Advances in Elliptic-Curve Cryptography

Tanja Lange

Technische Universiteit Eindhoven

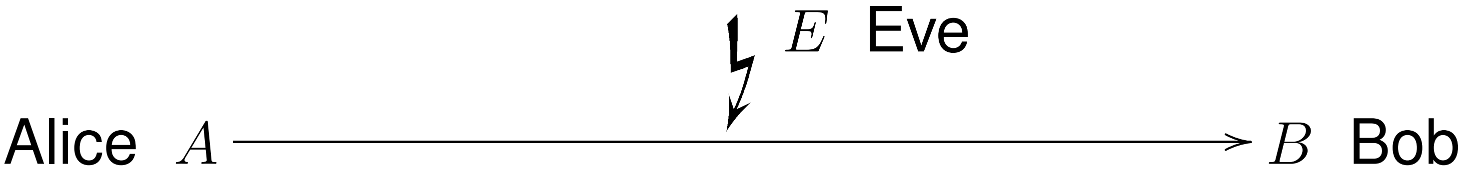
tanja@hyperelliptic.org

08.10.2011

Data Protection

Many factors influence the security and privacy of data

- Secure storage, physical security.
- Access control.
- Protection against alteration of data
  $\Rightarrow$ digital signatures, message authentication codes.
- Protection of sensitive content against reading
  $\Rightarrow$ encryption.

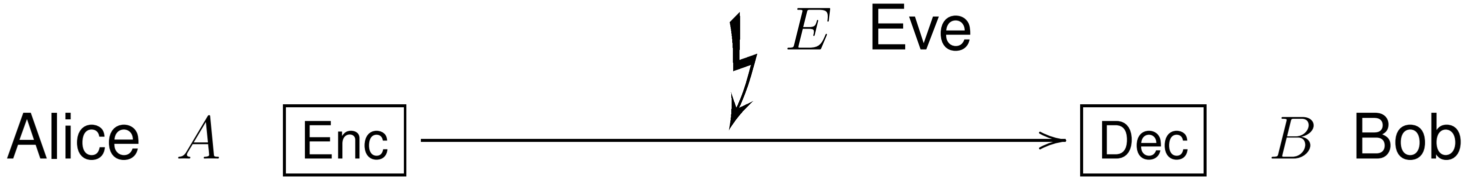


Eve hears everything on the line; she can intercept and modify contents.

Data Protection

Many factors influence the security and privacy of data

- Secure storage, physical security.
- Access control.
- Protection against alteration of data
  $\Rightarrow$ digital signatures, message authentication codes.
- Protection of sensitive content against reading
  $\Rightarrow$ encryption.



Eve hears everything on the line; she can intercept and modify contents.

Public Key cryptography

Each user has two keys

- a private key chosen by him which *never* leaves the secure environment,
- a public key which is made public in a kind of phone book.
- These keys are linked by a (computational) one-way function; the general principles are public.
- Sender uses public key of recipient to encrypt
  $\Rightarrow$ only recipient knows secret key and can decrypt.
- Sender uses her private key to sign a message
  $\Rightarrow$ signature must be due to her as only she knows the secret; every recipient can verify the signature using the sender's public key.

Common set-up

- No initial shared secret; use public key crypto
  - to get session key (e.g. Diffie-Hellman key-exchange);
  - to authenticate a message and ensure its integrity, verifiable by anybody, using digital signatures.
- Symmetric crypto methods with shared key
  - to encrypt the message (e.g.. block cipher AES);
  - to provide authenticity and integrity for the other holder of the key (e.g. message authentication codes);
- Cryptographic hash functions satisfy:
  - hash: $\{0,1\}^* \to \{0,1\}^\ell$ map to fixed length strings;
  - finding preimages and second preimages is hard;
  - finding collisions is hard.

ECDSA I

- Elliptic-Curve Digital Signature Algorithm.
- Suggested by Certicom, standardized by FIPS and others.
- Some other elliptic-curve signature schemes were proposed, e.g. by Korea and by Germany.
- Starting point is an elliptic curve E, in Weierstrass form over a finite field $\mathbb{F}_q$ and a point $P \in E(\mathbb{F}_q)$ of some prime order ℓ.
- Assume for simplicity $q > 3$ is an odd prime,

$$E : y^2 = x^3 + a_4 x + a_6;\ a_4, a_6 \in \mathbb{F}_q$$

and $E(\mathbb{F}_q) = \langle P \rangle$.

ECDSA II

- Alice's secret key is integer $a \in [1, \ell - 1]$ and her public key is $P_A = aP$.
- To sign a message m with hash value $h =$hash(m) she
  - picks random $k \in [1, \ell - 1]$;
  - computes $R = kP$ and puts $r \equiv x(R) \bmod \ell$;
  - computes $s \equiv k^{-1}(h + ra) \bmod \ell$.
  - The signature is the tuple (r, s), both values are smaller than ℓ.
- To verify the signature (r, s) on m Bob computes
  - $w \equiv s^{-1} \bmod \ell, u_1 \equiv hw \bmod \ell, u_2 \equiv rw \bmod \ell$;
  - and $Q = u_1 P + u_2 P_A$;
  - and accepts the signature if $r \equiv x(Q) \bmod \ell$; otherwise he rejects it.

ECDSA III

- Alice's signatures should be accepted, i.e. the x-coordinate of $R = kP$ should be the same as the x-coordinate of $Q = u_1 P + u_2 P_A$, where $P_A = aP$, $w \equiv s^{-1} \bmod \ell, u_1 \equiv hw \bmod \ell, u_2 \equiv rw \bmod \ell$, $r \equiv x(R) \bmod \ell$, and $s \equiv k^{-1}(h + ra) \bmod \ell$.

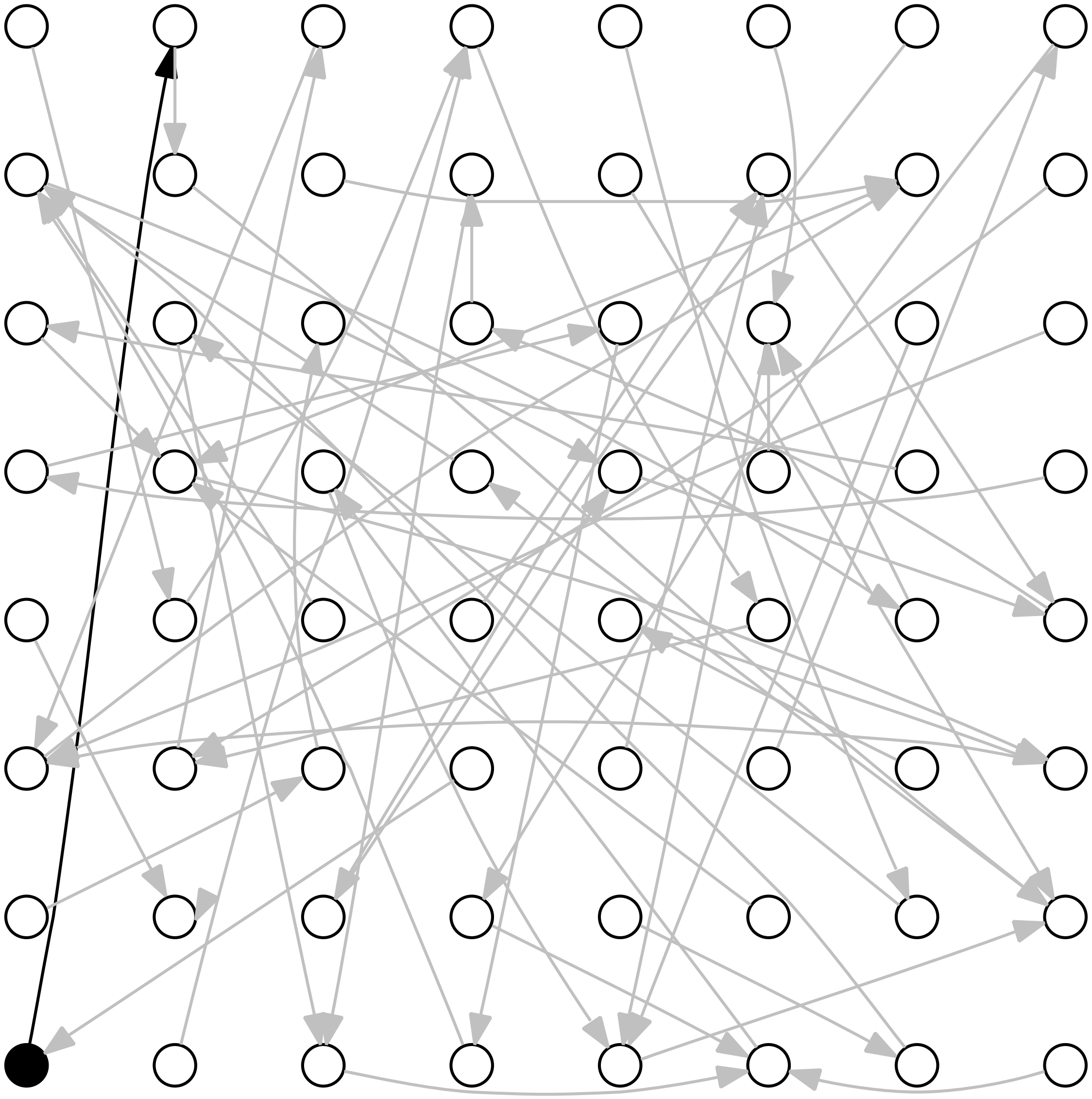
$$\begin{aligned} Q &= u_1 P + u_2 P_A = hwP + rwP_A = s^{-1}(hP + raP) \\ &= (k^{-1}(h + ra))^{-1}(h + ra)P = kP = R \end{aligned}$$

- Note that P having order ℓ takes care of $\mathrm{mod}\ell$ computations. Since $\ell \approx q$, $\pm R$ essentially given by r
- Only secret part in (r, s) is a; attacker Eve can choose any random k.
- Computing a from P and $P_A = aP$ is solving the Discrete Logarithm Problem (DLP) of P_A with base P.

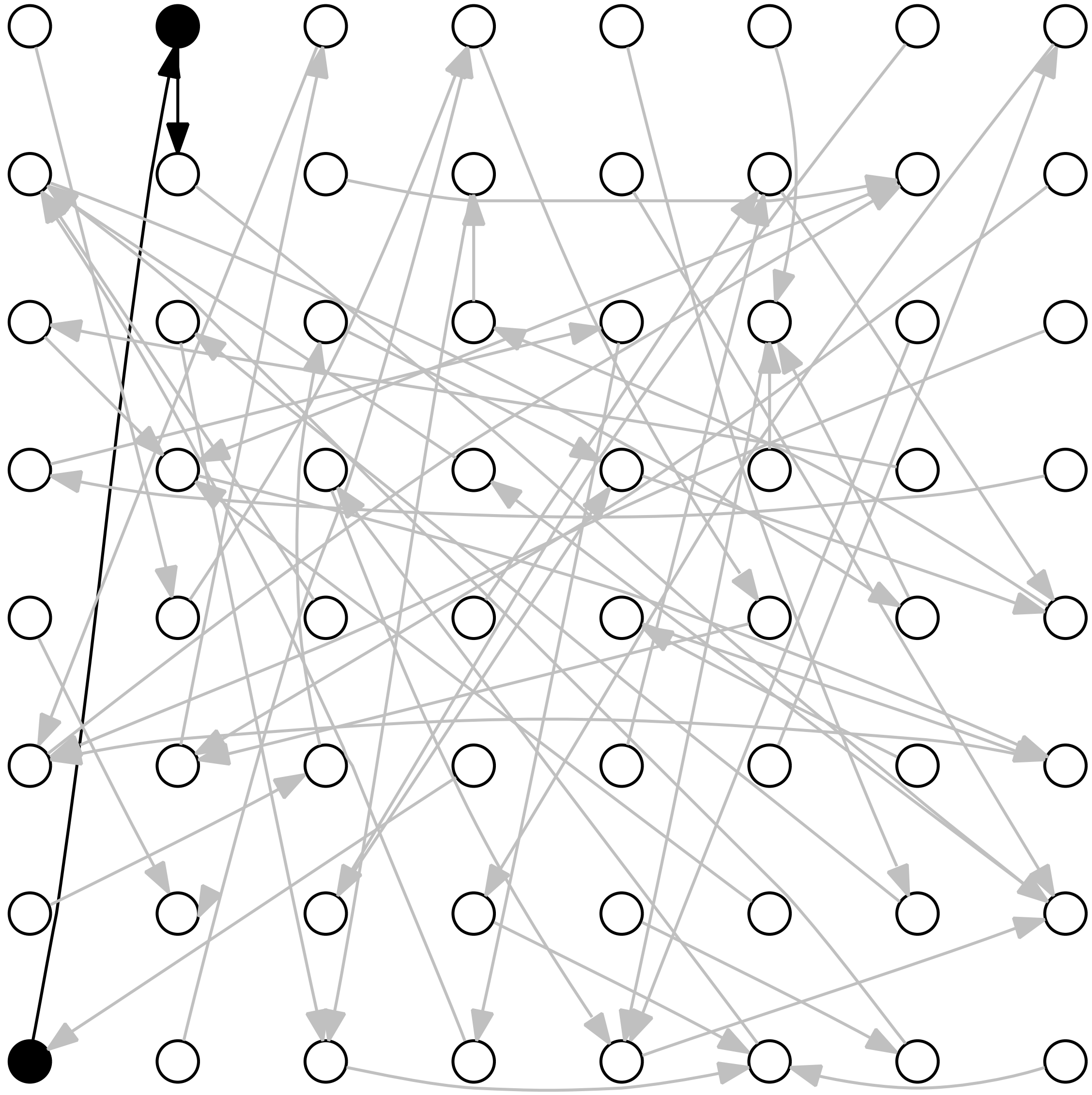
The Pollard rho method

- Standard method to compute a given P and $Q = aP$.
- 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 $\langle P \rangle$, 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.

