Linear Time Algorithm for Update Games
DIMAP Seminar
University of Warwick, UK
25 April 2017

Carlo Comin
On a joint ongoing work with Romeo Rizzi

University of Verona, Italy
An arena $\Gamma = (V, A, \langle V_\square, V_\bigcirc \rangle)$ is a finite directed graph $G^\Gamma = (V, A)$ whose vertices are divided into two classes, i.e., $V = V_\square \cup V_\bigcirc$.

$G^\Gamma = (V, A)$:
- has no sink vertices;
- has no loops nor parallel arcs;
- is not required to be a bipartite graph on colour classes $V_\square$ and $V_\bigcirc$. 
Infinite Pebble Games

Game

A game on $\Gamma = (V, A, \langle V\square, V\bigcirc \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\square, \bigcirc\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
- repeat rounds ad infinitum ...
A *game* on $\Gamma = (V, A, \langle V^{\square}, V^{\bigcirc} \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\square, \bigcirc\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
- repeat rounds ad infinitum ...
A game on $\Gamma = (V, A, \langle V\Box, V\circ \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \circ\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
A game on $\Gamma = (V, A, \langle V_\square, V_\bigcirc \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\square, \bigcirc\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
- repeat rounds ad infinitum...
A game on $\Gamma = (V, A, \langle V_{\square}, V_{\circ} \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\square, \circ\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
- repeat rounds ad infinitum...
A game on $\Gamma = (V, A, \langle V\Box, V\Diamond \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \Diamond\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
- repeat rounds ad infinitum ...
A game on $\Gamma = (V, A, \langle V_\square, V_\bigcirc \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\square, \bigcirc\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
- repeat rounds ad infinitum ...
A *game* on $\Gamma = (V, A, \langle V_{\square}, V_{\bigcirc} \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\square, \bigcirc\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
- repeat rounds ad infinitum ...
A game on $\Gamma = (V, A, \langle V_\Box, V_\bigcirc \rangle)$ 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 $v_s \in V$; say, $v_s = a$;
- At each round, if the pebble is currently on $v \in V_p$, for some $p \in \{\Box, \bigcirc\}$, Player $p$ chooses an arc $(v, v') \in A$; say, $(v, v') = (a, b)$;
- and then the next round starts with the pebble on $v'$;
- repeat rounds ad infinitum ... $\text{(abac)}^+ \text{ ...}$
Infinite Pebble Games

Play, Strategy, Outcome Play

- A **play** is an infinite path \( \pi = v_0v_1v_2 \ldots \in V^\omega \) such that \((v_i, v_{i+1}) \in A\);
- A **strategy** for Player \(p\), where \(p \in \{\Box, \bigcirc\}\), is a map,

  \[
  \sigma_p : V^* \times V_p \rightarrow V, \text{ such that } (v, \sigma_p(\pi', v)) \in A,
  \]

  for every finite path \(\pi'v\) in \(G^\Gamma\) where \(v \in V_i\);
- Given two strategies \(\sigma_{\Box} \in \Sigma^\Gamma_{\Box}\) and \(\sigma_{\bigcirc} \in \Sigma^\Gamma_{\bigcirc}\), and some \(v_s \in V\), the **outcome** play,

  \[
  \rho^\Gamma(v_s, \sigma_{\Box}, \sigma_{\bigcirc}),
  \]

  is the (unique) play that starts at \(v_s\) and is consistent with both \(\sigma_{\Box}, \sigma_{\bigcirc}\).
Winning Condition

- Let $\text{Inf}(\pi)$ be the set of all and only those vertices $v \in V$ that appear infinitely often in the play $\pi$,

$$\text{Inf}(\pi) \triangleq \{ v \in V \mid \forall j \in \mathbb{N} \exists k \in \mathbb{N}, k > j \text{ such that } v_k = v \}.$$ 

- Player $\square$ wins the Update Game $\Gamma$ iff all vertices are visited infinitely often,

$$\exists \sigma_\square \in \Sigma_\square \forall \sigma_\circ \in \Sigma_\circ \forall v_s \in V \text{ Inf}(\rho_\Gamma(v_s, \sigma_\square, \sigma_\circ)) = V.$$
**Main facts**

