arXiv:1802.07702v1 [cs.CC] 21 Feb 2018

ARRIVAL: Next Stop in CLS*

Bernd Gärtner[1], Thomas Dueholm Hansen[2], Pavel Hubáček[3], Karel Král[3], Hagar Mosaad[4], and Veronika Slívová[3]

[1]Department of Computer Science, ETH Zürich, Switzerland
gaertner@inf.ethz.ch
[2]Department of Computer Science, University of Copenhagen, Denmark
dueholm@di.ku.dk
[3]Computer Science Institute, Charles University, Prague, Czech Republic
{hubacek, kralka, slivova}@iuuk.mff.cuni.cz
[4]Department of Computer Science and Engineering, German University in Cairo, Egypt
hagar.omar@student.guc.edu.eg

February 22, 2018

Abstract

We study the computational complexity of Arrival, a zero-player game on n-vertex switch graphs introduced by Dohrau, Gärtner, Kohler, Matoušek, and Welzl. They showed that the problem of deciding termination of this game is contained in NP ∩ coNP. Karthik C. S. recently introduced a search variant of Arrival and showed that it is in the complexity class PLS. In this work, we significantly improve the known upper bounds for both the decision and the search variants of Arrival.

First, we resolve a question suggested by Dohrau et al. and show that the decision variant of Arrival is in UP ∩ coUP. Second, we prove that the search variant of Arrival is contained in CLS. Third, we give a randomized $O(1.4143^n)$-time algorithm to solve both variants.

Our main technical contributions are (a) an efficiently verifiable characterization of the unique witness for termination of the Arrival game, and (b) an efficient way of sampling from the state space of the game. We show that the problem of finding the unique witness is contained in CLS, whereas it was previously conjectured to be FPSPACE-complete. The efficient sampling procedure yields the first algorithm for the problem that has expected runtime $O(c^n)$ with $c < 2$.

1 Introduction

Variants of switch graphs have applications and are studied for example in combinatorics and in automata theory (cf. [10] and the references therein). Dohrau et al. [5] introduced Arrival, a natural computational problem on switch graphs, which they informally described as follows:

> 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? [5]

*The research leading to these results has received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013) / ERC Grant Agreement n. 616787 and was supported by the project 17-09142S of GA ČR. P. Hubáček and V. Slívová were partially supported by PRIMUS grant PRIMUS/17/SCI/9. T.D. Hansen was partially supported by BARC which is funded by a VILLUM Investigator grant (16582) to Mikkel Thorup.

The above rather straightforward question remains unresolved. Dohrau et al. [5] showed that deciding Arrival is unlikely to be NP-complete (by demonstrating that it is in NP ∩ coNP), but it is currently not known to be efficiently solvable.

To determine whether the train eventually reaches its destination, it is natural to consider a *run profile*, i.e., the complete transcript describing how many times the train traversed each edge. Dohrau et al. [5] presented a natural integer programming interpretation of run profiles called *switching flows*, which have the advantage of being trivial to verify. The downside of switching flows is that they do not guarantee to faithfully represent the number of times each edge has been traversed; a switching flow might contain superfluous circulations compared to a valid run profile. Nevertheless, Dohrau et al. [5] proved that the existence of a switching flow implies that the train reaches its destination, and thus a switching flow constitutes an NP witness for Arrival.

The coNP membership was shown by an insightful observation about the structure of switch graphs. Specifically, the train reaches its destination if and only if it never enters a node from which there is no directed path to d. The railway network can thus be altered so that all such vertices point to an additional "dead-end" vertex $\bar{d}$. The coNP witness is then simply a switching flow from the origin to the dead-end $\bar{d}$.

Given that the decision variant of Arrival is in NP ∩ coNP, it is natural to study the search complexity of Arrival in the context of total search problems with the guaranteed existence of a solution, i.e., within the complexity class TFNP (which contains the search analogue of NP∩coNP). Total search problems are classified into subclasses of TFNP using the methodology proposed by Papadimitriou [12] that clusters computational problems according to the type of argument assuring the existence of a solution. Karthik C. S. [13] noticed that the search for a switching flow is a prime candidate to fit into the hierarchy of TFNP problems. He introduced S-Arrival, a search version of Arrival that seeks a switching flow to either the destination d or the dead-end vertex $\bar{d}$, and showed that it is contained in the complexity class PLS [9] of total problems amenable to local search.

Fearnley et al. [6] recently studied multiple variants of reachability games on switch graphs and as one of their results gave a lower bound on the complexity of deciding ARRIVAL. Specifically, they showed that Arrival is NL-hard.

1.1 Our Results

One of the open problems suggested by Dohrau et al. [5] was whether deciding the termination of Arrival is contained in UP ∩ coUP. Recall that given a railway network with an origin o and a destination d, the transcript of the route of the train captured in the run profile from o to d (if it exists) is unique. We show that it is possible to efficiently decide whether a switching flow corresponds to a run profile, which provides a positive answer to the above question and places Arrival inside UP ∩ coUP. We similarly also improve the upper bound on the search complexity of Arrival: We show that S-Arrival is contained in the complexity class CLS. Daskalakis and Papadimitriou [4] introduced CLS to classify problems that can be reduced to local search over continuous domains. CLS contains multiple important search problems such as solving simple stochastic games, finding equilibria in congestion games, and solving linear complementarity problems on P-matrices. For all of these problems, as well as for S-Arrival, we currently do not have a polynomial time algorithm, and they are not known to be complete for some subclass of TFNP.

We establish the containment in CLS through a reduction to End-Of-Metered-Line (EOML), a total search problem that was recently introduced by Hubáček and Yogev [8] who also showed that it is in CLS. In EOML we are given a source in a directed graph with vertices of in-degree and out-degree at most one, and the task is to find a sink or a source different from the given trivial source. The access to the graph is given locally via information about the successor and predecessor of each vertex together with its distance from the trivial source (for the formal definition see Definition 19).

Our result makes it unlikely for S-Arrival to be PLS-hard, which was one of the possibilities suggested by the containment in PLS shown by Karthik C. S. [13]. This is due to known black-box separations among subclasses of TFNP [11, 2], which suggest that CLS is a proper subclass of PLS. Note that our reduction from S-Arrival to End-Of-Metered-Line results in instances with a significantly restricted structure: the End-Of-Metered-Line graph consists only of a single path and many isolated vertices. We believe that this structure may in future work be used to show that S-Arrival is contained in FP.

Our reduction from S-Arrival to End-Of-Metered-Line also implies that we can use an algorithm by Aldous [1] to solve S-Arrival. The algorithm is randomized and runs in $O(2^{n/2}poly(n))$ expected time on switch graphs with n vertices. This is the first algorithm with expected runtime $O(c^n)$ for $c < 2$. (A trivial $O(2^n poly(n))$ time algorithm can be obtained by following the path of the train through the network.) Aldous' algorithm, in fact, solves any problem in PLS. It samples a large number of candidate solutions and then performs a local search from the best sampled solution. The advantage of our reduction is that the resulting search space for End-Of-Metered-Line is small enough to make Aldous' algorithm useful, unlike in the previous reduction by Karthik C. S. [13] that showed containment in PLS.