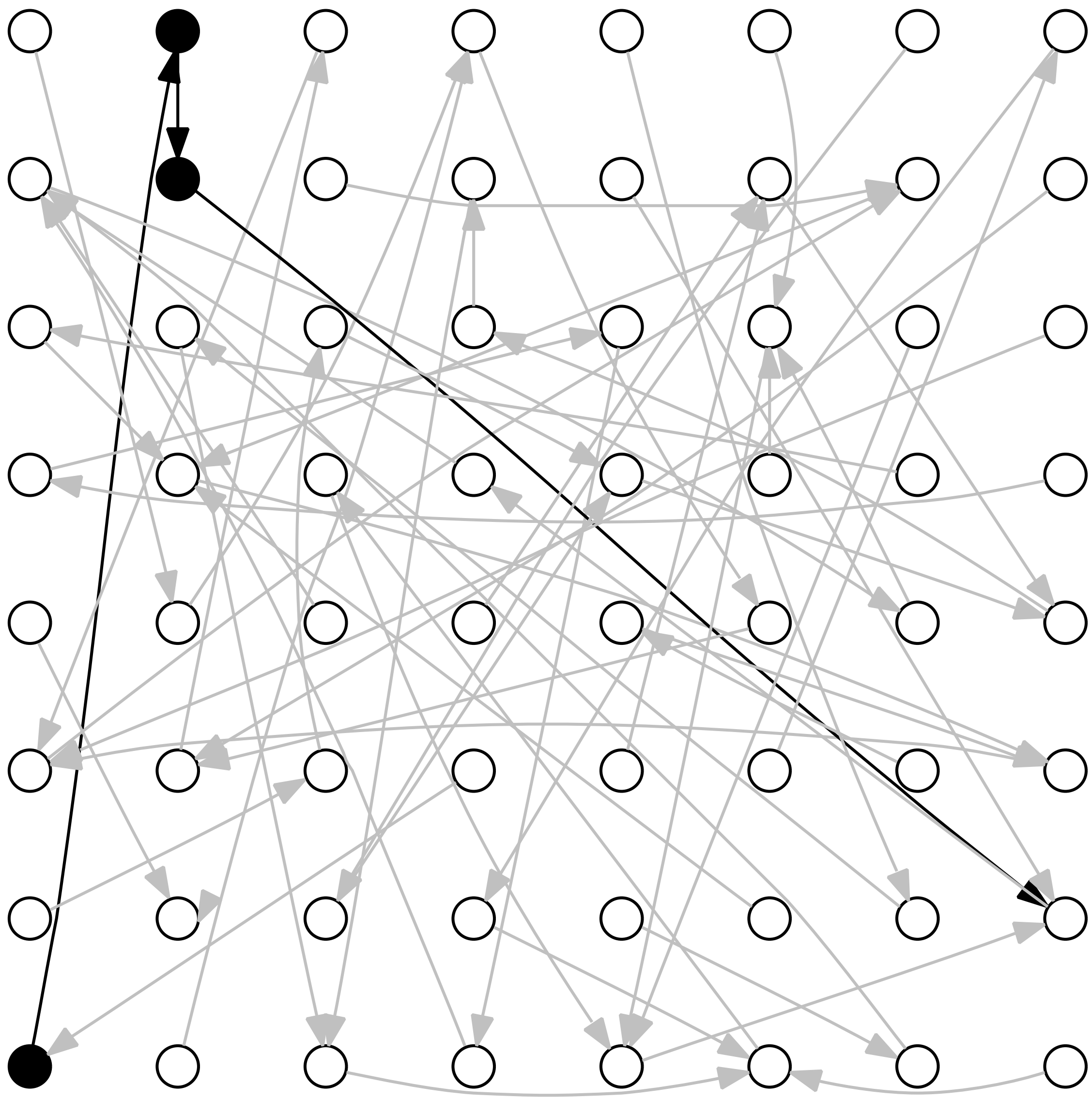
Random walk example



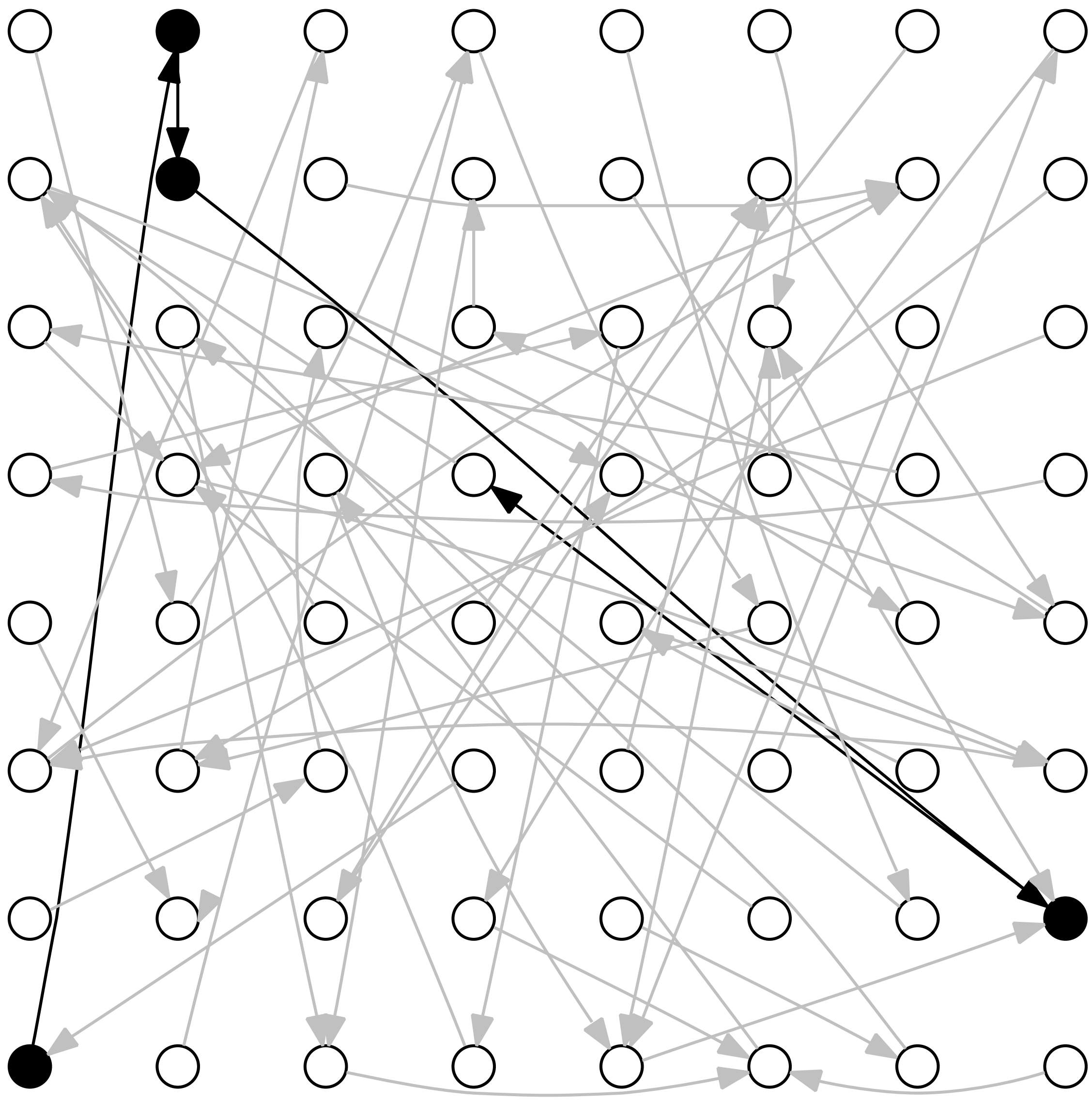
Random walk example



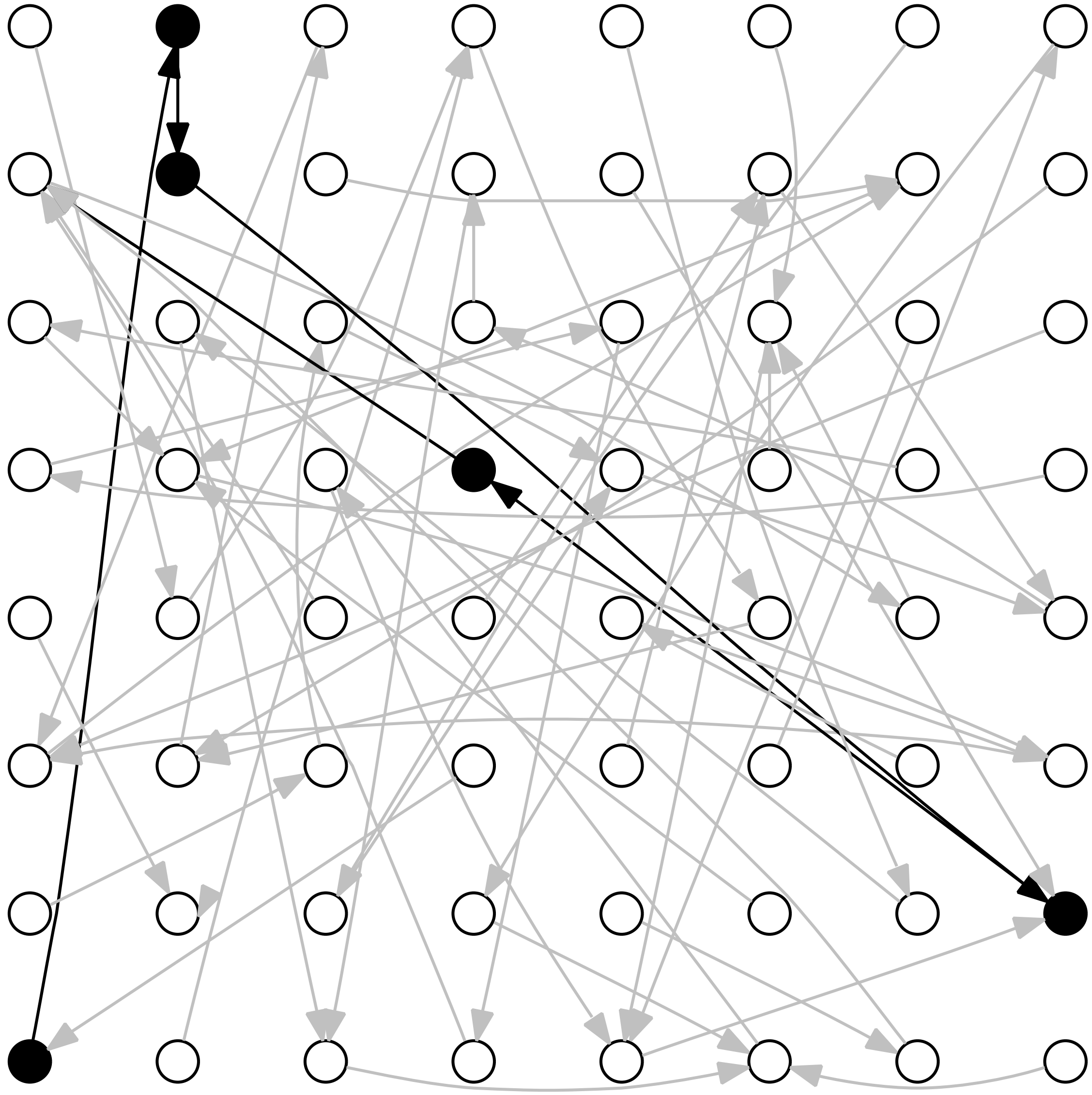
Random walk example



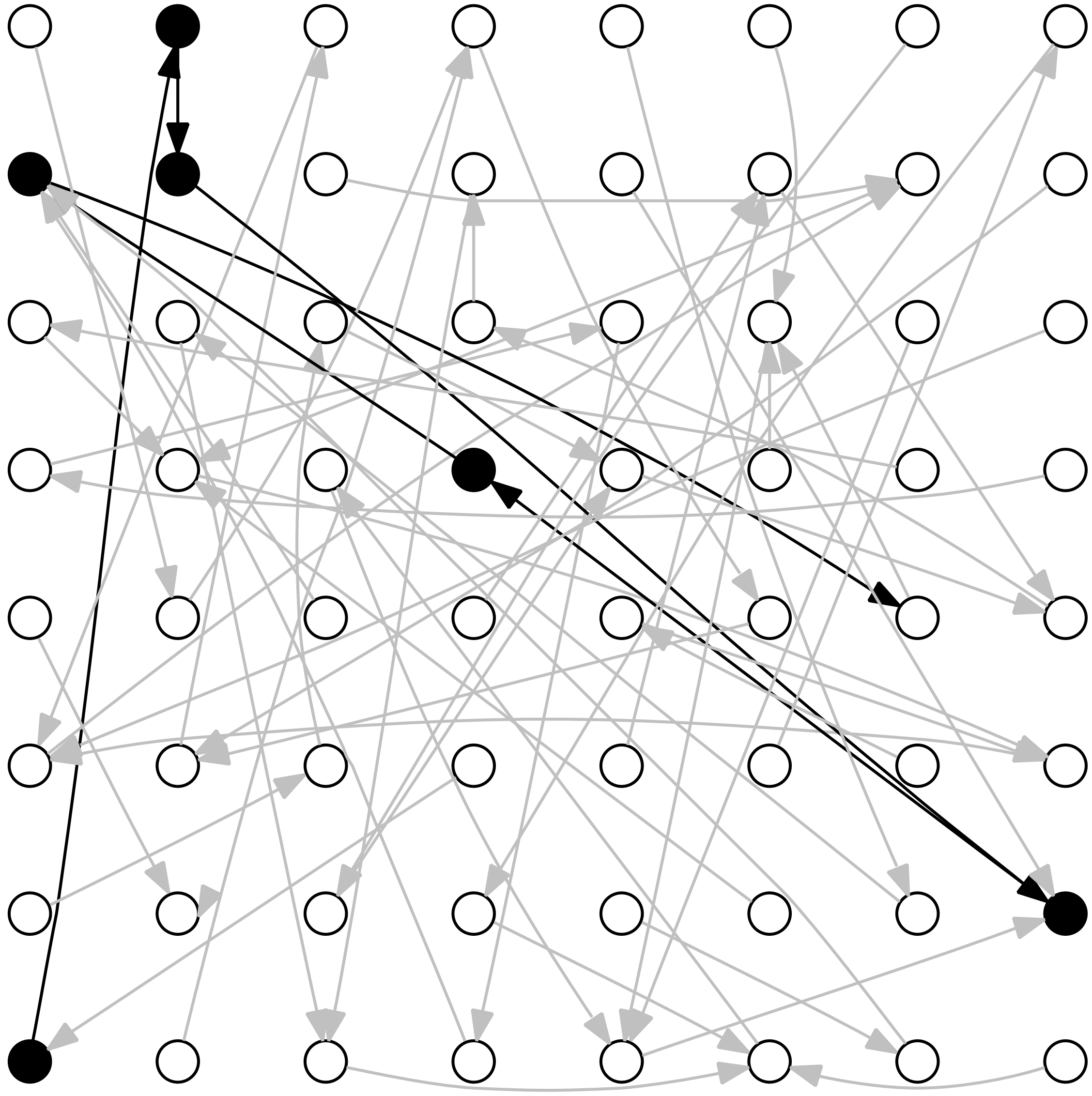
Random walk example



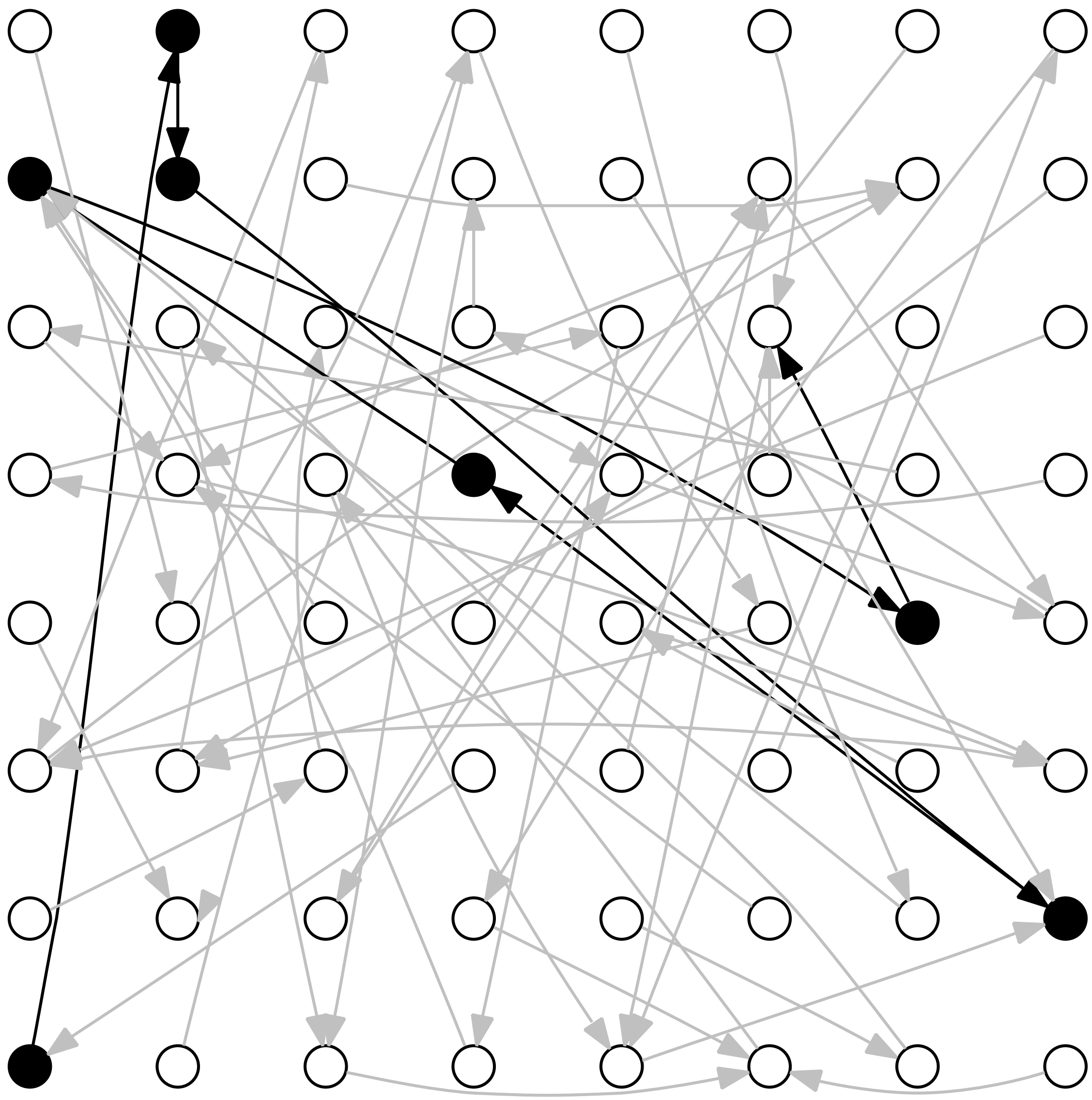
Random walk example



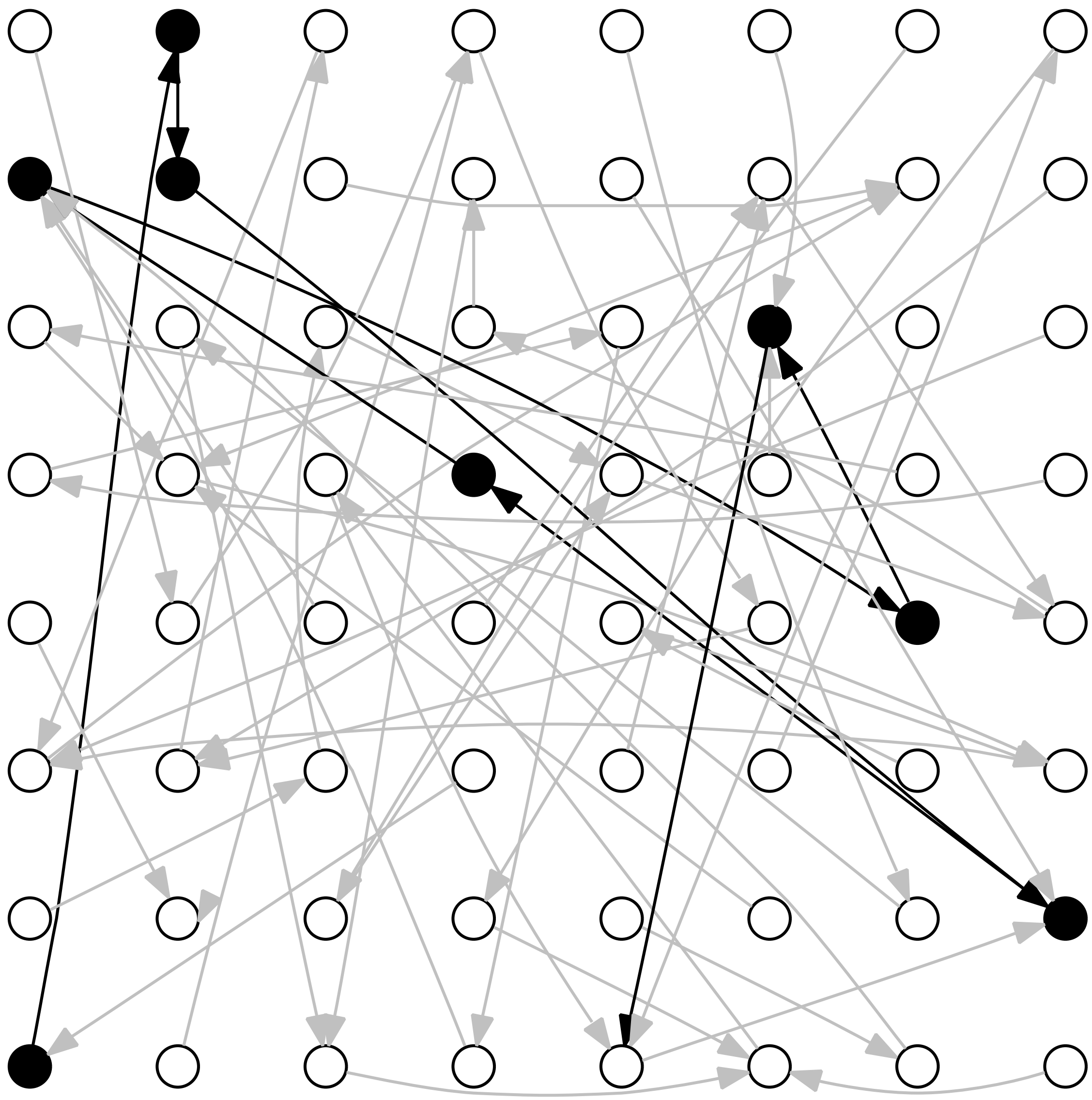
