Deterministic walks on graphs:

exploration and diffusion

Adrian Kosowski

INRIA Paris / LIAFA

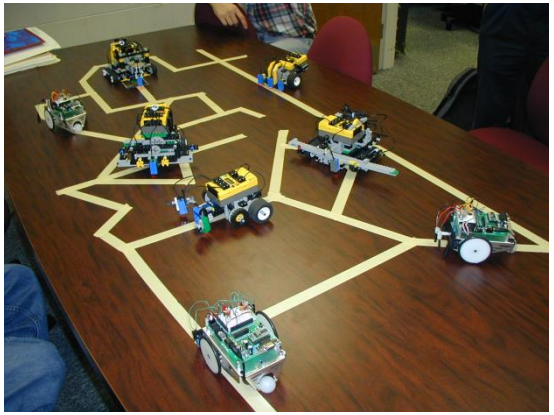
adrian.kosowski@inria.fr

Overview of the tutorial

- Introduction
- Graph exploration
  - The random walk and its variants
  - Deterministic walks
- Map construction in anonymous networks
- Diffusive load balancing
  - Continuous diffusion
  - The rotor-router model

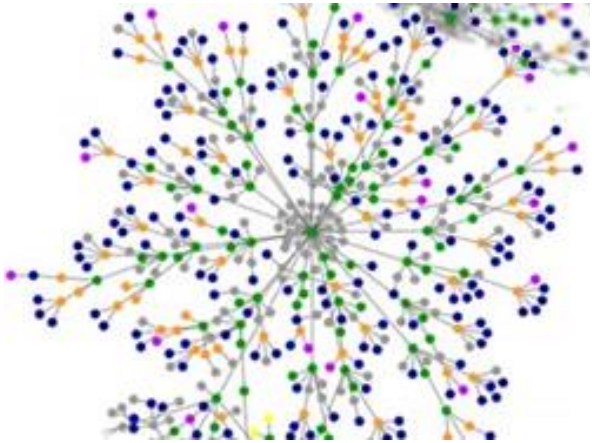
Where do we observe walks on networks?

Computational tasks involving "walkers" appear in diverse contexts:



Source: Georgia Tech

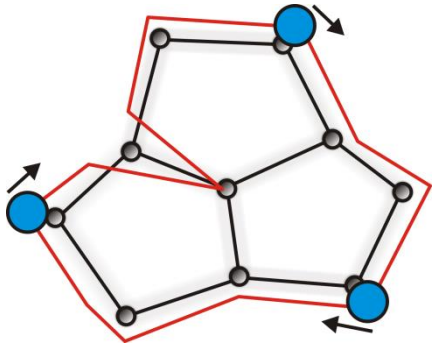
- **Mobile robots search for an exit from a labyrinth**
- **Biological agents hunt for prey**
- **Network crawler processes traverse hyperlinks**
  - GoogleBot web cache update
  - Facebook link prediction



- **Token circulation is at the heart of many distributed algorithms**

The theoretical framework

- An ***agent*** is a mobile unit hosted by a network node, capable of:
  - gathering local information about the environment
  - local computations at a node
  - migration to another node along a network link.



- The agent is equipped with ***state information*** (memory):
  - the state is modified only when the agent is located at network nodes
  - the state remains intact when traversing network links.

- Our task in Distributed Computing:

  Understanding the power and capabilities of agent-based algorithms.

Network Exploration

Network Exploration

The network exploration problem

- A walker is placed on some node of the input network
- The walker is allowed to traverse edges (links) of the network
- The goal is to visit all the nodes of the graph at least once

Objective: Minimize the ***cover time***

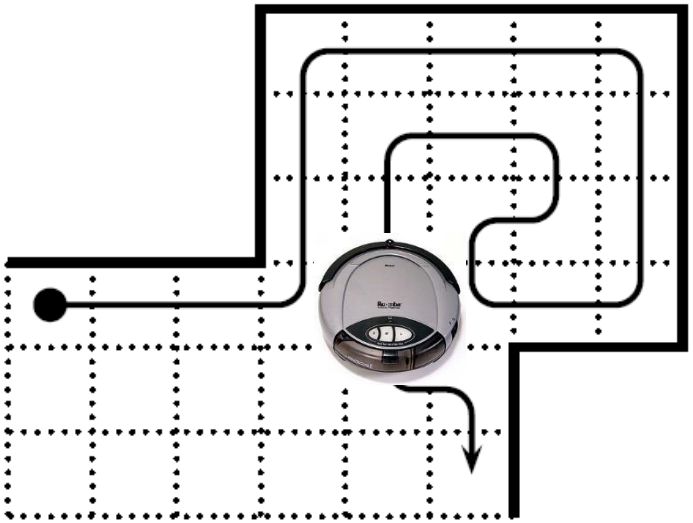
Complete the ”first exploration”
of all nodes as quickly as possible.

100 years of CS theory behind the problem
(automata on graphs, st-connectivity, L=SL,…)

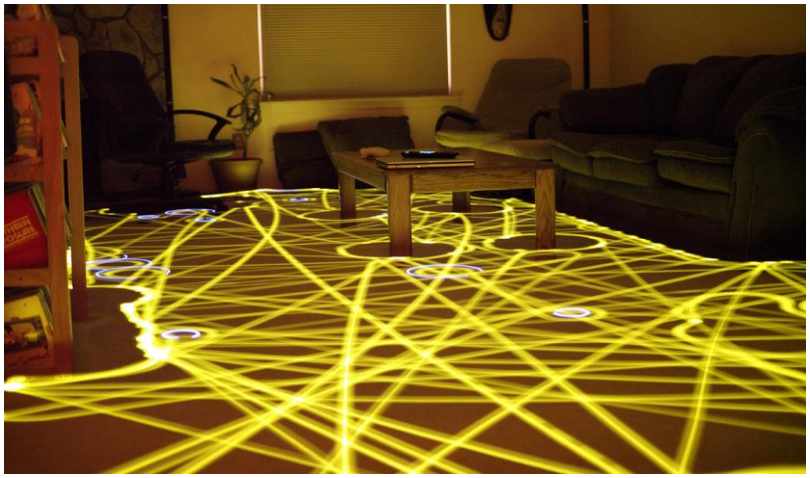
Example: How to deploy a robot vacuum cleaner?

Global knowledge
Global computation
Optimal solution

Local awareness
Restricted resources
Resilient solution



Traveling Salesman Problem



Random Walk

The local setting

The network:

- ***Think locally*: the global topology of the network is not known**
- The network may potentially change in time
- We may possibly know some global parameters
  - a bound on n – the number of network nodes
  - we may have a rough idea of the degree distribution in the graph
- Network links are undirected! (like Facebook)

The agent:

- For most of the time, we see the agent as a ”crawler process” (a bit like GoogleBot)
- When visiting a node, we learn its neighbors
  - this comes with a fixed cost
- Only following links is possible – teleportation is not allowed (e.g. because of no numeric node ID-s)

Computation on Anonymous Networks

Anonymity of nodes — computations performed by the agent cannot make use of any identifying information specific to the node at which the agent is located.

Motivation:

- Focuses our attention on mobile agent algorithms which behave "uniformly" over all nodes of the network.
- Profound implications in other areas: *log-space complexity theory, fault-tolerant routing, token distribution schemes…*
- Allows us to test limits of computability.

Anonymous Network Model [Yamashita & Kameda, 1996]

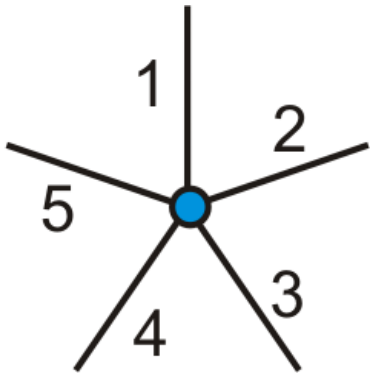
The Anonymous Network Model

- The explored graph $G = (V,E)$ is simple, undirected, and connected; $|V|=\boldsymbol{n}$, $|E|=\boldsymbol{m}$
- The nodes of the graph do not have any labels or colors which are known to the agent (anonymous graph property)

- The agent is an automaton with state memory
  - When located at a vertex, the agent can distinguish among the edges adjacent to the current node
  - The agent is aware of the edge by which it entered the current node
  - The agent is aware of a local port ordering at nodes.
  - ***No information may be written at nodes.***

The Anonymous Network Model

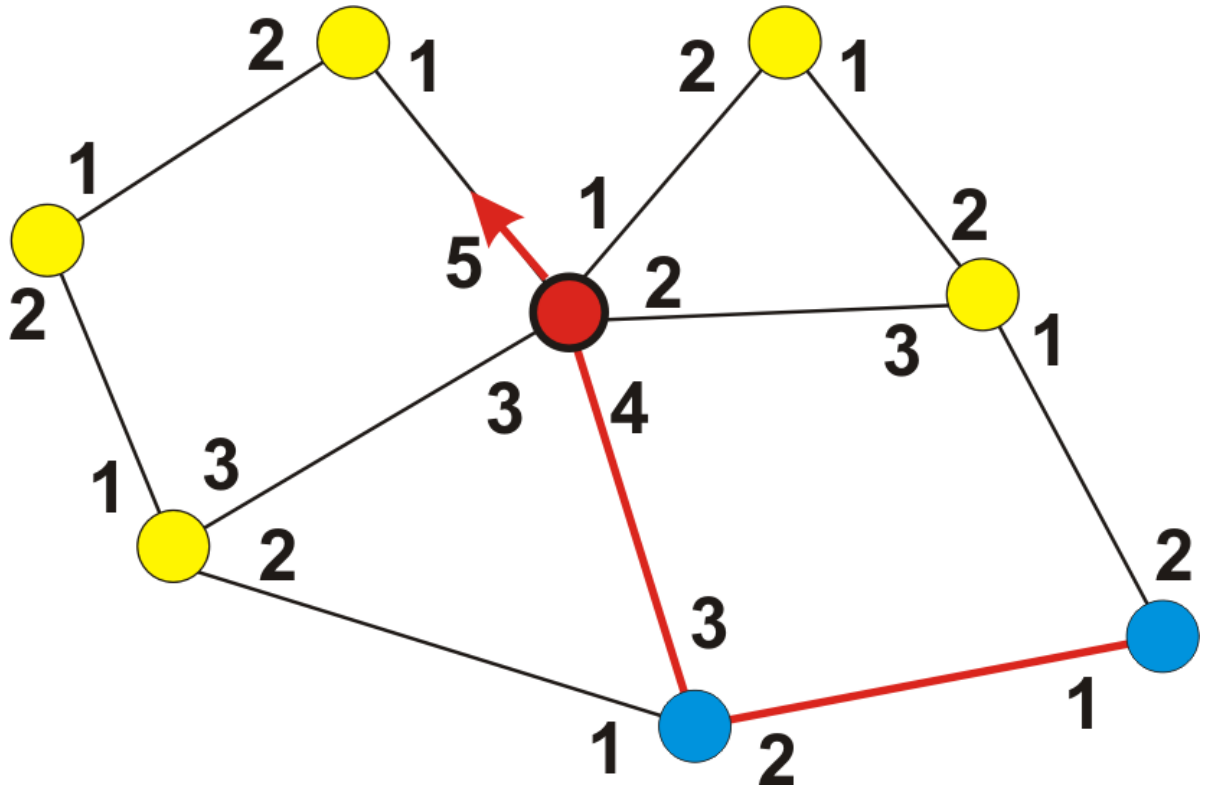
Example: The local view of an agent in the anonymous model



We assume throughout that in the anonymous model ports adjacent to each node are labeled with consecutive integers, from 1 to its degree.

Modeling network data

Example: The global perspective
(i.e., the anonymous network model, as seen by an external observer)



Some examples

Q1: We know that *G* is an anonymous tree. Design an agent which explores *G.*

- What is the cover time of the agent?
- What is the memory size of this agent?
- Does the algorithm need to know the number of nodes *n*?
- Does the problem get harder if we require the agent to stop after exploration?

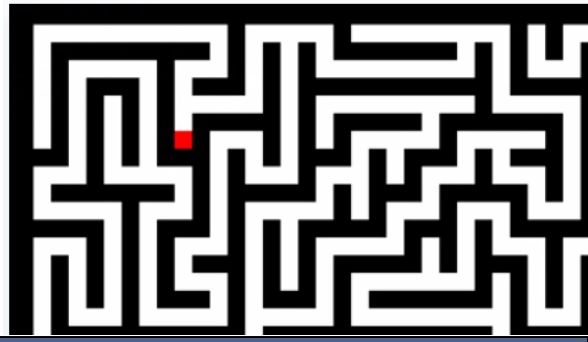
Some examples

Q2: Suppose you are lost in an unknown, dark maze (modeled by a graph on n nodes) and you wish to escape from it.

- How would you go about this, assuming all locations in the maze are numbered with unique identifiers from the set {1, 2, …, n}?
- How would you go about this, assuming all locations look exactly the same, but that you can tell by which port number you enter and leave a given location (=anonymous graph model).

In both cases, try to bound the **time** and **space** required by the algorithm (agent) in terms of n.

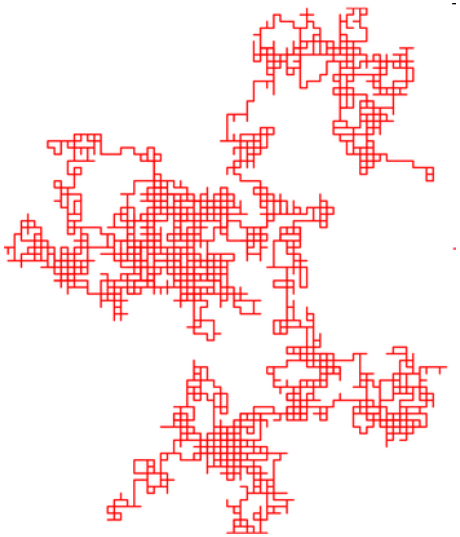
Note: You should not be assuming anything about the structure of the graph. Would it make life easier if we knew the graph was a subgraph of the grid, with a sense of direction (N-S-E-W), as shown in the figure?

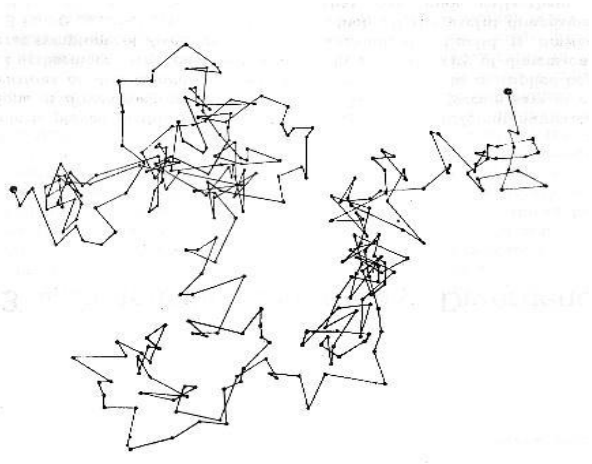


The Random Walk

The random walk

- The walker is placed at any node of an unknown network $G = (V,E)$…
- The walker leaves each node along one of the adjacent links, chosen **uniformly at random**
- The process is Markovian: the next step of exploration does not depend on the exploration history





Inspired by nature: **Brownian motion**

The random walk as an exploration strategy

Why use the random walk?

- Simple, resource-efficient, independent of network location
- Recovers quickly after a *non-adversarial* modification of the graph
- Covers web-type graphs quickly (in expectation)
- Covers grid-type graphs quickly (in expectation)

Disadvantages?

- Hopeless in a worst-case setting
- Does not learn anything from/about the environment
- Slow in some non-regular graphs
- Not always the best ”team strategy”

Where to use the random walk?

Classical networking applications:

- **Information search/packet circulation in p2p networks**
  – an alternative to flooding
  [Gkantsidis 2004]

- **Sampling of nodes in a web or social network**
  [Gjoka et al 2010]

- **Self-stabilizing mutual exclusion**
  (tokens following random walks meet and coalesce, until 1 remains)
  [Israeli-Jalfon 1990]

- **Picking a spanning tree of a graph** uniformly at random
  (applications of loop-erased walks)
  [Broder 1988, Wilson 1995]

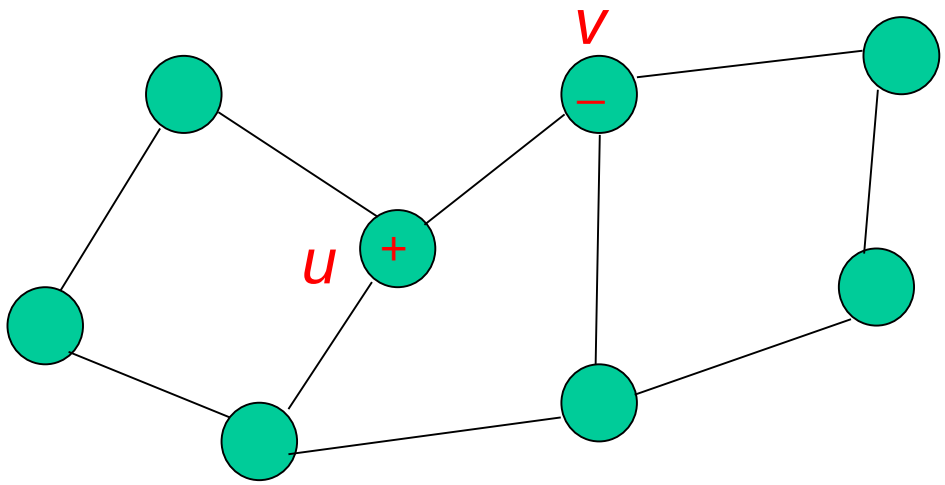
The random walk algorithm

How to analyze the random walk?

