

ARRIVAL: A Zero-Player Graph Game in $\text{NP} \cap \text{coNP}$

Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek,
and Emo Welzl

Abstract Suppose that a train is running along a railway network, starting from a designated origin, with the goal of reaching a designated destination. The network, however, is of a special nature: every time the train traverses a switch, the switch will change its position immediately afterwards. Hence, the next time the train traverses the same switch, the other direction will be taken, so that directions alternate with each traversal of the switch.

Given a network with origin and destination, what is the complexity of deciding whether the train, starting at the origin, will eventually reach the destination?

It is easy to see that this problem can be solved in exponential time, but we are not aware of any polynomial-time method. In this short paper, we prove that the problem is in $\text{NP} \cap \text{coNP}$. This raises the question whether we have just failed to find a (simple) polynomial-time solution, or whether the complexity status is more subtle, as for some other well-known (two-player) graph games (Halman, *Algorithmica* 49(1):37–50, 2007).

1 Introduction

In this paper, a *switch graph* is a directed graph G in which every vertex has at most two outgoing edges, pointing to its *even* and to its *odd* successor. Formally, a switch graph is a 4-tuple $G = (V, E, s_0, s_1)$, where $s_0, s_1 : V \rightarrow V$, $E = \{(v, s_0(v)) :$

This research was done in the 2014 undergraduate seminar *Wie funktioniert Forschung?* (How does research work?).

J. Dohrau • B. Gärtner (✉) • M. Kohler • E. Welzl

Department of Computer Science, Institute of Theoretical Computer Science, ETH Zürich,
CH-8092 Zürich, Switzerland

e-mail: gaertner@inf.ethz.ch

J. Matoušek

Department of Computer Science, Institute of Theoretical Computer Science, ETH Zürich,
CH-8092 Zürich, Switzerland

Department of Applied Mathematics, Charles University, Malostranské nám. 25, 118 00 Prague,
Czech Republic

$v \in V\} \cup \{(v, s_1(v)) : v \in V\}$, with loops (v, v) allowed. Here, $s_0(v)$ is the even successor of v , and $s_1(v)$ the odd successor. We may have $s_0(v) = s_1(v)$ in which case v has just one outgoing edge. We always let $n = |V|$; for $v \in V$, $E^+(v)$ denotes the set of outgoing edges at v , while $E^-(v)$ is the set of incoming edges.

Given a switch graph $G = (V, E, s_0, s_1)$ with origin and destination $o, d \in V$, the following procedure describes the train run that we want to analyze; our problem is to decide whether the procedure terminates. For the procedure, we assume arrays s_curr and s_next , indexed by V , such that initially $s_curr[v] = s_0(v)$ and $s_next[v] = s_1(v)$ for all $v \in V$.

```

procedure RUN( $G, o, d$ )
   $v := o$ 
  while  $v \neq d$  do
     $w := s\_curr[v]$ 
    swap( $s\_curr[v], s\_next[v]$ )
     $v := w$  ▷ traverse edge  $(v, w)$ 
  end while
end procedure

```

Definition 1 Problem *ARRIVAL* is to decide whether procedure $RUN(G, o, d)$ terminates for a given switch graph $G = (V, E, s_0, s_1)$ and $o, d \in V$.

Theorem 1 Problem *ARRIVAL* is decidable.

Proof The deterministic procedure RUN can be interpreted as a function that maps the current state (v, s_curr, s_next) to the next state. We can think of the state as the current location of the train, and the current positions of all the switches. As at most $n2^n$ different states may occur, RUN either terminates within this many iterations, or some state repeats, in which case RUN enters an infinite loop. Hence, to decide *ARRIVAL*, we have to go through at most $n2^n$ iterations of RUN . \square

Figure 1 shows that a terminating run may indeed take exponential time.

