Sinks and Ladders: ARRIVAL and SSG with Two Vertices per Level

Bernd Gärtner
Department of Computer Science, ETH Zürich, Switzerland

Sebastian Haslebacher
Department of Computer Science, ETH Zürich, Switzerland

Hung P. Hoang
Algorithms and Complexity Group, Faculty of Informatics, TU Wien, Austria

Abstract

ARRIVAL is the problem of deciding whether a token, following a deterministic process, eventually reaches a designated destination. While the problem is known to lie in $\mathsf{NP} \cap \mathsf{CoNP}$, whether it can be solved in polynomial time remains a major open question. In this article, we study *ladders*, a class of graphs that constitutes a family of worst-case instances for many existing algorithms, including the currently best known algorithm by Gärtner, Haslebacher, and Hoang (ICALP 2021). We show that ARRIVAL restricted to ladders can be solved in polynomial time, and we further extend this result to stopping binary simple stochastic games (SSG).

2012 ACM Subject Classification Theory of computation → Graph algorithms analysis

Keywords and phrases ARRIVAL, Rotor-Routing, Simple Stochastic Games

Digital Object Identifier 10.4230/LIPIcs.FUN.2026.19

Funding *Hung P. Hoang*: Austrian Science Foundation (FWF, project ESP1136425).

Acknowledgements We are grateful to three anonymous referees for their helpful comments.

1 Introduction

ARRIVAL is the following decision problem, first introduced by Dohrau, Gärtner, Kohler, Matoušek, and Welzl [7]. As the input, we are given a directed graph G with a designated origin $\mathbf{o}$ and two designated destinations, $\mathbf{1}$ and $\mathbf{0}$.[1] Every other vertex has two outgoing edges, with one edge marked as *even edge* and the other as *odd edge*. We place a token on $\mathbf{o}$ and move it according to the following deterministic rule: At every vertex, if the token has visited it an even number of times, the token traverses the even edge; otherwise, it takes the odd edge. The task is to decide whether the token eventually reaches $\mathbf{1}$. See Figure 1 for an example.

This simple deterministic process has appeared in many contexts. Most notable is the study of deterministic simulations of a random walk under different names, such as *Eulerian walkers* [16], *rotor-router walks* [15], and *Propp machines* [6]. It also naturally arises from combinatorial games. The introduction of ARRIVAL problem by [7] was inspired by an online game called *Looping Piggy* developed by Gärtner for *Kinderlabor*, an initiative for computer science education in kindergartens. The latest reporting of a similar problem is in a blog post by Aaronson [1], also motivated by a children's game. As the author pointed out in the post, this in turn is a rediscovery of a result back in 1994 [4].

This article is partly based on an unpublished work in the thesis of the last author [14].

[1] In the literature, the two sinks are often called d and $\bar{d}$. However, in this paper, as we want to emphasize a parallel between ARRIVAL and simple stochastic games, we call them $\mathbf{1}$ and $\mathbf{0}$ instead.

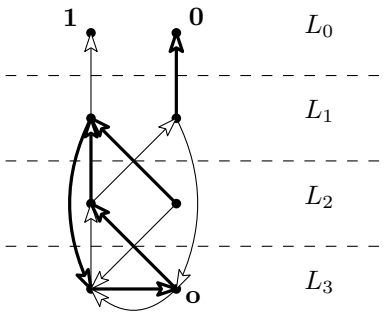

Figure 1 An ARRIVAL instance on ladder. Bold edges are even edges. The vertices are also grouped by level, where L_i are vertices with distance i from the destinations.

Arguably one of the main attractions of ARRIVAL is its curious complexity status. It is shown in [7] that the problem lies in $\mathsf{NP} \cap \mathsf{CoNP}$. Subsequent papers have established more specific results, such as the membership in $\mathsf{UP} \cap \mathsf{CoUP}$ [10] and NL-hardness [8] as well as containment in subclasses of TFNP [10, 9], but one key question remains wide open: Is ARRIVAL in P?

Two common approaches to tackle this question are finding a faster algorithm for a general instance and expanding the classes of graphs where a polynomial-time algorithm for ARRIVAL exists. For the first approach, the best algorithm so far runs in subexponential time $2^{\mathcal{O}(\sqrt{n}\log n)}$ [11], where n is the number of non-sinks in the graph. For the second approach, there have been (quasi-)polynomial-time algorithms on tree-like multigraphs [2, 12], path-like multigraphs [3], and bounded treewidth [13]. Notably, most of these results also extend to a more generalized variant with multiple tokens.

Contributions. This paper contributes towards the second approach above. In particular, we consider the class of graphs where for every $i \in \mathbb{N}$, there are either no or exactly two vertices of distance i to the destinations (i.e., the length of the shortest path to either **0** or **1** is i); see Figure 1 for an example. These graphs, which we call *ladders*, capture one of the worst cases for the current state-of-the-art subexponential algorithm [11] and also for another deterministic algorithm, which we elaborate in Section 3. It turns out that ladders have an interesting structural property, expressed in terms of hitting properties of a Markov chain, which we can exploit to design a polynomial-time algorithm for ARRIVAL; see Section 5. Note that the technique used for this case is different than most of the existing algorithms for ARRIVAL, as we do not simulate the movement of the token in any way. Instead, we progressively and simultaneously construct a witness for a YES-instance and a witness for a NO-instance, and whichever fails first helps us decide the instance.

Moreover, ARRIVAL has a very close connection to Simple Stochastic Games (SSG), a famous problem that shares the same complexity status: It is also in $\mathsf{NP} \cap \mathsf{CoNP}$ but not known to be in P. While there has been no reduction between these two problems, they share many parallels; see [13] for a recent discussion. Due to this connection, we also explore how SSG can be solved on ladders. Using an analogous observation on the hitting probabilities, we can derive a simple polynomial-time algorithm for SSG on ladders; see Section 4.

2 Preliminaries

A *binary graph* is a directed simple graph $G = (V, E)$ such that there are exactly two *sinks* $\mathbf{1}$ and $\mathbf{0}$ (i.e., these two vertices have no outgoing edges). All other vertices have exactly two outgoing edges. We also assume that it is possible to reach at least one of the two sinks starting at any vertex. In this paper, we define $n := |V| - 2$; that is n is the number of non-sinks in the graph.

For a vertex $v \in V$, denote by $d(v)$ the distance of v to either of the sinks, i.e. $d(v)$ is the length of a shortest path (number of edges) in the graph G from v to either $\mathbf{1}$ or $\mathbf{0}$. In particular, we have $d(\mathbf{1}) = d(\mathbf{0}) = 0$. For $i \in \mathbb{N}$, let $L_i := \{v \in V : d(v) = i\}$ denote the set of vertices of distance exactly i from the sinks. We call L_i the *i-th levelset* and call $(L_0, L_1, \dots)$ the *level decomposition* of G. Observe that we can compute the level decomposition of G in linear time by inverting all edge directions and starting a multi-source breadth-first search with sources $\mathbf{1}$ and $\mathbf{0}$.

An edge (u, v) is a *forward edge* if v is closer to the sinks than u (i.e., u in L_i and v in L_{i-1} for some i). Otherwise, we call (u, v) a *backward edge*. Note that a backward edge may connect two vertices in the same levelset.

A binary graph G is a *ladder*, if every non-empty levelset of G consists of exactly two vertices, except for the last non-empty levelset which may consist of a single vertex. In other words, let $(L_0, L_1, \dots)$ be the level decomposition of G; then G is a ladder, if $|L_i| \leq 2$ for all $i \in \mathbb{N}$ and $|L_i| < 2$ implies $|L_j| = 0$ for all $i, j \in \mathbb{N}$ with $i < j$.

