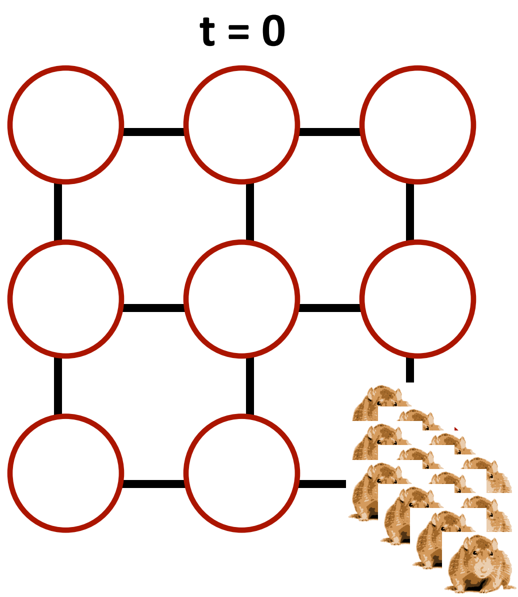
Assignment 3: GraphRats

GraphRats
Carnegie Mellon University

Topics

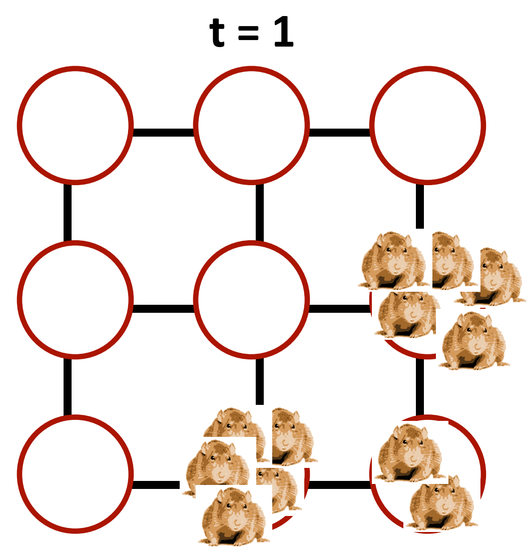
- Application
- Implementation Issues
- Optimizing for Parallel Performance
- Useful Advice

Basic Idea



- **Graph**
  - K X K grid
- **Initial State**
  - Start with all *R* rats in corner

- **Transitions**
  - Each rat decides where to move next
    - Don't like crowds
    - But also don't like to be alone
  - Weighted random choice



Node Count Representation (K = 12)

```
t = 0.
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |576|
+---+---+---+---+---+---+---+---+---+---+---+---+
```

```
t = 1.
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |165|164|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |168| 79|
+---+---+---+---+---+---+---+---+---+---+---+---+
```

Simulation Example

```
t = 0.
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |576|
+---+---+---+---+---+---+---+---+---+---+---+---+
```

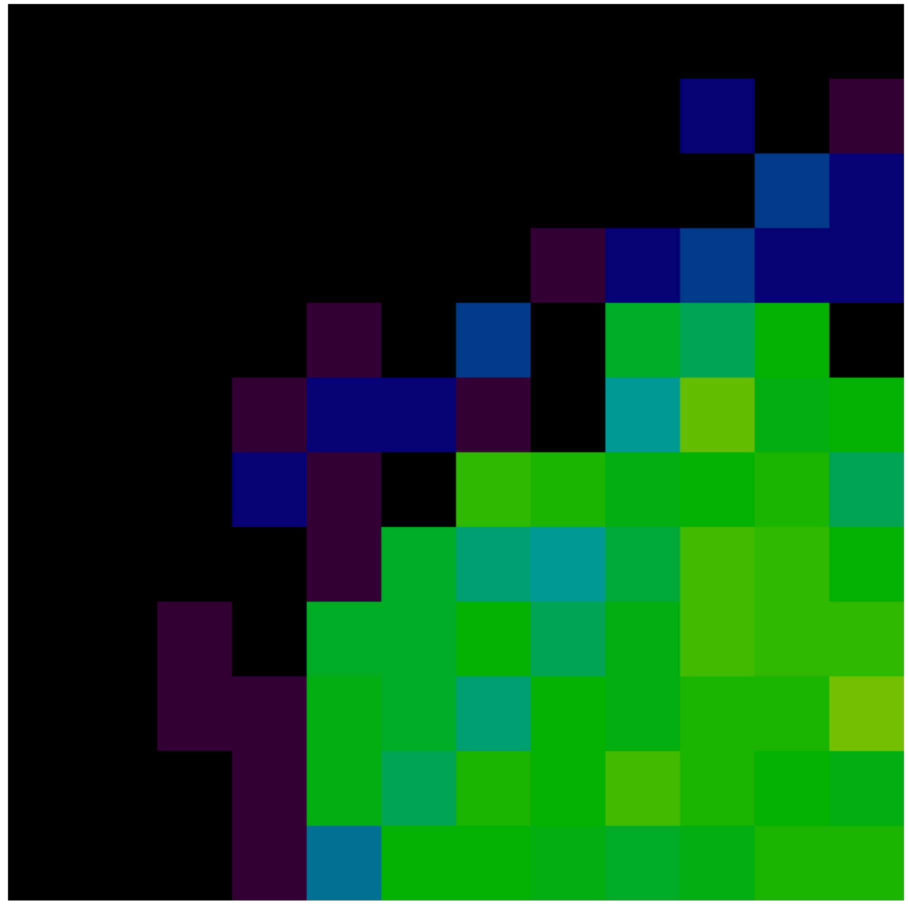
```
t = 30.
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   | 2 |   | 1 |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   | 3 | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 1 | 2 | 3 | 2 | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   | 1 |   | 3 |   | 9 | 7 | 11|   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 1 | 2 | 2 | 1 |   | 5 | 16| 10| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 2 | 1 |   | 13| 12| 10| 11| 12| 7 |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   | 1 | 9 | 6 | 5 | 8 | 14| 13| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   | 1 |   | 9 | 9 | 11| 7 | 10| 14| 13| 13|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   | 1 | 1 | 10| 9 | 6 | 11| 10| 12| 12| 17|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 1 | 10| 7 | 12| 11| 14| 12| 11| 10|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 1 | 4 | 11| 11| 10| 9 | 10| 12| 12|
+---+---+---+---+---+---+---+---+---+---+---+---+
```