Existing research on switch graphs (with the above, or similar definitions) has mostly focused on actively controlling the switches, with the goal of attaining some desired behavior of the network (e.g. reachability of the destination); see e.g. [5]. The question we address here rather fits into the theory of cellular automata. It is motivated by the online game *Looping Piggy* (<https://scratch.mit.edu/projects/1200078/>) that the second author has written for the *Kinderlabor*, a

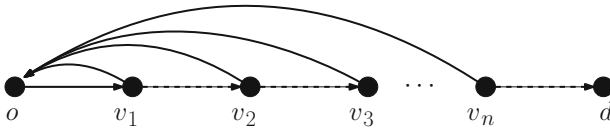


Fig. 1 Switch graph G with $n+2$ vertices on which $RUN(G, o, d)$ traverses an exponential number of edges. If we encode the current positions of the switches at v_n, \dots, v_1 with an n -bit binary number (0: even successor is next; 1: odd successor is next), then the run counts from 0 to $2^n - 1$, resets the counter to 0, and terminates. *Solid edges* point to even or unique successors, *dashed edges* to odd successors

Swiss-based initiative to educate children at the ages 4–12 in natural sciences and computer science (<http://kinderlabor.ch>).

In Sects. 2 and 3, we prove that ARRIVAL is in NP as well as in coNP; Sect. 4 shows that a terminating run can be interpreted as the unique solution of a flow-type integer program with balancing conditions whose LP relaxation may have only fractional optimal solutions.

2 ARRIVAL Is in NP

A natural candidate for an NP-certificate is the *run profile* of a terminating run. The run profile assigns to each edge the number of times it has been traversed during the run. The main difficulty is to show that fake run profiles cannot fool the verifier. We start with a necessary condition for a run profile: it has to be a *switching flow*.

Definition 2 Let $G = (V, E, s_0, s_1)$ be a switch graph, and let $o, d \in V$, $o \neq d$. A *switching flow* is a function $\mathbf{x} : E \rightarrow \mathbb{N}_0$ (where $\mathbf{x}(e)$ is denoted as x_e) such that the following two conditions hold for all $v \in V$.

$$\sum_{e \in E^+(v)} x_e - \sum_{e \in E^-(v)} x_e = \begin{cases} 1, & v = o, \\ -1, & v = d, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

$$0 \leq x_{(v, s_1(v))} \leq x_{(v, s_0(v))} \leq x_{(v, s_1(v))} + 1. \quad (2)$$

Observation 1 Let $G = (V, E, s_0, s_1)$ be a switch graph, and let $o, d \in V$, $o \neq d$, such that $\text{RUN}(G, o, d)$ terminates. Let $\mathbf{x}(G, o, d) : E \rightarrow \mathbb{N}_0$ (the run profile) be the function that assigns to each edge the number of times it has been traversed during $\text{RUN}(G, o, d)$. Then $\mathbf{x}(G, o, d)$ is a switching flow.

Proof Condition (1) is simply flow conservation (if the run enters a vertex, it has to leave it, except at o and d), while (2) follows from the run alternating between successors at any vertex v , with the even successor $s_0(v)$ being first. \square

While every run profile is a switching flow, the converse is not always true. Figure 2 shows two switching flows for the same switch graph, but only one of them

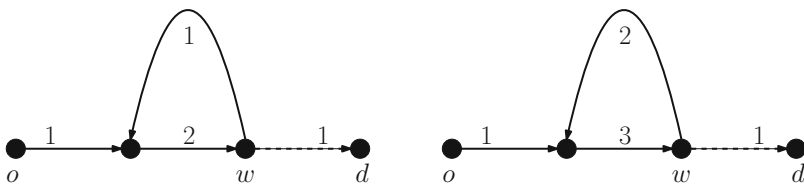


Fig. 2 Run profile (left) and fake run profile (right); both are switching flows. Solid edges point to even or unique successors, dashed edges to odd successors

is the actual run profile. The “fake” run results from going to the even successor of w twice in a row, before going to the odd successor d . This shows that the balancing condition (2) fails to capture the strict alternation between even and odd successors. Despite this, and maybe surprisingly, the existence of a switching flow implies termination of the run.

