DDA 2010, lecture 6: Exploration and rendezvous

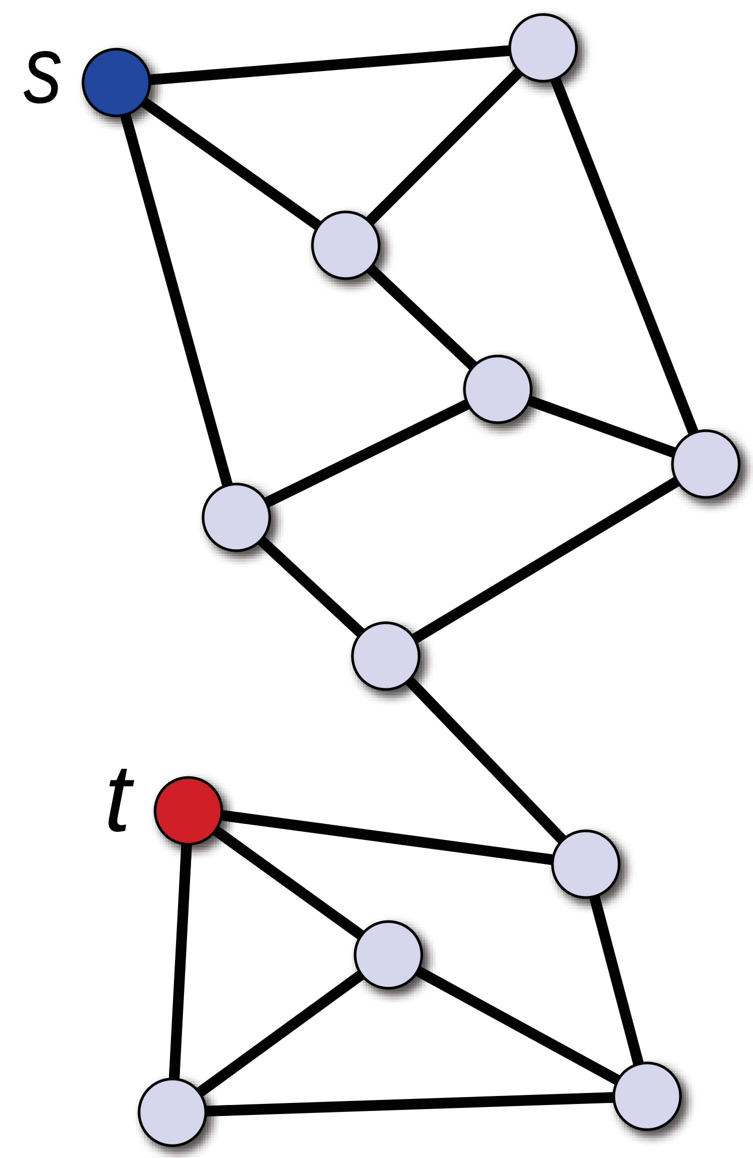
- Treasure hunt in port-numbered graphs

DDA 2010, lecture 6a: Treasure hunt in port-numbered graphs

- Universal traversal sequences exist

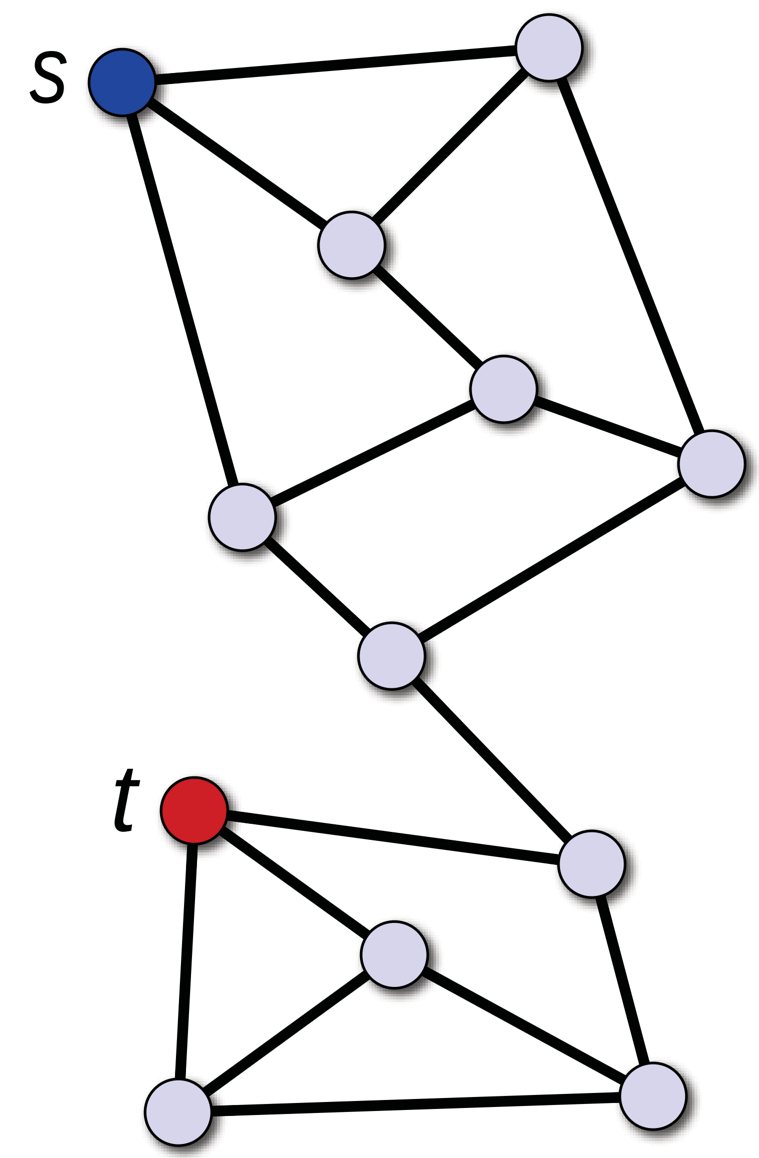
Graph exploration

- Connected graph
  with port-numbering
- **Robot** placed
  in some starting node *s*
- **Treasure** hidden
  in some target node *t*
- Program the robot
  so that it will find
  the treasure!



