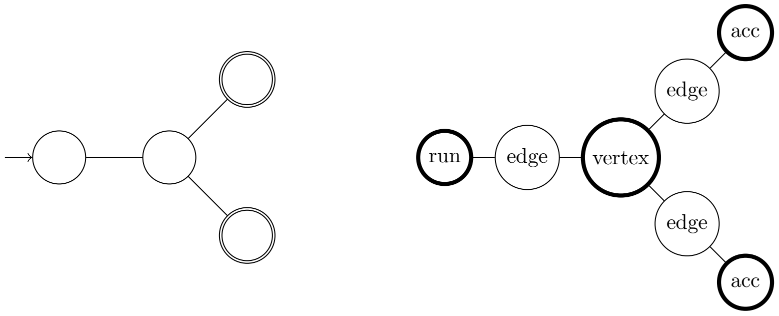

Figure 4. Initial graph G_s of a sabotage game and corresponding initial graph G of the reachability game.

the reachability version is PSPACE-hard.

Proof Consider a sabotage game $\mathcal{G}_s = (G_s, v_{\text{in}})$ on the initial game graph $G_s = (V_s, E_s)$ and a set $F \subseteq V_s$ of final vertices. We construct a game arena $\mathcal{G} = (G, R)$ with node labels 'vertex', 'edge', 'run', 'acc', 'reach' in the alphabet Σ with the purpose that Constructor can reach a network in $\mathcal{G}$ containing a label 'reach' iff Runner wins the sabotage game $\mathcal{G}_s$.

The idea is to simulate the Runner by labeling rules of the form $\langle\, \text{run}, * \longrightarrow *, \text{run}\,\rangle$. We simulate edge removal by node removal as follows: We make the vertices of the original initial graph G_s unerasable by making them to unmovable strong nodes. Then we add one vertex for each edge in E_S. Instead of deleting edges Destructor may delete these new vertices. In order to allow Constructor to "move" the run label from one unerasable vertex to another, she make two re-labeling steps in one turn.

Figure 4 shows the initial graph G_s of an example sabotage game and the equivalent initial game graph G of our game.

The initial game graph is $G = (V, E, A, S, (P_a)_{a\in\Sigma})$ with $V := V_s \cup E_s$, and E arises from E_s as in the figure. Additionally let $P_{\text{vertex}} := V_s \setminus (\{v_{\text{in}}\} \cup F)$, $P_{\text{edge}} := E_s$, $P_{\text{run}} := \{v_{\text{in}}\}$, $P_{\text{acc}} := F$, $P_{\text{reach}} := \emptyset$, and $S := V_s$.

In order to simulate the movement of Runner the set R contains the rules $\langle\, \text{run}, \text{edge} \longrightarrow \text{vertex}, \text{run}\,;\, \text{run}, \text{vertex} \longrightarrow \text{edge}, \text{run}\,\rangle$ and $\langle\, \text{run}, \text{edge} \longrightarrow \text{vertex}, \text{run}\,;\, \text{run}, \text{acc} \longrightarrow \text{edge}, \text{reach}\,\rangle$. These rules allow Constructor to propagate the 'run' label from one 'vertex'-labeled node to another by passing one non-deleted 'edge'-labeled node. If the 'run'-label reaches the 'acc'-label it is re-labeled to 'reach'. Note that there are no movement resp. restoration rules in R, so that all strong nodes (not labeled with 'edge') are considered unerasable.

In this proof we assume Constructor starts the game (alike Runner starts the sabotage game). (This could be easily achieved by connecting the 'run'-node to one 'acc'-node with an additional 'edge'-labeled node. So, Destructor deletes this extra 'edge'-labeled node in his first turn.)