2.1 ARRIVAL

For a binary graph $G = (V, E)$, a partition (E_0, E_1) of the edge set E is *balanced*, if every vertex $v \in V \setminus \{\mathbf{0}, \mathbf{1}\}$ has exactly one outgoing edge in E_0 and one outgoing edge in E_1. We call these edges the *even edge* and the *odd edge* of v and denote them by $(v, s_0(v))$ and $(v, s_1(v))$, respectively. Then an instance of ARRIVAL is defined formally as a tuple of a binary graph $G = (V, E)$, a designated source vertex $\mathbf{o} \in V$, and a balanced partition of E into E_0 and E_1.

Given such an instance of ARRIVAL, consider moving a token along the edges of the graph according to the following rules: Whenever the token reaches a vertex v, it follows the even edge of v if it has been at v an even number of times already. Otherwise, it follows the odd edge of v. This results in the token effectively switching between the two outgoing edges at every vertex. The process finishes once the token reaches one of the two sinks. The problem is to decide which of the two sinks the token will end up at.

2.2 (Binary) SSG

Following roughly the definition given by Condon [5], a (binary) Simple Stochastic Game (SSG) is played on a binary graph $G = (V, E)$ with $V = V_{\max} \sqcup V_{\min} \sqcup V_{\text{avg}} \sqcup \{\mathbf{0}, \mathbf{1}\}$. Moreover, one of the vertices $\mathbf{o} \in V \setminus \{\mathbf{0}, \mathbf{1}\}$ is considered the designated source.

The game works as follows: For every vertex $v \in V_{\max}$, the so-called max-player chooses one of the outgoing edges of v. Next, the min-player chooses one outgoing edge for every $v \in V_{\min}$. A token is then placed on $\mathbf{o}$ and starts moving along the directed edges of the graph. Whenever the token reaches a vertex $v \in V_{\max}$, the token follows the outgoing edge that was chosen by the max-player for this vertex. Analogously, the edges chosen by the min-player determine where the token moves to from vertices in $V_{\min}$. When the token reaches an average vertex $v \in V_{\text{avg}}$, an outgoing edge of v is chosen uniformly at random

and the token continues along this edge. The game ends when the token reaches either of the sinks **0** or **1**. The max-player wins if the token eventually reaches **1**. If the token never reaches **1**, the min-player wins. This can happen if the token reaches **0** instead or if it continues traversing the graph indefinitely without ever reaching any of the sinks.

A *(positional) strategy* σ for the max-player is a function $\sigma : V_{\max} \to E$ that assigns each vertex $v \in V_{\max}$ one of its outgoing edges. If the max-player plays according to the strategy σ, this means that whenever the token reaches a vertex $v \in V_{\max}$, it will continue along the outgoing edge $\sigma(v)$. Analogously, a strategy τ for the min-player is a function $\tau : V_{\min} \to E$ that assigns to each vertex in $V_{\min}$ one of its outgoing edges.

We denote by $G_{\sigma,\tau}$ the graph that we obtain if we delete outgoing edges from vertices in $V_{\max} \cup V_{\min}$ that are not used by σ and τ. In other words, $G_{\sigma,\tau}$ is the graph restricted to the strategies σ, τ. This can be thought of as a Markov chain: Vertices in $V_{\max} \cup V_{\min}$ have exactly one outgoing edge that is taken with probability 1, vertices in V_{avg} have two outgoing edges each with a transition probability of $\frac{1}{2}$. Given strategies σ, τ and a vertex $v \in V$, the *value* (or *hitting probability*) $h_{\sigma,\tau}(v)$ is the probability that a token starting at vertex v will eventually hit **1** in the Markov chain $G_{\sigma,\tau}$. In other words, this is the probability of the max-player winning the game if the players play according to σ and τ and the token starts at v. The optimal value $h(v)$ of a vertex $v \in V$ is defined as $h(v) \coloneqq \max_\sigma \min_\tau h_{\sigma,\tau}(v)$.

The computational problem SSG is to decide whether $h(\mathbf{o}) > \frac{1}{2}$ or not, where $h(\mathbf{o})$ is the optimal value of the designated source.

An SSG is called *stopping* if and only if a sink is reachable from every vertex in the graph $G_{\sigma,\tau}$ for all strategies σ and τ of the players. In other words, in a stopping game, the probability of the token looping indefinitely in the graph without ever reaching a sink is zero, regardless of the played strategies. There is a well-known trick that reduces SSGs to stopping SSGs [5]. However, using it for SSGs on ladders would destroy the ladder structure of the underlying graph. Hence, in this article, we only consider stopping SSGs on ladders.

3 Prelude: An $\mathcal{O}(2^{n/2})$-time algorithm for ARRIVAL

As a motivation for studying ladders, consider the following simple algorithm for ARRIVAL that runs in time $\mathcal{O}(2^{n/2})$. Note that this algorithm has been discovered independently by Rote [17], who also bootstrapped it into a $\mathcal{O}(2^{n/3})$-time algorithm.

Suppose we are given an instance of ARRIVAL consisting of the binary graph G, $\mathbf{o} \in V$, and a balanced partition (E_0, E_1) of the edge set. Let $(L_0, L_1, \dots)$ be the level decomposition of G. Now assume there is a vertex w in some levelset L_k such that all forward edges from L_{k+1} (i.e., all edges from a vertex in L_{k+1} to a vertex in L_k) have w as the target. Let G' be the graph obtained from G by contracting all vertices in $\bigcup_{j>k} L_j$ to w; that is, we replace every edge (u, v) such that $d(v) > k$ by (u, w) and then remove all vertices in $\bigcup_{j>k} L_j$ and their outgoing edges. Let $\mathbf{o}'$ be w if $d(\mathbf{o}) > k$ and be $\mathbf{o}$ otherwise. Finally, let (E'_0, E'_1) be the balanced partition of $E(G')$ obtained from (E_0, E_1) through the process of replacement and deletion above.

▶ **Lemma 1.** *$(G, \mathbf{o}, E_0, E_1)$ is a YES ARRIVAL instance, if and only if $(G', \mathbf{o}', E'_0, E'_1)$ is also a YES ARRIVAL instance.*

Proof. Let S and S' be the sequences of vertices that the token visits for $(G, \mathbf{o}, E_0, E_1)$ and $(G', \mathbf{o}', E'_0, E'_1)$, respectively. We prove by induction the claim that every prefix of S' can be obtained from some prefix of S by removing vertices in $\bigcup_{j>k} L_j$. For the base case, suppose $d(\mathbf{o}) > k$. Observe that the token only terminates when it visits a vertex in L_0 (i.e., **1** or **0**);

and by the construction of the level decomposition, after visiting u, the token has to visit at least one vertex in each of the levelsets between $L_{d(u)}$ and L_0. This implies that it has to visit w, and until the first time it visits w after this visit at u, the token only visits vertices in layers L_j for $j > k$. Hence, S contains a prefix that ends with w and otherwise contains only vertices in $\bigcup_{j>k} L_j$. Since $\mathbf{o}' = w$ when $d(\mathbf{o}) > k$, the claim then holds for the prefix $(\mathbf{o}')$ of S'. Note that it trivially holds for this prefix when $d(\mathbf{o}) \leq k$, since in this case $\mathbf{o} = \mathbf{o}'$.

For the inductive step, suppose for some prefix P' of S', we have the corresponding prefix P of S as guaranteed by the claim. Note that we can choose P such that P and P' end at the same vertex v. Suppose the vertex after P in S is u. If $d(u) \leq k$, then the vertex after P' in S' is also u, since at this point, the token should have visited v the same number of times in both G and G'. Otherwise, the token visits w next in P', since u should have been contracted to w when we constructed G' from G. By a similar argument as above, we also conclude that after P in S, there is a substring that ends with w and otherwise contains only vertices in $\bigcup_{j>k} L_j$. Concatenating this substring to P, we obtain a prefix in S corresponding to the prefix resulting from the concatenation of P' and w in S', as required by the claim. This completes the proof of the claim.

