Computing small discrete logarithms faster

D. J. Bernstein,
University of Illinois at Chicago &
Technische Universiteit Eindhoven

Tanja Lange,
Technische Universiteit Eindhoven

eprint.iacr.org/2012/458

Privacy for smart meters

2011 Kursawe–Danezis–Kolhweiss:
Provider should
not learn individual
consumptions c_i.

picture:
EVB Energy Ltd

Use DL group $\langle g \rangle$
of prime order ℓ.

Set of users
computes $g^{\sum c_j}$
(including blinding).

Provider knows consumption c.

Checks whether $\log_g(g^{\sum c_j}/g^c)$
lies within a tolerance interval.

Need to solve DL in interval.

Trapdoor DL

Use RSA modulus $n = pq$,
where $p - 1$ and $q - 1$ have
many medium-size factors ℓ_i.

With p, q and the ℓ_i
as trapdoor information,
can compute m from g^m
for specified $g \in (\mathbf{Z}/n)^*$.

Requires computation of DL
in each subgroup of order ℓ_i.

Applications: e.g.,
1991 Maurer–Yacobi IBE;
2010 Henry–Henry–Goldberg.

BGN homomorphic encryption

2005 Boneh–Goh–Nissim:
Can handle adding arbitrary subsets of encrypted data, multiplying the sums, and adding the products.

Uses pairings on elliptic curves.
2010 Freeman: very efficient prime-order version.

Decryption requires computing DL in interval.
Length depends on message size and number of additions.

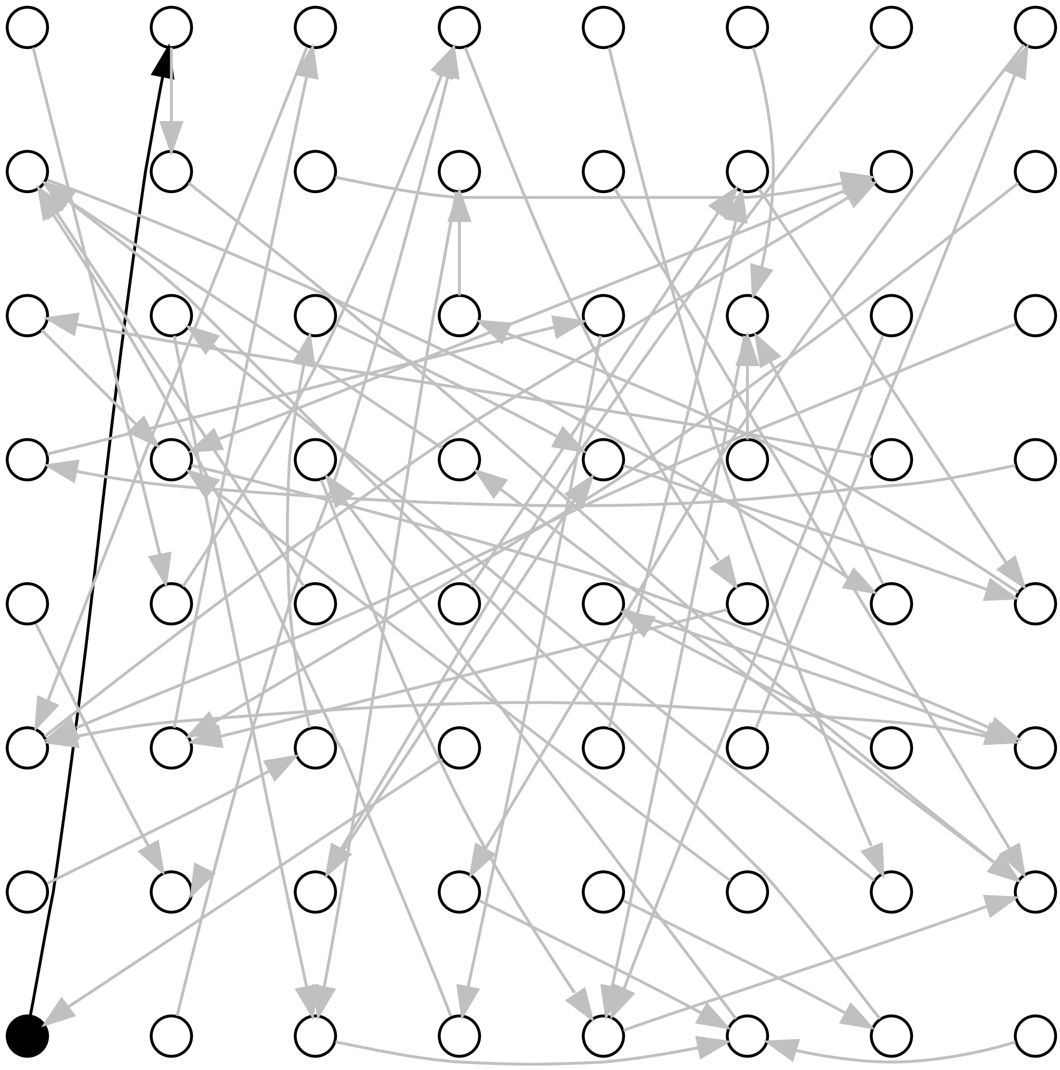
The rho method

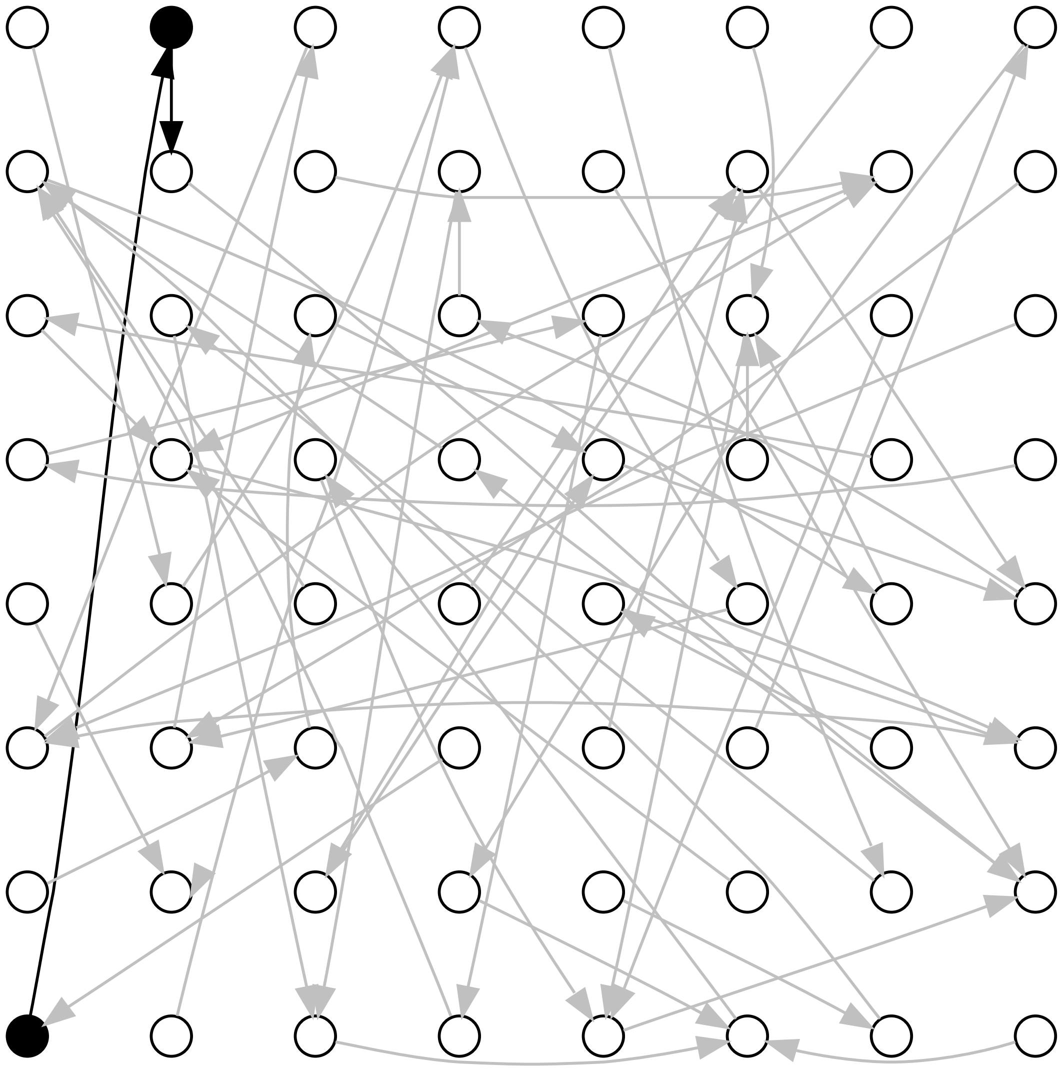
Simplified, non-parallel rho:

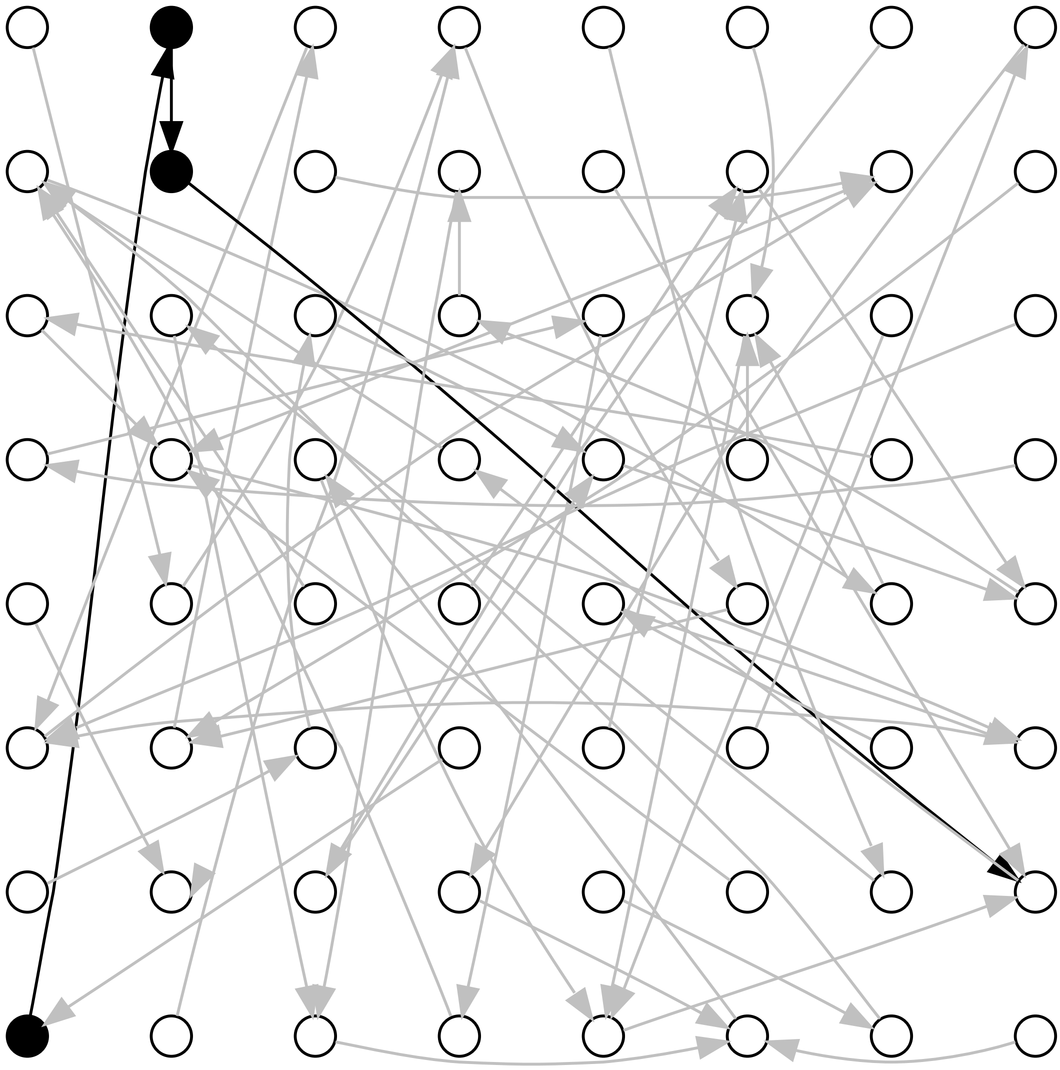
Make a pseudo-random walk $u_0, u_1, u_2, \ldots$ in the group $\langle g \rangle$, where current point determines the next point: $u_{i+1} = f(u_i)$.

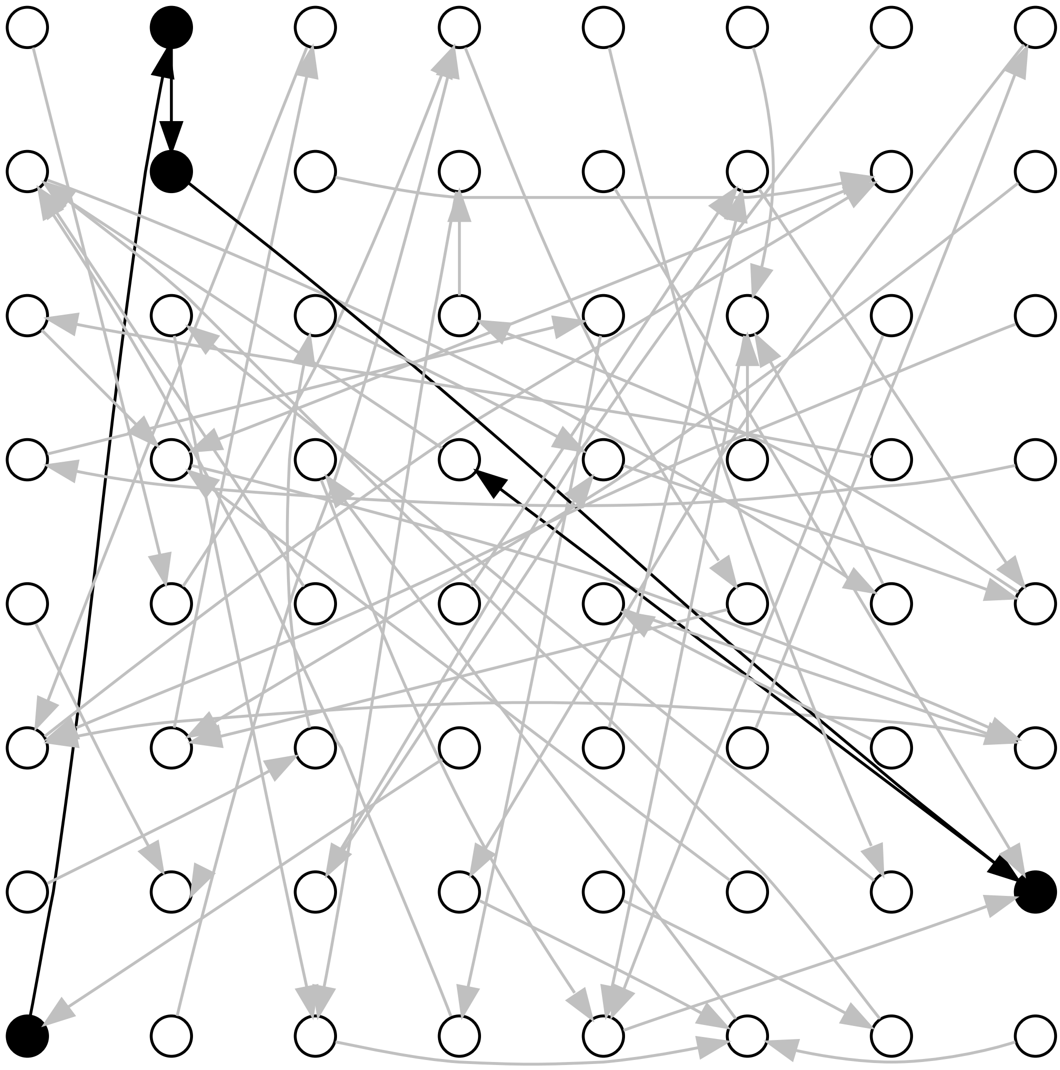
Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

