Algorithmic Problems Related to Internet Graphs

Thomas Erlebach

University of Leicester

Based on joint work with:

Zuzana Beerliova, Pino Di Battista, Felix Eberhard, Alexander Hall, Michael Hoffmann, Matúš Mihaľák, Alessandro Panconesi, Maurizio Patrignani, Maurizio Pizzonia, L. Shankar Ram, Thomas Schank, Danica Vukadinović

The Internet

- Size of the Internet (as of 2003):
  - $\sim$ 7–10M routers
  - $\sim$ 170M hosts
  - $\sim$ 650M users
- In recent years, significant interest in mapping the Internet.
- Different kinds of Internet graphs:
  - Router-level graph (routers and hosts)
    traceroute experiments
  - AS-level graph (autonomous systems)
    traceroute, BGP tables, registries
  - WWW graph (web pages and hyperlinks)
    crawling

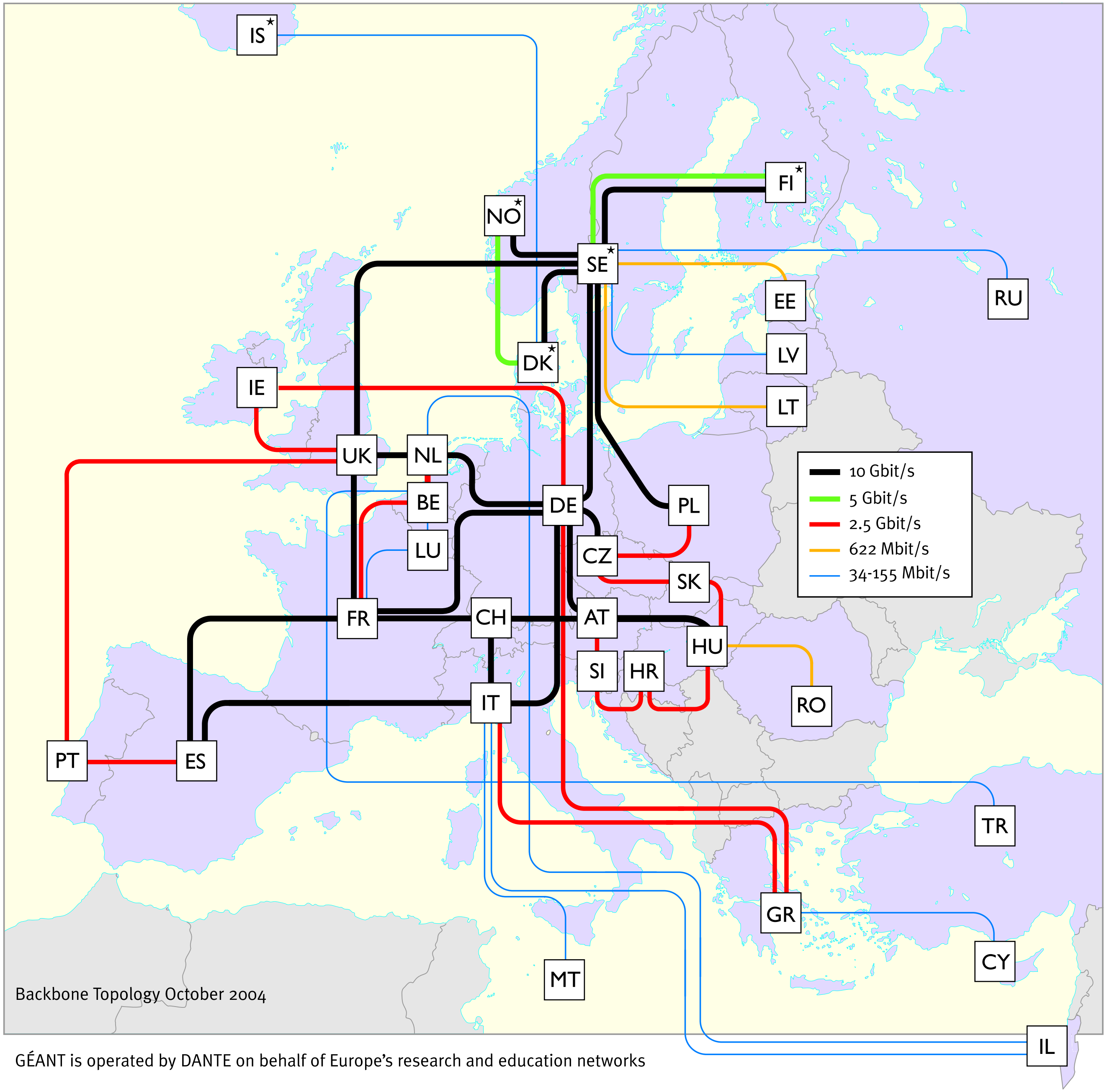
Autonomous Systems (ASs)

- AS: subnetwork under separate administrative control.
- Examples:
  - AS8: Rice University
  - AS378: ILAN
  - AS701: UUNET
  - AS768: JANET
  - AS20965: GEANT
- An AS can consist of tens to thousands of routers and hosts.
- roughly 15,000 ASs in 2003, 23,000 ASs in 2006.
- Routing between ASs: BGP (border gateway protocol)

AS786:


The JANET Backbone
Key
AbMAN Aberdeen MAN
C&NL MAN Cumbria and North Lancashire Area Network
Clydenet Clyde Area Network
EaStMAN Edinburgh and Stirling MAN
EastNet East of England Regional Network
EMMAN East Midlands MAN
FaTMAN Fife and Tayside MAN
LeNSE Learning Network South East
LMN London Metropolitan Network
MAN Metropolitan Area Network
MidMAN Midlands MAN
NNW Net North West
NorMAN North East MAN
SWERN South West England Regional Network
TVN Thames Valley Network
UHIMI Network University of the Highlands and Islands Millennium Institute Network
YHMAN Yorkshire and Humberside MAN
Core Point of Presence
Regional network entry point
Backbone link
Access link
UHIMI Network
AbMAN
FaTMAN
Glasgow
Edinburgh
Clydenet
EaStMAN
NorMAN
Northern Ireland
YHMAN
C&NL MAN
Leeds
Warrington
HEAnet
NNW
EMMAN
North Wales MAN
MidMAN
EastNet
South Wales MAN
TVN
Bristol
LMN
Reading
London
Kentish MAN
SWERN
LeNSE
Portsmouth

AS20965: GEANT



Traceroute: Leicester – Haifa

traceroute: pc14.mcs.le.ac.uk → www.haifa.ac.il

1 gate (143.210.72.1)
2 143.210.6.2 (143.210.6.2)
3 uol3-gw-7-1.emman.net (194.82.121.177)
4 uol1-gw-g3.emman.net (212.219.212.85)
5 uon6-gw-7-1.emman.net (194.82.121.25)
6 nottingham-bar.ja.net (146.97.40.21)
7 po12-0.lond-scr.ja.net (146.97.35.13)
8 po6-0.lond-scr3.ja.net (146.97.33.30)
9 po1-0.gn2-gw1.ja.net (146.97.35.98)
10 janet.rt1.lon.uk.geant2.net (62.40.124.197)
11 so-4-0-0.rt1.par.fr.geant2.net (62.40.112.105)
12 so-7-3-0.rt1.gen.ch.geant2.net (62.40.112.29)
13 so-2-0-0.rt1.mil.it.geant2.net (62.40.112.34)
14 so-1-2-0.rt1.tik.il.geant2.net (62.40.112.121)
15 iucc-gw.rt1.tik.il.geant2.net (62.40.124.126)
16 haifa-gp0-cel-g.ilan.net.il (128.139.234.2)
17 * * *

Internet Mapping Projects

A map of the Internet can be obtained by combining the local views from a number of locations (vantage points):

- Path data from traceroute experiments
- Path data from BGP routing tables

Examples:

- Bill Cheswick's Internet Mapping Project (traceroute, router-level)
- Oregon Route Views (based on BGP data, AS-level)
- DIMES (Yuval Shavitt): router-level and AS-level, based on volunteer community
- and others

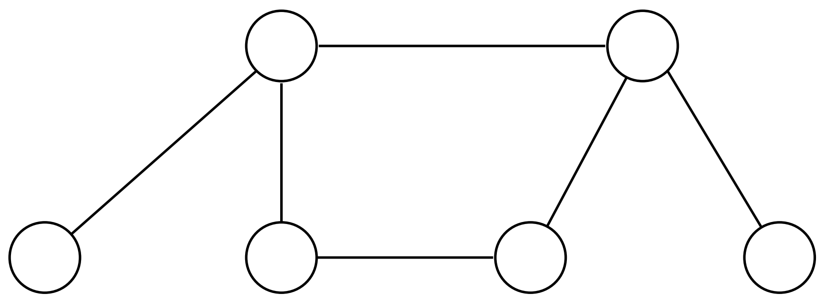
Outline

- AS Relationships and the Valley-Free Path Model
- **Inferring AS Relationships**
- **Cuts and Disjoint Paths in the Valley-Free Path Model**
- **Network Discovery and Verification**

AS Relationships and the Valley-Free Path Model

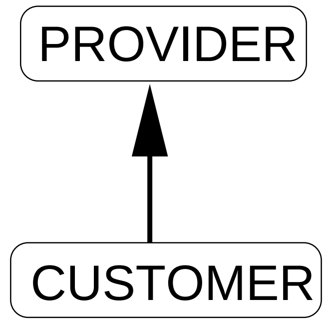
Undirected AS-Graph

- An **undirected AS-graph** is a simple, undirected graph with
  - a vertex for every AS
  - an edge joining two vertices if the corresponding ASs have at least one physical connection.
- Example:



AS Relationships

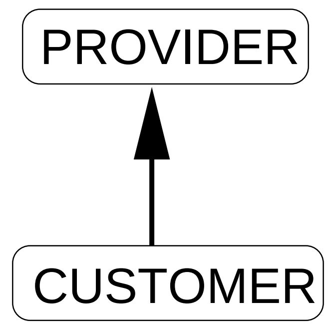
- Customer-Provider: directed edge



Customer pays provider for Internet access.

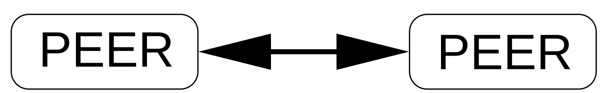
AS Relationships

- Customer-Provider: directed edge



Customer pays provider for Internet access.

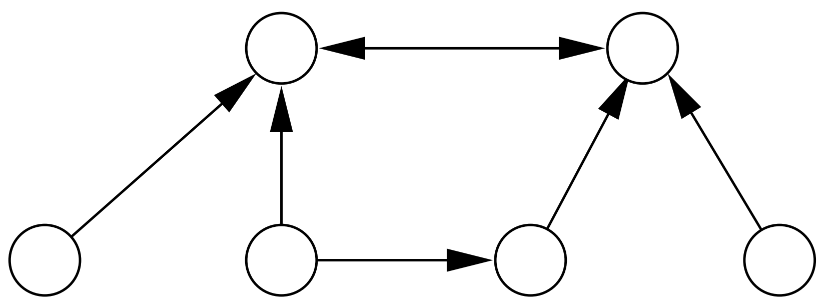
- Peer-to-Peer: bidirected edge



Peers exchange traffic of their subnetworks and their customers.

AS-Graph

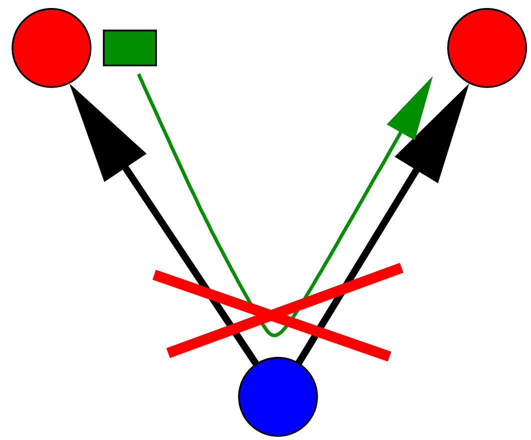
- An **AS-graph** is a graph $G = (V, E)$ in which any two vertices $u, v \in V$ can
  - be non-adjacent,
  - have a directed edge (u, v) or (v, u),
  - or have a bidirected edge $\{u, v\}$.
- Example:



- Model by Subramanian et al., 2002.

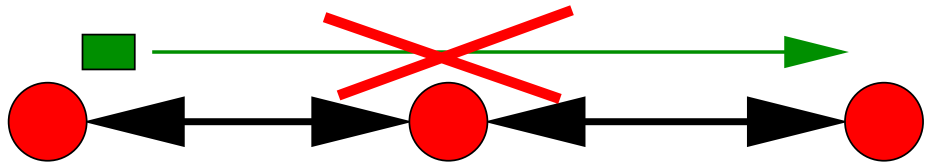
Routing Policies

- Customers do not route traffic from one provider to another:

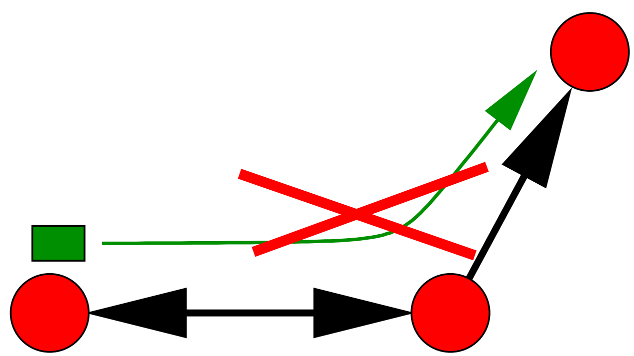


Routing Policies

- Peers do not forward to other peers:

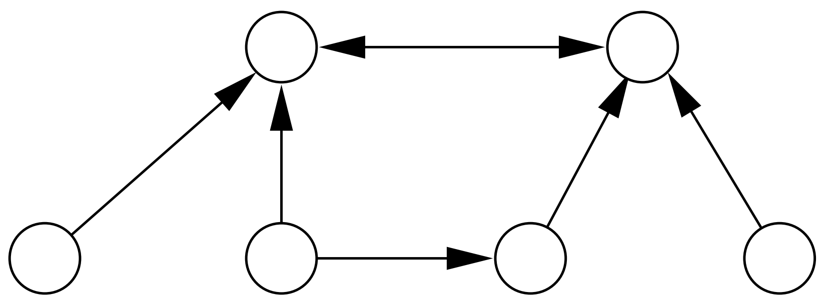


- Peers do not forward from peers to providers (and vice versa):



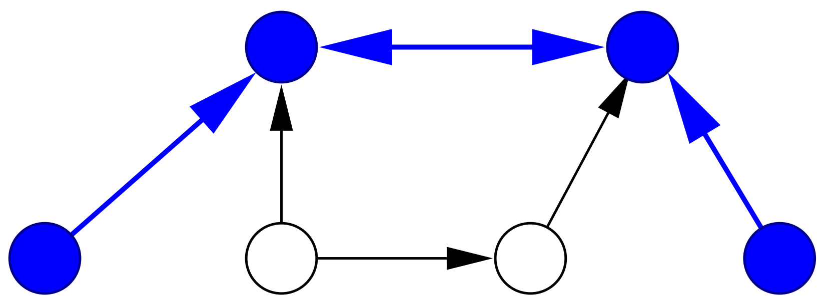
Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



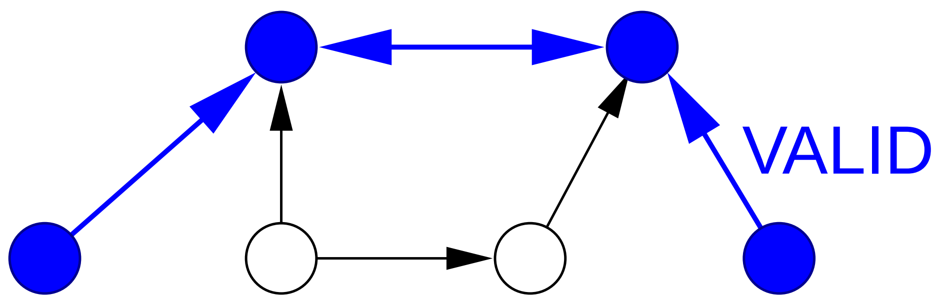
Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



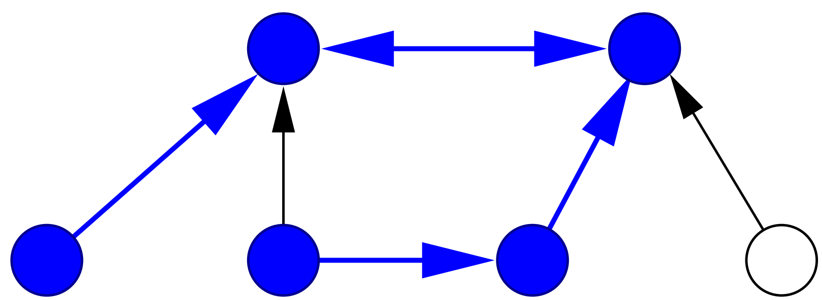
Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



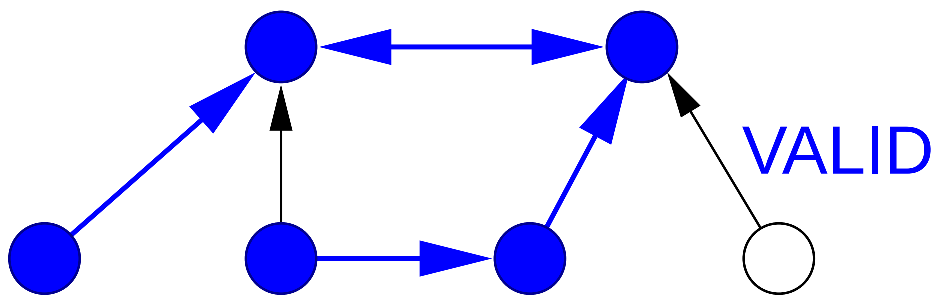
Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



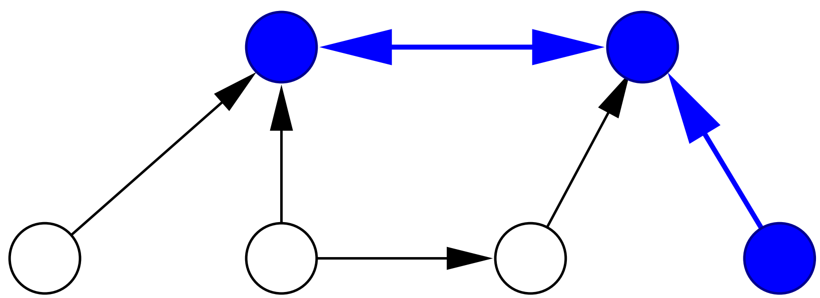
Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



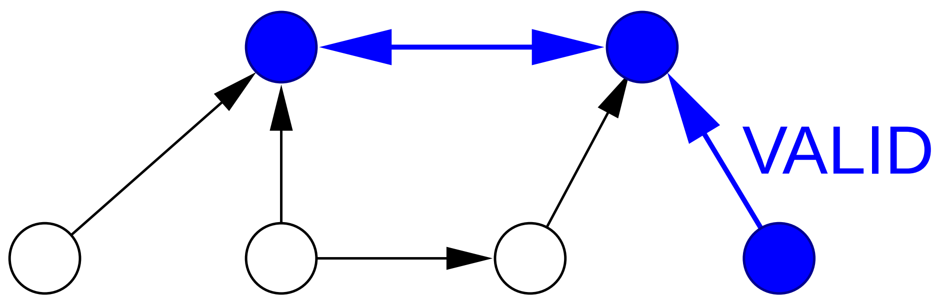
Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



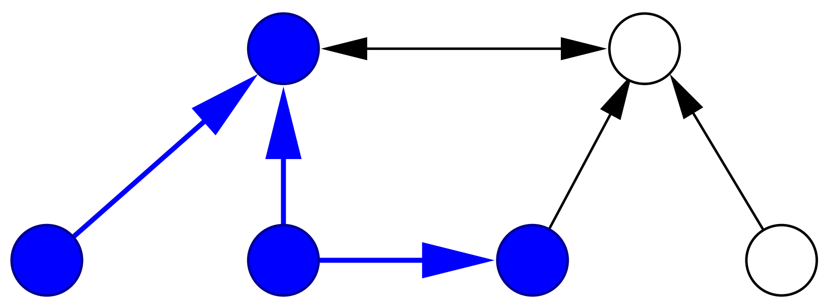
Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



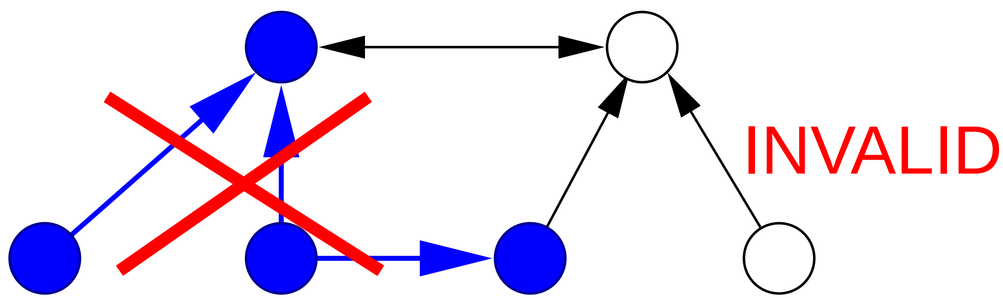
Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



Valley-Free Paths

- A path π from s to t in an AS-graph is **valid** in the valley-free path model, if it consists of
  - a sequence of ≥ 0 forward edges,
  - followed by 0 or 1 bidirected edges,
  - followed by a sequence of ≥ 0 reverse edges.
- Example:



Inferring AS Relationships

Motivation

- AS relationships are important for analyzing BGP routing, but difficult to obtain.
- Idea: Use information about BGP paths to infer AS relationships.
- Initiated by [Gao, 2001].
- Formalization as Type-of-Relationship (ToR) problem by Subramanian et al., 2002.

ToR-Problem

Given:

- undirected graph G, set P of paths in G.

Solution:

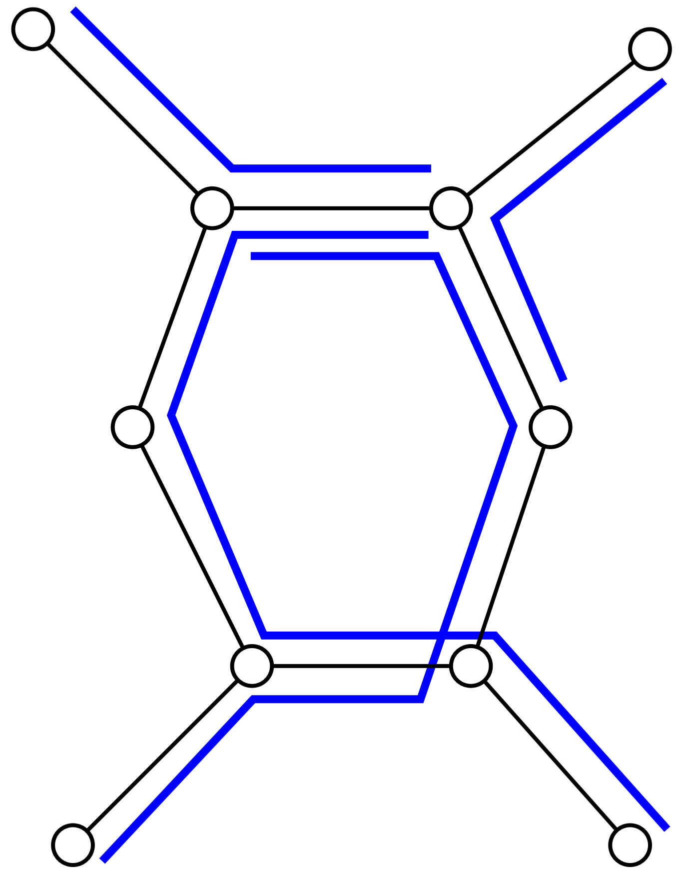
- classification of edges of G into customer-provider and peer-to-peer relationships.

Objective:

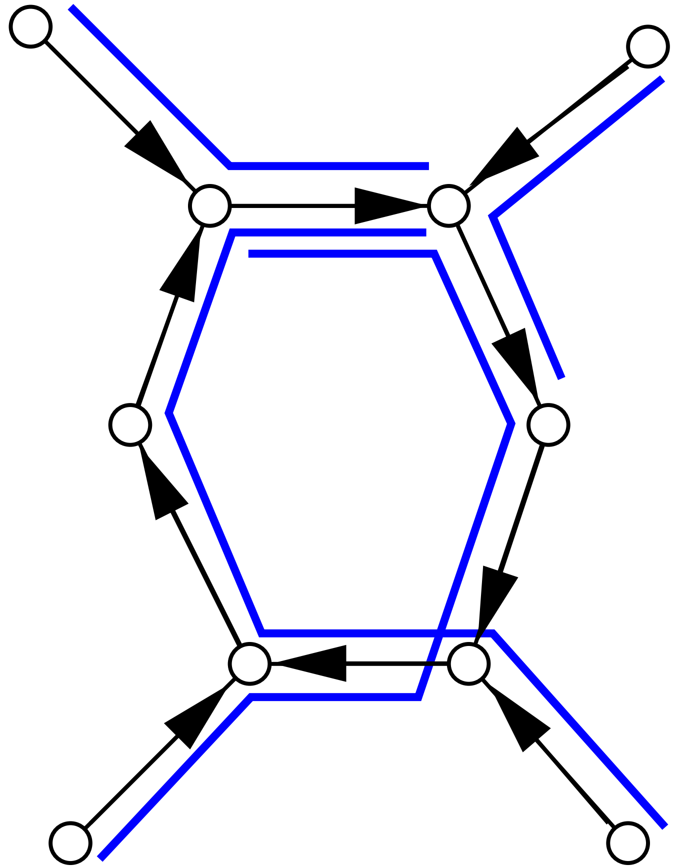
- maximize the number of paths in P that are made valid.

Special case: check if <u>all</u> paths in P can be valid.

