Discrete logarithm problem V

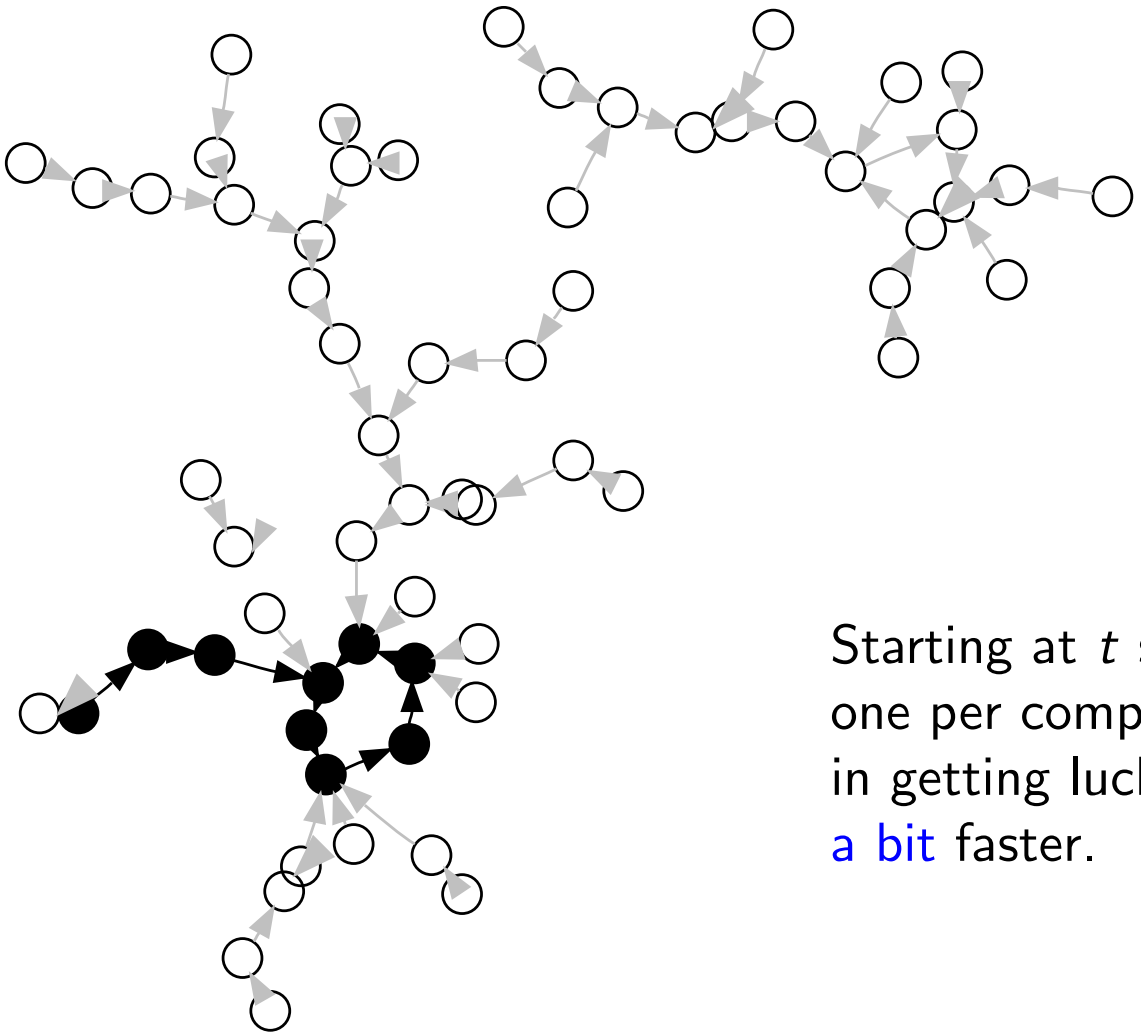
Paralel collision search

Tanja Lange

Eindhoven University of Technology

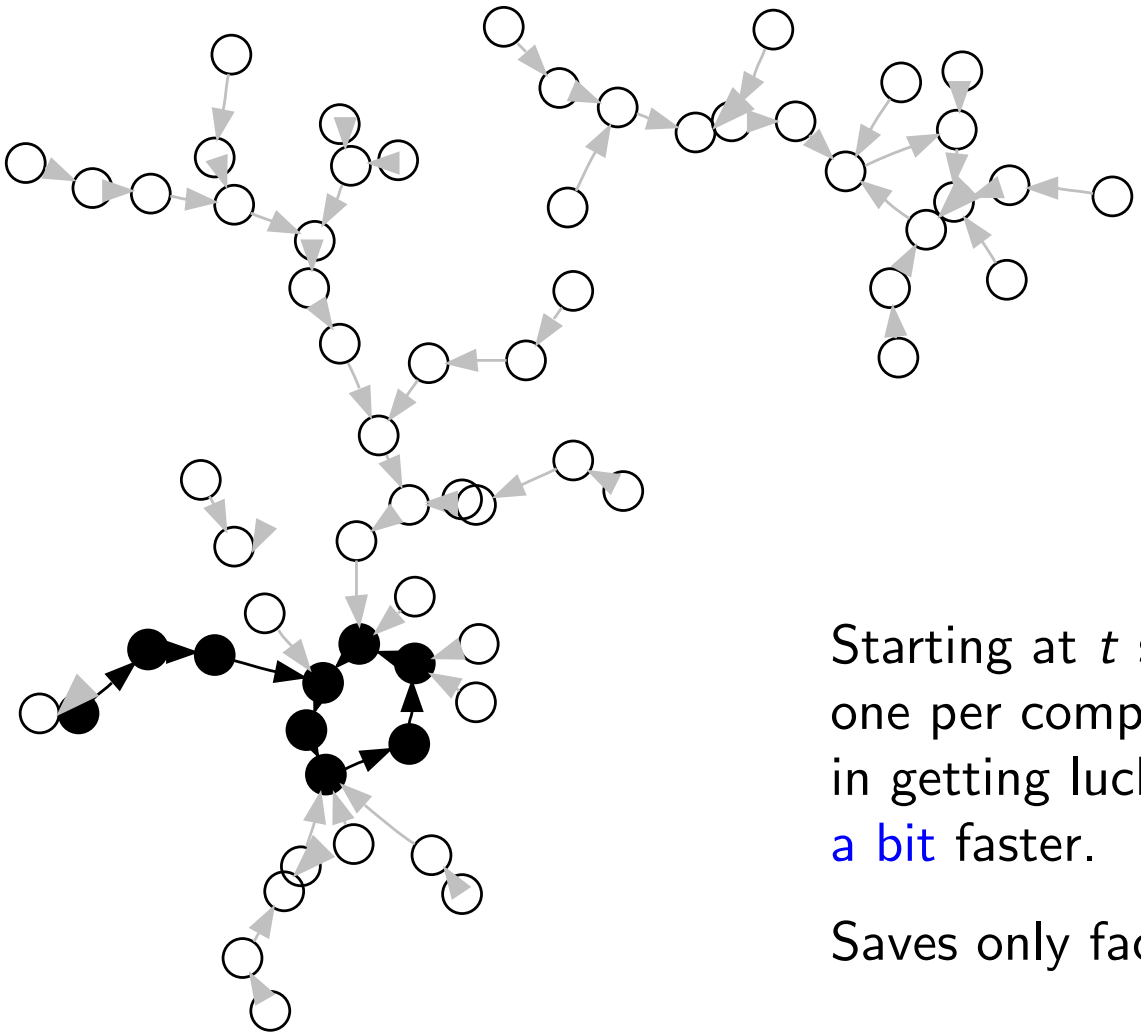
2MMC10 – Cryptology

How to use more than one computer efficiently?



Starting at t starting points,
one per computer, helps
in getting lucky
a bit faster.

How to use more than one computer efficiently?