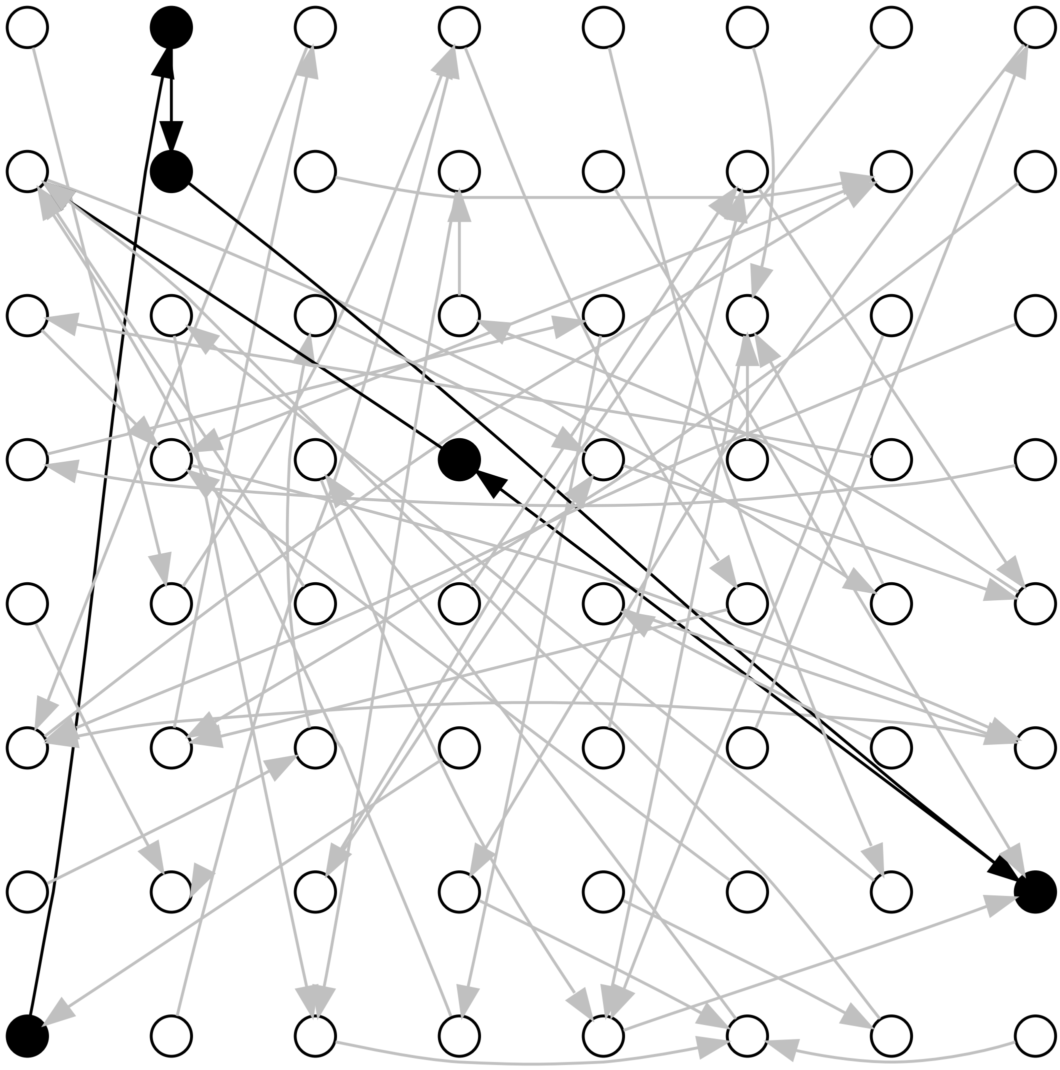
The walk now enters a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

