On the correct use of the negation map in the Pollard rho method

D. J. Bernstein
University of Illinois at Chicago

Tanja Lange
Technische Universiteit Eindhoven

Joint work with:
Peter Schwabe
Academia Sinica

Full version of paper with entertaining historical details:
eprint.iacr.org/2011/003

The rho method

Group $\langle P \rangle$ of prime order ℓ.
Discrete-log problem for $\langle P \rangle$:
given P, kP, find $k \bmod \ell$.

Standard attack: parallel rho.

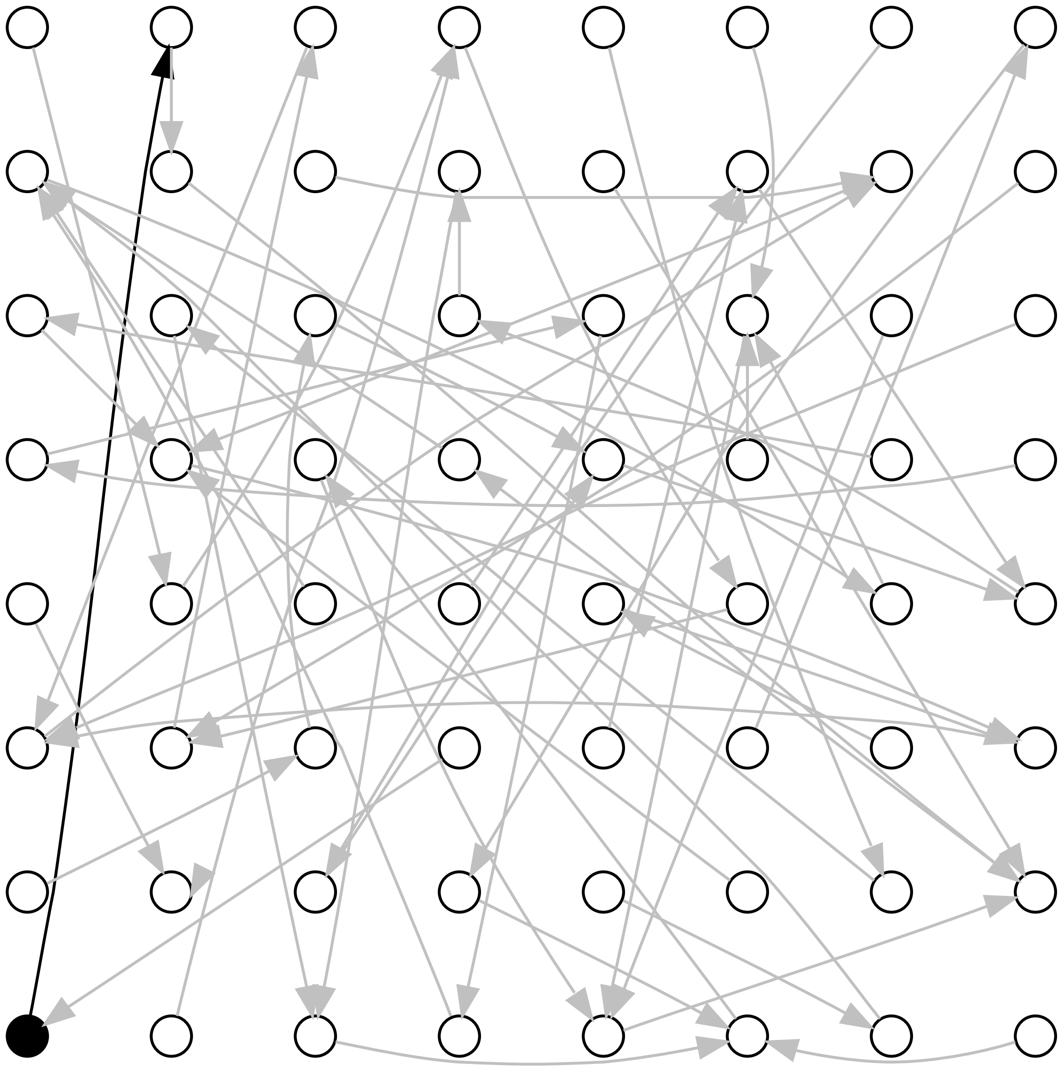
Expect $(1 + o(1))\sqrt{\pi\ell/2}$
group operations,
matching Nechaev/Shoup bound.
Easy to distribute across CPUs.
Very little memory consumption.
Very little communication.

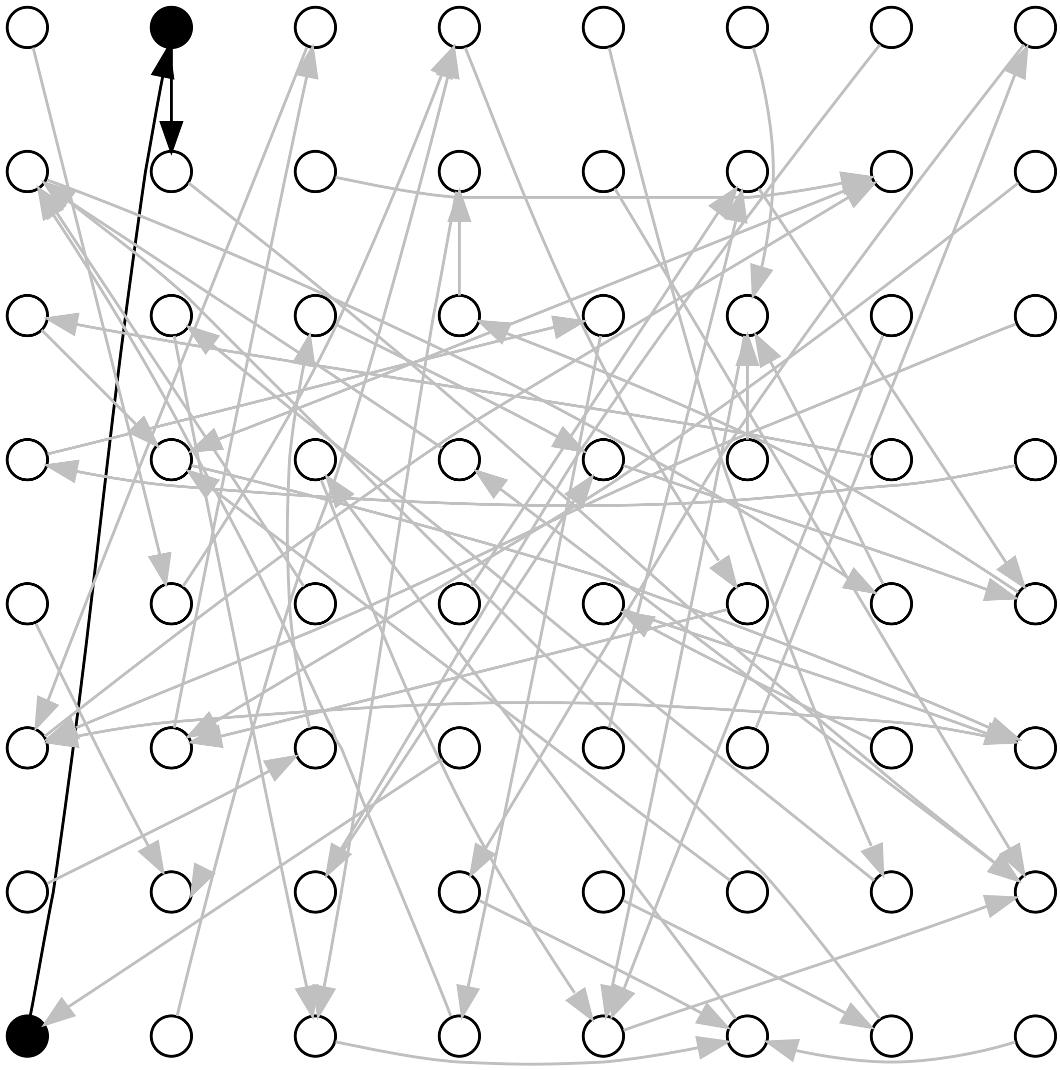
Simplified, non-parallel rho:

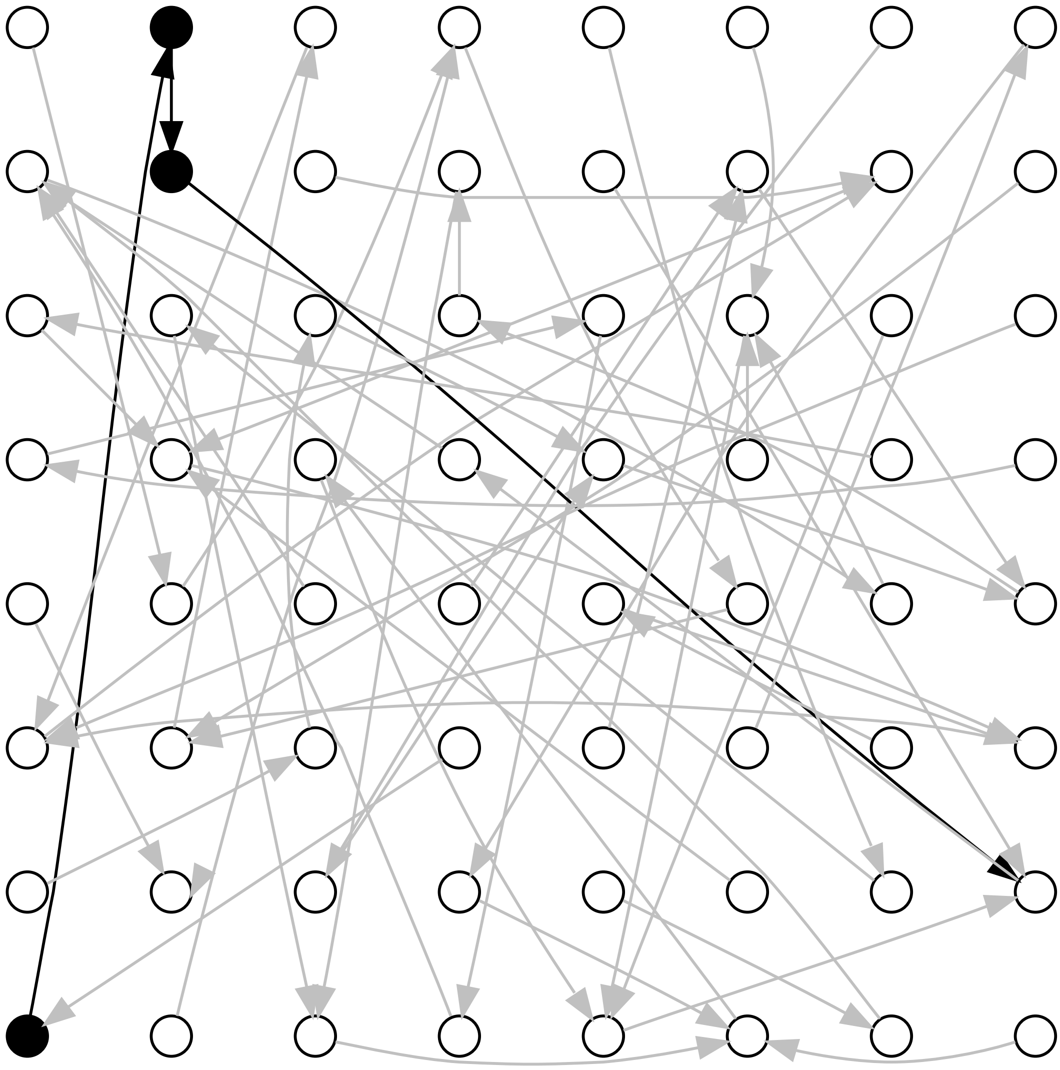
Make a pseudo-random walk in the group $\langle P \rangle$, where the next step depends on current point: $W_{i+1} = f(W_i)$.

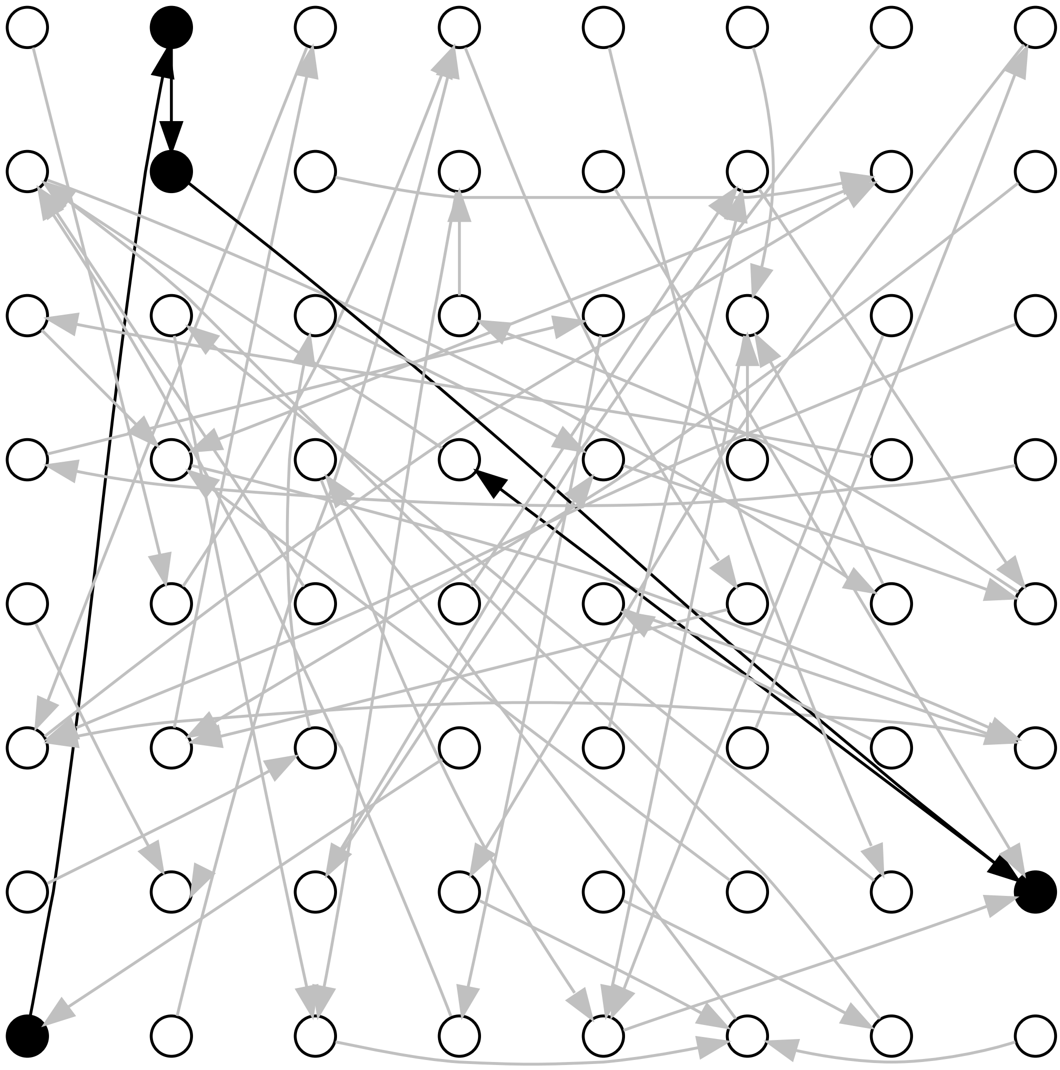
Birthday paradox: Randomly choosing from ℓ elements picks one element twice after about $\sqrt{\pi \ell / 2}$ draws.