Since **0** and **1** are not contracted, the lemma then follows from the claim above. ◀

We are now ready to describe our algorithm: If there exists a levelset with only one vertex, let k be the smallest index such that $|L_k| = 1$, and let w be the only vertex in layer L_k. Let $(G', \mathbf{o}', E'_0, E'_1)$ be the graph obtained by performing the contractions described above. If there exists no such levelset, then we define $(G', \mathbf{o}', E'_0, E'_1)$ to be the same as the original instance. Next, we simulate the movement of the token in G' until it reaches **0** or **1**. By Lemma 1, we can then decide the original instance accordingly.

As shown in [11], the number of steps the token takes is $\mathcal{O}(2^k)$, since k is the highest index of a nonempty levelset for G'. By the choice of k, we obtain that $k \leq \lceil n/2 \rceil$, and it attains the maximum, when there are exactly two vertices per non-empty levelset, except for potentially the last such levelset (i.e., when the graph is a ladder). In other words, the ladders capture a class of "hard" instances for the algorithm above (i.e., instances for which the algorithm runs the longest). Interestingly, they are also hard instances for the current state-of-the-art algorithms for ARRIVAL [11].

4 Stopping SSG on ladders in polynomial time

Since the arguments for SSG is simpler than for ARRIVAL, we present first the polynomial-time algorithm for stopping SSGs on ladders in this section.

▶ **Theorem 2.** *Stopping SSG can be solved in polynomial time on ladders.*

Our algorithm roughly works as follows: We start at the levelset L_0. It contains the two sinks **0** and **1** and we know that $h(\mathbf{0}) = 0 < 1 = h(\mathbf{1})$. Now consider the next levelset L_1 and assume it contains the two vertices u and v. We claim that by looking at forward edges of u and v, and by considering which types of vertices u and v are (max, min, or avg), we can figure out whether $h(u) < h(v)$ or not. Moreover, we can repeat this for the other levelsets. Combined with a further observation, we can deduce a total order on the optimal values of all vertices which allows us to solve the SSG.

For the rest of this section, assume that we are given a stopping SSG on a ladder as defined in Section 2 with $k \in \mathbb{N}$ being the smallest number such that $|L_k| < 2$. In particular, this implies $|L_i| = 2$ for all $i \in \mathbb{N}$ with $i < k$, and $|L_i| = 0$ for all $i \in \mathbb{N}$ with $i > k$. By adding up to two new vertices (e.g. to V_{avg}) with outgoing edges to vertices from L_{k-1}, we

also get $|L_k| = 2$. This does not change the game since these new vertices have no incoming edges and will never be visited by the token. Hence, from now on we assume $|L_i| = 2$ for all $i \in \{0, 1, \ldots, k\}$.

4.1 Optimal values within levelsets

The first ingredient for our algorithm is to find out which of the two vertices in each levelset has larger optimal value. For this, consider the following definition.

▶ **Definition 3.** *For every $i \in \{0, 1, \ldots, k\}$, define $\ell_i \in L_i$ to be a vertex $\ell_i \in \operatorname{argmin}_{v \in L_i} h(v)$, and define $u_i \in L_i$ such that $u_i \neq \ell_i$, i.e. u_i is the other vertex in L_i. Note that this implies $h(u_i) = \max_{v \in L_i} h(v)$.*

In particular, we have $\ell_0 = \mathbf{0}$ and $u_0 = \mathbf{1}$. The following observation will be crucial.

▶ **Lemma 4.** *For every $i \in \{1, \ldots, k\}$, we have $h(\ell_{i-1}) \leq h(\ell_i) \leq h(u_i) \leq h(u_{i-1})$.*

Proof. Let $i \in \{1, \ldots, k\}$ be arbitrary and assume σ, τ are optimal strategies. In particular, we have $h_{\sigma,\tau}(v) = h(v)$ for all $v \in V$. Assume the token is currently at ℓ_i and recall that we assume that the game is stopping. Consider any walk of the token through the graph eventually ending at a sink. The walk has to pass through ℓ_{i-1} or u_{i-1} which implies that $h(\ell_i)$ is a convex combination of $h(\ell_{i-1})$ and $h(u_{i-1})$. The same argument works for u_i (and in fact all vertices in levelsets $L_i, \ldots, L_k$). ◀

The goal of this subsection hence becomes determining ℓ_i and u_i for each $i \in \{0, 1, \ldots, k\}$. As we have observed, we already know this for $i = 0$.

We process the remaining levelsets iteratively going from $i = 1$ up to $i = k$. Assume that we are in iteration i and hence know ℓ_{i-1} and u_{i-1}. We are trying to determine ℓ_i and u_i.

▶ **Lemma 5** (one max-vertex)**.** *Assume there is a vertex $v \in L_i \cap V_{max}$, and let w denote the other vertex in L_i. Then $h(w) \leq h(v)$.*

Proof. We distinguish two cases: assume first that v has a forward edge with target u_{i-1}. The best strategy for the max-player is to use this edge since it will yield $h(v) = h(u_{i-1})$ which is best possible according to Lemma 4. We conclude that $h(w) \leq h(u_{i-1}) = h(v)$, as desired.

Thus, assume now that v has no forward edge to u_{i-1}. Then the outgoing edges of v are (v, ℓ_{i-1}) and (v, z) with $z \in L_j$ for some $j \geq i$. By Lemma 4, we have $h(z) \geq h(\ell_{i-1})$ and hence it is optimal for the max-player to choose the edge (v, z). This again means that under optimal strategies, a token starting at z needs to go through w in order to get to a sink. Hence, we get $h(z) = h(w) = h(v)$. ◀

Analogously, we can prove the following lemma.

▶ **Lemma 6** (one min-vertex)**.** *Assume there is a vertex $v \in L_i \cap V_{min}$, and let w denote the other vertex in L_i. Then $h(v) \leq h(w)$.*

Proof. We again distinguish two cases: assume first that v has a forward edge with target ℓ_{i-1}. The best strategy for the min-player is to use this edge since it will yield $h(v) = h(\ell_{i-1})$ which is best possible according to Lemma 4. We conclude that $h(w) \geq h(\ell_{i-1}) = h(v)$, as desired.

Thus, assume now that v has no forward edge to ℓ_{i-1}. Then the outgoing edges of v are (v, u_{i-1}) and (v, z) with $z \in L_j$ for some $j \geq i$. By Lemma 4, we have $h(z) \leq h(u_{i-1})$ and hence it is optimal for the min-player to choose the edge (v, z). This again means that under optimal strategies, a token starting at z needs to go through w in order to get to a sink. Hence, we get $h(z) = h(w) = h(v)$. ◀

Finally, we consider the case where both vertices are average vertices in Lemma 7 and 8.

▶ **Lemma 7** (average vertices I). *Assume that the two vertices $v, w \in L_i$ with $v \neq w$ are both average vertices, i.e. $v, w \in V_{avg}$. If the edge (v, ℓ_{i-1}) exists but the edge (v, u_{i-1}) does not exist, then we have $h(v) \leq h(w)$.*

Proof. Assume first that the edge (w, u_{i-1}) exists, and let (v, z) and (w, y) be the remaining outgoing edges of v and w. Observe that both z and y are in one of the levelsets $L_{i-1}, \dots, L_k$ and hence we have $h(\ell_{i-1}) \leq h(z) \leq h(u_{i-1})$ and $h(\ell_{i-1}) \leq h(y) \leq h(u_{i-1})$. We conclude that indeed $h(v) = \frac{1}{2}h(\ell_{i-1}) + \frac{1}{2}h(z) \leq \frac{1}{2}h(y) + \frac{1}{2}h(u_{i-1}) = h(w)$.