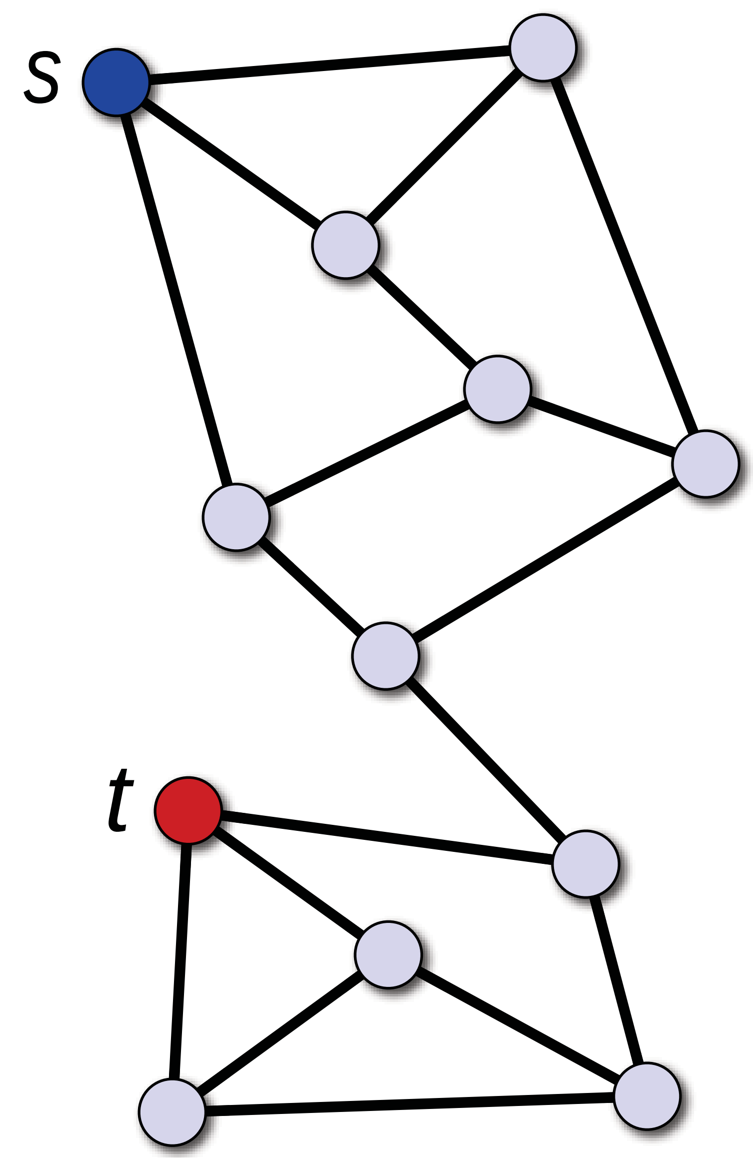
Graph exploration

- Connected graph with port-numbering
  - we will first focus on the case of d-regular graphs
  - assume that we know an upper bound on n



