Subway Shuffle is PSPACE–complete

(Preliminary draft)

Marzio De Biasi* Tim Ophelders†

February 2015

Abstract

Subway shuffle is an addicting puzzle game created by Bob Hearn. It is played on a graph with colored edges that represent subway lines; colored tokens that represent subway cars are placed on the nodes of the graph. A token can be moved from its current node to an empty one, but only if the two nodes are connected with an edge of the same color of the token. The aim of the game is to move a special token to its final target position. We prove that deciding if the game has a solution is PSPACE–complete even when the game graph is planar.

1 Introduction

In the last years the study of the complexity of puzzles and (video)games has gained much attention (see for example the survey [3]). Most games can be generalized to arbitrary instance size and transformed to decision problems in which the question is usually: "*Given an instance of size $m \times n$ of the game X, does it have a solution?*". It turns out that most static puzzles (sudoku, kakuro, binary puzzle, light up, ...) are NP–complete and that most dynamic puzzles (sokoban, rush hour, atomix, ...) are PSPACE–complete.

One of the puzzles for which the complexity was still unknown is Subway Shuffle (Figure 1), an addicting puzzle game created by Bob Hearn.

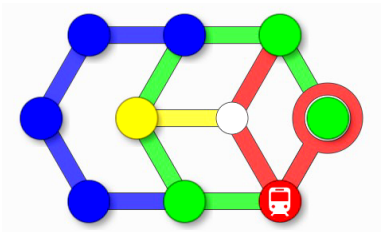

Figure 1: Level 14 of Subway Shuffle.

It is played on a graph with colored edges that represent subway lines; colored tokens that represent subway cars are placed on the nodes of the graph. A token

*marziodebiasi[at]gmail[dot]com
†t.a.e.ophelders[at]tue[dot]nl, Eindhoven University of Technology

can be moved from its current node to an empty one, but only if the two nodes are connected with an edge of the same color of the token. The aim of the game is to move a special token to its final target position.

We proved, as conjectured in [2], that its rules are rich enough to be PSPACE-complete. The proof uses the framework of the nondeterministic constraint logic model of computation ([2], [1]): given a planar constraint graph in normal form, it is PSPACE-complete to find a sequence of edge reversals (moves) that keep the constraint graph valid, ending in the reversal of a special edge e^*.

We build an equivalent subway shuffle board with gadgets that simulates the behaviour of the edges and vertices of the constraint graph and that has a solution (i.e. a sequence of moves that shift the special token to its final target position) if and only if there is a sequence of moves that reverses e^*.

2 Subway Shuffle decision problem

We can generalize the Subway Shuffle game to boards of arbitrary size and formulate it as a decision problem in this way:

Definition 2.1 (SUBWAY SHUFFLE problem)**.**
Input: Given a graph $G = (V, E)$, in which every edge e is colored with color $C(e) \in [1..c]$, and m colored tokens $T_1, T_2, ..., T_m$ placed on m distinct nodes of V and colored with color $C(T_i) \in [1..c]$; $T_1 = M$ is the *special token*, and one of the nodes $U \in V$ is the *target node*. A *legal move* is a pair $\langle T_i, (u_i, u_j)\rangle$ that represents the shift of token T_i placed on node u_i on an empty adjacent node u_j along an edge $e = (u_i, u_j) \in E$ of the same color, i.e. $C(T_i) = C(e)$;
Output: A sequence of legal moves, that shift the special token M to the target node U.

In our construction we use four colors ($c = 4$).

3 Nondeterministic Constraint Logic (NCL)

A *constraint graph* [2] is a directed graph G in which edges have a nonnegative integer *weight* and the nodes have a nonnegative integer *minimum in–flow constraint*. A *valid configuration* of the graph is an orientation of the edges such that, for each node, the total weight of its incoming edges is at least the minimum in–flow constraint. A move from one valid configuration to another is the reversal of a single edge such that the in–flow constraints remain satisfied. It is PSPACE–complete to decide if, given a valid initial configuration, there exists a sequence of moves that reverses a specified edge; this sequence can be viewed as a nondeterministic computation.

The problem remains PSPACE–complete even if the graph is in *normal form*: the weights are 1 or 2, all in–flow constraints are 2 and all vertices have degree 3. In [1] it is shown that two types of vertices are enough to simulate all NCL graphs in normal form: AND and OR vertices. The AND vertex has two (red) edges of weight 1, and a (blue) edge of weight 2 (Figure 2a); the OR vertex has three (blue) edges of weight 2 (Figure 2b).

