Modeling the Security of Cryptography, Part 2: Public-Key Cryptography

Tanja Lange
Technische Universiteit Eindhoven

Joint work with:
Daniel J. Bernstein
University of Illinois at Chicago &
Technische Universiteit Eindhoven

---

eprint.iacr.org/2012/318,
eprint.iacr.org/2012/458

Similar public-key story.

Define t-insecurity of RSA-1024 as maximum success probability of all attacks that cost $\leq t$.

Similar public-key story.

Define t-insecurity of RSA-1024 as maximum success probability of all attacks that cost $\leq t$.

Prove, e.g., that bounds on insecurity of RSA-1024 imply similar bounds on insecurity of RSA-1024-PSS.

Similar public-key story.

Define t-insecurity of RSA-1024 as maximum success probability of all attacks that cost $\leq t$.

Prove, e.g., that bounds on insecurity of RSA-1024 imply similar bounds on insecurity of RSA-1024-PSS.

Conjecture bounds on insecurity of RSA-1024: e.g., "it takes time $Ce^{1.923(\log N)^{1/3}(\log \log N)^{2/3}}$ to invert RSA".

DL-based systems

E.g. forward security setting in TLS uses DH-key exchange on elliptic curve NIST P-256.

Break by solving ECDL in group of prime order $\ell \approx 2^{256}$.
ECDL input: points P, Q, where P is a standard generator.
ECDL output: $\log_P Q$.

DL-based systems

E.g. forward security setting in TLS uses DH-key exchange on elliptic curve NIST P-256.

Break by solving ECDL in group of prime order $\ell \approx 2^{256}$. ECDL input: points P, Q, where P is a standard generator. ECDL output: $\log_P Q$.

Standard conjecture: For each $p \in [0, 1]$, each P-256 ECDL algorithm with success probability $\geq p$ takes "time" $\geq 2^{128} p^{1/2}$.

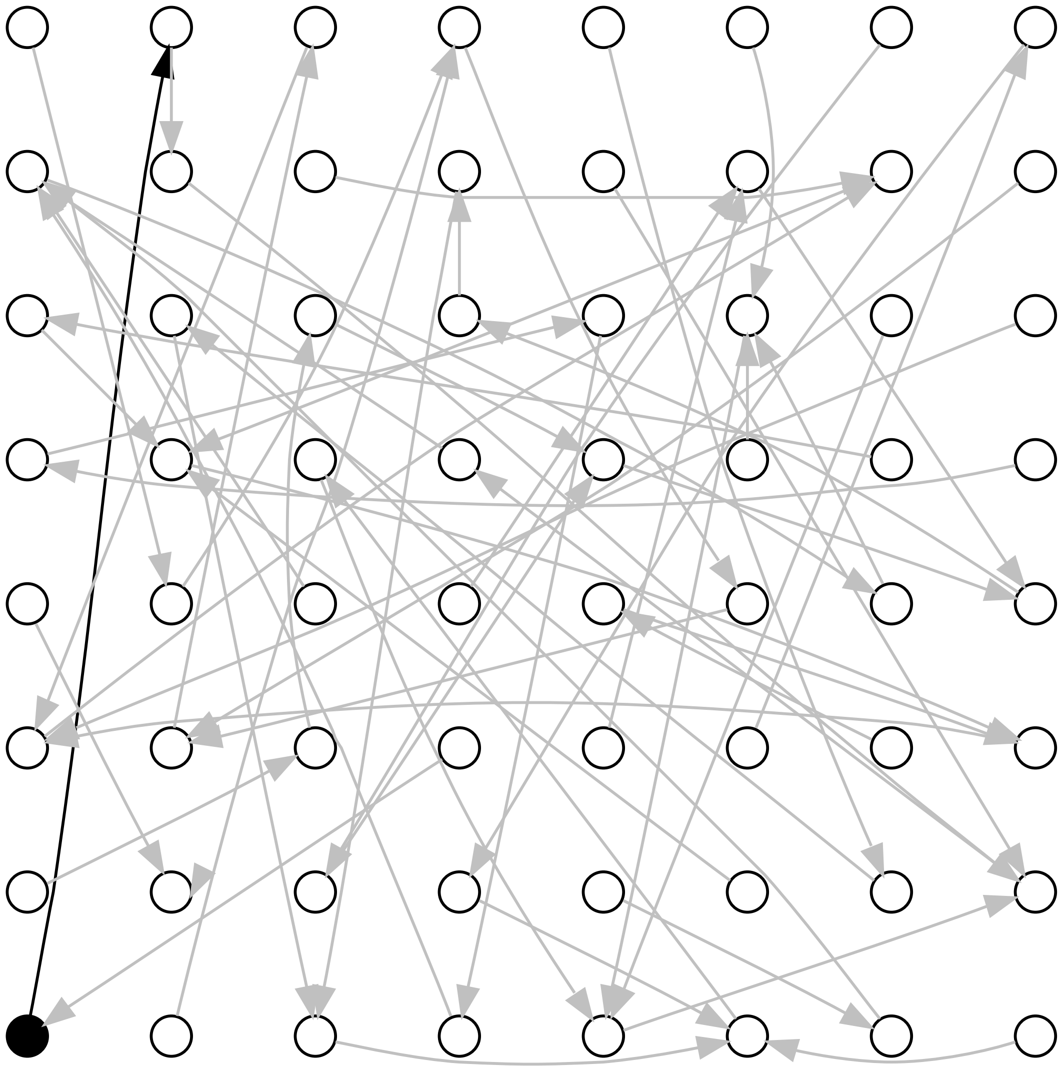
The rho method

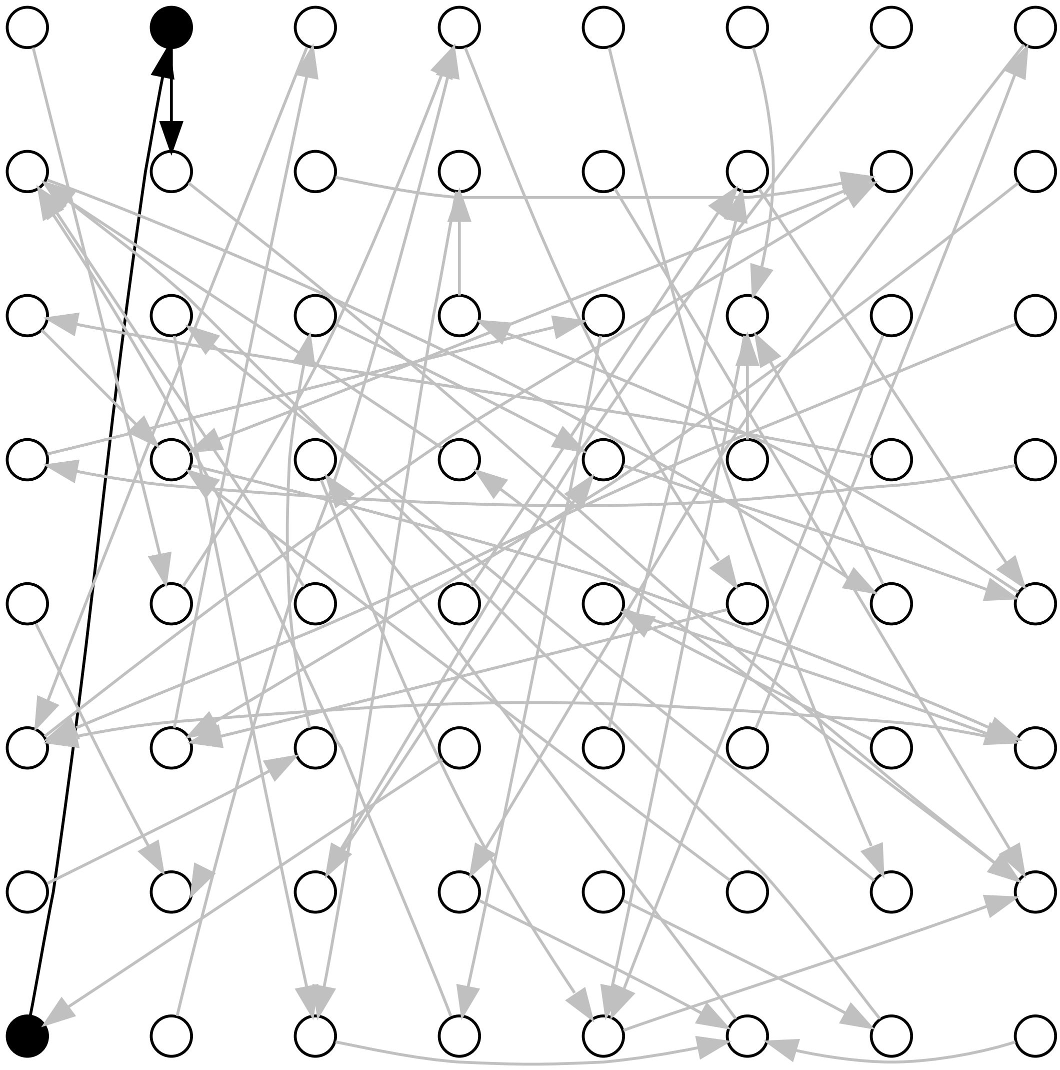
Simplified, non-parallel rho:

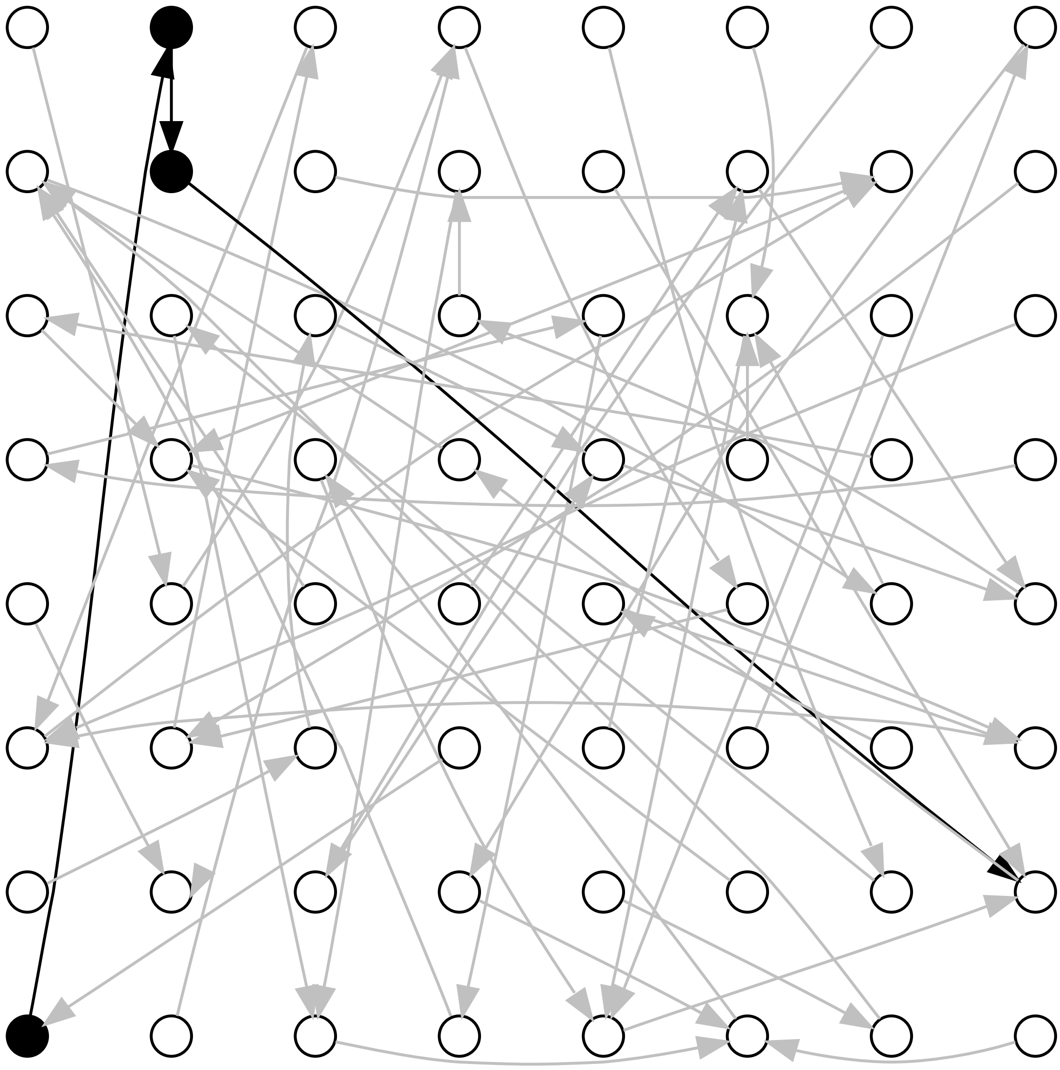
Make a pseudo-random walk $R_0, R_1, R_2, \ldots$ in the group $\langle P \rangle$, where current point determines the next point: $R_{i+1} = f(R_i)$.