Visualizations

Text ("a" for ASCII)

```
t = 30.
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   | 2 |   | 1 |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   |   |   |   | 3 | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   |   |   |   | 1 | 2 | 3 | 2 | 2 |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   | 1 |   | 3 |   | 9 | 7 | 11|   |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 1 | 2 | 2 | 1 |   | 5 | 16| 10| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 2 | 1 |   | 13| 12| 10| 11| 12| 7 |
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   |   | 1 | 9 | 6 | 5 | 8 | 14| 13| 11|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   | 1 |   | 9 | 9 | 11| 7 | 10| 14| 13| 13|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   | 1 | 1 | 10| 9 | 6 | 11| 10| 12| 12| 17|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 1 | 10| 7 | 12| 11| 14| 12| 11| 10|
+---+---+---+---+---+---+---+---+---+---+---+---+
|   |   |   | 1 | 4 | 11| 11| 10| 9 | 10| 12| 12|
+---+---+---+---+---+---+---+---+---+---+---+---+
```

Heat Map ("h")



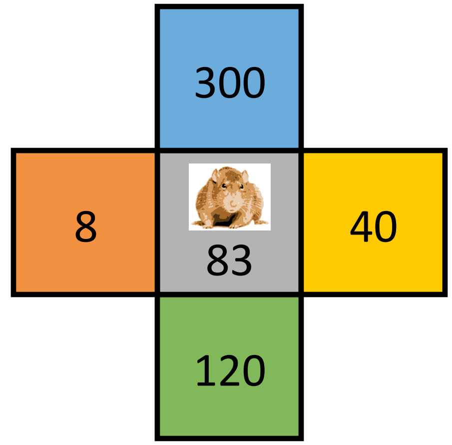
Running it yourself

```
linux> cd some directory
linux> git clone https//github.com/cmu15418/asst3-s19.git
linux> cd asst3-s19/code
Linux> make demoX
```

***X* from 1 to 10**

- **Demos**
  - 1: Text visualization, synchronous updates
  - 2: Heap-map, synchronous updates

