Sublinear algorithms for correlation clustering

Slobodan Mitrović
(UC Davis)

Mina Dalirrooyfard
Morgan Stanley Research

Konstantin Makarychev
Northwestern University

Correlation clustering

[Bansal, Blum, Chawla, 2002, 2004]

Input:

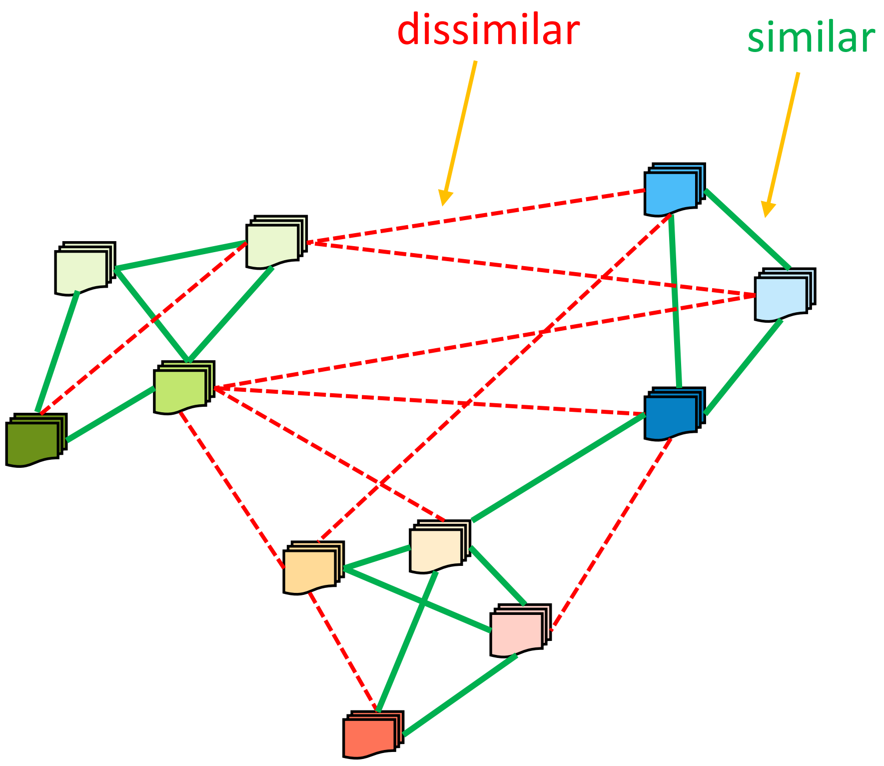
n objects

Similarity function ***f***

$f(a, b) \rightarrow \{similar, dissimilar\}$

Correlation clustering

[Bansal, Blum, Chawla, 2002, 2004]



Input:

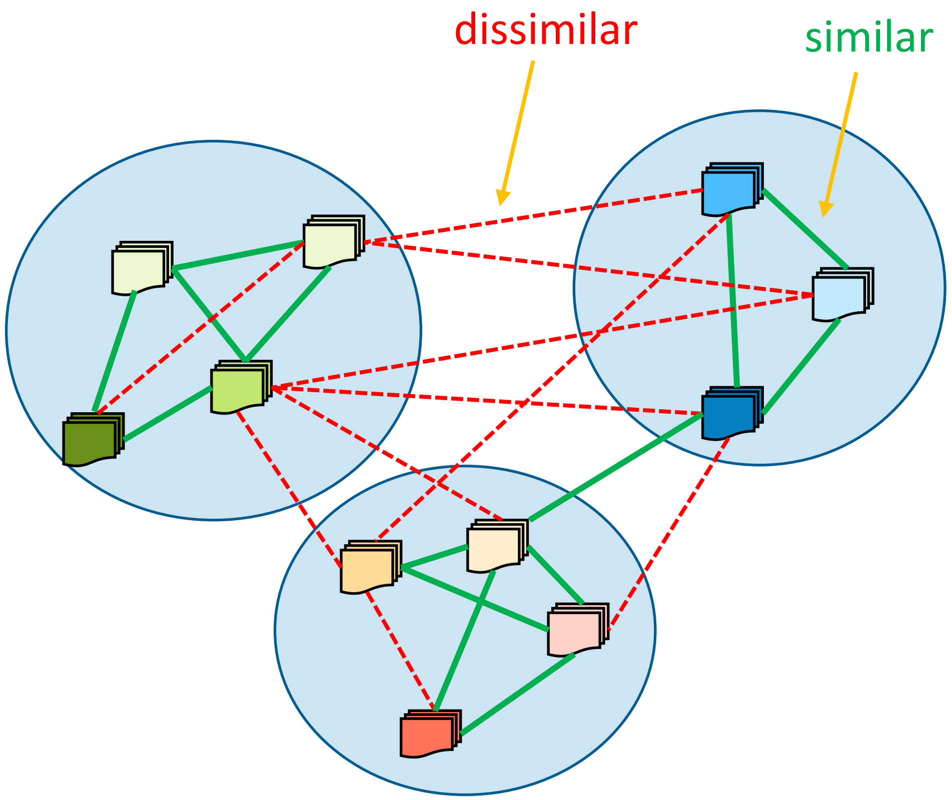
n objects

Similarity function ***f***

f$(a, b) \rightarrow \{$*similar*, *dissimilar*$\}$

Correlation clustering

[Bansal, Blum, Chawla, 2002, 2004]