The deletion of the nodes in E_s resp. the propagation of the 'run'-label from on node in V_s to another corresponds to the edge-deletion resp. to the movement of Runner in the sabotage game. Hence, Constructor can reach a network containing a label 'reach' in $\mathcal{G}$ iff Runner wins the sabotage game $\mathcal{G}_s$. □

Note that we do not have any movement resp. restoration steps in the proof, so the problem remains PSPACE-hard for restricting our games by prohibiting those rules.

One might have the impression that the high complexity for the case without node creation is only due to consideration of information flow resp. labeling rules. But even without these rules, simple questions like connectivity cause computational problems. We present a corresponding PSPACE-hardness result in Thm. 4.6 below using a reduction to sabotage games again. For readers who prefer a reduction to a standard problem, we include also a NP-hardness proof (invoking the vertex cover problem).

Theorem 4.5 *In the non-expansive case without consideration of information flow, the dynamic model-checking problem with respect to connectivity in the safety version is NP-hard.*

Proof We reduce the *vertex cover* problem which is well-known to be NP-hard to the question whether Constructor can guarantee connectivity of the network in the safety game.

We state the vertex cover problem in the following form: Given a graph $G_{\mathrm{VC}} = (V', E')$ and an integer k, is there a vertex cover of the size k or less for G_{VC}?

We construct a game arena $\mathcal{G} = (G, R)$ with labels 'vertex', 'edge', 'keeper', 'storage', $\sqcup$ in Σ. The idea is to adapt G_{VC} for the initial graph of $\mathcal{G}$ by adding an unmovable strong node for each edge. We keep the original nodes of G_{VC} as erasable nodes. These erasable vertices are connected with a "storage" of k movable strong nodes $s_1, \dots, s_k$. The erasable vertices are also connected with an unmovable strong node r which keeps the vertices connected. So, in a play Destructor can only try to isolate the 'edge'-labeled nodes by deleting the 'vertex'-labeled nodes. Constructor can only preserve connectivity of G by moving the k strong nodes of the storage to the vertices which are a vertex cover for G_{VC}.

Additionally, k nodes labeled with $\sqcup$ are connected with the 'edge'-labeled nodes to prevent Destructor from winning before Constructor can move the nodes of the storage to the vertex cover. Destructor can delete these $\sqcup$-nodes within k turns and there is no way for Constructor to restore them.

For example in the graph in Fig. 5 (on the left) there exists a vertex cover of size 2, so two strong storage nodes (and two additional nodes labeled with $\sqcup$) are necessary for Constructor to preserve the connectivity (in the right-hand graph).

Formally, the initial game graph is $G = (V, E, A, (P_a)_{a\in\Sigma}, S)$ with $V := V' \cup E' \cup \{r, s_1, \dots, s_k, t_1, \dots, t_k\}$, and E arises from E' as in the figure. Additionally let $P_{\text{vertex}} := V'$ $P_{\text{edge}} := E'$, $P_{\text{keeper}} := \{r\}$, $P_{\text{storage}} := \{s_1, \dots, s_k\}$, $P_{\sqcup} := \{t_1, \dots, t_k\}$, and $S := E' \cup \{r, s_1, \dots, s_k\}$.

In order to allow movement of the strong storage nodes to the nodes labeled with 'vertex' we add a rule $\langle \text{storage} \xrightarrow{\text{move}} \text{vertex} \rangle$ to R. Note that this is the only rule in R, so that the other strong nodes (not labeled with 'storage') are unmovable.

Clearly, if for G_{VC} a vertex cover of size k exists, player Constructor wins by moving k strong nodes to the vertex cover.