We will use another type of vertex that is called LATCH and has three blue edges, one *input* and two *outputs* (see Figure 2c). When the input edge is

directed outward at most one output edge can be directed outward (the edge that can be directed outward represents an *internal 0–1 state* S); when the input edge is directed inward then both outputs can be directed outward. As a result, the output edges can be reversed only when the input edge is directed inward. An example of status change from $S = 0$ to $S = 1$ is shown in Figure 2d ($S = *$ means that both states are active).

Using an AND and a LATCH we can build an OR, like shown in Figure 2e; the only thing to notice is the *junction* between a blue edge of weight 2 and a red edge of weight 1 that can be realized using a vertex of degree 2 with an inflow constraint of 1. Though this type of vertex can also be converted in normal form as shown in [2], its behavior is implicitly realized by the way in which we connect the gadgets in subway shuffle, so we ignore it in our OR construction.

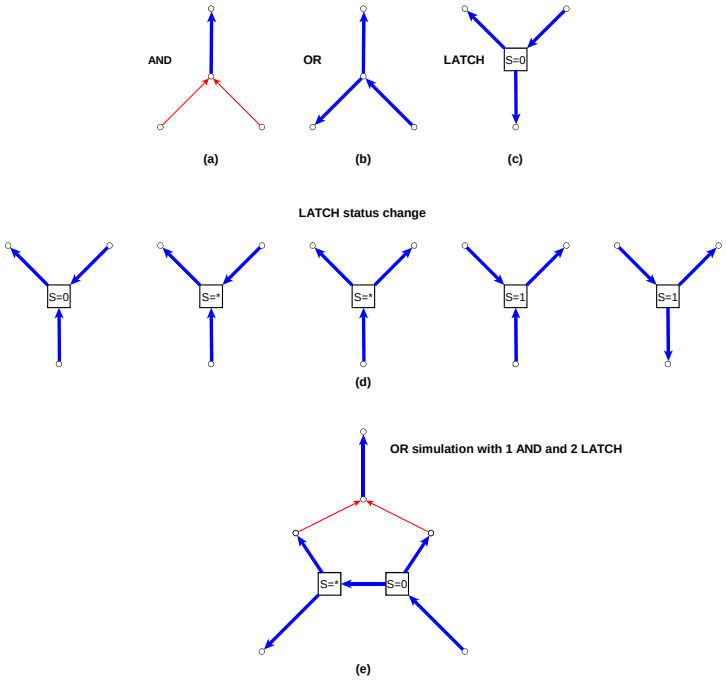

Figure 2: (a) AND, (b) OR and (c) LATCH vertices; (d) LATCH status change example; (e) OR simulation with an AND and two LATCH.

So given a planar constraint graph G we can build an equivalent constraint graph G' in normal form that contains only AND and LATCH vertices. The edge-reversal problem is PSPACE–complete even for *planar* graphs [2]; so we can immediately derive the following:

Corollary 3.1. *Given a planar constraint graph G in normal form that contains only AND and LATCH vertices, and given an edge e^* of G; it is PSPACE–complete to decide if there exists a sequence of moves that reverses e^*.*

4 Gadgets and reduction

Given a planar constraint graph in normal form with AND and LATCH vertices, we construct an instance of Subway Shuffle that has a solution if and only if the original constraint graph has a solution.

4.1 Reduction overview

In the graphs of the constructed instances of subway shuffle, all except one node is occupied by a token, so there is a single empty node at all times, which we refer to as the *empty token*. A solution to a constructed instance is then uniquely represented by the path traced by the empty token. Each (nonempty) token has one of four colors: yellow, green, blue or purple.

We use two types of subway shuffle *gadgets*:

- *Vertex gadgets* (*AND* and *LATCH*) that simulate the behavior of the vertices of the constraint graph;
- *EDGE gadgets* that simulate the behavior of the edges of the constraint graph (we will use the uppercase letters to distinguish an EDGE gadget from a single colored subway shuffle edge).

Every gadget is planar and has an *entry node* E. The gadgets are arranged like in the original constraint graph and there is a *connection track* made of purple edges (do not confuse them with EDGE gadgets) that connect together the entry nodes of every gadget. EDGE gadgets can be crossed by the purple edges of the connection track through a central *cross node* R, allowing the empty token to move from one side to the other; in this way it is possible to build the connection track without breaking the planarity of the resulting graph.