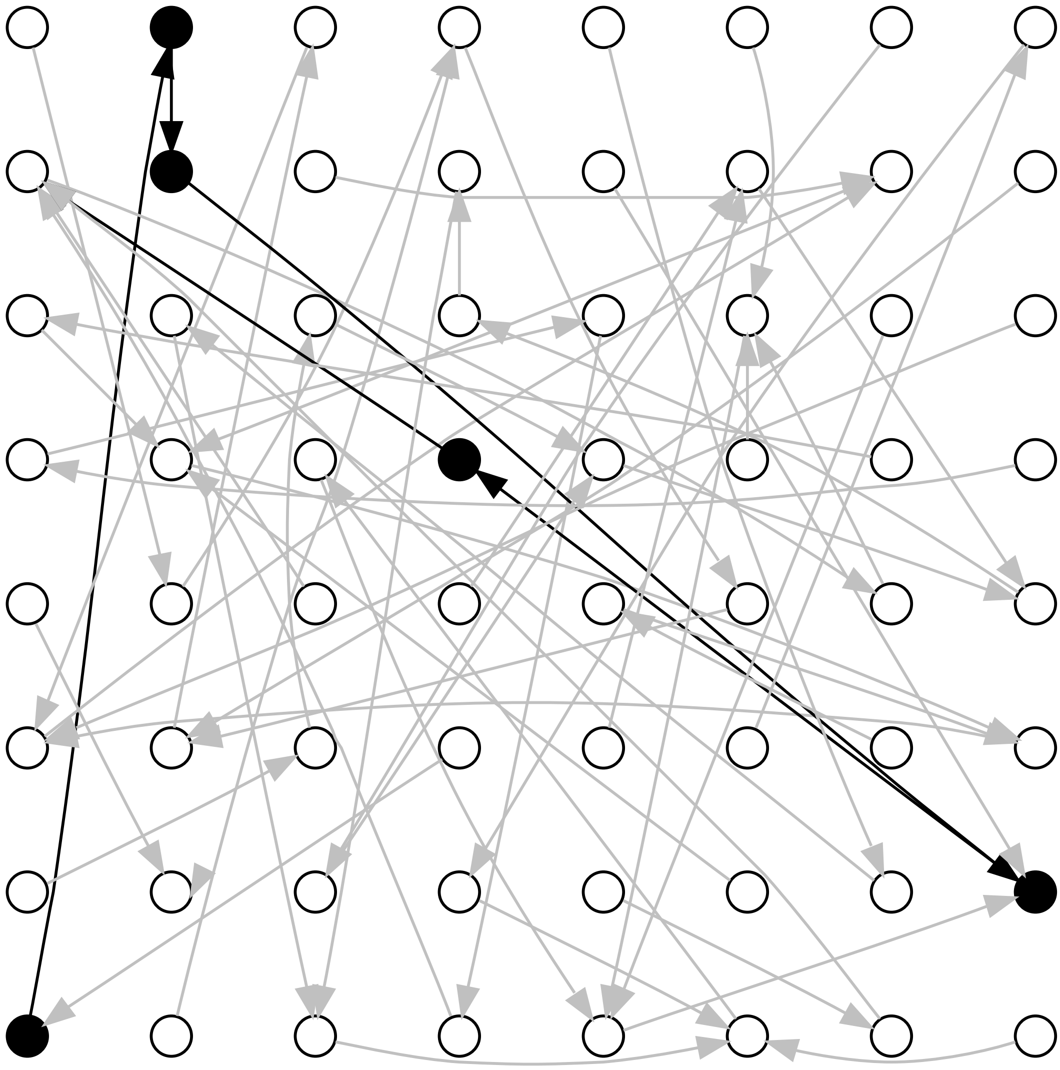
The walk now enters a cycle. Cycle-finding algorithm (e.g., Floyd) quickly detects this.