Fearnley et al. [7] recently gave a reduction from P-matrix linear complementarity problems (PLCP) to End-Of-Metered-Line. As in our case for Arrival, this implies that Aldous' algorithm can be used to solve PLCP. In fact this gives the fastest known randomized algorithm for PLCP, running in expected time $O(2^{n/2}poly(n))$ for input matrices of dimension $n \times n$. Fearnley et al. do not make this observation themselves, but it is straightforward to check that their reduction also gives an efficient representation of the search space. Although Aldous' algorithm is very simple, it thus non-trivially improves the best runtime of algorithms for multiple problems. We believe that this way of applying Aldous' algorithm is a powerful technique that will produce additional results in the future.

1.2 Technical remarks

Recall that a switching flow is a run profile with additional superfluous circulations compared to the valid run profile. Our main technical observation is a characterization of switching flows that correspond to the valid run profile. Given a switch graph G and a switching flow f, we consider the subgraph G^* induced over the railway network by the "last-used" edges; for every vertex v, we include in G^* only the outgoing edge that was, according to the switching flow, used by the train last time it left from v. Note that such last-used edges can be efficiently identified simply by considering the parity of the total number of visits at every vertex. When f is a valid run profile, then it is straightforward to see that the subgraph G^* is acyclic. We show that this property is in fact a characterization, i.e., any switching flow for which the induced graph G^* is acyclic must be a run profile. Given that this property is easy to check, we can use it to efficiently verify run profiles as UP witnesses. (The coUP witness is then a run profile to the dead-end at $\bar{d}$.)

For our reduction from S-Arrival to End-Of-Metered-Line we extend the above observation to partial switching flows that are not required to end at the destination. The vertices of the End-Of-Metered-Line graph created by our reduction correspond to partial switching flows in the S-Arrival instance. The directed edges connect partial run profiles to their natural successors and predecessors, i.e., the partial run extended or shortened by a single step of the train. Any switching flow that does not correspond to some partial run profile is an isolated vertex in the End-Of-Metered-Line graph. Finally, the trivial source is the empty switching flow, and the distance from it can be computed for any partial run simply as the number of steps taken by the train so far. Given that there is only a single path in the resulting End-Of-Metered-Line graph and that its sink is exactly the complete run, we get that the unique solution to the End-Of-Metered-Line instance gives us a solution for the original instance of S-Arrival.

To make the reduction efficiently computable, we need to address the verification of partial run profiles. As it turns out, partial run profiles can be efficiently verified using the graph G^*, in a similar way to complete run profiles discussed above. The main difference is that the graph of last-used edges for a partial run profile can contain a cycle, as the train might visit the same vertex multiple times on its route to the destination. However, we show that there is at most one cycle in G^*, which always contains the current end-vertex of the partial run. The complete characterization of partial run profiles (which covers also full run profiles) is given in Lemma 9, and the formal reduction is described in Section 4.2.1.

Finally, we show that every partial run profile is uniquely determined by its last-used edges and its end-vertex. This limits the size of the search space for the EOML instances that are produced by our reduction, which allows us to efficiently use Aldous' algorithm [1] to solve Arrival and S-Arrival.

2 Preliminaries

In the rest of the paper we use the following standard notation. For $k \in \mathbb{N}$, we denote by $[k]$ the set $\{1, \dots, k\}$. For a graph $G = (V, E)$, we reserve $n = |V|$ for the number of vertices. The basic object that we study are switch graphs, as defined by Dohrau et al. [5].

Definition 1 (switch graph)**.** A *switch graph* is a tuple $G = (V, E, s_0, s_1)$ where $s_0, s_1 \colon V \to V$ and $E = \{(v, s_0(v)), (v, s_1(v)) \mid \forall v \in V\}$.[1]

[1]Whenever $s_0(v) = s_1(v)$ for some vertex $v \in V$ we depict them as multiple edges in figures.

Algorithm 1: Run

```
Input  : a switch graph $G = (V, E, s_0, s_1)$ and two vertices $o, d \in V$
Output : for each edge $e \in E$, the number of times the train traversed $e$
$v \leftarrow o$                                                   // position of the train
$\forall u \in V$ set $s\_curr[u] \leftarrow s_0(u)$ and $s\_next[u] \leftarrow s_1(u)$
$\forall e \in E$ set $\boldsymbol{r}[e] \leftarrow 0$                                // initialize the run profile
$step \leftarrow 0$
while $v \neq d$ do
    $(v, w) \leftarrow s\_curr[v]$                                   // compute the next vertex
    $\boldsymbol{r}[s\_curr[v]]$++                                     // update the run profile
    swap($s\_curr[v], s\_next[v]$)
    $v \leftarrow w$                                                 // move the train
    $step \leftarrow step + 1$
return $\boldsymbol{r}$
```

In order to avoid cumbersome notation, we slightly overload the use of s_0, s_1 and treat both as functions from vertices to edges; that is by $s_b(v)$ we denote the edge $(v, s_b(v))$ for $b \in \{0, 1\}$. We use this convention throughout the paper unless stated otherwise.

The Arrival problem was formally defined by Dohrau et al. [5] as follows.

Definition 2 (Arrival [5])**.** Given a switch graph $G = (V, E, s_0, s_1)$ and two vertices $o, d \in V$, the Arrival problem is to decide whether the algorithm Run (Algorithm 1) terminates, i.e., whether the train reaches the destination d starting from the origin o.

To simplify theorem statements and our proofs, we assume without loss of generality that both $s_0(d)$ and $s_1(d)$ end in d.

A natural witness for termination of the Run procedure considered in previous work (e.g., [5]) is a switching flow. We extend the definition of a switching flow to allow for partial switching flows that do not necessarily end in the desired destination d.

Definition 3 ((partial) switching flow, end-vertex)**.** Let $G = (V, E, s_0, s_1)$ be a switch graph. For $o, d \in V$, we say that $\boldsymbol{f} \in \mathbb{N}^{2n}$ is a *switching flow from o to d* if the following two conditions hold.

Kirchhoff's Law (flow conservation):

$$\forall v \in V\colon \sum_{e=(u,v)\in E} \boldsymbol{f}_e - \sum_{e=(v,w)\in E} \boldsymbol{f}_e = [v = d] - [v = o]\,,$$

where $[\cdot]$ is the indicator variable of the event in brackets.

Parity Condition:

$$\forall v \in V\colon \boldsymbol{f}_{s_1(v)} \leq \boldsymbol{f}_{s_0(v)} \leq \boldsymbol{f}_{s_1(v)} + 1\,.$$

Kirchoff's law means that o emits one unit of flow, d absorbs one unit of flow, and at all other vertices, in-flow equals out-flow. If $d = o$, we have a circulation.

Given an instance $(G = (V, E, s_0, s_1), o, d)$ of Arrival, we say that $\boldsymbol{f}$ is a *switching flow* if it is a switching flow from o to d. A vector $\boldsymbol{f} \in \mathbb{N}^{2n}$ is called a *partial switching flow* iff $\boldsymbol{f}$ is a switching flow from o to v for some vertex $v \in V$. We say that v is the *end-vertex* of the partial switching flow. We denote the end-vertex of $\boldsymbol{f}$ by $v_{\boldsymbol{f}}$.

Definition 4 ((partial) run profile)**.** A *run profile* is the switching flow $\boldsymbol{r}$ returned by the algorithm Run (Algorithm 1) upon termination. A *partial run profile* is a partial switching flow corresponding to some intermediate value of $\boldsymbol{r}$ in the algorithm Run (Algorithm 1).

Observation 5 (Dohrau et al. [5, Observation 1])**.** Each (partial) run profile is a (partial) switching flow.