Initially every gadget is in a *valid configuration*, i.e. the tokens on its nodes are consistent with the logic of the corresponding edge or vertex of the constraint graph.

The empty token is placed on a node of the connection track; from there it can reach the entry node of every gadget H, *enter* it, and eventually change its configuration from the current one to another valid one. The empty token can leave the gadget only returning to its entry point. We will see that in some cases the empty token can move to an adjacent gadget H' instead of leaving through the entry point of H. In such case, the empty token cannot leave H' except through H, and this cannot change the configuration of H'.

Figure 3 summarizes the structure of the subway shuffle board corresponding to the simple constraint graph with one AND and two LATCH that simulate the OR behavior.

4.2 EDGE gadget

The *EDGE gadget* is a straight path of blue edges with two endpoint nodes A and B. Auxiliary connections allow a blue token positioned on one endpoint to be shifted on the other endpoint. It is used to connect two Vertex gadgets together; in particular the endpoints are shared with the Vertex gadgets. The only nodes that will be connected to the remaining part of the board are the two *endpoints* A, B, the *entry node* E and the central *cross node* R of the blue

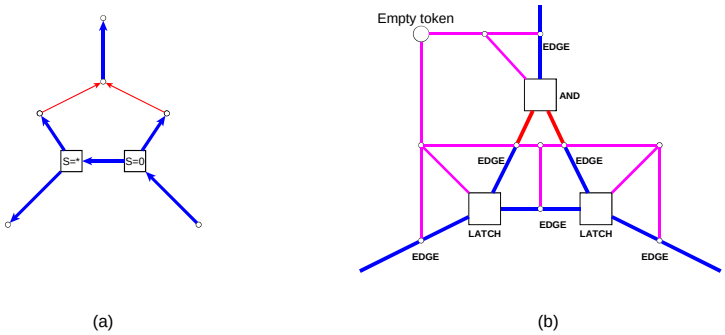

Figure 3: A simple constraint graph (a); and the skeleton structure of the corresponding subway shuffle board (b): the connection track made with purple edges allows the empty token to reach the entry node of each subway shuffle gadget (EDGE, AND, LATCH) and change its configuration.

path, which is connected to E with a purple edge and is part of the connection track and can be used by the empty token to cross the gadget. The EDGE gadget can be in three states:

- *Unlocked*: a blue token is present along the blue path, and the empty token, starting from the entry node E, can shift it from one endpoint to another (see Figure 4a,b). As we will see, an EDGE is in the unlocked state if and only if the corresponding edge of the constraint graph can be reversed.
- *Locked A*: the blue token is *locked* in the vertex gadget connected to the endpoint A (see Figure 4c). As we will see, an EDGE is the locked A state if and only if the corresponding edge of constraint graph is directed towards the vertex connected to the endpoint A and cannot be reversed.
- *Locked B*: the blue token is *locked* in the vertex gadget connected to the endpoint B (see Figure 4d). As we will see, an EDGE is the locked B state if and only if the corresponding edge of constraint graph is directed towards the vertex connected to the endpoint B and cannot be reversed.

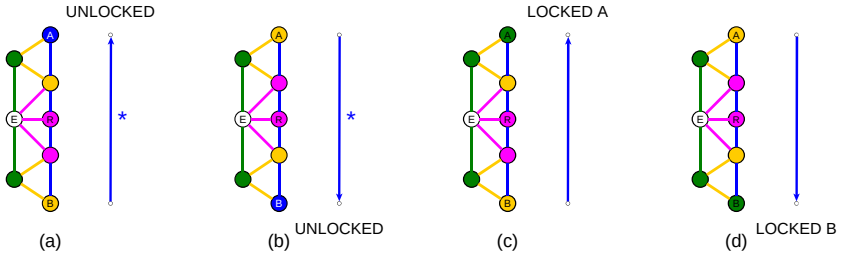

Figure 4: An EDGE gadget and its possible states: Unlocked (a,b), Locked A (c), Locked B (d).

The shift sequence to move the blue token from endpoint A to B is shown in Figure 5; the sequence can be reversed to move the blue token from B to A.