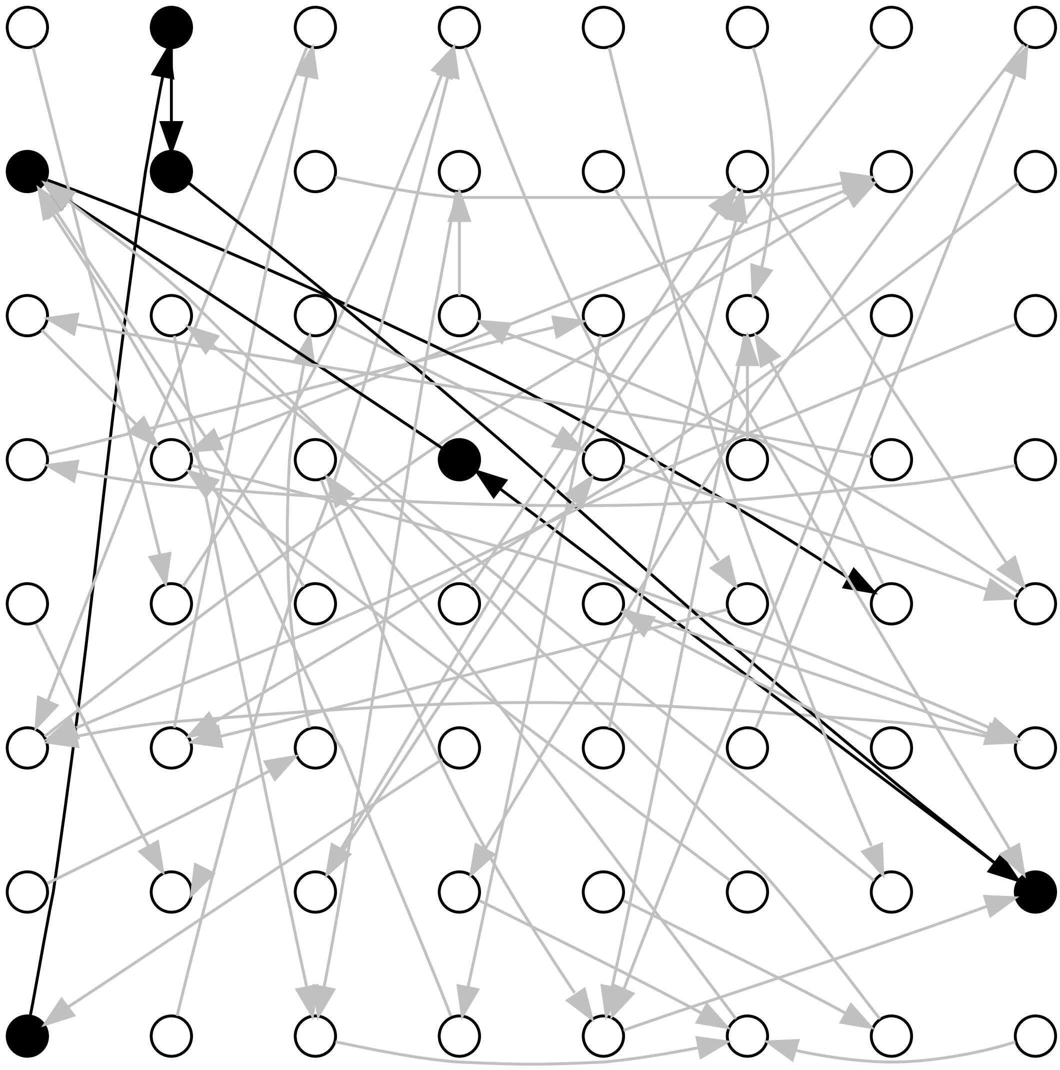


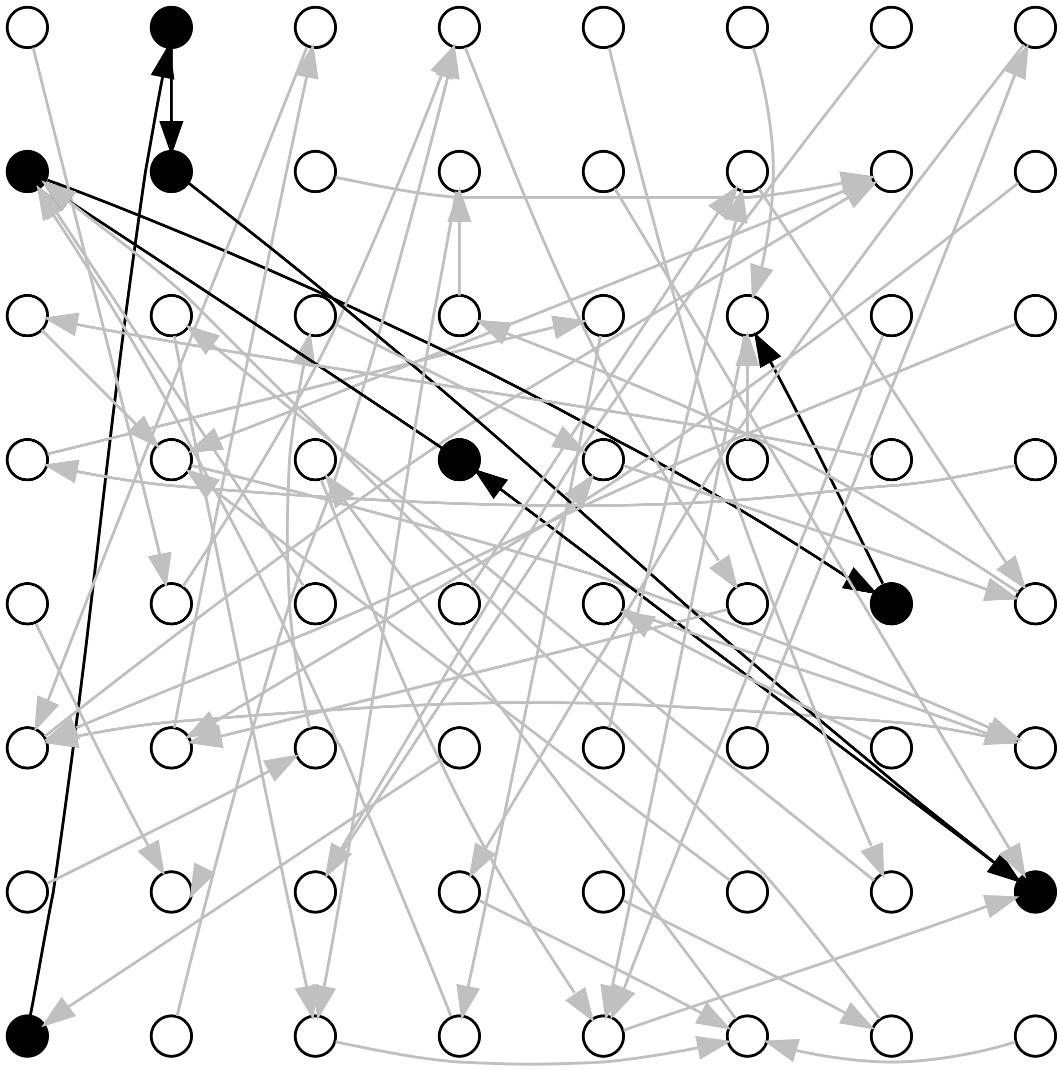


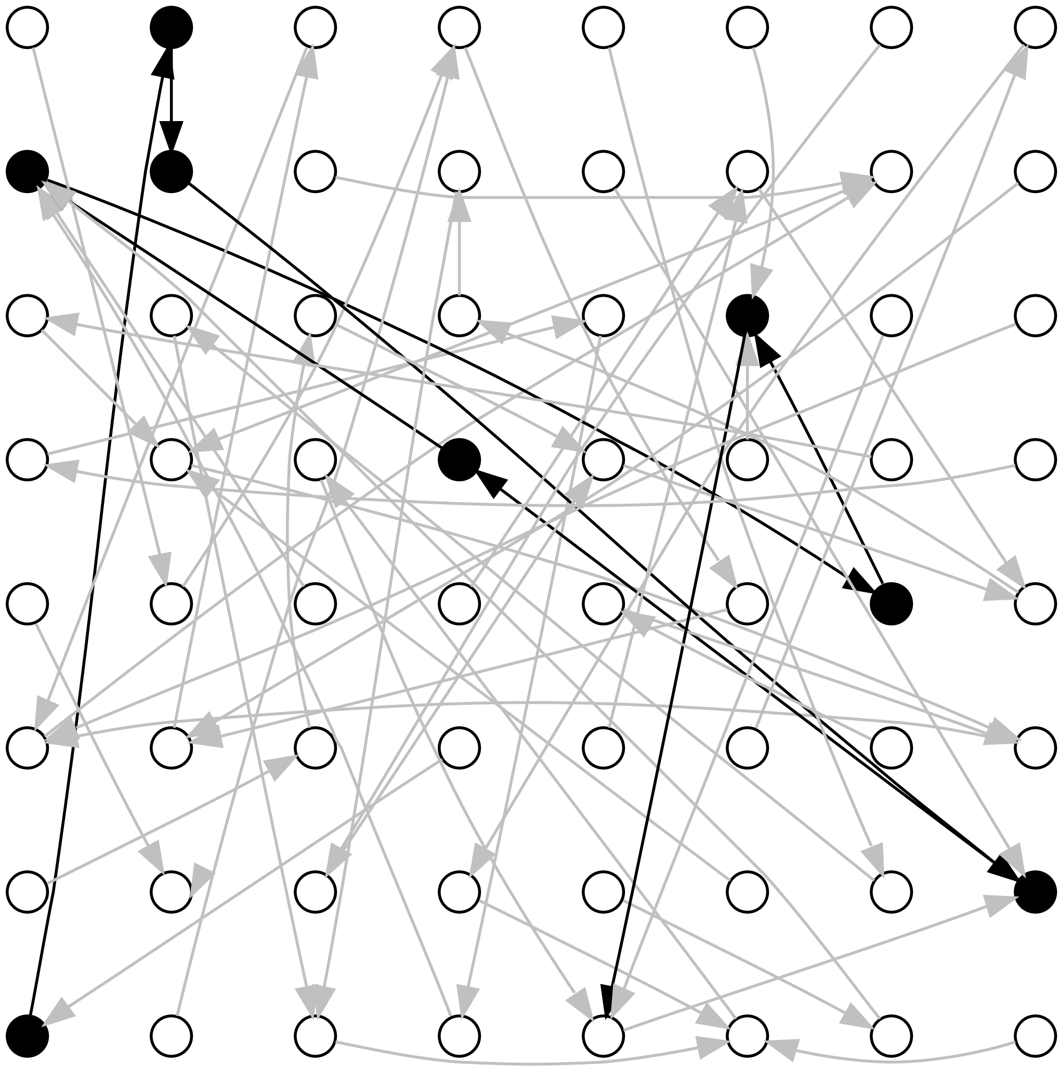


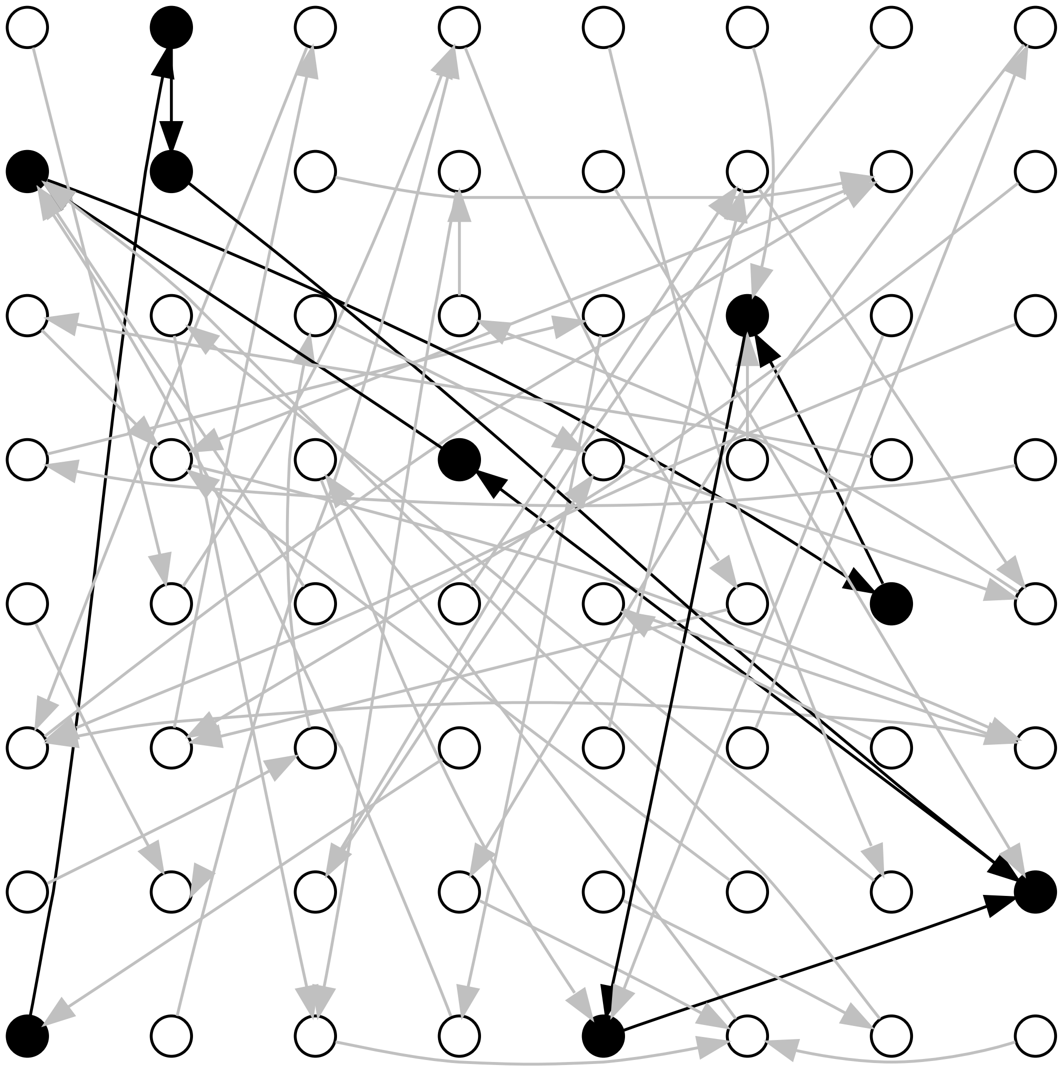


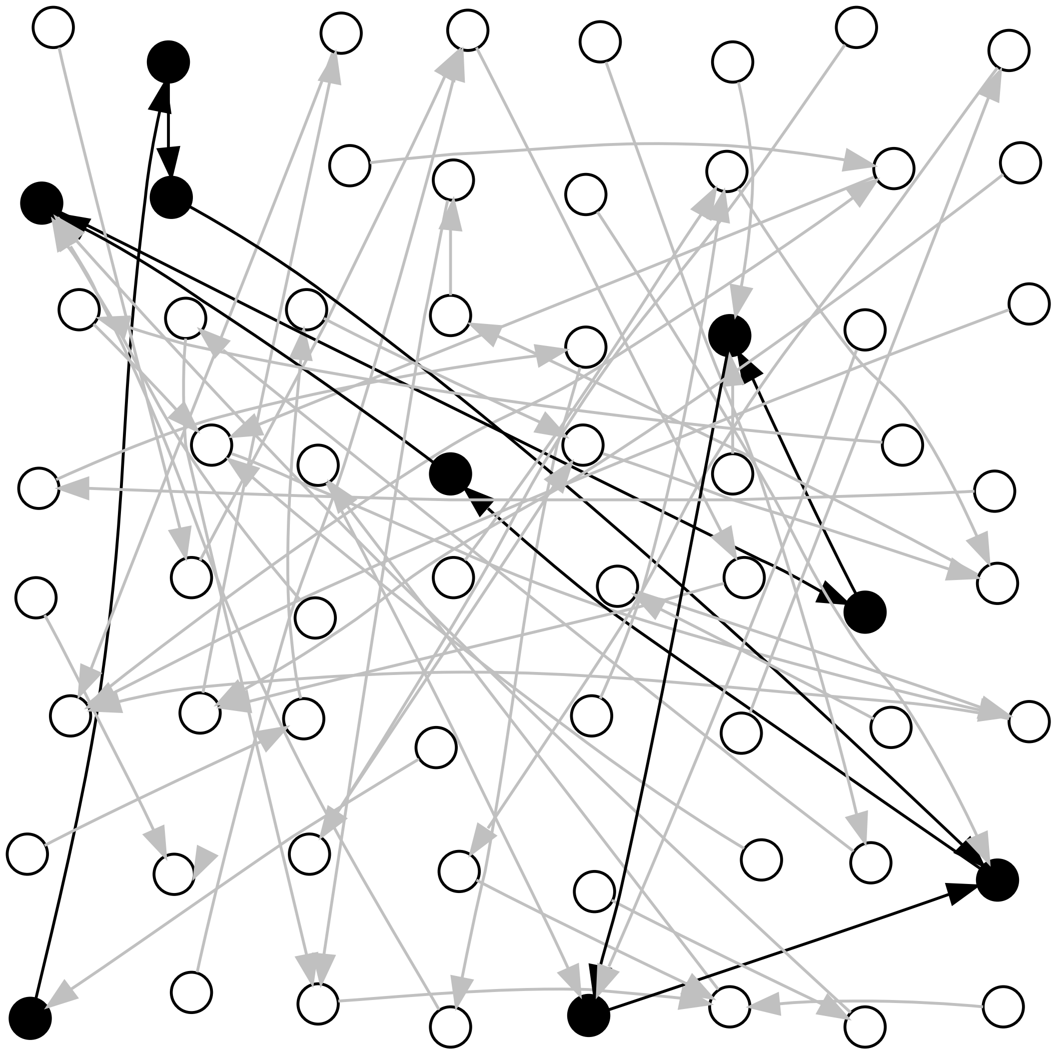


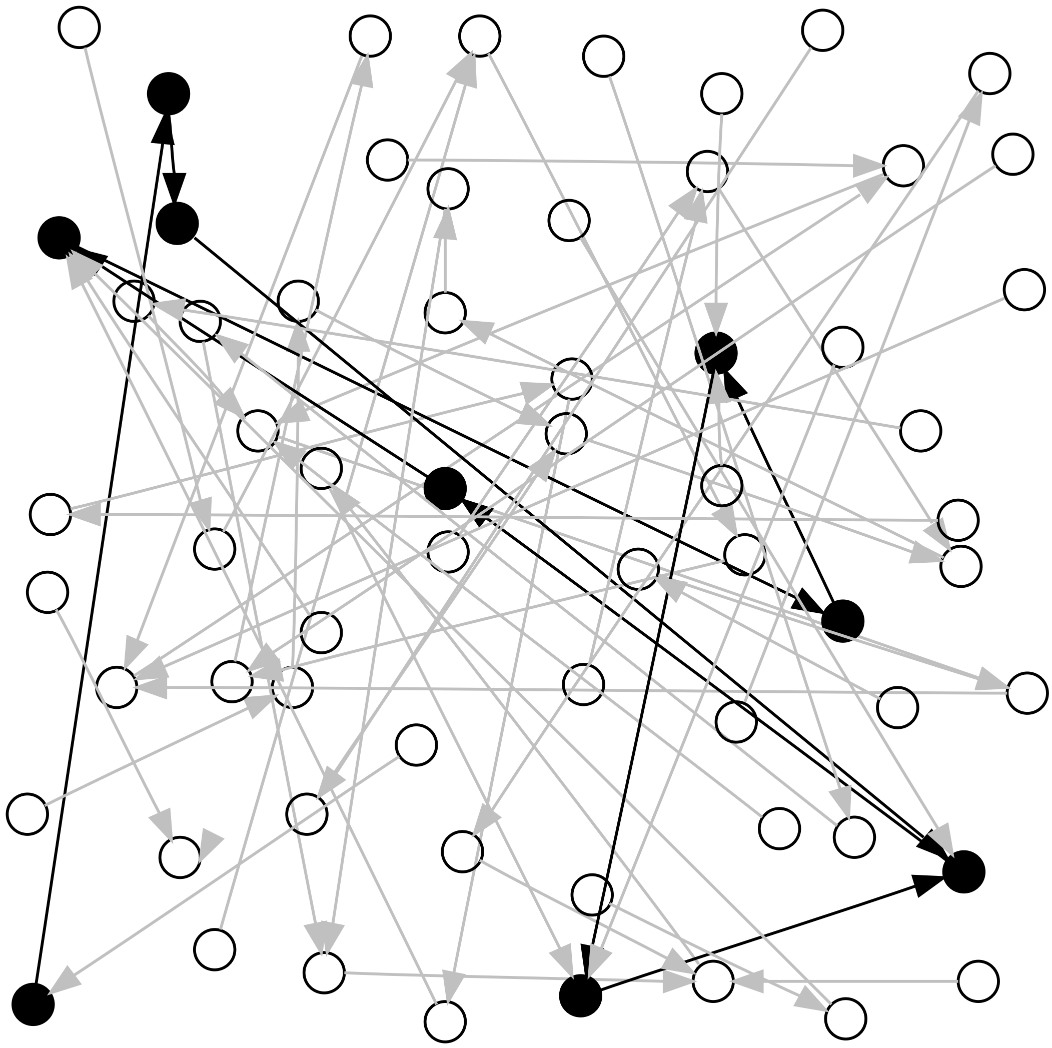


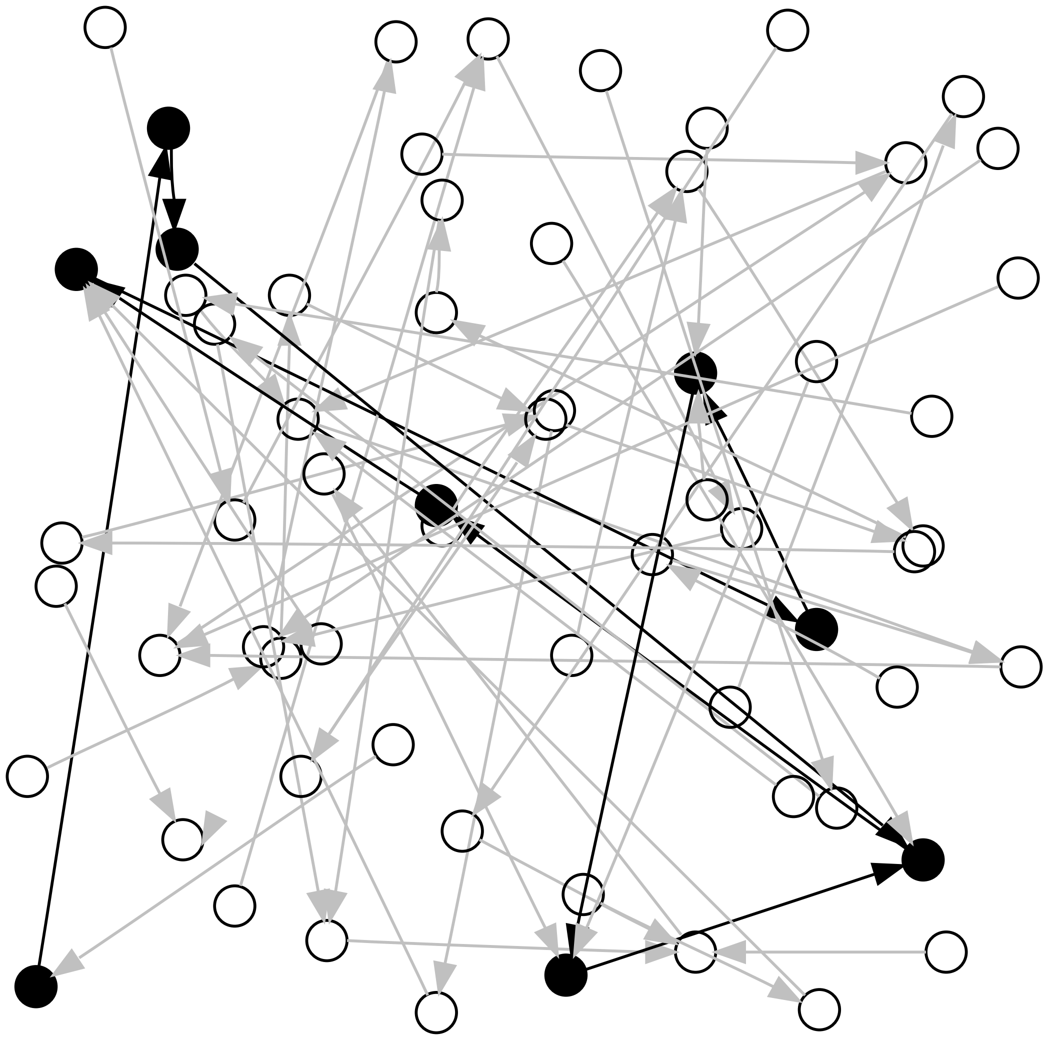


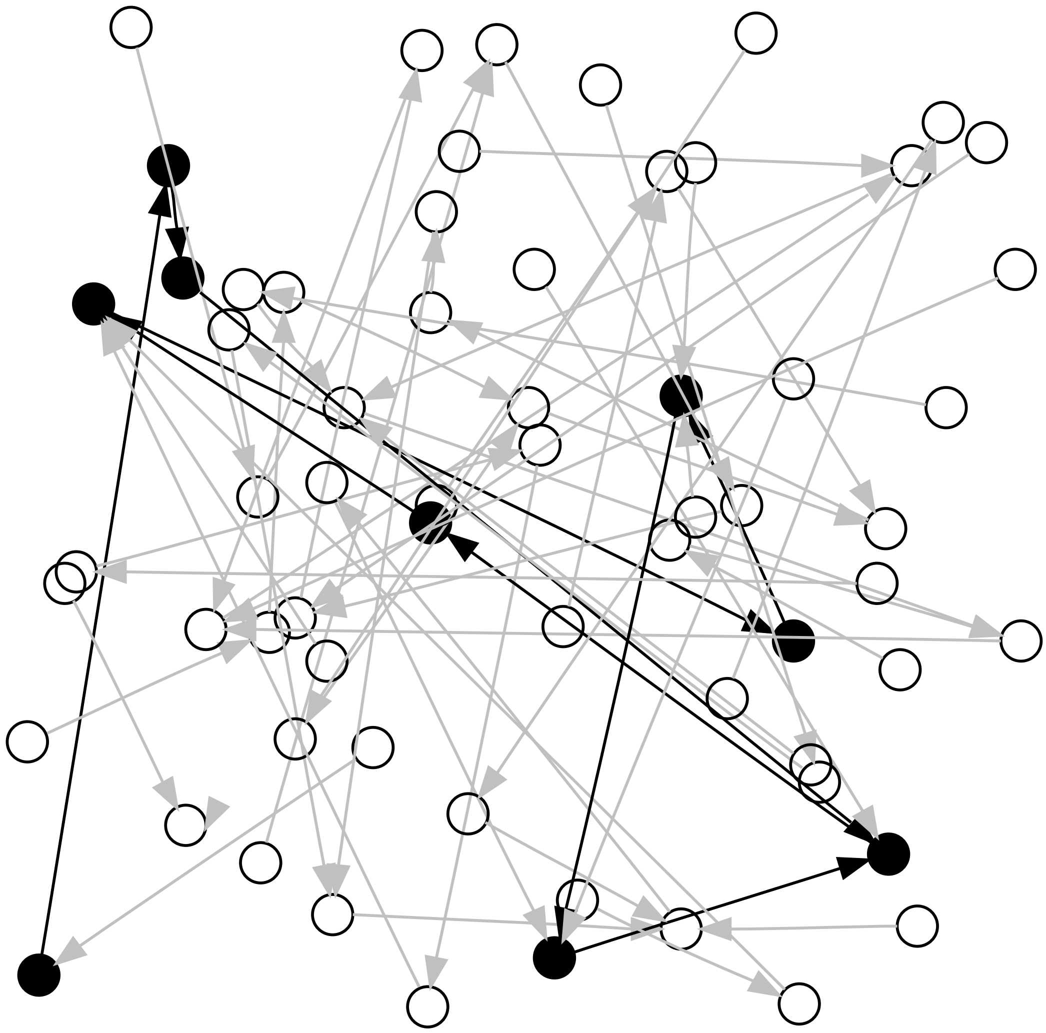


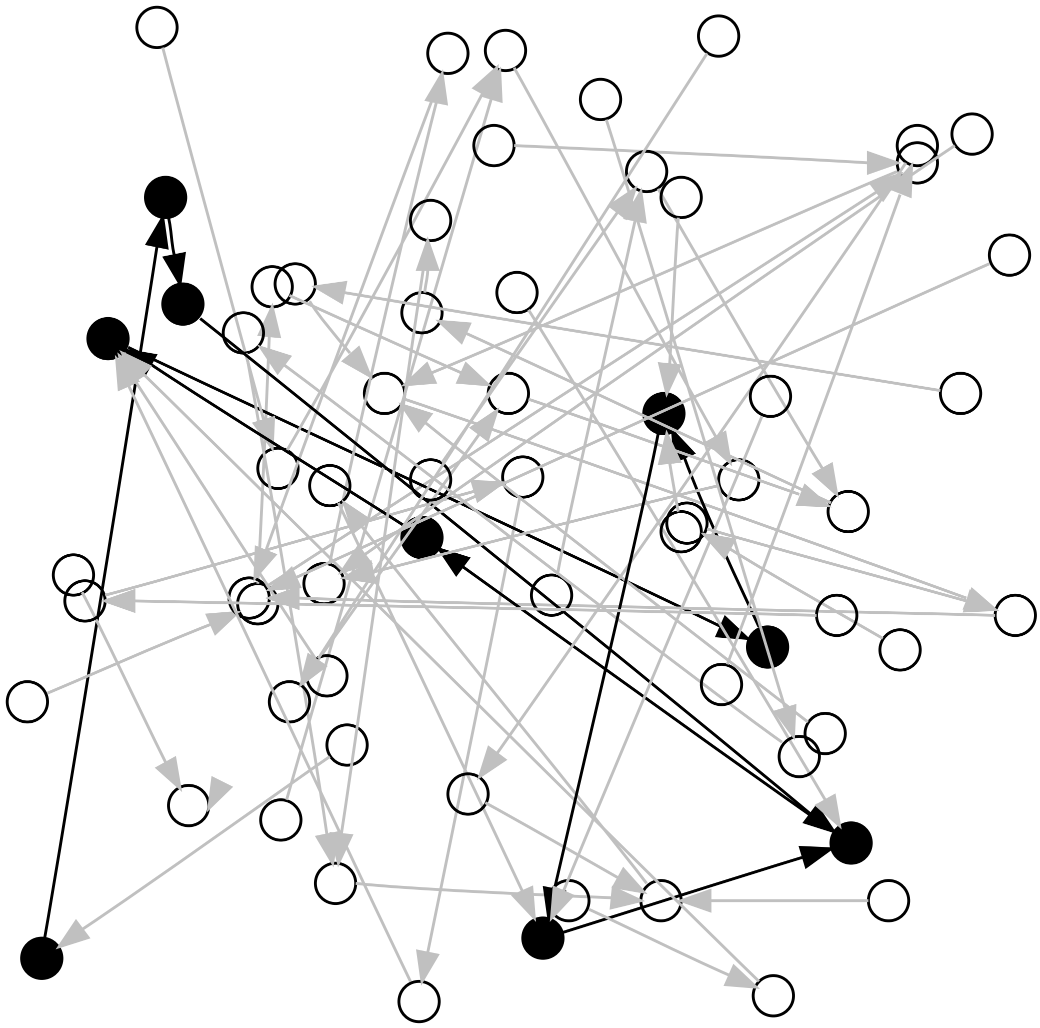


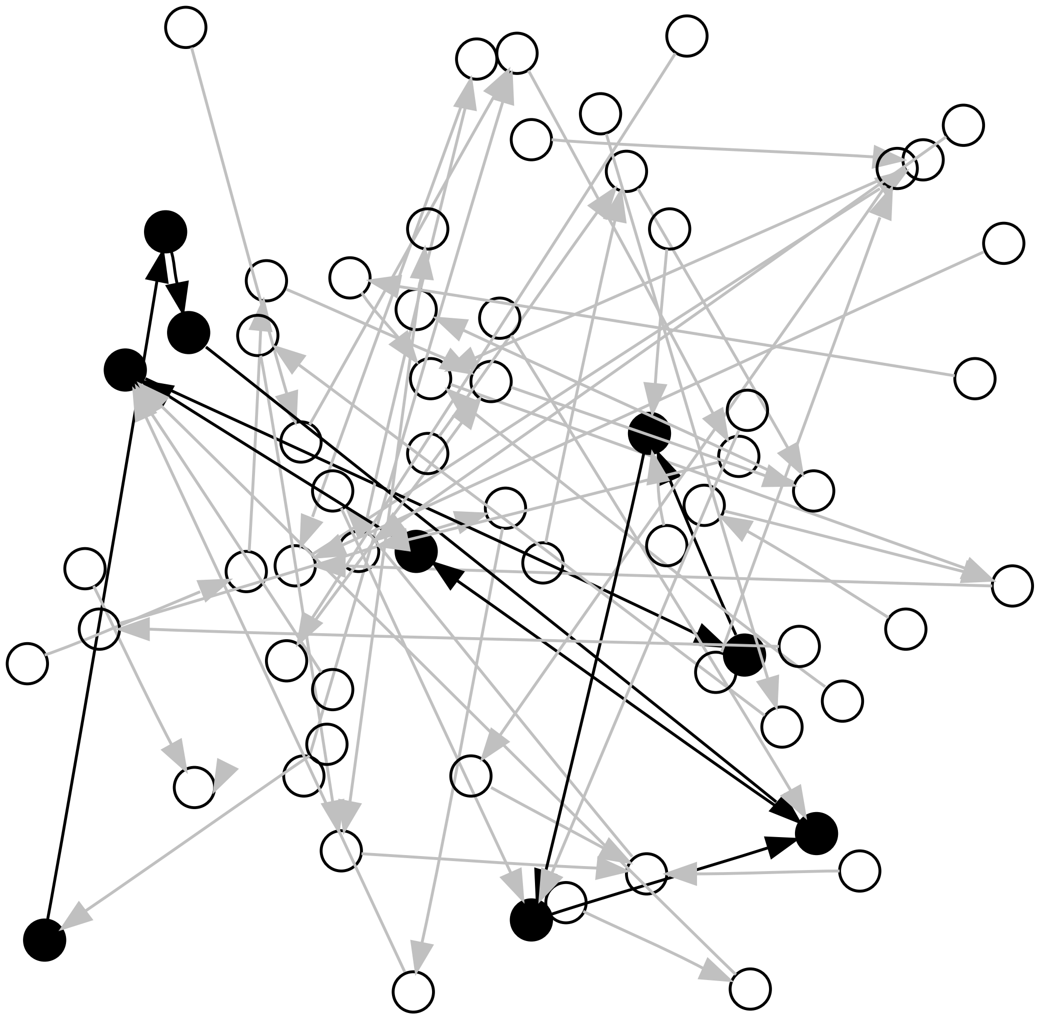


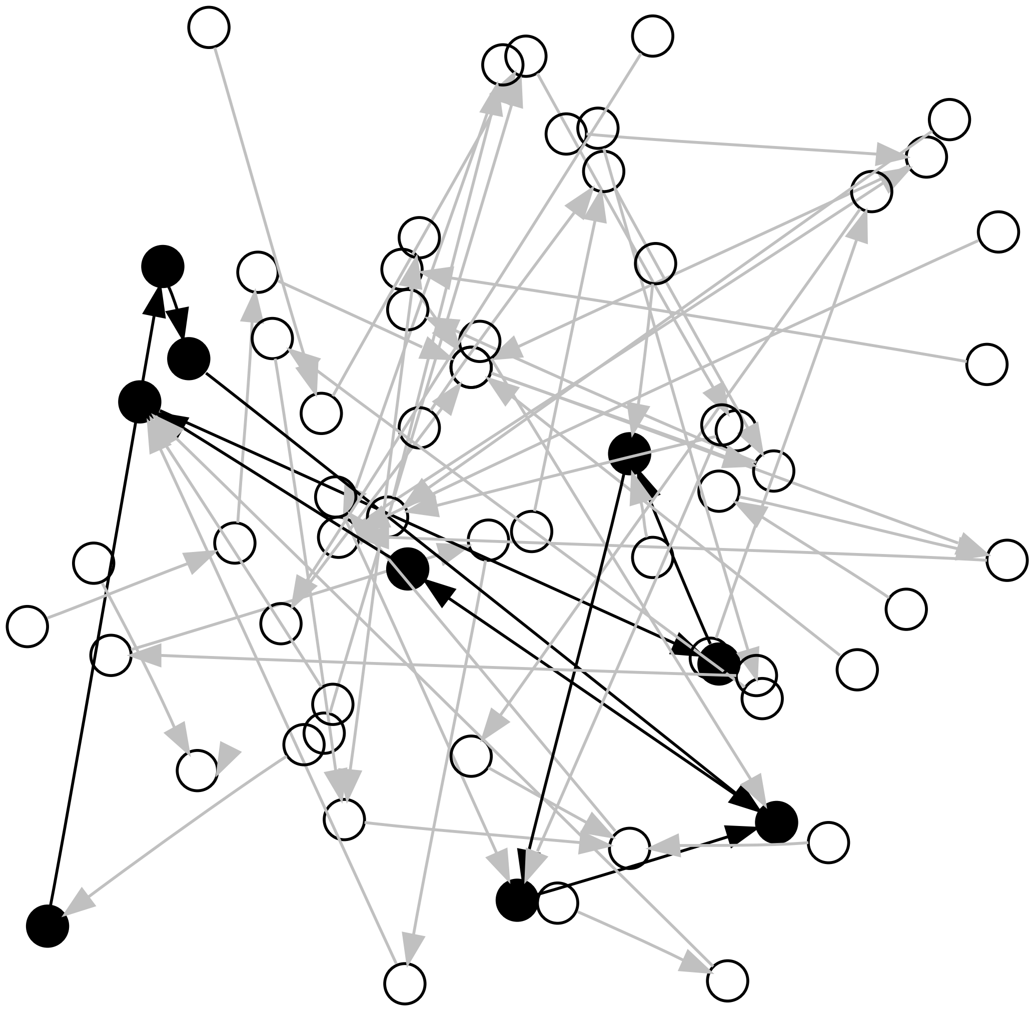


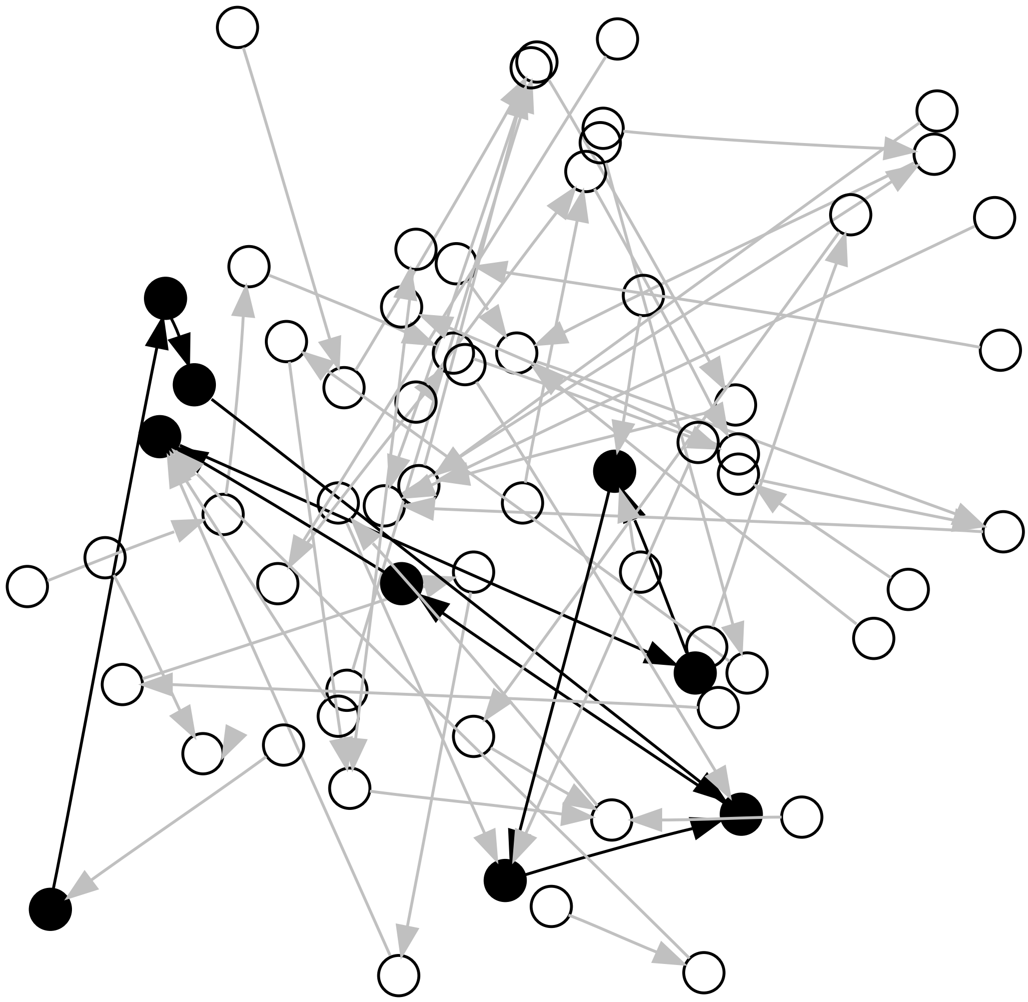


