APPENDIX

AN EXAMPLE RUN OF ALGORITHM `RootedAsync()`

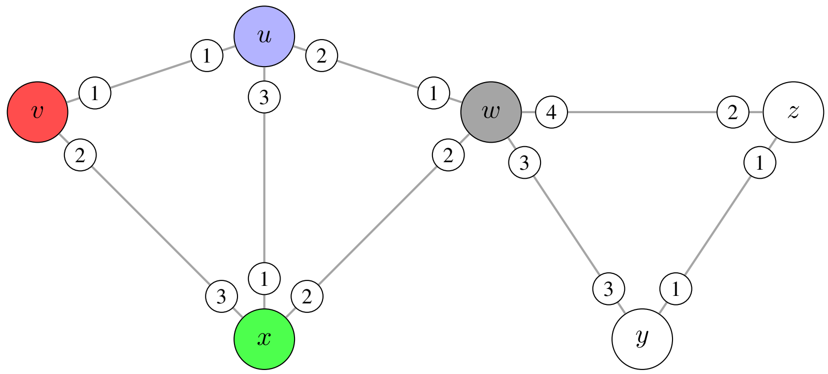

Fig. 5: Graph for the example run. Node colors are illustrative from the original image and may not reflect dynamic states in this trace.

Consider $k = 6$ agents, $a_1, a_2, a_3, a_4, a_5, a_6$, with IDs sorted in ascending order (i.e., $a_1.\mathsf{ID} < a_2.\mathsf{ID} < \cdots < a_6.\mathsf{ID}$). Initially, all agents are at node v (the root) and are in state *unsettled*. The DFS-based P1Tree construction begins. $\psi(N)$ denotes the agent settled at node N. $A_{unsettled}$ is the set of unsettled agents. $A_{vacated}$ is the set of agents whose nodes were vacated. $A_{scout} = A_{unsettled} \cup A_{vacated}$. The agent a_{min} is the agent in A_{scout} with the lowest ID.

1) **Settle at v (Root):** Agents $\{a_1, \ldots, a_6\}$ are at v. a_6 (highest ID) settles: $\psi(v) = a_6$. $a_6.state \leftarrow$ *settled*. $A_{unsettled} = \{a_1, \ldots, a_5\}$. $A_{scout} = \{a_1, \ldots, a_5\}$. $a_{min} = a_1$. $\psi(v).\mathsf{parentPort} = \perp$.
   `Parallel_Probe()` at v: Ports are 1 (to u) and 2 (to x). a_1 scouts port 1 (to u); a_2 scouts port 2 (to x).
   - a_1 (to u): Edge $\{v, u\}$ is type `t11`. Node u is **unvisited**. Scout a_1 finds $\xi(u) = \perp$ and $p_{uv} = 1$ (Rule R2). Reports u as EMPTY.
   - a_2 (to x): Edge $\{v, x\}$ is type `tpq`. Node x is **unvisited**. Scout a_2 finds $\xi(x) = \perp, p_{xv} \neq 1$. (Rule R3). port-1 neighbor (P1N) of x is u. a_2 visits u. $\xi(u) = \perp, p_{ux} \neq 1$. (Rule R3c). P1N of u is v. a_2 visits v. $\xi(v) = a_6$. Reports x as EMPTY.

   Probe results processed: u (`t11`, EMPTY), x (`tpq`, EMPTY). Based on probe results, node v is **visited** (it has unvisited neighbors).
   `Can_Vacate`($\psi(v) = a_6$): $\psi(v).\mathsf{parentPort} = \perp$ (root). a_6 remains *settled*. Node v is OCCUPIED. Next edge is to u (priority). A_{scout} moves to u. $a_6.\mathsf{recentChild} \leftarrow$ (port 1 to u). $a_{min}.\mathsf{childPort}$ (i.e. $a_1.\mathsf{childPort}$) $\leftarrow$ (port 1 to u).
2) **Settle at u:** a_5 (highest ID in $A_{unsettled} = \{a_1, \ldots, a_5\}$) settles: $\psi(u) = a_5$. $a_5.state \leftarrow$ *settled*. $a_5.\mathsf{parent} \leftarrow (a_6.\mathsf{ID}, \text{port 1 at } v)$. $a_5.\mathsf{parentPort} \leftarrow$ (port 1 at u). $A_{unsettled} = \{a_1, \ldots, a_4\}$.
   `Parallel_Probe()` at u: Ports (excl. parent port 1) are 2 (to w), 3 (to x). a_1 scouts port 2 (to w); a_2 scouts port 3 (to x).
   - a_1 (to w): Edge $\{u, w\}$ is type `tp1`. Node w is **unvisited**. Rule R2. Reports w as EMPTY.
   - a_2 (to x): Edge $\{u, x\}$ is type `tp1`. Node x is **unvisited**. Rule R2. Reports x as EMPTY.