- First parameter: commute time $Com(u,v)$
  - What is the expected number of steps for a random walk to reach node v from node u, and then return to node v?
  - **Theorem** [Chandra, Raghavan, Ruzzo, Smolensky, Tiwari 1989]:

    $Com(u,v) = 2\, m\, R(u,v)$.
  - **Theorem** [Foster 1949]:

    $\sum_{\{u,v\} \in E} R(u,v) = n - 1$.



The random walk algorithm

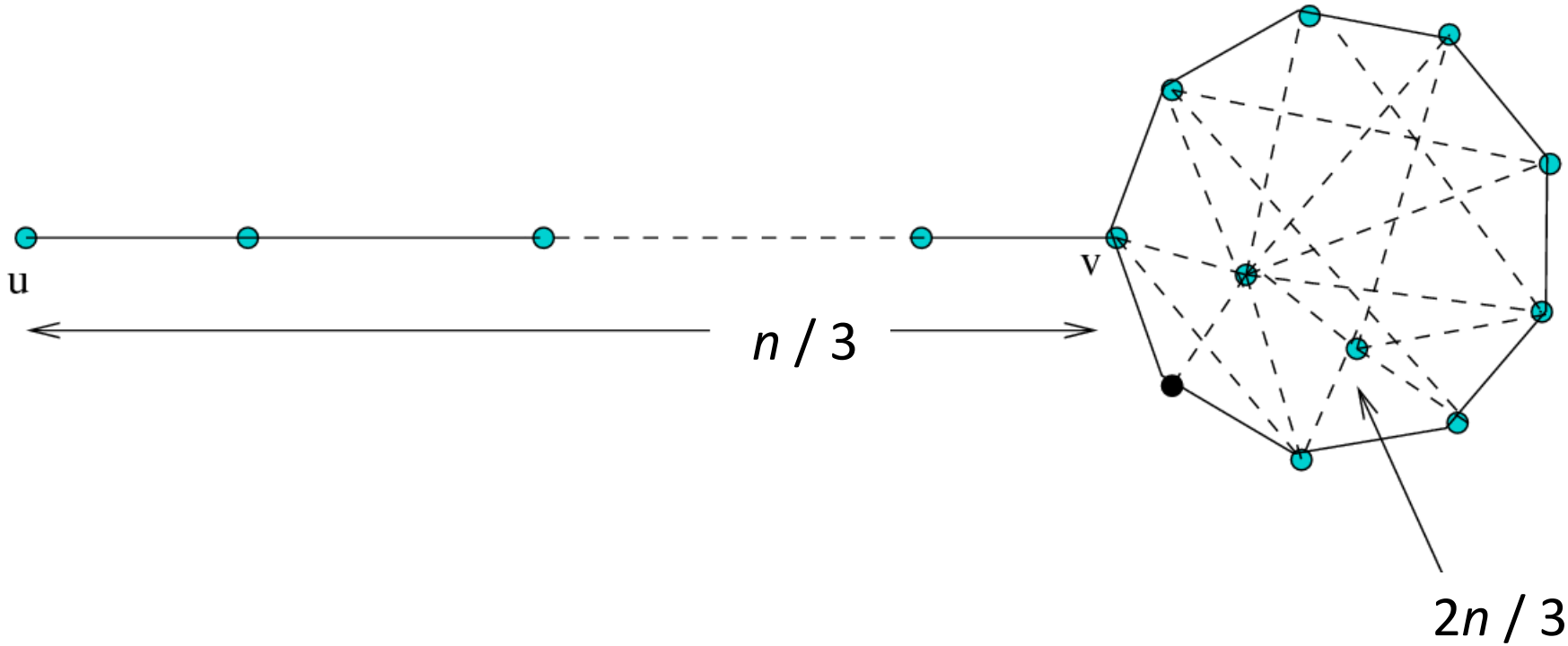
How to analyze the random walk?

- Second parameter: cover time
  - $Cov(u)$ - what is the expected number of steps for a random walk to reach all nodes of the graph, starting from node u?
  - $Cov = \max_{u \in V} Cov(u)$
  - **Theorem** [Aleliunas, Karp, Lipton, Lovasz, Rackoff 1979]
    Cover time is upper bounded by sum of commute times along the edges of any spanning tree of the graph.
  - **Theorem** [Feige 1995]
    By using the best spanning tree, we obtain for any graph:
    $Cov \leq 4n^3 / 27$
    - the bound is tight, and the worst-case example is precisely known

The random walk algorithm

How to analyze the random walk?

- The lollipop graph – worst-case cover time $4n^3 / 27 - o(n^3)$



The random walk algorithm

How to analyze the random walk?

Order of the cover time for different graph classes

- Cliques $n \log n$
- Paths, cycles n^2
- 2-dimensional grids $n \log^2 n$
- 3-dimensional grids $n \log n$
- Complete k-ary trees $n \log^2 n / \log k$
- Expanders $n \log n$
- Regular graphs not more than n^2

The random walk algorithm

Additional properties: how ”regular” is the random walk?

- In the limit, the random walk visits all edges with the same frequency ($1/|E|$)
- If the graph is not bipartite, then in the limit, at any given moment: The probability of finding ourselves at a vertex v is proportional to the degree of v.
- Rate of convergence to regularity: blanket time B
  - Intuitively, what is the expected number of steps of a random walk before all edges of the graph have been visited a similar number of times?
  - **Theorem** [Ding, Lee, Peres 2011]

    $Cov \leq B \leq \text{const} * Cov$

Tweaking the random walk…

Disadvantages of the random walk…

- *Completely useless in terms of worst-case performance*

  Partial remedy: use a deterministic strategy instead (*n^2 time overhead) **[TBD]**

- *Expected cover time of $\Theta(n^3)$ for some weakly connected graphs*

  Partial remedy: use Metropolis-Hastings biasing **[TBD]**

- *Short walks may get stuck in local network neighborhoods*

  More precisely: a random walk of small length *t* is expected to visit about √t edges [Broder et al. 1994], but may possibly visit very few nodes

  Partial remedy: use Metropolis-Hastings biasing **[TBD]**

Biased walks

and the Metropolis algorithm

Biased walks

- A biased walk is one in which the next node is chosen by the walker from among its neighbors, but transition probabilities need not be equal.
- The bias can be:
  - Topological – based on the structure of the graph, degrees/importance of nodes, etc.,
  - Dependant on exploration history – e.g. walks which never back-track to the node they have just come from
- In general, we want to keep the process (almost) reversible Markovian
- A simple way to obtain the desired form of bias (Markovian, reversible):
  - put positive real-valued weights on edges
  - at each step, choose an incident edge with probability proportional to its weight (relative to the sum of all weights of incident edges)

The probability of the agent being on an edge with weight w is $\mathbf{w / \sum_{e \in E} w(e)}$.

Metropolis walks

- There have been several recent papers showing how to bias random walks, given helper information about the topology of the graph, etc.
- The effort required to collect this information means that effectively a "normal" walker needs to do $\Theta(n^3)$ steps, anyway.
- There is an exception: the Metropolis-Hastings walk weighted by node degrees [Metropolis 1959, Nonaka et al. 2010]
- **Construction of the Markovian process:**
  - For each edge connecting *u* and *v,* put the following weight on it:

$$w(\{u,v\}) = \min \{ 1 / \deg(u),\ 1 / \deg(v) \}$$

  - Add self-loops at each node, so that the sum of weights of all incident edges sums to 1, for all nodes.
  - **Corollary.** In the Metropolis walk, all nodes are visited equally often during exploration (with the same limit probability of 1/n).

Metropolis walks

- There have been several recent papers showing how to bias random walks, given helper information about the topology of the graph, etc.
- The effort required to collect this information means that effectively a ”normal” walker needs to do $\Theta(n^3)$ steps, anyway.
- There is an exception: the Metropolis-Hastings walk weighted by node degrees [Metropolis 1959, Nonaka et al. 2010]
- **The walk can be implemented using an agent, as shown below:**

```
NextState (v: node)
    u <- neighbor of v in G chosen uniformly at random;
    move to u;
    with probability  max{1 - deg(v)/deg(u), 0} move back to v;
```

[Lee at al. 2012, K. 2013]

Metropolis walks

- The Metropolis walk has a worst-case performance superior to that of the random walk

A Metropolis walker explores a graph in $O(n^2 \log(n))$ steps w.h.p.

[Nonaka at al. 2010]

- **Note:** the random walk does not carry any state when traversing edges. **A little bit of memory** is necessary to implement Metropolis-Hastings.

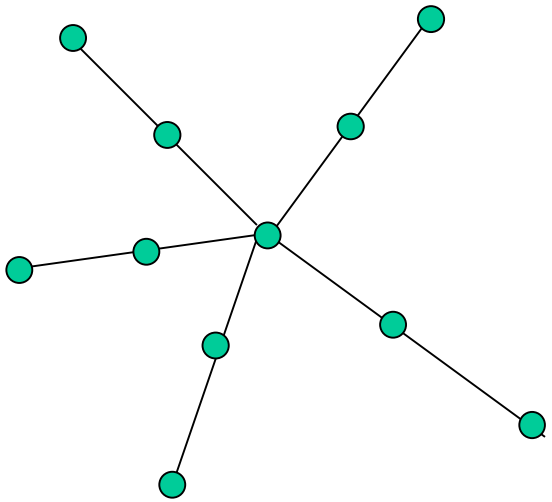
Any strategy with $o(n^3)$ cover time requires some state memory carried over edges.

A Metropolis walker can be implemented in O(log n) bits of memory.

Some questions

Q3: Consider the subdivided star graph on n = 2k + 1 nodes.

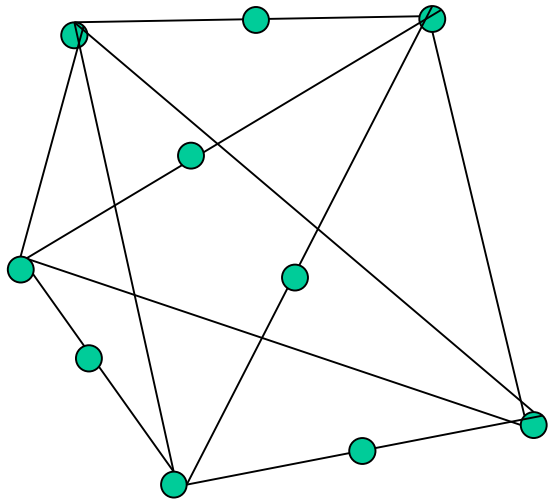
- Estimate (up to multiplicative constants) the cover time of the unweighted random walk on this graph.
- Estimate (up to multiplicative constants) the cover time of the Metropolis walk on this graph.



Some questions

Q4: Consider a modification of the k-node clique, such that on some k of its edges someone has inserted an additional node.

- Describe a weighting of edges of this graph such that the (vertex) cover time of the weighted random walk on this graph is as small as possible (up to multiplicative constants). What is this cover time?
- Can this weighted walk be implemented by a mobile agent without loss of time complexity? (And why not?)



Combining Metropolis walks and Random walks

- **Note:** the Metropolis walk is slower than the random walk on many graphs – even for the star.
- Is this strategy of practical importance?
  - **Yes. There are several elegant ways of combining the Metropolis walk with the random walk.**

A variant of the Metropolis walk explores all graphs $O(n^2 \log(n))$ steps w.h.p., and not more slowly (up to a factor of 2) than the random walk.

Combining Metropolis walks and Random walks

- **Note:** the Metropolis walk is slower than the random walk on many graphs – even for the star.
- Is this strategy of practical importance?
  - **Yes. There are several elegant ways of combining the Metropolis walk with the random walk.**

NextState (*v*: node)

u <- neighbor of *v* in *G* chosen uniformly at random;

move to *u*;

with probability $\max\{(\deg^{-1}(u)+d^{-1}) / (\deg^{-1}(v)+d^{-1}), 0\}$ **move back to** *v*;

  - The above method relies on knowledge of the average degree $d = 2m/n$. (can be done without.)

Short Metropolis walks

In expectation, a Metropolis walk of length $t > \Delta^2$ discovers at least $t^{1/2}$ nodes.

[K. 2013]

- Potentially useful property in local searches around network neighborhoods.

Metropolis walks

- Advantages?
  - Simple, resource-efficient, independent of network location
  - Equitable – uses all **nodes** fairly
  - Recovers quickly after a slight modification of the graph
  - Covers web-type graphs quickly in almost linear time
  - Expected cover time not worse than $\mathbf{O(n^2 \log n)}$
  - After some fine-tuning, short Metropolis walks visit nodes more quickly than short random walks

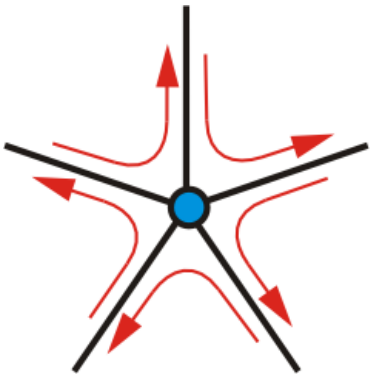
- Disadvantages?
  - Unbounded pessimistic cover time
  - In practical scenarios, a little slower than the Random Walk

Deterministic Network Exploration

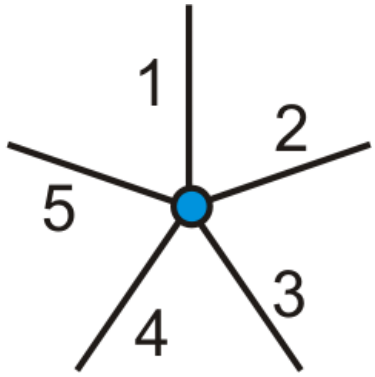
The labeled graph model

Assumptions of the labeled graph model

- The explored graph $G = (V,E)$ is simple, undirected, and connected
- The nodes of the graph do not have any labels or colors which are known to the agent (anonymous graph property)
  - When located at a vertex, the agent can distinguish among the edges adjacent to the current node
  - The agent is aware of the edge by which it entered the current node
- There are two distinct types of local orientations of edges at a node:



implicit cyclic ordering



explicit port labeling

De-randomizing random walks

How to make the random walk deterministic?

- We perform an exploration using a robot equipped with some memory (state) and knowledge of the ports in the graph:

  f (STATE, IN-PORT, DEGREE) = (STATE', OUT-PORT)

- The following properties are extremely desirable:
  - The number of states of the robot should be as small as possible
  - The worst-case cover time of the robot should be polynomial
  - If possible, other properties should be retained (e.g. equity of edge visits)
- First variant: we assume nothing about the port labeling of the graph (i.e., worst case labeling)
- First approach: sequences of port numbers that work for any graph…

Universal Sequences

Universal Traversal Sequences (UTS-s)

- A UTS(n,d) is a sequence of numbers (t_1… t_k) in 1..d, such that the robot
  f (STEP$_i$, PORT ?, DEGREE d) = (STEP$_{i+1}$, PORT t_i)
  covers any d-regular graph of (at most) n vertices in at most k steps.
- **Theorem** [Aleliunas, Karp, Lipton, Lovasz, Rackoff 1979]
  For any n, there exists a UTS(n,d) of length $k \leq n^5 \log n$
- *Proof: the probabilistic method*

  Intuition:

  If a uniformly random sequence of K moves in a graph explores any graph from graph class $\mathcal{G}$ with probability at least p,

  then:

  There exists a fixed sequence of K moves which explores at least $p\,|\mathcal{G}|$ graphs from $\mathcal{G}$.

Universal Sequences

How much memory is required to construct a UTS efficiently?

- Nisan's generic derandomizer (1992): $O(\log^2 n)$ memory
  - but the length of the sequence is no longer polynomial – $O(n^{\log n})$
- Not clear even if a sequence of polynomial length can be constructed in polynomial time…
- Some explicit constructions are known, e.g. for cycles…
- It turns out that it is easier to apply UXS-s instead!

Universal Exploration Sequences (UXS-s)

- A UXS(n,d) is a sequence of numbers (x_1… x_k) in 1..d, such that the robot
  f (STEP$_i$, PORT p, DEGREE d) = (STEP$_{i+1}$, PORT [$p + x_i$])
  covers any d-regular graph of (at most) n vertices in at most k steps.

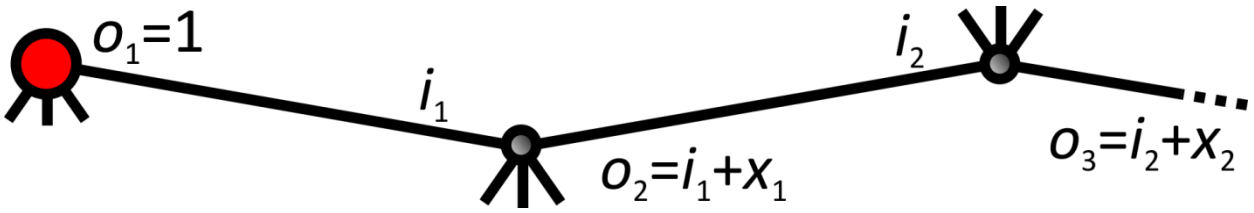
Deterministic Network Exploration Algorithms

- Deterministic exploration algorithms are obtained by de-randomizing randomized exploration algorithms.