Input:

n objects

Similarity function ***f***

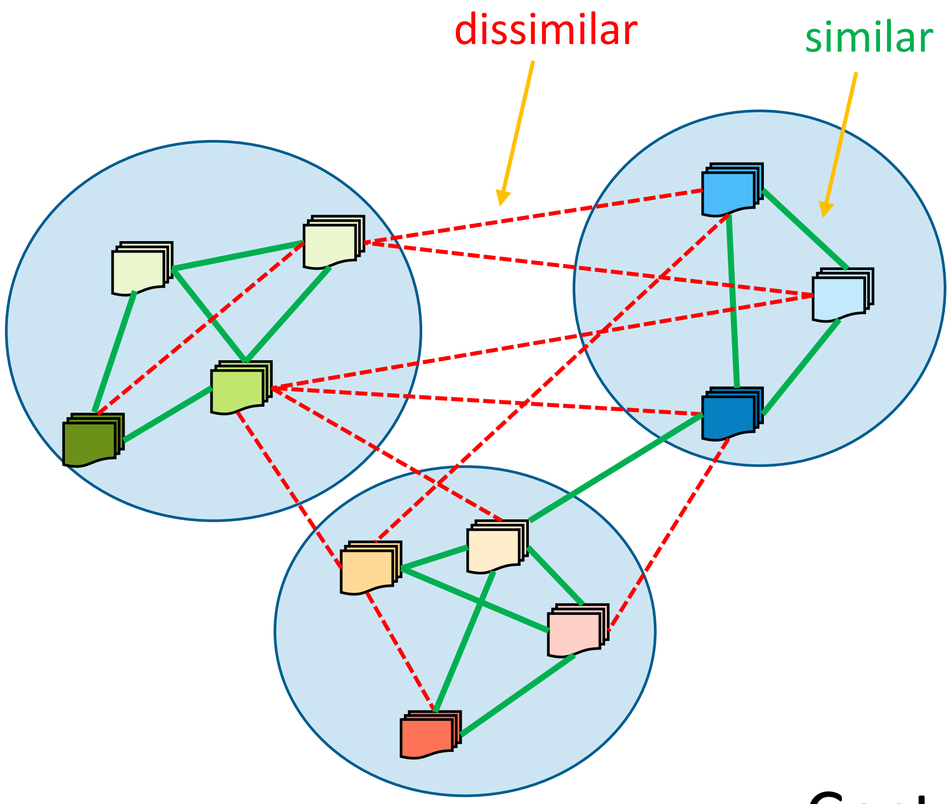
f$(a, b) \rightarrow \{$*similar*, *dissimilar*$\}$

Goal:

A clustering that aligns with ***f*** *as much as possible*.

Correlation clustering

[Bansal, Blum, Chawla, 2002, 2004]



Cost = 2 + 1

Input:

n objects

Similarity function ***f***

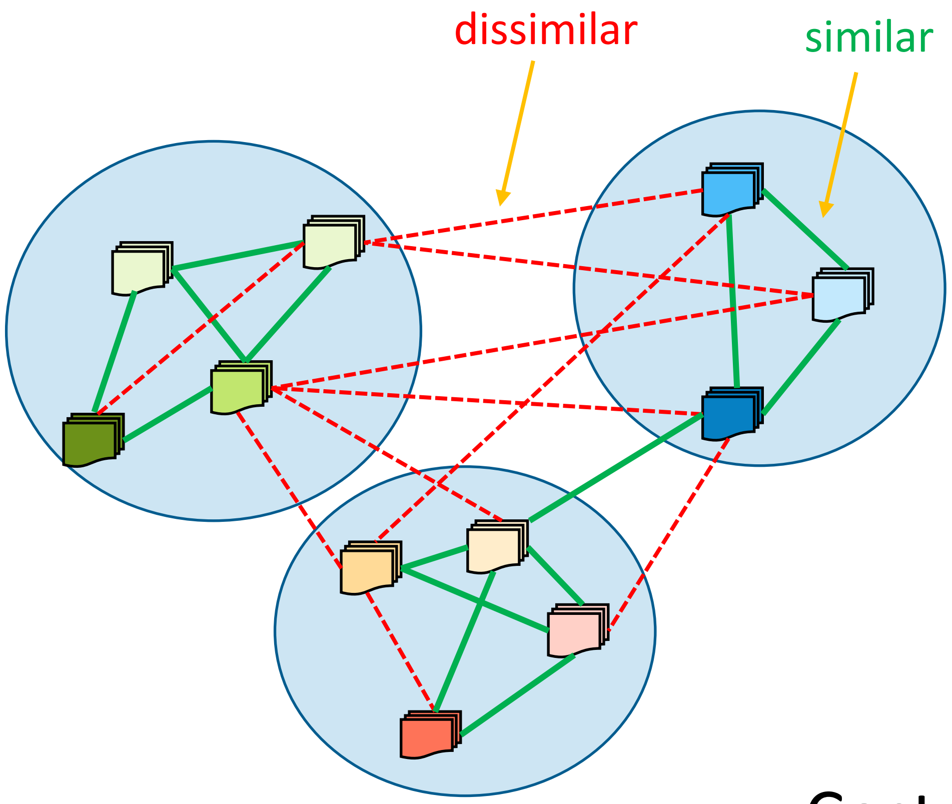
f(*a*, *b*) → {*similar*, *dissimilar*}

Goal:

A clustering that aligns with ***f*** *as much as possible*.