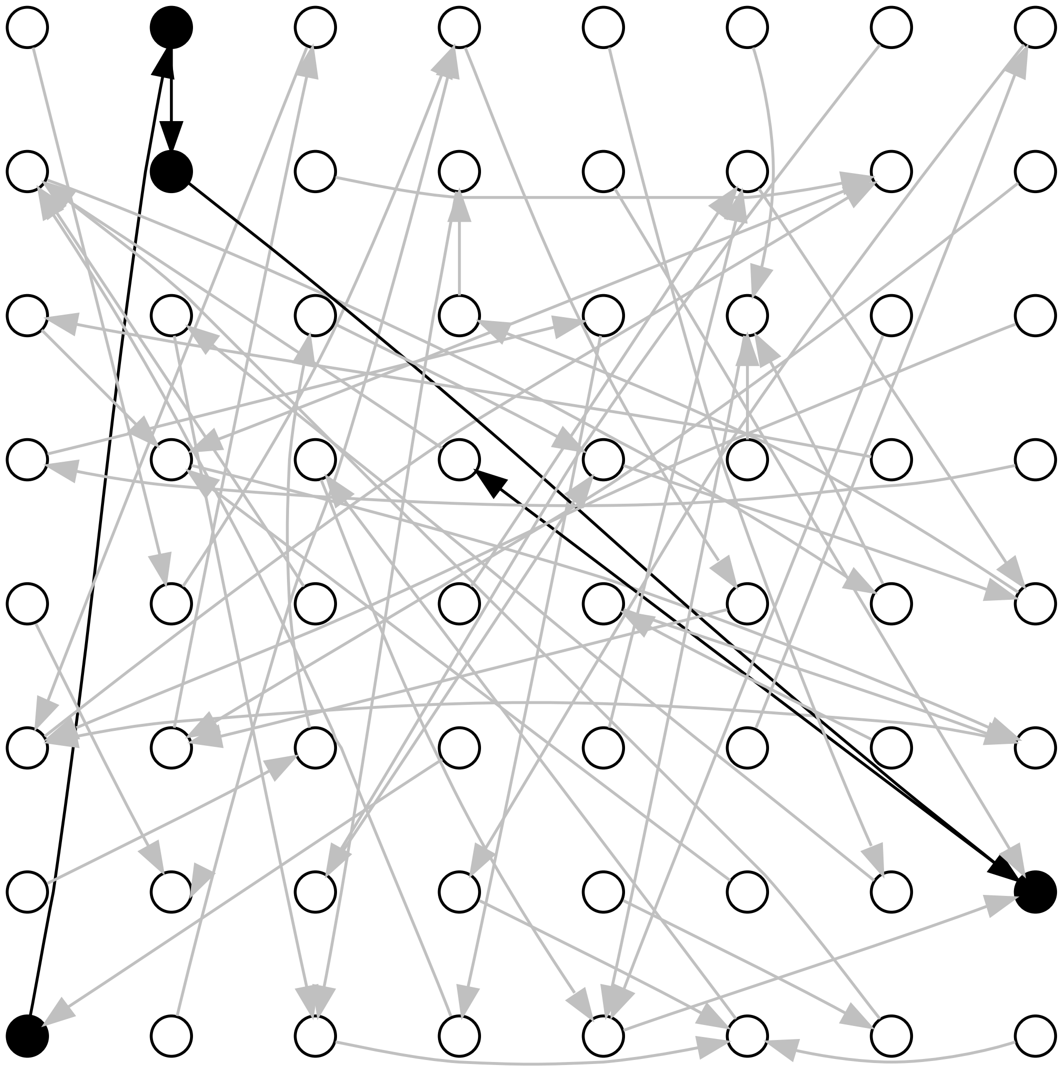
Birthday paradox:
Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi\ell/2}$ draws.

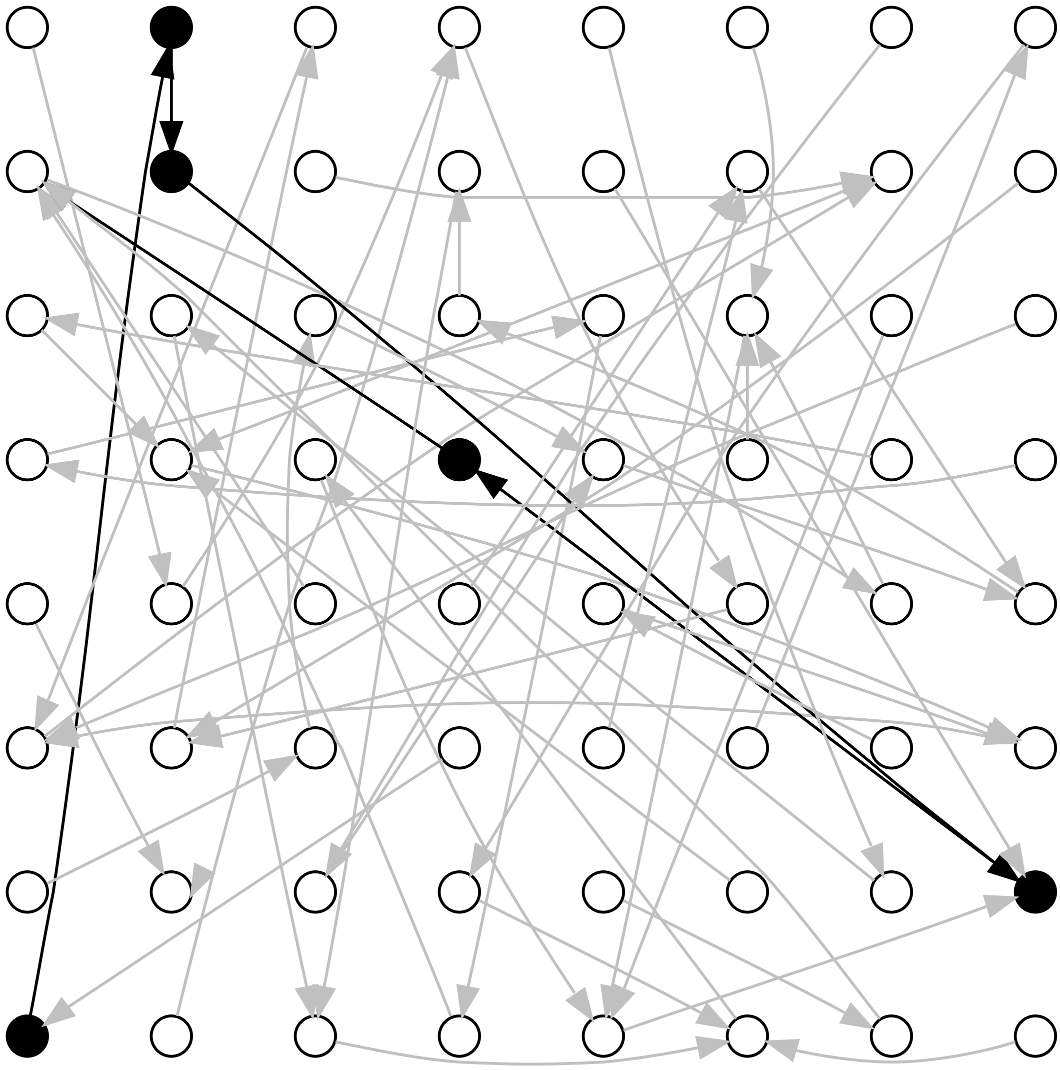
The walk now enters a cycle.
Cycle-finding algorithm (e.g., Floyd) quickly detects this.

