Electronic Colloquium on Computational Complexity, Report No. 80 (2018)

Edge Hop: A Framework to Show NP-Hardness of Combinatorial Games

Moritz Gobbert
gobbert@informatik.uni-trier.de
Fachbereich 4 - Abteilung Informatikwissenschaften
Universität Trier, D-54296 Trier, Germany

2017

Abstract

The topic of this paper is a game on graphs called *Edge Hop*. The game's goal is to move a marked token from a specific starting node to a specific target node. Further, there are other tokens on some nodes which can be moved by the player under suitable conditions. In addition, the graph has special properties. For instance: Every node can only hold a fixed amount of tokens and the marked token is only allowed to travel once across each edge. We show that the decision question whether the marked token can reach the goal node is *NP-complete*. For this we construct several gadgets to show a reduction via *Directed Hamiltonian cycles*. These gadgets can further be used as a framework for complexity analysis of combinatorial puzzles and similar questions. As an example we will show the NP-hardness of *Game about Squares* and of *2048*.

Keywords: complexity, combinatorial puzzles, NP, complexity of games, 2048, Game about Squares

ISSN 1433-8092

1 Introduction

Games are frequently examined in complexity theory. In this context, many types of games are considered. For instance combinatorial puzzles like *Rush Hour* [10, 9] or *Numberlink* [1], two-player games like *Checkers* [18] or *Phutball* [8] and video games like *Pokémon* [2] or *Lemmings* [20]. Even "zero-player" games like *Conway's Game of Life* [17] are researched.[1]

Game	*Complexity*	*Literature*
Rush Hour	PSPACE-complete respectively FPT	[10, 9]
Numberlink	NP-complete	[1]
Checkers	EXPTIME-complete	[18]
Phutball	PSPACE-hard	[8]
Pokémon	NP-complete	[2]
Lemmings	PSPACE-complete	[20]
Conway's Game of Life	Turing-complete	[17]

Table 1: Different games and the complexity of their related decision questions.

All these different results relating to the complexity of games or rather their related decision questions (see table 1) provide indication that the particular proofs are executed in different ways. With the idea to unify proofs for some of these problems, Hearn and Demaine developed a framework for hardness proofs – the so-called *Nondeterministic Constraint Logic (NCL)* [12]. Our goal is to develop another framework which can be used for NP-hardness proofs. Our framework will correspond to motion planning problems in a more "natural" way than the NCL.

1.1 Motivation

In some cases complexity analysis is relatively easy (especially if one reduces between "similar" problems) but in many cases it is time-consuming and complex (for example Cook's proof that SAT is NP-complete [6]). The complexity analysis of similar problems is often executed in a similar way which leads to the question if it is possible to simplify these processes by outsourcing the "time-consuming" parts to a *distinct* and *universal* proof. This distinct proof should provide "interfaces" (*gadgets*) which can be used to analyze the complexity of different problems. The construction of these gadgets should ideally be less time-consuming than "manually" analyzing the complexity. In addition, it should be possible to use the same gadgets for a plethora of problems. Hearn and Demain developed the *Nondeterministic Constraint Logic* as a framework to show PSPACE-hardness[2]. Even though there a variations of the NCL to show NP-hardness of problems, we will develop a different framework. Our framework will be an (abstract) one-player game called *Edge Hop*, in which it is the goal to find a path on graphs with special properties. This problem will allow us to provide gadgets which relate to motion planning problems. Thus we offer a

[1] A comprehensive list of NP-complete puzzles was put together by Graham Kendall, Andrew Parkes and Kristian Spoerer [14]. Édouard Bonnet dedicated likewise a chapter in his doctoral thesis to the complexity of games [3]. In addition, Erik Demaine maintains a website dedicated to combinatorial games. [7]

[2] At least in its original version from 2005 [13].

framework to show the NP-hardness of motion planning problems in a "natural" way.

2 Theory

In the course of this chapter we establish the rules of Edge Hop and the properties of the corresponding graph. In addition, we show that the question whether the dedicated token can reach the target node is *NP-complete*, by a reduction via *Directed Hamiltonian Cycles*.

2.1 Edge Hop

Edge Hop is a combinatorial puzzle (for one person). The game is played on an undirected graph $G = (V, E)$. The goal of Edge Hop is to move a specific token to a designated node. It is only allowed to move between adjacent nodes. In other words: The goal is to find a path between two specific nodes.

Definition 2.1 (Active token)**.** The *active token* is a uniquely marked token which the player moves from node to node. The player is only allowed to move the active token from a node u to a node v if $uv \in E$. Such a move is called *active move*. To indicate an active move from u to v we write $u \xrightarrow{a} v$. In addition, we call the node where the active token is located at the beginning of the game the *starting node*.

Definition 2.2 (Passive token)**.** Passive tokens are another kind of token placed on the graph. The player is only allowed to move a passive token from a node u to a node v if $uv \in E$ and the active token is currently located on v. We call such a move *passive move* and we write $v \xleftarrow{p} u$ for a passive move from u to v.

Definition 2.3 (Target node)**.** The target node is a dedicated node $z \in V$. The player wins the game iff the active move $u \xrightarrow{a} z$ (for any $u \in V$) is performed.

Definition 2.4 (Maximum capacity)**.** The maximum capacity $k_v \in \mathbb{N}$ of a node $v \in V$ is the maximal number of tokens (both passive tokens and the active token) which can be on v at the same time. For a node v with k tokens on it the player is only allowed to execute $u \xrightarrow{a} v$ or $v \xleftarrow{p} u$ if $k < k_v$.

Definition 2.5 (Used edges)**.** At any time[3] $t \geq 0$ in the course of the game we define $X_t \subseteq E$ as the set of all edges uv which the player used for an active move $u \xrightarrow{a} v$ prior to time t. We define inductively:

$$X_0 := \emptyset$$

$$X_t := \begin{cases} X_{t-1} \cup \{uv\}, \text{ if the move at time } t \text{ is } u \xrightarrow{a} v \\ X_{t-1}, \text{ otherwise} \end{cases}$$

$$X := \bigcup_{t \geq 0} X_t$$

We call the set of *all* used edges X.

[3] Here *time* corresponds to the number of executed active and passive moves combined.

Definition 2.6 (Valid sequence of moves)**.** A finite sequence of moves is a sequence of any active and passive moves (respecting definitions 2.1, 2.2 and 2.4). A sequence of moves $Z = (Z_1, Z_2, ..., Z_m)$ is valid if the following two properties hold:

(1) For $1 \leq i \leq m$: Z_i is an active move $\Rightarrow$ Z_i doesn't use an edge in X_{i-1}.

(2) The last move in the sequence is $u \xrightarrow{a} z$ for $u \in V$.

We also call single moves in a valid sequence of moves *valid moves*.

Definition 2.7 (Edge Hop graph)**.** An Edge Hop graph (or Edge Hop instance) is a 5-tuple $\mathcal{EH} = (G, k_G, a, z, P_G)$. Here, $G = (V, E)$ is a simple (connected) graph, $k_G : V \to \mathbb{N}$, $k_G(v) = k_v$ a total function which assigns every node its maximum capacity, $a \in V$ the starting node, $z \in V$ the target node and P_G a multiset over V which contains for every passive token the node which holds the token. If it is apparent which graph is meant we write $k(v)$ instead of $k_G(v)$ and P instead of P_G. In addition we define $V(\mathcal{EH}) := V(G)$ and $E(\mathcal{EH}) := E(G)$

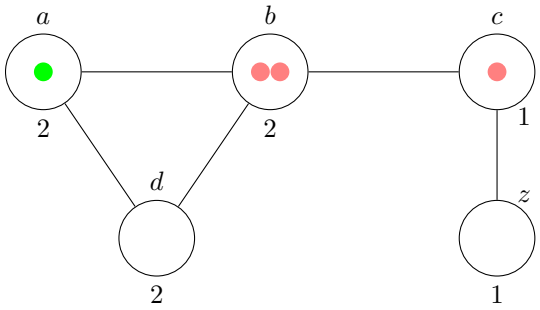


Figure 1: An example graph.