Correlation clustering

[Bansal, Blum, Chawla, 2002, 2004]



Cost = **2** + **1**

Input:

n objects

Similarity function ***f***

f(*a, b*) → {*similar*, *dissimilar*}

Goal:

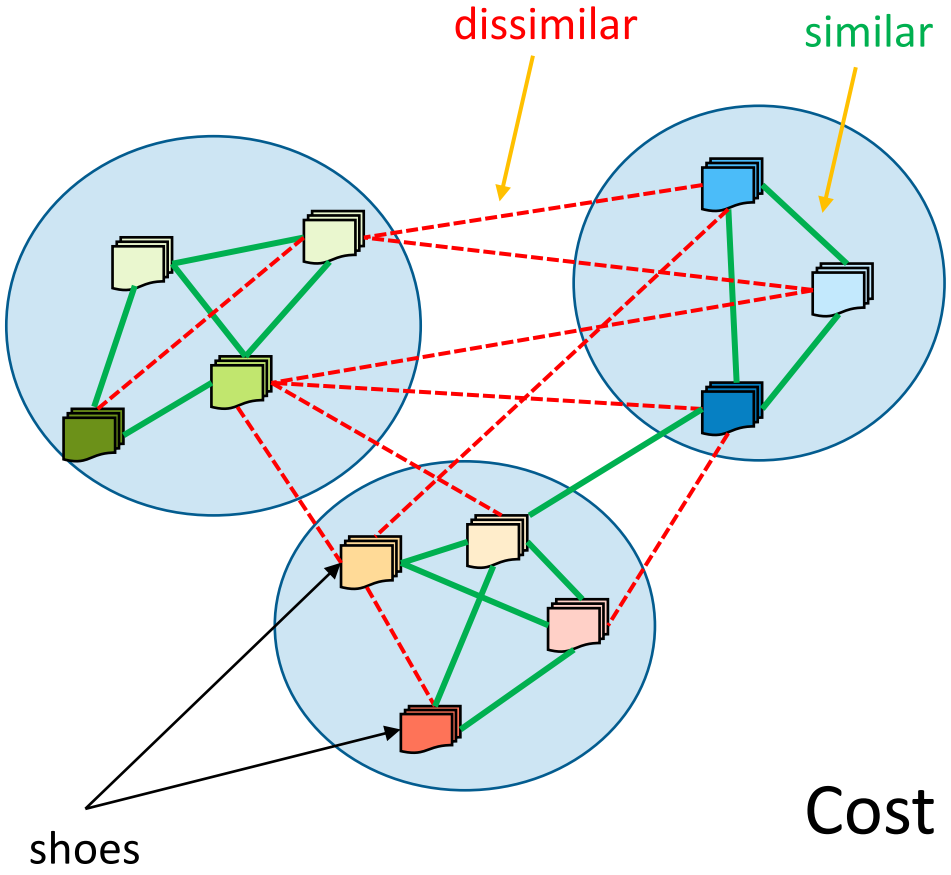
A clustering that aligns with ***f***
as much as possible.

Application:

- Aggregating accounts

Correlation clustering

[Bansal, Blum, Chawla, 2002, 2004]



Cost = **2** + **1**

Input:

n objects

Similarity function ***f***

f*(a, b)* → *{similar, dissimilar}*

Goal:

A clustering that aligns with ***f*** *as much as possible*.

Application:

- Aggregating accounts
- Semi-supervised learning

History (an overview)

- [Bansal, Blum, Chawla, 2002, 2004]
- [Charikar, Guruswami, Wirth, 2003] – APX-hard, 4 approximation
- [Demaine, Emanuel, Fiat, Immorlica, 2006] – O(log n) approximation for weighted
- …
- [Ailon, Charikar, Newman, 2005, 2008] – 3 approximation, **Pivot**
- [Chawla, Makarychev, Schramm, Yaroslavtsev, 2014] – 2.06 approximation
- [Cohen-Addad, Lee, Newman, 2022] – 1.994 approximation
- [Cohen-Addad, Lee, Li, Newman, 2023] – 1.73 approximation
- [Cao, Cohen-Addad, Lee, Li, Newman, Vogl, 2024] – 1.437 approximation

History (big data regimes)

n = number of vertices in the input graph
Δ = maximum vertex degree

Approx.	Model	Complexity	References
3	Centralized	$O(m)$	[Ailon, Charikar, Newman, 2005]
3	MPC	$O(\log^2 n)$	[Blelloch, Fineman, Shun, 2012]
3+ε	MPC	$O(\log n \log \Delta)$	[Chierichetti, Dalvi, Kumar, 2014]
3	MPC	$O(\log n)$	[Fischer, Noever, 2018]
3	MPC	$O(\log \Delta \log \log n)$	[Cambus, Choo, Miikonen, Uitto, 2021]
~700	MPC	$O(1)$	[Cohen-Addad, Lattanzi, **M**, Norouzi-Fard, Parotsidis, Tarnawski, 2021]
3+ε	MPC	$O(1/\epsilon)$	[Behnezhad, Charikar, Ma, Tan, 2022]
3+ε	MPC	$O(1)^*$	[Cambus, Kuhn, Lindy, Pai, Uitto, 2023]
3+ε	MPC	$O(\log 1/\epsilon)$	**this work**
1.846	MPC	$O(1)$	[Cohen-Addad, Lolck, Pilipczuk, Thorup, Yan, Zhang, 2024]
3+ε	LCA	$\Delta^{O\left(\frac{1}{\epsilon}\right)}$	[Behnezhad, Charikar, Ma, Tan, 2022]
3+ε	LCA	$O(\Delta/\epsilon)$	**this work**
3+ε	Dynamic	$O(\log^2 n \log^2 \Delta)$	[Behnezhad, Derakhshan, Hajiaghayi, Stein, Sudan, 2019]
3+ε	Dynamic	$O(\log^4 n)$	[Chechik, Zhang, 2019]
3+ε	Dynamic	$O(1/\epsilon)$	**this work**
2.997	Dynamic	$\operatorname{poly} \log n$	[Behnezhad, Charikar, Cohen-Addad, Ghafari, Ma, 2024]

Not complete picture.
We will return to this.