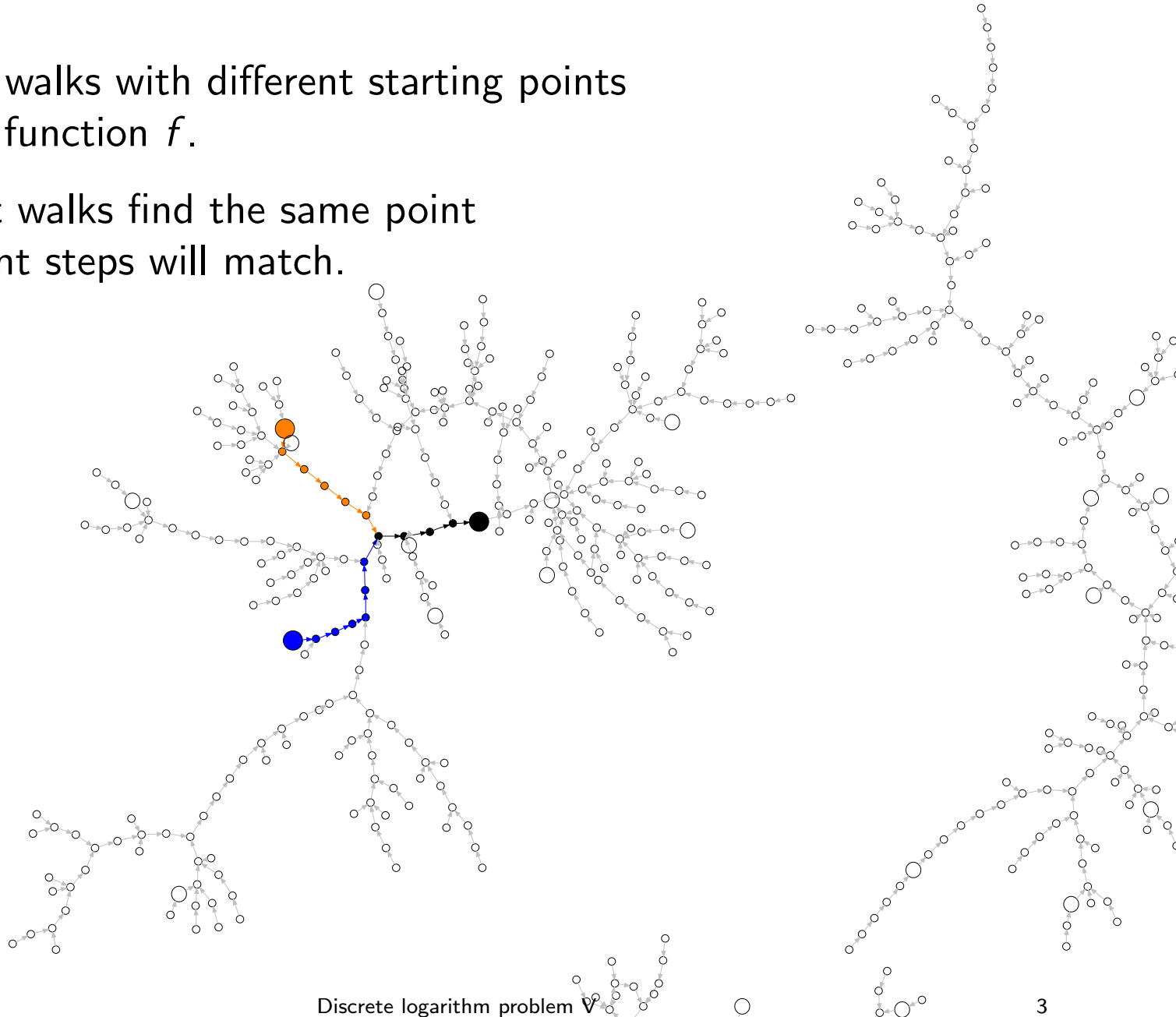
Starting at t starting points, one per computer, helps in getting lucky a bit faster.

Saves only factor of $\sqrt{t}$.

Van Oorschot and Wiener parallel collision search

Perform many walks with different starting points but same step function f.

If two different walks find the same point their subsequent steps will match.