- Firstly studied by:
  - [Dinneen, Khoussainov, Update Networks and Their Routing Strategies. WG 2000];
- \(O(mn)\) time algorithm [Dinneen, Khoussainov. WG 2000];
- Routing strategies are not positional.
- Basic subtask for solving Explicit Müller Games in polynomial time, as in:
  - [Horn, Explicit Müller Games are PTIME. FSTTCS 2008];
Update Games

□-Attractor

\[ \text{Reach}^\square(v, 0) \triangleq \{v\}, \forall v \in V; \]
\[ \text{Reach}^\square(v, i) \triangleq \{u \in V\big| \exists (u, u') \in A \ u' \in \bigcup_{j=0}^{i-1} \text{Reach}^\square(v, j)\} \cup \]
\[ \bigcup \{u \in V_\big| \forall (u, u') \in A \ u' \in \bigcup_{j=0}^{i-1} \text{Reach}^\square(v, j)\}, \forall v \in V \forall i > 0. \]
\[ \text{Attr}^\square(v) \triangleq \bigcup_{i \geq 0} \text{Reach}^\square(v, i). \]

\(O(mn)\) Time Algorithm [Dinneen, Khoussainov. WG 2000]

- Compute \(\text{Attr}^\square(v)\) for each \(v \in V\);
- If \(\exists v \in V \text{Attr}^\square(v) \neq V\), return NO;
- Otherwise, return YES.
Linear Time: what can be done?

**Note**

If $G^\Gamma$ is not *strongly-connected* as a graph, then return **NO**.

**Alternating Strongly Connected Components**

Say $u, v \in V$ are \(a\)-strongly-connected if $u \in \text{Attr}^\Gamma_{\square}(v) \land v \in \text{Attr}^\Gamma_{\square}(u)$.

\[
\text{Attr}^\Gamma_{\square}(a) = \{a, b, c\}, \text{Attr}^\Gamma_{\square}(b) = \{b\}, \text{Attr}^\Gamma_{\square}(c) = \{a, c\}.
\]
Linear Time: what can be done?

**Note**

If $G^\Gamma$ is not *strongly-connected* as a graph, then return *NO*.

**Alternating Strongly Connected Components**

Say $u, v \in V$ are *a-strongly-connected* if $u \in \text{Attr}^\Gamma_{\square}(v) \land v \in \text{Attr}^\Gamma_{\square}(u)$.

**Example**

\[
\begin{align*}
\text{Attr}^\Gamma_{\square}(a) &= \{a, b, c\}, \\
\text{Attr}^\Gamma_{\square}(b) &= \{b\}, \\
\text{Attr}^\Gamma_{\square}(c) &= \{a, c\}.
\end{align*}
\]
Linear Time: what can be done?

Note
If $G^\Gamma$ is not strongly-connected as a graph, then return NO.

Alternating Strongly Connected Components
Say $u, v \in V$ are a-strongly-connected if $u \in \text{Attr}^\Gamma(v) \land v \in \text{Attr}^\Gamma(u)$.

$D_{a-scC} = \left\{ \{a, c\}, \{b\} \right\}$. 
Linear Time: what can be done?

**Note**

If $G^\Gamma$ is not strongly-connected as a graph, then return NO.

**Alternating Strongly Connected Components**

Say $u, v \in V$ are $a$-strongly-connected if $u \in \text{Attr}^\Gamma(v) \land v \in \text{Attr}^\Gamma(u)$.

**Deciding Update Games (Sketch)**

If $\Gamma$ is $a$-strongly-connected, i.e., $D_{a\text{-scc}} = \{V\}$, return YES; otherwise, NO.
Figure: An arena $\Gamma$ (a), and a rev-palm-tree (b), generated by rev-DFS on $G^\Gamma$ (c).
If we run a DFS on $G^\Gamma$, then ...