Fig. 1 displays an example graph $\mathcal{EH}$. The set of nodes V is $\{a, b, c, d, z\}$ with z being the target node. The set of edges E is $\{ab, ad, bc, bd, cz\}$. The green dot on node a indicates the active token and the red dots on b and c indicate passive tokens. The numbers below the nodes indicate the corresponding maximal capacities. A valid sequence of moves for this particular graph is $a \xleftarrow{p} b, a \xrightarrow{a} d, d \xleftarrow{p} b, d \xrightarrow{a} b, b \xleftarrow{p} c, b \xrightarrow{a} c, c \xrightarrow{a} z$.

2.2 Complexity

In this section we will show that the decision question whether it is possible to move the active token to the target node is NP-complete.

▶ EdgeHop
Instance: An Edge Hop instance $\mathcal{EH} = (G, k_G, a, z, P_G)$.
Question: Is there a valid sequence of moves from a to z?

The NP-membership is shown straightforward with a guess-and-check algorithm: Guess a sequence of moves and check if it is valid. The important detail is, that a valid sequence of moves cannot be arbitrarily long. The number of active moves is bounded by the number of edges (since every edge can only be used

once) and the number of passive moves (per node) is bounded by the number of passive tokens. This means a valid sequence of moves has at most $|E|+|V|\cdot|P|$ moves. Thus, it is possible to check the sequence of moves in polynomial time.[4]

To show that EdgeHop is NP-hard, we will show a polynomial-time reduction via *Directed Hamiltonian Cycles* (DHC). Since DHC is NP-hard itself (cf. [21]) it will follow that EdgeHop is NP-hard. First we will design gadgets to construct a graph (suited for Edge Hop) from a given directed graph (*digraph*) G. For this, we will design four basic gadgets: An one-way gadget (fig. 3), a fork gadget (fig. 4a), a defork gadget (fig. 4b) and an unlock gadget (fig. 5). Fig. 2 shows the symbols we'll use for these gadgets. In our construction there will only be a valid sequence of moves, iff G has a Hamiltonian cycle.

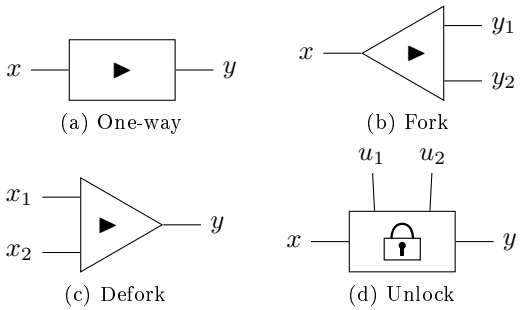


Figure 2: The symbols which we will use to indicate the basic gadgets.

Below we will assume that the active token cannot leave the currently analyzed gadget. This doesn't pose a problem because the gadgets will later be connected in such a way, that there are no ways to leave and enter the gadgets in unintended ways.

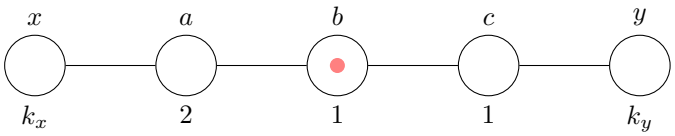


Figure 3: The one-way gadget as a partial Edge Hop graph. The active token cannot traverse the gadget in both directions. The capacities k_x and k_y can be chosen arbitrarily.

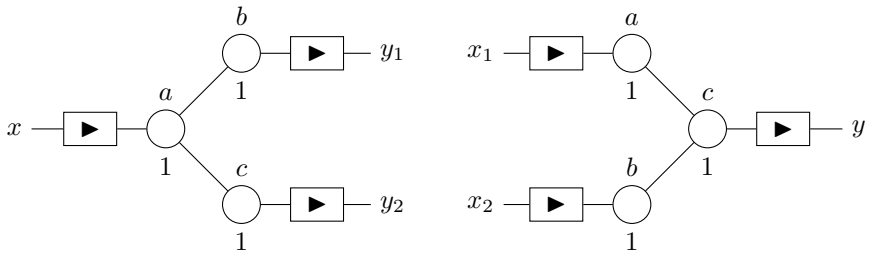
Lemma 2.1 (One-way gadget)**.** *Without leaving the one-way gadget, there is a valid sequence of moves from x to y and no valid sequence of moves from y to x.*

Proof. A valid sequence of moves from x to y is $x \xrightarrow{a} a, a \xleftarrow{p} b, a \xrightarrow{a} b, b \xrightarrow{a} c, c \xrightarrow{a} y$. Therefore the lemma's first part is true.

Now we have to show that there is *no* valid sequence of moves from y to x. First we consider the gadget in its initial state. Since we only consider sequences of moves inside the gadget, we know that the active token has to pass nodes c, b and a. Due to the fact that there is a passive token on node b and $k_b = 1$ it

[4] See [11] for a more detailed analysis.

follows that $c \xrightarrow{a} b$ is not a valid move. Likewise, $c \xleftarrow{p} b, c \xrightarrow{a} b$ is not a valid sequence of moves because $k_c = 1$. Therefore the active token cannot move from c to b in the gadget's initial state. This means also that the active token cannot move from y to x. Now we consider the case that the passive token is not located on b anymore. Since $k_c = 1$, it is not possible that the passive token passed node c. This means it had to move across node a. This on the other hand means that the active token had to visit node a at a previous point in time t and therefore $xa \in X_t$ (and subsequent sets of used edges). In this case $c \xrightarrow{a} b, b \xrightarrow{a} a$ is a valid sequence of moves. However xa is an used edge and hence the active token cannot reach node x. So there is also no valid sequence of moves from y to x in this case and therefore the gadget works as intended. □



(a) The active token can leave this gadget via one of two possible nodes.

(b) The active token has to leave the gadget via a specific node, regardless of which node it used to enter the gadget.

Figure 4: The Fork and defork gadgets as partial Edge Hop graphs.

Lemma 2.2 (Fork gadget)**.** *For every $i, j \in \{1, 2\}$ with $i \neq j$ there is a valid sequence of moves from x to y_i without leaving the gadget. In addition there are* no *valid sequences of moves from y_i to x or from y_i to y_j.*

Proof. Because of lemma 2.1 there is no valid sequence of moves from y_1 to b and from y_2 to c. This means that there is no valid sequence of moves from y_i to x and from y_i to y_j (for $i, j \in \{1, 2\}$ with $i \neq j$).