- **Universal Exploration Sequences (UXS)** [Aleliunas et al. 1979, Koucky 2001]

  The walk is defined by a sequence of port increments $[x_1, x_2, x_3, \ldots, x_T]$

  in step t: $\boldsymbol{OUTPORT_{t+1} = INPORT_t + x_t}$ (modulo the degree of the node)



f (STEP$_t$, PORT p, DEGREE d) = (STEP$_{t+1}$, PORT [$p + x_t$] mod d)

For all n, there exists a UXS which explores all graphs of at most n nodes in T = **O(n^5 log n) steps.** [= n^3 cover time of Random Walk × n^2 log n]

Deterministic Network Exploration Algorithms

- Deterministic exploration algorithms are obtained by de-randomizing randomized exploration algorithms.

- **Universal Tables**

  An extension of the concept of universal sequences for de-randomizing **stateful** (non-Markovian) randomized algorithms. [Also extends Dessmark et al. 2004]

  Theorem. For all n, there exists a Universal Table which explores all graphs of at most n nodes, in T = **$O(n^4 \log^2 n)$ steps.**

  [= $n^2 \log n$ cover time of Metropolis Walk × $n^2 \log n$]

  - Universal Tables are shorter than UXS-s, but do not easily allow the agent to revert to its initial location.
  - Universal Tables / UXS-s can be implemented in an agent with **$O(\log n)$ bits** of state memory**.**

Overview of Network Exploration Algorithms

Agent's Algorithm	State memory	Cover time	Reference
Random Walk:	none	$O(n^3)$ (expectation)	[Aleliunas et al. 1979]
Metropolis Walk:	$O(\log \log n)$	$O(n^2 \log n)$ (expectation)	[Nonaka et al. 2010, K. 13]
Universal Table:	$O(\log n)$	$O(n^4 \log^2 n)$	[Aleliunas et al. 1979, Nonaka et al. 2010]

Randomized approaches:

- Fast and memory-efficient
- No knowledge of global parameters is required
- Process termination is problematic.

Deterministic approaches:

- Lower bounds: $\Omega(n^2)$ steps, $\Omega(\log n)$ memory bits [Borodin et al. 1992, Fraigniaud et al. 2005]
- Termination is possible, given upper bound on n.

Touching the foundations of computer science

The L = SL complexity class problem

- L is the complexity class containing decision problems which can be solved by a deterministic Turing machine using a logarithmic amount of memory space.
- SL (Symmetric Logspace) is the complexity class of problems log-space reducible to USTCON (*undirected s-t connectivity*), which is the problem of determining whether two vertices of a graph are in the same connected component

A positive answer [Reingold, STOC 2005]

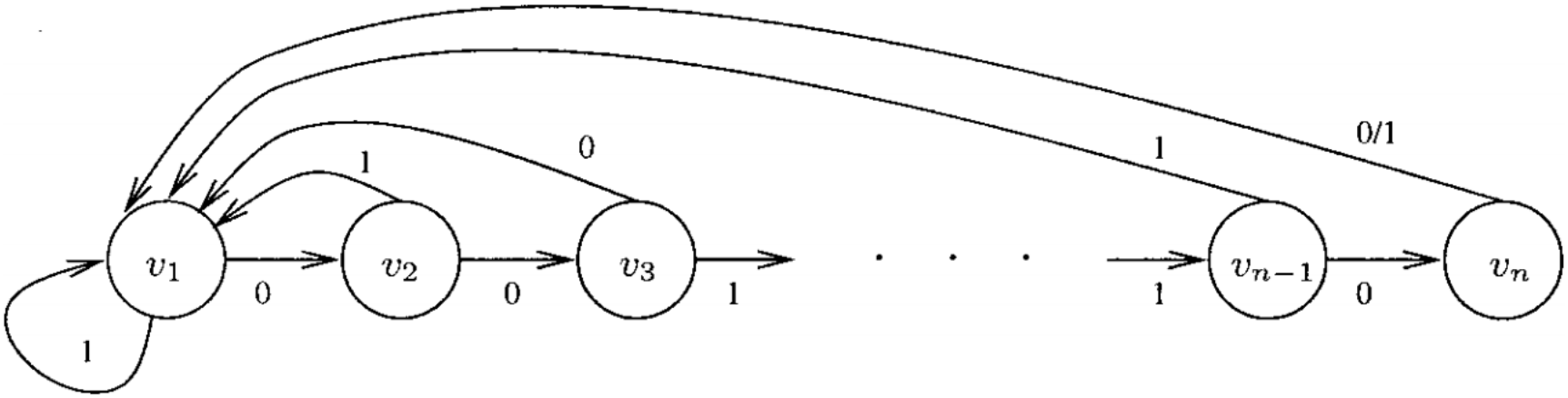
- UXS(n,d) can be ”printed” by a Turing machine with O(log n) memory
- By applying a slight modification of the sequence, we can explore any (not necessarily regular) graph of order at most n, thus solving USTCON.
  - Note: the problem for the related oblivious (UTS-based) variant is open!

Notes on other graph exploration models

- Models with whiteboards
  - The agent can write and read information on designated memory areas at nodes, known as **whiteboards**.
  - For exploration problems, it is sufficient to be able to mark a node as ``already visited''; this allows us to implement DFS (in $\boldsymbol{2m}$ steps).
  - A refined algorithm performs in $\boldsymbol{m+O(n)}$ steps. [Pelc and Panaite, 1999]
- Models with pebbles
  - The agent can drop a marker (pebble, token) at some node. If the pebble is moveable, it can move it to an adjacent node. The number of available pebbles is limited.
  - With one moveable pebble, deterministic exploration and identification of G can be performed in $\boldsymbol{O(mn)}$ time, *without knowledge of any upper bound on n* [Dudek et al., 1997]
  - Allowing the agent to put a pebble on its starting location is also sufficient, but the time is then $\boldsymbol{O(n^3\Delta)}$. [Chalopin, K., Das 2010]

Exploration of directed graphs

- We consider strongly connected, but not necessarily symmetric, directed graphs
- There exist directed graphs for which exploration requires exponential time (even for randomized algorithms, in expectation)
- Lower bound for random-walk type algorithms: the **combination lock** graph
  [Bender & Slonim 1994; figure from Bender et al. 2002]



Exploration of directed graphs

- We consider strongly connected, but not necessarily symmetric, directed graphs
- There exist directed graphs for which exploration requires exponential time (even for randomized algorithms, in expectation)
- Lower bound for random-walk type algorithms: the **combination lock** graph [Bender & Slonim 1994; figure from Bender et al. 2002]
- Allowing the agent to use pebbles reduces exploration time [Bender et al. 2002]
  - A single pebble: exploration and identification in polynomial time (roughly n^8), but knowledge of an upper bound on n is required
  - O(log log n) pebbles: exploration and identification in polynomial time without knowledge of n.

Map construction

Map Construction

The map construction problem

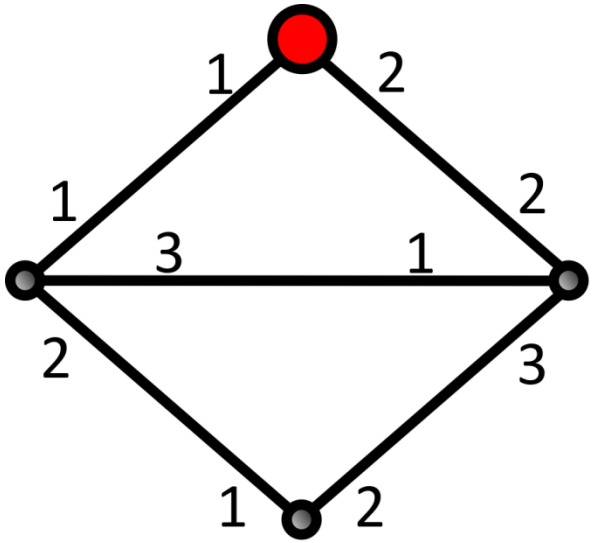
- A mobile agent is placed at some node of the input graph.
- The agent moves around the graph, traversing its edges, and collecting all possible information about the topology of the graph.
- The agent constructs a ”map” which encodes all of this information.

Why construct a map of an *anonymous* graph?

- Software agents for managing network resources may use information about the network topology to optimize their performance.
- Agents may need to break symmetries in order to, e.g., solve leader election or achieve rendezvous.

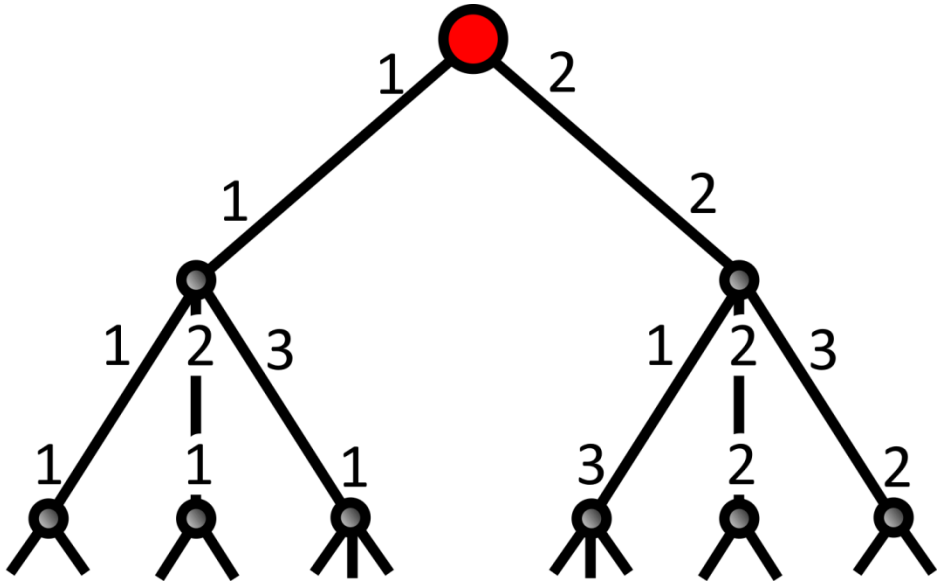
The simplest map: the View

What can an agent learn about the network?



explored graph

agent's view



All accessible information is encoded in the first *n* levels of the agent's view.

Constructing the view directly up to level *n* might require exponential time & memory…

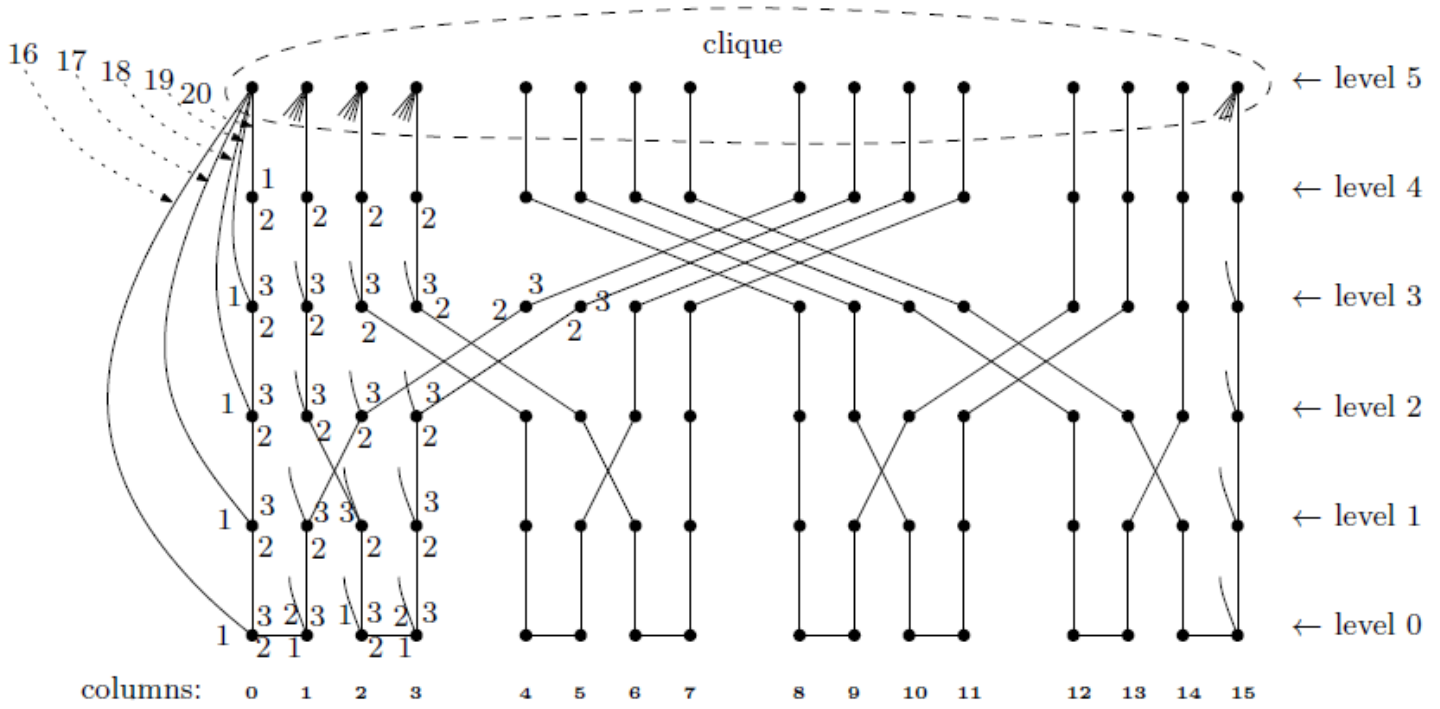
A question

Q5. For arbitrarily large n, show an example of an n-node labeled graph which contains two distinguished nodes u, v whose views are different, but identical up to as large a depth as possible.

(The best you can do is very close to n).

Distinguishability of views

Note: it is possible that for a pair of nodes the view is identical up to depth more than their diameter, but different at some greater depth!



In fact, for distinguishability of views, it is sufficient to consider views up to depth O(diam * log *n*). (Non-trivial) [Hendrickx'13]

A more compact map: the Quotient Graph

Assume that the agent starts at a node v in G.

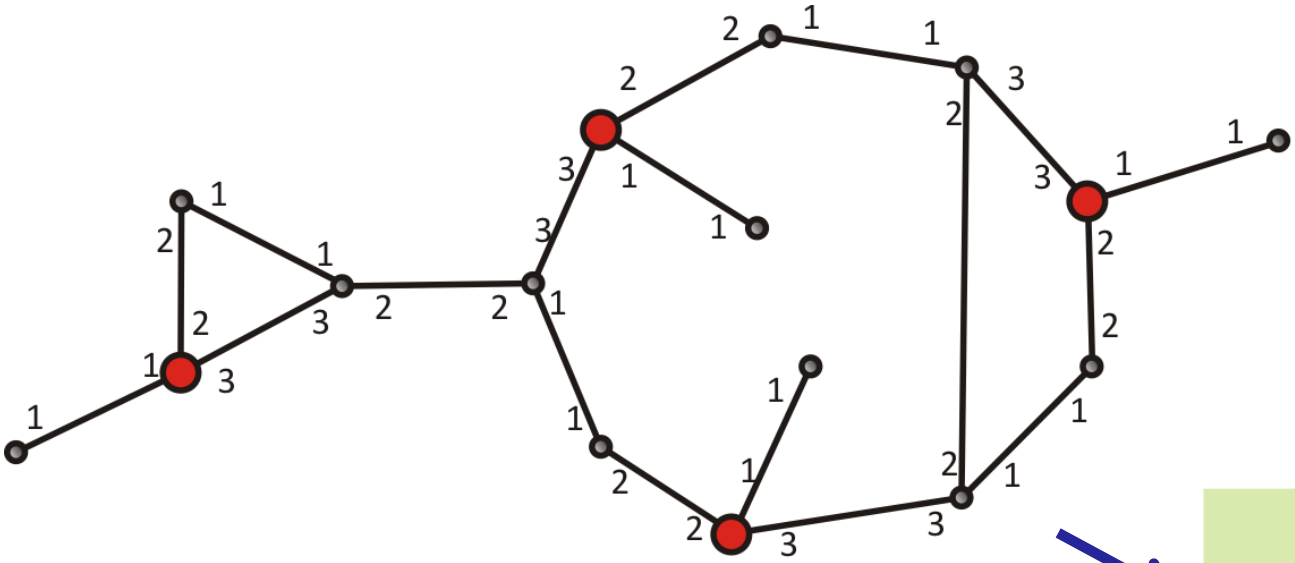
Quotient graph (H, v') of (G, v) : the (unique) smallest multigraph such that node v' in H has the same view as node v in G.

Theorem. The quotient graph exists & encodes the view of v in G.

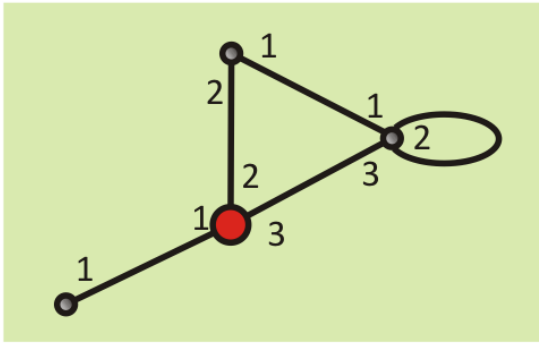
[Yamashita, Kameda '96]

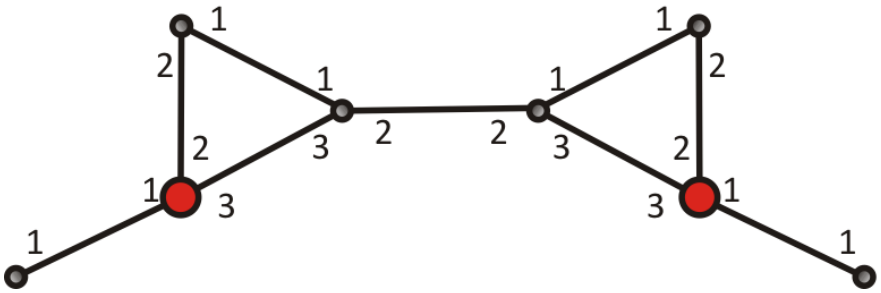
- **If two graphs** have identical quotient graphs, then no agent can tell these two graphs apart.
- **If two nodes** of a graph have identical quotient graphs, then no agent can tell these two nodes apart.