Recent History

in semi-streaming **single pass**

Approx.	References
5	[Behnezhad, Charikar, Ma, Tan, 2023]
3+ε	[Cambus, Kuhn, Lindy, Pai, Uitto ,2023]
3+ε	[Chakrabarty, Makarychev, 2023]
3+ε	**this work**

Outline

- Pivot [Ailon, Charikar, Newman, 2005]
- Our approach (Pruned Pivot)
- Implementations
- Implications on Maximal Independent Set
- Analysis

Pivot [Ailon, Charikar, Newman, 2005]

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

Pivot [Ailon, Charikar, Newman, 2005]

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

1. Let π be a random ordering of V

Pivot [Ailon, Charikar, Newman, 2005]

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

1. Let π be a random ordering of V
2. For i = 1 to n
   a. If π(i) is not clustered
      a. Cluster π(i) and its un-clustered neighbors together.

Pivot [Ailon, Charikar, Newman, 2005]

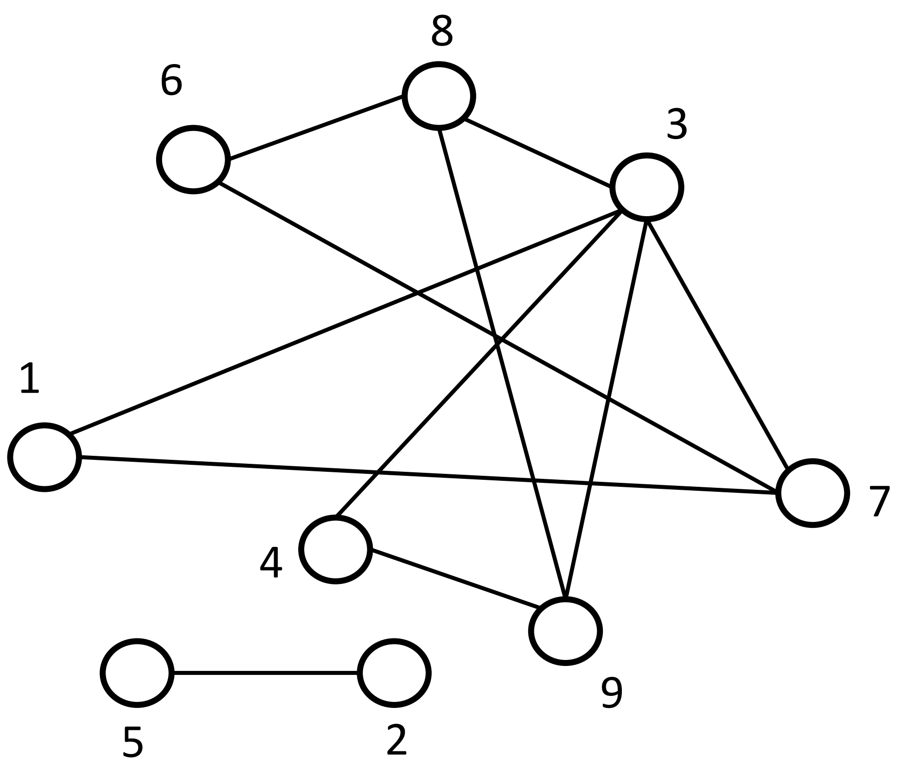
edge = similar
no-edge = dissimilar

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

1. Let π be a random ordering of V
2. For i = 1 to n
   a. If π(i) is not clustered
      a. Cluster π(i) and its un-clustered neighbors together.



Pivot [Ailon, Charikar, Newman, 2005]

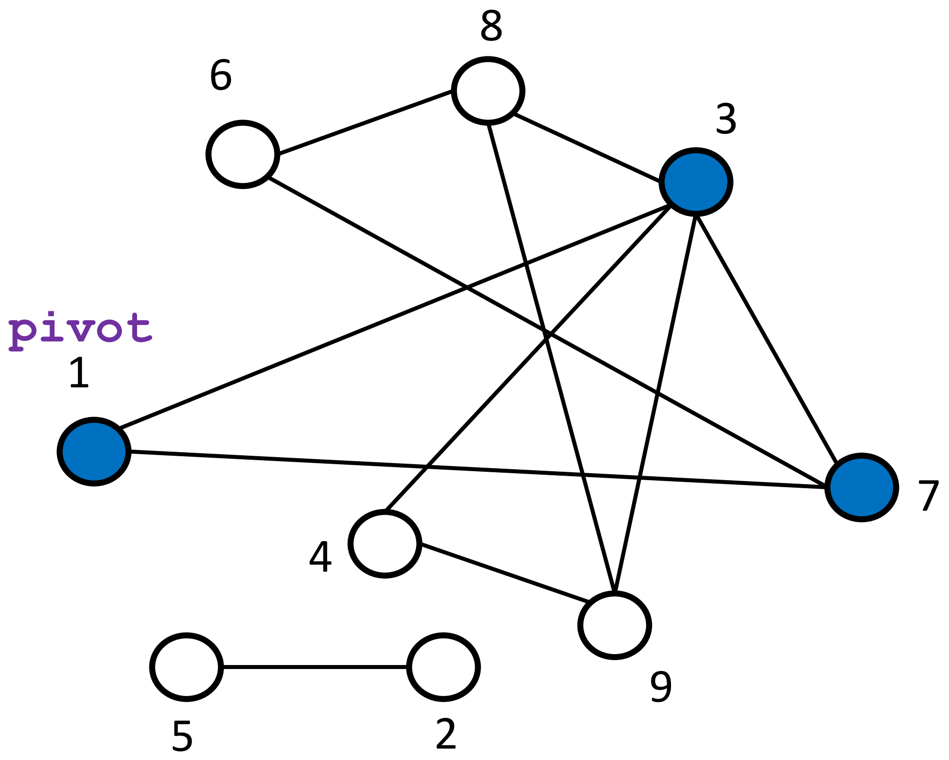
edge = similar
no-edge = dissimilar

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

1. Let π be a random ordering of V
2. For i = 1 to n
   a. If π(i) is not clustered
      a. Cluster π(i) and its un-clustered neighbors together.



