RWTH Aachen

Department of Computer Science
Technical Report

Solving the Sabotage Game is PSPACE-hard

Christof Löding and Philipp Rohde

ISSN 0935–3232 · Aachener Informatik Berichte · AIB-2003-05

RWTH Aachen · Department of Computer Science · March 2003

Solving the Sabotage Game is PSPACE-hard

Christof Löding and Philipp Rohde

Lehrstuhl für Informatik VII
RWTH Aachen
{loeding,rohde}@informatik.rwth-aachen.de

Abstract. We consider the sabotage game presented by van Benthem in an essay on the occasion of Jörg Siekmann's 60th birthday. In this game one player moves along the edges of a finite, directed or undirected multi-graph and the other player takes out a link after each step. There are several algorithmic tasks over graphs which can be considered as winning conditions for this game, for example reachability, Hamilton path or complete search. As the game definitely ends after at most the number of edges (counted with multiplicity) steps, it is easy to see that solving the sabotage game for the mentioned tasks takes at most PSPACE in the size of the graph. We will show that on the other hand solving this game in general is PSPACE-hard for all conditions. We extend this result to some variants of the game (removing more than one edge per round and removing vertices instead of edges). Finally we introduce a modal logic over changing models to express tasks corresponding to the sabotage games. We will show that model checking this logic is PSPACE-complete.

1 Introduction

In some fields of computer science, especially the controlling of reactive systems an interesting sort of tasks arises, which consider temporal changes of a systems itself. In contrast to the usual tasks over reactive systems, where movements *within* a system are considered, an additional process affects: the dynamic change of the system itself. Hence we have two different processes: a *local* movement within the system and a *global* change of the system.

For example consider a server network such that, during its work, some connections between the servers break down or some servers stop working (caused by technical malfunction or by wilful damage). We can model this situation by a network graph where from time to time some edges or vertices (with their associated edges) are removed. Some natural questions arise for this system: is it possible – regardless of the removed connections – to interchange information between two designated servers? Is there a protocol which guarantees that the destination can be reached?

Another example for a task of this kind was recently given by van Benthem [vB02], which can be described as the *real Travelling Salesman Problem*: is it possible to find your way between two cities within a railway network where a malevolent demon starts cancelling connections?

As usual one can model such kind of reactive system as a two-person game, where one player (the control program, the protocol, the salesman) plays against the other (the environment, the disturbance, the malevolent demon). As a winning condition one can consider several algorithmic tasks over graphs, for example reachability, Hamilton path or complete search. Determining the winner of these games gives us the answers for our original tasks.

In this paper we will show that solving the *sabotage games* where one player (the *Runner*) moves along edges in a multi-graph and the other player (the *Blocker*) removes an edge in each round is PSPACE-hard for the three mentioned winning conditions. Further we extend this result to some variants of the game. Finally we consider a modal logic over *changing models* to express tasks corresponding to sabotage games and we show that model checking this logic is PSPACE-complete.

The main aspect of the sabotage game is that the *Runner* can only act *locally* by moving one step further from his actual position whereas the *Blocker* has the possibility to behave *globally* on the arena of the game. So the sabotage game is in fact a match between a *local* and a *global* player.

This paper is organised as follows. In Sect. 2 we introduce the basic notions of the sabotage game and repeat the definitions of the mentioned winning conditions. In Sect. 3 we show the PSPACE-hardness for the sabotage game with the reachability condition on undirected graphs. For that we give a polynomial time reduction from the PSPACE-complete problem of *Quantified Boolean Formulas* to these games. In Sect. 4 we modify the construction for directed graphs. The power of the saboteur is modified in Sect. 5: we consider variants of the game (removing more than one edge per round and removing vertices instead of edges) and show that solving these variants has the same complexity. In Sect. 6 we give polynomial time reductions from sabotage games with reachability condition to the other winning conditions. In the last section we introduce the extension SML of modal logic over transitions systems which captures the concept of removing edges, i.e., SML is a modal logic over changing models. We give the syntax and the semantics of SML and provide a translation to first order logic. By applying the results of the first part we will show that model checking this logic is PSPACE-complete.

2 The sabotage game

In this section we give the definition of the sabotage game and we repeat three algorithmic tasks over graphs which can be considered as winning conditions for this game: reachability, Hamilton path and complete search.

A *multi-graph* is a pair (V, e) where V is a non-empty, finite set of vertices and $e : V \times V \to \mathbb{N}$ is an edge multiplicity function, i.e., $e(u, v)$ denotes the number of edges between the vertices u and v. $e(u, v) = 0$ means that u and v are not connected. In case of an undirected graph we have in addition $e(u, v) = e(v, u)$ for all $u, v \in V$. A *single-graph* is given by a multiplicity function with $e(u, v) \leq 1$ for all vertices $u, v \in V$. The size of a multi-graph (V, e) is given by $|V| + |E|$, where we set $|E| := \sum_{u,v \in V} e(u, v)$ for directed graphs and $|E| := \frac{1}{2} \sum_{u,v \in V} e(u, v)$ for undirected graphs.

Let (V, e_0) be a multi-graph and $v_0 \in V$ be an initial vertex. The two-person sabotage game is played as follows: initially the game arena is $\mathcal{A}_0 = (V, e_0, v_0)$. The two players, which we call *Runner* and *Blocker*, move alternatingly, where the *Runner* starts his run from vertex v_0. At the begin of round n the *Runner* moves one step further along an existing edge of the graph, i.e., if v_n is his actual position, he chooses a $v_{n+1} \in V$ with $e_n(v_n, v_{n+1}) > 0$ an moves to v_{n+1}. Afterwards the *Blocker* removes one edge of the graph, i.e., he chooses two vertices u and v somewhere in the graph with $e_n(u,v) > 0$. In the directed case we define $e_{n+1}(u,v) := e_n(u,v) - 1$ and $e_{n+1}(\cdot,\cdot) := e_n(\cdot,\cdot)$ otherwise. In the undirected case we let $e_{n+1}(u,v) := e_{n+1}(v,u) := e_n(u,v) - 1$. The multi-graph $\mathcal{A}_{n+1} = (V, e_{n+1}, v_{n+1})$ becomes the arena for the next round. The game ends, if either the *Runner* cannot make a move, i.e., there is no link starting from his actual position or if the winning condition is fulfilled.

As a winning condition for the sabotage game on an undirected or directed graph one can consider the usual tasks over graphs, for example:

1. *Reachability*: the *Runner* wins iff he can reach a given vertex (which we call the *goal*)
2. *Hamilton Path* or *Travelling Salesman*: the *Runner* wins iff he can move along a Hamilton Path, i.e., he visits each vertex exactly once
3. *Complete Search*: the *Runner* wins iff he can visit each vertex (possibly more than once)

For the reachability game with one single goal the use of multi-graphs is crucial:

Lemma 1. *The sabotage game with reachability condition on a single-graph and one single goal can be solved in linear time.*

Proof. The *Runner* wins the game iff the goal is the initial vertex or a vertex which is reachable in the first move. Checking this needs at most linear time. In the other cases the *Blocker* has a winning strategy: if the *Runner* moves to a vertex v such that there is a (single) edge connecting v and the goal then the *Blocker* removes this edge. Otherwise he removes an arbitrary edge. $\square$

Remark 1. With Hamilton path or complete search condition the sabotage game even can be solved in constant time: the *Runner* wins iff the graph contains only one single vertex or two connected vertices. In all other cases the *Blocker* can isolate a fixed vertex by the strategy given above.

In the sequel we only consider the sabotage game with reachability condition. For the representation of game arenas we use the following conventions: the label of edges gives us the multiplicity $e(u,v)$. If no number is displayed then the link is a single edge, i.e., it has multiplicity 1. Further we display the goal – the vertex which has to be reached by the *Runner* to win the game – as a circled bullet $\odot$. For better readability the goal can be displayed multiple times in a figure, but it is always meant as *one single* vertex. Capital letters always denote vertices.

For the reachability condition we can bound the multiplicity uniformly by two or, if we allow a second goal vertex, we even can transform every multi-graph game into a single-graph game:

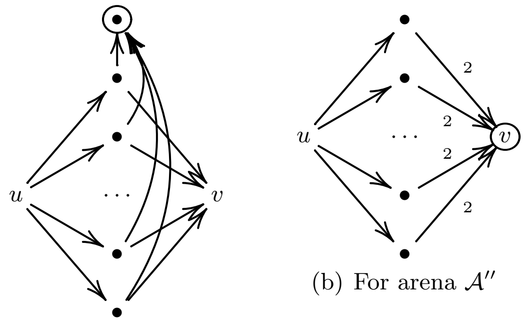
Lemma 2. *Let G be a sabotage game with reachability condition on a multi-graph arena $\mathcal{A}$. Then there are games G', G'' on arenas $\mathcal{A}', \mathcal{A}''$ with a size polynomial in the size of $\mathcal{A}$ such that the* Runner *wins G iff he wins G', resp. G'', and*

1. *$\mathcal{A}'$ is a single-graph with two goals*
2. *$\mathcal{A}''$ is a multi-graph with one goal and only single or double edges. Moreover the double edges occur only connected with the goal.*

Proof. We only consider directed graphs; undirected graphs are treated analogously. First, we construct $\mathcal{A}'$. For that we start with the original set of vertices and add a second goal vertex to the arena $\mathcal{A}'$. Single-edges are transferred directly. For every edge in $\mathcal{A}$ with multiplicity > 1 connecting vertices u, v we add k new vertices and connect each of them with u, v and the new goal vertex as depicted in Fig. 1(a). Moving from vertex u towards v the *Runner* has as many possibilities as in the original game. But the *Blocker* does not gain an additional move when the *Runner* reaches one of the new vertices: he has to delete the connection to the goal to prevent his loss. Applying the previous lemma it is clear that we actually need a new goal, if v is the original goal.