   Probe results: w (`tp1`, EMPTY), x (`tp1`, EMPTY). Node u is **visited**.
   `Can_Vacate`($\psi(u) = a_5$): $\psi(u).\mathsf{nodeType} =$ **visited**. P1N of u is v. $\xi(v) = a_6 \neq \perp$ (and v is OCCUPIED). Rule (V2) applies. a_5 becomes *settledScout*. Node u is VACATED. $A_{vacated} = \{a_5\}$. $A_{scout} = \{a_1, \ldots, a_4, a_5\}$. $a_{min} = a_1$. $\psi(u) = a_5$ stores $a_5.\mathsf{P1Neighbor} \leftarrow a_6.\mathsf{ID}$, $a_5.\mathsf{portAtP1Neighbor} \leftarrow p_{vu} = 1$. Next edge to w. A_{scout} moves to w. $\psi(u) = a_5$ (now scout) updates $a_5.\mathsf{recentChild} \leftarrow$ (port 2 to w). $a_1.\mathsf{childPort} \leftarrow$ (port 2 to w).
3) **Settle at w:** a_4 settles: $\psi(w) = a_4$. $a_4.state \leftarrow$ *settled*. $a_4.\mathsf{parent} \leftarrow (a_5.\mathsf{ID}, \text{port 2 at } u)$. $a_4.\mathsf{parentPort} \leftarrow$ (port 1 at w). $A_{unsettled} = \{a_1, a_2, a_3\}$.
   `Parallel_Probe()` at w: Ports (excl. parent port 1) are 2 (to x), 3 (to y), 4 (to z). a_1 scouts port 2 (to x); a_2 scouts port 3 (to y); a_3 scouts port 4 (to z).
   - a_1 (to x): Reports x as EMPTY (as per step 2b logic, after 2-hop check, finds no scout $\psi(x)$ or $\psi(u)$ in A_{scout} that are settled at those nodes for P1N purposes, as $a_5 = \psi(u)$ is a scout now, but its P1N info is about v).
   - a_2 (to y): Reports y as EMPTY (Rule R3b, P1N z is empty, $p_{zy} = 1$).
   - a_3 (to z): Reports z as EMPTY (Rule R3b, P1N y is empty, $p_{yz} = 1$).

   Probe results: x, y, z all (`tpq`, EMPTY). Node w is **visited**.

`Can_Vacate`$(\psi(w) = a_4)$: $\psi(w)$.nodeType = **visited**. P1N of w is u. $\xi(u) = \perp$ (since $a_5 = \psi(u)$ is a scout). Condition "P1N is OCCUPIED" is false. a_4 remains *settled*. Node w is OCCUPIED. $A_{vacated} = \{a_5\}$. $A_{scout} = \{a_1, a_2, a_3, a_5\}$. $a_{min} = a_1$. Next edge to x. A_{scout} moves to x. a_4.recentChild $\leftarrow$ (port 2 to x). a_1.childPort $\leftarrow$ (port 2 to x).

4) **Settle at x:** a_3 settles: $\psi(x) = a_3$. $a_3.state \leftarrow$ *settled*. a_3.parent $\leftarrow$ (a_4.ID, port 2 at w). a_3.parentPort $\leftarrow$ (port 2 at x). $A_{unsettled} = \{a_1, a_2\}$. $A_{scout} = \{a_1, a_2, a_5\}$. $a_{min} = a_1$.
   `Parallel_Probe()` at x: Ports (excl. parent port 2) are 1 (to u), 3 (to v).
   - a_1 (to u): Edge $\{x, u\}$ is `t1p`. $\xi(u) = \perp$. P1N of u is v. $\xi(v) = a_6$. At x, find $b = \psi(u)$, which is $a_5 \in A_{scout}$, with a_5.P1Neighbor = a_6.ID. Yes. Reports u as VACATED (**visited**, Rule R3a-ii).
   - a_2 (to v): Edge $\{x, v\}$ is `tpq`. $\xi(v) = a_6$. Rule R1. Reports v as OCCUPIED (**visited**).

