Polynomial Time Algorithm for ARRIVAL on Tree-Like Multigraphs

David Auger
DAVID Lab.,UVSQ, Université Paris Saclay, 45 avenue des Etats-Unis, 78000, Versailles, France

Pierre Coucheney
DAVID Lab.,UVSQ, Université Paris Saclay, 45 avenue des Etats-Unis, 78000, Versailles, France

Loric Duhazé
DAVID Lab., UVSQ, Université Paris Saclay, 45 avenue des Etats-Unis, 78000, Versailles, France
LISN, Université Paris Saclay, France

Abstract

A *rotor walk* in a directed graph can be thought of as a deterministic version of a Markov Chain, where a pebble moves from vertex to vertex following a simple rule until a terminal vertex, or sink, has been reached. The *ARRIVAL problem*, as defined by Dohrau et al. [8], consists in determining which sink will be reached. While the walk itself can take an exponential number of steps, this problem belongs to the complexity class NP ∩ co-NP without being known to be in P. In this work, we define a class of directed graphs, namely *tree-like multigraphs*, which are multigraphs having the global shape of an undirected tree. We prove that in this class, ARRIVAL can be solved in almost linear time, while the number of steps of a rotor walk can still be exponential. Then, we give an application of this result to solve some deterministic analogs of stochastic models (e.g., Markovian decision processes, Stochastic Games).

2012 ACM Subject Classification Mathematics of computing

Keywords and phrases Rotor-routing, Rotor Walk, Reachability Problem, Game Theory, Tree-like Multigraph

Digital Object Identifier 10.4230/LIPIcs.MFCS.2022.12

Related Version *Full Version*: `https://arxiv.org/abs/2204.13151`

Funding *Loric Duhazé*: This research was partially supported by Labex DigiCosme (project ANR11LABEX0045DIGICOSME) operated by ANR as part of the program "Investissement d'Avenir" Idex ParisSaclay (ANR11IDEX000302).

Acknowledgements We want to thank Johanne Cohen for her advises and attentive review.

1 Introduction

The *rotor routing*, or *rotor walk model*, has been studied under different names: *eulerian walkers* [17, 16] and *patrolling algorithm* [19]. It shares many properties with a more algebraic focused model: *abelian sandpiles* [4, 15]. We can cite [12] and [15] as general introductions to this cellular automaton.

Let us explain briefly how a rotor walk works. Consider a directed graph and for each vertex v, if v has k outgoing arcs, number these arcs from 1 to k. Then, we place a pebble on a starting vertex and proceed to the walk. On the initial vertex, the pebble moves to the next vertex according to arc 1. It does the same on the second vertex and so on. But the second time that a vertex is reached, the pebble will move according to arc 2, and so on until arc k has been used, and then we start again with arc 1.

In this work, we fix a set of vertices that we call sinks and stop the walking process when a sink is reached. The problem of determining, for a starting configuration (numbering) of arcs and an initial vertex, which sink will be reached first is the ARRIVAL problem. It is defined in [8] together with a proof that the problem belongs to the complexity class NP ∩ co-NP, but it is not known to be in P. It has then been shown in [10] that the problem is in the smaller complexity class UP ∩ co-UP, and a subexponential algorithm has been proposed in [11], based on computing a Tarski fixed point. This algorithm is even polynomial if the graph is almost acyclic (in a certain sense). A direct application of the rotor-routing automaton is that several structural properties of Markov chains can be approximated or bounded by rotor walks see [6, 7, 9]. It seems natural to extend these results to one and two player variants and define rotor analogs for *Markov decision processes* and *stochastic games* [18]. It is proved in [18] that deciding if a player can ensure some value is NP-complete for the one-player version and PSPACE-complete for the two-player version.

Contributions and Organization of the Paper

We define the class of *tree-like multigraphs*, and prove that while the number of steps needed to complete a rotor walk can still be exponential, the tree-like structure helps to efficiently solve ARRIVAL in linear time. We also extend our results to one-player and two-player variants. It is to be noted that tree-like multigraphs are not almost acyclic in the sense of [11], thus their algorithm does not run in polynomial time in our case.

In Section 2, we first give some standard definitions for multigraphs and proceed to define rotor walks in this context, together with different rotor-routing notions (*exit pattern, cycle pushing*, etc.). Next, in Section 3, we define tree-like multigraphs and our main tool to study ARRIVAL on these graphs, namely the *return flow.* Then, in Section 4, we sketch a polynomial algorithm that solves ARRIVAL and finally we show that this algorithm can be used to efficiently solve both the one and two player case.

The following table summarizes our results (bold), i.e. time complexity of computing the sink (or the optimal sink, for one-player and two-player) reached by a particle starting on a particular vertex in a graph $G = (V, A)$. Results on simple tree-like multigraphs depicted here and quickly presented in this paper are detailed in our extended version [1]. The first column states the complexity of the natural algorithm to solve these problems, namely simulating the *rotor walk.*

	Rotor Walk	0 player	1 player	2 players
General digraph	*exponential*	NP ∩ coNP	NP-complete	PSPACE-complete
Tree-like multigraph	**exponential**	$\boldsymbol{O(\lvert A\rvert)^\dagger}$	$\boldsymbol{O(\lvert A\rvert)}$	$\boldsymbol{O(\lvert A\rvert)}$
Simple Tree-like multigraph	$\boldsymbol{O(\lvert V\rvert^2)}$	$\boldsymbol{O(\lvert V\rvert)^\dagger}$	$\boldsymbol{O(\lvert V\rvert)^\dagger}$	$\boldsymbol{O(\lvert V\rvert)^\dagger}$

The † indicates the cases where we can solve the problem for all vertices of the graph at the same time with this complexity. Note that all proofs that do not appear directly in the document are detailed in our extended version [1].

2 Basic Definitions

2.1 Directed Multigraphs

In this paper, unless stated otherwise, we always consider a directed multigraph $G = (V, \mathcal{A}, h, t)$ where V is a finite set of *vertices*, $\mathcal{A}$ is a finite set of *arcs*, and h (for *head*) and t (for *tail*) are two maps from $\mathcal{A}$ to V defining incidence between arcs and vertices. For sake of clarity, we only consider graphs without arcs of the form $h(a) = t(a)$ (i.e. loops). All our complexity results would remain true if we authorized them. Note that multigraphs can have multiple arcs with the same head and tail. Let $u \in V$ be a vertex, we denote by $\mathcal{A}^+(u)$ (resp. $\mathcal{A}^-(u)$) the subset of arcs $a \in \mathcal{A}$ with tail u (resp. with head u).