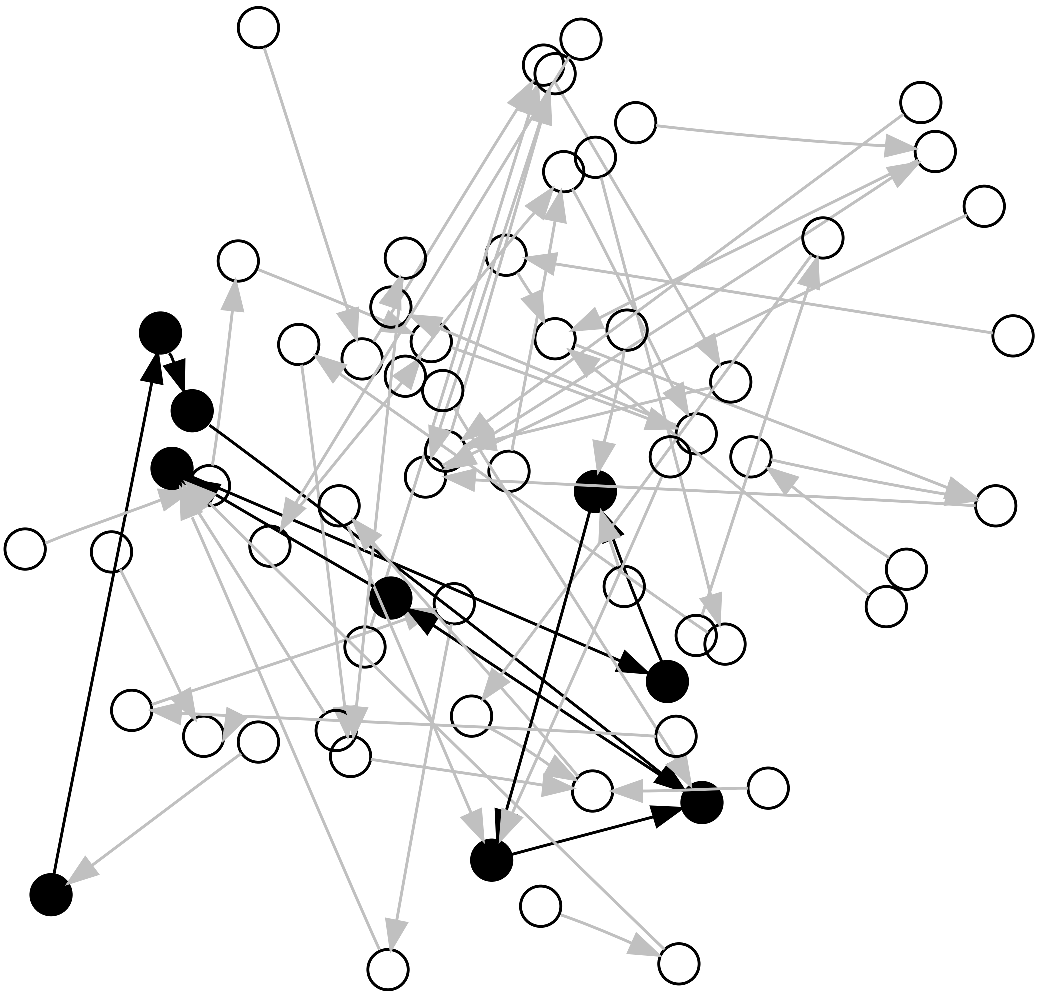


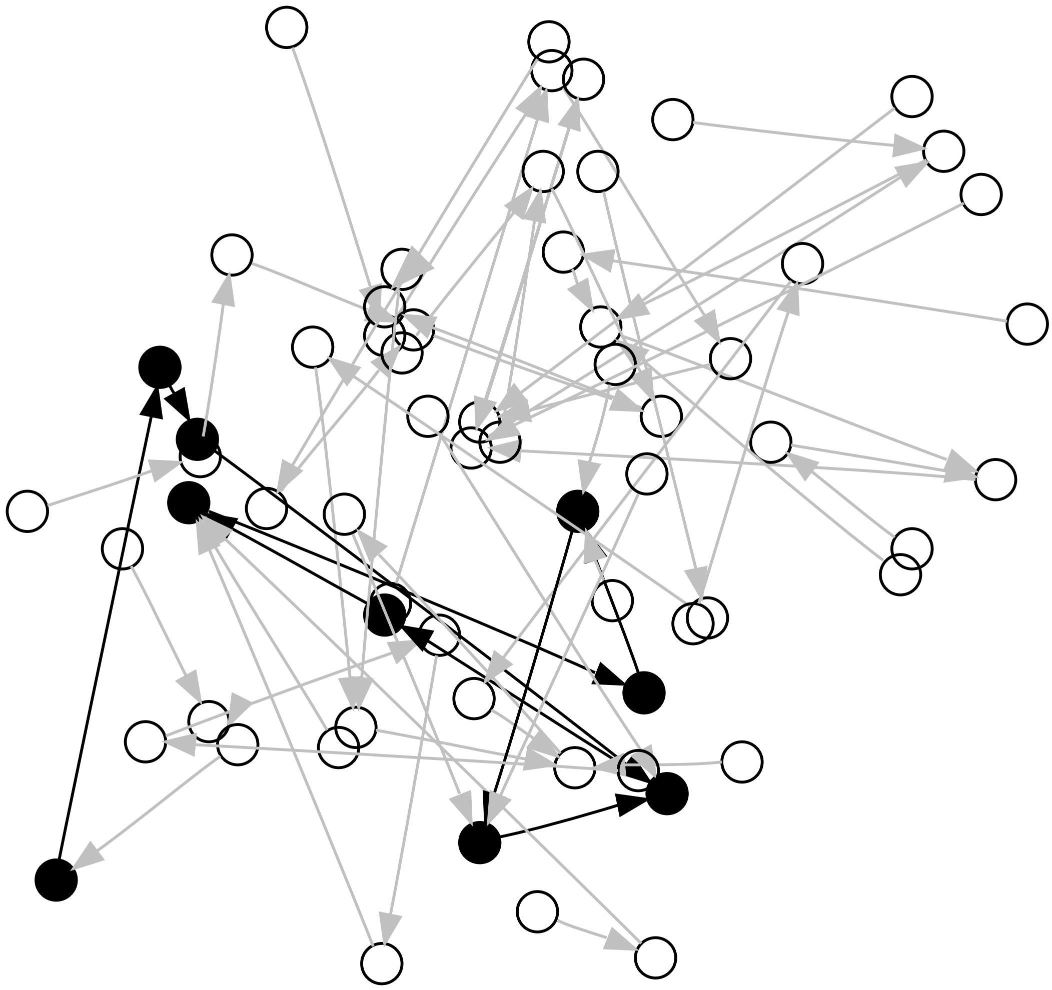


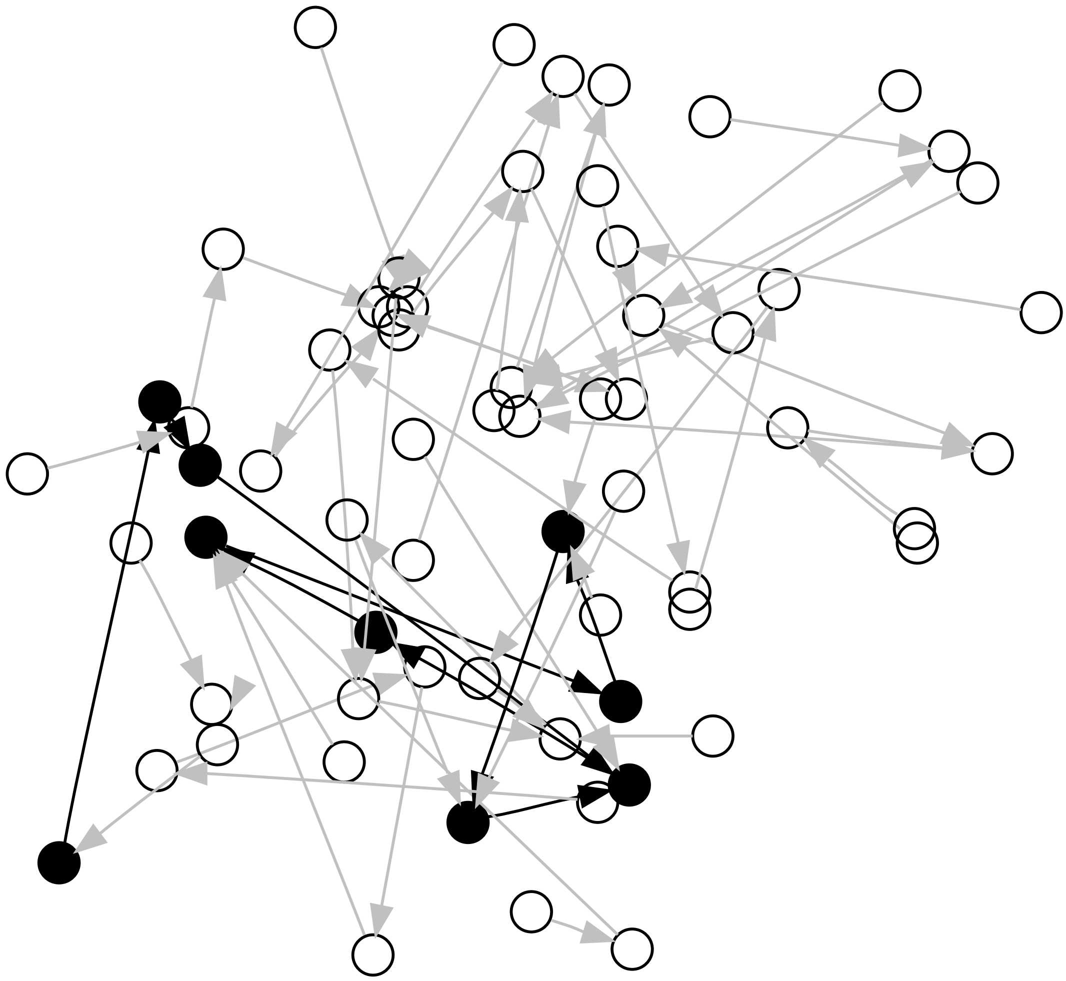


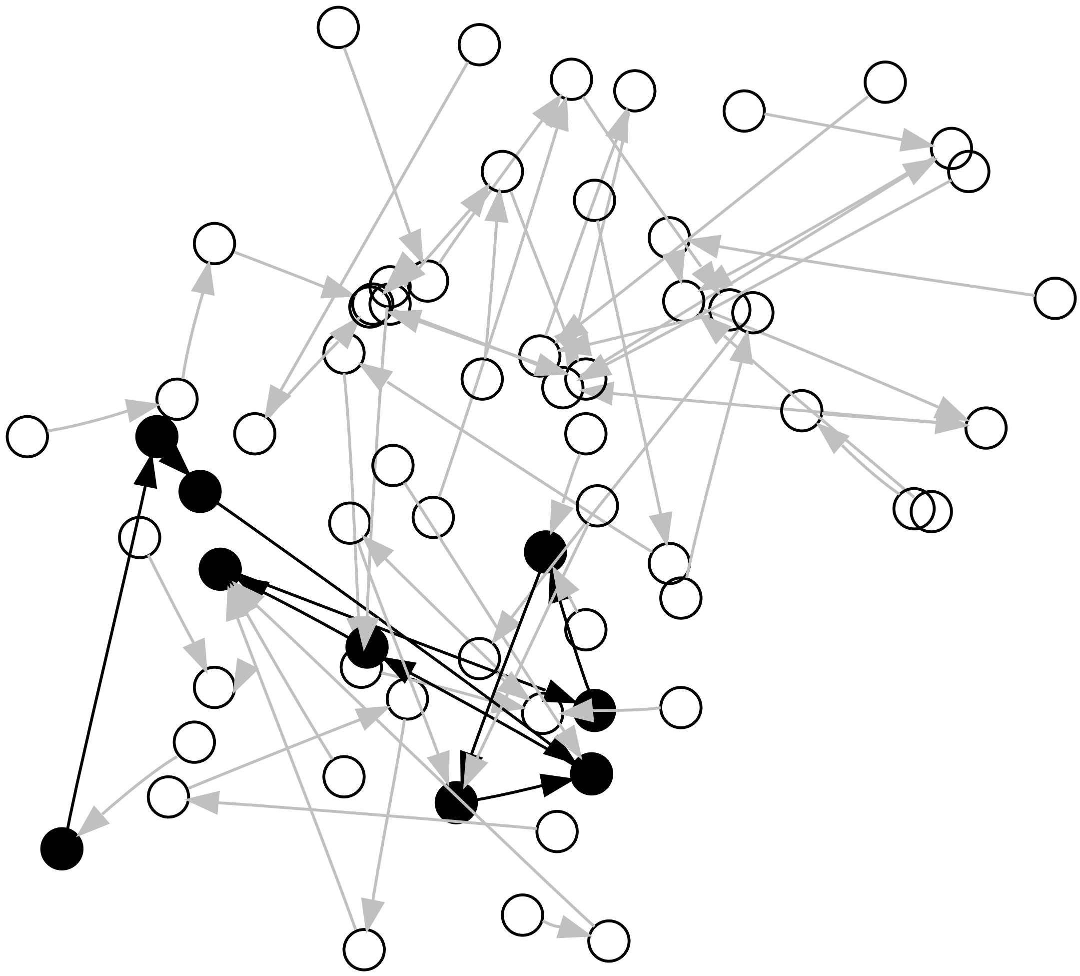


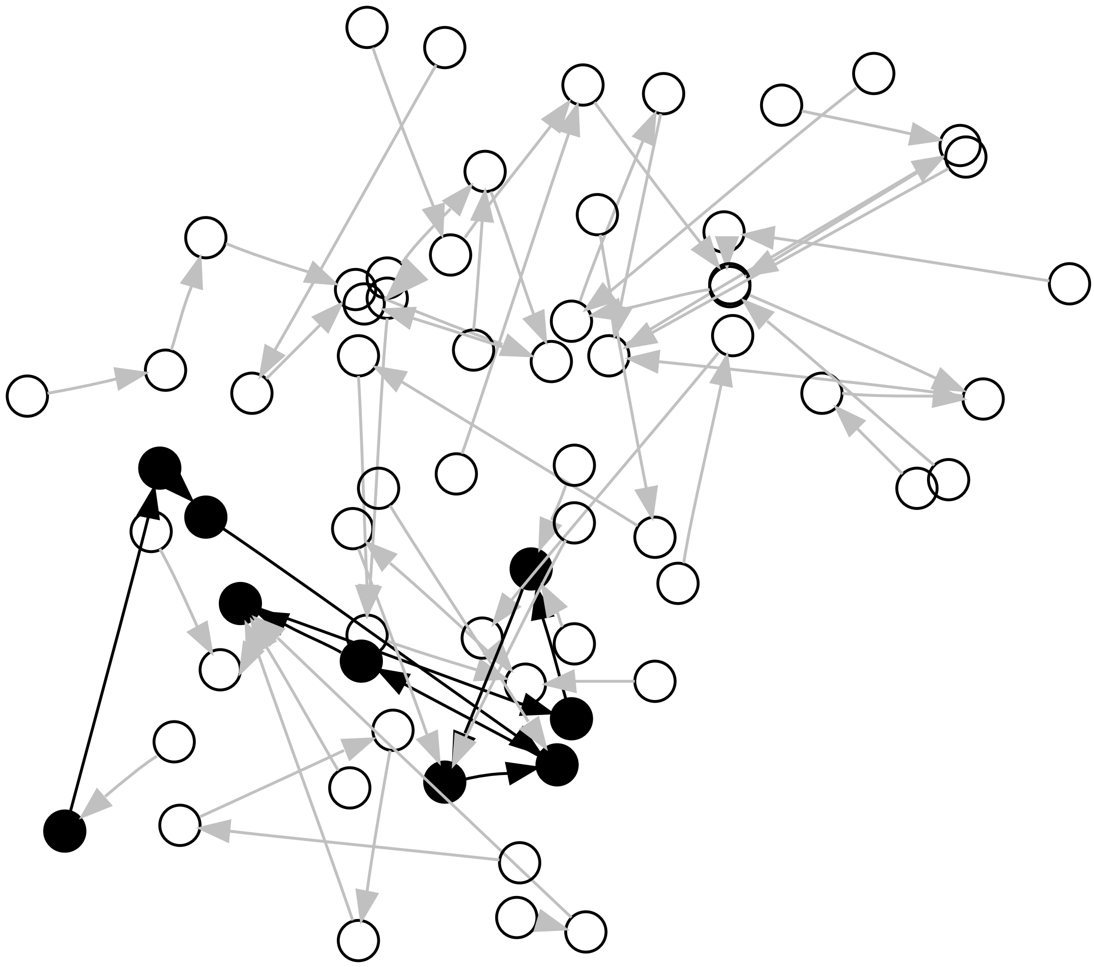


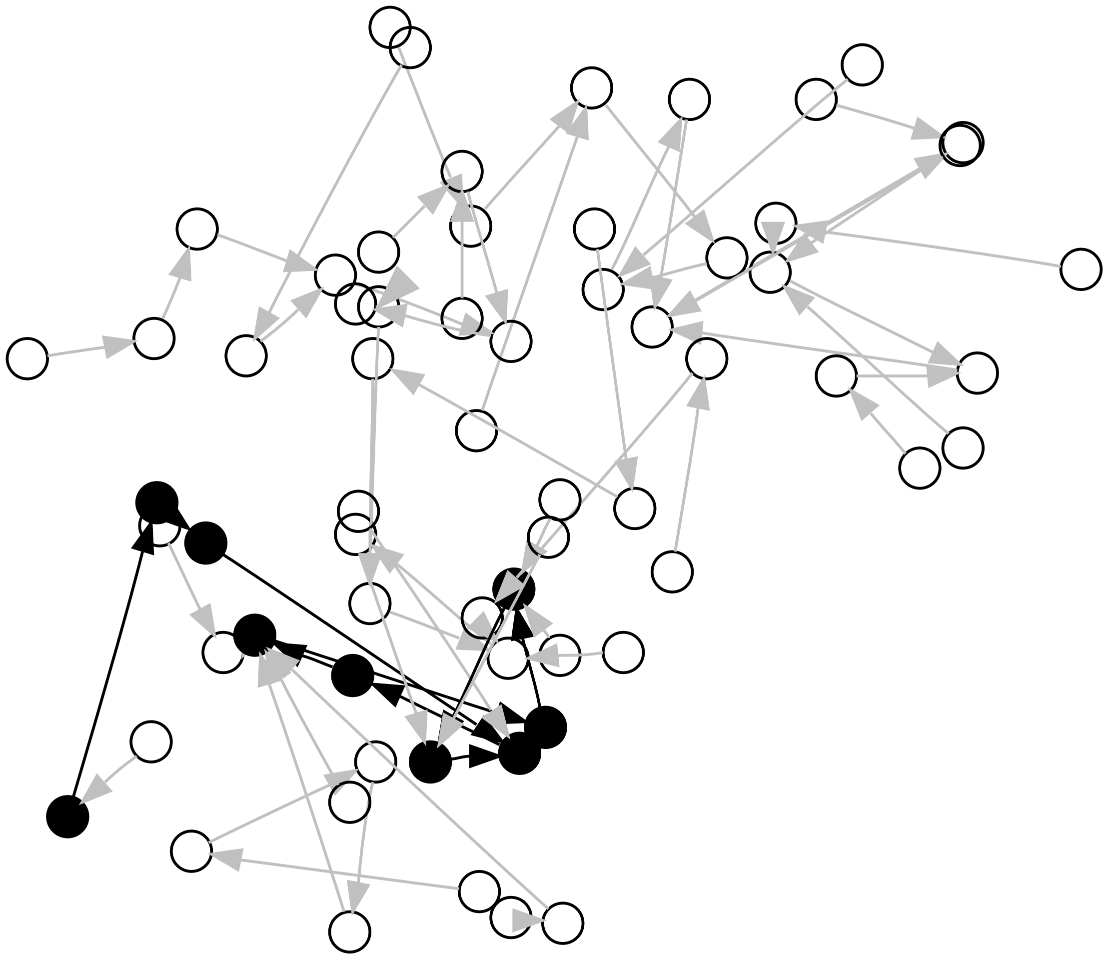


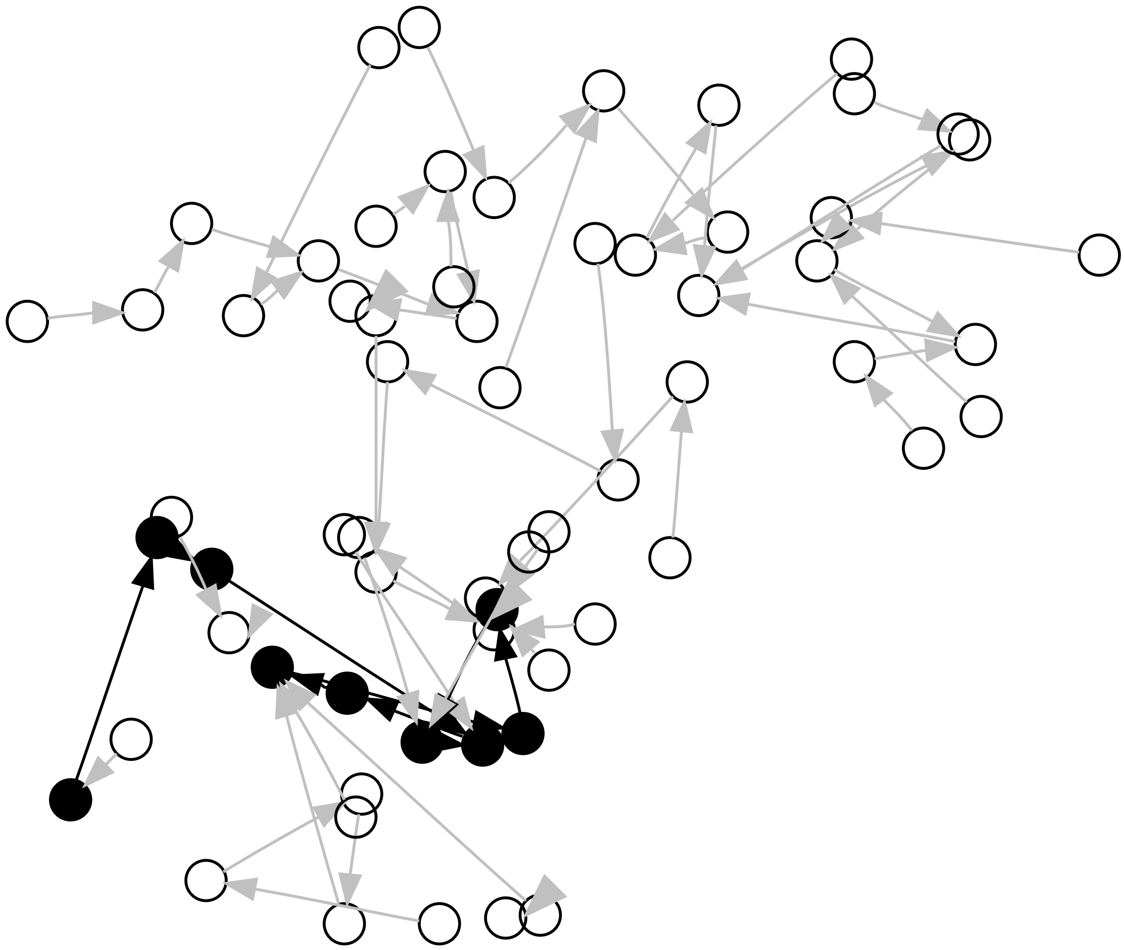


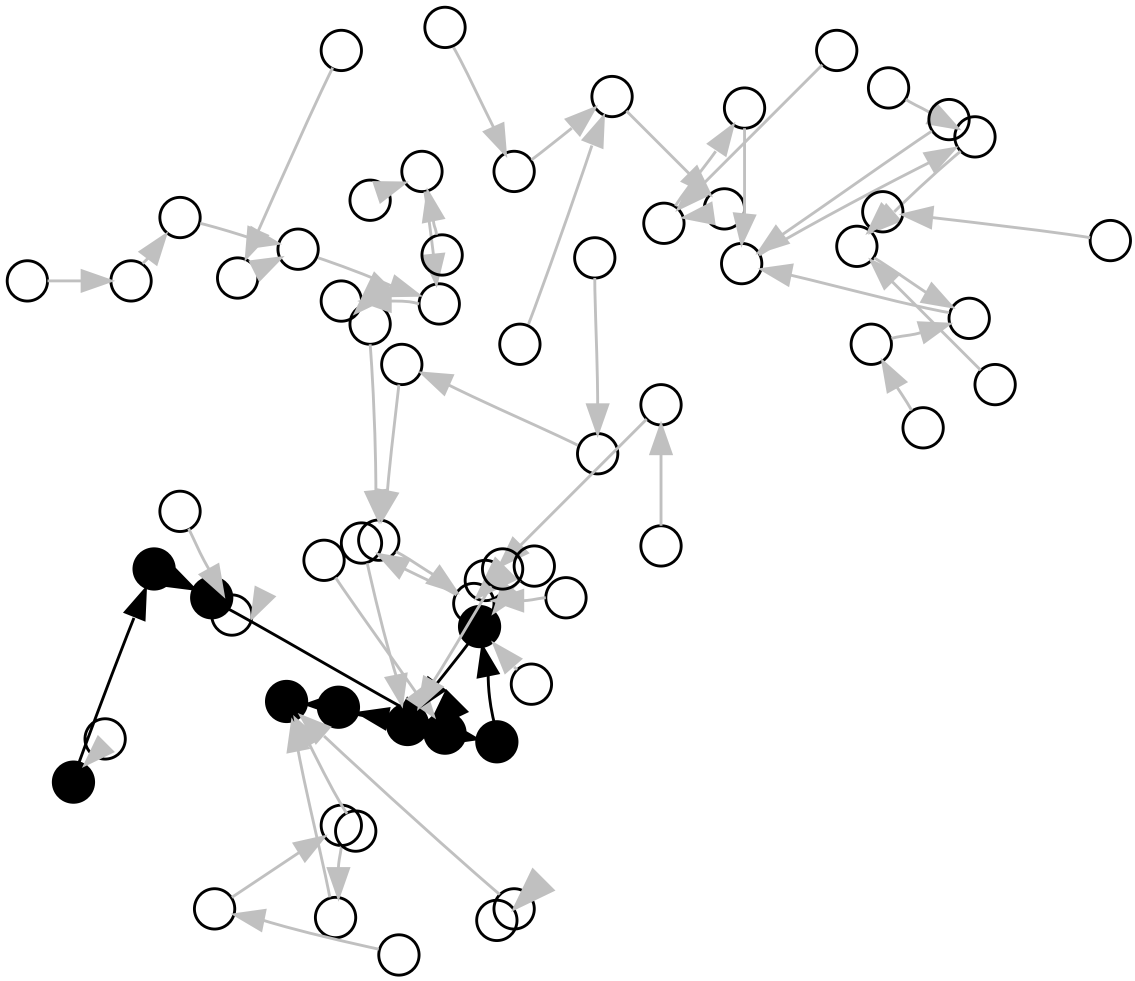


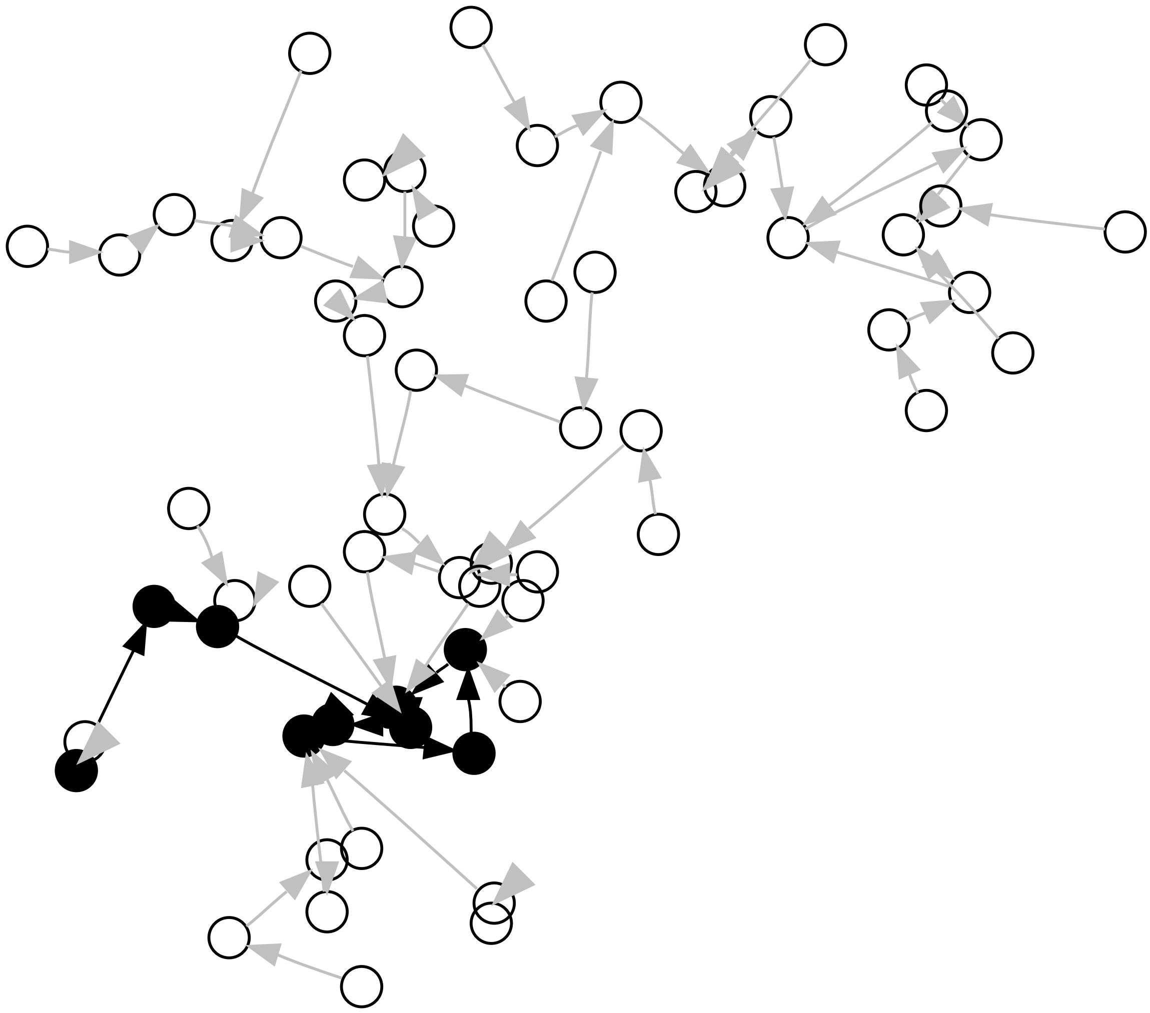


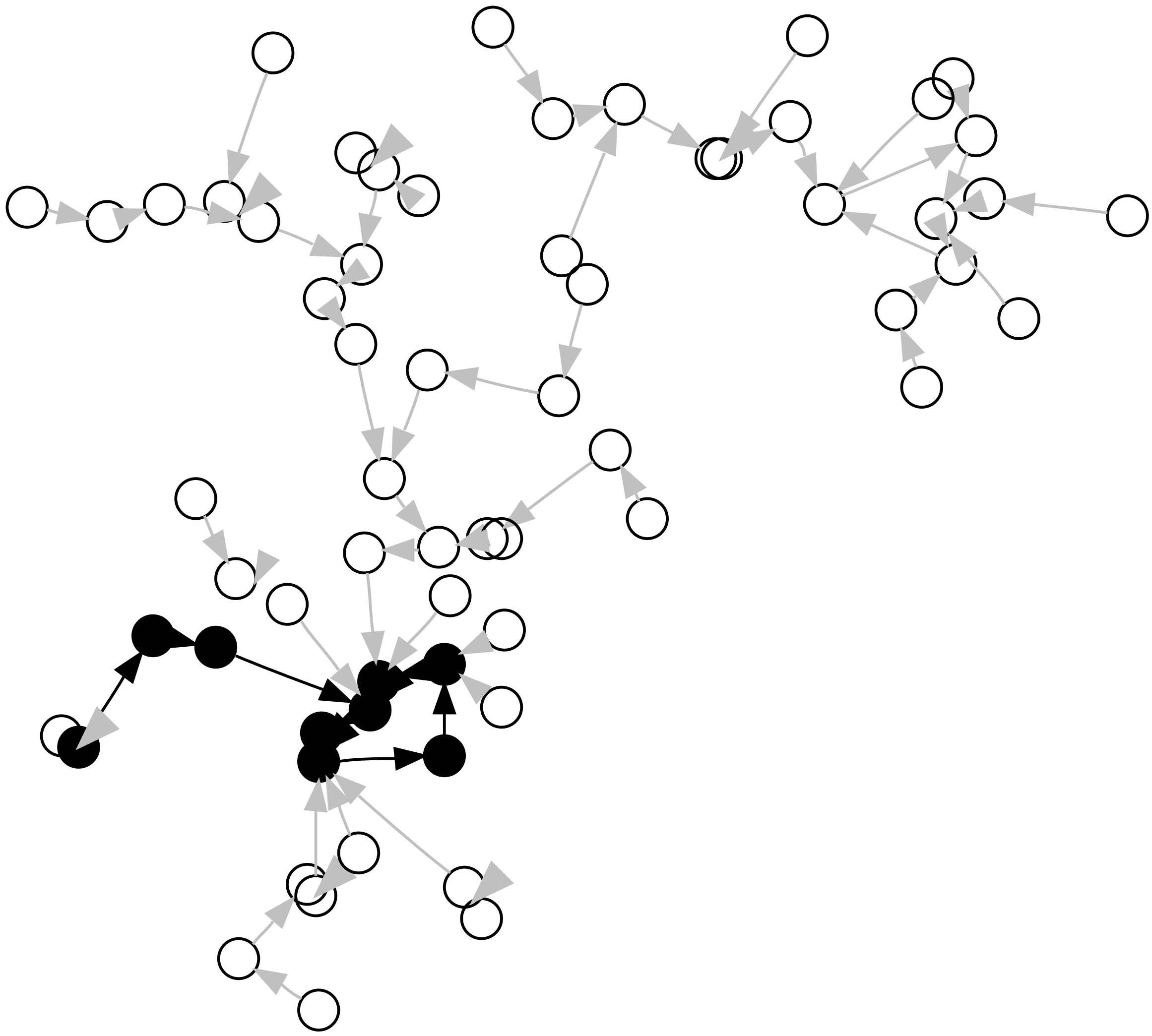


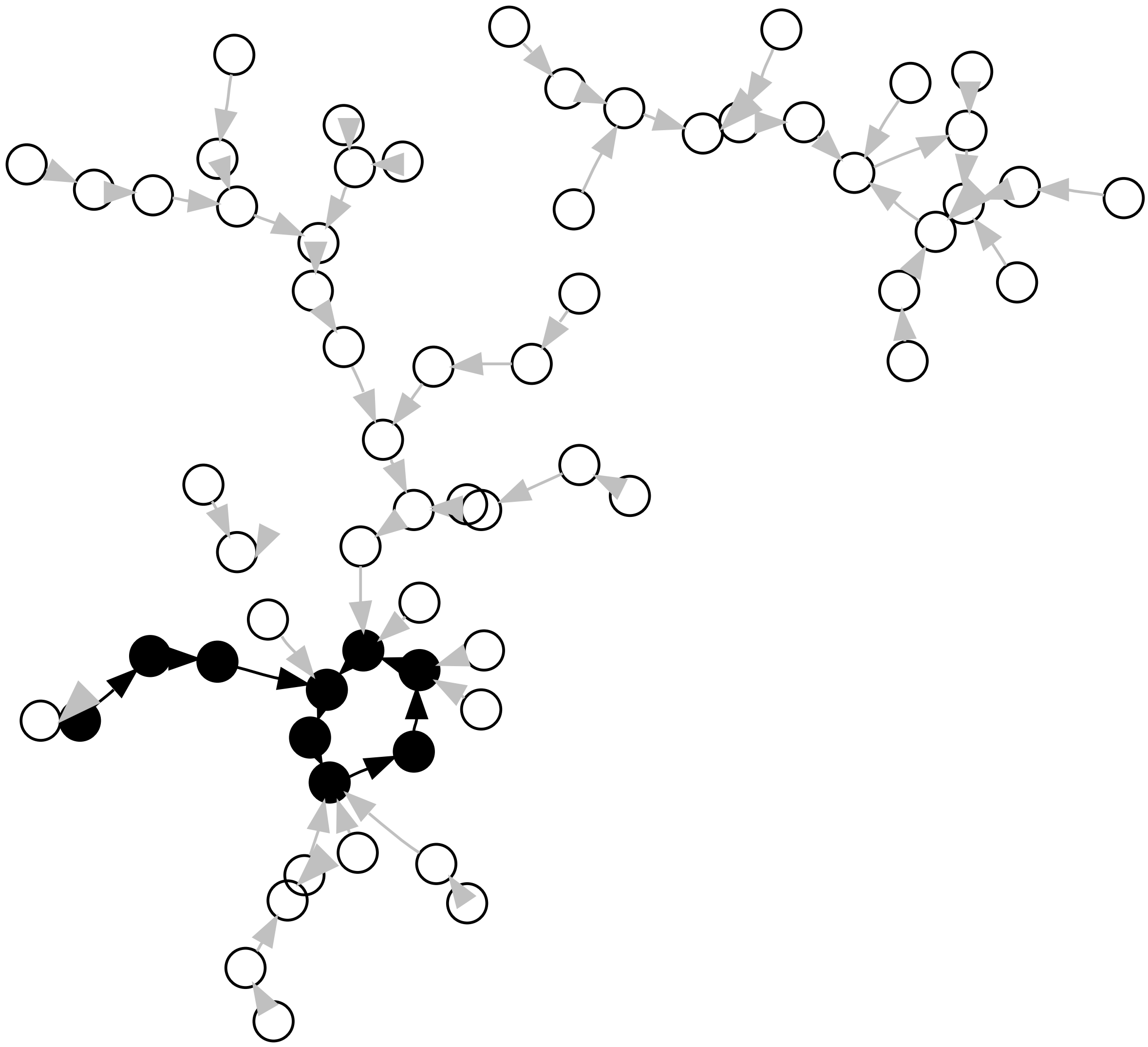