   Probe results: All explorable non-parent neighbors (u, v) are not EMPTY. Parent edge $\{w, x\}$ is `tpq`. Assume no other EMPTY neighbors via non-`tpq` edges. Node x becomes **partiallyVisited**.
   `Can_Vacate`$(\psi(x) = a_3)$: $\psi(x)$.nodeType = **partiallyVisited**. Rule (V4). a_3 becomes *settledScout*. Node x is VACATED. $A_{vacated} = \{a_5, a_3\}$. $A_{scout} = \{a_1, a_2, a_5, a_3\}$. DFS backtracks (nextPort is $\perp$). A_{scout} moves to w. a_1.childDetails $\leftarrow$ (a_3.ID, port 2 at w).
5) **At w (after backtrack from x):** $\psi(w) = a_4$. **`Parallel_Probe()` at w:** Next ports to probe: 3 (to y), 4 (to z).
   - a_1 scouts port 3 (to y): reports EMPTY. Same as before.
   - a_2 scouts port 4 (to z): reports EMPTY. Same as before.

   Probe results update: y and z are EMPTY. Node w remains **visited**.
   `Can_Vacate`$(\psi(w) = a_4)$: No change, a_4 remains *settled*, w is OCCUPIED. Priority to y (port 3). A_{scout} moves to y. a_4.recentChild $\leftarrow$ (port 3 to y). a_1.childPort $\leftarrow$ (port 3 to y). a_1.siblingDetails $\leftarrow$ (a_3.ID, port 2 at w) (sibling of y is x).
6) **Settle at y:** a_2 settles: $\psi(y) = a_2$. $a_2.state \leftarrow$ *settled*. a_2.parent $\leftarrow$ (a_4.ID, port 3 at w). a_2.parentPort $\leftarrow$ (port 3 at y). a_2.sibling $\leftarrow$ (a_3.ID, port 2 at w). $A_{unsettled} = \{a_1\}$. $A_{scout} = \{a_1, a_5, a_3\}$. $a_{min} = a_1$.
   `Parallel_Probe()` at y: Ports (excl. parent port 3) are 1 (to z), 2 (to x).
   - a_1 (to z): Edge $\{y, z\}$ is `t11`. Node z is **unvisited**. Rule R2. Reports z as EMPTY.
   - a_5 (to x): Edge $\{y, x\}$ is `tpq`. $\xi(x) = \perp$. P1N u, P1N of u is $v(\xi(v) = a_6)$. At y: find $c = \psi(u) = a_5 \in A_{scout}$ (yes). Find $b = \psi(x) = a_3 \in A_{scout}$ (yes). Rule R3c-ii(α). Reports x as VACATED (**partiallyVisited**).

   Probe results: z (`t11`, EMPTY), x (`tpq`, **partiallyVisited**). Node y is **visited**.
   `Can_Vacate`$(\psi(y) = a_2)$: $\psi(y)$.nodeType = **visited**. P1N of y is z. $\xi(z) = \perp$ (it's EMPTY). Returns *settled*. Node y is OCCUPIED. Next edge to z. A_{scout} moves to z. a_2.recentChild $\leftarrow$ (port 1 to z). a_1.childPort $\leftarrow$ (port 1 to z). a_1.siblingDetails $\leftarrow \perp$.
7) **Settle at z:** a_1 settles: $\psi(z) = a_1$. $a_1.state \leftarrow$ *settled*. a_1.parent $\leftarrow$ (a_2.ID, port 1 at y). a_1.parentPort $\leftarrow$ (port 1 at z). a_1.sibling $\leftarrow \perp$. $A_{unsettled} = \{\}$. $k = 6$ agents are settled.

Construction phase ends as $A_{unsettled}$ is empty. Current agent states and logical locations (physical location of scouts is z): $v : \psi(v) = a_6(\textit{settled})$ $u : \psi(u) = a_5(\textit{settledScout})$ $w : \psi(w) = a_4(\textit{settled})$ $x : \psi(x) = a_3(\textit{settledScout})$ $y : \psi(y) = a_2(\textit{settled})$ $z : \psi(z) = a_1(\textit{settled})$ $A_{vacated} = \{a_3, a_5\}$ (sorted by ID). They are physically co-located at z.