Distinguishing graphs: an example



different network graphs & locations





same quotient graph

Results on map construction (anonymous model, with knowledge of n)

Theorem. There exists an agent with **O(log n)** memory which resolves the following queries in any anonymous network G:

- What is the number $\boldsymbol{p}$ of nodes of the quotient graph of G?
- What is the identifier $v' \in \{1, \dots, p\}$ of the node of the quotient graph corresponding to the node v of G at which the agent is currently located?
- Given two nodes of the quotient graph H with identifiers $i, j \in \{1, \dots, p\}$, are they connected by an edge?

Proof relies on a variant of UXS.

Results on map construction (anonymous model, with knowledge of n)

Theorem. There exists an agent with **O(log n)** memory which resolves the following queries in any anonymous network G:

- What is the number $\boldsymbol{p}$ of nodes of the quotient graph of G?
- What is the identifier $v' \in \{1, \dots, p\}$ of the node of the quotient graph corresponding to the node v of G at which the agent is currently located?
- Given two nodes of the quotient graph H with identifiers $i, j \in \{1, \dots, p\}$, are they connected by an edge?

Proof relies on a variant of UXS.

Corollary. Any problem on a network G which can be formulated as a log-space Turing-decidable problem on the quotient graph of G, is decidable using a log-space agent on G.

(given knowledge of an upper bound on n)

Log-space computations with agents

Examples of problems decidable with log-space agents:

(given knowledge of an upper bound on n)

- *Decide if G is a tree.*
- *Locate an edge of G whose removal does not disconnect G.*
- *Given a graph G in which nodes have distinct views, perform "leader election" on the nodes of G.*

Q6. Problem: Provide an example of a graph G which contains an edge e, such that e belongs to some cycle in G, whereas the edge corresponding to e in the quotient graph of G does not belong to any cycle (or loop) of this quotient graph.
Draw both graph G and its quotient graph.

The case of trees…

Walking in tree networks

Exploration of trees

- Perpetual exploration can be trivially achieved, in **O(n) time** and **no memory**.
- Termination of exploration requires $\Omega(\log \log \log n)$ memory, or $\Omega(\log n)$ memory if the agent is required to stop at its starting location.

[Diks et al., 2004, Ambuhl et al. 2011]

Map construction & detecting if the network is a tree.

- For a tree, the quotient graph can be reconstructed in $O(\log n)$ memory, without knowledge of an upper bound on n.

Diffusive load balancing

Load balancing in a distributed system

Load balancing

- We are given a network whose nodes represent machines connected by communication links
- Tasks of fixed size arrive at the machines (in an adversarial scheduling)
- Machines use network links to spread out load over all network nodes

Objective: Minimize the number of communication rounds required to spread out the load fairly over the network.

Assumption: communication time << task processing time

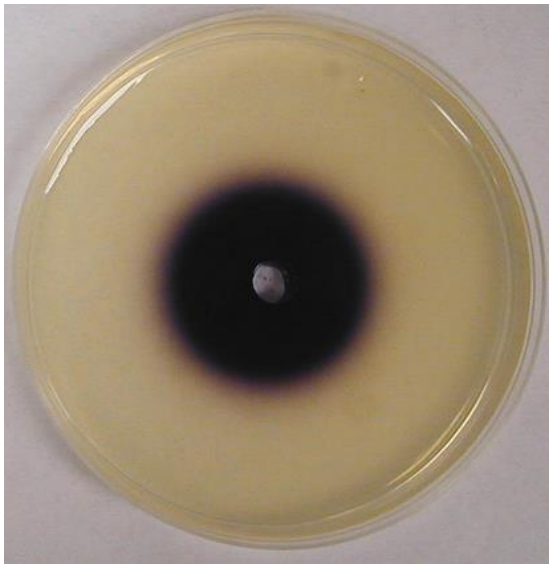
We think of the tasks as a form of token (or mobile agent).

Inspirations: The Diffusion Equation

Diffusion equation in R^k

$$\partial L / \partial t = \nabla (\alpha \nabla L)$$

- $L(x,y,z,t)$ is a scalar function (scalar field) – concentration of particles at points, temperature,…
- α is the diffusion coefficient
  - For constant α, we have uniform linear diffusion (heat equation)
- First formulation for diffusing particles [Fick, 1855]
  - $-\alpha \nabla L$ is the diffusion flux



Inspirations: The Diffusion Equation

Uniform diffusion equation in R^k

$$\partial L / \partial t = \nabla (\alpha \nabla L) = \alpha \Delta L$$

Next steps:

1. discretizing space: R^k -> graph with exchange between adjacent sites
2. we assume uniformity of diffusion coefficients on the sites of the graph
   - e.g. on the path: $\partial L_k / \partial t = \alpha (L_{k+1} + L_{k-1} - 2L_k) / 2$
3. discretizing time

Inspirations: The Diffusion Equation

Continuous Diffusion Model (in load balancing)

- Each of the nodes v of the graph starts with a certain amount of resource $L_0(v)$ (real-valued, non-negative) – call it **load**
- In each round, each of the nodes sends an equal part of its load to its neighbors

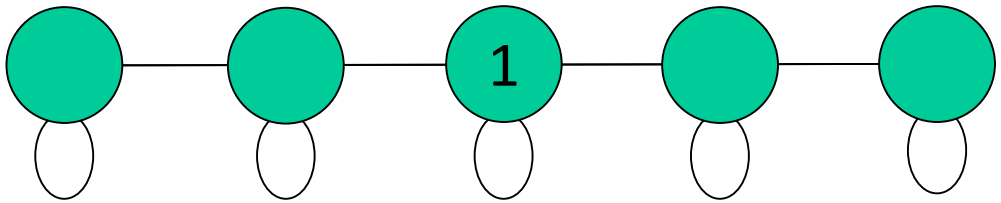
$$L_{t+1}(v) = \sum_{u \in N(v)} L_t(u) / \deg(u)$$

Inspirations: The Diffusion Equation

Continuous Diffusion Model (in load balancing)

- Each of the nodes v of the graph starts with a certain amount of resource $L_0(v)$ (real-valued, non-negative) – call it **load**
- In each round, each of the nodes sends an equal part of its load to its neighbors

$$L_{t+1}(v) = \sum_{u \in N(v)} L_t(u) / \deg(u)$$

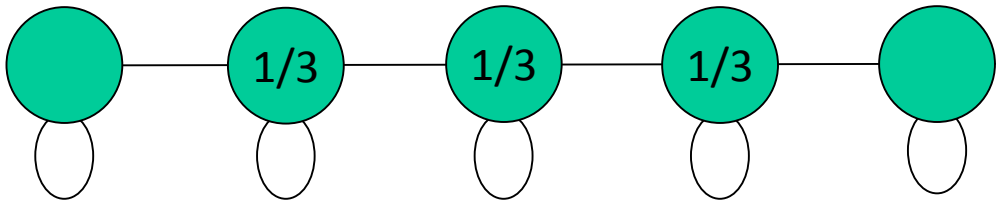


Inspirations: The Diffusion Equation

Continuous Diffusion Model (in load balancing)

- Each of the nodes v of the graph starts with a certain amount of resource $L_0(v)$ (real-valued, non-negative) – call it **load**
- In each round, each of the nodes sends an equal part of its load to its neighbors

$$L_{t+1}(v) = \sum_{u \in N(v)} L_t(u) / \deg(u)$$

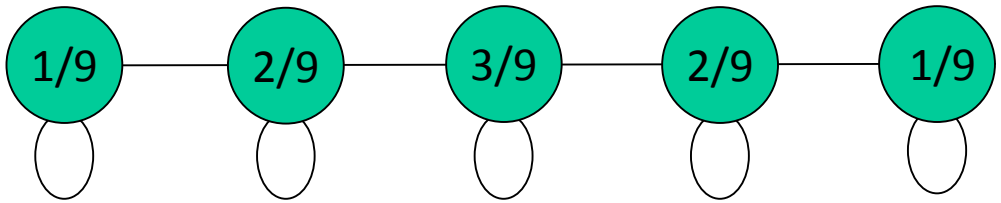


Inspirations: The Diffusion Equation

Continuous Diffusion Model (in load balancing)

- Each of the nodes v of the graph starts with a certain amount of resource $L_0(v)$ (real-valued, non-negative) – call it **load**
- In each round, each of the nodes sends an equal part of its load to its neighbors

$$L_{t+1}(v) = \sum_{u \in N(v)} L_t(u) / \deg(u)$$



[…]

Analyzing continuous diffusion

- Continuous diffusion is a linear and **deterministic (!)** process:

$$\mathbf{L}_{t+1} = \mathbf{M}\ \mathbf{L}_t \quad \Rightarrow \quad \mathbf{L}_t = \mathbf{M}^t\ \mathbf{L}_0$$

where **M** is the stochastic matrix (”random walk matrix”) of the graph.

- Two easy-to-handle time parameters:
  - After how many steps has some (non-zero) load reached the most distant nodes of the graph? O(diam)
  - After how many steps is load almost evenly spread throughout the graph? $O(\mu^{-1} \log (k\, n))$
    - μ – eigenvalue gap of the graph
    - k – total load (1-norm of L_0; total number of chips)

Analyzing continuous diffusion

- Continuous diffusion is a linear and **deterministic (!)** process:

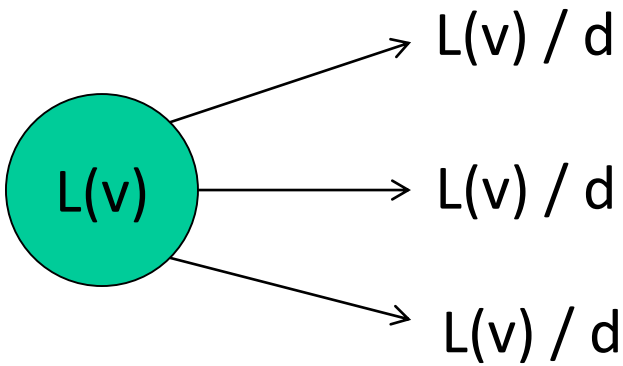
$$\mathbf{L}_{t+1} = \mathbf{M}\ \mathbf{L}_t \quad \Rightarrow \quad \mathbf{L}_t = \mathbf{M}^t\ \mathbf{L}_0$$

where **M** is the stochastic matrix (”random walk matrix”) of the graph.

- **Problem:** dealing with granular load (not infinitely divisable)
- Assume load is expressed in multiple of unit values, each of which is a chip propagated between neighboring nodes.
- We have *k* chips in total, each node v starting with $L_0(v)$ chips
- In general, it is no longer possible to follow the diffusion equation accurately.
- We discuss several discrete diffusion rules which **asymptotically converge** to continous diffusion.

Discrete diffusion rules

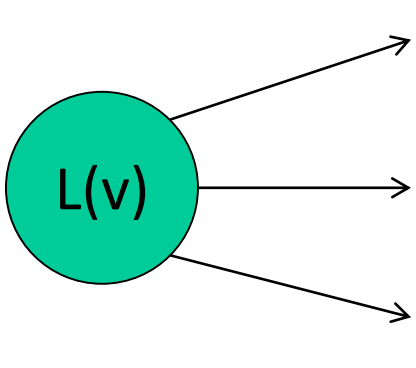
Reference point – continuous diffusion:



d = deg(v), for a while, we will be considering regular graphs.

Discrete diffusion rules

Rule 1: Independent random walk for each chip



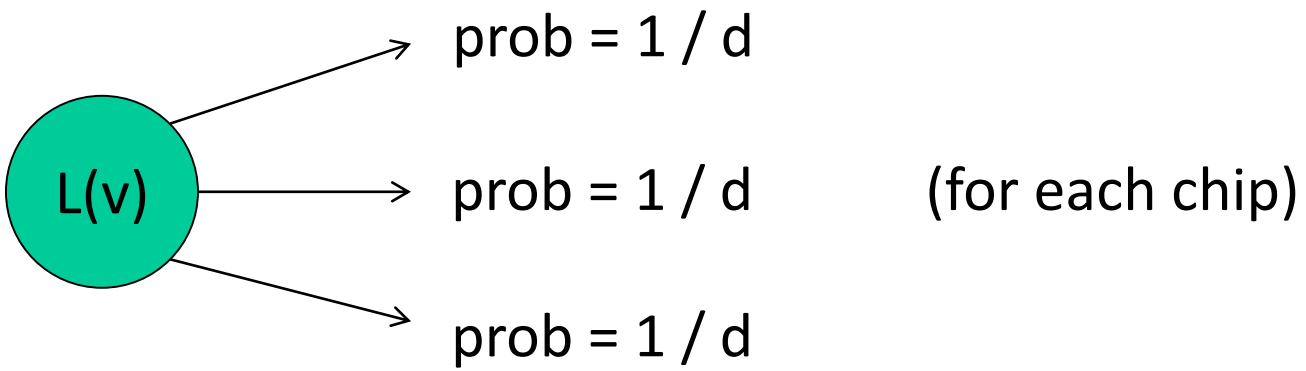
prob = 1 / d

prob = 1 / d (for each chip)

prob = 1 / d

Discrete diffusion rules

Rule 1: Independent random walk for each chip

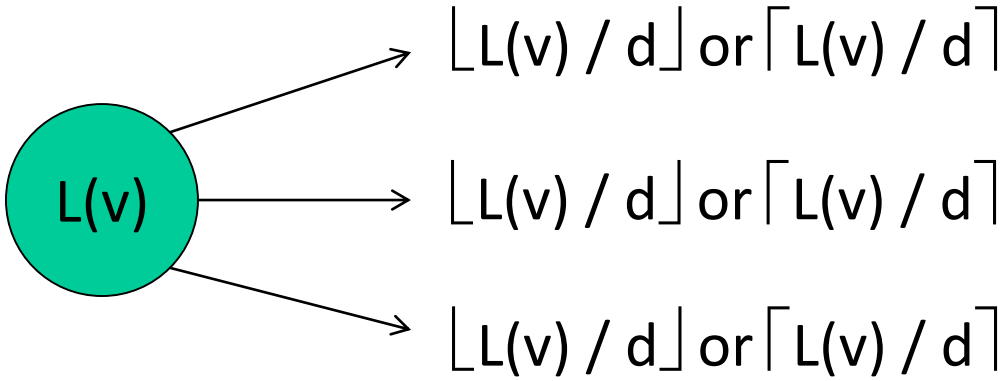


- Pros:
  - Expected number of chips at each location for the random walk matches that in continuous diffusion: $\mathbf{E}[\mathbf{L}_t] = \mathbf{M}^t \ \mathbf{E}[\mathbf{L}_0]$
- Cons:
  - Unboundedly bad worst-case behavior.

Let's look at some rules which do have worst-case guarantees…

Discrete diffusion rules

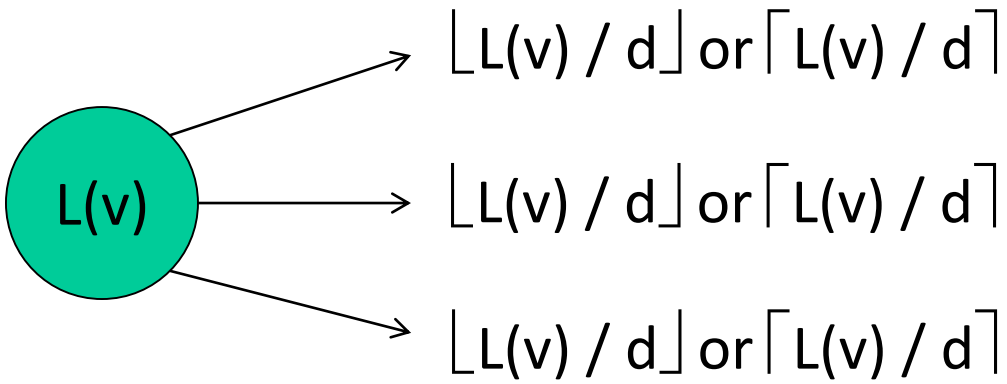
Rule 2: Perform rounding of the continuous diffusion process



(keeping sum intact)

Discrete diffusion rules

Rule 2: Perform rounding of the continuous diffusion process

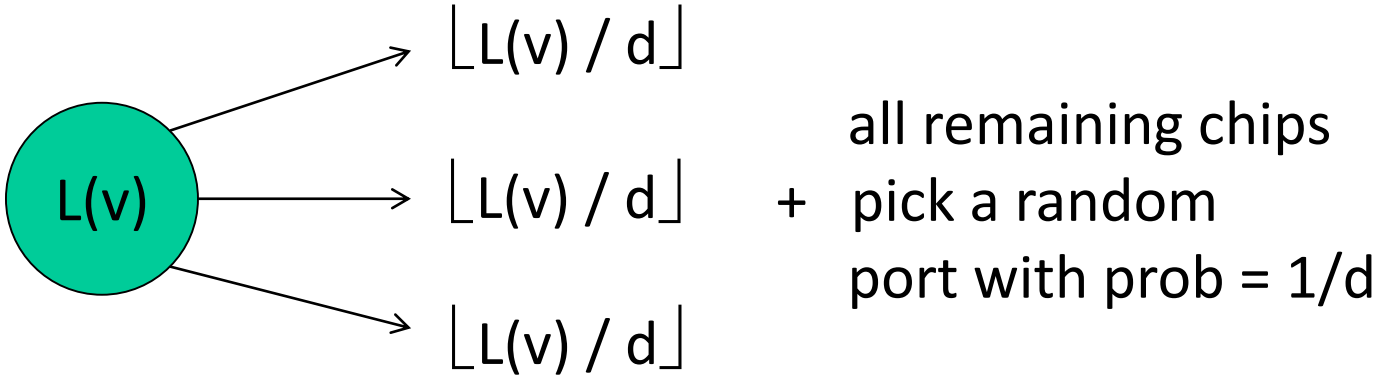


(keeping sum intact)

- Pros:
  - Load of each node off by **O (d log n / μ)** from continuous diffusion [Rabani et al., 1998]
  - Extremely general. **Can be made deterministic in many ways!**
- Cons:
  - Gap from continuous diffusion pretty big in general.
  - Says nothing about the behavior, of e.g., 1 chip only.

