Assignment 3: GraphRats

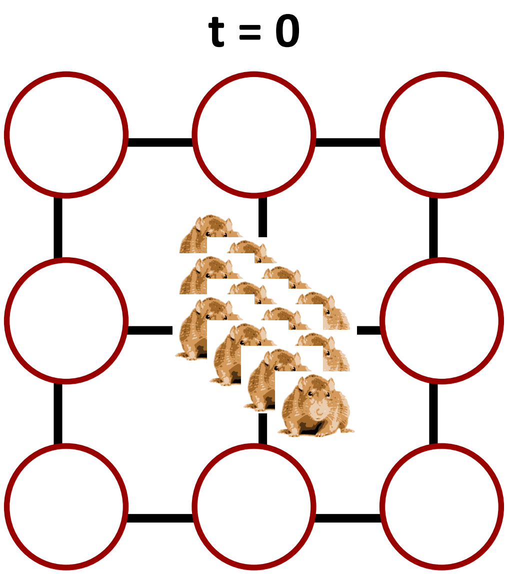
GraphRats 3.0

Carnegie Mellon University

Topics

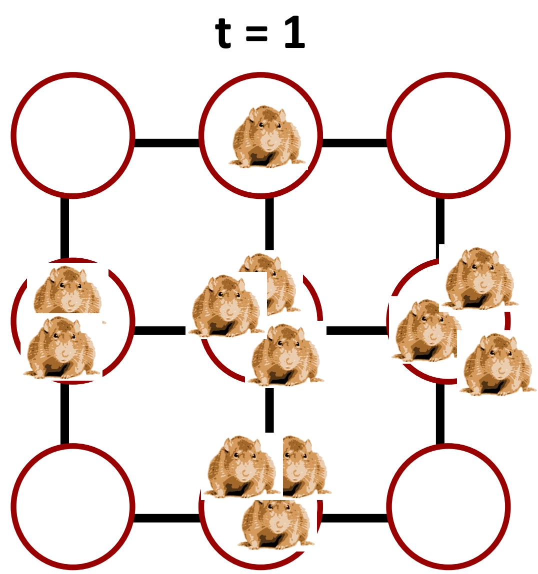
- Application
- Implementation Issues
- Optimizing for Parallel Performance
- Useful Advice

Basic Idea



- **Graph**
  - *W* X *H* grid
- **Initial State**
  - Start with all *R* rats in center

- **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 (18 x 12)

```
t = 0.
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |1080|    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

Simulation Example (18 x 12)

```
t = 1.
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    | 137|    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    | 144|  7 | 145|    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    | 147|    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    | 164|    |    |    | 168|    |    |    | 168|    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

- **Note moves to nodes not connected by grid**
  - Explanation to follow

Simulation Example (18 x 12)

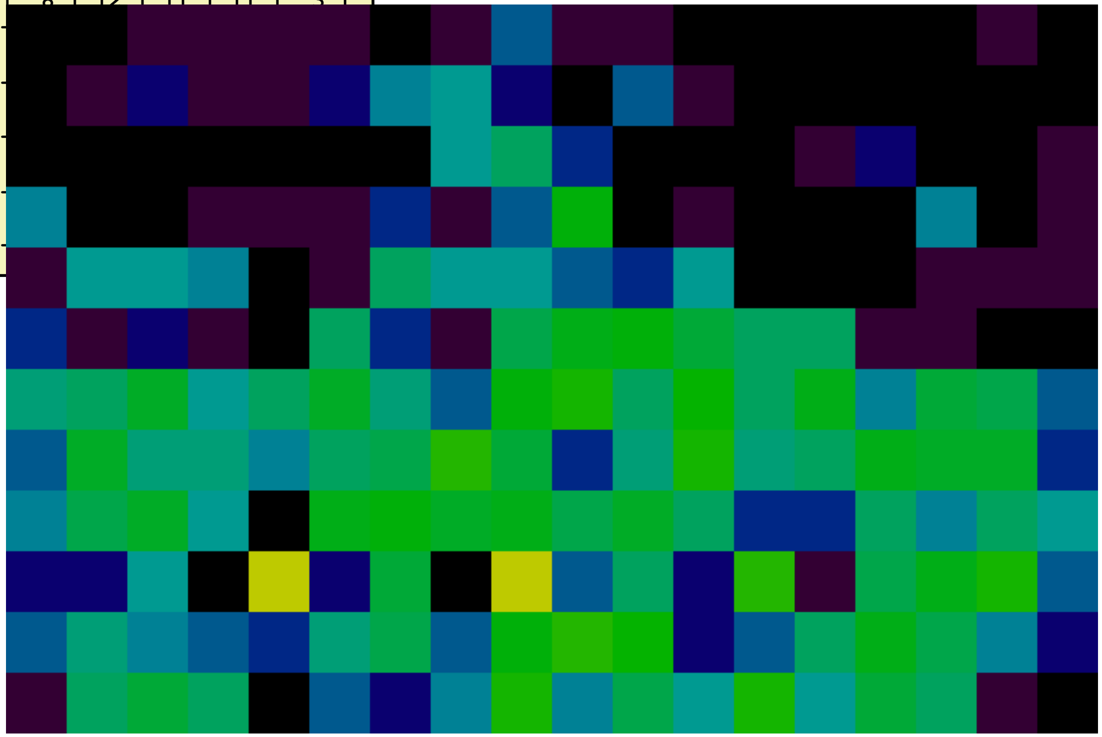
```
t = 20.
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |  1 |  1 |  1 |  1 |    |  1 |  4 |  1 |  1 |    |    |    |    |    |  1 |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |  1 |  2 |  1 |  1 |  2 |  5 |  6 |  2 |    |  4 |  1 |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |  6 |  8 |  3 |    |    |    |  1 |  2 |    |    |  1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  5 |    |    |  1 |  1 |  1 |  3 |  1 |  4 | 13 |    |  1 |    |    |    |  5 |    |  1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  1 |  6 |  6 |  5 |    |  1 |  8 |  6 |  6 |  4 |  3 |  6 |    |    |    |  1 |  1 |  1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  3 |  1 |  2 |  1 |    |  8 |  3 |  1 |  9 | 12 | 13 | 10 |  8 |  8 |  1 |  1 |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  7 |  8 | 11 |  6 |  8 | 11 |  7 |  4 | 13 | 15 |  8 | 14 |  8 | 12 |  5 | 10 |  9 |  4 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  4 | 11 |  7 |  7 |  5 |  8 |  9 | 16 | 10 |  3 |  7 | 15 |  7 |  8 | 12 | 11 | 11 |  3 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  5 |  9 | 11 |  6 |    | 12 | 13 | 11 | 12 |  9 | 11 |  8 |  3 |  3 |  8 |  5 |  8 |  6 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  2 |  2 |  6 |    | 31 |  2 | 10 |    | 31 |  4 |  8 |  2 | 16 |  1 |  9 | 12 | 15 |  4 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  4 |  7 |  5 |  4 |  3 |  7 |  9 |  4 | 13 | 16 | 14 |  2 |  4 |  8 | 12 |  9 |  5 |  2 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  1 |  8 | 10 |  8 |    |  4 |  2 |  5 | 15 |  5 |  9 |  6 | 15 |  6 | 10 |  8 |  1 |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
```