Thus, assume now that (w, u_{i-1}) does not exist. Then the edge (w, ℓ_{i-1}) must exist instead. Let (v, z) and (w, y) be the remaining outgoing edges of v and w. Observe that all paths from z, y, v, w to any of the sinks must go through ℓ_{i-1}. Hence, we get $h(y) = h(z) = h(v) = h(w) = h(\ell_{i-1})$. ◀

▶ **Lemma 8** (average vertices II). *Assume that the two vertices $v, w \in L_i$ with $v \neq w$ are both average vertices, i.e. $v, w \in V_{avg}$. If the edge (v, u_{i-1}) exists but the edge (v, ℓ_{i-1}) does not exist, then we have $h(v) \geq h(w)$.*

Proof. Assume first that the edge (w, ℓ_{i-1}) exists, and let (v, z) and (w, y) be the remaining outgoing edges of v and w. Observe that both z and y are in one of the levelsets $L_{i-1}, \dots, L_k$ and hence we have $h(\ell_{i-1}) \leq h(z) \leq h(u_{i-1})$ and $h(\ell_{i-1}) \leq h(y) \leq h(u_{i-1})$. We conclude that indeed $h(v) = \frac{1}{2}h(u_{i-1}) + \frac{1}{2}h(z) \geq \frac{1}{2}h(y) + \frac{1}{2}h(\ell_{i-1}) = h(w)$.

Thus, assume now that (w, ℓ_{i-1}) does not exist. Then the edge (w, u_{i-1}) must exist instead. Let (v, z) and (w, y) be the remaining outgoing edges of v and w. Observe that all paths from z, y, v, w to any of the sinks must go through u_{i-1}. Hence, we get $h(y) = h(z) = h(v) = h(w) = h(u_{i-1})$. ◀

The only remaining case is the one where both average vertices have edges to both ℓ_{i-1} and u_{i-1}.

▶ **Lemma 9** (average vertices III). *Let $v, w \in L_i$ with $v \neq w$ be the two vertices in levelset L_i and assume that none of the Lemmas 5-8 apply. Then $h(v) = h(w)$.*

Proof. If none of the Lemmas 5-8 apply, then v, w must be average vertices, both with edges to u_{i-1} and ℓ_{i-1}. Hence, we have $h(v) = h(w)$. ◀

We conclude that we can determine ℓ_i and u_i in constant time by just looking at ℓ_{i-1}, u_{i-1} and the two vertices $v, w \in L_i$ as well as their outgoing edges.

4.2 The algorithm

In Section 4.1 we proved that we can find ℓ_i and u_i in polynomial time for all $i \in \{0, 1, \dots, k\}$. From Lemma 4, we know that this gives us a total order

$$h(\ell_0) \leq h(\ell_1) \leq \cdots \leq h(\ell_k) \leq h(u_k) \leq h(u_{k-1}) \leq \cdots \leq h(u_0)$$

on the optimal values of all vertices in the graph. From this total order, we can directly derive optimal strategies σ, τ for both players. More precisely, for each vertex in $V_{\max}$, we keep the outgoing edges to the out-neighbor with the higher value, while for each vertex in $V_{\min}$, we keep the edge to the one with the lower value. Finally, we can solve the Markov chain $G_{\sigma,\tau}$ to obtain the optimal value $h(\mathbf{o})$ and decide the game in polynomial time.

5 ARRIVAL on ladders in polynomial time

This section is wholly dedicated to show the following.

▶ **Theorem 10.** *ARRIVAL can be solved in polynomial time on ladders.*

We start by some structural observations on ladders in Section 5.1. Next, in Section 5.2, we define switching flows and recap the result that integral switching flows are certificates for whether an ARRIVAL instance is a YES- or a NO-instance. Then we give an overview of our polynomial-time algorithm in Section 5.3. After that, we explain the key lemma in Section 5.4 and the main technical recursive argument of the algorithm in Section 5.5. Finally, we wrap up the algorithm in Section 5.6.

For the rest of this section, we consider an ARRIVAL instance on a ladder $G = (V, E)$ with a designated source $\mathbf{o}$ and a balanced partition (E_0, E_1) of the edges E. Let $(L_0, L_1, \dots)$ be the level decomposition of G. Similar to Section 4, we also assume that there exists $k \in \mathbb{N}$ such that $|L_i| = 2$ for $i \in \{0, \dots, k\}$ and $|L_i| = 0$ for $i > k$.

5.1 Zero-player hitting probabilities

Consider the Markov chain on the ladder G, where the transition probability of every edge is $\frac{1}{2}$. That is, from every vertex in $V \setminus \{\mathbf{1}, \mathbf{0}\}$, there is an equal probability to visit one of its two out-neighbors next. Let $\tilde{h}_G(v)$ be the hitting probability at vertex v in this Markov chain. In other words, $\tilde{h}_G(v)$ is exactly the optimal $h(v)$ of the SSG on G where $V_{\max} = V_{\min} = \varnothing$. For this reason, we call this the *zero-player hitting probability*. We drop the subscript G from $\tilde{h}_G$ when the graph G is clear from context.

As explained in Section 4, in polynomial time, we can label the two vertices in each levelset L_i for $i \in \{0, \dots, k\}$ as ℓ_i and u_i such that $\ell_0 = \mathbf{0}$, $u_0 = \mathbf{1}$, and

$$0 = \tilde{h}(\ell_0) \le \tilde{h}(\ell_1) \le \cdots \le \tilde{h}(\ell_k) \le \tilde{h}(u_k) \le \tilde{h}(u_{k-1}) \le \cdots \le \tilde{h}(u_0) = 1. \tag{1}$$

Next, we can assume without loss of generality that $(\ell_k, \ell_{k-1}, \dots, \ell_0)$ and $(u_k, u_{k-1}, \dots, u_0)$ are two directed paths in G. Indeed, suppose for some $i \in \{1, \dots, k\}$, there is no edge from ℓ_i to ℓ_{i-1}. (The case of the missing edge (u_i, u_{i-1}) can be argued analogously.) Then ℓ_i must have an edge to u_{i-1}. By Lemma 8, $\tilde{h}(u_i) \le \tilde{h}(\ell_i)$. Combined with (1), this implies that $\tilde{h}(u_i) = \tilde{h}(\ell_i)$. If u_i has an edge to ℓ_{i-1}, we can exchange the labels of u_i and ℓ_i, and after this relabeling, we have the edges (ℓ_i, ℓ_{i-1}) and (u_i, u_{i-1}). Otherwise, the target of all forward edge(s) of u_i is u_{i-1}. This means that all vertices of L_i have their forward edges to u_{i-1}. Then by using the same argument as in Section 3, we can contract all vertices in $\bigcup_{j \ge i} L_j$ to u_{i-1}.

Finally, for each vertex $v \in V \setminus \{\mathbf{0}, \mathbf{1}\}$, we define the *gap* $g_G(v) := \tilde{h}_G(v) - \tilde{h}_G(s_0(v)) = -\big(\tilde{h}_G(v) - \tilde{h}_G(s_1(v))\big)$. We also drop the subscript G from g_G when the graph G is clear. The following lemma shows a crucial structural property of ladders that allows us to design an efficient algorithm later.

▶ **Lemma 11.** $\sum_{v \in V \setminus \{\mathbf{0},\mathbf{1}\}} |g(v)| = 1 + \tilde{h}(\ell_k) - \tilde{h}(u_k) \le 1$.

Proof. See Figure 2 for an illustration of the following proof. For each vertex $v \in V \setminus \{\mathbf{0}, \mathbf{1}\}$, $|g(v)|$ is the absolute difference between the $\tilde{h}$-value of v and that of any its out-neighbor. Combining this with (1) and the fact that $(\ell_k, \ell_{k-1}, \dots, \ell_0)$ and $(u_k, u_{k-1}, \dots, u_0)$ are two directed paths in G, we have for $i \in \{1, \dots, k\}$,

$$|g(\ell_i)| = \tilde{h}(\ell_i) - \tilde{h}(\ell_{i-1}),$$
$$|g(u_i)| = \tilde{h}(u_{i-1}) - \tilde{h}(u_i).$$

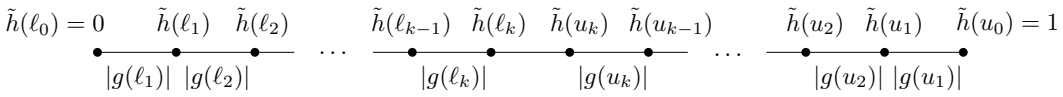

Figure 2 Illustration of Lemma 11. The points represent the $\tilde{h}$ values from smallest to largest. Note that each $|g(v)|$ is the distance between some two consecutive points. Hence, the sum $\sum_v g(v) + (\tilde{h}(u_k) - \tilde{h}(\ell_k))$ is the distance between $\tilde{h}(u_0)$ and $\tilde{h}(\ell_0)$, which is exactly one.