- Arcs can be partitioned into: tree, fronds, cross (and forward).
- If $G^\Gamma$ is strongly-connected, the rev-palm-tree is a spanning tree.
- Each vertex is reachable in $G^\Gamma$ from any of its descendants,
  - however, e.g. $\text{Attr}^\Gamma(B) = \{B\}$; thus, rev-palm-trees are not (yet) certificates of a-reachability.
Safe-Reachability

Given an arena $\Gamma$ on vertex set $V$, let $U \subseteq V$ and $u, v \in U$. We say that $v$ is $U$-safe-reachable from $u$ (i.e. $u \xrightarrow{U} v$) when $\exists \sigma_\Box \in \Sigma_\Box^\Gamma$ such that $\forall \sigma_\bigcirc \in \Sigma_\bigcirc^\Gamma$:

- [a-reachability] $v \in \Occ(\rho^\Gamma(u, \sigma_\Box, \sigma_\bigcirc))$; and,
- [safety] $\Occ(\rho^\Gamma(u, \sigma_\Box, \sigma_\bigcirc, v)) \subseteq U$.

where for any finite (or infinite) path $p \in V^*$ (or $p \in V^\omega$), the occurrence set of $p$ is:

$$\Occ(p) \triangleq \{ v \in V \mid v \text{ appears in } p \};$$

and $\rho^\Gamma(u, \sigma_\Box, \sigma_\bigcirc, v)$ is the shortest prefix of $\rho^\Gamma(u, \sigma_\Box, \sigma_\bigcirc)$ ending with $v$.

Alternating Depth-First-Search (a-DFS)

Perform a DFS on the arena $\Gamma$ in such a way as to always preserve $V_T$-safe-reachability within the palm-tree $T$ that is under construction. (Invariant Property)

- i.e., $v \in V_\bigcirc$ joins the palm-tree $T$ as soon as all of its outgoing neighbours $v' \in N^\text{out}_v$ have already did it; and its parent $\pi_v$ is the LCA of $N^\text{out}_v$ in $T$. 
Given an arena $\Gamma$ on vertex set $V$, let $U \subseteq V$ and $u, v \in U$. We say that $v$ is $U$-safe-reachable from $u$ (i.e. $u \overset{U}{\sim} v$) when $\exists \sigma_\square \in \Sigma_\square^{\Gamma}$ such that $\forall \sigma_\bigcirc \in \Sigma_\bigcirc^{\Gamma}$:

**[a-reachability]** $v \in \overline{\text{Occ}}(\rho_\Gamma(u, \sigma_\square, \sigma_\bigcirc))$; and,

**[safety]** $\overline{\text{Occ}}(\rho_\Gamma(u, \sigma_\square, \sigma_\bigcirc, v)) \subseteq U$.

where for any finite (or infinite) path $p \in V^*$ (or $p \in V^\omega$), the occurrence set of $p$ is:

$$\overline{\text{Occ}}(p) \triangleq \{v \in V \mid v \text{ appears in } p\};$$

and $\rho_\Gamma(u, \sigma_\square, \sigma_\bigcirc, v)$ is the shortest prefix of $\rho_\Gamma(u, \sigma_\square, \sigma_\bigcirc)$ ending with $v$.

**Alternating Depth-First-Search (a-DFS)**

Perform a DFS on the arena $\Gamma$ in such a way as to always preserve $V_T$-safe-reachability within the palm-tree $T$ that is under construction. (Invariant Property)

- *i.e.,* $v \in V_\bigcirc$ joins the palm-tree $T$ as soon as all of its outgoing neighbours $v' \in N_v^{\text{out}}$ have already did it; and its parent $\pi_v$ is the LCA of $N_v^{\text{out}}$ in $T$. 

Linear Time Algorithm for Update Games

Carlo Comin
Alternating Depth-First-Search: an example

For every \( v \in V \), \( \text{idx}[v] \in \mathbb{N} \), \( \text{ready}_\text{St}[v] \subseteq V_\circ \); and, for every \( u \in V_\circ \), \( c[u] \in \mathbb{N} \).

Figure: An arena (a), and an a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every \( v \in V \), \( \mathbf{idx}[v] \in \mathbb{N}, \mathbf{ready}_\text{St}[v] \subseteq V_\circ \); and, for every \( u \in V_\circ \), \( c[u] \in \mathbb{N} \).

(a) An arena \( \Gamma \).

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

(c) The order of arcs’ exploration.

Figure: An arena (a), and an a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_\text{St}[v] \subseteq V_\circ$; and, for every $u \in V_\circ$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

(c) The order of arcs’ exploration.