Lemma 1 *Let $G = (V, E, s_0, s_1)$ be a switch graph, and let $o, d \in V$, $o \neq d$. If there exists a switching flow \mathbf{x} , then $\text{RUN}(G, o, d)$ terminates, and $\mathbf{x}(G, o, d) \leq \mathbf{x}$ (componentwise).*

Proof We imagine that for all $e \in E$ we put x_e pebbles on edge e , and then start $\text{RUN}(G, o, d)$. Every time an edge is traversed, we let the run collect one pebble. The claim is that we never run out of pebbles, which proves termination as well as the inequality for the run profile.

To prove the claim, we first observe two invariants: during the run, flow conservation (w.r.t. to the remaining pebbles) always holds, except at d , and at the current vertex which has one more pebble on its outgoing edges. Moreover, by alternation, starting with the even successor, the numbers of pebbles on $(v, s_0(v))$ and $(v, s_1(v))$ always differ by at most one, for every vertex v .

For contradiction, consider now the first iteration of $\text{RUN}(G, o, d)$ where we run out of pebbles, and let $e = (v, w)$ be the edge (now holding -1 pebbles) traversed in the offending iteration. By the above alternation invariant, the other outgoing edge at v cannot have any pebbles left, either. Then the flow conservation invariant at v shows that already some incoming edge of v has a deficit of pebbles, so we have run out of pebbles before, which is a contradiction. \square

Theorem 2 *Problem ARRIVAL is in NP.*

Proof Given an instance (G, o, d) , the verifier receives a function $\mathbf{x} : E \rightarrow \mathbb{N}_0$, in form of binary encodings of the values x_e , and checks whether it is a switching flow. For a Yes-instance, the run profile of $\text{RUN}(G, o, d)$ is a witness by Observation 1; the proof of Theorem 1 implies that the verification can be made to run in polynomial time, since every value x_e is bounded by $n2^n$. For a No-instance, the check will fail by Lemma 1. \square

3 ARRIVAL Is in coNP

Given an instance (G, o, d) of ARRIVAL, the main idea is to construct in polynomial time an instance $(\tilde{G}, o, \tilde{d})$ such that $\text{RUN}(G, o, d)$ terminates if and only if $\text{RUN}(\tilde{G}, o, \tilde{d})$ does not terminate. As the main technical tool, we prove that nontermination is equivalent to the arrival at a “dead end”.

Definition 3 Let $G = (V, E, s_0, s_1)$ be a switch graph, and let $o, d \in V$, $o \neq d$. A *dead end* is a vertex from which there is no directed path to the destination d in the graph (V, E) . A *dead edge* is an edge $e = (v, w)$ whose head w is a dead end.

An edge that is not dead is called *hopeful*; the length of the shortest directed path from its head w to d is called its *desperation*.

By computing the tree of shortest paths to d , using inverse breadth-first search from d , we can identify the dead ends in polynomial time. Obviously, if $\text{RUN}(G, o, d)$ ever reaches a dead end, it will not terminate, but the converse is also true. For this, we need one auxiliary result.

Lemma 2 *Let $G = (V, E, s_0, s_1)$ be a switch graph, $o, d \in V$, $o \neq d$, and let $e = (v, w) \in E$ be a hopeful edge of desperation k . Then $\text{RUN}(G, o, d)$ will traverse e at most $2^{k+1} - 1$ times.*

Proof Induction on the desperation k of $e = (v, w)$. If $k = 0$, then $w = d$, and indeed, the run will traverse e at most $2^1 - 1 = 1$ times. Now suppose $k > 0$ and assume that the statement is true for all hopeful edges of desperation $k - 1$. In particular, one of the two successor edges $(w, s_0(w))$ and $(w, s_1(w))$ is such a hopeful edge, and is therefore traversed at most $2^k - 1$ times. By alternation at w , the other successor edge is traversed at most once more, hence at most 2^k times. By flow conservation, the edges entering w (in particular e) can be traversed at most $2^k + 2^k - 1 = 2^{k+1} - 1$ times. \square

Lemma 3 *Let $G = (V, E, s_0, s_1)$ be a switch graph, and let $o, d \in V$, $o \neq d$. If $\text{RUN}(G, o, d)$ does not terminate, it will reach a dead end.*

Proof By Lemma 2, hopeful edges can be traversed only finitely many times, hence if the run cycles, it eventually has to traverse a dead edge and thus reach a dead end. \square

Now we can prove the main result of this section.