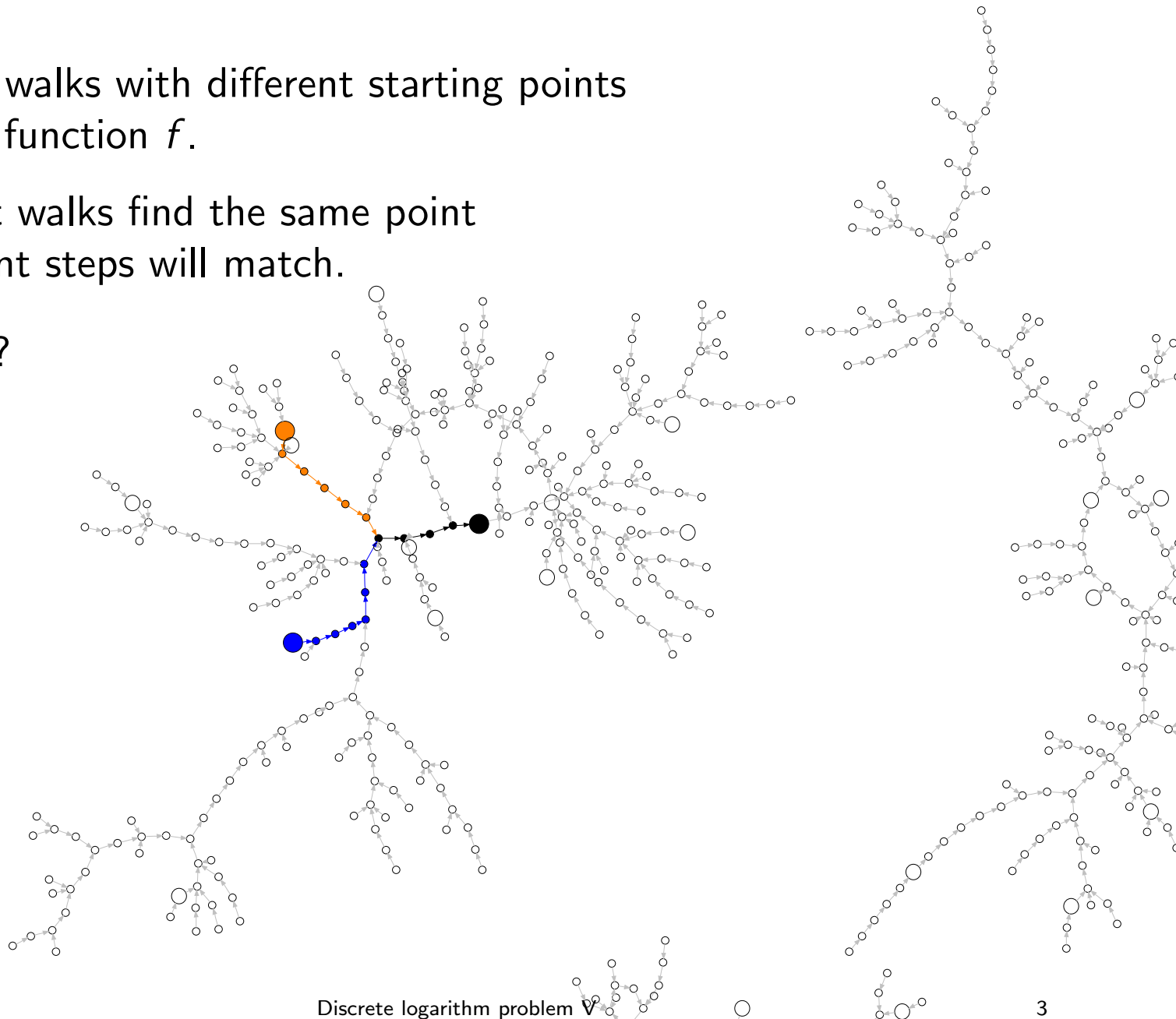


Van Oorschot and Wiener parallel collision search

Perform many walks with different starting points but same step function f.

If two different walks find the same point their subsequent steps will match.