Therefore,

$$\begin{aligned}\sum_{v \in V \setminus \{\mathbf{0},\mathbf{1}\}} |g(v)| &= \sum_{i \in \{1,\dots,k\}} |g(\ell_i)| + \sum_{i \in \{1,\dots,k\}} |g(u_i)| \\ &= \sum_{i \in \{1,\dots,k\}} \big(\tilde{h}(\ell_i) - \tilde{h}(\ell_{i-1})\big) + \sum_{i \in \{1,\dots,k\}} \big(\tilde{h}(u_{i-1}) - \tilde{h}(u_i)\big) \\ &= \big(\tilde{h}(\ell_k) - \tilde{h}(\ell_0)\big) + \big(\tilde{h}(u_0) - \tilde{h}(u_k)\big) \\ &= 1 + \big(\tilde{h}(\ell_k) - \tilde{h}(u_k)\big) \leq 1.\end{aligned}$$

The last line above follows from (1). Specifically, the equality follows from the facts that $\tilde{h}(\ell_0) = 0$ and $\tilde{h}(u_0) = 1$, while the inequality follows from the fact that $\tilde{h}(\ell_k) \leq \tilde{h}(u_k)$. ◀

5.2 Switching Flows

Switching flows were first introduced in [7]. We define them here more generally.

▶ **Definition 12** (Switching Flow). *Given a binary graph $G = (V, E)$, a balanced partition (E_0, E_1) of E, and a function $\varphi : V \setminus \{\mathbf{0}, \mathbf{1}\} \to \mathbb{N}$, a* switching flow *with respect to (V, E_0, E_1, φ) is a function $x : E \to \mathbb{R}_{\geq 0}$ satisfying the equations*

$$\sum_{u:(v,u)\in E} x(v,u) - \sum_{u:(u,v)\in E} x(u,v) = \varphi(v)$$
$$0 \leq x(v, s_0(v)) - x(v, s_1(v)) \leq 1$$

for all $v \in V \setminus \{\mathbf{0}, \mathbf{1}\}$.

▶ **Definition 13** (Inflow and Outflow). *Given a binary graph $G = (V, E)$, a balanced partition $E = E_0 \cup E_1$, and a function $\varphi : V \setminus \{\mathbf{0}, \mathbf{1}\} \to \mathbb{N}$, and a switching flow x with respect to (V, E_0, E_1, φ), we call $x^-(v) := \sum_{u:(u,v)\in E} x(u, v)$ the* inflow *of $v \in V$. Analogously, we call $x^+(v) := \sum_{u:(v,u)\in E} x(v, u)$ the* outflow *of $v \in V$.*

We distinguish between different types of switching flows.

▶ **Definition 14** (Types of Switching Flows). *Suppose we are given a binary graph $G = (V, E)$, a balanced partition (E_0, E_1) of E, a function $\varphi : V \setminus \{\mathbf{0}, \mathbf{1}\} \to \mathbb{N}$, and a switching flow x with respect to (V, E_0, E_1, φ). We use the following adjectives to define different types of switching flows.*

- *We call x* integral *if $x(e) \in \mathbb{N}$ for all $e \in E$.*
- *Suppose for some $\mathbf{o} \in V$, φ satisfies $\varphi(\mathbf{o}) = 1$ and $\varphi(v) = 0$ for $v \neq \mathbf{o}$. Then we say x is a switching flow* from $\mathbf{o}$.
- *Finally, we call x a switching flow* to $\mathbf{1}$ *if $x^-(\mathbf{0}) = 0$, and we call it a switching flow* to $\mathbf{0}$ *if $x^-(\mathbf{1}) = 0$.*

The following result goes back to Dohrau, Gärtner, Kohler, Matoušek, and Welzl [7] and implicitly proves that ARRIVAL is in NP ∩ CoNP.

▶ **Theorem 15** (Switching Flows are Certificates [7]). *An instance of ARRIVAL has an integral switching flow from the origin* $\mathbf{o}$ *to* $\mathbf{1}$ *if and only if the token eventually arrives at* $\mathbf{1}$*. Analogously, it has an integral switching flow from the origin* $\mathbf{o}$ *to* $\mathbf{0}$ *if and only if the token ends up at* $\mathbf{0}$*.*

In other words, in order to solve ARRIVAL, it suffices to find an integral switching flow.

5.3 Overview of the algorithm

As we have seen in the previous subsection, one way to try to solve ARRIVAL is to solve its integer linear program (ILP) formulation implicitly given in Definition 12 together with the constraints that all variables are integral. Unfortunately, we do not know how to solve this ILP fast. However, we can easily obtain a (not necessarily integral) switching flow by solving the linear system in Definition 12 in polynomial time. In fact, for each of the two sinks, we can efficiently check if there exists a switching flow to that sink. The non-existence of a switching flow to one sink suffices to decide ARRIVAL. Thus, the interesting case is when there are two switching flows from $\mathbf{o}$, one to $\mathbf{1}$ and one to $\mathbf{0}$.

The general idea to solve this case is as follows. We start with the two switching flows above. At every iteration, we attempt to make more values in the two switching flows to be integral, while maintaining that one switching flow is from $\mathbf{o}$ to $\mathbf{1}$ and the other to $\mathbf{0}$. More specifically, at iteration i, we require the flow on the edges starting from a vertex in $\bigcup_{j\leq i} L_j$ to be integral. We show that for each switching flow, there is at most one choice of integral values for these edges, and we can compute it in polynomial time. Hence, either we can decide the instance immediately (because no integral values are possible for one switching flow) or we can progressively fix more integral values in both switching flows.

5.4 Difference between two switching flows

This subsection is dedicated to Lemma 16 below, which is the key lemma to our algorithm and discusses the relationship between *any* two switching flows with respect to the same tuple (V, E_0, E_1, φ). In particular, it shows that the inflows to $\mathbf{1}$ in any two switching flows cannot differ by more than one. This is trivial when the switching flows are from an origin $\mathbf{o}$, since the inflows to $\mathbf{1}$ can then be at most one. However, it is not obvious for an arbitrary function φ, where the inflow to $\mathbf{1}$ can potentially take any value between 0 and $\sum_{v\in V\setminus\{\mathbf{0},\mathbf{1}\}} \varphi(v)$. Further, Lemma 16 also states that when the difference between the inflows to $\mathbf{1}$ in the two switching flows is exactly one, we know exactly the difference between the flows of the even and odd edges from each vertex v such that $g(v) \neq 0$, in both switching flows.