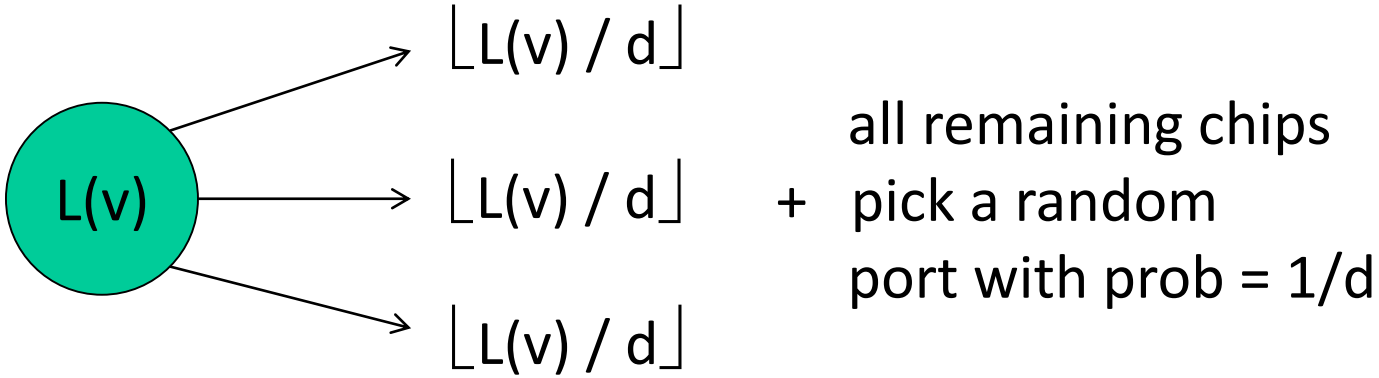
Discrete diffusion rules

Rule 2a: Split chips evenly for multiples of the degree; remainder follows random walks [Sauerwald and Sun, 2012]



Discrete diffusion rules

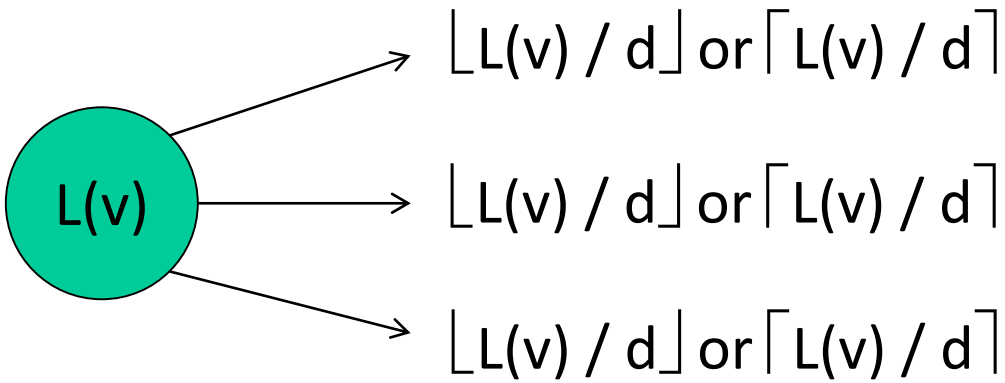
Rule 2a: Split chips evenly for multiples of the degree; remainder follows random walks [Sauerwald and Sun, 2012]



- Pros:
  - Expected number of chips at each location matches continuous diffusion
  - At about the time when continuous diffusion has evened out, off by **O(1)** from continuous diffusion **in expectation** [Sauerwald and Sun, 2012]
- Cons:
  - Still no better worst-case guarantee than previously.

Discrete diffusion rules

Rule 2b: emulate continuous diffusion explicitly

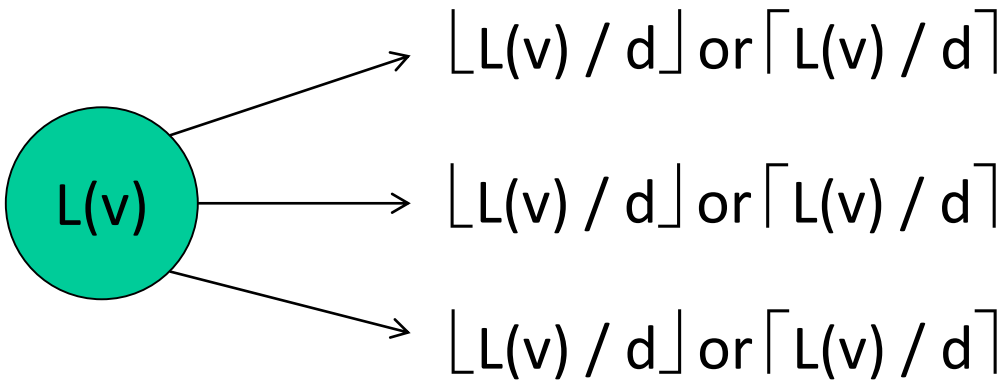


Rounding rules:

1) keep sums intact
2) try to create new loads which are as close as possible to those in continuous diffusion

Discrete diffusion rules

Rule 2b: emulate continuous diffusion explicitly



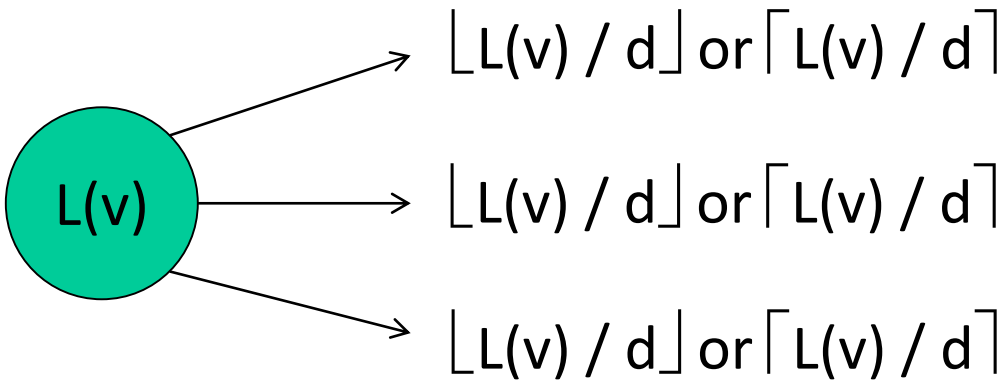
Rounding rules:

1) keep sums intact
2) try to create new loads which are as close as possible to those in continuous diffusion

- Pros:
  - Best possible simulation of continuous process: Load of each node off by **O (d) deterministically** w.r.t. continuous diffusion – in a slightly modified process which generates some extra artificial tokens and communication [Akbari et al. 2012]
- Cons:
  - A lot of communication, computation, state memory at nodes,...

Discrete diffusion rules

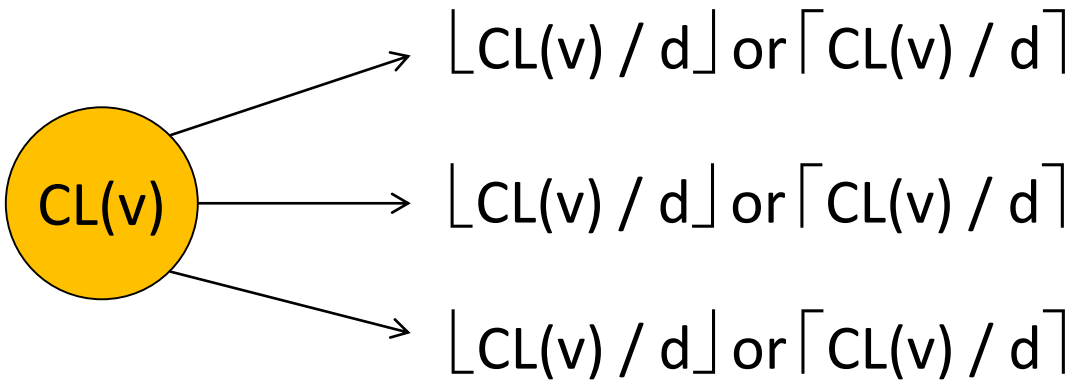
Rule 2c: Local fairness rules (our focus!)



Rounding rule: release chips so that every arc outgoing from node *v* is used the same number of times (+/-1) up to time *t*

Discrete diffusion rules

Rule 2c: Local fairness rules (our focus!), cumulative definition:



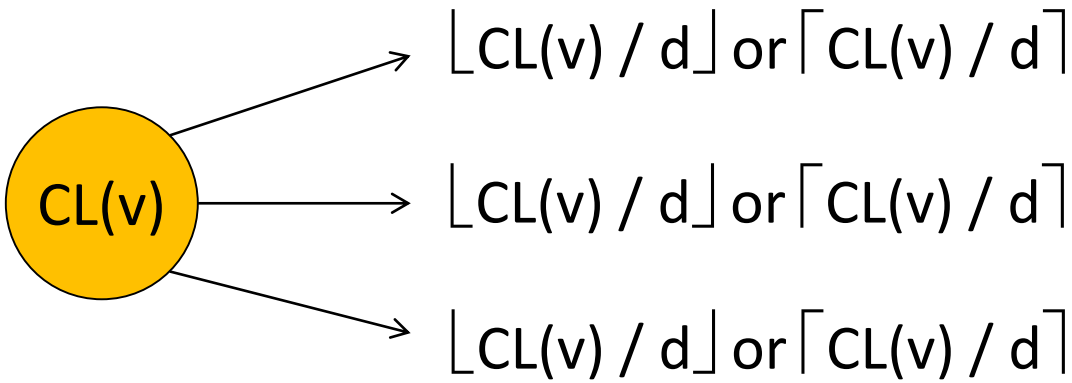
Define cumulative (integrated) load CL for nodes:

$$CL_T(v) = \sum_{t \le T} L_t(v)$$

(similar definition for arcs.)

Discrete diffusion rules

Rule 2c: Local fairness rules (our focus!), cumulative definition:



Some features:

- Simple to implement, very little local state information required
- Admits refinements (including a stateless one!) which provide better bounds on discrepancy from continuous diffusion then the general bound for Rule 2.
- Admits a deterministic refinement which is particularly useful in **graph exploration** – the rotor-router model.

The rotor-router model

The rotor-router model

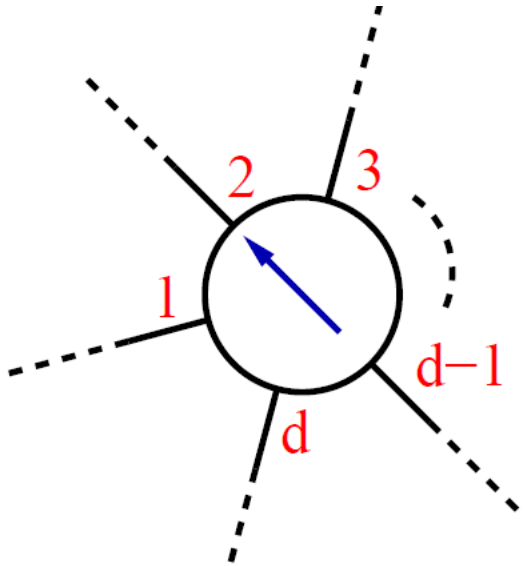
- Anonymous undirected graphs / networks
- No node identifiers, no auxiliary labels
- Each node v has a fixed local port numbering from 1 to deg(v)

Rotor-Router Mechanism

The state of each node v is a pointer **p(v)** $\in\{1, \ldots, \deg(v)\}$.

For each chip located at node v
at the start of time round t:

- The chip is pushed to the neighbor along port p(v)
- Pointer p(v) is incremented modulo the degree.



The rotor-router model

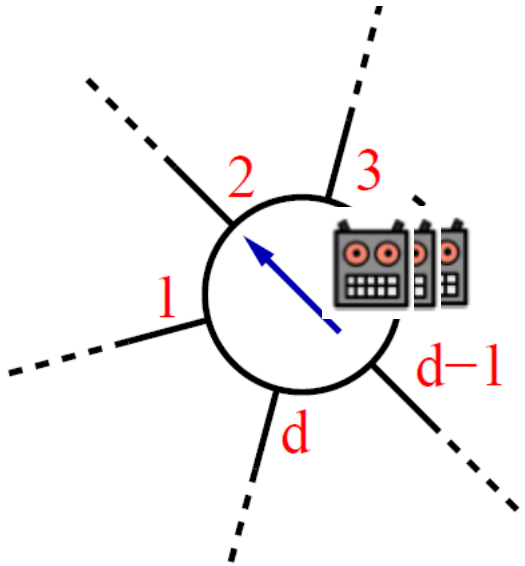
- Anonymous undirected graphs / networks
- No node identifiers, no auxiliary labels
- Each node v has a fixed local port numbering from 1 to deg(v)

Rotor-Router Mechanism

The state of each node v is a pointer **p(v)** $\in\{1, \ldots, \deg(v)\}$.

For each chip located at node v
at the start of time round t:

- The chip is pushed to the neighbor along port p(v)
- Pointer p(v) is incremented modulo the degree.



The rotor-router model

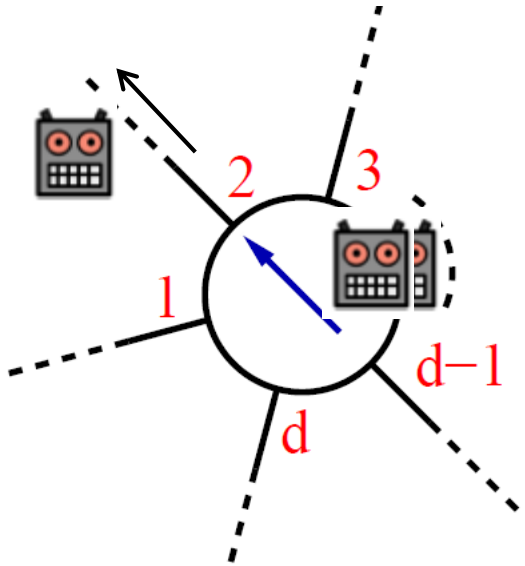
- Anonymous undirected graphs / networks
- No node identifiers, no auxiliary labels
- Each node v has a fixed local port numbering from 1 to deg(v)

Rotor-Router Mechanism

The state of each node v is a pointer **p(v)** $\in\{1, \ldots, \deg(v)\}$.

For each chip located at node v
at the start of time round t:

- The chip is pushed to the neighbor along port p(v)
- Pointer p(v) is incremented modulo the degree.



The rotor-router model

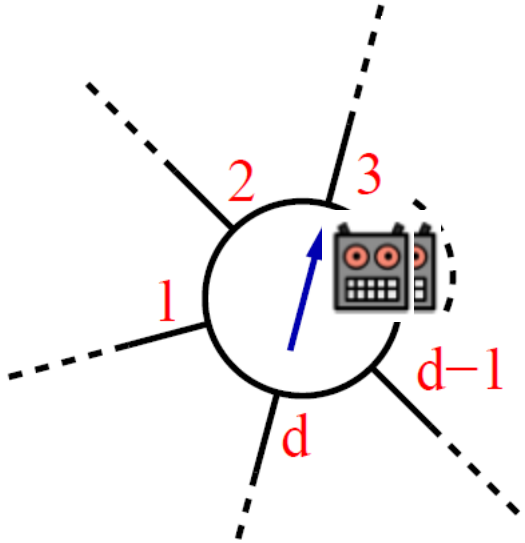
- Anonymous undirected graphs / networks
- No node identifiers, no auxiliary labels
- Each node v has a fixed local port numbering from 1 to deg(v)

Rotor-Router Mechanism

The state of each node v is a pointer **p(v)** $\in\{1, \ldots, \deg(v)\}$.

For each chip located at node v
at the start of time round t:

- The chip is pushed to the neighbor along port p(v)
- Pointer p(v) is incremented modulo the degree.