The same lemma also proves that there are valid sequences of moves from x to a, from b to y_1 and from c to y_2. Therefore it is sufficient to consider the subgraph $U_V = (\{a, b, c\}, \{ab, ac\})$ to prove the first part. The moves $a \xrightarrow{a} b$ and $a \xrightarrow{a} c$ satisfy the postulated properties. □

Lemma 2.3 (Defork gadget)**.** *For every $i, j \in \{1, 2\}$ with $i \neq j$ there is a valid sequence of moves from x_i to y without leaving the gadget. There are also no valid sequences of moves from y to x_i and from x_i to x_j without leaving the gadget.*

Proof. Analogously to lemma 2.2. □

Lemma 2.4 (Unlock gadget)**.** *Without leaving the unlock gadget the following is true for all $v, w \in \{u_1, u_2, x, y\}$ with $v \neq w$: There is a valid sequence of moves from v to w iff $v = u_1$ and $w = u_2$ and there is a valid sequence of moves from v to w iff $v = x$ and $w = y$ and the active token has entered the gadget via u_1 at a previous point in time.*

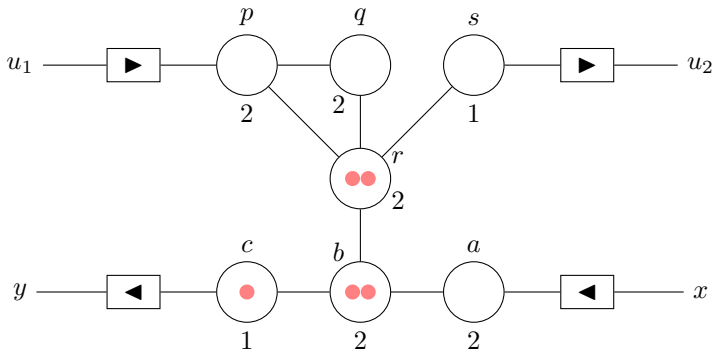


Figure 5: The unlock gadget as partial Edge Hop graph. The active token is only able to move from x to y if it entered the gadget via u_1 at a previous point in time.

Proof. First of all we show that the claimed sequences of moves actually exist. Lemma 2.1 indicates that there are valid sequences of moves from u_1 to p, from s to u_2, from x to a and from c to y. Therefore it is sufficient to consider the subgraph $U_E = (\{a, b, c, p, q, r, s\}, \{ab, bc, br, pq, pr, qr, rs\})$. A valid sequence of moves from p to s would be $p \overset{p}{\leftarrow} r, p \overset{a}{\rightarrow} r, r \overset{a}{\rightarrow} s$.

Every valid sequence of moves from a to c requires that the player moves the passive token off of node c (because $k_c = 1$). It is not possible to move the passive token inside the one-way gadget due to lemma 2.1.[5] Therefore the move $b \overset{p}{\leftarrow} c$ is required. This move is only possible if there are no passive tokens on b because $k_b = 2$. Since $k_a = 2$ the move $a \overset{p}{\leftarrow} b$ can only be executed *once*. The second passive token on node b cannot get moved to a or c. This means that the move $r \overset{p}{\leftarrow} b$ has to be executed at least once to move both passive tokens off of node b. There are two passive tokens on node r as well. To move them off of r, the player has to perform the moves $p \overset{p}{\leftarrow} r$ and $q \overset{p}{\leftarrow} r$ (since $k_p = k_q = 2$ and $k_s = 1$). This means that the active token has to move to node p. It is not possible to move (inside the gadget) from a to p and then to c because edge br would be used more than once.[6]

The only way for the active token to reach node p is by entering the gadget via u_1. This means that there is no valid sequence of moves from a to c unless the active token entered the gadget previously via u_1. Let us assume that the player executed the moves $p \overset{p}{\leftarrow} r$, $p \overset{a}{\rightarrow} q$, $q \overset{p}{\leftarrow} r$, $q \overset{a}{\rightarrow} r$, $r \overset{p}{\leftarrow} b$, $r \overset{a}{\rightarrow} s$ anytime before the active token reached node a. Then the following is a valid sequence of moves from a to c (and therefore from x to y): $a \overset{p}{\leftarrow} b$, $a \overset{a}{\rightarrow} b$, $b \overset{p}{\leftarrow} c$, $b \overset{a}{\rightarrow} c$.

Now we have to show that there are no other valid sequences of moves. Lemma 2.1 proves that there are no valid sequences of moves from u_1 to x or from x to u_1. Likewise there are no valid sequences of moves inside the gadget originating from y or from u_2. The only thing now to show is that there are no valid sequences of moves from u_1 to y or from x to u_2.

[5] Actually it is possible to move the passive token to node a of the one-way gadget, but after this move it wouldn't be possible for the active token to reenter the unlock gadget.

[6] It is also not possible for the active token to move from a to p, then *outside* of the gadget to x and afterwards to c because edge ab would be used more than once.

First we show that there is no valid sequence of moves from u_1 to y. As aforementioned it is sufficient to show that there is no valid sequence of moves from p to c. As discussed before every such sequence of moves has to include $c \overset{p}{\leftarrow} b$. This implies that the player has to execute both $a \overset{p}{\leftarrow} b$ and $r \overset{p}{\leftarrow} b$. Let us assume that the active token it located on node p at time t. We have to consider two distinct cases: Either $a \overset{p}{\leftarrow} b$ was executed at a previous point in time or the move was not yet executed.

Case 1: $a \overset{p}{\leftarrow} b$ was not yet executed. This means that the active token has to move from p to a to execute $a \overset{p}{\leftarrow} b$. Then it needs to move from a to c. This is not possible because every sequence of moves from p to a as well as every sequence of moves from a to c uses edge ab.

Case 2: $a \overset{p}{\leftarrow} b$ was executed at a previous point in time. Let us assume that $ab \notin X_t$. This means that $a \overset{a}{\rightarrow} b$ was not yet executed. Additionally lemma 2.1 shows that it is not possible for the active token to move from a to x. These two restrictions imply that the active token is located on node a. This contradicts our assumption that the active token is on node p. Therefore it has to be true that $ab \in X_t$. This means that the active move $a \overset{a}{\rightarrow} b$ was executed. Because the active token is currently located on node p one of the active moves $b \overset{a}{\rightarrow} c$ or $b \overset{a}{\rightarrow} r$ was executed beforehand. In the case of $bc \in X_t$ there is no valid sequence of moves from p to c because every such sequence of moves includes $b \overset{a}{\rightarrow} c$. In the case of $br \in X_t$ there is also no valid sequence of moves from p to c because every such sequence of moves includes $r \overset{a}{\rightarrow} b$.[7]

In a final step we show that there are no valid sequences of moves from x to u_2. Here, too, is it sufficient to show that there are no valid sequences of moves from a to s. It is easy to see that every valid sequence of moves from a to s requires that at least one of the two passive tokens on r gets moved. Here we consider the two distinct cases: Either $p \overset{p}{\leftarrow} r$ or $b \overset{p}{\leftarrow} r$ was executed.[8]

Case 1: $p \overset{p}{\leftarrow} r$ was executed. Lemma 2.1 and the fact that there is no valid sequence of moves from p to c imply that the active token has to have moved from p to s to exit the gadget (to reach node a at a later time). This means that the move $r \overset{a}{\rightarrow} s$ was executed. When the token reaches node a at a point in time t, there are no more valid sequences of moves from a to s because $rs \in X_t$ and every such sequence includes $r \overset{a}{\rightarrow} s$.

Case 2: $b \overset{p}{\leftarrow} r$ was executed. This is only possible if *both* tokens get moved off of b. However, this is not possible because $k_c = 1$ and $k_a = 2$. □

Now we have all basic gadgets which are required to construct an Edge Hop graph out of a given digraph. Fig. 6 shows schematically how to construct an Edge Hop graph. The rectangles labeled v_1 to v_n denote *node gadgets* which we will assemble out of our basic gadgets. The essential idea is, that the active token has to enter every node gadget *exactly once* to reach node z.

Until now, we used fork and defork gadgets with exactly two in-/outputs. For the following gadgets we will use these two gadgets with an arbitrary number of inputs and outputs. For this we just cascade our old gadgets.

As already mentioned, our task is to substitute every node of a digraph with an (modified) unlock gadget, called *node gadget*. We will connect the outputs of

[7] A sequence of moves from r to u_2, then to x and then to b would also not be valid because it would use ab more than once.

[8] The case $q \overset{p}{\leftarrow} r$ is essentially the same as the case $p \overset{p}{\leftarrow} r$.