The size of the arena $\mathcal{A}' = (V', e')$ is estimated by the inequalities $|V|' \leq |V| + |E| + 1$ and $|E|' \leq 3\,|E|$.

The arena $\mathcal{A}''$ is constructed similarly. If v is not the original goal we apply the same construction, but reusing the existing goal instead of adding a new one. If v is the goal then we add double edges from the new vertices to v as depicted in Fig. 1(b). The estimation of the size of $\mathcal{A}''$ is the same as above. $\square$



(a) For arena $\mathcal{A}'$

(b) For arena $\mathcal{A}''$

Fig. 1. Transformation of multi-edges

Since edges are only deleted but not added during the play the following fact is easy to see:

Lemma 3. *If the* Runner *has a winning strategy in the sabotage game with reachability condition then he can win without visiting any vertex twice.*

In the sequel we will introduce several game arenas where we use edges with a multiplicity 'high enough' to ensure that the *Blocker* cannot win the game by reducing these edges. In figures these edges are represented by a curly link $\bullet \sim\!\sim\!\sim \bullet$. For the moment we can consider these links to be 'unremovable'. Due to the last lemma we have: if the *Runner* can win the reachability game at all, then he can do so within at most $|V| - 1$ rounds. Hence we can set the multiplicity of the 'unremovable' edges to $|V| - 1$. To bound the multiplicity of edges uniformly one can apply Lemma 2.

3 PSPACE-hardness for sabotage reachability games on undirected graphs

In this section we prove that the PSPACE-complete problem of *Quantified Boolean Formulas*, QBF for short, can be reduced by a polynomial time reduction to sabotage games on undirected graphs with the reachability condition.

Let φ be an instance of QBF, i.e., a formula of the form[1]

$$\varphi \equiv \exists x_1 \forall x_2 \exists x_3 \dots Q x_n \psi,$$

where Q is $\exists$ for n odd and $\forall$ otherwise and ψ is a quantifier-free boolean formula in conjunctive normal form.

We will construct an undirected game arena for a sabotage game G_φ with a reachability condition such that the *Runner* has a winning strategy in the game iff the formula φ is satisfiable, i.e., there is an assignment for x_1 such that for all assignments for x_2 there exists ... and ψ is satisfied by the overall assignment.

In the first approach one would like to use a reduction like the classical one from QBF to the *Geography Game*,[2] but then the *Blocker* can destroy connections in a part of the graph which should be visited only later in the game. So possibly he is able to prevent the intended sequence of choices and block the game in an unwished manner. One could solve this problem by arranging the 'choice gadgets' of the *Geography Game* with increasing multiplicity. But then one has to blow up the distance between two successive gadgets to give the *Blocker* the chance to remove all necessary edges at each 'choice point' before the *Runner* reaches these vertices. It is easy to see that this 'accordion approach' results in an arena with a size exponential in the size n of φ. So the key to find a reduction which provides an arena with a size polynomial in the size of φ is to restrict the liberty of the *Blocker* in a more sophisticated way, i.e., to force him removing edges only 'locally'.

Similar to the construction of the *Geography Game* the game arena G_φ consists of two parts: a chain of n gadgets where first the *Runner* chooses an assignment for x_1, then the *Blocker* chooses an assignment for x_2 before the *Runner* chooses an assignment for x_3 and so on. The second part gives the *Blocker* the possibility to select one of the clauses of ψ. The *Runner* must certify that this clause is indeed satisfied by the chosen assignment: he can reach the goal vertex and win the game iff at least one literal in the clause is true under the assignment.

Figure 15 shows an example of the sabotage game G_φ for the formula $\varphi \equiv \exists x_1 \forall x_2 \exists x_3\ (c_1 \wedge c_2 \wedge c_3 \wedge c_4)$ where we assume that each clause consists of exactly three literals. In the following we describe in detail the several components of G_φ and their arrangement. The main step of the construction is to take care about the opportunity of the *Blocker* to remove edges *somewhere* in the graph.

3.1 The $\exists$-gadget

The gadget where the *Runner* chooses an assignment for the x_i with i odd is displayed in Fig. 2. We are assuming that the run reaches this gadget at vertex A at the first time. Vertex B is intended to be the exit. In the complete construction

[1] Without loss of generality we consider only formulas beginning with an existential quantifier.
[2] See for [Pap94], p. 460.

there are also edges from X_i, resp. $\overline{X_i}$ leading to the last gadget of the graph, represented as dotted lines labelled by *back*. We will see later that taking these edges as a shortcut, *starting from* the ∃-gadget directly to the last gadget is useless for the *Runner*. The only meaningful direction is *coming from* the last gadget back to the ∃-gadget. So we temporary assume that the *Runner* does not take these edges.

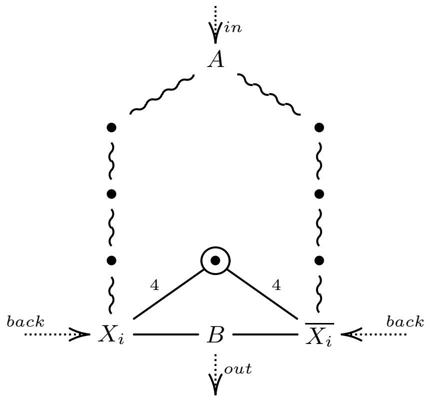


Fig. 2. The ∃-gadget for x_i with i odd

The *Runner* makes his choice simply by moving from A either to the left or to the right. Thereby he moves either towards X_i if he wants x_i to be `false` or towards $\overline{X_i}$ if he wants x_i to be `true`. We only consider the case that he chooses the left hand side since the other case is similar. Because of Lemma 3 we can further assume that he does not move backwards. Then the *Blocker* has exactly four steps to remove all the links between X_i and the goal before the *Runner* can reach X_i. On the other hand the *Blocker* must delete these links since otherwise the *Runner* can reach the goal directly. So the *Blocker* cannot remove edges somewhere else in the graph without loosing the game.

Why we use four steps here will be clarified later on. If the *Runner* has reached X_i and he moves towards B then the *Blocker* has to delete the edge between B and $\overline{X_i}$ since otherwise the *Runner* can reach the goal on this way (there are still four edges left between $\overline{X_i}$ and the goal). Altogether we have:

Lemma 4. *Suppose that the* Blocker *wants to isolate the goal and that the* Runner *does not move backwards. Then the ∃-gadget, when the* Runner *reaches the exit, has one of the two appearances displayed in Fig. 3 and it is for the* Runner *to decide which case arises. During the run through this gadget the* Blocker *cannot delete edges somewhere else in the graph without loosing the game immediately.*

3.2 The ∀-gadget

The gadget where the *Blocker* chooses an assignment for the x_i with i even is a little bit more sophisticated. Figure 4 shows the construction.

In the sequel we are again assuming, due to Lemma 3, that the *Runner* does not move backwards. Furthermore, we are assuming as in the last case that he does not take the dotted *back*-edges as a shortcut.

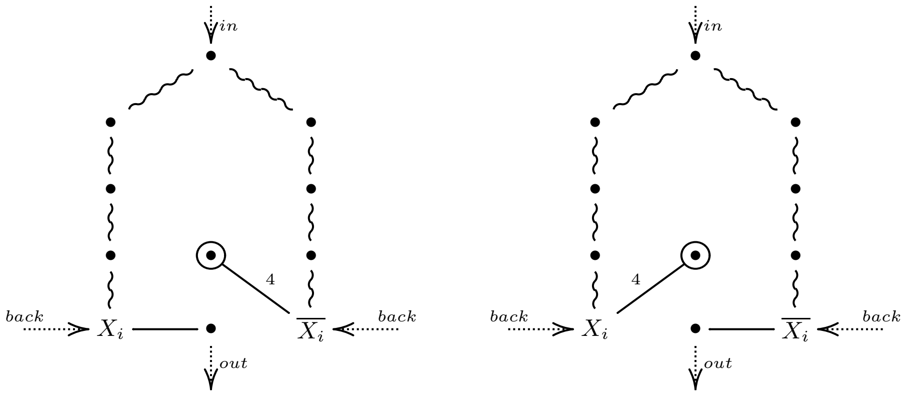


Fig. 3. The two cases after traversing the ∃-gadget

If the *Blocker* wants x_i to be `false` he tries to lead the *Runner* towards X_i. In this case he simply removes the three edges between C and $\overline{X_i}$ during the first three steps. Then the *Runner* has to move across D and in the meantime the *Blocker* deletes the four edges between X_i and the goal to ensure that the *Runner* cannot win directly. Just the same way as above he removes in the last step the link between B and $\overline{X_i}$ to prevent a premature end of the game (since the four edges between $\overline{X_i}$ and the goal are still left).