How to notice?

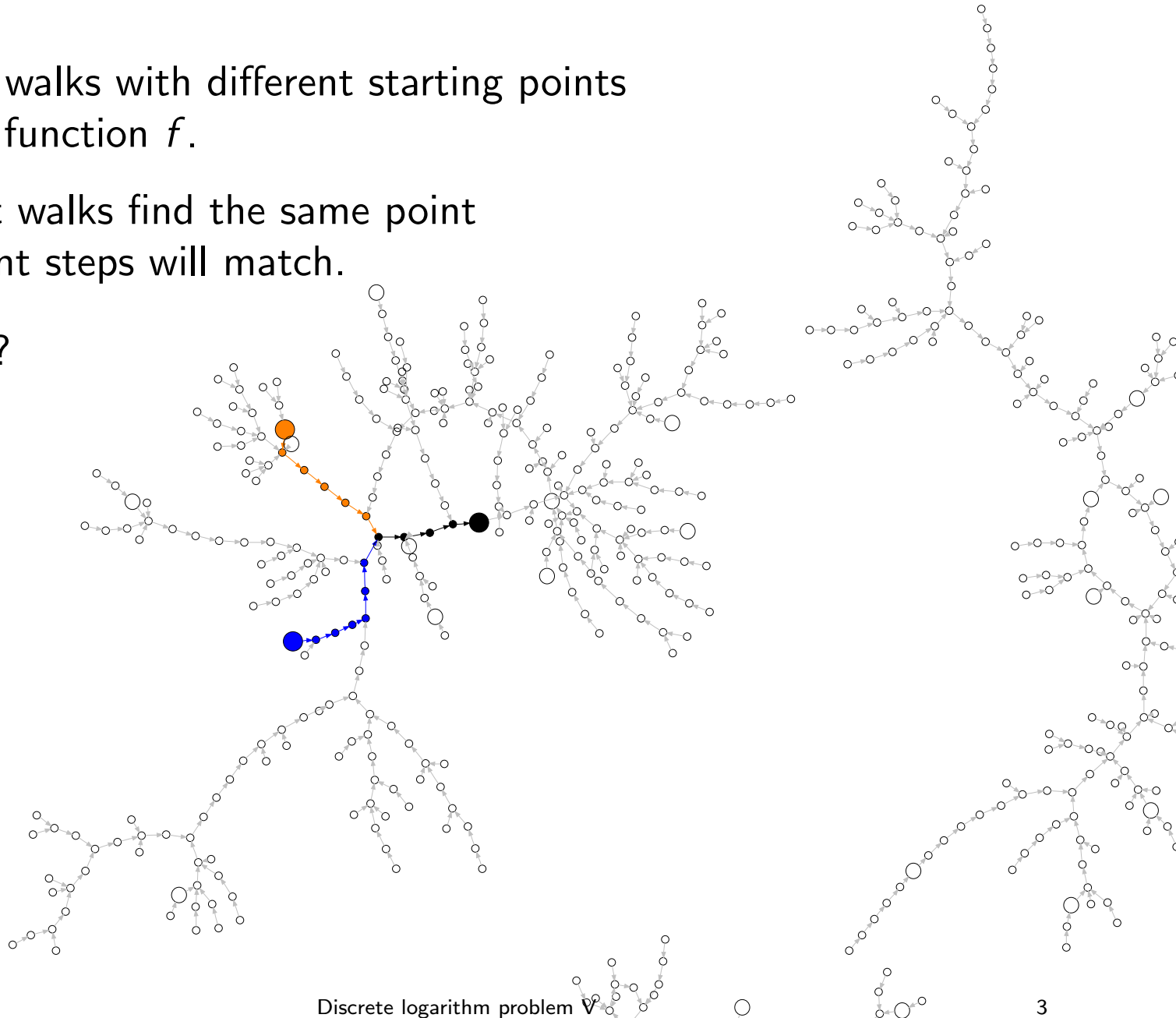


Van Oorschot and Wiener parallel collision search

Perform many walks with different starting points but same step function f.