- **Rats dispersed across graph**

Visualizations

```
t = 20.
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |  1 |  1 |  1 |  1 |    |  1 |  4 |  1 |  1 |    |    |    |    |    |  1 |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |  1 |  2 |  1 |  1 |  2 |  5 |  6 |  2 |    |  4 |  1 |    |    |    |    |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|    |    |    |    |    |    |    |  6 |  8 |  3 |    |    |    |  1 |  2 |    |    |  1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  5 |    |    |  1 |  1 |  1 |  3 |  1 |  4 | 13 |    |  1 |    |    |    |  5 |    |  1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  1 |  6 |  6 |  5 |    |  1 |  8 |  6 |  6 |  4 |  3 |  6 |    |    |    |  1 |  1 |  1 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  3 |  1 |  2 |  1 |    |  8 |  3 |  1 |  9 | 12 | 13 | 10 |  8 |  8 |  1 |  1 |    |    |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  7 |  8 | 11 |  6 |  8 | 11 |  7 |  4 | 13 | 15 |  8 | 14 |  8 | 12 |  5 | 10 |  9 |  4 |
+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+----+
|  4 | 11 |  7 |  7 |  5 |  8 |  9 | 16 | 10 |  3 |  7 | 15 |  7 |  8 | 12 | 11 | 11 |  3 |
+----+----+----+----+----+----+----+----+----+----+----+----+----
|  5 |  9 | 11 |  6 |    | 12 | 13 | 11 | 12 |  9 | 11 |  8 |  3
+----+----+----+----+----+----+----+----+----+----+----+----+----
|  2 |  2 |  6 |    | 31 |  2 | 10 |    | 31 |  4 |  8 |  2 | 16
+----+----+----+----+----+----+----+----+----+----+----+----+----
|  4 |  7 |  5 |  4 |  3 |  7 |  9 |  4 | 13 | 16 | 14 |  2 |  4
+----+----+----+----+----+----+----+----+----+----+----+----+----
|  1 |  8 | 10 |  8 |    |  4 |  2 |  5 | 15 |  5 |  9 |  6 | 15
+----+----+----+----+----+----+----+----+----+----+----+----+----
```

Text ("a" for ASCII)



Heat Map ("h")

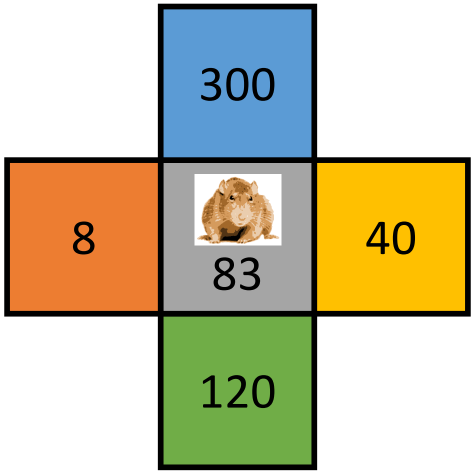
Running it yourself

```
linux> cd some directory
linux> git clone https://github.com/cmu15418/asst3-s20.git
linux> cd asst3-s20/code
Linux> make demoX
```

X from 1 to 11

- **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)

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

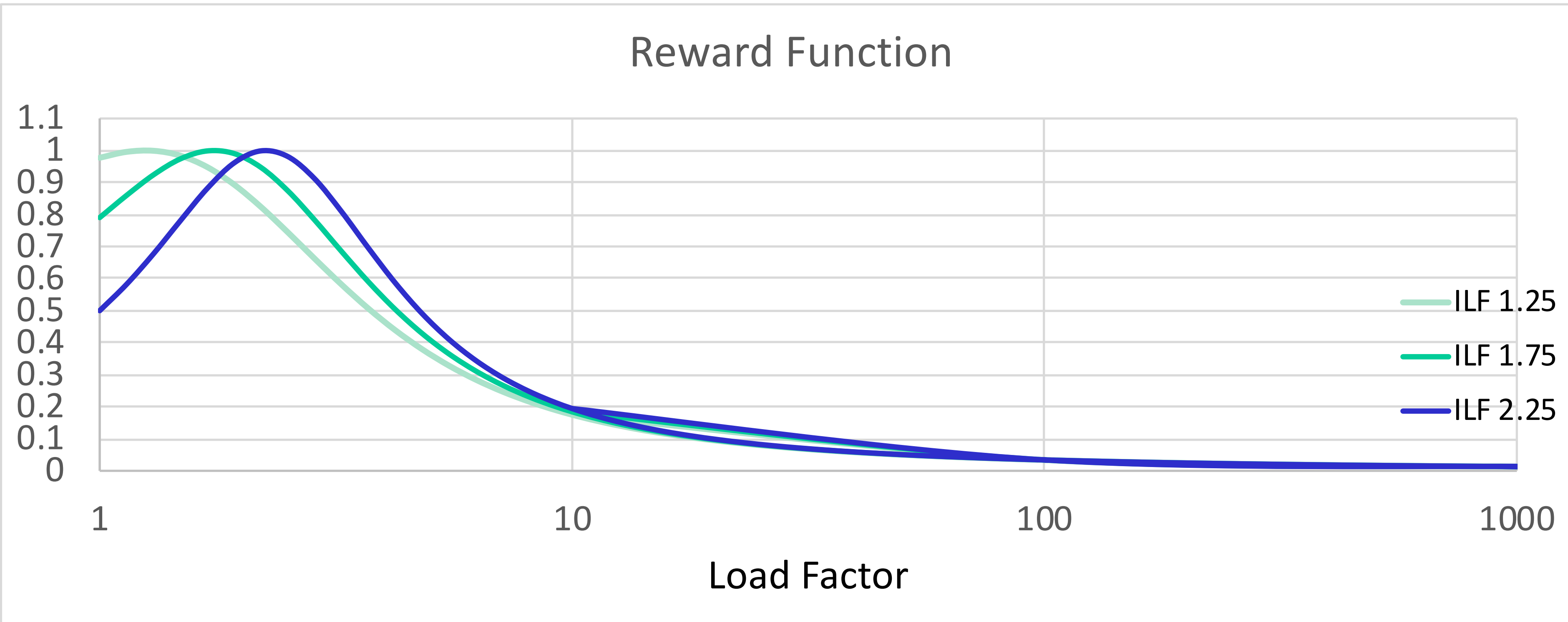
Reward Function

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

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

Reward Function (cont.)