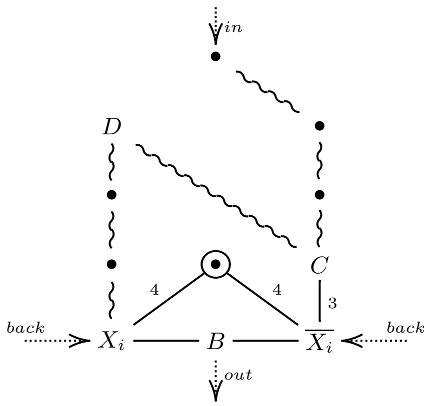


Fig. 4. The ∀-gadget for x_i with i even

On the other hand if the *Blocker* wants to assign `true` to x_i he should lead the *Runner* towards $\overline{X_i}$. To achieve this aim he removes three of the four links between $\overline{X_i}$ and the goal before the *Runner* reaches C. Nevertheless the *Runner* has the free choice at vertex C whether he moves towards X_i or towards $\overline{X_i}$, i.e., the *Blocker cannot guarantee* that the run goes across $\overline{X_i}$. But let us consider the two possible cases: first we assume that the *Runner* moves as intended and uses the edge between C and $\overline{X_i}$. In this round the *Blocker* removes the last link from $\overline{X_i}$ to the goal. Then the *Runner* moves to B and the *Blocker* deletes the edge from B to X_i as before.

Now assume that the *Runner* 'misbehaves' and moves from C to D and further towards X_i. Then the *Blocker* first removes the four edges between X_i and the goal. When the *Runner* now moves from X_i to B the *Blocker* has to take care that the *Runner* cannot reach the goal via the link between B and $\overline{X_i}$ (there is still one edge left from $\overline{X_i}$ to the goal). There are several ways to realise that: 1) He can delete the last link between $\overline{X_i}$ and the goal and therefore isolate the goal completely within this gadget, 2) he cuts the link between B and $\overline{X_i}$ such that the *Runner* has to exit the gadget or 3) he waits for the decision of the *Runner* and deletes an edge *somewhere* in the game graph. Only if the *Runner* moves from B to $\overline{X_i}$ he removes the last link from $\overline{X_i}$ to the goal. In this way the *Blocker* obtains one additional opportunity to remove edges. When we have introduced the verification gadget in the next subsection it will be clear that all three alternatives are of no advantage for the *Runner*: after traversing this last gadget the way leads back to one of the literal vertices (which one depends on the formula and on choices of both players) and the *Runner* will only win if there are at least two edges left connecting the goal with this literal vertex (the number of edges depends on the chosen assignment). But if the *Runner* misbehaves within the $\forall$-gadget and the *Blocker* plays as described above then, if the way leads back to this gadget there are certainly not enough edges left.

To summarise the traversing of the $\forall$-gadget we obtain:

Lemma 5. *Suppose that the* Blocker *wants to isolate the goal, that the* Runner *does not move backwards, and that he wants to prevent an extra chance for the* Blocker *to delete edges. Then the $\forall$-gadget, when the* Runner *reaches the exit, has one of the two appearances displayed in Fig. 5 and it is for the* Blocker *to decide which case arises. During the run through this gadget the* Blocker *cannot delete edges somewhere else in the graph without loosing the game immediately.*

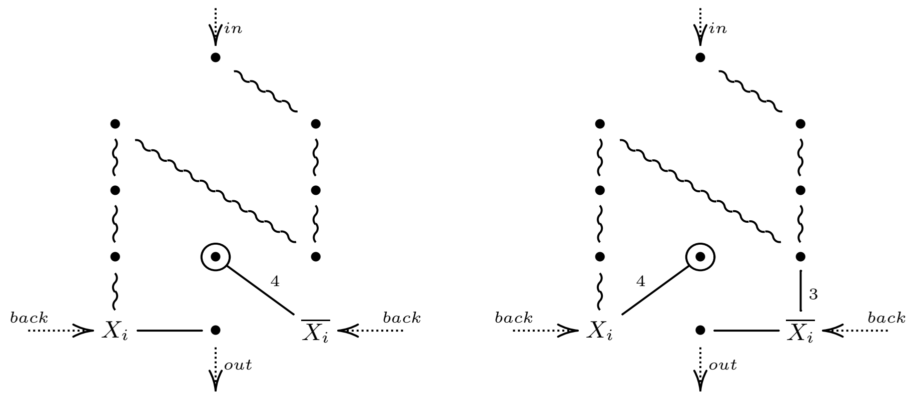


Fig. 5. The two essential cases after traversing the $\forall$-gadget

3.3 The verification gadget

The last component of the arena is a gadget where the *Blocker* can choose one of the clauses of the formula ψ. Before we give the representation of this gadget let us explain the idea. If the *Blocker* chooses the clause c then the *Runner* can select for his part one literal x_i of c. There is an edge 'back' to the $\exists$-gadget if i is odd or to the $\forall$-gadget if i is even, videlicet to X_i if x_i is positive in c, resp. to $\overline{X_i}$ if x_i is negative in c (these are the edges represented by dotted lines and labelled by *back* in the two previous gadgets). So if the chosen assignment satisfies ψ, then for all clauses of ψ there is at least one literal which is `true` (ψ is in conjunctive normal form). Since the path through the assignment gadgets visits the opposite truth values this means that there is at least one edge back to an X_i, resp. $\overline{X_i}$, which itself is connected to the goal by an edge with a *multiplicity of four* (assuming that the *Runner* did not misbehave in the $\forall$-gadget), see Lemmas 4 and 5. Therefore the *Runner* can reach the goal and wins the game.

For the converse if the chosen assignment does not satisfy ψ, then there is a clause c in ψ such that every literal in c is assigned `false`. If the *Blocker* chooses this clause c then every edge back to the assignment gadgets ends in an X_i, resp. $\overline{X_i}$, which is *unconnected* to the goal (again by Lemmas 4 and 5). If we show that there is no other way to reach the goal this means that the *Runner* looses the game.

But we have to be very careful neither to allow any shortcuts for the *Runner* nor to give the *Blocker* to much liberty. Figure 6 shows the verification gadget for $\psi \equiv c_1 \wedge c_2 \wedge c_3 \wedge c_4$ where each clause c_i has exactly three literals. The curly edges at the bottom of the gadget lead back to the corresponding literals of each clause. For example if $c_1 \equiv x_1 \vee \neg x_2 \vee \neg x_4$ then the first curly edge under the vertex C_1 leads to X_1 in the first gadget ($\exists$), the second to $\overline{X_2}$ in the second gadget ($\forall$), and the third to $\overline{X_4}$ in the fourth gadget ($\forall$).

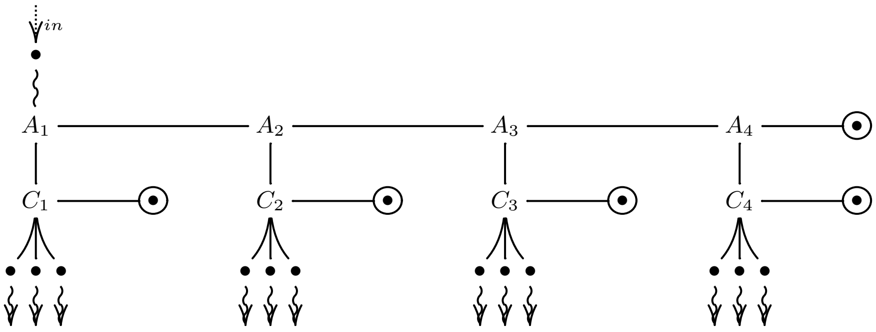


Fig. 6. The verification gadget with four clauses

The *Blocker* chooses the clause c_k by first removing the edges from A_j to C_j for $j < k$ one after the other. Then he cuts the link between A_k and A_{k+1}, resp. between A_k and the goal if c_k is the last clause. By Lemma 3 it is useless for the *Runner* to go back, thus he can only follow the given path to C_k. If he reaches this vertex the *Blocker* must remove the link from C_k to the goal to prevent the win for the opponent. In the next step the *Runner* selects a literal x_i, resp. $\neg x_i$ in

c_k, moves towards the corresponding vertex and afterwards along the curly edge back to the assignment gadgets as described above. At this point the *Blocker* has exactly *two* moves left, i.e., he is allowed to remove two edges somewhere in the graph. But by Lemma 4 and 5 we have: if the 'right' assignment for this literal has been chosen then there are exactly *four* edges left connecting the corresponding vertex and the goal. So the *Blocker* has not the opportunity to isolate the goal and the *Runner* wins the game. Otherwise, if the 'wrong' assignment has been chosen then there is no link from X_i, resp. $\overline{X_i}$ to the goal left. Any continuation which the *Runner* could take either leads him back to an already visited vertex (which is a loss by Lemma 3) or, by taking another *back*-edge in the 'wrong' direction, to another vertex in the verification gadget. We handle the latter case in general.

What happens if the *Runner* uses one of the shortcuts starting from a literal vertex within the assignment gadgets which leads him directly to the bottom of the verification gadget? In this case the *Blocker* can prevent the continuation of the run at the bottom of the verification gadget by removing the corresponding single edge between the clause vertex C_k and the vertex beneath. So the *Runner* has to move back. This cannot be an advantage for him by Lemma 3, i.e., he wins the game if and only if he wins it without using any shortcut of this kind.