Let $\Gamma^+(u)$ (resp. $\Gamma^-(u)$) be the subset of vertices $v \in V$ such that there is an arc $a \in \mathcal{A}$ with $h(a) = v$ and $t(a) = u$ (resp.$h(a) = u$ and $t(a) = v$). A graph such that for all $u \in V$ we have $|\mathcal{A}^+(u)| = |\Gamma^+(u)|$ is called *simple.* A vertex u for which $|\Gamma^+(u) \cup \Gamma^-(u)| = 1$ is called a *leaf.*

2.2 Rotor Routing Mechanics

Rotor Graphs

Let $G = (V, \mathcal{A}, h, t)$ be a multigraph.

▶ **Definition 1** (Rotor Order). *We define a* rotor order *at $u \in V$ as an operator denoted by θ_u such that:*

- $\theta_u : \mathcal{A}^+(u) \to \mathcal{A}^+(u)$
- *for all $a \in \mathcal{A}^+(u)$, the orbit $\{a, \theta_u(a), \theta_u^2(a), ..., \theta_u^{|\mathcal{A}^+(u)|-1}(a)\}$ of arc a under θ_u is equal to $\mathcal{A}^+(u)$, where $\theta_u^k(a)$ is the composition of θ_u applied to arc a exactly k times.*

Observe that each arc of $\mathcal{A}^+(u)$ appears exactly once in any orbit of θ_u. Now, we will integrate the operator θ_u to our graph structure as follows.

▶ **Definition 2** (Rotor Graph). *A* rotor graph *G is a (multi)graph G together with:*

- *a partition $V = V_0 \cup S_0$ of vertices, where $S_0 \neq \emptyset$ is a particular set of vertices called sinks, and V_0 is the rest of the vertices;*
- *a rotor order θ_u at each $u \in V_0$.*

In this document, unless stated otherwise, all the graphs we consider are rotor graphs with $G = (V_0, S_0, \mathcal{A}, h, t, \theta)$.

▶ **Definition 3** (Rotor Configuration). *A* rotor configuration *(or simply configuration) of a rotor graph G is a mapping ρ from V_0 to $\mathcal{A}$ such that $\rho(u) \in \mathcal{A}^+(u)$ for all $u \in V_0$. We denote by $\mathcal{C}(G)$ the set of all rotor configurations on the rotor graph G.*

What will be called a *particle* in the remaining of this paper is a pebble which will move from one vertex to another; hence the position of the particle is characterized by a single vertex. This movement, called rotor walk, follows specific rules that we detail after.

▶ **Definition 4** (Rotor-particle configuration). *A* rotor-particle configuration *is a couple (ρ, u) where ρ is a rotor configuration and $u \in V$ denotes the position of a particle.*

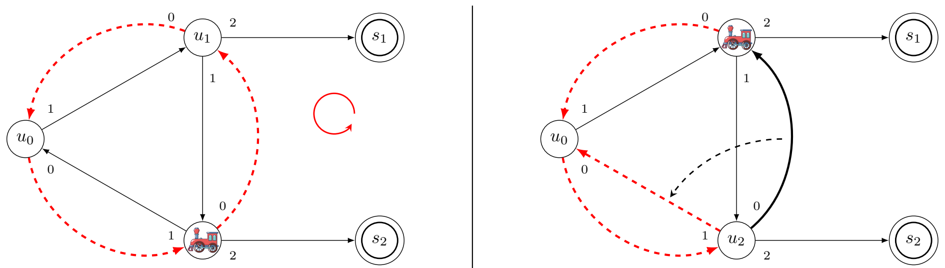
Rotor Walk

▶ **Definition 5.** *Let us define two mappings on $\mathcal{C}(G) \times V_0$:*

- **turn**, *with values in $\mathcal{C}(G)$, is defined by* **turn**$(\rho, u) = \rho'$ *where ρ' is equal to ρ except at u where $\rho'(u) = \theta_u(\rho(u))$.*
- **move**, *with values in V, is defined by* **move**$(\rho, u) = h(\rho(u))$.

By composing those mappings, we are now ready to define the *routing of a particle* which is a single step of a rotor walk.

▶ **Definition 6** (Routing of a Particle). *The* ***routing*** *of a particle (illustrated in Figure 1) from a rotor-particle configuration (ρ, u) is a mapping:* **routing** $: \mathcal{C}(G) \times V_0 \longrightarrow \mathcal{C}(G) \times V$ *defined by* **routing**$(\rho, u) = (\rho', v)$, *with $\rho' =$* **turn**(ρ, u) *and $v =$* **move**(ρ, u). *This can be viewed as the particle first travelling through $\rho(u)$ and then $\rho(u)$ is replaced by $\theta_u(\rho(u))$.*



(a) Let ρ be the rotor configuration depicted by the red arcs in dashes.

(b) The red rotor configuration in dashes is obtained by processing the operation: $\mathbf{routing}(\rho, u_2)$.

Figure 1 A rotor-routing where the particle is depicted by a train and starts on u_2. The sink-vertices are s_1 and s_2. The red arcs in dashes represent the current rotor configuration. The rotor orders on the different vertices are anticlockwise, i.e. they are: θ_{u_0}:(u_0,u_2),(u_0,u_1); θ_{u_1}: (u_1,u_0),(u_1,u_2),(u_1,s_1); θ_{u_2}: (u_2,u_1),(u_2,u_0),(u_2,s_2). These orders are also depicted by the numbers around each vertex.

▶ Remark 7. Our routing rule (move, then turn) is slightly different than the one defined in [17] which is mostly used in the literature (turn, then move) but is more convenient to study ARRIVAL. The two rules are equivalent up to applying **turn** mapping on all vertices.

A rotor-routing is in fact a single step of a rotor walk.