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



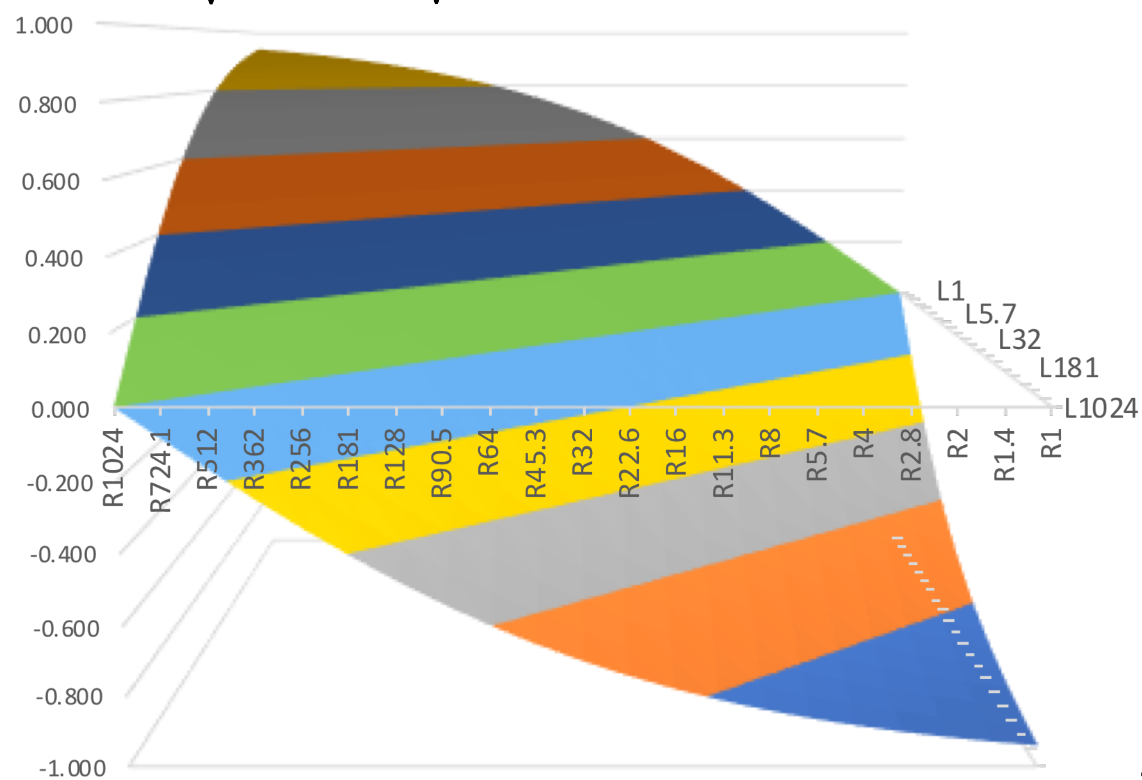
- Falls off gradually
  - $Reward(1000) = 0.0132$

Computing Ideal Load Factor (ILF)

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

$$\beta(c_l, c_r) = \frac{\sqrt{c_r} - \sqrt{c_l}}{\sqrt{c_l} + \sqrt{c_r}}$$

- **Maximum +1** $c_r \gg c_l$
- **Minimum –1** $c_l \gg c_r$



Computing Ideal Load Factor (cont.)

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

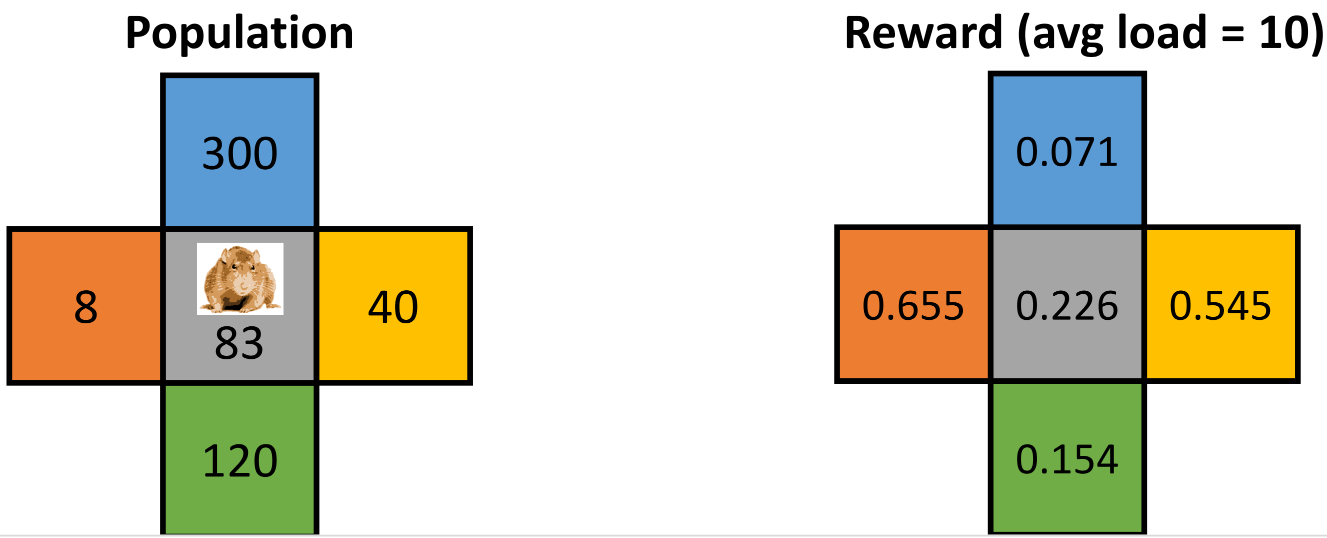
$$\hat{\beta}(u) \quad = \quad Avg_{(u,v)\in E}\left[\beta(p(u), p(v))\right]$$