We have to consider more special cases: what happens if the *Blocker* does not behave as intended in the verification gadget? First, if the *Runner* reaches a vertex A_k and the *Blocker* removes either the edge between A_k and C_k *or* the one between C_k and the goal *or* one of the edges leading to the vertices beneath C_k (one for each literal in c_k) – in our example, if the *Blocker* removes *one* of the five solid edges displayed in Fig. 7 – then the *Runner* moves towards A_{k+1}, resp. towards the goal if c_k is the last clause. The *Runner* has to do so since, in the latter two cases, entering the 'damaged' area around C_k could be a disadvantage for him.

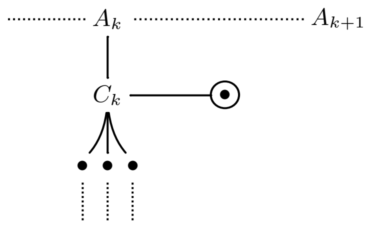


Fig. 7. A special case

Finally we consider the case that the *Blocker* removes an edge somewhere else in the graph instead. This behaviour is only reasonable if the *Blocker* cannot win by the 'regular' strategy described above, i.e., only if the chosen assignment satisfies ψ. So consider the round when the *Runner* reaches for the first time an A_k such that the edges from A_k to A_{k+1}, resp. the goal, *as well as* all edges connected to C_k are still left (i.e. all five solid edges in Fig. 7). If c_k is the last clause then the *Runner* just reaches the goal and wins the game. Otherwise he moves to C_k, chooses an appropriate literal x_i, resp. $\neg x_i$ such that the four edges from the corresponding vertex are still left (the chosen assignment satisfies φ, so at least one literal of this kind exists in each clause) and moves back to the assignment gadgets. Since A_k is the first vertex with this property the *Blocker* has gained only *one* additional move, so nevertheless it remains at least *one* edge from the vertex X_i, resp. $\overline{X_i}$ to the goal. So if the *Runner* can chose a satisfying assignment at all then the *Blocker* cannot prevent the win for the *Runner* by this behaviour. Since we had to cover this situation it is now clear why we used the multiplicity of four within the gadgets.

This completes the construction of the game G_φ. Figure 15 depict the complete arena for the example formula $\exists x_1 \forall x_2 \exists x_3\ (c_1 \wedge c_2 \wedge c_3 \wedge c_4)$. To summarise the results we obtain:

Lemma 6. *The* Runner *has a winning strategy in the sabotage game G_φ iff φ is satisfiable.*

Since the reduction presented above can easily be done in polynomial time we therefore obtain:

Theorem 1. *There is a polynomial time reduction from QBF to sabotage games with reachability winning condition on undirected graphs. In particular solving these games is PSPACE-hard.*

4 Modifying the game for directed graphs

Since each edge of the game G_φ presented in the last section has an 'intended direction', it is straight forward to check that the same construction works for directed graphs as well: just replace every undirected edge by one directed edge with the intended direction.

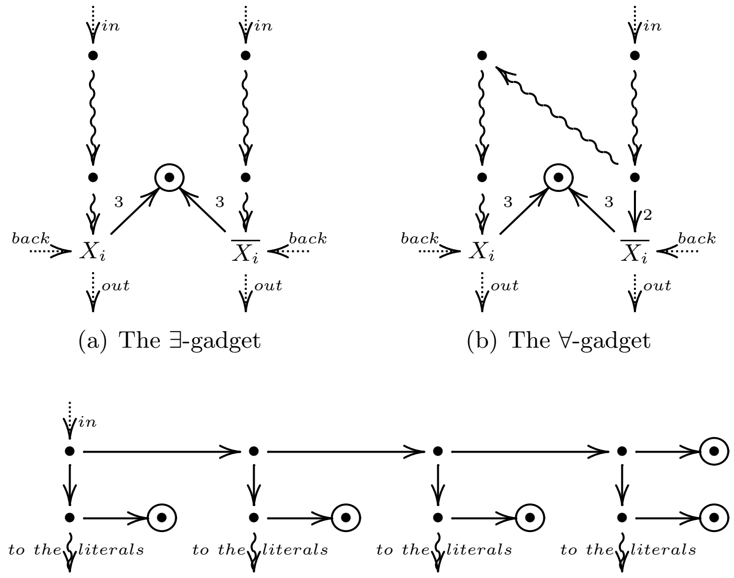
But since we do not have to take care of the shortcuts for the *Runner* there are in fact simpler arenas for the directed case. We can omit the ways back from B to X_i resp. $\overline{X_i}$ in the assignment gadgets and furthermore the shortcuts starting from the literal vertices to the bottom of the verification gadget. We present only the modified gadgets here and leave it to the reader to check that this arrangement works. Figure 8(a) and Fig. 8(b) show the two assignment gadgets and Fig. 8(c) the verification gadget with four clauses. To arrange the gadgets in line connect each *out* vertex to each *in* vertex of the next gadget by an 'unremovable' edge. The undermost arrows in the verification gadget represent 'unremovable' links back to each corresponding literal vertex in the assignment gadgets.

Therefore we obtain:

Theorem 2. *Solving sabotage games with reachability winning condition on directed graphs is PSPACE-hard.*

5 Extending the power of the saboteur

Regarding the last theorem a natural question arises: is the PSPACE-hardness still given if we liberate the movements of the *Blocker*? In this section we show how to modify the game graph G_φ to capture extensions of the sabotage game such that the reduction from QBF still works. We consider the following liberations for the *Blocker* for *undirected* graphs:



(a) The ∃-gadget (b) The ∀-gadget

(c) The verification gadget with four clauses

Fig. 8. The gadgets for directed graphs

1. Given a fixed number n he is allowed to remove up to n edges (counted by their multiplicity) in each round. It is also allowed that he removes none of them.
2. He removes vertices instead of edges, where the game arena is a single-graph. First we consider the case that he is allowed to remove one vertex in each round (together with the connected edges). To obtain a useful game we have to impose two restrictions: he is not allowed to remove the actual position where the *Runner* is and the goal is not affected.
3. The same as 1 for vertices of a single-graph: given a fixed number n he is allowed to remove up to n vertices in each round.

Variant 1

This extension is captured by the following fact:

Lemma 7. *For each of the three winning conditions reachability, Hamilton path and complete search there is a (polynomial time) reduction from sabotage games where the* Blocker *removes single edges to the one where he removes up to* n *edges in each round. In particular, solving Variant 1 of the game is PSPACE-hard.*

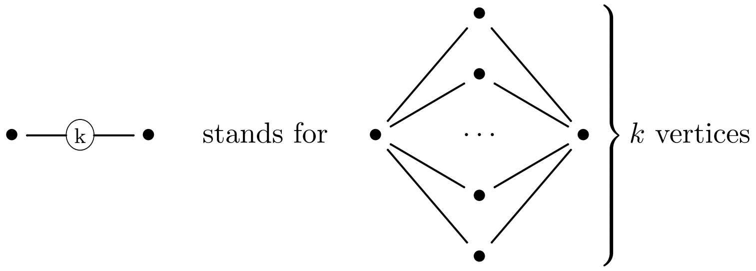
Proof. Just replace the multiplicity $e(u, v)$ of each edge in the original arena by $n \cdot e(u, v)$. Then the *Runner* wins the original game iff he wins the new game where the *Blocker* removes up to n edges in each round. □

Variant 2

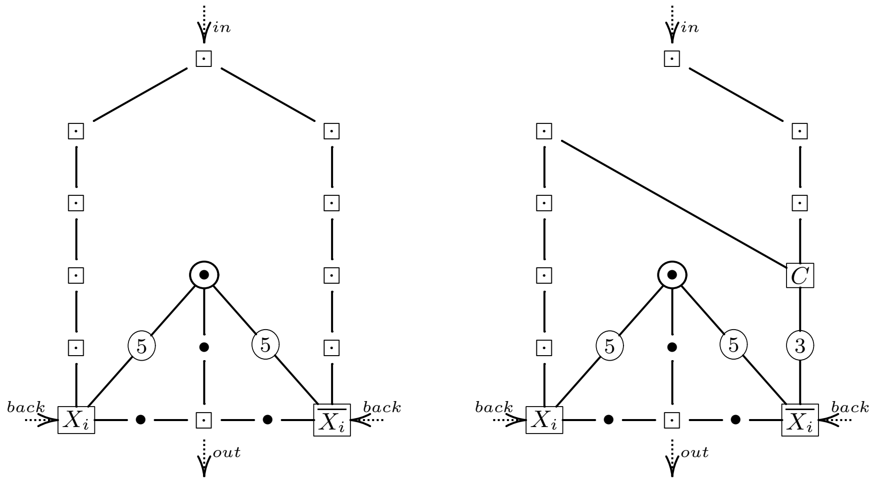
Although 'vertix deleting' can be viewed as a special kind of 'edge deleting' where simply each edge connected to the desired vertex is removed one cannot directly apply Theorem 1, resp., Theorem 2 for this variant of the game.

Also, it is not immediately clear that the complexity for 'vertex deleting' is the same as for 'edge deleting': for example Even and Tarjan [ET76] proved that a generalisation of the *Hex game* to graphs is PSPACE-complete when the players colour vertices. Surprisingly, the closely related game for edge colouring, which is known as the *Shannon switching game*, can be solved in polynomial time (see [BW70]). For an overview of these games and the corresponding results see [Dem01].