Figure: An arena (a), and a $a$-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{id}x[v] \in \mathbb{N}$, $\text{ready}_\text{St}[v] \subseteq V_\circ$; and, for every $u \in V_\circ$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every \( v \in V \), \( \text{idx}[v] \in \mathbb{N} \), \( \text{ready}_\text{St}[v] \subseteq V_\circ \); and, for every \( u \in V_\circ \), \( c[u] \in \mathbb{N} \).

(a) An arena \( \Gamma \).

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

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_St[v] \subseteq V_\circ$; and, for every $u \in V_\circ$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

(c) The order of arcs’ exploration.

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

Linear Time Algorithm for Update Games
Carlo Comin
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_St[v] \subseteq V_\circ$; and, for every $u \in V_\circ$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}, \text{ready}_\text{St}[v] \subseteq V_\triangledown$; and, for every $u \in V_\triangledown$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

1. $(B, A)$ 7. $(A, G')$
2. $(D, B)$ 8. $(F, G')$
3. $(E, D)$ 9. $(C, B)$
4. $(C, E)$ 10. $(H, A)$
5. $(F, E)$ 11. $(C, H)$
6. $(G, D)$ 12. $(F, H)$

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_St[v] \subseteq V_\circ$; and, for every $u \in V_\circ$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

1. $(B, A)$ 7. $(A, G')$
2. $(D, B)$ 8. $(F, G')$
3. $(E, D)$ 9. $(C, B)$
4. $(C, E)$ 10. $(H, A)$
5. $(F, E)$ 11. $(C, H)$
6. $(G, D)$ 12. $(F, H)$

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_St[v] \subseteq V_\bigotimes$; and, for every $u \in V_\bigotimes$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_\text{St}[v] \subseteq V_{\circ}$; and, for every $u \in V_{\circ}$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

(c) The order of arcs' exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_\text{St}[v] \subseteq V_\circ$; and, for every $u \in V_\circ$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$. (b) The a-palm-tree generated by a-DFS rooted at $A$, with indices of vertices and labelled arcs. (c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_\text{St}[v] \subseteq V_\bigcirc$; and, for every $u \in V_\bigcirc$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_St[v] \subseteq V_\odot$; and, for every $u \in V_\odot$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

(c) The order of arcs' exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every \( v \in V \), \( \text{idx}[v] \in \mathbb{N} \), \( \text{ready}_St[v] \subseteq V_\bigcirc \); and, for every \( u \in V_\bigcirc \), \( c[u] \in \mathbb{N} \).

(a) An arena \( \Gamma \).

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

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Alternating Depth-First-Search: an example

For every $v \in V$, $\text{idx}[v] \in \mathbb{N}$, $\text{ready}_St[v] \subseteq V_{\bigcirc}$; and, for every $u \in V_{\bigcirc}$, $c[u] \in \mathbb{N}$.

(a) An arena $\Gamma$.

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

1. $(B, A)$
2. $(D, B)$
3. $(E, D)$
4. $(C, E)$
5. $(F, E)$
6. $(G, D)$
7. $(A, G)$
8. $(F, G)$
9. $(C, B)$
10. $(H, A)$
11. $(C, H)$
12. $(F, H)$

(c) The order of arcs’ exploration.

Figure: An arena (a), and a a-palm-tree (b), generated by a-DFS (c).
Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on $V$, i.e., init $\mathcal{D}.\text{MakeSet}(v) \ \forall v \in V$;
- For all $v \in V_\text{O}$, the following invariant is maintained during the a-DFS:

$$\text{low}_\text{ready}[v] = \min \{ \text{idx}[u] \mid u \in N^\text{out}_\text{A}(v) \}.$$  

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^\text{in}(v) \cup \text{ready}_\text{St}[v]$, soon after that call, execute $\mathcal{D}.\text{Union}(u, v)$.
- When the condition $c[u] = 0$ is met, for some $u \in V_\text{O}$:
  1. let $\text{low}_v$ be the unique $x \in N^\text{out}(u)$ with $\text{idx}[x] = \text{low}_\text{ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\text{Find}(\text{low}_v)$,
  3. if $\gamma$ is still active, execute $\text{ready}_\text{St}[\gamma].\text{push}(u)$.