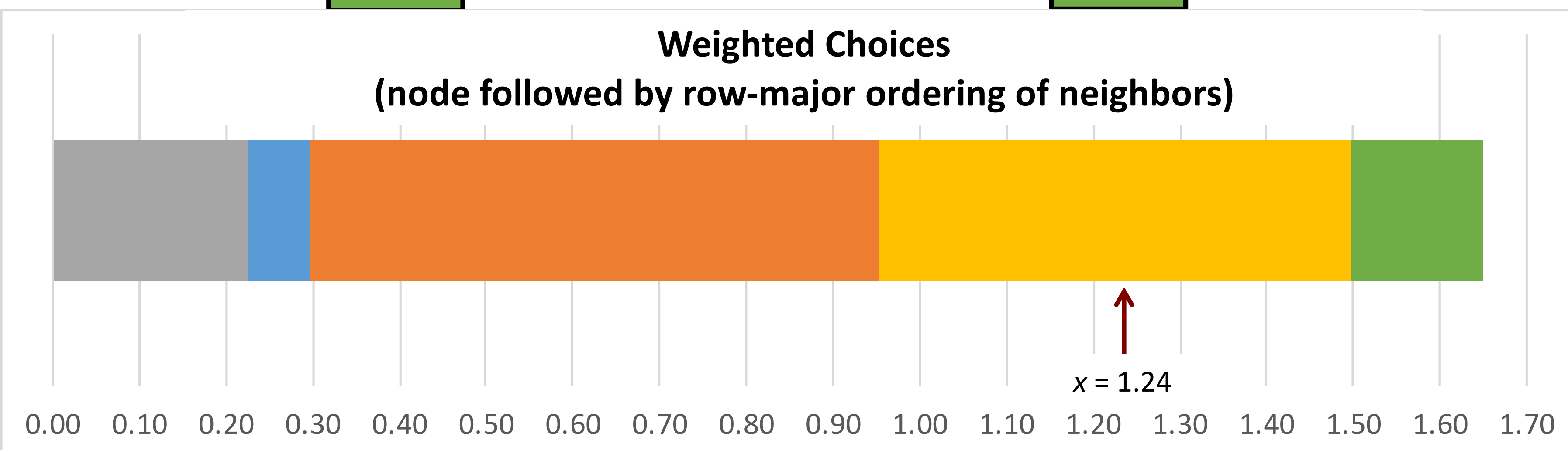
- **Define ILF as**

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

- **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





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

Update Models

- **Synchronous**
  - Demo 3
  - Compute next positions for all rats, and then move them
  - Causes oscillations/instabilities
- **Rat-order**
  - Demo 4
  - For each rat, compute its next position and then move it
  - Smooth transitions, but costly
- **Batch**
  - Demo 5
  - For each batch of B rats, compute next moves and then move them
  - $B = 0.02 * R$
  - Smooth enough, with better performance possibilities

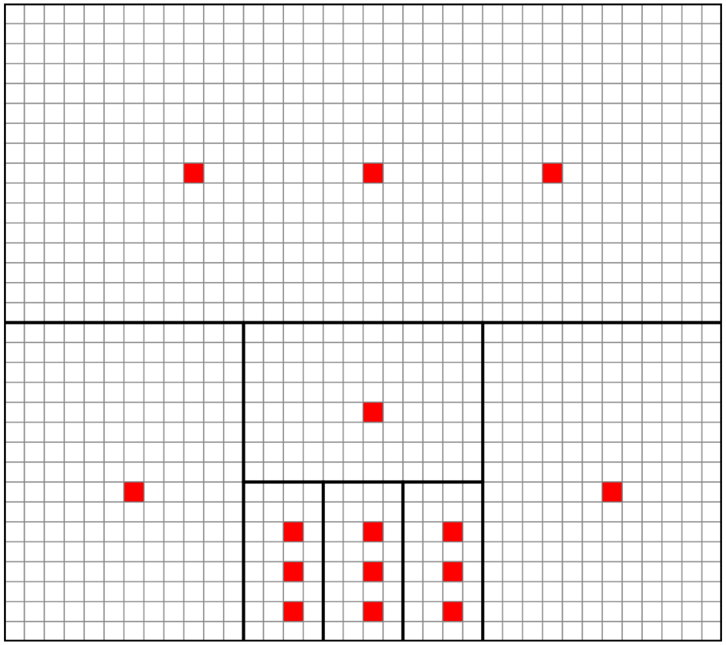
What We Provide

- **Python version of simulator**
  - Demos 1–2
  - Very slow
- **C version of simulator**
  - Fast sequential implementation
  - Demos 3–5: 36X32 grid, 11,520 rats
  - Demos 6–11: 180X160 grid, 1,008,000 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

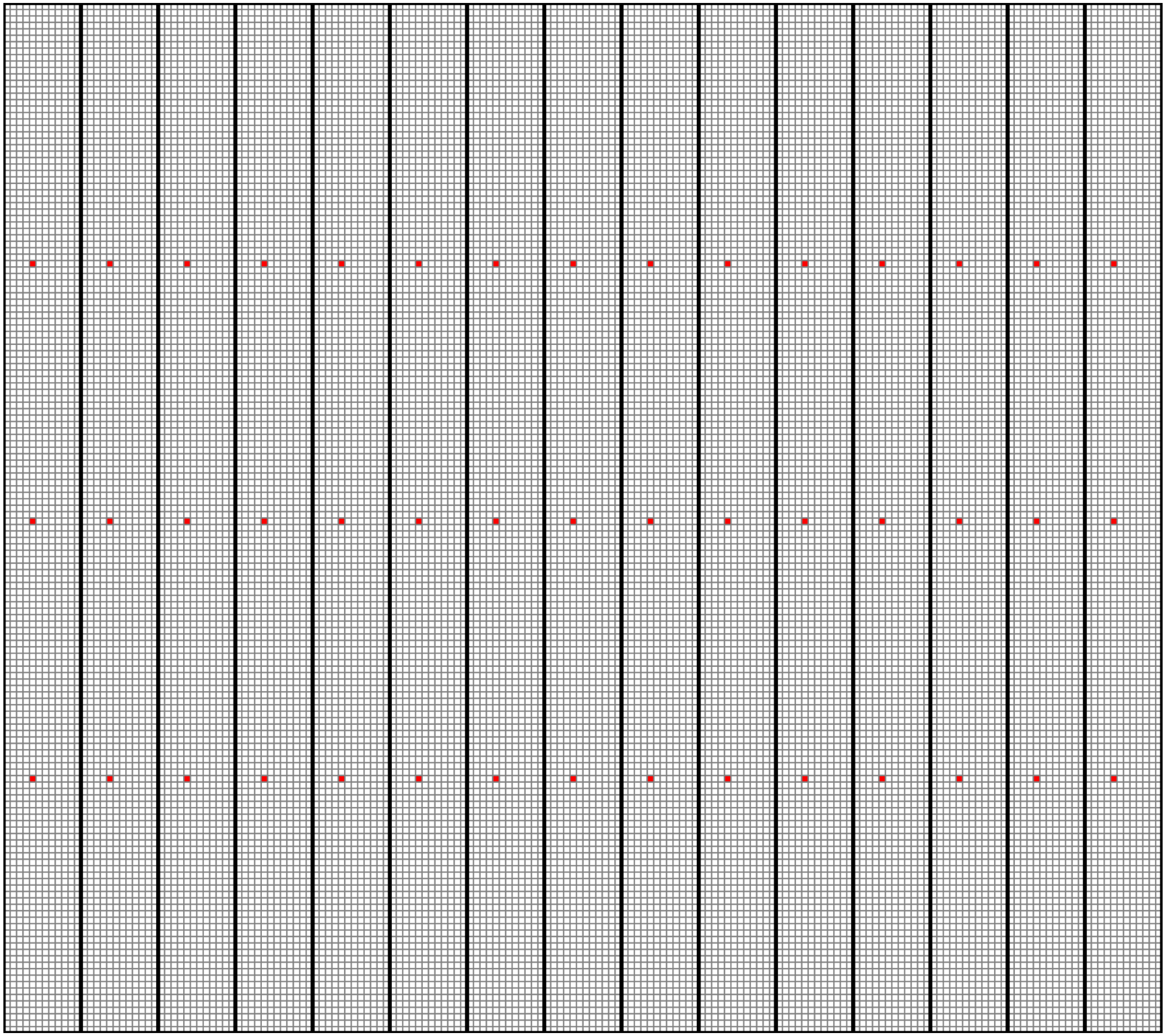
Fractal Graph fracZ (Demos 1–5)