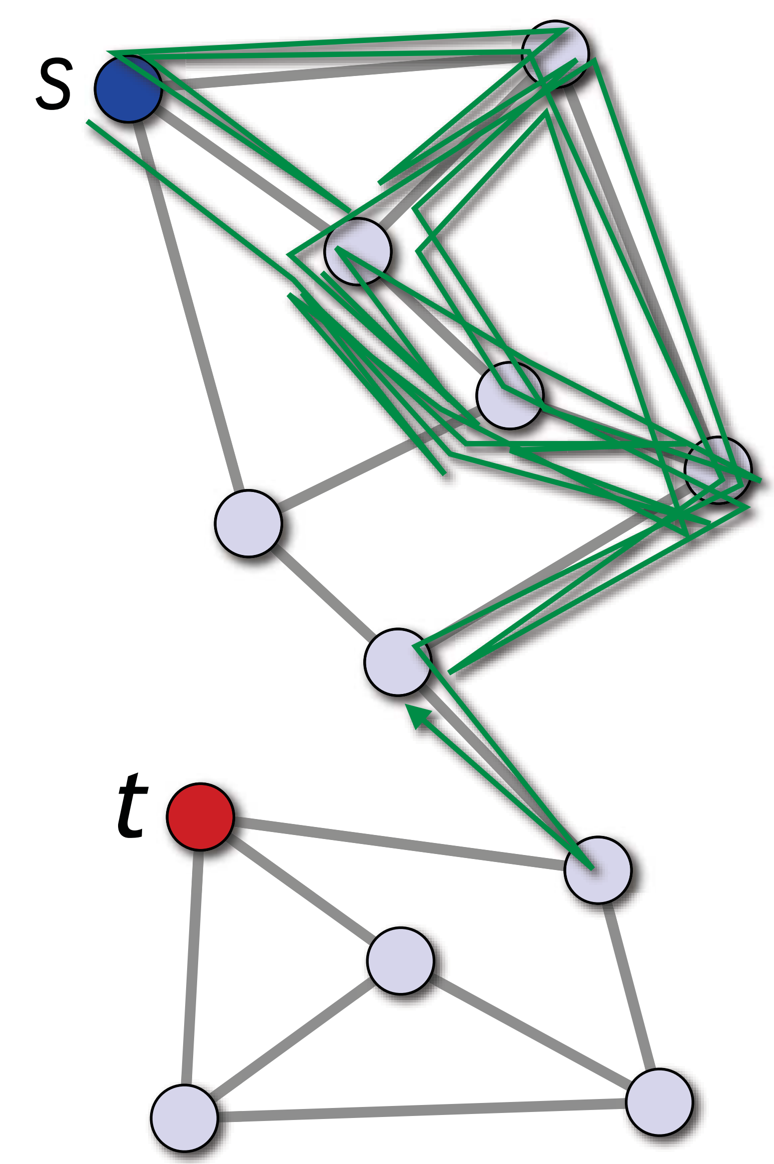
Graph exploration: random walks

- Simple solution with randomness:
  - just take a **random walk**



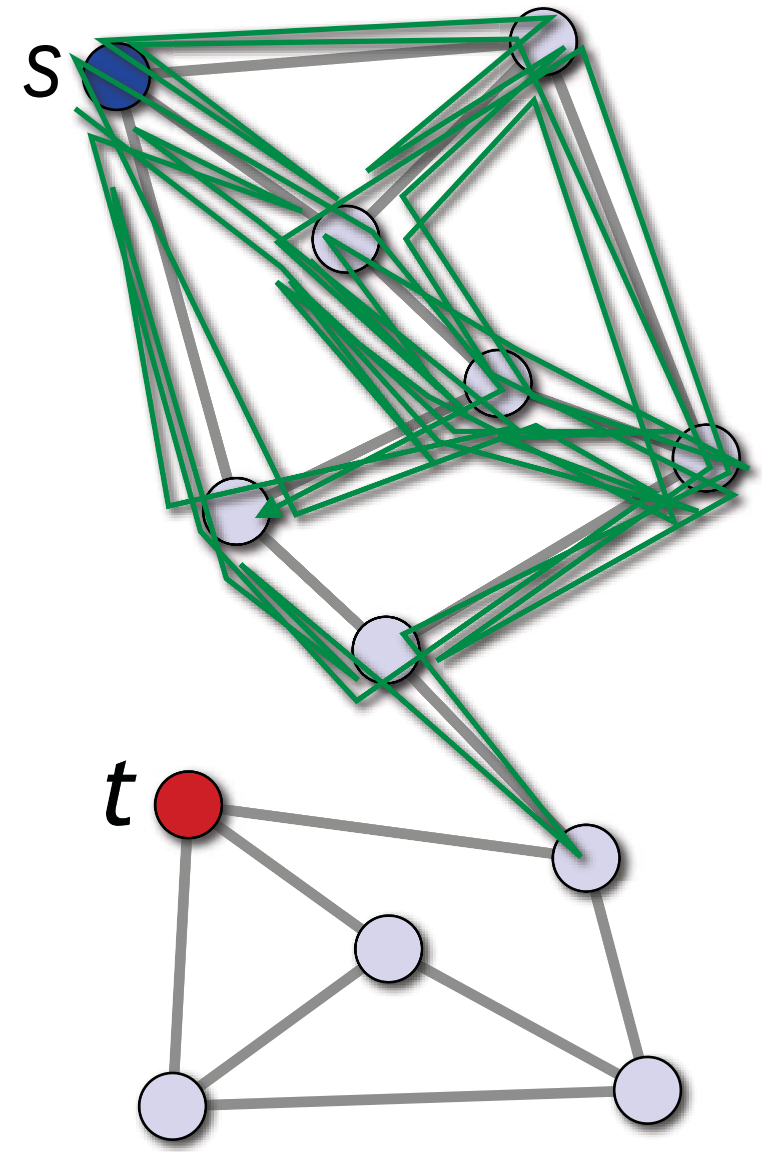
Graph exploration: random walks

- Simple solution with randomness:
  - just take a **random walk**



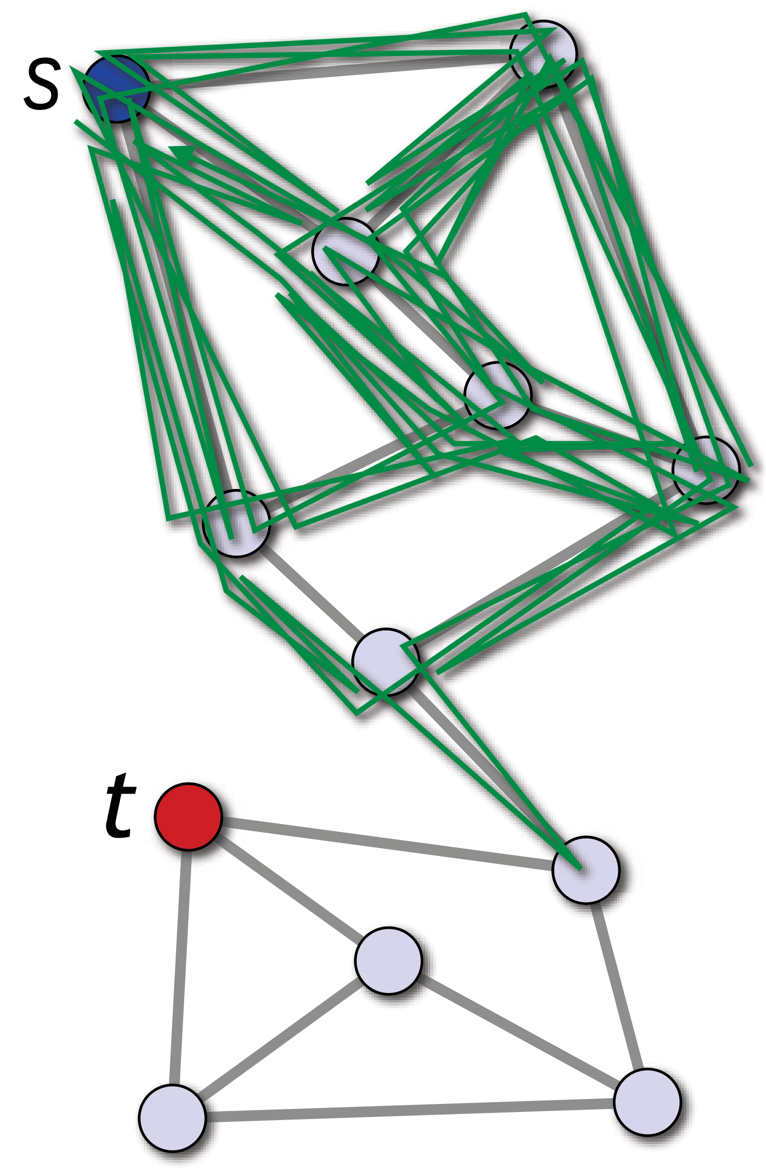
Graph exploration: random walks

- Simple solution with randomness:
  - just take a **random walk**
  - might take a while…



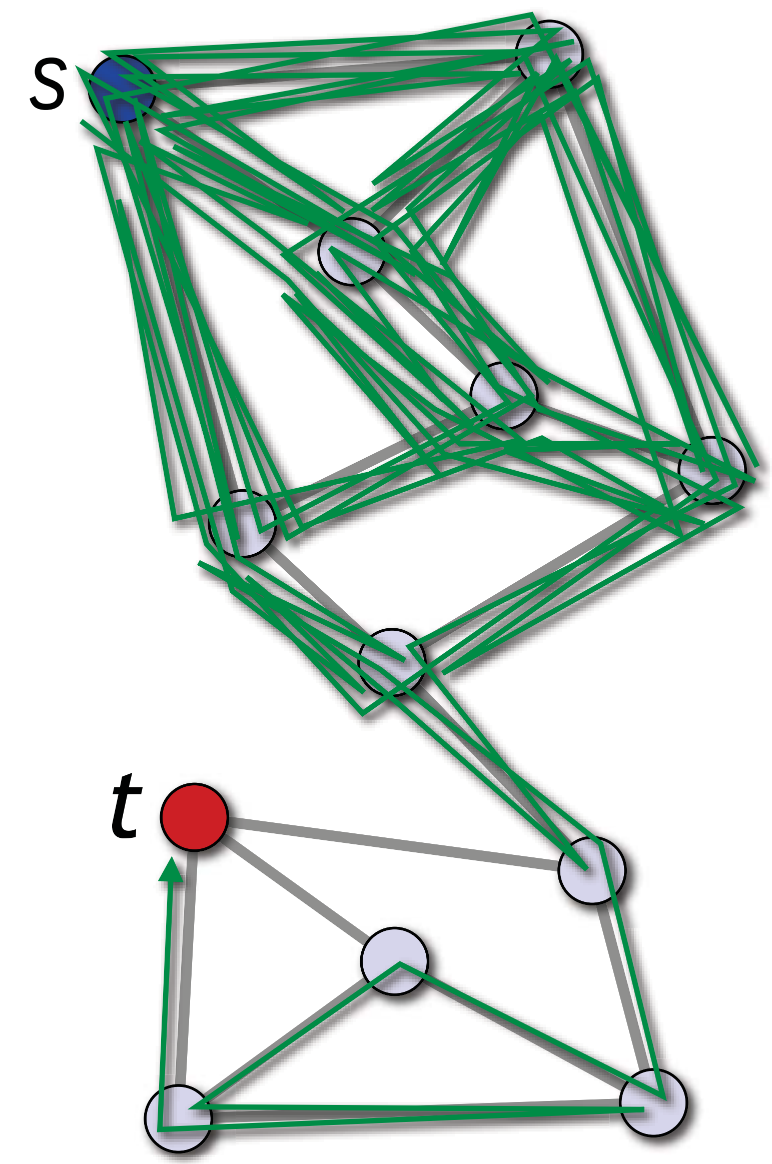
Graph exploration: random walks

- Simple solution with randomness:
  - just take a **random walk**
  - might take a while…



Graph exploration: random walks

- Simple solution with randomness:
  - just take a **random walk**
  - might take a while…
  - but eventually we will stumble on the treasure
  - expected time: poly(n)
    - see references on course web page



Graph exploration: random walks

- Random walk = **sequence of port numbers**
  - any such sequence can be applied in any regular graph!
- Expected time from s to any v and back is $O(n^2)$
  - here we assume that graph is d-regular and $d = O(1)$
  - proof: e.g., Motwani–Raghavan (1995), Section 6.4