We have to modify the construction of the game arena for a 'vertex deleting' saboteur. We first consider this game with the additional restriction that there are 'unremovable' vertices (similar to the curly edges in the previous game). These vertices are displayed as $\boxdot$ if they are unlabelled or as a box containing the label. Later we will give a method to eliminate them. Further we use the following abbreviation:



It is clear that Lemma 3 is still valid. The general construction of the game G_φ for an instance φ of QBF is the same as in the previous section. We only have to modify the three kinds of gadgets. The new $\exists$-gadget is displayed in Fig. 9(a).



(a) The $\exists$-gadget (b) The $\forall$-gadget

Fig. 9. Gadgets for the 'vertex removing' game

The run through this gadget is similar to the previous case: the *Runner* decides whether x_i should be `true`, resp. `false` by moving right, resp. left at the beginning. We consider the case where he moves towards X_i (letting x_i be `false`). In the moment the *Runner* reaches X_i the *Blocker* by then has to remove the five vertices between X_i and the goal. Afterwards, when the *Runner* moves towards the exit of the gadget, the *Blocker* has to remove the two vertices above and on the right of the exit to keep his chance to win.

The new $\forall$-gadget is depicted in Fig. 9(b). Again, if the *Blocker* wants x_i to be `false` he leads the *Runner* towards X_i by removing the three vertices between C and $\overline{X_i}$. Afterwards, he has to act like in the $\exists$-gadget. If he wants x_i to be `true` he deletes three of the five vertices between $\overline{X_i}$ and the goal. Again, he cannot guarantee that the *Runner* moves towards $\overline{X_i}$, but if the *Runner* does so, he removes the remaining two vertices. If the *Runner* reaches the exit the *Blocker* has to delete the vertices above and on the left of the goal by then. If the *Runner* 'misbehaves' and moves from C towards X_i then the *Blocker* deletes the five vertices between X_i and the goal, afterwards the vertex above the goal. If the *Runner* then reaches the exit, the *Blocker* has obtained one extra move to delete a vertex somewhere in the graph, since, if the *Runner* moves from the exit towards $\overline{X_i}$, he has enough time to delete the remaining two vertices between $\overline{X_i}$ and the goal to isolate the goal completely. So 'misbehaving' is definitely of no advantage for the *Runner*. We leave it to reader to check the details.

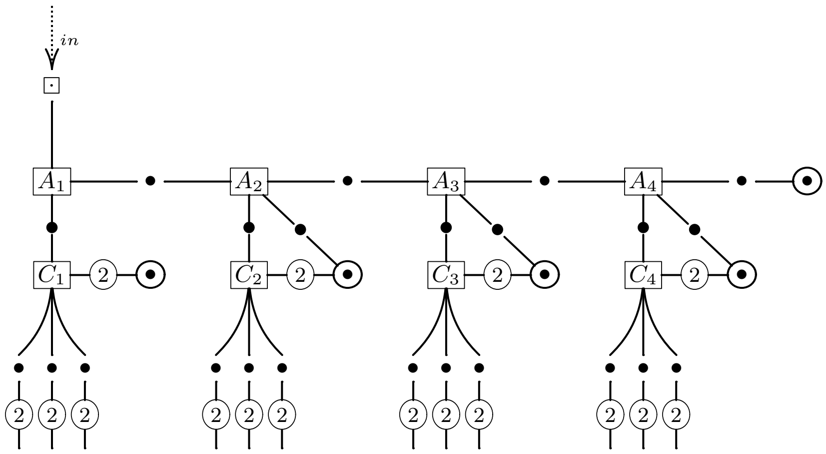


Fig. 10. The verification gadget with four clauses

We finish the construction by the modification of the verification gadget which is displayed in Fig. 10, again for the case that φ consists of four clauses where each clause has exactly three literals. If the *Blocker* wants to check clause c_1 he removes the vertex between A_1 and A_2 in the first step. In the next two steps he has to remove the two vertices between C_1 and the goal. The continuation is analogous to the previous game: if the 'right' assignment has been chosen then there is a literal in c_1 such that there are five connections left between the corresponding literal vertex and the goal – provided that the *Runner* has acted reasonable. Since the *Blocker* cannot block the the way towards this vertex and on this way he can only delete three vertices somewhere in the graph the *Runner* wins the game. On the other hand, if the 'wrong' assignment has been chosen then there are no connections left from each literal vertex corresponding to the literals in c_1.

If the *Blocker* wants to check clause c_k for $k > 1$ he leads the *Runner* along the horizontal way. First, he removes the vertex between A_1 and C_1. In the two steps where the *Runner* moves towards A_2 he removes the two vertices between A_2 and C_2, resp. A_2 and the goal. This repeats until the *Runner* moves from A_{k-1} towards A_k. During the two steps the *Blocker* now removes the vertices between the next A_{k+1}, resp. the goal if c_k is the last clause, and between A_k and the goal. Now the *Runner* has to turn down and the following movements are as in the C_1-case.

Again we have to consider several special cases: first, if the *Runner* reaches A_k and the *Blocker* does not remove the vertex between A_k and C_k but another vertex beside or below the vertex C_k, then the *Runner* moves towards A_{k+1}, resp. the goal. If the area around C_k is 'complete' then he moves directly towards C_k and continues his play as above: since the *Blocker* has gained only on additional move by this there will be at least one connection from the appropriate literal vertex to the goal left (this is the reason why we shifted the multiplicity from four to five). Second, if the *Runner* at any time uses a shortcut by moving from a literal vertex to the bottom of the verification gadget the *Blocker* can prevent the continuation by removing the appropriate vertex beneath C_k. We leave it to the reader to check the details.

To get rid of the 'unremovable' vertices one has to replace each of them by five copies of a simple vertex such that in- and outgoing edges are preserved. The construction is similar to the one for $\bullet$ — k — $\bullet$. Two successive vertices of this kind are replaced by the complete bipartite graph $K_{5,5}$ as depicted in Fig. 11; for more than two successive vertices the construction is analogous. The initial vertex, i.e. the first vertex in the first $\exists$-gadget where the game starts is left as a single vertex. It is easy to see that a 'multiplicity' of five is sufficient to preserve all opportunities for the *Runner*. Since the number of vertices is increased at most fivefold and the number of edges at most 25-fold the size of the game arena is still polynomial in the size of φ.

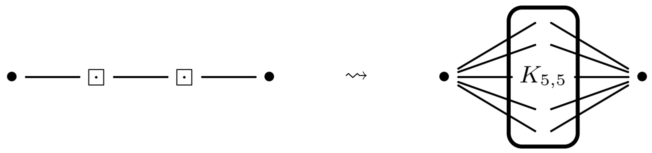


Fig. 11. The replacement for two unremovable vertices

Variant 3

To capture the extension where the *Blocker* is allowed to remove up to n vertices at once one has to replace each vertex in the previous game – except the first one and the goal – by n copies of it and connect them analogously to the construction above where we replaced the unremovable vertices. This results in a game arena with $O(n|V|)$ vertices and $O(n^2|E|)$ edges, where (V, E) is the original game arena.

Directed graphs

Again, since the game played on the constructed arenas for the three variants has an intended direction the cases for directed graphs are easy to capture by replacing each original edge by an directed edges with the intended direction, i.e. from top to bottom and from the bottom of the verification gadget back to the corresponding literal vertices.

To summarise the results for the extended games we have:

Theorem 3. *Solving the variants* 1, 2 *and* 3 *of the sabotage game on directed or undirected graphs with the reachability winning condition is PSPACE-hard.*

6 The remaining winning conditions

In this section we give polynomial time reductions from sabotage games with reachability condition (where the *Blocker* removes exactly one edge per round) to the ones with Hamilton path or complete search condition. We only consider the games on undirected graphs. Each reduction works for the directed case as well by simply replacing each edge added in the construction by a forth and a back edge. These reductions are not optimal but it is easy to modify them to omit some useless edges.

We do not consider Variants 2 or 3 of the game where the *Blocker* removes vertices since the Hamilton path or the complete search condition does not make sense for these conditions. Variant 1 of the game is already captured by Lemma 7.

In the sequel let G be a 'single edge deleting' sabotage game on an undirected graph with the reachability condition.

6.1 Complete search condition

We have to give a sabotage game G' with complete search condition such that the *Runner* wins G iff he wins G'. To obtain G' we add several vertices to G: one 'power vertex' P together with $(|V|-2)\cdot(\max\{|V|,|E|\}-1)$ additional vertices, see Fig. 12 (the additional vertices are displayed outside the frame). The original goal vertex in G is connected to P by an edge with multiplicity $|V|$, whereas all other vertices in G – except the goal and the initial vertex – are connected to P by a chain of $\max\{|V|,|E|\}$ new vertices which are linked among each other by 'unremovable' edges.

If the *Runner* can reach the goal in the original game G then by Lemma 3 he can do so within at most $|V|-1$ steps. In this case there is at least one edge from the goal to the power vertex P left which he uses to reach P. Afterwards he visits one by one all of the remaining vertices in G which have not been visited yet, using the chains in both directions. Thus the *Runner* is able to visit all vertices in G' and therefore he wins the game G'.