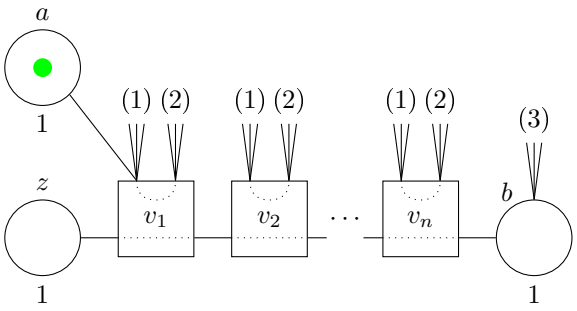


Figure 6: A schematic representation of an Edge Hop graph, constructed out of a digraph G. As usual, a is the starting node and z is the target node. The edges at (1) correspond to the respective node's inward edges and the edges at (2) correspond to the respective node's outward edges. The edges at (3) are connected to the nodes which have an outward edge to v_1.

a node gadget with the inputs of another node gadget if there is an edge leading from the first node to the other node. Fig. 7 shows a node gadget. We call the nodes $x_1, ..., x_k$ and $y_1, ..., y_l$ of a node gadget *unlocking* nodes.

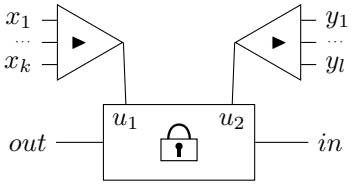


Figure 7: The node gadget. The active token is only able to move from *in* to *out* if it entered the gadget via one of the nodes $x_1, ..., x_k$ at a previous point in time. The nodes $x_1, ..., x_k$ correspond to the inward edges and the nodes $y_1, ..., y_l$ correspond to the outward edges of the respecting nodes of the original graph.

Lemma 2.5 (Node gadget)**.** *Choose $i, i' \in \{1, ..., k\}$ and $j, j' \in \{1, ..., l\}$ (with $i \neq i'$ and $j \neq j'$) arbitrarily. The following properties hold for the node gadget (Fig. 7):*

1. *There is only a valid sequence of moves from in to out if the gadget was entered at a previous point in time via node x_i.*
2. *There is only a valid sequence of moves from x_i to y_j if the gadget was not entered at a previous point in time via node $x_{i'}$.*
3. *There are no valid sequences of moves from y_i to x_j, from x_i to $x_{i'}$ or from y_j to $y_{j'}$.*
4. *There are no valid sequences of moves from x_i to out or from in to y_j.*

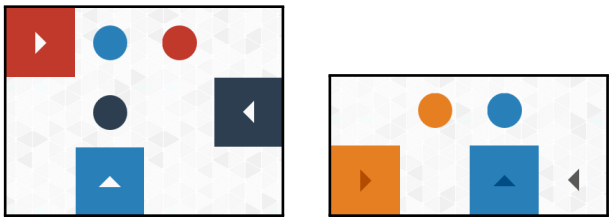
Proof. Properties 1 and 4 follow from lemma 2.4, property 3 follows from lemma 2.2 and lemma 2.3. Property 2 is also a result of lemma 2.4 and the fact that no edge in an Edge Hop graph can be used more than once. □

Now we have designed the node gadget and therefore we can devise an Edge Hop graph out of any digraph G and since DHC is NP-hard it follows that EdgeHop is NP-hard.[9] Combined with the previous mentioned NP-membership we have:

Theorem 2.6. EdgeHop *is NP-complete.*

3 Examples

3.1 Game about Squares



(a) Level 2 (b) Level 9

Figure 8: Two screenshots taken from the browser version of Game about Squares. See fig. 19 in the appendix for an example solution for level 9.

Game about Squares is a sliding block puzzle developed by Andrey Shevchuk in 2014 [19]. The game is played on an unbounded grid and the goal is to push colored squares to specific spaces (which are indicated by circles of the same color). Each square can only be pushed in a specific direction (indicated by a white triangle on the square) which is assigned at the start of the game. There are spaces on the grid (indicated by black triangles) which can change the assigned direction of a square by pushing it on that space. In contrast to e. g. *Sokoban*, squares can push other squares. In addition there is no "player character" on the grid – the player moves squares by clicking on them (with the mouse cursor). Every click moves the square one space in the assigned direction.

We will now show the NP-hardness of Gas via EdgeHop which presents an alternative to Jens Maßberg's reduction via Sat [15].

We define an instance of *Game about Squares* as 5-tuple $\mathcal{G} = (C, r, p, z, d)$ with:

- $C \subset \mathbb{N}$ is a finite set of colors (which correspond to the squares)
- $r : C \to \{\triangle, \triangleright, \triangledown, \triangleleft\}$ is a function which assigns each square an initial direction
- $p : C \to \mathbb{Z}^2$ is a function which assigns each square its initial position
- $z : C \to \mathbb{Z}^2$ is a partial function which assigns some squares their corresponding target positions
- $d : \{\blacktriangle, \blacktriangleright, \blacktriangledown, \blacktriangleleft\} \to \mathcal{P}(\mathbb{Z}^2)$ is a function which assigns each direction-changing triangle a set of positions

[9] Again: See [11] for a more detailed analysis.

In addition, the following properties should apply:

- p is injective (i. e. squares have pairwise distinct positions)
- z is injective (i. e. squares have pairwise distinct target positions)
- $\forall x, y \in \{\blacktriangle, \blacktriangleright, \blacktriangledown, \blacktriangleleft\}: x \neq y \Rightarrow d(x) \cap d(y) = \emptyset$ (i. e. no space holds two different black triangles)
- $\forall c \in C, x \in \{\blacktriangle, \blacktriangleright, \blacktriangledown, \blacktriangleleft\}: p(c) \in d(x) \Rightarrow r(c) = x$ (i. e. if the initial position of a square is the position of a black triangle then the initial direction of that square has to match the triangle)

▶ Game about Squares (GaS)
Instance: A *Game about Squares* instance $\mathcal{G} = (C, r, p, z, d)$.
Question: Is there a sequence of moves so that every target space is occupied by the corresponding square?

Since every square in the game has a well-defined direction and since all squares is are distinguishable from each other, we will write sequences of moves as a sequences of colors. These sequences correspond to the order in which the player clicks on the respective squares. Additionally we will write (c^n) for a (sub-)sequence which moves the c-colored square n times instead of writing all c's explicitly. We also abbreviate colors by their first letter (e.g. b denotes a blue square). Furthermore we specify spaces on the grid as tuples $(x, y) \in \mathbb{Z}$ with $(0, 0)$ being the space on the bottom left in the corresponding figure.

(a) Empty space

(b) Target space of a blue square

(c) A space which changes directions – When a square moves to this space it changes its direction to ▽

(d) A red square with direction ▷

(e) A blue square on a space with black triangle

(f) An orange square on a space with an orange circle

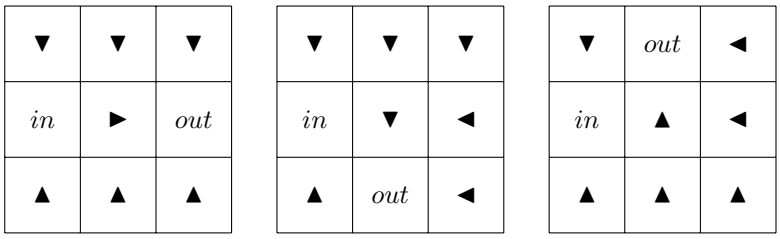
Figure 9: The different symbols used in *Game about Squares*

All our gadgets will be surrounded by a border of inwards facing triangles to indicate that the gadgets can only be entered or left via the designated input and output spaces. Technically this border needs to have a width of k spaces of inwards facing triangles (with $k \geq |C|$) to ensure this property.[10] Fig. 9 shows the different symbols we will use in the following figures. W. l. o. g., we will use

[10] Technically, the border also needs to contain k spaces of outwards facing triangles so that no outside square can enter the gadget. Initially however there are no squares outside of a gadget and since no square can leave a gadget in an undesired way, there also will be no square outside of a gadget anytime in the course of the game.

the two colors *blue* (b) and *orange* (o) for squares (and corresponding target positions) inside our gadgets[11] and we assume that there is a *red* (r) square which the player has to push across all gadgets. This means the main question will be whether the red square can reach its target position. The gadgets will be designed in such a way that the "inner" squares can reach their target positions without problems.