Retrace Phase: $A_{vacated} = \{a_3, a_5\}$. Current location of $A_{vacated}$ group: z. The lead retrace agent $a_{min\_retrace}$ is a_3. Goal: a_3 to x, a_5 to u.

a. **At z (settled agent $\psi(z) = a_1$):** $A_{vacated} = \{a_3, a_5\}$ is present. $a_{min} = a_3$. Node z is occupied by $a_1(\xi(z) = a_1)$. $\psi(z) = a_1$ has a_1.recentChild = $\perp$ (leaf in construction). The group moves to parent of z. a_3.nextAgentID $\leftarrow \psi(z)$.parent.ID (i.e., a_2.ID). a_3.nextPort $\leftarrow \psi(z)$.parentPort (port 1 at z). a_3.siblingDetails $\leftarrow \psi(z)$.sibling ($\perp$). $A_{vacated} = \{a_3, a_5\}$ moves to y via z's port 1. a_3.arrivalPort at y becomes port 1.
b. **At y (settled agent $\psi(y) = a_2$):** $A_{vacated} = \{a_3, a_5\}$ arrives. $a_{min} = a_3$. Node y is occupied by $a_2(\xi(y) = a_2)$. $\psi(y) = a_2$ has a_2.recentChild = (port 1 to z). a_3.arrivalPort (port 1 at y) matches $\psi(y)$.recentChild. a_3.siblingDetails is $\perp$. $\psi(y)$.recentChild $\leftarrow \perp$. The group moves to parent of y. a_3.nextAgentID $\leftarrow \psi(y)$.parent.ID (i.e., a_4.ID). a_3.nextPort $\leftarrow \psi(y)$.parentPort (port 3 at y). a_3.siblingDetails $\leftarrow \psi(y)$.sibling ((a_3.ID, port 2 at w)). $A_{vacated} = \{a_3, a_5\}$ moves to w via y's port 3. a_3.arrivalPort at w becomes port 3.
c. **At w (settled agent $\psi(w) = a_4$):** $A_{vacated} = \{a_3, a_5\}$ arrives. $a_{min} = a_3$. Node w is occupied by $a_4(\xi(w) = a_4)$. $\psi(w) = a_4$ has a_4.recentChild = (port 3 to y). a_3.arrivalPort (port 3 at w) matches $\psi(w)$.recentChild. a_3.siblingDetails is (a_3.ID, port 2 at w), which is not $\perp$. The group moves to the sibling node x. a_3.nextAgentID $\leftarrow a_3$.ID (from siblingDetails). a_3.nextPort $\leftarrow$ (port 2 at w) (from siblingDetails). $\psi(w)$.recentChild $\leftarrow$ (port 2 at w) (updates to current traversal direction). a_3.siblingDetails $\leftarrow \perp$. $A_{vacated} = \{a_3, a_5\}$ moves to x via w's port 2. a_3.arrivalPort at x becomes port 2.
d. **At x (original node of a_3, currently VACATED):** $A_{vacated} = \{a_3, a_5\}$ arrives. $a_{min} = a_3$. Node x is VACATED

($\xi(x) = \perp$). a_3.nextAgentID is a_3.ID. Agent $a_3 \in A_{vacated}$ matches. $a_3.state \leftarrow$ *settled*. a_3 occupies x. $\psi(x) \leftarrow a_3$. $A_{vacated}$ becomes $\{a_5\}$. a_{min} (for the remaining $A_{vacated}$) is now a_5. $\psi(x) = a_3$ has a_3.recentChild $= \perp$. The group (now just $\{a_5\}$) moves to parent of x. a_5.nextAgentID $\leftarrow \psi(x)$.parent.ID (i.e., a_4.ID). a_5.nextPort $\leftarrow \psi(x)$.parentPort (port 2 at x). a_5.siblingDetails $\leftarrow \psi(x)$.sibling($\perp$). $A_{vacated} = \{a_5\}$ moves to w via x's port 2. a_5.arrivalPort at w becomes port 2.

e. **At w (settled agent $\psi(w) = a_4$):** $A_{vacated} = \{a_5\}$ arrives. $a_{min} = a_5$. Node w is occupied by $a_4(\xi(w) = a_4)$. $\psi(w) = a_4$ has a_4.recentChild = (port 2 at w) (updated in step c). a_5.arrivalPort (port 2 at w) matches $\psi(w)$.recentChild. a_5.siblingDetails is $\perp$. $\psi(w)$.recentChild $\leftarrow \perp$. The group moves to parent of w. a_5.nextAgentID $\leftarrow \psi(w)$.parent.ID (i.e., a_5.ID). a_5.nextPort $\leftarrow \psi(w)$.parentPort (port 1 at w). a_5.siblingDetails $\leftarrow \psi(w)$.sibling($\perp$). $A_{vacated} = \{a_5\}$ moves to u via w's port 1. a_5.arrivalPort at u becomes port 2.

f. **At u (original node of a_5, currently VACATED):** $A_{vacated} = \{a_5\}$ arrives. $a_{min} = a_5$. Node u is VACATED ($\xi(u) = \perp$). a_5.nextAgentID is a_5.ID. Agent $a_5 \in A_{vacated}$ matches. $a_5.state \leftarrow$ *settled*. a_5 occupies u. $\psi(u) \leftarrow a_5$. $A_{vacated}$ becomes $\emptyset$. Retrace phase ends as $A_{vacated}$ is empty.