▶ **Definition 8** (Rotor Walk). *A* rotor walk *is a (finite or infinite) sequence of rotor-particle configurations $(\rho_i, u_i)_{i \geq 0}$, which is recursively defined by $(\rho_{i+1}, u_{i+1}) = \mathbf{routing}(\rho_i, u_i)$ as long as $u_i \in V_0$.*

One can check that the sequence of vertices in the rotor walk starting from the rotor-particle configuration depicted on Figure 1(b) until a sink is reached is $u_1, u_0, u_2, u_0, u_1, u_2, s_2$.

Routing to Sinks

Now that we have defined our version of rotor-walk, we proceed to the corresponding version of the problem *ARRIVAL* similar to the one stated in [8].

▶ **Definition 9** (Maximal Rotor Walk). *A maximal rotor walk is a rotor walk such that in the case where it is finite, the last vertex must be a sink vertex $s \in S_0 = V \setminus V_0$.*

▶ **Definition 10** (Stopping Rotor Graph). *If, for any vertex $u \in V$, there exists a directed path from u to a sink $s \in S_0$, the graph is said to be* stopping.

The next lemma is a classical result on rotor walks (cf Lemma 16 in [12]).

▶ **Lemma 11** (Finite number of steps). *If G is stopping then any rotor walk in G is finite.*

The main objective of our work is to study the sink that will be reached by a maximal rotor walk from a rotor-particle configuration, if the rotor walk is finite.

▶ **Definition 12** (Exit Sink). *Let $u \in V$, let ρ be a configuration, if the maximal rotor walk starting from (ρ, u) is finite in G, then the sink reached by such a rotor walk is denoted by $\mathbf{S}_G(\rho, u)$ and called* exit sink *of u for the rotor configuration ρ in G.*

▶ **Definition 13** (Exit Pattern). *For a rotor configuration ρ on a stopping rotor graph G, the* exit pattern *is the mapping that associates to each vertex $u \in V$, its exit sink $\mathbf{S}_G(\rho, u)$.*

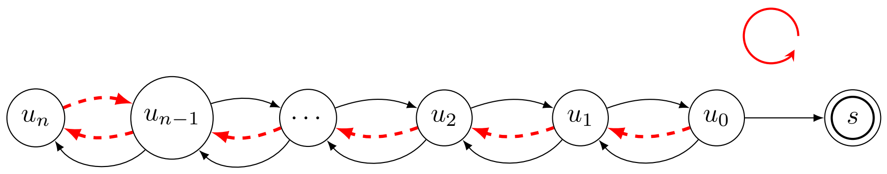


Figure 2 Family of path-like multigraphs where maximal routing can take an exponential number of steps in the number of vertices, here equal to $n+2$. The interior vertices (u_0 to u_{n-1}) have two arcs going left and one going right. Routing a particle from u_0 to sink s with the initial configuration ρ drawn with red arcs in dashes takes a non-polynomial time considering the anticlockwise rotor ordering on each vertex, depicted by the curved arc in red. One can check that the number of times a particle starting from u_0 will travel from u_i to u_{i+1} before reaching s is 2^{i+1}.

2.3 ARRIVAL and Complexity Issues

With our notations, ARRIVAL (see [8]) can be expressed as the following decision problem:

In a stopping rotor graph G,
given (ρ, u) and $s \in S_0$
does $\mathbf{S}_G(\rho, u) = s$?

This problem belongs to NP $\cap$ co-NP for simple graphs as shown in [8], but there is still no polynomial algorithm known to solve it. The case of eulerian simple graphs can be solved in time $O(|V + \mathcal{A}|^3)$, since a finite maximal rotor walk from (ρ, v) ends in at most $O(|V + \mathcal{A}|^3)$ routings (see [19]).

In the case of multigraphs, ARRIVAL still belongs to NP $\cap$ co-NP, since the polynomial certificate used for simple graphs in [8] remains valid. Our goal in this paper is to solve the problem in polynomial time for a particular class of multigraphs. Despite that, even for a path multigraph, the routing can be exponential, as shown on Figure 2.

2.4 Cycle Pushing

In order to speed-up the rotor walk process, a simple tool to avoid computing every step of the walk is the use of *cycle pushing*. We keep the terminology of *cycle pushing* used in [12] – even if what is called a cycle in this terminology is usually called a directed cycle or circuit in graph theory.

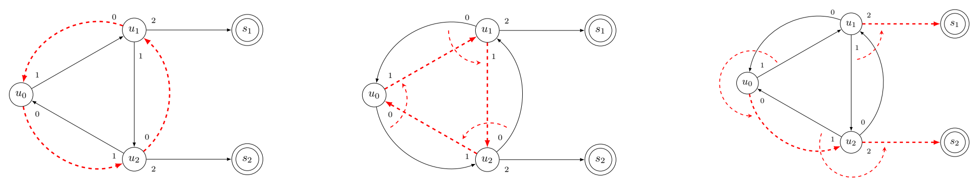
▶ **Definition 14** (Cycle *Pushing*). *Let ρ be a rotor configuration on G containing a directed cycle $C = (u_1, u_2, u_3, ..., u_k)$. We call* cycle push *the operation on ρ that leads to a configuration ρ' such that:*

- *for all $u_i \in C$, $\rho'(u_i) = \theta_{u_i}(\rho(u_i))$;*
- $\forall u \in V_0 \setminus C, \rho'(u) = \rho(u)$

*i.e. we process a **turn** on all $u_i \in C$.*

Note that a cycle push on a cycle C can also be computed by putting a particle on a vertex u of C and routing the particle until it comes back to u for the first time. Hence, cycle pushing is in a sense a shortcut on the rotor walk process, so in a manner similar to Lemma 11 it follows that in a stopping rotor graph, any sequence of cycle pushes is finite. This is a well known result in rotor walk studies (see [15]). It implies that, by processing a long enough sequence of successive cycle pushes, the resulting configuration contains no directed cycles. Such a sequence of cycle pushes is called *maximal*.

The two following results can be found in [12].



(a) Red Rotor Configuration ρ in dashes with rotor order on each vertex described by increasing numbers.

(b) The red configuration in dashes is obtained from a Cycle Push on the cycle $\{(u_0,u_2),(u_2,u_1),(u_1,u_0)\}$.

(c) Destination Forest (depicted by red arcs in dashes) computed by two successive cycle pushes on ρ.

Figure 3 Computation of the Destination Forest by successive cycle pushing.