The rotor-router model

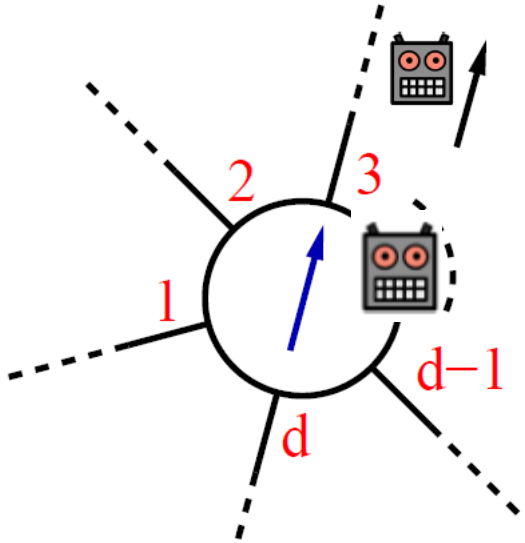
- Anonymous undirected graphs / networks
- No node identifiers, no auxiliary labels
- Each node v has a fixed local port numbering from 1 to deg(v)

Rotor-Router Mechanism

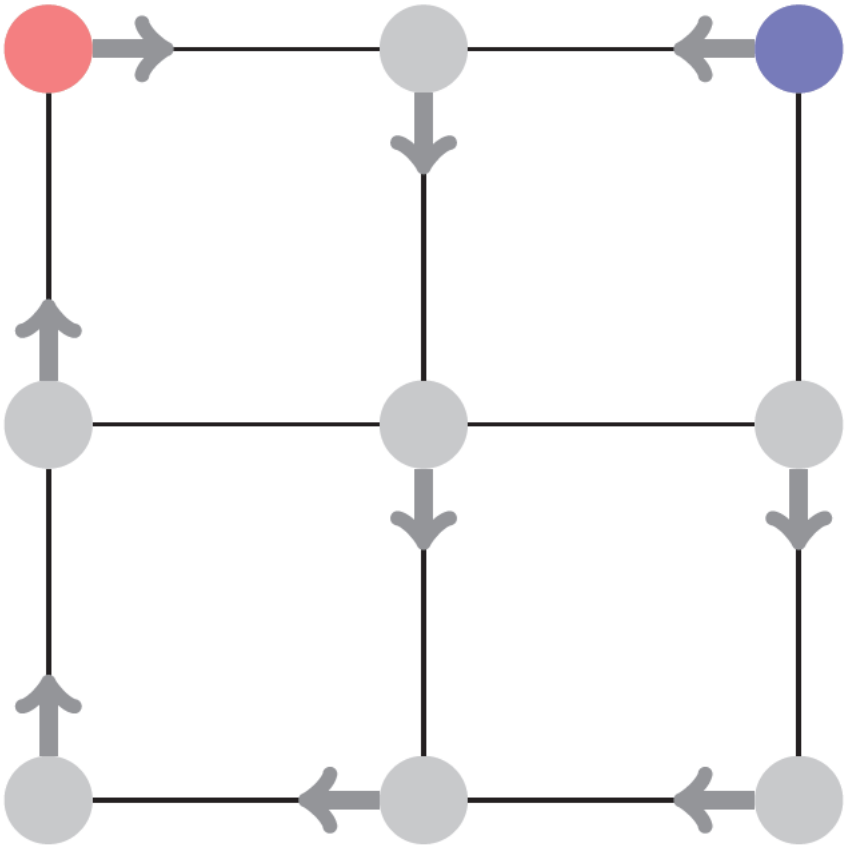
The state of each node v is a pointer **p(v)** $\in\{1, \ldots, \deg(v)\}$.

For each chip located at node v
at the start of time round t:

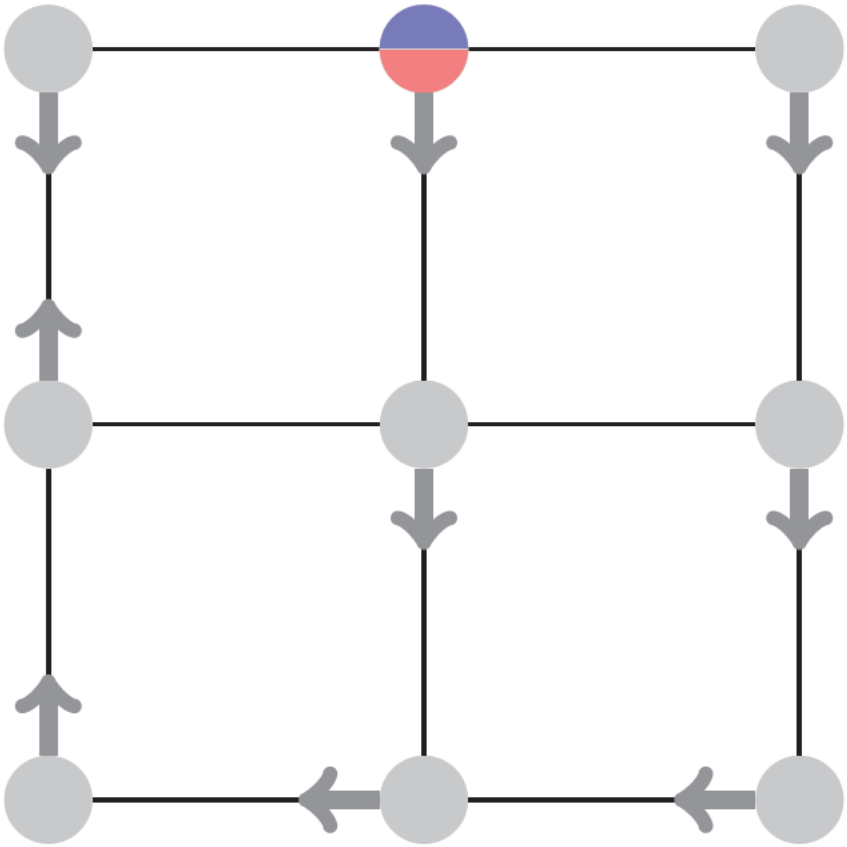
- The chip is pushed to the neighbor along port p(v)
- Pointer p(v) is incremented modulo the degree.



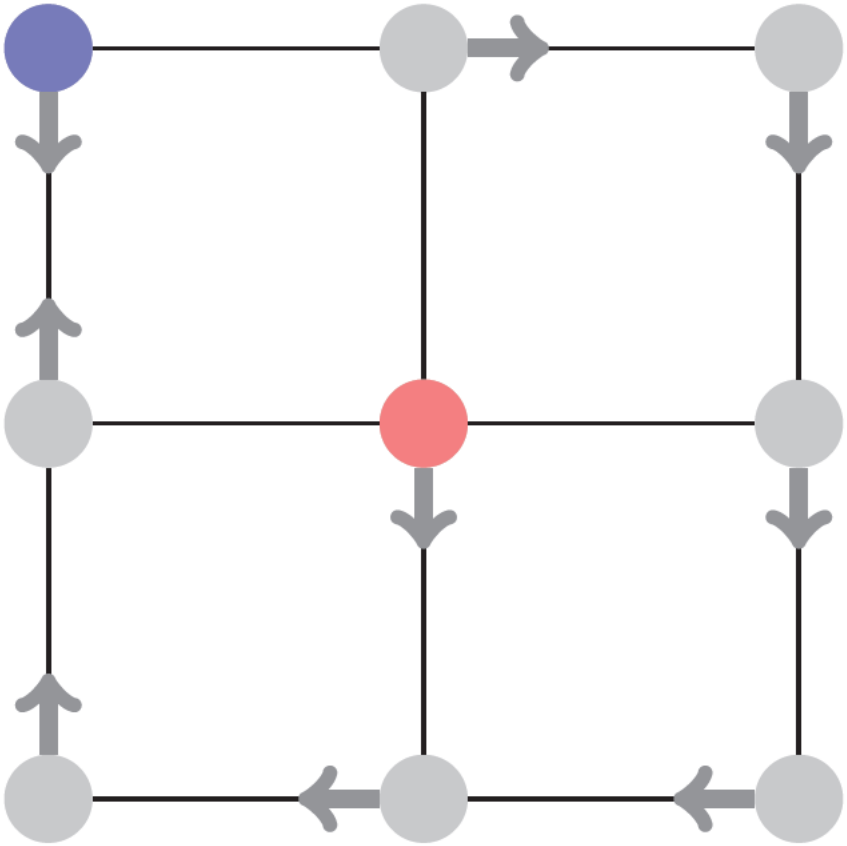
The rotor-router model: an example



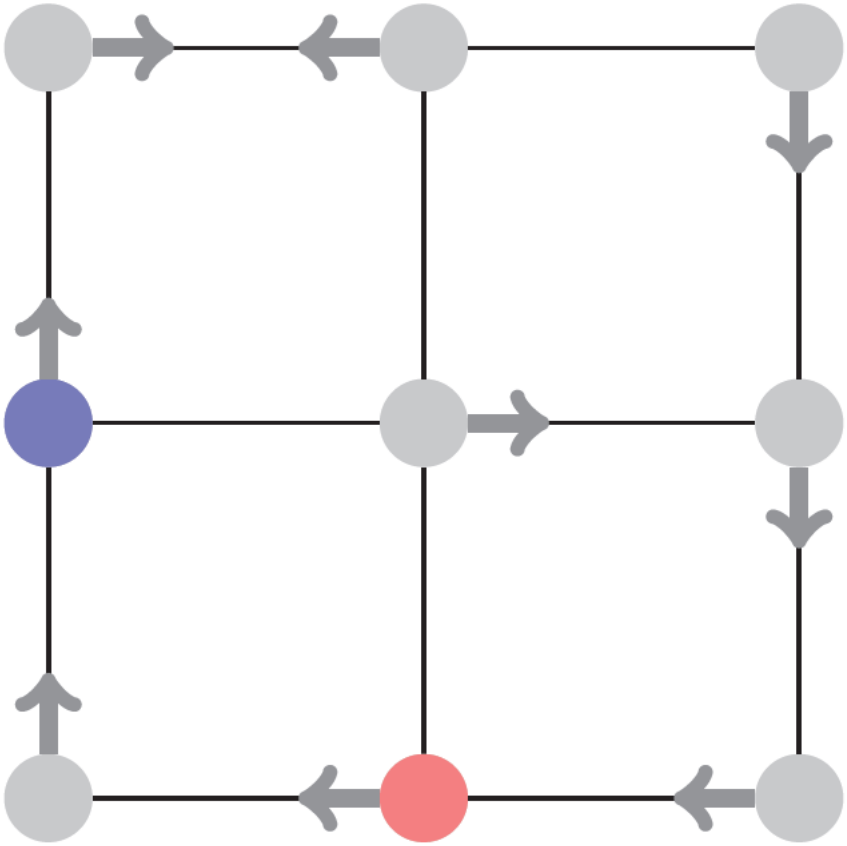
The rotor-router model: an example



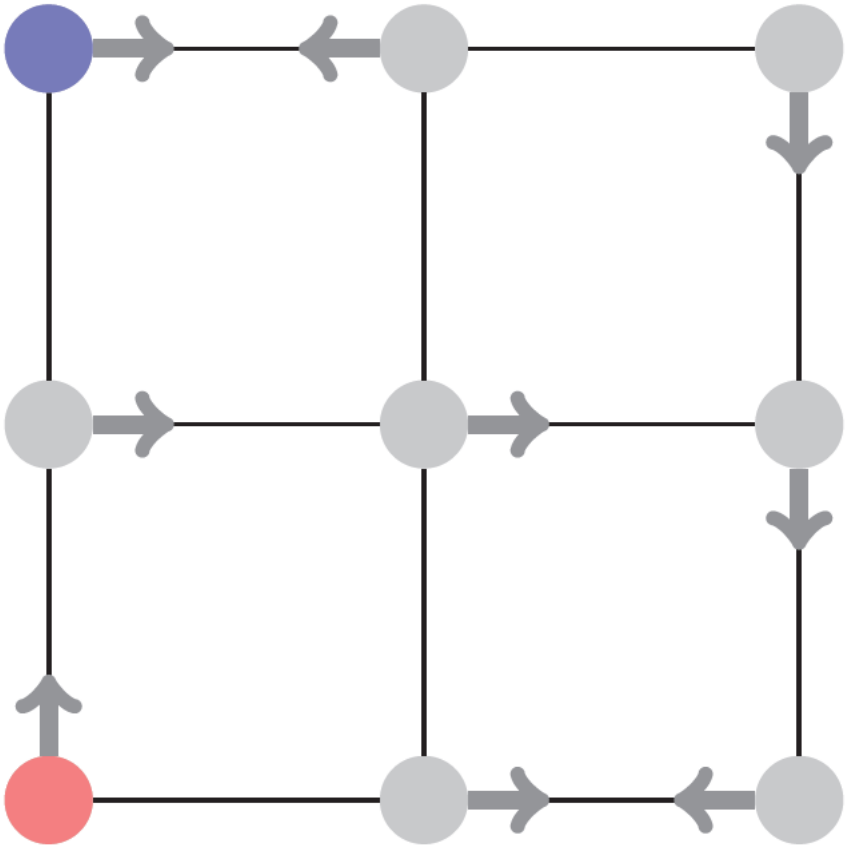
The rotor-router model: an example



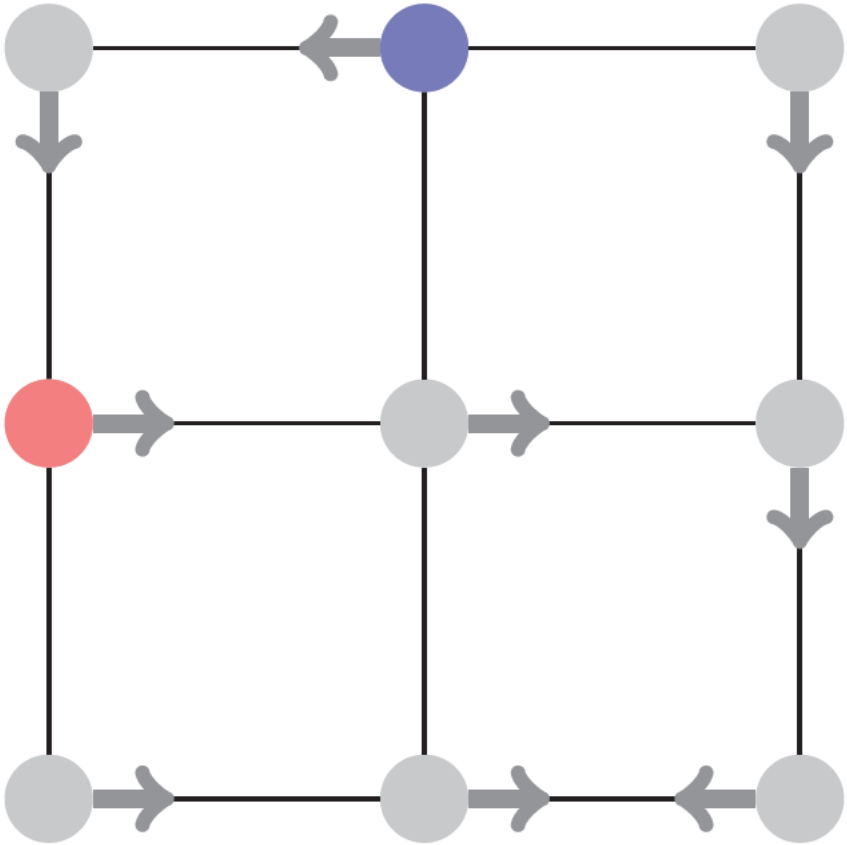
The rotor-router model: an example



The rotor-router model: an example



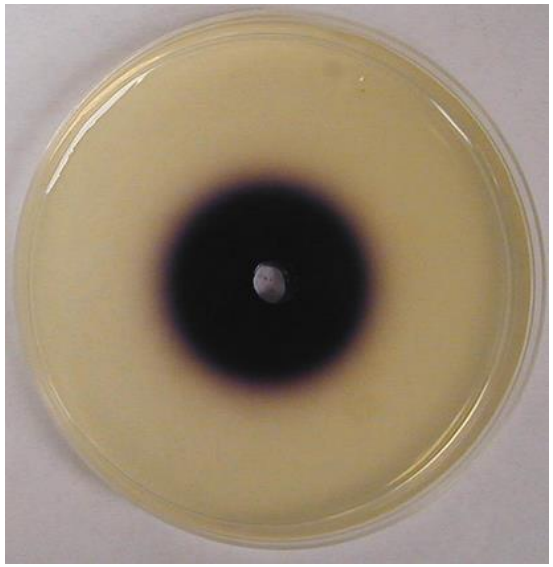
The rotor-router model: an example



Some load-balancing results for the rotor-router

Discrepancy w.r.t. continuous diffusion

- **Bounded by 2.29 for the line** [Cooper, Doerr, Spencer and Tardos, 2006]
- **Bounded by 7.8 for the grid** [Doerr and Friedrich, 2007]



Some load-balancing results for the rotor-router

Discrepancy w.r.t. continuous diffusion

- **Bounded by 2.29 for the line** [Cooper, Doerr, Spencer and Tardos, 2006]
- **Bounded by 7.8 for the grid** [Doerr and Friedrich, 2007]
- **Bounded by $O(d \mu^{-1} \log n)$** [Rabani et al., 1998]
- **We have recently obtained tighter bounds** for d-regular graphs, given enough self-loops [Berenbrink, Klasing, K., Mallmann-Trenn, Uznanski, to appear]
  - **These results can also be obtained for stateless processes similar to a rotor-router (without a pointer).**