Final agent settlement: $\psi(v) = a_6, \psi(u) = a_5, \psi(w) = a_4, \psi(x) = a_3, \psi(y) = a_2, \psi(z) = a_1$. Dispersion is achieved.

TABLE OF VARIABLES

TABLE II: Variables Used by Agents

Variable Name	**Description**
Generic Agent Properties (applicable to any agent a)	
a.ID	Unique identifier of the agent.
a.state	Current operational state of the agent (e.g., *unsettled*, *settled*, *settledScout*).
a.arrivalPort	The port number through which agent a arrived at its current node.
a.treeLabel	For general dispersion: a tuple $\langle \texttt{leaderID}, \texttt{level}, \texttt{weight} \rangle$ identifying the P1Tree exploration the agent is part of. Contains the ID of the tree's root agent, the tree's merger level, and its current weight (number of agents).
Variables for Settled Agents (e.g., agent $a = \psi(v)$ at node v)	
a.nodeType	The type of node v where the agent is settled (e.g., **unvisited**, **partiallyVisited**, **fullyVisited**, **visited**).
a.parent	A tuple: (ID of the agent settled at v's parent node in the P1Tree, port number at the parent node leading to v). Is $\perp$ for the root agent.
a.parentPort	The port number at node v that leads to its parent in the P1Tree. Is $\perp$ for the root agent.
a.P1Neighbor	ID of the agent settled at the port-1 neighbor of node v. Stores $\perp$ if the port-1 neighbor is EMPTY or unvisited.
a.portAtP1Neighbor	The port number at v's port-1 neighbor (say w) that leads back to v.
a.vacatedNeighbor	Boolean flag. True if a neighbor of v (for which v is a port-1 neighbor and would need to be OCCUPIED for that neighbor to be VACATED) has itself become VACATED. Used in Algorithm `Can_Vacate`.
a.recentChild	The port number at node v that leads to the child most recently visited by the DFS traversal originating from v.
a.sibling	A tuple: (ID of the agent at the previous sibling node in the DFS tree, port number at v's parent leading to that sibling). Is $\perp$ if v is the first child.
a.recentPort	The port number most recently used by the scout agents to depart from node v (either towards a child or back to the parent).
a.probeResult	Stores the overall highest priority result (e.g., next edge to traverse) obtained from the `Parallel_Probe` procedure executed at node v.
a.checked	The count of incident ports at node v that have already been explored during the `Parallel_Probe` procedure.
Temporary Variables for Scouting Agents (agent $a \in A_{\text{SCOUT}}$ during `Parallel_Probe`)	
a.scoutPort	The port number at the current DFS head node that this scout agent a is assigned to explore.
a.scoutEdgeType	The type of the edge (e.g., `tp1`, `t11`) discovered by scout a along its `scoutPort`.

Continued on next page

TABLE II – continued from previous page

`Variable Name`	**Description**
a.scoutP1Neighbor	(During probe of neighbor y) Stores ID of the agent at port-1 neighbor of y (say z), or $\perp$.
a.scoutPortAtP1Neighbor	(During probe of y) Stores port at z leading to y.
a.scoutP1P1Neighbor	(During probe of y's P1N z) Stores ID of agent at port-1 neighbor of z (say w), or $\perp$.
a.scoutPortAtP1P1Neighbor	(During probe of z) Stores port at w leading to z.
a.scoutResult	A tuple $\langle p_{xy}, \texttt{edgeType}, \texttt{nodeType}_y, a' \rangle$ storing the individual result found by scout a for its assigned port.
Context Variables for the Lead Scout Agent (e.g., $a = a_{\min}$)	
a.prevID	ID of the agent settled at the node from which the DFS head (and scout group) just departed.
a.childPort	Port at the current DFS head that will be taken to visit the next child. This info is used to set up the child's parent information.
a.siblingDetails	A tuple carrying information about the current child's previous sibling, to be passed to the agent settling at the next child. Format: (Sibling Agent ID, Port at Parent to Sibling).
a.childDetails	A tuple carrying information about the child node just exited during a backtrack operation, to be used by the parent. Format: (Child Agent ID, Port at Parent to Child).
a.nextAgentID	(During Retrace phase) The ID of the agent whose original settled node the A_{VACATED} group is currently moving towards.
a.nextPort	(During Retrace phase) The port number the A_{VACATED} group will take to reach the node associated with `nextAgentID`.