Observation 6. An end-vertex $v_{\boldsymbol{f}}$ of a switching flow $\boldsymbol{f}$ is computable in polynomial time.

Proof. It is sufficient to determine which vertex has a net in-flow of one. □

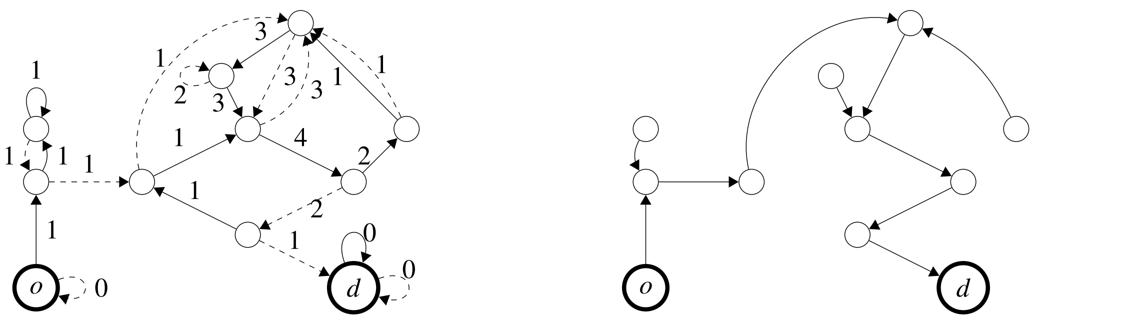
Figure 1: An example of a switch graph G with a switching flow $\boldsymbol{f}$ (on the left) and the corresponding graph $G^*_{\boldsymbol{f}}$ (on the right). The s_0 edges are denoted by full arrows and the s_1 edges are denoted by dashed arrows. For each edge in G, the switching flow $\boldsymbol{f}$ is specified by the adjacent integer value.

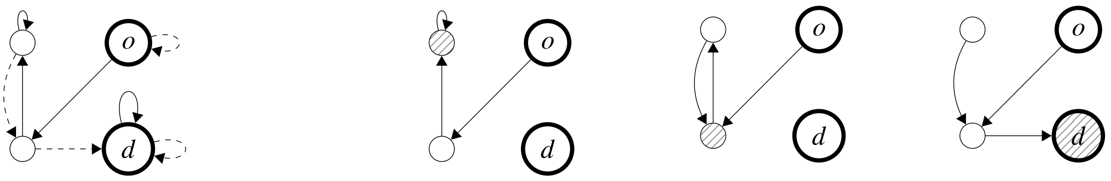
Figure 2: An example of a switch graph G and cycles in the graphs G^* corresponding to partial run profiles after 3, 4, and 5 steps of the train (respectively from left to right). We use hatching to highlight the current end-vertex.

3 The Complexity of Run Profile Verification

Dohrau et al. [5] proved that it is possible to efficiently verify whether a given vector is a switching flow. In this section we show that we can also efficiently verify whether a switching flow is a run profile. Combining this with the results by Dohrau et al. [5], we prove that the decision problem of Arrival is in UP ∩ coUP (see Section 4.1) and that the search problem of Arrival lies in the complexity class CLS (see Section 4.2). As outlined in Section 1.2, our approach for verification of run profiles is based on finding a cycle in a natural subgraph of the railway network G defined below. Specifically, we consider the subgraph of G that contains only the last visited outgoing edge of each vertex, i.e., every vertex has out-degree at most one (see Figure 1 for an illustration).

Definition 7 ($G^*_{\boldsymbol{f}}$)**.** Let $(G = (V, E, s_0, s_1), o, d)$ be an instance of Arrival, and let $\boldsymbol{f} \in \mathbb{N}^{2n}$ be a partial switching flow. We define a graph $G^*_{\boldsymbol{f}} = (V, E^*)$ as follows

$$\begin{aligned} E^* = &\{s_0(v) \colon \forall v \in V \text{ s.t. } \boldsymbol{f}(s_0(v)) \neq \boldsymbol{f}(s_1(v))\} \cup \\ &\{s_1(v) \colon \forall v \in V \text{ s.t. } \boldsymbol{f}(s_0(v)) = \boldsymbol{f}(s_1(v)) > 0\}\,. \end{aligned}$$

Observation 8. Given a partial switching flow $\boldsymbol{f}$, the graph $G^*_{\boldsymbol{f}}$ can be computed in polynomial time.

Lemma 9. *A partial switching flow $\boldsymbol{f}$ is a partial run profile iff $\boldsymbol{f}_{s_0(d)} = \boldsymbol{f}_{s_1(d)} = 0$ and one of the following two conditions holds:*

1. *There exists no cycle in $G^*_{\boldsymbol{f}}$.*
2. *There exists exactly one cycle in $G^*_{\boldsymbol{f}}$ and this cycle contains the end-vertex of $\boldsymbol{f}$.*

The main idea of the proof is based on the following fact: a switching flow f which is *not* a run profile must contain a circulation (as shown by Dohrau et al. [5]). Let f be a switching flow that we get from a run profile r by adding some flows on cycles, then the last added circulation (the last added cycle) must form a cycle in the corresponding graph G^*_f. On the other hand, a cycle containing the end-vertex is formed in G^*_f whenever the train arrives to a previously visited vertex. An illustration of the graph G^* at consecutive steps of the algorithm Run, with the corresponding evolution of the end-vertices and cycles, is given in Figure 2.

We prove the "⇐" implication of Lemma 9 by contradiction. Given a switching flow $\boldsymbol{f}$ we consider the longest run profile $\boldsymbol{r}$ that is everywhere at most $\boldsymbol{f}$. We denote the difference $\boldsymbol{f} - \boldsymbol{r}$ by $\boldsymbol{\delta}$. We utilize the following observation at two points in our proof. First, to prove that both $\boldsymbol{f}$ and $\boldsymbol{r}$ have the same end-vertex, that is $v_{\boldsymbol{r}} = v_{\boldsymbol{f}}$. Second, to prove that any cycle we find in the intersection of $G^*_{\boldsymbol{f}}$ and the non-zero edges of $\boldsymbol{\delta}$ avoids $v_{\boldsymbol{f}}$.

Observation 10. Let $\boldsymbol{f}$ be a switching flow and let $\boldsymbol{r}$ be the longest partial run profile such that all coordinates are at most $\boldsymbol{f}$ and denote $\boldsymbol{\delta} = \boldsymbol{f} - \boldsymbol{r}$. Then $\boldsymbol{\delta}_{s_0(v_{\boldsymbol{r}})} = \boldsymbol{\delta}_{s_1(v_{\boldsymbol{r}})} = 0$.

Proof of Observation 10. Suppose that either $\boldsymbol{\delta}_{s_0(v_{\boldsymbol{r}})} \neq 0$ or $\boldsymbol{\delta}_{s_1(v_{\boldsymbol{r}})} \neq 0$, and that $\boldsymbol{r}$ is such that the next edge for the train to continue on is $s_0(v_{\boldsymbol{r}})$. From the maximality of $\boldsymbol{r}$ we get that $\boldsymbol{\delta}_{s_0(v_{\boldsymbol{r}})}$ is equal to zero. Then we get

$$\begin{aligned}
\boldsymbol{f}_{s_1(v_{\boldsymbol{r}})} &> \boldsymbol{r}_{s_1(v_{\boldsymbol{r}})} && \text{(since } \boldsymbol{\delta}_{s_1(v_{\boldsymbol{r}})} \text{ is non-zero)}\\
&= \boldsymbol{r}_{s_0(v_{\boldsymbol{r}})} && \text{(from the parity condition for } \boldsymbol{r}\text{)}\\
&= \boldsymbol{f}_{s_0(v_{\boldsymbol{r}})} && \text{(from the maximality of } \boldsymbol{r}\text{)}
\end{aligned}$$

This leads to a contradiction with the parity condition of Definition 3 as $\boldsymbol{f}_{s_1(v_{\boldsymbol{r}})} > \boldsymbol{f}_{s_0(v_{\boldsymbol{r}})}$.