Random walk example



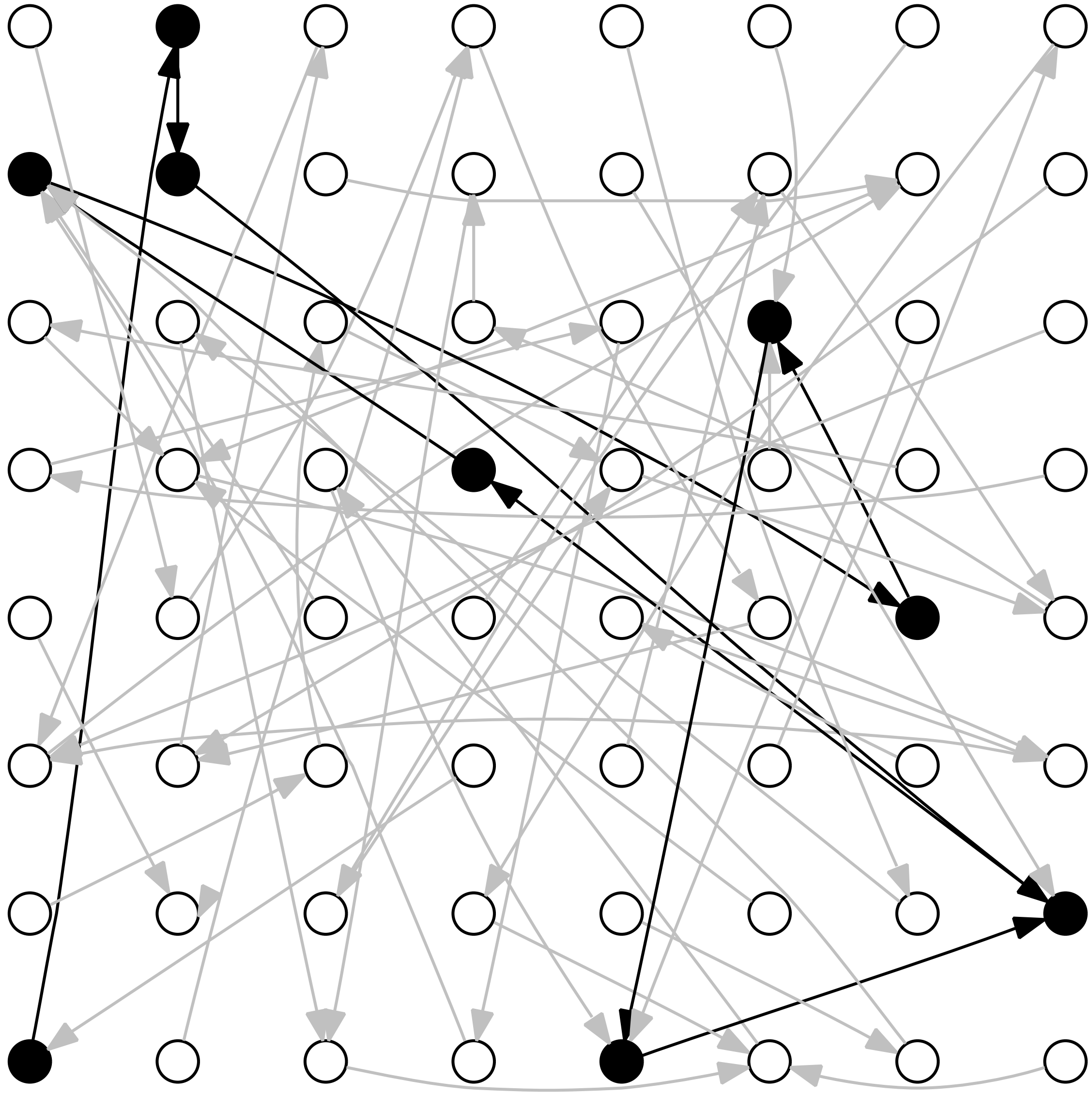
Random walk example



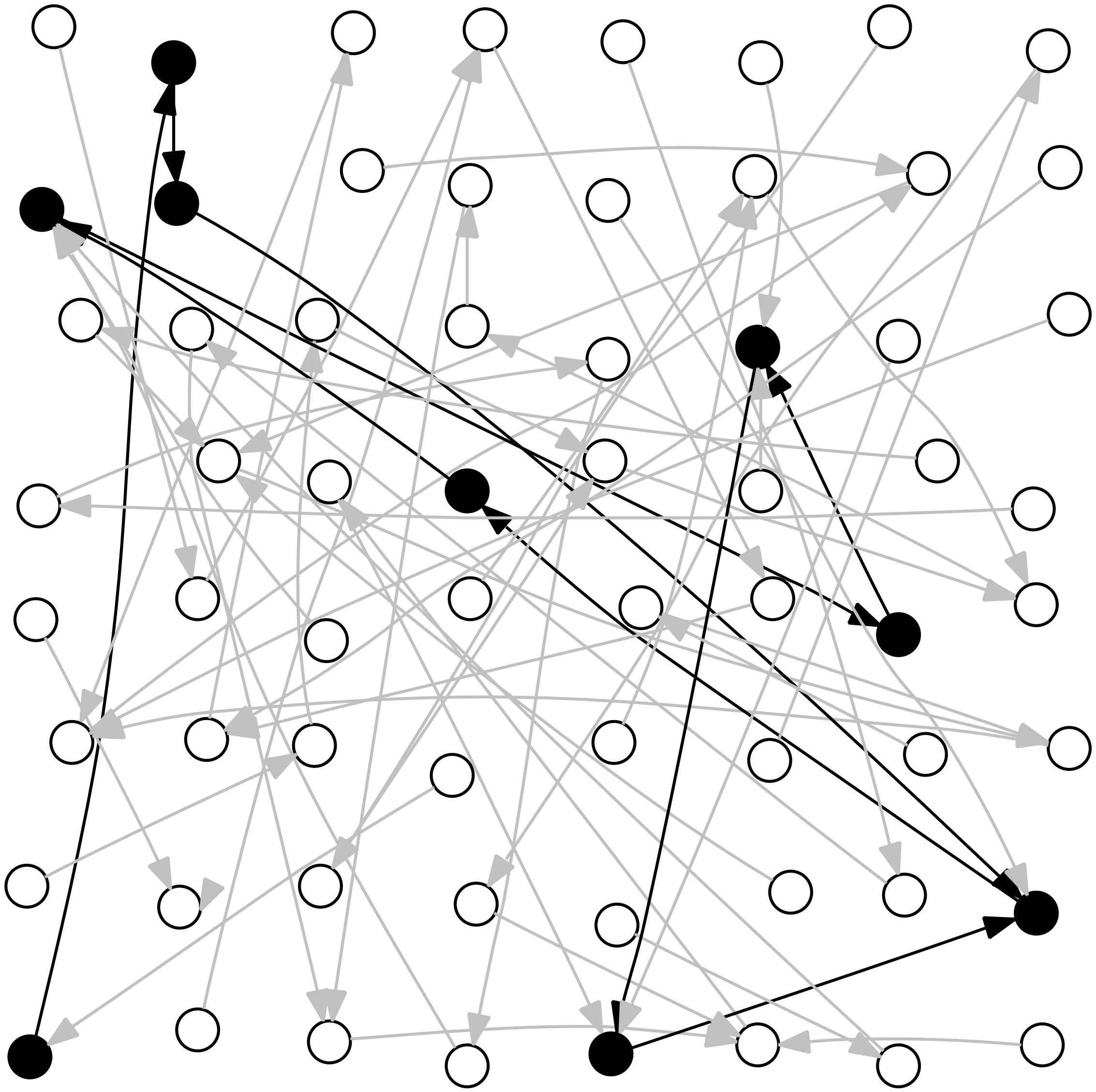
Random walk example



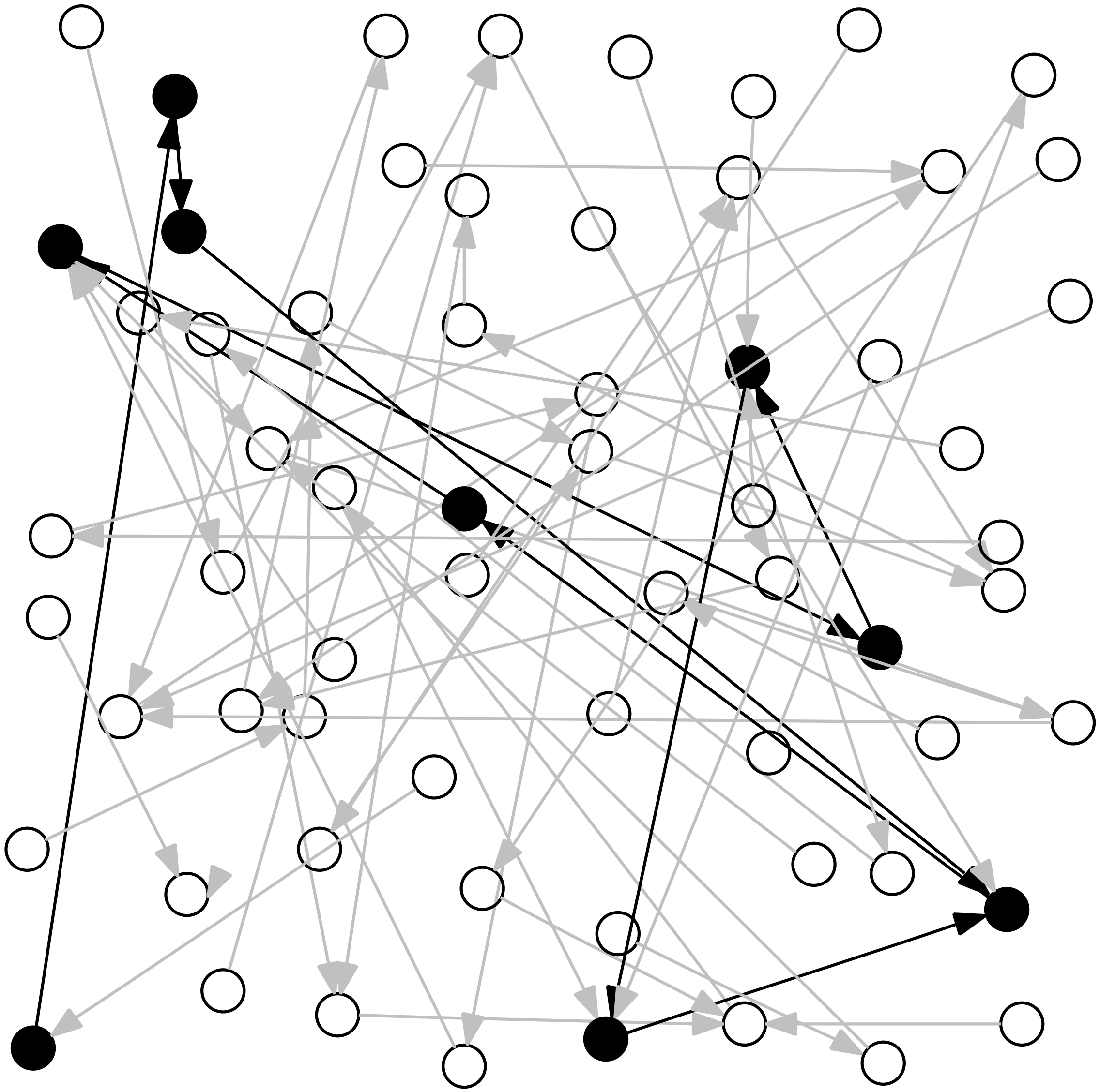
Random walk example



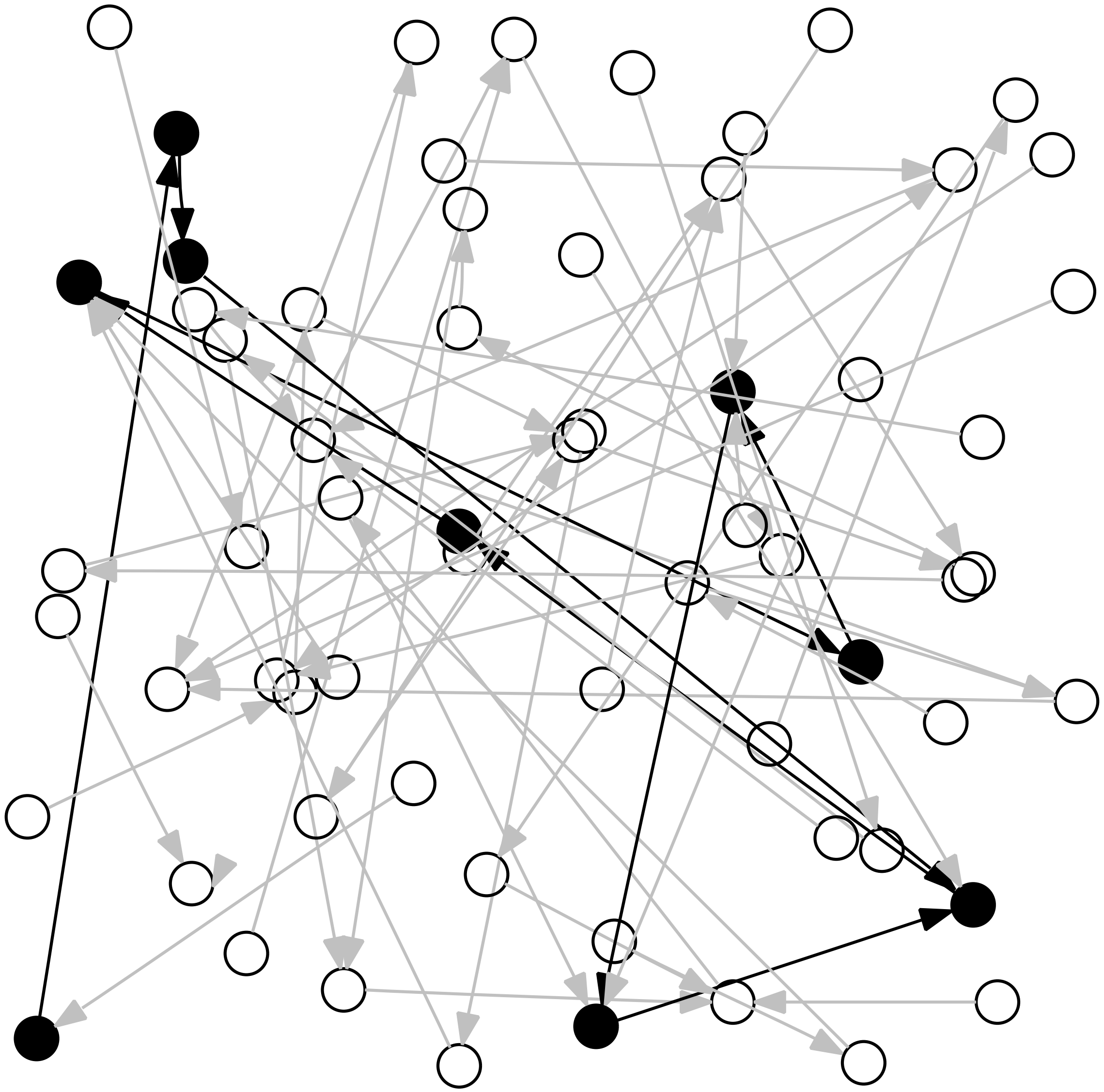
Random walk example



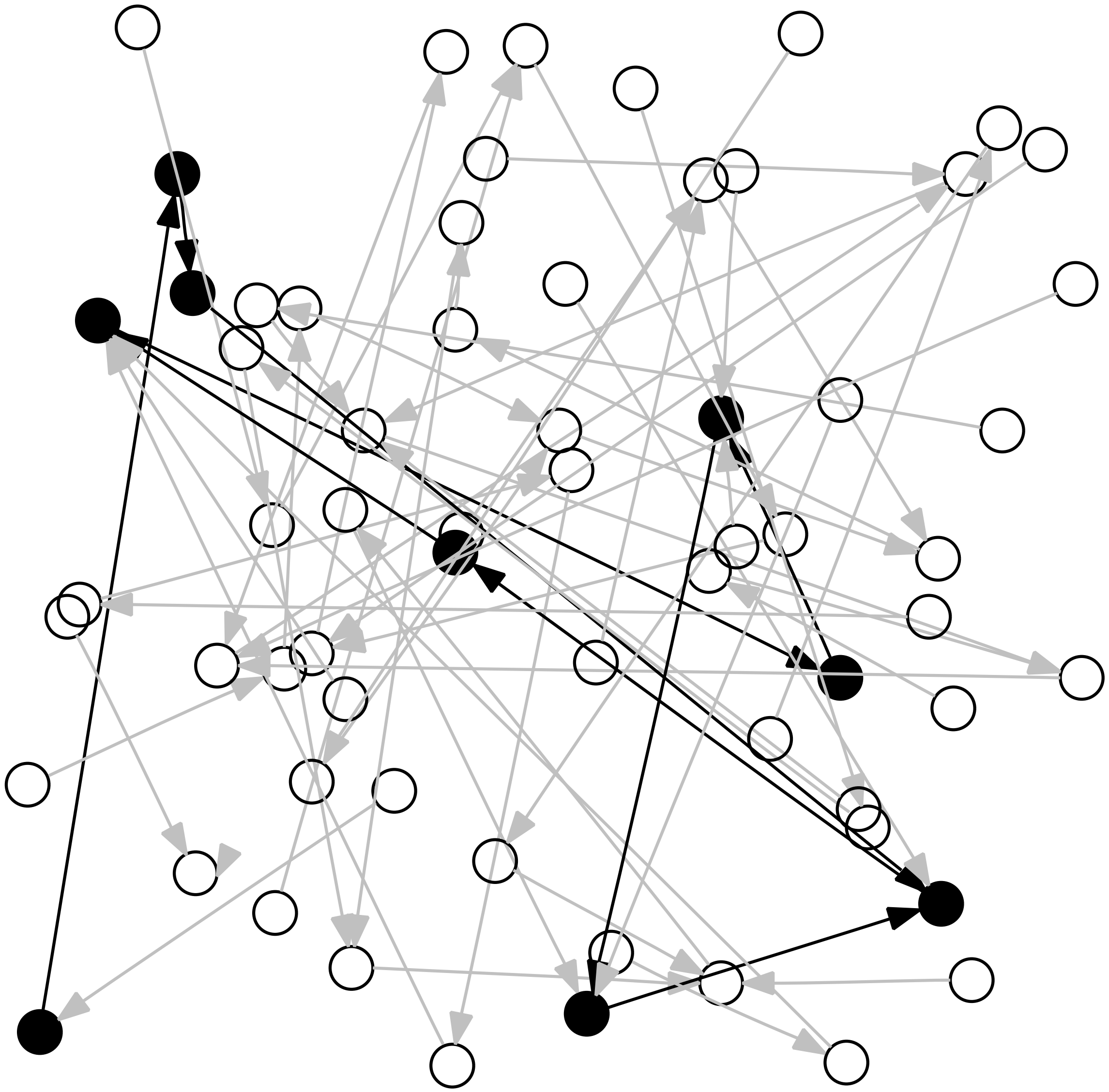
Random walk example



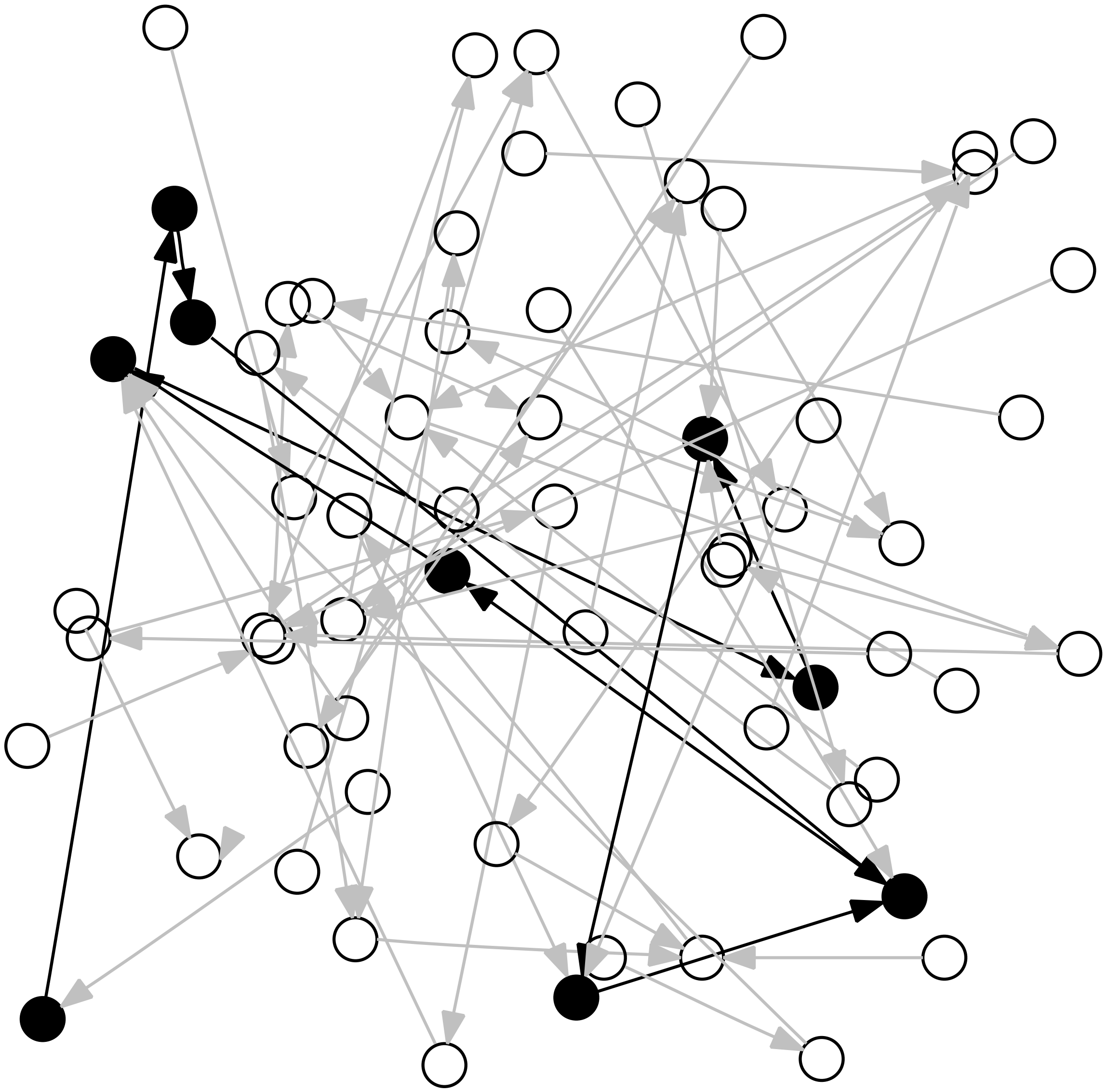