If two different walks find the same point their subsequent steps will match.

How to notice?
when to stop?



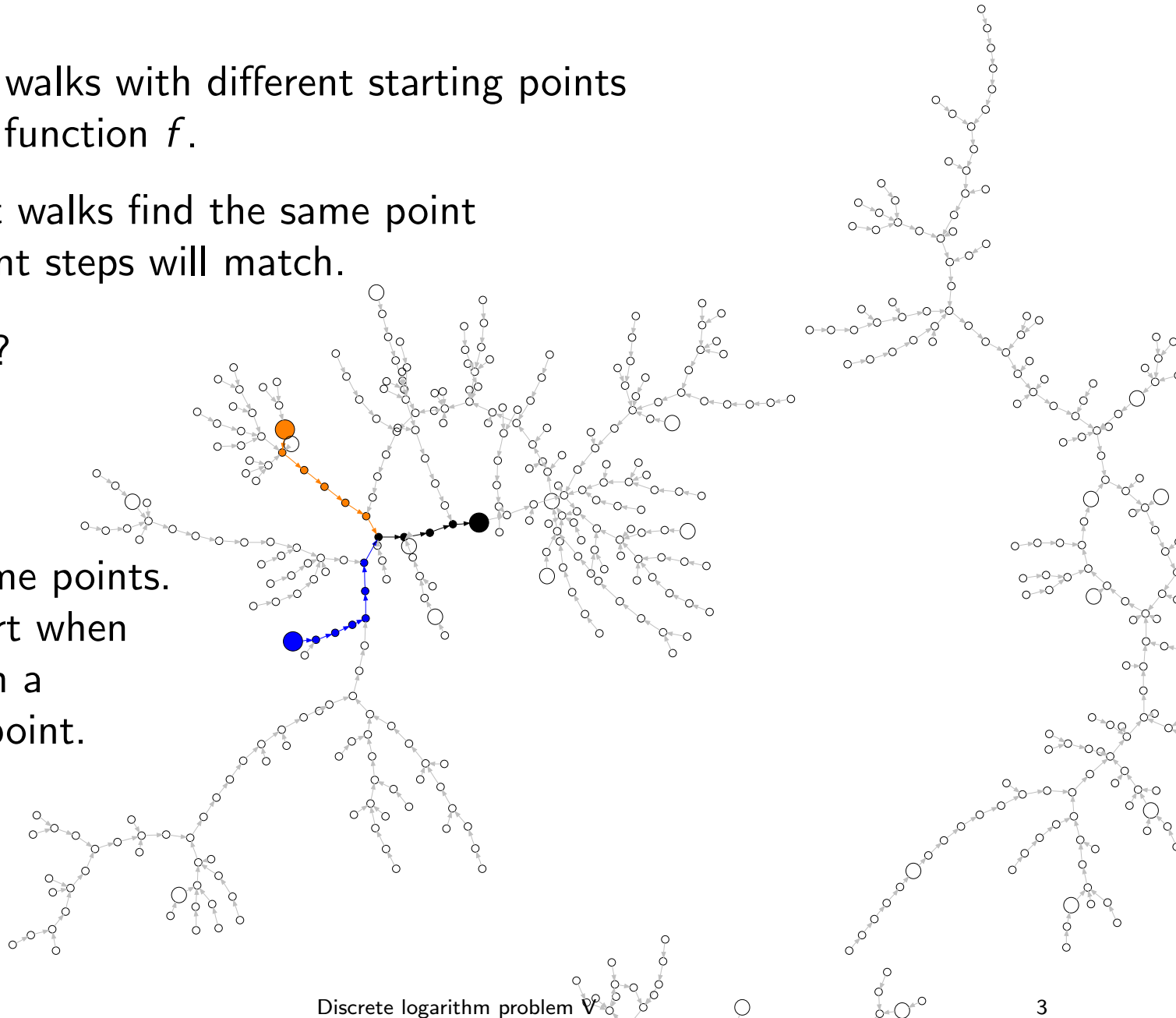
Van Oorschot and Wiener parallel collision search