PSEUDOCODES OF ALGORITHMS

A. *Pseudocode of Algorithm 1 `Centralized_P1Tree()`*

Algorithm 1: CENTRALIZED_P1TREE(G)

Input: connected, port-labelled graph $G = (V, E)$
Output: a Port-One tree $\mathcal{T}$ of G

$\mathcal{T} \leftarrow \emptyset$;
mark all vertices **unvisited**;
while *there exists an* **unvisited** *vertex* **do**
    pick any **unvisited** vertex v, push v on a stack S;
    while $S \neq \emptyset$ **do**
        $u \leftarrow$ pop S;
        foreach *incident edge* $e = [u, p_{uv}, p_{vu}, v]$ *of type* `tp1`, `t11` *or* `t1q` **do**
            if v *is* **unvisited then**
                $\mathcal{T} \leftarrow \mathcal{T} \cup \{e\}$;
                push v on S;
                mark v **visited**;
sort all edges of type `tpq` in lexicographical order;
foreach *edge* e *in sorted order* **do**
    if $\mathcal{T} \cup \{e\}$ *is acyclic* **then**
        $\mathcal{T} \leftarrow \mathcal{T} \cup \{e\}$;
        if $\mathcal{T}$ *forms a single connected component* **then**
            break;
return $\mathcal{T}$;

B. *Pseudocode of Algorithm 2 DFS_P1Tree()*

Algorithm 2: `DFS_P1Tree()`

Input: Root vertex v_0, port-labelled graph $G = (V, E)$
Output: P1Tree $\mathcal{T}$

edge priority: $\texttt{tp1} \succ \texttt{t11} \sim \texttt{t1q} \succ \texttt{tpq}$, smallest incident port number under each type;
initialize: $\mathcal{T} \leftarrow \emptyset$, mark all vertices **unvisited**, stack $S \leftarrow \emptyset$;
$S.push(v_0)$;
$\texttt{type}(v_0) \leftarrow$ **visited**;
while $S \neq \emptyset$ **do**
    $u \leftarrow S.top()$;
    $e_{next} \leftarrow \varnothing$;
    $\mathcal{E} \leftarrow$ sorted list of edges incident to u in order of edge-priority;
    for $e \in \mathcal{E}$ **do**
        let $e = [u, p_{uv}, p_{vu}, v]$ be the edge;
        if $\texttt{type}(v) =$ **unvisited then**
            $e_{next} \leftarrow e$;
            break;
        else
            if $\texttt{type}(v)$ **= partiallyVisited** and $\texttt{type}(\{u, v\}) \in \{\texttt{tp1}, \texttt{t11}\}$ **then**
                $e_{next} \leftarrow e$;
                break;
    if $e_{next} \neq \varnothing$ **then**
        let $e = [u, p_{uv}, p_{vu}, v]$ be the edge;
        let $e_{\uparrow} = [w, p_{wu}, p_{uw}, u]$ be the parent edge of u;
        if *$e, e_{\uparrow}$ are $\texttt{tpq}$ and no incident edge at u in $\mathcal{T}$ is of type $\langle \texttt{tp1}, \texttt{t11}, \texttt{t1q} \rangle$* **then**
            $\texttt{type}(u) \leftarrow$ **partiallyVisited**;
            $S.pop()$;
        else
            $parent(v) \leftarrow u$;
            $S.push(v)$;
            $\texttt{type}(v) \leftarrow$ **visited**;
    else
        $\texttt{type}(u) \leftarrow$**fullyVisited**;
        $S.pop()$;

C. *Pseudocode of Algorithm 3* `Can_Vacate`()

Algorithm 3: `Can_Vacate()`

Input: Agent $\psi(x)$ at node x
Output: State of $\psi(x)$

if $\psi(x)$.*parentPort* $= \perp$ **then**
    return *settled*;
else if $\psi(x)$.*nodeType* = ***visited*** **then**
    visit port 1 neighbor w;
    if $\xi(w) \neq \perp$ **then**
        $\psi(w) \leftarrow \xi(w)$;
        set $\psi(w)$.vacatedNeighbor $= true$;
        return to x;
        return *settledScout*;
    return *settled*;
else if $\psi(x)$.*nodeType* = ***fullyVisited*** *and* $\psi(x)$.*vacatedNeighbor* $= false$ **then**
    return *settledScout*;
else if $\psi(x)$.*nodeType* = ***partiallyVisited*** **then**
    return *settledScout*;
else if $\psi(x)$.*portAtParent* $= 1$ **then**
    Visit parent z of x;
    if $\psi(z)$.*vacatedNeighbor* $= false$ **then**
        $\psi(z)$.state $\leftarrow$ *settledScout*;
        $\psi(z)$ joins $A_{vacated}$;
        return to x;
        set $\psi(x)$.vacatedNeighbor $= true$;
        return *settled*;
    else
        return to x;
        return *settled*;