Theorem 3 *Problem ARRIVAL is in coNP.*

Proof Let (G, o, d) be an instance, $G = (V, E, s_0, s_1)$. We transform (G, o, d) into a new instance (\bar{G}, o, \bar{d}) , $\bar{G} = (\bar{V}, \bar{E}, \bar{s}_0, \bar{s}_1)$ as follows. We set $\bar{V} = V \cup \{\bar{d}\}$, where \bar{d} is an additional vertex, the new destination. We define \bar{s}_0, \bar{s}_1 as follows. For every dead end w , we set

$$\bar{s}_0(w) = \bar{s}_1(w) := \bar{d}. \quad (3)$$

For the old destination d , we install the loop

$$\bar{s}_0(d) = \bar{s}_1(d) := d. \quad (4)$$

For the new destination, $\bar{s}_0(\bar{d})$ and $\bar{s}_1(\bar{d})$ are chosen arbitrarily. In all other cases, $\bar{s}_0(v) := s_0(v)$ and $\bar{s}_1(v) := s_1(v)$. This defines \bar{E} and hence \bar{G} .

The crucial properties of this construction are the following:

- (i) If $\text{RUN}(G, o, d)$ reaches the destination d , it has not visited any dead ends, hence s_0 and \bar{s}_0 as well as s_1 and \bar{s}_1 agree on all visited vertices except d . This means

that $\text{RUN}(\bar{G}, o, \bar{d})$ will also reach d , but then cycle due to the loop that we have installed in (4).

- (ii) If $\text{RUN}(G, o, d)$ cycles, it will at some point reach a first dead end w , by Lemma 3. As s_0 and \bar{s}_0 as well as s_1 and \bar{s}_1 agree on all previously visited vertices, $\text{RUN}(\bar{G}, o, \bar{d})$ will also reach w , but then terminate due to the edges from w to \bar{d} that we have installed in (3).

To summarize, $\text{RUN}(G, o, d)$ terminates if and only if $\text{RUN}(\bar{G}, o, \bar{d})$ does not terminate. Since (\bar{G}, o, \bar{d}) can be constructed in polynomial time, we can verify in polynomial time that (G, o, d) is a No-instance by verifying that (\bar{G}, o, \bar{d}) is a Yes-Instance via Theorem 2. \square

4 Is ARRIVAL in P?

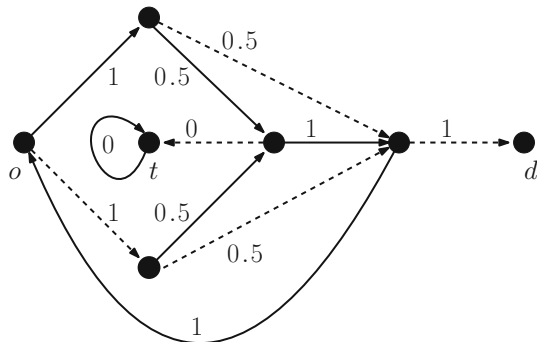
Observation 1 and Lemma 1 show that ARRIVAL can be decided by checking the solvability of a system of linear (in)equalities (1) and (2) over the nonnegative integers.

The latter is an NP-complete problem in general: many of the standard NP-complete problems, e.g. SAT (satisfiability of boolean formulas) can easily be reduced to finding an integral vector that satisfies a system of linear (in)equalities.

In our case, we have a flow structure, though, and finding integral flows in a network is a well-studied and easy problem [6, Chapter 8]. In particular, if only the flow conservation constraints (1) are taken into account, the existence of a nonnegative integral solution is equivalent to the existence of a nonnegative real solution. This follows from the classical *Integral Flow Theorem*, see [6, Corollary 8.7]. Real solutions to systems of linear (in)equalities can be found in polynomial time through linear programming [6, Chapter 4].

However, the additional balancing constraints (2) induced by alternation at the switches, make the situation more complicated. Figure 3 depicts an instance which has a real-valued “switching flow” satisfying constraints (1) and (2), but no integral one (since the run does not terminate).