**Proposition**

If $\gamma$ is still active, the LCA of $N^\text{out}(u)$ exists in $\mathcal{T}$ and it is really $\gamma$. 

Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $D$ on $V$, i.e., $\text{init } D.\text{MakeSet}(v) \forall v \in V$;
- For all $v \in V$, the following invariant is maintained during the a-DFS:
  
  $$\text{low\_ready}[v] = \min \{ \text{idx}[u] \mid u \in N_{A}^{\text{out}}(v) \}.$$  

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \text{ready\_St}[v]$, soon after that call, execute $D.\text{Union}(u, v)$.
- When the condition $c[u] = 0$ is met, for some $u \in V$:
  1. let $\text{low\_v}$ be the unique $x \in N_{A}^{\text{out}}(u)$ with $\text{idx}[x] = \text{low\_ready}[u]$,
  2. let $\gamma \leftarrow D.\text{Find}(\text{low\_v})$;
  3. if $\gamma$ is still active, execute $\text{ready\_St}[\gamma].\text{push}(u)$.

**Proposition**

If $\gamma$ is still active, the LCA of $N_{A}^{\text{out}}(u)$ exists in $T$ and it is really $\gamma$. 
Least Common Ancestors

To compute *Least Common Ancestors*: 

- We have a Disjoint Sets Forest $D$ on $V$, i.e., $\text{init } D.\text{MakeSet}(v) \forall v \in V$;
- For all $v \in V_\infty$, the following invariant is maintained during the $a$-DFS:

$$\text{low\_ready}[v] = \min \{\text{idx}[u] \mid u \in N^{\text{out}}_A(v)\}.$$ 

- Whenever the $a$-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \text{ready\_St}[v]$, soon after that call, execute $D.\text{Union}(u, v)$.

- When the condition $c[u] = 0$ is met, for some $u \in V_\infty$:
  1. let low_v be the unique $x \in N^{\text{out}}(u)$ with $\text{idx}[x] = \text{low\_ready}[u]$,
  2. let $\gamma \leftarrow D.\text{Find}(\text{low}_v)$;
  3. if $\gamma$ is still active, execute $\text{ready\_St}[\gamma].\text{push}(u)$.

**Proposition**

If $\gamma$ is still active, the LCA of $N^{\text{out}}(u)$ exists in $T$ and it is really $\gamma$. 
Least Common Ancestors

To compute *Least Common Ancestors*:  

- We have a Disjoint Sets Forest $\mathcal{D}$ on $V$, i.e., init $\mathcal{D}.\text{MakeSet}(v) \; \forall v \in V$;  
- For all $v \in V_\circ$, the following invariant is maintained during the a-DFS:

$$\text{low}_\text{ready}[v] = \min \{ \text{idx}[u] \mid u \in N_A^{\text{out}}(v) \}.$$  

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \text{ready}_\text{St}[v]$, soon after that call, execute $\mathcal{D}.\text{Union}(u, v)$.  
- When the condition $c[u] = 0$ is met, for some $u \in V_\circ$:

1. let $\text{low}_v$ be the unique $x \in N^{\text{out}}(u)$ with $\text{idx}[x] = \text{low}_\text{ready}[u]$,  
2. let $\gamma \leftarrow \mathcal{D}.\text{Find}(\text{low}_v)$;  
3. if $\gamma$ is still active, execute $\text{ready}_\text{St}[\gamma].\text{push}(u)$.

**Proposition**

If $\gamma$ is still active, the LCA of $N^{\text{out}}(u)$ exists in $T$ and it is really $\gamma$.  

Linear Time Algorithm for Update Games  

Carlo Comin  

27/41
Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $D$ on $V$, i.e., $\text{init } D.\text{MakeSet}(v) \ \forall v \in V$;
- For all $v \in V_\circ$, the following invariant is maintained during the a-DFS:

  \[ \text{low}_\text{ready}[v] = \min \{ \text{idx}[u] \mid u \in N^\text{out}_A(v) \} \]

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^\text{in}(v) \cup \text{ready}_\text{St}[v]$, soon after that call, execute $D.\text{Union}(u, v)$.
- When the condition $c[u] = 0$ is met, for some $u \in V_\circ$:
  1. let $\text{low}_u$ be the unique $x \in N^\text{out}(u)$ with $\text{idx}[x] = \text{low}_\text{ready}[u]$,
  2. let $\gamma \leftarrow D.\text{Find}(\text{low}_u)$;
  3. if $\gamma$ is still active, execute $\text{ready}_\text{St}[\gamma].\text{push}(u)$.