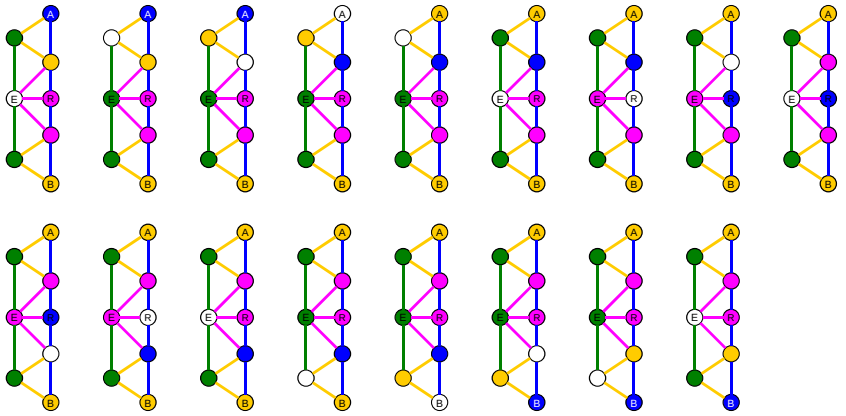

Figure 5: The move sequence to move the blue token from one endpoint to the other in an EDGE gadget.

4.3 AND gadget

The *AND gadget*, shown in Figure 6a, is connected to three EDGE gadgets H_A, H_B, H_C through nodes A, B and C (which are also the endpoints of the EDGEs). Once both H_A and H_B edges are pointing inward, i.e. there is a blue token on nodes A and B, the empty token, from the entry node E, can make a counter–clockwise loop, shift one of the blue tokens on node C so that the edge H_C can be later be reversed outward, and finally reach the entry node again (Figure 6b).

4.4 LATCH gadget

The *LATCH gadget*, shown in Figure 7a, is connected to three EDGE gadgets H_A, H_B, H_C through nodes A, B and C (which are also the endpoints of the EDGEs). Node B contains a blue token so edge H_B can be directed outward. Once H_A is directed inward, i.e. there is a blue token on node A, the empty token, from the entry node E can pop the blue token and move it to node C. At this point both H_B and H_C can be directed outward. In order to make H_A point outward again, a blue token must be picked from node B (or C) and moved back to node A (Figure 7b).

4.5 FINAL gadget

The *FINAL gadget*, shown in Figure 8a, is used to simulate the reversal of the target edge e^* of the constraint graph. It is simply a circuit linked to an EDGE gadget H_A through the endpoint A (so the FINAL gadget and the attached EDGE gadget H_A represent together the edge e^* of the original constraint graph), and it contains the special token M. If H_A is directed inward, i.e. a blue token is placed on node A, then the empty token, after reaching the entry node E can make a counter–clockwise loop placing the token M on node B; then it can leave the gadget and reach B from outside, placing M on its target node U (Figure 8b).