Fig. 3 The run will enter the loop at t and cycle, so there is no (integral) switching flow. But a real-valued “switching flow” (given by the numbers) exists. *Solid edges* point to even or unique successors, *dashed edges* to odd successors



We conclude with a result that summarizes the situation and may be the basis for further investigations.

Theorem 4 *Let $G = (V, E, s_0, s_1)$ be a switch graph, and let $o, d \in V$, $o \neq d$. $\text{RUN}(G, o, d)$ terminates if and only if there exists an integral solution satisfying the constraints (1) and (2). In this case, the run profile $\mathbf{x}(G, o, d)$ is the unique integral solution that minimizes the linear objective function $\Sigma(\mathbf{x}) = \sum_{e \in E} x_e$ subject to the constraints (1) and (2).*

Proof Observation 1 and Lemma 1 show the equivalence between termination and existence of an integral solution (a switching flow). Suppose that the run terminates with run profile $\mathbf{x}(G, o, d)$. We have $\mathbf{x}(G, o, d) \leq \mathbf{x}$ for every switching flow \mathbf{x} , by Lemma 1. In particular, $\Sigma(\mathbf{x}(G, o, d)) \leq \Sigma(\mathbf{x})$, so the run profile has minimum value among all switching flows. A different switching flow \mathbf{x} of the same value would have to be smaller in at least one coordinate, contradicting $\mathbf{x}(G, o, d) \leq \mathbf{x}$. \square

Theorem 4 shows that the existence of $\mathbf{x}(G, o, d)$ and its value can be established by solving an integer program [6, Chapter 5]. Moreover, this integer program is of a special kind: its unique optimal solution is at the same time a least element w.r.t. the partial order “ \leq ” over the set of feasible solutions.

5 Conclusion

The main question left open is whether the zero-player graph game ARRIVAL is in P. There are three well-known two-player graph games in $\text{NP} \cap \text{coNP}$ for which membership in P is also not established: *simple stochastic games*, *parity games*, and *mean-payoff games*. All three are even in $\text{UP} \cap \text{coUP}$, meaning that there exist efficient verifiers for Yes- and No-instances that accept *unique* certificates [1, 3]. In all three cases, the way to prove this is to assign payoffs to the vertices in such a way that they form a certificate if and only if they solve a system of equations with a unique solution.

It is natural to ask whether also ARRIVAL is in $\text{UP} \cap \text{coUP}$. We do not know the answer. The natural approach suggested by Theorem 4 is to come up with a verifier that does not accept just any switching flow, but only the unique one of minimum norm corresponding to the run profile. However, verifying optimality of a feasible integer program solution is hard in general, so for this approach to work, one would have to exploit specific structure of the integer program at hand. We do not know how to do this.

As problems in $\text{NP} \cap \text{coNP}$ cannot be NP-hard (unless NP and coNP collapse), other concepts of hardness could be considered for ARRIVAL. As a first step in this direction, Karthik [4] has shown that a natural search version of ARRIVAL is contained in the complexity class PLS (Polynomial Local Search) which has complete problems not known to be solvable in polynomial time. PLS-hardness of

ARRIVAL would not contradict common complexity theoretic beliefs; establishing such a hardness result would at least provide a satisfactory explanation why we have not been able to find a polynomial-time algorithm for ARRIVAL.

Acknowledgements We thank the referees for valuable comments and Rico Zenklusen for constructive discussions.

References

1. A. Condon, The complexity of stochastic games. *Inf. Comput.* **96**(2), 203–224 (1992)
2. N. Halman, Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica* **49**(1), 37–50 (2007)
3. M. Jurdziński, Deciding the winner in parity games is in $UP \cap co-UP$. *Inf. Process. Lett.* **68**(3), 119–124 (1998)
4. C.S. Karthik, Did the train reach its destination: the complexity of finding a witness (2016). <https://arxiv.org/abs/1609.03840>
5. B. Katz, I. Rutter, G. Woeginger, An algorithmic study of switch graphs. *Acta Inf.* **49**(5), 295–312 (2012)
6. B. Korte, J. Vygen, *Combinatorial Optimization: Theory and Algorithms*, 5th edn. (Springer, Berlin, 2012)