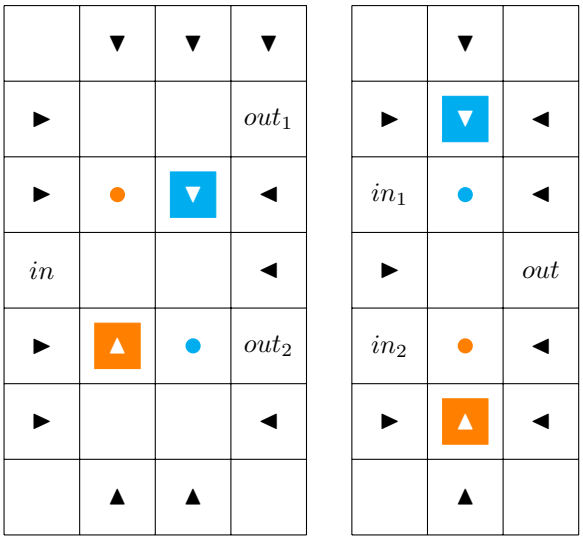
Lastly, when describing a sequence of moves, we assume that the red square has the correct direction – usually facing into the corresponding gadget.



(a) One-way (b) Turn (right) (c) Turn (left)

Figure 10: One-way and turn gadgets as *Game about Squares* subinstances

Fig. 10 shows how to construct a one-way gadget. It also shows how to construct turn gadgets. In all three gadgets (r^2) is a sequence which moves the red square from the input (in) to the output (out). The triangle on the center space $(1, 1)$ makes sure that the red square cannot traverse the gadgets the other way.



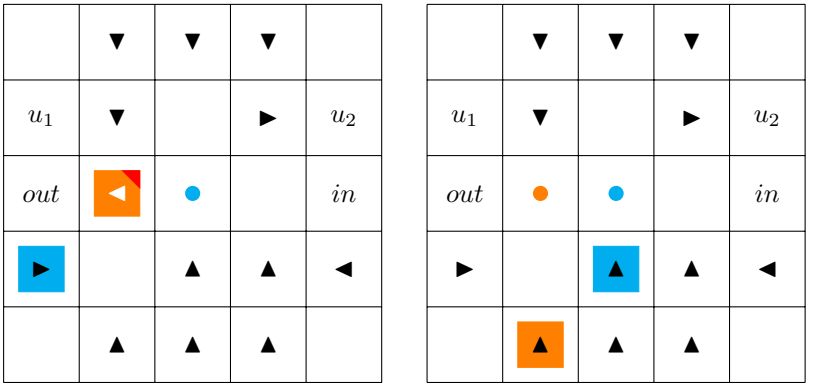
(a) Fork (b) Defork

Figure 11: Fork and defork gadgets as *Game about Squares* subinstances

Fig. 11a shows how to construct a fork gadget. The sequence (b^2, r, o^2, r^2)

[11] In the total construction, we have to make sure that different gadgets use different colored squares/circles.

moves the red square from in to out_1 and both inner squares to their target positions. The sequence (o^2, r^2, b, r, b) moves the red square from in to out_2 and both inner squares to their target position. Similarily, fig. 11b shows how to construct a defork gadget. The sequences (o, r, b, r) or (b, r, o, r) move the red square from in_1 or in_2 to out and the inner squares to their target positions.



(a) Initial state

(b) State after the red square moved from u_1 to u_2

Figure 12: An unlock gadget in different states as *Game about Squares* subinstances

Fig. 12a shows how to construct the unlock gadget. We have to show that there is a sequence of moves from u_1 to u_2 and we have to show that there is a sequence which moves the red square from in to out and the inner squares to their corresponding target positions when a square moved from u_1 to u_2 at a previous point in time. In addition, we have to show that there is no sequence of moves from u_1 to out and there is no sequence of moves from in to u_2. A sequence of moves from u_1 to u_2 is (r^3, b^2, r^3). Fig. 12b shows the unlock gadget after this sequence of moves. Moving the red square now from in to out (and the inner squares to their target positions) is achieved with (r^4, b, o^2). It is also easy to see that there is no sequence of moves from in to out when no square moved from u_1 to u_2: The orange square blocks the path. It is not possible to push the orange square to the left because then it cannot reach its target position anymore. It is also not possible to move it to the right because its direction is $\triangleleft$ and no square can enter the gadget via its output. Lastly, it is not possible to move the orange square up because there is no square on either $(0, 0)$, $(1, 0)$ or $(1, 1)$. This means, the only way to "unblock" the path is by pushing the orange square down. This is only possible by a square on $(1, 3)$ and this space can only be reached from u_1. Analogously there is no sequence which moves a square (with direction $\triangleright$) from u_1 to out and that there is no sequence which moves a square (with direction $\triangleleft$) from in to u_2.

Now we have seen how to construct the four essential gadgets. We don't need any technical gadgets (apart from the turn gadgets in fig. 10): Since squares cannot change their direction at will, we just need rows or columns of empty spaces to simulate straight edges. For non-straight edges we take advantage of the fact, that graphs can be drawn orthogonally. This means we only need straight edges and 90° turns. Further we need no special construction for crossovers

because, again, squares cannot change their direction. Additionally, we don't need dedicated starting or target gadgets: Instead of a starting gadget, we just put a red square on the input field of the first gadget in our total construction (a node gadget's defork "sub gadget") and instead of a target gadget, we just put a red circle on the output field of the last gadget in our total construction (a node gadget's unlock "sub gadget"). Therefore, we have shown that GAS is NP-hard, which strengthens Jens Maßberg's results.

The question that remains is the NP-membership. GAS doesn't feature an implicit (or explicit) bound on the length of sequences of moves. On the one hand, GAS is similar to games like SOKOBAN, which are PSPACE-complete but on the other hand, GAS also features similarities to games like PUSH-∗-X, which are NP-complete [12]. Roughly speaking, the difference between both games is that SOKOBAN asks for a "complete" configuration of the game, whereas PUSH-∗-X only asks if a single block (or rather the player character) can reach a specific position. Both problems have counterparts in GAS – either every square has a corresponding target position or there is only one square which has a corresponding target position.

3.2 2048$^+$

(a) Ongoing game (b) Win (c) Loss

Figure 13: Three screenshots taken from the browser version of 2048. The first screenshot shows an ongoing game, the second one a win because the player reached a tile with value 2048 and the third one loss because no more moves are possible.

2048 is a sliding block puzzle developed in 2014 by Gabriele Cirulli [5]. The game is played on a 4×4 grid where some squares are occupied by numbered tiles. The player subsequently chooses directions in $\{\uparrow, \rightarrow, \downarrow, \leftarrow\}$ which move *all* tiles in the chosen direction. Tiles slide in the desired direction until they hit an obstacle (the edge of the grid or another tile with a different value) or until they hit another tile of the same value. When two tiles of the same value w meet, they *merge* and create a new tile with value $2w$. Every move the player makes generates a new tile of value either 2 or 4 on an empty square on the grid. At the start the game creates two tiles with either value 2 or 4 on an otherwise empty grid. The player's goal is to create a tile of value 2048. If the grid is completely filled and the player cannot move any tiles, the game is over. Fig. 13 shows three example screenshots. To get from the first depicted situation (fig. 13a) to the second one (fig. 13b), the player needs to execute the following sequence of moves: $\downarrow, \leftarrow, \leftarrow, \leftarrow, \downarrow, \rightarrow, \rightarrow$.