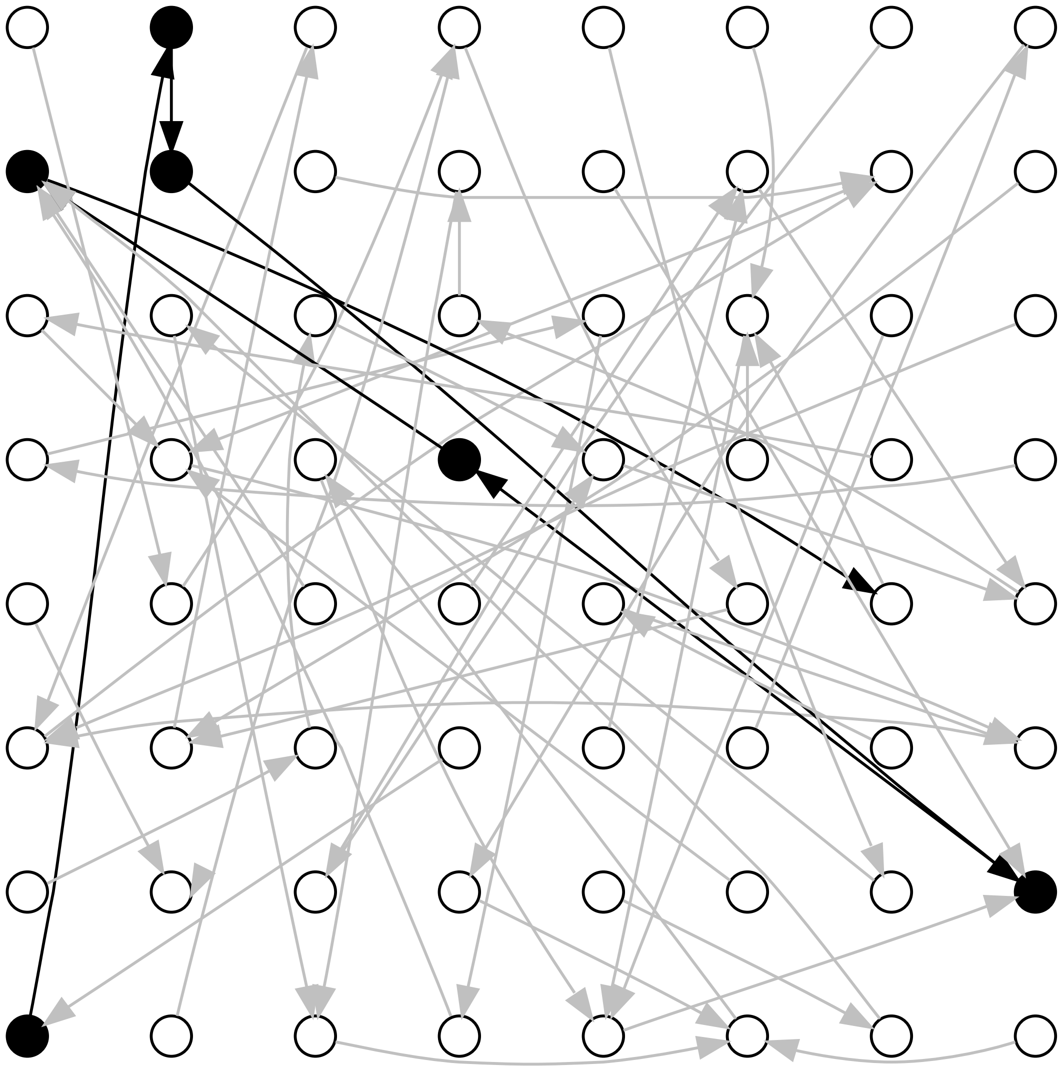


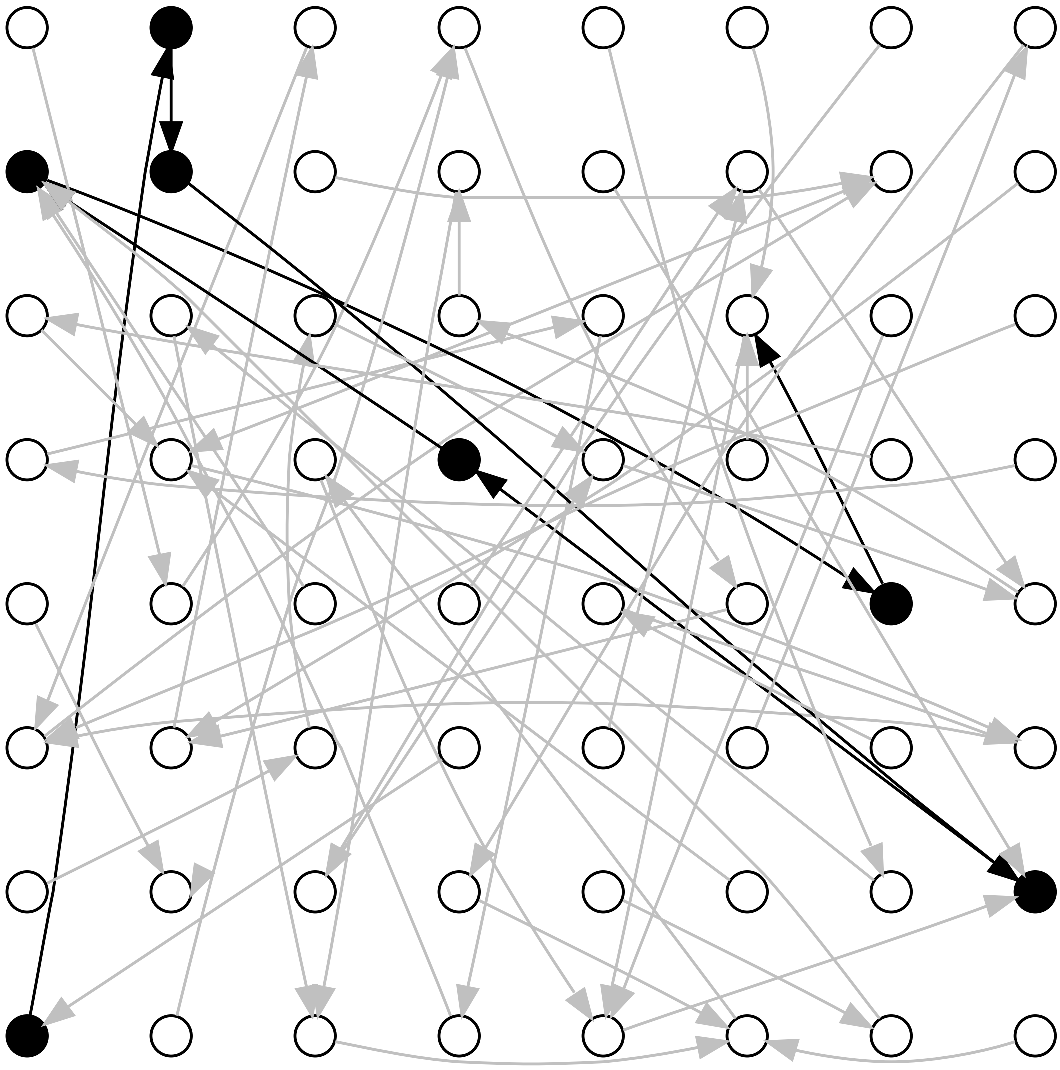


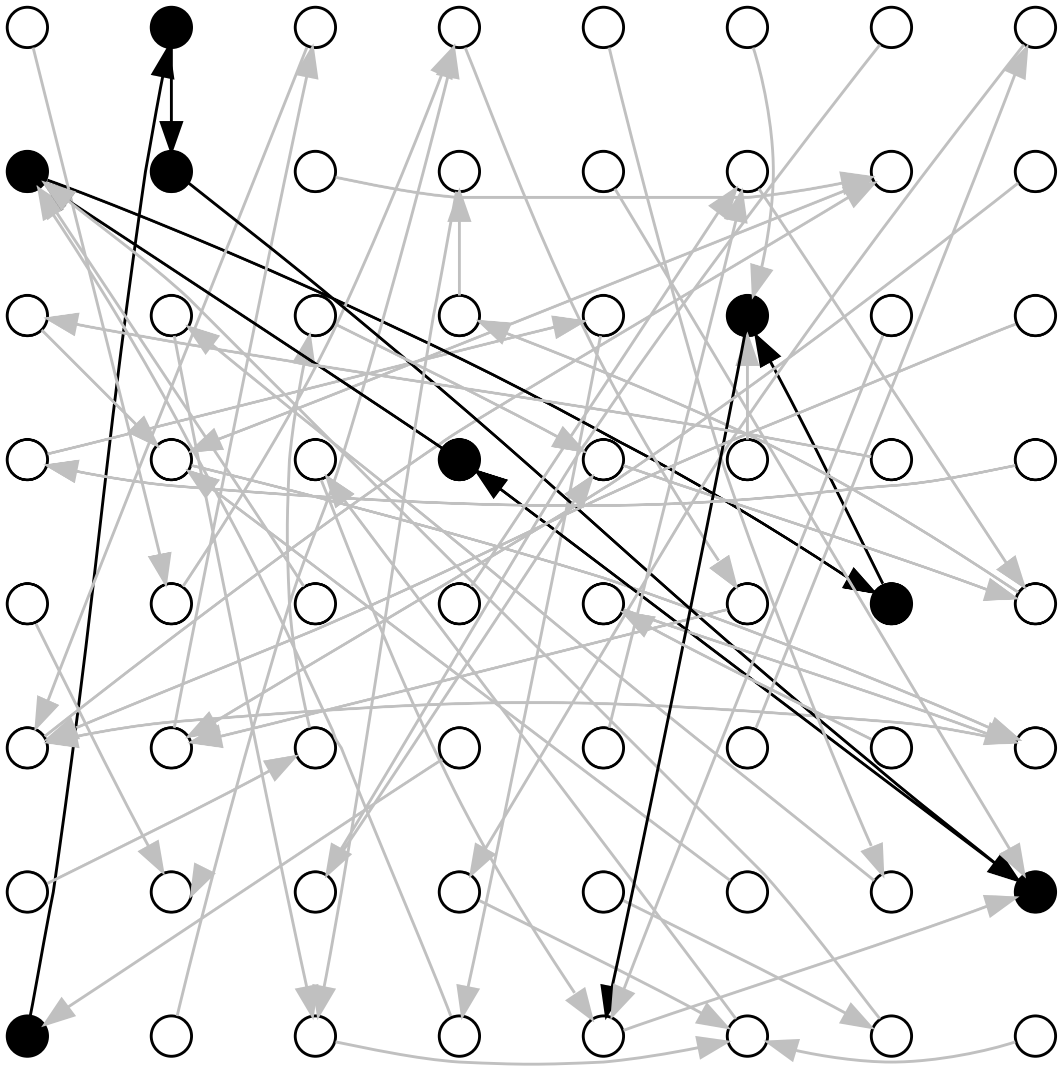


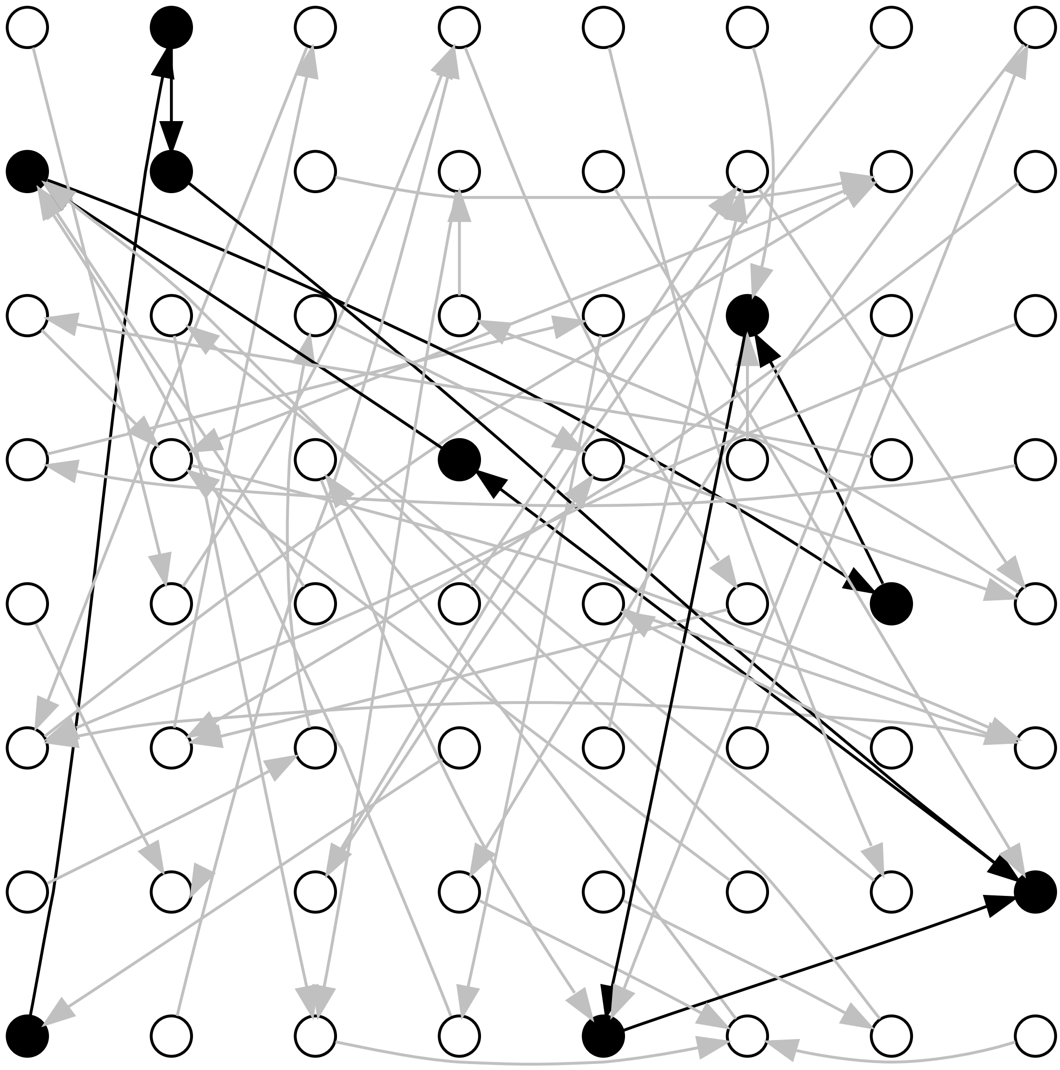


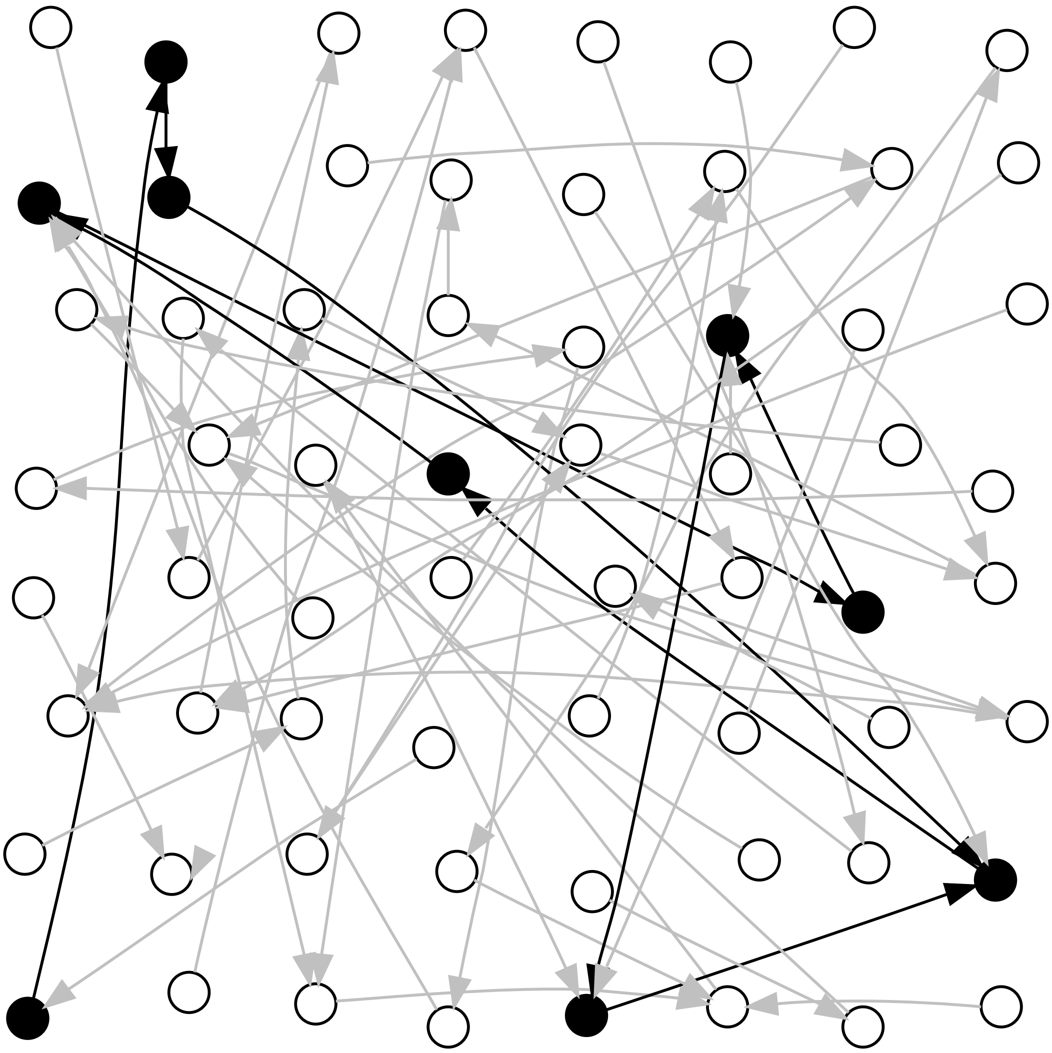


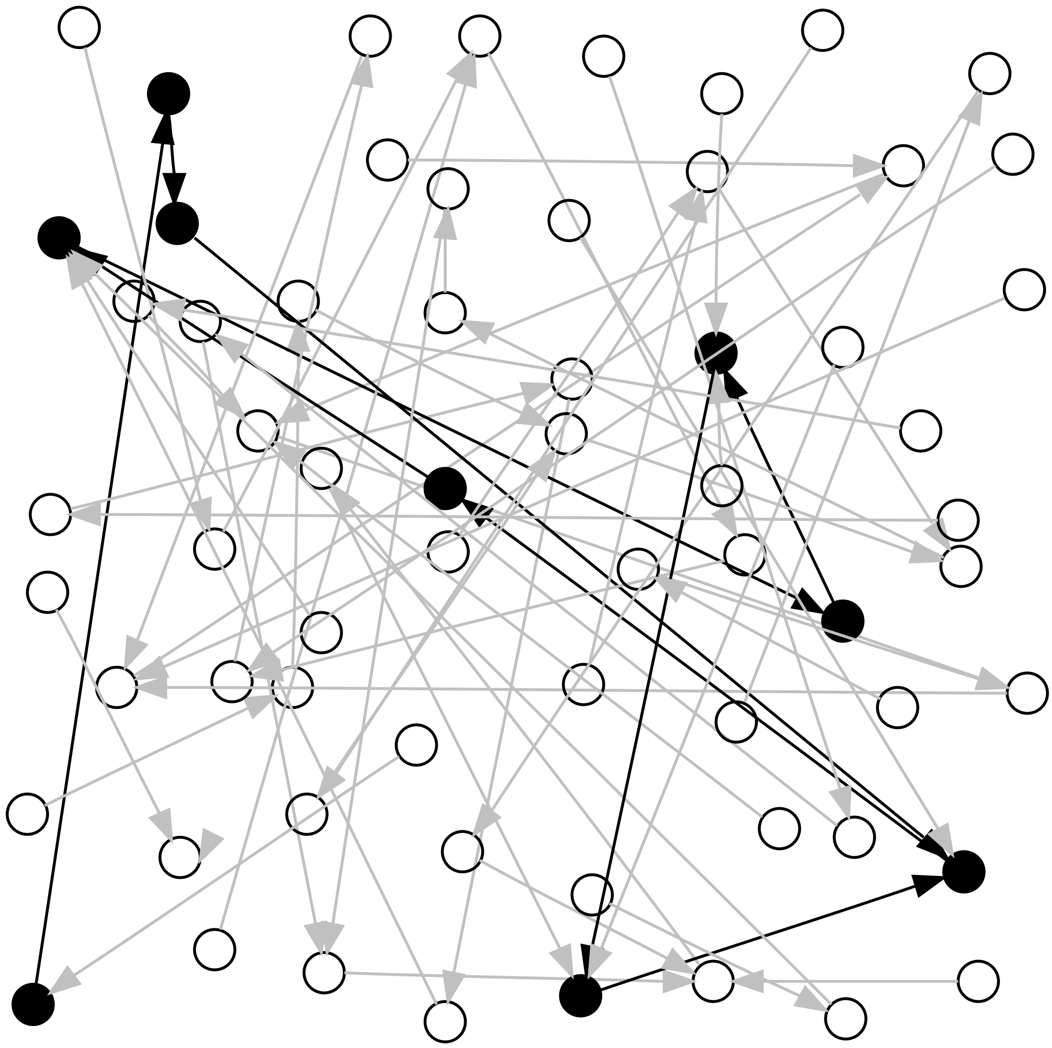


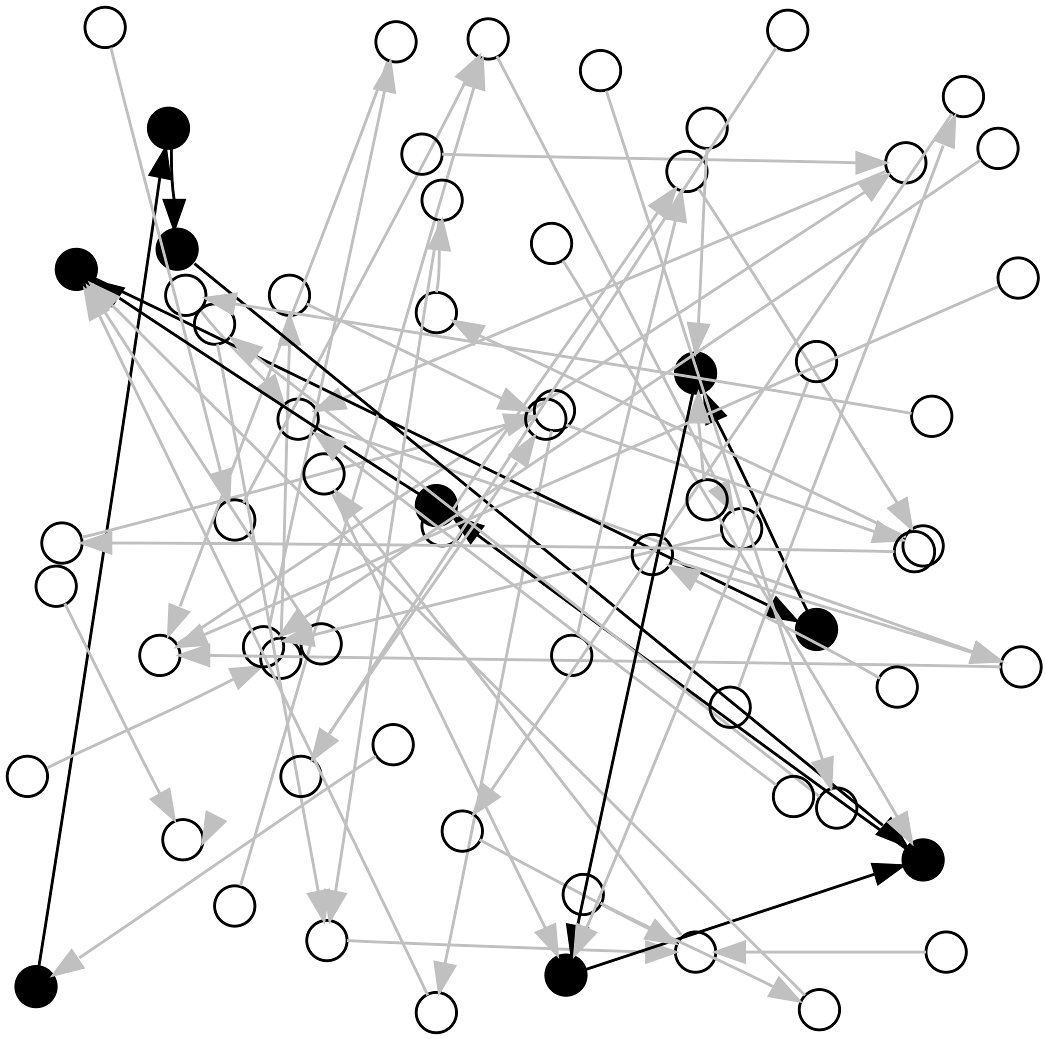


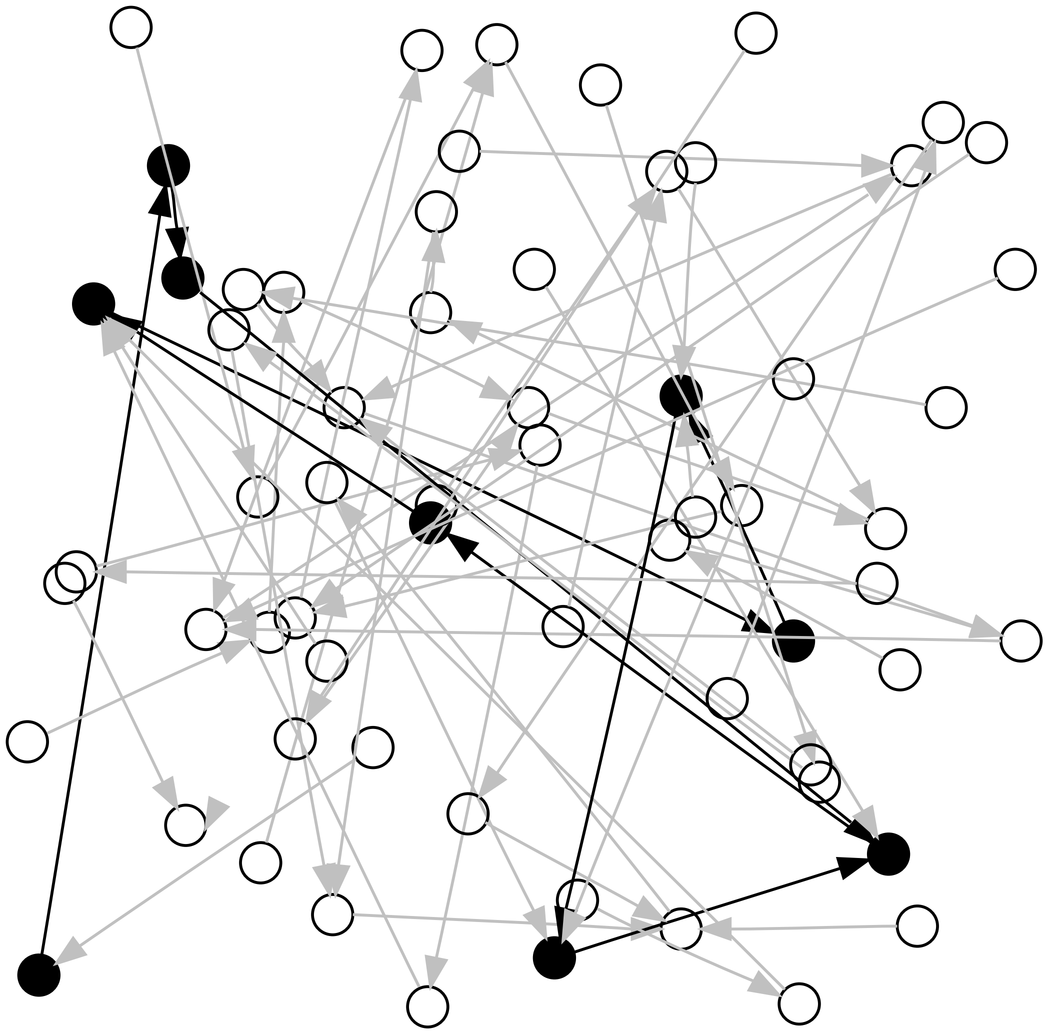


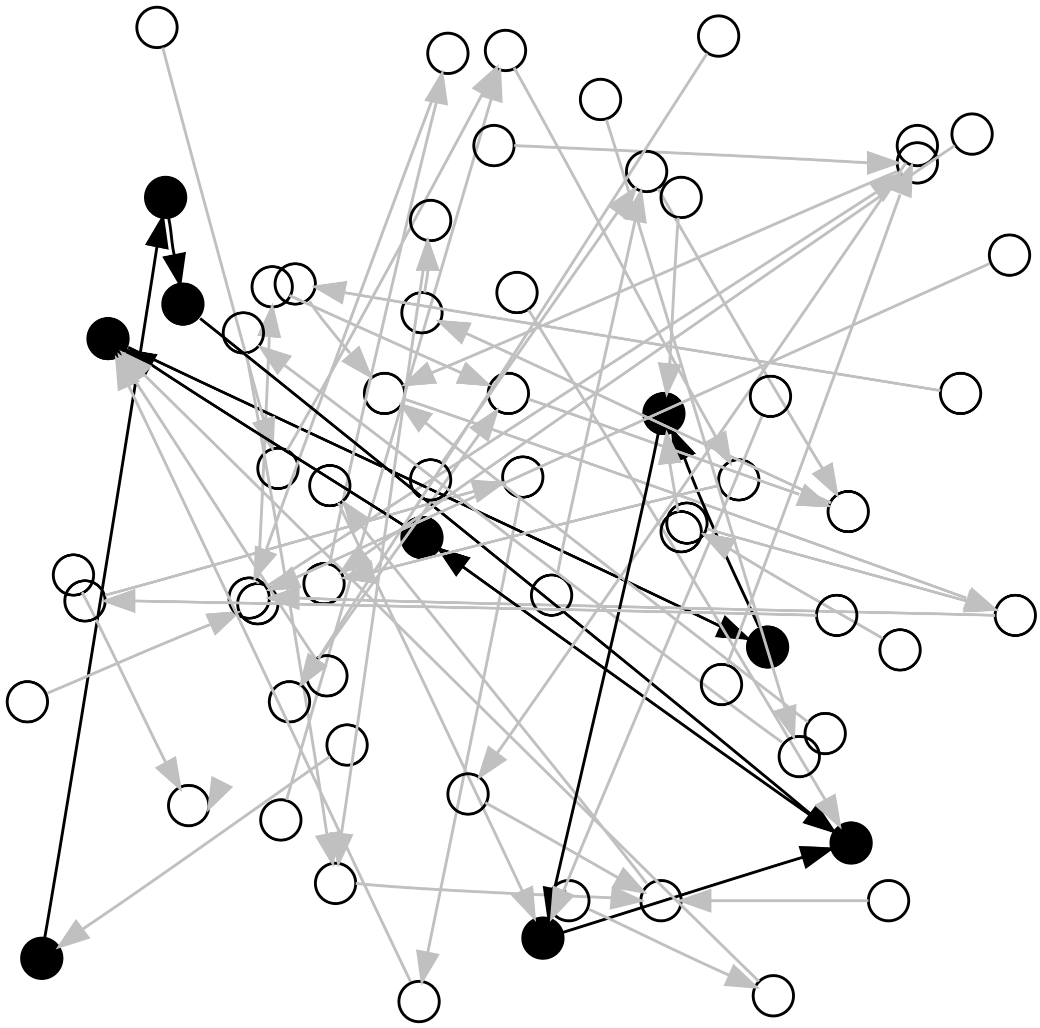


