Linear Time Algorithm for Update Games via Strongly-Trap-Connected Components

A 2-Player Infinite Pebble Game Generalization of Strongly-Connected Components

Carlo Comin[*], Romeo Rizzi[†]

Abstract

An arena is a finite directed graph whose vertices are divided into two classes, i.e., $V = V_\square \cup V_\bigcirc$; this forms the basic playground for a plethora of 2-player infinite pebble games. We introduce and study a refined notion of reachability for arenas, named *trap-reachability*, where Player $\square$ attempts to reach vertices without leaving a prescribed subset $U \subseteq V$, while Player $\bigcirc$ works against. It is shown that every arena decomposes into *strongly-trap-connected components (STCCs)*. Our main result is a linear time algorithm for computing this unique decomposition. Both the graph structures and the algorithm generalize the classical decomposition of a directed graph into its strongly-connected components (SCCs). The algorithm builds on a generalization of the depth-first search (DFS), taking inspiration from Tarjan's SCCs classical algorithm. The structures of palm-trees and jungles described in Tarjan's original paper need to be revisited and generalized (i.e., tr-palm-trees and tr-jungles) in order to handle the 2-player infinite pebble game's setting. This theory has direct applications in solving Update Games (UGs) faster. Dinneen and Khoussainov showed in 1999 that deciding who's the winner in a given UG costs $O(mn)$ time, where n is the number of vertices and m is that of arcs. We solve that problem in $\Theta(m + n)$ linear time. The result is obtained by observing that the UG is a win for Player $\square$ if and only if the arena comprises one single STCC. It is also observed that the tr-palm-tree returned by the algorithm encodes routing information that an $\Theta(n)$-space agent can consult to win the UG in $O(1)$ time per move. With this, the polynomial-time complexity for deciding Explicit McNaughton-Müller Games is also improved, from cubic to quadratic.

Keywords: Trap-Reachability, Strongly-Trap-Connected Components, Update Games, Update Networks, Explicit McNaughton-Müller Games, Linear Time Algorithm, Strategy Synthesis.

1 Introduction

In the construction of reactive systems, like communication protocols or control systems, a central aim is to put the development of hardware and software on a mathematical basis which is both firm and practical. A characteristic feature of such systems is their perpetual interaction with the environment as well as their non-terminating behaviour. The theory of infinite duration games offers many appealing results under this prospect [4]. For instance, consider the following communication network problem. Suppose we have data stored on each node of a network and we want to continuously update all nodes with consistent data: often one requirement is to share key information between all nodes of a network, this can be done by having a data packet of current information continuously going through all nodes. Unfortunately not all routing choices are always under our control, some of them may be controlled by the network environment, which could play against us. This is essentially an infinite 2-player pebble game played on an *arena*, i.e., a finite directed simple graph in which the vertices are divided into two classes, i.e., $V_\square$ and $V_\bigcirc$, where Player $\square$ wants to visit all vertices infinitely often by moving the pebble on them, while Player $\bigcirc$ works against. This is called *Update Game (UG)* in [1–3]. Dinneen and Khoussainov [2] showed that deciding who's the winner in a given UG costs $O(mn)$ time, where n is the number of vertices and m is that of the arcs.

[*]Department of Mathematics, University of Trento, Italy; and LIGM, Université Paris-Est, Marne-la-Vallée, Paris, France. (e-mail: carlo.comin@unitn.it)

[†]Department of Computer Science, University of Verona, Verona, Italy. (e-mail: romeo.rizzi@univr.it)

Contribution and Organization. In Section 6, as a main result, the same problem of deciding who's the winner in a given UG is solved in $\Theta(m+n)$ linear time; it is also observed that the graph structure returned by the algorithm encodes routing information that an $\Theta(n)$-space agent can consult to win the UG in $O(1)$ time per move. For this, in Section 2, we introduce and study a refined notion of reachability for arenas, named *trap-reachability*, where Player $\square$ attempts to reach vertices without leaving a prescribed subset $U \subseteq V$, while Player $\bigcirc$ works against. In Section 3, it is shown that every arena decomposes into *strongly-trap-connected components (STCCs)*, and a linear time algorithm for computing this unique decomposition is offered in Section 5. Both the graph structures and the STCCs algorithm generalize the classical decomposition of a directed graph into its strongly-connected components (SCCs) [7]. The algorithm builds on a generalization of the depth-first search (DFS), taking inspiration from Tarjan's SCCs classical algorithm, the structures of palm-trees and jungles described in Tarjan's original paper [7] need to be revisited and generalized (i.e., tr-palm-trees and tr-jungles) in order to handle the 2-player infinite pebble game's setting, this is done in Section 4. With this, in Section 7, the polynomial-time complexity for deciding Explicit McNaughton-Müller Games is also improved, from cubic to quadratic.

1.1 Notation and Preliminaries.

An *arena* is a tuple $\mathcal{A} \triangleq (V, A, (V_\square, V_\bigcirc))$ where $G^{\mathcal{A}} \triangleq (V, A)$ is a finite directed simple graph (i.e., there are no loops nor parallel arcs) and $(V_\square, V_\bigcirc)$ is a partition of V into the set $V_\square$ of vertices owned by Player $\square$, and the set $V_\bigcirc$ of those owned by Player $\bigcirc$. Still $G^{\mathcal{A}}$ is not required to be a bipartite graph on colour classes $V_\square$ and $V_\bigcirc$. The ingoing and outgoing neighbourhoods of $u \in V$ are $N^{\text{in}}_{\mathcal{A}}(u) \triangleq \{v \in V \mid (v,u) \in A\}$ and $N^{\text{out}}_{\mathcal{A}}(u) \triangleq \{v \in V \mid (u,v) \in A\}$, respectively. Disjoint-union is denoted by $\cup\!\!\!\!\cdot$, e.g., $V = V_\square \cup\!\!\!\!\cdot\, V_\bigcirc$.

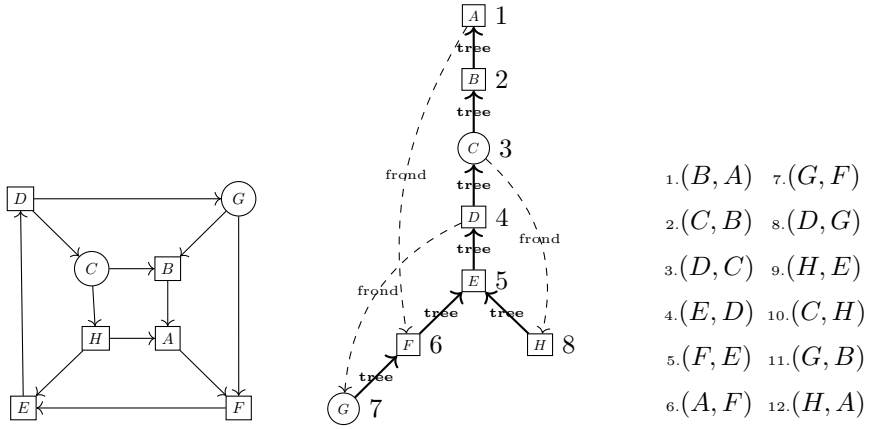
A *game* on $\mathcal{A}$ is played for infinitely many rounds by moving a pebble along the arcs, from one vertex to an adjacent one. Initially the pebble is located on some $v_s \in V$, this is the *starting position*. At each round, if the pebble is currently on $v \in V_i$, for some $i \in \{\square, \bigcirc\}$, Player i chooses an arc $(v, v') \in A$; and then the next round starts with the pebble on v'.

A finite (or infinite) *path* in $G^{\mathcal{A}}$ is a sequence $v_0 v_1 \dots v_n \dots \in V^*$ (or V^ω) such that $\forall_{j \geq 0}\ (v_j, v_{j+1}) \in A$; the *length* of $v_0 v_1 \dots v_n$ is n. A *play* on $\mathcal{A}$ is any infinite path in $G^{\mathcal{A}}$. A *strategy* for Player i, where $i \in \{\square, \bigcirc\}$, is a map $\sigma_i : V^* \times V_i \to V$ such that for every finite path $p'v$ in $G^{\mathcal{A}}$, where $p' \in V^*$ and $v \in V_i$, it holds that $(v, \sigma_i(p', v)) \in A$. The set of all strategies of Player i in $\mathcal{A}$ is denoted by $\Sigma^{\mathcal{A}}_i$. A play $v_0 v_1 \dots v_n \dots$ is *consistent* with some $\sigma \in \Sigma^{\mathcal{A}}_i$ if $v_{j+1} = \sigma(v_0 v_1 \dots v_j)$ whenever $v_j \in V_i$. Given two strategies $\sigma_\square \in \Sigma^{\mathcal{A}}_\square$ and $\sigma_\bigcirc \in \Sigma^{\mathcal{A}}_\bigcirc$, and some $v_s \in V$, the *outcome* play $\rho_{\mathcal{A}}(v_s, \sigma_\square, \sigma_\bigcirc)$ is the (unique) play that starts at v_s and is consistent with both $\sigma_\square$ and $\sigma_\bigcirc$. For any $v \in V$, we denote by $\rho_{\mathcal{A}}(v_s, \sigma_\square, \sigma_\bigcirc, v)$ the (unique) prefix of $\rho_{\mathcal{A}}(v_s, \sigma_\square, \sigma_\bigcirc)$ which ends at the first occurence of v, if any; otherwise, $\rho_{\mathcal{A}}(v_s, \sigma_\square, \sigma_\bigcirc, v) \triangleq \rho_{\mathcal{A}}(v_s, \sigma_\square, \sigma_\bigcirc)$. For any finite (or infinite) path $p \in V^*$ (or $p \in V^\omega$), the *alphabet* of p is $\Xi(p) \triangleq \{v \in V \mid v \text{ appears in } p\}$.

Let $T \triangleq (V_T, A_T)$ be an inward directed tree, rooted at $r_T \in V_T$. We simply write $u \in T$ for $u \in V_T$. For each $u \in T$, there is only one *path* p_u going from u to r_T; the *depth* $d(u)$ of u is the length of p_u. An *ancestor* of $u \in T$ is any $v \in \Xi(p_u)$; it's a *proper ancestor* if $v \neq u$, and it's the *parent* $\pi_T(u)$ of u if $(u, v) \in A_T$. The *children* of $u \in T$ are all the $v \in T$ such that $\pi_T(v) = u$. A *descendant* of $u \in T$ is any $v \in T$ such that $u \in \Xi(p_v)$; it's a *proper descendant* if $v \neq u$. A *leaf* of T is any $u \in T$ having no children, i.e., $N^{\text{in}}_T(u) = \emptyset$. The *lowest common ancestor (LCA)* of a subset of vertices $S \subseteq T$ is $\gamma_S \triangleq \arg\max\{d(\gamma) \mid \gamma \in T \text{ and } \forall_{s \in S}\ s \text{ is a descendant of } \gamma \text{ in } T\}$. The subtree of T that is rooted at $u \in T$ is denoted by T_u. Given a LIFO stack `St` containing some element $v \in$ `St`, we denote by `St`(v) the set of all elements $u \in$ `St` going from the top of `St` down 'til the first occurence of v, extremes included.

2 Trap-Reachability

Recalling palm-trees and jungles. In a seminal work of Tarjan [7] some foundamental properties and applications of the *depth-first search (DFS)* were analyzed. Particularly, specific graph structures underlying the DFS were discussed in detail, namely *palm-trees* and *jungles*. This allowed the author to provide a linear time procedure, nowadays known as *Tarjan's SCCs algorithm*, for computing strongly-connected components (SCCs) in finite directed simple graphs.



(a) An Arena $\mathcal{A}$.

(b) The rev-palm-tree generated by rev-DFS, with indices of vertices.

(c) The order of arcs' exploration.

Figure 1: An arena (a), and a rev-palm-tree (b), generated by rev-DFS (c).

Following [7], assume G is a finite directed simple graph that we wish to explore. Initially all the vertices of G are unexplored. We start from some vertex of G and choose an *outgoing* arc to follow. At each step, we select an unexplored arc (leading from a vertex already reached) and explore (traverse) that arc. When selecting an arc to explore, we always choose an arc emanating from the vertex *most recently* reached which still has unexplored arcs. Traversing the selected arc leads to some vertex, either new or already reached; if already reached, we backtrack and select another unexplored arc. Whenever we run out of arcs leading from old vertices, we choose some unreached vertex, if any exists, and begin a new exploration from this point. Eventually, the procedure will traverse all the arcs of G, each exactly once. This is a *depth-first search (DFS)* of G; one may call it *fwd*-DFS, because at each step the chosen arc is *outgoing*.

Recalling *palm-trees* from [7], consider in more detail what happens when a DFS is performed on G. The set of arcs leading to an unexplored vertex, when traversed during the search, forms an outward directed *tree* T. The other arcs fall into four categories: (i) some arcs are running from ancestors to descendants in T, these may well be ignored as they do not affect the SCCs of G; still, (ii) some other arcs run from descendants to ancestors in T, these are quite relevant instead, they are called *fronds*; (iii) other arcs run from one subtree to another within the same tree T, these are *internal cross-links*; (iv) suppose to continue the DFS until all arcs are explored, the process creates a family of trees which contains all vertices of G, i.e., a *spanning forest* F of G, plus sets of (fronds and) cross-links which may also connect two different trees in F; these are *external cross-links*. It is easy to see that if the vertices of G are numbered in the order in which they are reached during the search, e.g., by $\texttt{idx} : V \to \{1, \ldots, |V|\}$, then any (internal or external) cross-link (u, v) always has $\texttt{idx}[u] > \texttt{idx}[v]$. Any tree T of F, comprising fronds and internal cross-links, it is called *palm-tree*.

A directed graph consisting of a spanning forest, plus fronds and cross-links, it is named *jungle*, i.e., a family of palm-trees plus external cross-links, which is a natural representation of the graph-reachability structure of the input graph G.

Rev-DFS, rev-palm-trees and rev-jungles. In this work we need to impose an *opposite* direction w.r.t. that in which the arcs are traversed, so at each step of the DFS one actually chooses an *ingoing* arc to follow instead of an outgoing one. In this way, the corresponding search algorithm may be called *rev*-DFS. A moment's reflection reveals that this symmetric twist doesn't affect the basic properties of the DFS. For instance, if the vertices are numbered in the order in which they are reached during the rev-DFS, e.g., by $\texttt{idx} : V \to \{1, \ldots, |V|\}$, now a cross-link (u, v) always has $\texttt{idx}[u] < \texttt{idx}[v]$. So, a family of *rev-palm-trees* is constructed during rev-DFS. Let us call *rev-jungle* the graph structure underlying a rev-DFS, that is a family of rev-palm-trees comprising *fronds* and *cross-links*.

