Efficient Dispersion of Mobile Robots on Graphs

Ajay D. Kshemkalyani    Faizan Ali

University of Illinois at Chicago

ajay@uic.edu
Roadmap

1. Introduction

2. System Model

3. Overview of Results

4. Algorithms
   - Helping-Sync and Helping-Async
   - Independent-Async
   - Independent-Bounded-Async
   - Tree-Switching-Async

5. Discussion and Conclusions
Problem Definition

$k$ robots placed arbitrarily at the $n$ nodes of an anonymous graph, where $k \leq n$, to coordinate with each other to reach a final configuration in which each robot is at a distinct node of the graph.

- Related to graph exploration by mobile robots, scattering on a graph, and load balancing on a graph.
- App: relocation of self-driven electric cars (robots) to recharge stations (nodes).
- App: minimize the total cost of $k$ agents sharing $n$ resources, located at various places, subject to the constraint that the cost of moving an agent to a different resource is much smaller than the cost of multiple agents sharing a resource.

Contribution

Five deterministic algorithms, for the sync and async models, to solve dispersion on graphs.
Related Work

- Dispersion in 2-D plane well-studied
- Dispersion on graphs introduced by Augustine and Moses Jr. [ICDCN’18]
  - $\Omega(D)$ time, $\Omega(\log n)$ bits/robot
  - Dispersion algorithms for synchronous systems for
    - paths, rings, trees, rooted trees, rooted graphs
    - General graphs: $O(\log n)$ bits/robot and $O(\Delta^D)$ time Log-N algorithm
    - General graphs: $O(n \log n)$ bits/robot and $O(m)$ time N-Log-N algorithm
- We claim Log-N and N-Log-N algorithm are incorrect (trapped in cycles while backtracking and fail to search entire graph) due to:
  - do not correctly coordinate concurrent searches which interfere
  - backtracking strategy not consistent with forward exploration
  - while backtracking, robot uses parent pointer of docked robot

- Related to graph exploration by mobile robots, scattering on a graph, and load balancing on a graph.
System Model

1. Robots have no visibility, are deterministic and distinguishable.
2. Robots can only communicate with co-located robots.
3. Undirected graph: $m$ edges, $n$ nodes, diameter $D$, degree $\Delta$, node degree $\delta$.
4. Graph is anonymous, i.e., nodes have no labels.
5. Nodes have no memory but the ports at a node have locally unique labels.
6. Robots do not know graph parameters.
## Overview of Results

**Table:** Comparison of the proposed algorithms for dispersion on graphs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Model</th>
<th>Memory Requirement at Each Robot (in bits)</th>
<th>Time Complexity</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log-N [AM18]</td>
<td>Sync.</td>
<td>$O(\log n)$</td>
<td>$O(\Delta^D)$</td>
<td>WRONG</td>
</tr>
<tr>
<td>N-Log-N [AM18]</td>
<td>Sync.</td>
<td>$O(n \log n)$</td>
<td>$O(m)$</td>
<td>WRONG</td>
</tr>
<tr>
<td>Helping-Sync</td>
<td>Sync.</td>
<td>$O(k \log \Delta)$</td>
<td>$O(m)$ steps</td>
<td>need to know $m$ for termination</td>
</tr>
<tr>
<td>Helping-Async</td>
<td>Async.</td>
<td>$O(k \log \Delta)$</td>
<td>$O(m)$ steps</td>
<td>no termination</td>
</tr>
<tr>
<td>Independent-Async</td>
<td>Async.</td>
<td>$O(k \log \Delta)$</td>
<td>$O(m)$ steps</td>
<td>no termination</td>
</tr>
<tr>
<td>Indep.-Bounded-Async</td>
<td>Async.</td>
<td>$O(D \log \Delta)$</td>
<td>$O(\Delta^D)$ steps</td>
<td>termination</td>
</tr>
<tr>
<td>Tree-Switching-Async</td>
<td>Async.</td>
<td>$O(\max(\log k, \log \Delta))$</td>
<td>$O((m - n)k)$ steps</td>
<td>no termination</td>
</tr>
</tbody>
</table>
Helping-Sync and Helping-Async

- A docked robot maintains data structures on behalf of visiting robots
  - \texttt{port\_entered} and \texttt{parent\_ptr}: \(O(\log \Delta)\) bits
  - \texttt{state} \(\in\) \{explore, backtrack, settled\}: 2 bits
  - \texttt{seen}: 1 bit
  - \texttt{round}: \(O(\log n)\) bits
  - \texttt{visited}[1, k] of type boolean: \(k\) bits
  - \texttt{entry\_port}[1, k] of type port: \(O(k \log \Delta)\) bits