The other case is similar: suppose that the train should continue with the edge $s_1(v_{\boldsymbol{r}})$. From the maximality of $\boldsymbol{r}$ we get that $\boldsymbol{\delta}_{s_1(v_{\boldsymbol{r}})}$ is equal to zero. And thus we get

$$\begin{aligned}
\boldsymbol{f}_{s_0(v_{\boldsymbol{r}})} &> \boldsymbol{r}_{s_0(v_{\boldsymbol{r}})} && \text{(since } \boldsymbol{\delta}_{s_0(v_{\boldsymbol{r}})} \text{ is non-zero)}\\
&= \boldsymbol{r}_{s_1(v_{\boldsymbol{r}})} + 1 && \text{(from the parity condition for } \boldsymbol{r}\text{)}\\
&= \boldsymbol{f}_{s_1(v_{\boldsymbol{r}})} + 1 && \text{(from the maximality of } \boldsymbol{r}\text{)}
\end{aligned}$$

This gives a contradiction with the parity condition of Definition 3 as $\boldsymbol{f}_{s_0(v_{\boldsymbol{r}})} > \boldsymbol{f}_{s_1(v_{\boldsymbol{r}})} + 1$. □

Proof of Lemma 9. We start by proving that any partial run profile $\boldsymbol{r}$ has at most one cycle and that this cycle always contains the end-vertex $v_{\boldsymbol{r}}$. We proceed by induction on the length of the run profile, i.e., the number of steps in the algorithm Run. The base case is the initial run profile 0^{2n} for which $G^*_{0^{2n}}$ contains no cycle. For the induction step, let us assume that a vector $\boldsymbol{r}^i$ is a partial run profile and the vector $\boldsymbol{r}^{i+1}$ is the partial run profile after one step from $\boldsymbol{r}^i$.

If $G^*_{\boldsymbol{r}^i}$ contains a cycle then removing the edge from the end-vertex $v_{\boldsymbol{r}^i}$ to its successor in $G^*_{\boldsymbol{r}^i}$ produces a cycle-free graph. Adding an edge between $v_{\boldsymbol{r}^i}$ and $v_{\boldsymbol{r}^{i+1}}$ creates a graph with at most one cycle. Moreover, if there is a cycle then it has to contain $v_{\boldsymbol{r}^{i+1}}$.

To prove the reverse implication, assume we have a partial switching flow $\boldsymbol{f}$ that satisfies the conditions from the statement of this lemma, i.e., $G^*_{\boldsymbol{f}}$ has at most one cycle and if the cycle is present it contains the end-vertex $v_{\boldsymbol{f}}$. Let us denote by $\boldsymbol{r}$ the longest partial run profile such that all its coordinates are at most $\boldsymbol{f}$, that is $\boldsymbol{r}_e \leq \boldsymbol{f}_e$ for each $e \in E$. Consider the difference $\boldsymbol{\delta} = \boldsymbol{f} - \boldsymbol{r}$ of the switching flow and its longest run profile. We complete the proof by the following case analysis:

1. If the end-vertex $v_{\boldsymbol{f}}$ is the same as the end-vertex $v_{\boldsymbol{r}}$ ($v_{\boldsymbol{f}} = v_{\boldsymbol{r}}$), then the difference $\boldsymbol{\delta} = \boldsymbol{f} - \boldsymbol{r}$ is a flow, and moreover it is a circulation. That is the Kirchhoff's law is satisfied in all vertices of G:

$$\forall v \in G\colon \sum_{e=(u,v)\in E} \boldsymbol{\delta}_e - \sum_{e=(v,w)\in E} \boldsymbol{\delta}_e = 0\,.$$

   If $\boldsymbol{\delta}$ is identically zero we are done as $\boldsymbol{f} = \boldsymbol{r}$, and thus $\boldsymbol{f}$ is a partial run profile.

   Otherwise, we show that the circulation $\boldsymbol{\delta}$ will result in a cycle in $G^*_{\boldsymbol{f}}$. Namely we prove that $G^*_{\boldsymbol{f}}$ contains an outgoing edge from v with a non-zero value in $\boldsymbol{\delta}$ iff v has a non-zero outgoing edge in $\boldsymbol{\delta}$. Using this and the fact that $\boldsymbol{\delta}$ satisfies the Kirchhoff's law everywhere, we find a cycle in $G^*_{\boldsymbol{f}}$.

   Let u be any vertex of G such that only one of the edges $s_0(v)$ and $s_1(v)$ is non-zero in $\boldsymbol{\delta}$. We claim that then the non-zero edge is contained in the graph $G^*_{\boldsymbol{f}}$. There are only two possible cases:

   (a) Either $\boldsymbol{r}_{s_0(u)} = \boldsymbol{f}_{s_0(u)}$, and thus $\boldsymbol{r}_{s_0(u)} = \boldsymbol{r}_{s_1(u)} - 1$ and $\boldsymbol{f}_{s_0(u)} = \boldsymbol{f}_{s_1(u)}$ (see Table 1a), and $G^*_{\boldsymbol{f}}$ contains the edge $s_1(u)$,

   (b) or $\boldsymbol{r}_{s_1(u)} = \boldsymbol{f}_{s_1(u)}$, and thus $\boldsymbol{r}_{s_0(u)} = \boldsymbol{r}_{s_1(u)}$ and $\boldsymbol{f}_{s_0(u)} = \boldsymbol{f}_{s_1(u)} + 1$ (see Table 1b), and $G^*_{\boldsymbol{f}}$ contains the edge $s_0(u)$.

	$s_0(u)$	$s_1(u)$
$\boldsymbol{\delta}$	0	1
$\boldsymbol{r}$	a	$a-1$
$\boldsymbol{f}$	a	a

(a) Case (1.a).

	$s_0(u)$	$s_1(u)$
$\boldsymbol{\delta}$	1	0
$\boldsymbol{r}$	a	a
$\boldsymbol{f}$	$a+1$	a

(b) Case (1.b).

Table 1: Case analysis from the proof of Lemma 9 when only one of the edges $s_0(u)$ and $s_1(u)$ is non-zero in $\boldsymbol{\delta}$.

On the other hand, if both $s_0(u)$ and $s_1(u)$ are non-zero in $\boldsymbol{\delta}$ then $G^*_{\boldsymbol{f}}$ contains one of them. Thus we may choose any non-zero edge in $\boldsymbol{\delta}$ which is in $G^*_{\boldsymbol{f}}$ and proceed by another adjacent non-zero directed edge (in $\boldsymbol{\delta}$) in $G^*_{\boldsymbol{f}}$ until we construct a directed cycle in $G^*_{\boldsymbol{f}}$.

By the Observation 10, we know that all outgoing edges from $v_{\boldsymbol{r}} = v_{\boldsymbol{f}}$ are zero in $\boldsymbol{\delta}$, and thus the end-vertex $v_{\boldsymbol{f}}$ is not contained in the cycle we have found.