Example

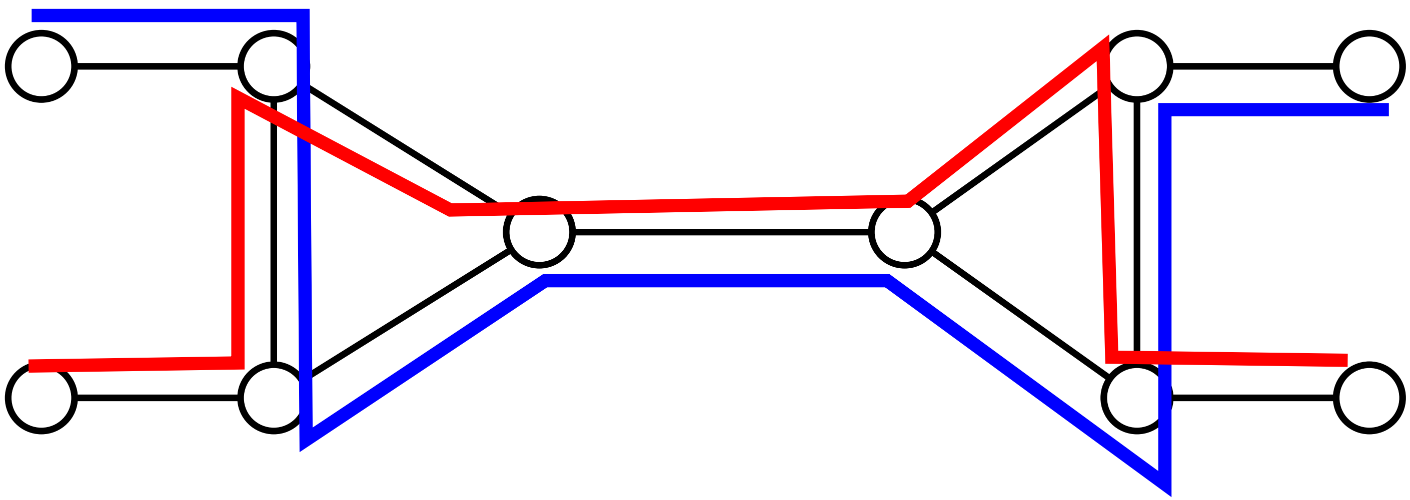


Example

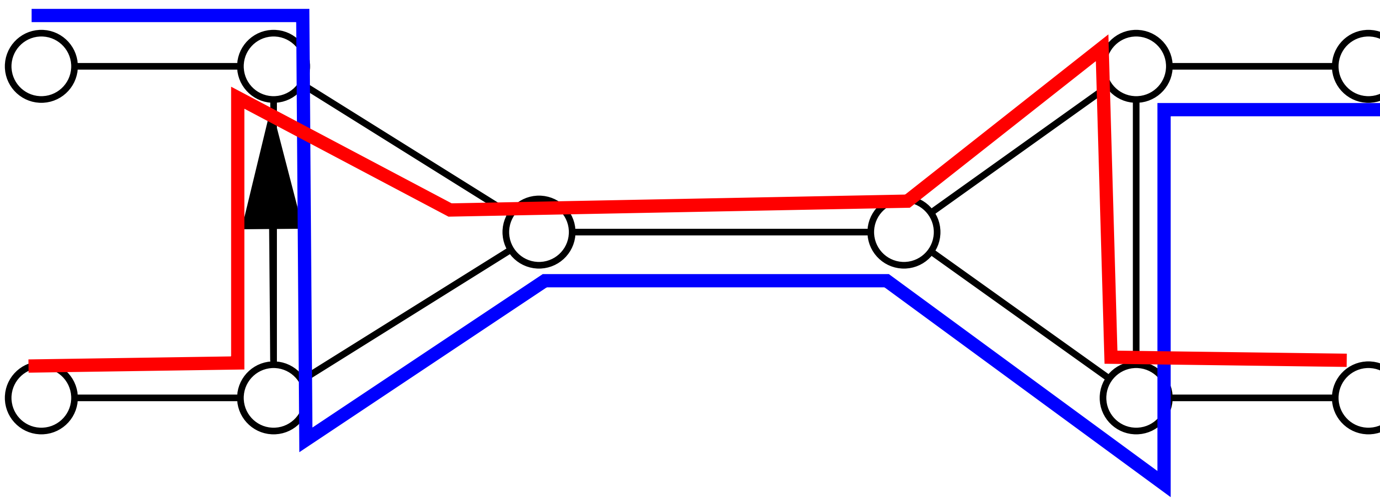


All paths are valid!

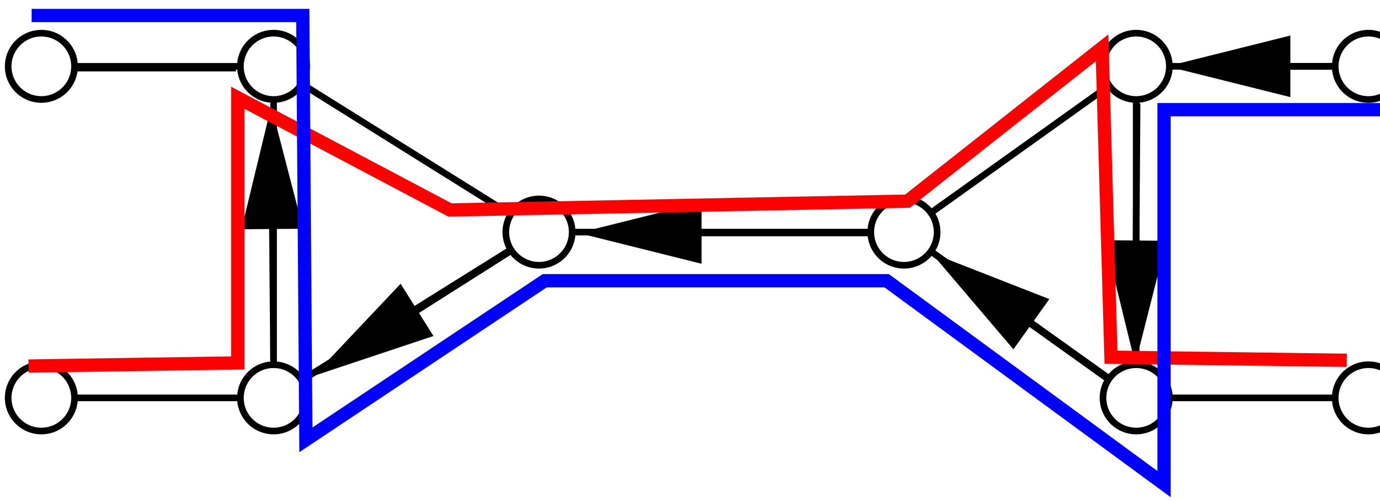
Example 2



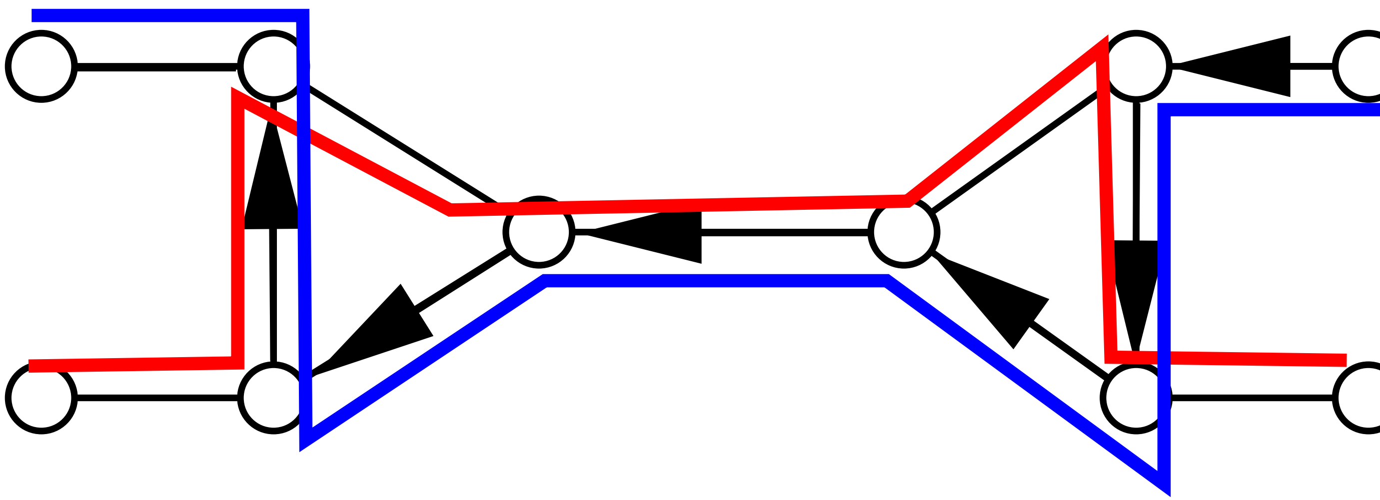
Example 2



Example 2



Example 2



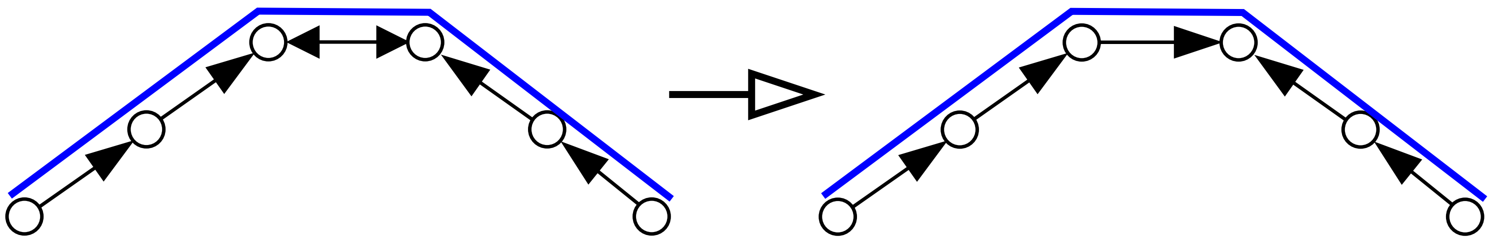
Only one of the two paths can be valid!

Results

- There is a linear-time algorithm for deciding whether all paths can be made valid (➠ 2SAT).
- If not all paths can be made valid, the ToR-problem is NP-hard and APX-hard even if all paths have length 2.
- In general, the ToR-problem cannot be approximated within $\frac{1}{n^{1-\varepsilon}}$ for n paths, unless $NP = ZPP$.
- If the path lengths are bounded by a constant, the ToR-problem can be approximated within a constant factor (trivial algorithm: random orientation).
- If the path length is at most 2, 3, or 4, we obtain approximation ratio 0.94, 0.84, or 0.36 (using MAX2SAT [Goemans, Williamson 1994; Lewin, Livnat, Zwick 2002]).

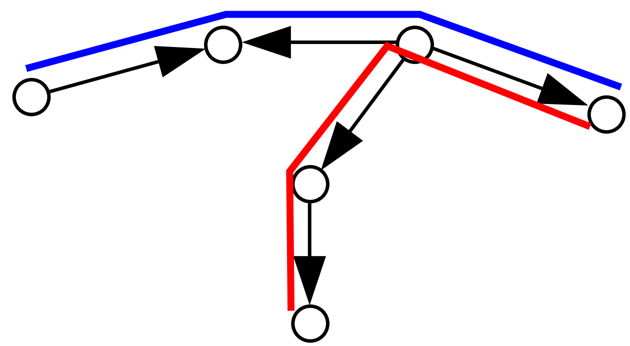
Sketch of Algorithm

- Don't use peer-to-peer edges at all!



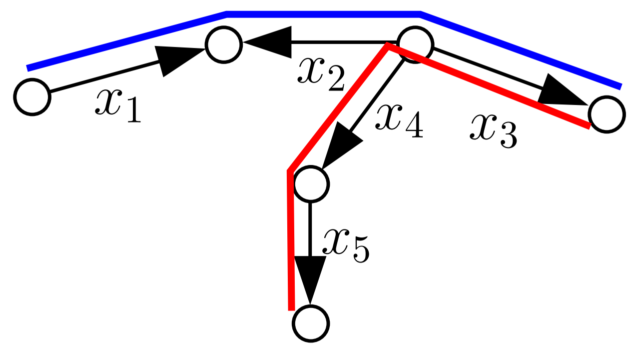
Sketch of Algorithm

- Initially, classify each edge arbitrarily.



Sketch of Algorithm

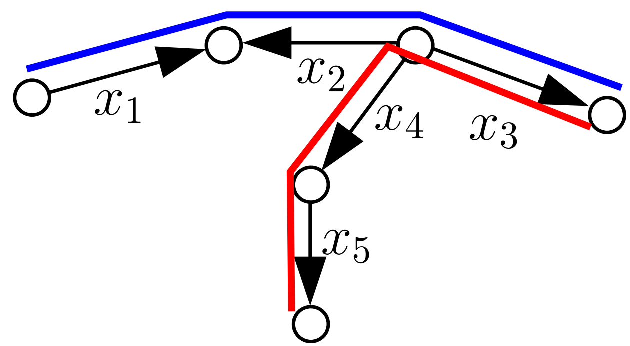
- Build a 2SAT formula representing a solution that makes all paths valid.



$$(\overline{x_1} \vee \overline{x_2}) \wedge (x_2 \vee x_3) \wedge (x_4 \vee x_3) \wedge (x_5 \vee \overline{x_4})$$

Sketch of Algorithm

- Use MAX2SAT algorithm to obtain good truth assignment for the variables.

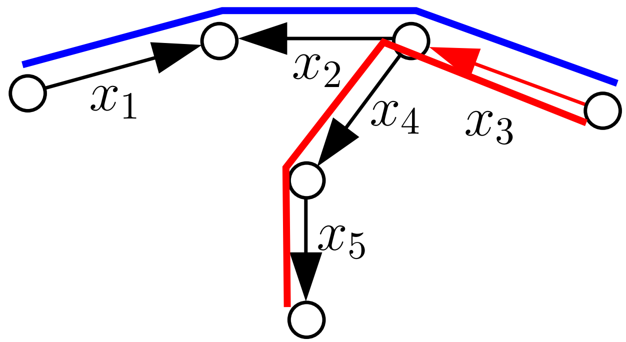


$$(\overline{x_1} \vee \overline{x_2}) \wedge (x_2 \vee x_3) \wedge (x_4 \vee x_3) \wedge (x_5 \vee \overline{x_4})$$

$$x_1 = \mathrm{F}, x_2 = \mathrm{F}, x_3 = \mathrm{T}, x_4 = \mathrm{F}, x_5 = \mathrm{F}$$

Sketch of Algorithm

- Flip directions of true variables.



$$(\overline{x_1} \vee \overline{x_2}) \wedge (x_2 \vee x_3) \wedge (x_4 \vee x_3) \wedge (x_5 \vee \overline{x_4})$$

$$x_1 = \mathrm{F}, x_2 = \mathrm{F}, x_3 = \mathrm{T}, x_4 = \mathrm{F}, x_5 = \mathrm{F}$$

Comments on Relationship Inference

- Maximizing the number of valid paths is not really the right objective function. We need to find a formulation of the ToR problem that yields more realistic classifications:
  - Avoid customer-provider cycles.
  - Include peer-to-peer edges.
  - Include sibling edes.
- Other direction: Use **active probing** methods to obtain better classifications.

Cuts and Disjoint Paths in the Valley-Free Path Model

Robustness Considerations

- Robustness of connectivity between s and t:
  - Minimum size of a cut separating s and t.
  - Maximum number of disjoint paths between s and t.
- Efficiently computable using network flow techniques in standard undirected or directed graphs.
- But: should take into account routing policies!
  ➠ valley-free path model

$\Rightarrow$ Problems Min Valid s-t-Cut and Max Disjoint Valid s-t-Paths (**vertex version** and edge version).

Min Valid s-t-Vertex-Cut

Given:

- Directed graph $G = (V, E)$ and two non-adjacent vertices $s, t \in V$

Feasible solution:

- A valid s-t-vertex-cut C ($C \subseteq V \setminus \{s, t\}$ s.t. $\nexists$ valid s-t-path in $G \setminus C$)

Objective:

- Minimize $|C|$.

Smallest number of ASs that must fail in order to disconnect s and t with respect to valley-free paths.

Max Vertex-Disjoint Valid s-t-Paths

Given:

- Directed graph $G = (V, E)$ and two non-adjacent vertices $s, t \in V$

Feasible solution:

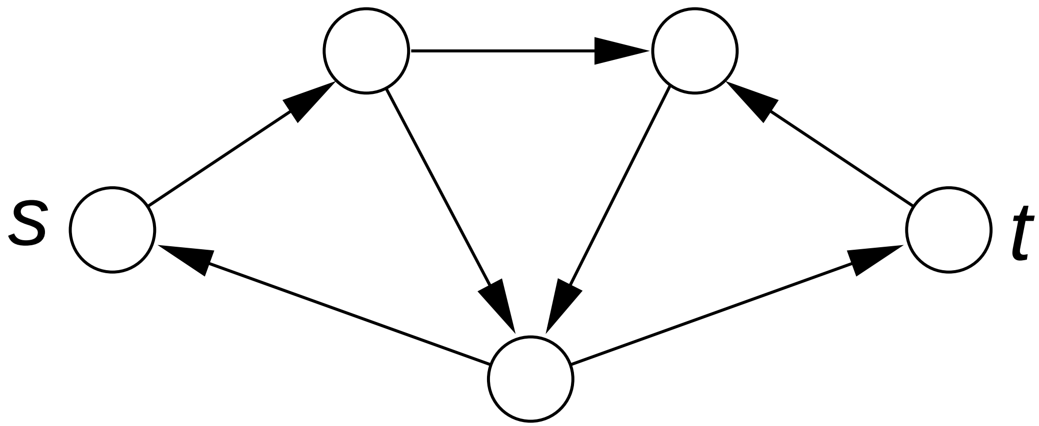
- Set $\mathcal{P}$ of vertex-disjoint valid s-t-paths in G

Objective:

- Maximize $|\mathcal{P}|$.

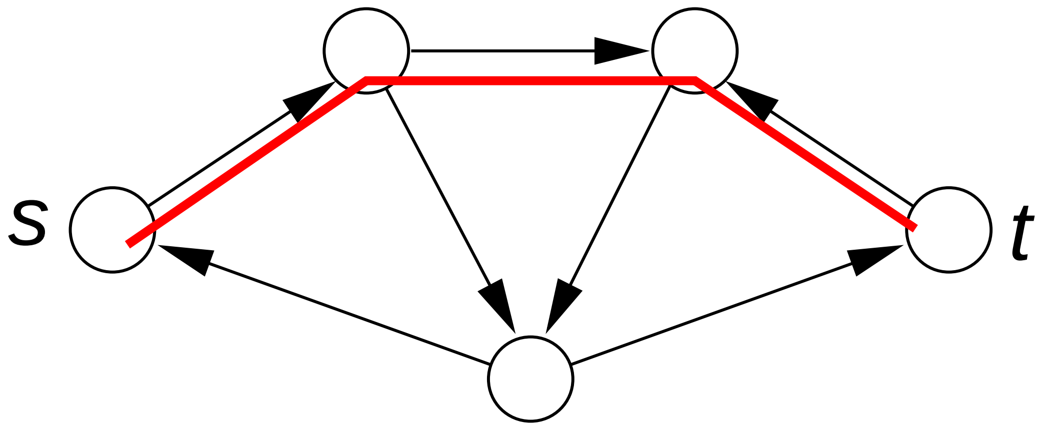
Largest number of disjoint valley-free paths connecting ASs s and t.

Example



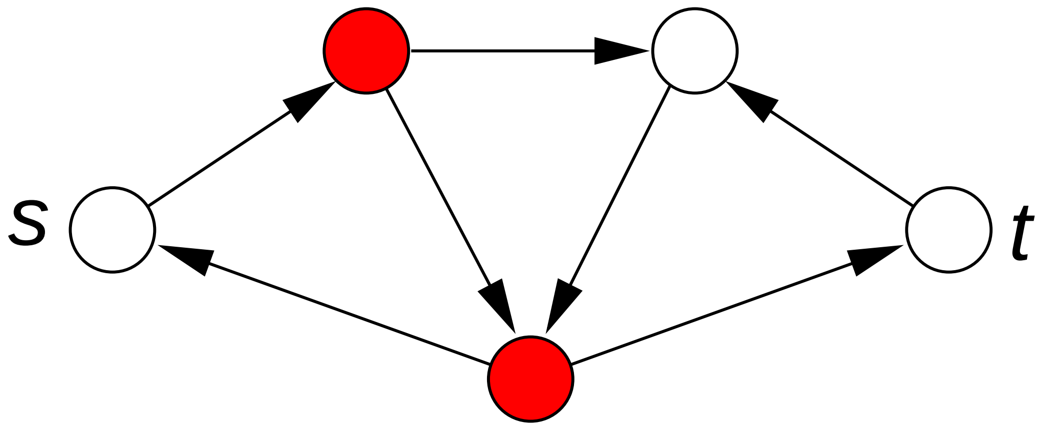
- max number of vertex-disjoint s-t-paths:
- min valid s-t-vertex-cut:

Example



- max number of vertex-disjoint s-t-paths: 1
- min valid s-t-vertex-cut:

Example



- max number of vertex-disjoint s-t-paths: 1
- min valid s-t-vertex-cut: 2

Hardness Results

Theorem. Min Valid s-t-Vertex-Cut is APX-hard.

Proof. By reduction from 3-WAY EDGE CUT.

Theorem. Max Vertex-Disjoint Valid s-t-Paths is NP-hard and cannot be approximated with ratio $2-\varepsilon$ for any $\varepsilon > 0$ unless $P = NP$.

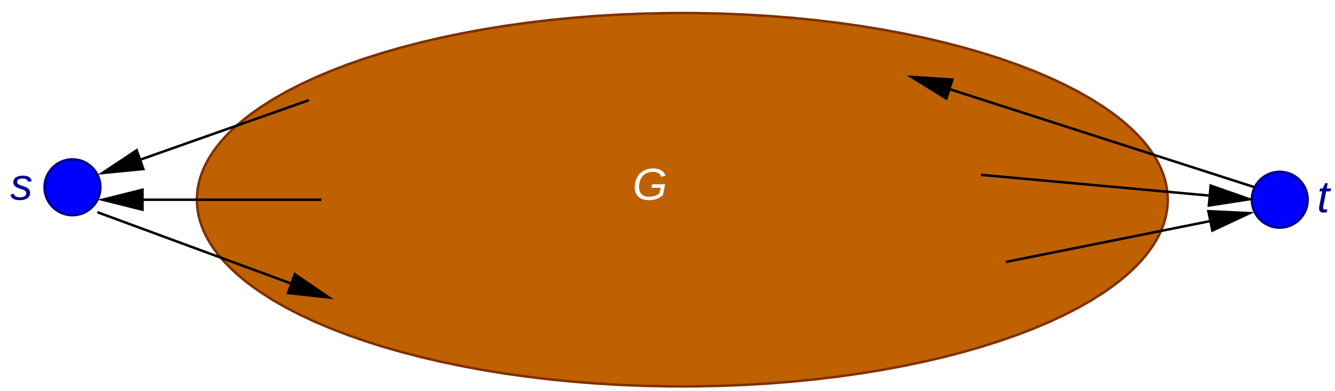
Proof. By reduction from 2DIRPATH.

Main Result

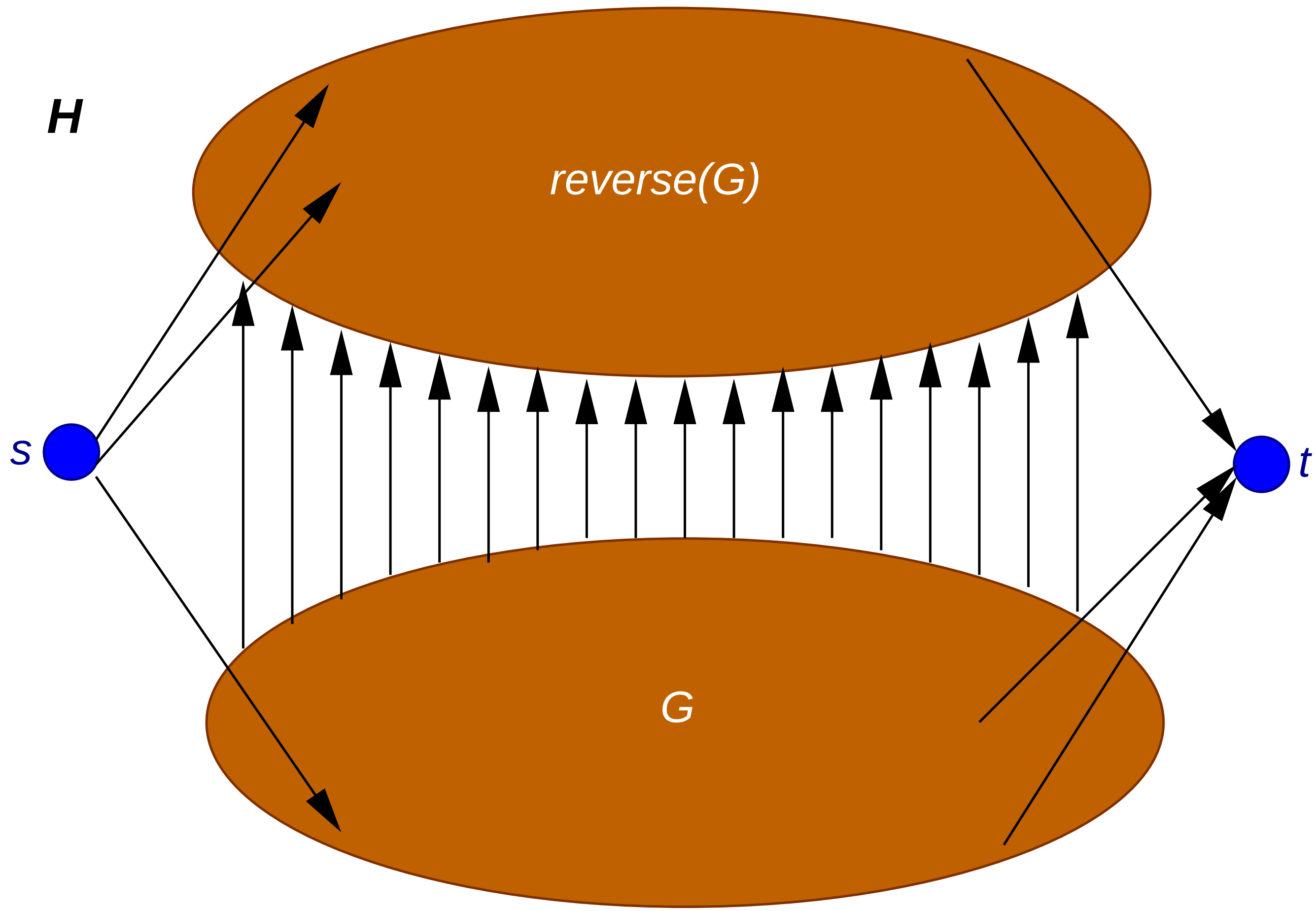
Theorem. There is an efficient algorithm that computes a valid s-t-vertex-cut of size c and a set of d vertex-disjoint valid s-t-paths such that $c \leq 2 \cdot d$.

Corollary. There is a 2-approximation algorithm for Min Valid s-t-Vertex-Cut and a 2-approximation algorithm for Max Vertex-Disjoint Valid s-t-Paths.

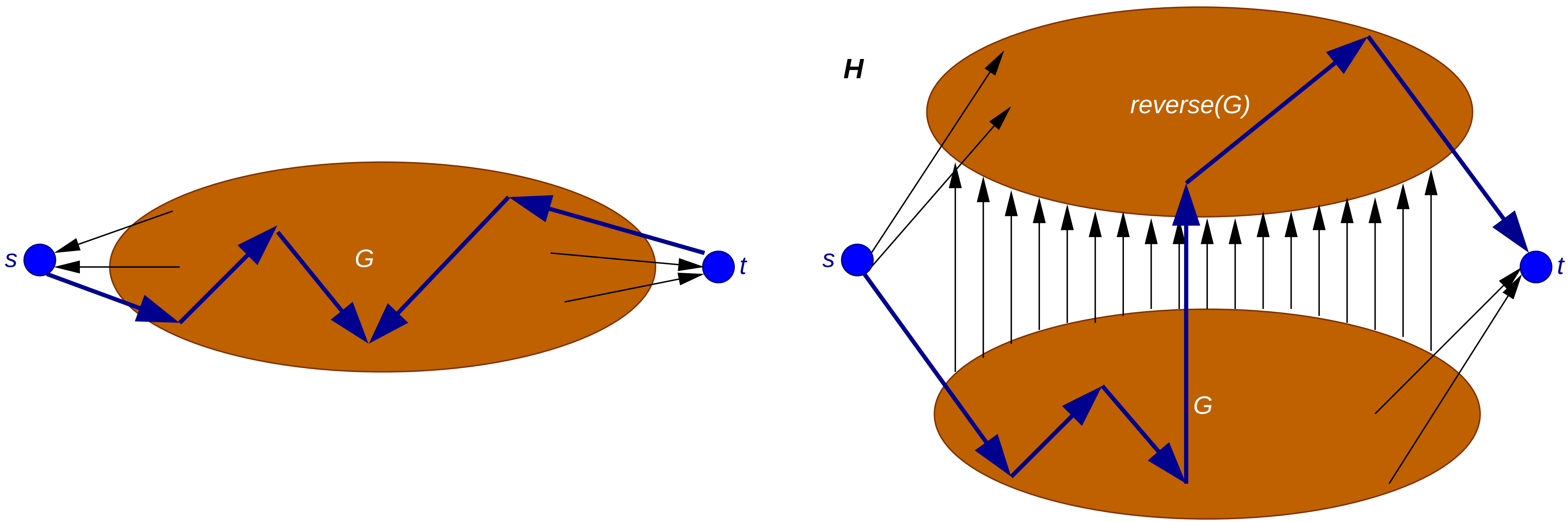
Two-Layer Model



Two-Layer Model



Paths in G and H



valid path in $G \equiv$ directed path in H

Cut-Algorithm

① Compute minimum s-t-vertex-cut C_H in H.