\textbf{Theorem}

\textit{Algorithm Helping-Sync achieves dispersion in a synchronous system in} \(O(m)\) \textit{rounds with} \(O(k \log \Delta)\) \textit{bits at each robot.}

\textbf{Theorem}

\textit{Algorithm Helping-Async achieves dispersion (without termination) in an asynchronous system in} \(O(m)\) \textit{steps with} \(O(k \log \Delta)\) \textit{bits at each robot.}
Independent-Async

- Each robot maintains its own data structures
  - `port_entered`: $O(\log \Delta)$ bits
  - `state` $\in \{\text{explore, backtrack, settled}\}$: 2 bits
  - `visited[1, k]` of type boolean: $k$ bits
  - `stack` of type `port`: $O(k \log \Delta)$ bits
    - allows backtracking correctly

**Theorem**

*Algorithm Independent-Async achieves dispersion (without termination) in an asynchronous system in $O(m)$ steps with $O(k \log \Delta)$ bits at each robot.*
Independent-Bounded-Async

Each robot maintains its own data structures to do a depth-bounded DFS

- *port_entered*: \(O(\log \Delta)\) bits
- *state* \(\in\) \{explore, backtrack, settled\}: 2 bits
- *stack* of type port: \(O(D \log \Delta)\) bits
  - allows backtracking correctly
- *depth, depth_bound*: \(O(\log D)\) bits

Guaranteed to terminate

**Theorem**

*Algorithm Independent-Bounded-Async achieves dispersion in an asynchronous system in* \(O(\Delta^D)\) *steps with* \(O(D \log \Delta)\) *bits at each robot.*
Tree-Switching-Async

- Previous algos: each robot performed a separate DFS;
  - docked robot stored up to $k - 1$ `parent_ptrs` for $k - 1$ DFSs, or
  - traversing robot tracked up to $k - 1$ `parent_ptrs` for its own DFS

  $O(k \log \Delta)$ bits/robot

- We are allowed $O(\max(\log k, \log \Delta))$ bits/robot

- Robots must coordinate in associating with a DFS tree and its `parent_ptrs`

- `virtual_id` (of type robot identifier) tracks DFS tree instance robot is associated with currently
  - Strict (total) priority order defined on `virtual_ids`
  - When two robots meet, their DFS trees intersect
    - lower priority robot abandons its partially computed DFS tree and switches to higher priority DFS tree
  - Robot makes at most $k - 1$ tree switches
Tree-Switching-Async

- Each robot maintains its own data structures; interacts with docked robot
  - `port_entered, parent_ptr` (at docked robot): $O(\log \Delta)$ bits
  - `state ∈ {explore, backtrack, settled}`: 2 bits
  - `virtual_id` of type identifier: $O(\log k)$ bits
    - allows backtracking correctly
  - `depth`: $O(\log k)$ bits
    - associated with `virtual_id`
- lower priority robot switches to higher priority DFS tree

**Lemma**

For any value of `virtual_id`, an undocked robot docks or switches to a higher priority `virtual_id` within $4m - 2n + 2$ steps.

**Theorem**

Algorithm Tree-Switching-Async achieves dispersion (without termination) in an asynchronous system in $O((m - n)k)$ steps with $O(\max(\log k, \log \Delta))$ bits at each robot.
In doing a switch, the lower priority robot

1. updates its `virtual_id` to the higher priority
2. updates its `depth` variable to the new depth in the higher priority tree, and
3. updates its `parent_ptr` (if docked) to `port_entered` of the traversing robot or its `port_entered` (if traversing) to `parent_ptr` of the docked robot

If the traversing robot (whether in `explore` or `backtrack` state) does the switch, it continues the DFS in the newly-switched-to tree as if it had just entered that node where the switch occurs in `explore` state for the first time

Multiple robots may be executing the same tree instance possibly in different parts of the graph if they share the same `virtual_id`. 
Five algorithms for dispersion on graphs

Formulate *ongoing dispersion* problem

Rather than a one-shot dispersion, a robot, after docking and recharging, moves again on the graph and after some time, finds itself at some node from where it wants to search for an unoccupied node to dock again. Every time a docked robot moves, it creates a free node. This cycle repeats.

Open problem: to analyze our proposed algorithms and design new algorithms for ongoing dispersion
Thank You!