2. If the end-vertex $v_{\boldsymbol{f}}$ is not the same as the end-vertex $v_{\boldsymbol{r}}$ ($v_{\boldsymbol{f}} \neq v_{\boldsymbol{r}}$) then we get a contradiction with $\boldsymbol{f}$ being a partial switching flow. It follows from Observation 10 that

$$\sum_{e=(u,v_{\boldsymbol{r}})\in E} \boldsymbol{f}_e - \sum_{e=(v_{\boldsymbol{r}},w)\in E} \boldsymbol{f}_e \geq \sum_{e=(u,v_{\boldsymbol{r}})\in E} \boldsymbol{r}_e - \sum_{e=(v_{\boldsymbol{r}},w)\in E} \boldsymbol{r}_e = \begin{cases} 0 & v_{\boldsymbol{r}} = o, \\ 1 & \text{otherwise,} \end{cases}$$

which is in contradiction with $\boldsymbol{f}$ being a partial switching flow for which the end-vertex $v_{\boldsymbol{f}}$ differs from $v_{\boldsymbol{r}}$.

This concludes the proof of Lemma 9. □

Lemma 11. It is possible to verify in polynomial time whether a vector is a run profile.

Proof. We can check that a vector $\boldsymbol{f}$ is a switching flow in polynomial time due to Dohrau et al. [5]. The construction of the graph $G^*_{\boldsymbol{f}}$ is polynomial by Observation 8. Lemma 9 gives us a polynomial time procedure to check if $\boldsymbol{f}$ is also a run profile as it is sufficient to check if $G^*_{\boldsymbol{f}}$ contains more than one cycle or whether it has a cycle not containing the end-vertex. This check can be done by a simple modification of the standard depth-first search on $G^*_{\boldsymbol{f}}$. □

4 The Computational Complexity of Arrival

In this section we use our efficient structural characterization of run profiles from Lemma 9 to improve the known results about the computational complexity of Arrival. Specifically, we show that the decision version of Arrival is in UP ∩ coUP and the search version is in CLS.

4.1 The Decision Complexity of Arrival

Our upper bound on the decision complexity of Arrival follows directly from the work of Dohrau et al. [5] by application of Lemma 11.

Theorem 12. Arrival is in UP ∩ coUP.

Proof. The unique UP certificate for a YES-instance of Arrival is the run profile $\boldsymbol{r}$ returned by the algorithm Run. Clearly, for each YES-instance there exists only one such vector $\boldsymbol{r}$ and $\boldsymbol{r}$ does not exist for NO-instances. By Lemma 11, we can determine whether a candidate switching flow $\boldsymbol{r}$ is a run profile in polynomial time.

The coUP membership follows directly from the reduction of NO-instances of Arrival to YES-instances of Arrival as suggested by Dohrau et al. [5]. The reduction adds to the original graph G a new vertex $\bar{d}$, and for each vertex $v \in V$ such that there is no directed path from v to the destination d, the edges $s_0(v)$ and $s_1(v)$ are replaced with edges $(v, \bar{d})$. This alteration of the original switch graph can be performed in polynomial time. Dohrau et al. [5] proved that the train eventually arrives either at d or $\bar{d}$. The unique coUP witness for Arrival is then a run profile from o to the dead-end $\bar{d}$. □

4.2 The Search Complexity of Arrival

The search complexity of Arrival was first studied by Karthik C. S. [13], who introduced a total search variant of Arrival as follows.

Definition 13 (S-Arrival [13]). Given a switch graph $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$, define a graph G' as follows:

1. Add a new vertex $\bar{d}$.
2. For each vertex v such that there is no directed path from v to d, replace edges $s_0(v)$ and $s_1(v)$ with edges $(v, \bar{d})$.
3. Edges $s_0(d), s_1(d), s_0(\bar{d})$, and $s_1(\bar{d})$ are self-loops.

The problem S-Arrival is to find a switching flow in G' either from o to d or from o to $\bar{d}$.

The above Definition 13 is motivated by the proof of membership in $\mathsf{NP} \cap \mathsf{coNP}$ by Dohrau et al. [5]. Namely, in order to ensure that a solution for S-Arrival always exists, it was necessary to add to the switch graph G the dead-end vertex $\bar{d}$.

Note that our method for efficient verification of run profiles from Lemma 11 allows us to define a more natural version of S-Arrival directly on the graph G without any modifications. Instead of relying on the dead-end vertices, we can use the fact that a partial run profile with an edge that was visited for $2^n + 1$ times is an efficiently verifiable witness for NO-instances of Arrival.

Definition 14 (S-Arrival - simplified). Given a switch graph $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$, the S-Arrival problem asks us to find one of the following:

1. a run profile $\boldsymbol{r} \in \mathbb{N}^{[2n]}$ from o to d, or
2. a run profile $\boldsymbol{r} \in \mathbb{N}^{[2n]}$ from o to any $v \in V$ such that
   - $\boldsymbol{r}_{(u,v)} = 2^n + 1$, where u is the last vertex visited by the train before it reached the end-vertex v of $\boldsymbol{r}$, and
   - $\boldsymbol{r}_{e'} \leq 2^n$ for all $e' \neq (u, v)$.

The correspondence of the above version of S-Arrival to the original one follows formally from the following lemma.

Lemma 15 (Karthik C. S. [13, Lemma 1]). For any $G = (V, E, s_0, s_1)$ and a pair of vertices $o, d \in V$. Let $\boldsymbol{r}$ be a run profile (thus $v_{\boldsymbol{r}} = d$), then $\boldsymbol{r}_e \leq 2^n$ for each edge $e \in E$.

To argue membership of our version of S-Arrival in TFNP, we need to show that both types of solutions in Definition 14 can be verified efficiently. Solutions of the first type are simply run profiles, and we have already shown that they can be verified in polynomial time in Lemma 11. In order to be able to verify solutions of the second type, it remains to argue that for any partial run profile, the immediate predecessor of its end-vertex can be determined in polynomial time.

Lemma 16. Let $\boldsymbol{r}$ be a partial run profile after $R \geq 1$ steps and u be the vertex visited by the train at step $R - 1$. Then

1. either u is the unique predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$, or
2. there is a single cycle in $G^*_{\boldsymbol{r}}$ containing $v_{\boldsymbol{r}}$ and u is the predecessor of $v_{\boldsymbol{r}}$ on this cycle.

Proof. First, note that if u is the end-vertex one step before $v_{\boldsymbol{r}}$ becomes the end-vertex then $G^*_{\boldsymbol{r}}$ must contain the edge $(u, v_{\boldsymbol{r}})$, as it is the last edge used by the train to leave u. Thus, in the first case the immediate predecessor of $v_{\boldsymbol{r}}$ in the partial run $\boldsymbol{r}$ is unambiguously given by the only predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$.

For the second case we show that $G^*_{\boldsymbol{r}}$ contains a directed cycle C (containing the end-vertex $v_{\boldsymbol{r}}$) and u is unambiguously given by the predecessor of $v_{\boldsymbol{r}}$ in $G^*_{\boldsymbol{r}}$ that lies on C. We find the cycle C by constructing the longest possible directed path $c_0 = v_{\boldsymbol{r}}, c_1, \ldots, c_k$ in $G^*_{\boldsymbol{r}}$ without repeating vertices. Note that it cannot happen that c_k has no outgoing edge in $G^*_{\boldsymbol{r}}$. Otherwise, $\boldsymbol{r}$ would have two different end-vertices $v_{\boldsymbol{r}}$ and c_k (as having no outgoing edge in $G^*_{\boldsymbol{r}}$ means that the train has never left this vertex). By Lemma 9, the directed edge from c_k has to end in the end-vertex $v_{\boldsymbol{r}}$, or else there would be a cycle in $G^*_{\boldsymbol{r}}$ that avoids $v_{\boldsymbol{r}}$.