Trap-Reachability. A trivial *graph-reachability* property holds in any rev-palm-tree $T = (V_T, A_T)$:

for any $u, v \in T$ such that v is an ancestor of u in T, there exists a simple path from u to v in T, i.e., v is graph-reachable from u in T. With this in mind, let's explore an *arena* $\mathcal{A} = (V, A, (V_\square, V_\bigcirc))$ by a rev-DFS. Let $J^{\mathcal{A}}$ be the resulting rev-jungle, and let $T^{\mathcal{A}}$ be any rev-palm-tree of $J^{\mathcal{A}}$. An example is depicted in Fig. 1a and the corresponding rev-palm-tree $T^{\mathcal{A}}$ is in Fig. 1b; notice, $T^{\mathcal{A}}$ is still an arena. So, let's consider *reachability* on arenas, which is most relevant to 2-player infinite pebble games: given $\mathcal{A}$, and any two $u, v \in V$, we say that v is *reachable* from u in $\mathcal{A}$ if and only if there is some $\sigma_\square \in \Sigma_\square^{\mathcal{A}}$ (i.e., $\sigma_\square = \sigma_\square(u, v)$) such that for every $\sigma_\bigcirc \in \Sigma_\bigcirc^{\mathcal{A}}$, it holds $v \in \Xi\big(\rho_{\mathcal{A}}(u, \sigma_\square, \sigma_\bigcirc)\big)$. Then, the rev-palm-tree $T^{\mathcal{A}}$, constructed as above, doesn't respect reachability: consider the two vertices $F, B \in V_\square$ in the rev-palm-tree $T^{\mathcal{A}}$ shown in Fig. 1b; starting from F, Player $\square$ admits no strategy which allows him to reach B, even though B is an ancestor of F in T; indeed, any play starting from F must first reach D, at that point, if Player $\square$ plays (D, G) then Player $\bigcirc$ can go back to F by playing (G, F), otherwise, if Player $\square$ plays (D, C), then Player $\bigcirc$ can play (C, H) thus reaching H, and notice that once on H the continuation of the play must reach D back again. So, starting from F, Player $\bigcirc$ can prevent Player $\square$ to reach B. Thus we now aim at generalizing the classical DFS, palm-trees and jungles, from directed graphs to arenas, in such a way as to preserve reachability *within* the (suitably adapted) palm-trees. Particularly, a desirable "DFS on arenas" should maintain the following basic property: for any (suitably adapted) palm-tree T, if $u, v \in T$ and v is an ancestor of u in T, there exists $\sigma_\square \in \Sigma_\square^{\mathcal{A}}$ which allows Player $\square$ to eventually reach v starting from u, *without leaving* T at the same time, no matter which $\sigma_\bigcirc \in \Sigma_\bigcirc^{\mathcal{A}}$ is chosen by Player $\bigcirc$.

This is the genesis of *trap-reachability.*

Definition 1. *Given an arena $\mathcal{A}$ on vertex set V, let $U \subseteq V$ and $u, v \in U$. We say that v is U-*trap-reachable *from u when* there exists $\sigma_\square \in \Sigma_\square^{\mathcal{A}}$ *(i.e., $\sigma_\square = \sigma_\square(u, v)$) such that* for every $\sigma_\bigcirc \in \Sigma_\bigcirc^{\mathcal{A}}$*:*

[reachability] $v \in \Xi\big(\rho_{\mathcal{A}}(u, \sigma_\square, \sigma_\bigcirc)\big)$*; and,*

[entrapment] $\Xi\big(\rho_{\mathcal{A}}(u, \sigma_\square, \sigma_\bigcirc, v)\big) \subseteq U$.

In this case, we denote $\sigma_\square : u \overset{U}{\rightsquigarrow} v$, or $u \overset{U}{\rightsquigarrow} v$ when $\sigma_\square$ is implicit; if $U = V$, $\sigma_\square : u \rightsquigarrow v$ and $u \rightsquigarrow v$ will be enough notation.

Remark: Notice that any $u \in U$ is always U-trap-reachable from itself, for every $U \subseteq V$.

3 Strongly-Trap-Connectedness

In the rest of this work, $\mathcal{A} = (V, A, (V_\square, V_\bigcirc))$ denotes the generic arena taken as input.

Definition 2. *We say that $U \subseteq V$ is* strongly-trap-connected *when for every $(u, v) \in U \times U$ there exists some $\sigma_\square \in \Sigma_\square^{\mathcal{A}}$ (i.e., $\sigma_\square = \sigma_\square(u, v)$) such that $\sigma_\square : u \overset{U}{\rightsquigarrow} v$.*

Notice, $\emptyset$ and $\{v\}$ are strongly-trap-connected for any $v \in V$.

Definition 3. *A* strongly-trap-connected component (STCC) *is a maximal strongly-trap-connected $\mathcal{C} \subseteq V$ (i.e., such that if $\mathcal{C} \subseteq \mathcal{C}'$ and $\mathcal{C}'$ is strongly-trap-connected, then $\mathcal{C} = \mathcal{C}'$).*

Next, we observe the following property concerning strongly-trap-connectedness.

Lemma 1. *Let $V_1, V_2 \subseteq V$ be strongly-trap-connected. If $V_1 \cap V_2 \neq \emptyset$, then $V_1 \cup V_2$ is strongly-trap-connected.*

Proof. Pick some $u, v \in V_1 \cup V_2$ and $z \in V_1 \cap V_2$, arbitrarily. Since $\{u, z\} \subseteq V_1$, and since V_1 is strongly-trap-connected, there exists some $\sigma_\square(u, z) \in \Sigma_\square^{\mathcal{A}}$ such that $\sigma_\square(u, z) : u \overset{V_1}{\rightsquigarrow} z$; similarly, there is $\sigma_\square(z, v) \in \Sigma_\square^{\mathcal{A}}$ such that $\sigma_\square(z, v) : z \overset{V_2}{\rightsquigarrow} v$. Then, consider the following $\sigma_\square(u, v) \in \Sigma_\square^{\mathcal{A}}$:

$$\sigma_\square(u, v) \triangleq \begin{cases} \textit{(1)} \text{ Starting from } u\text{, play } \sigma_\square(u, z) \text{ until } z \text{ is first reached; then,} \\ \textit{(2)} \text{ once on } z\text{, play } \sigma_\square(z, v) \text{ until } v \text{ is finally reached.} \end{cases}$$

Clearly, $\sigma_\square(u, v) : u \overset{V_1 \cup V_2}{\rightsquigarrow} v$. Since u and v were chosen arbitrarily, then $V_1 \cup V_2$ is strongly-trap-connected. $\square$

Lemma 1 allows us to define and study an equivalence relation, i.e., $\sim_{\text{stc}} \subseteq V \times V$; it will turn out that the STCCs of $\mathcal{A}$ are the equivalence classes of $\sim_{\text{stc}}$.

Definition 4. *The binary relation $\sim_{stc}$ on V is defined as follows:*

$$\sim_{stc} \triangleq \big\{(u,v) \in V \times V \mid \exists_{U \subseteq V} \text{ such that } U \text{ is strongly-trap-connected and } \{u,v\} \subseteq U\big\}.$$

Lemma 2. *It holds that $\sim_{stc}$ is an equivalence relation on V.*

So, let $\{\mathcal{C}_i\}_{i=1}^{k}$ be the distinct equivalence classes of $\sim_{stc}$, for some $k \in \mathbf{N}$. Then, the following holds.

1. *If $U \subseteq V$ is strongly-trap-connected and $U \cap \mathcal{C}_i \neq \emptyset$, then $U \subseteq \mathcal{C}_i$;*
2. *$\mathcal{C}_i$ is strongly-trap-connected for each $i \in [k]$;*
3. *Let $U \subseteq V$ be strongly-trap-connected. Then, $\mathcal{C}_i \subsetneq U$ for* no *$i \in [k]$.*

Proof of ($\sim_{stc}$ is an equivalence relation on V). To begin, (i) $\sim_{\text{stc}}$ is *reflexive*: for any $u \in V$, let $U \triangleq \{u\}$; then, $u \overset{U}{\rightsquigarrow} u$, so U is strongly-trap-connected; this shows $u \sim_{\text{stc}} u$. (ii) $\sim_{\text{stc}}$ is *symmetric*, (actually, by definition): for any $u, v \in V$, assume $u \sim_{\text{stc}} v$; then, there exists some $U \subseteq V$ which is strongly-trap-connected and $u, v \in U$; so, the same set U certifies $v \sim_{\text{stc}} u$. (iii) $\sim_{\text{stc}}$ is *transitive*: indeed, for any $a, b, c \in V$, assume $a \sim_{\text{stc}} b$ and $b \sim_{\text{stc}} c$. Since $a \sim_{\text{stc}} b$, there exists V_1 which is strongly-trap-connected and such that $a, b \in V_1$; similarly, there exists V_2 which is strongly-trap-connected and such that $b, c \in V_2$. Consider $U \triangleq V_1 \cup V_2$. Since $b \in V_1 \cap V_2$, and V_1, V_2 are both strongly-trap-connected, then U is strongly-trap-connected by Lemma 1. Moreover, $a, c \in U$. So, $a \sim_{\text{stc}} c$. □

Proof of (1). Since $U \cap \mathcal{C}_i \neq \emptyset$, let $z \in U \cap \mathcal{C}_i$. Let $v \in U$, arbitrarily. Since U is strongly-trap-connected and $z, v \in U$, then $v \sim_{\text{stc}} z$. Therefore, $v \in \mathcal{C}_i$ (because $z \in \mathcal{C}_i$, which is an equivalence class of $\sim_{\text{stc}}$). □

Proof of (2). Let $u, v \in \mathcal{C}_i$, arbitrarily. Then, $u \sim_{\text{stc}} v$. So, there exists some $U \subseteq V$ which is strongly-trap-connected and such that $u, v \in U$. Thus, $u \overset{U}{\rightsquigarrow} v$. Notice, $u, v \in U \cap \mathcal{C}_i \neq \emptyset$. Then, by Item 1 of Lemma 2, $U \subseteq \mathcal{C}_i$. Since $u \overset{U}{\rightsquigarrow} v$ and $U \subseteq \mathcal{C}_i$, then $u \overset{\mathcal{C}_i}{\rightsquigarrow} v$. So, $\mathcal{C}_i$ is strongly-trap-connected. □

Proof of (3). Assume that $\mathcal{C}_i \subseteq U$, for some $i \in [k]$, and some $U \subseteq V$ which is strongly-trap-connected. Then, since $U \cap \mathcal{C}_i = \mathcal{C}_i \neq \emptyset$, by Item 1 of Lemma 2 we have $U \subseteq \mathcal{C}_i$. So, $\mathcal{C}_i = U$. □

Proposition 1. *Let $\mathcal{C} \subseteq V$, and consider the $\sim_{stc}$ relation on V. It holds that $\mathcal{C}$ is a STCC of $\mathcal{A}$ if and only if $\mathcal{C}$ is an equivalence class of $\sim_{stc}$.*

Proof. ($\Rightarrow$) If $\mathcal{C}$ is a STCC of $\mathcal{A}$, then $\mathcal{C}$ is strongly-trap-connected. So, $u \sim_{\text{stc}} v$ for every $u, v \in \mathcal{C}$. Then, $\mathcal{C} \subseteq \mathcal{C}'$ holds for some equivalence class $\mathcal{C}'$ of $\sim_{\text{stc}}$. By Item 2 of Lemma 2, $\mathcal{C}'$ is strongly-trap-connected. Thus, by maximality, $\mathcal{C}$ is not a proper subset of $\mathcal{C}'$. Therefore, $\mathcal{C} = \mathcal{C}'$.

($\Leftarrow$) If $\mathcal{C}$ is an equivalence class of $\sim_{\text{stc}}$, then: $\mathcal{C}$ is strongly-trap-connected by Item 2 of Lemma 2; and $\mathcal{C}$ is maximal by Item 3 of Lemma 2. Therefore, $\mathcal{C}$ is a STCC of $\mathcal{A}$. □

4 TR-Depth-First-Search

In this section, a DFS algorithm for the exploration on arenas is designed. Its rationale is that a new node $u \in V$ is attached to the r_T-rooted DFS-tree T under formation as soon as trap-reachability of r_T from u, within T itself, is already guaranteed (rather than requiring the weaker graph-reachability of r_T from u, like rev-DFS would do on graphs). The algorithm is called *Trap-Reachability-Depth-First-Search (tr-DFS)*. The pseudo-code of `tr-DFS()` is given in Algo. 1, and that of subprocedure `tr-DFS-visit()` is in Proc. 1.

Given $\mathcal{A}$, `tr-DFS`($\mathcal{A}$) explores $\mathcal{A}$ so to construct another arena $\mathcal{J}_{\mathcal{A}}$, like rev-DFS constructs a jungle. The `tr-DFS`($\mathcal{A}$) (Algo. 1) is a generalization of rev-DFS, in the sense that, if $V_{\bigcirc} = \emptyset$, it works as rev-DFS and $\mathcal{J}_{\mathcal{A}}$ coincides with a Tarjan's jungle. So, $\mathcal{J}_{\mathcal{A}}$ comprises a forest of trees, called *tr-palm-trees*, with *fronds* and *cross-links*. Initially, four sets of arcs $A_{\text{tree}}, A_{\text{frond}}, A_{\text{petiole}}, A_{\text{cross}}$ are initialized to $\emptyset$, then some arcs will be added to them during `tr-DFS`($\mathcal{A}$); when `tr-DFS`($\mathcal{A}$) will halt, $A' \leftarrow A_{\text{tree}} \uplus A_{\text{frond}} \uplus A_{\text{petiole}} \uplus A_{\text{cross}}$ will be the arc set of $\mathcal{J}_{\mathcal{A}}$. Let's say $u \in V$ *joins* $\mathcal{J}_{\mathcal{A}}$ precisely when (u, v) is added to A_{tree}, for some $v \in V$.

An index $\texttt{idx} : V \to \{1, \dots, |V|\}$ numbers the vertices in the order in which they join $\mathcal{J}_\mathcal{A}$; initially, $\forall_{u \in V}\ \texttt{idx}[u] \leftarrow +\infty$. Let's say $u \in V$ is *visited* if $\texttt{idx}[u] < +\infty$, and *unvisited* if $\texttt{idx}[u] = +\infty$. Any $u \in V_\square$ joins $\mathcal{J}_\mathcal{A}$ as soon as it is visited by the search (see lines 6-8 of `tr-DFS-visit()`, Proc. 1); but the $V_\bigcirc$*-rule* (i.e., that allowing $u \in V_\bigcirc$ to join $\mathcal{J}_\mathcal{A}$) is more involved: any $u \in V_\bigcirc$ joins $\mathcal{J}_\mathcal{A}$ as soon as all $u' \in N^{\text{out}}_\mathcal{A}(u)$ have already did it; and when $u \in V_\bigcirc$ joins some tr-palm-tree $\mathcal{P}$ of $\mathcal{J}_\mathcal{A}$, with parent π (i.e., when $u \in V_\bigcirc$ *and* $(u, \pi) \in A_{\text{tree}}$ for some $\pi \in V$), then *all* arcs going out of u are tagged *petiole-arcs*; and π is the *LCA* of $N^{\text{out}}_\mathcal{A}(u)$ in $\mathcal{P}$. In fact, besides fronds and cross-links, tr-palm-trees have an additional class of arcs, the *petiole-arcs*: these are those arcs thanks to which $u \in V_\bigcirc$ can join $\mathcal{J}_\mathcal{A}$. By considering LCAs the $V_\bigcirc$*-rule* allows us to preserve trap-reachability, as shown in Proposition 4. To implement the $V_\bigcirc$*-rule*, an additional counter $\texttt{cnt} : V_\bigcirc \to \mathbf{N}$ is employed, and this invariant is maintained:

Algorithm 1: tr-Depth-First Search

```
Procedure tr-DFS(A)
    input  : An arena A = (V, A, (V_○, V_□)).
    output: A tr-jungle J_A.
1   A_tree, A_frond, A_petiole, A_cross ← ∅;
2   foreach u ∈ V do
3       idx[u] ← +∞;
4       active[u] ← false;
5       ready_St[u] ← ∅;
6       if u ∈ V_○ then
7           cnt[u] ← |N^out_A(u)|;
8   next_idx ← 1;
9   foreach u ∈ V_□ do
10      if idx[u] = +∞ then
11          tr-DFS-visit(u, A);
12  foreach u ∈ V_○ do
13      if idx[u] = +∞ then
14          idx[u] ← next_idx;
15          next_idx ← next_idx + 1;
16  A' ← A_tree ⊎ A_frond ⊎ A_petiole ⊎ A_cross;
17  return J_A ← (V, A', (V_□, V_○));
```

$$\forall_{u \in V_\bigcirc}\ \texttt{cnt}[u] = \big|\{v \in N^{\text{out}}_\mathcal{A}(u) \mid \texttt{idx}[v] = +\infty\}\big|, \qquad (I_{\texttt{cnt}})$$

also, for each $u \in V$, there's a LIFO stack of vertices named $\texttt{ready_St}[u]$.