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

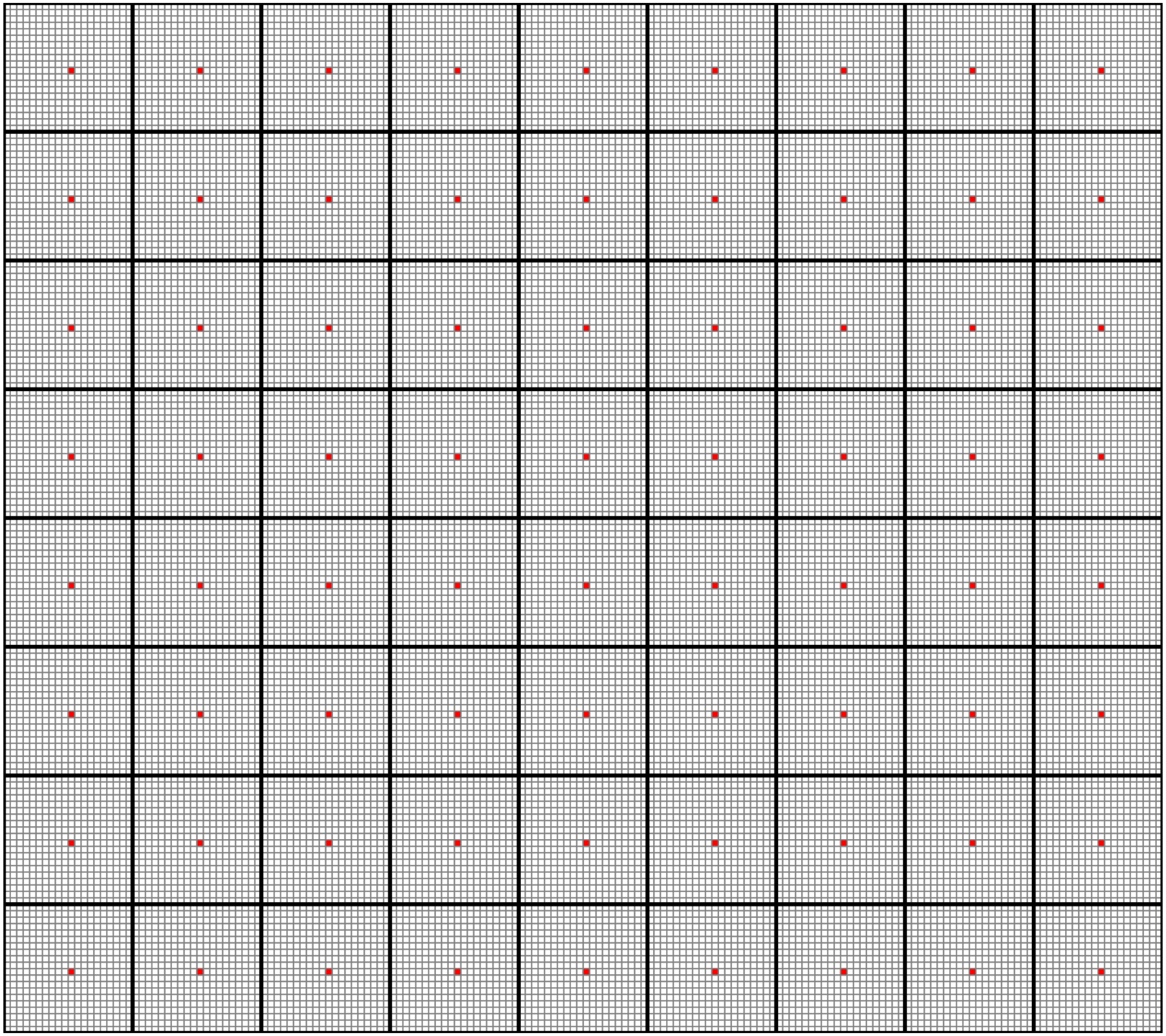
- **Base grid**
  - *W* X *H* nodes, each with nearest neighbor connectivity
- **Regions**
  - Recursively partition rectangles into two or three subrectangles
  - Leaves of tree form regions
- **Hubs**
  - Connect to every other node in region
  - Each region has one or three hubs

Benchmark Graph UniA (Demo 7)