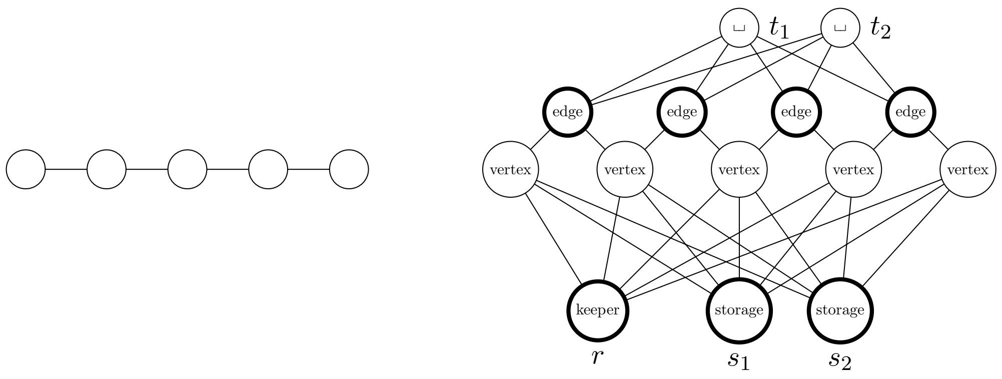

Figure 5. An ordinary graph G_{VC} and corresponding initial graph G of the safety game for testing G_{VC} for a vertex cover of size 2.

For the other direction notice that it is best for Constructor to move nodes to the erasable nodes labeled with 'vertex'. Since only the k strong storage nodes are movable, Constructor can only keep the persistence of k nodes of V_G of her choice. If Constructor wins, these nodes are a vertex cover of size k for G_{VC}. $\square$

As announced, we now sharpen this result to PSPACE-hardness, using a reduction to the solution of sabotage games (which is PSPACE-complete [17,18,20]). We show that the problem is PSPACE-hard even in the case we omit the distinction of node labels (i. e., all nodes are labeled with $\sqcup$).

Theorem 4.6 *In the non-expansive case without consideration of information flow and without distinction of node labels, the dynamic model-checking problem with respect to connectivity in the safety version is PSPACE-hard.*

Proof Consider a sabotage game $\mathcal{G}_s = (G_s, v_{\mathrm{in}})$ on the initial game graph $G_s = (V_s, E_s)$ with a set $F \subseteq V_s$ of final vertices. We construct a game arena $\mathcal{G} = (G, R)$ where all vertices are labeled with $\sqcup$, so that Constructor can preserve the connectivity of the graph in $\mathcal{G}$ iff Runner wins the sabotage game $\mathcal{G}_s$.

The idea is to simulate the movement of player Runner by the movement of a strong node. Intuitively, Constructor should be able to move a strong node to a "final" vertex in $\mathcal{G}$ iff Runner reaches this final vertex of F in the game $\mathcal{G}_s$. Then, Constructor can prevent Destructor from destroying the connectivity iff Constructor can move a strong node to a "final" vertex. We will ensure the latter by connecting the final vertices with a complete graph $K_{|V_s|}$ of $|V_s|$ nodes. Each final vertex is connected with all nodes of $K_{|V_s|}$, and each node of $K_{|V_s|}$ is connected with an additional node called v_{target}. Destructor can isolate v_{target} by deleting the complete subgraph $K_{|V_s|}$ iff Constructor cannot move a strong node towards a "final" vertex to $K_{|V_s|}$. For the rest of the arena Constructor can always guarantee connectivity in $\mathcal{G}$, but with the handicap that a strong node can be moved to a final vertex iff Runner wins in $\mathcal{G}_s$.

In order to realize the construction, we shall replace the edges of G_s by so-called *gates* which we depict by a bold square; an example for the simulation is depicted in Fig. 6 which shows the initial graph G_s of an example sabotage game and the