Initially, $\forall_{u \in V}\ \texttt{ready_St}[u] \leftarrow \emptyset$ and $\forall_{u \in V_\bigcirc}\ \texttt{cnt}[u] \leftarrow |N^{\text{out}}_\mathcal{A}(u)|$ (lines 5-7); then, $\texttt{cnt}[u]$ is decremented whenever some $v \in N^{\text{out}}_\mathcal{A}(u)$ is visited. When $\texttt{cnt}[u] = 0$ (at line 11 of $\texttt{tr-DFS-visit}(v, \mathcal{A})$, Proc. 1), all $v \in N^{\text{out}}_\mathcal{A}(u)$ have already joined $\mathcal{J}_\mathcal{A}$: notice, if any two vertices in $N^{\text{out}}_\mathcal{A}(u)$ belong to two distinct tr-palm-trees in $\mathcal{J}_\mathcal{A}$, there would be no way to preserve trap-reachability, in case u already joined $\mathcal{J}_\mathcal{A}$, because Player $\bigcirc$ can always choose to move from u to any of the two shafts, and the LCA γ of $N^{\text{out}}_\mathcal{A}(u)$ in (V, A_{tree}) is undefined; still, if all vertices in $N^{\text{out}}_\mathcal{A}(u)$ belong to the same tr-palm-tree, the LCA γ of $N^{\text{out}}_\mathcal{A}(u)$ in (V, A_{tree}) does exist; so, firstly we seek for the LCA γ, and if it exists, push u on top of $\texttt{ready_St}[\gamma]$ (lines 11-13 of `tr-DFS-visit()`, Proc. 1). So doing, $u \in V_\bigcirc$ joins $\mathcal{J}_\mathcal{A}$ as soon as `tr-DFS-visit()` backtracks, from the last $v \in N^{\text{out}}_\mathcal{A}(u)$ that has been visited, up to γ (possibly $\gamma = v$): at that point, $\texttt{ready_St}[\gamma]$ will be checked and emptied (lines 17-21), and u will be found there inside, so (u, γ) will be added to A_{tree} (line 19); also, for each $t \in N^{\text{out}}_\mathcal{A}(u)$ the arc (u, t) will be added to A_{petiole}, and $\texttt{tr-DFS-visit}(u, \mathcal{A})$ will be invoked.

Procedure 1: tr-DFS-visit

```
Procedure tr-DFS-visit(v, A)
    input  : One vertex v ∈ V of A.
1   active[v] ← true;
2   idx[v] ← next_idx;
3   next_idx ← next_idx + 1;
    // Check the in-neighbourhood of v
4   foreach u ∈ N^in_A(v) do
5       if idx[u] = +∞ then
6           if u ∈ V_□ then
7               add (u, v) to A_tree;
8               tr-DFS-visit(u, A);
9           else
10              cnt[u] ← cnt[u] − 1;
11              if cnt[u] = 0 and ∃(LCA of N^out_A(u) in
                (V, A_tree)) then
12                  γ ← the LCA of N^out_A(u) in (V, A_tree);
13                  ready_St[γ].push(u);
14      else if active[u] = true then
15          add (u, v) to A_frond;
16          else add (u, v) to A_cross;
    // Check the ready-stack of v, i.e., ready_St[v]
17  while ready_St[v] ≠ ∅ do
18      u ← ready_St[v].pop(); // u ∈ V_○
19      add (u, v) to A_tree;
20      for each t ∈ N^out_A(u) do add (u, t) to A_petiole;
21      tr-DFS-visit(u, A);
22  active[v] ← false;
```

To classify the remaining arcs into fronds or cross-links, an additional flag $\texttt{active} : V \to \{\texttt{true}, \texttt{false}\}$ is employed; initially, $\forall_{u \in V}\ \texttt{active}[u] \leftarrow \texttt{false}$; then, $\texttt{active}[u]$ is set to `true` when u is visited by $\texttt{tr-DFS-visit}(u, \mathcal{A})$ (line 1), finally, $\texttt{active}[u]$ is set back to `false` when $\texttt{tr-DFS-visit}(u, \mathcal{A})$ ends.

During $\texttt{tr-DFS-visit}(v, \mathcal{A})$, when some $u \in N^{\text{in}}_\mathcal{A}(v)$ such that $\texttt{idx}[u] \neq +\infty$ is explored, if $\texttt{active}[u] = \texttt{true}$ then (u, v) is added to A_{frond}, otherwise to A_{cross}.

There's still one point which is worth mentioning. During `tr-DFS()`, firstly, all $u \in V_\square$ are considered, see lines 9-11 of `tr-DFS()` (Algo. 1); so, for each unvisited $u \in V_\square$, $\texttt{tr-DFS-visit}(u, \mathcal{A})$ is invoked; after that, for each $u \in V_\bigcirc$ which is still unvisited, it is assigned $\texttt{idx}[u]$ incrementally, and $\texttt{tr-DFS-visit}(u, \mathcal{A})$ is not invoked anymore. Indeed, w.l.o.g we can assume that $\forall_{v \in V} |N^{\text{out}}_\mathcal{A}(v)| \geq 2$ holds, by pre-processing

$\mathcal{A}$ as follows: for any $v \in V$, if $N^{\text{out}}_{\mathcal{A}}(v) = \emptyset$, remove v and all of its ingoing arcs; if $N^{\text{out}}_{\mathcal{A}}(v) = \{v'\}$, add (u, v') to A for each $u \in N^{\text{in}}_{\mathcal{A}}(v)$, finally remove v and all of its arcs. So doing, even if `tr-DFS-visit`$(v, \mathcal{A})$ would've been invoked for some $v \in V_{\bigcirc}$, say at line 14 of `tr-DFS()`, there would've been no $u \in V$ such that $(u, v) \in A_{\text{tree}}$: consider `tr-DFS-visit()` (Proc. 1), notice (u, v) could not have been added to A_{tree} neither at line 7 (because all $u \in V_{\square}$ would've been already visited before at that time) nor at line 19 (since $\forall_{v \in V_{\bigcirc}} |N^{\text{out}}_{\mathcal{A}}(v)| \geq 2$, there would've been no way for "$\exists$ LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ in (V, A_{tree})" at line 11). So, this way of going is fine. Let us now analyze the complexity of `tr-DFS()` (Algo. 1).

Proposition 2. *Given $\mathcal{A}$, the `tr-DFS`$(\mathcal{A})$ (Algo. 1) halts in time $\Theta\big(|V| + |A| + \mathit{Time}[\textsc{lca}]\big)$, and it works with space $\Theta\big(|V| + |A| + \mathit{Space}[\textsc{lca}]\big)$, where $\mathit{Time}[\textsc{lca}]$ ($\mathit{Space}[\textsc{lca}]$) is the aggregate total time (space) taken by all of the LCA computations at lines 11-12 of `tr-DFS-visit`() (Proc. 1). Moreover, each $v \in V$ is numbered by `idx` exactly once.*

Proof. Firstly notice that the init-phase (lines 1-7 of Algo. 1) takes $\Theta(|V| + |A|)$ time. Secondly, Algo. 1 basically performs a sequence of invocations to `tr-DFS-visit`$(v, \mathcal{A})$ (Proc. 1), each one is for some $v \in V$. Any such `tr-DFS-visit`$(v, \mathcal{A})$ is invoked iff `idx`$[v] = +\infty$, and then `idx`$[v]$ is set to some non-zero value at line 2. Thus, the total number of invocations of `tr-DFS-visit()` (Proc. 1) is at most $|V|$; actually, by line 9 of `tr-DFS()` (Algo. 1), it is exactly $|V|$; so, each vertex $v \in V$ is numbered by `idx` $: V \to \{1, \ldots, |V|\}$ exactly once. Concerning time complexity, each `tr-DFS-visit`$(v, \mathcal{A})$ explores $N^{\text{in}}_{\mathcal{A}}(v)$ as follows: for each $u \in N^{\text{in}}_{\mathcal{A}}(v)$, the LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ is computed at lines 11-12. Also, at the end of `tr-DFS-visit`$(v, \mathcal{A})$, the stack `ready_St`$[v]$ is emptied; still, due to the condition `cnt`$[u] = 0$ that is checked at line 11 of `tr-DFS-visit()`, any $u \in V_{\bigcirc}$ can be pushed on `ready_St`$[v]$ at most once and for at most one $v \in V$. Therefore, the $\Theta\big(|V|+|A|+\text{Time}[\textsc{lca}]\big)$ time bound holds. Concerning space complexity, a similar argument shows that the aggregate total space for storing $\{$`ready_St`$[v]\}_{v \in V}$ is only $O(|V|)$. Also, the total size of `idx`, `active` and `cnt` is $\Theta(|V|)$, and that of A' is $\Theta(|A|)$ (see line 16 of `tr-DFS()`, Algo. 1). So, the working space is $\Theta\big(|V| + |A| + \text{Space}[\textsc{lca}]\big)$. $\square$

Next, we analyze the structure of the arena $\mathcal{J}_{\mathcal{A}}$ which is constructed by `tr-DFS`$(\mathcal{A})$ (Algo. 1). Let's start by formally defining *tr-palm-trees*. Some examples are shown in Fig. 2 and Fig. 3.

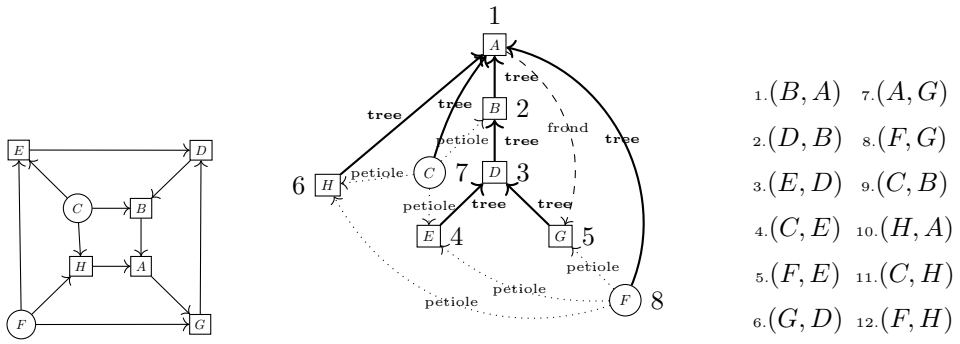
Definition 5. *A* tr-palm-tree *is a pair $(\mathcal{P}, \mathit{idx})$, where:*
(i) $\mathcal{P} \triangleq (V, A, (V_{\square}, V_{\bigcirc}))$ is an arena, $V \triangleq V_{\square} \uplus V_{\bigcirc}$ and $A \triangleq A_{\mathit{tree}} \uplus A_{\mathit{frond}} \uplus A_{\mathit{petiole}} \uplus A_{\mathit{cross}}$;
(ii) $\mathit{idx} : V \to \{1 + j, \ldots, |V| + j\}$, for some fixed $j \in \mathbf{N}$, is a labelling of V;
(iii) the following four main properties hold:

(tr-pt-1) $\mathcal{T}_{\mathcal{P}} \triangleq (V, A_{\mathit{tree}})$ is an inward directed rooted tree such that:
- *(a) the root $r_{\mathcal{T}_{\mathcal{P}}}$ of $\mathcal{T}_{\mathcal{P}}$ is controlled by Player $\square$, i.e., $r_{\mathcal{T}_{\mathcal{P}}} \in V_{\square}$;*
- *(b) $\mathit{idx}[u] > \mathit{idx}[v]$ if $(u, v) \in A_{\mathit{tree}}$;*

(tr-pt-2) Each $(u, v) \in A_{\mathit{frond}}$ connects some $u \in V_{\square}$ to one of its proper descendants $v \in V$ in $\mathcal{T}_{\mathcal{P}}$;

(tr-pt-3) Each $(u, v) \in A_{\mathit{petiole}}$ connects some $u \in V_{\bigcirc}$ to one of the descendants v of its parent $\pi_{\mathcal{T}_{\mathcal{P}}}(u)$ (i.e., possibly to $\pi_{\mathcal{T}_{\mathcal{P}}}(u)$); in particular, given any $u \in V_{\bigcirc}$, the following hold:
- *(a) $\{v \in V \mid (u, v) \in A_{\mathit{petiole}}\} \uplus \{\pi_{\mathcal{T}_{\mathcal{P}}}(u)\} = N^{\mathit{out}}_{\mathcal{P}}(u)$;*
- *(b) $\pi_{\mathcal{T}_{\mathcal{P}}}(u)$ is the LCA of $\{v \in V \mid (u, v) \in A_{\mathit{petiole}}\}$ in $\mathcal{T}_{\mathcal{P}}$;*
- *(c) $\mathit{idx}[u] > \mathit{idx}[v]$ for every $v \in N^{\mathit{out}}_{\mathcal{P}}(u)$.*

(tr-pt-4) Each arc $(u, v) \in A_{\mathit{cross}}$ connects some $u \in V_{\square}$ to some $v \in V$ such that:
- *(a) v is not a descendant of u in $\mathcal{T}_{\mathcal{P}}$;*
- *(b) either v is a proper ancestor of u in $\mathcal{T}_{\mathcal{P}}$ or $\mathit{idx}[u] < \mathit{idx}[v]$.*



(a) An arena $\mathcal{A}$.

(b) The tr-palm-tree generated by tr-DFS rooted at A, with indices of vertices and labelled arcs.

(c) The order of arcs' exploration.

Figure 2: An arena (a), and a tr-palm-tree (b), generated by tr-DFS (c).