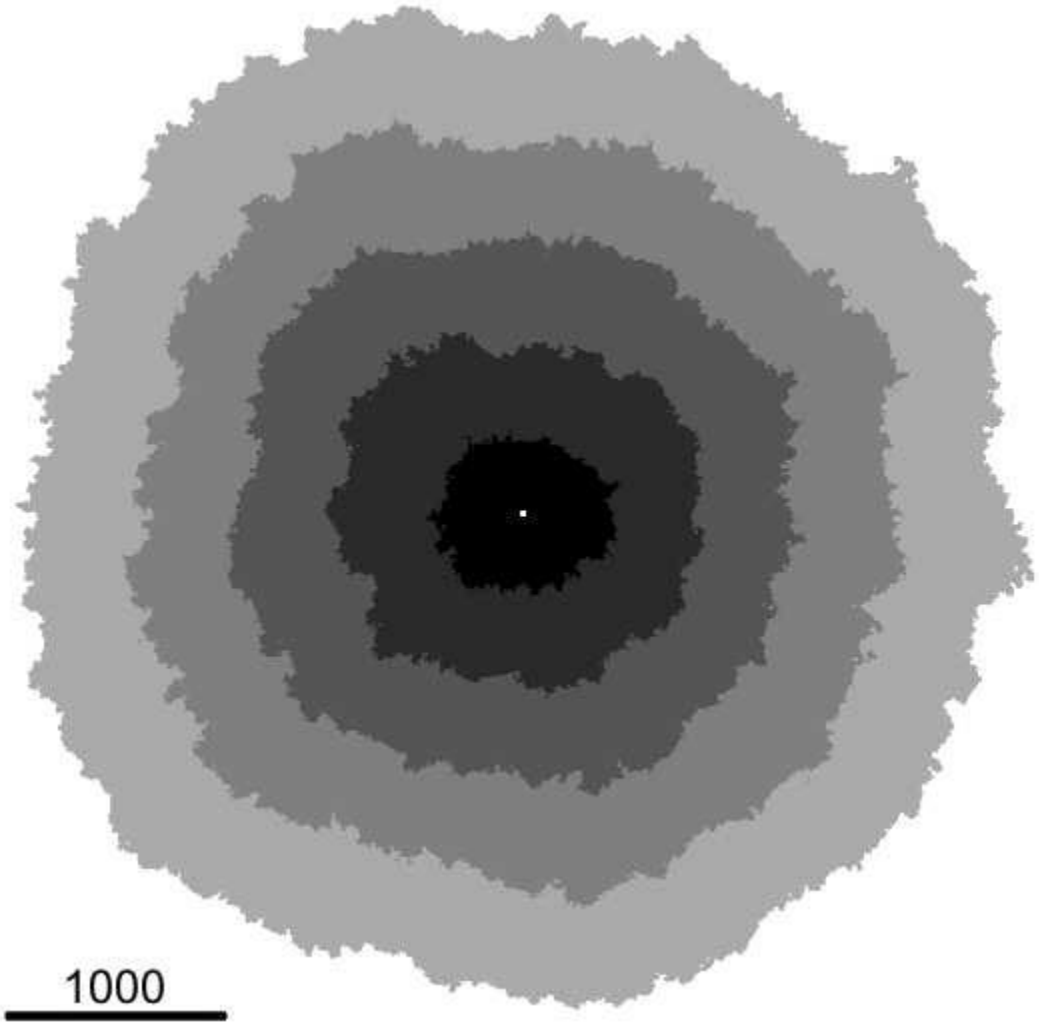
Side questions for a small number of chips

How many times is each node visited by a single chip, starting from the origin of an infinite grid?

black – 100

80
60
40
20
white – 0

(after about 10^8 steps,
random port initialization)



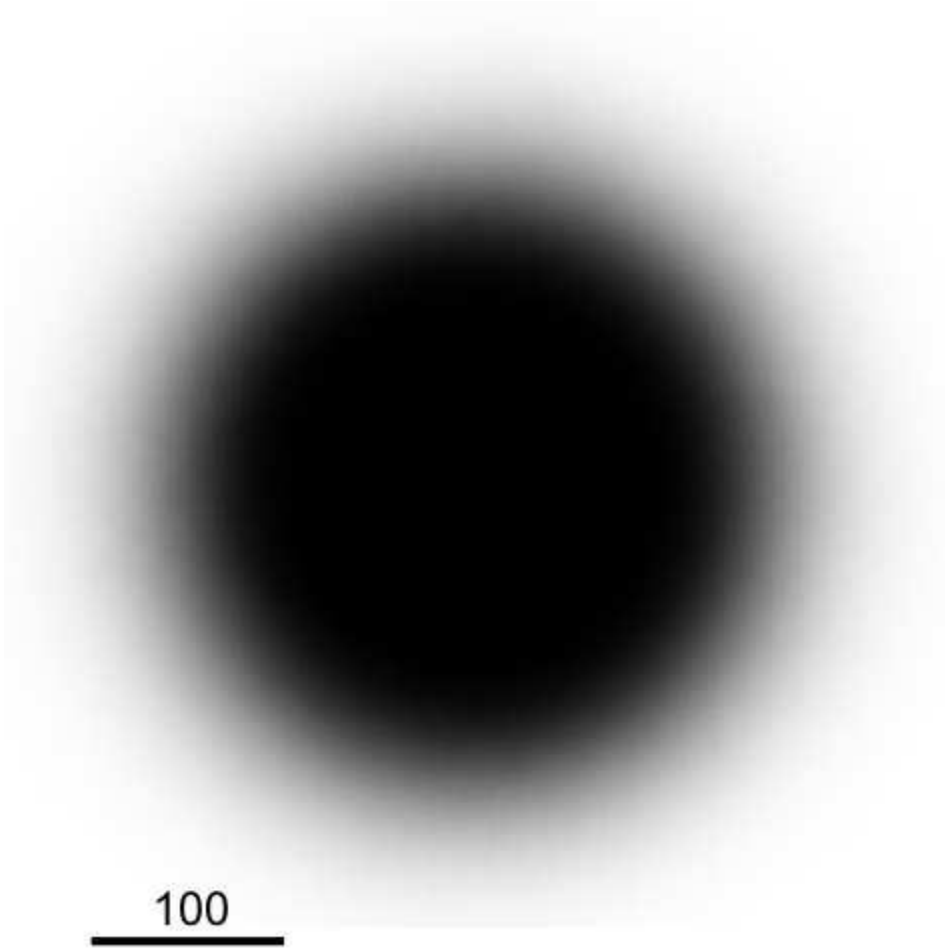
Side questions for a small number of chips

What is the probability that a node has been visited a given number of times, for uniformly random port initialization, during a fixed number of steps of the walk?

black – 1

white – 0

(after about 10^6 steps,
prob. of 10 visits to a node,
averaging over
5000 Monte-Carlo samples)



Exploration-related parameters of the rotor-router

- **Cover time**
  - When will have each node of the graph been reached by some chip, for a worst-case starting configuration?
  - Same as maximum hitting time (for a deterministic process)
- **Lock-in time**
  - The rotor-router is a deterministic process with a finite number of states, hence it must stabilize to a periodic traversal of some cycle in its state space after some initialization phase
  - After what time does the rotor-router enter its limit cycle?
- **Cycle length**
  - What is the length of the limit cycle of the rotor-router system?

Exploration with the rotor-router for 1 chip

The rotor-router with 1 chip exhibits elegant structural properties.

Theorem [Yanovski et al., 2001]:

- For any graph with diameter diam and m edges, cover time and lock-in time are bounded by $O(m \cdot diam)$.
- After this lock-in period, the rotor-router stabilizes to an **Eulerian traversal** of the directed version of the graph (traversing each edge once in each direction).

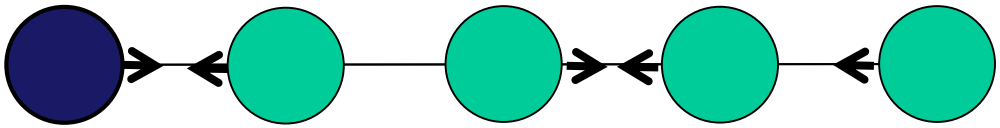
Theorem [Bampas, Gasieniec, Hanusse, Ilcinkas, Klasing, K. 2009]:

- For any graph with diameter diam and m edges, there exists an initial configuration of the rotor-router for which cover time and lock-in time are $\Theta(m \cdot diam)$.

Proof intuition (Eulerian lock-in)

The walker visits successively larger Eulerian subgraphs of *G*.

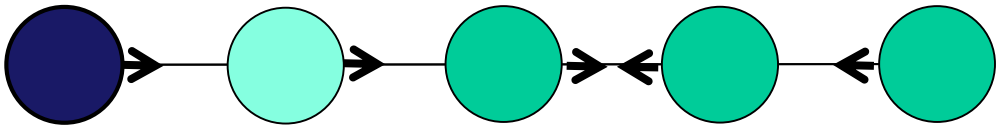
Initial configuration:



Proof intuition (Eulerian lock-in)

The walker visits successively larger Eulerian subgraphs of *G*.

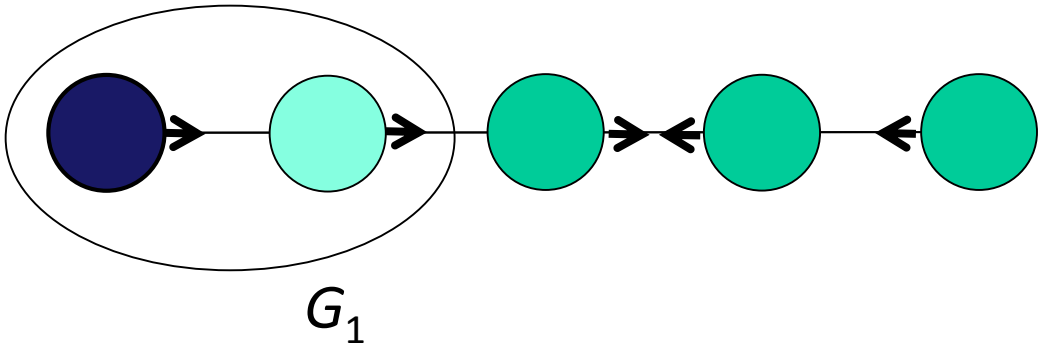
After the pointer at the starting node has made ”one full turn”:



Proof intuition (Eulerian lock-in)

The walker visits successively larger Eulerian subgraphs of *G*.

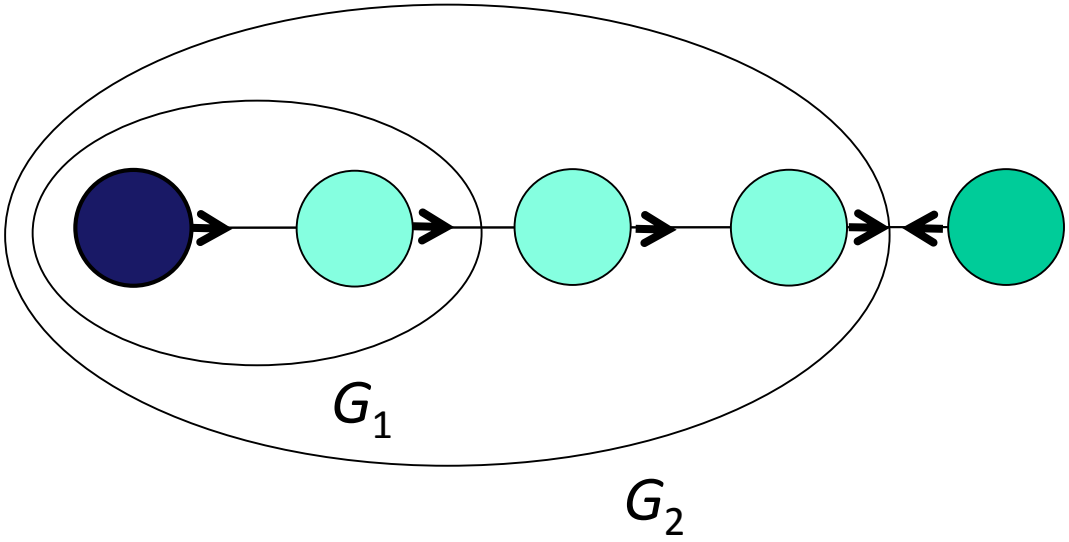
After the pointer at the starting node has made ”one full turn”:



Proof intuition (Eulerian lock-in)

The walker visits successively larger Eulerian subgraphs of *G*.

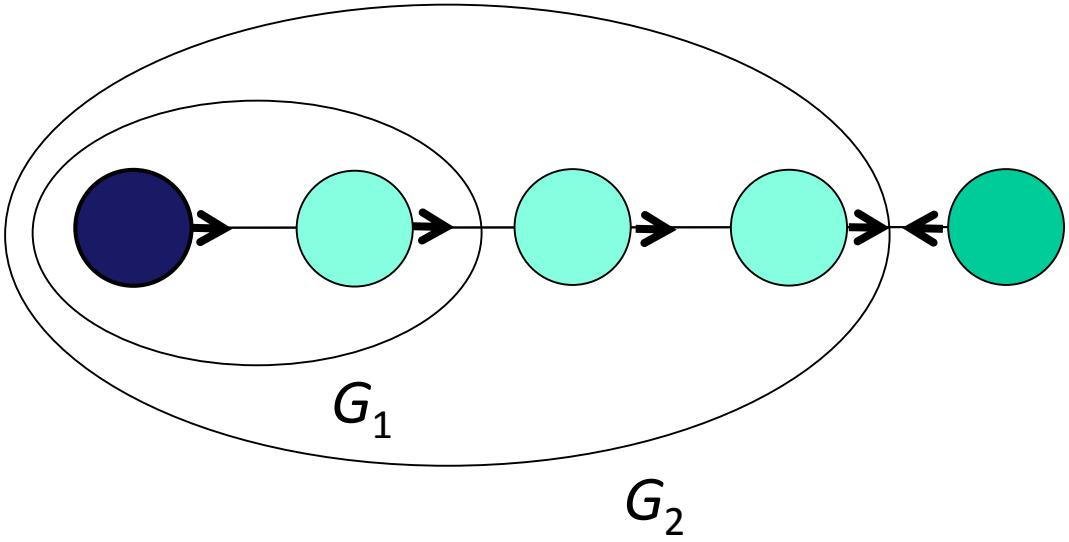
After the pointer at the starting node has made "two full turns":



Proof intuition (Eulerian lock-in)

The walker visits successively larger Eulerian subgraphs of *G*.

After the pointer at the starting node has made "two full turns":

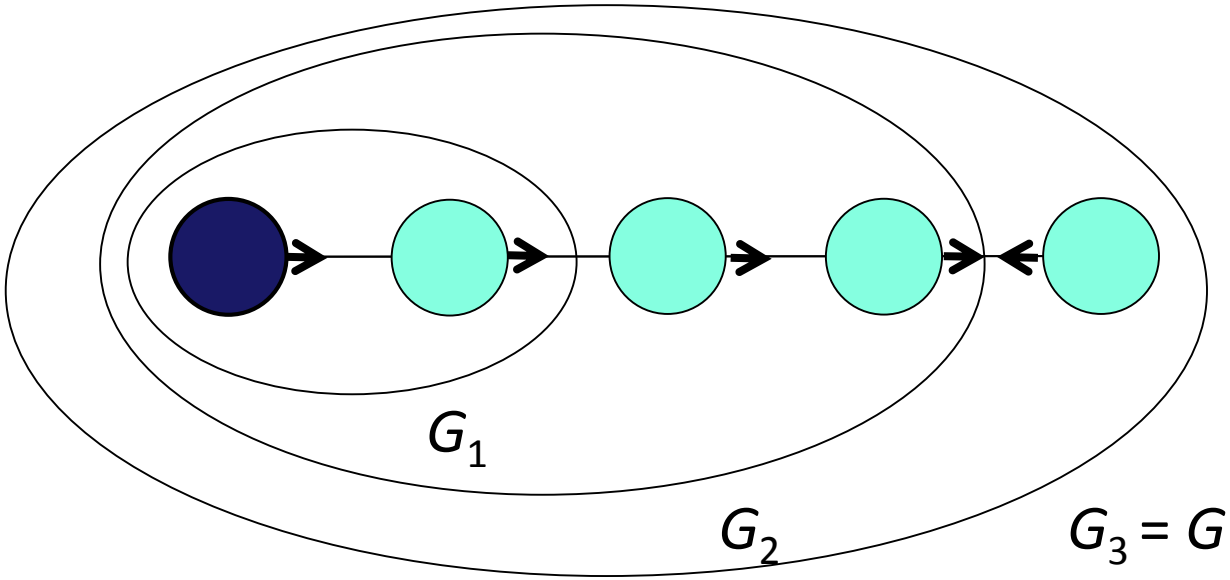


After every full turn of the pointer at the starting location, the walker has traversed all of the arcs of its previous Eulerian subgraph (and new arcs "one level deeper").

Proof intuition (Eulerian lock-in)

The walker visits successively larger Eulerian subgraphs of *G*.

After the pointer at the starting node has made "two full turns":



Stabilization is completed after O(diam) stages G_1, G_2,…,
each requiring O(m) time.

Exploration with the rotor-router for k chips

The rotor-router with $k>1$ chips exhibits more complex structural properties

- **Observation** [Uznanski and Das, personal communication, 2014]:
  There exist configurations with $k = O(n)$ chips having exponential period, but the lock-in time is always polynomial.
  - This says nothing about the cover time (which is of course still polynomial in n).
- The period of the system is always a multiple of $2m/k$.
- Experimental evidence suggests that for randomly sampled instances, the period seems to be not more than $2m$ a.s.
  [Yanovski et al. 2001]

Exploration with the rotor-router for *k* chips

We can make use of diffusive properties of the rotor-router

- **Recall the continuous diffusion equation:**

$$L_{t+1}(v) = \sum_{u \in N(v)} L_t(u) / \deg(u)$$

- The rotor-router satisfies it up to rounding:

$$L'_{t+1}(v) = \sum_{u \in N(v)} L'_t(u) / \deg(u) \quad +/- \quad O(1)$$

Exploration with the rotor-router for *k* chips

We can make use of diffusive properties of the rotor-router

- **Recall the continuous diffusion equation:**

$$L_{t+1}(v) = \sum_{u \in N(v)} L_t(u) / \deg(u)$$

- The rotor-router satisfies it up to rounding:

$$L'_{t+1}(v) = \sum_{u \in N(v)} L'_t(u) / \deg(u) \quad +/- \quad O(1)$$

… not strong enough.