For the converse assume that the *Runner* cannot reach the goal in G. Then the only way for him to visit the goal in G' is to use a shortcut from some vertex in G to the power vertex P by moving along one of the chains. But on the way towards P the *Blocker* can cut all links from P to the goal. On *Runner's* way back from P he can remove all edges in the original game G isolating the goal completely. Thus the goal cannot be visited in G' as well and the *Runner* looses G'.

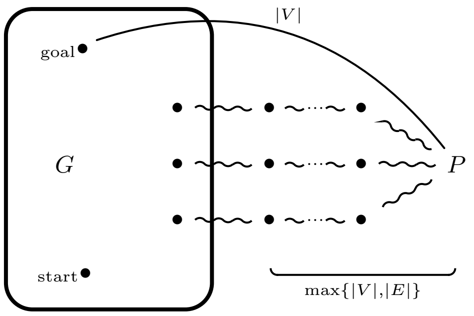


Fig. 12. Game arena G' for the reduction to complete search

To determine an adequate multiplicity of the curly edges notice that the *Runner* needs at most $|V|$ steps to reach P (provided he can visit the goal at all). In the worst case it remains to visit $|V| - 2$ vertices in G (all vertices except the initial vertex and the goal). Each visitation takes $2 \cdot \max\{|V|, |E|\}$ for traversing the chain in both directions. So a rough estimate gives us a sufficient multiplicity of $2 \cdot |V| \cdot \max\{|V|, |E|\}$.

So we have

Theorem 4. *There is a polynomial time reduction from sabotage games with reachability condition to sabotage games with complete search condition. In particular solving the latter games is PSPACE-hard.*

6.2 Hamilton path condition

The reduction to the Hamilton path condition is quite similar to the last one. But we have to prevent repeated visitations of P, so we need to 'unravel' the loops. Let $m := |V| - 2$ and let $v_1, \ldots, v_m$ be an enumeration of all vertices in G except the initial vertex and the goal. We add a sequence $P_1, \ldots, P_m$ of new vertices to G' together with several chains of new vertices such that each has length $\max\{|V|, |E|\}$. We add these chains from P_i as well as from P_{i+1} to vertex v_i for $i < m$ and one chain from P_m to vertex v_m. Furthermore we add for $i < m$ shortcuts from the last vertices in the chains between P_i and v_i to the last vertices in the chains between P_{i+1} and v_i to give the *Runner* the possibility to skip the visitation of v_i. Additionally there is one link with multiplicity $|V|$ from P_1 to the goal in G, see Fig. 13.

Again we have: if the *Runner* can reach the goal in G then within at most $|V| - 1$ steps without visiting any vertex twice. So there is at least one link to P_1 which he uses to reach P_1. He follows the chain to v_1. If he had already visited v_1 on his way to the goal he uses the shortcut at the last vertex in the chain, otherwise he visits v_1. Afterwards he moves to P_2 using the next chain. Continuing like this he reaches P_m and moves towards the last vertex v_m. If he had already visited v_m he just stops one vertex before. Otherwise he stops at v_m. Moving this way he visits each vertex of G' exactly once and wins the game.

For the converse: if the *Runner* cannot reach the goal in G then he cannot do so in G' as well. If he tries to use a shortcut via some P_i the *Blocker* has enough time on the way to P_i to cut all the links between the goal and P_1. On *Runner's*

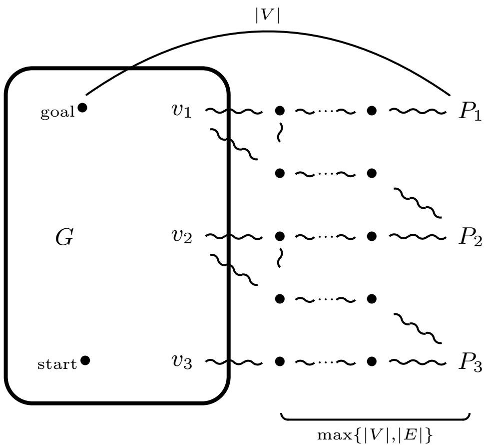


Fig. 13. Game arena G' for the reduction to Hamilton path

way back from some P_j to a vertex in G he is able to remove all edges in the original game G to isolate the goal completely. Thus the *Runner* looses G'.

The estimation of the multiplicity for the 'unremovable' edges is the same as in the previous construction.

Hence we obtain

Theorem 5. *There is a polynomial time reduction from sabotage games with reachability condition to sabotage games with Hamilton path condition. In particular solving the latter games is PSPACE-hard.*

7 A sabotage modal logic

In [vB02] van Benthem considered a 'sabotage modal logic',i.e., a modal logic over *changing models* to express tasks corresponding to sabotage games. He introduced a cross-model modality referring to submodels from which objects have been removed. In this section we will give a formal definition of a sabotage modal logic with a 'transition-deleting' modality and we will show how to apply the results of the previous sections to determine the complexity of uniform model checking for this logic.

We only treat the 'transition-deleting' logic here. We omit the corresponding 'state-deleting' logic, since – to apply Theorem 3– one has to introduce additional semantic requirements to capture the extra game rules (no deletion of the current position, no deletion of the goal), which seems to be very artificial in the logic context.

To realise the use of multi-graphs we will interpret the logic over edge-labelled transition systems. By applying Lemma 2 the complexity results for the reachability game can be obtained for multi-graphs with a uniformly bounded multiplicity. Hence we can do with a finite alphabet Σ. Let Prop $= \{p, p', p'', \ldots\}$ be a set of unary predicate symbols. A (finite) transition system $\mathcal{T}$ is a tuple $(S, \{R_a \mid a \in \Sigma\}, L)$ with binary relations $R_a \subseteq S \times S$ for each $a \in \Sigma$ and a labelling function $L : S \to 2^{\text{Prop}}$.

Definition 1. *Let* $p \in$ Prop *and* $a \in \Sigma$. *Formulae of the* relational sabotage modal logic SML *over transition systems are inductively defined by the grammar*

$$\varphi ::= p \mid \neg\varphi \mid \varphi \vee \varphi \mid \Diamond_a\varphi \mid \diamondminus_a\varphi$$

The dual modalities are denoted by $\Box_a$*, resp.* $\boxminus_a$*, and are defined by*

$$\Box_a\varphi := \neg\Diamond_a\neg\varphi \quad \text{, resp.,} \quad \boxminus_a\varphi := \neg\diamondminus_a\neg\varphi.$$

Further we define

$$\Diamond\varphi := \bigvee_{a\in\Sigma} \Diamond_a\varphi \quad \text{and} \quad \Box\varphi := \bigwedge_{a\in\Sigma} \Box_a\varphi.$$

The formulae $\diamondminus\varphi$ *and* $\boxminus\varphi$ *are defined analogously.*

Let $\mathcal{T} = (S, \{R_a \mid a \in \Sigma\}, L)$ be a transition system. For $t, t' \in S$ and $a \in \Sigma$ we define the submodel $\mathcal{T}^a_{(t,t')} := (S, \{R_b \mid b \in \Sigma \setminus \{a\}\} \cup R_a \setminus \{(t,t')\}, L)$. Note that $\mathcal{T}^a_{(t,t')} = \mathcal{T}$ iff $(t,t') \notin R_a$. For a given state $s \in S$ we define the semantics of SML inductively by

$(\mathcal{T}, s) \models p$ iff $p \in L(s)$
$(\mathcal{T}, s) \models \neg\varphi$ iff not $(\mathcal{T}, s) \models \varphi$
$(\mathcal{T}, s) \models \varphi \vee \psi$ iff $(\mathcal{T}, s) \models \varphi$ or $(\mathcal{T}, s) \models \psi$
$(\mathcal{T}, s) \models \Diamond_a\varphi$ iff there is $s' \in S$ with $(s,s') \in R_a$ such that $(\mathcal{T}, s') \models \varphi$
$(\mathcal{T}, s) \models \diamondminus_a\varphi$ iff there are $t, t' \in S$ such that $(\mathcal{T}^a_{(t,t')}, s) \models \varphi$

For a transition system $\mathcal{T}$ let $\hat{\mathcal{T}}$ be the corresponding first order structure with an unary relation P for each $p \in \text{Prop}$ such that $P(s)$ iff $p \in L(s)$ for each $s \in S$. Similar to the usual modal logic one can translate the logic SML into first order logic. Since FO-model checking is in PSPACE we obtain:

Theorem 6. *For every* SML*-formula* φ *there is an effectively constructible* FO*-formula* $\hat{\varphi}(x)$ *such that for every transition system* $\mathcal{M}$ *and state* s *of* $\mathcal{M}$ *one has:*

$$(\mathcal{T}, s) \models_{\text{SML}} \varphi \iff \hat{\mathcal{T}} \models_{\text{FO}} \hat{\varphi}[s].$$

The size of $\hat{\varphi}(x)$ *is polynomial in the size of* φ*. In particular, SML-model checking is in PSPACE.*

Proof. By induction on the structure of φ.