Definition 6. *A* tr-jungle *is any arena $\mathcal{J} \triangleq (V, A, (V_\square, V_\bigcirc))$ comprising a family of tr-palm-trees $\{\mathcal{P}^i\}_{i=1}^k$, for some $k \in \mathbf{N}$, and satisfying the following properties:*

(tr-jn-1) $\forall_{i\in[k]}\ \mathcal{P}^i \triangleq (V^i, A^i, (V^i_\square, V^i_\bigcirc))$, where $V^i_\square \subseteq V_\square, V^i_\bigcirc \subseteq V_\bigcirc, A^i \subseteq A$;

(tr-jn-2) $\forall_{i,j\in[k]}\ V^i \cap V^j = \emptyset$ if $i \neq j$;

(tr-jn-3) If $(u, v) \in A$ for some $u \in V^i$ and $v \in V^j$ such that $i \neq j$, then $u \in V^i_\square$ and $i < j$;

(tr-jn-4) If $v \in V \setminus \bigcup_{i=1}^k V^i$, then $v \in V_\bigcirc$ and $N^{out}_{\mathcal{J}}(v) \subseteq V^i$ for no *$i \in [k]$.*

Definition 7. *Given a tr-palm-tree $(\mathcal{P}, \texttt{idx})$, for $\mathcal{P} = (V, A, (V_\square, V_\bigcirc))$, $A = A_{tree} \cup A_{frond} \cup A_{petiole} \cup A_{cross}$, the* support *of $\mathcal{P}$ is the arena $\mathcal{P}_* \triangleq (V, A_*, (V_\square, V_\bigcirc))$, where: $A_* \triangleq \big\{(u, v) \in A \mid u \in V_\square\big\} \cup A_{petiole}$.*

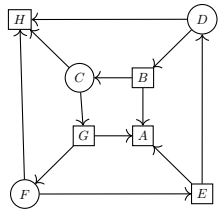
Notice that $A_* = A \setminus \big\{(u, v) \in A_{tree} \mid u \in V_\bigcirc\big\}$ holds by (tr-pt-3).

Given any tr-jungle $\mathcal{J}$ with tr-palm-trees's family $\{\mathcal{P}^i\}_{i=1}^k$, for some $k \in \mathbf{N}$, let $\overline{V} \triangleq V \setminus \bigcup_{i=1}^k V^i$ (where V^i is the vertex set of $\mathcal{P}^i$). The support *of $\mathcal{J}$ is the arena $\mathcal{J}_*$ which is obtained from $\mathcal{J}$ by replacing $\mathcal{P}^i$ with its support $\mathcal{P}^i_*$, for every $i \in [k]$, and by leaving intact all the vertices in $\overline{V}$ and all the arcs (u, v) of $\mathcal{J}$ such that: either, (i) $u \in V^i$ and $v \in V^j$ for some $i \neq j$; or, (ii) $u \in \overline{V}$ or $v \in \overline{V}$ (possibly both of them).*

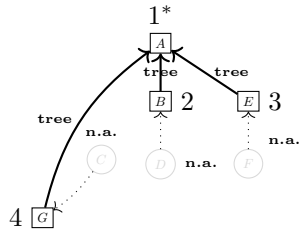
Proposition 3. *Let $\mathcal{A} = (V, A, (V_\square, V_\bigcirc))$ be an arena. The following two propositions hold.*

1. *Let J be the arena constructed by* `tr-DFS`$(\mathcal{A})$ *(Algo. 1). Then, J is a tr-jungle.*
2. *Let J be a tr-jungle with support J_*. Then,* `tr-DFS`(J_*) *(Algo. 1) constructs J itself, i.e., $\mathcal{J}_{J_*} = J$.*

Proof of (1). Recall, `tr-DFS`$(\mathcal{A})$ performs a sequence of invocations to `tr-DFS-visit`$(\cdot, \mathcal{A})$; by Proposition 2, each $v \in V$ is numbered by `idx` exactly once. Let k be the number of times that `tr-DFS-visit`$(u_i, \mathcal{A})$ is invoked at line 11 of `tr-DFS`() (Algo. 1). For each $i = 1, 2, \ldots, k$, it holds $u_i \in V_\square$ by line 9, then, let $V^i \subseteq V$ be the set of vertices that are numbered by `idx` during `tr-DFS-visit`$(u_i, \mathcal{A})$, recursive calls included. Let A^i be the set of arcs that are added to any of A_{tree}, A_{frond}, A_{petiole}, A_{cross} during that same `tr-DFS-visit`$(u_i, \mathcal{A})$, and let $A^i_t \triangleq \{(a, b) \in A^i \mid a, b \in V^i\}$ and $A^i_c \triangleq A^i \setminus A^i_t$. Let $\mathcal{P}^i \triangleq (V^i, A^i_t, (V_\square \cap V^i, V_\bigcirc \cap V^i))$, and let $\texttt{idx}^i$ be the restriction of `idx` to V^i. It is not difficult to see that $(\mathcal{P}^i, \texttt{idx}^i)$ is a tr-palm-tree: indeed, for each $i \in [k]$, $(\mathcal{P}^i, \texttt{idx}^i)$ is constructed by `tr-DFS-visit`$(u_i, \mathcal{A})$ and thus it satisfies *(tr-pt-1)* to *(tr-pt-4)*, where any $u \in V_\bigcirc$ joins $\mathcal{P}^i$ according to: (i) the checking of the $\texttt{cnt}[u] = 0$ condition at line 11, (ii) the LCA computation at lines 11-12, (iii) the emptying of $\texttt{ready_St}[v]$ at lines 17-21; also recall that, for every $u \in V_\bigcirc$, $\texttt{cnt}[u]$ was initialized to $|N^{\text{out}}_{\mathcal{A}}(u)|$ at line 7 of `tr-DFS`() (Algo. 1), and then $\texttt{cnt}[u]$ is decremented at line 10 of `tr-DFS-visit`$(v, \mathcal{A})$ each time some $v \in N^{\text{out}}_{\mathcal{A}}(u)$ is visited; with this in mind, it is easy to check that *(tr-pt-1)* to *(tr-pt-4)* are satisfied. Next, we claim that J is a tr-jungle with tr-palm-tree family $\{\mathcal{P}^i\}_{i\in[k]}$. Indeed, *(tr-jn-1)* and *(tr-jn-2)* clearly hold. Concerning *(tr-jn-3)*, let $(u, v) \in A$ for some $u \in V^i$ and $v \in V^j$ such that $i \neq j$; then $u \in V^i_\square$, because $\mathcal{P}^i$ is a



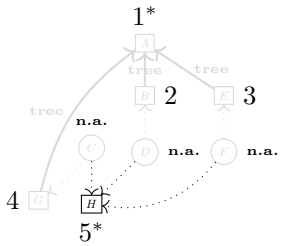
(a) An arena $\mathcal{A}$.



(b) The tr-palm-tree generated by a tr-DFS rooted at A, with indices of vertices and labelled arcs.

1. (B, A)
2. (D, B)
3. (E, A)
4. (F, E)
5. (G, A)
6. (C, G)

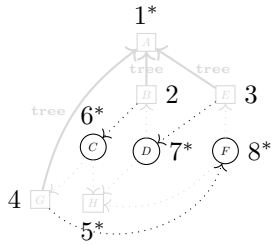
(c) The order of arcs' exploration.



(d) The tr-palm-tree generated by a tr-DFS rooted at H, with indices of vertices and labelled arcs.

7. (C, H)
8. (D, H)
9. (F, H)

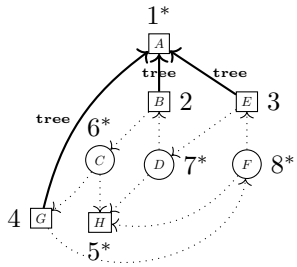
(e) The order of arcs' exploration.



(f) The tr-palm-trees generated by an tr-DFS rooted at C, D, F, with indices of vertices and labelled arcs.

10. (B, C)
11. (E, D)
12. (G, F)

(g) The order of arcs' exploration.



(h) The resulting tr-jungle, which is generated by multiple tr-DFSs rooted at A, H, C, D and F, with indices of vertices and labelled arcs.

Figure 3: An arena (a), and the construction of a tr-jungle (b-h).

tr-palm-tree so that *(tr-pt-3)* holds for $V_\bigcirc$; also, $i < j$ since otherwise u would've joined $\mathcal{P}^i$ at lines 6-8 of `tr-DFS-visit`(). Concerning *(tr-jn-4)*, let $v \in V \setminus \bigcup_{i=1}^{k} V^i$, then $v \in V_\bigcirc$ by lines 9-15 of `tr-DFS`(); also, $N^{\text{out}}_{\mathcal{J}}(v) \subseteq V^i$ holds for *no* $i \in [k]$, otherwise v would've joined $\mathcal{P}^i$ thanks to lines 9-13 and 17-21 of `tr-DFS-visit`(). So J is a tr-jungle. □

Proof of (2). Notice that J_* is obtained from J simply by removing from the tr-palm-trees of J all the arcs $(u, v) \in A_{\text{tree}}$ such that $u \in V_\bigcirc$. Consider the ordering $<_{\texttt{idx}}$ on V induced by the labelling `idx` of J, i.e., $\forall_{a,b \in V}\ a <_{\texttt{idx}} b \iff \texttt{idx}[a] < \texttt{idx}[b]$. Construct an adjacency list of J_* such that: (i) the main list of vertices is ordered according to $<_{\texttt{idx}}$; (ii) for each $u \in V$, also $N^{\text{in}}_J(u)$ is ordered according to $<_{\texttt{idx}}$. So doing, since J satisfies *(tr-jn-1)* to *(tr-jn-4)* and their tr-palm-trees satisfy *(tr-pt-1)* to *(tr-pt-4)*, then `tr-DFS`(J_*) (Algo. 1) reconstructs J itself, i.e., that $\mathcal{J}_{J_*} = J$. □

The next proposition shows that tr-jungles do respect trap-reachability.

Proposition 4. *Let $\mathcal{J}$ be a tr-jungle with family of tr-palm-trees $\{\mathcal{P}^i\}_{i=1}^k$, for some $k \in \mathbf{N}$, assume that $\mathcal{P}^i = (V^i, A^i, (V^i_\square, V^i_\bigcirc))$ holds for each $i \in [k]$. There exists $\sigma_\square \in \Sigma^{\mathcal{J}}_\square$ (i.e., $\sigma_\square = \sigma_\square(i)$) such that, for any two vertices $u, v \in V^i$, if u is a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$, then: $\sigma_\square : u \overset{V^i}{\rightsquigarrow} v$.*

Proof. By lines 9-11 of `tr-DFS`(), for every $u \in V_\square$ there exists some $i_u \in [k]$ such that $u \in V^{i_u}$.

Then, consider the strategy $\sigma_\square \in \Sigma^{\mathcal{J}}_\square$ (i.e., $\sigma_\square = \sigma_\square(i)$) defined as follows:

$$\forall_{u \in V_\square}\ \sigma_\square(u) \triangleq \begin{cases} \pi_{\mathcal{T}_{\mathcal{P}^{i_u}}}(u), & \text{if } u \text{ is } \textit{not} \text{ the root of } \mathcal{T}_{\mathcal{P}^{i_u}}; \\ \text{any } u' \in N^{\text{out}}_{\mathcal{J}}(u), & \text{if } u \text{ is the root of } \mathcal{T}_{\mathcal{P}^{i_u}}. \end{cases}$$

Let $i \in [k]$ and $u, v \in V^i$ be fixed, arbitrarily. Recall that, by *(tr-pt-1)*, the vertices of $\mathcal{T}_{\mathcal{P}^i}$ are numbered by `idx` so that $\texttt{idx}[v] < \texttt{idx}[u]$ if v is a proper ancestor of u in $\mathcal{T}_{\mathcal{P}^i}$. To prove the thesis, we argue by induction on the value of $\texttt{idx}[u]$. Let $z \triangleq \min_{x \in V^i} \texttt{idx}[x]$. Assume $\texttt{idx}[u] = z$. Then, $u = r_{\mathcal{T}_{\mathcal{P}^i}}$ is the root of $\mathcal{T}_{\mathcal{P}^i}$. Since $u, v \in V^i$ and u is a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$, then $v = u$; so, the thesis trivially holds. Instead, assume $\texttt{idx}[u] > z$. Again, if $u = v$ the claim holds trivially. So, let $u \neq v$. Assume the thesis holds for every $x \in V^i$ which is still a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$ and such that $\texttt{idx}[v] \leq \texttt{idx}[x] < \texttt{idx}[u]$.

We have two cases to analyze, either $u \in V_\square$ or $u \in V_\bigcirc$.

(i) If $u \in V_\square$, then $\sigma_\square(u) = \pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$. By *(tr-pt-1)*, $\texttt{idx}[\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)] < \texttt{idx}[u]$. Since $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is the parent of u in $\mathcal{T}_{\mathcal{P}^i}$ and $u \neq v$, then $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is still a descendant of v; thus, by induction hypothesis: $\sigma_\square : \pi_{\mathcal{T}_{\mathcal{P}^i}}(u) \overset{V^i}{\rightsquigarrow} v$. Therefore, since $\sigma_\square : u \overset{V^i}{\rightsquigarrow} \pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ and $\sigma_\square : \pi_{\mathcal{T}_{\mathcal{P}^i}}(u) \overset{V^i}{\rightsquigarrow} v$, the thesis holds.

(ii) If $u \in V_\bigcirc$, firstly recall that by *(tr-pt-3)*, $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is the LCA of $\Lambda^i(u) \triangleq \{u' \in V \mid (u, u') \in A^i_{\text{petiole}}\}$; notice that $\Lambda^i(u) = N^{\text{out}}_{\mathcal{J}_*}(u)$, where $\mathcal{J}_*$ is the support of $\mathcal{J}$. Fix $u' \in N^{\text{out}}_{\mathcal{J}_*}(u)$, arbitrarily. It holds that u' is still a descendant of $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ in $\mathcal{T}_{\mathcal{P}^i}$, because $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is the LCA of $N^{\text{out}}_{\mathcal{J}_*}(u)$. Thus, since $\pi_{\mathcal{T}_{\mathcal{P}^i}}(u)$ is a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$, then u' is also a descendant of v in $\mathcal{T}_{\mathcal{P}^i}$. And, by item *(c)* of *(tr-pt-3)*, $\texttt{idx}[u'] < \texttt{idx}[u]$. Thus, by induction hypothesis, $\sigma_\square : u' \overset{V^i}{\rightsquigarrow} v$. Since u' was chosen arbitrarily, then $\sigma_\square : u \overset{V^i}{\rightsquigarrow} v$. This concludes the inductive step of the proof. So, in any case, $\sigma_\square : u \overset{V^i}{\rightsquigarrow} v$. □

Still it remains to be seen how to perform the LCAs computations that are needed at lines 11-12 of `tr-DFS()` (Proc. 1). In the next paragraph, we suggest to adopt a *disjoint-set forest* data structure with *non-ranked* ***Union()*** and *path-compression* ***Find()***.

LCAs by Disjoint-Set Forest. A *disjoint-set forest (DSF)* data structure, hereby denoted by $\mathcal{D}$, also called *union-find* data structure or *merge-find* set, is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets, each of which is represented by a tree.

It supports the following operations:

$\mathcal{D}$.`MakeSet`$(\cdot)$, $\mathcal{D}$.`Union`$(\cdot,\cdot)$ and $\mathcal{D}$.`Find`$(\cdot)$, such that:

(dsf-1) The representative element of each set is the root of that set's tree;

(dsf-2) `MakeSet`(v) initializes the parent of a vertex $v \in V$ to be v itself;

(dsf-3) `Union`(u, v) combines two trees, T_1 rooted at u and T_2 rooted at v, into a new tree T_3 still rooted at v, i.e., by adding u as a child of v (*non-ranked union*).

(dsf-4) `Find`(v), starting from v, it traverses the ancestors of v until the root r of the tree containing v is finally reached. While doing this, it changes each ancestor's parent reference to point to r (*path-compression*); the resulting tree is much flatter, speeding up future operations, not only on these traversed elements but also on those referencing them.