### Proposition

If $\gamma$ is still active, the LCA of $N^\text{out}(u)$ exists in $T$ and it is really $\gamma$. 
Least Common Ancestors

To compute **Least Common Ancestors**:  
- We have a Disjoint Sets Forest $D$ on $V$, i.e., \( \text{init } D.\text{MakeSet}(v) \ \forall v \in V \);  
- For all $v \in V_\circ$, the following invariant is maintained during the a-DFS:
  \[
  \text{low}_\text{ready}[v] = \min \{ \text{idx}[u] \mid u \in N_\text{out}(v) \}.
  \]
- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^\text{in}(v) \cup \text{ready}_\text{St}[v]$, soon after that call, execute $D.\text{Union}(u, v)$.
- When the condition $c[u] = 0$ is met, for some $u \in V_\circ$:
  1. let $\text{low}_u$ be the unique $x \in N_\text{out}(u)$ with $\text{idx}[x] = \text{low}_\text{ready}[u]$,
  2. let $\gamma \leftarrow D.\text{Find}(\text{low}_u)$;
  3. if $\gamma$ is still active, execute $\text{ready}_\text{St}[\gamma].\text{push}(u)$.

**Proposition**

If $\gamma$ is still active, the LCA of $N_\text{out}(u)$ exists in $T$ and it is really $\gamma$.  

Linear Time Algorithm for Update Games

Carlo Comin
Least Common Ancestors

To compute *Least Common Ancestors*:  
▶ We have a Disjoint Sets Forest $D$ on $V$, i.e., init $D.$MakeSet($v$) $\forall v \in V$;  
▶ For all $v \in V_\bigcirc$, the following invariant is maintained during the a-DFS:

$$\text{low\_ready}[v] = \min\{\text{idx}[u] \mid u \in N_{\text{out}}^A(v)\}.$$  

▶ Whenever the a-DFS makes a recursive call to visit some vertex $u \in N_{\text{in}}(v) \cup \text{ready\_St}[v]$, soon after that call, execute $D.$Union($u, v$).  
▶ When the condition $c[u] = 0$ is met, for some $u \in V_\bigcirc$:  
  1. let low$_v$ be the unique $x \in N_{\text{out}}^A(u)$ with $\text{idx}[x] = \text{low\_ready}[u]$,  
  2. let $\gamma \leftarrow D.$Find(low$_v$);  
  3. if $\gamma$ is still active, execute ready$_\text{St}[\gamma].\text{push}(u)$.  

**Proposition**

If $\gamma$ is still active, the LCA of $N_{\text{out}}^A(u)$ exists in $T$ and it is really $\gamma$. 

Linear Time Algorithm for Update Games  
Carlo Comin
To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on $V$, i.e., $\text{init } \mathcal{D}.\text{MakeSet}(v) \quad \forall v \in V$;
- For all $v \in V_{\varnothing}$, the following invariant is maintained during the a-DFS:

$$\text{low}_\text{ready}[v] = \min \{ \text{idx}[u] \mid u \in N_{A}^{\text{out}}(v) \}.$$  

- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N_{\text{in}}(v) \cup \text{ready}_\text{St}[v]$, soon after that call, execute $\mathcal{D}.\text{Union}(u, v)$.
- When the condition $c[u] = 0$ is met, for some $u \in V_{\varnothing}$:
  1. let $\text{low}_v$ be the unique $x \in N_{\text{out}}(u)$ with $\text{idx}[x] = \text{low}_\text{ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\text{Find}(\text{low}_v)$;
  3. if $\gamma$ is still active, execute $\text{ready}_\text{St}[\gamma].\text{push}(u)$.

**Proposition**

If $\gamma$ is still active, the LCA of $N_{\text{out}}(u)$ exists in $\mathcal{T}$ and it is really $\gamma$. 