Random walk example



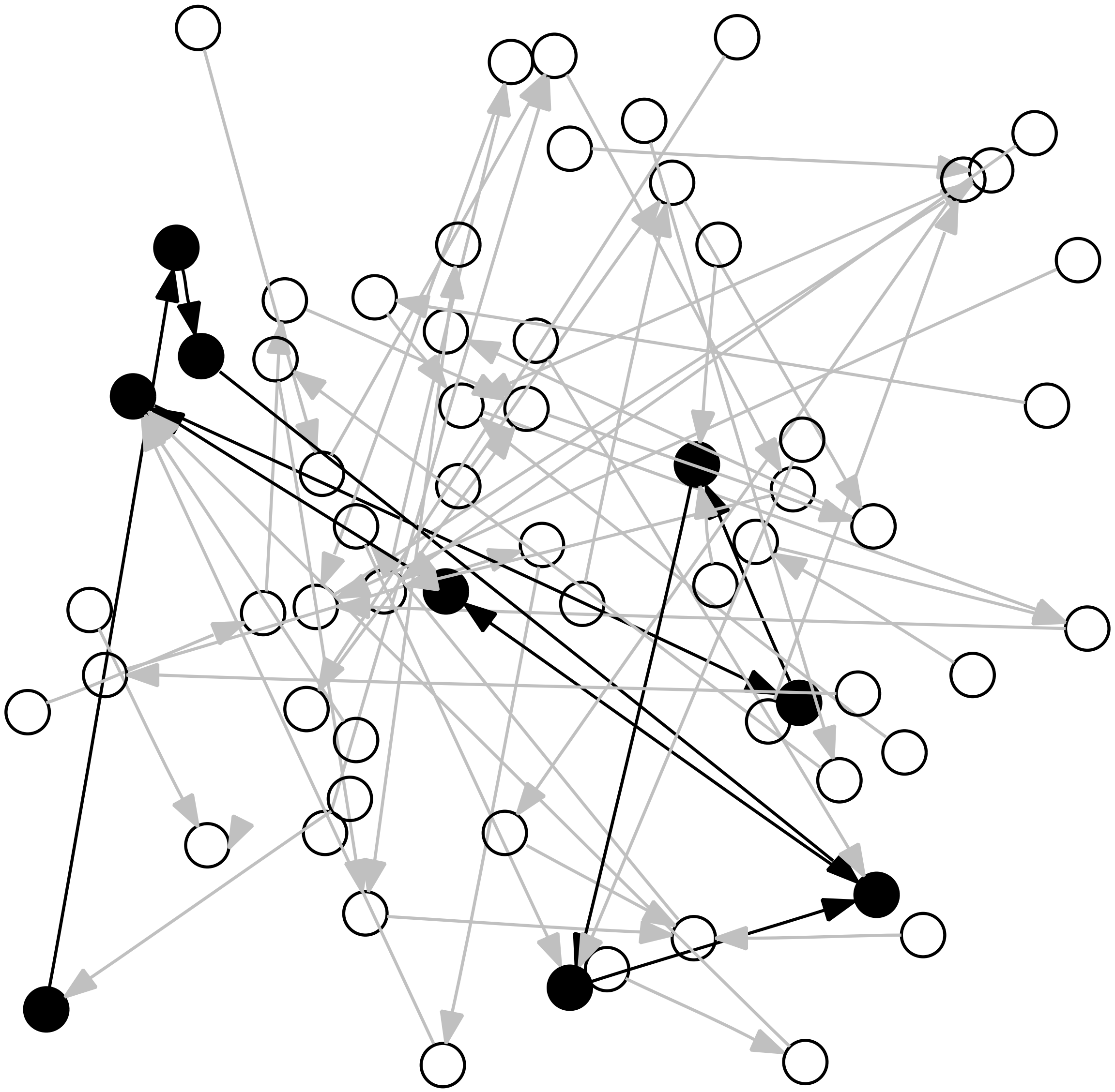
Random walk example



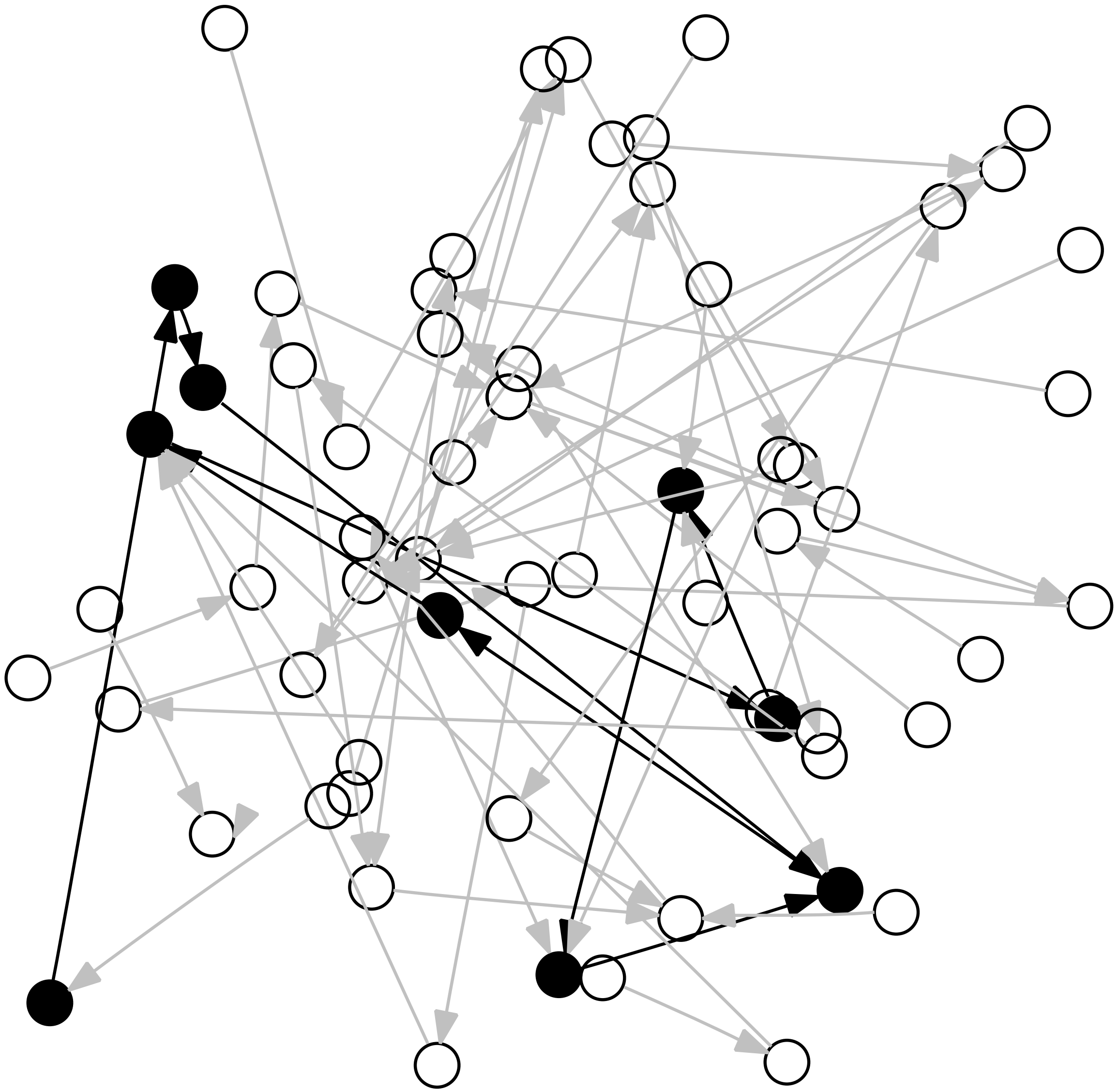
Random walk example



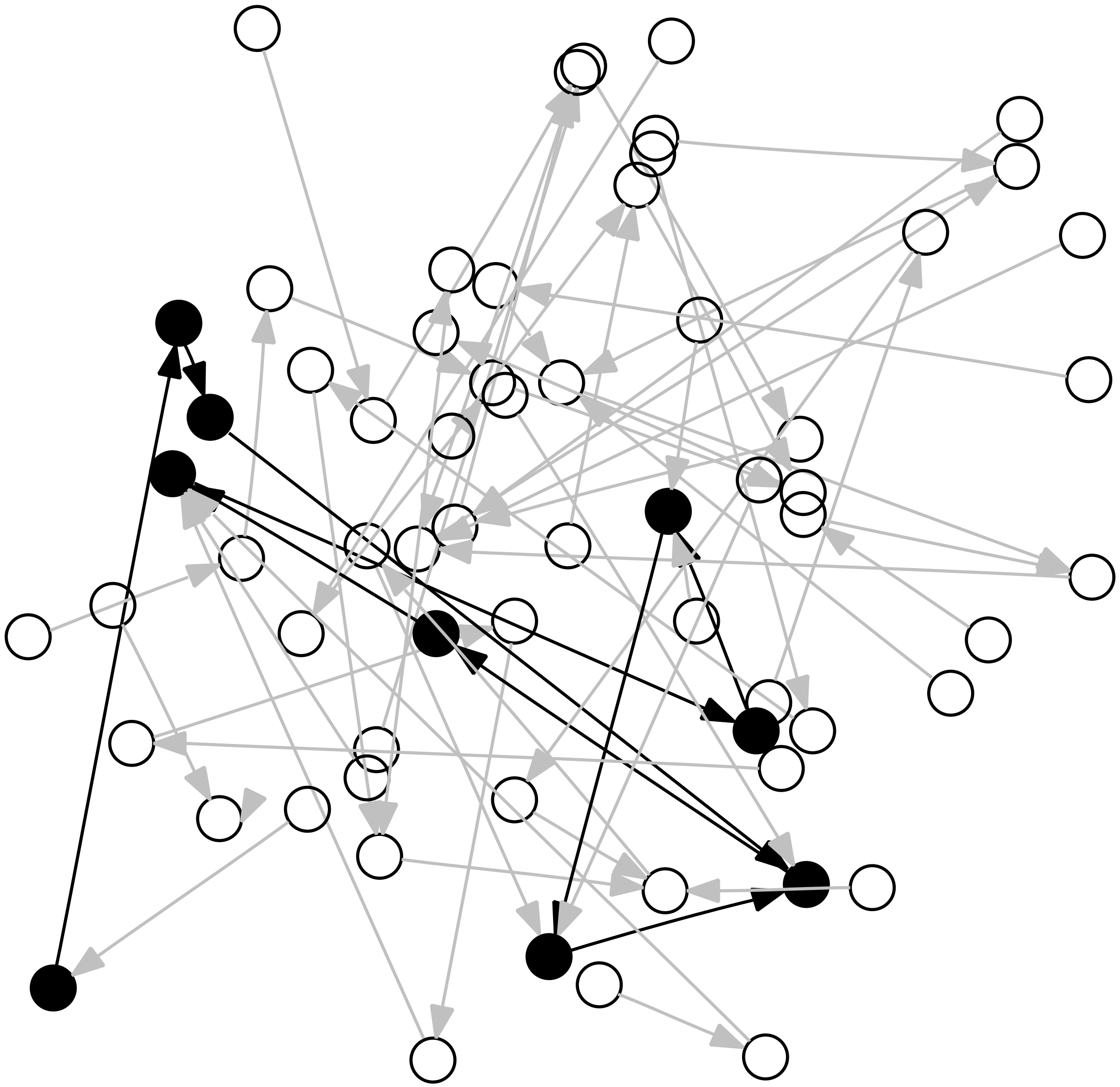
Random walk example



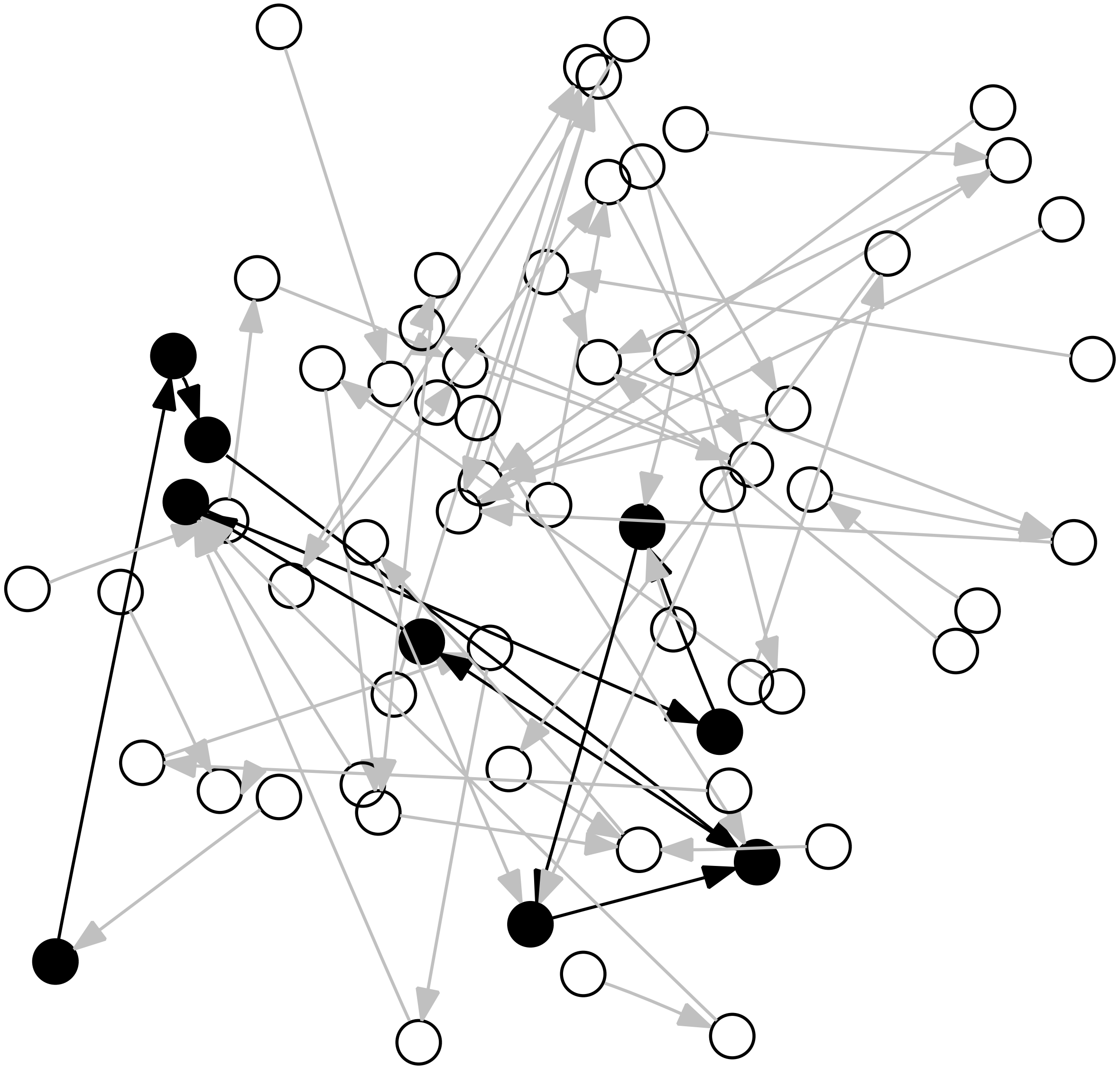
Random walk example



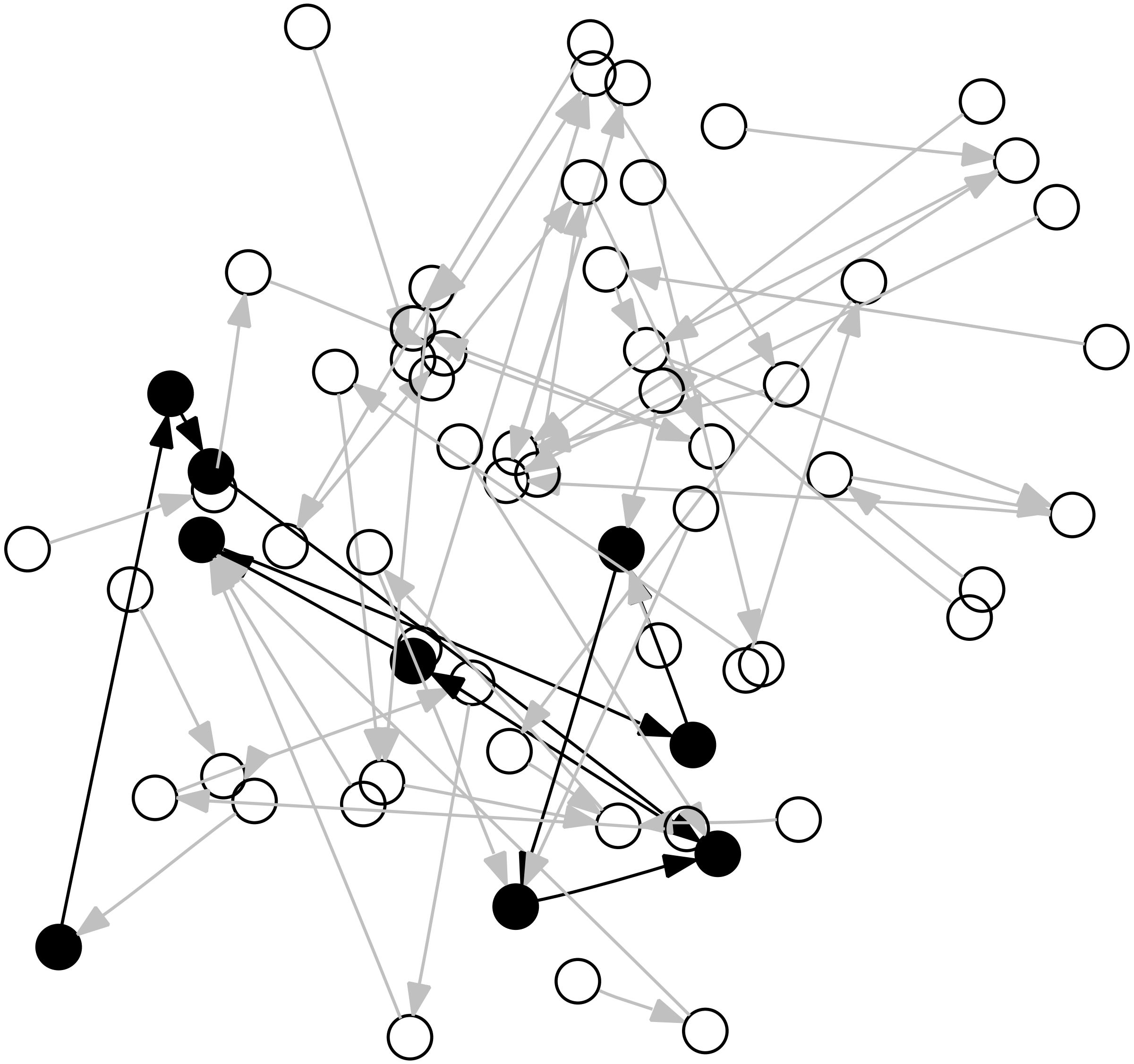
Random walk example



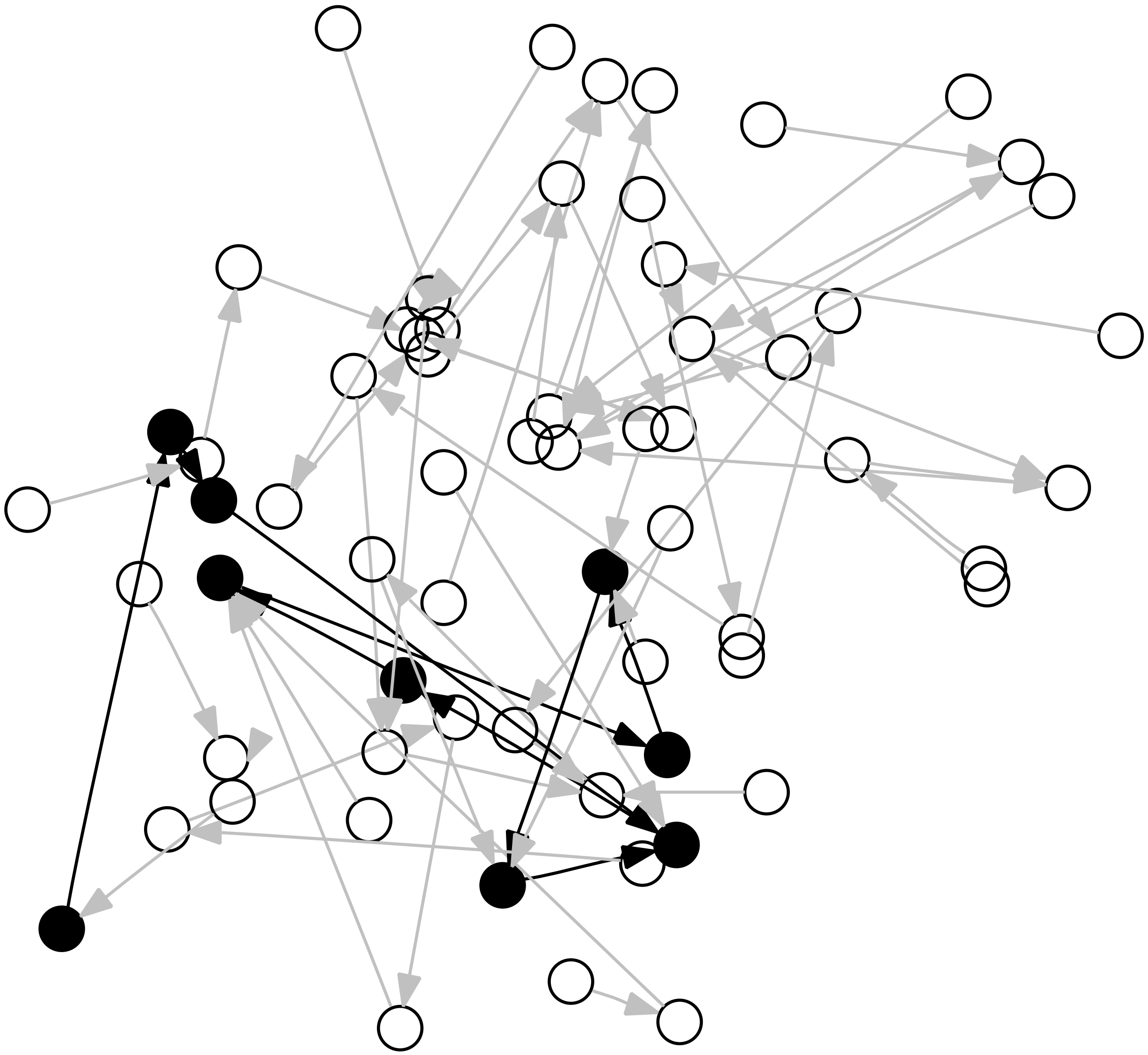