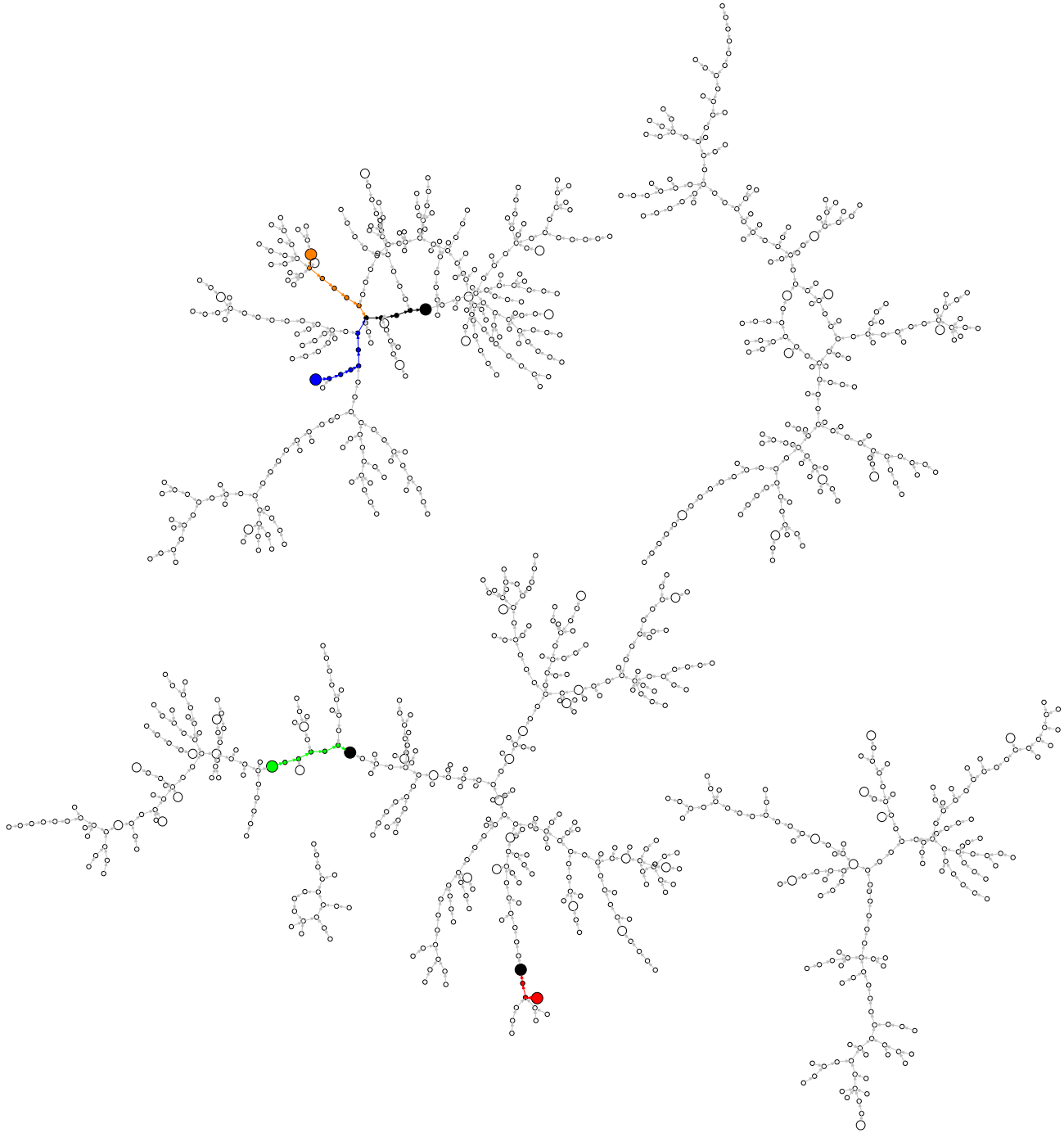
Perform many walks with different starting points but same step function f.

If two different walks find the same point their subsequent steps will match.