▶ **Lemma 15** (Exit Pattern conservation for Cycle Push). *If G is a stopping rotor graph, for any rotor configuration ρ and configuration ρ' obtained from ρ by a cycle push, the exit pattern for ρ and ρ' is the same.*

▶ **Lemma 16** (Commutativity of Cycle Push). *In a stopping rotor graph, any maximal sequence of cycle pushes starting from a given rotor configuration ρ leads to the same acyclic configuration ρ' (ρ' does not contain a cycle).*

▶ **Definition 17** (Destination Forest). *We call the configuration obtained by a maximal cycle push sequence on ρ the* Destination Forest *of ρ, denoted by $D(\rho)$.*

The destination forest has a simple interpretation in terms of rotor walks: start a rotor walk by putting a particle on any vertex of a stopping graph G; consider a vertex $u \in V_0$; if the particle ever reaches u, it will leave u by arc $D(\rho)(u)$ on the last time it enters u.

In an acyclic configuration like $D(\rho)$, finding the exit pattern is simple, precisely:

▶ **Lemma 18** (Path to a sink). *If there is a directed path between $u \in V$ and $s \in S_0$ in ρ then $\mathbf{S}_G(\rho, u) = s$. It follows that from $D(\rho)$ one can compute the exit pattern of ρ in time complexity $O(|\mathcal{A}|)$.*

This gives us a new approach, since computing the exit pattern of a configuration ρ can be done by computing its Destination Forest $D(\rho)$. Note that by doing this we are solving a problem harder than ARRIVAL because we compute the exit sink of all vertices simultaneously.

▶ Remark 19. The strategy consisting in pushing cycles until the Destination Forest is reached can take an exponential number of steps. This is the case in the example drawn in Figure 2.

3 Tree-Like Multigraphs: Return Flow Definition

3.1 Tree-Like Multigraphs

To a directed multigraph $G = (V, \mathcal{A}, h, t)$ we associate:

- A simple directed graph $\hat{G} = (V, \hat{\mathcal{A}})$ such that, for $u, v \in V$ there is an arc from u to v in $\hat{G}$ if there is at least one arc $a \in \mathcal{A}$ with $t(a) = u$ and $h(a) = v$. Please note that even if there are multiple arcs a that satisfy this property, there is only one arc with tail u and head v in $\hat{\mathcal{A}}$. As it is unique, an arc from u to v in $\hat{\mathcal{A}}$ will simply be denoted by (u, v).
- A simple undirected graph $\overline{G} = (V, E)$ such that, for $u, v \in V$ there is an edge between u and v in $\overline{G}$ if and only if there is at least one arc $a \in \mathcal{A}$ such that $t(a) = u$ and $h(a) = v$ or $h(a) = u$ and $t(a) = v$.

▶ **Definition 20** (Tree-Like Rotor Multigraph and Tree-Like Multigraph). *A rotor multigraph $G = (V_0, S_0, \mathcal{A}, h, t, \theta)$ is tree-like if $\overline{G}$ is a tree and its set of leaves contains S_0. In that case, we say that $G = (V, \mathcal{A}, h, t)$ is a* tree-like multigraph*.*

Without loss of generality, in order to avoid some complexity in the notation and proofs, we will only study stopping tree-like rotor multigraphs.

▶ **Definition 21** (Sink Component). *A sink component is a strongly connected component in G that does not contain a sink vertex and such that there is no arc leaving the component.*

Note that all sink components can be computed in linear time.

▶ **Lemma 22** (Lemma 36 in [1]). *Consider a configuration ρ on a (not necessarily stopping) tree-like rotor multigraph $G = (V_0, S_0, \mathcal{A}, h, t, \theta)$. Consider a configuration ρ' on the* stopping *tree-like rotor multigraph $G' = (V_0', S_0', \mathcal{A}', h, t, \theta)$ where G' is obtained from G by replacing each sink component by a unique sink, and where $\rho'(u) = \rho(u)$ for each $u \in V_0'$. For any $u \in V$, finding the exit sink of u (if any) or the sink component reached by u in G for the configuration ρ can be directly determined by solving ARRIVAL for the configuration ρ' in G'.*

Note that, after replacing sink components by sinks, the multigraph G' may no longer be a tree-like multigraph but a forest-like multigraph. However we can split the study of ARRIVAL in each tree-like component of this forest since a particle cannot travel between those trees in a rotor walk.

3.2 Return Flows

Let us consider the simple example depicted on Figure 4 to motivate the introduction of (u, v)-subtrees and return flows, which is our main tool. In this figure, consider the routing of a particle starting at u:

- the particle moves from u to v_1, and stays for a while in the subtree T_1 – where it either reaches a sink or comes back to u. Suppose it comes back to u. Then:
- the particle moves from u to v_3, and either reaches a sink in T_3 or comes back to u. Suppose it comes back to u once again;
- the rotor walk goes on, in T_3, then in $T_2, T_1, T_1, T_3, \ldots$
- until finally the particle ends in a sink in one of the subtrees, say T_2.

Now consider only the relative movement that the particle had in T_2: it went from u into T_2 and back to u a number of times, before it ended in a sink. If we were to replace T_1 and T_3 by single arc leading back automatically to u, the relative movement in T_2 would have been exactly the same. The return flow will be a quantity that counts exactly the ability of each subtree to bounce back the particle to u. During the process described above, every time the particle enters a subtree and comes back to u, we can think of it as consuming a single unit of return flow in this subtree. The first time that a particle enters a subtree that has exactly one unit of return flow left, then the particle must end in a sink of that subtree.