**Linear Time Algorithm for Update Games**

Carlo Comin 27/41
Least Common Ancestors

To compute *Least Common Ancestors*:

- We have a Disjoint Sets Forest $\mathcal{D}$ on $V$, i.e., init $\mathcal{D}.\text{MakeSet}(v) \ \forall v \in V$;
- For all $v \in V_{\circ}$, the following invariant is maintained during the a-DFS:
  \[
  \text{low\_ready}[v] = \min \{ \text{idx}[u] \mid u \in N_{A}^{\text{out}}(v) \}.
  \]
- Whenever the a-DFS makes a recursive call to visit some vertex $u \in N^{\text{in}}(v) \cup \text{ready\_St}[v]$, soon after that call, execute $\mathcal{D}.\text{Union}(u, v)$.
- When the condition $c[u] = 0$ is met, for some $u \in V_{\circ}$:
  1. let $\text{low\_v}$ be the unique $x \in N_{\text{out}}^{\text{out}}(u)$ with $\text{idx}[x] = \text{low\_ready}[u]$,
  2. let $\gamma \leftarrow \mathcal{D}.\text{Find}(\text{low\_v})$;
  3. if $\gamma$ is still active, execute $\text{ready\_St}[\gamma].\text{push}(u)$.

**Proposition**

If $\gamma$ is still active, the LCA of $N_{\text{out}}^{\text{out}}(u)$ exists in $T$ and it is really $\gamma$. 
Strong Post-Order Path-Compression Systems

Theorem (Loebl, Nesetril, 1997)

Let $S$ be a strong post-order path compression system, then $|S| \leq 5n$.


Proposition

The sequence of path compressions performed by the $a$-DFS is a Loebl-Nesetril strong post-order path compression system.

- Then the $a$-DFS halts in linear time.
Theorem (Loebl, Nesetril, 1997)
Let $S$ be a strong post-order path compression system, then $|S| \leq 5n$.


Proposition
The sequence of path compressions performed by the a-DFS is a Loebl-Nesetril strong post-order path compression system.
- Then the a-DFS halts in linear time.
Safe-Strongly-Connected Components

Safe-Strongly-Connectedness

Say $U \subseteq V$ is *safe-strongly-connected* ($s$-$SC$) when for every $(u, v) \in U \times U$

\[ \sigma \sqsubseteq : u \overset{U}{\sim} v \text{ for some } \sigma \sqsubseteq \in \Sigma^\Gamma. \]

Safe-Strongly-Connected Components

Consider the following binary relation $\sim_{s$-$scc}$ on $V$:

\[ \sim_{s$-$scc} \triangleq \left\{ (u, v) \in V \times V \mid \exists U \subseteq V \text{ s.t. } U \text{ is } s$-$SC \text{ and } u, v \in U \right\}. \]

- It holds that $\sim_{s$-$scc}$ is an equivalence relation on $V$.

Equivalence classes of $\sim_{s$-$scc}$ are *safe-strongly-connected components* of $\Gamma$. 
s-SCC $\subseteq$ a-SCC

\[ D_{a\text{-scc}} = \left\{ \{a, c\}, \{b\} \right\}; \]

\[ D_{s\text{-scc}} = \left\{ \{a\}, \{b\}, \{c\} \right\}. \]
s-SCC ⊆ a-SCC

\[ D_{a-\text{scc}} = \left\{ \{a, c\}, \{b\} \right\}; \]

\[ D_{s-\text{scc}} = \left\{ \{a\}, \{b\}, \{c\} \right\}. \]
Each s-SCC $C$ induces a subtree $T_C$ in one of the a-palm-trees constructed by the a-DFS.
(a) An arena $\Gamma$.

(b) The a-palm-tree generated by a-DFS rooted at $A$, with indices of vertices and labelled arcs.
(c) An arena $\Gamma$.

(d) The s-SCCs of $\Gamma$. 
Subtrees, Roots, and LowLinks

- Each s-SCC $C$ induces a subtree $T_C$ in one of the a-palm-trees constructed by the a-DFS.
- The problem of computing the s-SCCs reduces to that of finding the roots of the s-SCCs in $T$, as the classical problem of finding the SCCs of a directed graph reduced to that of finding the roots of the SCCs.
(e) An arena $\Gamma$.