The algorithm Run takes R steps to generate the run profile $\boldsymbol{r}$, i.e., $\sum_{e\in E} \boldsymbol{r}_e = R$. Let $t_{\boldsymbol{r}} : V \to \{0, 1, \dots, R-1\}$ be the function returning the last step after which a vertex was left by the train in the partial run profile $\boldsymbol{r}$. Observe that, except for the edge through which the train arrived to $v_{\boldsymbol{r}}$,[2] it holds for all edges $(x, y) \in G^*_{\boldsymbol{r}}$ that $t_{\boldsymbol{r}}(x) < t_{\boldsymbol{r}}(x) + 1 \leq t_{\boldsymbol{r}}(y)$. However, the above inequality cannot hold for all edges on the cycle C, and thus C has to contain the last used edge and the train had to be in c_k at step $R-1$. □

Observation 17. S-Arrival from Definition 13 reduces to simplified S-Arrival from Definition 14.

Proof. Given a solution of the second type of the simplified S-Arrival, i.e., the long run profile, we can get a run profile $\boldsymbol{r}$ to $\bar{d}$ in polynomial time. For each vertex u we can determine whether there is an oriented path from it to the destination d, and if there is no such path we set $\boldsymbol{r}_{s_0(v)} = \boldsymbol{r}_{s_1(v)} = 0$. We compute the end vertex $v_{\boldsymbol{r}}$ and set $s_0(v_{\boldsymbol{r}}) = 1$. All other components of $\boldsymbol{r}$ are set according to the original solution of the simplified S-Arrival. □

4.2.1 S-Arrival is in CLS

Karthik C. S. [13] showed that S-Arrival is contained in the class PLS. We improve this result and prove that S-Arrival is in fact contained in CLS. As a by-product, we also obtain a randomized algorithm for S-Arrival with runtime $O(1.4143^n)$ which is the first algorithm for this problem with expected runtime $O(c^n)$ for $c < 2$.

The class of total search problems that are amenable to "continuous" local search was defined by Daskalakis and Papadimitriou [4] using the following canonical problem.

Definition 18 (CLS [4])**.** CLS is the class of total search problems reducible to the following problem called CLOpt.

Given two arithmetic circuits $f\colon [0,1]^3 \to [0,1]^3$ and $p\colon [0,1]^3 \to [0,1]$, and two real constants $\varepsilon, \lambda > 0$, find either a point $x \in [0,1]^3$ such that $p(f(x)) \leq p(x) + \varepsilon$ or a pair of points $x, x' \in [0,1]^3$ certifying that either p or f is not λ-Lipschitz.

Instead of working with CLOpt, we use as a gateway for our reduction a problem called End-Of-Metered-Line (EOML) which was recently defined and shown to lie in CLS by Hubáček and Yogev [8].

Definition 19 (End-Of-Metered-Line)**.** Given circuits $S, P\colon \{0,1\}^m \to \{0,1\}^m$, and $V\colon \{0,1\}^m \to [2^m] \cup \{0\}$ such that $P(0^m) = 0^m \neq S(0^m)$ and $V(0^m) = 1$, find a string $x \in \{0,1\}^m$ satisfying one of the following:

1. either $P(S(x)) \neq x$ or $S(P(x)) \neq x \neq 0^m$,
2. $x \neq 0^m$ and $V(x) = 1$,
3. either $V(x) > 0$ and $V(S(x)) - V(x) \neq 1$ or $V(x) > 1$ and $V(x) - V(P(x)) \neq 1$.

The circuits S, P from Definition 19 implicitly represent a directed graph with vertices labelled by binary strings of length m, where each vertex has both out-degree and in-degree at most one. The circuit P represents the predecessor and the circuit S represents the successor of a given vertex as follows: there is an edge from a vertex u to a vertex v iff $S(u) = v$ and $P(v) = u$. Finally, the circuit V can be thought of as an odometer that returns the distance from the trivial source at 0^m or value 0 for vertices lying off the path starting at the trivial source. The task in End-Of-Metered-Line is to find a sink or a source different from the trivial one at 0^m (the solutions of the second and of the third type in Definition 19 ensure that V behaves as explained above).

We are now ready to present our reduction from S-Arrival to End-Of-Metered-Line.

Theorem 20. S-Arrival can be reduced to End-Of-Metered-Line, and thus it is contained in CLS.

Proof. Let (G, o, d) be an instance of S-Arrival. We construct an instance of EOML that contains a vertex for each candidate partial switching flow over the switch graph G, i.e., for each vector with $2n$ coordinates and values from $[2^n + 1] \cup \{0\}$. The EOML instance will comprise of a directed path starting at the initial (empty) partial run profile 0^{2n}. Each vertex on the path has an outgoing edge to its consecutive partial run profile. Any vertex that does not correspond to a partial run profile becomes a self-loop. Finally, the valuation circuit V returns either the number of steps in the corresponding partial run profile or the zero value if the vertex does not correspond to a partial run profile.

Algorithm 2: Successor

```
Input  : a vector $\boldsymbol{x} \in ([2^n + 1] \cup \{0\})^{2n}$
Output : a vector $\boldsymbol{y} \in ([2^n + 1] \cup \{0\})^{2n}$
1  if $\boldsymbol{x}$ is a partial run profile then              // efficiently testable by Lemma 11
2      compute the end-vertex $v_{\boldsymbol{x}}$
3      if $v_{\boldsymbol{x}} = d$ or $\boldsymbol{x}_e = 2^n + 1$ for some $e \in E$ then
4          return $\boldsymbol{x}$                                  // the train terminates or runs for too long
5      else
6          $b \leftarrow \boldsymbol{x}(s_1(v_{\boldsymbol{x}})) - \boldsymbol{x}(s_0(v_{\boldsymbol{x}}))$    // parity of the current visit at $v_{\boldsymbol{x}}$
7          $e \leftarrow s_b(v_{\boldsymbol{x}})$                     // the next edge to traverse
8          $y_h \leftarrow \begin{cases} \boldsymbol{x}_h + 1 & \text{if } h = e \\ \boldsymbol{x}_h & \text{otherwise} \end{cases}$    // run profile update
9          return $\boldsymbol{y}$
10 else
11     return $\boldsymbol{x}$                                      // self-loop
```

Algorithm 3: Predecessor

```
Input  : a vector $\boldsymbol{x} \in ([2^n + 1] \cup \{0\})^{2n}$
Output : a vector $\boldsymbol{y} \in ([2^n + 1] \cup \{0\})^{2n}$
1  if $\boldsymbol{x} = \boldsymbol{0}$ then
2      return $\boldsymbol{0}$                                      // the trivial source
3  if $\boldsymbol{x}$ is a partial run profile then              // efficiently testable by Lemma 11
4      compute the end-vertex $v_{\boldsymbol{x}}$
       // correctness by Lemma 16
5      if $v_{\boldsymbol{x}}$ has a single predecessor in $G^*_{\boldsymbol{x}}$ then
6          $e \leftarrow$ the only incoming edge of $v_{\boldsymbol{x}}$ in $G^*_{\boldsymbol{x}}$
7      else
8          $e \leftarrow$ the only incoming edge of $v_{\boldsymbol{x}}$ which lies on a directed cycle
9      if $\boldsymbol{x}_e \leq 2^n + 1$ and $\boldsymbol{x}_{e'} < 2^n + 1$ for all $e' \neq e$ then
10         $y_h \leftarrow \begin{cases} \boldsymbol{x}_h - 1 & \text{if } h = e \\ \boldsymbol{x}_h & \text{otherwise} \end{cases}$
11         return $\boldsymbol{y}$
12 return $\boldsymbol{x}$                                          // self-loop
```