▶ **Definition 23** ((u, v)-subtree). *Let $(u, v) \in \hat{\mathcal{A}}$. The (u, v)-subtree $T_{(u,v)}$ is a sub(multi)graph of G:*

- *whose vertices are all the vertices of the connected component of $\bar{G} \setminus \{u\}$ that contains v, together with u;*
- *whose arcs are all the arcs of G that link the vertices above, excepted in u where we remove all arcs of $\mathcal{A}^+(u)$ but a single arc a with head v. Such an arc a always exists because $(u, v) \in \hat{\mathcal{A}}$;*
- *whose rotor orders are unchanged except at u where $\theta_u(a) = a$.*

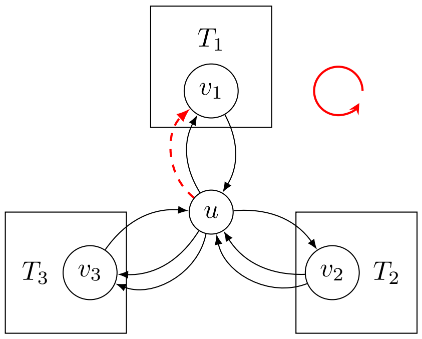


Figure 4 We sketch a stopping tree-like rotor multigraph as follows: a vertex u, its neighbours v_1, v_2, v_3 (that might be sinks), respectively belonging to T_1, T_2, T_3, the three connected components of $\bar{G} \setminus \{u\}$. In particular, we have $\hat{\mathcal{A}}^+(u) = \{(u, v_1); (u, v_2); (u, v_3)\}$. We consider the rotor configuration in red on u and θ_u is the anticlockwise order on the arcs of $\mathcal{A}^+(u)$.

Such a subtree is a (not necessarily stopping) tree-like rotor multigraph. A rotor configuration ρ in G can be thought of as a rotor configuration ρ' in $T_{(u,v)}$ by defining that $\rho'(u) = a$ and $\rho'(w) = \rho(w)$ for all $w \in T_{(u,v)}$.

We define a notion of *flow* for a particular starting vertex.

▶ **Definition 24** (Flow of (u, v))**.** *We define the* flow *on arc $(u, v) \in \hat{\mathcal{A}}$ for configuration ρ, denoted by $F_\rho(u, v)$, the number of times (possibly infinite) that an arc with tail u and head v is visited during the maximal rotor walk of a particle starting from the rotor-particle configuration (ρ, u). We denote by $F_\rho(u)$ the flow vector of (u, v) for every $v \in \Gamma^+(u)$.*

▶ **Definition 25** (Return flow)**.** *The* return flow *of arc $(u, v) \in \hat{\mathcal{A}}$ for configuration ρ, denoted by $r_\rho(u, v)$, is the flow on (u, v) in the (u, v)-subtree $T_{(u,v)}$.*

By definition of return flow, if $u \in S_0$, then $r_\rho(u, v) = 0$, and if $v \in S_0$, or if $(v, u) \notin \hat{\mathcal{A}}$ then $r_\rho(u, v) = 1$. Remark also that, even if the tree-like multigraph is stopping, it is not necessarily the case of any (u, v)-subtree: this is for instance the case of a leaf v which is not a sink such that $(u, v) \in \hat{\mathcal{A}}$. Finiteness of the return flow characterizes the subtrees that are stopping as stated in Lemma 26.

▶ **Lemma 26** (Lemma 40 in [1])**.** *Given a stopping tree-like multigraph G and an arc $(u, v) \in \hat{\mathcal{A}}$, the (u, v)-subtree $T_{(u,v)}$ is stopping if and only if for any rotor configuration on G, the return flow of (u, v) is finite.*

We give a bound on the maximal value of the return flow in a multigraph as it will be used to express our complexity results later.

▶ **Lemma 27** (Return flow bound, Lemma 41 in [1])**.** *Return flows can be written in at most $O(|\mathcal{A}|)$ bits.*

Return flows and flows are linked by the following result:

▶ **Lemma 28** (Lemma 42 in [1])**.** *Given a stopping tree-like rotor multigraph G, consider $u \in V_0$ and suppose that $h(D(\rho)(u)) = v$. Then:*

- *$F_\rho(u, v) = r_\rho(u, v)$;*
- *for all $w \in \Gamma^+(u) \setminus \{v\}$, $\quad F_\rho(u, w) < r_\rho(u, w)$;*
- *for all $w \in \Gamma^+(u) \cap \Gamma^-(u) \setminus \{v\}, r_\rho(w, u) = F_\rho(u, w) + 1$.*

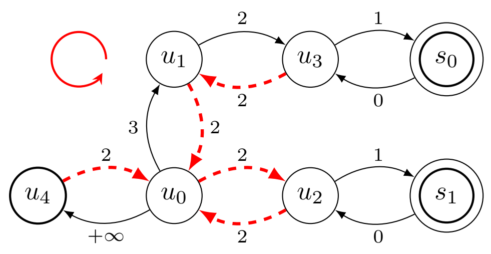


Figure 5 Examples of return flows in a simple graph. The rotor configuration is depicted by red arcs in dashes, with $S_0 = \{s_0, s_1\}$, and θ_{u_i} is the anticlockwise order on every vertex. We write the return flow of all arcs of $\hat{\mathcal{A}}$ next to their corresponding arc in $\mathcal{A}$. As a tutorial example, we detail the computation of $r_\rho(u_1, u_0)$ and $r_\rho(u_0, u_1)$. In the (u_1, u_0)-subtree, the particle will visit the following sequence of vertices $u_1, u_0, u_2, u_0, u_1, u_0, u_4, u_0, u_2, s_1$, where it crosses (u_1, u_0) twice, thus $r_\rho(u_1, u_0) = 2$. For the (u_0, u_1)-subtree, the sequence of vertices visited by the particle is $u_0, u_1, u_0, u_1, u_3, u_1, u_0, u_1, u_3, s_0$ hence $r_\rho(u_0, u_1) = 3$.

Thus, to compute $D(\rho)(u)$, we need to compute the flow of all arcs (u, w) with $w \in \Gamma^+(u)$ and compare it to $r_\rho(u, w)$. Theorem 29 gives the time complexity of such operation where $c(a, b)$ is the time complexity to divide an integer a by an integer b.

▶ **Theorem 29** (Theorem 43 in [1]). *For any vertex $u \in V_0$, given the return flows of all arcs $(u, v) \in \hat{\mathcal{A}}$ with $v \in \Gamma^+(u)$, one can compute $D(\rho)(u)$ and the flow on each arc (u, v) in time $O(|\Gamma^+(u)| \cdot c(r_{\max}, |\mathcal{A}^+(u)|))$ with r_{max} being the maximum (finite) value of $r_\rho(u, v)$ for all $v \in \Gamma^+(u)$.*

But in the case of simple graphs, we can improve this computation (see Lemma 30).