We can now describe how to perform the LCAs computations at lines 11-12 of `tr-DFS-visit`() (Proc. 1). We refer to the following procedure as to the *"DSF-based* ***tr-DFS()****"*. We have a global DSF data structure named $\mathcal{D}$. The init-phase is almost the same as Algo. 1, the only additions being that, for each $v \in V$:

(dsf-init-1) $\mathcal{D}$.`MakeSet`(v) is executed;

(dsf-init-2) If $v \in V_{\bigcirc}$, an array $\texttt{low_ready} : V \to \mathbf{N} \cup \{+\infty\}$ is initialized as $\texttt{low_ready}[v] \leftarrow +\infty$. Given $\mathcal{A}$ in input, the DSF-based $\texttt{tr-DFS}(\mathcal{A})$ is going to keep the following invariant property:

$$\forall_{v \in V_{\bigcirc}}\ \texttt{low_ready}[v] = \min\big\{\texttt{idx}[u] \mid u \in N^{\text{out}}_{\mathcal{A}}(v)\big\}. \tag{I_{low}}$$

Next, the visit-phase begins as at lines 9-15 of $\texttt{tr-DFS}()$ (Algo. 1): for each $v \in V_{\square}$, if v is still unvisited (i.e., if $\texttt{idx}[v] = +\infty$) then $\texttt{tr-DFS-visit}(v, \mathcal{A})$ is invoked. As soon as all $V_{\square}$ vertices have been visited, then, all $v \in V_{\bigcirc}$ that are still unvisited are handled as at lines 12-15 of $\texttt{tr-DFS}()$ (Algo. 1).

Also, the halting-phase is the same as before, see lines 16-17 of $\texttt{tr-DFS}()$ (Algo. 1).

Let us now describe the distinctive features of the DSF-based $\texttt{tr-DFS}()$. Let $v \in V$, then:

(dsf-visit-1) Whenever the DSF-based $\texttt{tr-DFS-visit}(v, \mathcal{A})$ makes a recursive call on some input vertex $u \in N^{\text{in}}_{\mathcal{A}}(v) \cup \texttt{ready_St}[v]$ (see lines 8 and 21 of Proc. 1), soon after that, it is executed $\mathcal{D}.\texttt{Union}(u, v)$.

(dsf-visit-2) Suppose that the DSF-based $\texttt{tr-DFS-visit}(v, \mathcal{A})$ is currently exploring some $v \in V$, and that it comes to consider some $u \in N^{\text{in}}_{\mathcal{A}}(v) \cap V_{\bigcirc}$ (at line 4 and 9). Then, $\texttt{low_ready}$ is updated as follows:

$$\texttt{low_ready}[u] \leftarrow \min(\texttt{low_ready}[u], \texttt{idx}[v]);$$

this aims at satisfying the I_{low} invariant. Next, $\texttt{cnt}[u]$ is decremented (as at line 10 of Proc. 1).

If the condition $\texttt{cnt}[u] = 0$ is met (see line 11 of Proc. 1), the following is done:

(a) It is assigned $\texttt{low_}v \leftarrow$ "the unique $x \in N^{\text{out}}_{\mathcal{A}}(u)$ such that $\texttt{idx}[x] = \texttt{low_ready}[u]$";

(b) Then, it is computed $\gamma \leftarrow \mathcal{D}.\texttt{Find}(\texttt{low_}v)$;

(c) Then, if $\texttt{active}[\gamma] = \texttt{true}$, it is executed $\texttt{ready_St}[\gamma].\texttt{push}(u)$; indeed, in that case, we can prove (see Proposition 5) that the LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ in (V, A_{tree}) does exist, and it is really γ.

The rest of the DSF-based $\texttt{tr-DFS-visit}()$ is the same as Proc. 1.

This concludes the description of the DSF-based $\texttt{tr-DFS}()$.

At this point we shall prove that the above mentioned property concerning γ and LCAs holds.

Proposition 5. *Suppose that $\texttt{tr-DFS-visit}(v, \mathcal{A})$ (DSF-based) is invoked, for some $v \in V$, and that it comes to consider some $u \in N^{in}_{\mathcal{A}}(v) \cap V_{\bigcirc}$ (at line 4 and 9). Assume that $\texttt{idx}[u] = +\infty$ at line 5 and that $\texttt{cnt}[u] = 0$ at line 11. Let γ be the vertex returned by $\mathcal{D}.\texttt{find}(\texttt{low_}v)$, where $\texttt{low_}v$ is the unique $x \in V$ such that $\texttt{idx}[x] = \texttt{low_ready}[u]$. If $\texttt{active}[\gamma] = \texttt{true}$ holds, then the LCA of $N^{out}_{\mathcal{A}}(u)$ in (V, A_{tree}) is γ.*

Proof. Firstly, during the execution of the DSF-based $\texttt{tr-DFS}(\mathcal{A})$, the (V, A_{tree}) still grows as a forest. Indeed, if a new arc (u, v) is added to A_{tree} it holds that $\texttt{idx}[u] = +\infty$ (by line 5 of $\texttt{tr-DFS-visit}()$) and that $\texttt{idx}[v] < +\infty$ (by line 2 of $\texttt{tr-DFS-visit}(v, \mathcal{A})$); so, no cycle can be formed. Thus, when $\texttt{tr-DFS-visit}(v, \mathcal{A})$ is invoked for $v \in V$, we can consider the unique maximal tree $\mathcal{T}^v$ in (V, A_{tree}) which contains v (constructed up to that point). Let p_v be the path in $\mathcal{T}^v$ going from v to the root $r_{\mathcal{T}^v}$. By *(dsf-visit-1)*, by definition of $\texttt{low_}v$, and since $\gamma = \mathcal{D}.\texttt{find}(\texttt{low_}v)$ and $\texttt{active}[\gamma] = \texttt{true}$, then γ lies on p_v. Thus, γ is the LCA of $\texttt{low_}v$ and v in $\mathcal{T}^v$ (possibly $\gamma = \texttt{low_}v$). Next, we argue that $N^{\text{out}}_{\mathcal{A}}(u) \subseteq \mathcal{T}^v_{\gamma}$, where $\mathcal{T}^v_{\gamma}$ is the subtree of $\mathcal{T}^v$ comprising all and only the descendants of γ (i.e., the subtree of $\mathcal{T}^v$ rooted at γ). Indeed, by *(dsf-visit-2)*, it holds that:

$$\texttt{idx}[\texttt{low_}v] = \min\big\{\texttt{idx}[x] \mid x \in N^{\text{out}}_{\mathcal{A}}(u)\big\}.$$

So, when $\texttt{cnt}[u] = 0$ holds at line 11 of $\texttt{tr-DFS-visit}(v, \mathcal{A})$, and since γ is an ancestor of $\texttt{low_}v$, then:

$$\forall_{x \in N^{\text{out}}_{\mathcal{A}}(u)} \texttt{idx}[\gamma] \leq \texttt{idx}[\texttt{low_}v] \leq \texttt{idx}[x] < +\infty.$$

Notice that all vertices in $\mathcal{T}^v$ which are not descendants of γ still have a smaller $\texttt{idx}$ than γ (i.e., they were all visited before γ), and all those which are proper descendants of γ have a greater $\texttt{idx}$ than γ. All these combined, it follows $N^{\text{out}}_{\mathcal{A}}(u) \subseteq \mathcal{T}^v_{\gamma}$. So, γ is a common ancestor of $N^{\text{out}}_{\mathcal{A}}(u)$ in $\mathcal{T}^v$; but γ is also the LCA of $\{\texttt{low_}v, v\} \subseteq N^{\text{out}}_{\mathcal{A}}(u)$, therefore, γ is actually the LCA of $N^{\text{out}}_{\mathcal{A}}(u)$ in $\mathcal{T}^v$. $\square$

By Proposition 5, then, Proposition 3 holds even for the DSF-based $\texttt{tr-DFS}()$. Concerning complexity, by relying on the result offered in [6], we now show that the DSF-based $\texttt{tr-DFS}()$ halts in linear time.

Definition 8 ([6]). *Let $T = (V, A)$ be any rooted tree. Let $u_1, \ldots, u_k$ be a path in T listed from a leaf u_1 in the direction towards the root of T (i.e., u_k is some ancestor of u_1). The* path compression *$C = (u_1, \ldots, u_k)$ is an operation that modifies T as follows:*

(i) It deletes from T all the arcs (u_i, u_{i+1}), for $i = 1, \ldots, k-1$;

(ii) It makes each of the vertices u_i, for $i = 1, \ldots, k-1$, a new son of u_k;

(iii) It deletes all new sons of u_k of degree 1 which may occur (particularly, u_1 is deleted).

The vertex u_k is called the root *of C. We also say that C* starts *from u_1. The* length *of C is $|C| \triangleq k-1$.*

Any sequence $S = (C_1, \ldots, C_n)$ of path compressions on a tree T is called a strong postorder path compression system (SPPCS) *if and only if the following four properties hold:*

(i) Each C_i is a path compression on the tree T^i obtained from T after that the path compressions $C_1, \ldots, C_{i-1}$ have been executed (where $T^1 = T$);

(ii) Each leaf of T is a starting point of exactly one path compression of S;

(iii) $(1, 2, \ldots, n)$ is a linear ordering of all the n leaves of T induced by a fixed postorder of T;

(iv) Let the root of a compression C_i, for any $2 \leq i \leq n$, be some vertex u of T. Then, all the compressions C_j such that $j < i$ and $j \in T_u$ have roots in a descendant of u in T.

The length *of S is defined as $|S| \triangleq \sum_{i=1}^{n} |C_i|$.*

Theorem 1 ([6]). *Let S be a SPPCS on a tree T with n leaves. Then, $|S| \leq 5 \cdot n$.*

Proposition 6. *Given $\mathcal{A}$, assume that the DSF-based* `tr-DFS`$(\mathcal{A})$ *is invoked. Then, it halts in $\Theta\big(|V| + |A|\big)$ time.*

Proof. Recall that during the `tr-DFS()`, the graph (V, A_{tree}) grows as a forest. By *(dsf-visit-1)*, that forest coincides with the DSF that is constructed by means of the $\mathcal{D}$.`Union`$(\cdot, \cdot)$ operations.

So, in order to rely on Theorem 1, let us consider the following directed rooted tree $T^* \triangleq (V_{T^*}, A_{T^*})$:

$$\begin{aligned} V_{T^*} \triangleq & V \cup \big\{r_{T^*}\big\} \cup \big\{l_{(u,v)} \mid (u,v) \in A, u \in V_{\bigcirc}\big\}; \\ A_{T^*} \triangleq & A_{\texttt{tree}} \cup \big\{(r_{\mathcal{T}}, r_{T^*}) \mid \mathcal{T} \text{ is a tree in } (V, A_{\texttt{tree}}) \text{ and } r_{\mathcal{T}} \text{ is its root}\big\} \\ & \cup \big\{(l_{(u,v)}, v) \mid l_{(u,v)} \in V_{T^*}\big\}. \end{aligned}$$

where $r_{T^*}, l_{(u,v)} \notin V$ for every $l_{(u,v)} \in V_{T^*}$. Notice that r_{T^*} is the root of T^* and $\{l_{(u,v)} \in V_{T^*}\}$ is a subset of the leaves of T^*; so, for each $u \in V_{\bigcirc}$ and $v \in N^{\text{out}}_{\mathcal{A}}(u)$, there is a new leaf $l_{(u,v)}$ attached to v in T^*. Also notice that $|V_{T^*}| = 1 + |V| + |\{(u,v) \in A, u \in V_{\bigcirc}\}| \leq 1 + |V| + |A|$ and $|A_{T^*}| = |V_{T^*}| - 1$. Now, observe that each $\mathcal{D}$.`Find`() operation, that is possibly made by `tr-DFS`$(\mathcal{A})$, it is made only by `tr-DFS-visit`$(v, \mathcal{A})$ (for some $v \in V$) as prescribed by *items (a) and (b)* of *(dsf-visit-2)* and only if `cnt`$[u] = 0$ holds at line 11, i.e., $\gamma \leftarrow \mathcal{D}$.`Find`(`low_v`), where `low_v` is the unique $x \in N^{\text{out}}_{\mathcal{A}}(u)$ such that `idx`$[x]$ = `low_ready`$[u]$, and $u \in V_{\bigcirc} \cap N^{\text{in}}_{\mathcal{A}}(v)$ at lines 9-10. Each of these $\mathcal{D}$.`Find()` operations acts in a natural manner on T^*: indeed, $\mathcal{D}$.`Find`(`low_v`) induces a path compression $C_{\texttt{low_}v}$ on T^*, if we assume that $C_{\texttt{low_}v}$ starts at the leaf $l_{(u,\texttt{low_}v)}$ and that it terminates at γ (i.e., γ is the root of $C_{\texttt{low_}v}$). Since $\gamma \leftarrow \mathcal{D}$.`Find`(`low_v`) is executed only if `cnt`$[u] = 0$ holds at line 11 of `tr-DFS-visit`$(v, \mathcal{A})$, then each path compression on T^* starts from a distinct leaf $l_{(u,\texttt{low_}v)}$. It is safe to assume that each leaf of T^* is a starting point of exactly one path compression, because for each leaf l' of T^* that has not been the starting point of any path compression, we can impose a *void* path compression, i.e., one that starts and terminates at l'. Then, we argue that the family of all path compressions on T^* that are induced by the whole execution of `tr-DFS`$(\mathcal{A})$ is a SPPCS: indeed, during the search, T^* is (implicitly) visited in a post-ordering; when some $v \in V$ is visited, and some $u \in N^{\text{in}}_{\mathcal{A}}(v) \cap V_{\bigcirc}$ is considered at line 4 of `tr-DFS-visit`$(v, \mathcal{A})$, then the root γ of the path compression $C_{\texttt{low_}v}$ is the LCA of $\{v, \texttt{low_}v\}$ in T^* (as shown in Proposition 5). Thus, we argue that *(sppcs-4)* holds. Assume some path compression $C_{\texttt{low_}v'}$ was done before $C_{\texttt{low_}v}$ and that $\texttt{low_}v' \in T^*_{\gamma}$. So, `idx`$[\gamma] <$ `idx`$[\texttt{low_}v']$. Also, by *(dsf-visit-2)*, $C_{\texttt{low_}v'}$ was induced during `tr-DFS-visit`$(v', \mathcal{A})$, for some $v' \in V$. Thus, since $C_{\texttt{low_}v'}$ was done

before $C_{\texttt{low_}v}$, then $\texttt{idx}[v'] < \texttt{idx}[v]$; and by definition of $\texttt{low_}v'$, then $\texttt{idx}[\texttt{low_}v'] < \texttt{idx}[v']$. Therefore, $\texttt{idx}[\gamma] < \texttt{idx}[v'] < \texttt{idx}[v] < +\infty$ holds when $C_{\texttt{low_}v'}$ is performed. This means that $v' \in T^*_\gamma$. Since the root γ' of $C_{\texttt{low_}v'}$ is the LCA of $\{v', \texttt{low_}v'\}$ in T^* (as shown in Proposition 5), and since $\{v', \texttt{low_}v\} \subseteq T^*_\gamma$, then $\gamma' \in T^*_\gamma$; so, *(sppcs-4)* holds. At this point, by Theorem 1, the total length of all path compressions that are induced by `tr-DFS()` on T^* is $O(|V_{T^*}|) = O(|V| + |A|)$. It is clear that the space required for storing $\mathcal{D}$ is $\Theta(|V| + |A|)$. So, also by Proposition 2, the complexity of the DSF-based `tr-DFS()` is $\Theta(|V| + |A|)$. □

5 Linear Time Algorithm for STCCs

Lemma 3. *Let $C_0, \ldots, C_{k-1} \subseteq V$ be some STCCs of $\mathcal{A}$, for some $k \geq 2$. For each $i \in \{0, \ldots, k-1\}$ fix some $u_i \in C_i$, arbitrarily; and let $i' \triangleq (i+1) \mod k$. Assume that the following* tr-cycle *(i.e., cyclic trap-reachability relation) holds for some $\{\sigma_\square(u_i, u_{i'})\}_{i=0}^{k-1} \subseteq \Sigma^{\mathcal{A}}_\square$:*

$$\sigma_\square(u_i, u_{i'}) : u_i \overset{C_i \cup \{u_{i'}\}}{\rightsquigarrow} u_{i'}, \text{ for each } i = 0, \ldots, k-1.$$

Then, $C_i = C_{i'}$ for every $i \in \{0, \ldots, k-1\}$.