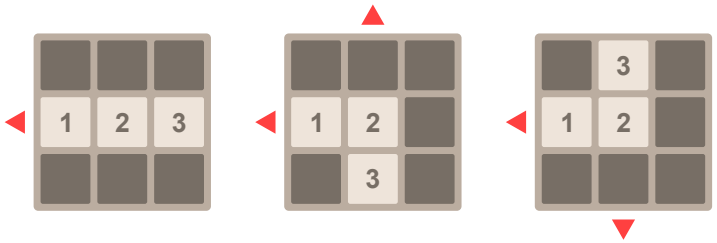
To generalize the game we allow grids of size $n \times m$. To make the construction of the different gadgets easier, we also assume that the value the player tries to reach is dependent on the grid size. We define the goal value for a grid of size $n \times m$ (with $n \geq m$) as 2^{3n-1}. We also assume that every newly created tile has a unique positive integer value $w < 2^{3n-1}$. We call this version 2048^+.

▶ 2048^+
Instance: A $n \times m$ grid where some squares are occupied by tiles with positive integer values $w < 2^{3n-1}$.
Question: Is it possible to create a tile with value 2^{3n-1}?

Below we will interpret the squares as coordinates $(x, y) \in \mathbb{Z}^2$ where the square on the bottom left is assigned the coordinate $(0, 0)$. Since all relevant tile values are powers of two, we will only write the respective exponent on the tile. Further we will use dark tinted squares (without any value) to indicate tiles which will never merge with other tiles. These will be used as "barriers".[12] In addition, we will construct the gadgets in such a way that the inputs always start with a tile of value 2^1 – later we will see why this doesn't pose a problem.

All our gadgets will be designed in such a way that every move only enforces a single merge. This means we can specify the single tile which was generated by this merge. We will call this tile the *active tile* and its position the *active square*. With this we can interpret successive merges as a sequence of moves through the corresponding gadget by listing the active squares. To write down sequences of moves, we will give sequences of directions $r \in \{\uparrow, \rightarrow, \downarrow, \leftarrow\}$. Additionally we will write (r^n) for a (sub-)sequence of n consecutive moves in direction r.

Due to the fact that moves in 2048 are global and shift whole or at least partial rows or columns, it is possible that moves will destroy gadgets placed on the same row or column. When placing the gadgets, we have to make sure that this doesn't happen. For this we will use red triangles to indicate which partial rows and columns will be "destroyed" by using the respective gadget. These marks show where we are not allowed to place other gadgets which will be used at a later point in time.

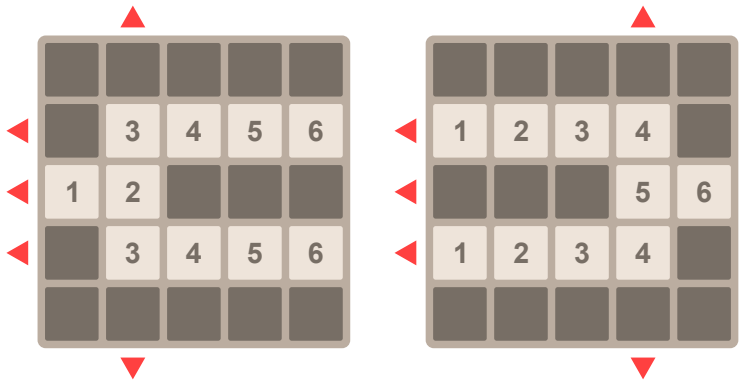


(a) One-way (b) Turn (right) (c) Turn (left)

Figure 14: One-way and turn gadgets as 2048 subinstances.

Fig. 14 shows how to construct a one-way gadget. It also shows how to construct turn gadgets. We assume that a tile with value 2^1 is located on square $(-1, 1)$. In all three cases it is possible to make the square containing the tile with value 2^3 the active square. The respective sequences of moves are $(\rightarrow^3)$, $(\rightarrow^2, \downarrow)$ and $(\rightarrow^2, \uparrow)$.

[12] We will show which values we need to assign to these tiles.



(a) Fork (b) Defork

Figure 15: Fork and defork gadgets as 2048 subinstances.

Fig. 15 shows how to construct fork and defork gadgets. For the fork gadget we assume that a tile with value 2^1 is located on $(-1, 2)$. Then there are sequences of moves to either make $(4, 3)$ or $(4, 1)$ the active square. Said sequences are $(\rightarrow^2, \uparrow, \rightarrow^3)$ and $(\rightarrow^2, \downarrow, \rightarrow^3)$. In the first case, the active squares are $(0, 2)$, $(1, 2)$, $(1, 3)$, $(2, 3)$, $(3, 3)$ and $(4, 3)$. This sequence of moves is possible because each merge increments the power of the tile on the active square by one and the power of the next tile in the sequence of moves has the same value. Thus, the merges are possible. See 5.2 in the appendix for an example. The defork gadget works in a similar way: There are sequences of moves which makes $(4, 2)$ the active square, if there is a tile with value 2^1 on either $(-1, 1)$ or $(-1, 3)$

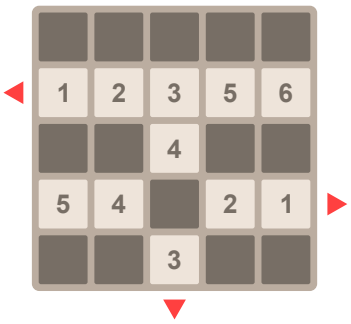


Figure 16: An unlock gadget as 2048 subinstance. The vertical distance between the second and fourth row can be arbitrarily large. The important detail is that a tile with value 2^4 (square $(2, 2)$) is located directly below the "upper" tile with value 2^3 (square $(2, 3)$).

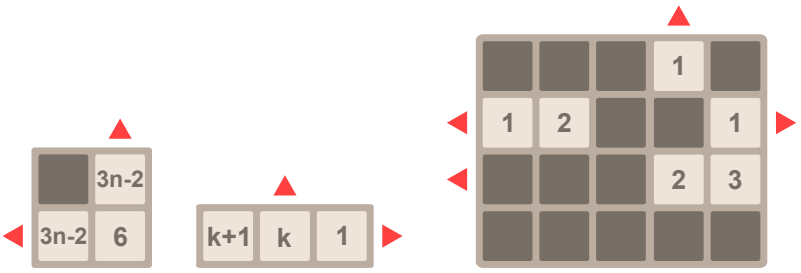
Fig. 16 shows how to construct the unlock gadget. We have to show that there is a sequence of moves which makes square $(4, 3)$ the active square (when there is a tile wich value 2^1 on square $(-1, 3)$) and we have to show that there is a sequence of moves which makes square $(0, 1)$ the active square when there is a tile with value 2^1 on square $(5, 1)$ and square $(4, 3)$ was the active square at a previous point in time. In addition, we have to show that there is no sequence of moves which makes first $(0, 3)$ and then $(0, 1)$ the active square and there is

no sequence of moves which makes first $(4,1)$ and then $(4,3)$ the active square.

A sequence of moves for the first property is $(\rightarrow^3, \uparrow, \rightarrow^2)$.