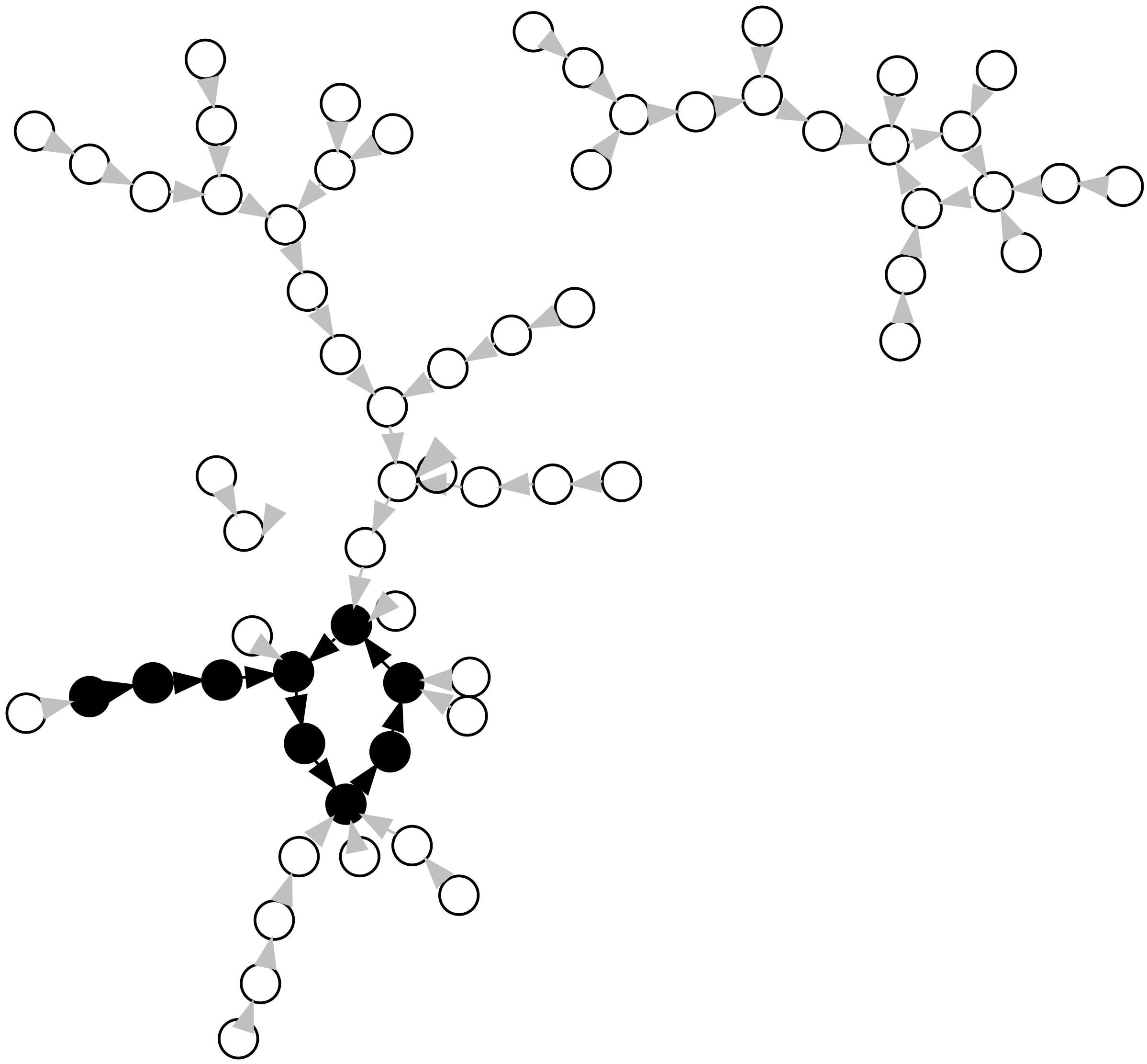












Goal: Compute $\log_g h$.

Assume that for each i
we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $u_i = g^{y_i} h^{x_i}$.

Then $u_i = u_j$ means that
$g^{y_i} h^{x_i} = g^{y_j} h^{x_j}$
so $g^{y_i - y_j} = h^{x_j - x_i}$.
If $x_i \neq x_j$ the DLP is solved:
$\log_g h = (y_j - y_i)/(x_i - x_j)$.

Goal: Compute $\log_g h$.

Assume that for each i
we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $u_i = g^{y_i} h^{x_i}$.

Then $u_i = u_j$ means that