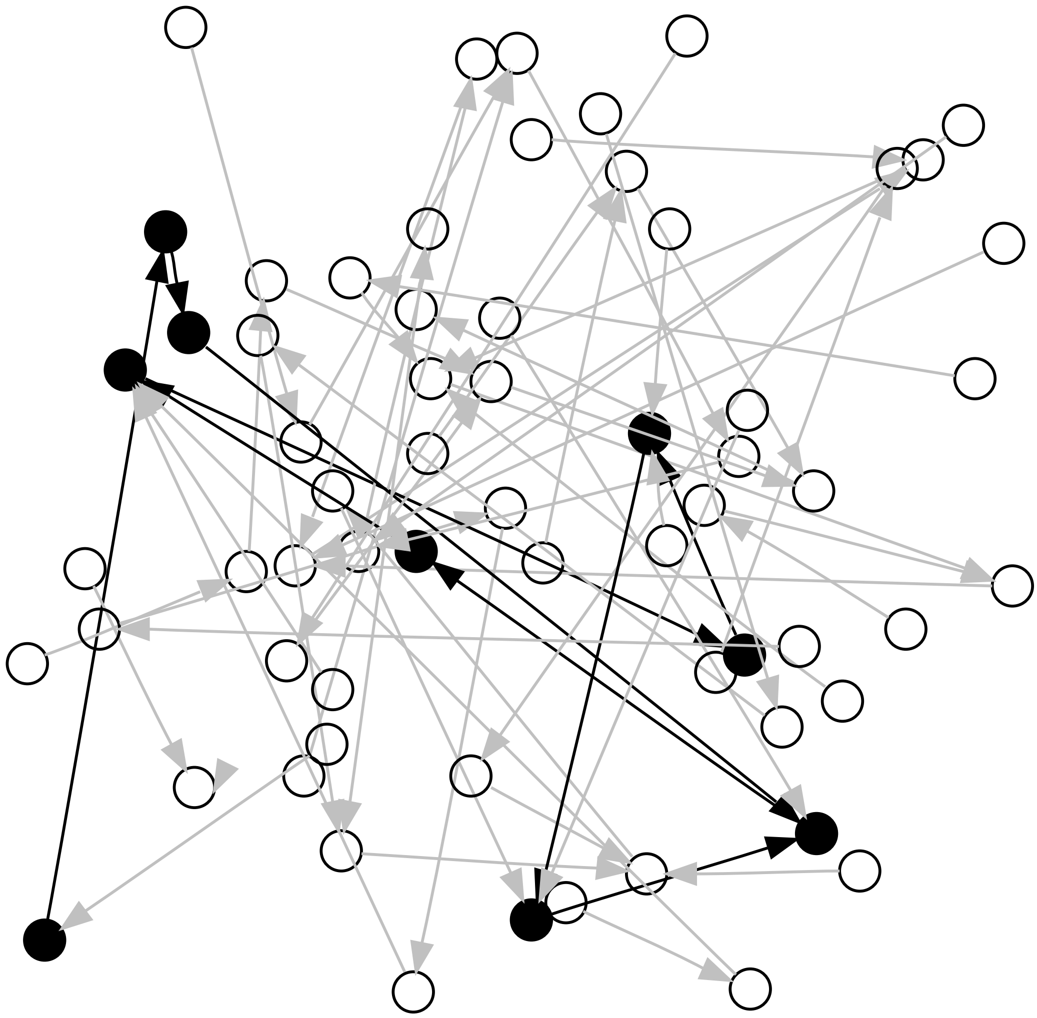


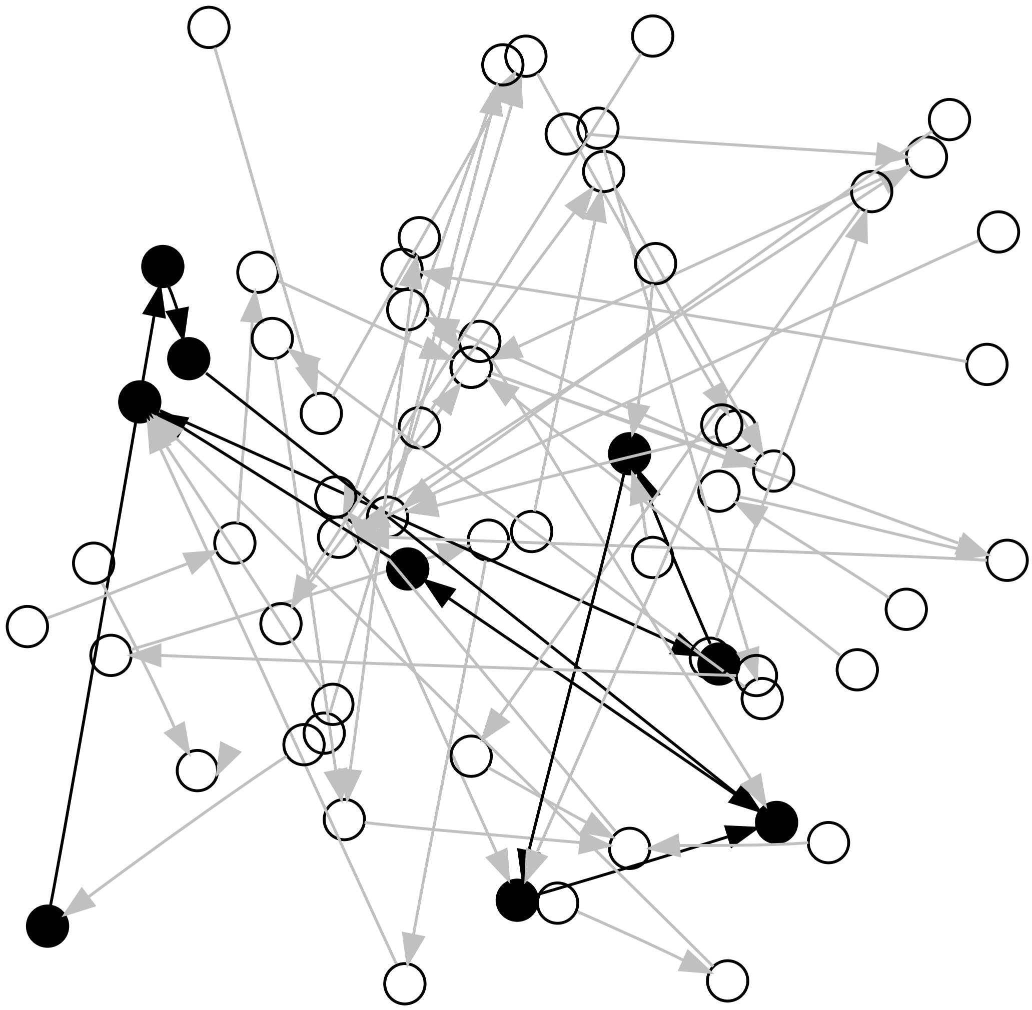


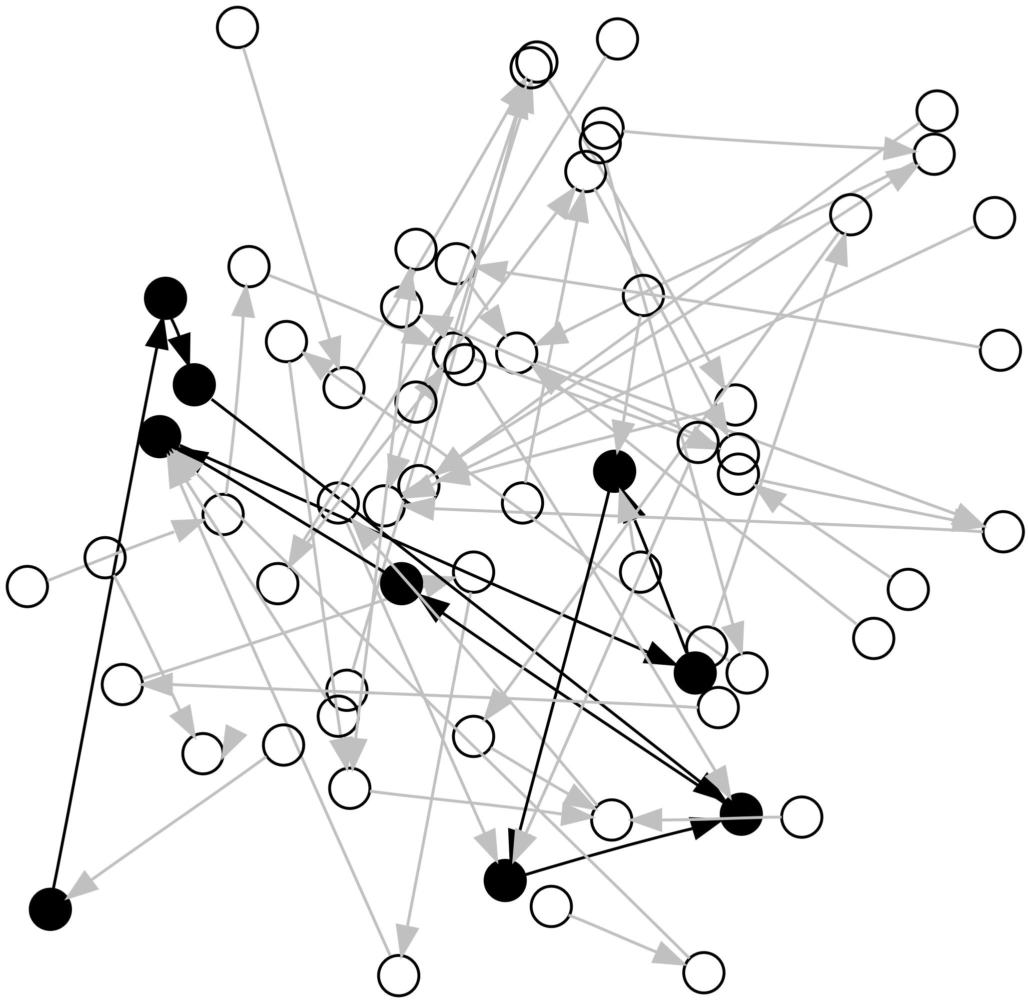


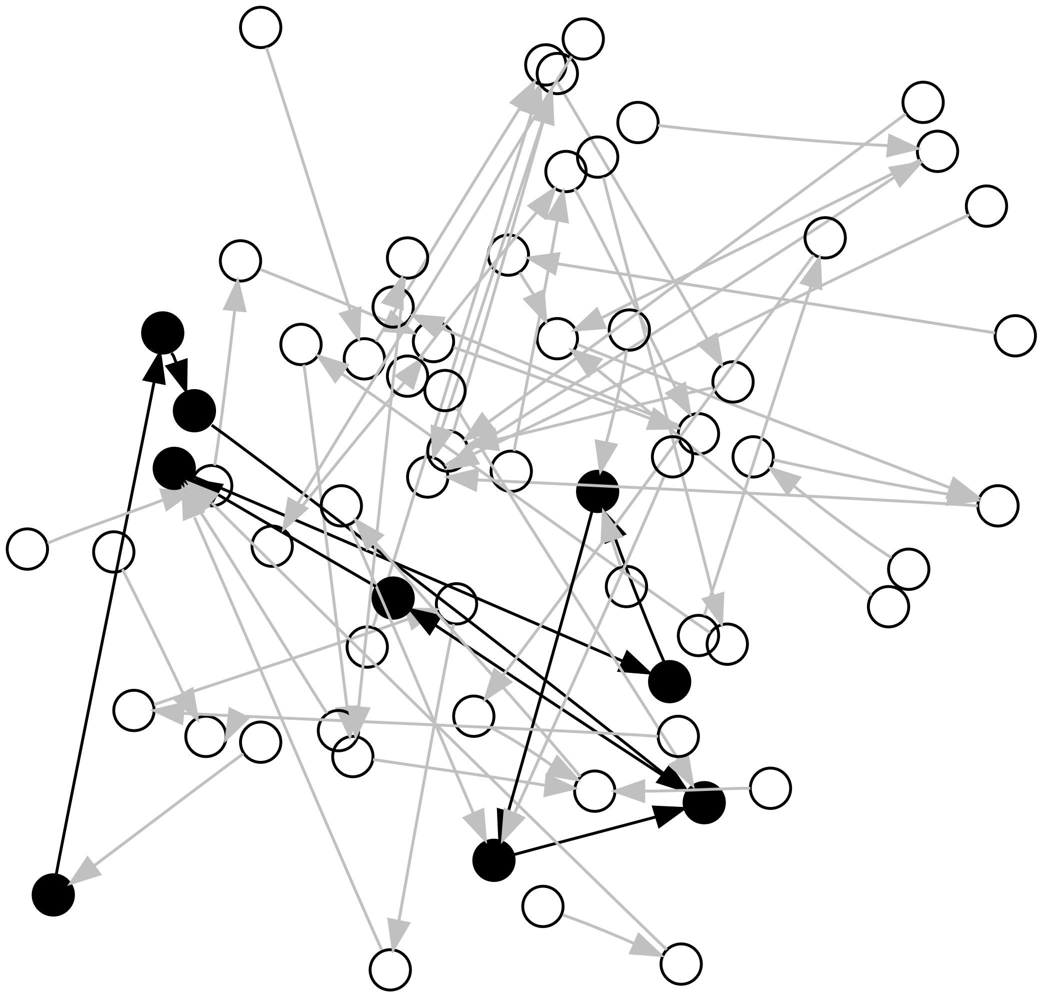


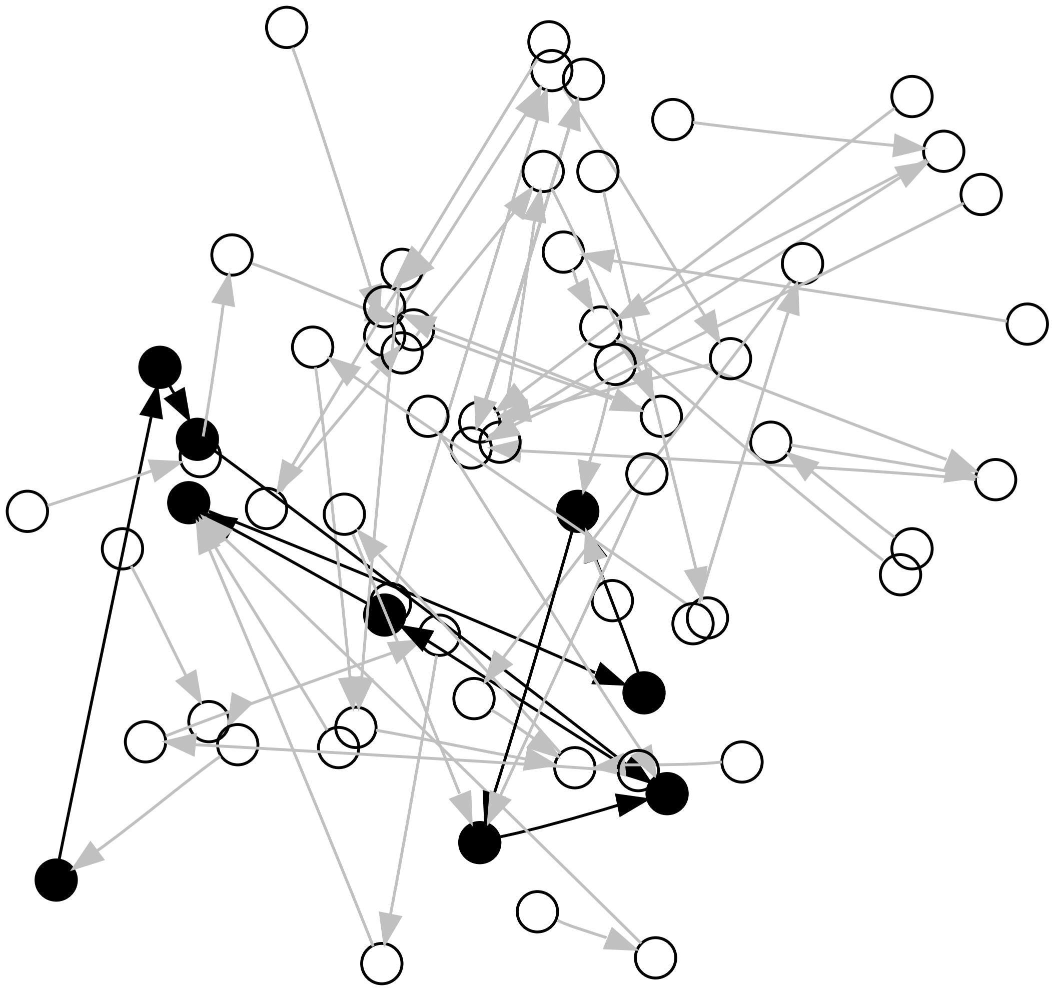


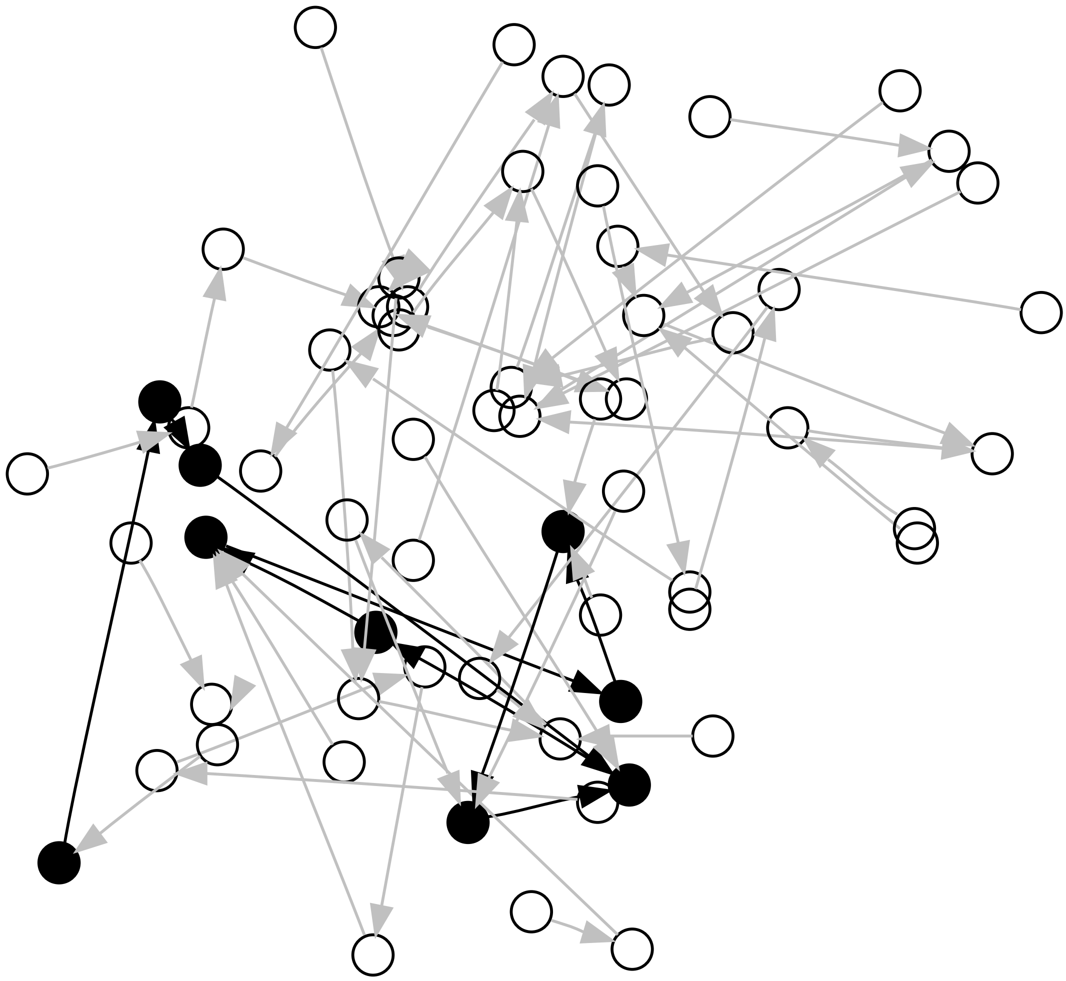


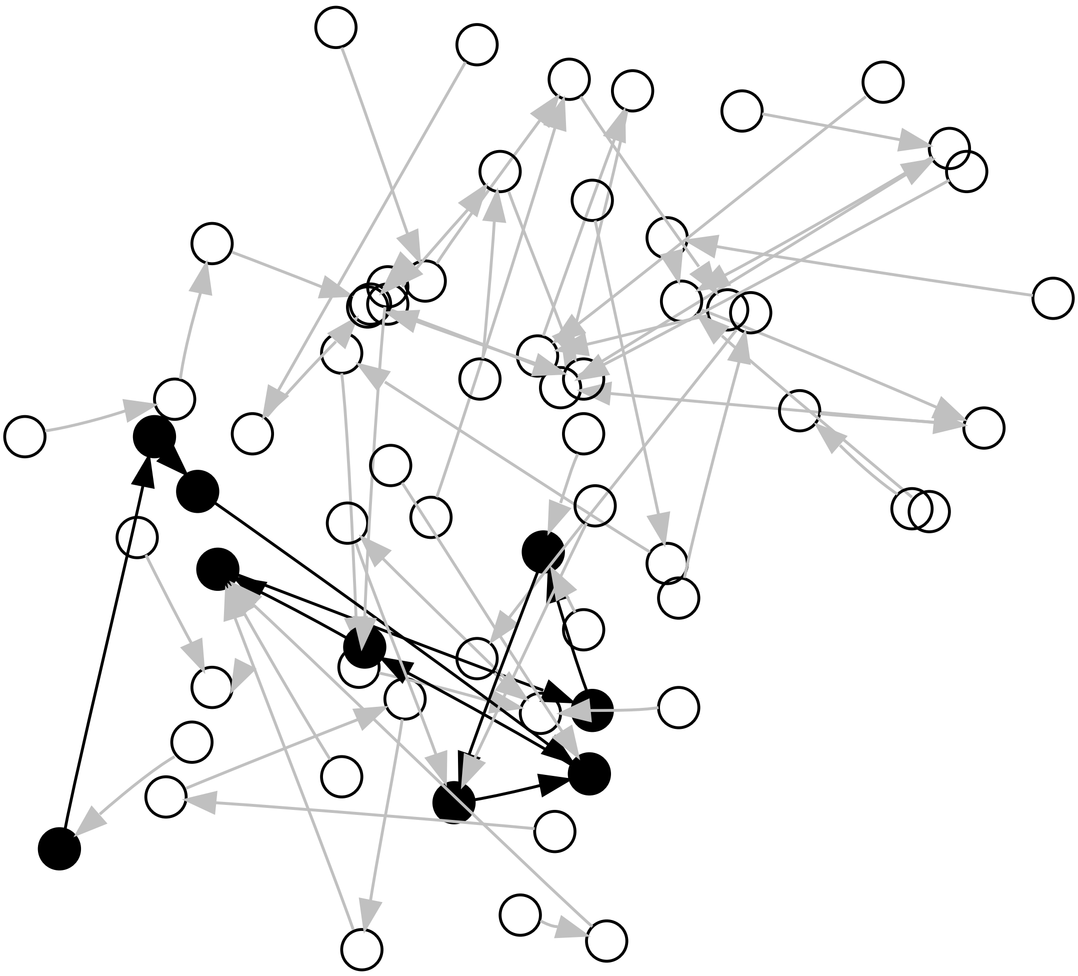


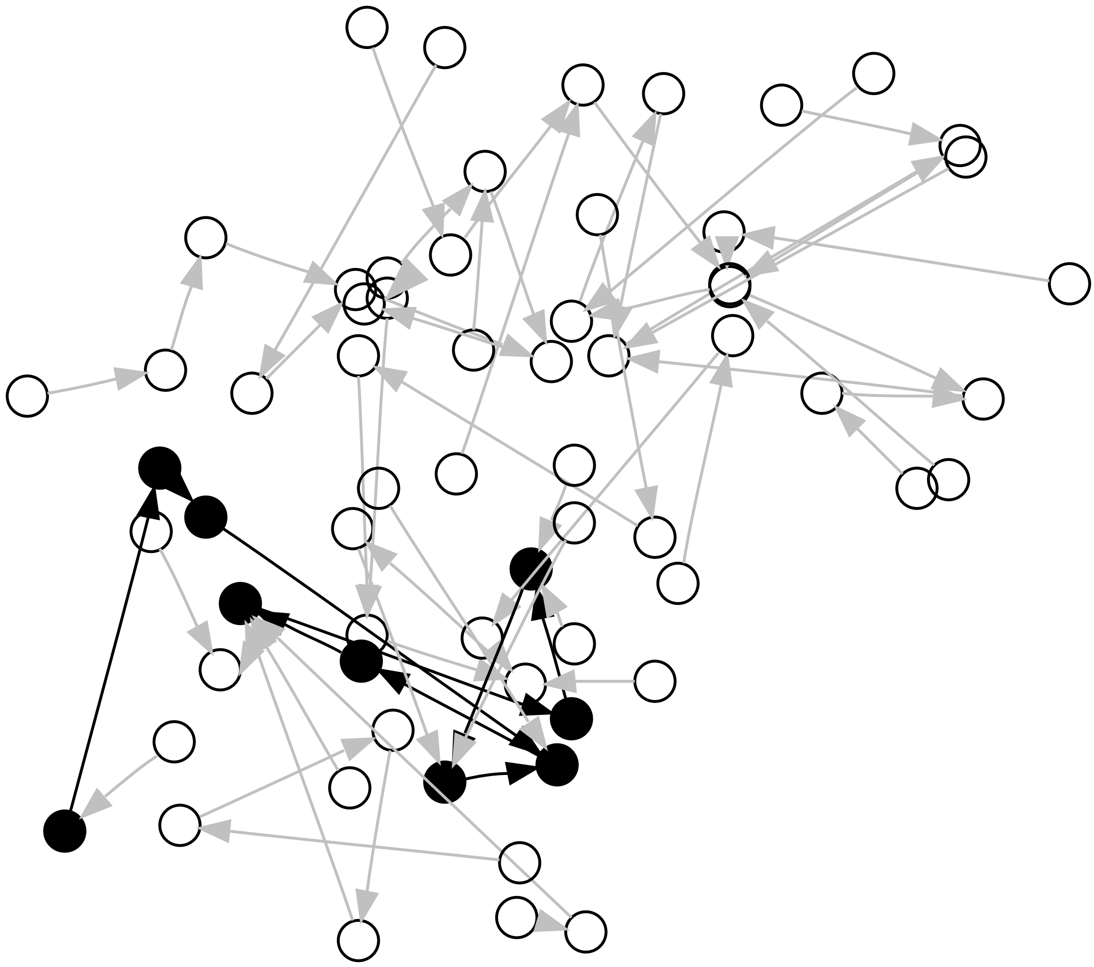


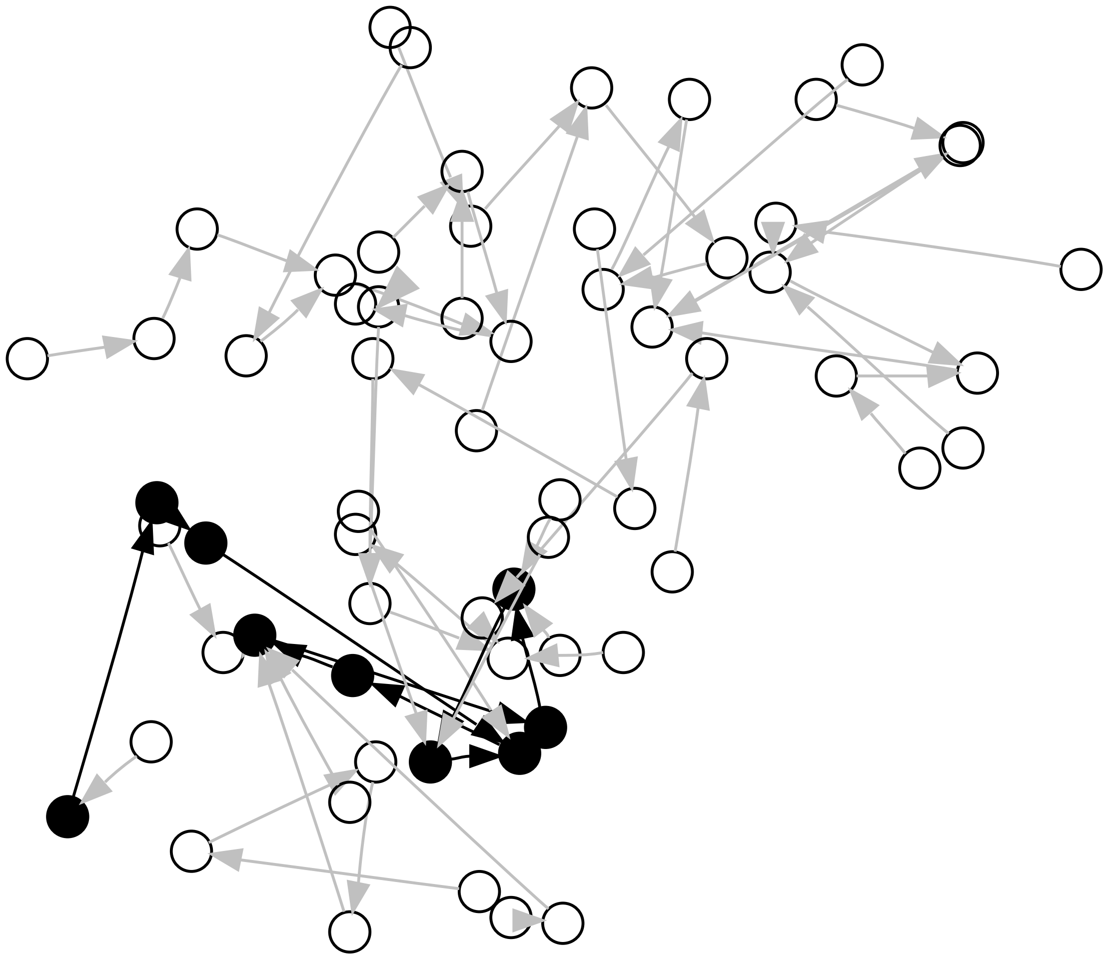