Pivot [Ailon, Charikar, Newman, 2005]

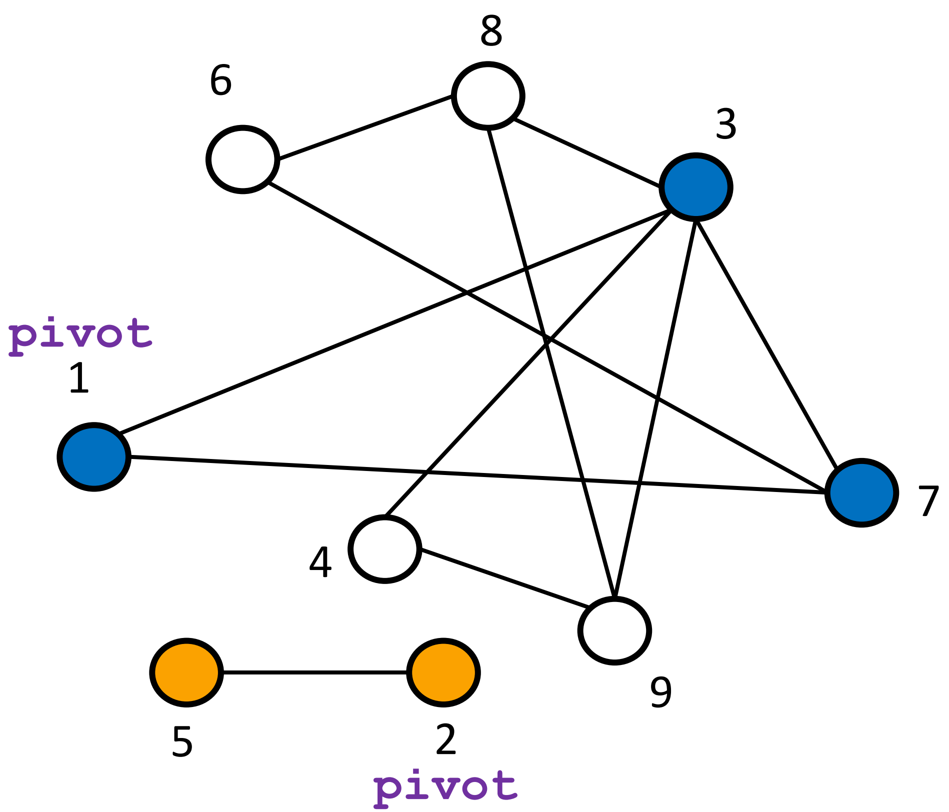
edge = similar
no-edge = dissimilar

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

1. Let π be a random ordering of V
2. For i = 1 to n
    a. If π(i) is not clustered
        a. Cluster π(i) and its un-clustered neighbors together.



Pivot [Ailon, Charikar, Newman, 2005]

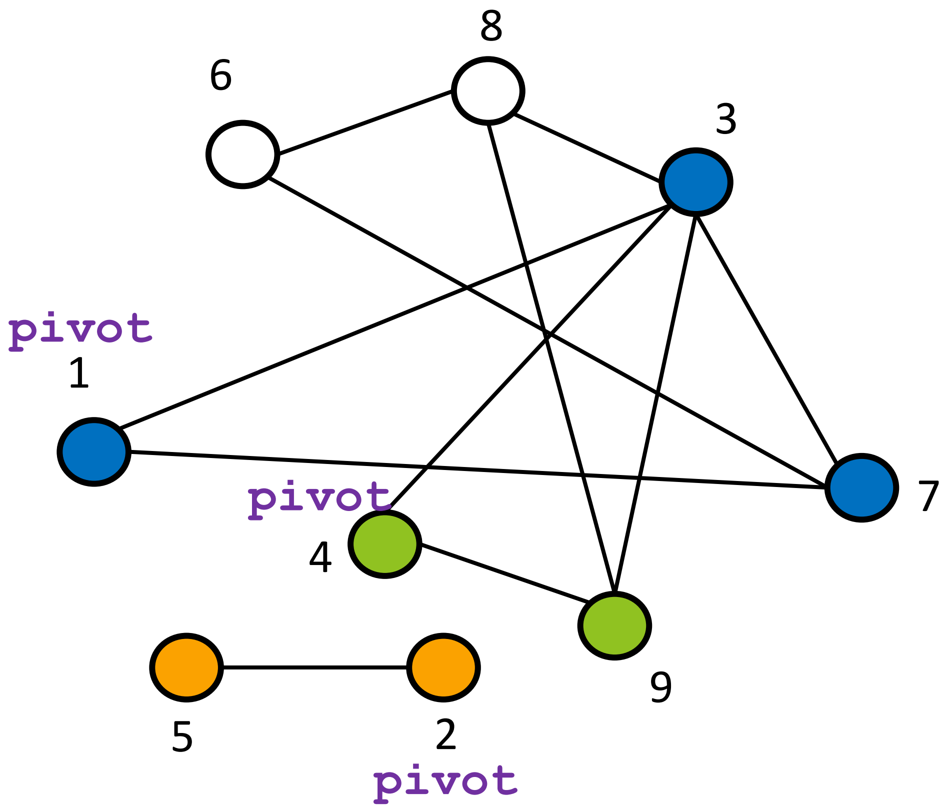
edge = similar
no-edge = dissimilar

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

1. Let π be a random ordering of V
2. For i = 1 to n
   a. If π(i) is not clustered
      a. Cluster π(i) and its un-clustered neighbors together.