Proof. Let $C^* \triangleq \cup_{i=0}^{k-1} C_i$. We argue that C^* is strongly-trap-connected. Let $x, y \in C^*$ be fixed arbitrarily, where $x \in C_{i_x}$ and $y \in C_{i_y}$, for some $i_x, i_y \in \{0, \ldots, k-1\}$. If $i_x = i_y$, the following holds for some $\sigma_\square(x, y) \in \Sigma^{\mathcal{A}}_\square$ (because $C_{i_x} = C_{i_y}$ is strongly-trap-connected):

$$\sigma_\square(x, y) : x \overset{C_{i_x} = C_{i_y}}{\rightsquigarrow} y.$$

Otherwise, $i_x \neq i_y$. Then, $\sigma_\square(x, u_{i_x}) : x \overset{C_{i_x}}{\rightsquigarrow} u_{i_x}$ for some $\sigma_\square(x, u_{i_x}) \in \Sigma^{\mathcal{A}}_\square$ (because C_{i_x} is strongly-trap-connected). Next, this holds for each $i = i_{i_x}, i_{i_x}+1, \ldots, k-1, 0, 1, \ldots, i_y - 1$ and for $i' \triangleq (i+1) \mod k$:

$$\sigma_\square(u_i, u_{i'}) : u_i \overset{C_i \cup \{u_{i'}\}}{\rightsquigarrow} u_{i'}.$$

Finally, $\sigma_\square(u_{i_y}, y) : u_{i_y} \overset{C_{i_y}}{\rightsquigarrow} y$ for some $\sigma_\square(u_{i_y}, y) \in \Sigma^{\mathcal{A}}_\square$ (because C_{i_y} is strongly-trap-connected). Therefore, by composition, there exists $\sigma_\square(x, y) \in \Sigma^{\mathcal{A}}_\square$ such that $\sigma_\square(x, y) : x \overset{C^*}{\rightsquigarrow} y$; so C^* is strongly-trap-connected. At this point, for every $i \in \{0, \ldots, k-1\}$, since $C_i \subseteq C^*$, since C^* is strongly-trap-connected, and since C_i is a STCC of $\mathcal{A}$ (so the maximality property mentioned in Def. 3 holds), then, $C_i = C^*$. □

Proposition 7. *Let $\mathcal{J}_\mathcal{A}$ be the tr-jungle constructed by `tr-DFS`($\mathcal{A}$) (Algo. 1) (see Proposition 3). Let $\{\mathcal{P}^i\}_{i=1}^k$ be the tr-palm-tree's family of $\mathcal{J}_\mathcal{A}$, for some $k \in \mathbf{N}$, where $\mathcal{P}^i \triangleq (V^i, A^i, (V_\square \cap V^i, V_\bigcirc \cap V^i))$ and $A^i \triangleq A^i_{tree} \uplus A^i_{frond} \uplus A^i_{petiole} \uplus A^i_{cross}$; also, $F \triangleq (V, \bigcup_{i=0}^k A^i_{tree})$ is a forest by Defs. 5-6. Let $\mathcal{C} \subseteq V$ be any STCC of $\mathcal{A}$. Then, $\mathcal{C}$ induces a subtree $T_\mathcal{C}$ in F (i.e., $F[\mathcal{C}]$ is an inward directed rooted tree).*

Proof. By Proposition 2, each $v \in V$ is numbered by `idx` exactly once. Let $v^* \triangleq \arg\min_{x \in \mathcal{C}} \texttt{idx}[x]$ be the first vertex $v^* \in \mathcal{C}$ that is visited during `tr-DFS`($\mathcal{A}$) (Algo. 1). By Proposition 3 and Defs. 5-6, the set of all vertices that are visited during the whole (i.e., including recursive calls) execution of `tr-DFS-visit`($v^*, \mathcal{A}$) induces a subtree $T_{v^*} \triangleq (V_{T_{v^*}}, A_{T_{v^*}})$ of F which is rooted at v^*; so, $v^* \in \mathcal{C} \cap T_{v^*} \neq \emptyset$.

<u>*Fact-1*</u>*:* $\mathcal{C} \subseteq T_{v^*}$. For the sake of contradiction, suppose $\mathcal{C} \setminus T_{v^*} \neq \emptyset$. Then, since $\mathcal{C}$ is strongly-trap-connected, there exists $\hat{u} \in \mathcal{C} \setminus T_{v^*}$ such that one of the following two holds: either (i) $\hat{u} \in V_\square$ and there exists $u' \in N^{\text{out}}_\mathcal{A}(\hat{u})$ such that $u' \in \mathcal{C} \cap T_{v^*}$; or (ii) $\hat{u} \in V_\bigcirc$ and for all $u' \in N^{\text{out}}_\mathcal{A}(\hat{u})$ it holds that $u' \in \mathcal{C} \cap T_{v^*}$. Also notice, since $v^* \triangleq \arg\min_{x \in \mathcal{C}} \texttt{idx}[x]$, then $\texttt{idx}[v^*] < \texttt{idx}[\hat{u}]$; thus $\hat{u}$ was not visited before v^*. All these combined, by definition of `tr-DFS-visit()` (Proc. 2) and since T^*_v is a subtree of F, it must be that $\hat{u}$ is visited and that it joins F during the execution of `tr-DFS-visit`($v^*, \mathcal{A}$); so, $\hat{u} \in T_{v^*}$. But this contradicts our assumption $\hat{u} \in \mathcal{C} \setminus T_{v^*}$. Therefore, $\mathcal{C} \setminus T_{v^*} = \emptyset$; so, $\mathcal{C} \subseteq T_{v^*}$.

<u>*Fact-2*</u>*:* If $u \in \mathcal{C} \setminus \{v^*\}$ and $u' \triangleq \pi_{T_{v^*}}(u)$, then $u' \in \mathcal{C}$. Indeed, since $u \in \mathcal{C}$ and $\mathcal{C} \subseteq T_{v^*}$, then u is a descendant of v^* in T_{v^*}. Since $u \neq v^*$, then u' is also a descendant of v^* in T_{v^*}. Thus, by Proposition 4, there exists some $\sigma_\square \in \Sigma^{\mathcal{A}}_\square$ such that:

$$\sigma_\square : u \overset{T_{v^*}}{\rightsquigarrow} u' \text{ and } \sigma_\square : u' \overset{T_{v^*}}{\rightsquigarrow} v^*;$$

thus, since $u, v^* \in \mathcal{C}$ and $\mathcal{C}$ is a STCC, then $u' \in \mathcal{C}$ holds by Lemma 3.

By *(fact-1)* and *(fact-2)*, $\mathcal{C}$ induces a tree $T_{\mathcal{C}}$ in F (i.e., $T_{\mathcal{C}}$ is a subtree of T_{v^*} still rooted at v^*). □

Definition 9. *The root v^* of the tree $T_{\mathcal{C}}$ (as in the proof of Proposition 7) is the* root *of the STCC $\mathcal{C}$.*

The problem of computing the STCCs of any arena $\mathcal{A}$ reduces to that of finding the roots of the STCCs; just as the classical problem of finding the SCCs of a directed graph reduced to that of finding the roots of the SCCs. We have identified a simple test to determine if a vertex is the root of a STCC. It is based on a *lowlink* indexing, generalizing the lowlink calculation performed by Tarjan's SCC algorithm [7].

Definition 10. *Let $\mathcal{J}_{\mathcal{A}}$ be the tr-jungle constructed by* `tr-DFS`$(\mathcal{A})$ *(Algo. 1). Let* `idx` $: V \to \{1, \ldots, |V|\}$ *be the indexing computed during that execution, and let $\{\mathcal{P}^i\}_{i=1}^k$ be the tr-palm-tree's family of $\mathcal{J}_{\mathcal{A}}$, for some $k \in \mathbf{N}$, where $\mathcal{P}^i \triangleq (V^i, A^i, (V_\square \cap V^i, V_\bigcirc \cap V^i))$ and $A^i \triangleq A^i_{tree} \uplus A^i_{frond} \uplus A^i_{petiole} \uplus A^i_{cross}$; also, $F \triangleq (V, \bigcup_{i=0}^k A^i_{tree})$ is a forest by Defs. 5-6. Given any $v \in V$, the tr-lowlink $: V \to \mathbf{N}$ is defined as follows:*

$$\textit{tr-lowlink}(v) \triangleq \min\Big(\big\{\texttt{idx}[v]\big\} \uplus \big\{\texttt{idx}[u] \mid u \in V \setminus \{v\} \text{ and } \exists_{i\in[k]} \text{ such that the following two hold:}$$

$$\textit{(tr-ll-1)}\ \exists_{t\geq 1}\ \exists_{(u,v_1,\ldots,v_{t-1},(v_t=v))\in(V^i)^+} \text{ such that:}$$

$$\textit{(a)}\ (u, v_1) \in A_{frond} \uplus A_{cross};$$

$$\textit{(b)}\ \text{ if } t \geq 2, \forall_{j\in\{1,\ldots,t-1\}} \text{ it holds } (v_j, v_{j+1}) \in A_{tree}.$$

$$\textit{(tr-ll-2)}\ \exists_{\gamma\in V^i} \text{ such that:}$$

$$\textit{(a)}\ \gamma \text{ is a common ancestor of } u \text{ and } v \text{ in } (V^i, A^i_{tree});$$

$$\textit{(b)}\ \gamma \text{ and } u \text{ are in the same STCC of } \mathcal{A}.$$

$$\big\}$$

$$\Big).$$

Moreover, given $v \in V$, let $N^{in}_{\mathcal{A},LCA}(v) \triangleq \{u \in N^{in}_{\mathcal{A}}(v) \mid$ the LCA γ of $\{u, v\}$ in F exists and $\gamma \in \mathcal{C}_u\}$.

Let us prove some useful properties of tr-lowlink() and $N^{\text{in}}_{\mathcal{A},\text{LCA}}$.

Proposition 8. *Let $\mathcal{A}$,* `idx`$, F$ *and tr-lowlink() be as in Def. 10. Given any $v \in V$:*

1. *If tr-lowlink$(v) =$* `idx`$[u]$ *for some $u \in V \setminus \{v\}$ such that* (tr-ll-1) *and* (tr-ll-2) *hold, then $u \in V_\square$.*
2. *tr-lowlink$(v) = \min\big\{$*`idx`$[v]\big\} \uplus \big\{$*`idx`$[u] \mid u \in N^{in}_{\mathcal{A},LCA}(v)\big\} \uplus \big\{$tr-lowlink$(u) \mid u$ is a child of v in $F\big\}$.*

Proof of (1). By Item (a) of (tr-ll-1), it holds $(u, v_1) \in A_{\text{frond}} \uplus A_{\text{cross}}$. Recall that (u, v_1) can be added to $A_{\text{frond}} \uplus A_{\text{cross}}$ only at lines 15-16 of `tr-DFS-visit`$(v_1, \mathcal{A})$ (Proc. 1). So, u was visited before v_1. Still, $u \in N^{\text{in}}_{\mathcal{A}}(v_1)$ by Item (a) of (tr-ll-1); then, it is not possible that $u \in V_\bigcirc$, because any $x \in V_\bigcirc$ can join F only if `cnt`$[x] = 0$ holds at line 11 of `tr-DFS-visit()` (Proc. 1), and u was visited before v_1 and yet it joined F. Therefore, $u \in V_\square$. □

Proof of (2). Assume that tr-lowlink$(v) =$ `idx`$[u]$ for some $u \in V \setminus \{v\}$ such that (tr-ll-1) and (tr-ll-2) hold. Then, $\exists_{t\geq 1}\ \exists_{(u,v_1,\ldots,v_{t-1},(v_t=v))\in(V^i)^+}$ as in Def. 10. If $t = 1$, then $v_1 = v$; so, $u \in N^{\text{in}}_{\mathcal{A},\text{LCA}}(v)$ (it is easy to see that, if (tr-ll-2) holds for *some* common ancestor of $\{u, v\}$ in F, then it holds for the LCA). If $t > 1$, then v_1 is a proper descendant of v in F. At this point, it is easy to check tr-lowlink$(v) =$ `idx`$[u] =$ tr-lowlink$(v_1) =$ tr-lowlink(c) holds for some child c of v in F, indeed, this follows from Def. 10 and Proposition 7. □

Similarly to Tarjan's lowlink based algorithm [7], the tr-lowlink(v) is thus the smallest index of any vertex u which is in the same STCC as v and such that u can reach v by traversing: at most one frond (i.e., A^i_{frond}) or cross-link (i.e., A^i_{cross}) arc by *item (a)* of *(tr-ll-1)*, then, zero or more tree (i.e., A^i_{tree}) arcs by *item (b)* of *(tr-ll-1)*.

What follows is of a pivotal importance for computing STCCs by relying on tr-lowlinks.

Proposition 9. *Let $\mathcal{J}_{\mathcal{A}}$ be the tr-jungle constructed by* `tr-DFS`$(\mathcal{A})$ *(Algo. 1), and let* $\mathtt{idx} : V \to \{1, \ldots, |V|\}$ *be the indexing constructed during that execution. Finally, let tr-lowlink* $: V \to \mathbf{N}$ *be as in Def. 10.*

For any $v \in V$, it holds that v is the root of some STCC of $\mathcal{A}$ iff tr-lowlink$(v) = \mathtt{idx}[v]$.

Proof. Let $v \in V$. By Proposition 7, the STCC $\mathcal{C}_v$ induces a subtree $T_{\mathcal{C}_v}$ in F. Let v^* be the root of $T_{\mathcal{C}_v}$.

($\Rightarrow$) Assume $v = v^*$. Then, we argue there can be *no* $u \in V \setminus \{v\}$ such that tr-lowlink$(v) = \mathtt{idx}[v]$, i.e., such that $\mathtt{idx}[u] < \mathtt{idx}[v]$ and both *(tr-ll-1)* and *(tr-ll-2)* (see Def. 10) hold on u. For the sake of contradiction, assume the existence of such an u. Then, since $\mathtt{idx}[u] < \mathtt{idx}[v]$ and v is the root of $T_{\mathcal{C}_v}$, it would be $u \notin \mathcal{C}_v$. By *(tr-ll-1)* (Def. 10), there exists a path $\langle u, v_1, \ldots, v_{t-1}, (v_t = v)\rangle$ in $\mathcal{A}$, for some $t \geq 1$, such that $(u, v_1) \in A_{\text{frond}} \uplus A_{\text{cross}}$ and, if $t \geq 2$, $\forall_{j \in \{1,\ldots,t-1\}}$ it holds that $(v_j, v_{j+1}) \in A_{\text{tree}}$. Also, by *(tr-ll-2)* (Def. 10), there exists a common ancestor γ of u and v in $(V, A_{\mathtt{tree}})$, and $\gamma \in \mathcal{C}_u$. All these combined, by Proposition 4 and Lemma 3 (applied to the tr-cycle $v\gamma uv$), it would be $\mathcal{C}_v = \mathcal{C}_u$. This is absurd, because $u \in \mathcal{C}_u$ and $u \notin \mathcal{C}_v$. Indeed, there is no such u. Therefore, tr-lowlink$(v) = \mathtt{idx}[v]$.