D. *Pseudocode of Algorithm 4 `Parallel_Probe`()*

Algorithm 4: `Parallel_Probe()`

Input: Current DFS-head x with settled agent $\psi(x)$, and A_{scout}

Output: Next port p_{xy}

```
ψ(x).probeResult ← ⊥; ψ(x).checked ← 0;
while ψ(x).checked < δ_x do
    A_scout = {a_1, ..., a_s} in the increasing order ID;
    Δ' ← min(s, δ_x − ψ(x).checked);
    for j ← 1 to Δ' do
        a ← next agent in A_scout;
        if ψ(x).parent.Port = j + ψ(x).checked then
            j ← j + 1; Δ' ← min(s + 1, δ_x − ψ(x).checked);
        a.scoutPort ← j + ψ(x).checked;
        move via a.scoutPort to reach y;
        a.scoutEdgeType ← type({x, y});
        if ξ(y) ≠ ⊥ then
            ψ(y) ← ξ(y);
            a returns to x;
        else
            if ξ(y) = ⊥ ∧ p_yx = 1 then
                ψ(y) ← ⊥;
                a returns to x;
            else
                z ← port-1 neighbor of y;
                a.scoutP1Neighbor ← ξ(z);
                a.scoutPortAtP1Neighbor ← p_zy;
                if ξ(z) ≠ ⊥ then
                    a returns to x;
                    check ∃ b ∈ A_scout : b.scoutP1Neighbor = ξ(z) ∧ b.scoutPortAtP1Neighbor = p_zy;
                    if b found then
                        ψ(y) ← b
                    else
                        ψ(y) ← ⊥
                else
                    if ξ(z) = ⊥ ∧ p_zy = 1 then
                        ψ(y) ← ⊥;
                        a returns to x;
                    else
                        w ← port-1 neighbor of z;
                        a.scoutP1P1Neighbor ← ξ(w);
                        a.scoutPortAtP1P1Neighbor ← p_wz;
                        if ξ(w) = ⊥ then
                            ψ(y) ← ⊥
                        else
                            a returns to x;
                            check ∃ c ∈ A_scout : c.scoutP1Neighbor = ξ(w) ∧ c.scoutPortAtP1Neighbor = p_wz;
                            if c found then
                                check ∃ b ∈ A_scout : b.scoutP1Neighbor = c ∧ b.scoutPortAtP1Neighbor = p_zy;
                                if b found then
                                    ψ(y) ← b
                                else
                                    ψ(y) ← ⊥
                            else
                                ψ(y) ← ⊥
        a.scoutResult ← ⟨p_xy, a.scoutEdgeType, ψ(y).nodeType, ψ(y)⟩;
    ψ(x).checked ← ψ(x).checked + Δ';
    ψ(x).probeResult ← highest priority edge from a ∈ A_scout based on ψ(y).nodeType;
return p_xy from ψ(x).probeResult;
```