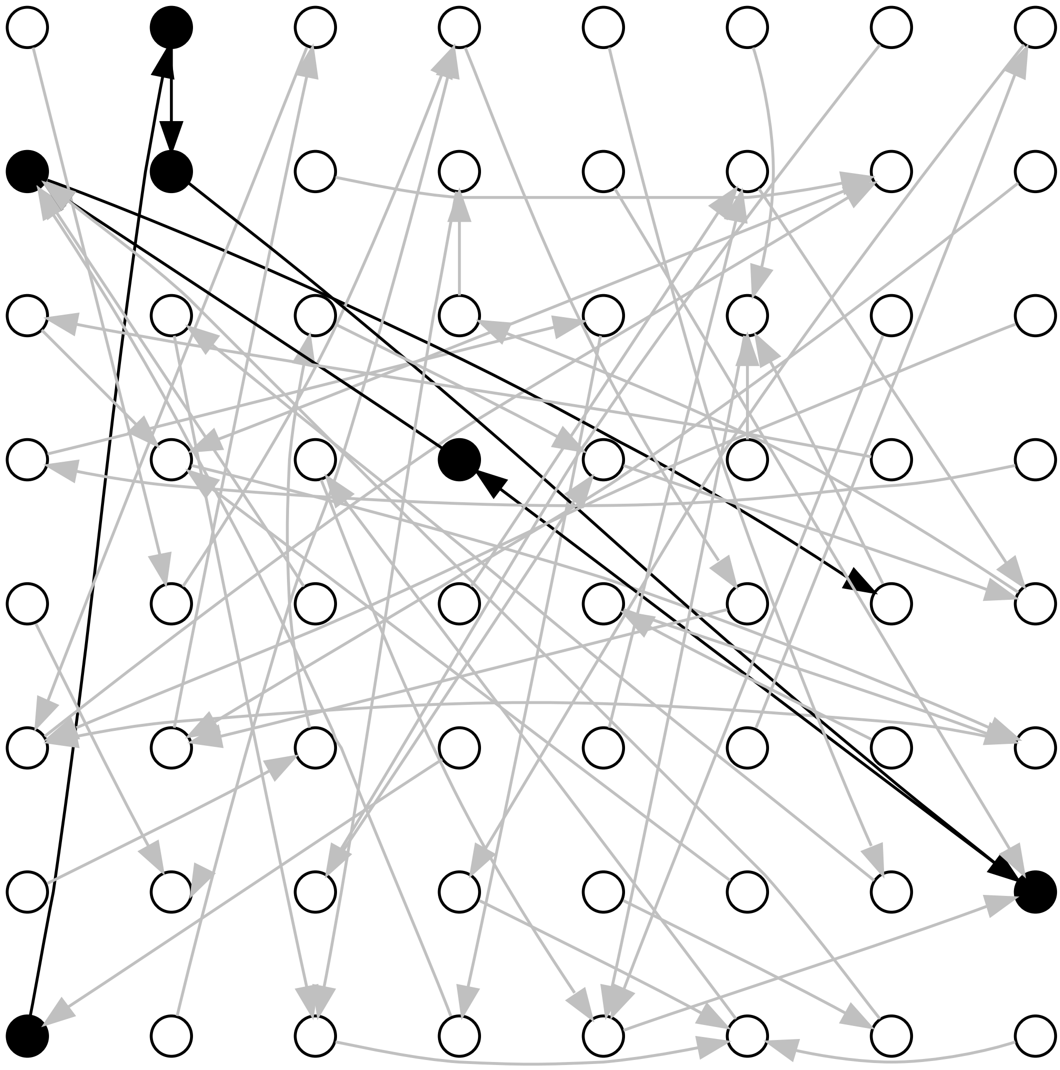


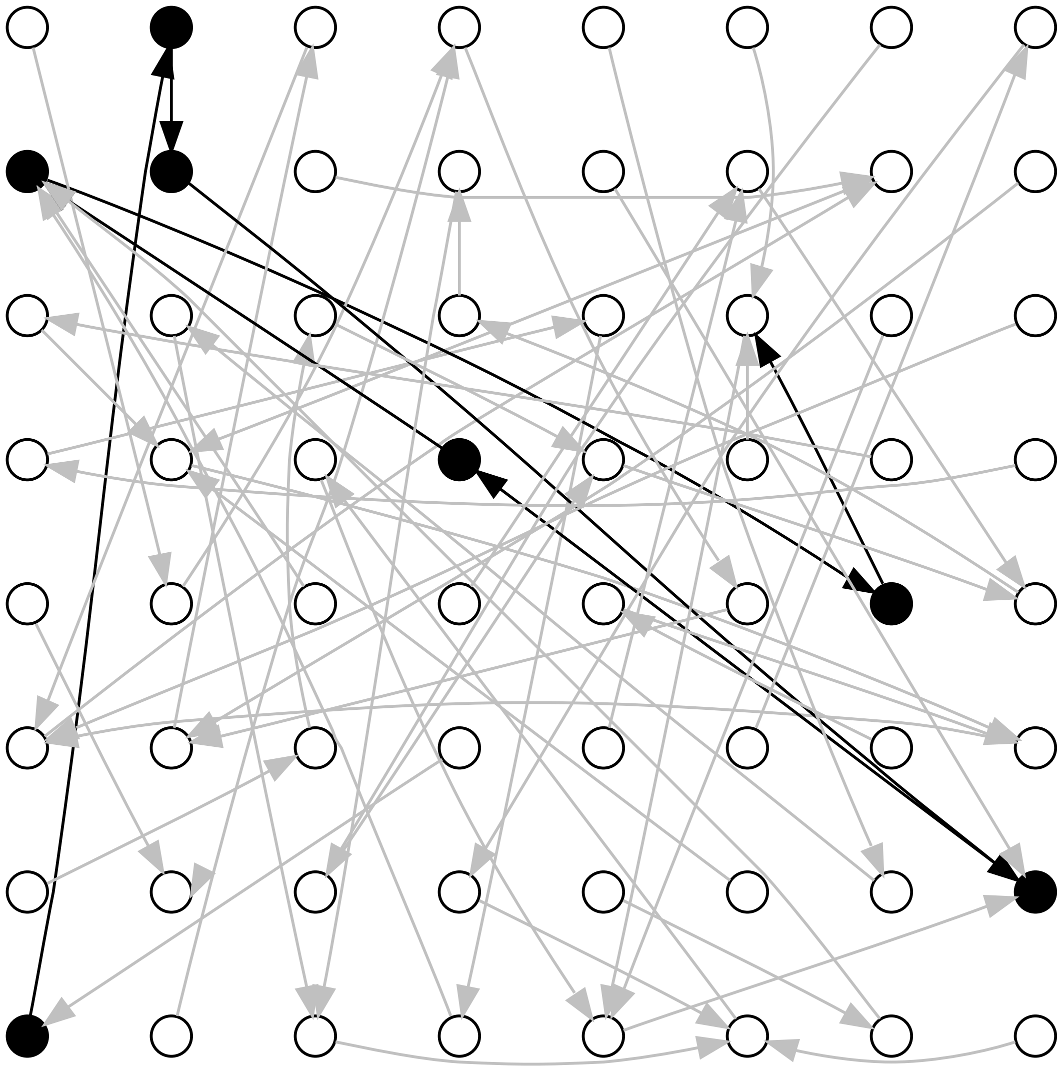


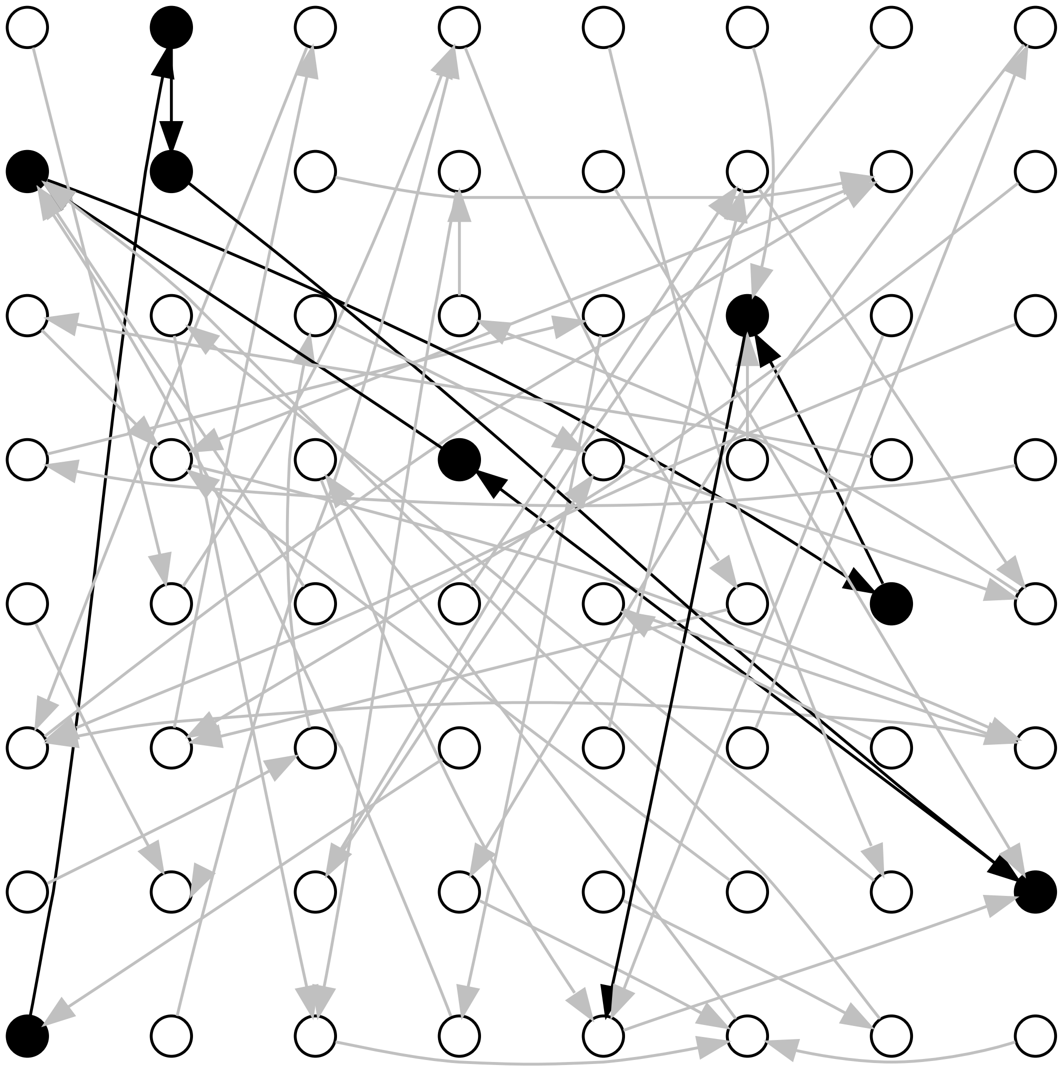


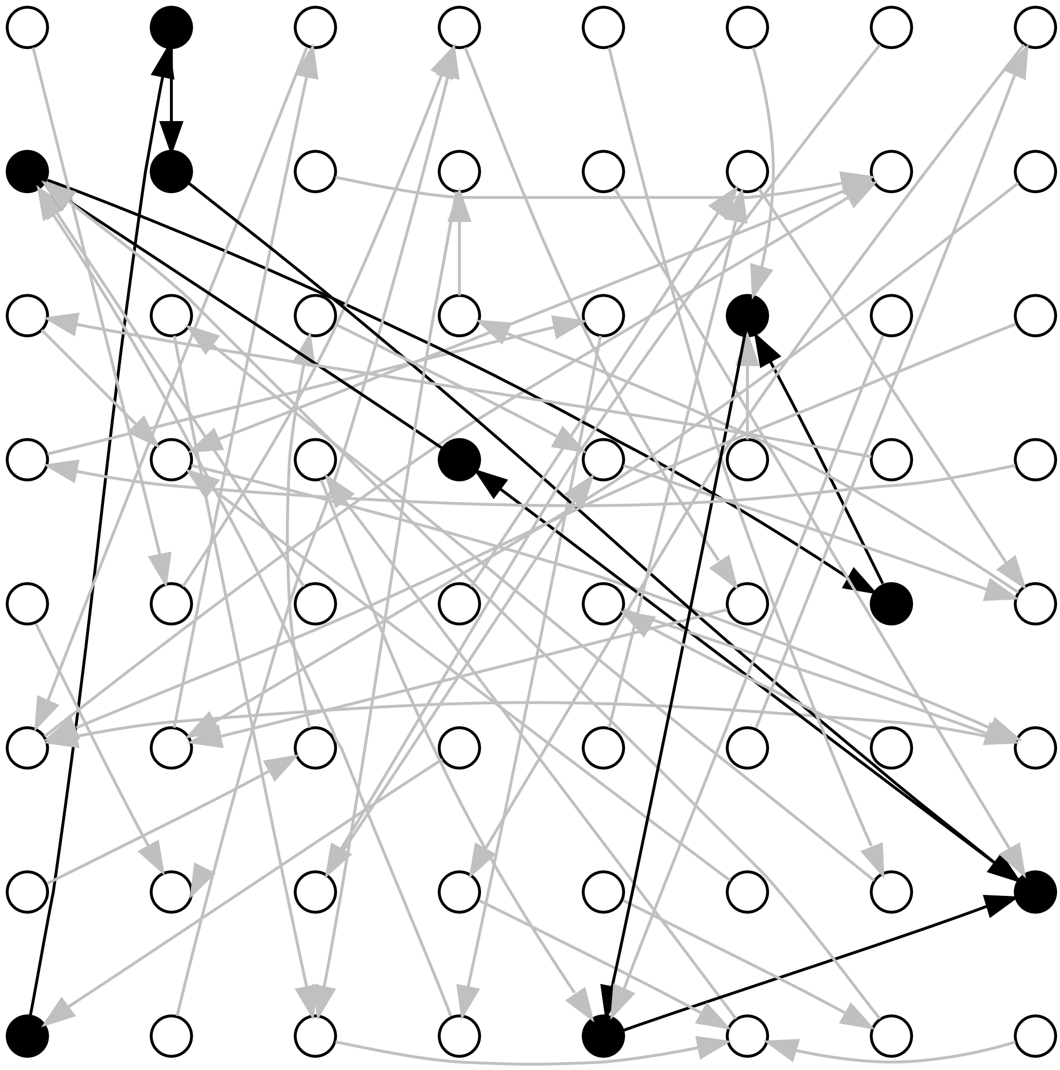


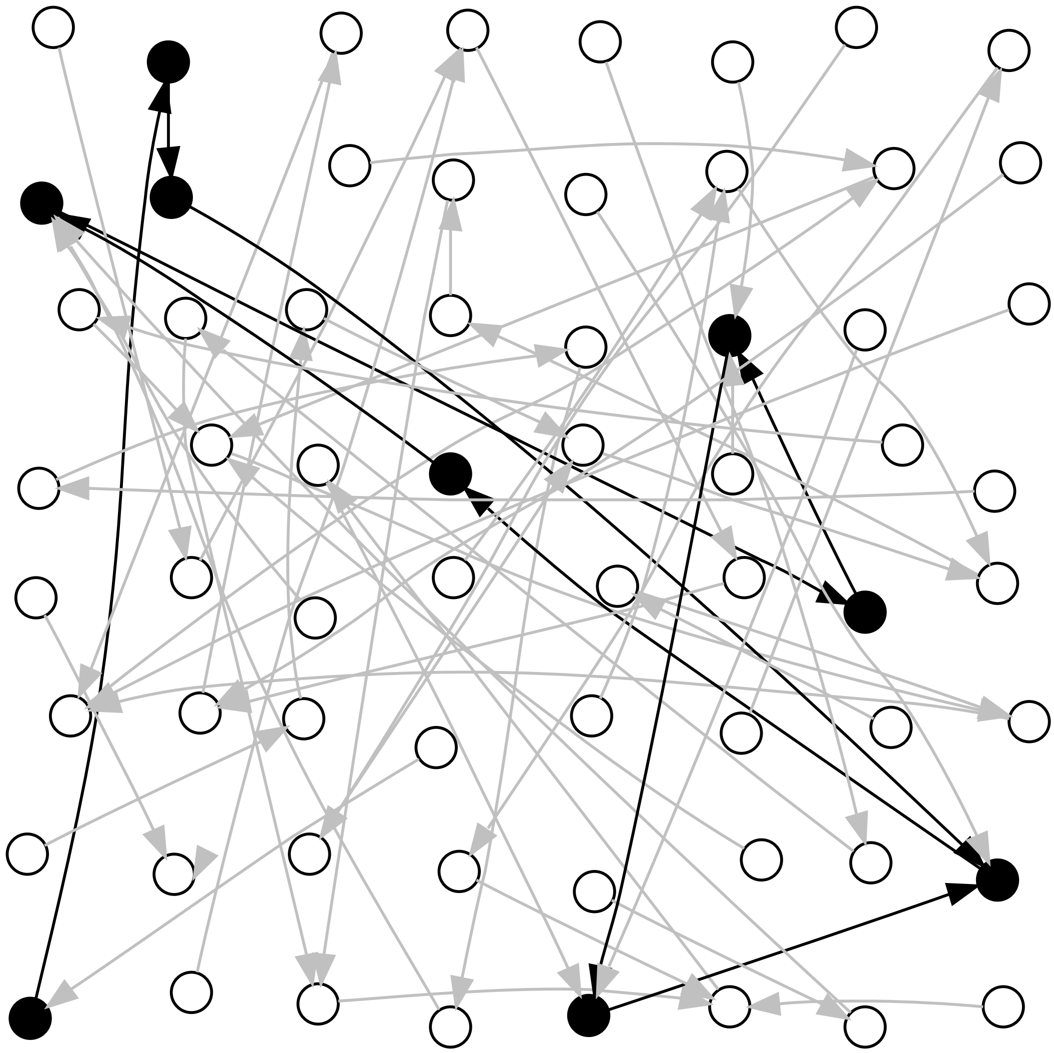


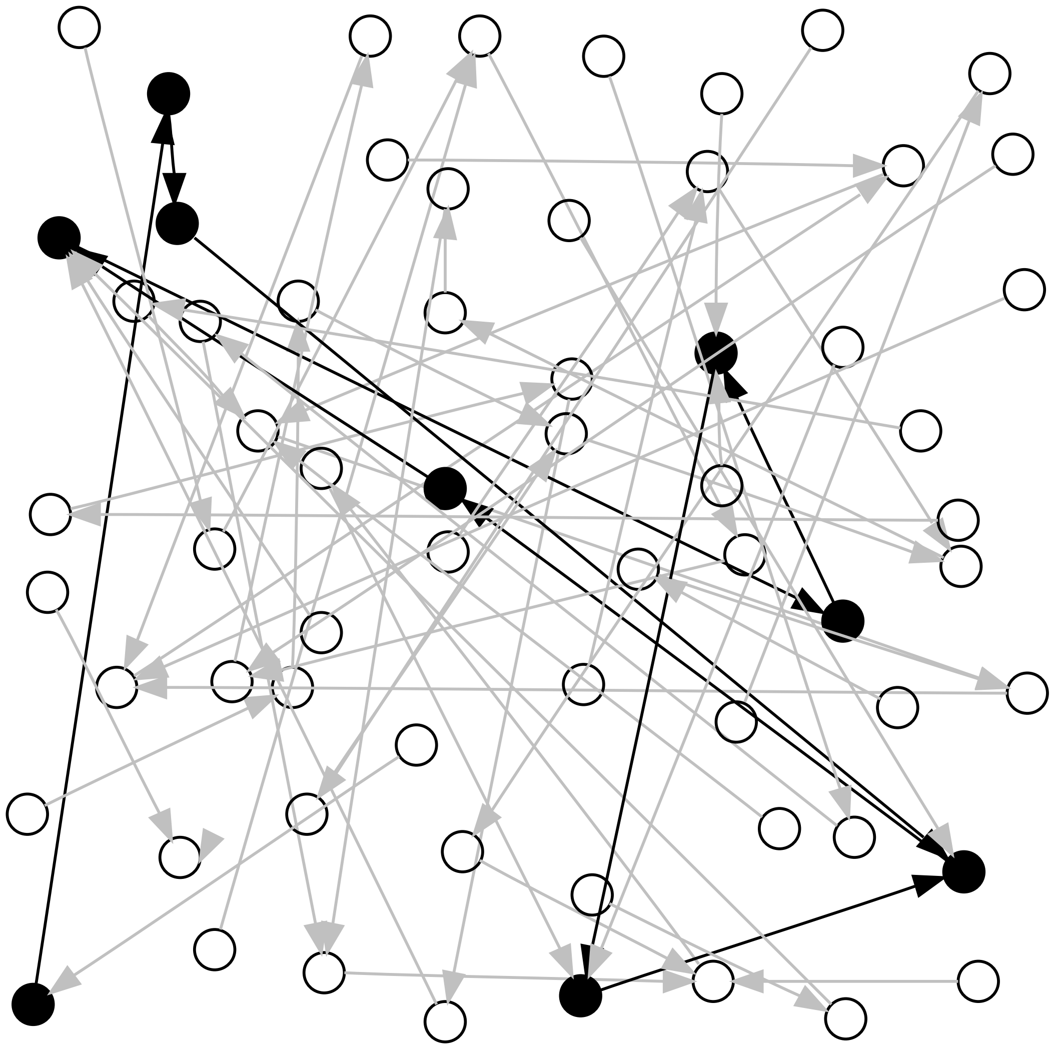


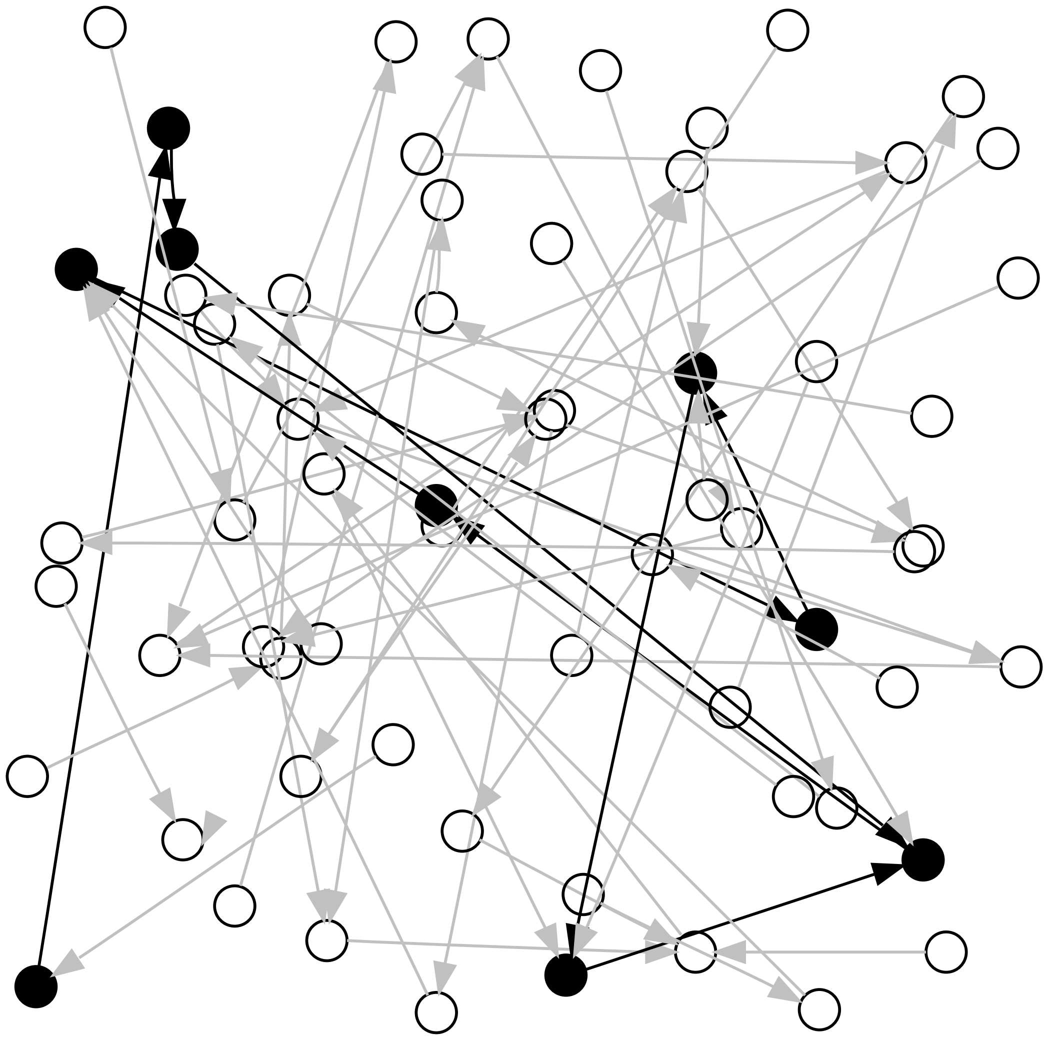


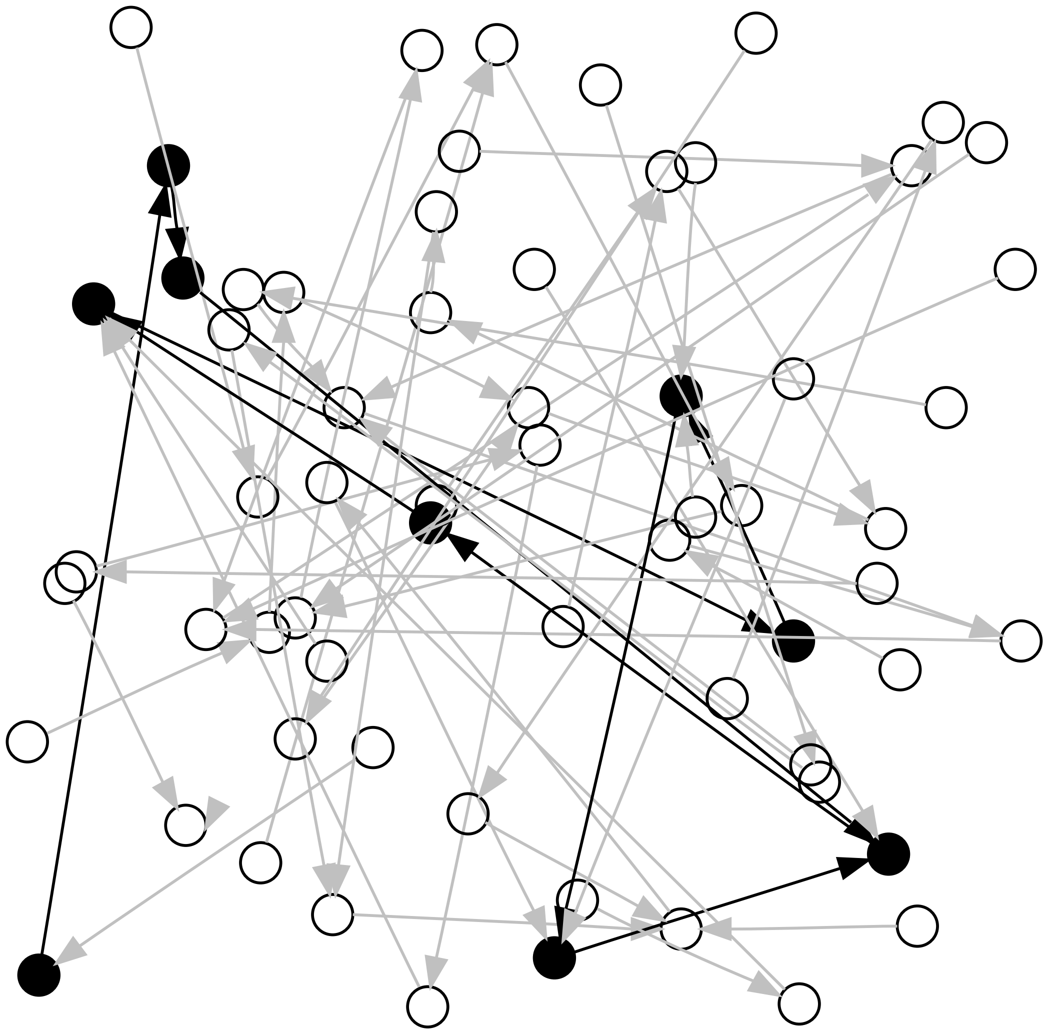


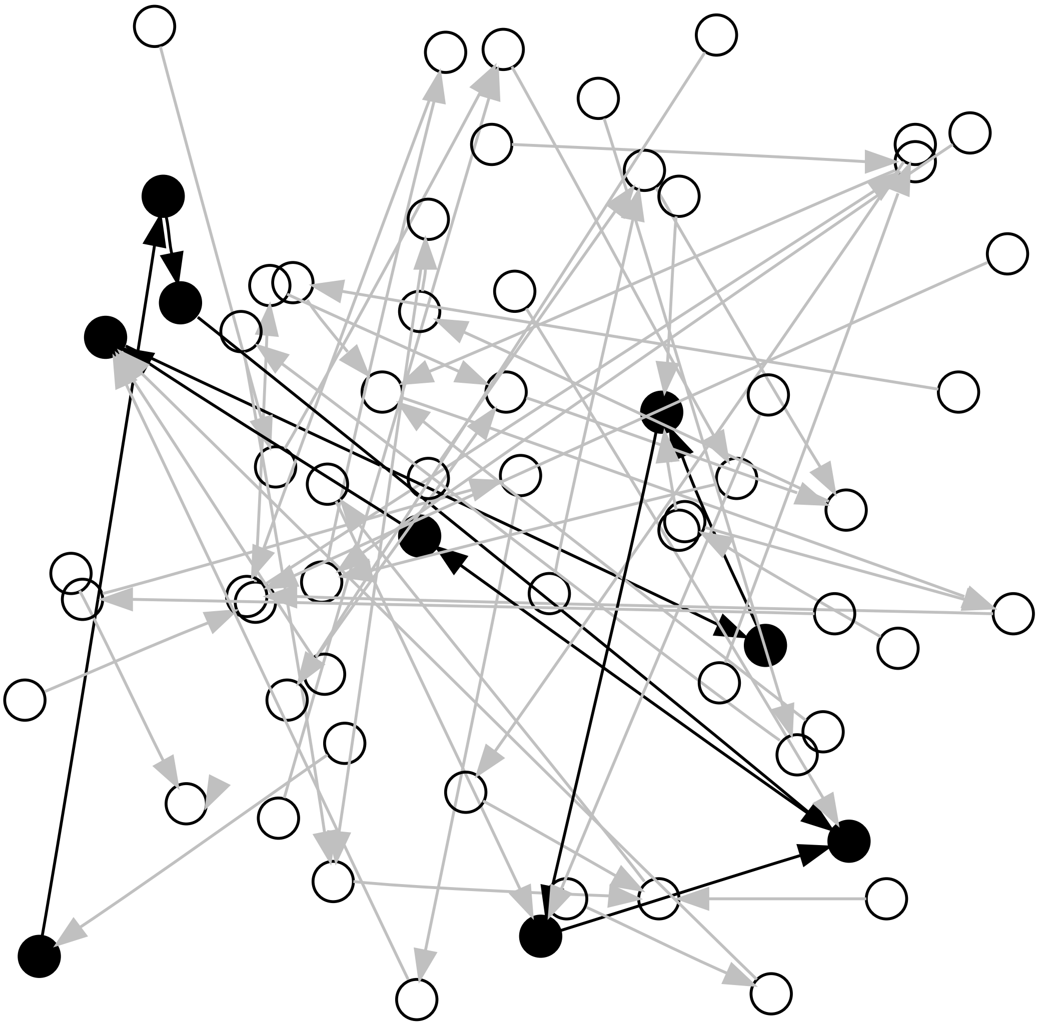


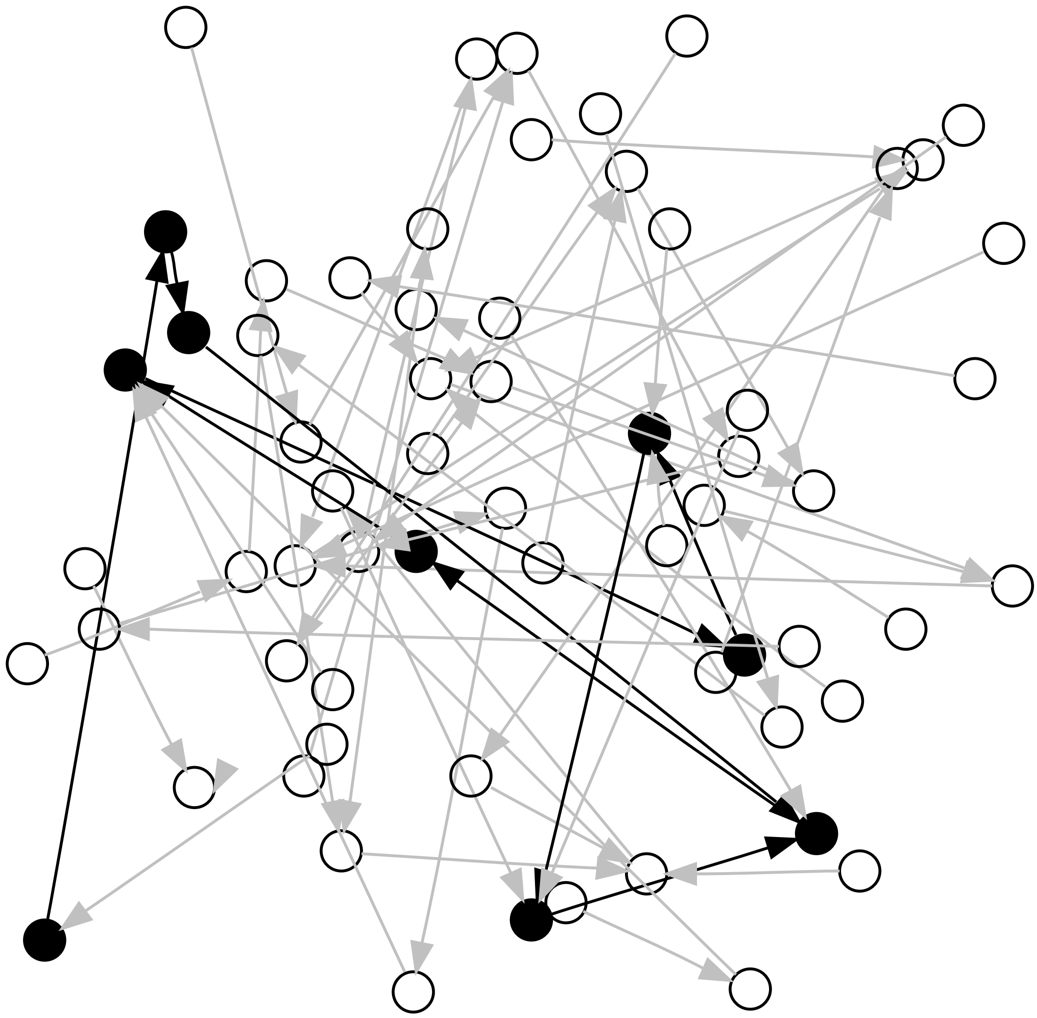


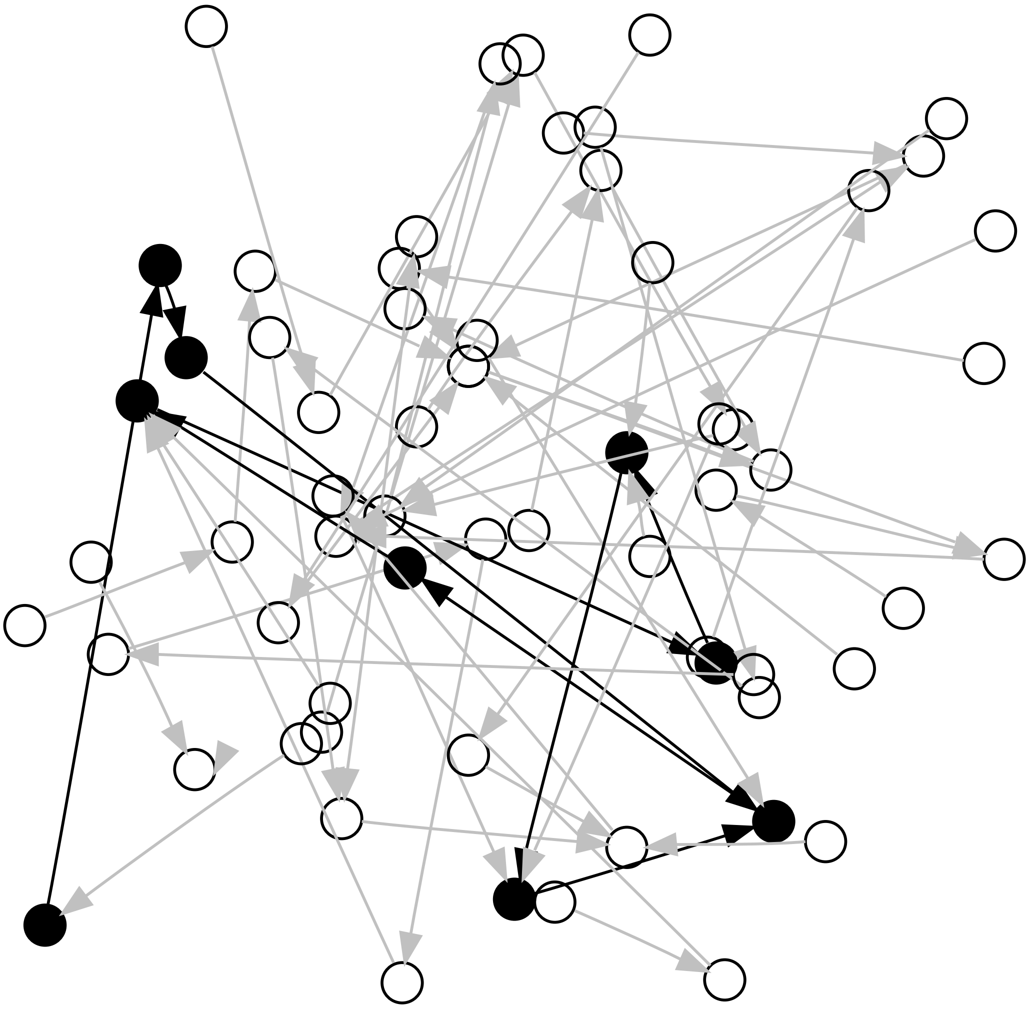


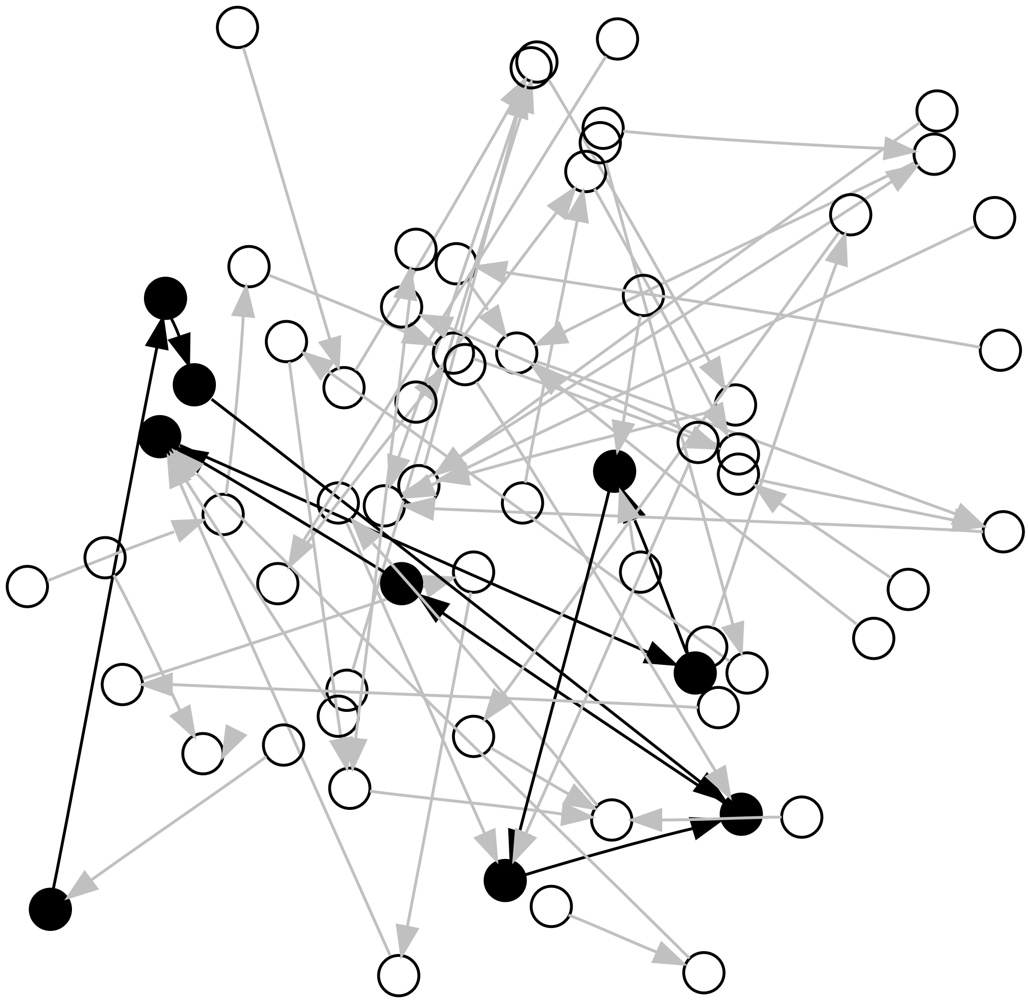


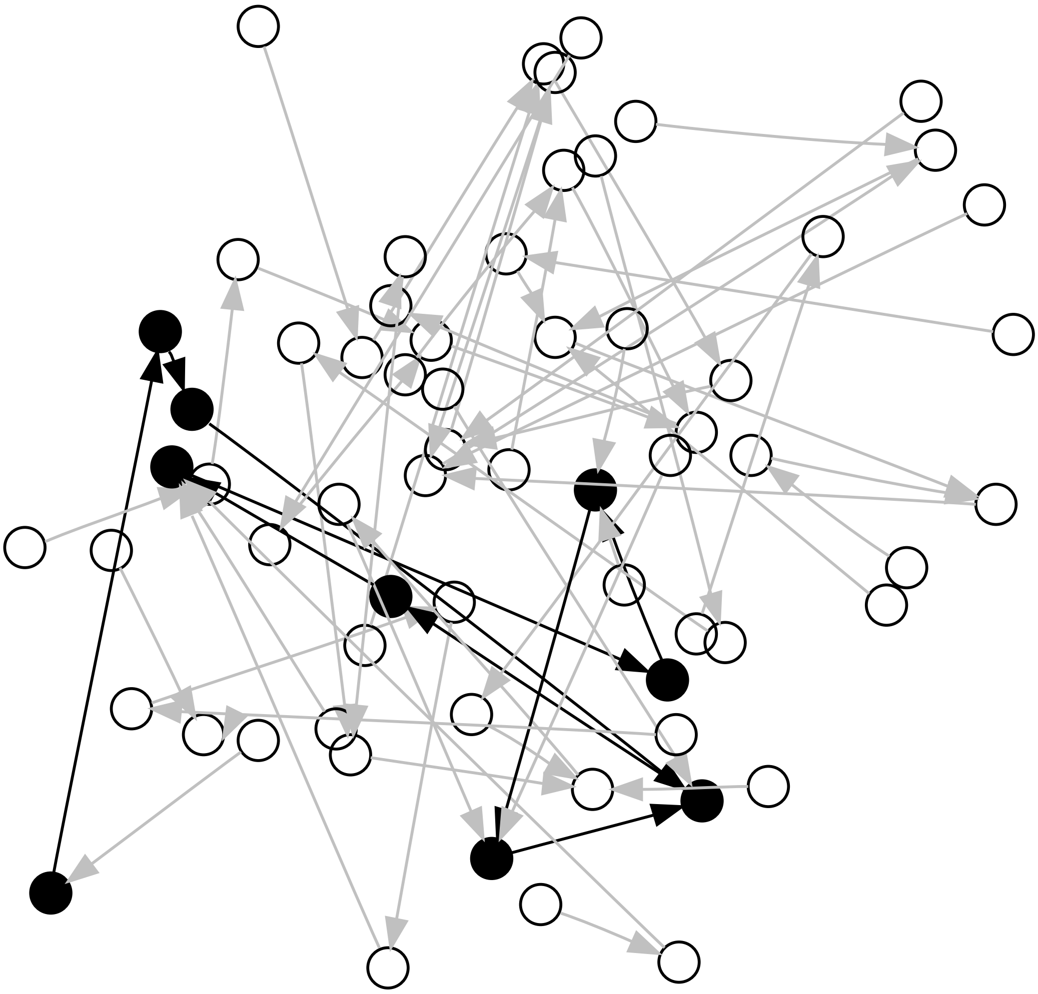


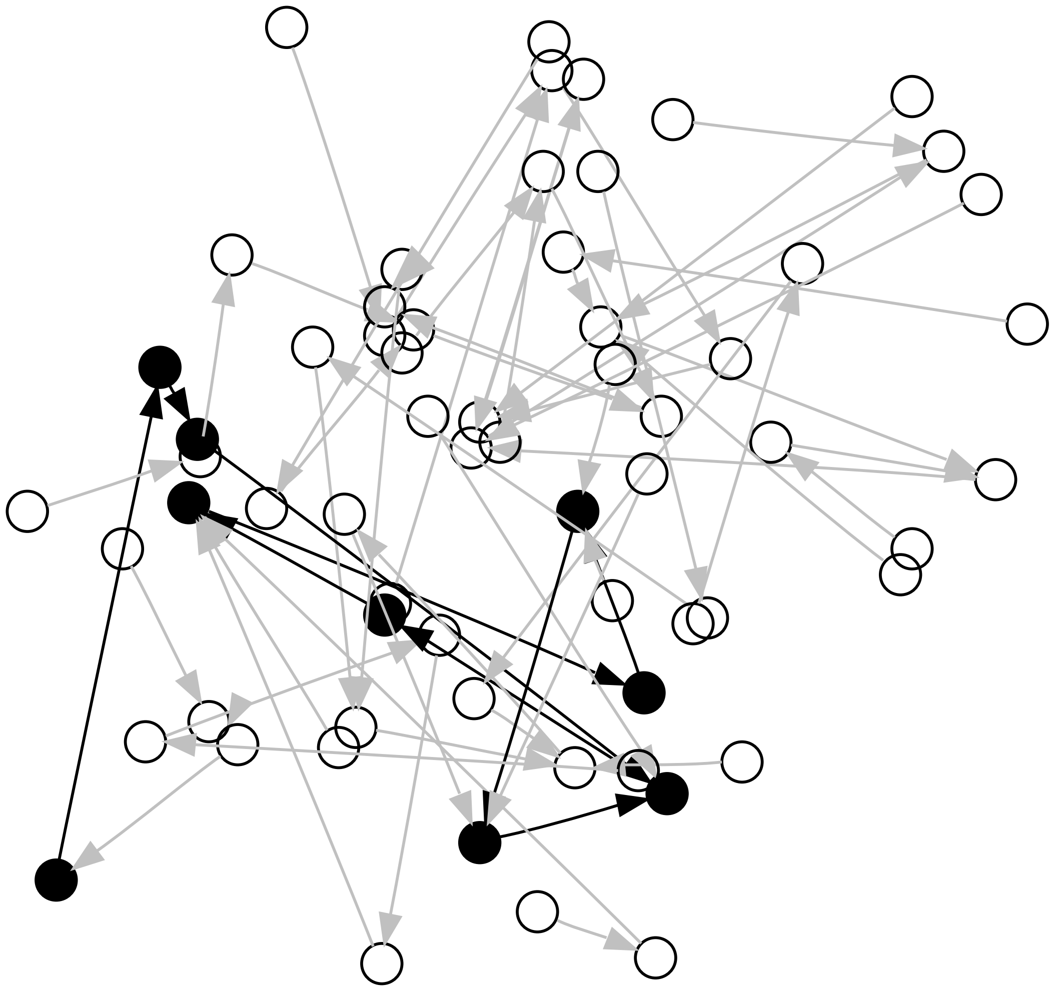


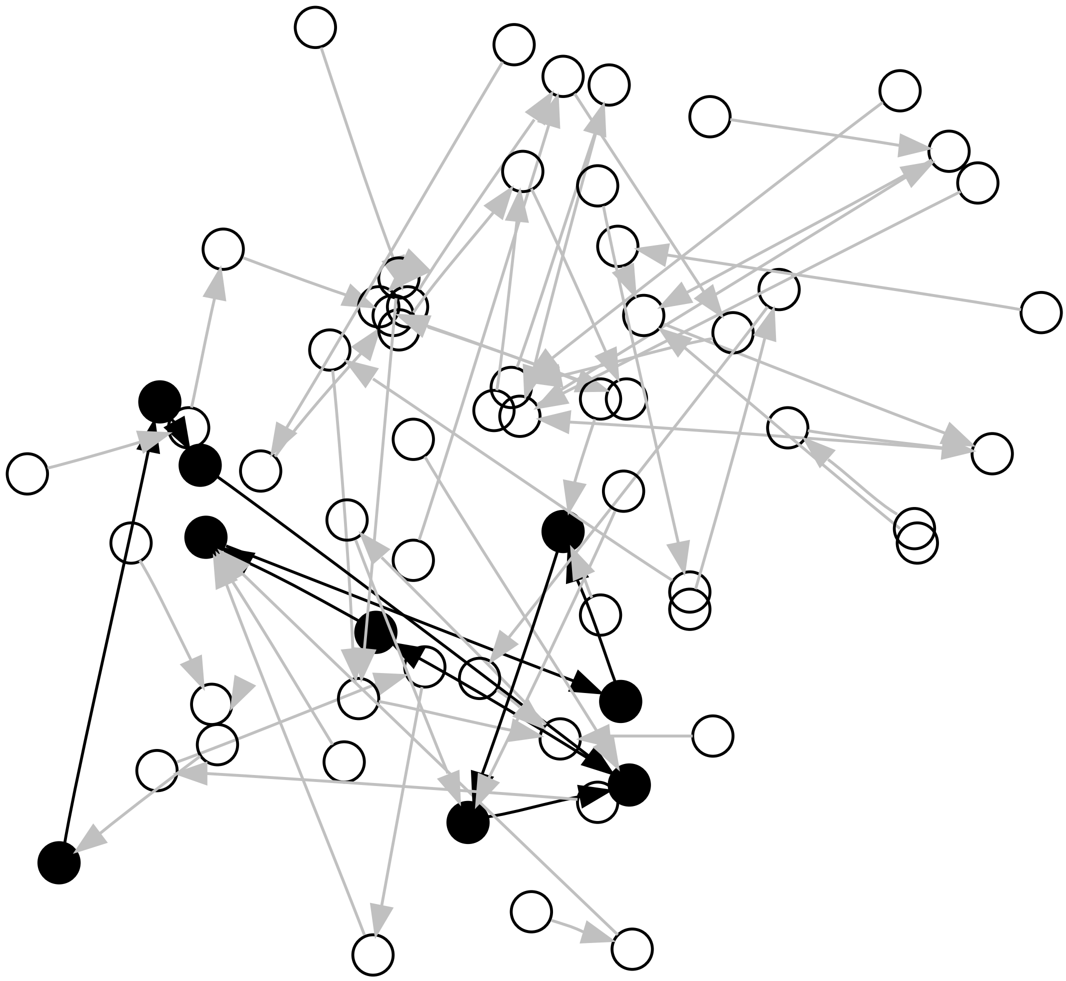


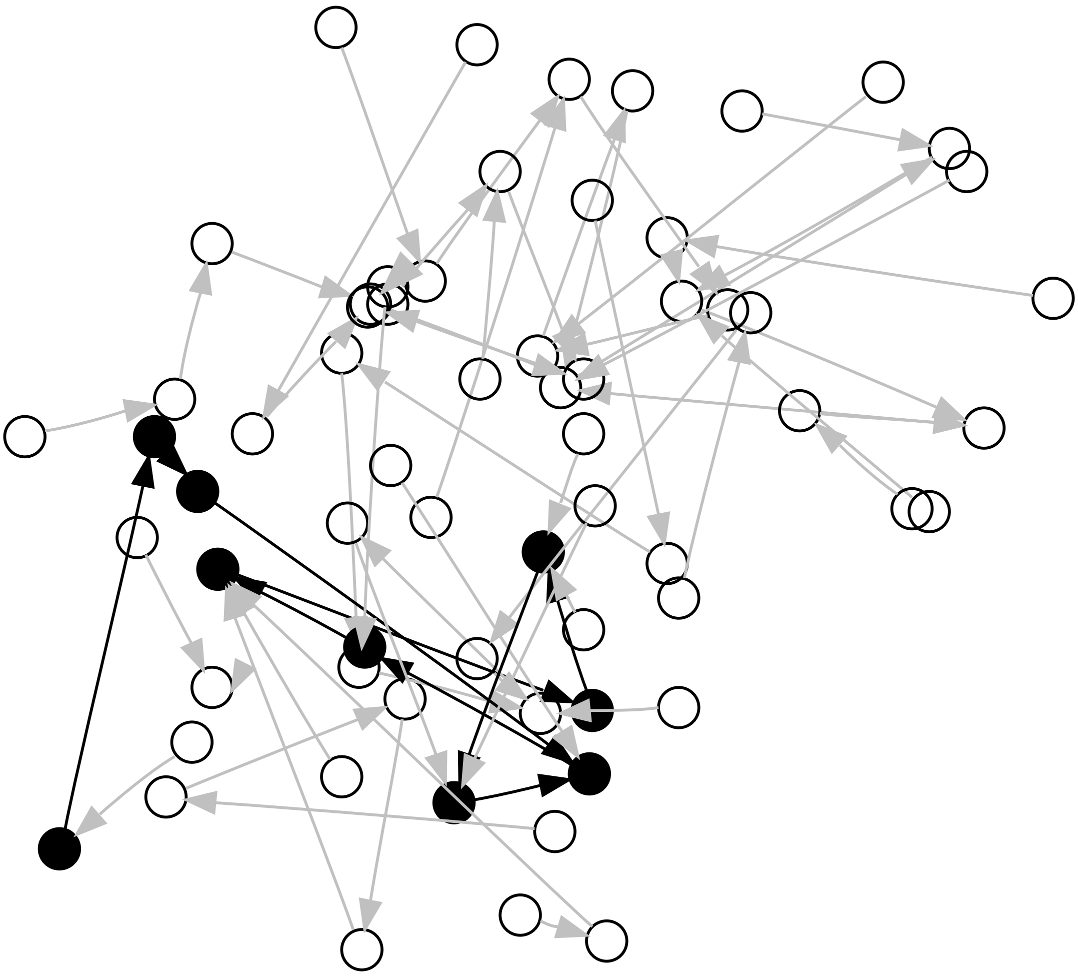


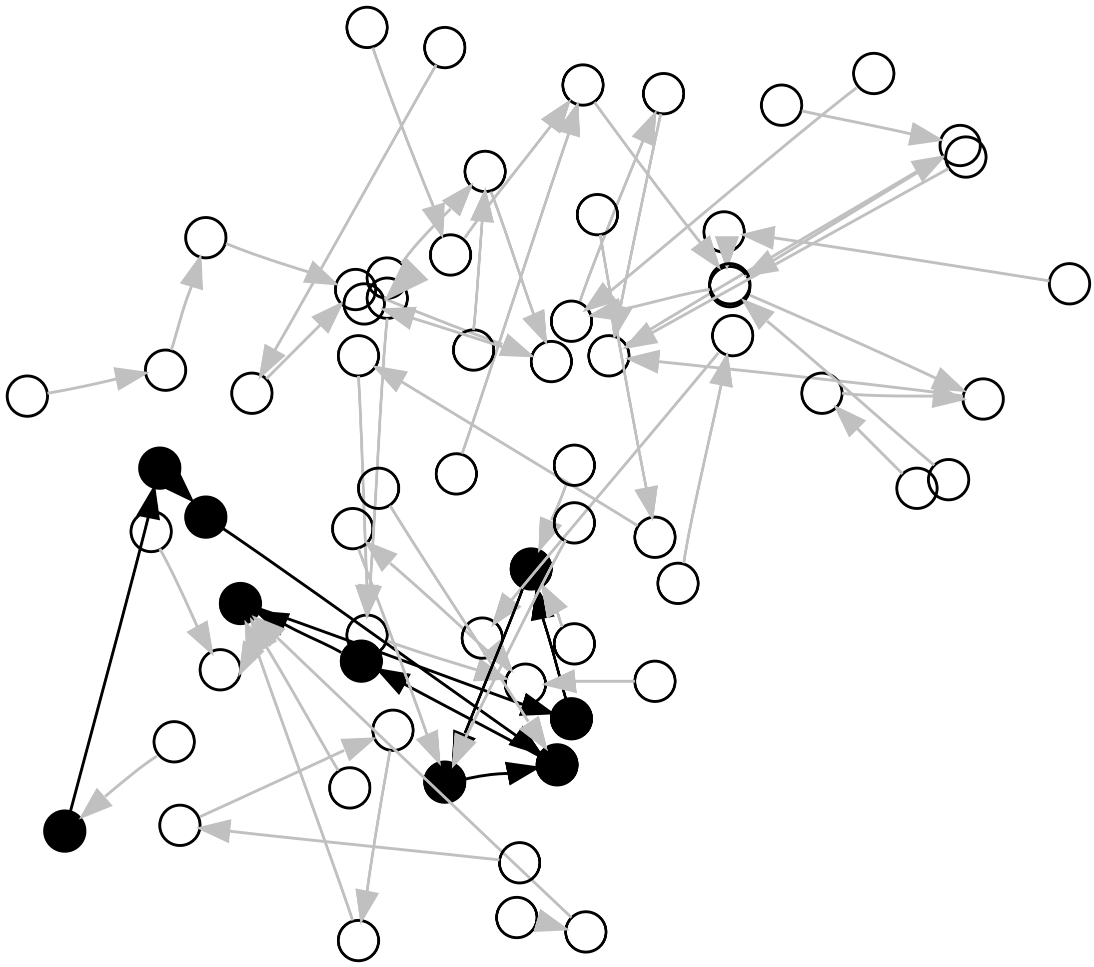


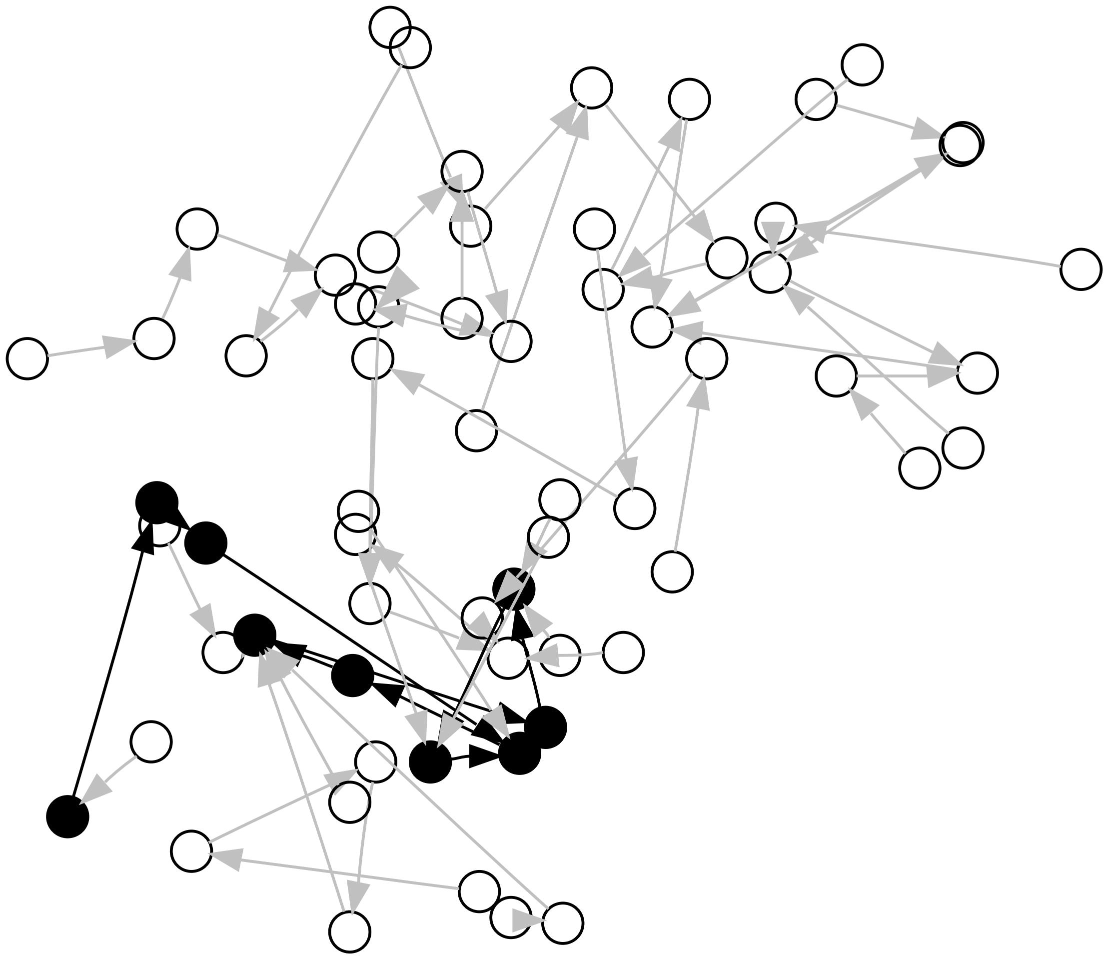


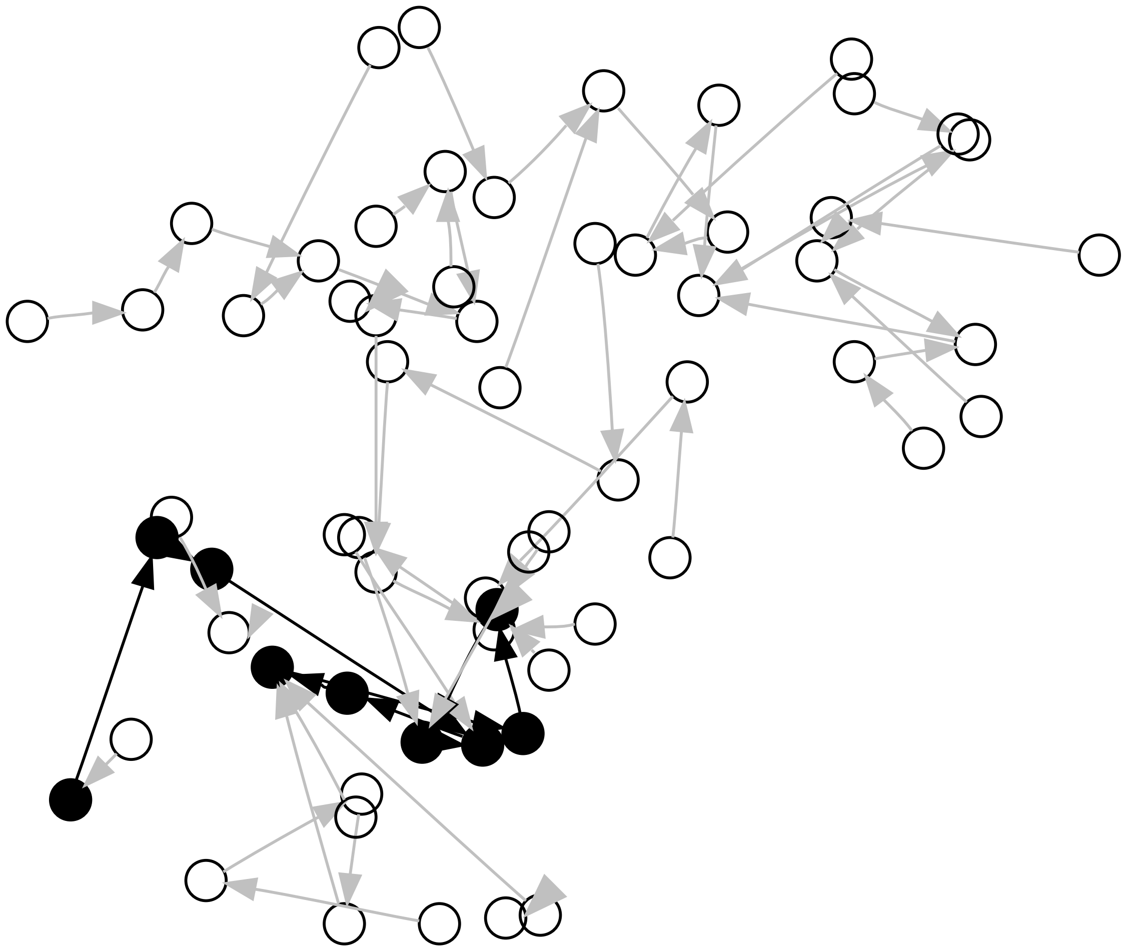


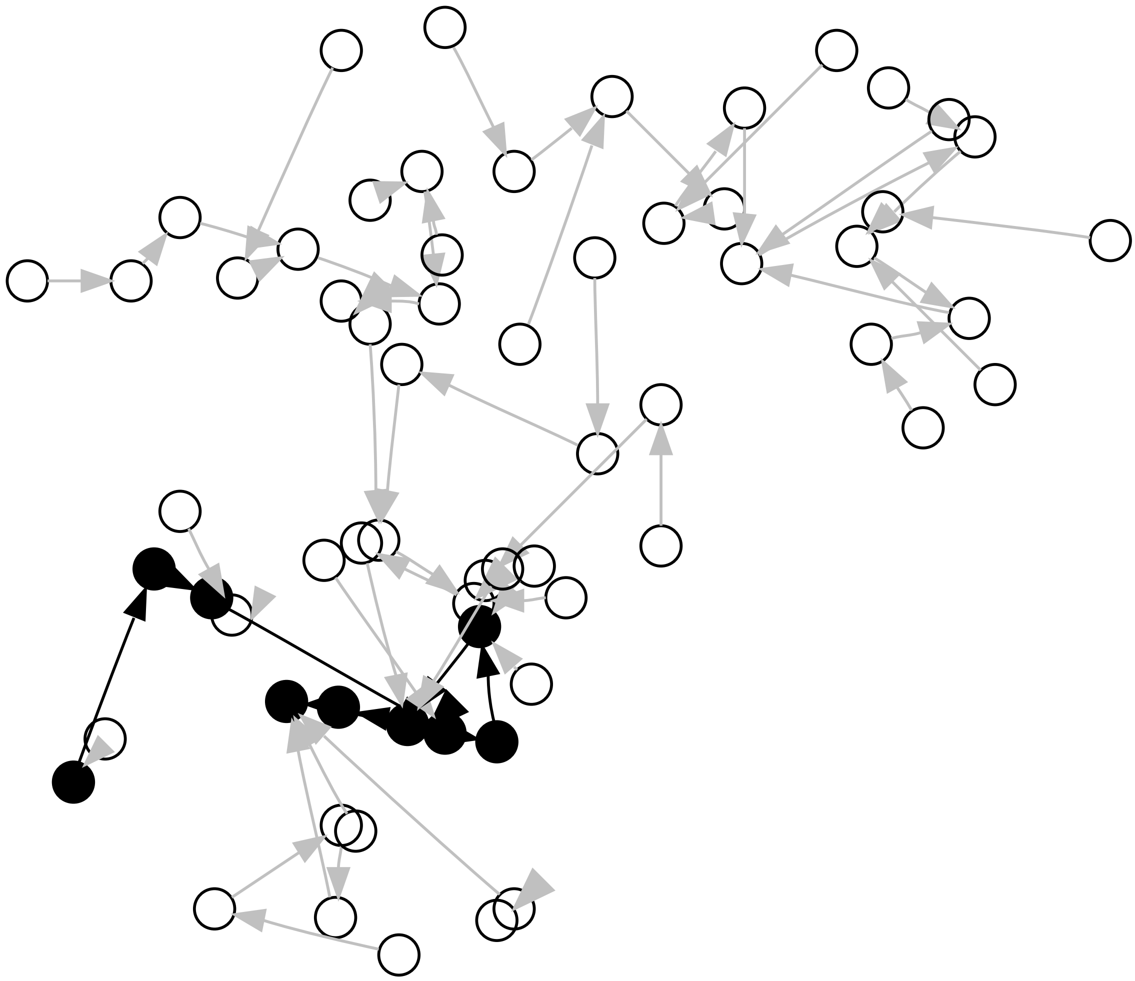


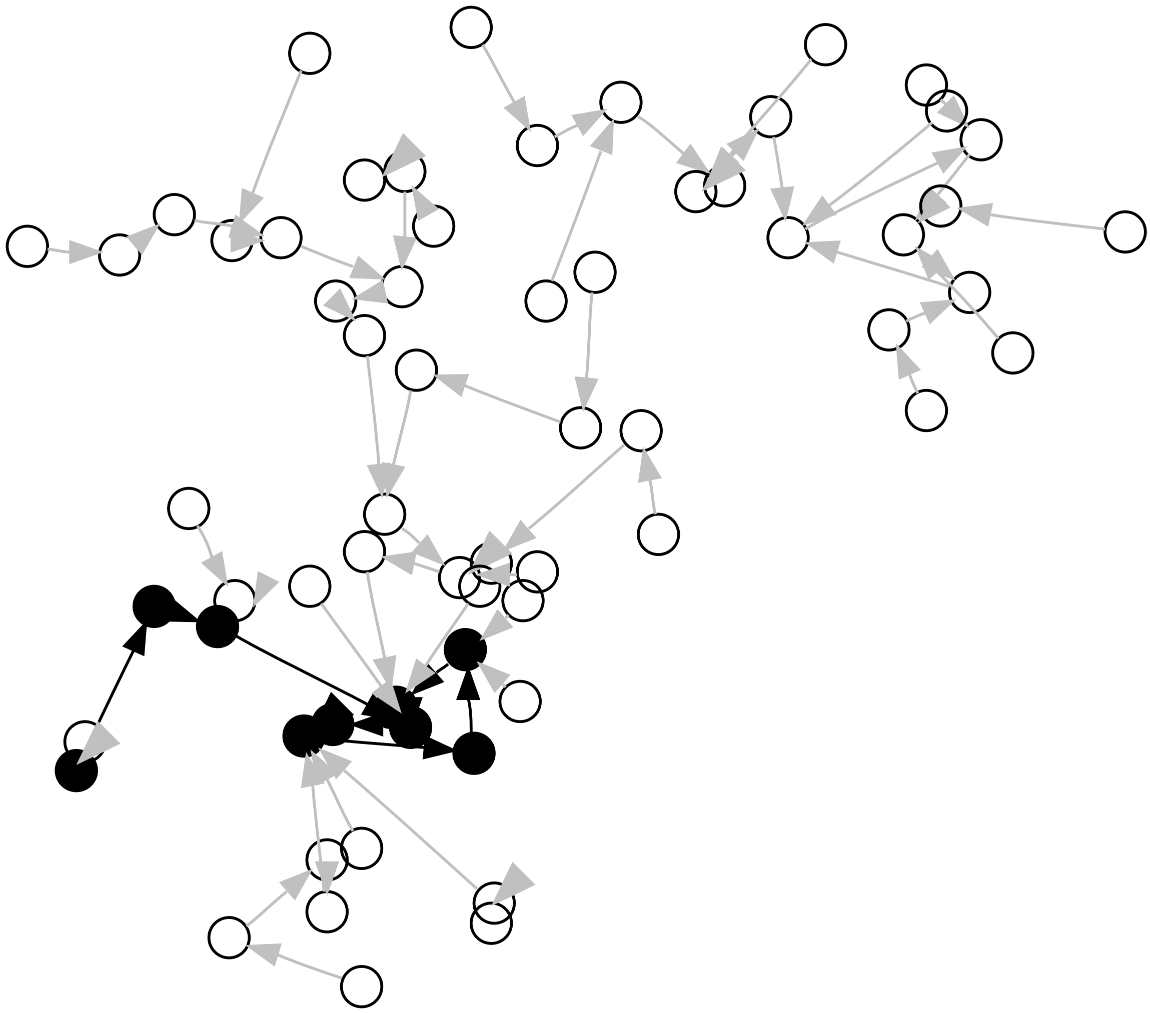


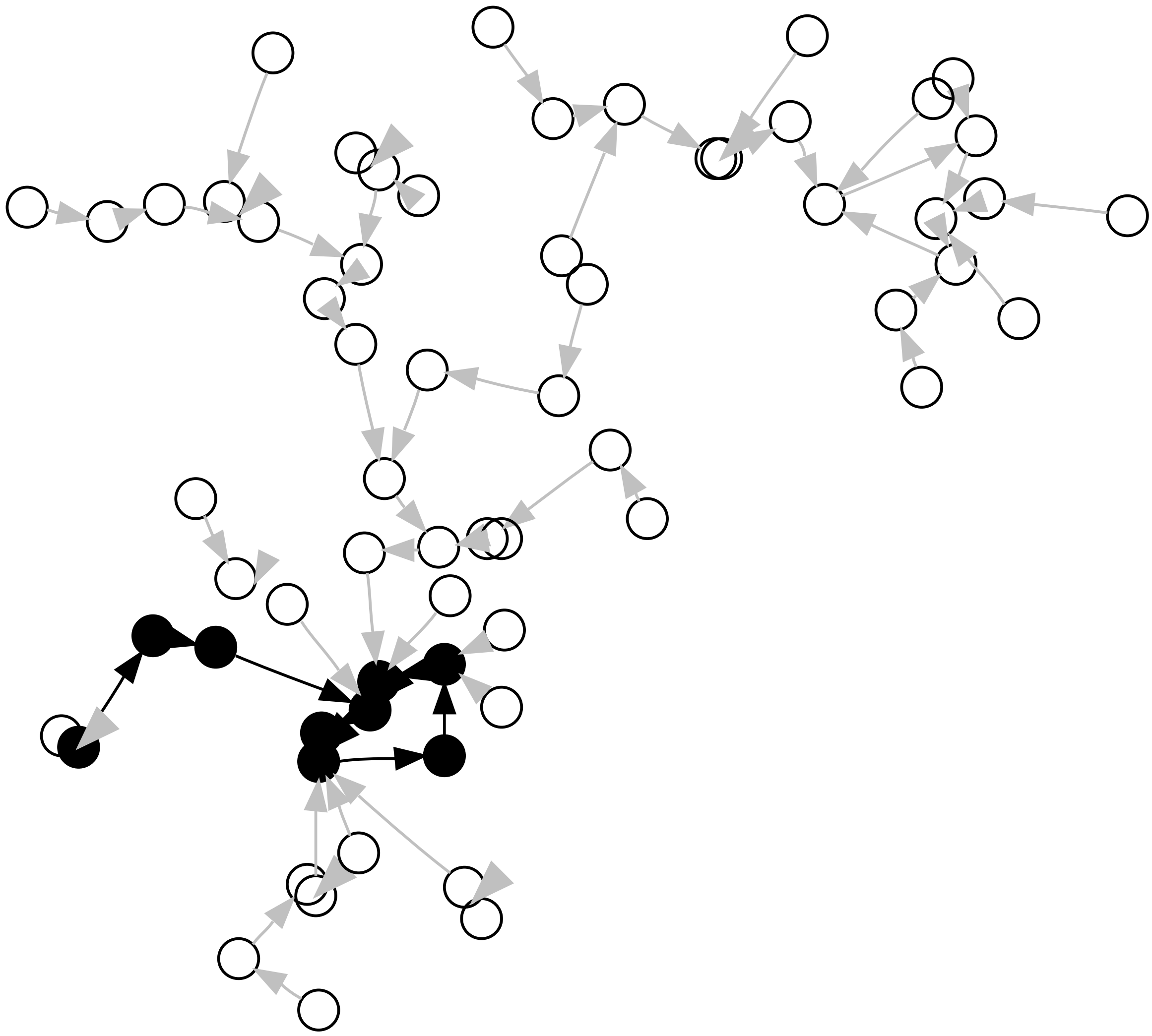


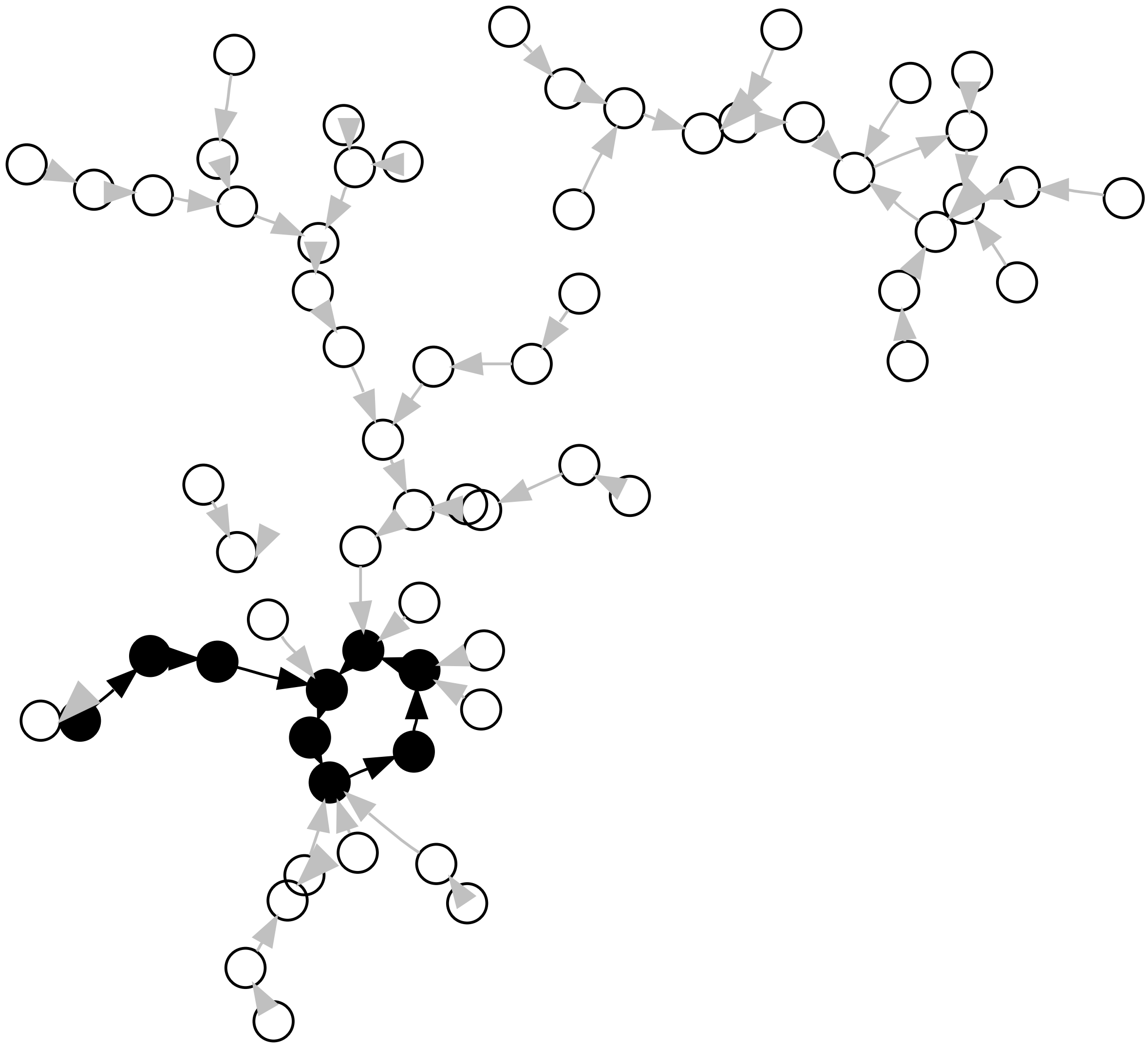


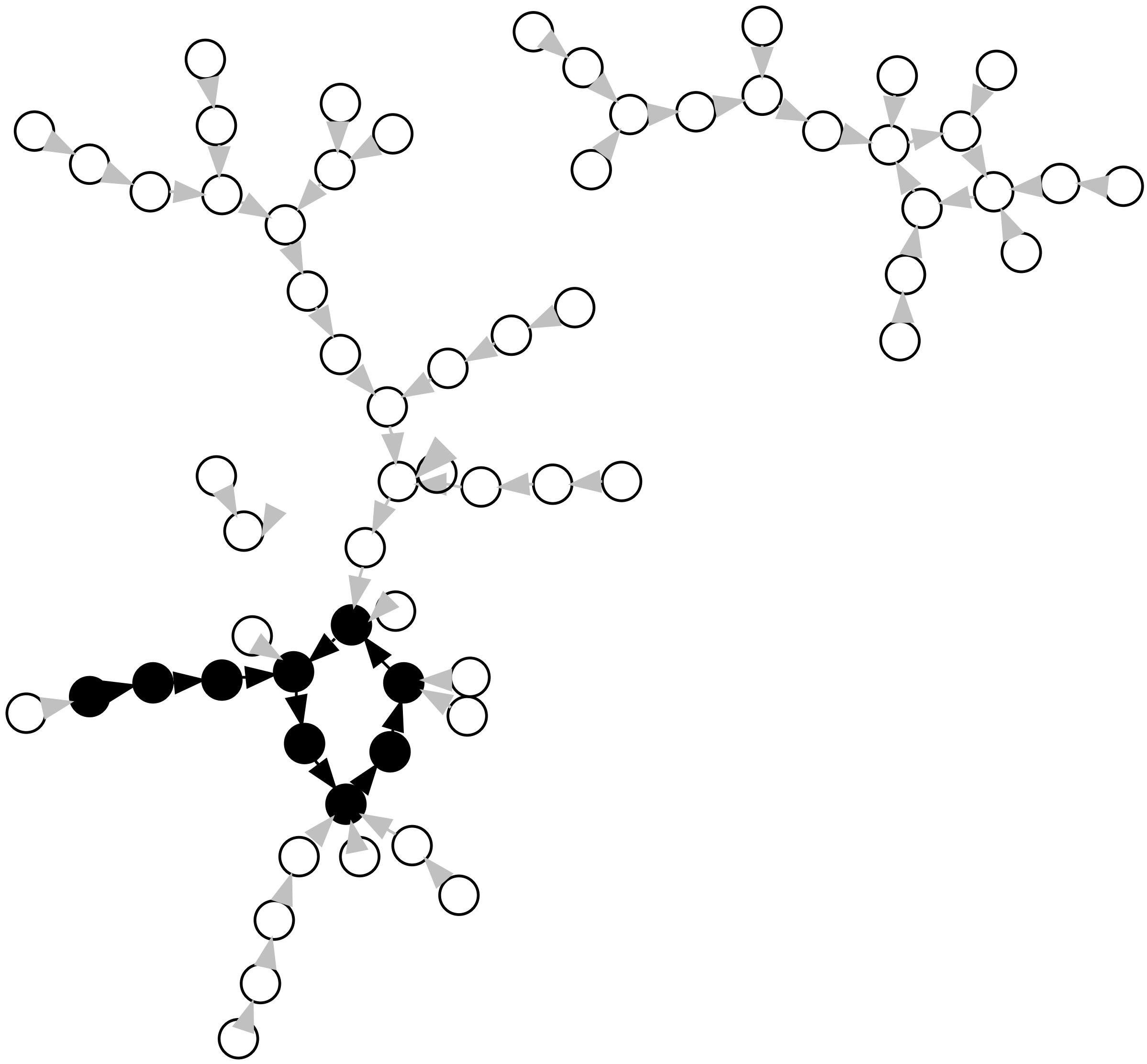












Goal: Compute $\log_P Q$.

Assume that for each i
we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that
$y_i P + x_i Q = y_j P + x_j Q$
so $(y_i - y_j)P = (x_j - x_i)Q$.
If $x_i \neq x_j$ the DLP is solved:
$\log_P Q = (y_j - y_i)/(x_i - x_j)$.

Goal: Compute $\log_P Q$.

Assume that for each i
we know $x_i, y_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $R_i = y_i P + x_i Q$.