Determining Rat Moves

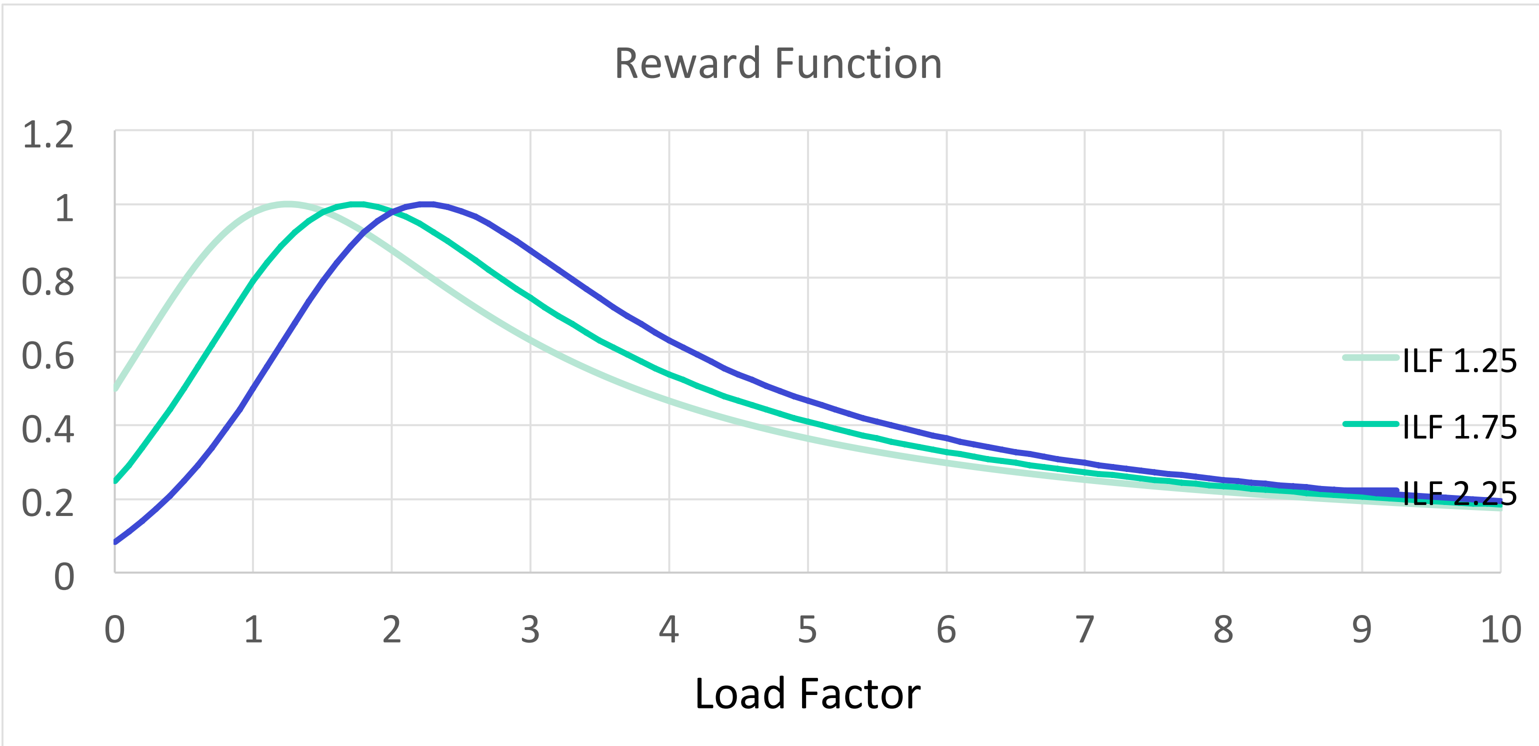


- **Count number of rats at current and adjacent locations**
  - Adjacency structure represented as graph
- **Compute reward value for each location**
  - Based on *load factor* $\boldsymbol{l}$ = count/average count
  - $\boldsymbol{l}^*$ Ideal load factor (ILF) (varying)
  - $\boldsymbol{\alpha}$ Fitting parameter (= 0.4)

$$Reward(\boldsymbol{l}) = \frac{1}{1 + (\log_2 [1 + \alpha(\boldsymbol{l} - \boldsymbol{l}^*)])^2}$$

Reward Function

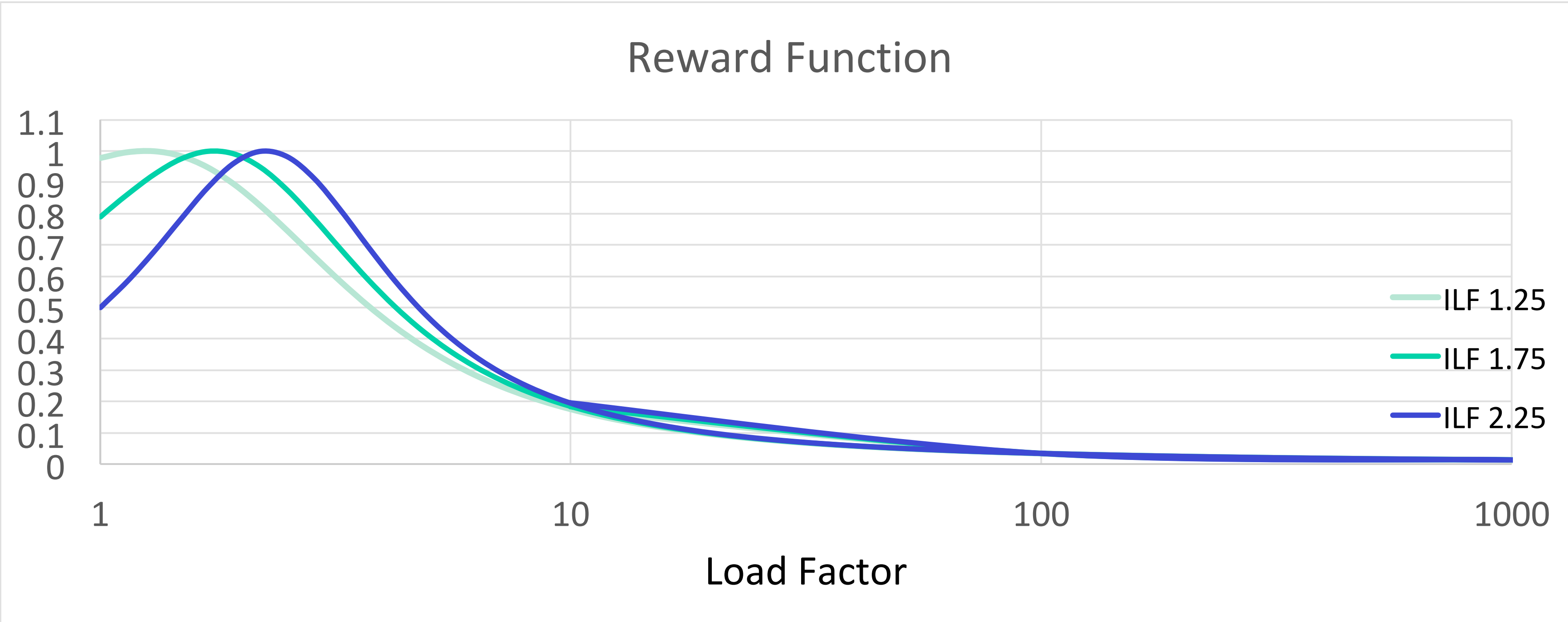
$$Reward(l) = \frac{1}{1 + (\log_2 [1 + \alpha(l - l^*)])^2}$$



- Maximized at ILF
  - Just above average population
  - Drops for smaller loads (too few) and larger loads (too crowded)

Reward Function (cont.)