Algorithm 4: Valuation

```
Input  : a vector $\boldsymbol{x} \in ([2^n + 1] \cup \{0\})^{2n}$
Output : a value $v \in \mathbb{N}$
1  $v \leftarrow 0$                                                 // default value for self-loops
2  if $\boldsymbol{x}$ is a partial run profile then              // efficiently testable by Lemma 11
3      $v \leftarrow 1 + \sum_{i=1}^{2n} \boldsymbol{x}_i$
4  return $v$
```

Formal description of the circuits S, P, and V defining the above EOML graph is given by algorithms Successor (Algorithm 2), Predecessor (Algorithm 3), and Valuation (Algorithm 4).

A polynomial bound on the size of the circuits S, P, and V follows directly from Observation 8 (computing G^*), Lemma 11 (testing whether a given vector is a partial run profile), Observation 6 (computing the end-vertex), and Lemma 16 (computing the previous position of the train).

[2]The inequality does not hold for $v_{\boldsymbol{r}}$, since $t(v_{\boldsymbol{r}})$ has not been updated to the time R yet.

Lemma 9 and Lemma 16 imply that the EOML graph indeed consists of a single directed path and isolated vertices with self-loops. By the construction of V (it outputs the number of steps of the train), there are no solutions of the second or the third type (cf. Definition 19). Thus, the EOML instance has a unique solution which has to correspond to a run profile in the original S-Arrival instance or to a partial run profile certifying that the train ran for too long (see the second type of solution in Definition 14). □

5 An $\tilde{O}(1.4143^n)$ Algorithm for S-Arrival

Consider any problem that can be put into the complexity class PLS, i.e., can be reduced to the canonical PLS-complete problem LocalOpt (see also [13, Definition 1]):

Definition 21 (LocalOpt)**.** Given circuits $S\colon \{0,1\}^m \to \{0,1\}^m$, and $V\colon \{0,1\}^m \to [2^m] \cup \{0\}$, find a string $x \in \{0,1\}^m$ such that $V(x) \geq V(S(x))$.

Aldous [1] introduced the following simple algorithm that can be used to solve LocalOpt: pick $2^{m/2}$ binary strings uniformly and independently at random from $\{0,1\}^m$, and let $x_{\max}$ be the selected string that maximizes the value $V(x)$. Starting from $x = x_{\max}$, repeatedly move to the successor $S(x)$, until $V(x) \geq V(S(x))$. He showed that the expected number of circuit evaluations performed by the algorithm before finding a local optimum is at most $O(m2^{m/2})$. Note that in the case of EOML it is possible that all sampled solutions are isolated vertices in the EOML graph. In this case the 0^m string has the best known value, and the search is started from there.

In the case of S-Arrival, the PLS membership proof of Karthik C. S. constructs the circuits for successor and valuation with $O(n^2)$ input bits [13, Theorem 2]; the EOML instance constructed in Theorem 20—proving CLS and in particular PLS membership—yields circuits of $O(n^2)$ input bits as well. This number is too high to yield a randomized algorithm of non-trivial runtime.

This number of $O(n^2)$ input bits comes from the obvious encoding of a partial run profile: each of the $2n$ edges has a nonnegative integer flow value of at most $2^n + 1$. But in fact, a *terminating* run has at most $n2^n$ partial run profiles, as no *vertex-state pair* (v, s_curr) can repeat in Algorithm 1. In other words, a partial run profile $\boldsymbol{f}$ is determined by its end-vertex $v_{\boldsymbol{f}}$ as well as the positions of all switches at the time of the corresponding visit of $v_{\boldsymbol{f}}$. This means that a partial run profile can be encoded with $n + \log_2 n$ bits, and if we had a PLS or CLS membership proof of S-Arrival with circuits of this many inputs only, we could solve S-Arrival in time $O(\text{poly}(n) \times 2^{n/2})$.

Next, we show that such membership proofs indeed exist. For this, we show that the above encoding (of a partial run profile by an end-vertex and the positions of all switches) can be efficiently decoded: given an end-vertex and the positions of all switches, we can efficiently compute a unique candidate for a corresponding partial run profile. The resulting encoding and decoding circuits can be composed with the ones in Theorem 20 to obtain PLS and CLS membership proofs with circuits of $n + \log_2 n$ input bits, and hence yield a randomized algorithm of runtime $O(\text{poly}(n) \times 2^{n/2}) = \tilde{O}(1.4143^n)$, as explained above.

5.1 Decoding Partial Run Profiles

We work with instances of S-Arrival as in Definition 13, i.e., there is always a run profile either to d or to $\bar{d}$. This is without loss of generality [5, 13].

Definition 22. Let $G = (V, E, s_0, s_1)$ be a switch graph, and let $o, d, \bar{d} \in V$ be as in Definition 13. The *parity* of a run profile $\boldsymbol{f} \in \mathbb{N}^{2n}$ is the vector $\boldsymbol{p}_{\boldsymbol{f}} \in \{0,1\}^{n-2}$ defined by $\boldsymbol{p}_v = \boldsymbol{f}_{s_0(v)} - \boldsymbol{f}_{s_1(v)} \in \{0,1\}$, $v \in V \setminus \{d, \bar{d}\}$.

Note that we do not care about (the parity of) the switches at d and $\bar{d}$, since the algorithm stops as soon as d or $\bar{d}$ is reached.

Here is the main result of this section. For a given target vertex $t \in V$ and given parity $\boldsymbol{p}$, there is exactly one candidate for a partial run profile from o to t with parity $\boldsymbol{p}$. Moreover, this candidate can be computed by solving a system of linear equations.

Lemma 23. Let $(G, o, d, \bar{d})$ be an instance of S-Arrival, let $t \in V$ and $\boldsymbol{p} \in \{0,1\}^{n-2}$. Then there exists exactly one vector $\boldsymbol{f} \in \mathbb{R}^{2(n-2)}$ such that the following conditions hold.

Kirchhoff's Law (flow conservation):

$$\forall v \in V \setminus \{d, \bar{d}\}\colon \sum_{e=(u,v)\in E} \boldsymbol{f}_e - \sum_{e=(v,w)\in E} \boldsymbol{f}_e = [v = t] - [v = o] \tag{1}$$

where $[\cdot]$ is the indicator variable of the event in brackets.