Then $R_i = R_j$ means that
$y_i P + x_i Q = y_j P + x_j Q$
so $(y_i - y_j)P = (x_j - x_i)Q$.
If $x_i \neq x_j$ the DLP is solved:
$\log_P Q = (y_j - y_i)/(x_i - x_j)$.

e.g. "base-(P, Q) r-adding walk":
precompute $S_1, S_2, \ldots, S_r$
as random combinations $aP + bQ$;
define $f(R) = R + S_{H(R)}$
where H hashes to $\{1, 2, \ldots, r\}$.

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

Ample experimental evidence that base-(P, Q) 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-(P, Q) 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.

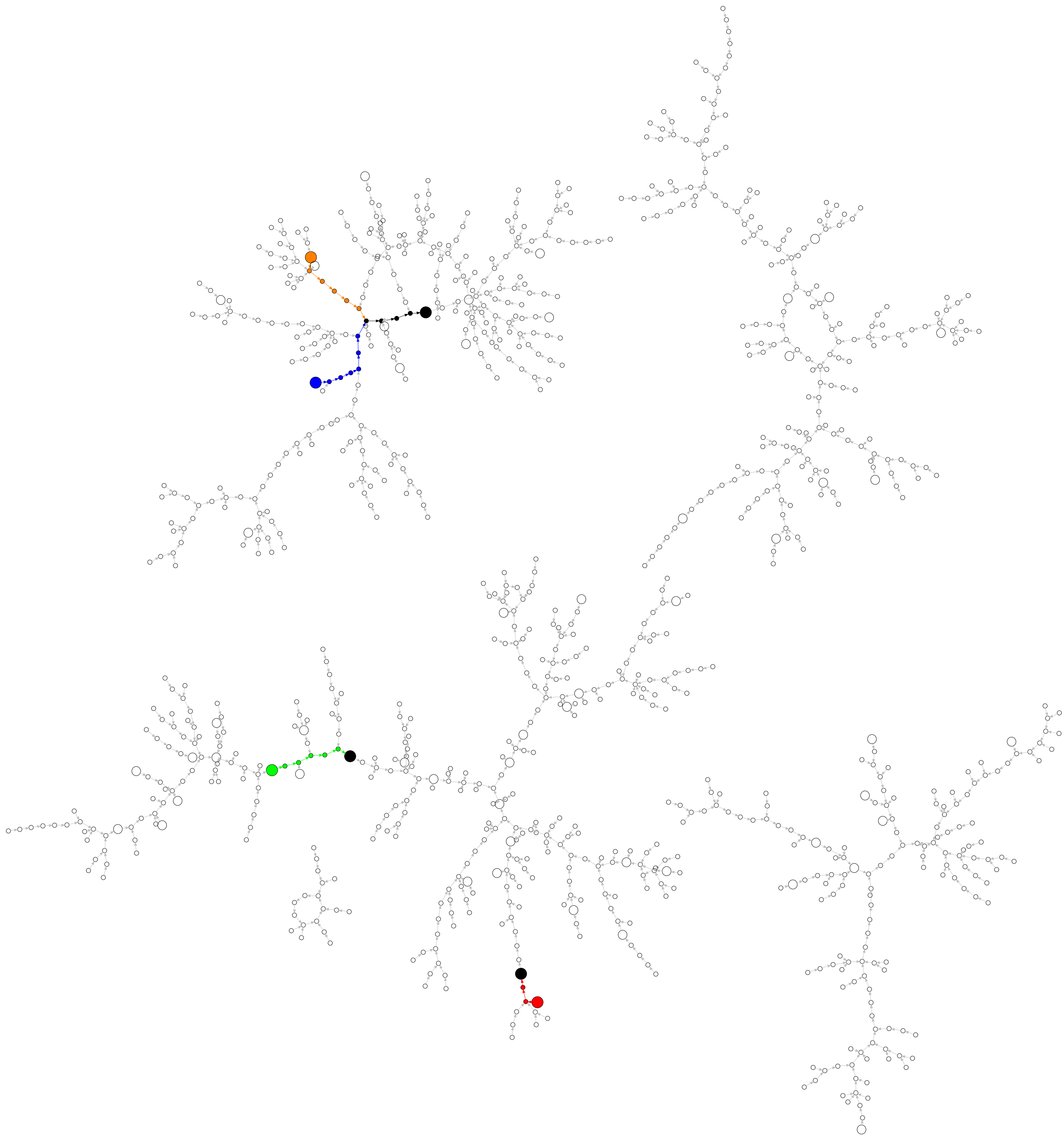
Parallel rho

1994 van Oorschot–Wiener:

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

Perform, in parallel, walks for different starting points $Q+yP$ 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.

State of the art

Can break DLP in group of order ℓ in $\sqrt{\pi\ell/2}$ group operations.

For elliptic curves can gain factor of $\sqrt{2}$ by using negation map.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operations.

State of the art

Can break DLP in group of order ℓ in $\sqrt{\pi\ell/2}$ group operations.

For elliptic curves can gain factor of $\sqrt{2}$ by using negation map.

Solving DLP on NIST P-256 takes $\approx 2^{128}$ group operations.

Let's see what free precomputation does to this . . .