($\Leftarrow$) Assume $v \neq v^*$. Since $\mathcal{C}_v$ is strongly-trap-connected and $v, v^* \in \mathcal{C}_v$, then $v^* \rightsquigarrow v$. Let T_v be the subtree of $T_{\mathcal{C}_v}$ that is rooted at v. Since $v^* \rightsquigarrow v$, there exists some $u \in V \setminus V_{T_v}$ (possibly, $u = v^*$) and some $v_1 \in T_v$ (possibly, $v_1 = v$) such that: (i) $v^* \rightsquigarrow u$ and $u \rightsquigarrow v_1$; plus, (ii) $(u, v_1) \in A_{\text{frond}} \uplus A_{\text{cross}}$. Since $u \notin T_v$ and $v_1 \in T_v$, and since $(u, v_1) \in A_{\text{frond}} \uplus A_{\text{cross}}$, then $u \in V_\square$ and $\mathtt{idx}[u] < \mathtt{idx}[v]$. Moreover, since $v^* \rightsquigarrow u \rightsquigarrow v_1 \rightsquigarrow v$, and finally (by Proposition 4) $v \rightsquigarrow v^*$, then u is in the same STCC as $\{v^*, v_1, v\}$ (by Lemma 3, applied to the tr-cycle $v^*uv_1vv^*$); i.e., $u \in \mathcal{C}_v$. Thus, u satisfies both *(tr-ll-1)*, *(tr-ll-2)*, so tr-lowlink$(v) \leq \mathtt{idx}[u]$; and since $\mathtt{idx}[u] < \mathtt{idx}[v]$, then tr-lowlink$(v) < \mathtt{idx}[v]$. $\square$

When `tr-lowlink`$(v) = \mathtt{idx}[v]$ holds, as in Proposition 9, we say that v is a fixpoint of tr-lowlink(); so, given any arena $\mathcal{A}$, the roots of the STCCs of $\mathcal{A}$ are exactly the fixpoints of tr-lowlink().

Our algorithm for decomposing $\mathcal{A}$ into STCCs is described next, it is based on the DSF-based `tr-DFS()` (see Algo. 1); still, it has some additional and distinctive features:

(1) All vertices that have already been reached during the `tr-DFS()`, but whose STCC has not yet been completely identified, are stored on a stack named `St`;

(2) The stack `St` is (partially) emptied, and a brand new STCC $\mathcal{C}$ is completely identified, when the tr-lowlink's fixpoint condition `tr-lowlink`$(v) = \mathtt{idx}[v]$ is met (see Proposition 9).

(3) The STCC algorithm does not build any tr-jungle's forest F explicitly (i.e., there is no real need to keep track of $A_{\text{tree}}, A_{\text{frond}}, A_{\text{petiole}}, A_{\text{cross}}$); still, a tr-jungle's forest F is defined implicitly, by the sequence of vertices that are visited and backtracked during the search process.

It will be convenient to consider this tr-jumgle during the proof of correctness, so let us refer to the corresponding (implicitly constructed) F as to the *STCC forest* (recall that it would have been $F \triangleq (V, A_{\text{tree}})$ in Algo. 1).

The STCCs main procedure is called `compute-STCCs()`, it takes in input an arena $\mathcal{A}$, and it aims at printing out all the STCCs $\mathcal{C}_1, \ldots, \mathcal{C}_k$ of $\mathcal{A}$ (w/o repetitions). A procedure named `STCCs-visit()` is also employed. The pseudo-code is given in Algo. 2 and Proc. 2, respectively.

The initialization phase goes from line 1 to 9 of Algo. 2. The visit-phase starts by setting `next_idx` $\leftarrow 1$ and `St` $\leftarrow \emptyset$ (lines 10-18 of Algo. 2). Firstly, $V_\square$ is considered: for each unvisited $u \in V_\square$, `STCCs-visit`$(u, \mathcal{A})$ is invoked. Then, all the $u \in V_\bigcirc$ which are still unvisited are handled as in Algo. 1.

Consider `STCCs-visit`$(v, \mathcal{A})$ (Proc. 2), for $v \in V$. This is similar to the DSF-based implementation of `tr-DFS-visit()` (Proc. 1), the significant changes going as follows. Initially, v is pushed on top of `St` and `on_Stack`$[v] \leftarrow$ `true` is set (lines 4-5).

Algorithm 2: Computing STCCs

```
Procedure compute-STCCs(A)
    input  : An arena A = (V, A, (V_○, V_□)).
    output: The STCCs of A.
1   foreach u ∈ V do
2       idx[u] ← +∞;
3       tr-lowlink[u] ← +∞;
4       on_Stack[u] ← false;
5       D.make_set(u);
6       ready_St[u] ← ∅;
7       if u ∈ V_○ then
8           low_ready[u] ← +∞;
9           cnt[u] ← |N_A^out(u)|;
10  next_idx ← 1; St ← ∅;
11  foreach u ∈ V_□ do
12      if idx[u] = +∞ then
13          STCCs-visit(u, A);
14  foreach u ∈ V_○ do
15      if idx[u] = +∞ then
16          idx[u] ← next_idx;
17          next_idx ← next_idx + 1;
18          ta_lowlink[u] ← idx[u];
```

Then, each time `STCCs-visit`$(u, \mathcal{A})$ is invoked recursively (line 9 and 25), for some $u \in N^{\text{in}}_{\mathcal{A}}(v)$, `tr-lowlink`$[v] \leftarrow \min($`tr-lowlink`$[v]$, `tr-lowlink`$[u])$ is updated and $\mathcal{D}$.`Union`(u, v) is executed.

When exploring $N^{\text{in}}_{\mathcal{A}}(v)$ (line 6): If $u \in N^{\text{in}}_{\mathcal{A}}(v) \cap V_\bigcirc$ is unvisited (i.e., $\mathtt{idx}[u] = +\infty$), and if $\mathtt{cnt}[u] = 0$ holds, we seek for the LCA γ of $N^{\text{out}}_{\mathcal{A}}(u)$ in F, as in the DSF-based implementation of `tr-DFS-visit()`;

Procedure 2: The `STCCs-visit()` procedure.

```
Procedure STCCs-visit(v, A)
    input : A vertex v ∈ V.
1   idx[v] ← next_idx;
2   tr-lowlink[v] ← next_idx;
3   next_idx ← next_idx + 1;
4   St.push(v);
5   on_Stack[v] ← true
    // Check the in-neighbourhood of v
6   foreach u ∈ N_A^in(v) do
7       if idx[u] = +∞ then
8           if u ∈ V_□ then
9               STCCs-visit(u, A);
10              tr-lowlink[v] ← min(tr-lowlink[v], tr-lowlink[u]);
11              D.Union(u, v);
12          else
13              low_ready[u] ← min(low_ready[u], idx[v]);
14              cnt[u] ← cnt[u] − 1;
15              if cnt[u] = 0 then
16                  low_v ← the unique x such that idx[x] = low_ready[u];
17                  γ ← D.find(low_v);
18                  if on_Stack[γ] = true then
19                      ready_St[γ].push(u);
20      else if on_Stack[u] = true then
21          tr-lowlink[v] ← min(tr-lowlink[v], idx[u]);
    // Check the ready-stack of v, i.e., ready_St[v]
22  while ready_St[v] ≠ ∅ do
23      u ← ready_St[v].pop(); // u ∈ V_○
24      if ∀(x ∈ N_A^out(u)) on_Stack[x] = true then
25          SCCs-visit(u, A);
26          tr-lowlink[v] ← min(tr-lowlink[v], tr-lowlink[u]);
27          D.union(u, v);
    // Check whether a new STCC has to be constructed and printed to output
28  if tr-lowlink[v] = idx[v] then
29      C ← ∅;
30      repeat
31          u ← St.pop();
32          on_Stack[u] ← false;
33          add u to C;
34      until u = v
35      output(C);
```

but then u is pushed on `ready_St`$[\gamma]$ if and only if `on_Stack`$[\gamma]$ = `true` (i.e., there's no `active` array to make this decision anymore). Else, if $u \in N^{\text{in}}_{\mathcal{A}}(v)$ has been already visited (i.e., if `idx`$[u] \neq +\infty$), and if `on_Stack`$[u]$ = `true`, then `tr-lowlink`$[v] \leftarrow \min($`tr-lowlink`$[v]$, `idx`$[u])$ is updated (lines 20-21).

After that, anyway, `ready_St`$[v]$ is emptied and checked as in the DSF-based `tr-DFS-visit()` (see lines 22-27). Finally, if `tr-lowlink`$[v]$ = `idx`$[v]$ (see Proposition 9), a new STCC $\mathcal{C}$ is constructed: so, yet another vertex u is removed from `St` and added to $\mathcal{C}$, until $u = v$; soon after that, $\mathcal{C}$ is printed out.

This concludes the description of the STCCs algorithm (Algo. 2).

5.1 Proof of Correctness of `compute-STCCs()` (Algo. 2)

Firstly, we need to prove that Algo. 2 computes tr-lowlink() correctly.

Proposition 10. *Assume* `compute-STCCs`$(\mathcal{A})$ *(Algo. 2) is invoked.*
When it halts, it holds that $\forall_{v \in V}$ `tr-lowlink`$[v]$ = *tr-lowlink*(v).

Proof. Let us say that v is *active* from when `STCCs-visit`$(v, \mathcal{A})$ is invoked 'til it halts; and from when `STCCs-visit`$(v, \mathcal{A})$ finally halts hereafter, we say that v is *deactivated*. So, let $(v_1, \ldots, v_i, \ldots, v_{|V|})$ be the order in which the vertices in V are deactivated during `compute-STCCs`$(\mathcal{A})$ (Algo. 2).

For every $v \in V$, let's define:

$$N^{\text{in}}_{\mathcal{A},\texttt{St}}(v) \triangleq \big\{u \in N^{\text{in}}_{\mathcal{A}}(v) \mid u \in \texttt{St} \text{ at line 22 of } \texttt{STCCs-visit}(v, \mathcal{A}) \text{ (Proc. 2)}\big\}.$$

The proof proceeds by induction on $i = 1, \ldots, |V|$. Let F be the corresponding STCC forest.

Base Case: $i = 1$. Notice that v_1 is a leaf of some tr-palm-tree in F. In this case, $\texttt{tr-lowlink}[v_1]$ can be assigned only at line 21 of $\texttt{STCCs-visit}(v_1, \mathcal{A})$, particularly, as follows:

$$\texttt{tr-lowlink}[v_1] = \min\{\texttt{idx}[v_1]\} \cup \{\texttt{idx}[u] \mid u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1)\}. \tag{eq. 1}$$

Since v_1 is the first vertex in V which is ever deactivated (particularly, v_1 is a leaf in F), then:

$$N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1) = \big\{u \in N^{\text{in}}_{\mathcal{A}}(v_1) \mid u \text{ is a proper ancestor of } v_1 \text{ in } F\big\}, \tag{eq. 2}$$

for the same reason, plus Item 1 of Proposition 8, it holds that:

$$\text{tr-lowlink}(v_1) = \min\big\{\texttt{idx}[v_1]\} \uplus \{\texttt{idx}[u] \mid u \in N^{\text{in}}_{\mathcal{A}}(v_1) \cap \mathcal{C}_{v_1} \cap V_{\square}\big\}. \tag{eq. 3}$$

Observe, since v_1 is a leaf of F, and by (eq. 2) plus lines 8-11 of $\texttt{STCCs-visit}(v_1, \mathcal{A})$ (Proc. 2), it holds that $N^{\text{in}}_{\mathcal{A}}(v_1) \cap \mathcal{C}_{v_1} \cap V_{\square} \subseteq N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1)$; also, by (eq. 2) and Proposition 4, $N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1) \subseteq N^{\text{in}}_{\mathcal{A}}(v_1) \cap \mathcal{C}_{v_1} \cap V_{\square}$. Then, $N^{\text{in}}_{\mathcal{A}}(v_1) \cap \mathcal{C}_{v_1} \cap V_{\square} = N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_1)$. Therefore, by (eq. 1) and (eq. 3), $\texttt{tr-lowlink}[v_1] = \text{tr-lowlink}(v_1)$.

Inductive Step: $i > 1$. In this case, $\texttt{tr-lowlink}[v_i]$ can be assigned either at line 2 or 10 or 21 or 26 of $\texttt{STCCs-visit}(v_i, \mathcal{A})$, particularly, as follows:

$$\texttt{tr-lowlink}[v_i] = \min\big\{\texttt{idx}[v_1]\big\} \uplus \big\{\texttt{idx}[u] \mid u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v)\big\} \uplus \big\{\texttt{tr-lowlink}[u] \mid u \text{ is a child of } v \text{ in } F\big\}.$$

On the other side, let $N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i) \triangleq \{u \in N^{\text{in}}_{\mathcal{A}}(v_i) \mid \text{the LCA } \gamma \text{ of } \{u, v_i\} \text{ in } F \text{ exists and } \gamma \in \mathcal{C}_u\}$, then the following holds by Item 2 of Proposition 8:

$$\begin{aligned}\text{tr-lowlink}(v_i) = \min\big\{\texttt{idx}[v_i]\big\} &\uplus \big\{\texttt{idx}[u] \mid u \in N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)\big\} \\ &\uplus \big\{\text{tr-lowlink}(u) \mid u \text{ is a child of } v_i \text{ in } F\big\}.\end{aligned}$$

Notice that, if u is a child of v_i in F, then $\texttt{compute-STCCs}(\mathcal{A})$ deactivates u before v_i. Thus, by induction hypothesis, $\texttt{tr-lowlink}[u] = \text{tr-lowlink}(u)$ holds for every child u of v_i in F that is considered either at line 10 or 26 of $\texttt{STCCs-visit}(v_i, \mathcal{A})$. To finish the proof, we need to show that $N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i) = N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$. For this, let's recall the following facts. (i) Any vertex $v \in V$ can be added to $\texttt{St}$ only at line 4 of $\texttt{STCCs-visit}(v, \mathcal{A})$. (ii) Any vertex can be removed from $\texttt{St}$ only at line 31 of $\texttt{STCCs-visit}(v, \mathcal{A})$, for some $v \in V$, and only if $\texttt{tr-lowlink}[v] = \texttt{idx}[v]$ at line 28; this (possibly) happens only after that $N^{\text{in}}_{\mathcal{A}}(v)$ has been fully explored at lines 6-21. With this in mind, we can proceed.