$$Reward(l) = \frac{1}{1 + (\log_2 [1 + \alpha(l - l^*)])^2}$$

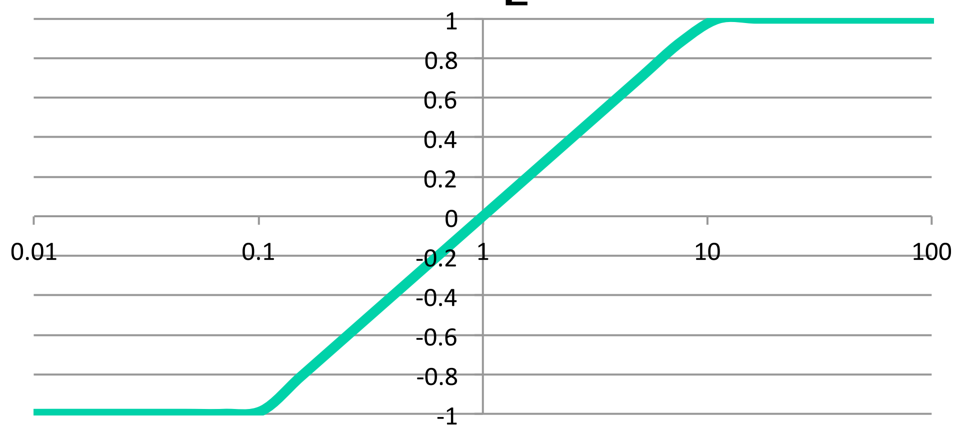


- Falls off gradually
  - $Reward(1000) = 0.0132$

Computing Ideal Load Factor (ILF)

- **Suppose node has count c_l and neighbor has count c_r**
- **Compute *imbalance* as**

$$\beta(c_l, c_r) \quad = \quad Clamp\left[\log_{10}\frac{c_r}{c_l}, -1, +1\right]$$



Computing Ideal Load Factor (cont.)

- **For node *u* with population *p*(*u*)**

$$\hat{\beta}(u) \quad = \quad Avg_{(u,v)\in E}\left[\beta(p(u), p(v))\right]$$

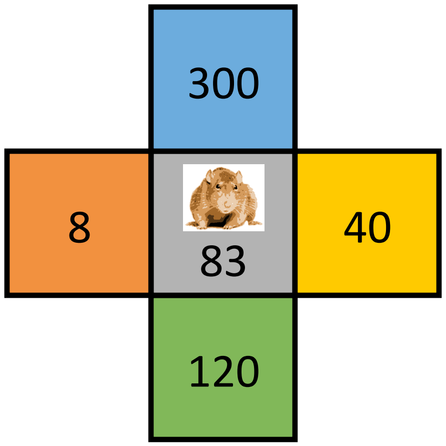
- **Define ILF as**

$$l^{*}(u) \quad = \quad 1.75 + 0.5 \cdot \hat{\beta}(u)$$

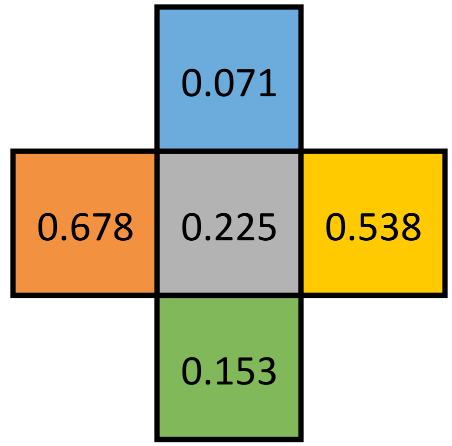
- **Minimum 1.25**
  - When adjacent nodes much less crowded
- **Maximum 2.25**
  - When adjacent nodes much more crowded
- **Changes as rats move around**

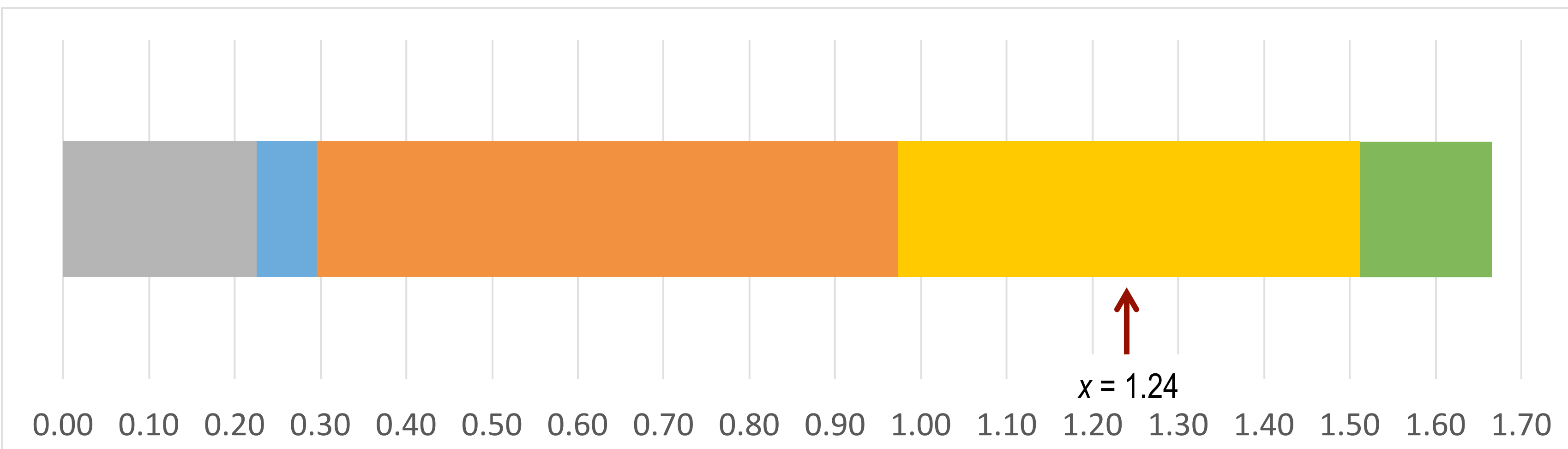
Selecting Next Move

Population



Reward (avg load = 10)