Cube-root ECDL algorithms

Assuming plausible heuristics, overwhelmingly verified by computer experiment:

There exists a P-256 ECDL algorithm that takes "time" $\approx 2^{85}$ and has success probability ≈ 1.

"Time" includes algorithm length.

Inescapable conclusion: **The standard conjectures** (regarding P-256 ECDL hardness, P-256 ECDSA security, etc.) **are false.**

Should P-256 ECDSA users be worried about this P-256 ECDL algorithm A?

No!

We have a program B that prints out A, but B takes "time" $\approx 2^{170}$.

We conjecture that nobody will ever print out A.

Should P-256 ECDSA users be worried about this P-256 ECDL algorithm A?

No!

We have a program B that prints out A, but B takes "time" $\approx 2^{170}$.

We conjecture that nobody will ever print out A.

But A *exists*, and the standard conjecture doesn't see the 2^{170}.

Cryptanalysts *do* see the 2^{170}.

Common parlance: We have a 2^{170} "precomputation" (independent of Q) followed by a 2^{85} "main computation".

For cryptanalysts: This costs 2^{170}, much worse than 2^{128}.

For the standard security definitions and conjectures: The main computation costs 2^{85}, much better than 2^{128}.

What the algorithm does

What the algorithm does

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

Computing (e.g.) $\log_P Q_1$, $\log_P Q_2$, $\log_P Q_3$, $\log_P Q_4$, and $\log_P Q_5$ costs only $2.49\times$ more than computing $\log_P Q$.