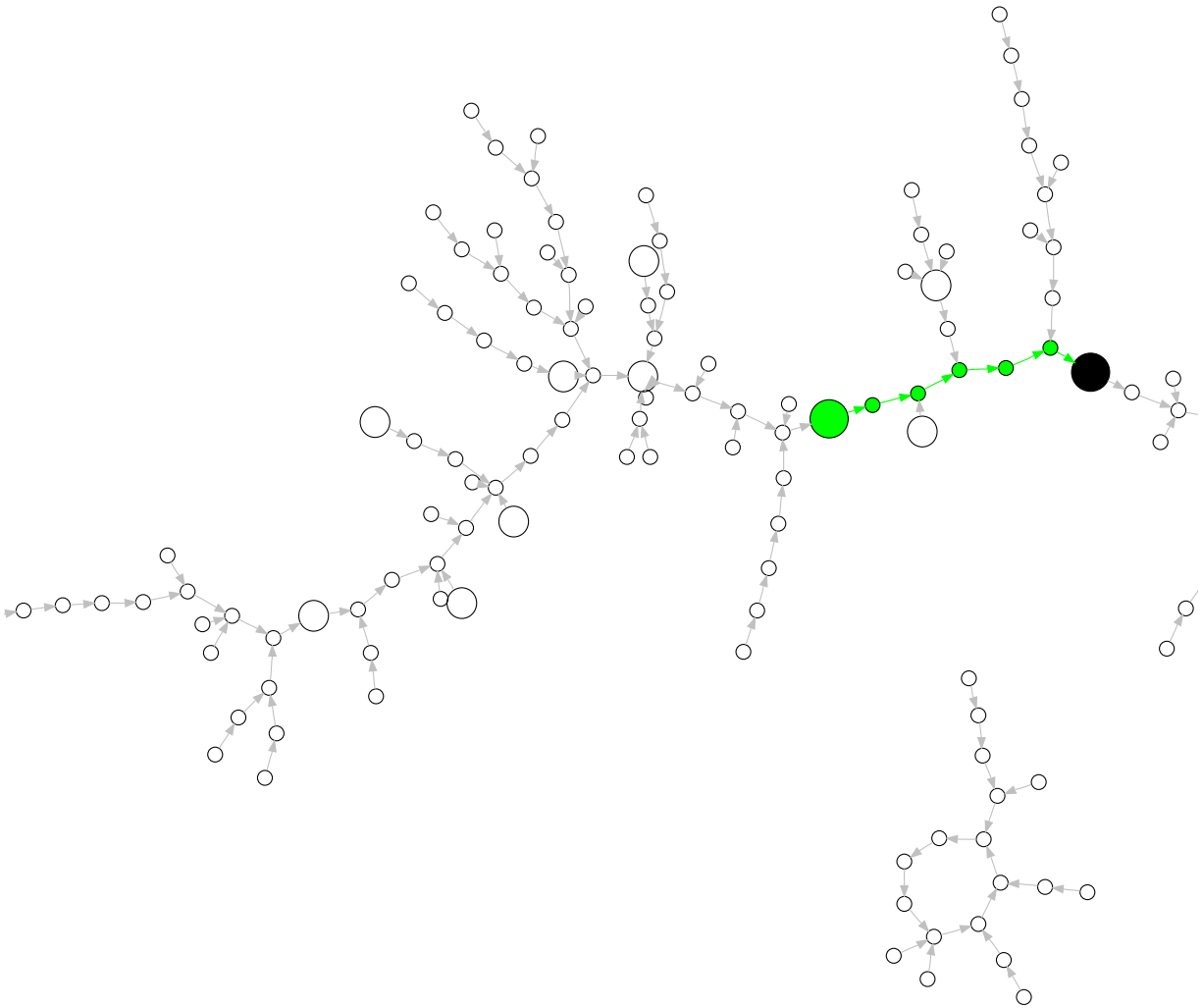
How to notice?
when to stop?

Idea: mark some points. Stop and report when getting to such a distinguished point.