Pivot [Ailon, Charikar, Newman, 2005]

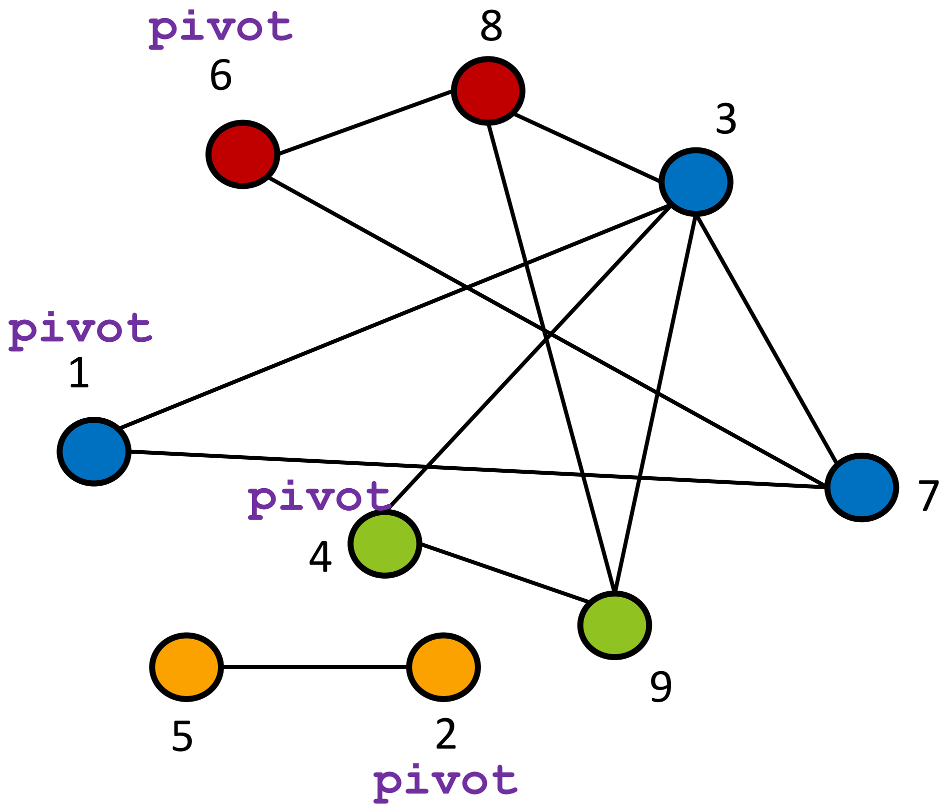
edge = similar
no-edge = dissimilar

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

1. Let π be a random ordering of V
2. For i = 1 to n
   a. If π(i) is not clustered
      a. Cluster π(i) and its un-clustered neighbors together.



Pivot [Ailon, Charikar, Newman, 2005]

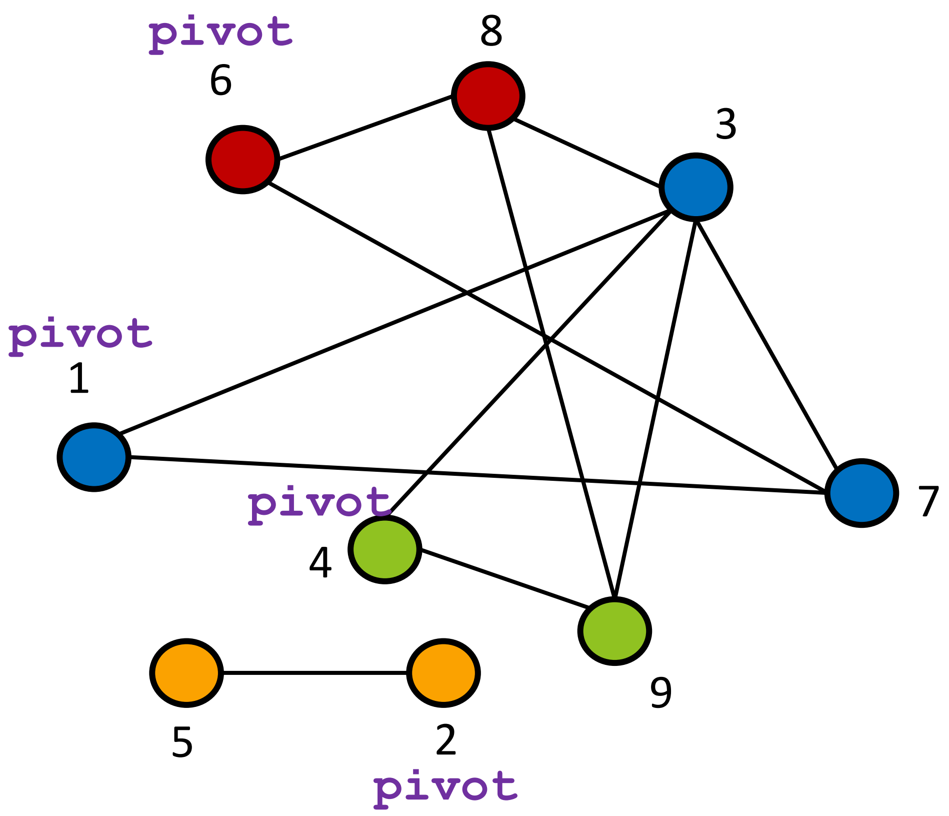
edge = similar
no-edge = dissimilar

n = number of vertices in the input graph

Pivot

Input: G = (V, E)

1. Let π be a random ordering of V
2. For i = 1 to n
   a. If π(i) is not clustered
      a. Cluster π(i) and its un-clustered neighbors together.