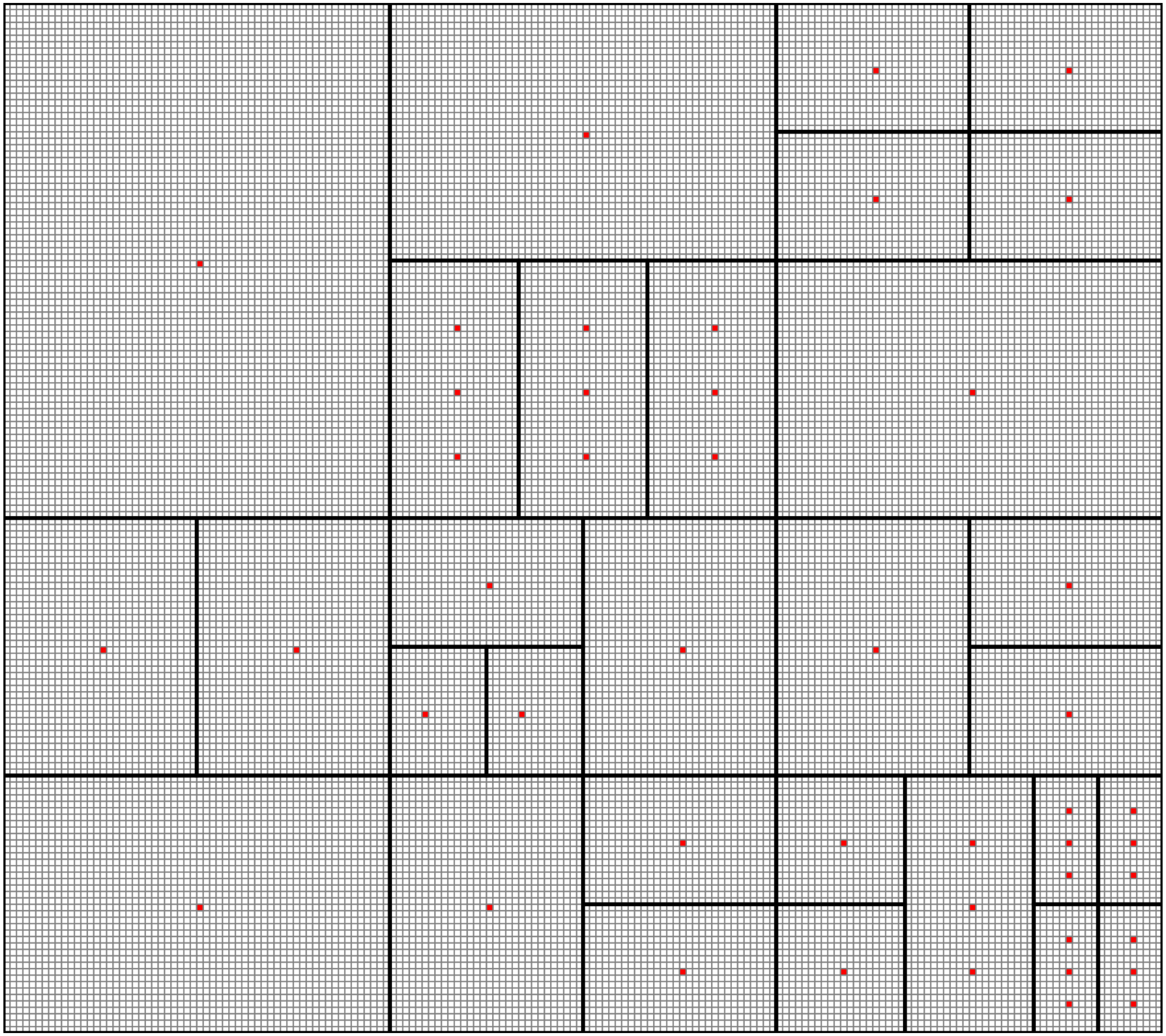
Nodes	28,800
Edges	286,780
Regions	15
Hubs	45
Max Degree	1,919

Benchmark Graph UniB (Demo 8)



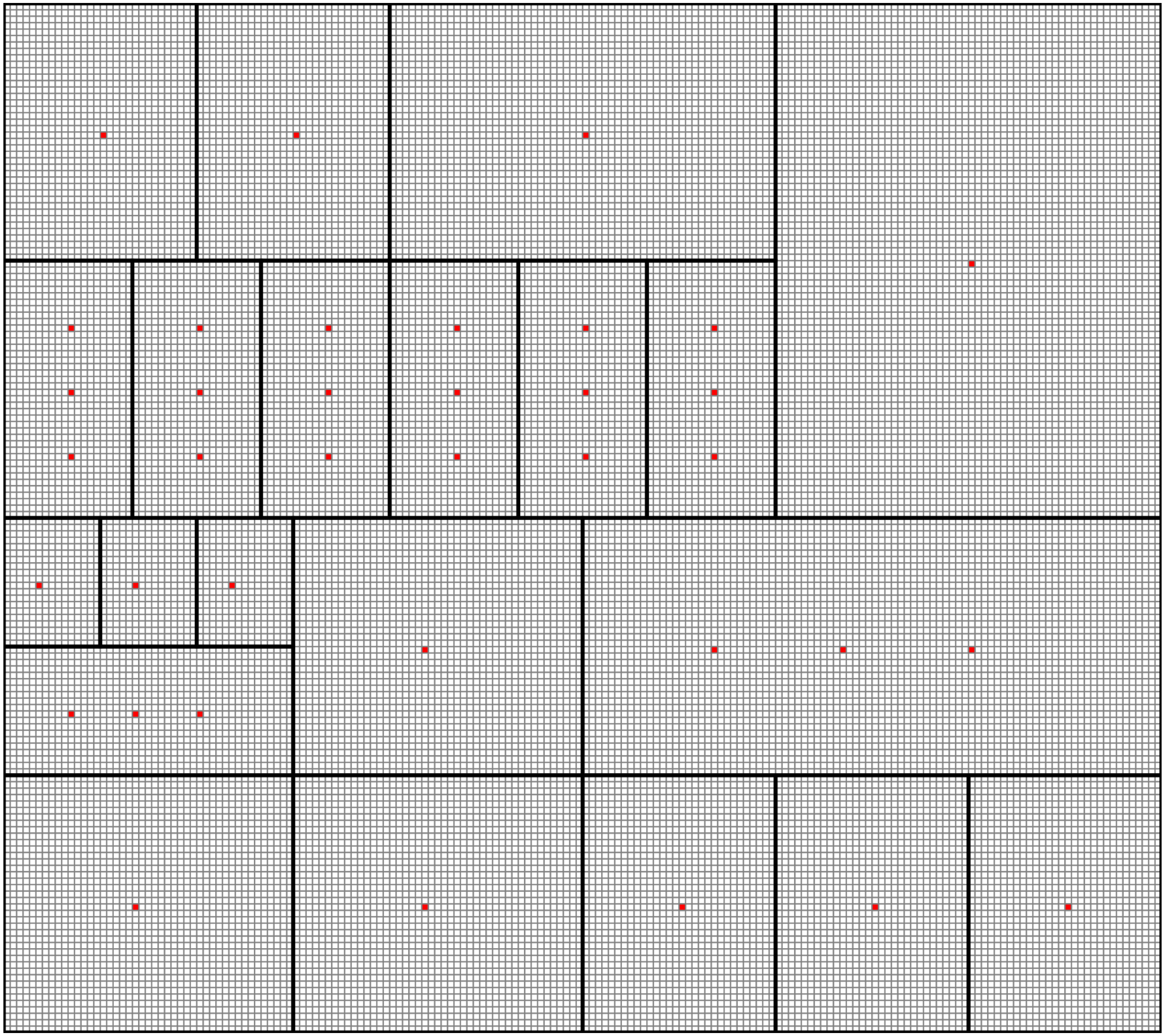
Nodes	28,800
Edges	171,400
Regions	72
Hubs	72
Max Degree	399