- Choose random number between 0 and sum of rewards
- Move according to interval hit

Update Models

- **Synchronous**
  - Demo 2
  - Compute next positions for all rats, and then move them
  - Causes oscillations/instabilities
- **Rat-order**
  - Demo 3
  - For each rat, compute its next position and then move it
  - Smooth transitions, but costly
- **Batch**
  - Demo 4
  - For each batch of B rats, compute next moves and then move them
  - B = 0.02 * *R*
  - Smooth enough, with better performance possibilities

What We Provide

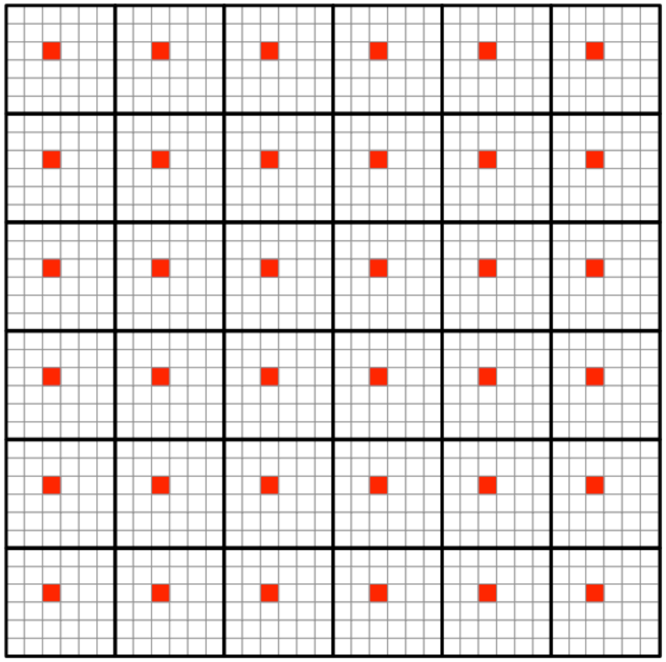
- **Python version of simulator**
  - Demo 4
  - Very slow
- **C version of simulator**
  - Fast sequential implementation
  - Demo 5: 36X36 grid, 1,290 rats
  - Demo 6: 180X180 grid, 1,036,800 rats
    - That's what we'll be using for benchmarks!
- **Generate visualizations by piping C simulator output into Python simulator**
  - Operating in visualization mode
  - See Makefile for examples

Correctness

- **Simulator is Deterministic**
  - Global random seed
  - Random seeds for each rat
  - Process rats in fixed order
- **You Must Preserve Exact Same Behavior**
  - Python simulator generates same result as C simulator
  - Use `regress.py` to check
    - Only checks small cases
    - Useful sanity check
  - Benchmark program compares your results to reference solution
    - Handles full-sized graphs

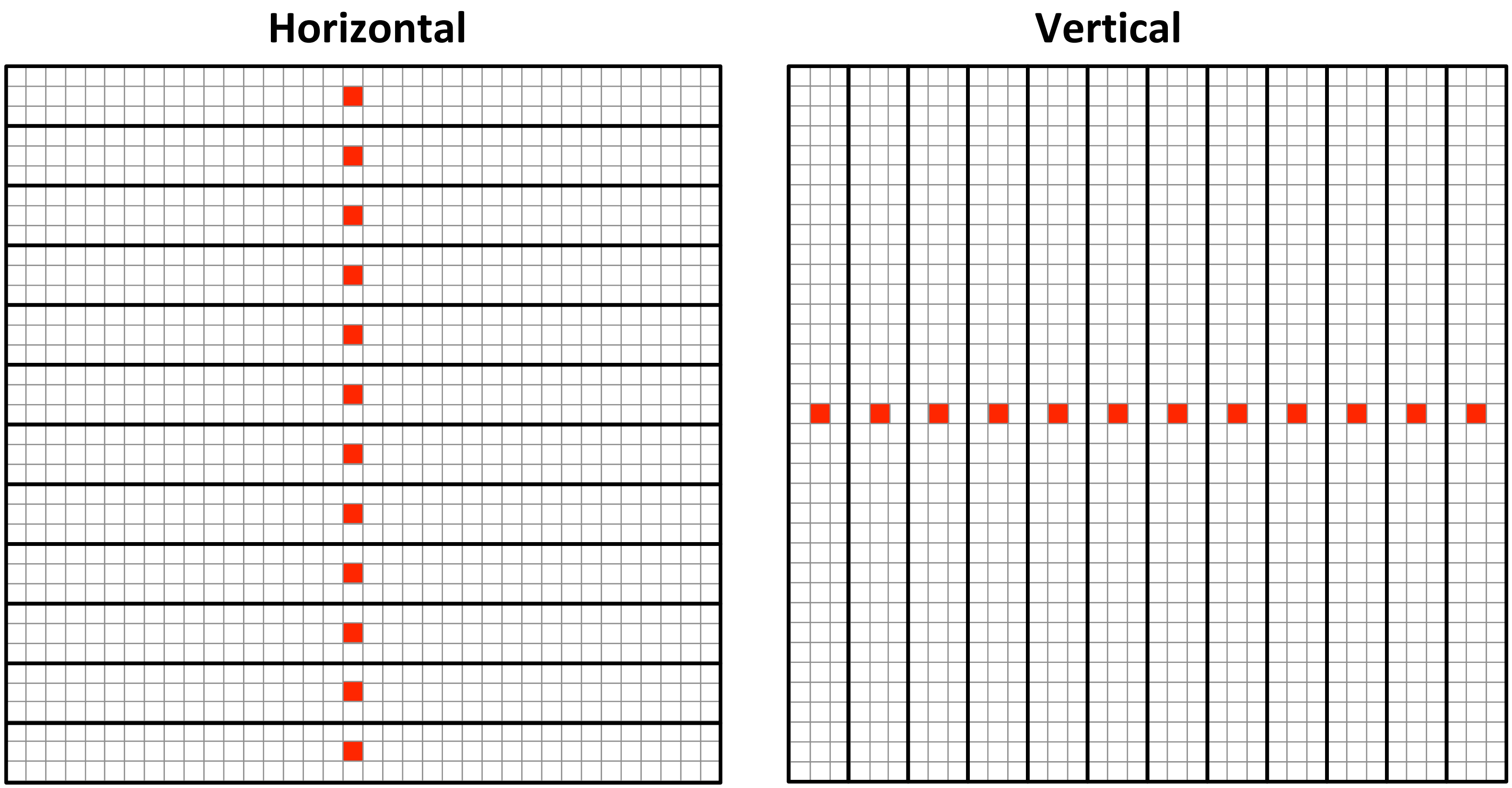
Graphs: Tiled (Demos 1–6)