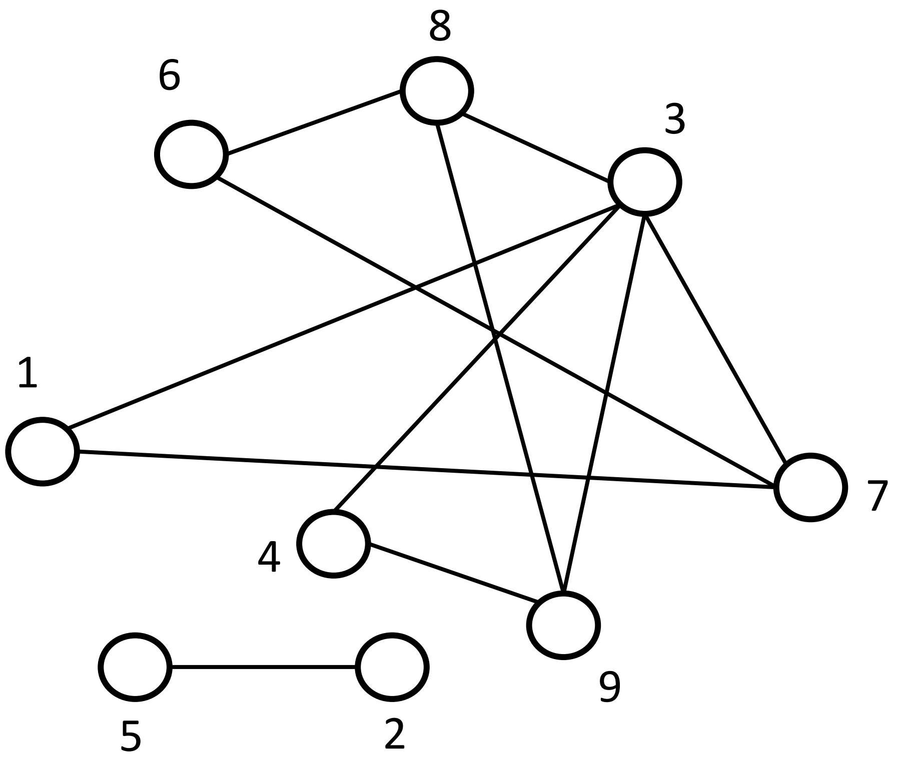


Claim: In expectation, Pivot outputs a 3-approximate correlation clustering.

Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

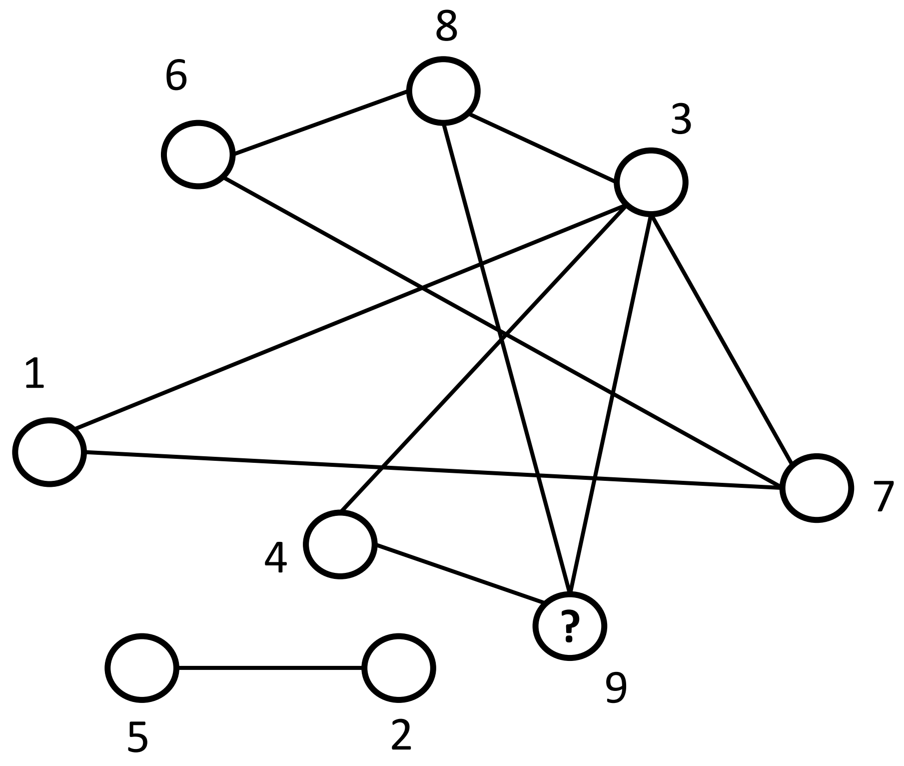
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



Pivot – Recursive Query View

Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.