▶ **Lemma 30** (Lemma 61 in [1]). *If the graph is simple, $D(\rho)(u)$ is the first arc with respect to order θ_u, starting at $\rho(u)$, among all arcs of $\mathcal{A}^+(u)$ with minimal return flow. The flow on all arcs $(u, v) \in \hat{\mathcal{A}}$ can be computed in time $O(|\Gamma^+(u)|)$.*

4 ARRIVAL for Tree-Like Multigraphs

In this section we show that, for a given configuration ρ, we can compute the Destination Forest $D(\rho)$ in time complexity $O(|\mathcal{A}| \cdot c(r_{\max}, |\mathcal{A}|))$, hence solve the ARRIVAL problem for every vertex at the same time. To achieve this, we recursively compute return flows for all arcs in $\hat{\mathcal{A}}$ and then use these flows to compute the destination forest.

▶ **Theorem 31** (Complete Destination Algorithm, Theorem 47 in [1]). *The configuration $D(\rho)$ can be computed in time $O(|\mathcal{A}|)$ for a stopping tree-like rotor multigraph in a model where arithmetic operations can be made in constant time, or alternatively in $O(|\mathcal{A}| \cdot c(r_{\max}, |\mathcal{A}|))$ on a Turing Machine.*

Sketch of the Proof. Consider an arbitrary vertex x. We proceed with a Breadth-First Search (BFS) to compute an order such that all necessary return flows are already computed when we use Theorem 29. The algorithm is split into two phases:

1. Computation of the return flows of arcs directed from x towards leaves, starting from the arcs closest to the leaves and recursively coming back to x, using Theorem 29.
2. Computation of the return flows of all remaining arcs, starting from the arcs closest to x and recursively coming back to the leaves. We use Theorem 29 only twice for each vertex u to compute the return flows of all arcs (u, v) with (u, v) being directed from the leaves towards x.

All in all, we use Theorem 29 three times for each vertex, hence the time complexity is $O(\sum_{u \in V} (|\hat{\mathcal{A}}^+(u)| \cdot c(r_{\max}, |\mathcal{A}|)))$ which amounts to $O(|\mathcal{A}| \cdot c(r_{\max}, |\mathcal{A}|))$. ◀

We showed in Lemma 27 that return flows could be written in at most $O(|\mathcal{A}|)$ bits which gives an upper bound for $c(r_{\max}, |\mathcal{A}|)$ of $k|\mathcal{A}| \log(|\mathcal{A}|)$ for some constant $k > 0$. It is proved in [14] that the multiplication of two n bits integers can be done in time $O(n \log(n))$ and as the complexity of the division is equivalent to the complexity of multiplication (see [5]), the bound follows. Thus the complexity of our algorithm is $O(|\mathcal{A}|^2 \log(|\mathcal{A}|))$ in this context.

Simple Graph

In the case of simple graphs, this algorithm has a better complexity as we can use Lemma 30 instead of Theorem 29. Nevertheless, the algorithm remains the same and the complexity sums up to $O(|A|)$ which is also $O(|V|)$ as the graph is simple (details can be found in Theorem 63 in [1]).

5 One and Two player Variants

Problem ARRIVAL can be seen as a zero-player game where the winning condition is that the particle reaches a particular sink (or set of sinks). The one and two players variants of ARRIVAL (i.e. deterministic analogs of *Markov decision processes* and *Stochastic games*) we address in this section are inspired from [18], but differ by the choice of the set of strategies (see the discussion hereafter).

We specifically consider a game with a single player that controls a subset of vertices $V_{\mathcal{MAX}}$ of V_0. Given a rotor configuration on the rest of the vertices of V_0, a starting vertex and a set of targets among the sinks, his goal is to wisely choose the initial rotor configuration of the vertices he controls (his strategy) such that the particle reaches one of the targets.

Our definition of the way a player controls his vertices is somewhat different from the one in the seminal paper [18], where a strategy consists in choosing an outgoing-arc each time the particle is on a vertex controlled by the player. In this sense the set of strategies that we allow (which seems to us a very natural extension of the zero player case) is a finite subset of the version from [18] and could seem easier, but the latter can in fact be solved by some slight modifications of our algorithm. Moreover, the proof of NP-completeness from [18] is still valid in our case, despite the fact that we reduce the number of available strategies.

▶ **Definition 32** (Partial Configuration). *Let V' be a subset of V_0, a* partial rotor configuration *on V' is a mapping ρ' from V' to $\mathcal{A}$ such that $\rho'(u) \in \mathcal{A}^+(u)$ for all $u \in V'$.*

Given the disjoint sets of vertices V_r, $V_{\mathcal{MAX}}$ and S_0, a one-player rotor game (resp. one-player tree-like rotor game) is defined by $(V_r, V_{\mathcal{MAX}}, S_0, \mathcal{A}, h, t, \theta, \text{val}, \rho)$ where:

- $(V_0, S_0, \mathcal{A}, h, t, \theta)$ is a rotor graph (resp. tree-like rotor graph) with $V_0 = V_r \cup V_{\mathcal{MAX}}$;
- val is a map from S_0 to $\{0, 1\}$ defining the target sinks (sinks of value 1);
- ρ is a partial configuration on V_r, i.e. on the vertices not controlled by the player.

The tree-like rotor game is *stopping* if and only if the induced rotor graph $(V_0, S_0, \mathcal{A}, h, t, \theta)$ is stopping. The player is called $\mathcal{MAX}$, and a *strategy* for $\mathcal{MAX}$ is a partial rotor configuration on $V_{\mathcal{MAX}}$. We denote by $\Sigma_{\mathcal{MAX}}$ the finite set of strategies for this player.