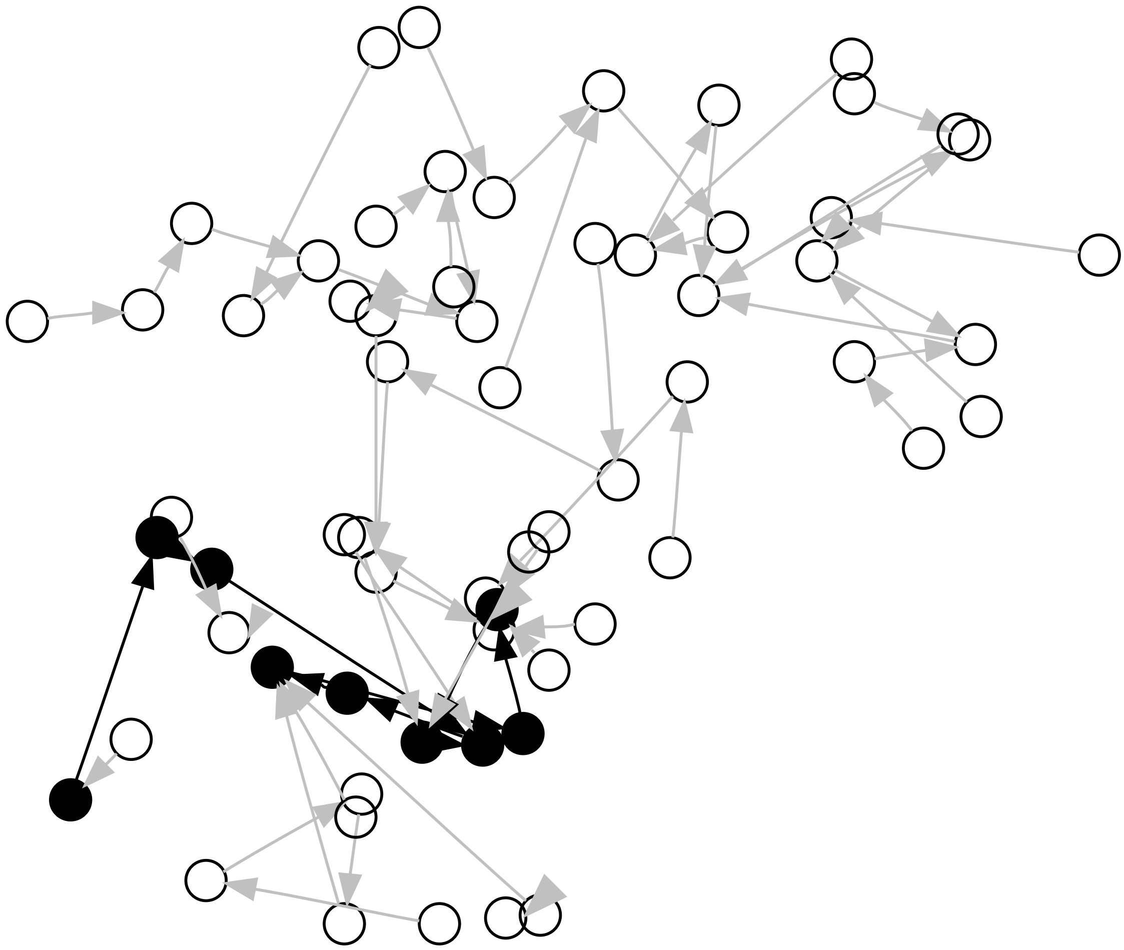


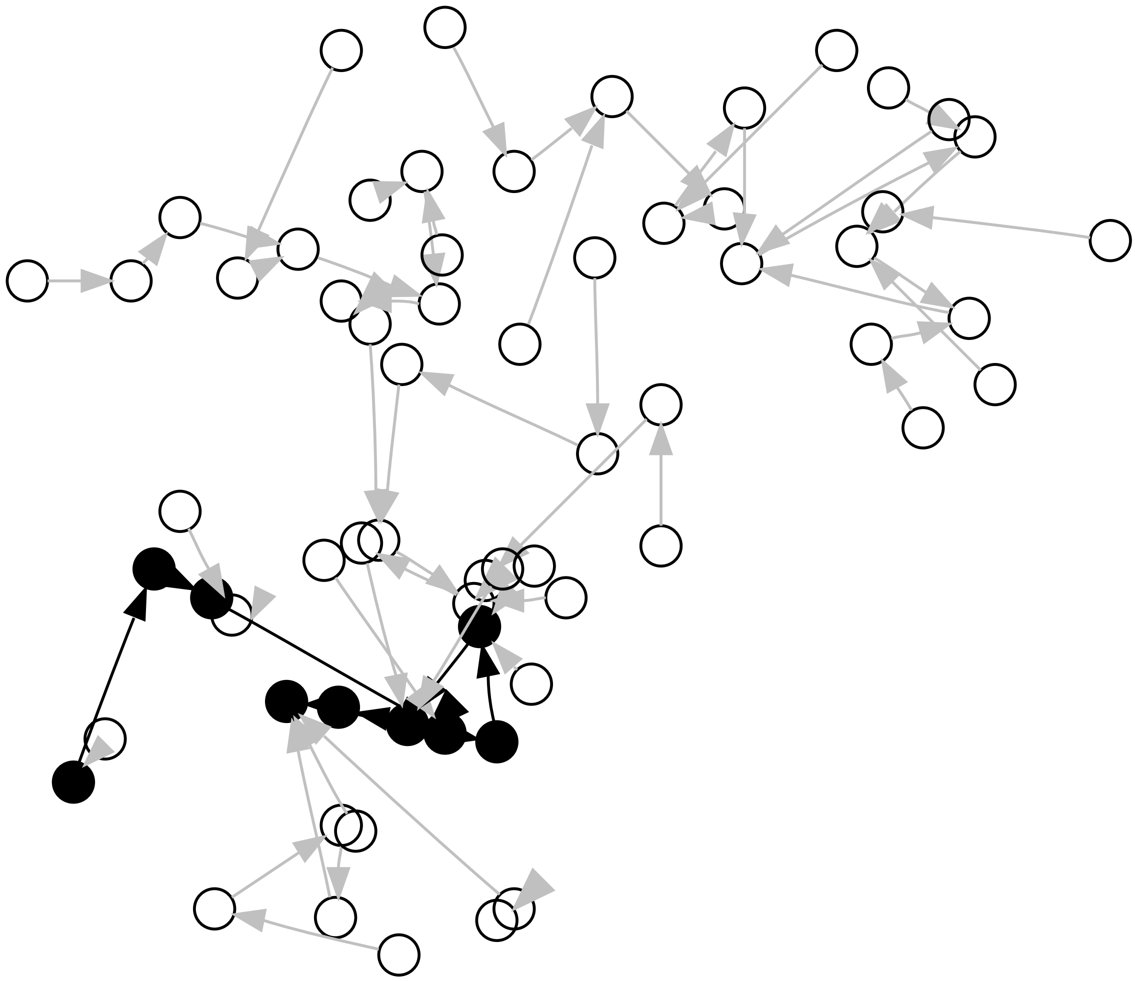


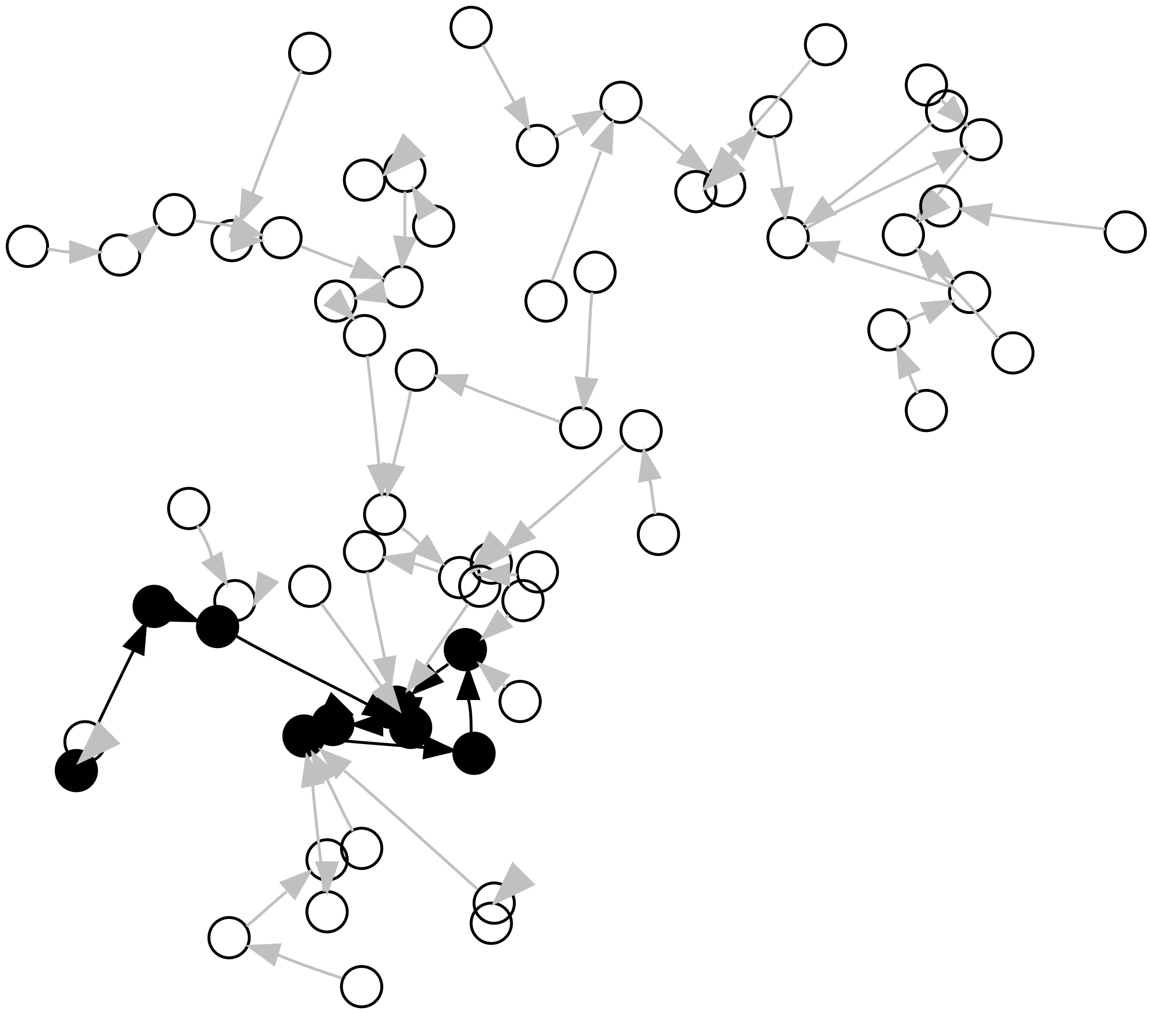


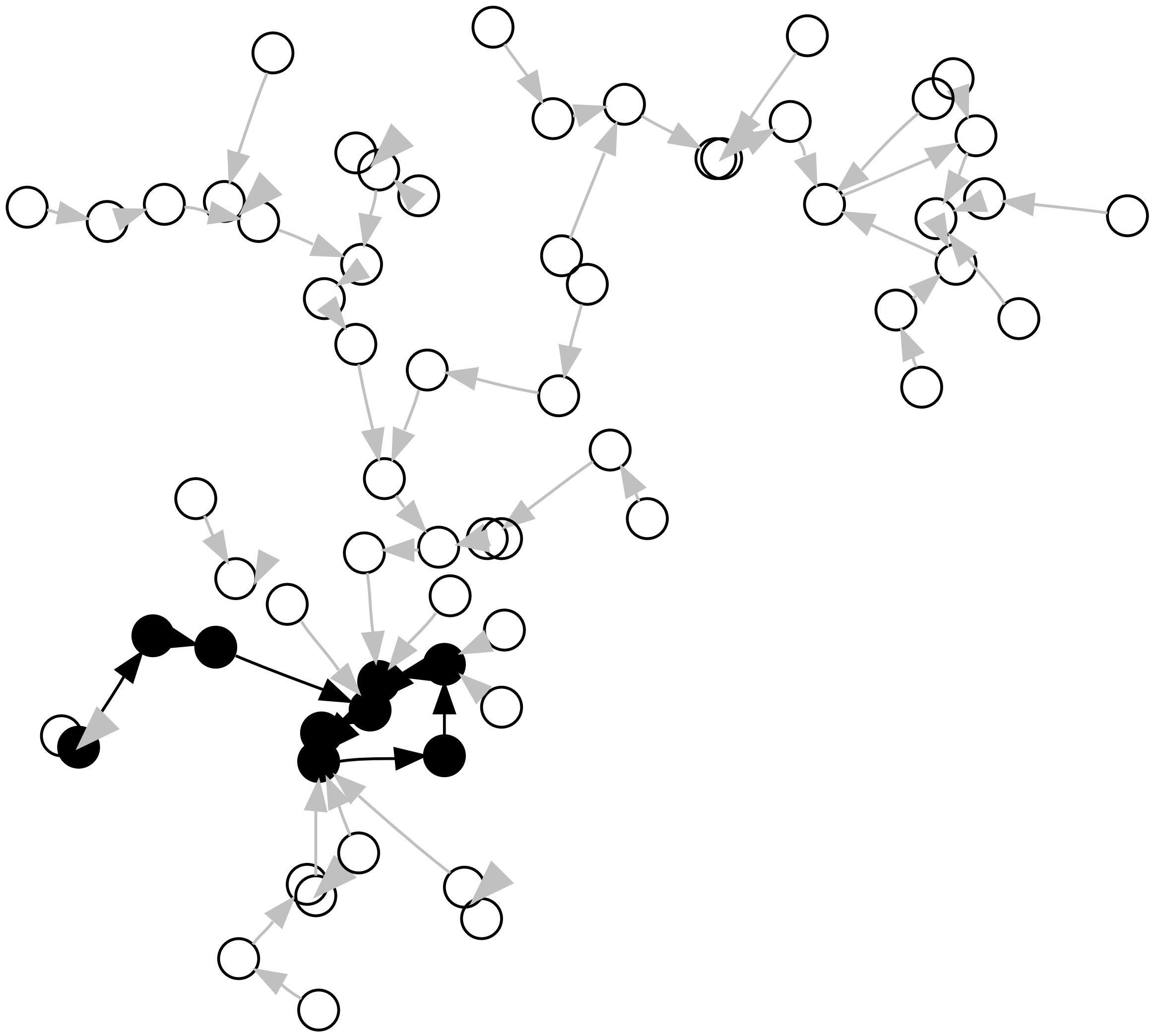


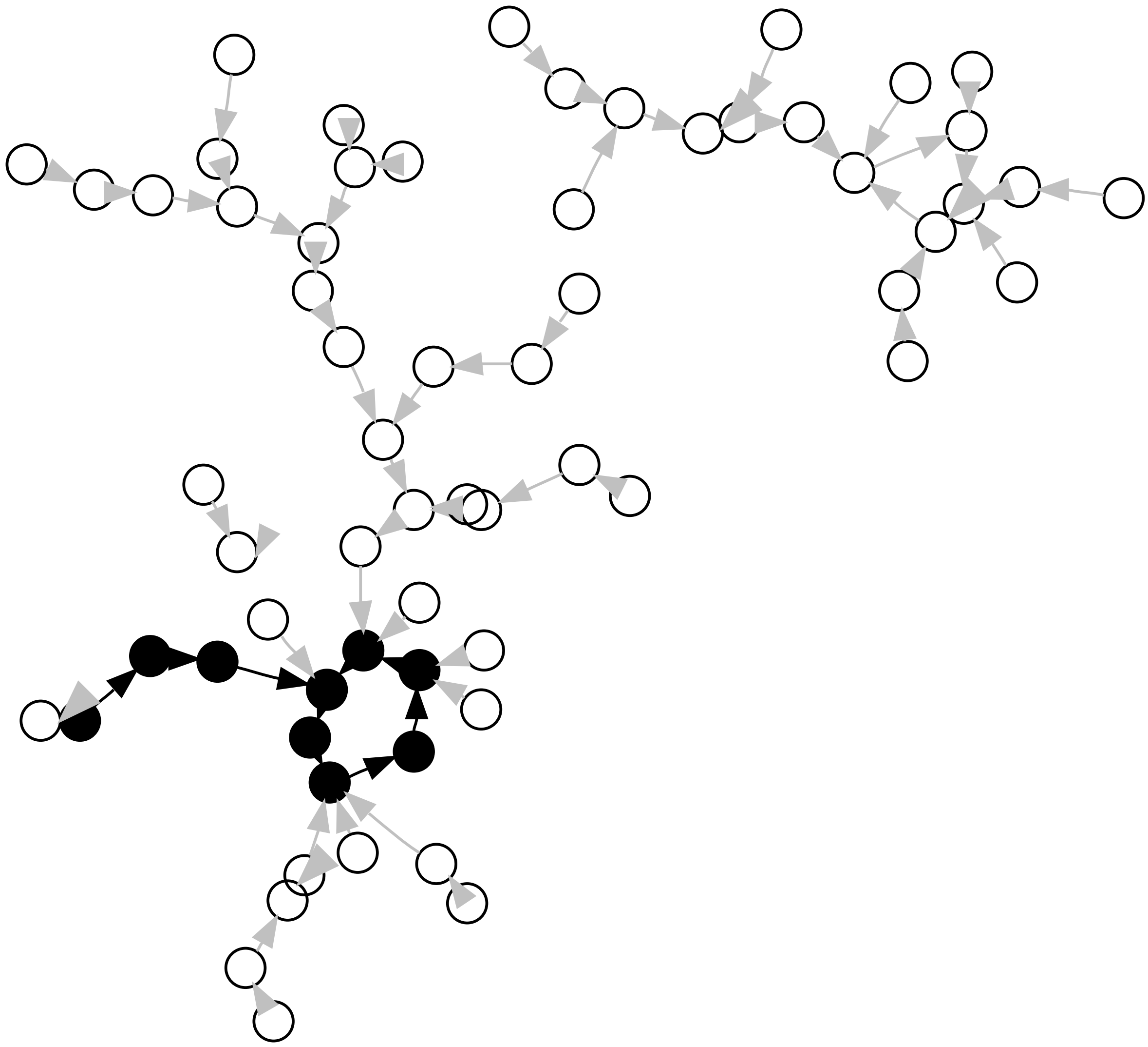


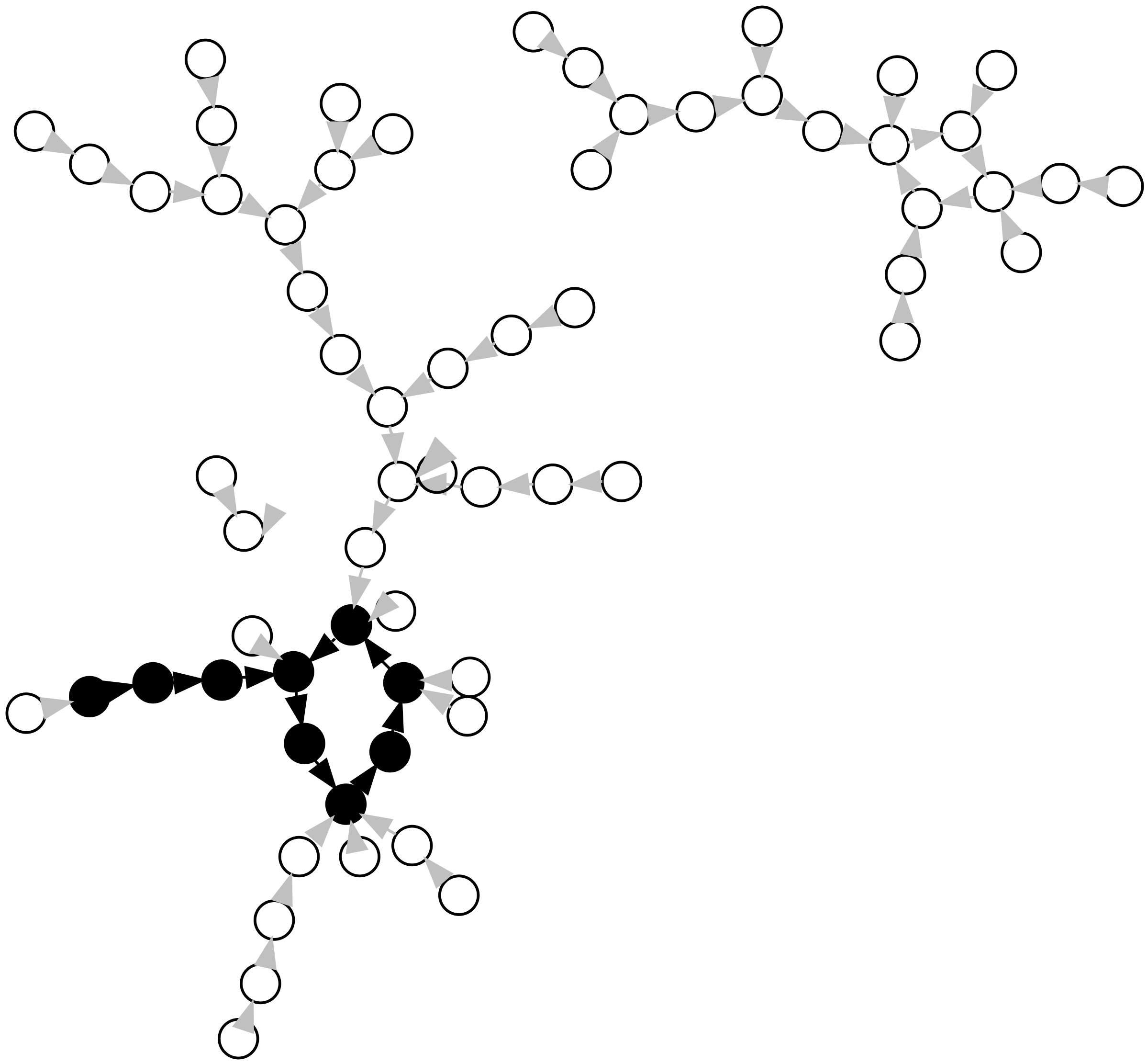












Assume that for each point
we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $W_i = [a_i]P + [b_i]Q$.

Then $W_i = W_j$ means that