(f) The roots of the s-SCCs of $\Gamma$. 
Subtrees, Roots, and LowLinks

- Each s-SCC $C$ induces a subtree $T_C$ in one of the a-palm-trees constructed by the a-DFS.

- The problem of computing the s-SCCs reduces to that of finding the roots of the s-SCCs in $T$, 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 s-SCCs. It is based on an a-lowlink indexing, similar to the lowlink calculation performed by Tarjan’s SCC algorithm.
  - i.e., $v \in V$ is the root of some s-SCCs of $\Gamma$ iff $a\text{-lowlink}(v) = \text{idx}[v]$. 
The a-lowlink\([v]\) is the smallest index of any vertex \(u\) which is in the same s-SCC as \(v\) and such that \(u\) can reach \(v\) by traversing: at most one frond or cross arc, and then zero or more tree arcs.
Procedure `compute-s-SCC(A)`

- **input**: An arena $A = (V, A, (V\circ, V\n))$.
- **output**: The s-SCC of $A$.

1. **foreach** $u \in V$ **do**
   1.1. `idx[u] ← +∞;`
   1.2. `a-lowlink[u] ← +∞;`
   1.3. `on_Stack[u] ← false;`
   1.4. `D.make_set(u);`
   1.5. `ready_Stack[u] ← ∅;`
   1.6. **if** $u \in V\circ$ **then**
      1.6.1. `low_ready[u] ← +∞;`
      1.6.2. `cnt[u] ← |N^\text{out}_A(u)|;`

2. `next_idx ← 1; St ← ∅;`
3. **foreach** $u \in V\n$ **do**
4. 4.1. **if** `idx[u] = +∞` **then**
5. 4.2. 4.1.1. `s-SCC-visit(u, A);`

6. **foreach** $u \in V\circ$ **do**
7. 6.1. **if** `idx[u] = +∞` **then**
8. 6.2. 6.1.1. `idx[u] ← next_idx;`
9. 6.3. `next_idx ← next_idx + 1;`
10. 6.4. `ta_lowlink[u] ← idx[u];`
Procedure \textit{s-SCC-visit}(v, A)
\begin{algorithmic}
    \Comment{input: A vertex $v \in V$.}
    \State $\text{idx}[v] \leftarrow \text{next_idx}$;
    \State $\text{a-lowlink}[v] \leftarrow \text{next_idx}$;
    \State $\text{next_idx} \leftarrow \text{next_idx} + 1$;
    \State $\text{St}.\text{push}(v)$;
    \State $\text{on}_{-}\text{Stack}[v] \leftarrow \text{true}$
    \Comment{Check the in-neighbourhood of $v$}
    \ForEach{$u \in \mathcal{N}_{A}^{\text{in}}(v)$}
        \If{$\text{idx}[u] = +\infty$}
            \If{$u \in V_{\square}$}
                \State $\text{s-SCC-visit}(u, A)$;
                \State $\text{a-lowlink}[v] \leftarrow \min(\text{a-lowlink}[v], \text{a-lowlink}[u])$;
                \State $\mathcal{D}.\text{Union}(u, v)$;
            \Else
                \State $\text{low\_ready}[u] \leftarrow \min(\text{low\_ready}[u], \text{idx}[v])$;
                \State $\text{cnt}[u] \leftarrow \text{cnt}[u] - 1$;
                \If{$\text{cnt}[u] = 0$}
                    \State $\text{low\_v} \leftarrow \text{the unique } x \text{ such that } \text{idx}[x] = \text{low\_ready}[u]$;
                    \State $\gamma \leftarrow \mathcal{D}.\text{find}(\text{low\_v})$;
                    \If{$\text{on}_{-}\text{Stack}[\gamma] = \text{true}$}
                        \State $\text{ready\_St}[\gamma].\text{push}(u)$;
                    \EndIf
                \ElseIf{$\text{on}_{-}\text{Stack}[u] = \text{true}$}
                    \State $\text{a-lowlink}[v] \leftarrow \min(\text{a-lowlink}[v], \text{idx}[u])$;
                \EndIf
            \EndIf
        \EndIf
    \EndFor
\end{algorithmic}
Deciding Update Games

If $C = V$, return YES; otherwise, NO.
Deciding Update Games

If $C = V$, return YES; otherwise, NO.
Thank you for the attention