▶ **Lemma 16.** *Let* $G = (V, E)$ *be a ladder,* (E_0, E_1) *a balanced partition of* E*, and* $\varphi : V \setminus \{\mathbf{0}, \mathbf{1}\} \to \mathbb{N}$ *a function. Let* x *and* y *be two switching flows with respect to* (V, E_0, E_1, φ)*. Then* $|x^-(\mathbf{1}) - y^-(\mathbf{1})| \leq 1$*.*

Further, if $x^-(\mathbf{1}) - y^-(\mathbf{1}) = 1$*, then* $\tilde{h}(\ell_k) = \tilde{h}(u_k)$*, and for each* $v \in V \setminus \{\mathbf{0}, \mathbf{1}\}$*,*

- *if* $g(v) < 0$*, then* $x(v, s_0(v)) - x(v, s_1(v)) = 1$ *and* $y(v, s_0(v)) - y(v, s_1(v)) = 0$*; and*
- *if* $g(v) > 0$*, then* $x(v, s_0(v)) - x(v, s_1(v)) = 0$ *and* $y(v, s_0(v)) - y(v, s_1(v)) = 1$*.*

Proof. Consider the following sum over all edges of E:

$$M := \sum_{(u,v)\in E} \big(y(u,v) - x(u,v)\big)\big(\tilde{h}(u) - \tilde{h}(v)\big).$$

On one hand, we can sum over the vertices and obtain

$$\begin{aligned} M &= \sum_{v \in V} \tilde{h}(v) \left(y^+(v) - y^-(v) - x^+(v) + x^-(v)\right) \\ &= x^-(\mathbf{1}) - y^-(\mathbf{1}). \end{aligned} \tag{2}$$

In the last equality, we use the facts that $\tilde{h}(\mathbf{1}) = 1$, that $\tilde{h}(\mathbf{0}) = 0$, that $x^+(\mathbf{1}) = y^+(\mathbf{1}) = 0$, and that for $v \in V \setminus \{\mathbf{0}, \mathbf{1}\}$, $x^+(v) - x^-(v) = y^+(v) - y^-(v) = \varphi(v)$.

On the other hand, we can consider the two cases of whether (u, v) is an odd edge or even edge and obtain

$$\begin{aligned} M &= \sum_{v \in V \setminus \{\mathbf{0},\mathbf{1}\}} \big(y(v, s_0(v)) - x(v, s_0(v))\big) g(v) - \big(y(v, s_1(v)) - x(v, s_1(v))\big) g(v) \\ &= \sum_{v \in V \setminus \{\mathbf{0},\mathbf{1}\}} \big(y(v, s_0(v)) - y(v, s_1(v)) - x(v, s_0(v)) + x(v, s_1(v))\big) g(v) \\ &\leq \sum_{v \in V \setminus \{\mathbf{0},\mathbf{1}\}} |g(v)| \leq 1 + \tilde{h}(\ell_k) - \tilde{h}(u_k) \leq 1. \end{aligned} \tag{3}$$

The second and third inequalities above follow from Lemma 11, while in the first inequality, we use the fact that for $v \in V \setminus \{\mathbf{0}, \mathbf{1}\}$, $0 \leq y(v, s_0(v)) - y(v, s_1(v)) \leq 1$ and $0 \leq x(v, s_0(v)) - x(v, s_1(v)) \leq 1$ to deduce that

$$-1 \leq y(v, s_0(v)) - y(v, s_1(v)) - x(v, s_0(v)) + x(v, s_1(v)) \leq 1.$$

Combining (2) and (3), we obtain $x^-(\mathbf{1}) - y^-(\mathbf{1}) \leq 1$. Symmetrically, we also have $y^-(\mathbf{1}) - x^-(\mathbf{1}) \leq 1$. Hence, $|x^-(\mathbf{1}) - y^-(\mathbf{1})| \leq 1$, as claimed.

If $x^-(\mathbf{1}) - y^-(\mathbf{1}) = 1$, the equalities of (3) must hold at equalities. This implies that $\tilde{h}(\ell_k) = \tilde{h}(u_k)$ and for $v \in V \setminus \{\mathbf{0}, \mathbf{1}\}$,

- if $g(v) < 0$, then $x(v, s_0(v)) - x(v, s_1(v)) = 1$ and $y(v, s_0(v)) - y(v, s_1(v)) = 0$; and
- if $g(v) > 0$, then $x(v, s_0(v)) - x(v, s_1(v)) = 0$ and $y(v, s_0(v)) - y(v, s_1(v)) = 1$. ◀

5.5 Fixing flow values from the next levelset

For convenience, we first define a few notations. For $i \in \{0, \ldots, k\}$, define $L_{\leq i} := \bigcup_{j=0}^{i} L_j$ and $L_{\geq i} := \bigcup_{j=i}^{k} L_j$. Moreover, we say a switching flow x is *i-integral* if for every edge e from a vertex in $L_{\leq i}$, $x(e) \in \mathbb{N}$. (x, y) is an *i-pair*, if x and y are i-integral switching flows from $\mathbf{o}$ to $\mathbf{1}$ and $\mathbf{0}$, respectively. In this subsection, we package the idea of the algorithm in Section 5.3 in the following lemma.

▶ **Lemma 17.** *For $i \in \{0, \ldots, k\}$, if there exists an i-pair (x, y), then*

1. *for every outgoing edge e of a vertex in $L_{\leq i}$, there exists only one possible value for each of $x(e)$ and $y(e)$;*
2. *if such e is a backward edge, then $x(e) = y(e)$; and*
3. *for $v \in L_{\leq i}$, $|x^-(v) - y^-(v)| = 1$.*

Moreover, in polynomial time, we can output either an i-pair, or that there is no i-integral switching flow from $\mathbf{o}$ to $\mathbf{1}$, or that there is no i-integral switching flow from $\mathbf{o}$ to $\mathbf{0}$.

Proof. We prove by induction on i. For the base case $i = 0$, a 0-pair (x, y) is simply a pair of switching flows from $\mathbf{o}$ to $\mathbf{1}$ and to $\mathbf{0}$, without any integrality constraint. Then statements 1 and 2 are vacuously true, while statement 3 holds since $x^-(\mathbf{1}) - y^-(\mathbf{1}) = 1$ and $x^-(\mathbf{0}) - y^-(\mathbf{0}) = -1$. Hence, we only need to argue about the run time. As noted before, x

can be obtained by solving the linear program that comprises the inequalities in Definition 12 and the constraint that the inflow to **1** is one. Likewise, y can be obtained by solving a similar linear program but with the constraint that the inflow to **1** is instead zero. These linear programs can be solved in polynomial time. If one of them is infeasible, we conclude that the corresponding 0-integral switching flow does not exist. Otherwise, combining a solution of each linear program yields a 0-pair.

For the inductive step, assume that the lemma statement holds for some $i < k$. We show that it also holds for $i+1$. Suppose that there exists an $(i+1)$-pair (x, y). Note that this is also an i-pair. Hence, statements 1–3 already holds for vertices in $L_{\leq i}$, and we only need to show them for vertices in L_{i+1}. The idea is to apply Lemma 16 to fix the differences $x(v, s_0(v)) - x(v, s_1(v))$ and $y(v, s_0(v)) - y(v, s_1(v))$ for $v \in L_{i+1}$ and from this information deduce the unique values for $x(v, s_0(v))$, $x(v, s_1(v))$, $y(v, s_0(v))$, and $y(v, s_1(v))$. However, the issue here is that we require $g(v) \neq 0$ in order to apply the lemma, and this may not hold. In order to circumvent this, instead of applying Lemma 16 directly to the original ladder G, we apply it to the ladder induced on the layers $L_{\geq i}$ instead.

Formally, let $G^i = (V^i, E^i)$ be the graph obtained from G by removing the vertices in $L_{\leq i-1}$, removing the outgoing edges from ℓ_i and u_i, and relabeling ℓ_i and u_i to **0** and **1**. It is easy to see that G^i is a ladder with the level decomposition $(L_i, L_{i+1}, \dots)$. To avoid confusion, we still refer to the vertices in L_i by their old labels ℓ_i and u_i. Let $E^i_0 := E^i \cap E_0$ and $E^i_1 := E^i \cap E_1$. Then (E^i_0, E^i_1) is a balanced partition of E^i. Next, let φ^i be the function such that for $v \in L_{\geq i+1}$,

$$\varphi^i(v) = \begin{cases} \sum_{(u,v)\in E : u \in L_{\leq i}} x(u,v) & \text{if } v \neq \mathbf{o} \\ 1 + \sum_{(u,v)\in E : u \in L_{\leq i}} x(u,v) & \text{if } v = \mathbf{o}. \end{cases}$$

Finally, let x^i and y^i be the functions obtained by restricting x and y to the set $\{(u,v) \in E \mid u \in L_{\geq i+1}\}$. Then by construction, x^i is a switching flow with respect to $(V^i, E^i_0, E^i_1, \varphi^i)$. Moreover, applying statement 2 in the inductive hypothesis, we also have that y^i is a switching flow with respect to $(V^i, E^i_0, E^i_1, \varphi^i)$.