② Output the set $C_G = \{v \in V(G) \mid \geq 1$ copy of v is in $C_H\}$ as valid s-t-cut.

Analysis:

- $|C_G| \leq |C_H|$, C_G is valid s-t-vertex-cut
- $|C_H| \leq 2\cdot$ size of min valid s-t-vertex-cut in G

➥ 2-approximation algorithm

Path-Algorithm

① Compute max disjoint s-t-paths $\mathcal{P}_H$ in H.

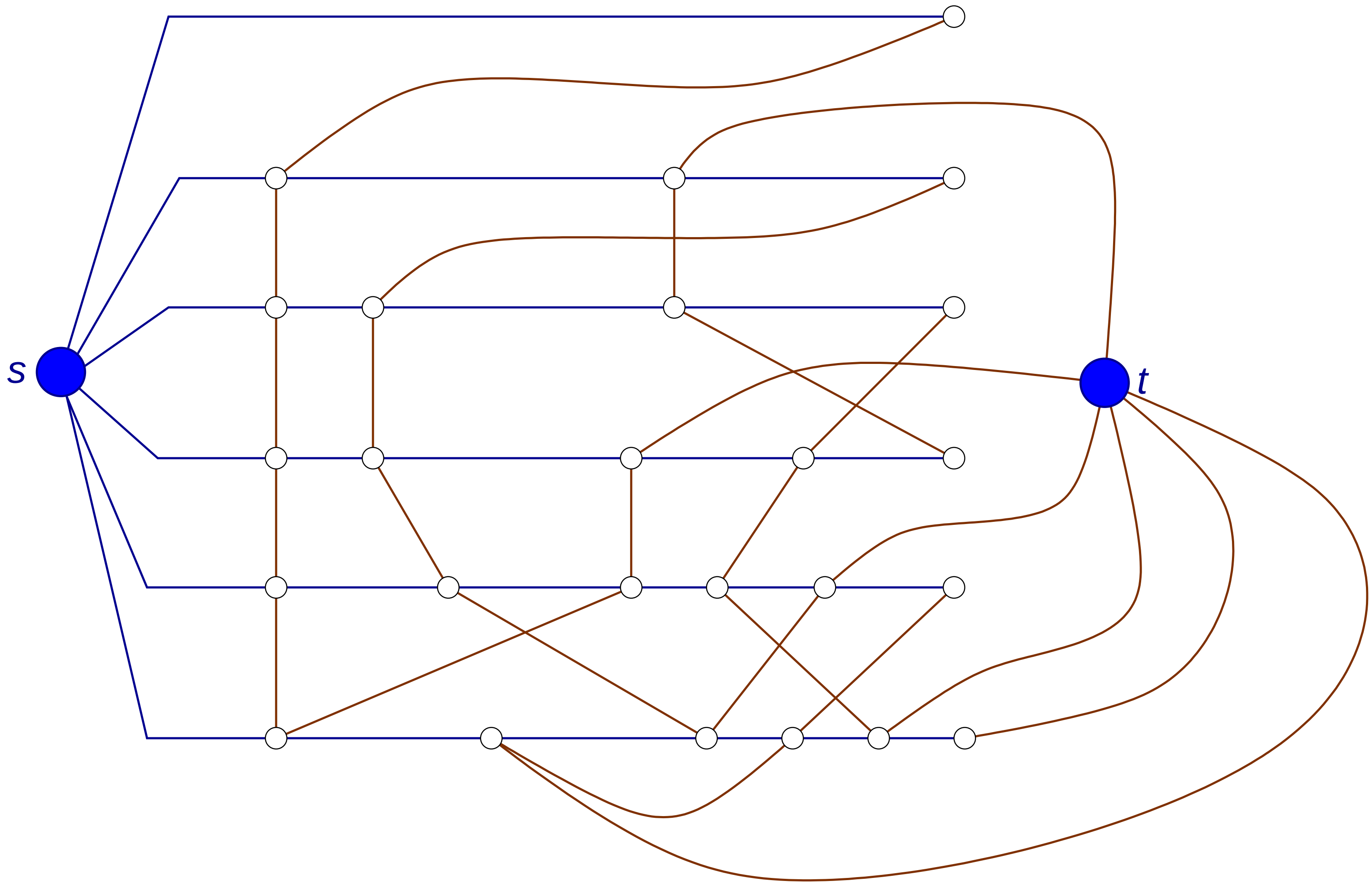
② Interpret $\mathcal{P}_H$ as set $\mathcal{P}_G$ of valid s-t-paths in G.

③ Recombine parts of paths in $\mathcal{P}_G$ to get at least $\frac{1}{2}|\mathcal{P}_G|$ disjoint valid s-t-paths in G.

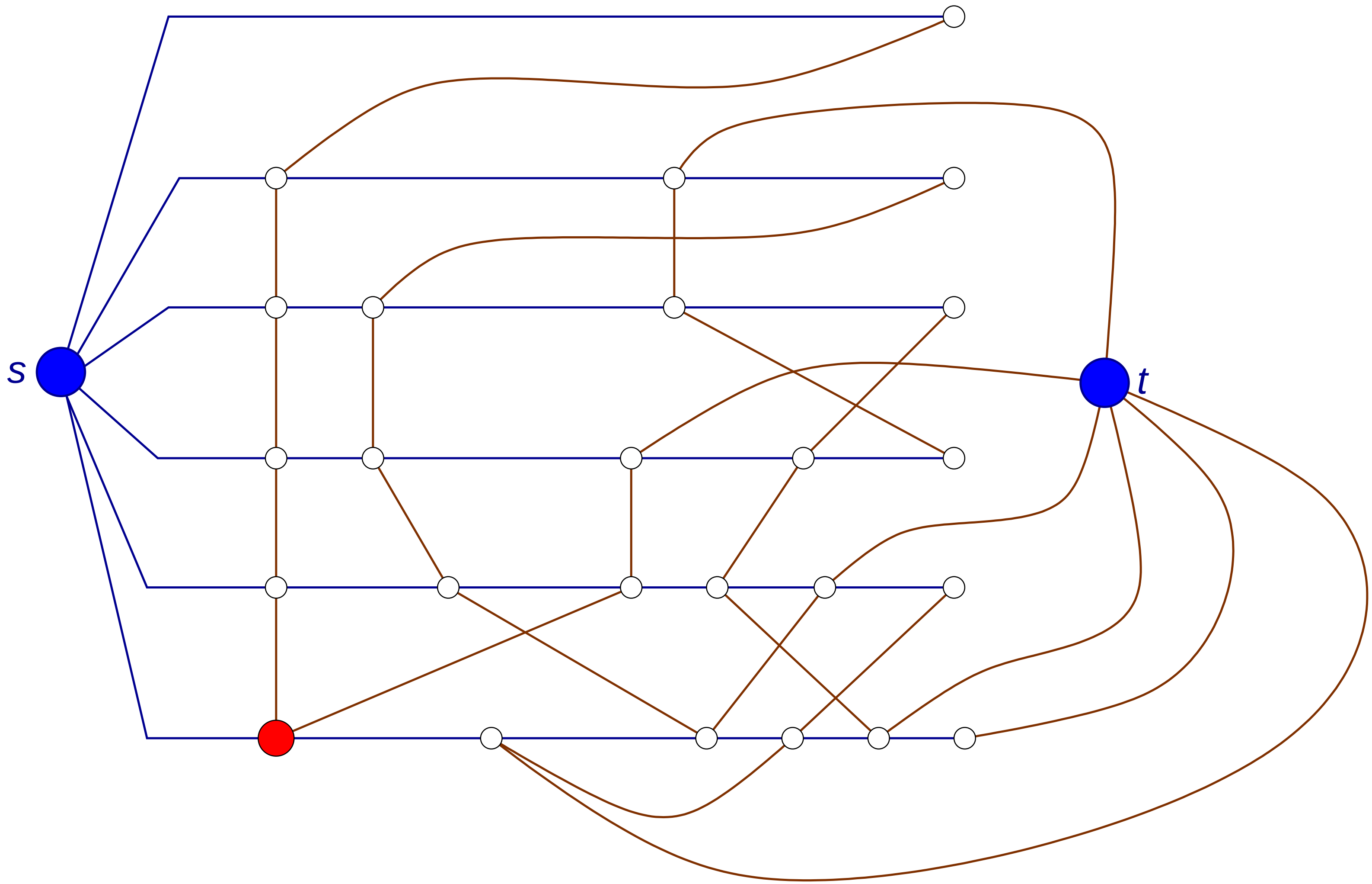
Observations:

- Forward parts of paths in $\mathcal{P}_G$ are disjoint.
- Backward parts of paths in $\mathcal{P}_G$ are disjoint.
- Forward part of one path may intersect backward parts of other paths.