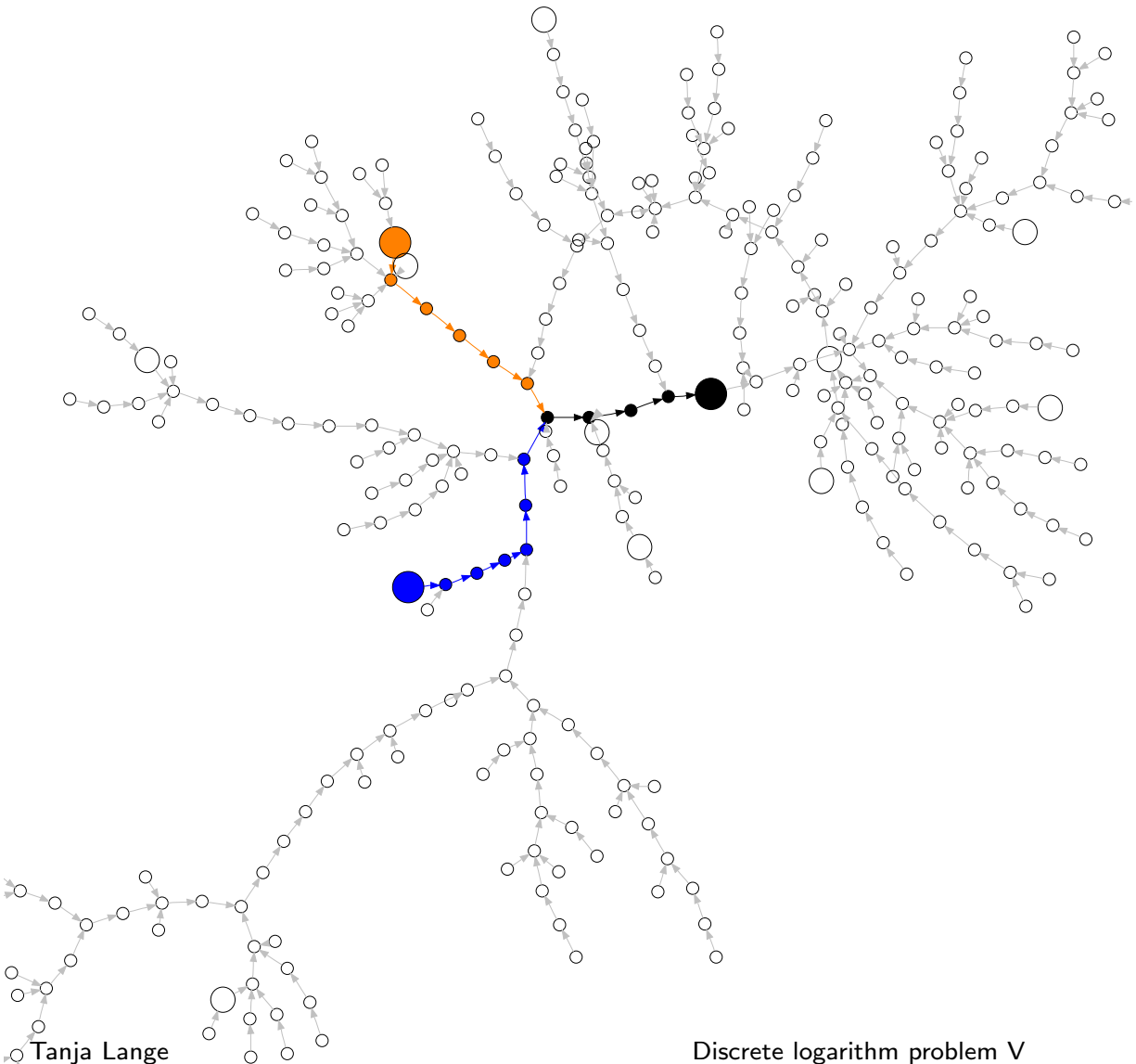




Walk to distinguished point, report to some server



Lucky case: two walks end in same distinguished point



Parallel Pollard rho

- Perform many walks with different starting points.
- Terminate each walk once it hits a distinguished point.
- Report the distinguished point and its a_i and b_i to server.
- Server receives, stores, and sorts all distinguished points.
- Two walks reaching same distinguished point give collision.
- As before, this collision solves the DLP.

Parallel Pollard rho

- Perform many walks with different starting points.
- Terminate each walk once it hits a distinguished point.
- Report the distinguished point and its a_i and b_i to server.
- Server receives, stores, and sorts all distinguished points.
- Two walks reaching same distinguished point give collision.
- As before, this collision solves the DLP.

How to identify distinguished points? Want something efficient to test.
Typical: top r bits of $x(W)$ are 0, takes $\approx 2^r$ steps to reach.

Parallel Pollard rho

- Perform many walks with different starting points.
- Terminate each walk once it hits a distinguished point.
- Report the distinguished point and its a_i and b_i to server.
- Server receives, stores, and sorts all distinguished points.
- Two walks reaching same distinguished point give collision.
- As before, this collision solves the DLP.

How to identify distinguished points? Want something efficient to test.
Typical: top r bits of $x(W)$ are 0, takes $\approx 2^r$ steps to reach.

How frequent should they be?
Infrequent: small number of very long walks, little storage on server, long delay before a collision is recognized. Must not hit cycle!

More frequent: shorter walks, more storage, more communication.
Too frequent: very short, degrades to random search.

Parallel Pollard rho

- Perform many walks with different starting points.
- Terminate each walk once it hits a distinguished point.
- Report the distinguished point and its a_i and b_i to server.
- Server receives, stores, and sorts all distinguished points.
- Two walks reaching same distinguished point give collision.
- As before, this collision solves the DLP.

How to identify distinguished points? Want something efficient to test.
Typical: top r bits of $x(W)$ are 0, takes $\approx 2^r$ steps to reach.

How frequent should they be?
Infrequent: small number of very long walks, little storage on server, long delay before a collision is recognized. Must not hit cycle!

More frequent: shorter walks, more storage, more communication.
Too frequent: very short, degrades to random search.

Any choice (unless walk enters cycle without DP) needs $\sqrt{n}$ steps.

Parallel Pollard rho

- Perform many walks with different starting points.
- Terminate each walk once it hits a distinguished point.
- Report the distinguished point and its a_i and b_i to server.
- Server receives, stores, and sorts all distinguished points.
- Two walks reaching same distinguished point give collision.
- As before, this collision solves the DLP.

How to identify distinguished points? Want something efficient to test.
Typical: top r bits of $x(W)$ are 0, takes $\approx 2^r$ steps to reach.

How frequent should they be?
Infrequent: small number of very long walks, little storage on server, long delay before a collision is recognized. Must not hit cycle!

More frequent: shorter walks, more storage, more communication.
Too frequent: very short, degrades to random search.

Any choice (unless walk enters cycle without DP) needs $\sqrt{n}$ steps.

Several speedups, e.g. identifying W and $-W$, some pitfalls.