Graph exploration: random walks

- Random walk = **sequence of port numbers**
  - any such sequence can be applied in any regular graph!
- Expected time from s to any v and back is $O(n^2)$
- Take a random walk w of length $\Theta(n^2)$ and apply it to any $O(1)$-regular graph G:
  - for any v, walk w fails to visit v with probability < 1/4
  - Markov's inequality

Graph exploration: random walks

- Random walk = **sequence of port numbers**
  - any such sequence can be applied in any regular graph!
- Expected time from s to any v and back is $O(n^2)$
- Take a random walk w of length $\Theta(n^2)$ and apply it to any $O(1)$-regular graph G:
  - for any v, walk w fails to visit v with probability < 1/4
- Take log x **consecutive walks** of length $\Theta(n^2)$:
  - failure probability $< (1/4)^{\log x} = 1/(x^2)$

Graph exploration: random walks

- Take log x **consecutive walks** of length $\Theta(n^2)$:
  - failure probability $< (1/4)^{\log x} = 1/(x^2)$
- Let x = number of possible choices (G, v)
  - number of d-regular port-numbered graphs with at most n nodes: $n^{O(n)}$
  - number of possible choices of v: $O(n)$
  - $\log x = O(n \log n)$

Graph exploration: random walks

- Take $\log x$ **consecutive walks** of length $\Theta(n^2)$:
  - failure probability $< (1/4)^{\log x} = 1/(x^2)$
- Let x = number of possible choices (G, v)
  - $\log x = O(n \log n)$
- Expected number of failures is $< x/(x^2) < 1$
  - failure = walk does not reach v in G
  - total length of walk = $O(n^3 \log n)$
- **There exists a walk that never fails for any *G*, *v*!**

Graph exploration: universal traversal sequences

- Therefore for every n, d there exists a **universal traversal sequence** w:
  - w consists of poly(n) port numbers
  - w always guides the robot from s to t in any d-regular port-numbered graph with at most n nodes
- This is completely deterministic!
  - e.g., choose the first w in lexicographic order
  - however, constructing w is not easy…

Graph exploration: universal traversal sequences

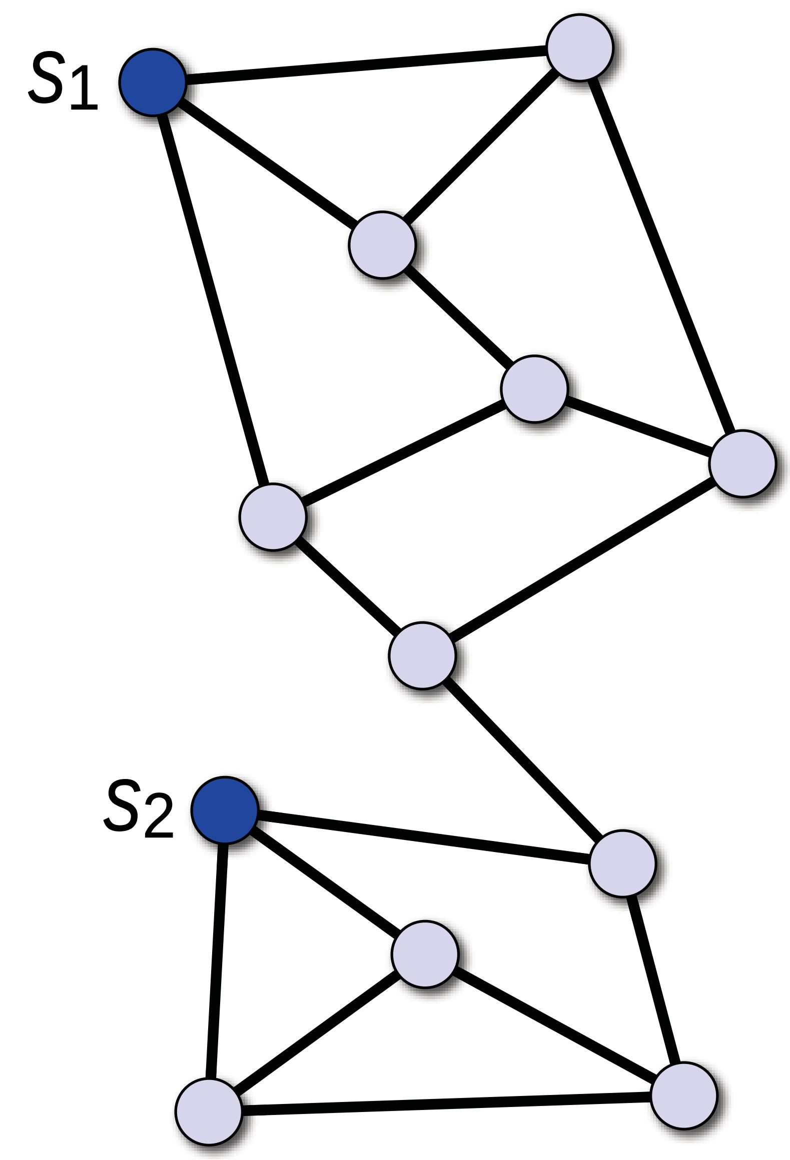
- Slightly simpler case:
  universal exploration sequence
  - next outgoing port depends on previous incoming port
- Omer Reingold (2005) showed how to construct universal exploration sequences efficiently
  - together with many other techniques, the paper shows that connectivity in undirected graphs can be solved by using deterministic log-space algorithms…

DDA 2010, lecture 6a: Meeting in a maze

- Dessmark et al. (2006):
  “Deterministic rendezvous in graphs”