Random walk example



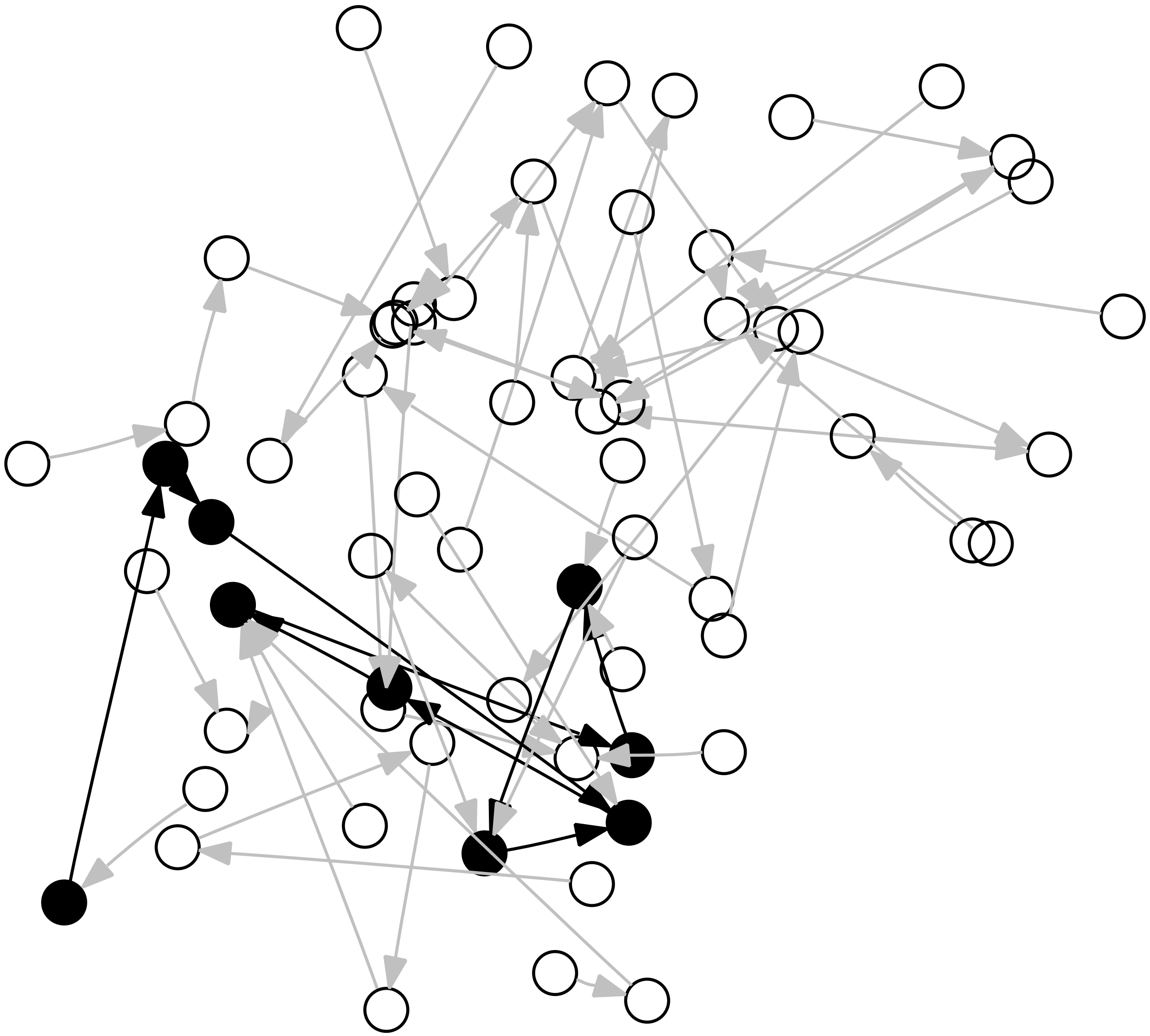
Random walk example



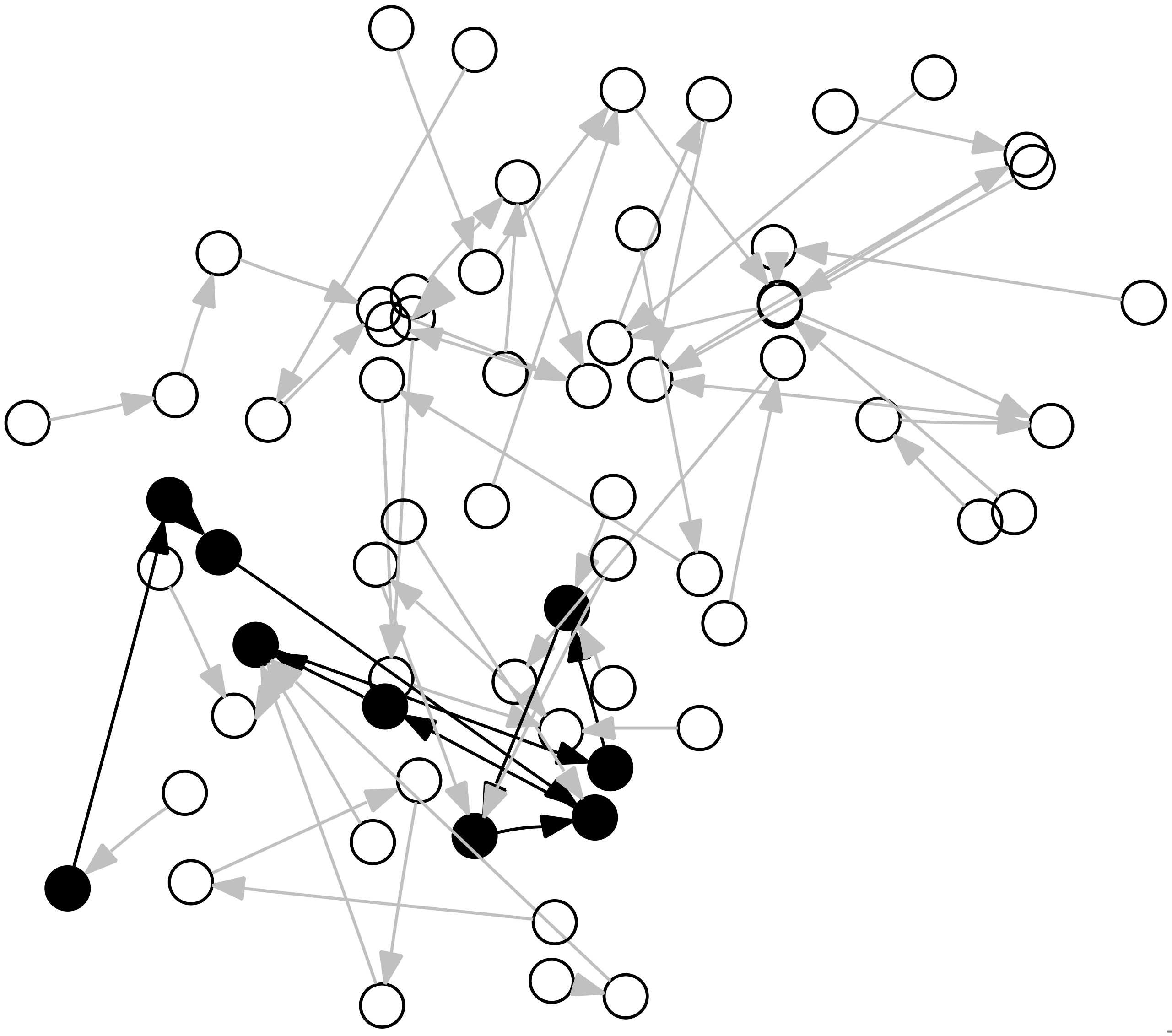
Random walk example



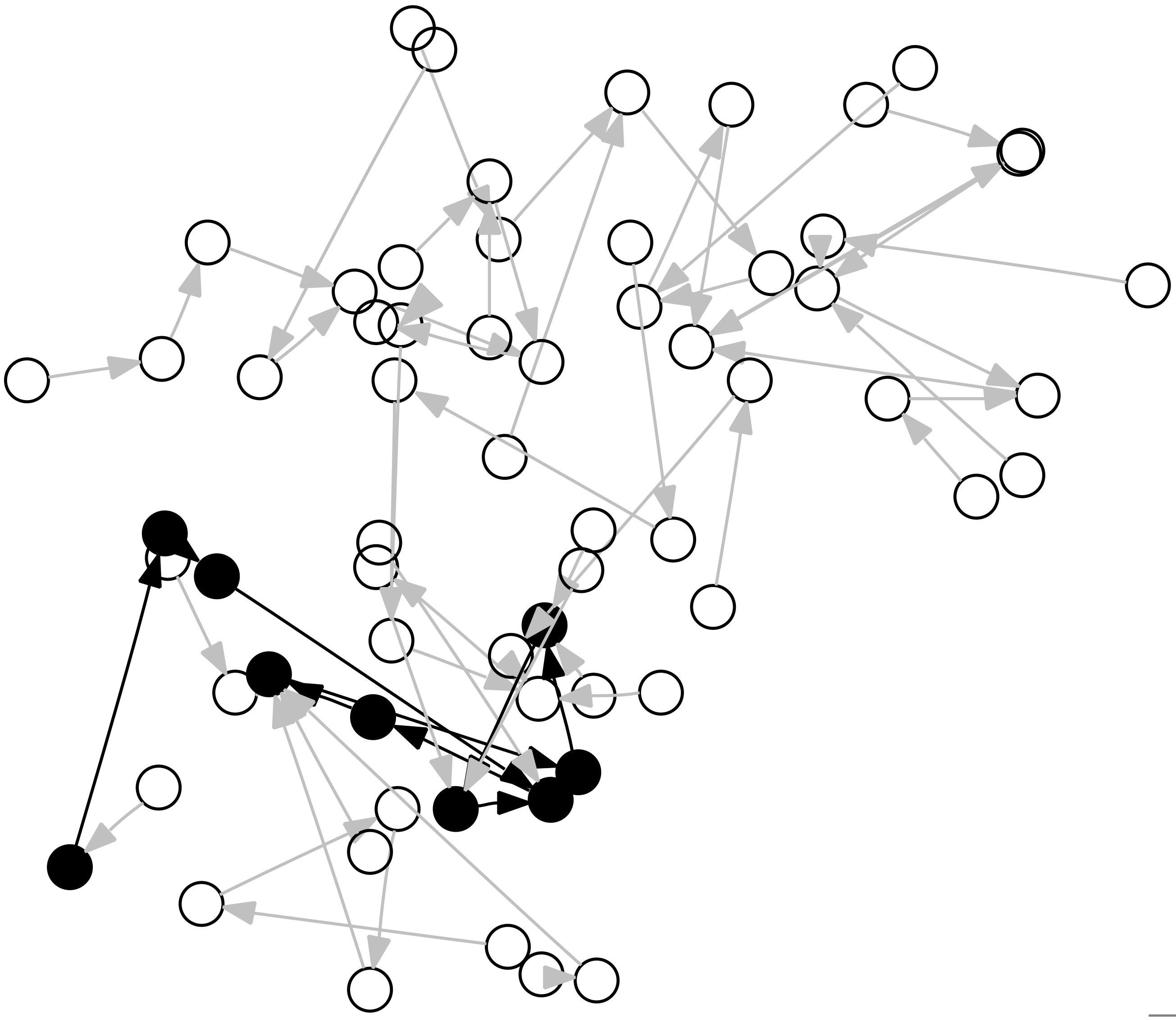
Random walk example



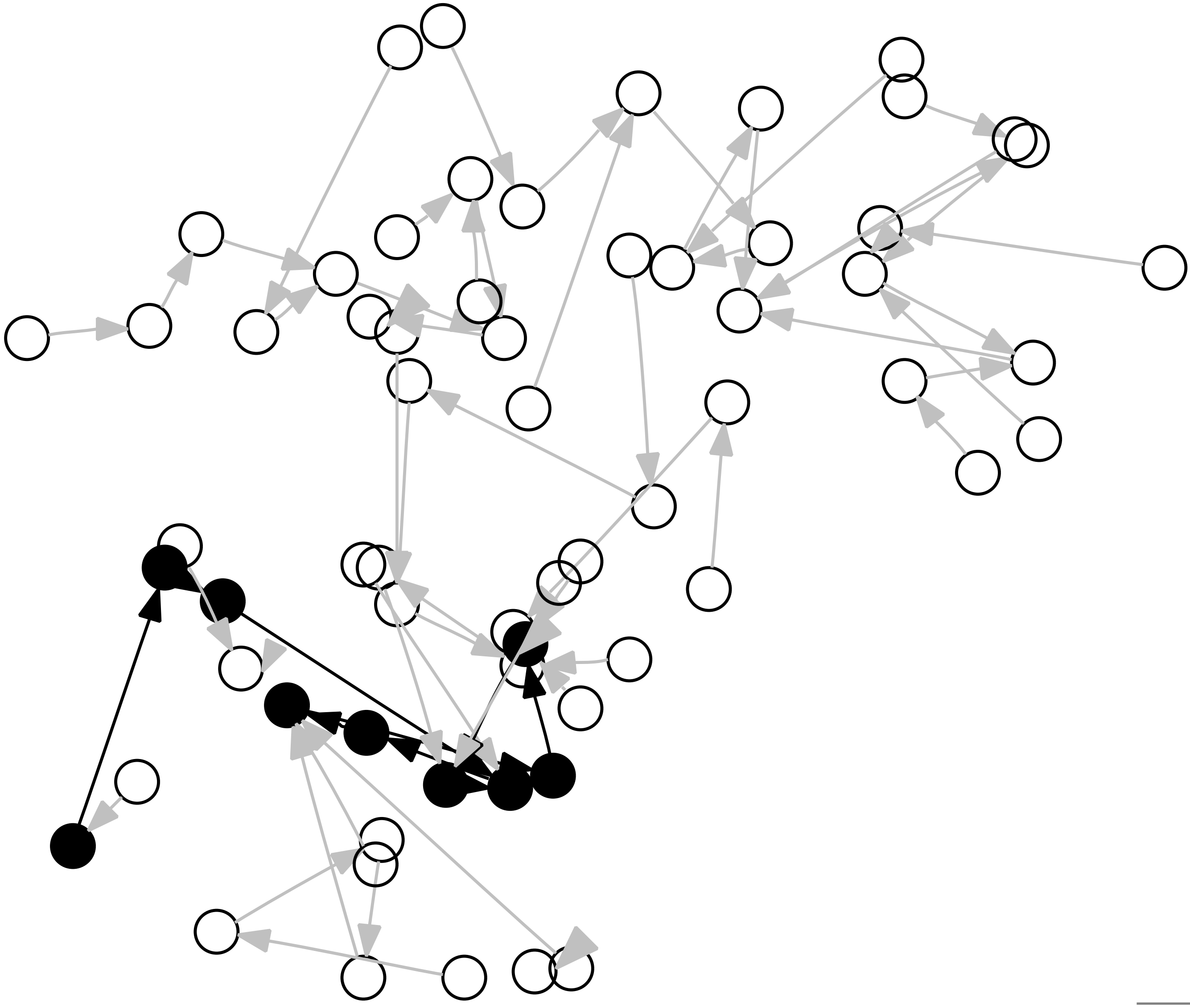
Random walk example



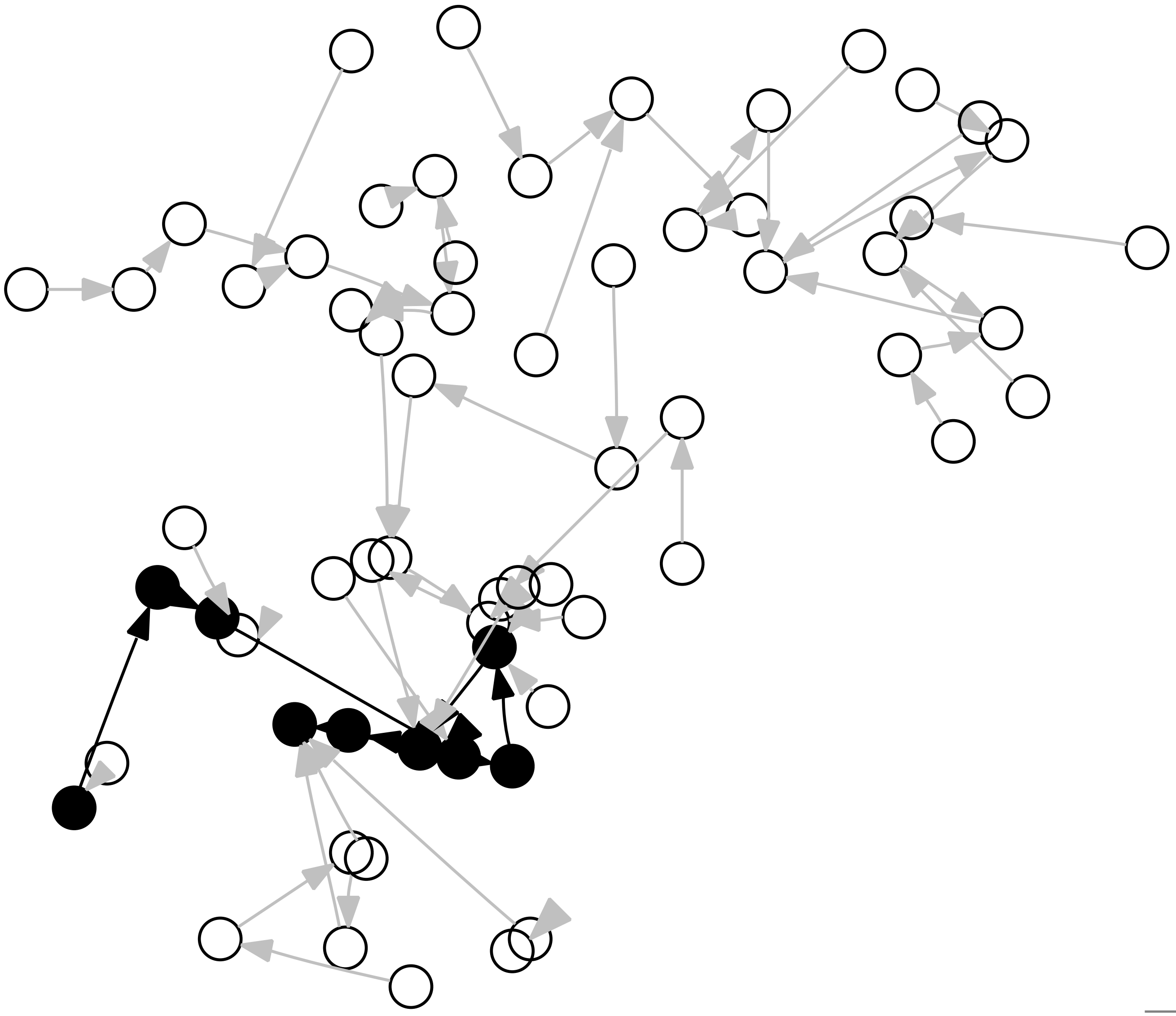
Random walk example