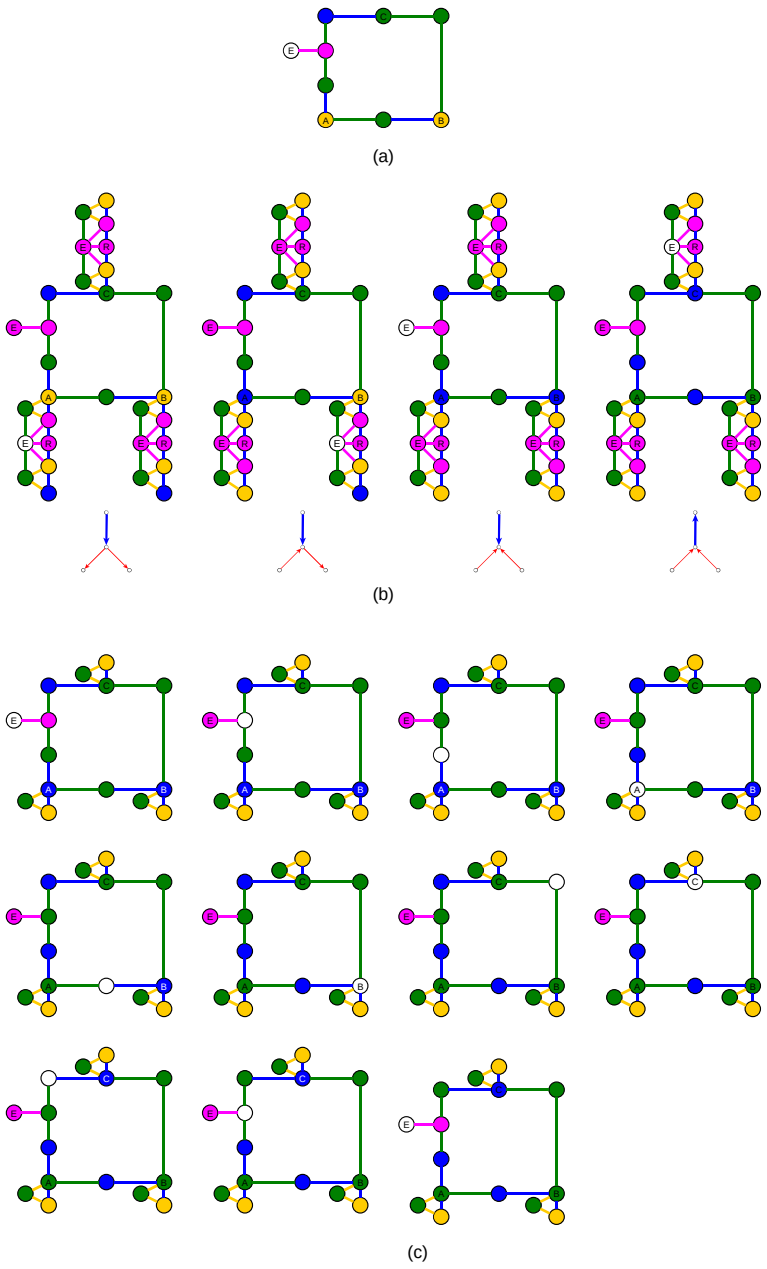

Figure 6: The AND gadget (a). How it is connected to the EDGE gadgets and the valid configurations that simulate the corresponding NCL vertex behavior (b). The sequence of moves that changes its configuration (c).

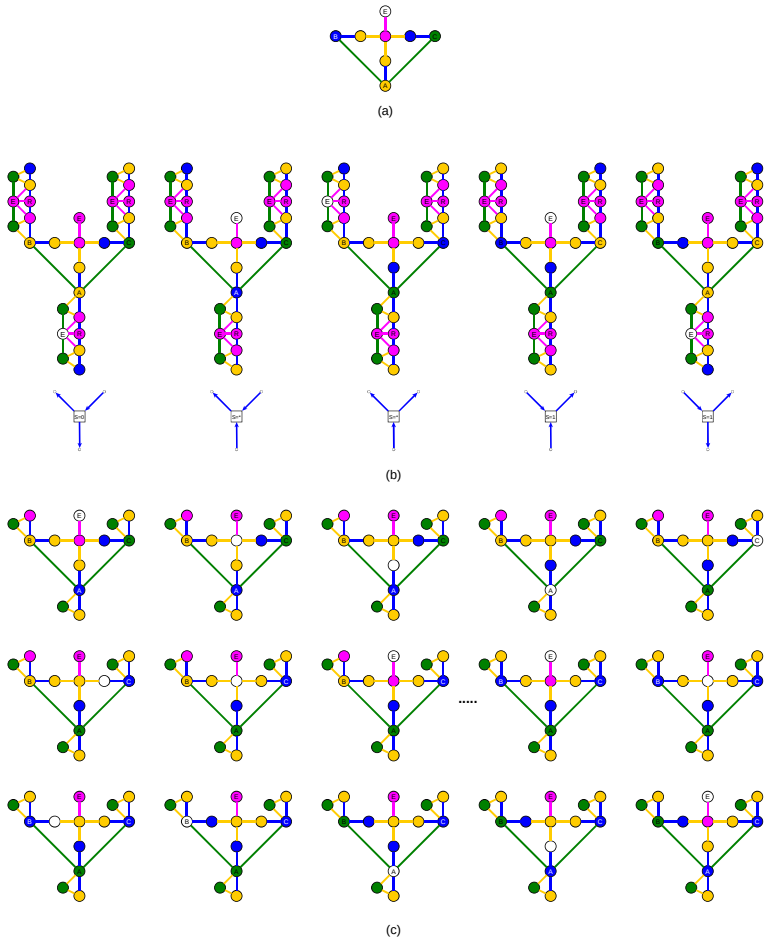

Figure 7: The LATCH gadget (a). How it is connected to the EDGE gadgets and the valid configurations that simulate the corresponding NCL vertex behavior (b). The sequence of moves that changes its configuration (c).