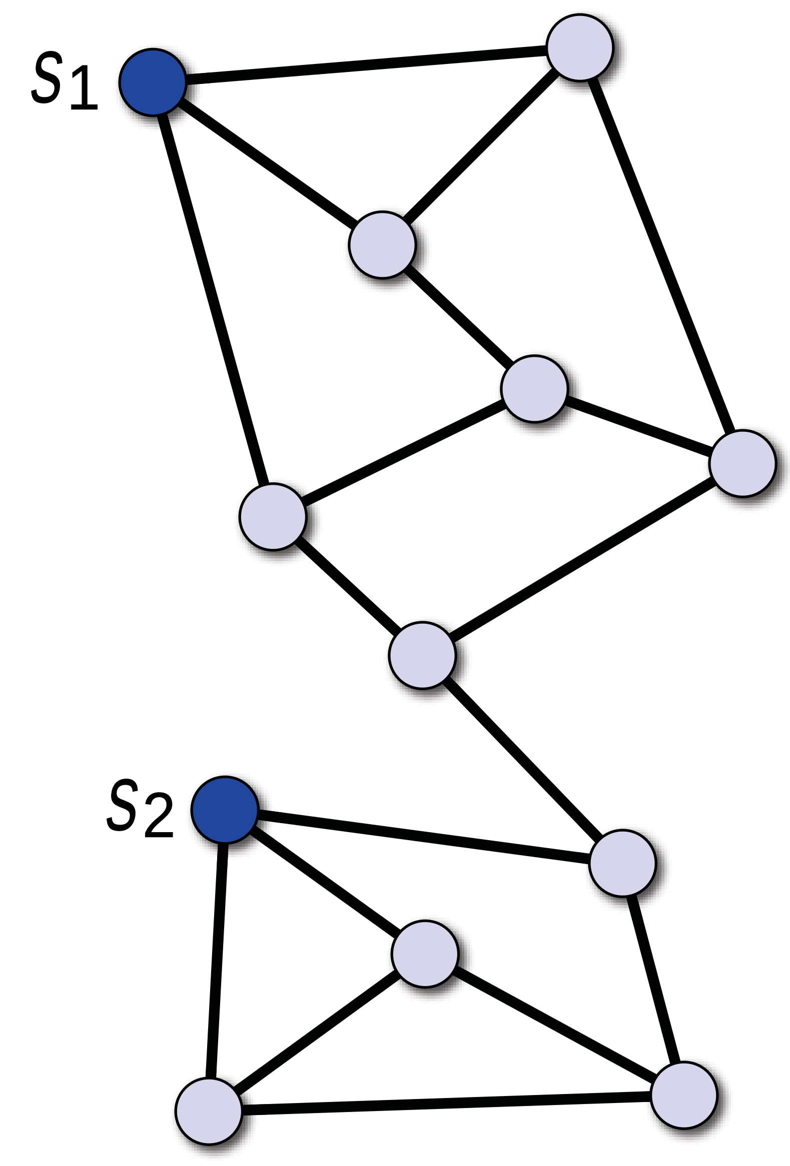
Rendezvous

- Connected graph with port-numbering
- **Two robots** placed in some nodes s_1, s_2
- Program the robots so that they will meet each other!



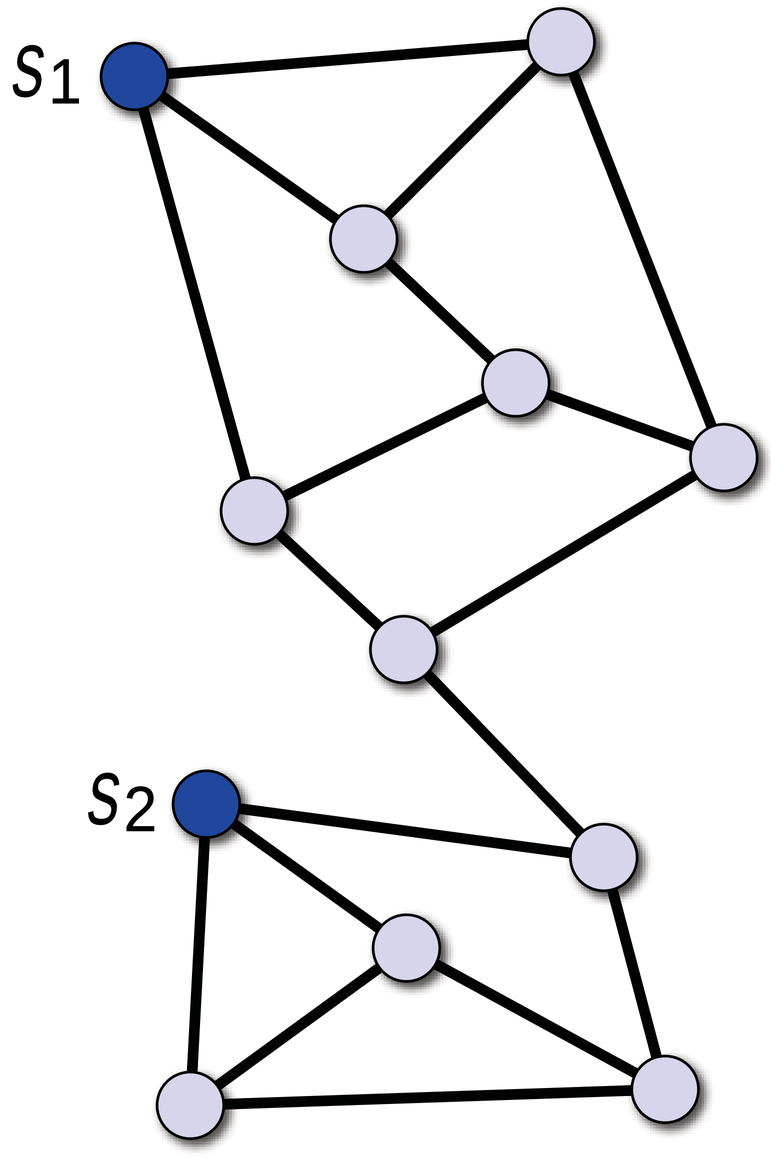
Rendezvous

- Identical robots:
  - not solvable by using a deterministic algorithm
  - counterexample:
    - symmetric cycle
    - both robots move in sync