To make square $(0,1)$ the active square, we have to merge the tile with the tile on $(1,1)$ with $(\leftarrow)$.[13] This is only possible if the tile on square $(1,1)$ has an incremented exponent – which means it had to be the active square beforehand. The only tile (of non-unique value) which can be moved to a neighboring square is the tile with value 2^3 on square $(2,0)$. This happens when the previously mentioned sequence of moves is executed. We assume that a tile with value 2^1 is located on square $(5,1)$ and that square $(4,3)$ was already active. Then the sequence $(\leftarrow^5)$ make square $(5,1)$ the active square and thus the second property holds.

The last two properties hold because there is no possibility to move a tile with value 2^3 on square $(1,2)$ and there is no possibility to move a tile with value 2^4 on square $(3,2)$.



(a) Target (b) Reset (c) Skip

Figure 17: Target and skip gadgets as 2048 subinstances.

Now we have seen how to construct the four essential gadgets. The technical gadgets that remain are those to simulate edges, the starting and the target gadget and crossovers. Again we assume that the corresponding Edge Hop graph is orthogonal. The subinstances for turns are shown in fig. 14. The starting gadget is a single tile with value 2^1. The target gadget is shown in fig. 17a. In our reduction from DHC we put a target node next to the output of a node gadget. The output of a node gadget corresponds to the output of an unlock gadget. In our 2048^+ unlock gadget (fig. 16), the output is a tile with value 2^5. Upon traversing this gadget (which makes this "output tile" the active tile), it has a value of 2^6. Therefore the tile on $(1,0)$ in our target gadget has a value of 2^6. If we put the target gadget right beside the last unlock gadget in a way that the mentioned tiles are neighbors, the sequence of moves $(\rightarrow, \downarrow)$ will lead to a tile with value 2^{3n-1}. Though, the value on square $(1,0)$ can be an arbitrary power of two less then 2^{3n-2}. It is possible to extend one-way gadgets to reach arbitrarily high values and the reset gadget can reset values back to 2^1. Fig. 17b shows a reset gadget. We assume that a tile with value 2^k is located on $(1,1)$. Then $(\downarrow, \leftarrow)$ is a sequence of moves which makes $(1,1)$ the active square with a tile of value 2^1 again. The reset gadget is needed to reset too high tile values back to 2^1. In our construction, all gadgets start with an "input tile" of value 2^1 but end with tiles of a higher value. If we put a reset gadget right after

[13] The tiles on $(0,0)$ and $(0,2)$ have unique values and we assume that we cannot enter the gadget via its output $(-1,1)$.

each gadget, we can keep the values of all input tiles at 2^1.

To simulate straight edges, we use the skip gadget (fig. 17c). As the name suggests, the horizontal distance between squares $(1, 2)$ and $(3, 2)$ can be arbitrarily large. Since the tiles between these squares are irrelevant (for the skip gadget itself), it doesn't matter whether the player executes $(\uparrow)$ or $(\downarrow)$ on these columns. We assume that there is a tile with value 2^1 on square $(-1, 2)$. Then the sequence $(\rightarrow, \leftarrow, \downarrow^2, \rightarrow)$ makes square $(4, 1)$ the active one. Since we didn't use any tiles in the middle column, it doesn't matter if there were $\uparrow$ or $\downarrow$ moves beforehand. It is also easy to see that the distance between squares $(1, 2)$ and $(3, 2)$ can be arbitrarily large because the $\leftarrow$ move in our sequence moves all tiles in the same row to the right of square $(0, 2)$ one square to the left.

To construct a crossover gadget we can combine two skip gadgets. Fig. 18 shows an example.

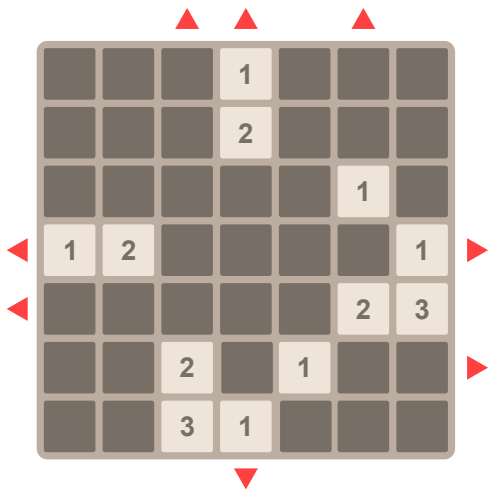


Figure 18: A crossover gadget as 2048 subinstance.

In all our gadgets we assumed that every dark tinted tile and every newly created tile has a unique value which is not a power of two. It is possible to assign the values in such a way because we have at most $n \cdot m < n^2$ tiles and $n^2 \leq 2^{3n-1} - 3n$ (for $n \in \mathbb{N}$). Thus, 2048^+ is NP-hard.

In addition, the "uniqueness" of values also gives a simple NP-membership: Every move generates a new tile which can never be merged with another tile. Since the grid has a size of $n \times m$ we cannot execute more than $n \cdot m$ moves before the grid is filled and therefore every sequence of moves which leads to a tile with value 2^{3n-1} has length at most $n \cdot m$.

A question that remains is the complexity of "proper" 2048. It is possible to assign every dark tinted tile a unique power of two (greater than 2048) and repose the question as: "Is it possible to create another tile with value 2048?" Though in this case one has to make sure that the newly created tiles (of value 2 or 4) cannot compromise the construction. To see other generalizations and analyses see the works of Christopher Chen [4] (a variant where no new tiles are created) and Rahul Mehta [16] (a variant which interprets 2048 as two-player game).

4 Conclusion

We have seen that EdgeHop is NP-complete and we have seen how to show NP-hardness of other problems by constructing a handful of different gadgets, namely:

- a gadget which features a "path"[14] from x to y (but not the other way round)
- a gadget which features a path from x to y_1 or from x to y_2
- a gadget which features a path from x_1 to y or from x_2 to y
- a gadget which features a path from u_1 to u_2 and a path from x to y if the path from u_1 to u_2 was used at a previous point in time (In addition, it has to be ensured that there is no path from u_1 to y or from x to u_2.)
- a gadget for the starting node
- a gadget for the target node
- a gadget to simulate the edges of the graph (either universal edge gadgets or separate gadgets for straight lines and for 90° turns)
- a crossover gadget

The first three gadgets and the gadget to simulate edges are a natural fit for motion-planning problems. Often, starting and target gadgets are also easily implementable in motion planning problems. In addition, the unlock gadget can be realized in many different ways. An obvious example are locked doors and corresponding keys which have to be collected. The only "problematic" gadget is the crossover gadget, at least for two-dimensional motion planning problems.

4.1 Further Work

Even though we have shown the NP-hardness of EdgeHop there are some questions which remain open. Our Edge Hop graph is not planar. Since it is hard or impossible to design crossover gadgets for some problems, one question is if there is a way to design a crossover gadget out of our defined gadgets, to make the Edge Hop graph planar.

Another open question is how the complexity of EdgeHop changes if we modify our rules:

- What changes if we allow that edges can be crossed arbitrarily often?
- Does it make a difference if we introduce multiple *active* tokens, potentially with individual target nodes?
- Does the complexity change if we use other graphs instead of undirected graphs? (Digraphs, multigraphs, ...)
- Do the results differ if we allow "pushing" passive moves instead of (or in addition to) the defined "pulling" passive moves?

[14] Here, *path* can be an actual path (for example motion planning problems), or it can be just a sequence of actions and decisions.

If the active token is allowed to cross edges arbitrarily often then it is maybe possible to design "memory gadgets" with which it would be possible to reduce from *QBF*. If we introduce multiple target nodes, then our game will become more similar to Sokoban. This could also change the complexity.

One other direction for further research would be to show the NP-hardness of other problems with the aid of EdgeHop. We have seen the NP-hardness for *Game about Squares* and for a variation of *2048*. Further, the NP-hardness for specific variants of *Latrunculi* and *Minecraft* was shown in [11].