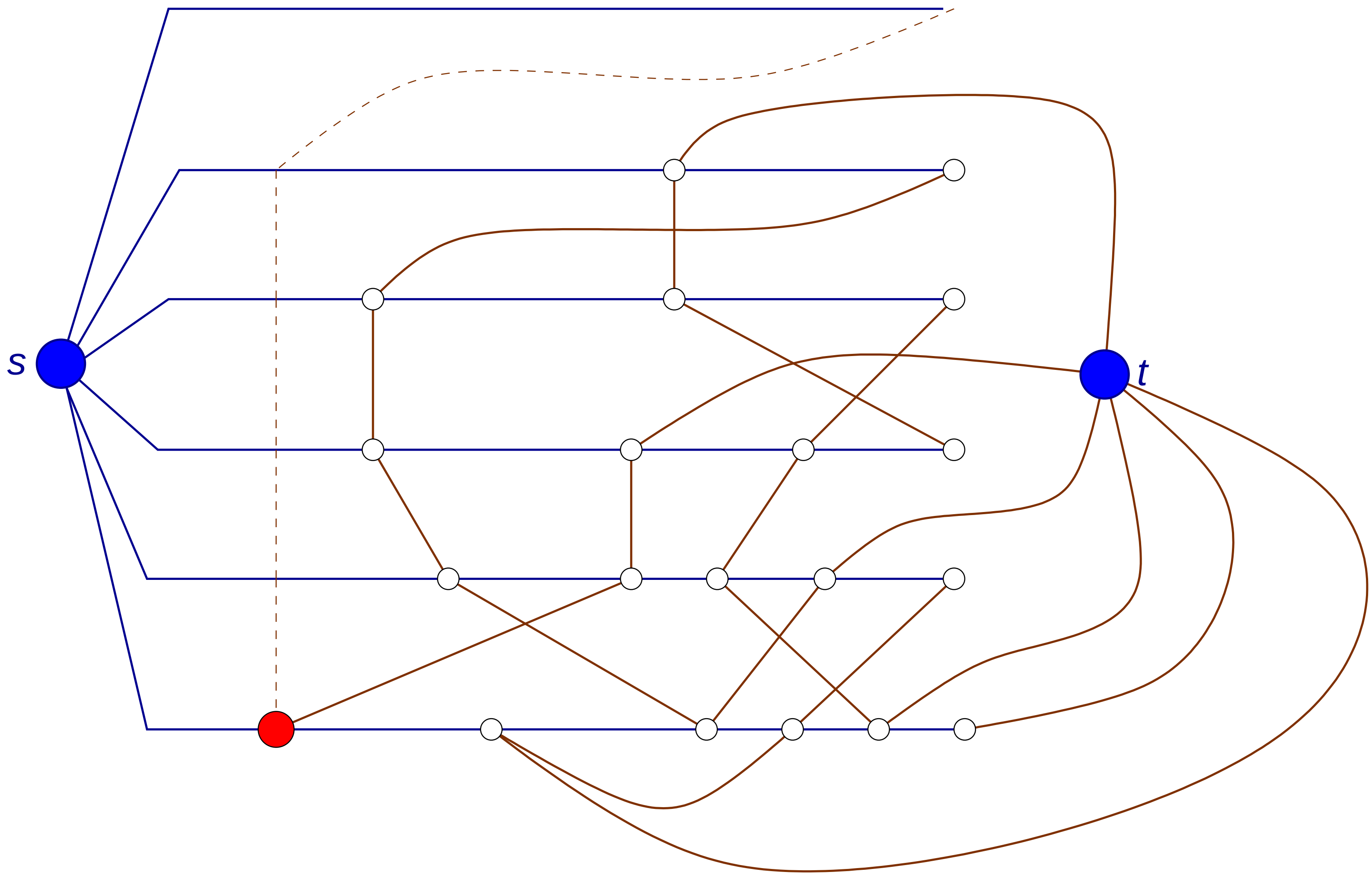
Recombination



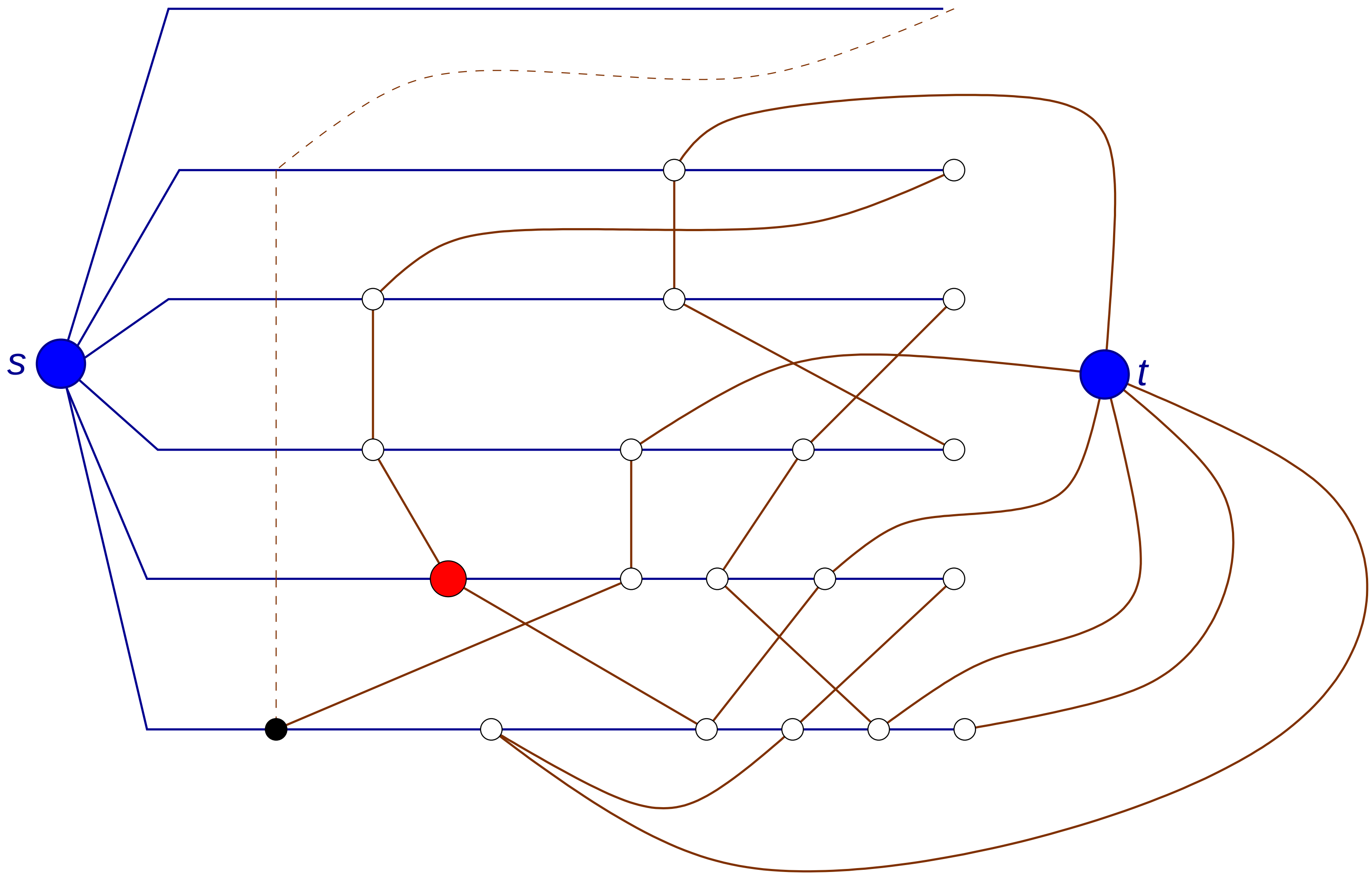
Recombination



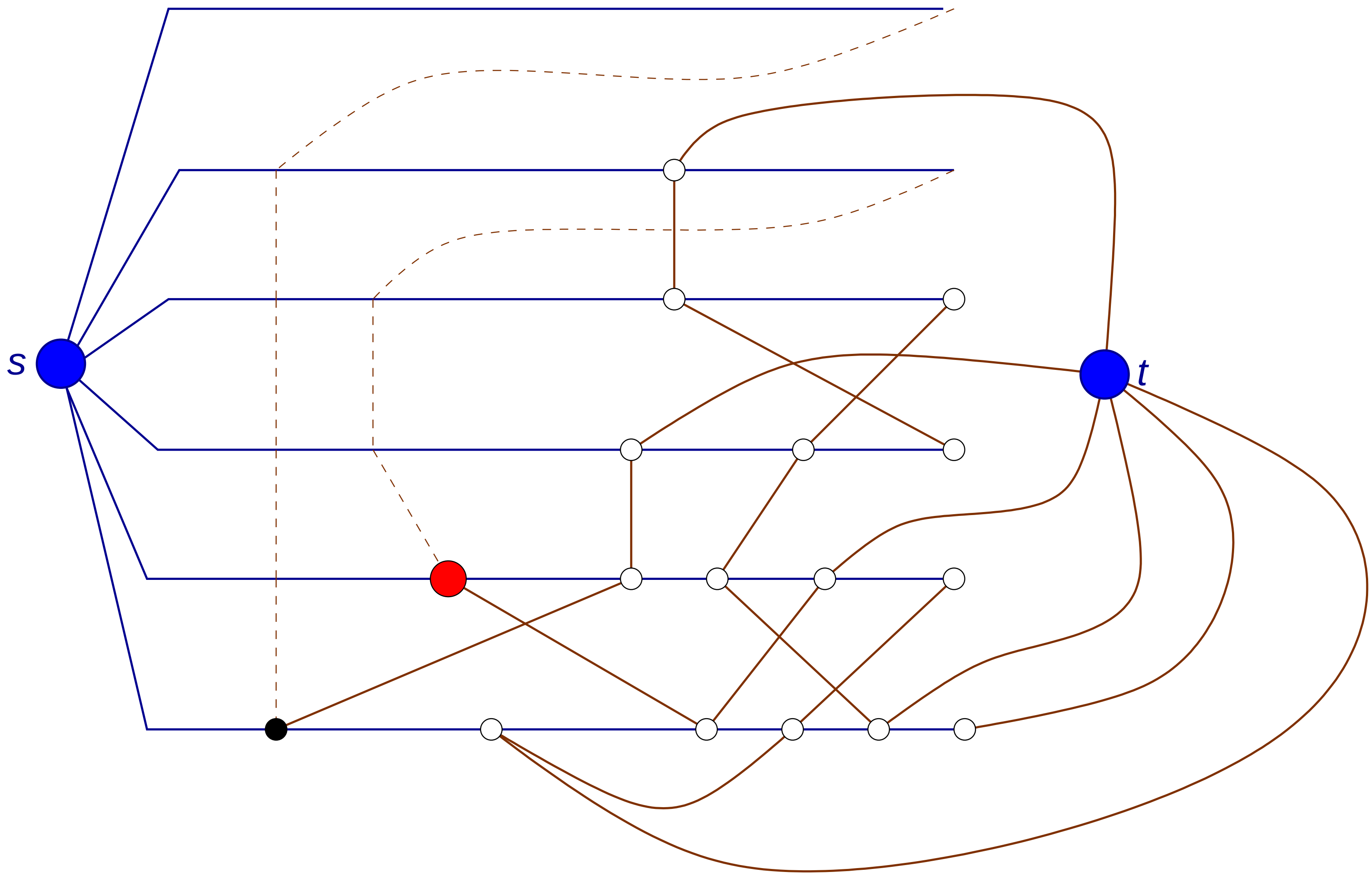
Recombination



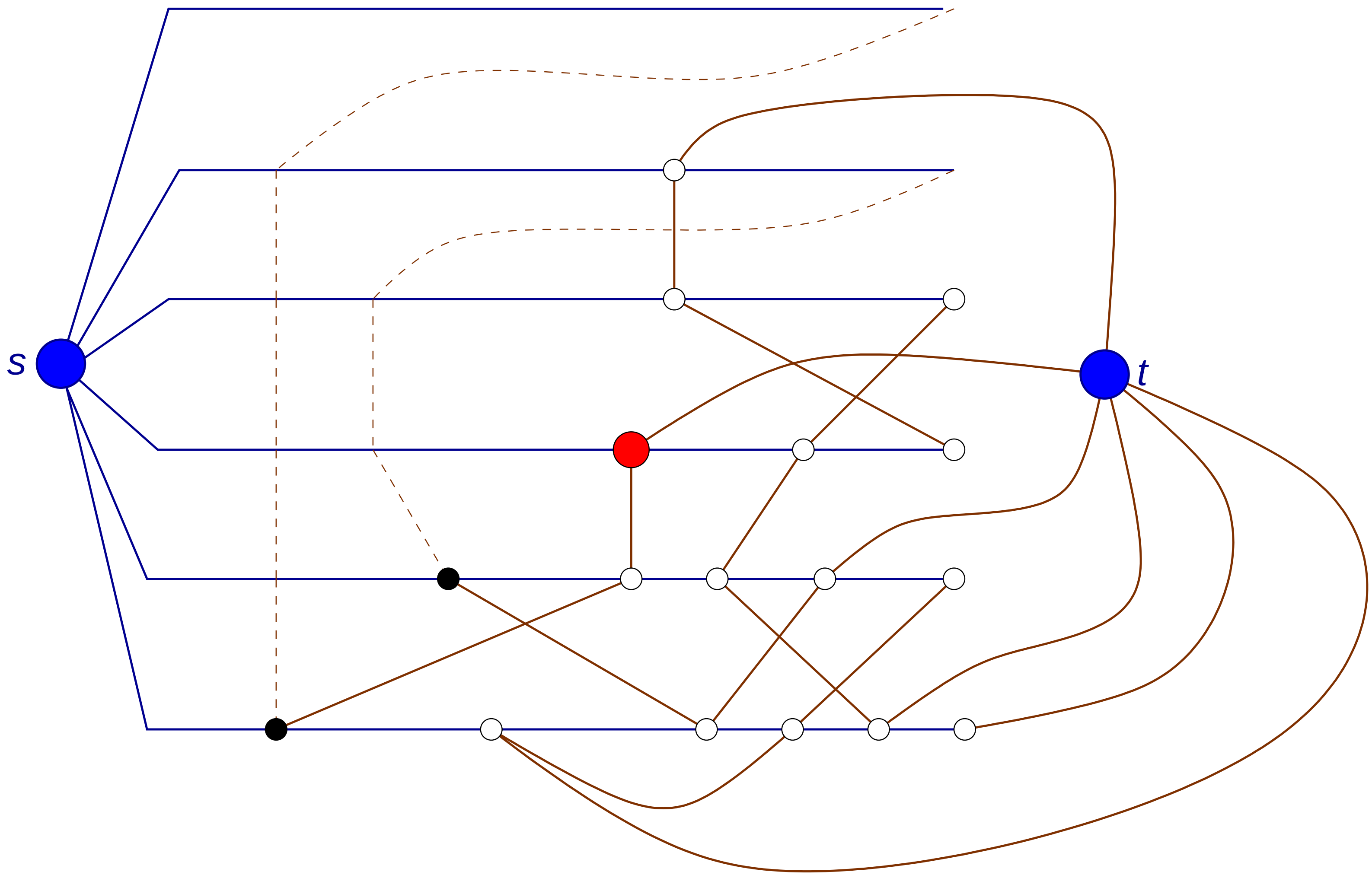
Recombination



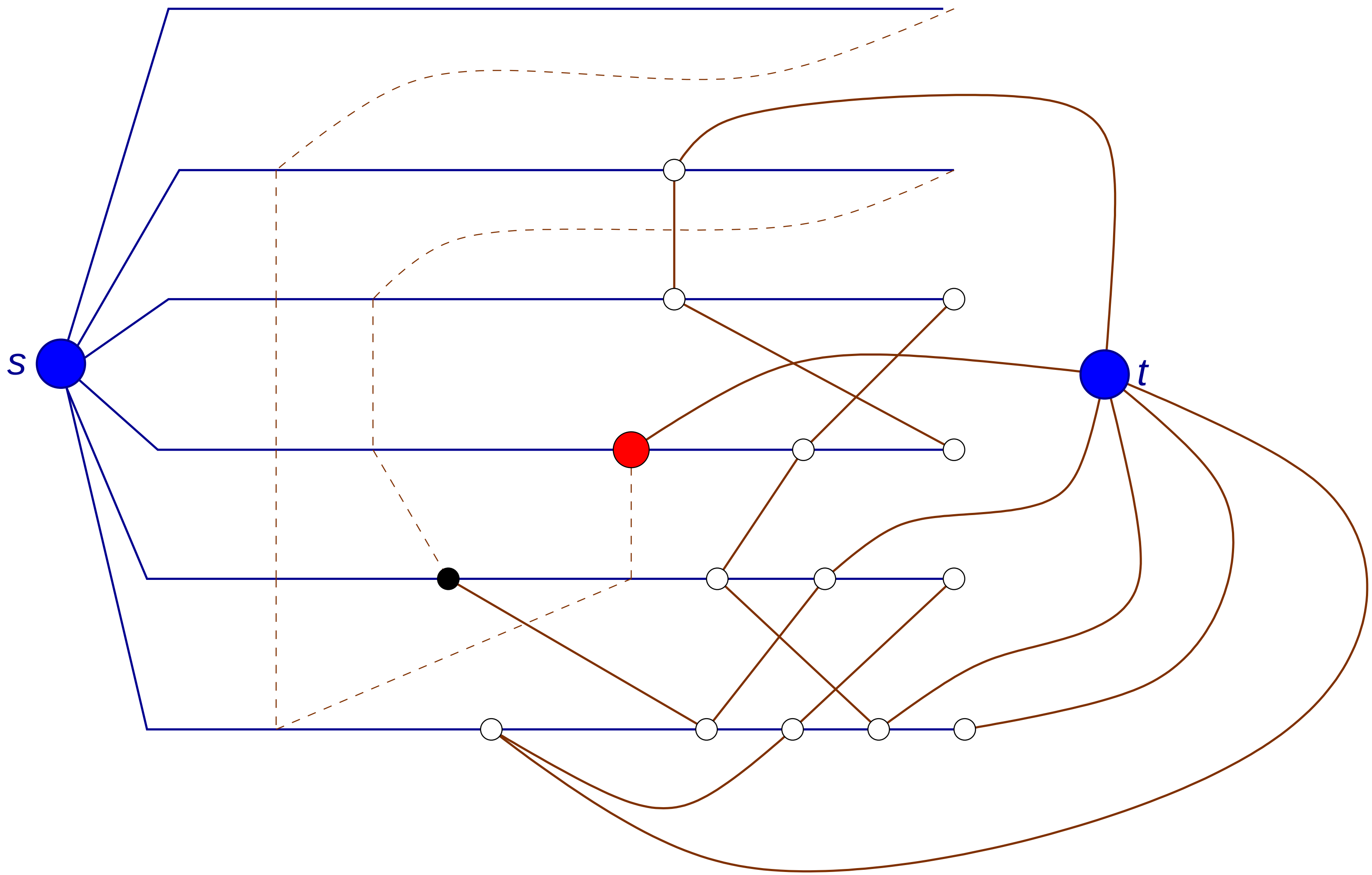
Recombination



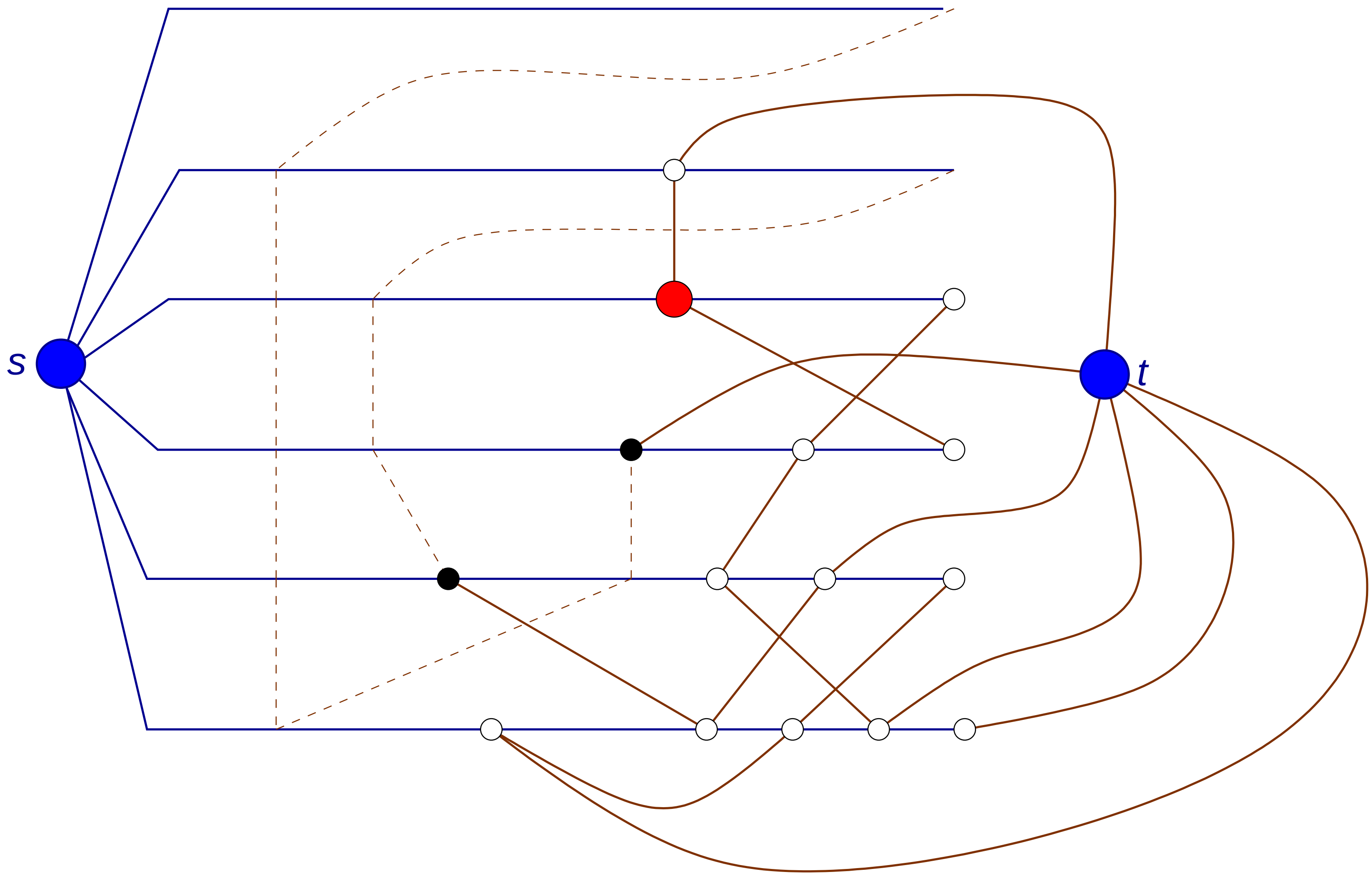
Recombination



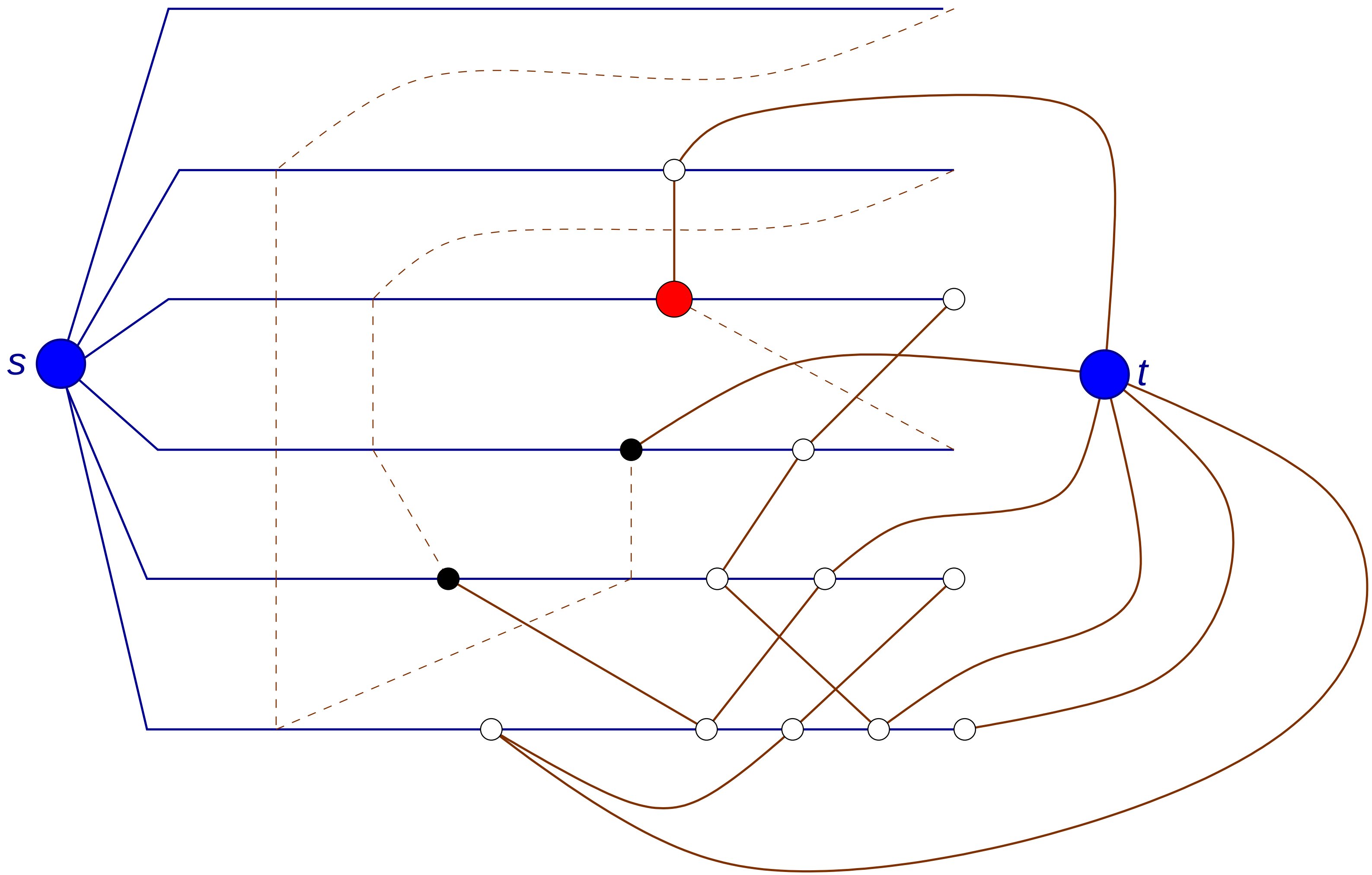
Recombination



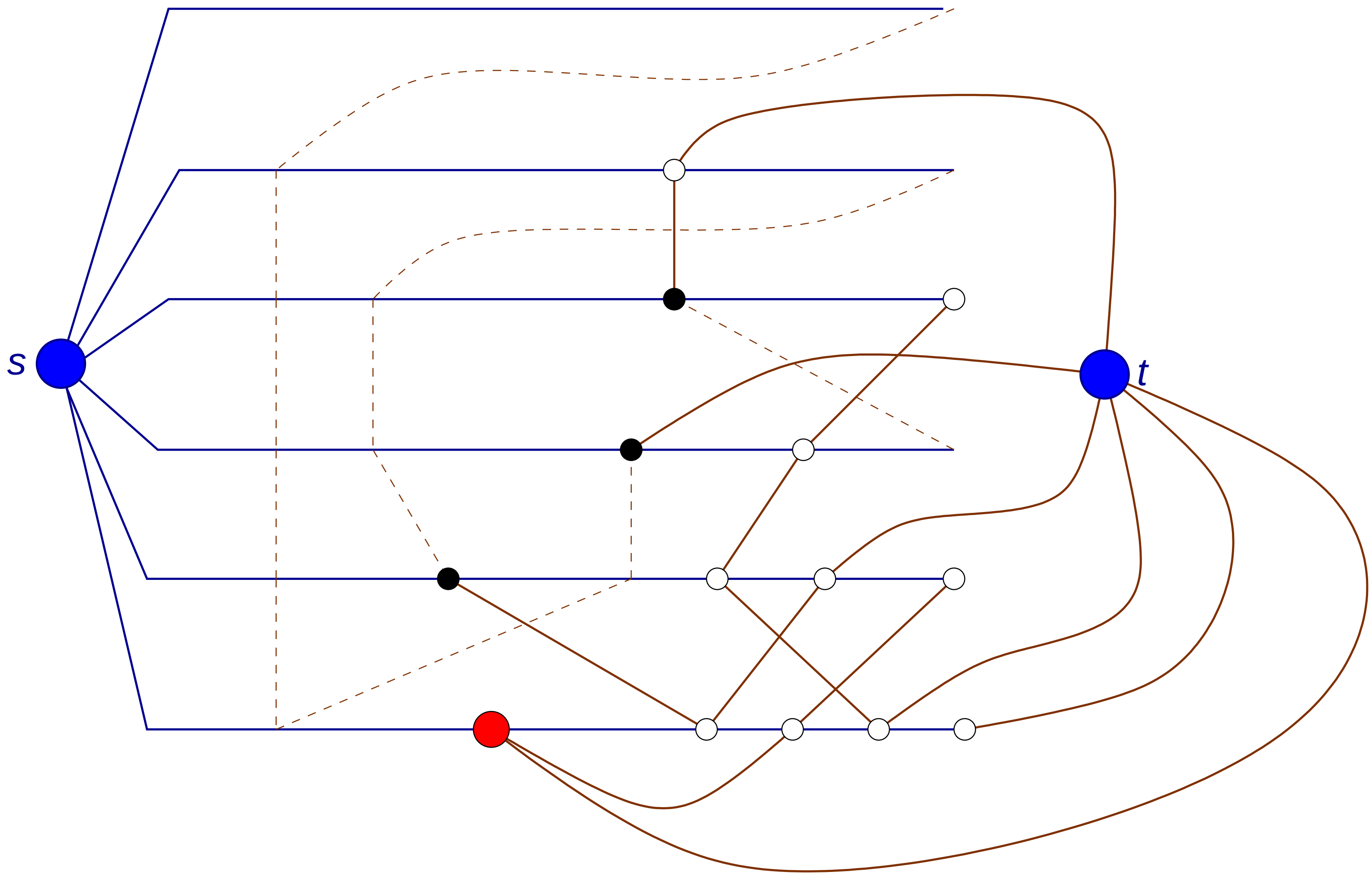
Recombination



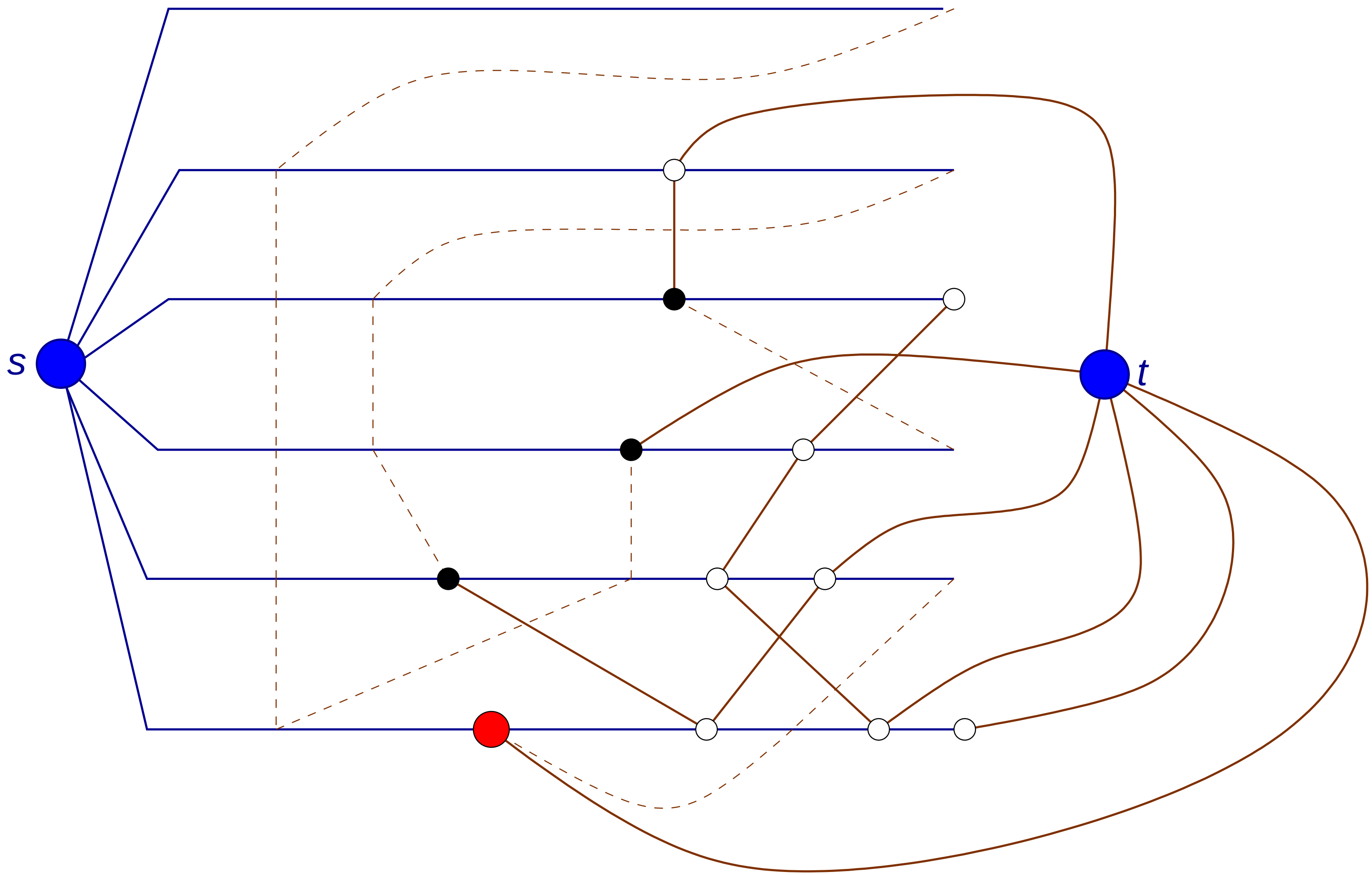
Recombination



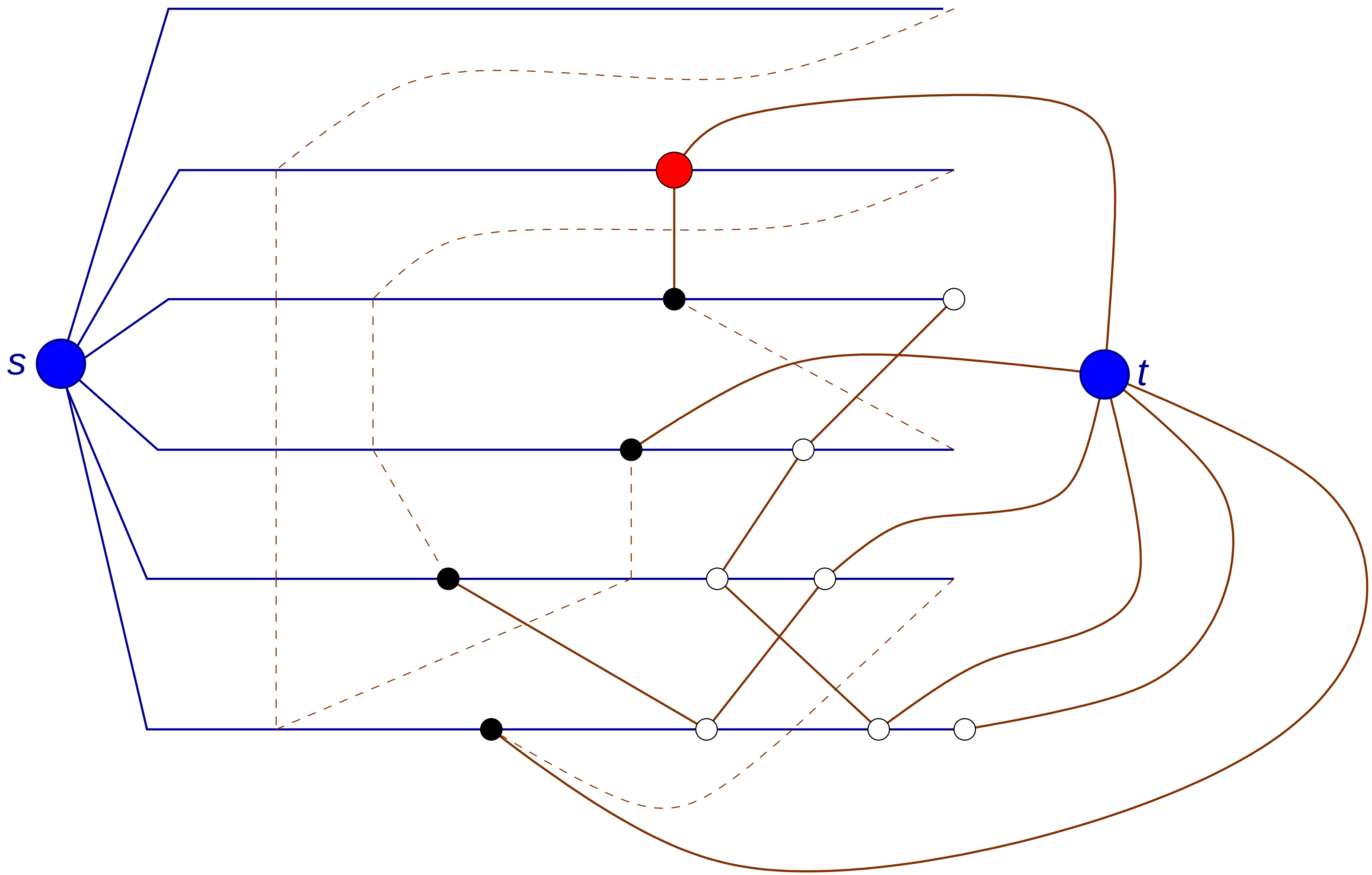
Recombination



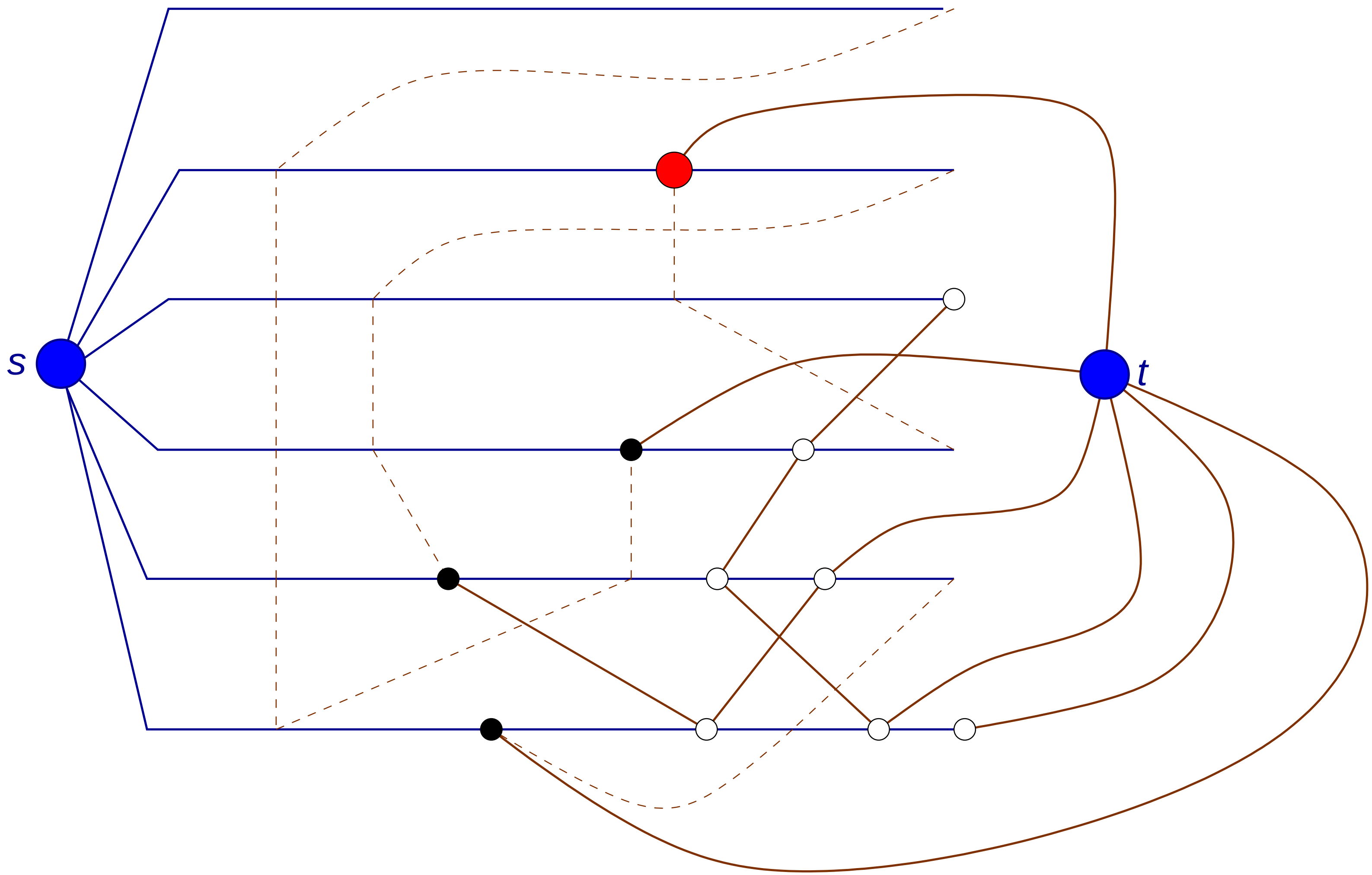
Recombination



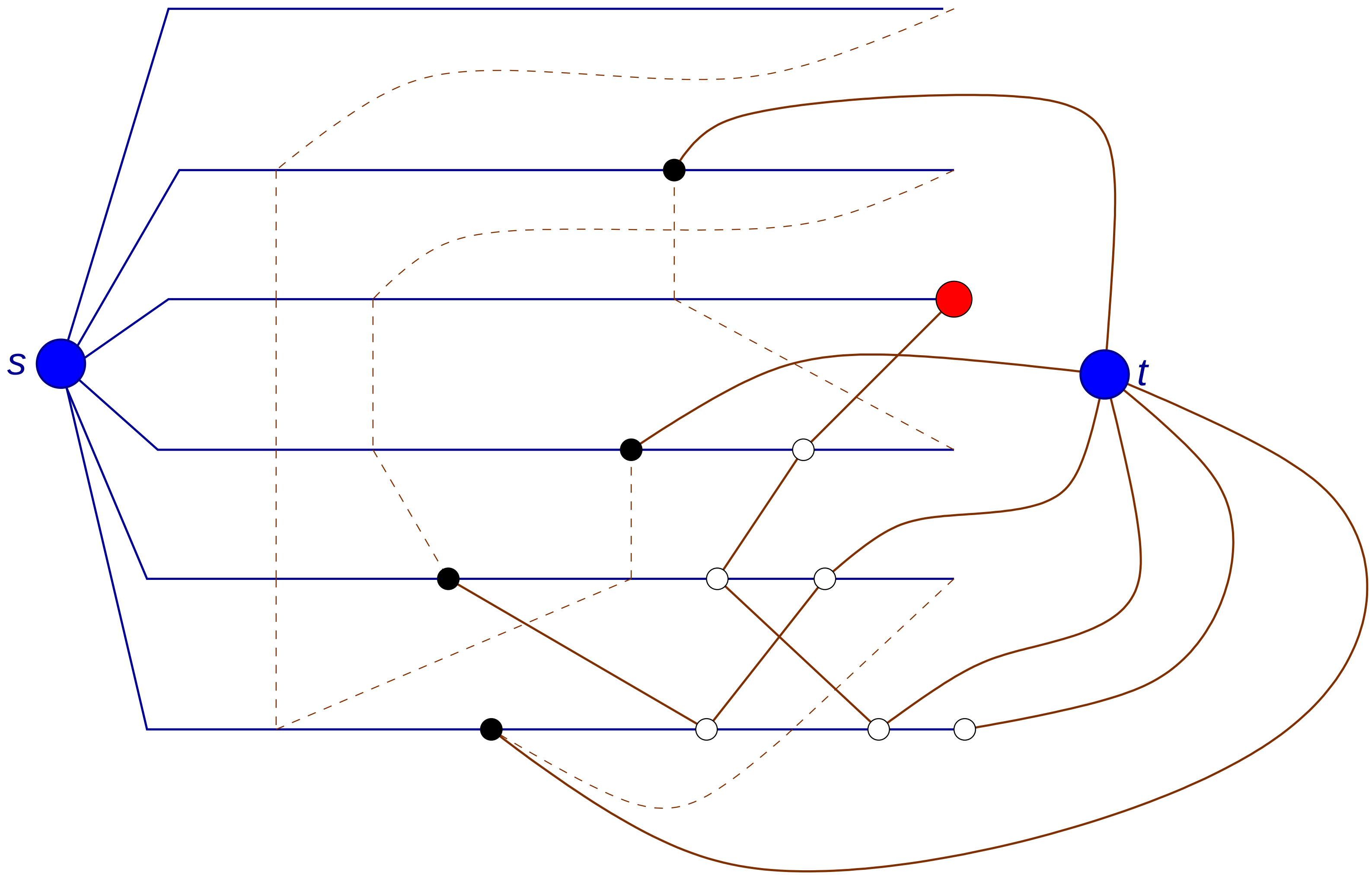
Recombination



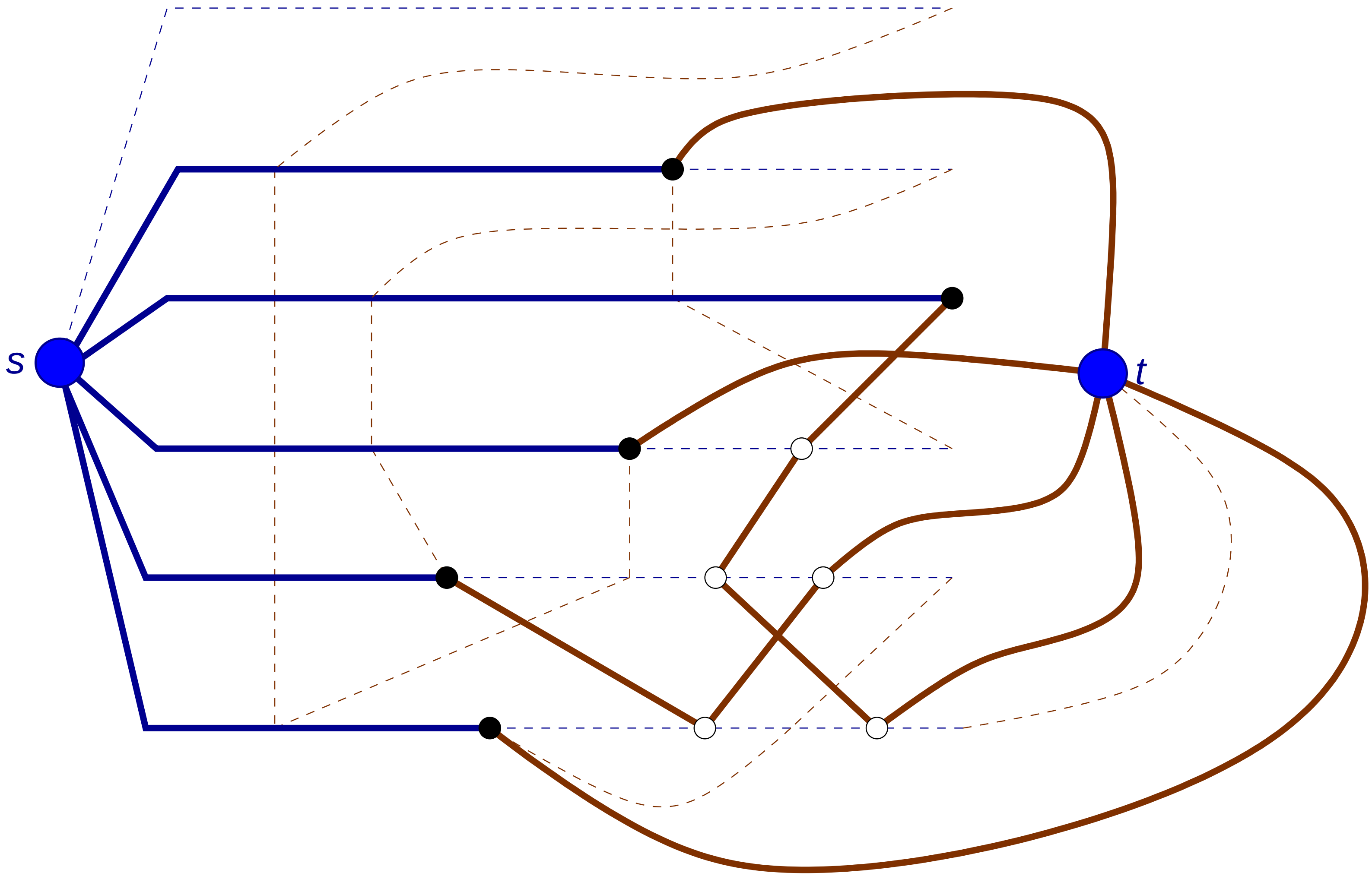
Recombination



Recombination



Recombination



Summary of Results

For arbitrary directed graphs, valley-free path model:

	Min s-t-Cut	Max Disjoint s-t-Paths
vertex version	APX-hard 2-approx	no $(2-\varepsilon)$-apx unless $P = NP$ 2-approx
edge version	polynomial	no $(2-\varepsilon)$-apx unless $P = NP$ 2-approx

(plus some additional results for DAGs)

Remark. Interesting cut and disjoint paths problems arise also from paths with other restrictions (e.g. length-bounded paths).

Network Discovery and Verification

General Setting

- **Discover** information about an unknown network using **queries**.
- **Verify** information about a network using **queries**.
- Here, “network” means connected, undirected graph.