$g^{y_i} h^{x_i} = g^{y_j} h^{x_j}$
so $g^{y_i - y_j} = h^{x_j - x_i}$.
If $x_i \neq x_j$ the DLP is solved:
$\log_g h = (y_j - y_i)/(x_i - x_j)$.

e.g. "base-(g, h) r-adding walk":
precompute $s_1, s_2, \ldots, s_r$
as random products $g^{\cdots} h^{\cdots}$;
define $f(u) = u s_{H(u)}$
where H hashes to $\{1, 2, \ldots, r\}$.

Ample experimental evidence that base-(g, h) r-adding walk resembles a random walk: solves DLP in about $\sqrt{\pi\ell/2}$ steps on average.

Ample experimental evidence that base-(g, h) r-adding walk resembles a random walk: solves DLP in about $\sqrt{\pi \ell / 2}$ steps on average.

2001 Teske: need big r; e.g., $r = 20$. Clear slowdown for small r; Blackburn and Murphy say $\sqrt{\pi \ell / 2} / \sqrt{1 - 1/r}$.

Ample experimental evidence that base-(g, h) r-adding walk resembles a random walk: solves DLP in about $\sqrt{\pi\ell/2}$ steps on average.

2001 Teske: need big r; e.g., $r = 20$. Clear slowdown for small r; Blackburn and Murphy say $\sqrt{\pi\ell/2}/\sqrt{1 - 1/r}$.