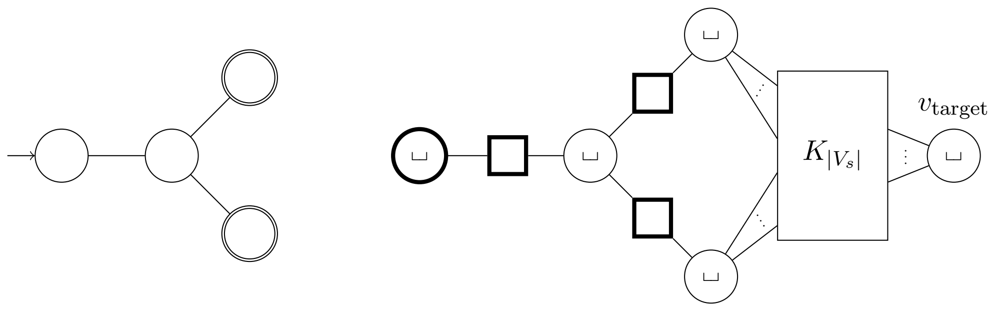

Figure 6. Initial graph G_s of a sabotage game and corresponding initial graph G of the safety game with respect to connectivity.

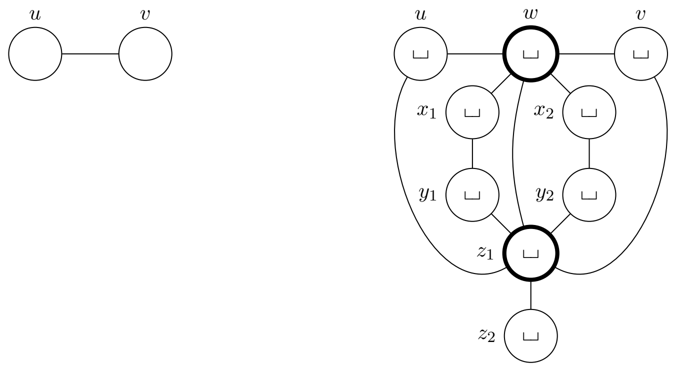

Figure 7. An edge between nodes u and v in the sabotage game and its replacement gate.

equivalent initial game graph G of our game.

The replacement gate for an edge between two nodes u and v is depicted in Fig. 7. All gates in the graph share the same vertices z_1 and z_2. Each node of the complete subgraph $K_{|V_s|}$ described above is connected to z_1 as well. Thus, all vertices other than v_{target} stay connected if Constructor always skips. The strong node z_1 is unmovable, otherwise Destructor isolates z_2 by deletion of z_1. The movement of Runner from u to v is simulated by the situation that Constructor can move the strong node at w to v without giving Destructor the opportunity to isolate one of the nodes x_i. Therefore Constructor needs a strong node at u which she can move towards w in the case that Destructor deletes a y_i. Conversely, if u is not strong and Destructor deletes one of the nodes y_i, Constructor cannot move the strong node at w to v anymore (even in the case that u becomes strong later), because Destructor could subsequently isolate x_i by deletion of w. The x_i-nodes and y_i-nodes exist twice to prevent Constructor of "securing" the gate by moving the strong node at w to a x_i node. (If Constructor moves the strongness of w to one x_i-node, Destructor can achieve the isolation of the other one.)

Formally, we construct the initial game graph $G = (V, E, A, S, (P_\sqcup))$ with $V := V_s \cup E_s \cup \{w, x_1, x_2, y_1, y_2\} \times E_s \cup \{z_1, z_2\} \cup V(K_{|V_s|}) \cup \{v_{\text{target}}\}$, and E arises from E_s as in the figures. Additionally let $P_\sqcup := V$, and $S := \{v_{\text{in}}, z_1\} \cup \{w\} \times E_s$. With $V(K_{|V_s|})$ we denote the vertices of the complete subgraph $K_{|V_s|}$, and v_{target} is the node which Destructor tries to isolate. The vertex v_{target} is only connected to the nodes in $K_{|V_s|}$, and each node of $K_{|V_s|}$ is additionally connected to each “final” vertex and to the vertex z_1. The only rule in the set R is $\langle \sqcup \xrightarrow{\text{move}} \sqcup \rangle$ which allows arbitrary movement of strongness (to weak nodes).