- $\varphi = p$. Set $\hat{\varphi}(x) := P(x)$.
- $\varphi = \neg\psi$. Set $\hat{\varphi}(x) := \neg\hat{\psi}(x)$. Similarly for $\varphi = \psi_1 \vee \psi_2$.
- $\varphi = \Diamond_a\psi$. Choose a fresh variable name x' which does not occur in $\hat{\psi}(x)$ and let
$$\hat{\varphi}(x) := \exists x' : (x, x') \in R_a \wedge \hat{\psi}(x').$$
- $\varphi = \diamondminus_a\psi$. Choose fresh variable names y, y' which do not occur in $\hat{\psi}(x)$ and let
$$\hat{\varphi}(x) := \exists y, y' : (y, y') \in R_a \wedge \text{Sub}^a_{(y,y')}(\hat{\psi}(x)),$$
where $\text{Sub}^a_{(y,y')}(\chi)$ is the FO-formula where each occurrence of the subformula $(v, v') \in R_a$ within the FO-formula χ is substituted by the formula $(v, v') \in R_a \wedge \neg(v = y \wedge v' = y')$.

It is easy to see that the size of $\hat{\varphi}(x)$ is polynomial in the size of φ. $\square$

We can express the winning of the *Runner* in the sabotage game G on directed graphs with the reachability condition by a SML-formula. For that we consider the game arena as a transition system $\mathcal{T}(G)$ such that the multiplicity of edges is captured by the edge labelling and such that the goal vertex of the game is viewed as the only state with predicate p. We inductively define the SML-formula γ_n by

$$\gamma_0 := p \quad \text{and} \quad \gamma_{i+1} := (\Diamond\boxminus\gamma_i) \vee p.$$

Theorem 7. *The* Runner *has a winning strategy from vertex s in the sabotage game G iff $(\mathcal{T}(G), s) \models \gamma_n$ where n is the number of edges of the game arena (counted with multiplicity).*

Proof. The proof goes by induction on the number n of edges in the game arena G. For $n = 0$ it is obvious that $(\mathcal{T}(G), s) \models \gamma_n$ iff the predicate p is true in s. By the definition of $\mathcal{T}(G)$ this is the case iff s is the goal vertex in the game arena G, which is equivalent to *Runner* having a winning strategy from the vertex s in a sabotage game without edges.

For the induction step first note that $(\mathcal{T}(G), s) \models \gamma_n$ for each n if s is the goal vertex in the game G. Assume that G has $n > 0$ edges and that *Runner* has a winning strategy in G starting from s. If s is the goal vertex, then $(\mathcal{T}(G), s) \models \gamma_n$ according to the above remark. Otherwise, there is a vertex s' that *Runner* moves to in his first move according to the winning strategy. Let G' be the game arena after *Blocker* removed some edge. Then *Runner* has a winning strategy from s' in G' and G' contains $n - 1$ edges. By induction we get that $(\mathcal{T}(G'), s') \models \gamma_{n-1}$. Since the edge that was removed from G to obtain G' was chosen arbitrarily, we get that $(\mathcal{T}(G), s') \models \boxminus\gamma_{n-1}$ and hence $(\mathcal{T}(G), s) \models \Diamond\boxminus\gamma_{n-1}$ since there is an edge between s and s' in G. This obviously implies that $(\mathcal{T}(G), s) \models \gamma_n$.

Conversely, assume that $(\mathcal{T}(G), s) \models \gamma_n$. If $(\mathcal{T}(G), s) \models p$, then s is the goal vertex and *Runner* wins immediately. Otherwise, there exists a vertex s' such that $(\mathcal{T}(G), s') \models \boxminus\gamma_{n-1}$. This means that $(\mathcal{T}(G), s') \models \gamma_{n-1}$ for all G' obtained from G by removing one edge. By induction we get that *Runner* has a winning strategy from s' in all these game arenas G'. Therefore, a winning strategy for *Runner* from s in G is to move to s' and then apply the corresponding winning strategy in the game arena G' after the removal of an edge by *Blocker*. $\square$

Remark 2. To express the winning of the *Runner* it is not enough to consider the formula $p \vee \bigvee_{i=0}^{n}(\Diamond\boxminus)^i\Diamond p$: there are transition systems where none of the disjuncts is satisfied but the *Runner* still wins the game. For example consider the transition system displayed in Fig. 14: neither $\Diamond\boxminus\Diamond p$ nor $\Diamond\boxminus\Diamond\boxminus\Diamond p$ is satisfied in s_0.

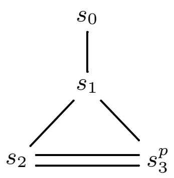


Fig. 14. A counter example

Combining the last theorem and Theorem 2 we obtain:

Theorem 8. *Model checking for the sabotage logic SML is PSPACE-complete.*

References

[BW70] John Bruno and Louis Weinberg. A constructive graph-theoretic solution of the Shannon switching game. *IEEE Transactions on Circuit Theory CT-17*, 1:74–81, 1970.

[Dem01] Erik D. Demaine. Playing games with algorithms: Algorithmic combinatorial game theory. In *Mathematical Foundations of Computer Science 2001, 26th International Symposium, MFCS 2001 Marianske Lazne, Czech Republic, August 27-31, 2001*, volume 2136 of *Lecture Notes in Computer Science*, pages 18–32. Springer, 2001.

[ET76] Shimon Even and Robert Endre Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the Association for Computing Machinery*, 23(4):710–719, 1976.

[Pap94] Christos H. Papadimitriou. *Computational Complexity*. Addison–Wesley, 1994.

[vB02] Johan van Benthem. An essay on sabotage and obstruction. In D. Hutter and S. Werner, editors, *Festschrift in Honour of Prof. Joerg Siekmann*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 2002.

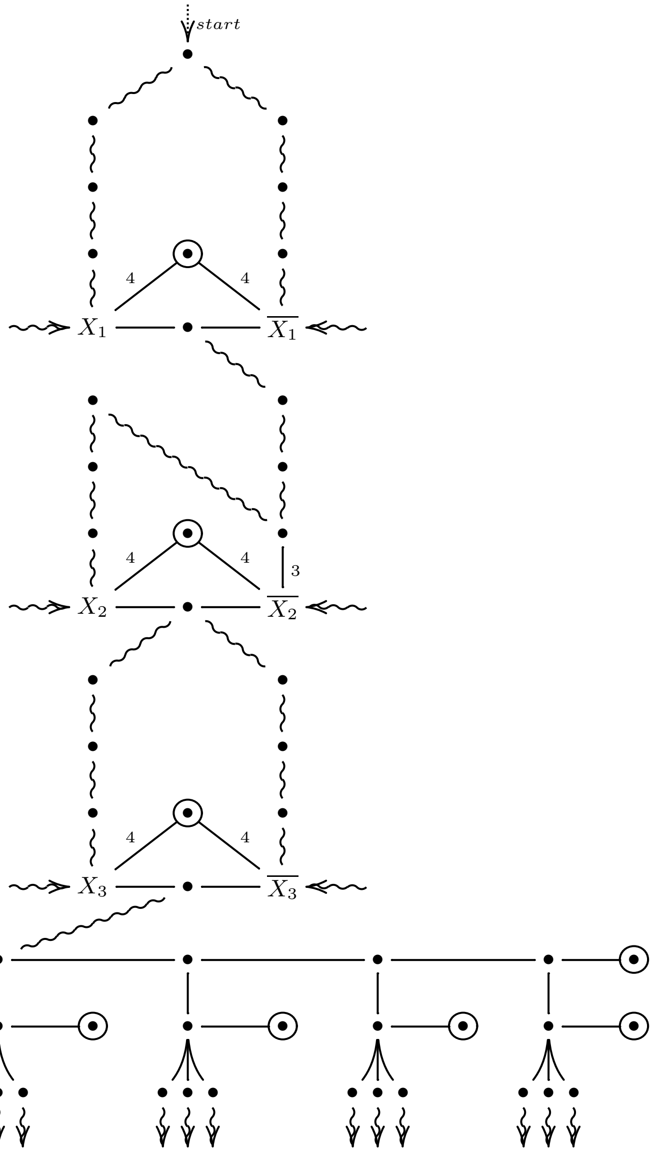


Fig. 15. The arena for $\exists x_1 \forall x_2 \exists x_3 \; (c_1 \wedge c_2 \wedge c_3 \wedge c_4)$

Aachener Informatik-Berichte

This is a list of recent technical reports. To obtain copies of technical reports please consult http://aib.informatik.rwth-aachen.de/ or send your request to: Informatik-Bibliothek, RWTH Aachen, Ahornstr. 55, 52056 Aachen, Email: `biblio@informatik.rwth-aachen.de`