Consider a partial rotor configuration ρ on V_r together with strategy σ and denote by (ρ, σ) the rotor configuration where we apply the partial configuration ρ or σ depending on whether the vertex is in V_r or $V_{\mathcal{MAX}}$. The *value of the game* for strategy σ and starting

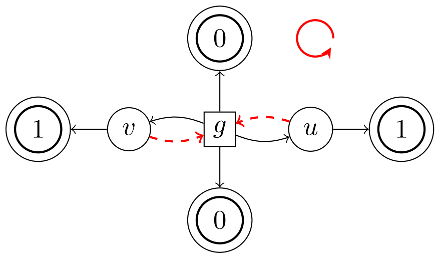


Figure 6 Simple graph where the optimal strategy depends on the starting vertex u_0, with $V_{\mathcal{MAX}} = \{g\}$, with $u, v \in V_0$ and with all other vertices being sinks. As in previous examples, the starting configuration is depicted by red arcs in dashes, and the rotor order on all vertices is an anticlockwise order on their outgoing arcs. In the case $u_0 = v$, the only optimal strategy is $\sigma(g) = (g, v)$ and the game has value 1. In the case $u_0 = u$, the only optimal strategy is $\sigma(g) = (g, u)$ and the game has value 1.

vertex u_0 is denoted by $\mathrm{val}_\sigma(u_0)$ and is equal to $\mathrm{val}(s)$ where s is the sink reached by a maximal rotor walk from the rotor particle configuration $((\rho, \sigma), u_0)$ if any, and 0 otherwise. As in the zero-player framework, up to computing strongly connected components that do not contain sinks and replacing each of them with a sink of value 0, we can suppose that the tree-like rotor game is stopping. In the following, all rotor games we consider are tree-like and stopping unless stated otherwise.

When u_0 is fixed, the maximal value of $\mathrm{val}_\sigma(u_0)$ over strategies $\sigma \in \Sigma_{\mathcal{MAX}}$ is called the *optimal value* of the game with starting vertex u_0 and is denoted by $\mathrm{val}^*(u_0)$. Any strategy $\sigma \in \Sigma_{\mathcal{MAX}}$ such that $\mathrm{val}_\sigma(u_0) = \mathrm{val}^*(u_0)$ is called an *optimal strategy* for the game starting in u_0. Observe that optimal strategies may depend on the choice of u_0 (see Figure 6).

The one-player ARRIVAL problem consists in computing the optimal value of a given starting vertex in a one-player rotor game.

Recall that in the tree-like rotor graph, $T_{(u,v)}$ denotes the (u, v)-subtree. We extend this notation to denote the one-player, not necessarily stopping, game played on the (u, v)-subtree where we restrict $V_{\mathcal{MAX}}$ and V_r to the subtree. For this game, we only consider the case where the starting vertex is u.

▶ **Definition 33** (Value under strategy). *Let (u, v) be an arc of $\hat{\mathcal{A}}$. For a strategy σ on the (u, v)-subtree, we denote by $val_\sigma(u, v)$ (resp. $val^*(u, v)$) the value of the game under strategy σ (resp. under an optimal strategy) in $T_{(u,v)}$. This is called the value (resp. the optimal value) of arc (u, v) for strategy σ.*

▶ **Definition 34** (Optimal return flow r^*). *Let (u, v) be an arc of $\hat{\mathcal{A}}$. If $val^*(u, v) = 0$, then $r^*(u, v)$ is defined as the maximum of $r_\sigma(u, v)$ over all strategies σ on $T_{(u,v)}$, otherwise it is the minimum of $r_{\sigma^*}(u, v)$ among* optimal *strategies σ^* on $T_{(u,v)}$.*

Lemma 35 connects the value $\mathrm{val}_\sigma(u)$ with the value of the last outgoing arc of vertex u while processing a maximal rotor walk from the rotor-particle configuration $((\rho, \sigma), u)$.

▶ **Lemma 35.** *Let a be an arc of $\mathcal{A}^+(u)$ such that $D(\rho, \sigma)(u) = a$ with $h(a) = v$. We have $val_\sigma(u) = val_\sigma(u, v)$.*

To recursively compute an optimal strategy, we need a stronger notion of optimality, namely a *subtree optimal strategy.*

▶ **Definition 36** (Subtree optimal strategy). *A strategy σ^* is* subtree optimal *at u_0 if it is optimal at u_0 and, moreover, $val_{\sigma^*}(u, v) = val^*(u, v)$ and $r_{\sigma^*}(u, v) = r^*(u, v)$ for every (u, v)-subtree such that (u, v) is directed from u_0 towards the leaves.*

Instead of recursively computing only the return flow as in the zero-player game, we now propagate both the optimal value and the optimal return flow to construct a subtree optimal strategy (details in the proof of Theorem 54 in [1]) and give an equivalent of Theorem 31 for the one-player game.

▶ **Theorem 37** (Computation of $\mathrm{val}^*(u_0)$, Theorem 54 in [1]). *The optimal value $val^*(u_0)$ can be computed in the same time complexity as the computation of $D(\rho)$ in the zero-player game (see Theorem 31).*

Some remarks are in order here. (*i*) The algorithm used to compute $\mathrm{val}^*(u_0)$ (Algorithm 3 in [1]) provides the optimal value of u_0 as well as a subtree optimal strategy at u_0 for every decisional vertex. A tutorial example is given in [1] (Figure 11). (*ii*) This algorithm can also be adapted to compute optimal values for games with more general type of strategies, particularly those in [18]. (*iii*) In the case of a simple graph, one can compute $\mathrm{val}^*(u_0)$ for every vertex u_0 with the same time complexity as in Theorem 37 as detailed in our extended version [1]. (*iv*) If we consider a game with integer values on the sinks, one can use a dichotomic process to solve it in $\log(|S|)$ steps where a step consists in solving a binary game.

Two-player Game

In the two-player game, each player controls the initial configuration on disjoint sets of vertices, and the second player wants to minimize the value of the game. In a general graph, there may not exist an equilibrium in pure strategies (see Figure 7 where we define a game equivalent to *matching pennies*). However, in the case of tree-like rotor multigraphs, we can show that there always is an equilibrium in pure strategies and compute it (see [1]). As in the one-player case, if we consider integer values, we can simply proceed using a dichotomy technique.

Conclusion and Future Work

Concerning ARRIVAL, one remaining fundamental question is to determine whether there exists a polynomial algorithm to solve the zero-player game. Similarly, problems such as *simple stochastic games, parity games* and *mean-payoff games* are also in NP $\cap$ co-NP, and there are no polynomial algorithm known to solve them (see [13]). For those different problems, considering subclasses of graphs where we can find polynomial algorithms is a fruitful approach (see [2] and [3]). This paper is a first step in this direction.

Thus, we would like to study more general classes of graphs. To begin with, even graphs that are well-studied in terms of the *sandpile group* such that ladders or grids remain now an open problem for ARRIVAL. The problem of finding the destination of multiple particles at the same time is also an important open problem in nearly all cases – we solve it for the (simple) path graph in our extended version (see [1]).