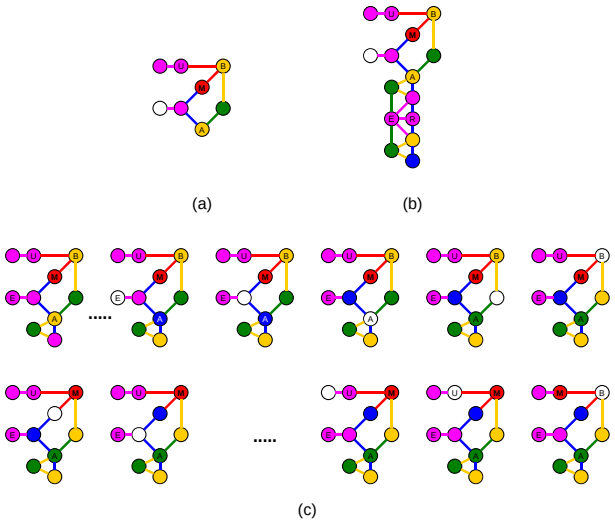

Figure 8: The FINAL gadget (a). How it is connected to the EDGE gadget in order to simulate the behavior of e^* (b). The sequence of moves that solves the puzzle leading the special token M to the target node U.

Note that in the figure, for better clarity, we used the red color for the M token and the path to U, but we can use the green color as well, without altering the gadget behavior.

4.6 Combining the gadgets

Like shown in Figure 2e we can combine the gadgets above to build a Subway Shuffle gadget that behaves like an NCL OR vertex (see Figure 9); note that edge weights are "embedded" in the AND and LATCH gadgets, so an EDGE gadget acts like an implicit degree 2 junction node that can connect a blue edge of weight 2 to a red edge of weight 1. Furthermore we can also arbitrarily extend the length of the EDGE gadgets and rearrange their nodes to make planar turns, without changing their behavior, like shown in Figure-10.

5 PSPACE–completeness

We prove that the behavior of the gadgets described in the previous section are *consistent* with the corresponding elements of the constraint graph; in particular the direction of an EDGE gadget can be reversed if and only if the corresponding edge in the constraint graph can be reversed.

It is important to notice that exactly one blue token is initially placed in each EDGE gadget, in one of the three states: unlocked, locked A, or locked B according to the corresponding constraint edge status, and by construction there is no way for the blue token to move from one edge gadget to another (i.e. a blue token cannot cross the AND and LATCH gadgets).

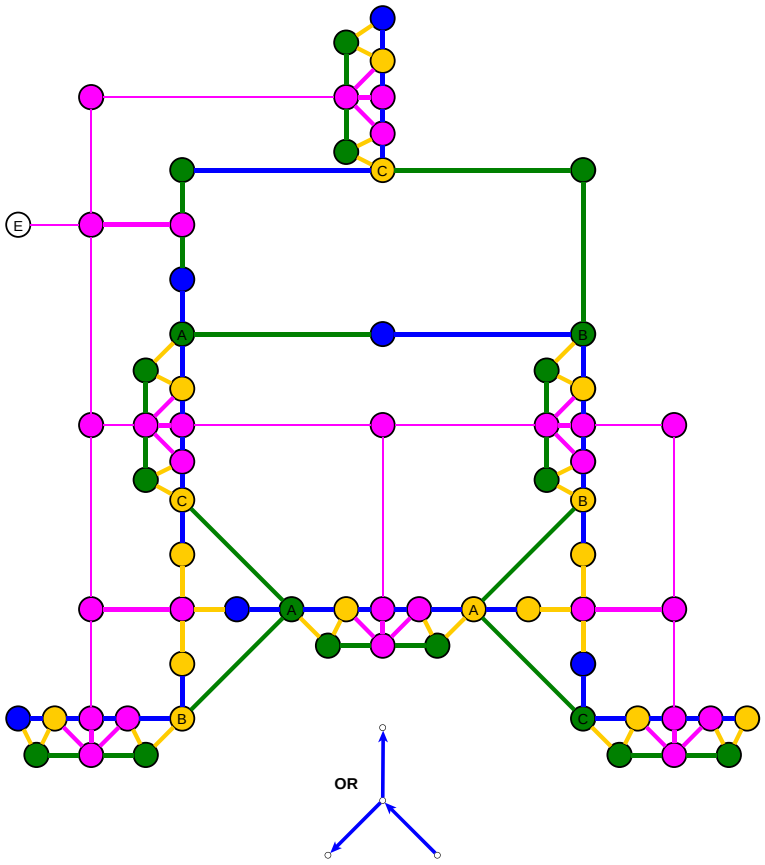

Figure 9: The simulation of an OR vertex using an AND and two LATCH.