$[a_i]P + [b_i]Q = [a_j]P + [b_j]Q$
so $[b_i - b_j]Q = [a_j - a_i]P$.
If $b_i \neq b_j$ the DLP is solved:
$k = (a_j - a_i)/(b_i - b_j)$.

Assume that for each point
we know $a_i, b_i \in \mathbf{Z}/\ell\mathbf{Z}$
so that $W_i = [a_i]P + [b_i]Q$.

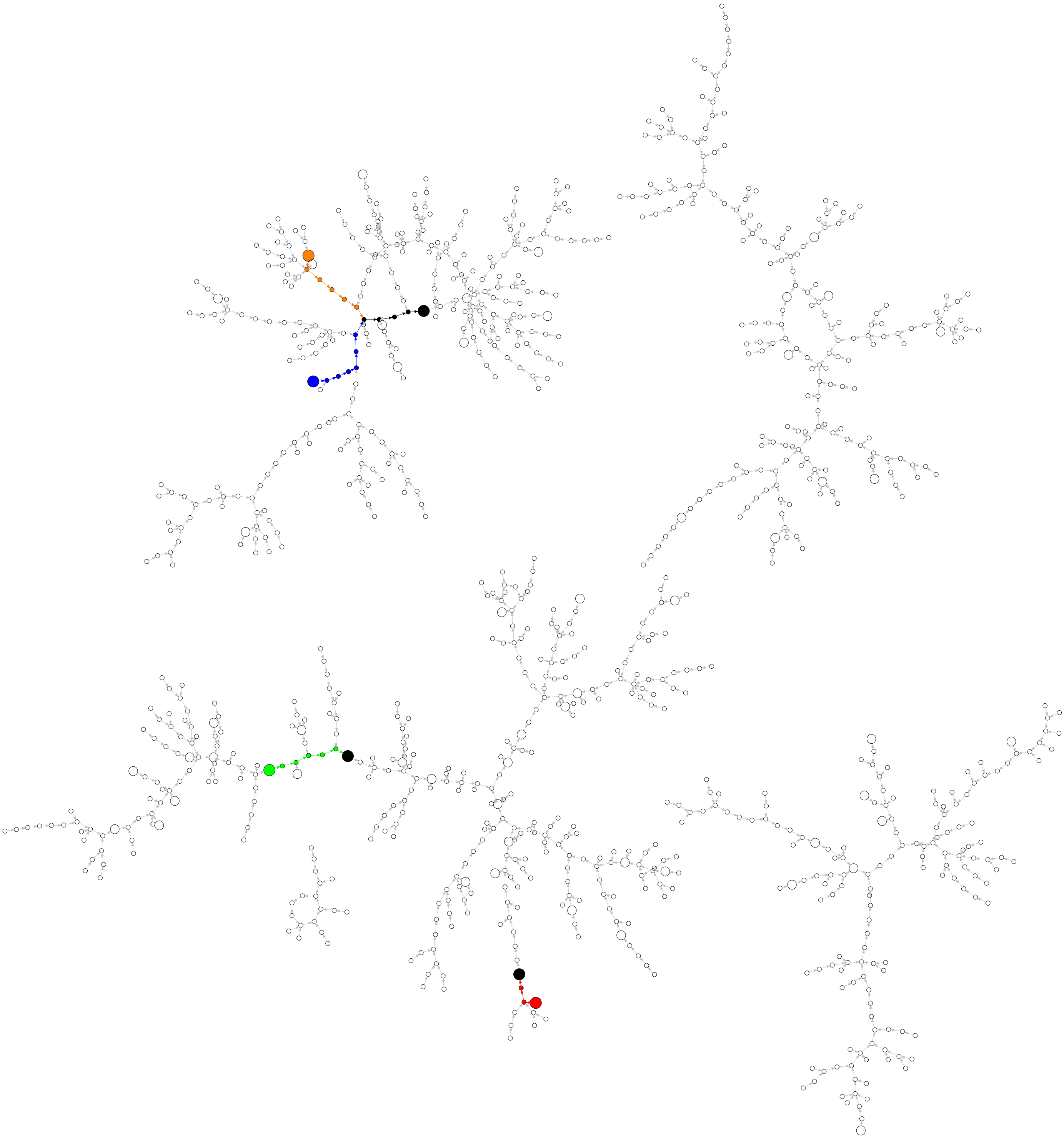
Then $W_i = W_j$ means that
$[a_i]P + [b_i]Q = [a_j]P + [b_j]Q$
so $[b_i - b_j]Q = [a_j - a_i]P$.