We now argue that $g_{G^i}(\ell_{i+1}) \neq 0$. Let S be the sequence $(\ell_i, \ell_{i+1}, \dots, \ell_k, u_k, \dots, u_i)$. Let v be a vertex in the sequence except for ℓ_i and u_i. Suppose the two out-neighbors of v precede it in the sequence S. Since there are two directed paths $(\ell_k, \dots, \ell_i)$ and $(u_k, \dots, u_i)$, v must be ℓ_j for some $j \in \{i+1, \dots, k\}$. However, due to the definition of the level decomposition, the preceding out-neighbors of ℓ_j in the sequence can only be ℓ_{j-1}. Now together with the fact that the graph is simple, we arrive at a contradiction. Hence, v must have at least one out-neighbor succeeding it in S. By an analogous argument, v must also have at least one out-neighbor preceding it in S. Hence, v has exactly one preceding and one succeeding out-neighbor. Now, if $g_{G^i}(\ell_{i+1}) = 0$, then since $\ell_i \in \{s_0(\ell_{i+1}), s_1(\ell_{i+1})\}$ and $\tilde{h}_{G^i}(\ell_i) = 0$, we have $\tilde{h}_{G^i}(s_0(\ell_i)) = \tilde{h}_{G^i}(s_1(\ell_i)) = 0$. Combined with (1), this implies that $\tilde{h}_{G^i}(u) = 0$ for all u between ℓ_i and $w := \max\{s_0(\ell_i), s_1(\ell_i)\}$ in S, where max indicates the further vertex among the two in the sequence. As shown before, w has one preceding and one succeeding out-neighbor in S. We then use the same argument as above to derive that $\tilde{h}_{G^i}(u) = 0$ for all u between ℓ_i and $\max\{s_0(w), s_1(w)\}$ in S. Repeating this argument, we obtain that the whole sequence S has the same $\tilde{h}_{G^i}$ values. However, this is false, since $\tilde{h}_{G^i}(\ell_i) = 0 \neq 1 = \tilde{h}_{G^i}(u_i)$. Hence, $g_{G^i}(\ell_{i+1}) \neq 0$. By a similar argument, $g_{G^i}(u_{i+1}) \neq 0$.

Therefore, we can now apply Lemma 16 on G^i, (E^i_0, E^i_1), φ^i, and the two switching flows x^i and y^i. By statements 2 and 3 of the inductive hypothesis, we obtain that $|x^{i-}(u_i) - y^{i-}(u_i)| = 1$. Further, as argued above, $g_{G^i}(v) \neq 0$ for $v \in L_{i+1}$. Therefore, Lemma 16 implies that for $v \in \{\ell_{i+1}, u_{i+1}\}$ exactly one of $x(v, s_0(v)) - x(v, s_1(v))$ and $y(v, s_0(v)) - y(v, s_1(v))$ has value

0 and the other has value 1. Further, we know exactly these values, since we can compute $g_{G^i}(\ell_{i+1})$ and $g_{G^i}(u_{i+1})$. (Recall that g_{G^i} can easily computed from $\tilde{h}_{G^i}$, which in turn can be computed in polynomial time as explained in Section 4.)

In the remainder of the proof, we assume that

$$x(u_{i+1}, s_0(u_{i+1})) - x(u_{i+1}, s_1(u_{i+1})) = 1 \text{ and } y(u_{i+1}, s_0(u_{i+1})) - y(u_{i+1}, s_1(u_{i+1})) = 0; \quad (4)$$

the other case can be argued analogously. By Lemma 16, in this case, one of the following holds:

- $x^-(u_i) - y^-(u_i) = 1$ and $g_{G^i}(u_{i+1}) < 0$; or
- $x^-(u_i) - y^-(u_i) = 0$ and $g_{G^i}(u_{i+1}) > 0$.

Note that since $\tilde{h}_{G^i}(u_i) > \tilde{h}_{G^i}(u_{i+1})$, in the former case, we have $u_i = s_0(u_{i+1})$, and in the latter case, $u_i = s_1(u_{i+1})$.

We now complete the proof of the inductive step by considering the two following cases.

Case 1. *At least one of ℓ_{i+1} and u_{i+1} have only one forward edge.* Suppose ℓ_{i+1} has only one forward edge. Then for u_i, other than its incoming edge (u_{i+1}, u_i), all its other incoming edges are from vertices in $L_{\leq i}$. We have

$$x(u_{i+1}, u_i) = x^+(u_i) - \sum_{(v,u_i)\in E : v \in L_{\leq i}} x(v, u_i) - \delta(u_i),$$
$$y(u_{i+1}, u_i) = y^+(u_i) - \sum_{(v,u_i)\in E : v \in L_{\leq i}} y(v, u_i) - \delta(u_i),$$

where $\delta(u_i)$ takes value one if $u_i = \mathbf{o}$ and zero otherwise.

Note that the edges in the sums in the two expressions above are backward edges from vertices in $L_{\leq i}$. Hence, by the inductive hypothesis, the two sums have the same value. Further, also by the inductive hypothesis, the values of $x^+(u_i)$ and $y^+(u_i)$ are fixed and differ by one. Hence, the values of $x(u_{i+1}, u_i)$ and $y(u_{i+1}, u_i)$ are also unique and

$$x(u_{i+1}, u_i) - y(u_{i+1}, u_i) = x^+(u_i) - y^+(u_i) = x^-(u_i) - y^-(u_i) = x^{i-}(u_i) - y^{i-}(u_i). \quad (5)$$

Now suppose $x^+(u_i) - y^+(u_i) = 1$. Then $x^-(u_i) - y^-(u_i) = 1$. As argued above, this implies $u_i = s_0(u_{i+1})$ and $g_{G^i}(u_{i+1}) < 0$. Then combining this with (4) and (5), we obtain that $x(u_{i+1}, s_1(u_{i+1}))$ and $y(u_{i+1}, s_1(u_{i+1}))$ have the same unique value. Hence, the statements 1–3 hold for the outgoing edges of u_{i+1}. With a similar analysis for the incident edges of ℓ_i, we also conclude statements 1–3 hold for the outgoing edges of ℓ_{i+1}. Further, note that all these unique values can be computed in polynomial time. Once we have all the unique values for the outgoing edges from $L_{\leq i+1}$, we add constraints to fix these values in the linear program to compute the switching flows from $\mathbf{o}$ to $\mathbf{0}$ and to $\mathbf{1}$. If one of these linear programs is infeasible, we conclude that the corresponding $(i+1)$-switching flow does not exist; otherwise, their solutions form an $(i+1)$-pair.

The case for $x^+(u_i) - y^+(u_i) = -1$ can be argued analogously. It remains to consider the case when the two outgoing edges of ℓ_{i+1} are forward edges. Then u_{i+1} must have only one forward edge. We then use similar arguments as before to show that statements 1–3 hold for all outgoing edges of u_{i+1} and ℓ_{i+1} and that the relevant computation can be done in polynomial time.

Case 2. *All outgoing edges of ℓ_{i+1} and u_{i+1} are forward edges.* Since the values of the outgoing edges from ℓ_{i+1} and u_{i+1} in x and y are integral, the assumption of (4) implies that $x^+(u_{i+1})$ is odd and $y^+(u_{i+1})$ is even.