The basic idea:
compute $\log_P Q_1$ with rho;
compute $\log_P Q_2$ with rho, *reusing* distinguished points produced by Q_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
$\log_P Q_1, \ldots, \log_P Q_{n-1}$ as
precomputatation for main
computation of $\log_P Q_n$.
Analyze tradeoffs between
main-computation time and
precomputation time.

2012 Bernstein–Lange:

(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) Disprove the standard 2^{128} P-256 security conjectures.

(8) Accelerate trapdoor DL etc.

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

Bonus:

(10) Disprove the standard 2^{128} AES, DSA-3072, RSA-3072 security conjectures.

Credit to earlier Lee–Cheon–Hong paper for (2), (6), (8).

Almost standard walk function:
redefine steps S_i
to depend on P only ($S_i = c_i P$),
c_i chosen uniform random.

Almost standard walk function:
redefine steps S_i
to depend on P only ($S_i = c_i P$),
c_i chosen uniform random.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Almost standard walk function: redefine steps S_i to depend on P only ($S_i = c_i P$), c_i chosen uniform random.

Precomputation: Start some walks at yP for random choices of y. Build table of distinct distinguished points D along with $\log_P D$.

Main computation: Starting from Q, walk to distinguished point $Q + yP$. Check for $Q + yP$ in table.