Benchmark Graph FracC (Demos 9–10)



Nodes	28,800
Edges	187,612
Regions	30
Hubs	46
Max Degree	4,899

Benchmark Graph FracD



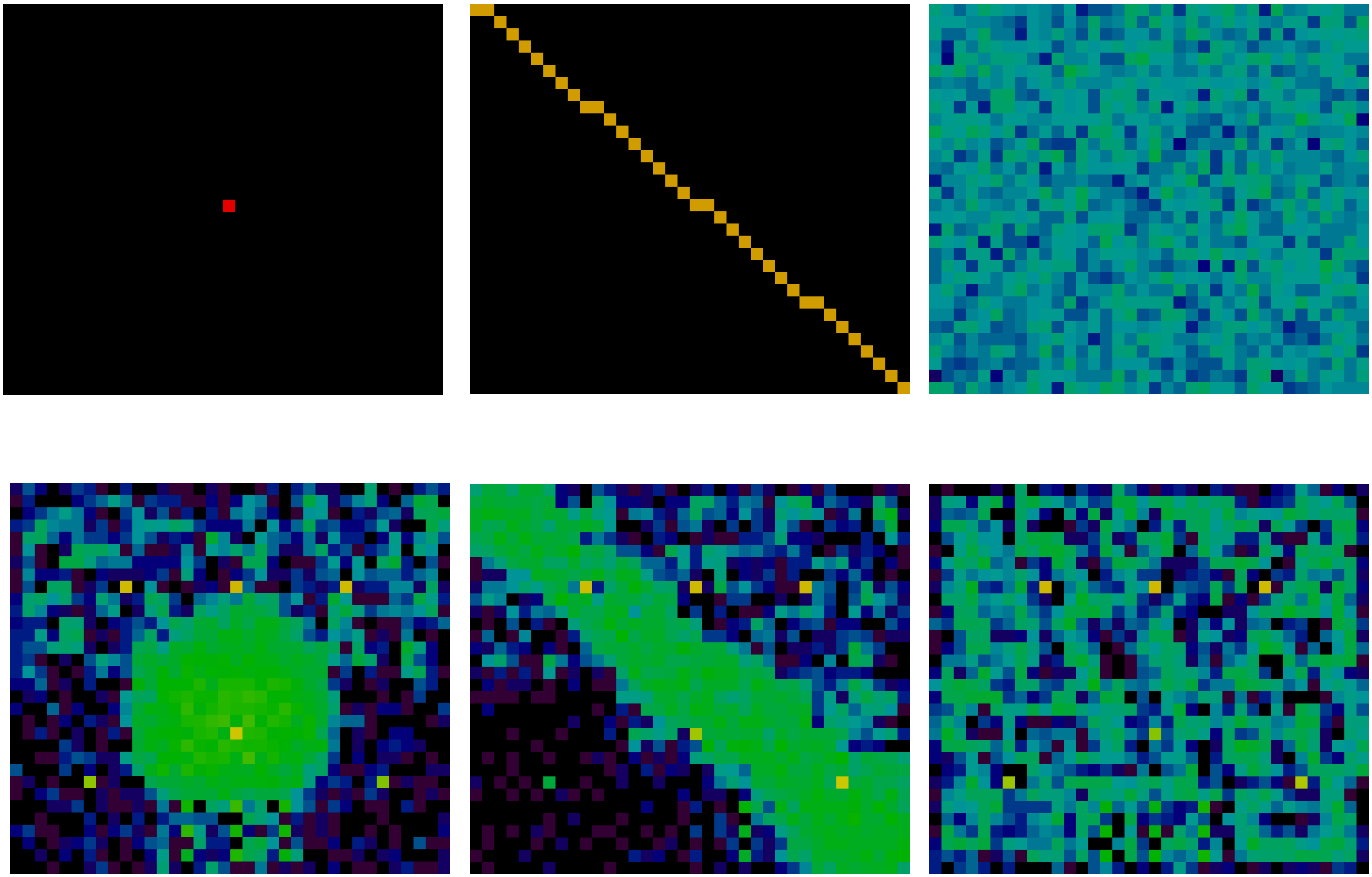
Nodes	28,800
Edges	208,902
Regions	21
Hubs	37
Max Degree	4,899

Initial States (fracZ)