- **Motivation:** Internet mapping; discovering the link structure of peer-to-peer networks.

Two Problems

Network Discovery:

- **Task:** Identify all edges and non-edges of the network using a small number of queries.
- On-line problem (incomplete information), competitive analysis

Network Verification:

- **Task:** Check whether an existing network “map” is correct, using a small number of queries.
- Off-line problem (full information), approximation algorithms

Query Models

Layered-Graph (LG) Query Model

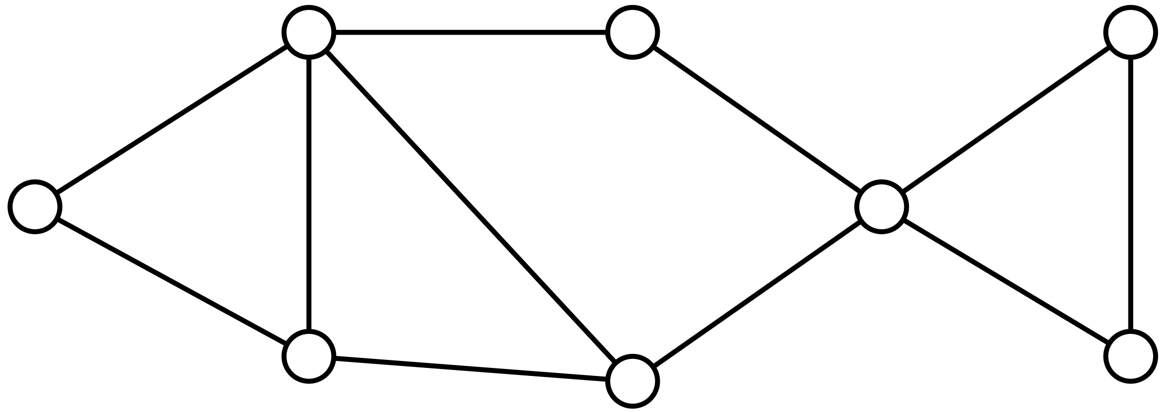
- Connected graph $G = (V, E)$ with $|V| = n$ (in the on-line case, only V is known in advance)
- Query at node $v \in V$ yields the subgraph containing all shortest paths from v to all other nodes of G.
- Problem LG-ALL-DISCOVERY (LG-ALL-VERIFICATON): Minimize the number of queries required to discover (verify) all edges and non-edges of G.

Layered-Graph (LG) Query Model

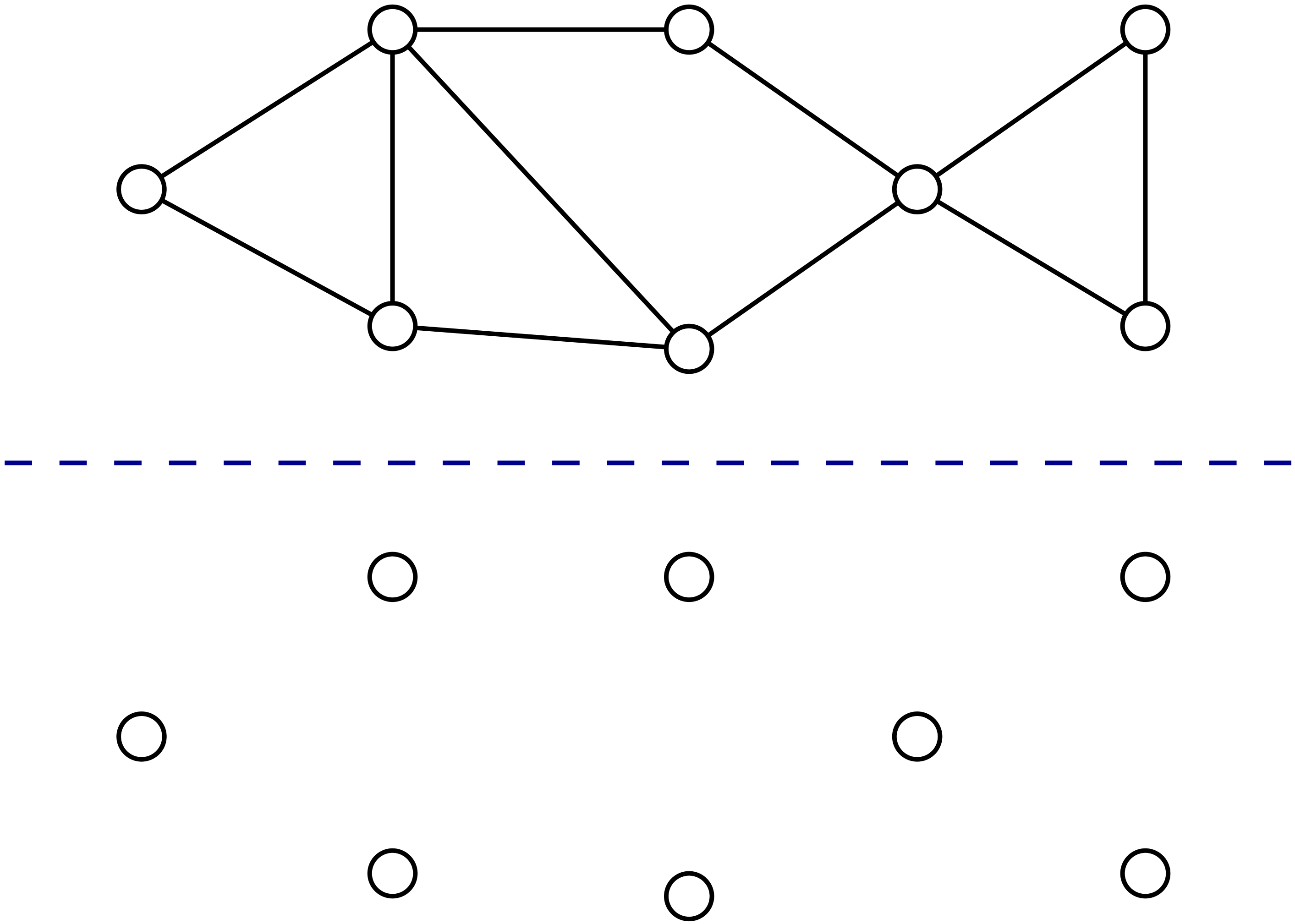
- Connected graph $G = (V, E)$ with $|V| = n$ (in the on-line case, only V is known in advance)
- Query at node $v \in V$ yields the subgraph containing all shortest paths from v to all other nodes of G.
- Problem LG-ALL-DISCOVERY (LG-ALL-VERIFICATON): Minimize the number of queries required to discover (verify) all edges and non-edges of G.

Observation. Query at v discovers all edges and non-edges between vertices with different distance from v.