Rats spread quickly within region
More slowly across regions
Hub nodes tend to have high counts



- **Base grid**
  - K X K nodes, each with nearest neighbor connectivity
- **Hub (red) nodes connect to all other nodes in region**
- **For K = 180**
  - Most nodes have degree ≤ 5
  - Hubs have degree 899

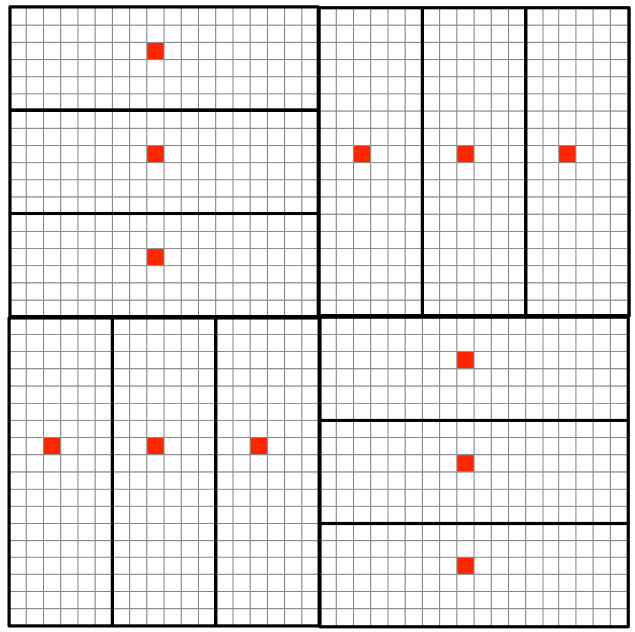
Other graphs



- Larger regions
- k = 180:   Max degree = 2,699

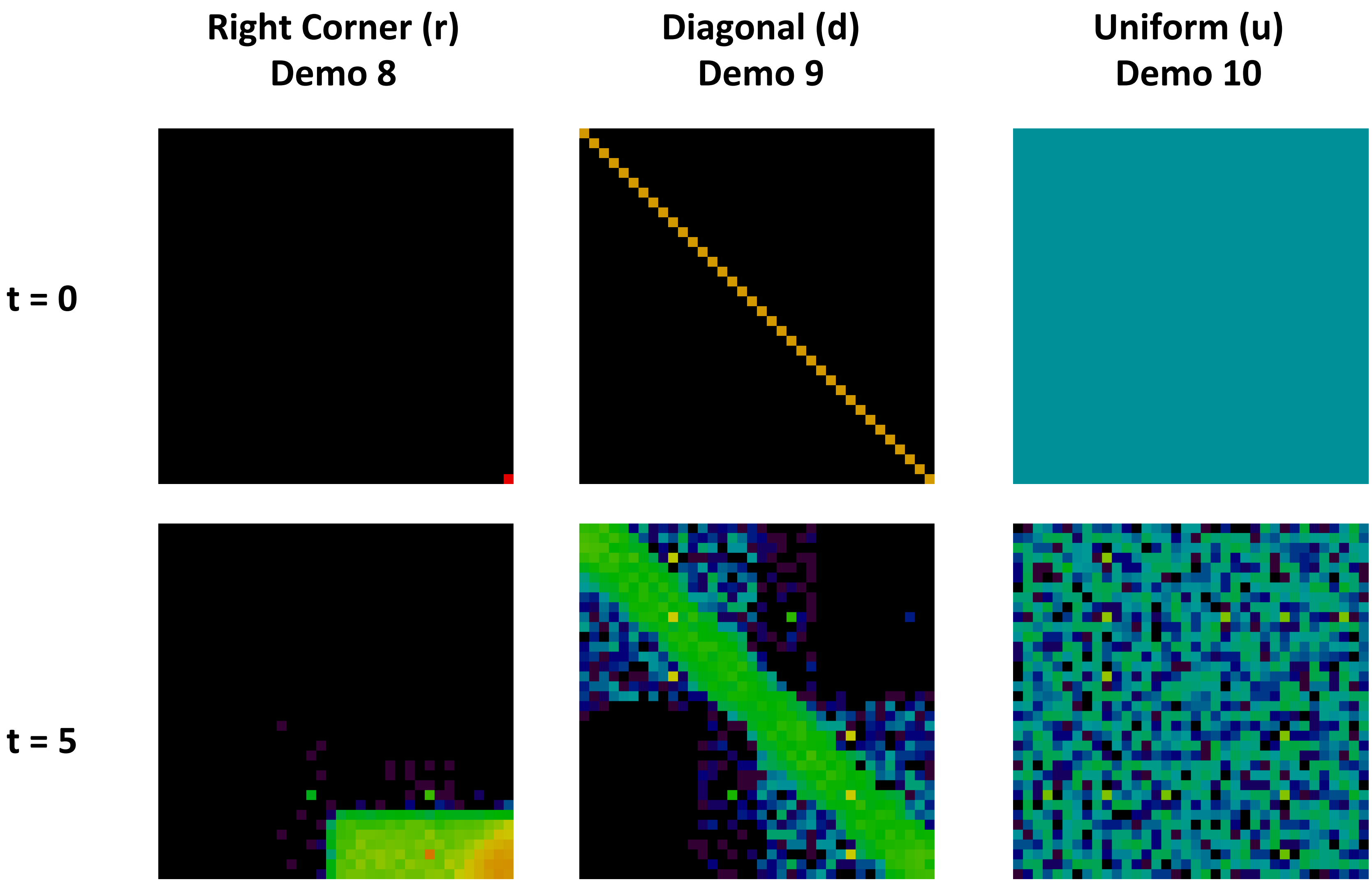
Other graphs

Parquet



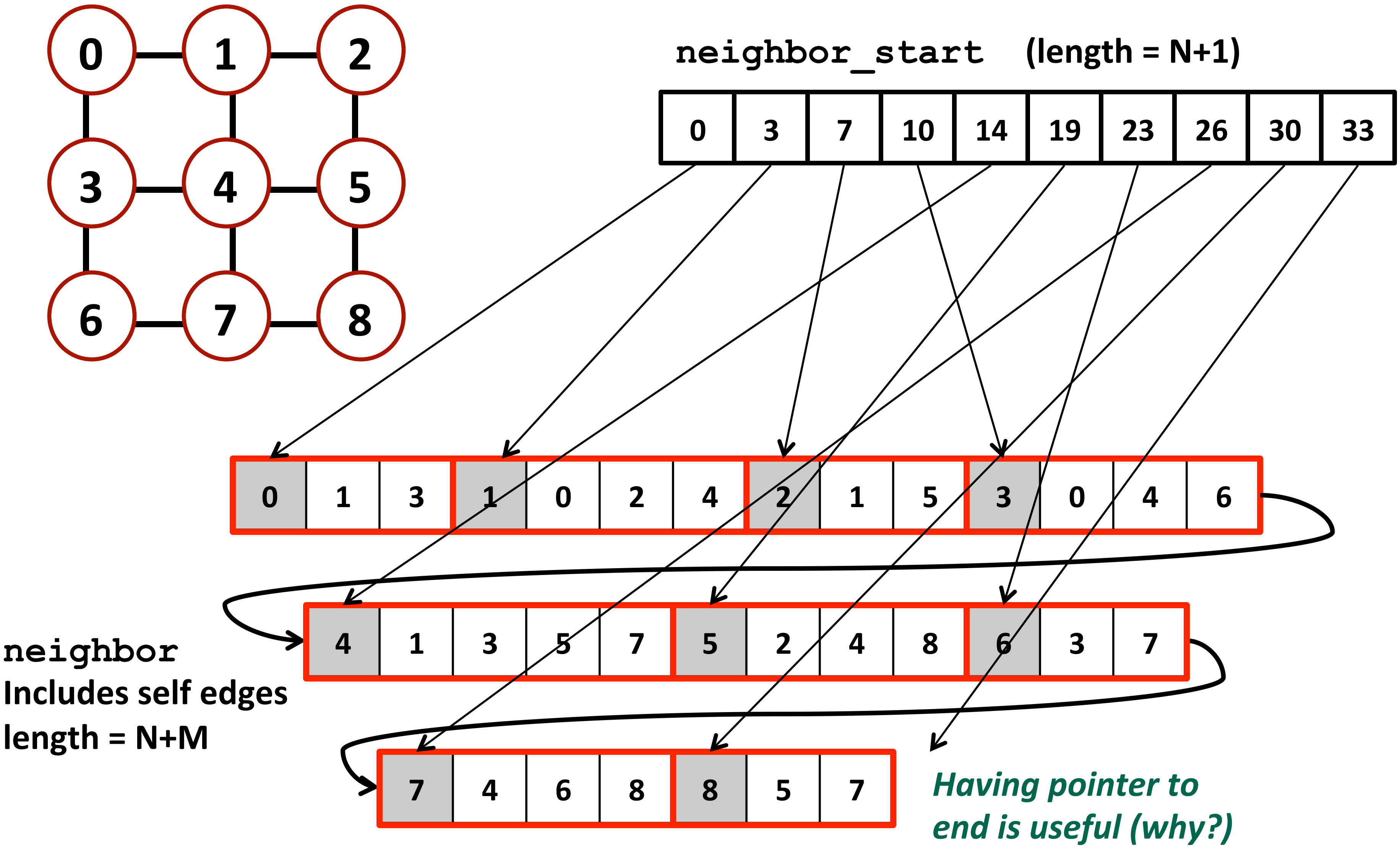
- **Larger regions**
- **k = 180: Max degree = 2,699**