In this proof we assume Constructor starts the game (alike Runner starts the sabotage game). (This could be easily achieved by connecting the strong node in $\mathcal{G}$ which corresponds to the initial node in $\mathcal{G}_s$ with one of the “final” nodes by an additional gate. So, Destructor deletes a y_i-node of this gate in his first turn.)

Assume, Runner has a winning strategy in $\mathcal{G}_s$. For each turn where Runner moves from a node u to a node v in $\mathcal{G}_s$, Constructor moves the strongness of w to v in the corresponding replacement gate for the edge (u, v) in $\mathcal{G}$. If Destructor deletes one of the nodes y_1, y_2, or w in this gate later on, Constructor reacts by securing this gate by moving the strongness of u to w (otherwise this move is not necessary). Since Runner reaches a final state in $\mathcal{G}_s$ within $|V_s| - 1$ turns, Constructor can move the strongness to a associated node in $\mathcal{G}$ before Destructor can delete $|V_s| - 1$ vertices of $K_{|V_s|}$. Constructor can finally move this strong node to a node of $K_{|V_s|}$, hence she wins in $\mathcal{G}$.

Conversely, assume that Blocker has a winning strategy in $\mathcal{G}_s$. For each deletion of an edge (u, v) in $\mathcal{G}_s$, Destructor deletes a y_i-node of the associated gate. In the case that Constructor moves the strongness of a w-node to a v-node without having the required strongness at the associated u-node, Destructor reacts by deleting a y_i-node of this gate (and successively the w-node if Constructor does not move back). Analogously, if Constructor moves strongness of a w-node to a x_i-node Destructor tries to isolate the other x_i-node. And in the case that Constructor moves the strongness of z_1, Destructor wins by deleting z_1. Since Runner loses the sabotage game, Constructor cannot move a strong node towards a “final” node to the complete subgraph $K_{|V_s|}$ without losing the game. After blocking the replacement gates according to Blocker’s strategy in $\mathcal{G}_s$, Destructor can delete all vertices of $K_{|V_s|}$. In this case v_{target} becomes isolated and Destructor wins, too.

Hence, Constructor can preserve the connectivity in $\mathcal{G}$ iff Runner wins the sabotage game $\mathcal{G}_s$. □

In the present paper we do not settle the question, whether the considered non-expansive cases of the model-checking problem are actually solvable in PSPACE.

5 Perspectives

We have presented in this paper a very flexible model of dynamic networks based on the paradigm of a game between the two players Destructor and Constructor. We showed some preliminary results on the status of the dynamic model-checking problem associated with this game.

Our current work addresses various restrictions of the framework. It is obvious that interesting algorithmic solutions can only be obtained in more constrained scenarios.

However, this does not apply to the specifications. In practice it may be relevant to guarantee a behavior of dynamical networks which transcends the range of reachability or safety properties. Already the use of the "until" operator may be interesting; it allows to express reachability properties under an additional constraint (whether a certain property is reached and until then another constraint is met).

Another leading aspect is that yes/no questions as studied in this paper (i. e., whether a dynamic model-checking specification is satisfied or not) has to be refined. From a practical point of view the formulation of optimization problems is more useful. As an example consider a network in grid form, say a two-dimensional $n \times n$ array of nodes. In the associated safety game without labeling constraints the connectivity is to be preserved. The question is how many strong nodes are required to ensure this. It is clear that Destructor wins if around some node v all nodes of distance < 6 (including v itself) fail to be strong: In this case, Destructor can successively delete the at most four neighbors of v and thus isolate v from the rest of the network, destroying connectivity. To determine the precise minimal number of strong nodes necessary to ensure connectivity is an optimization issue extending the question whether a given network (with a given set of strong nodes) allows to satisfy a safety specification.

References

[1] Awerbuch, B., P. Berenbrink, A. Brinkmann and C. Scheideler, *Simple routing strategies for adversarial systems*, in: *Proceedings of FOCS* (2001), pp. 158–167.