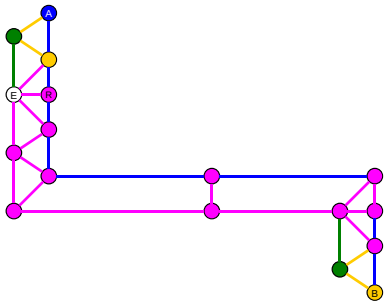

Figure 10: The EDGE gadgets can be arbitrarily extended and arranged to make planar turns.

5.1 Gadget behavior

We first analyze the behavior of the AND and LATCH gadgets when the empty token enters them through the entry node E and the gadget is in a consistent configuration.

The consistent configurations of the AND gadget are shown in figure 11 (for simplicity we represent two possible token colors with two overlapped tokens).

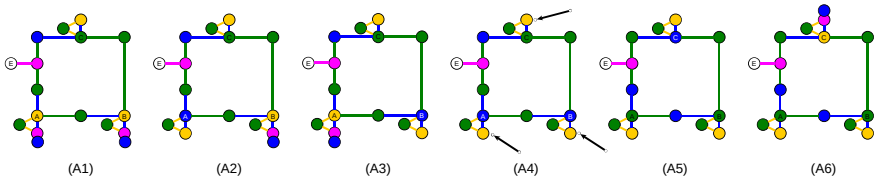

Figure 11: The consistent configurations of the AND gadget.

.

Notice that the only configuration change that can be made by the empty token on E is from $A4$ to $A5$ making a counter–clockwise loop or from $A5$ to $A4$ making a clockwise loop (see Figure 6c for the detailed sequence)). In both cases the neighbors of nodes A, B, C (marked in the picture with a black arrow) cannot contain another blue token because every EDGE gadget contains exactly one blue token, this prevents the empty token to exit the gadget from nodes A, B or C while performing the loop. So, if the empty token enters the AND gadget in a consistent configuration through the entry point E, we have the following cases:

- it leaves the gadget through E without changing its configuration;
- if there are two blue tokens on A and B ($A4$) it can lock them, perform a counter–clockwise loop unlocking the blue token on C, and finally exit the gadget through E ($A5$);
- conversely, if the two blue tokens on A and B are locked and there is a blue token on C ($A5$) then it can make a clockwise loop, locking the token on C and releasing the two tokens on A and B, and finally exit the gadget through E ($A4$).

The consistent configurations of the LATCH gadget are shown in figure 12 .

The only configuration changes that can be made by the empty token on E is from $L3$ to $L4$ or from $L5$ to $L6$ making a counter–clockwise loop on the right half–triangle, from $L6$ to $L10$ or from $L8$ to $L11$ making a counter–clockwise loop on the left half–triangle; and it can obviously reverse the change (see Figure 7c for the detailed sequence). In all cases there is only a blue token per EDGE gadget, so it cannot escape from the LATCH. So, if the empty token enters the LATCH gadget in a consistent configuration through the entry point E, we have the following cases:

- it leaves the gadget through E without changing its configuration;
- if there is a blue token on A and C is locked ($L3, L5$) it can lock the token on A and unlock the blue token on C ($L4, L6$);

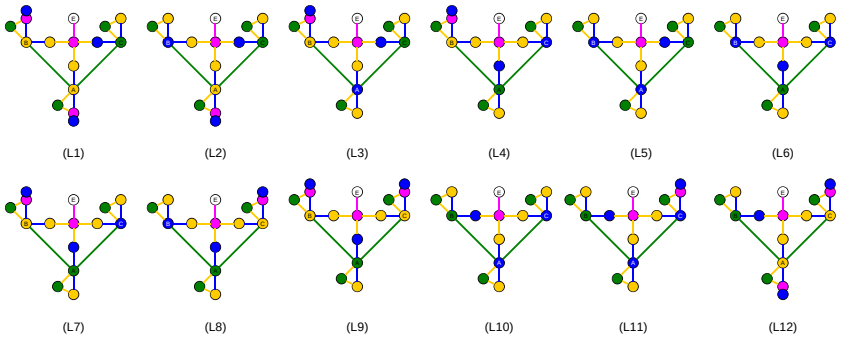

Figure 12: The consistent configurations of the LATCH gadget.

- if there is a blue token on A and B is locked $(L10, L11)$ it can lock the token on A and unlock the blue token on B $(L6, L8)$;
- conversely, if A is locked and there is a blue token on B (or C), it can lock the token on B (or C) and unlock the blue token on A.