Exploration with the rotor-router for *k* chips

We can make use of diffusive properties of the rotor-router

- **Recall the continuous diffusion equation:**

$$L_{t+1}(v) = \sum_{u \in N(v)} L_t(u) / \deg(u)$$

- Recall the definition of cumulative load (=total number of visits up to time T):

$$CL_T(v) = \sum_{t \le T} L_t(v)$$

- Summing the diffusion equation over time, we obtain:

$$CL_{T+1}(v) = CL_0(v) + \sum_{u \in N(v)} CL_t(u) / \deg(u)$$

Exploration with the rotor-router for *k* chips

We can make use of diffusive properties of the rotor-router

- **Recall the continuous diffusion equation:**

$$L_{t+1}(v) = \sum_{u \in N(v)} L_t(u) / \deg(u)$$

- Recall the definition of cumulative load (=total number of visits up to time T):

$$CL_T(v) = \sum_{t \le T} L_t(v)$$

- Summing the diffusion equation over time, we obtain:

$$CL_{T+1}(v) = CL_0(v) + \sum_{u \in N(v)} CL_t(u) / \deg(u)$$

- The rotor-router satisfies this up to rounding:

$$CL'_{T+1}(v) = CL'_0(v) + \sum_{u \in N(v)} \lceil (CL'_t(u) + \text{port}(u,v)) / \deg(u) \rceil$$

Exploration with the rotor-router for *k* chips

Corollaries:

- **Monotonicity property:** [Yanovski et al., 2001]
  Adding a new walker cannot increase cover time.

- **Stronger monotonicity property:** [Klasing, K., Pajak, Sauerwald, 2013]
  - Delaying a walker for any number of rounds (”holding it down with your finger at a node”) cannot decrease cover time.
  - Concept of a **delayed deployment *D*,** obtained from a rotor-router initialization by defining how many walkers to delay at which times and at which nodes.

Exploration with the rotor-router for *k* chips

The Slow-down Lemma

(a tool for proving upper bounds and lower bounds on cover time)

[Klasing, K., Pajak, Sauerwald, 2013]

- Let R[k] be a k-walker rotor router system with an arbitrarily chosen initialization.
- Let D be any delayed deployment of R[k].
- Suppose that:
  - Deployment D covers all the vertices of the graph after T rounds.
  - In at least τ of these rounds, all walkers were active in D.

Lemma: The cover time C(R[k]) of the system can be bounded by:

$$\tau \leq C(R[k]) \leq T$$

Exploration with the rotor-router for *k* chips

How to use the slow-down lemma when analyzing cover times?

- We can analyse the cover time of R[k] by constructing some easy to analyse delayed deployment D.
- This allows us to think of the rotor-router as an algorithm, rather than a process which is imposed upon us.
- If the deployment D is defined so that agents in D are delayed in at most a constant proportion of the first C(D) rounds then the above inequalities lead to an asymptotic bound on the value of the undelayed rotor-router: $C(R[k]) = \Theta(C(D))$.

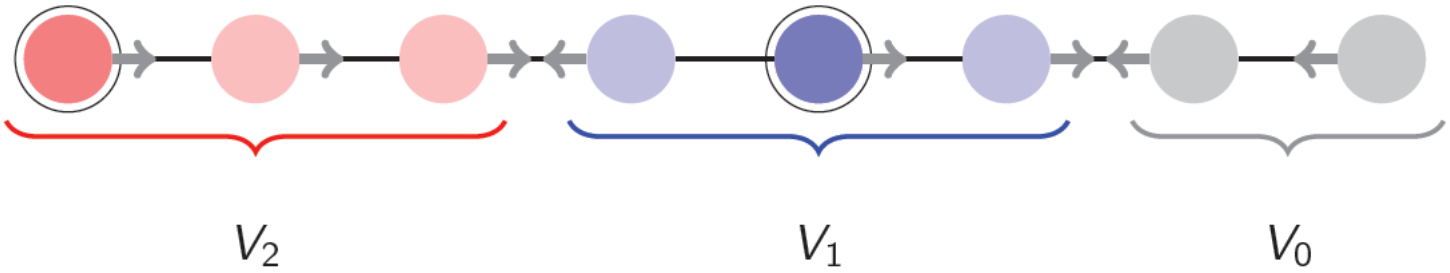
The *k* chip rotor-router on the path/ring

Case study: the rotor-router on the path (or ring) for *k* << *n*

- Intuition: Each walker occupies a ”domain”, which it patrols.
- A node *v* belongs to domain V_i (t) of the i-th walker if this walker was the last agent visiting node v until round t, inclusive.
- A special domain V_0(t) contains all nodes which have not yet been visited.
- One can show that domains either form spontaneously as segments, or by holding back a few walkers we can force them to form (delayed deployment). [Klasing, K., Pajak, Sauerwald, 2013]
- Within a domain, all ports are aligned ”towards” the walker which is its owner.

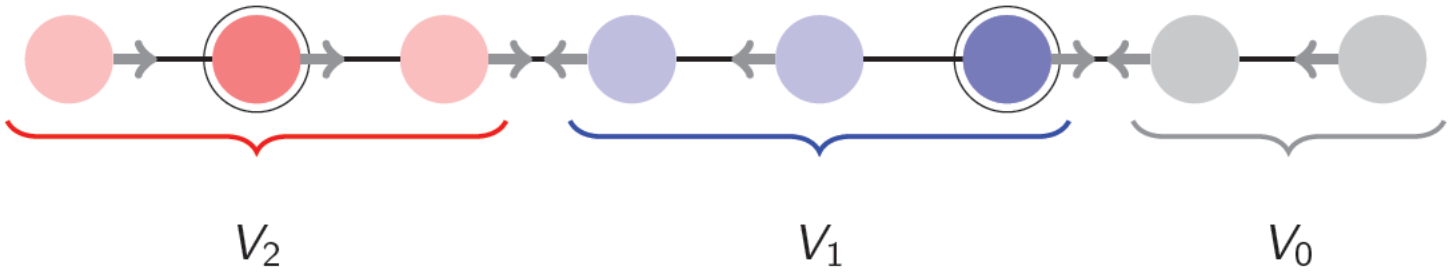
The *k* chip rotor-router on the path/ring

Example on the line, *k*=2 (starting from some moment…)



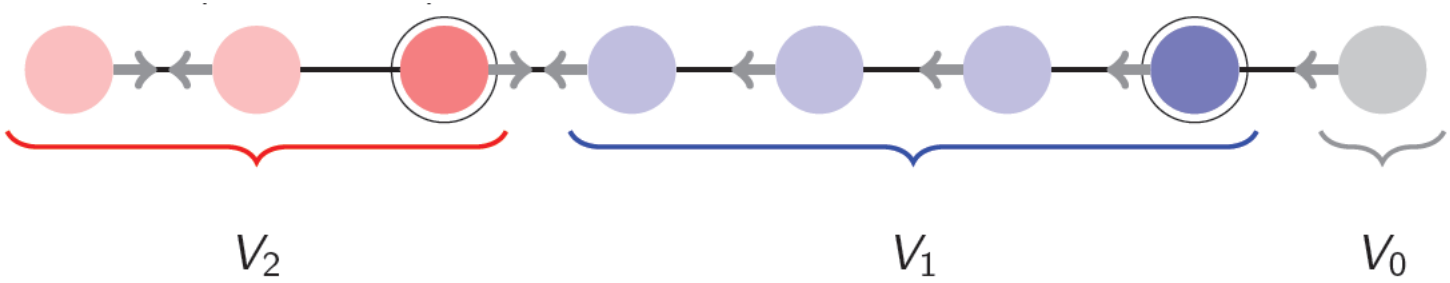
The *k* chip rotor-router on the path/ring

Example on the line, *k*=2 (starting from some moment…)



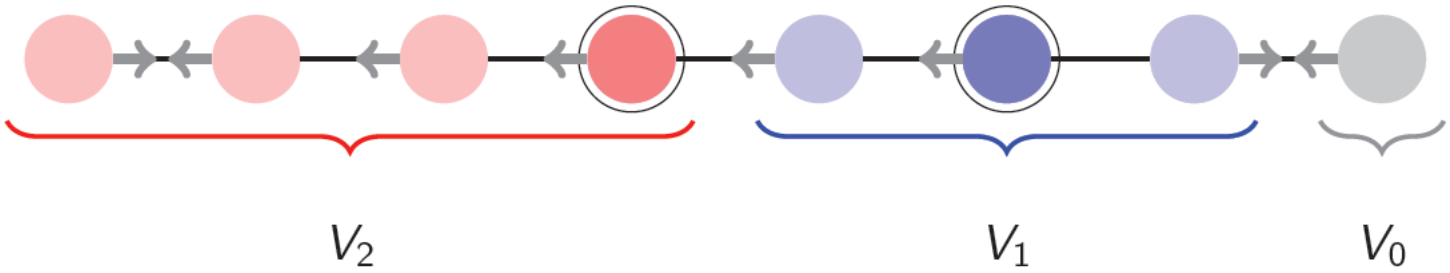
The *k* chip rotor-router on the path/ring

Example on the line, *k*=2 (starting from some moment…)



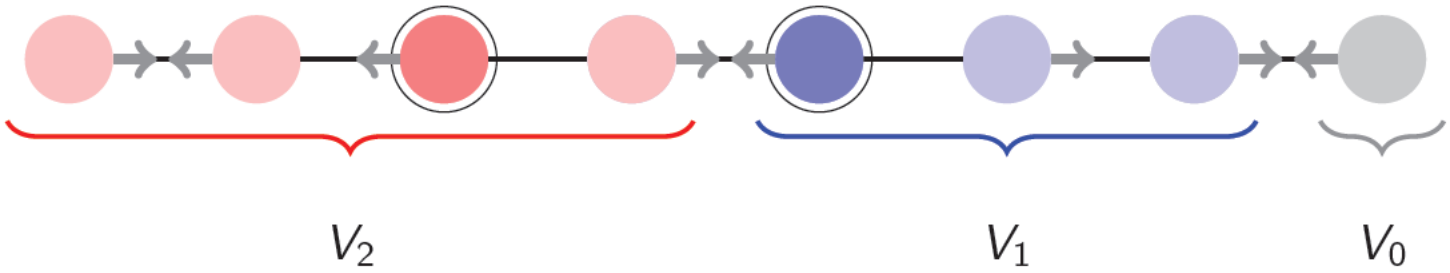
The *k* chip rotor-router on the path/ring

Example on the line, *k*=2 (starting from some moment…)



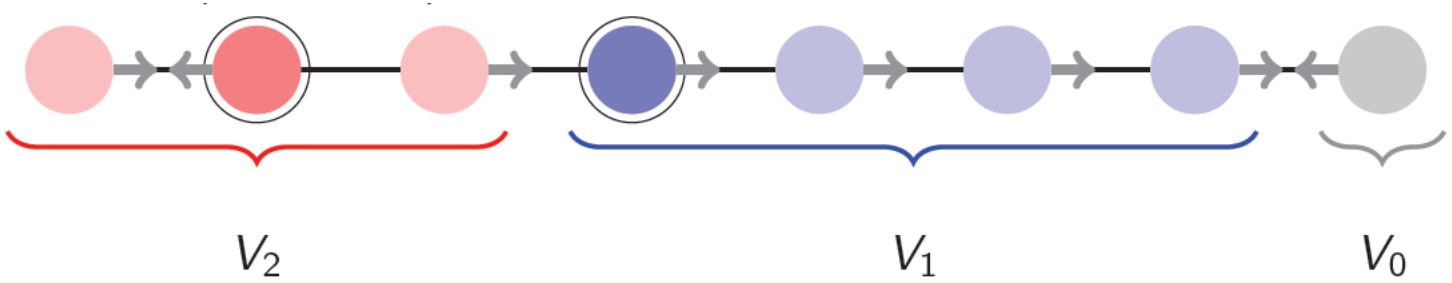
The *k* chip rotor-router on the path/ring

Example on the line, *k*=2 (starting from some moment…)



The *k* chip rotor-router on the path/ring

Example on the line, *k*=2 (starting from some moment…)



The *k* chip rotor-router on the path/ring

Case study: the rotor-router on the path (or ring) for *k* << *n*

- Roughly speaking, each walker *i* enlarges its own domain of size $\nu_i(t) = |V_i(t)|$ once every $\nu_i(t)$ steps (once at the left end, once at the right end)
- At each of the ends, the size of the domain is reduced by the adjacent agent (except from the side with $V_0(t)$, if applicable).
- We obtain the following continuous-time approximation:

$$\frac{d\nu_i(t)}{dt} = \frac{1}{\nu_i(t)} - \frac{1}{2\nu_{i-1}(t)} - \frac{1}{2\nu_{i+1}(t)}$$

- This approximation is accurate in the sense that one can construct a delayed deployment which (almost) adheres to its solution.

The *k* chip rotor-router on the path/ring

Theorem. Cover time of the rotor-router on the path [Klasing, K., Pajak, Sauerwald, 2013]

In the case when all walkers are initially placed at the same node v with all pointers are initialized along the shortest path to v, the *k*-walker rotor-router explores the path/ring of size n in time $\Theta(n^2 / \log k)$ when $k < n^{1/11}$.

Model	*Cover time*		*Return time*
	worst placement	*best placement*	
k-agent rotor-router	$\Theta(n^2/\log k)$	$\Theta(n^2/k^2)$	$\Theta(n/k)$
k random walks (expectations)	$\Theta(n^2/\log k)$ in literature	$\Theta\left(n^2/\frac{k^2}{\log^2 k}\right)$	$\Theta(n/k)$ in literature

The *k* chip rotor-router in general graphs

What happens for general graphs?

- Even less structure – forget about domains.
- Slowdown lemma still holds and proves useful.
- One can extract some additional properties from the cumulutive load equation

Theorem. Cover time of the k-walker rotor-router

[Dereniowski, K., Pajak, Uznanski, 2014]

The *k*-walker rotor-router covers any graph n in worst-case time **O(m diam / log k)** and **Ω(m diam / k),** whenever k is polynomial in n.

Both of these bounds are achieved for some graph classes.

The *k* chip rotor-router in general graphs

1 walker versus *k* walkers: comparison of speed-up

Graph class	Speedup of Rotor-Router *for cover time*	Speedup of Random Walk *for cover time*	Speedup of Random Walk *for max hitting time*
General case:	$\Omega(\log k)$, $O(k)$	$O(k^2)$, $O(k \log n)$	$O(k)$
Cycle:	$\Theta(\log k)$	$\Theta(\log k)$	$\Theta(\log k)$
Star:	$\Theta(k)$	$\Theta(k)$	$\Theta(k)$

(all results hold up to *k* polynomially large with respect to *n*)

Examples:

Worst-case graph (logarithmic speed-up): path, cycle

Best-case graph (linear speed-up): star, clique

Different behavior for the rotor-router and r.w. 2D grid

Load balancing parameters of the rotor-router

- **Blanket time (for small *k*)**
  After what time T will each node of the graph been visited by a similar number of chips up to time T (in total)?
  - Rougly like the cover time for 1 chip.
  - No visible speed-up with more chips for some graphs.

- **Discrepancy w.r.t. continuous diffusion (only for large *k*)**
  What is the maximum difference between the number of chips per node in the considered process and in continuous diffusion?
  (possibly starting from some moment of time)

- **Diffusion time (only for large *k*)**
  After what time t will the discrepancy between the deterministic process and continuous diffusion be bounded?

Research directions

Some research directions we are exploring

- **Exploring trade-offs between time, memory, knowledge, and randomness.**

- **Understanding randomness as an agent resource**
  - What adversaries can you fool with pseudorandomness?
  - Can a limited-entropy source help fool an adversary?

- **”Massively parallel computations” with mobile agents**
  - Can other problems (apart from load balancing) be approached using agent-based techniques?

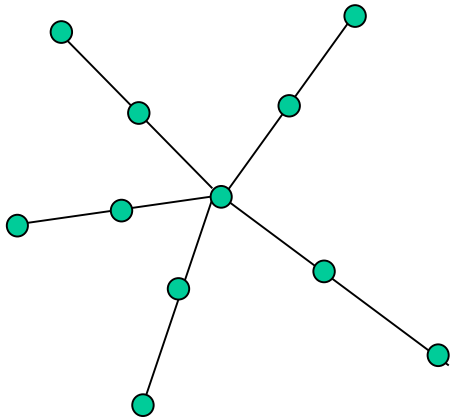
- **Modeling biological agents**
  - Behavior of immunological cells as a search problem with advice.

Questions for tonight

Some questions

A. Consider the subdivided star graph on n = 2k + 1 nodes.

- Estimate (up to multiplicative constants) the cover time of the simple random walk on this graph.
- Estimate (up to multiplicative constants) the cover time of the Metropolis walk on this graph.



B. For arbitrarily large *n*, show an example of an *n*-node labeled graph which contains two distinguished nodes *u*, *v* whose views are different, but identical up to as large a depth as possible.

(The best you can do is very close to *n*).

C. Suppose that a load balancing strategy satisfies the "rounding bound" (node v always sends $\lfloor L(v) / d \rfloor$ or $\lceil L(v) / d \rceil$ to each neighbor).
Give a tight upper bound on the value of:

$$\lim_{t \to \infty} \left(\max_{v \in V} L(v) - \min_{v \in V} L(v) \right)$$

Thank you!