2010 Bernstein–Lange (ANTS 2012): actually more complicated; higher-degree anticollisions.

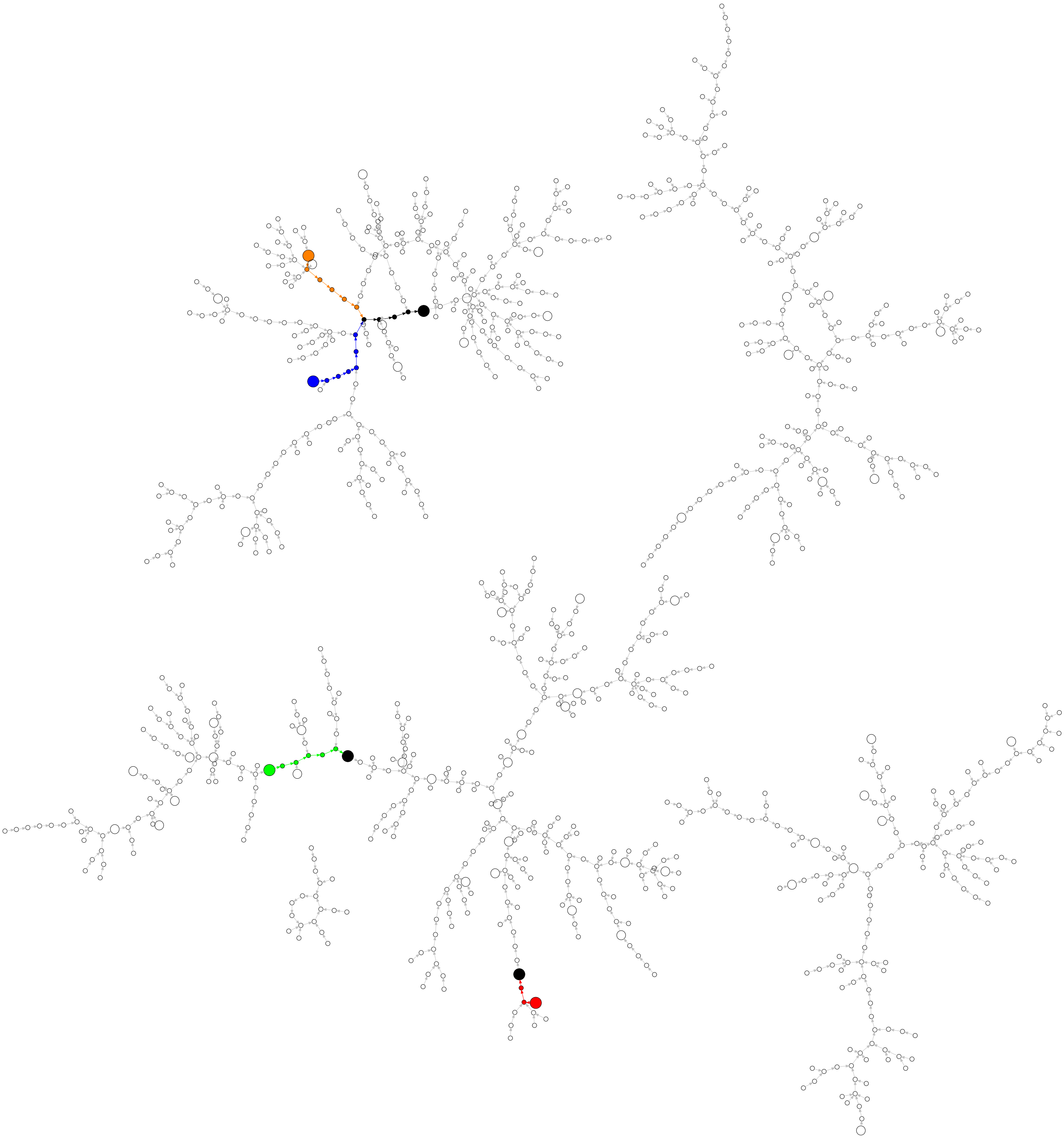
Parallel rho

1994 van Oorschot–Wiener:

Declare some subset of $\langle g \rangle$ to be the set of *distinguished points*: e.g., all $u \in \langle g \rangle$ where last 20 bits of representation of u are 0.

Perform, in parallel, many walks with different starting points hg^y but same update function f.

Terminate each walk once it hits a distinguished point. Report point to central server. Server receives, stores, and sorts all distinguished points.



Two colliding walks will reach
the same distinguished point.
Server sees collision, finds DL.

Many discrete logarithms

1999 Escott–Sager–Selkirk–Tsapakidis, also crediting Silverman–Stapleton:

Computing (e.g.) $\log_g h_1$, $\log_g h_2$, $\log_g h_3$, $\log_g h_4$, and $\log_g h_5$ costs only 2.49× more than computing just $\log_g h$.

The basic idea:
compute $\log_g h_1$ with rho;
compute $\log_g h_2$ with rho, *reusing* distinguished points produced by h_1; etc.

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

2001 Kuhn–Struik analysis:
cost $\Theta(n^{1/2}\ell^{1/2})$
for n discrete logarithms
in group of order ℓ
if $n \ll \ell^{1/4}$.

2004 Hitchcock–Montague–Carter–Dawson:
View computations of
$\log_g h_1, \ldots, \log_g h_{n-1}$ as
precomputatation for main
computation of $\log_g h_n$.
Analyze tradeoffs between
main-computation time and
precomputation time.

2012 Bernstein–Lange, this paper:

(1) Adapt to interval of length ℓ inside much larger group.

(2) Analyze tradeoffs between main-computation time and precomputed table size.

(3) Choose table entries more carefully to reduce main-computation time.

(4) Also choose iteration function more carefully.

(5) Reduce space required for each table entry.

(6) Break $\ell^{1/4}$ barrier.

Applications:

(7) Accelerate trapdoor DL etc.

(8) Accelerate BGN etc.;
this needs (1).

Further applications in 2012 Bernstein–Lange "Non-uniform cracks in the concrete", eprint.iacr.org/2012/318: these algorithms disprove standard security conjectures, demonstrating flaw in standard formal definitions of security.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (7).

The basic algorithm

Precomputation:
Start some walks at g^y
for random choices of y.
Build table of distinct
distinguished points d
along with $\log_g d$.

Use base-g r-adding walk,
not base-(g, h) r-adding walk!

Main computation:
Starting from h, walk to
distinguished point hg^y.
Check for hg^y in table.
(If this fails, rerandomize h.)

Standard base-g r-adding walk chooses uniform random $c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$; precomputes $s_i = g^{c_i}$; walks from u to $u s_{H(u)}$.

Nonstandard tweak: reduce $\ell - 1$ to, e.g., $0.25\ell/W$, where W is average walk length.

Intuition: This tweak compromises performance by only a small constant factor.

Standard base-g r-adding walk chooses uniform random $c_1, \ldots, c_r \in \{1, 2, \ldots, \ell - 1\}$; precomputes $s_i = g^{c_i}$; walks from u to $us_{H(u)}$.

Nonstandard tweak: reduce $\ell - 1$ to, e.g., $0.25\ell/W$, where W is average walk length.

Intuition: This tweak compromises performance by only a small constant factor.

If tweaked algorithm works for a group of order ℓ, what will it do for an interval of order ℓ?

Standard interval method:
Pollard's kangaroo method.

Pollard's kangaroos do small jumps around the interval.
Real kangaroos sleep.

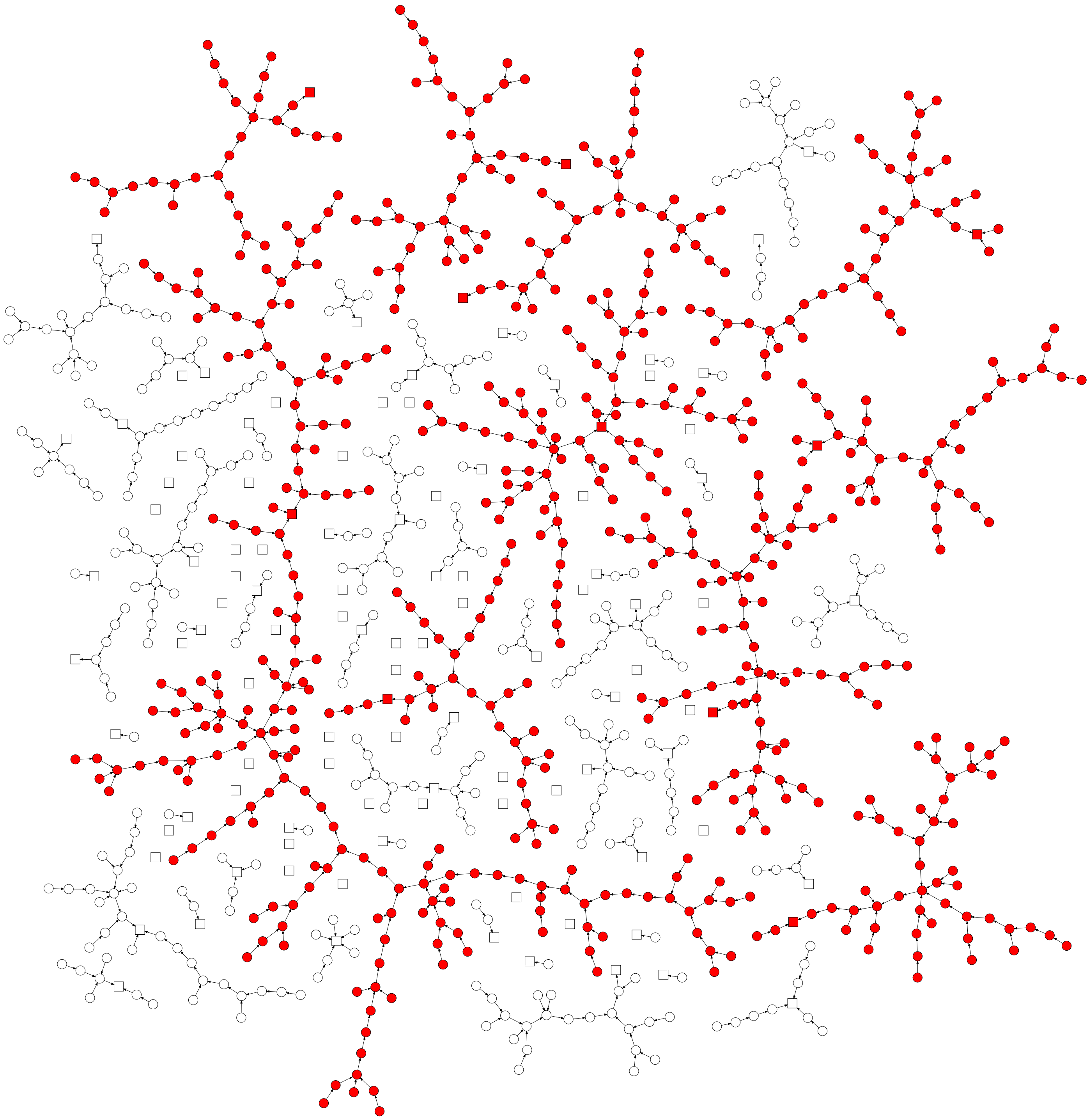
Are rho and kangaroo really
so different? Seek unification:
“kangarho”?

Are rho and kangaroo really so different? Seek unification: “kangarho”? Approved by Dan, not by Tanja: “kangarhoach”?

Are rho and kangaroo really so different? Seek unification: "kangarho"? Approved by Dan, not by Tanja: "kangarhoach"?

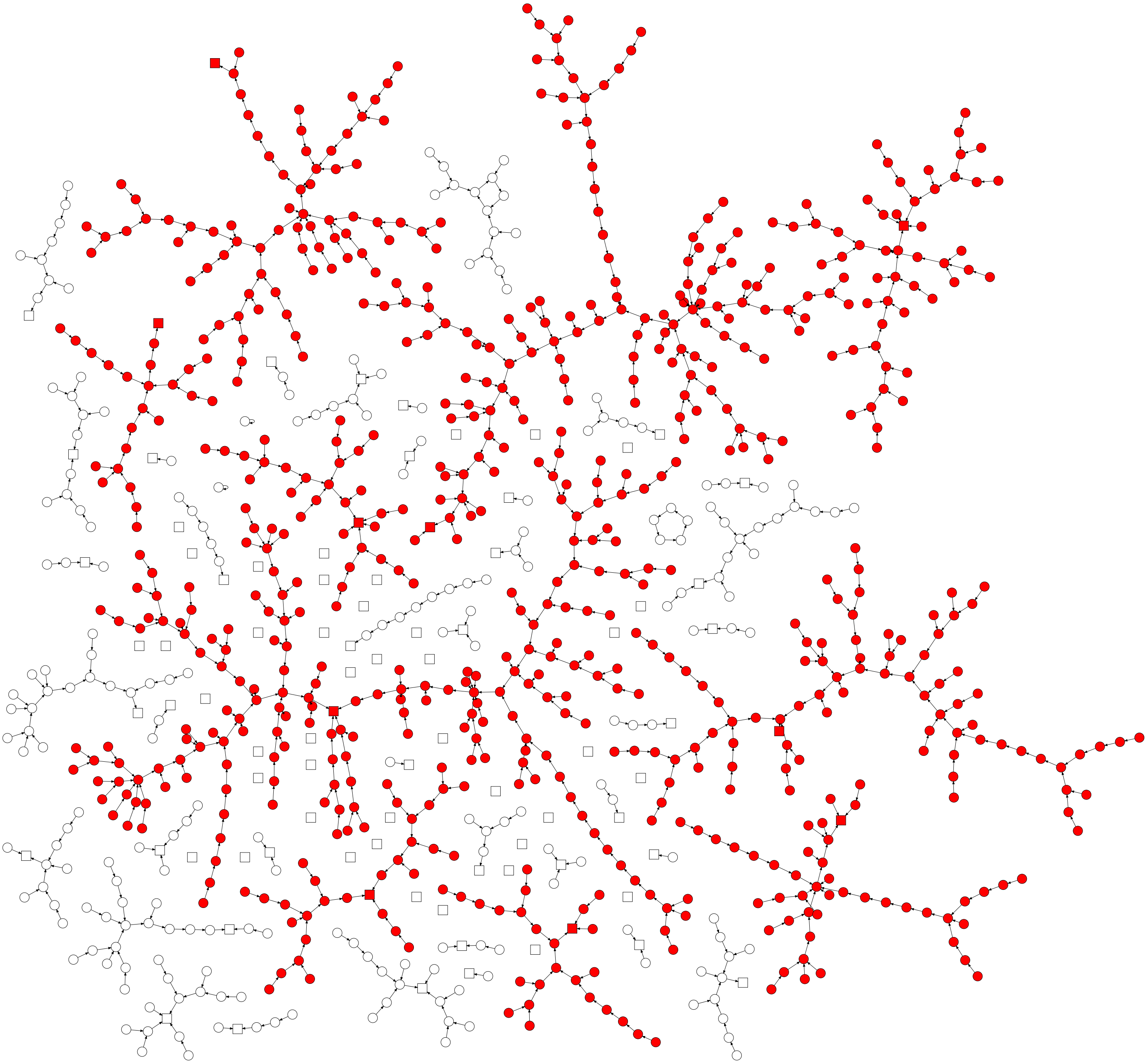
Some of our experiments for average ECDL computations using table of size $\approx\ell^{1/3}$ (selected from somewhat larger table):
for group of order ℓ,
precomputation $\approx 1.24\ell^{2/3}$,
main computation $\approx 1.77\ell^{1/3}$;
for interval of order ℓ,
precomputation $\approx 1.21\ell^{2/3}$,
main computation $\approx 1.93\ell^{1/3}$.

Not all DPs are equal



Ancestors of top 10 distinguished points are marked in red.

Not all f's are equal



697 red ancestors.

Previous picture had 603.