	Center (r) ~Demo 7	Diagonal (d) ~Demo 9	Random (u) ~Demo 10

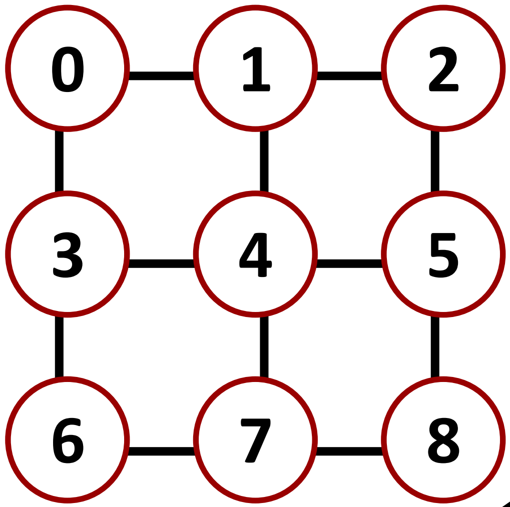
t = 0

t = 20

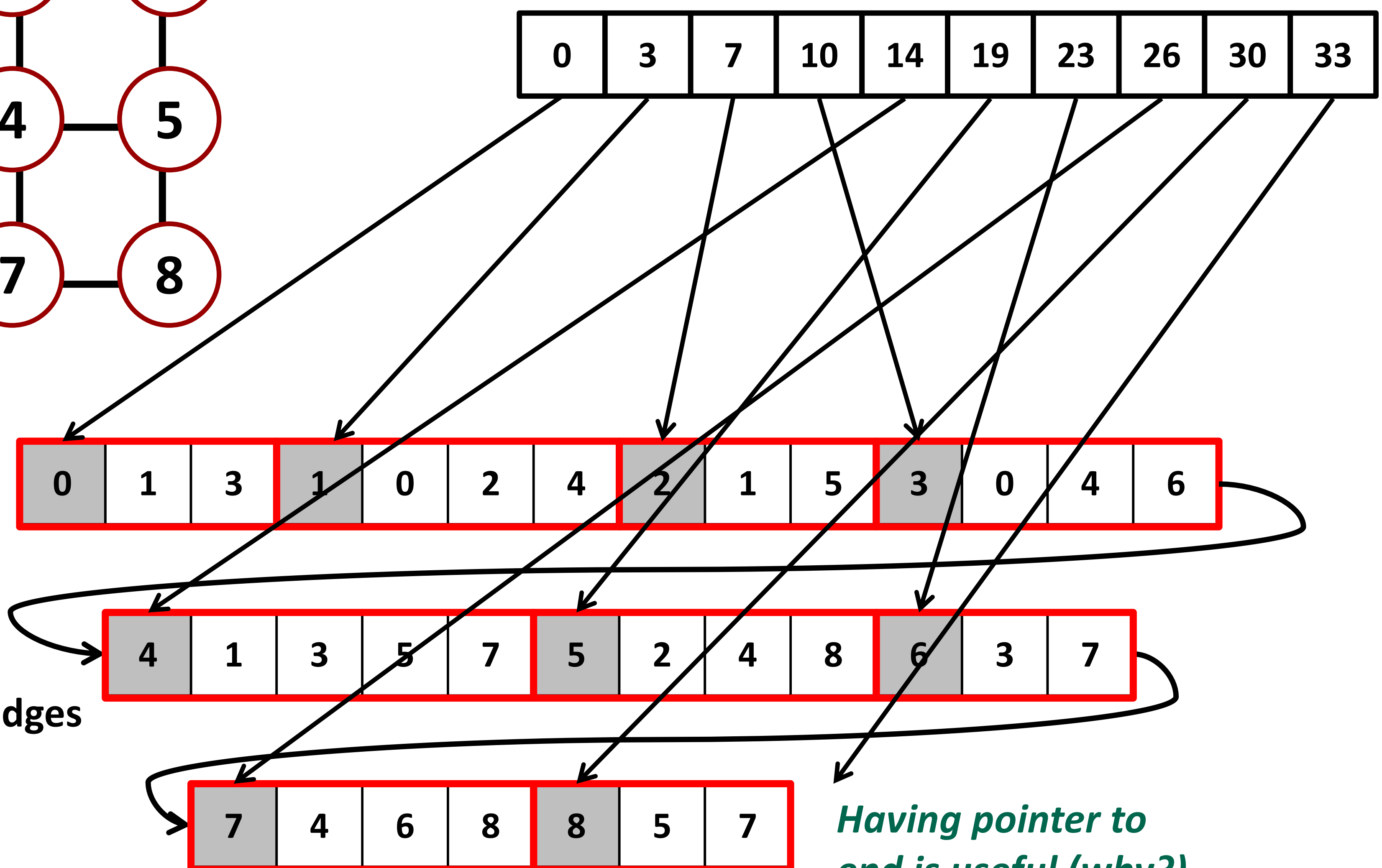


Graph Representation

N node, M edges



`neighbor_start` **(length = N+1)**



`neighbor`

Includes self edges

length = N+M

Having pointer to end is useful (why?)

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 reward value for all nodes

```
static inline void compute_all_weights(state_t *s)  {
    graph_t *g = s->g;
    double *node_weight = s->node_weight;
    int nid;
    for (nid = 0; nid < g->nnode; nid++)
        node_weight[nid] = compute_weight(s, nid);
}
```

- Simulation state stored in `state_t` struct

Sample Code

- From sim.c
- Compute sum of reward values for node
- Store cumulative value for each edge
- Store total sum for later reuse

```
static inline void find_all_sums(state_t *s) {
    graph_t *g = s->g;
    int nid, eid;
    for (nid = 0; nid < g->nnode; nid++) {
        double sum = 0.0;
        for (eid = g->neighbor_start[nid];
             eid < g->neighbor_start[nid+1];
             eid++) {
            sum += s->node_weight[g->neighbor[eid]];
            s->neighbor_accum_weight[eid] = sum;
        }
        s->sum_weight[nid] = sum;
    }
}
```

Sample Code

- Compute next move for rat

```
static inline int fast_next_random_move(state_t *s, int r) {
    int nid = s->rat_position[r];
    graph_t *g = s->g;
    random_t *seedp = &s->rat_seed[r];
    double tsum = s->sum_weight[nid];
    double val = next_random_float(seedp, tsum);
    int estart = g->neighbor_start[nid];
    int elen = g->neighbor_start[nid+1] - estart;

    /* Find location by binary search */
    int offset = locate_value(val,
                              &s->neighbor_accum_weight[estart],
                              elen);

    return g->neighbor[estart + offset];
}
```

Instrumented Code

- From sim.c
- Wrap major sections with instrumentation macros

```
static inline void find_all_sums(state_t *s) {
    graph_t *g = s->g;
    START_ACTIVITY(ACTIVITY_SUMS);
    int nid, eid;
    for (nid = 0; nid < g->nnode; nid++) {
        double sum = 0.0;
        for (eid = g->neighbor_start[nid];
             eid < g->neighbor_start[nid+1];
             eid++) {
            sum += s->node_weight[g->neighbor[eid]];
            s->neighbor_accum_weight[eid] = sum;
        }
        s->sum_weight[nid] = sum;
    }
    FINISH_ACTIVITY(ACTIVITY_SUMS);
}
```

Running Instrumented Code

- **Demo 11**

```
./crun-seq -g data/g-180x160-fracC.gph
    -r data/r-180x160-r35.rats -u b -n 50 -I -q

50 steps, 1008000 rats, 13.770 seconds

        228 ms      1.6 %    startup
      10677 ms     76.3 %    compute_weights
        750 ms      5.4 %    compute_sums
       2340 ms     16.7 %    find_moves
          2 ms      0.0 %    unknown
      13998 ms    100.0 %    elapsed
```

- **Shows breakdown of where time spent**
- **See speedups of different parts of code**
- **Can instrument both your code & reference version**

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
    - Computing cumulative sums
  - 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 seconds
  - NPM = 10^9 * T / (R * S)
  - Reference solution:
    - Average 290 NPM for 1 thread
    - Average 44.6 NPM for 12 threads
    - Speedups:
      - UniA 6.78
      - UniB 6.43
      - FracC 6.45
      - FracD 6.55
  - Maybe you can do better!

Performance Targets

- **Benchmarks**
  - 4 combinations of graph/initial state
  - Each counts 16 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 ≥ 0.95
    - Partial when ≥ 0.60

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
  - Max threads on GHC cluster = 8
  - Code should run for any number of threads (up to machine limit)

Some Logos

GraphLab
Carnegie Mellon

GraphChi: Going small with GraphLab

GraphRats
Carnegie Mellon University