[2] Awerbuch, B., A. Brinkmann and C. Scheideler, *Anycasting in adversarial systems: Routing and admission control*, in: *Proceedings of ICALP*, Lecture Notes in Computer Science **2719** (2003), pp. 1153–1168.

[3] Bouajjani, A., P. Habermehl, A. Rogalewicz and T. Vojnar, *Abstract regular tree model checking of complex dynamic data structures*, in: *Proceedings of SAS*, Lecture Notes in Computer Science **4134** (2006), pp. 52–70.

[4] Corradini, A., *GETGRATS: A summary of scientific results (with annotated bibliography)*, Electronic Notes in Theoretical Computer Science **51** (2001).

[5] Demri, S., *A reduction from DLP to PDL*, Journal of Logic and Computation **15** (2005), pp. 767–785.

[6] Edmund M. Clarke, J., O. Grumberg and D. A. Peled, "Model checking," MIT Press, Cambridge, MA, USA, 1999.

[7] Gadducci, F., R. Heckel and M. Koch, *A fully abstract model for graph-interpreted temporal logic*, in: *Proceedings of TAGT*, Lecture Notes in Computer Science **1764** (1998), pp. 310–322.

[8] Göller, S., *On the complexity of reasoning about dynamic policies*, in: *Proceedings of CSL*, Lecture Notes in Computer Science **4646** (2007), pp. 358–373.

[9] Grädel, E., W. Thomas and T. Wilke, "Automata, Logics, and Infinite Games," Lecture Notes in Computer Science **2500**, Springer, 2002.

[10] Hesse, W., *The dynamic complexity of transitive closure is in DynTC^0*, Theoretical Computer Science **3** (2003), pp. 473–485.

[11] Hesse, W. and N. Immerman, *Complete problems for dynamic complexity classes*, in: *Proceedings of LICS* (2002), pp. 313–322.

[12] Holm, J., K. de Lichtenberg and M. Thorup, *Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity*, Journal of the ACM **48** (2001), pp. 723–760.

[13] Jussila, T., J. Dubrovin, T. Junttila, T. Latvala and I. Porres, *Model checking dynamic and hierarchical UML state machines*, in: *Proceedings of MoDeV^{2}a* (2006), pp. 94–110, http://modeva.itee.uq.edu.au.

[14] King, V. and G. Sagert, *A fully dynamic algorithm for maintaining the transitive closure*, Journal of Computer and System Sciences **65** (2002), pp. 150–167.

[15] Lerda, F. and W. Visser, *Addressing dynamic issues of program model checking*, in: *Proceedings of SPIN*, Lecture Notes in Computer Science **2057** (2001), pp. 80–102.

[16] Löding, C. and P. Rohde, *Model checking and satisfiability for sabotage modal logic*, in: *Proceedings of FSTTCS*, Lecture Notes in Computer Science **2914** (2003), pp. 302–313.

[17] Löding, C. and P. Rohde, *Solving the sabotage game is PSPACE-hard*, Technical Report AIB-05-2003, RWTH Aachen (2003).

[18] Löding, C. and P. Rohde, *Solving the sabotage game is PSPACE-hard*, in: *Proceedings of MFCS*, Lecture Notes in Computer Science **2747** (2003), pp. 531–540.

[19] Rohde, P., *Moving in a crumbling network: The balanced case*, in: *Proceedings of CSL*, Lecture Notes in Computer Science **3210** (2004), pp. 310–324.

[20] Rohde, P., "On Games and Logics over Dynamically Changing Structures," Ph.D. thesis, RWTH Aachen (2005).

[21] Rozenberg, G., editor, "Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations," World Scientific, 1997.

[22] van Benthem, J., *An essay on sabotage and obstruction*, in: D. Hutter and W. Stephan, editors, *Mechanizing Mathematical Reasoning, Essays in Honor of Jörg H. Siekmann on the Occasion of His 60th Birthday*, Lecture Notes in Computer Science **2605** (2005), pp. 268–276.