If $b_i \neq b_j$ the DLP is solved:
$k = (a_j - a_i)/(b_i - b_j)$.

e.g. "Additive walk":
Start with $W_0 = P$ and put
$f(W_i) = W_i + c_j P + d_j Q$
where $j = h(W_i)$.

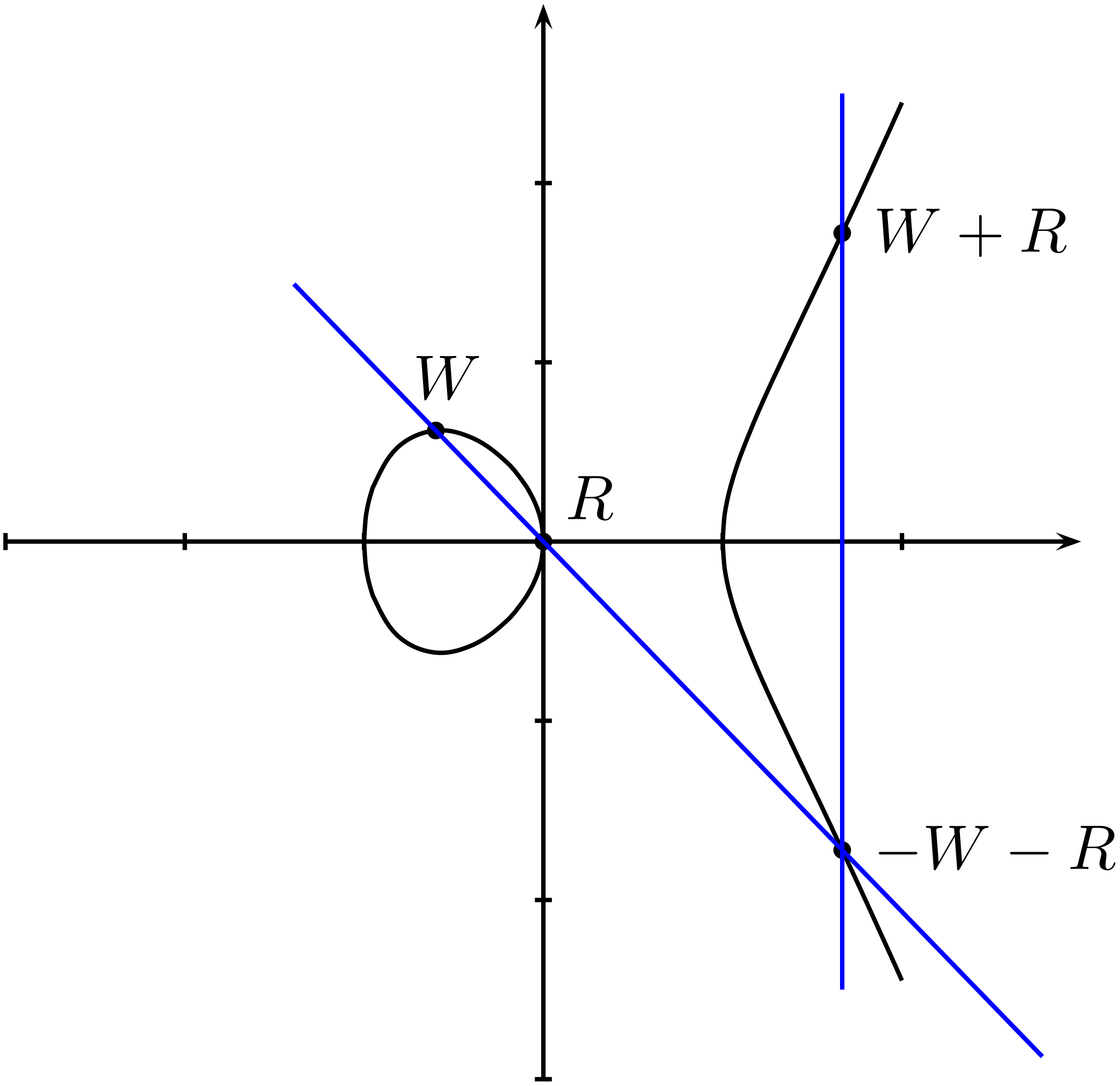
Parallel rho: Perform many walks with different starting points but same update function f. If two different walks find the same point then their subsequent steps will match.