Almost standard walk function:
redefine steps S_i
to depend on P only ($S_i = c_i P$),
c_i chosen uniform random.

Precomputation:
Start some walks at yP
for random choices of y.
Build table of distinct
distinguished points D
along with $\log_P D$.

Main computation:
Starting from Q, walk to
distinguished point $Q + yP$.
Check for $Q + yP$ in table.
(If this fails, rerandomize Q.)

DSA-3072

Assume that DLP subgroup is extended to 384 bits to counter previous attack.

DSA-3072

Assume that DLP subgroup
is extended to 384 bits
to counter previous attack.

The following sketch
is not the state of the art —
but good enough to break
the 2^{128} assumption.

Let $g \in \mathbf{F}_p^*$ have order q, $h = g^k$.
Goal: Find k.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \leq y$.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \le y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \le y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

and then try to factor h_1, h_2
into primes $\le y$.

Precomputation:
Take $y = 2^{110}$,
compute $\log_g x^{(p-1)/q}$
for every prime number $x \le y$.

Main computation:
Try to write h as
quotient h_1/h_2 in $\mathbf{F}_p^*$
with $h_2 \in \{1, 2, 3, \ldots, 2^{1535}\}$,
$h_1 \in \{-2^{1535}, \ldots, 0, 1, \ldots, 2^{1535}\}$,
and $\gcd\{h_1, h_2\} = 1$;

and then try to factor h_1, h_2
into primes $\le y$.

If this fails, try again