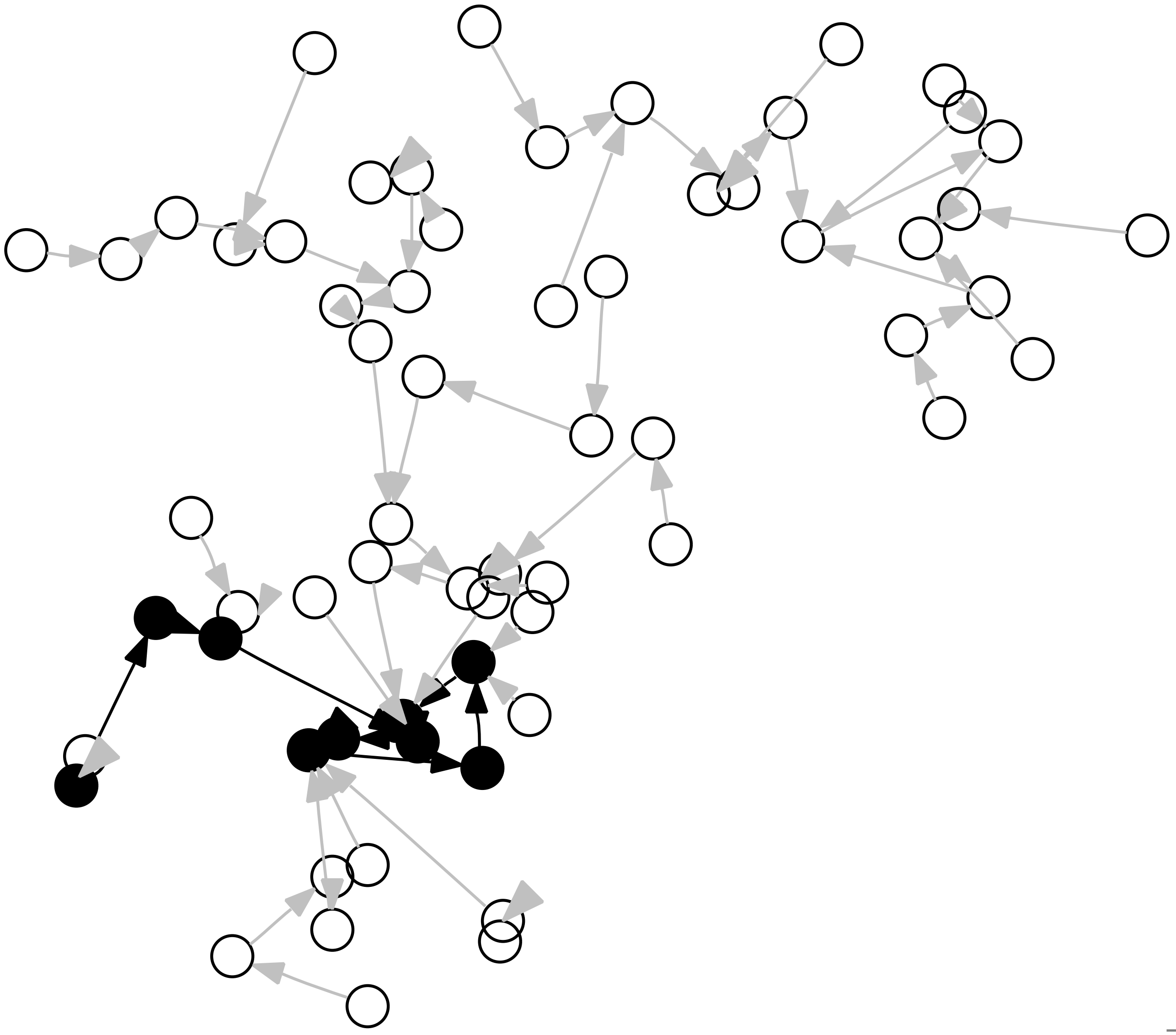
Random walk example



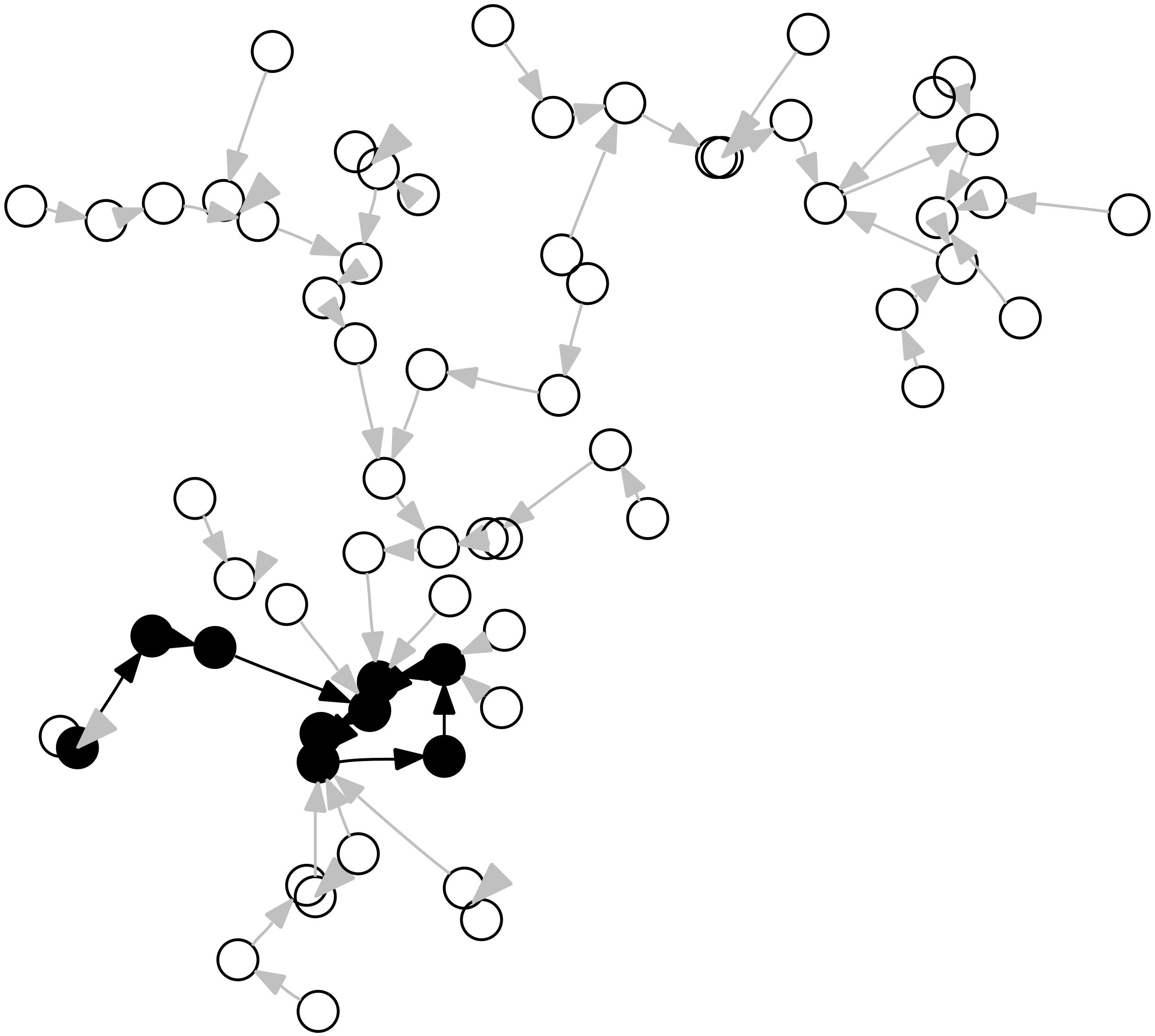
Random walk example



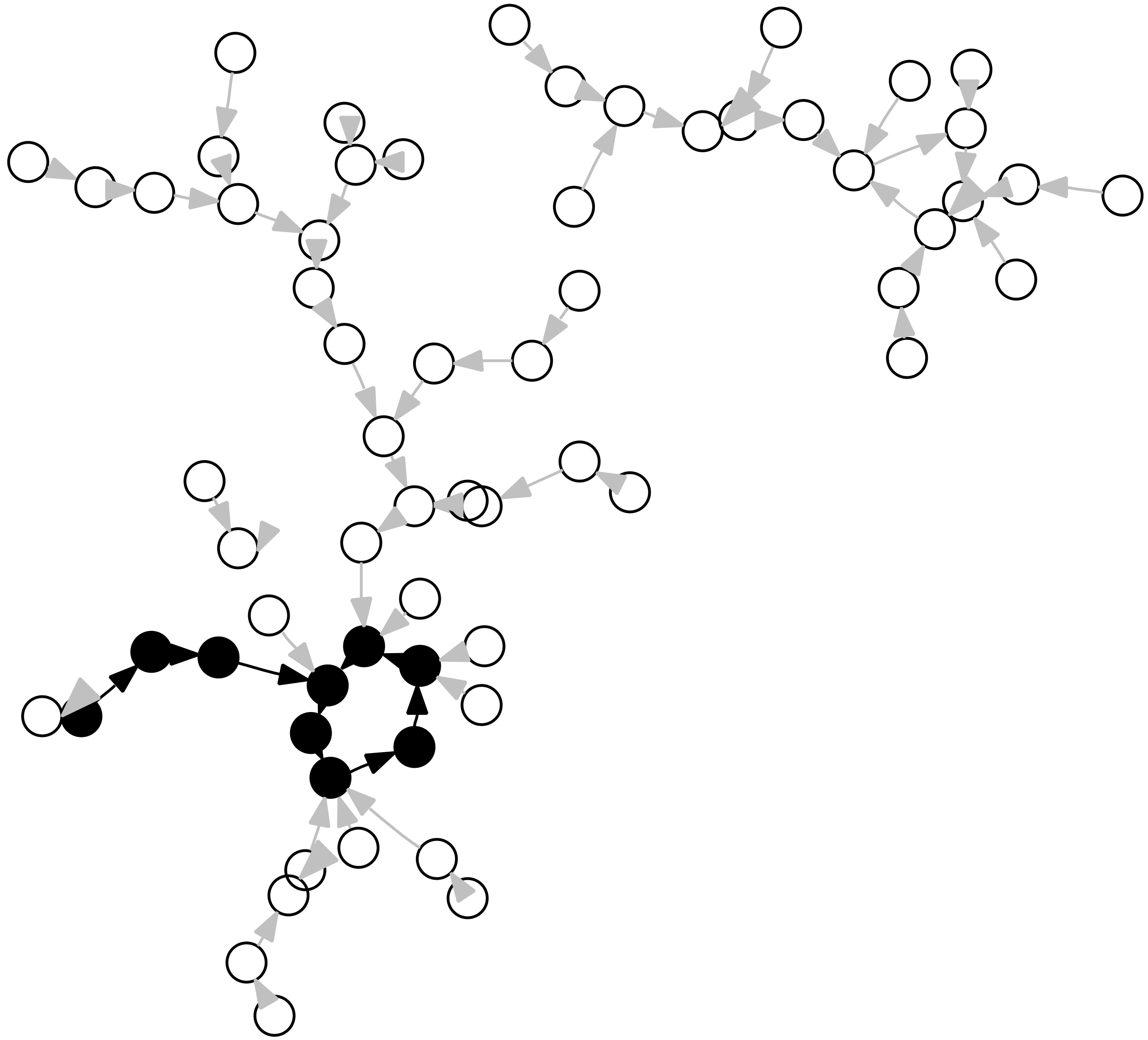
Random walk example



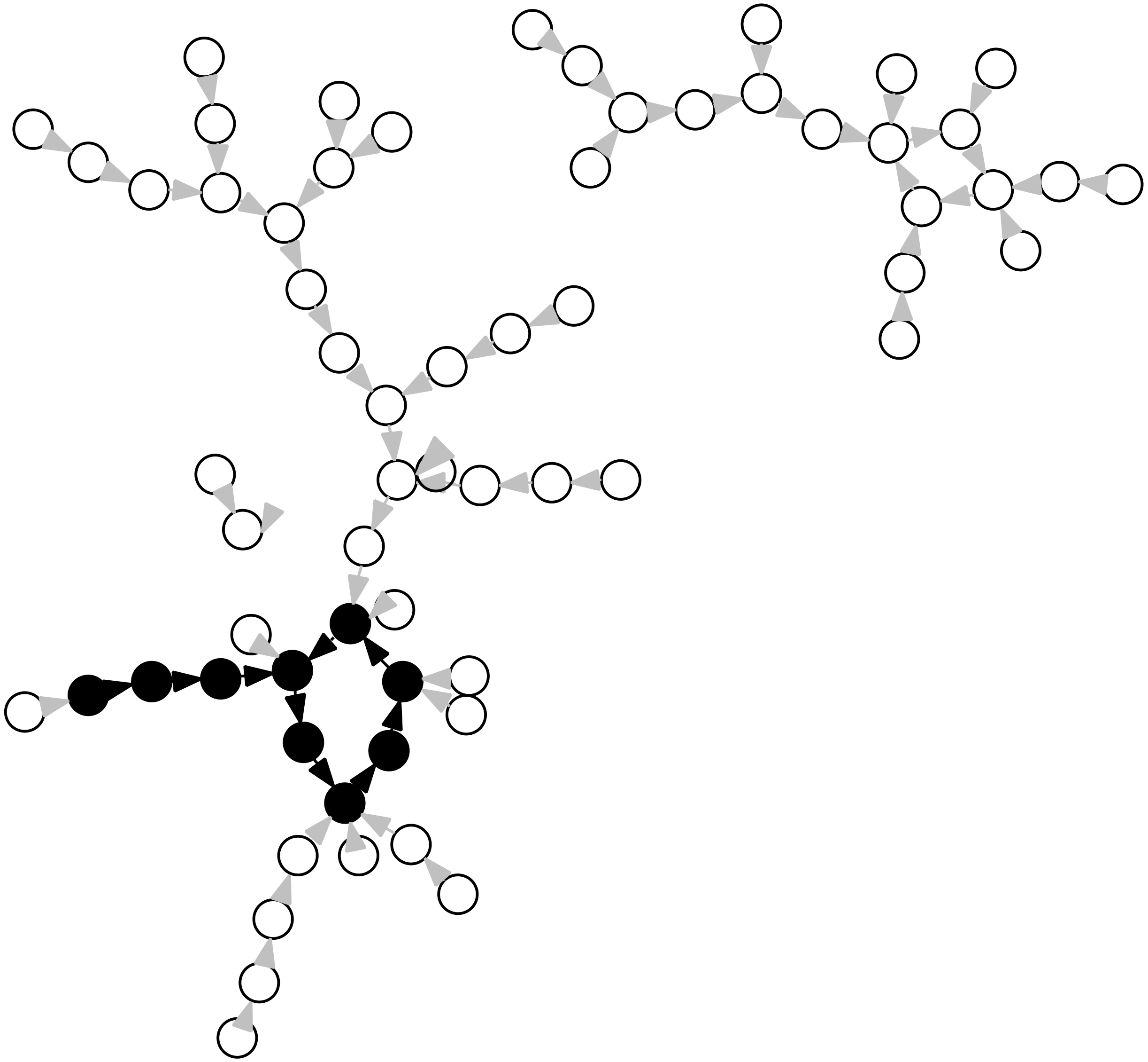
Random walk example



Random walk example



Random walk example



Connection with DLP

- Assume that for each point we know $a_i, b_i \in \mathbb{Z}/\ell\mathbb{Z}$ 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:

$$a = (a_j - a_i)/(b_i - b_j).$$

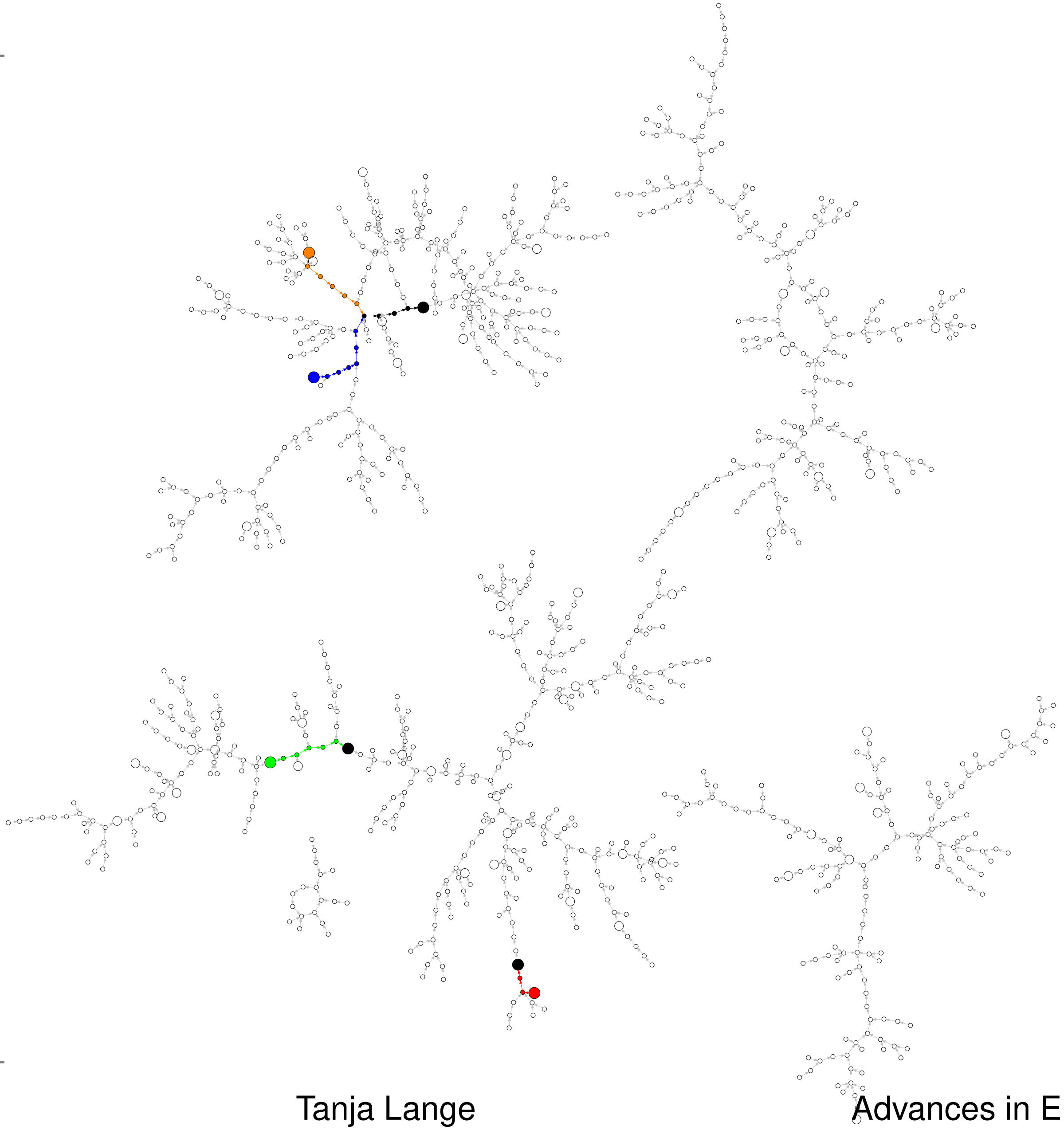
- e.g. additive walk: Start with $W_0 = P$ and put

$$f(W_i) = W_i + c_j P + d_j Q,$$

where $j = h(W_i)$.

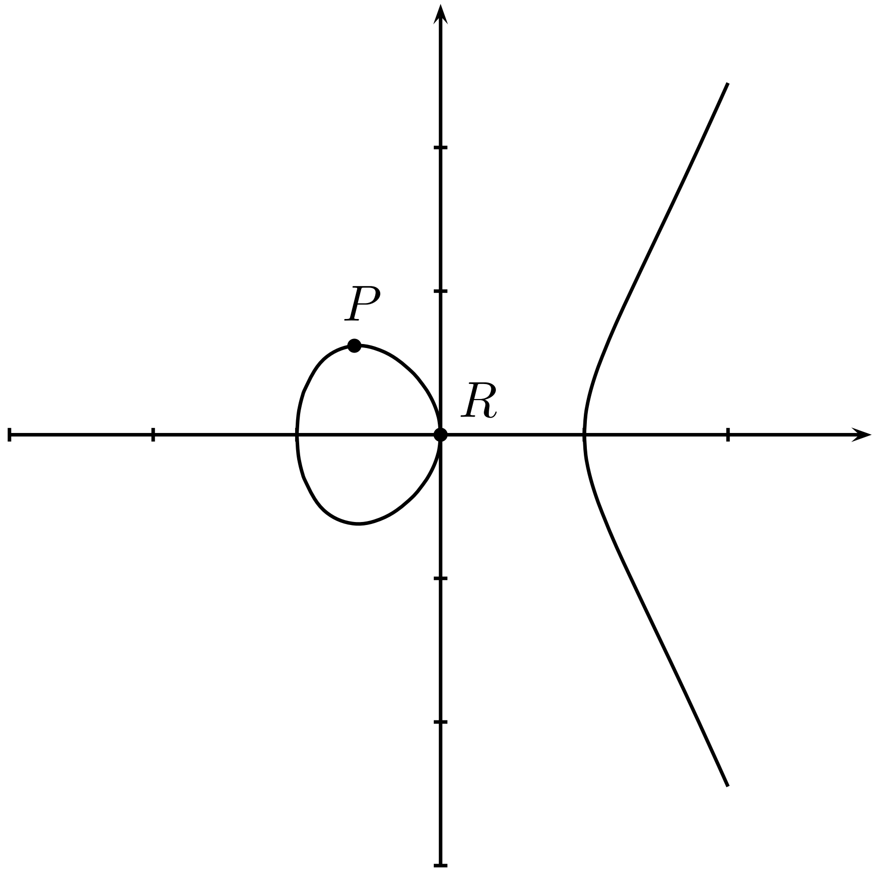
Parallel rho

- 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.