Terminate each walk once it hits a **distinguished point**. Attacker chooses frequency and definition of distinguished points. Do not wait for cycle.

Collect all distinguished points. Two walks ending in same distinguished point solve DLP.

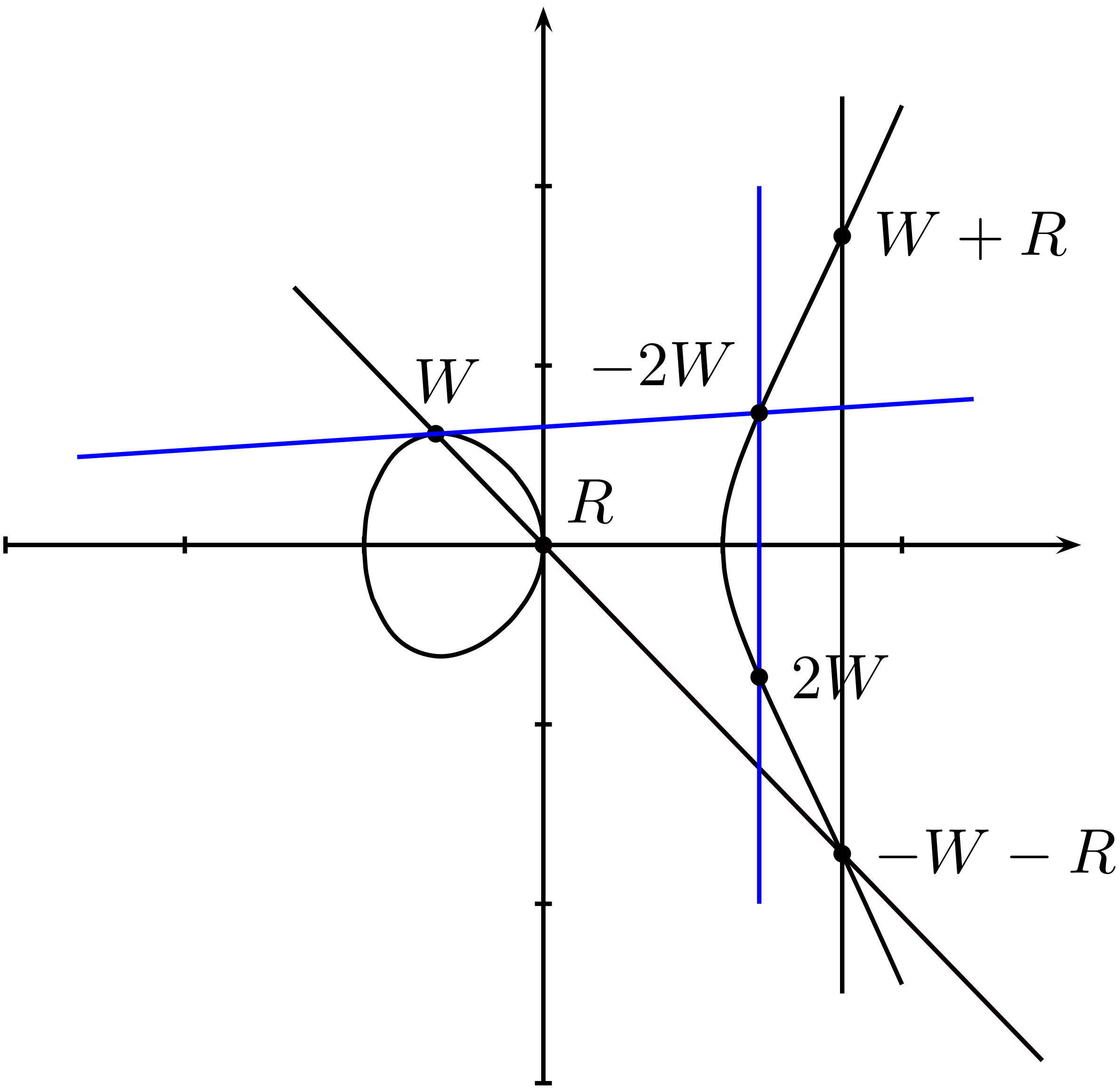


Elliptic-curve groups



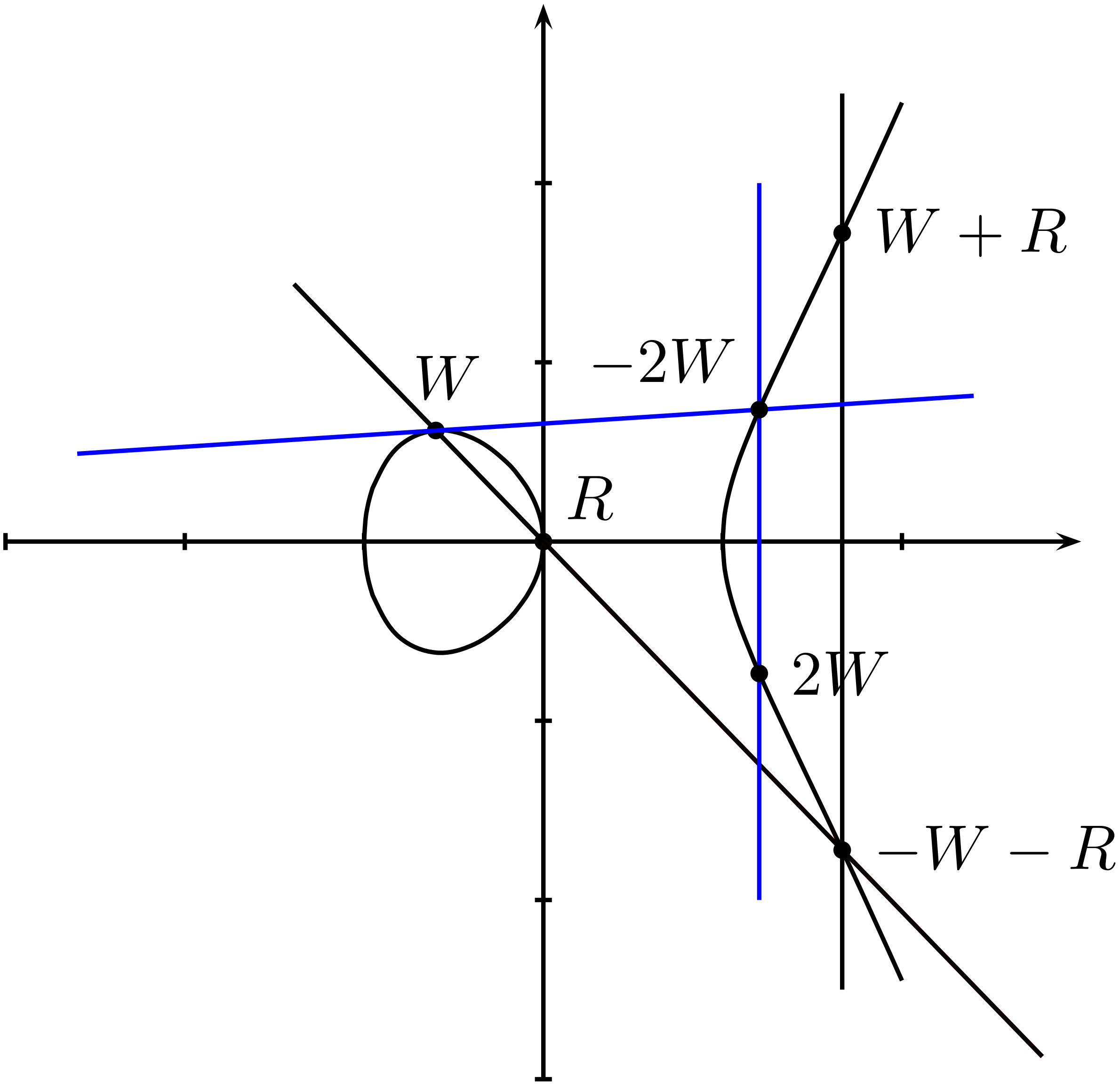
$$y^2 = x^3 + ax + b.$$

Elliptic-curve groups



$$y^2 = x^3 + ax + b.$$

Elliptic-curve groups



$y^2 = x^3 + ax + b$.

Also neutral element at ∞.

$-(x, y) = (x, -y)$.

$(x_W, y_W) + (x_R, y_R) =$
$(x_{W+R}, y_{W+R}) =$
$(\lambda^2 - x_W - x_R, \lambda(x_W - x_{W+R}) - y_W).$

$x_W \neq x_R$, "addition":
$\lambda = (y_R - y_W)/(x_R - x_W).$
Total cost $1\mathbf{I} + 2\mathbf{M} + 1\mathbf{S}$.

$W = R$ and $y_W \neq 0$, "doubling":
$\lambda = (3x_W^2 + a)/(2y_W).$
Total cost $1\mathbf{I} + 2\mathbf{M} + 2\mathbf{S}$.

Also handle some exceptions:
$(x_W, y_W) = (x_R, -y_R)$;
inputs at ∞.

Negation and rho

$W = (x, y)$ and $-W = (x, -y)$ have same x-coordinate. Search for x-coordinate collision.

Search space for collisions is only $\lceil \ell/2 \rceil$; this gives factor $\sqrt{2}$ speedup ... if $f(W_i) = f(-W_i)$.

To ensure $f(W_i) = f(-W_i)$: Define $j = h(|W_i|)$ and $f(W_i) = |W_i| + c_j P + d_j Q$. Define $|W_i|$ as, e.g., lexicographic minimum of $W_i, -W_i$.

Problem: this walk can
run into fruitless cycles!

Example: If $|W_{i+1}| = -W_{i+1}$
and $h(|W_{i+1}|) = j = h(|W_i|)$
then $W_{i+2} = f(W_{i+1}) =$
$-W_{i+1} + c_j P + d_j Q =$
$-(|W_i| + c_j P + d_j Q) + c_j P + d_j Q =$
$-|W_i|$ so $|W_{i+2}| = |W_i|$
so $W_{i+3} = W_{i+1}$
so $W_{i+4} = W_{i+2}$ etc.

If h maps to r different values
then expect this example to occur
with probability $1/(2r)$
at each step.

Current ECDL record:

2009.07 Bos–Kaihara–
Kleinjung–Lenstra–Montgomery
"PlayStation 3 computing
breaks 2^{60} barrier:
112-bit prime ECDLP solved".

Standard curve over $\mathbf{F}_p$
where $p = (2^{128} - 3)/(11 \cdot 6949)$.

Current ECDL record:

2009.07 Bos–Kaihara–
Kleinjung–Lenstra–Montgomery
"PlayStation 3 computing
breaks 2^{60} barrier:
112-bit prime ECDLP solved".

Standard curve over $\mathbf{F}_p$
where $p = (2^{128} - 3)/(11 \cdot 6949)$.

"We did not use
the common negation map
since it requires branching
and results in code that runs
slower in a SIMD environment."
All modern CPUs are SIMD.

2009.07 Bos–Kaihara–Kleinjung–Lenstra–Montgomery "On the security of 1024-bit RSA and 160-bit elliptic curve cryptography":

Group order $q \approx p$; "expected number of iterations" is "$\sqrt{\frac{\pi \cdot q}{2}} \approx 8.4 \cdot 10^{16}$"; "we do not use the negation map"; "456 clock cycles per iteration per SPU"; "24-bit distinguishing property" $\Rightarrow$ "260 gigabytes".

"The overall calculation can be expected to take approximately 60 PS3 years."

2009.09 Bos–Kaihara–Montgomery "Pollard rho on the PlayStation 3":

"Our software implementation is optimized for the SPE . . . the computational overhead for [the negation map], due to the conditional branches required to check for fruitless cycles [13], results (in our implementation on this architecture) in an overall performance degradation."

"[13]" is 2000 Gallant–Lambert–Vanstone.

2010.07 Bos–Kleinjung–Lenstra "On the use of the negation map in the Pollard rho method":

"If the Pollard rho method is parallelized in SIMD fashion, it is a challenge to achieve any speedup at all. . . . Dealing with cycles entails administrative overhead and branching, which cause a non-negligible slowdown when running multiple walks in SIMD-parallel fashion. . . . [This] is a major obstacle to the negation map in SIMD environments."

This paper: Our software solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.

For comparison: Bos–Kaihara–Kleinjung–Lenstra–Montgomery software uses 65 PS3 years on average.

This paper: Our software solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.

For comparison: Bos–Kaihara–Kleinjung–Lenstra–Montgomery software uses 65 PS3 years on average.

Computation used 158000 kWh (if PS3 ran at only 300W), wasting >70000 kWh, unnecessarily generating >10000 kilograms of carbon dioxide. (0.143 kg CO2 per Swiss kWh.)

Several levels of speedups, starting with fast arithmetic mod $p = (2^{128} - 3)/(11 \cdot 6949)$ and continuing up through rho.

Most important speedup: We use the negation map.

Several levels of speedups,
starting with fast arithmetic
mod $p = (2^{128} - 3)/(11 \cdot 6949)$
and continuing up through rho.

Most important speedup:
We use the negation map.

Extra cost in each iteration:
extract bit of "s"
(normalized y, needed anyway);
expand bit into mask;
use mask to conditionally
replace (s, y) by $(-s, -y)$.
5.5 SPU cycles ($\approx$ 1.5% of total).
No conditional branches.

Bos–Kleinjung–Lenstra say that "on average more elliptic curve group operations are required per step of each walk. This is unavoidable" etc.

Specifically: If the precomputed additive-walk table has r points, need 1 extra doubling to escape a cycle after $\approx 2r$ additions. And more: "cycle reduction" etc.

Bos–Kleinjung–Lenstra say that the benefit of large r is "wiped out by cache inefficiencies."

There's really no problem here!

We use $r = 2048$.
$1/(2r) = 1/4096$; negligible.

Recall: p has 112 bits.
28 bytes for table entry (x, y).

We expand to 36 bytes
to accelerate arithmetic.
We compress to 32 bytes
by insisting on small x, y;
very fast initial computation.

Only 64KB for table.
Our Cell table-load cost: 0,
overlapping loads with arithmetic.
No "cache inefficiencies."

What about fruitless cycles?

We run 45 iterations.
We then save s;
run 2 slightly slower iterations
tracking minimum (s, x, y);
then double tracked (x, y)
if new s equals saved s.

(Occasionally replace 2 by 12
to detect 4-cycles, 6-cycles.
Such cycles are almost
too rare to worry about,
but detecting them has a
completely negligible cost.)

Maybe fruitless cycles waste some of the 47 iterations.
. . . but this is infrequent.
Lose $\approx 0.6\%$ of all iterations.

Tracking minimum isn't free, but most iterations skip it!
Same for final s comparison.
Still no conditional branches.
Overall cost $\approx 1.3\%$.

Doubling occurs for only $\approx 1/4096$ of all iterations.
We use SIMD quite lazily here; overall cost $\approx 0.6\%$.
Can reduce this cost further.

To confirm iteration effectiveness
we have run many experiments
on $y^2 = x^3 - 3x + 9$
over the same $\mathbf{F}_p$,
using smaller-order P.
Matched DL cost predictions.

Final conclusions:
Sensible use of negation,
with or without SIMD,
has negligible impact
on cost of each iteration.
Impact on number of iterations
is almost exactly $\sqrt{2}$.
Overall benefit is
extremely close to $\sqrt{2}$.