with hg, hg^2, etc.

Analysis

About $y/\log y \approx 2^{103.75}$ primes $\le y$ for a total of $2^{109.33}$ bytes to store all small DLs.

Can write h as h_1/h_2 with probability $\approx (6/\pi^2) 2^{3071}/p$.

h_i is y-smooth with probability very close to $u^{-u} \approx 2^{-53.06}$ where $u = 1535/110$.

Overall the attack requires between $2^{107.85}$ and $2^{108.85}$ iterations; batch smoothness detection is fast.

Possible responses

Possible responses

(1) Accept 2^{85} etc. as security; live with it. Protect the proofs!

Possible responses

(1) Accept 2^{85} etc. as security;
live with it. Protect the proofs!

(2) Switch to NAND metric; or
(3) switch to AT metric.
Breaks most theorems;
still bogus results in NAND.

<u>Possible responses</u>

(1) Accept 2^{85} etc. as security; live with it. Protect the proofs!

(2) Switch to NAND metric; or
(3) switch to AT metric.
Breaks most theorems;
still bogus results in NAND.

(4) Add effectivity. Include cost for finding the algorithm.

Possible responses

(1) Accept 2^{85} etc. as security; live with it. Protect the proofs!

(2) Switch to NAND metric; or

(3) switch to AT metric.

Breaks most theorems; still bogus results in NAND.

(4) Add effectivity. Include cost for finding the algorithm.

(5) Add uniformity. Clearly stops attacks but breaks most theorems.

Abandons goal of defining concrete security of AES etc.