E. Pseudocode of Algorithm 5 RootedAsync()

Algorithm 5: `RootedAsync()`

Input: A set of k agents at root node v_0 in G

$A \leftarrow$ set of agents;
For each $a \in A$, initialize all variables to $\perp$;
for $a \in A$ **do**
    a.state $\leftarrow$ *unsettled*;
$A_{unsettled} \leftarrow A$;
$A_{vacated} \leftarrow \emptyset$;
while $A_{unsettled} \neq \emptyset$ **do**
    $v \leftarrow$ current node;
    $A_{scout} \leftarrow A_{unsettled} \cup A_{vacated}$;
    $a_{min} \leftarrow$ Lowest ID agent in A_{scout} ;
    if *there is no settled agent in* v **then**
        $\psi(v) \leftarrow$ agent with highest ID in $A_{unsettled}$;
        $\psi(v)$.state $\leftarrow$ *settled*;
        $\psi(v)$.parent $\leftarrow$ (a_{min}.prevID, a_{min}.childPort);
        a_{min}.childPort $\leftarrow \perp$;
        $\psi(v)$.parentPort $\leftarrow a_{min}$.arrivalPort;
        $A_{unsettled} \leftarrow A_{unsettled} - \{\psi(v)\}$;
        if $A_{unsettled} = \emptyset$ **then**
            break;
    a_{min}.prevID $\leftarrow \psi(v)$.ID;
    if $\delta_v \geq k-1$ **then**
        run `Parallel_Probe`($\psi(v), A_{scout}$) for $k-1$ ports;
        send unsettled agents to empty neighbors;
        break;
    $\psi(v)$.sibling $\leftarrow a_{min}$.siblingDetails;
    a_{min}.siblingDetails $\leftarrow \perp$;
    nextPort $\leftarrow$ `Parallel_Probe`($\psi(x), A_{scout}$);
    $\psi(v)$.state $\leftarrow$ `Can_Vacate()`;
    if $\psi(v)$.*state* =*settledScout* **then**
        $A_{vacated} \leftarrow A_{vacated} \cup \{\psi(v)\}$;
        $A_{scout} \leftarrow A_{unsettled} \cup A_{vacated}$;
    if *nextPort* $\neq \perp$ **then**
        $\psi(v)$.recentPort $\leftarrow$ nextPort;
        a_{min}.childPort $\leftarrow$ nextPort;
        if $\psi(v)$.*recentChild* $= \perp$ **then**
            $\psi(v)$.recentChild $\leftarrow$ nextPort;
        else
            a_{min}.siblingDetails $\leftarrow a_{min}$.childDetails;
            a_{min}.childDetails $\leftarrow \perp$;
            $\psi(v)$.recentChild $\leftarrow$ nextPort;
        All agents in A_{scout} move through nextPort;
    else
        a_{min}.childDetails $\leftarrow$ ($\psi(v)$.ID, $\psi(v)$.portAtParent);
        a_{min}.childPort $\leftarrow \perp$;
        $\psi(v)$.recentPort $\leftarrow \psi(v)$.parentPort;
        All agents in A_{scout} move though $\psi(v)$.parentPort;
`Retrace`($A_{vacated}$);

F. *Pseudocode of Algorithm 6* `Retrace()`

Algorithm 6: `Retrace()`

Input: $A_{vacated}$ - set of agents with state *settledScout*

while $A_{vacated} \neq \emptyset$ **do**
- $v \leftarrow$ current node;
- $a_{min} \leftarrow$ Lowest ID agent in $A_{vacated}$;
- **if** $\xi(v) = \perp$ **then**
  - `// no settled agent present at` v`, it must be in` $A_{vacated}$
  - find $a \in A_{vacated}$ with a.ID $= a_{min}$.nextAgentID at v;
  - a.state $\leftarrow$ *settled*;
  - $A_{vacated} \leftarrow A_{vacated} - \{a\}$;
  - $a_{min} \leftarrow$ Lowest ID agent in $A_{vacated}$;
  - $\psi(v) \leftarrow a$;
- `// If all agents are settled, retrace is complete`
- **if** $A_{vacated} = \emptyset$ **then**
  - break;
- `// Determine next move in post-order traversal`
- **if** $\psi(v)$.*recentChild* $\neq \perp$ **then**
  - **if** $\psi(v)$.*recentChild* $= a_{min}$.*arrivalPort* **then**
    - **if** a_{min}.*siblingDetails* $= \perp$ **then**
      - $\psi(v)$.recentChild $\leftarrow \perp$;
      - $(a_{min}$.nextAgentID$, a_{min}$.nextPort$) \leftarrow \psi(v).parent$;
      - a_{min}.siblingDetails $\leftarrow \psi(v)$.sibling;
    - **else**
      - $(a_{min}$.nextAgentID$, a_{min}$.nextPort$) \leftarrow a_{min}$.siblingDetails;
      - a_{min}.siblingDetails $\leftarrow \perp$;
      - $\psi(v)$.recentChild $\leftarrow a_{min}$.nextPort;
  - **else**
    - a_{min}.nextPort $\leftarrow \psi(v)$.recentChild;
    - Check if $\exists a \in A_{vacated} : a$.parent $= (\psi(v)$.ID$, \psi(v)$.recentChild$)$;
    - **if** a *found* **then**
      - a_{min}.nextAgentID $\leftarrow a$.ID;
      - a_{min}.nextPort $\leftarrow \psi(v)$.recentChild;
- **else**
  - $(parentID, portAtParent) \leftarrow \psi(v)$.parent;
  - a_{min}.nextAgentID $\leftarrow parentID$;
  - a_{min}.nextPort $\leftarrow \psi(v)$.parentPort;
  - a_{min}.siblingDetails $\leftarrow \psi(v)$.sibling;
- All agents in $A_{vacated}$ move through a_{min}.nextPort;