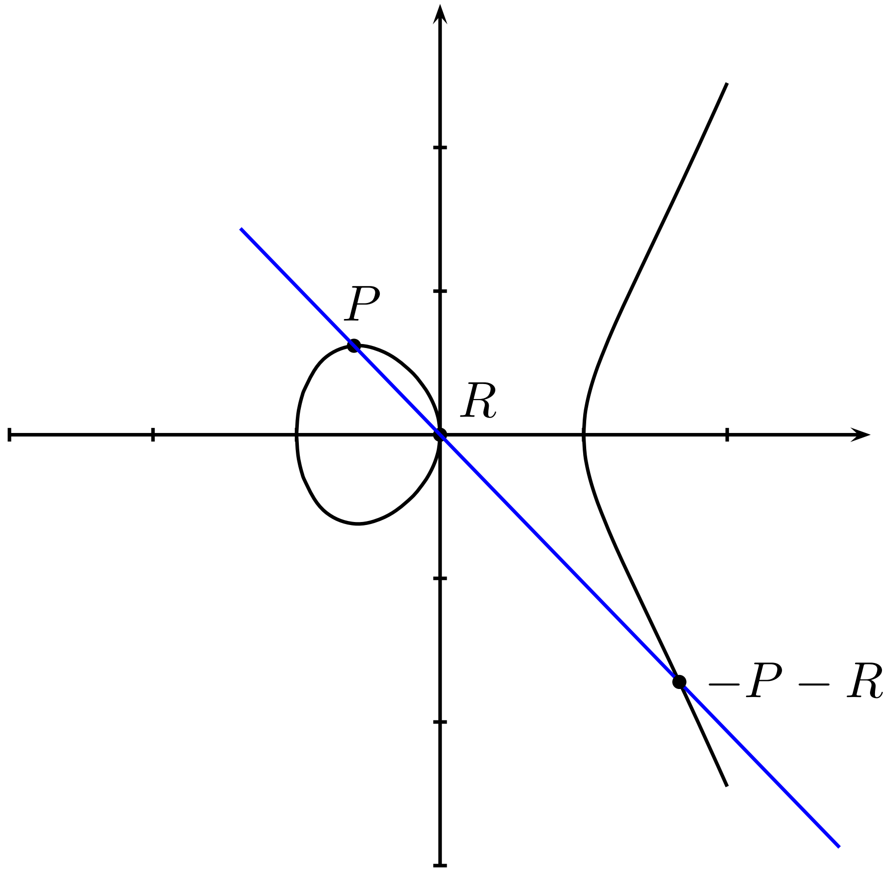
Group law in $E(\mathbb{R})$

$y^2 = x^3 - x$



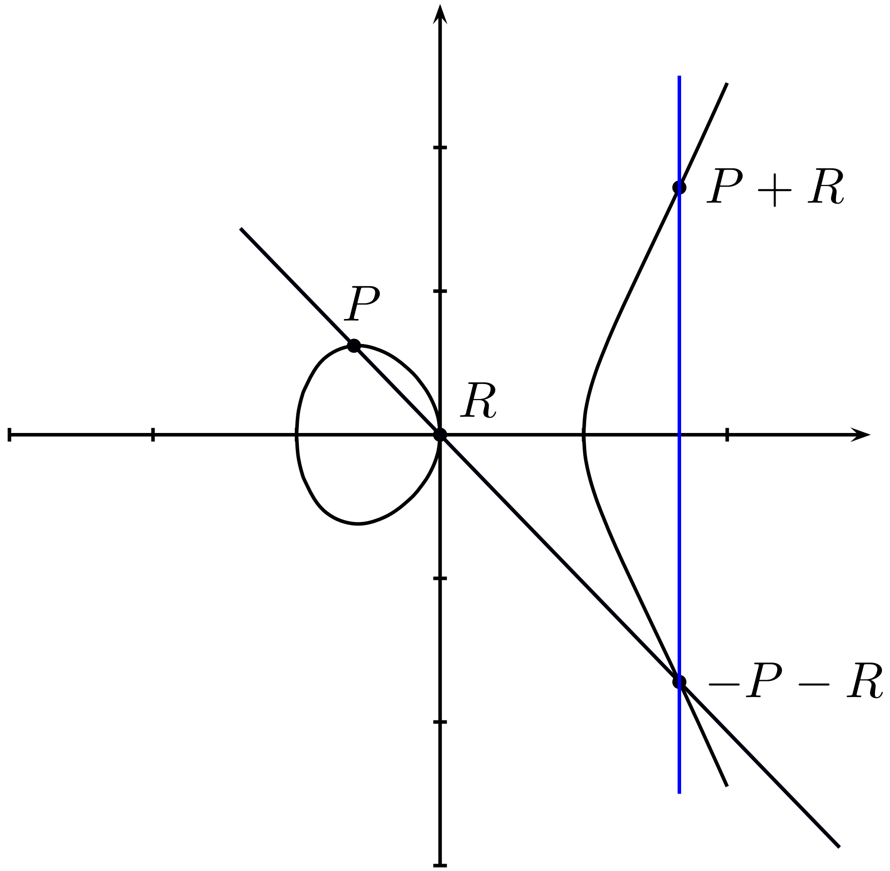
Group law in $E(\mathbb{R})$

$y^2 = x^3 - x$



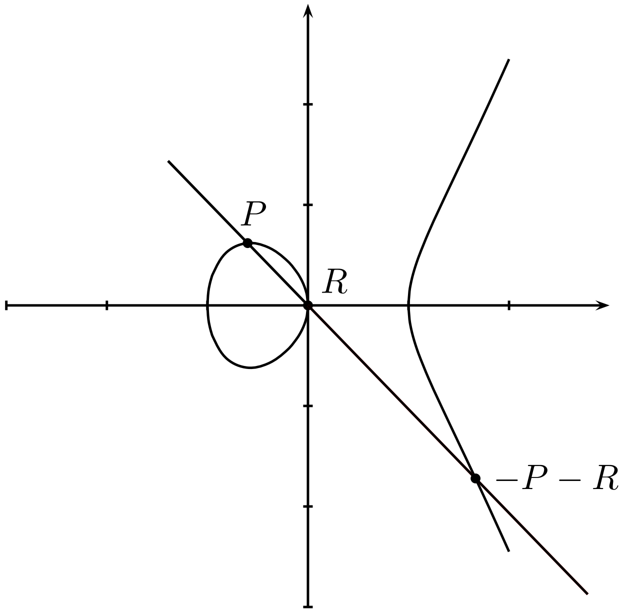
Group law in $E(\mathbb{R})$

$y^2 = x^3 - x$



Group law in formulas

$$E : y^2 = x^3 + a_4 x + a_6,\ a_i \in \mathbb{F}_q$$



Line $y = \lambda x + \mu$ has slope

$$\lambda = \frac{y_R - y_P}{x_R - x_P}.$$

Equating gives

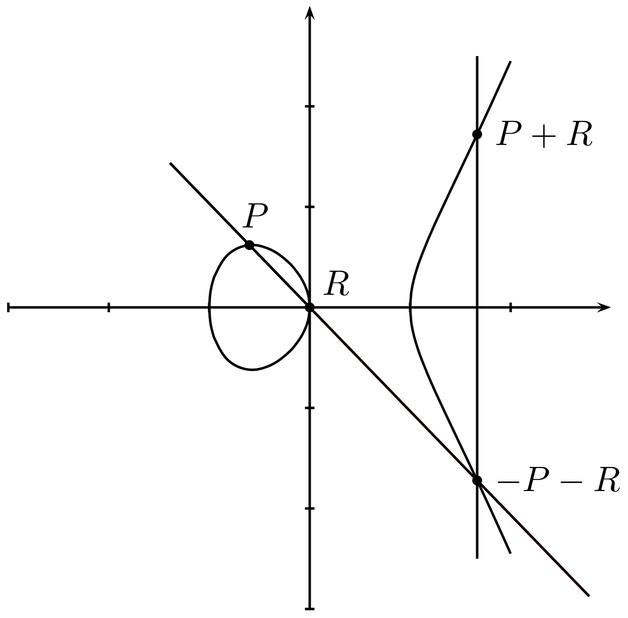
$$(\lambda x + \mu)^2 = x^3 + a_4 x + a_6.$$

This equation has 3 solutions, the x-coordinates of P, R and $-P-R$, thus

$$\begin{aligned}(x - x_P)(x - x_R)(x - x_{-P-R}) &= x^3 - \lambda^2 x^2 + (a_4 - 2\lambda\mu)x + a_6 - \mu^2 \\ x_{-P-R} &= \lambda^2 - x_P - x_R\end{aligned}$$

Group law in formulas

$$E : y^2 = x^3 + a_4x + a_6,\ a_i \in \mathbb{F}_q$$



Point P is on line, thus

$$y_P = \lambda x_P + \mu, \text{ i.e.}$$
$$\mu = y_P - \lambda x_P,$$

and

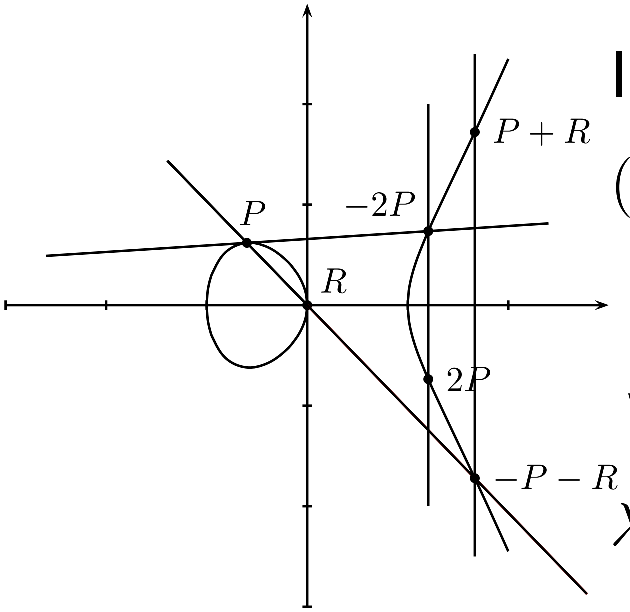
$$\begin{aligned} y_{-P-R} &= \lambda x_{-P-R} + \mu \\ &= \lambda x_{-P-R} + y_P - \lambda x_P \\ &= \lambda(x_{-P-R} - x_P) + y_P \end{aligned}$$

Point $P + R$ has the same x-coordinate but negative y-coordinate:

$$x_{P+R} = \lambda^2 - x_P - x_R, \quad y_{P+R} = \lambda(x_P - x_{P+R}) - y_P$$

Group law in formulas

$$E: y^2 = x^3 + a_4 x + a_6,\ a_i \in \mathbb{F}_q$$



In general, for $(x_P, y_P) \neq (x_R, -y_R)$:

$$\begin{aligned}(x_P, y_P) + (x_R, y_R) &= \\ &= (x_{P+R}, y_{P+R}) = \\ &= (\lambda^2 - x_P - x_R, \lambda(x_P - x_{P+R}) - y_P),\end{aligned}$$

where

$$\lambda = \begin{cases} (y_R - y_P)/(x_R - x_P) & \text{if } x_P \neq x_R, \\ (3x_P^2 + a_4)/(2y_P) & \text{else.} \end{cases}$$

$\Rightarrow$ Addition and Doubling need
1 I, 2M, 1S and 1 I, 2M, 2S, respectively

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 $W_i, -W_i$.

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!
- E.g. if $|W_{i+1}| = -W_{i+1}$ and $h(|W_{i+1}|) = j = h(|W_i|)$ then

$$\begin{aligned} W_{i+2} &= f(W_{i+1}) = -W_{i+1} + c_j P + d_j Q \\ &= -(|W_i| + c_j P + d_j Q) + c_j P + d_j Q = -|W_i|, \end{aligned}$$

so $|W_{i+2}| = |W_i|$, so $W_{i+3} = W_{i+1}$, so $W_{i+4} = W_{i+2}$ etc.

Fruitless cycles

- If h maps to r different values then expect fruitless cycles of length 2 to occur with probability $1/(2r)$ at each step.
- Longer cycles occur less frequently.
- Current ECDL record, achieved 07.2009 by Bos–Kaihara–Kleinjung–Lenstra–Montgomery, "PlayStation 3 computing breaks 2^{60} barrier – 112-bit prime ECDLP solved" breaks standard curve over $\mathbb{F}_p$, 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.)
- Several more papers justifying not using negation map.

Correct use of the negation map

- Our software (Bernstein-L-Schwabe, PKC 2011) solves random ECDL on the same curve (with no precomputation) in 35.6 PS3 years on average.
- For comparison, software from record computation uses 65 PS3 years on average.
- Several levels of speedups, starting with fast arithmetic mod p 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 $(-s, -y)$.
  - $\approx 1.5\%$ of total; no conditional branches.

Experiments

- To confirm iteration effectiveness we have run many experiments on $y^2 = x^3 - 3x + 9$ over the same $\mathbb{F}_p$, using smaller-order P.
- Results matched DL cost predictions.
- Conclusions of this part:
  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 $\sqrt{2}$.
- Lessons
  - Breaking the ECDLP is no harder than textbooks tell you – but that's pretty hard.
  - If a system gets broken it's not the fault of the ECDLP.

Much bigger experiments

- ECC2K-130 is the next unbroken Certicom challenge. The curve is $y^2 + xy = x^3 + 1$ considered over $\mathbb{F}_{2^{131}}$.
- For this curve we can use the negation map and the Frobenius endomorphism; this means we expect to need about $\sqrt{\pi 2^{131}/(2 \cdot 4 \cdot 2 \cdot 131)} \approx 2^{60.9}$ group operations (co-factor is 4, Frobenius has order 131).
- With a group of more than 20 collaborators have written optimized implementations for FPGAs, GPUs, the Cell, several standard CPUs – and we're running the attack.
- This will be the largest cryptanalytic computation publicly performed.
- Get more details, and watch our progress!
  http://ecc-challenge.info
  https://twitter.com/ECCchallenge

How are ECDSA systems broken?

- Anybody can produce an $R = kP$. Alice's private key is only used in $s \equiv k^{-1}(h(m) + r \cdot a) \bmod \ell$.
- Can fake signatures if one can break the DLP, i.e., if one can compute a from P_A. ...sometimes attacks are easier...

How are ECDSA systems broken?

- Anybody can produce an $R = kP$. Alice's private key is only used in $s \equiv k^{-1}(h(m) + r \cdot a) \bmod \ell$.
- Can fake signatures if one can break the DLP, i.e., if one can compute a from P_A. ...sometimes attacks are easier...
- If k is known for some $m, (r, s)$ then $a \equiv (sk - h(m))/r \bmod \ell$.

How are ECDSA systems broken?

- Anybody can produce an $R = kP$. Alice's private key is only used in $s \equiv k^{-1}(h(m) + r \cdot a) \bmod \ell$.
- Can fake signatures if one can break the DLP, i.e., if one can compute a from P_A. …sometimes attacks are easier…
- If k is known for some $m, (r, s)$ then $a \equiv (sk - h(m))/r \bmod \ell$.
- If two signatures $m_1, (r, s_1)$ and $m_2, (r, s_2)$ have the same value for r: assume $k_1 = k_2$; observe $s_1 - s_2 = k_1^{-1}(h(m_1) + ra - (h(m_2) + ra))$; compute $k = (s_1 - s_2)/(h(m_1) - h(m_2))$. Continue as above.

How are ECDSA systems broken?

- Anybody can produce an $R = kP$. Alice's private key is only used in $s \equiv k^{-1}(h(m) + r \cdot a) \bmod \ell$.
- Can fake signatures if one can break the DLP, i.e., if one can compute a from P_A. …sometimes attacks are easier…
- If k is known for some $m, (r, s)$ then $a \equiv (sk - h(m))/r \bmod \ell$.
- If two signatures $m_1, (r, s_1)$ and $m_2, (r, s_2)$ have the same value for r: assume $k_1 = k_2$; observe $s_1 - s_2 = k_1^{-1}(h(m_1) + ra - (h(m_2) + ra))$; compute $k = (s_1 - s_2)/(h(m_1) - h(m_2))$. Continue as above.
- If bits of many k's are known (biased PRNG) can attack $s \equiv k^{-1}(h(m) + r \cdot a) \bmod \ell$ as hidden number problem using lattice basis reduction.

Avoid trouble and be faster

aka "High-speed high-security signatures"
(Bernstein–Duif–L–Schwabe–Yang, CHES 2011).

- Fast signature verification. Intel Nehalem/Westmere CPUs: 280880 cycles for verification. (official eBATS measurement.)
- Even faster batch verification: < 134000 cycles/signature to verify 64 signatures of 64 messages under 64 public keys. 4×2.4GHz: 71000 verif/second!
- Very fast signing. 88328 cycles for signing. (eBATS measurement.) 4×2.4GHz: 108000 signs/second!
- Fast key generation. Almost as fast as signing.
- High security level > 2^{128} (without quantum computers).
- Small signatures and keys.

Bonus features

- No secret array indices. No information flow from secret data to addresses. $\Rightarrow$ No cache-timing attacks.
- No secret branch conditions. No information flow from secret data to branch unit.
- Collision resilience. Hash collisions do not break this signature system.
- Foolproof session keys. Signing is deterministic.
- Paper contains new representation of points which is faster for adding precomputed points.
- Curve is an Edwards curve, so it looks nicer :–)
- New efficient point compression and decompression for Edwards curves.

Some details

- Secret key of user produces a used in $P_A = aP$ and bitstring b.
- Nonce k is computed as $k =$hash(b, m).
- $R = kP$ as before but h is made dependent on R and P_A as $h =$hash(R, P_A, m).
- Signing and verification avoid inversions $s \equiv k + ha \bmod \ell$.
- Verification: $sP \stackrel{?}{=} R + hP_A$.
- Batch verification of (R_i, s_i) with $h_i =$hash(R_i, P_{A_i}, m_i) picks random scalars z_i and checks whether

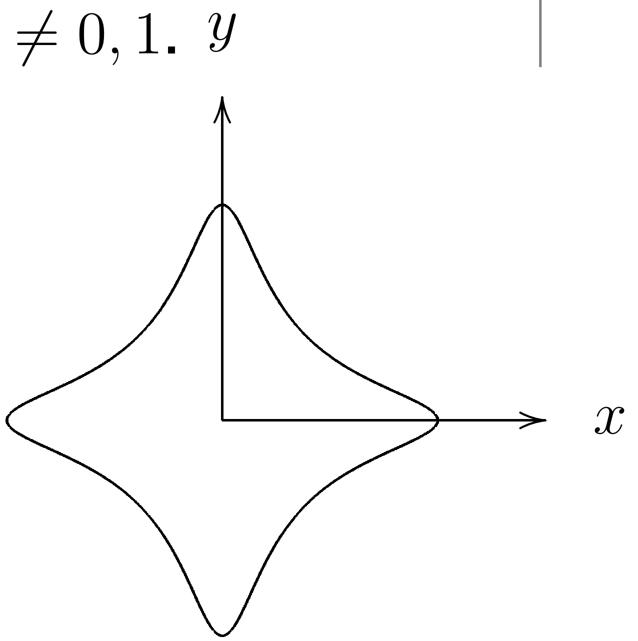
$$(-\sum_i z_i s_i \bmod \ell)P + \sum_i z_i R_i + \sum_i (z_i h_i \bmod \ell) P_{A_i} = 0.$$

Thank you for your attention!

Addition on Edwards curves

Let K be a field with $2 \neq 0$. Let $d \in K$ with $d \neq 0, 1$.
Edwards curve (nice form of elliptic curve):

$$\{(x, y) \in K \times K | x^2 + y^2 = 1 + dx^2y^2\}$$



Harold M. Edwards,
(Bulletin of the AMS, **44**, 393–422, 2007)

Associative operation on points
$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$
defined by Edwards addition law

$$x_3 = \frac{x_1y_2 + y_1x_2}{1 + dx_1x_2y_1y_2} \text{ and } y_3 = \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2}.$$

- Neutral element is $(0, 1)$.
- $-(x_1, y_1) = (-x_1, y_1)$.

Explicit formulas: addition

- $(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_2}, \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_2} \right).$
- Avoid inversions: Use $(X_1 : Y_1 : Z_1)$ with $Z_1 \neq 0$ to represent $(x_1, y_1) = (X_1/Z_1, Y_1/Z_1)$, i. e., $(X_1 : Y_1 : Z_1) = (\lambda X_1 : \lambda Y_1 : \lambda Z_1)$ for $\lambda \neq 0$.
- Addition formulas in projective coordinates:

$$\begin{aligned} A &= Z_1 \cdot Z_2;\; B = A^2;\; C = X_1 \cdot X_2;\; D = Y_1 \cdot Y_2; \\ E &= d \cdot C \cdot D;\; F = B - E;\; G = B + E; \\ X_3 &= A \cdot F \cdot ((X_1 + Y_1) \cdot (X_2 + Y_2) - C - D); \\ Y_3 &= A \cdot G \cdot (D - C);\; Z_3 = F \cdot G. \end{aligned}$$

- Needs 10M + 1S + 1D + 7A, i.e. 10 multiplications, 1 squaring, 1 multiplication involving d, and 7 additions.
- Extended coordinates $(X : Y : Z : XY/Z)$

More benefits

- Doubling formulas in projective coordinates:

$$\begin{aligned} B &= (X_1+Y_1)^2;\ C = X_1^2;\ D = Y_1^2; \\ E &= C+D;\ H = Z_1^2;\ J = E-2H; \\ X_3 &= (B-E)\cdot J;\ Y_3 = E\cdot(C-D);\ Z_3 = E\cdot J. \end{aligned}$$

- Needs 3M + 4S + 6A.
- Does not benefit from extended coordinates, so output XY/Z only if next operation is addition; skip this coordinate as output of addition if followed by doubling.
- Precomputed points get stored as $(Y-X, Y+X, 2dXY)$; this corresponds to affine points $Z=1$ and saves the operations that depend only on one input point.

Beyond speed

- Neutral element of addition law is affine point.
- Addition law is symmetric in both inputs.
- $P+Q = \left(\frac{x_1y_2+y_1x_2}{1+dx_1x_2y_1y_2}, \frac{y_1y_2-x_1x_2}{1-dx_1x_2y_1y_2}\right).$
- $[2]P = \left(\frac{x_1y_1+y_1x_1}{1+dx_1x_1y_1y_1}, \frac{y_1y_1-x_1x_1}{1-dx_1x_1y_1y_1}\right).$
  Can compute this using the same formulas as general addition.
- If d is not a square in K the denominators $1+dx_1x_2y_1y_2$ and $1-dx_1x_2y_1y_2$ are never 0; addition law is complete.
- Having addition law work for doubling removes some checks from the code; addition law also works for adding $P+(-P)$ or the neutral element.