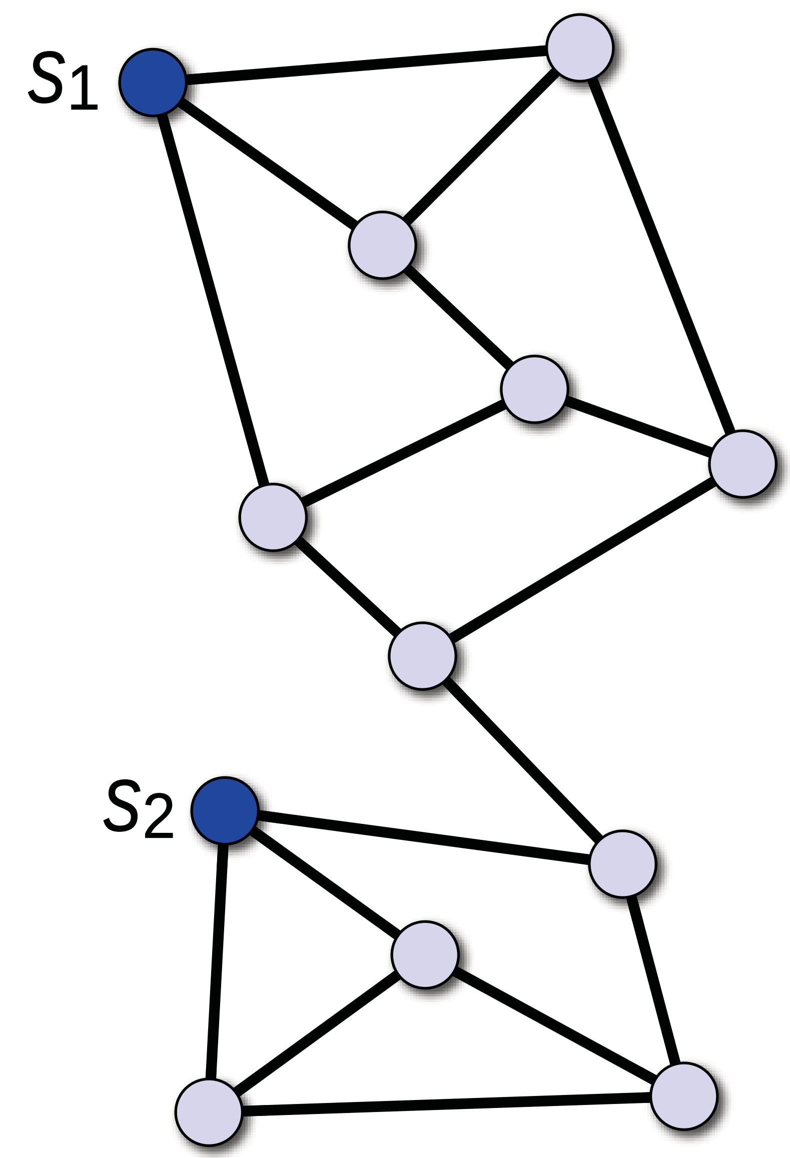
Rendezvous

- Robots with labels 1 and 2:
  - as easy as exploration
  - robot 1 explores
  - robot 2 stands still



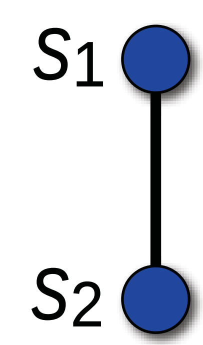
Rendezvous

- Robots with unknown unique labels L_1 and L_2:
  - can't choose which one waits and which one explores
  - random walks would solve the problem
  - but how to make it deterministic?



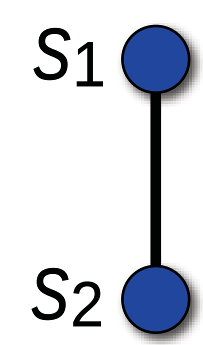
Rendezvous in K_2

- Robots with unknown unique labels L_1 and L_2
- Simplest special case: path with 2 nodes
  - bad if neither moves
  - bad if both move
- How to break symmetry using the labels?