Initial States (Parquet Graph)



Graph Representation

N node, M edges



Sample Code

- From sim.c
- Compute reward value for node

```
/* Compute weight for node nid */
static inline double compute_weight(state_t *s, int nid)
{
    int count = s->rat_count[nid];
    double ilf = neighbor_ilf(s, nid);
    return mweight((double) count/s->load_factor, ilf);
}
```

- Simulation state stored in `state_t` struct
- Reward function computed by `mweight`

Sample Code

- From sim.c
- Compute sum of reward values for node
- Store for later reuse

```
/* Compute sum of weights in region of nid */
static inline double compute_sum_weight(state_t *s, int nid)
{
    graph_t *g    = s->g;
    double sum    = 0.0;
    int eid;
    int eid_start = g->neighbor_start[nid];
    int eid_end   = g->neighbor_start[nid+1];
    for (eid = eid_start; eid < eid_end; eid++) {
        int nbrnid = g->neighbor[eid];
        double w = compute_weight(s, nbrnid);
        s->node_weight[nbrnid] = w;
        sum += w;
    }
    return sum;
}
```

Sample Code

- Compute next move for rat

```
static inline int next_random_move(state_t *s, int r)
{
    int nid          = s->rat_position[r];
    random_t *seedp = &s->rat_seed[r];
    double tsum      = compute_sum_weight(s, nid);
    graph_t *g       = s->g;
    double val       = next_random_float(seedp, tsum);
    double psum      = 0.0;
    int eid;
    int eid_start    = g->neighbor_start[nid];
    int eid_end      = g->neighbor_start[nid+1];
    for (eid = eid_start; eid < eid_end; eid++) {
        psum += s->node_weight[neighbor[eid]];
        if (val < psum) {
            return g->neighbor[eid];
        }
    }
}
```

Sequential Efficiency Considerations

- **Consider move computation for rat at node with degree D**
  - How many (on average) iterations of loop in `next_random_move`?
  - Is there a better way?
- **Provided code uses many optimizations**
  - Precompute weights at start of batch
  - Fast search

Finding Parallelism

- **Sequential constraints**
  - Must complete time steps sequentially
  - Must complete each batch before starting next
    - ILF values and weights then need to be recomputed
- **Sources of parallelism**
  - Over nodes
    - Computing ILFs and reward functions
  - Over rats (within a batch)
    - Computing next moves
    - Updating node counts

Performance Measurements

- **Nanoseconds per move (NPM)**
  - R rats running for S steps
  - Requires time T
  - NPM = 10^9 * T / (R * S)
  - Reference solution:
    - 665 NPM for 1 thread
    - 84 NPM for 12 threads
    - 7.9 X speedup

Performance Targets

- **Benchmarks**
  - 6 combinations of graph/initial state
  - Each counts 15 points
- **Target performance**
  - T = measured time
  - T_r = time for reference solution
  - T_r / T = How well you reach reference solution performance
    - Full credit when ≥ 0.9
    - Partial when ≥ 0.5

Machines

- **Latedays cluster**
  - 16 worker nodes + 1 head node
  - Each is 12-core Xeon processor (dual socket with 6 cores each)
  - You submit jobs to batch queue
  - Assigned single processor for entire run
  - Python script provided
- **Code Development**
  - OK to do code development and testing on other machines
  - But, they have different performance characteristics
  - Make sure to use 6 or 12 threads to ensure correct partitioning of nodes across processors

Instrumenting Your Code

- **How do you know how much time each activity takes?**
  - Create simple library using cycletimer code
  - Bracket steps in your code with library calls
  - Use macros so that you can disable code for maximum performance

```
START_ACTIVITY(ACTIVITY_NEXT);
#pragma omp parallel for schedule(static)
for (ri = 0; ri < local_count; ri++) {
    int rid = ri + local_start;
    s->rat_position[rid] = fast_next_random_move(s, rid);
}
FINISH_ACTIVITY(ACTIVITY_NEXT);
```

Evaluating Your Instrumented Code

1 thread

```
   194 ms     1.0 %    startup
  2077 ms    11.1 %    compute_weights
  4029 ms    21.6 %    compute_sums
 11733 ms    62.8 %    find_moves
   651 ms     3.5 %    set_ops
     3 ms     0.0 %    unknown
```

12 threads

```
   192 ms     3.2 %    startup
   426 ms     7.0 %    compute_weights
   940 ms    15.5 %    compute_sums
  3168 ms    52.3 %    find_moves
  1325 ms    21.9 %    set_ops
     2 ms     0.0 %    unknown
```

- Can see which activities account for most time
- Can see which activities limit parallel speedup

Some Logos

GraphLab
Carnegie Mellon

GraphChi: Going small with GraphLab

GraphRats
Carnegie Mellon University