We now define $V^{i+1}, E_0^{i+1}, E_1^{i+1}, \varphi^{i+1}, x^{i+1}, y^{i+1}$ similar to how we defined $V^i, E_0^i, E_1^i, \varphi^i$, x^i, y^i. Since the outgoing edges of ℓ_{i+1} and u_{i+1} are both forward edges, and since the values for the backward edges from vertices in $L_{\leq i}$ are the same in x and y, we obtain that x^{i+1} and y^{i+1} are switching flows with respect to $(V^{i+1}, E_0^{i+1}, E_1^{i+1}, \varphi^{i+1})$. Lemma 16 states that for *any* two switching flows with respect to $(V^{i+1}, E_0^{i+1}, E_1^{i+1}, \varphi^{i+1})$, the inflows to u_{i+1} differ by at most one. This means that if we know the parity of the inflow to u_{i+1} then there can only be a unique value for this inflow, since if any distinct values of the same parity have difference at least two. This implies statement 1 for the outgoing edges of u_{i+1}. Statement 2 for these edges hold vacuously, as these edges are not backward edges. Statement 3 is also implied from the analysis above: We know that the inflows to u_{i+1} in x and y differ by at most one. Since the parities of these inflows are different, they have to differ exactly by one.

It remains to show that we can do the computation in polynomial time. By the inductive hypothesis, in polynomial time, we can obtain an i-pair $(\bar{x}, \bar{y})$. As argued above, we know that $x^+(u_{i+1})$ is an odd number in the range $[\bar{x}^+(u_{i+1}) - 1, \bar{x}^+(u_{i+1}) + 1]$. If there is only one odd number in the range, then that is the value for $x^+(u_{i+1})$. If there are two odd numbers in the range, we use the earlier observation that one of these numbers cannot be the outflow of u_{i+1} for *any* i-switching flow from $\mathbf{o}$ to $\mathbf{1}$. Hence, by using the same linear program to obtain $\bar{x}$ with the additional constraint to fix the outflow of u_{i+1} to one of odd numbers in this range, we can verify which odd number is the unique possible value for $x^+(u_{i+1})$ based on the feasibility of this linear program. After obtaining the unique value for $x^+(u_{i+1})$, we can easily deduce the unique values of $x(u_{i+1}, s_0(u_{i+1}))$ and $x(u_{i+1}, s_1(u_{i+1}))$. By the same procedure, we can also obtain the unique values for $y(u_{i+1}, s_0(u_{i+1}))$ and $y(u_{i+1}, s_1(u_{i+1}))$, as well as the values for the outgoing edges of ℓ_{i+1}. Note that in the entire process, if something is infeasible, we can immediate conclude that the corresponding $(i+1)$-switching flow does not exist. ◀

5.6 Summary

Theorem 15 states that a k-integral switching flow from $\mathbf{o}$ to $\mathbf{1}$ (resp., to $\mathbf{0}$) is a certificate of a YES-instance (resp., NO-instance). This implies that there is no k-pair. Hence, using the algorithm as guaranteed by Lemma 17 for $i = k$, in polynomial time, we conclude that either the non-existence of a k-integral switching flow from $\mathbf{o}$ to $\mathbf{1}$ or non-existence of a k-integral switching flow from $\mathbf{o}$ to $\mathbf{0}$. We can then decide the ARRIVAL instance accordingly.

6 Concluding remarks

In this article, we prove that ARRIVAL and SSG on ladders can be solved in polynomial time. Note that our proof of Lemma 17 above mainly aims at making the run time polynomial rather than optimizing it as much as possible. Instead of solving a linear number of linear programs as this proof implies, we can solve as few as $\mathcal{O}(t)$ linear programs, where t is the number of levelsets i such that all four outgoing edges from L_i are forward edges. Further, the results can also be extended to the generalization of ARRIVAL with multiple tokens. We refer to [14] for more detail on both of these results.

Whether ARRIVAL can be solved in polynomial time remains open. While ladders capture a subset of worst-case instances for the subexponential time algorithm by [11], there are still other instances with the same asymptotic run time, e.g., when all non-empty levelsets have r vertices for some constant $r > 2$. Unfortunately, the analysis in this article does not extend to these cases, and they could form another interesting class of graphs for future work.

References

1 Scott Aaronson. The computational expressiveness of a model train set: A paperlet. https://scottaaronson.blog/?p=5402, 2021. Accessed: 6 July 2022.

2 David Auger, Pierre Coucheney, and Loric Duhazé. Polynomial Time Algorithm for ARRIVAL on Tree-Like Multigraphs. In Stefan Szeider, Robert Ganian, and Alexandra Silva, editors, *47th International Symposium on Mathematical Foundations of Computer Science (MFCS 2022)*, volume 241 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 12:1–12:14, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.MFCS.2022.12.

3 David Auger, Pierre Coucheney, Loric Duhazé, and Kossi Roland Etse. Generalized ARRIVAL problem for rotor walks in path multigraphs. In *Reachability problems*, volume 14235 of *Lecture Notes in Comput. Sci.*, pages 183–198. Springer, Cham, [2023] ©2023. doi:10.1007/978-3-031-45286-4_14.

4 Adam Chalcraft and Michael Greene. Train sets. *Eureka*, 53:5–12, 1994.

5 Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203–224, 1992. doi:10.1016/0890-5401(92)90048-K.

6 Joshua Cooper, Benjamin Doerr, Joel Spencer, and Gábor Tardos. Deterministic random walks on the integers. *European Journal of Combinatorics*, 28(8):2072–2090, 2007. EuroComb '05 - Combinatorics, Graph Theory and Applications. doi:10.1016/J.EJC.2007.04.018.

7 Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. ARRIVAL: A Zero-Player Graph Game in NP ∩ coNP. In Martin Loebl, Jaroslav Nešetřil, and Robin Thomas, editors, *A Journey Through Discrete Mathematics: A Tribute to Jiří Matoušek*, pages 367–374. Springer International Publishing, Cham, 2017. doi:10.1007/978-3-319-44479-6_14.

8 John Fearnley, Martin Gairing, Matthias Mnich, and Rahul Savani. Reachability switching games. In *45th International Colloquium on Automata, Languages, and Programming*, volume 107 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 124, 14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

9 John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. Unique end of potential line. *Journal of Computer and System Sciences*, 114:1–35, December 2020. doi:10.1016/j.jcss.2020.05.007.

10 Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubácek, Karel Král, Hagar Mosaad, and Veronika Slívová. ARRIVAL: Next Stop in CLS. In Ioannis Chatzigiannakis, Christos Kaklamanis, Dániel Marx, and Donald Sannella, editors, *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 60:1–60:13, Dagstuhl, Germany, 2018. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. ISSN: 1868-8969. doi:10.4230/LIPIcs.ICALP.2018.60.

11 Bernd Gärtner, Sebastian Haslebacher, and Hung P. Hoang. A subexponential algorithm for ARRIVAL. In *48th International Colloquium on Automata, Languages, and Programming*, volume 198 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 69, 14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021.

12 Ebrahim Ghorbani, Jonah Leander Hoff, and Matthias Mnich. A quasi-polynomial time algorithm for multi-arrival on tree-like multigraphs. In *42nd International Symposium on Theoretical Aspects of Computer Science*, volume 327 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 39, 19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/lipics.stacs.2025.39.

13 Sebastian Haslebacher. ARRIVAL: recursive framework & ℓ_1-contraction. In *52nd International Colloquium on Automata, Languages, and Programming*, volume 334 of *LIPIcs. Leibniz Int. Proc. Inform.*, pages Art. No. 95, 17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/lipics.icalp.2025.95.

14 Hung P. Hoang. *On Two Combinatorial Reconfiguration Problems: Reachability and Hamiltonicity*. PhD thesis, ETH Zurich, 2022. doi:10.3929/ethz-b-000572947.

15 Alexander E. Holroyd and James Propp. Rotor walks and Markov chains. In *Algorithmic probability and combinatorics*, volume 520 of *Contemp. Math.*, pages 105–126. Amer. Math. Soc., Providence, RI, 2010.

16 Viacheslav B. Priezzhev, Deepak Dhar, Abhishek Dhar, and Supriya Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Phys. Rev. Lett.*, 77:5079–5082, 1996.

17 Günter Rote. Personal communication, 2020.