Parity Condition: $\boldsymbol{p}_f = \boldsymbol{p}$, i.e.,

$$\forall v \in V \setminus \{d, \bar{d}\} \colon \boldsymbol{f}_{s_0(v)} - \boldsymbol{f}_{s_1(v)} = \boldsymbol{p}_v. \tag{2}$$

Before we prove Lemma 23, let us draw a crucial conclusion: The unique partial run profile $\boldsymbol{f} \in \mathbb{N}^{2n}$ with end-vertex t and parity $\boldsymbol{p}$ (if such f exists – note that we are only guaranteed a real-valued $\boldsymbol{f}$) necessarily satisfies (1) and (2). Hence, we may use Lemma 23 to get the entries $\boldsymbol{f}_e$ for all edges except the ones leaving d and $\bar{d}$. Only if all the entries are nonnegative and integral and satisfy (1) at d and $\bar{d}$ (under $\boldsymbol{f}_{s_0(d)} = \boldsymbol{f}_{s_1(d)} = \boldsymbol{f}_{s_0(\bar{d})} = \boldsymbol{f}_{s_1(\bar{d})} = 0$) do we have a candidate for a partial run profile. Hence, there is a unique candidate, and given t and a $\boldsymbol{p}$, this candidate can be efficiently found.

Proof of Lemma 23. Set $m = 2(n-2)$, $V' = V \setminus \{d, \bar{d}\}$ and let $A \in \mathbb{Z}^{m\times m}$ be the coefficient matrix of the linear system (1), (2) in the variables $\boldsymbol{f}_e$. We show that A is invertible.

Let $\boldsymbol{q} \in \mathbb{R}^m$ be the vector such that $\boldsymbol{q}_{(v,s_i(v))} = -1$ if $s_i(v) = d$ and $\boldsymbol{q}_{(v,s_i(v))} = 0$ otherwise. We show that $\boldsymbol{q}$ can be expressed as a linear combination of the rows of A in a unique way, from which invertibility of A and the statement of the lemma follow.

Let us use coefficients λ_v for each $v \in V'$ for the rows corresponding to the flow conservation constraints (1), and coefficients μ_v for each $v \in V'$ for the rows corresponding to the parity constraints (2). The column of A corresponding to variable $\boldsymbol{f}_{(v,s_i(v))}$, has a -1 entry from the flow conservation constraint at v, and a 1 entry (if $i = 0$) or a -1 entry (if $i = 1$) from the parity constraint at v. If $s_i(v) \neq d, \bar{d}$, there is another 1 entry from the flow conservation constraint at $s_i(v)$. All other entries are zero. The equations that express $\boldsymbol{q}$ as a linear combination of rows of A are therefore the following.

$$\begin{aligned}
\forall v \in V' : \quad -\lambda_v + \mu_v + \lambda_{s_0(v)} \cdot [s_0(v) \neq d, \bar{d}] &= \boldsymbol{q}_{(v,s_0(v))}, \\
\forall v \in V' : \quad -\lambda_v - \mu_v + \lambda_{s_1(v)} \cdot [s_1(v) \neq d, \bar{d}] &= \boldsymbol{q}_{(v,s_1(v))}.
\end{aligned}$$

Or equivalently:

$$\lambda_v - \mu_v = \begin{cases} \lambda_{s_0(v)}, & \text{if } s_0(v) \neq d, \bar{d} \\ 1, & \text{if } s_0(v) = d \\ 0, & \text{if } s_0(v) = \bar{d} \end{cases} \qquad v \in V', \tag{3}$$

$$\lambda_v + \mu_v = \begin{cases} \lambda_{s_1(v)}, & \text{if } s_1(v) \neq d, \bar{d} \\ 1, & \text{if } s_1(v) = d \\ 0, & \text{if } s_1(v) = \bar{d} \end{cases} \qquad v \in V'. \tag{4}$$

We now show that there are unique coefficients λ_v, μ_v satisfying these equations. Let us define $\lambda_d = 1$ and $\lambda_{\bar{d}} = 0$. Adding corresponding equations of (3) and (4) then yields

$$\lambda_d = 1, \quad \lambda_{\bar{d}} = 0, \quad \lambda_v = \frac{1}{2}\left(\lambda_{s_0(v)} + \lambda_{s_1(v)}\right), \quad \forall v \in V'.$$

These are exactly the equations for the vertex values in a *stoppping simple stochastic game* on the graph G with only average or degree-1 vertices and sinks d and $\bar{d}$ (stopping means that d or $\bar{d}$ are reachable from everywhere which is exactly what we require in a switch graph). Condon proved that these values are unique [3]. This also determines the μ_v's uniquely. □

6 Conclusion and Open Problems

We showed that candidate run profiles in Arrival can be efficiently verified due to their structure. This allowed us to improve the known upper bounds for the search complexity of Arrival and S-Arrival. Here we mention some natural questions arising from our work.

- Are there any non-trivial graph properties that make Arrival or S-Arrival efficiently solvable? Given that we currently do not know of any polynomial time algorithm for Arrival on general switch graphs, we could study the complexity of Arrival on some interesting restricted classes of switch graphs.
- Are there other natural problems in UP ∩ coUP such that their corresponding search variant is reducible to EOML? Does End-Of-Metered-Line capture the computational complexity of any TFNP problem with unique solution? Fearnley et al. [7] recently gave a reduction from the PLCP to EOML. Given that Arrival and PLCP can be both reduced to EOML, yet another intriguing question is whether there exists any reduction between the two.

- As mentioned in Section 1.1, the reduction from PLCP to EOML by Fearnley et al. [7] implies that PLCP can be solved faster with Aldous' algorithm [1] than with any other known algorithm. It would be interesting to see whether Aldous' algorithm can similarly give improved runtimes for other problems than ARRIVAL and PLCP.

Acknowledgements. We wish to thank Karthik C. S. for helpful discussions and suggestions.

References

[1] David Aldous. Minimization algorithms and random walk on the d-cube. *The Annals of Probability*, 11(2):403–413, 1983.

[2] Josh Buresh-Oppenheim and Tsuyoshi Morioka. Relativized NP search problems and propositional proof systems. In *19th Annual IEEE Conference on Computational Complexity (CCC 2004), 21-24 June 2004, Amherst, MA, USA*, pages 54–67, 2004.

[3] Anne Condon. The complexity of stochastic games. *Information and Computation*, 96(2):203 – 224, 1992.

[4] Constantinos Daskalakis and Christos H. Papadimitriou. Continuous local search. In *Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25, 2011*, pages 790–804, 2011.

[5] 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, 2017.

[6] John Fearnley, Martin Gairing, Matthias Mnich, and Rahul Savani. Reachability switching games. *CoRR*, abs/1709.08991, 2017. URL: `http://arxiv.org/abs/1709.08991`.

[7] John Fearnley, Spencer Gordon, Ruta Mehta, and Rahul Savani. CLS: new problems and completeness. *CoRR*, abs/1702.06017, 2017. URL: `http://arxiv.org/abs/1702.06017`.

[8] Pavel Hubáček and Eylon Yogev. Hardness of continuous local search: Query complexity and cryptographic lower bounds. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2017, Barcelona, Spain, Hotel Porta Fira, January 16-19*, pages 1352–1371, 2017.

[9] David S. Johnson, Christos H. Papadimitriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37(1):79–100, 1988.

[10] Bastian Katz, Ignaz Rutter, and Gerhard J. Woeginger. An algorithmic study of switch graphs. *Acta Inf.*, 49(5):295–312, 2012.

[11] Tsuyoshi Morioka. Classification of search problems and their definability in bounded arithmetic. *Electronic Colloquium on Computational Complexity (ECCC)*, (082), 2001.

[12] Christos H. Papadimitriou. On the complexity of the parity argument and other inefficient proofs of existence. *J. Comput. Syst. Sci.*, 48(3):498–532, 1994.

[13] Karthik C. S. Did the train reach its destination: The complexity of finding a witness. *Inf. Process. Lett.*, 121:17–21, 2017.