Now that the Tree-like multigraph case is settled, it seems natural to try and extend these results to classes of graphs with bounded width (e.g. treewidth, pathwidth, etc.). However, this extension is not direct and requires further work. If we were to replace a single node by a bag of nodes, defining an analog of the rotor ordering, the routine and return flows is much more complicated since a particle can enter and leave the bag in different ways. We need to compute and store all the exit arcs for all entering arcs in the bag for every rotor configuration, and combine these informations for neighbouring bags in order to find locally the destination forest. We intend to finish this work in a near future.

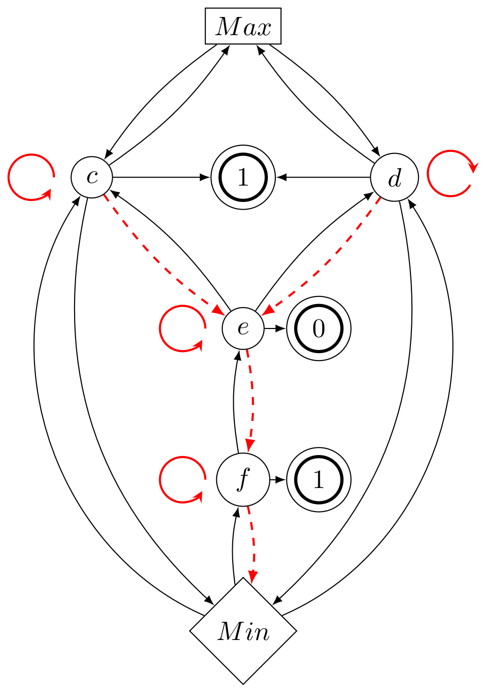


Figure 7 This example is a simple undirected graph where each edge is replaced by two arcs. We have $V_{\mathcal{MAX}} = \{Max\}$, $V_{\mathcal{MIN}} = \{Min\}$ and $V_0 = \{c, d, e, f\}$ and S_0 is the rest of the vertices with their value written inside them. The particle starts on the vertex Max. In this game, the only optimal strategy for $\mathcal{MAX}$ when the strategy for $\mathcal{MIN}$ is the arc (Min, x) with $x \in \{c, d\}$ is (Max, x). On the other hand the only optimal strategy for $\mathcal{MIN}$ when the strategy for $\mathcal{MAX}$ is the arc (Max, c) (resp. (Max, d)) is (Min, d) (resp. (Min, c)). The situation is like the classical matching pennies game where one player tries to match the strategy of the opponent whereas the other player has the opposite objective. It is known that such game does not admit a Nash equilibrium in pure strategies.

References

1 David Auger, Pierre Coucheney, and Loric Duhaze. Polynomial time algorithm for arrival on tree-like multigraphs. *arXiv preprint arXiv:2204.13151*, 2022.

2 David Auger, Pierre Coucheney, and Yann Strozecki. Finding optimal strategies of almost acyclic simple stochastic games. In *International Conference on Theory and Applications of Models of Computation*, pages 67–85. Springer, 2014.

3 David Auger, Pierre Coucheney, and Yann Strozecki. Solving simple stochastic games with few random nodes faster using bland's rule. In *36th International Symposium on Theoretical Aspects of Computer Science*, 2019.

4 Anders Björner, László Lovász, and Peter W Shor. Chip-firing games on graphs. *European Journal of Combinatorics*, 12(4):283–291, 1991.

5 Richard P Brent and Paul Zimmermann. *Modern computer arithmetic*, volume 18. Cambridge University Press, 2010.

6 Swee Hong Chan, Lila Greco, Lionel Levine, and Peter Li. Random walks with local memory. *Journal of Statistical Physics*, 184(1):1–28, 2021.

7 Joshua N Cooper and Joel Spencer. Simulating a random walk with constant error. *Combinatorics, Probability and Computing*, 15(6):815–822, 2006.

8 Jérôme Dohrau, Bernd Gärtner, Manuel Kohler, Jiří Matoušek, and Emo Welzl. Arrival: A zero-player graph game in NP ∩ coNP. In *A journey through discrete mathematics*, pages 367–374. Springer, 2017.

9 Tobias Friedrich and Thomas Sauerwald. The cover time of deterministic random walks. In *International Computing and Combinatorics Conference*, pages 130–139. Springer, 2010.

10 Bernd Gärtner, Thomas Dueholm Hansen, Pavel Hubácek, Karel Král, Hagar Mosaad, and Veronika Slívová. Arrival: Next stop in cls. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

11 Bernd Gärtner, Sebastian Haslebacher, and Hung P. Hoang. A Subexponential Algorithm for ARRIVAL. In *ICALP 2021*, volume 198, pages 69:1–69:14, 2021.

12 Giuliano Pezzolo Giacaglia, Lionel Levine, James Propp, and Linda Zayas-Palmer. Local-to-global principles for rotor walk. *arXiv preprint arXiv:1107.4442*, 2011.

13 Nir Halman. Simple stochastic games, parity games, mean payoff games and discounted payoff games are all LP-type problems. *Algorithmica*, 49(1):37–50, 2007.

14 David Harvey and Joris Van Der Hoeven. Integer multiplication in time O(n log n). *Annals of Mathematics*, 193(2):563–617, 2021.

15 Alexander E. Holroyd, Lionel Levine, Karola Mészáros, Yuyal Peres, James Propp, and David B. Wilson. *Chip-Firing and Rotor-Routing on Directed Graphs*, pages 331–364. Springer, 2008.

16 AM Povolotsky, VB Priezzhev, and RR Shcherbakov. Dynamics of eulerian walkers. *Physical review E*, 58(5):5449, 1998.

17 Vyatcheslav B Priezzhev, Deepak Dhar, Abhishek Dhar, and Supriya Krishnamurthy. Eulerian walkers as a model of self-organized criticality. *Physical Review Letters*, 77(25):5079, 1996.

18 Rahul Savani, Matthias Mnich, Martin Gairing, and John Fearnley. Reachability switching games. *Logical Methods in Computer Science*, 17, 2021.

19 Vladimir Yanovski, Israel A Wagner, and Alfred M Bruckstein. A distributed ant algorithm for protect efficiently patrolling a network. *Algorithmica*, 37(3):165–186, 2003.