Finally we analyze the behavior of the EDGE gadget. Again we suppose that both the EDGE gadgets and the Vertex gadgets are in a consistent configuration and that the empty token is placed on the entry node E of an EDGE gadget.

We analyze what can happen on endpoint A (the other is symmetric). If the edge is in the Locked A state, then the endpoint A contains a green token (see Figure 4c), and the empty token cannot leave the EDGE gadget through A. If the EDGE is unlocked, (see Figure 4a–b) then the empty token can reach the endpoint A (directly if it contains a yellow token or after moving down the blue token if it contains a blue token); at this point the empty token can enter the attached AND (or LATCH) gadget X; but we are assuming that the gadget X is in a consistent configuration, so the empty token can neither *(i)* leave X and jump to another EDGE (easily verified by case analysis on the valid configurations $A1 - A6$, $L1 - L12$), nor *(ii)* reach the entry node of X, because the purple token adjacent to E cannot be shifted. So, if the empty token enters the EDGE gadget in a consistent configuration through the entry point E, we have the following cases:

- it leaves the gadget through E without changing its configuration;
- it crosses the gadget through R without changing its configuration;
- if the EDGE is unlocked, it can move the blue token up and down according to the sequence shown in Figure 5.

Note that the blue token can be left in the middle of the EDGE gadget (and not necessarily on one of its endpoint), but this simply imply that when entering one of the attached gadget the empty token will not be able to lock the EDGE (and change the attached gadget configuration if it requires a blue token on the endpoint).

So we can state the following lemma:

Lemma 5.1. *The behavior of the EDGE, AND and LATCH gadgets are consistent with the behavior of the edges and vertices of the corresponding constraint graph.*

Proof. As seen above the EDGE attached to node C of an AND gadget can be directed outward (unlocked) if and only if the EDGEs attached to A and B are directed inward (locked). The EDGEs attached to nodes B, C of a LATCH gadget can be both directed outward (unlocked) only when the EDGE attached to A is pointing inward (locked); one of the EDGEs B, C must be directed inward (locked) before being able to point A outward (unlock). □

5.2 Main result

Theorem 5.2. Subway Shuffle *is PSPACE–complete.*

Proof. Given a planar constraint graph in normal form with only AND and LATCH vertices, we can build in polynomial time an equivalent Subway Shuffle board using the EDGE, AND, LATCH gadgets. The target edge e^* that must be reversed in the constraint graph is simulated by the FINAL gadget. We have seen that the empty token can freely traverse an EDGE gadget whatever its current configuration (direction) is from the entry node E through the cross node R, so we can build a purple track that connects every entry point of every subway shuffle gadget. As we have seen, by Lemma 5.1 the behavior of the gadgets is consistent with the logic of the corresponding constraint edges and vertices: the empty token can enter a gadget only through its entry node, and can leave it only through the same node leaving the gadget in a consistent configuration, furthermore it cannot change the configuration of the adjacent gadgets; so a sequence of moves that reverses e^* in the constraint graph exists, if and only if there is a sequence of moves in the subway shuffle board that leads the target token M to its final node U. It is easy to see that the solution can be found in polynomial space – which is equal to nondeterministic polynomial space (PSPACE = NPSPACE by Savitch's theorem [4]) – using a simple recursive program that nondeterministically picks the next move and verifies if the special token has reached the target node. □

6 Conclusion and open problems

We proved that the rules of Subway Shuffle are sufficiently complex to make solving instances PSPACE–complete, as conjectured by Hearn and Demaine in [2]. Using different gadgets, it may be possible to use less than four colors in a reduction for Subway Shuffle, this might lead to reductions for related games. An example of a related game whose complexity remains open is a variant of Rush Hour in which each block (car) has unitary size 1x1 and can move either horizontally or vertically. In particular, this variant of Rush Hour can be modeled as a version of Subway Shuffle with only two colors (representing horizontal and vertical cars and edges) and a more restricted class of graphs.

References

[1] Robert A. Hearn and Erik D. Demaine. The nondeterministic constraint logic model of computation: Reductions and applications. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 401–413, 2002.

[2] Robert A. Hearn and Erik D. Demaine. *Games, puzzles and computation.* A K Peters, 2009.

[3] Graham Kendall, Andrew J. Parkes, and Kristian Spoerer. A survey of np-complete puzzles. *ICGA Journal*, 31(1):13–34, 2008.

[4] Michael Sipser. *Introduction to the theory of computation.* PWS Publishing Company, 1997.