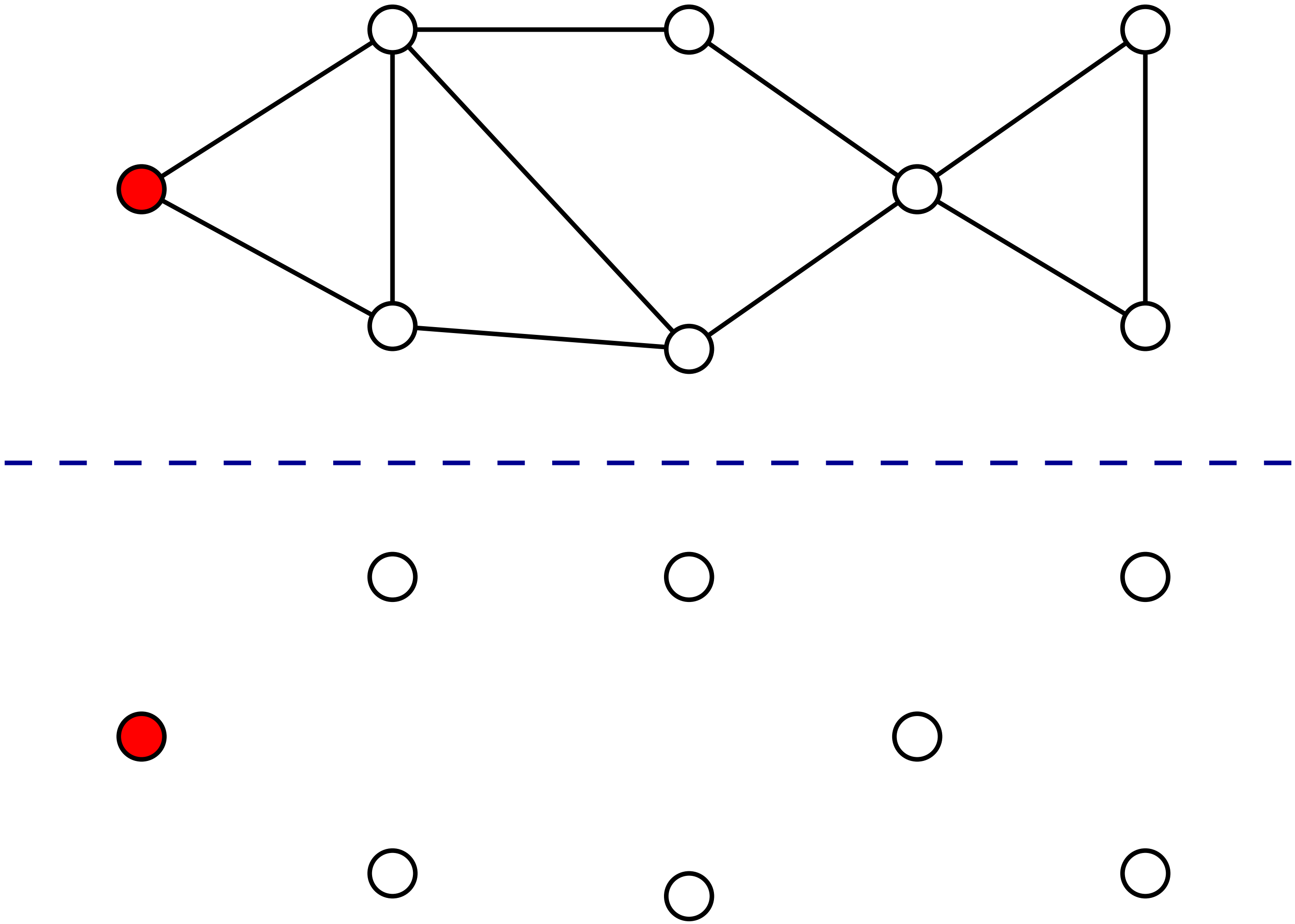
Layered-Graph Query Example



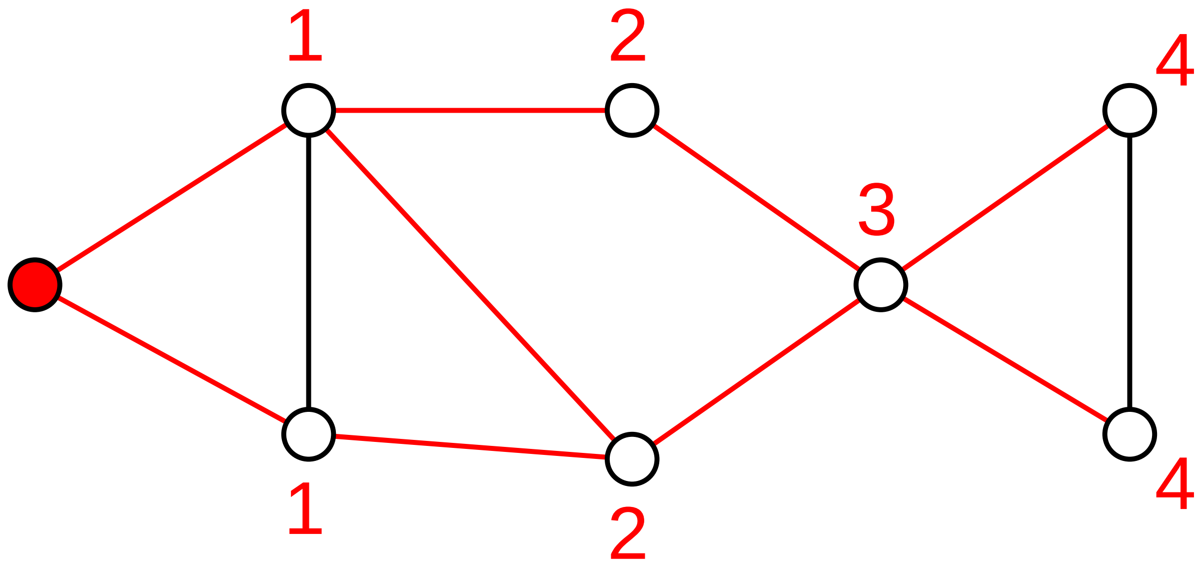
Layered-Graph Query Example

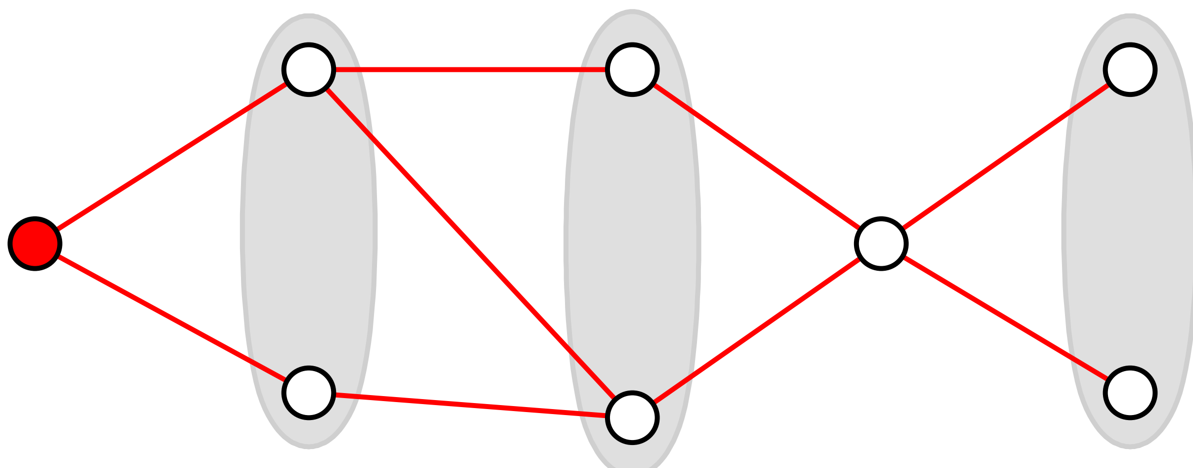


Layered-Graph Query Example

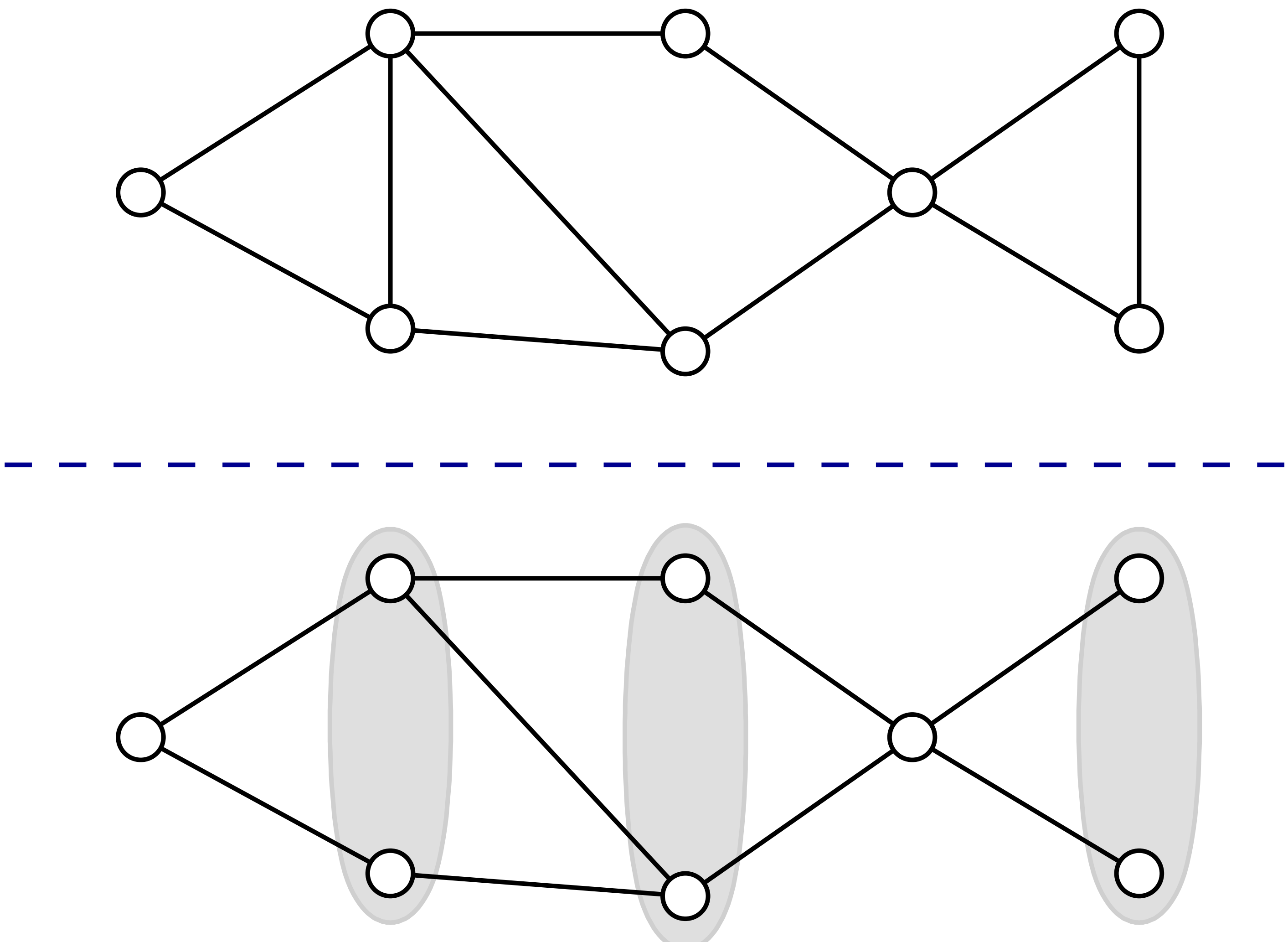


Layered-Graph Query Example

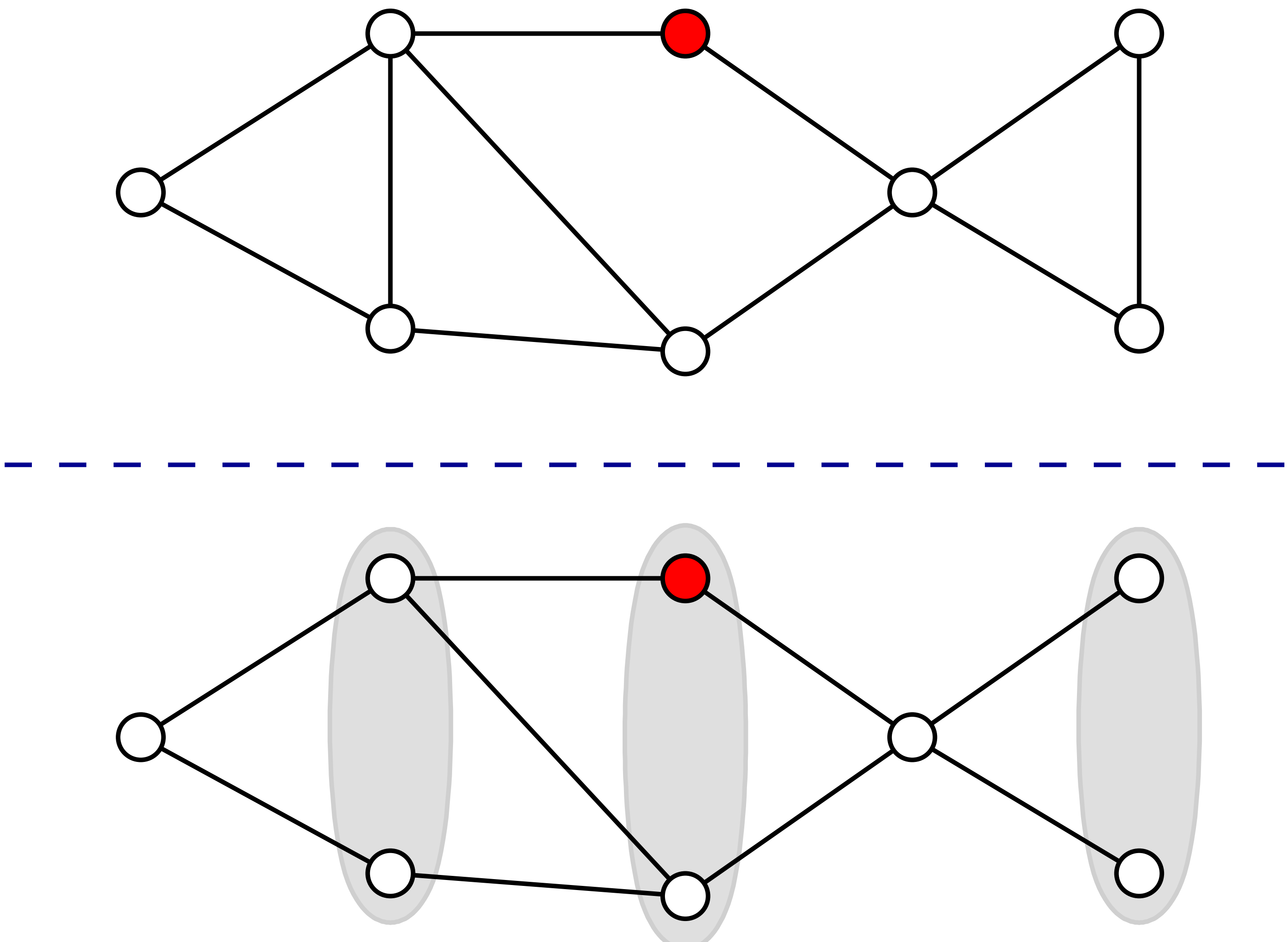




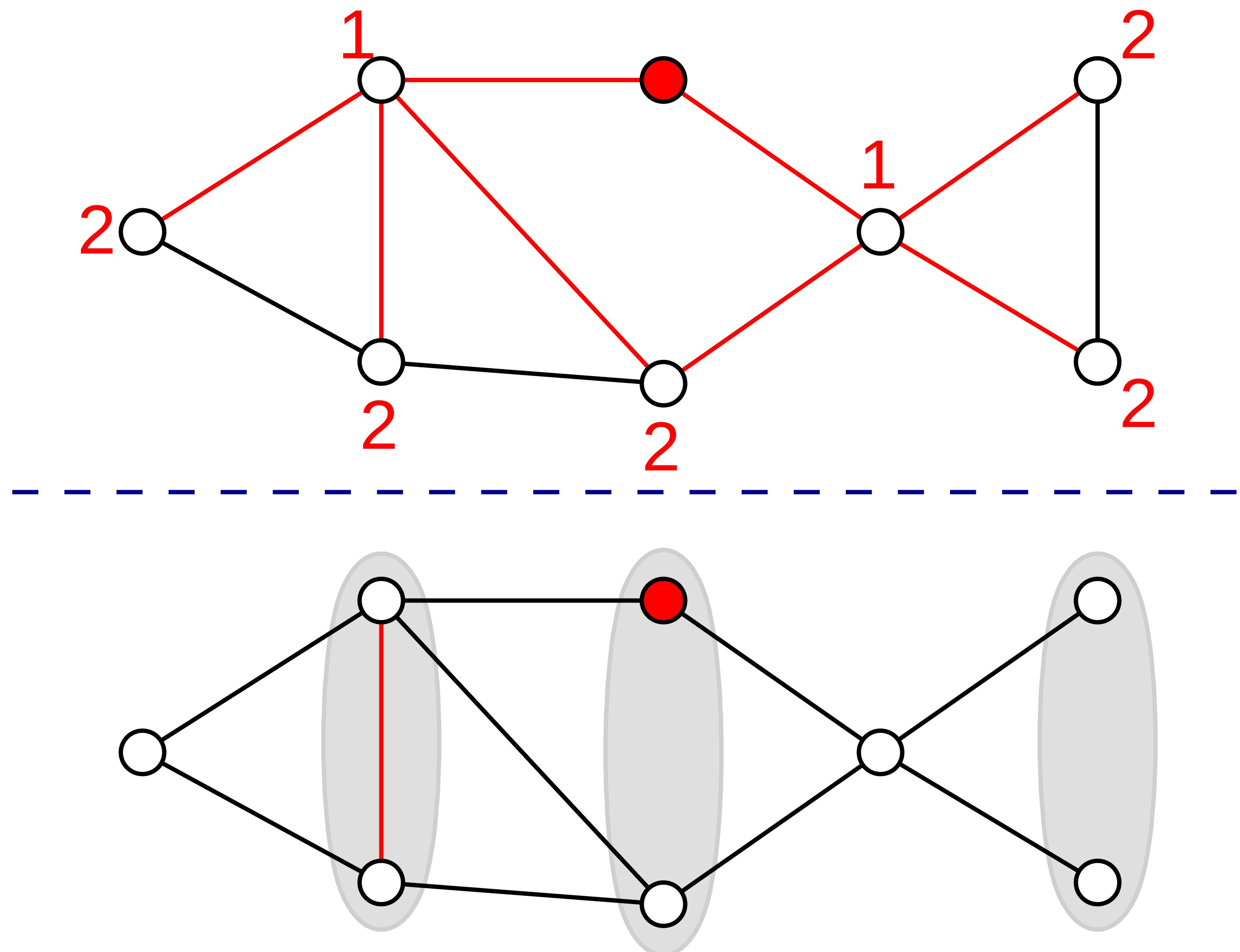
Layered-Graph Query Example



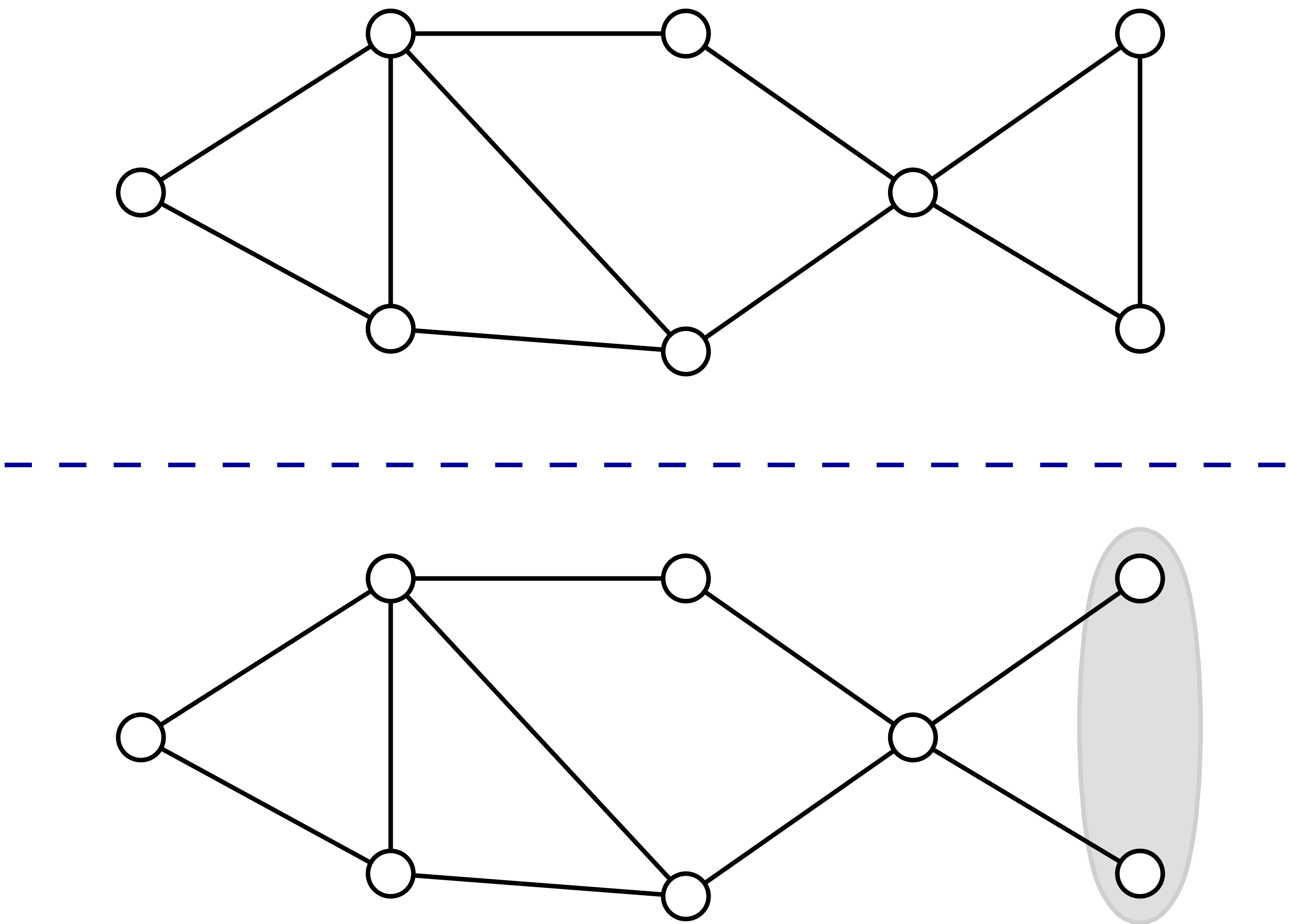
Layered-Graph Query Example



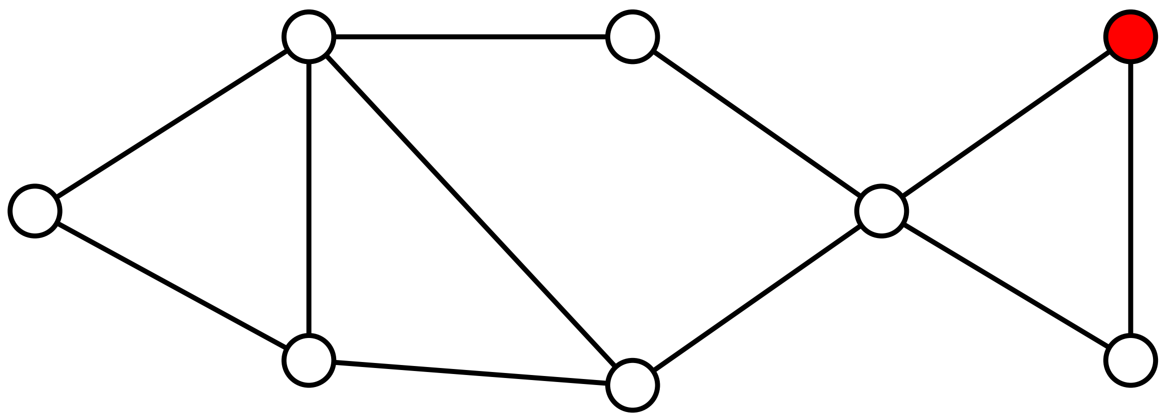
Layered-Graph Query Example

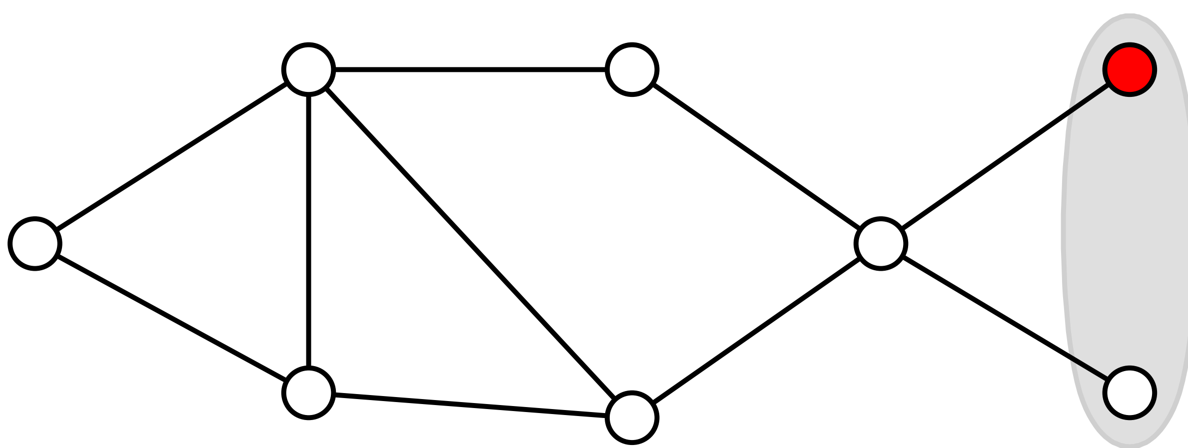


Layered-Graph Query Example

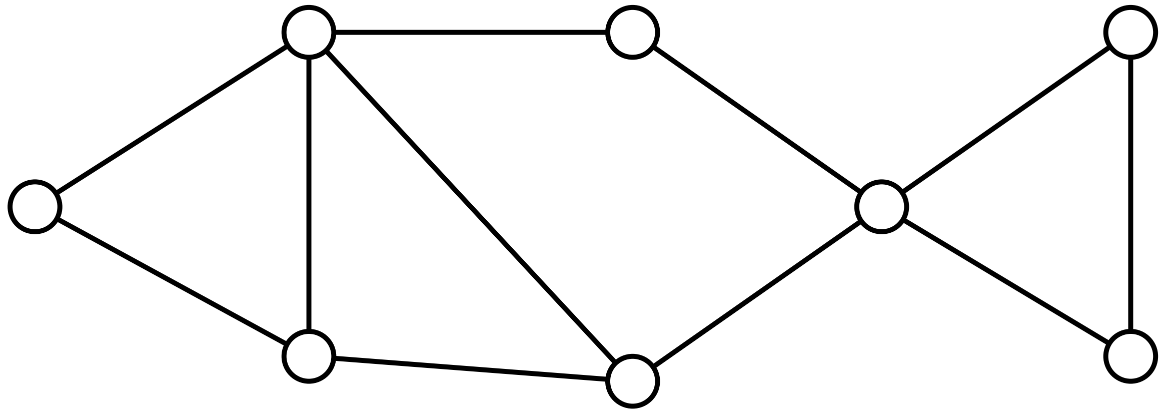


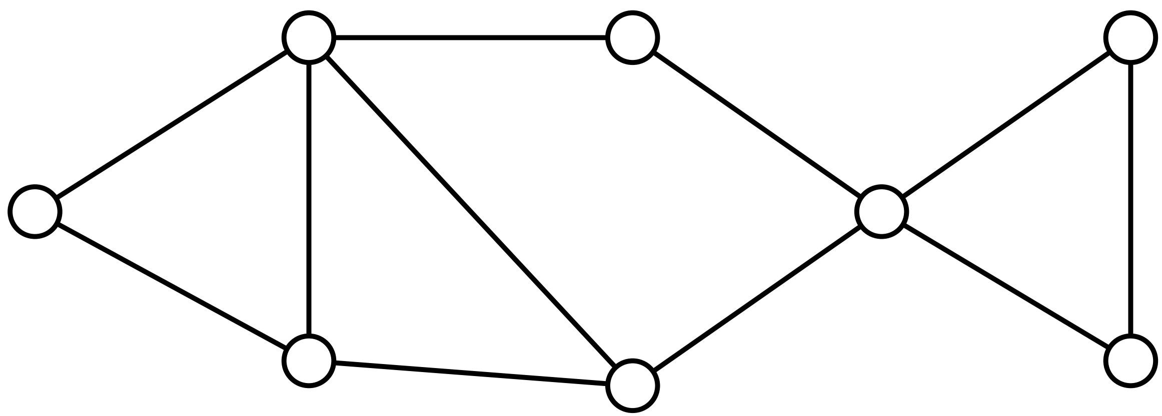
Layered-Graph Query Example



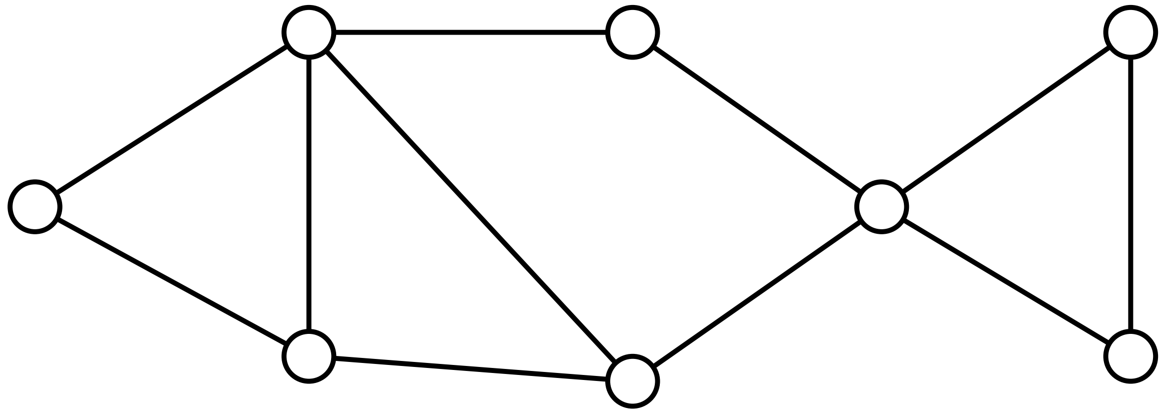


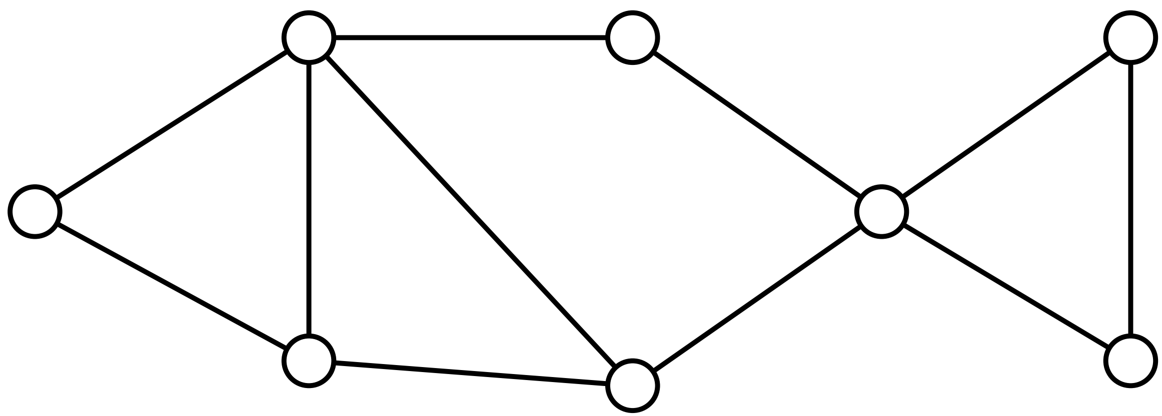
Layered-Graph Query Example





Layered-Graph Query Example



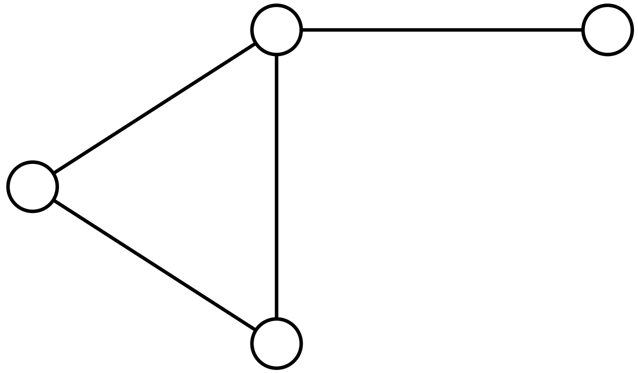


Three queries are sufficient!

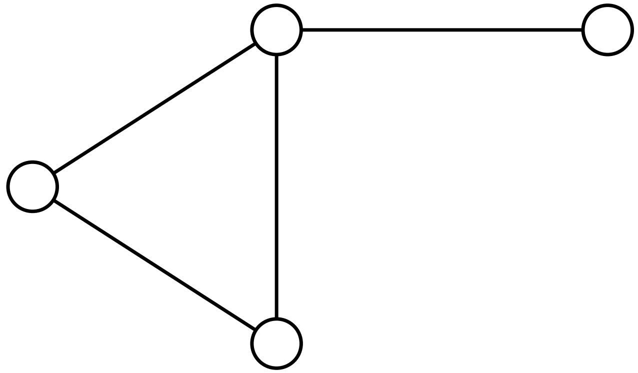
Distance (D) Query Model

- Connected graph $G = (V, E)$ with $|V| = n$ (in the on-line case, only V is known in advance)
- Query at node $v \in V$ yields the distances between v and all other nodes of G.
- Problem D-ALL-DISCOVERY (D-ALL-VERIFICATON): Minimize the number of queries required to discover (verify) all edges and non-edges of G.

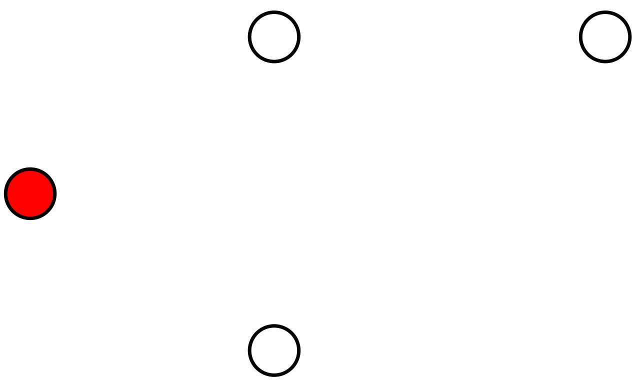
Distance Query Example



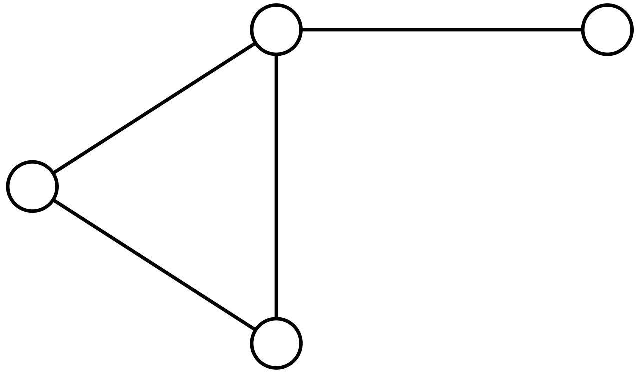
Distance Query Example



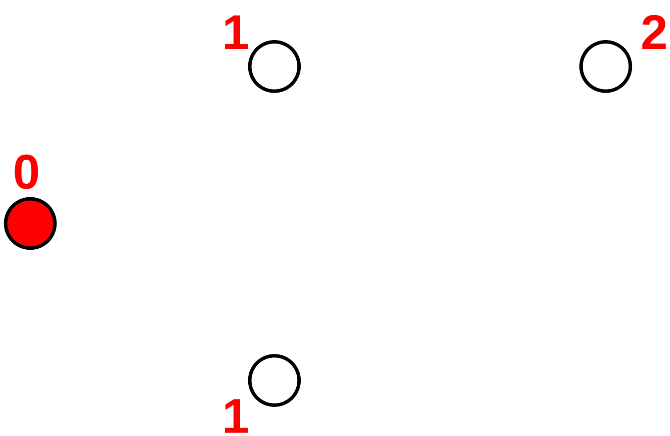
Query 1:



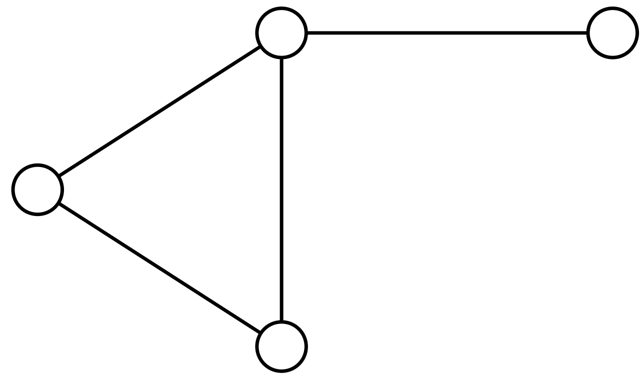
Distance Query Example



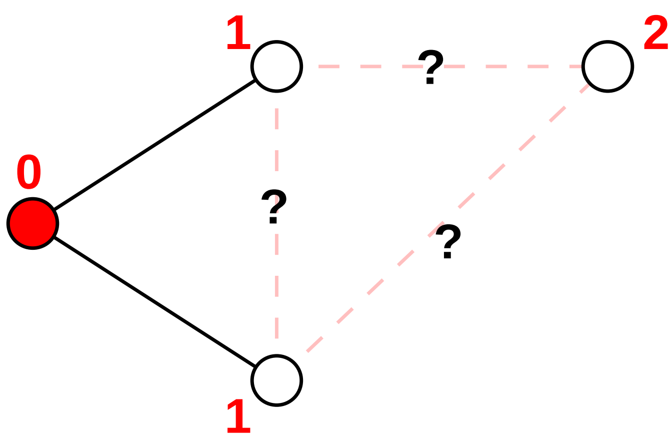
Query 1:



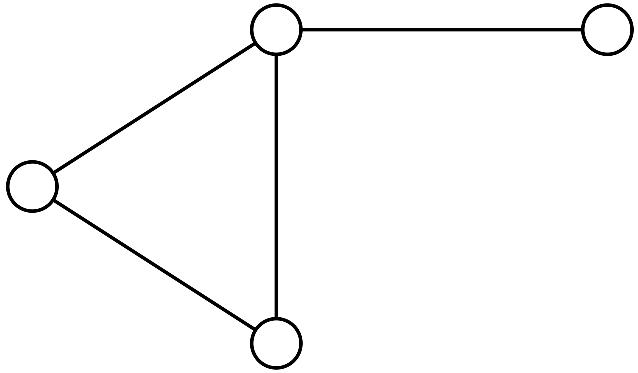
Distance Query Example

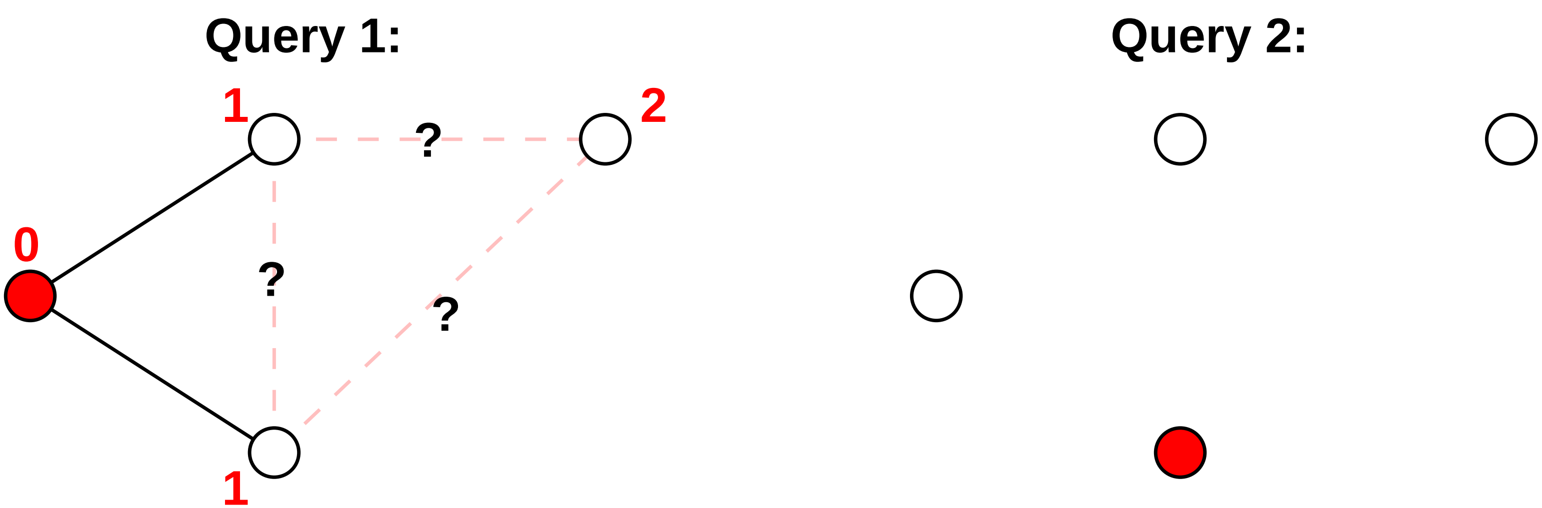


Query 1:

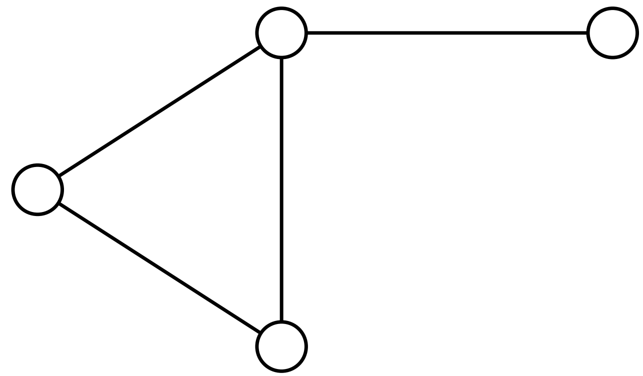


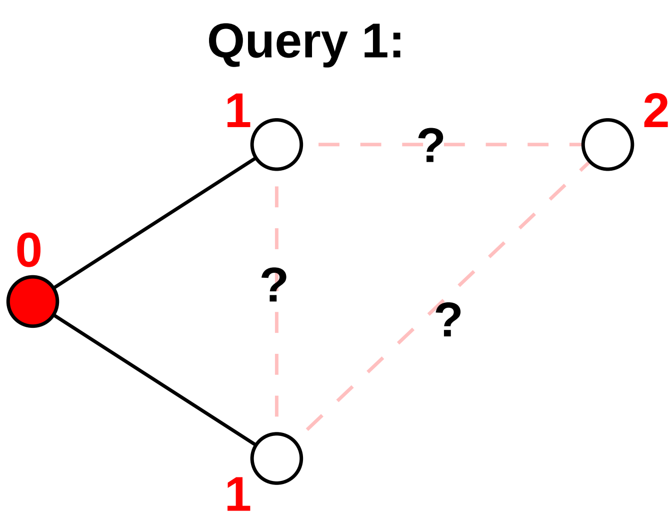
Distance Query Example

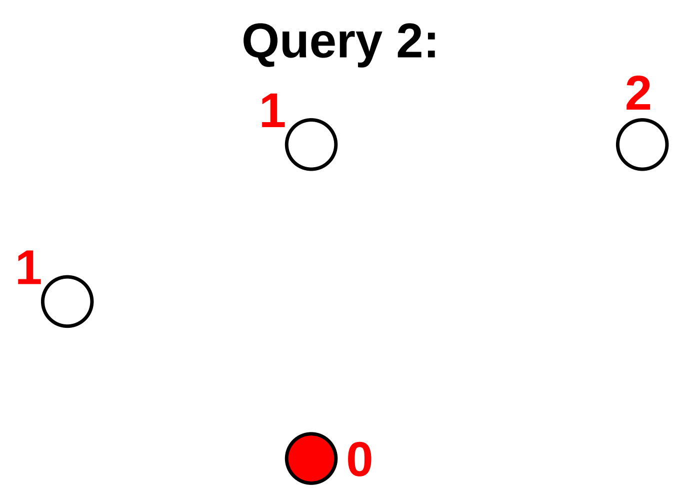




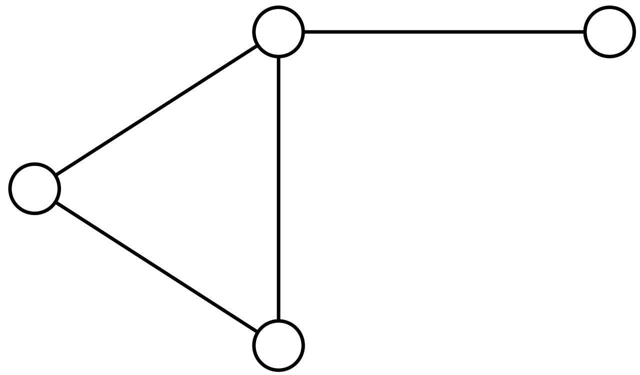
Distance Query Example



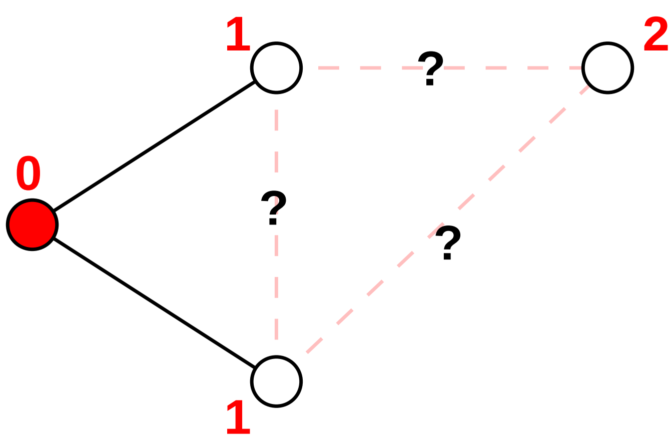




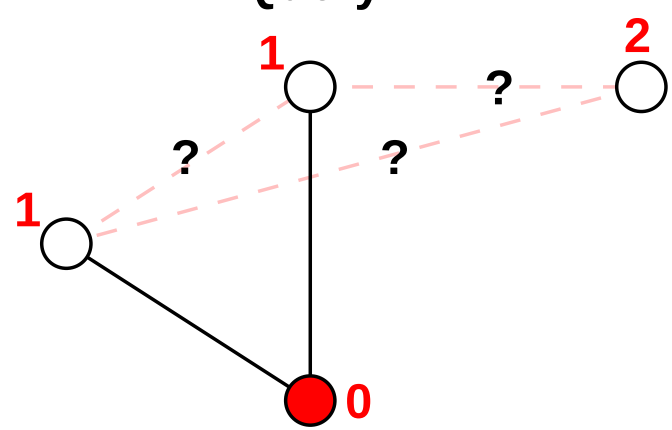
Distance Query Example



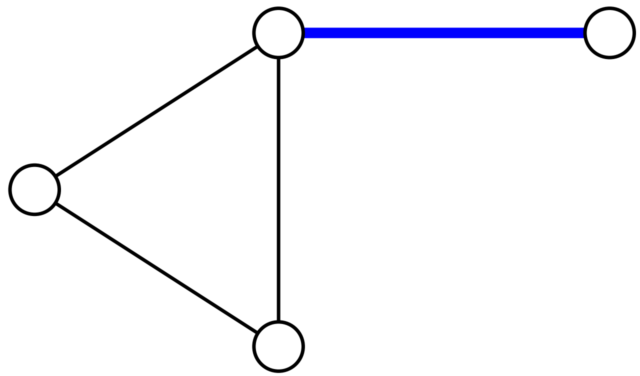
Query 1:

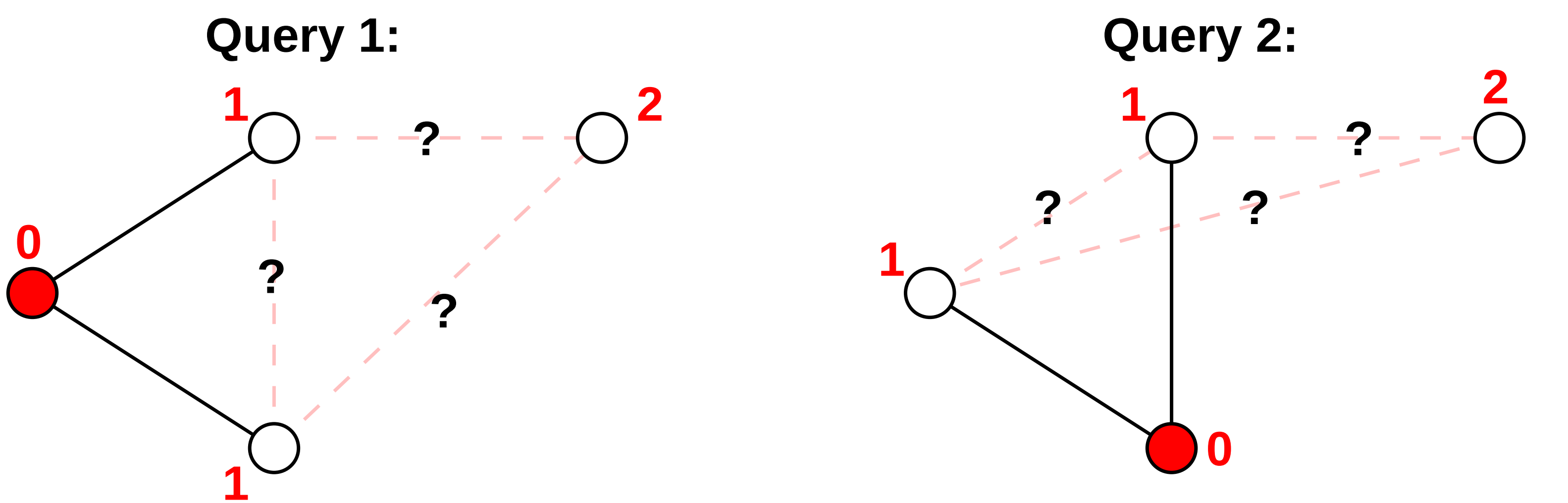


Query 2:



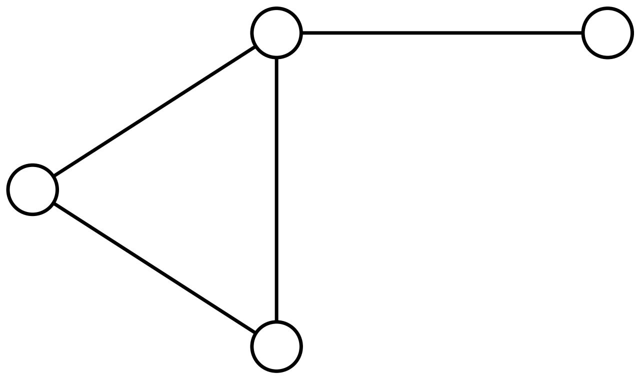
Distance Query Example

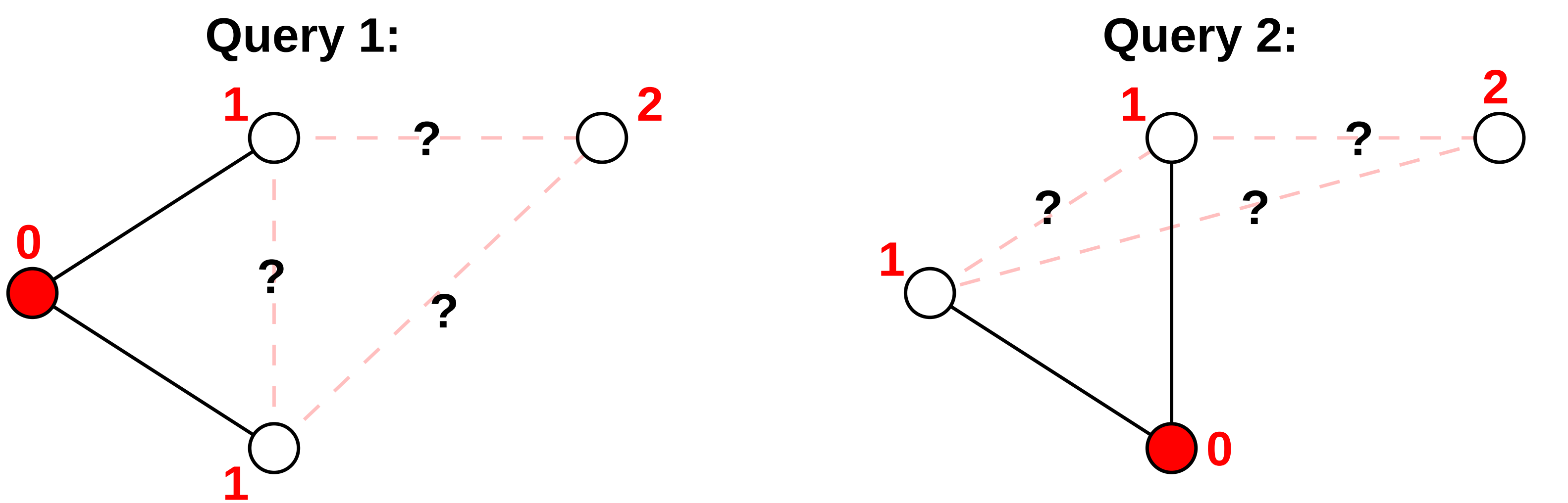




Blue edge is discovered by combination of queries!

Distance Query Example





Two queries are sufficient!

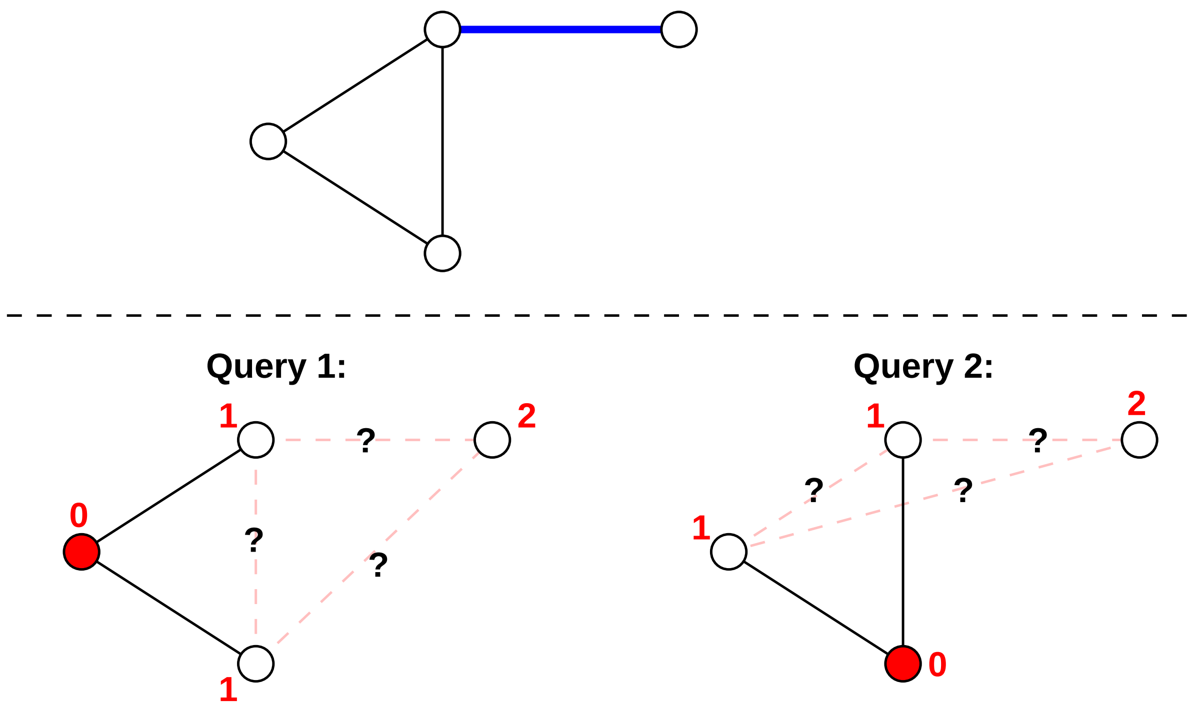
Results for LG Query Model

- **LG-ALL-DISCOVERY:**
  - No deterministic algorithm can be better than 3-competitive.
  - There is a randomized algorithm that is $O(\sqrt{n \log n})$-competitive.

- **LG-ALL-VERIFICATION:**
  - Optimal number of queries is equivalent to metric dimension of the graph.
  - NP-hard to approximate within $o(\log n)$
  - $O(\log n)$-approximation using greedy set cover algorithm [Khuller et al., 1996]

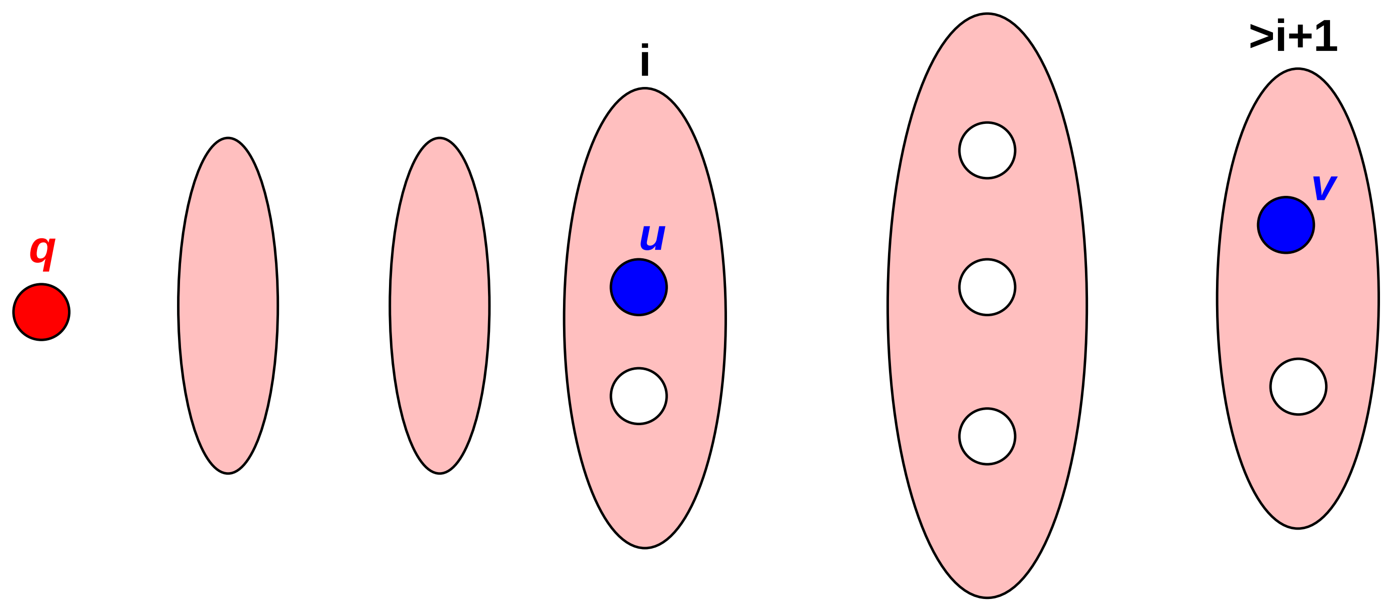
Distance Query Model

- A query at v discovers the distances to all other nodes.
- For the LG model, the edges and non-edges discovered by a set of queries were simply the union of those discovered by the individual queries. This is **not true** for edges in the distance query model!



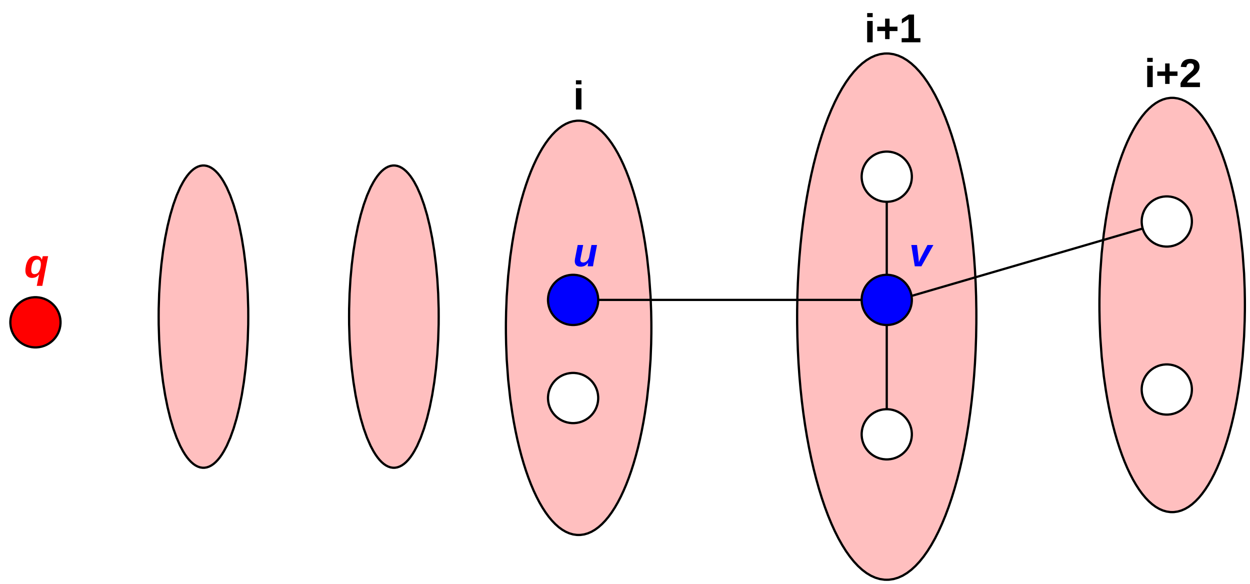
Discovering Non-edges in the D Model

Lemma. A set Q of queries discovers a non-edge $\{u, v\}$ if and only if there is $q \in Q$ with $|d(q, u) - d(q, v)| \geq 2$.



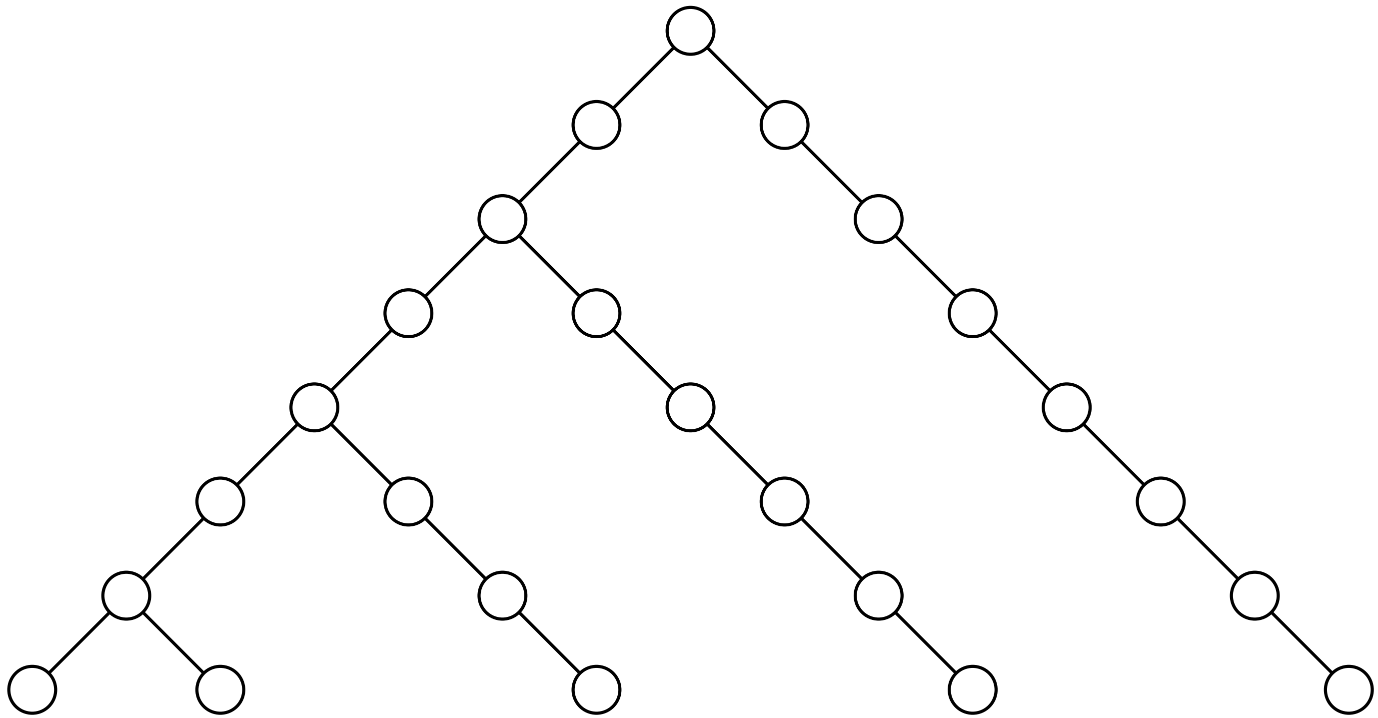
Discovering Edges in the D Model

Definition. A query q is a **partial witness** for edge $\{u, v\}$ if $d(q, u) \neq d(q, v)$ (say, $d(q, u) = i$ and $d(q, v) = i + 1$) and u is the only neighbor of v at distance i from q.

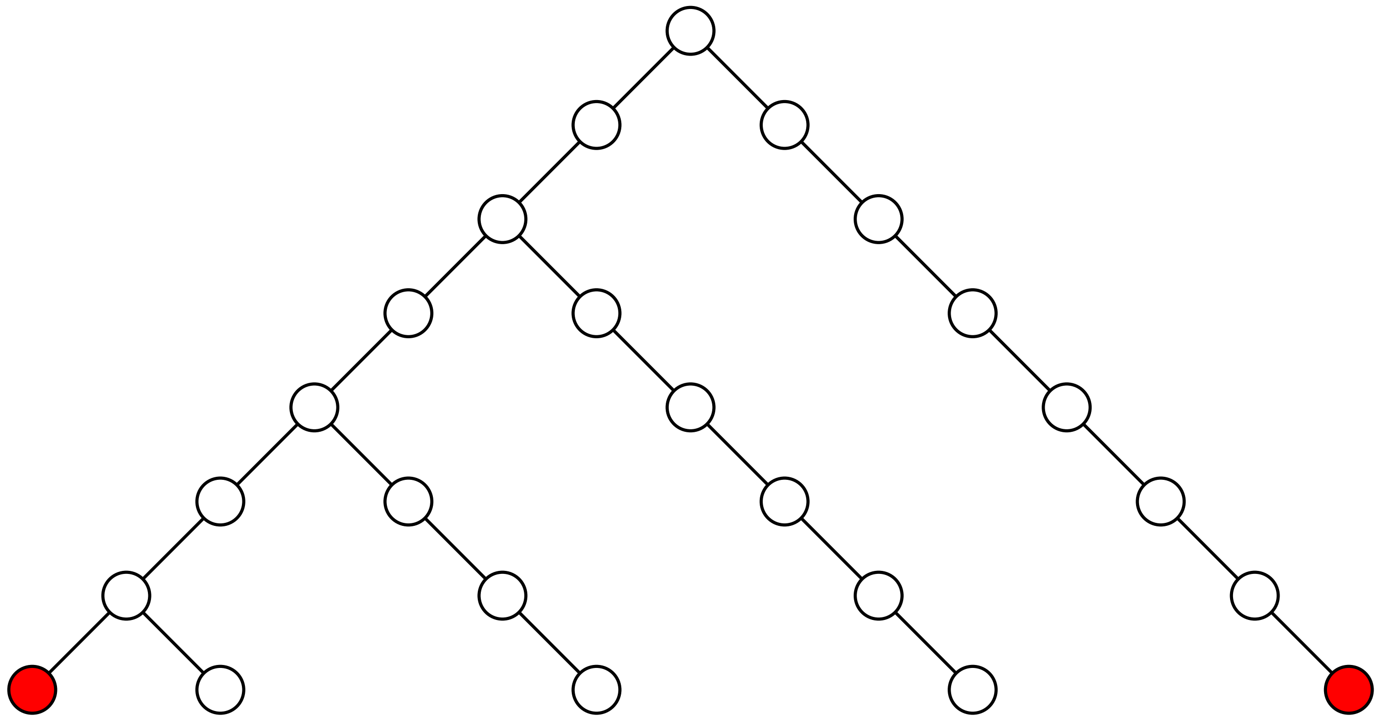


Lemma. A set Q of queries discovers all edges and non-edges of G if and only if it discovers all non-edges and contains a partial witness for each edge.