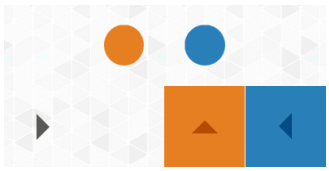
Lastly, to strengthen our results, it remains to be shown how EdgeHop relates to different variants of the NCL.

5 Appendix

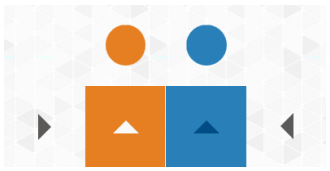
5.1 Sequence of moves to solve level 9 (Game about Squares)

(a) Initial situation

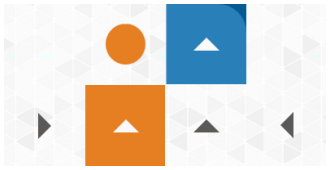
(b) Move 1, pushing the orange square



(c) Move 2, pushing the orange square



(d) Move 3, pushing the blue square

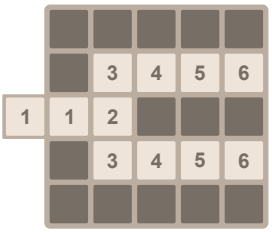


(e) Move 4, pushing the blue square

(f) Move 1, pushing the orange square

Figure 19: A sequence of moves to solve level 9 of *Game about Squares*. Using our notation, this sequence of moves is (o^2, b^2, o)

5.2 Sequence of moves to traverse the fork gadget (2048)



(a) Initial situation

(b) Move 1, $\rightarrow$. The active square is now $(0, 2)$

(c) Move 2, $\rightarrow$. The active square is now $(1, 2)$.

(d) Move 3, $\downarrow$. The active square is now $(1, 1)$. In addition, the upper tile with value 2^3 (square $(1, 3)$) moved one position down.

(e) Move 4, $\rightarrow$. The active square is now $(2, 1)$.

(f) Move 5, $\rightarrow$. The active square is now $(3, 1)$.

(g) Move 6, $\rightarrow$. The active square is now $(4, 1)$.

Figure 20: A sequence of moves in a fork gadget which makes $(4, 1)$ the active square.

References

[1] Aaron B. Adcock et al. "Zig-Zag Numberlink is NP-Complete". In: *CoRR* abs/1410.5845 (2014). URL: http://arxiv.org/abs/1410.5845.

[2] Greg Aloupis et al. "Classic Nintendo Games Are (Computationally) Hard". In: *Fun with Algorithms - 7th International Conference, FUN 2014, Lipari Island, Sicily, Italy, July 1-3, 2014. Proceedings*. Ed. by Alfredo Ferro, Fabrizio Luccio, and Peter Widmayer. Vol. 8496. Lecture Notes in Computer Science. Springer, 2014, pp. 40–51. DOI: 10.1007/978-3-319-07890-8_4. URL: http://dx.doi.org/10.1007/978-3-319-07890-8_4.

[3] Édouard Bonnet. "Résultats Positifs et Négatifs en Approximation et Complexité Paramétrée". PhD thesis. Université Paris-Dauphine, Nov. 2014.

[4] Christopher Chen. *2048 is in NP | Open Endings*. 2014. URL: http://blog.openendings.net/2014/03/2048-is-in-np.html (visited on 03/02/2018).

[5] Gabriele Cirulli. *2048 – Gabriele Cirulli*. 2014. URL: https://gabrielecirulli.com/2048 (visited on 03/02/2018).

[6] Stephen A. Cook. "The Complexity of Theorem-proving Procedures". In: *Proceedings of the Third Annual ACM Symposium on Theory of Computing*. STOC '71. Shaker Heights, Ohio, USA: ACM, 1971, pp. 151–158. DOI: 10.1145/800157.805047. URL: http://doi.acm.org/10.1145/800157.805047.

[7] Erik Demaine. *Erik Demaine's Combinatorial Games Page*. 2010. URL: http://erikdemaine.org/games/ (visited on 03/02/2018).

[8] Dariusz Dereniowski. "Phutball is PSPACE-hard". In: *Theoretical Computer Science* 411.44-46 (2010), pp. 3971–3978. DOI: 10.1016/j.tcs.2010.08.019.

[9] Henning Fernau et al. "On the parameterized complexity of the generalized rush hour puzzle". In: *Proceedings of the 15th Canadian Conference on Computational Geometry, CCCG'03, Halifax, Canada, August 11-13, 2003*. 2003, pp. 6–9. URL: http://www.cccg.ca/proceedings/2003/22.pdf.

[10] Gary William Flake and Eric B. Baum. "Rush Hour is PSPACE-complete, or "Why you should generously tip parking lot attendants"". In: *Theor. Comput. Sci.* 270.1-2 (2002), pp. 895–911. DOI: 10.1016/S0304-3975(01)00173-6. URL: http://dx.doi.org/10.1016/S0304-3975(01)00173-6.

[11] Moritz Gobbert. "Edge Hop – Ein Modell zur Komplexitätsanalyse von kombinatorischen Spielen". MA thesis. Universität Trier, May 2015. URL: https://www.uni-trier.de/fileadmin/fb4/prof/INF/TIN/Veroeffentlichungen/Gob2015.pdf.

[12] Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation*. A K Peters, 2009. ISBN: 978-1-56881-322-6.

[13] Robert A. Hearn and Erik D. Demaine. "PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation". In: *Theoretical Computer Science* 343 (2005), pp. 72–96. DOI: doi:10.1016/j.tcs.2005.05.008.

[14] G. Kendall, A. Parkes, and K. Spoerer. "A Survey of NP-Complete Puzzles". In: *International Computer Games Association Journal (ICGA)* 31 (2008), pp. 13–34.

[15] Jens Maßberg. "The "Game about Squares" is NP-hard". In: *CoRR* abs/1408.3645 (2014). URL: http://arxiv.org/abs/1408.3645.

[16] Rahul Mehta. "2048 is (PSPACE) Hard, but Sometimes Easy". In: *CoRR* abs/1408.6315 (2014). URL: http://arxiv.org/abs/1408.6315.

[17] Paul Rendell. "A Universal Turing Machine in Conway's Game of Life". In: *2011 International Conference on High Performance Computing & Simulation, HPCS 2012, Istanbul, Turkey, July 4-8, 2011*. Ed. by Waleed W. Smari and John P. McIntire. IEEE, 2011, pp. 764–772. DOI: 10.1109/HPCSim.2011.5999906. URL: http://dx.doi.org/10.1109/HPCSim.2011.5999906.

[18] J. M. Robson. "N by N Checkers is Exptime Complete". In: *SIAM J. Comput.* 13.2 (1984), pp. 252–267. DOI: 10.1137/0213018. URL: http://dx.doi.org/10.1137/0213018.

[19] Andrey Shevchuk. *Game about Squares*. 2014. URL: http://gameaboutsquares.com/ (visited on 03/02/2018).

[20] Giovanni Viglietta. "Lemmings Is PSPACE-Complete". In: *Fun with Algorithms - 7th International Conference, FUN 2014, Lipari Island, Sicily, Italy, July 1-3, 2014. Proceedings*. Ed. by Alfredo Ferro, Fabrizio Luccio, and Peter Widmayer. Vol. 8496. Lecture Notes in Computer Science. Springer, 2014, pp. 340–351. DOI: 10.1007/978-3-319-07890-8_29. URL: http://dx.doi.org/10.1007/978-3-319-07890-8_29.

[21] Ingo Wegener. *Komplexitätstheorie: Grenzen der Effizienz von Algorithmen*. Springer, 2003. ISBN: 3-540-00161-1.

ECCC ISSN 1433-8092

https://eccc.weizmann.ac.il