95-11 *	M. Staudt / K. von Thadden: Subsumption Checking in Knowledge Bases
95-12 *	G.V. Zemanek / H.W. Nissen / H. Hubert / M. Jarke: Requirements Analysis from Multiple Perspectives: Experiences with Conceptual Modeling Technology
95-13 *	M. Staudt / M. Jarke: Incremental Maintenance of Externally Materialized Views
95-14 *	P. Peters / P. Szczurko / M. Jeusfeld: Business Process Oriented Information Management: Conceptual Models at Work
95-15 *	S. Rams / M. Jarke: Proceedings of the Fifth Annual Workshop on Information Technologies & Systems
95-16 *	W. Hans / St. Winkler / F. Sáenz: Distributed Execution in Functional Logic Programming
96-1 *	Jahresbericht 1995
96-2	M. Hanus / Chr. Prehofer: Higher-Order Narrowing with Definitional Trees
96-3 *	W. Scheufele / G. Moerkotte: Optimal Ordering of Selections and Joins in Acyclic Queries with Expensive Predicates
96-4	K. Pohl: PRO-ART: Enabling Requirements Pre-Traceability
96-5	K. Pohl: Requirements Engineering: An Overview
96-6 *	M. Jarke / W. Marquardt: Design and Evaluation of Computer–Aided Process Modelling Tools
96-7	O. Chitil: The ς-Semantics: A Comprehensive Semantics for Functional Programs
96-8 *	S. Sripada: On Entropy and the Limitations of the Second Law of Thermodynamics
96-9	M. Hanus (Ed.): Proceedings of the Poster Session of ALP'96 — Fifth International Conference on Algebraic and Logic Programming
96-10	R. Conradi / B. Westfechtel: Version Models for Software Configuration Management
96-11 *	C. Weise / D. Lenzkes: A Fast Decision Algorithm for Timed Refinement
96-12 *	R. Dömges / K. Pohl / M. Jarke / B. Lohmann / W. Marquardt: PRO-ART/CE* — An Environment for Managing the Evolution of Chemical Process Simulation Models
96-13 *	K. Pohl / R. Klamma / K. Weidenhaupt / R. Dömges / P. Haumer / M. Jarke: A Framework for Process-Integrated Tools
96-14 *	R. Gallersdörfer / K. Klabunde / A. Stolz / M. Eßmajor: INDIA — Intelligent Networks as a Data Intensive Application, Final Project Report, June 1996
96-15 *	H. Schimpe / M. Staudt: VAREX: An Environment for Validating and Refining Rule Bases

96-16 * M. Jarke / M. Gebhardt, S. Jacobs, H. Nissen: Conflict Analysis Across Heterogeneous Viewpoints: Formalization and Visualization
96-17 M. Jeusfeld / T. X. Bui: Decision Support Components on the Internet
96-18 M. Jeusfeld / M. Papazoglou: Information Brokering: Design, Search and Transformation
96-19 * P. Peters / M. Jarke: Simulating the impact of information flows in networked organizations
96-20 M. Jarke / P. Peters / M. Jeusfeld: Model-driven planning and design of cooperative information systems
96-21 * G. de Michelis / E. Dubois / M. Jarke / F. Matthes / J. Mylopoulos / K. Pohl / J. Schmidt / C. Woo / E. Yu: Cooperative information systems: a manifesto
96-22 * S. Jacobs / M. Gebhardt, S. Kethers, W. Rzasa: Filling HTML forms simultaneously: CoWeb architecture and functionality
96-23 * M. Gebhardt / S. Jacobs: Conflict Management in Design
97-01 Jahresbericht 1996
97-02 J. Faassen: Using full parallel Boltzmann Machines for Optimization
97-03 A. Winter / A. Schürr: Modules and Updatable Graph Views for PROgrammed Graph REwriting Systems
97-04 M. Mohnen / S. Tobies: Implementing Context Patterns in the Glasgow Haskell Compiler
97-05 * S. Gruner: Schemakorrespondenzaxiome unterstützen die paargrammatische Spezifikation inkrementeller Integrationswerkzeuge
97-06 M. Nicola / M. Jarke: Design and Evaluation of Wireless Health Care Information Systems in Developing Countries
97-07 P. Hofstedt: Taskparallele Skelette für irregulär strukturierte Probleme in deklarativen Sprachen
97-08 D. Blostein / A. Schürr: Computing with Graphs and Graph Rewriting
97-09 C.-A. Krapp / B. Westfechtel: Feedback Handling in Dynamic Task Nets
97-10 M. Nicola / M. Jarke: Integrating Replication and Communication in Performance Models of Distributed Databases
97-13 M. Mohnen: Optimising the Memory Management of Higher-Order Functional Programs
97-14 R. Baumann: Client/Server Distribution in a Structure-Oriented Database Management System
97-15 G. H. Botorog: High-Level Parallel Programming and the Efficient Implementation of Numerical Algorithms
98-01 * Jahresbericht 1997
98-02 S. Gruner/ M. Nagel / A. Schürr: Fine-grained and Structure-oriented Integration Tools are Needed for Product Development Processes
98-03 S. Gruner: Einige Anmerkungen zur graphgrammatischen Spezifikation von Integrationswerkzeugen nach Westfechtel, Janning, Lefering und Schürr
98-04 * O. Kubitz: Mobile Robots in Dynamic Environments
98-05 M. Leucker / St. Tobies: Truth — A Verification Platform for Distributed Systems

98-07 M. Arnold / M. Erdmann / M. Glinz / P. Haumer / R. Knoll / B. Paech / K. Pohl / J. Ryser / R. Studer / K. Weidenhaupt: Survey on the Scenario Use in Twelve Selected Industrial Projects

98-08 * H. Aust: Sprachverstehen und Dialogmodellierung in natürlichsprachlichen Informationssystemen

98-09 * Th. Lehmann: Geometrische Ausrichtung medizinischer Bilder am Beispiel intraoraler Radiographien

98-10 * M. Nicola / M. Jarke: Performance Modeling of Distributed and Replicated Databases

98-11 * A. Schleicher / B. Westfechtel / D. Jäger: Modeling Dynamic Software Processes in UML

98-12 * W. Appelt / M. Jarke: Interoperable Tools for Cooperation Support using the World Wide Web

98-13 K. Indermark: Semantik rekursiver Funktionsdefinitionen mit Striktheitsinformation

99-01 * Jahresbericht 1998

99-02 * F. Huch: Verifcation of Erlang Programs using Abstract Interpretation and Model Checking — Extended Version

99-03 * R. Gallersdörfer / M. Jarke / M. Nicola: The ADR Replication Manager

99-04 M. Alpuente / M. Hanus / S. Lucas / G. Vidal: Specialization of Functional Logic Programs Based on Needed Narrowing

99-07 Th. Wilke: CTL+ is exponentially more succinct than CTL

99-08 O. Matz: Dot-Depth and Monadic Quantifier Alternation over Pictures

2000-01 * Jahresbericht 1999

2000-02 Jens Vöge / Marcin Jurdzinski: A Discrete Strategy Improvement Algorithm for Solving Parity Games

2000-04 Andreas Becks / Stefan Sklorz / Matthias Jarke: Exploring the Semantic Structure of Technical Document Collections: A Cooperative Systems Approach

2000-05 Mareike Schoop: Cooperative Document Management

2000-06 Mareike Schoop / Christoph Quix (eds.): Proceedings of the Fifth International Workshop on the Language-Action Perspective on Communication Modelling

2000-07 * Markus Mohnen / Pieter Koopman (Eds.): Proceedings of the 12th International Workshop of Functional Languages

2000-08 Thomas Arts / Thomas Noll: Verifying Generic Erlang Client-Server Implementations

2001-01 * Jahresbericht 2000

2001-02 Benedikt Bollig / Martin Leucker: Deciding LTL over Mazurkiewicz Traces

2001-03 Thierry Cachat: The power of one-letter rational languages

2001-04 Benedikt Bollig / Martin Leucker / Michael Weber: Local Parallel Model Checking for the Alternation Free mu-Calculus

2001-05 Benedikt Bollig / Martin Leucker / Thomas Noll: Regular MSC Languages

2001-06 Achim Blumensath: Prefix-Recognisable Graphs and Monadic Second-Order Logic

2001-07 Martin Grohe / Stefan Wöhrle: An Existential Locality Theorem

2001-08 Mareike Schoop / James Taylor (eds.): Proceedings of the Sixth International Workshop on the Language-Action Perspective on Communication Modelling

2001-09 Thomas Arts / Jürgen Giesl: A collection of examples for termination of term rewriting using dependency pairs

2001-10 Achim Blumensath: Axiomatising Tree-interpretable Structures

2001-11 Klaus Indermark / Thomas Noll (eds.): Kolloquium Programmiersprachen und Grundlagen der Programmierung

2002-01 * Jahresbericht 2001

2002-02 Jürgen Giesl / Aart Middeldorp: Transformation Techniques for Context-Sensitive Rewrite Systems

2002-03 Benedikt Bollig / Martin Leucker / Thomas Noll: Generalised Regular MSC Languages

2002-04 Jürgen Giesl / Aart Middeldorp: Innermost Termination of Context-Sensitive Rewriting

2002-05 Horst Lichter / Thomas von der Maßen / Thomas Weiler: Modelling Requirements and Architectures for Software Product Lines

2002-06 Henry N. Adorna: 3-Party Message Complexity is Better than 2-Party Ones for Proving Lower Bounds on the Size of Minimal Nondeterministic Finite Automata

2002-07 Jörg Dahmen: Invariant Image Object Recognition using Gaussian Mixture Densities

2002-08 Markus Mohnen: An Open Framework for Data-Flow Analysis in Java

2002-09 Markus Mohnen: Interfaces with Default Implementations in Java

2002-10 Martin Leucker: Logics for Mazurkiewicz traces

2002-11 Jürgen Giesl / Hans Zantema: Liveness in Rewriting

2003-01 * Jahresbericht 2002

2003-02 Jürgen Giesl / René Thiemann: Size-Change Termination for Term Rewriting

2003-03 Jürgen Giesl / Deepak Kapur: Deciding Inductive Validity of Equations

2003-04 Jürgen Giesl / René Thiemann / Peter Schneider-Kamp / Stephan Falke: Automated Termination Proofs with AProVE

* These reports are only available as a printed version.

Please contact `biblio@informatik.rwth-aachen.de` to obtain copies.