Competitive Lower Bound

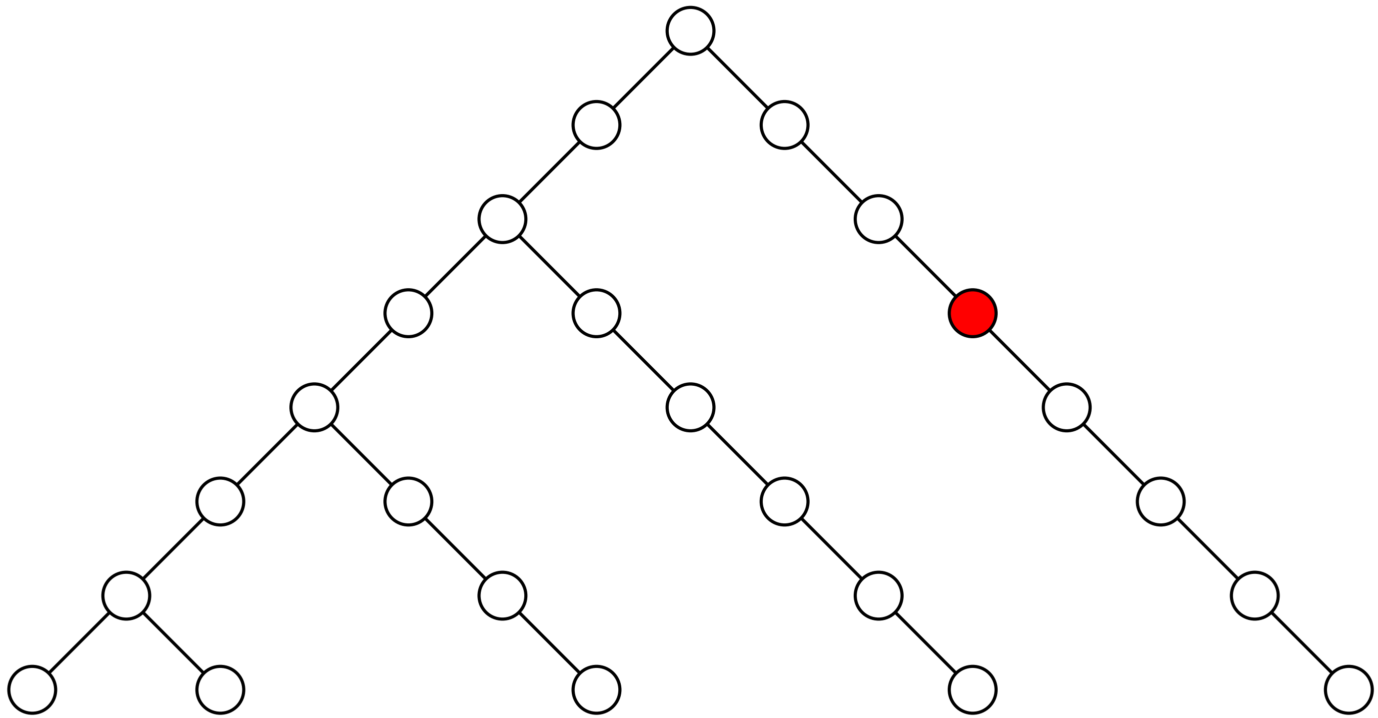


Competitive Lower Bound



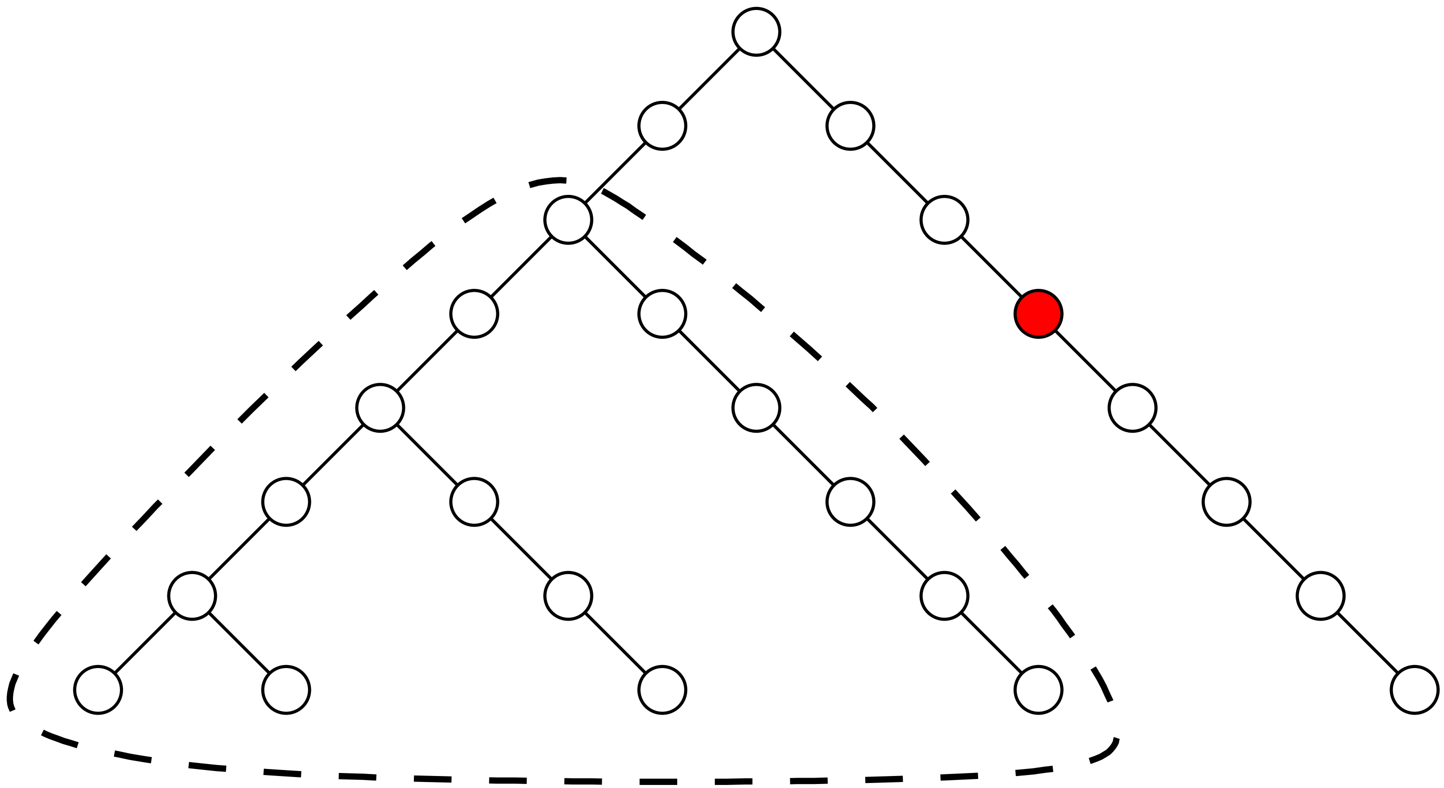
Optimal number of queries: 2

Competitive Lower Bound



Deterministic algorithm: First query in rightmost branch.

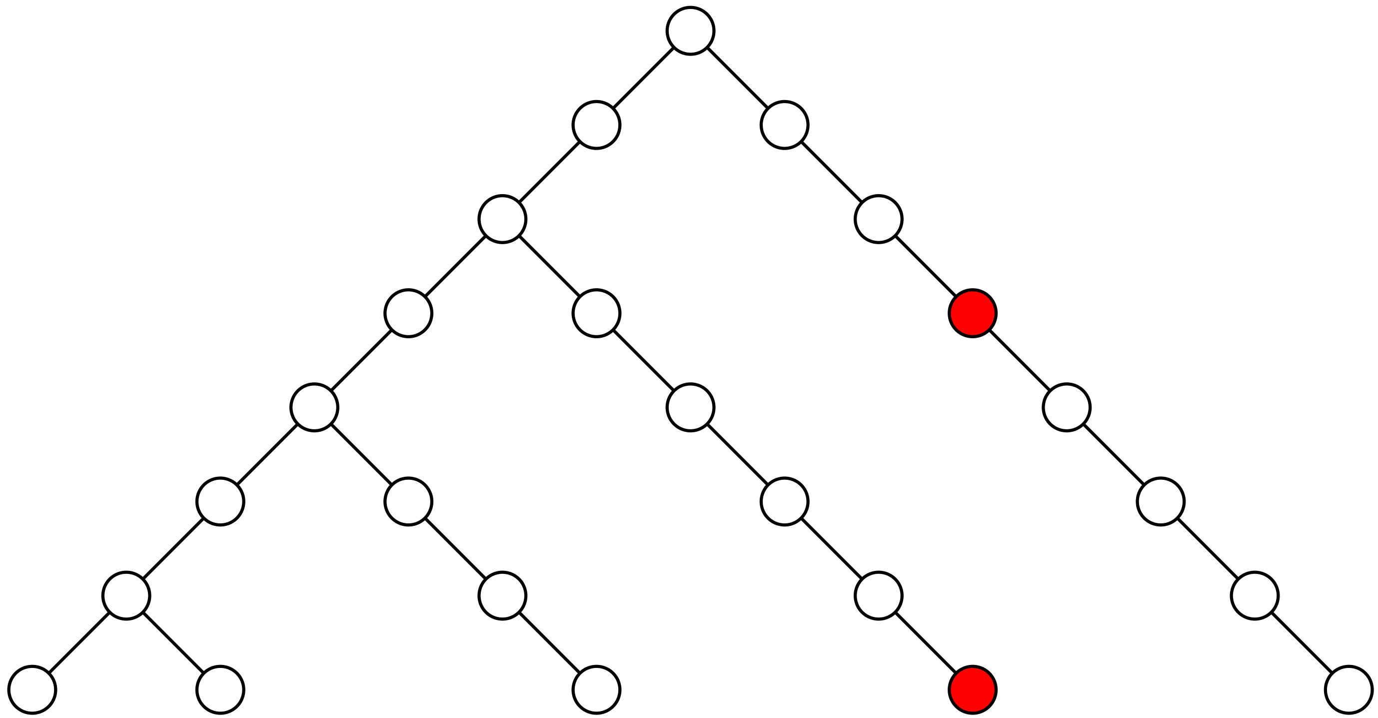
Competitive Lower Bound



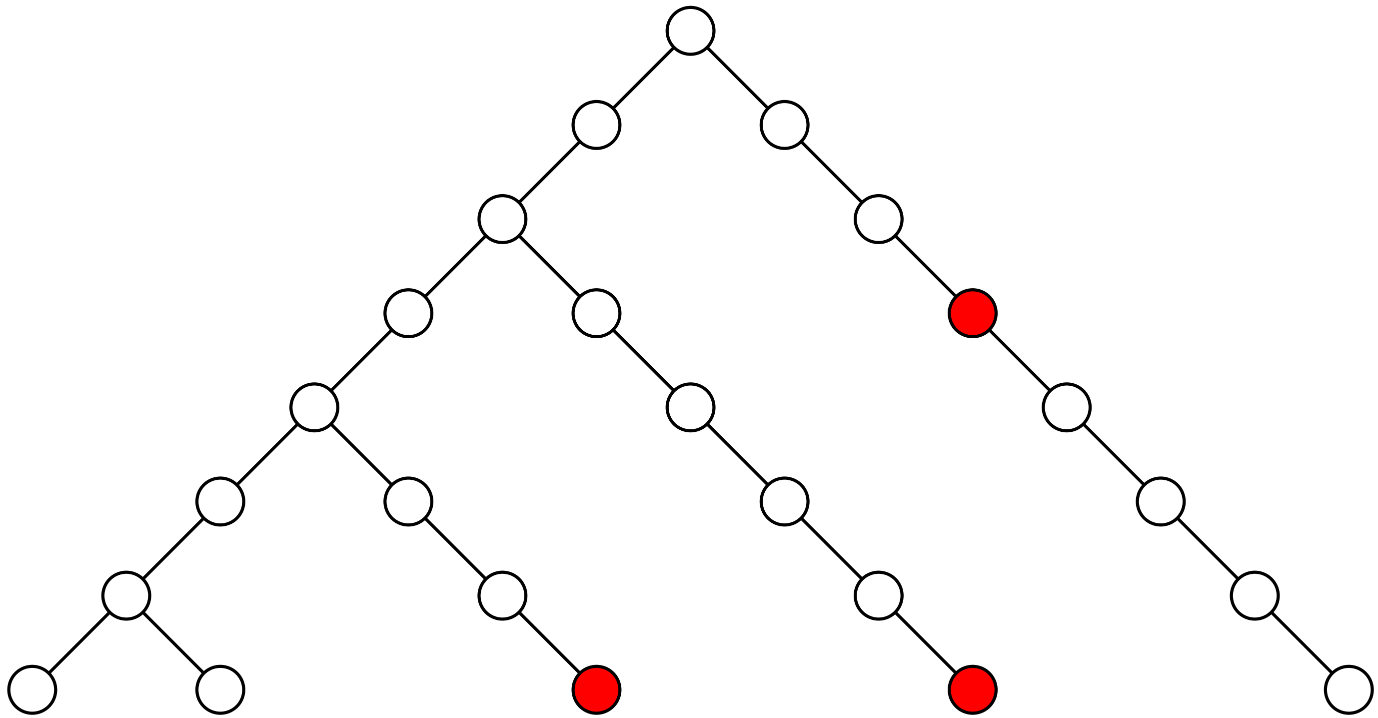
A smaller tree of the same kind remains.

Nodes in each level indistinguishable to the algorithm.

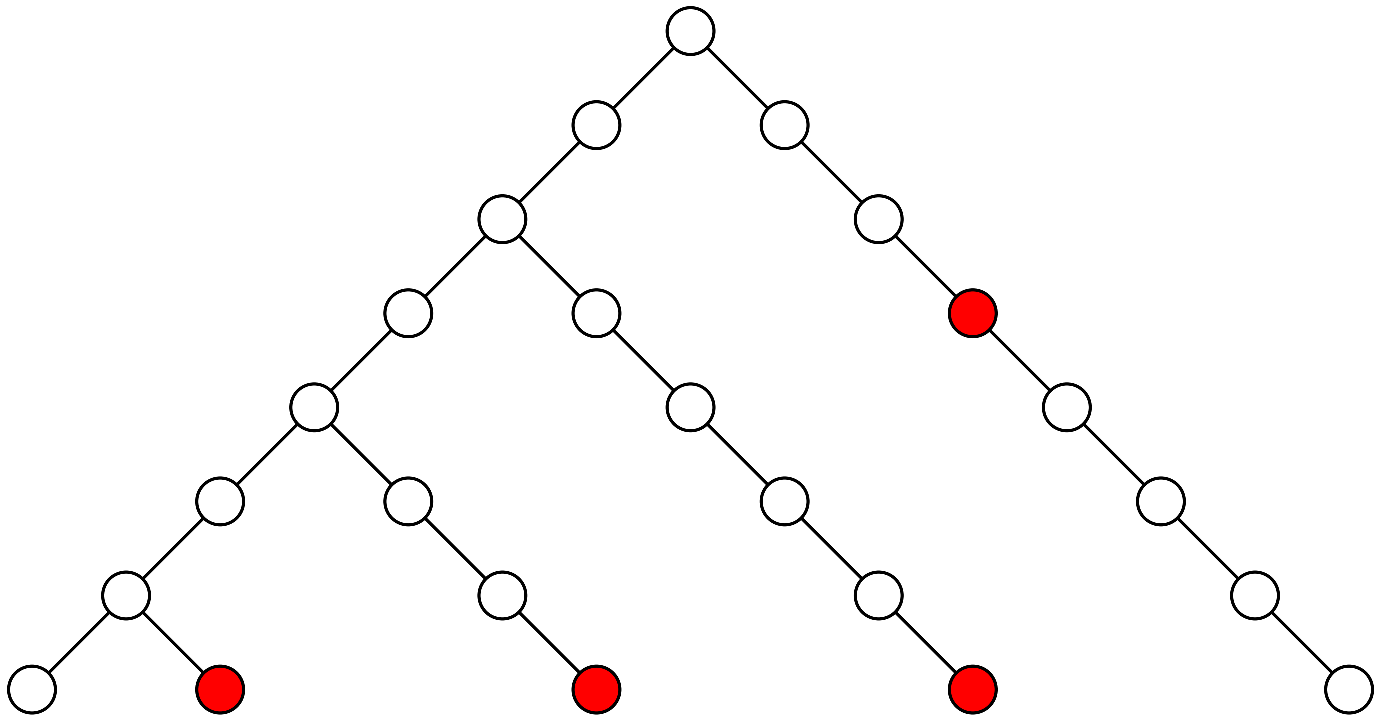
Competitive Lower Bound



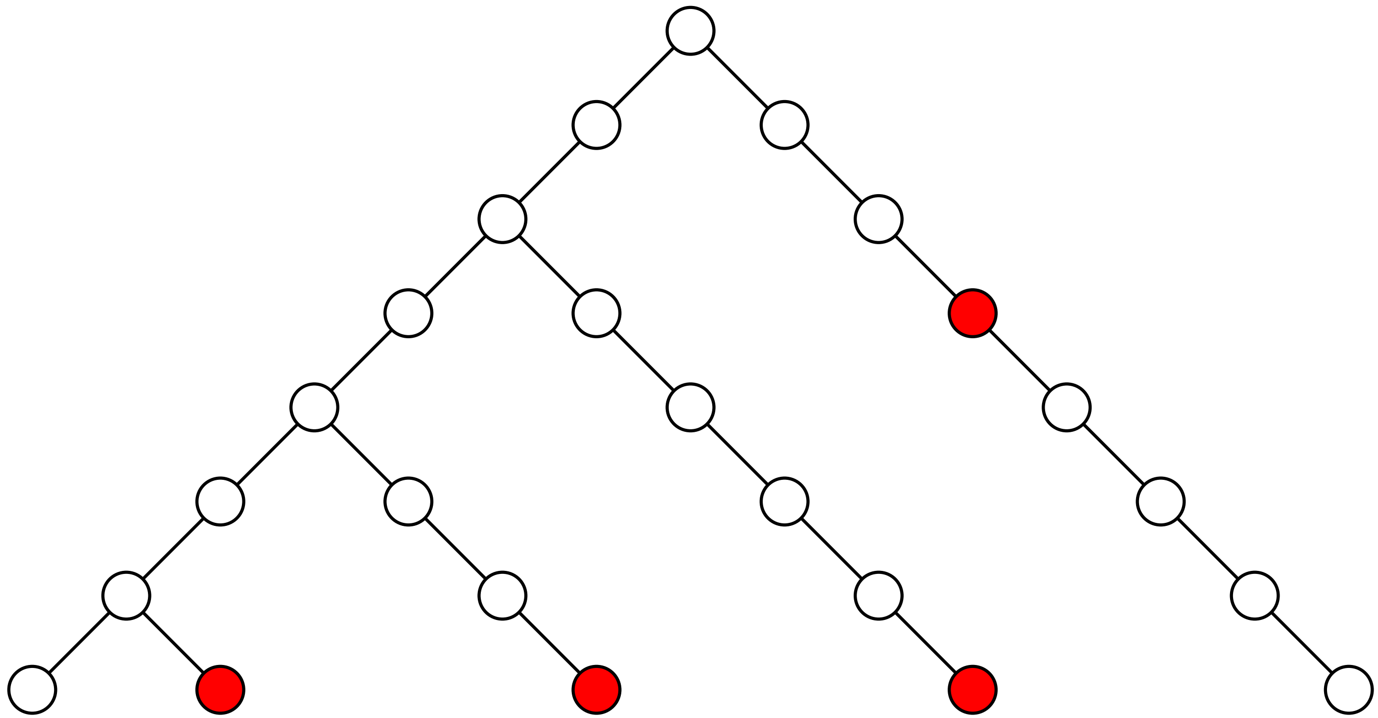
Competitive Lower Bound



Competitive Lower Bound

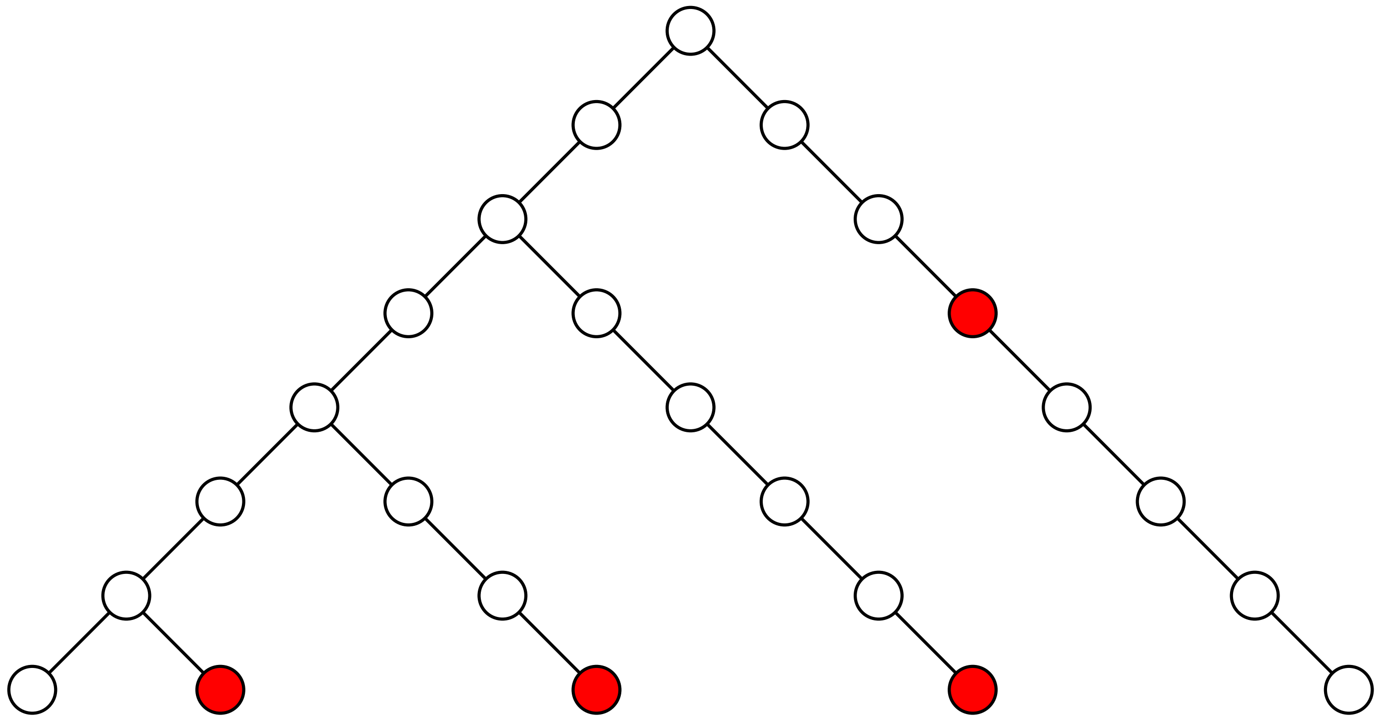


Competitive Lower Bound



Theorem. No **deterministic** algorithm can have competitive ratio better than $\Theta(\sqrt{n})$ for D-ALL-DISCOVERY in graphs with n nodes.

Competitive Lower Bound



Theorem. No **randomized** algorithm can have competitive ratio better than $\boldsymbol{\Theta(\log n)}$ for D-ALL-DISCOVERY in graphs with n nodes.

Ideas for an On-Line Algorithm

- View problem as a **hitting set** problem: For edge e, hit the set of its partial witnesses, and for non-edge $\bar{e}$, hit the set of queries that discover it.

Ideas for an On-Line Algorithm

- View problem as a **hitting set** problem: For edge e, hit the set of its partial witnesses, and for non-edge $\bar{e}$, hit the set of queries that discover it.
- Every non-edge can either be discovered by **many** (more than T) queries or by **few** (at most T) queries.

Ideas for an On-Line Algorithm

- View problem as a **hitting set** problem: For edge e, hit the set of its partial witnesses, and for non-edge $\bar{e}$, hit the set of queries that discover it.
- Every non-edge can either be discovered by **many** (more than T) queries or by **few** (at most T) queries.
- Similarly, every edge has either many partial witnesses or few.

Ideas for an On-Line Algorithm

- View problem as a **hitting set** problem: For edge e, hit the set of its partial witnesses, and for non-edge $\bar{e}$, hit the set of queries that discover it.
- Every non-edge can either be discovered by **many** (more than T) queries or by **few** (at most T) queries.
- Similarly, every edge has either many partial witnesses or few.
- Use **random** queries to discover all non-edges that can be discovered by many queries, and to get a partial witness for every edge that has many partial witnesses.

Ideas for an On-Line Algorithm

- View problem as a **hitting set** problem: For edge e, hit the set of its partial witnesses, and for non-edge $\bar{e}$, hit the set of queries that discover it.
- Every non-edge can either be discovered by **many** (more than T) queries or by **few** (at most T) queries.
- Similarly, every edge has either many partial witnesses or few.
- Use **random** queries to discover all non-edges that can be discovered by many queries, and to get a partial witness for every edge that has many partial witnesses.
- For each remaining undiscovered non-edge [edge], query **all** vertices that discover it [all partial witnesses].

Ideas for an On-Line Algorithm

- View problem as a **hitting set** problem: For edge e, hit the set of its partial witnesses, and for non-edge $\bar{e}$, hit the set of queries that discover it.
- Every non-edge can either be discovered by **many** (more than T) queries or by **few** (at most T) queries.
- Similarly, every edge has either many partial witnesses or few.
- Use **random** queries to discover all non-edges that can be discovered by many queries, and to get a partial witness for every edge that has many partial witnesses.
- For each remaining undiscovered non-edge [edge], query **all** vertices that discover it [all partial witnesses].
- With $T = \sqrt{n \ln n}$ we get competitive ratio $O(\sqrt{n \log n})$.

Algorithm

- **Phase 1:** Choose $3\sqrt{n \ln n}$ vertices uniformly at random and query them.
- **Phase 2:** While there is an undiscovered (non-)edge between some vertices u and v, do:
  - query u and v
  - if $\{u, v\}$ is non-edge, query all vertices that discover $\{u, v\}$.
  - if $\{u, v\}$ is an edge and $d(u), d(v) \leq \sqrt{n / \ln n}$, query all neighbors of u and v and then all vertices that are partial witnesses for $\{u, v\}$.
  - otherwise, proceed with another undiscovered (non)-edge

Algorithm for D-ALL-DISCOVERY

Theorem. There is a randomized on-line algorithm for D-ALL-DISCOVERY that achieves competitive ratio $O(\sqrt{n \log n})$.

Proof Ideas:

- With probability at least $1 - \frac{1}{n}$, Phase 1 discovers all non-edges that are discovered by many (i.e., more than $T = \sqrt{n \ln n}$) queries and contains partial witnesses for all edges that have many partial witnesses.
- In Phase 2, if the case that u or v has more than $\sqrt{n/\ln n}$ neighbors happens k times, OPT is at least $k\frac{\sqrt{n/\ln n}}{2n} = \frac{k}{2\sqrt{n \ln n}}$, so these iterations do not hurt the competitive ratio.

Results for D-ALL-VERIFICATION

- D-ALL-VERIFICATION is NP-hard.
  - Proof by reduction from vertex cover problem.
- There is an $O(\log n)$-approximation algorithm for D-ALL-VERIFICATION.
  - Simply apply the greedy set cover approximation algorithm.
- The cycle C_n, $n > 6$, can be verified optimally with 2 queries.
- The hypercube H_d, $d \geq 3$, can be verified optimally with 2^{d-1} queries.
- There is a polynomial algorithm for D-ALL-VERIFICATION in trees.

Summary of Network Discovery Results

- **LG model:**
  - Discovery: randomized upper bound $O(\sqrt{n \log n})$, deterministic lower bound 3.
  - Verification: $\Theta(\log n)$-approximable.
- **D model:**
  - Discovery: randomized upper bound $O(\sqrt{n \log n})$, deterministic lower bound $\Omega(\sqrt{n})$, randomized lower bound $\Omega(\log n)$.
  - Verification: NP-hard, $O(\log n)$-approximation.

Open Problems

- Close the gaps between upper and lower bounds for competitive ratio of network discovery problems.
- Deterministic on-line algorithms for network discovery?
- Better approximation for D-ALL-VERIFICATION?
- Better results for special graph classes?
- Models where queries can be made only at a subset of the nodes of the graph (motivated by practical applications).
- Approximate discovery/verification: e.g., discover 95% of edges and 95% of non-edges.
- Discovering graph properties.

Thank you!