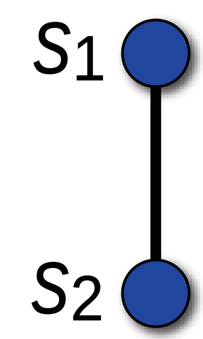
Rendezvous in K_2 – simple idea

- s_1 moves at time step L_1
- s_2 moves at time step L_2
  - will meet at time min $\{L_1, L_2\}$
- **Slow** if labels are large
- Requires **global time**!
  - assumes that robots are activated simultaneously



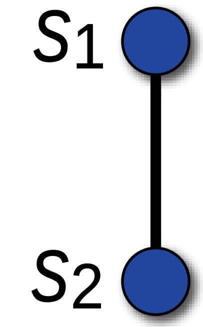
Rendezvous in K_2 – better idea

- Labels are bit strings (possibly different lengths)
  - agent with label $L_i = b_1 b_2 \ldots b_k$
    creates the string $X_i = 10 b_1 b_1 b_2 b_2 \ldots b_k b_k$
  - most significant bit 1, X_i begins 1011…
  - move according to X_i repeatedly:
    bit 1 = move, bit 0 = wait
- **Lemma**: $X_1 X_1$ cannot be a substring of $X_2 X_2 \ldots$, and vice versa



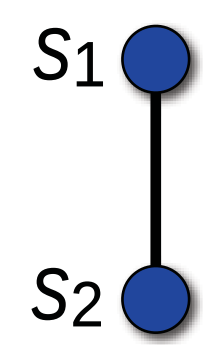
Rendezvous in K_2 – better idea

- **Lemma**: X_1X_1 cannot be a substring of $X_2X_2\ldots$, and vice versa
  - X_1 begins 101…
  - $X_2X_2\ldots$ contains …101… only at the beginning of each fragment X_2
- Same length, bit pairs differ:
  - $X_1X_1 = 1011aa\ldots bb\mathbf{00}cc\ldots dd1011\ldots$
    $X_2X_2 = 1011aa\ldots bb\mathbf{11}cc\ldots dd1011\ldots$



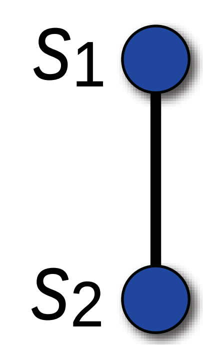
Rendezvous in K_2 – better idea

- **Lemma**: X_1X_1 cannot be a substring of $X_2X_2\ldots$, and vice versa
  - X_1 begins $101\ldots$
  - $X_2X_2\ldots$ contains $\ldots101\ldots$ only at the beginning of each fragment X_2
- Different lengths, boundary differs:
  - $X_1X_1 = 1011aa\ldots bb\mathbf{10}11\ldots$
    $X_2X_2 = 1011aa\ldots bb\mathbf{11}cc\ldots dd1011\ldots$



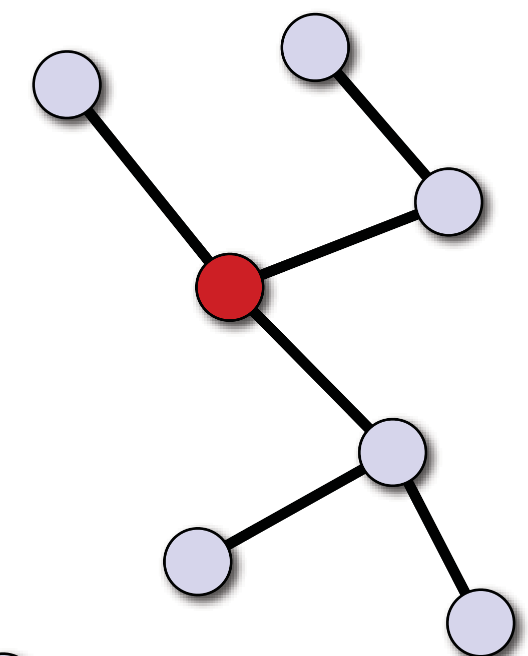
Rendezvous in K_2 – better idea

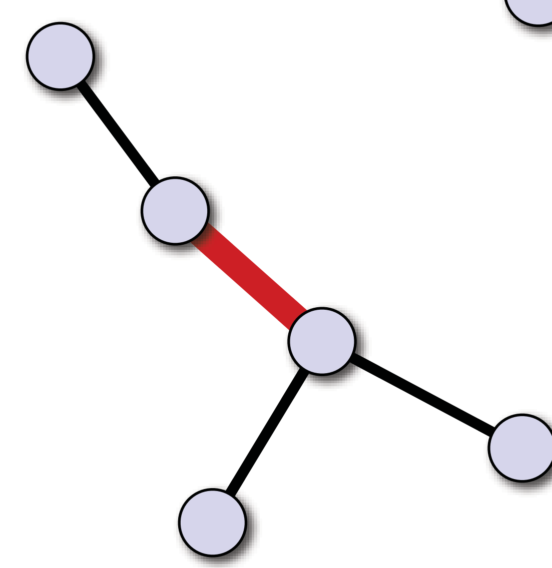
- **Lemma**: X_1X_1 cannot be a substring of $X_2X_2\ldots$, and vice versa
  - agents can't stay in sync forever
- **Corollary**: Even if s_1 and s_2 are activated at different times, they will meet after $O(\log l)$ rounds, where $l = \min\{L_1, L_2\}$



Rendezvous in trees

- First explore the tree
  - depth-first search,
    keep stack of port numbers
- There is a unique **central node** ***or*** **central edge** that minimises maximum distance to other nodes
  - central node: meet there
  - central edge: go to one endpoint,
    apply algorithm for K_2





Rendezvous in general graphs

- We have seen:
  - how to solve the case that labels are 1 and 2: treasure hunt
  - how to solve the case of arbitrary labels but simple graphs
  - similar ideas can be combined and generalised to arbitrary graphs
- Rendezvous can be solved using a deterministic algorithm in general graphs!

Rendezvous in general graphs

- What if we have more than 2 robots?
  - just pretend that we have the case of 2 robots
  - when any 2 robots with labels L_i and L_k meet, they form a group and then act as if they were one robot with label min $\{L_i, L_k\}$