- Firstly, we show $N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i) \subseteq N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$. Let $u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i)$. Then, u and v_i lie within the same tr-palm-tree in F: this is easily seen by induction on the number of tr-palm-trees of F. Then u is a proper ancestor of v_i in F, i.e., $\gamma = u$; thus, $\gamma \in \mathcal{C}_u$. So, the LCA γ of $\{u, v_i\}$ in F exists if u is still active at line 22 of $\texttt{STCCs-visit}(v_i, \mathcal{A})$. Otherwise, u has already been deactivated when $\texttt{STCCs-visit}(v_i, \mathcal{A})$ reaches line 22 (so, $\gamma \neq u$); in this case, also every ancestor of u that is a proper descendant of γ in F has already been deactivated before. So, by induction hypothesis, $\texttt{tr-lowlink}[\hat{v}] = \text{tr-lowlink}(\hat{v})$ for every ancestor $\hat{v}$ of u that is also proper descendant of γ in F. On the other hand, since $u \in \texttt{St}$ at line 22 of $\texttt{STCCs-visit}(v_i, \mathcal{A})$, all those $\hat{v}$ (including u) can't be already been removed from $\texttt{St}$ when $\texttt{STCCs-visit}(v_i, \mathcal{A})$ reaches line 22. Therefore, by lines 28-33 of $\texttt{STCCs-visit}()$, by Proposition 9, and since $\texttt{tr-lowlink}[\hat{v}] = \text{tr-lowlink}(\hat{v})$ for all of those $\hat{v}$, then none of those $\hat{v}$ can be the root of some STCC of $\mathcal{A}$. Thus, $\gamma \in \mathcal{C}_u$. So, $u \in N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$.

- Secondly, we show $N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i) \subseteq N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i)$. Let $u \in N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$, and let $\gamma \in \mathcal{C}_u$ be the LCA of $\{u, v_i\}$ in F. If u is still active at line 22 of $\texttt{STCCs-visit}(v_i, \mathcal{A})$, u is a proper ancestor of v_i in F (i.e., $\gamma = u$); so, $u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i)$. Otherwise, u has already been deactivated when $\texttt{STCCs-visit}(v_i, \mathcal{A})$ reaches line 22 (so, $\gamma \neq u$); also, in this case, every ancestor of u that is also a proper descendant of γ in F has already been deactivated before. So, by induction hypothesis, $\texttt{tr-lowlink}[\hat{v}] = \text{tr-lowlink}(\hat{v})$ for every ancestor $\hat{v}$ of u that is also proper descendant of γ in F; but, then, since $\gamma \in \mathcal{C}_u$, all those $\hat{v}$ (including u) can't be already been removed from $\texttt{St}$ when $\texttt{STCCs-visit}(v_i, \mathcal{A})$ reaches line 22. Therefore, $u \in N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i)$.

So, $N^{\text{in}}_{\mathcal{A},\texttt{St}}(v_i) = N^{\text{in}}_{\mathcal{A},\text{LCA}}(v_i)$. □

Proposition 11. *Assume that* `compute-STCCs`$(\mathcal{A})$ *(Algo. 2) is invoked, and that* `STCCs-visit`$(v, \mathcal{A})$ *(Proc. 2) outputs $\mathcal{C} \subseteq V$ at line 35, for some $v \in V$. Then, $\mathcal{C} = \mathcal{C}_v$.*

Proof. Notice that `STCCs-visit`$(v, \mathcal{A})$ outputs $\mathcal{C}$ at line 35 iff `tr-lowlink`$[v]$ = `idx`$[v]$ holds at line 28; assume such a condition holds. Recall `tr-lowlink`$[u]$ = tr-lowlink(u) for every $u \in V$, by Proposition 10. We argue that $\mathcal{C} = \mathcal{C}_v$. Indeed, $\mathcal{C} = \texttt{St}(v)$ by lines 28-33 of `STCCs-visit`$(v, \mathcal{A})$ (Proc. 2). Firstly, $\mathcal{C}_v \subseteq \texttt{St}(v)$: in fact, by Proposition 7, $\mathcal{C}_v$ induces a subtree $T_{\mathcal{C}_v}$ in F (i.e., the STCC forest) which is rooted at v, so all vertices in $\mathcal{C}_v$ must have been inserted into `St` at this point; and notice no vertex of $\mathcal{C}_v$ could have been removed earlier, as removals happen only at the root v of $T_{\mathcal{C}_v}$ (by Proposition 9 and lines 28-33 of `STCCs-visit()`). Secondly, $\texttt{St}(v) \subseteq \mathcal{C}_v$: indeed, assume $u \notin \mathcal{C}_v$ has been inserted into `St` after v, then u is a descendant of v in F, and since $\mathcal{C}_u$ induces a subtree $T_{\mathcal{C}_u}$ in F, then u must have been removed from `St` when the root of $T_{\mathcal{C}_u}$ was visited, i.e., before that `STCCs-visit`$(v, \mathcal{A})$ reaches line 28. Therefore, $\texttt{St}(v) = \mathcal{C}_v$; and since $\mathcal{C} = \texttt{St}(v)$, then $\mathcal{C} = \mathcal{C}_v$. □

Proposition 12. *Let $\mathcal{C} \subseteq V$ be some STCC of $\mathcal{A}$. Then,* `compute-STCCs`$(\mathcal{A})$ *(Algo. 2) eventually outputs $\mathcal{C}$ at line 35.*

Proof. By Proposition 7, $\mathcal{C}$ induces a sub-tree $T_{\mathcal{C}}$ in F; then, let v^* be its root. When `STCCs-visit`$(v^*, \mathcal{A})$ is invoked, `tr-lowlink`$[v^*]$ = `tr-lowlink`(v^*) = `idx`$[v^*]$ holds at line 28 by Propositions 9 and 10.

Then, $\mathcal{C}_{v^*}$ is outputted at line 35, by lines 28-33 and Proposition 12. □

In summary, we obtain the following result.

Theorem 2. *Let $\mathcal{A}$ be an arena,* `compute-STCCs`$(\mathcal{A})$ *(Algo. 2) outputs all and only the STCCs of $\mathcal{A}$.*

6 Application to Update Games

An *Update Game (UG)* [1–3] is played on an arena $\mathcal{A}$ for an infinite number of rounds, a *play* is thus an infinite path $p = v_0 v_1 v_2 \ldots \in V^\omega$ such that $(v_i, v_{i+1}) \in A$ for every $i \in \mathbf{N}$. Let $\text{Inf}(\pi)$ be the set of all and only those vertices $v \in V$ that appear infinitely often in π; namely,

$$\text{Inf}(\pi) \triangleq \big\{v \in V \mid \forall_{j \in \mathbf{N}}\ \exists_{k \in \mathbf{N}} \text{ such that } k > j \text{ and } v_k = v\big\}.$$

Player □ wins the UG $\mathcal{A}$ iff there exists $\sigma_\square \in \Sigma^{\mathcal{A}}_\square$ such that, for every $\sigma_\bigcirc \in \Sigma^{\mathcal{A}}_\bigcirc$, every vertex is visited infinitely often in the play consistent with $\sigma_\square$ and $\sigma_\bigcirc$, independently w.r.t. the starting position $v_s \in V$; namely,

$$\forall_{v_s \in V}\ \text{Inf}\big(\rho_{\mathcal{A}}(v_s, \sigma_\square, \sigma_\bigcirc)\big) = V;$$

otherwise, Player ○ wins. When Player □ wins an UG $\mathcal{A}$, we say that $\mathcal{A}$ is an *Update Network (UN)* [1–3].

Proposition 13. *An UG $\mathcal{A}$ is UN if and only if V is strongly-trap-connected.*

Proof. If $\mathcal{A}$ is UN, then V is strongly-trap-connected (it follows directly from definitions). Let $i' \triangleq (i+1) \bmod |V|$ for every $i \in \{0, \ldots, |V|-1\}$. Conversely, if $V = \{v_i\}_{i=0}^{|V|-1}$ is strongly-trap-connected, for every i there exists $\sigma_\square(i) \in \Sigma^M_\square$ such that $\sigma_\square(i) : v_i \rightsquigarrow v_{i'}$. Starting from any v_i, Player □ can visit infinitely often all vertices in V simply by playing $\sigma_\square(i), \sigma_\square(i'), \sigma_\square((i')'), \ldots$ in cascade; therefore, $\mathcal{A}$ is UN. □

So, we obtain the following main result.

Theorem 3. *Deciding whether or not a given UG $\mathcal{A}$ is UN takes $\Theta(|V| + |A|)$ time.*

Proof. On input $\mathcal{A}$, invoke `compute-STCCs`$(\mathcal{A})$ (Algorithm 2), and return `YES` if $\mathcal{A}$ has *only one* STCC; otherwise, $\mathcal{A}$ has at least two STCCs, so return `NO`. By Theorem 2 and Proposition 13, this correctly decides whether or not $\mathcal{A}$ is UN. By Proposition 6, the decision is made in $\Theta(|V| + |A|)$ time. □

Also, when the input UG is UN, Algorithm 2 is able to provide a winning strategy as shown next.

Theorem 4. *Algorithm 2 can be implemented so that, when* `compute-STCCs`$(\mathcal{A})$ *halts, and if the UG* $\mathcal{A}$ *is UN, it is returned a tr-palm-tree encoding routing information that an* $O(|V|)$*-space agent can consult to win the UG* $\mathcal{A}$ *in* $O(1)$ *time per move.*

Proof. During the execution of `compute-STCCs`$(\mathcal{A})$ (Algorithm 2), construct the STCC forest $F = (V_F, A_F)$ explicitly, as follows: $V_F = V$; whenever $\mathcal{D}.\texttt{Union}(u, v)$ is executed at line 11 or line 27 of `STCCs-visit`$(v, \mathcal{A})$, add (u, v) to A_F (*tree-arcs*); also, if `on_Stack`$[u]$ = `true` holds at line 20 of `STCCs-visit`$(v, \mathcal{A})$, add (u, v) to A_F (*cross-links*). Define $\sigma_\square \in \Sigma_\square$ as follows: for each $u \in V_\square$, the arcs exiting from u are selected one at a time, one after the other; and when they have all been traversed once, the selection starts again, cyclically. Since $\mathcal{A}$ is UN, then $\mathcal{A}$ has only one single STCC by Proposition 13, so F comprises only one single tr-palm-tree T_F. We argue that, if $\mathcal{A}$ is UN, then $\sigma_\square$ allows Player $\square$ to win the UG $\mathcal{A}$. Let v_s be any starting position. For any $\sigma_\bigcirc \in \Sigma_\bigcirc$ and $I \triangleq \mathrm{Inf}\big(\rho_{\mathcal{A}}(v_s, \sigma_\square, \sigma_\bigcirc)\big)$, it is not possible that $I \subsetneq V$: there can be no tree-arc nor cross-link going from some vertex $u \in I \cap V_\square$ to some vertex in $V \setminus I$ (otherwise such an arc would have eventually been selected by $\sigma_\square$); and there can be no $u \in I \cap V_\bigcirc$ such that $N^{\mathrm{out}}_{\mathcal{A}}(u) \subseteq V \setminus I$. Thus, all vertices in $V \setminus I$ are descendants in T_F of some of those in I; but, since they are all descendants, there must be at least one cross-link going from I to in $V \setminus I$ (because $\mathcal{A}$ has only one single STCC), which is a contradiction. Therefore, $I = V$. Notice the size of T_F is $|T_F| = |V_{T_F}| + |A_{T_F}| = O(|V|)$, and $\sigma_\square$ can be implemented with $O(|V|)$ additional memory (the total number of cross-links in T_F is less than $|V|$); so, $\sigma_\square$ can be implemented with $O(|V|)$ space. By handling pointers in a suitable way, the time spent for each single move of $\sigma_\square$ is $O(1)$. $\square$

7 Application to Explicit McNaughton-Müller Games

McNaughton-Müller Games (MGs) provide a useful model for the synthesis of controllers in reactive systems, but their complexity depends on the representation of the winning conditions [5]. The most straightforward way to represent a (regular; see [5]) winning condition $\mathcal{F} \subseteq 2^V$ is to provide an explicit list of subsets of vertices, i.e., $\mathcal{F} = \{\mathcal{F}_i \subseteq V \mid 1 \leq i \leq \ell\}$, for some $\ell \in \mathbf{N}$. A play $\rho \in V^\omega$ is winning for Player $\square$ if and only if $\mathrm{Inf}(\rho) \in \mathcal{F}$. So, *Explicit MGs (E-MGs)* can be solved in polynomial time, where an effective algorithm is given in [5]. Concerning time complexity, given an input arena $\mathcal{A}$ and explicit winning condition $\mathcal{F}$, there are at most $|\mathcal{F}|$ loops in a run of that algorithm, and the most time consuming operation at each iteration is to decide an UG of size at most $|\mathcal{A}| + |\mathcal{F}|$. By Theorem 3, we can decide such an UG in $\Theta(|\mathcal{A}| + |\mathcal{F}|)$ time. So the E-MG algorithm given in [5] is improved by a factor $|\mathcal{A}| + |\mathcal{F}|$ (i.e., from cubic to quadratic time). In summary, we obtain the following result.

Theorem 5. *Deciding the winner in a given E-MG* $(\mathcal{A}, \mathcal{F})$ *takes* $O\big(|\mathcal{F}| \cdot (|\mathcal{A}| + |\mathcal{F}|)\big)$ *time.*

8 Conclusion

This work presented an algorithm for solving Update Games in linear time. With this, also the polynomial-time complexity of deciding Explicit McNaughton-Müller Games improves, from cubic to quadratic. The result was obtained by: (a) introducing a refined notion of reachability for arenas, named *trap-reachability*; (b) showing that every arena decomposes into *strongly-trap-connected components (STCCs)*; (c) devising a linear time algorithm for computing this unique decomposition.

We expect that trap-reachability, and the corresponding linear time STCCs' decomposition, can play a role for speeding up computations in other problems concerning infinite 2-player pebble games.

Future works will likely investigate further on this way.

Acknowledgments This work was supported by *Department of Computer Science, University of Verona, Verona, Italy*, under PhD grant "Computational Mathematics and Biology", on a co-tutelle agreement with *LIGM, Université Paris-Est in Marne-la-Vallée, Paris, France.*

References

[1] Bodlaender, H. L., Dinneen, M. J., and Khoussainov, B. *Algorithms and Computation: 12th International Symposium, ISAAC 2001 Christchurch, New Zealand, December 19–21, 2001 Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001, ch. On Game-Theoretic Models of Networks, pp. 550–561.

[2] Dinneen, M. J., and Khoussainov, B. Update games and update networks. In *Proceedings of the 10th Australasian Workshop on Combinatorial Algorithms* (1999), AWOCA '99, pp. 7–18.

[3] Dinneen, M. J., and Khoussainov, B. *Graph-Theoretic Concepts in Computer Science: 26th International Workshop, WG 2000 Konstanz, Germany, June 15–17, 2000 Proceedings*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000, ch. Update Networks and Their Routing Strategies, pp. 127–136.

[4] Grädel, E., Thomas, W., and Wilke, T., Eds. *Automata Logics, and Infinite Games: A Guide to Current Research*. Springer-Verlag New York, Inc., New York, NY, USA, 2002.

[5] Horn, F. Explicit muller games are PTIME. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2008, December 9-11, 2008, Bangalore, India* (2008), pp. 235–243.

[6] Loebl, M., and Nešetřil, J. Linearity and unprovability of set union problem strategies. *Journal of Algorithms 23*, 2 (1997), 207 – 220.

[7] Tarjan, R. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing 1*, 2 (1972), 146–160.