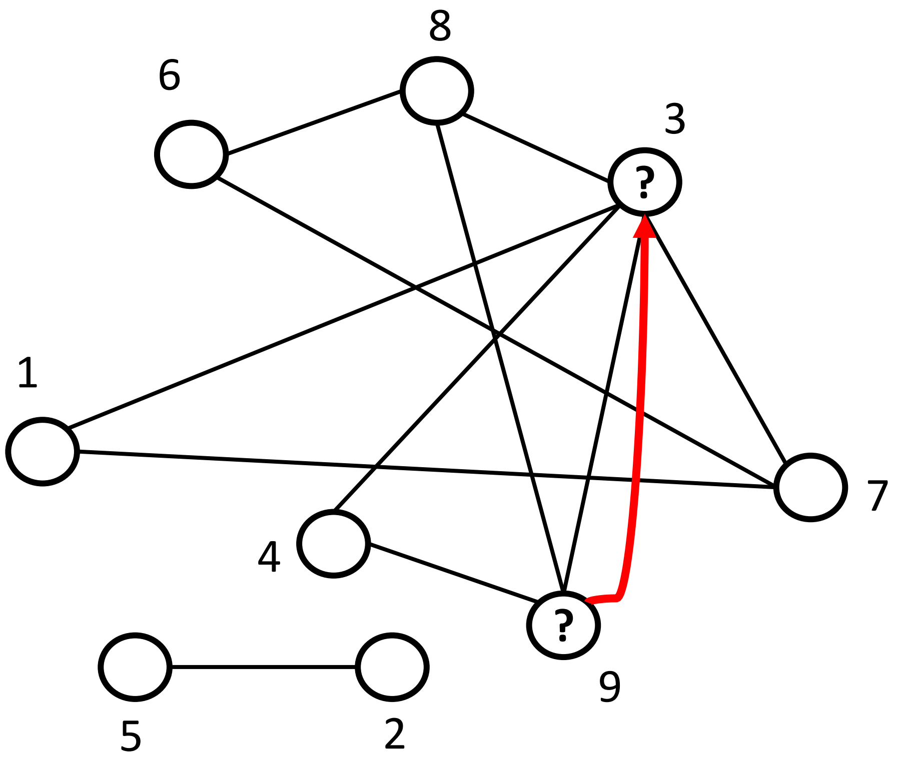
edge = similar
no-edge = dissimilar



Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

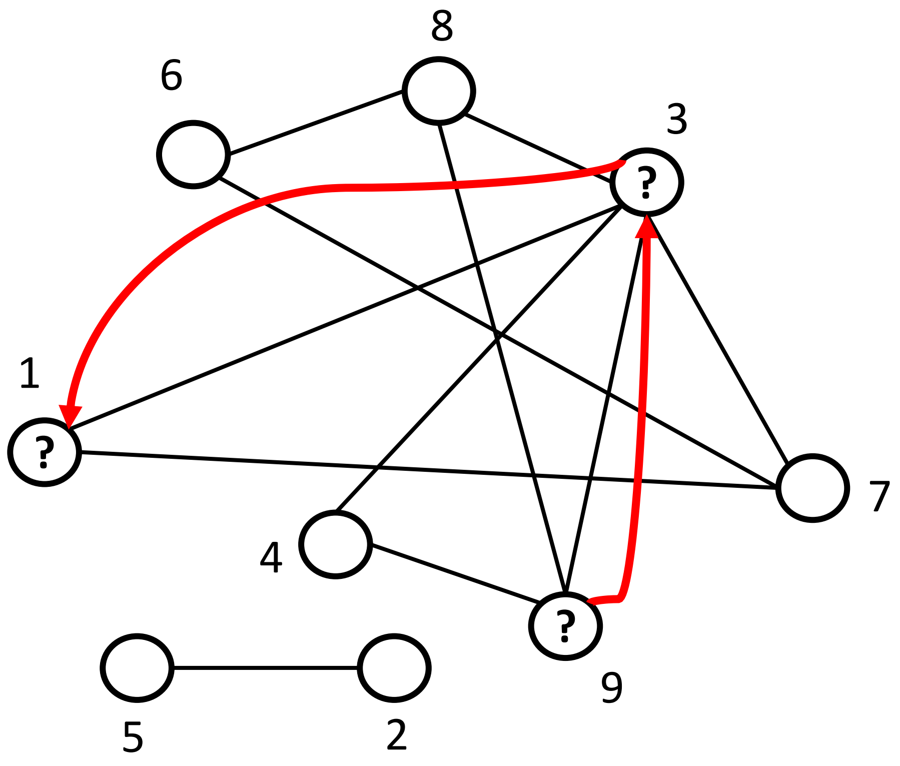
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

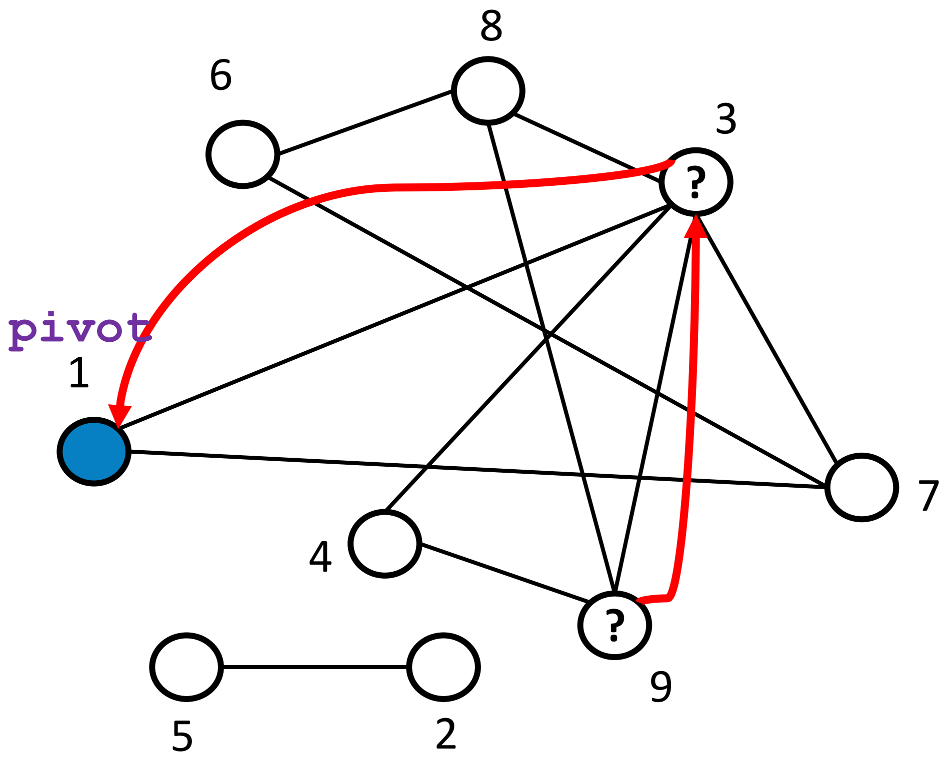
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

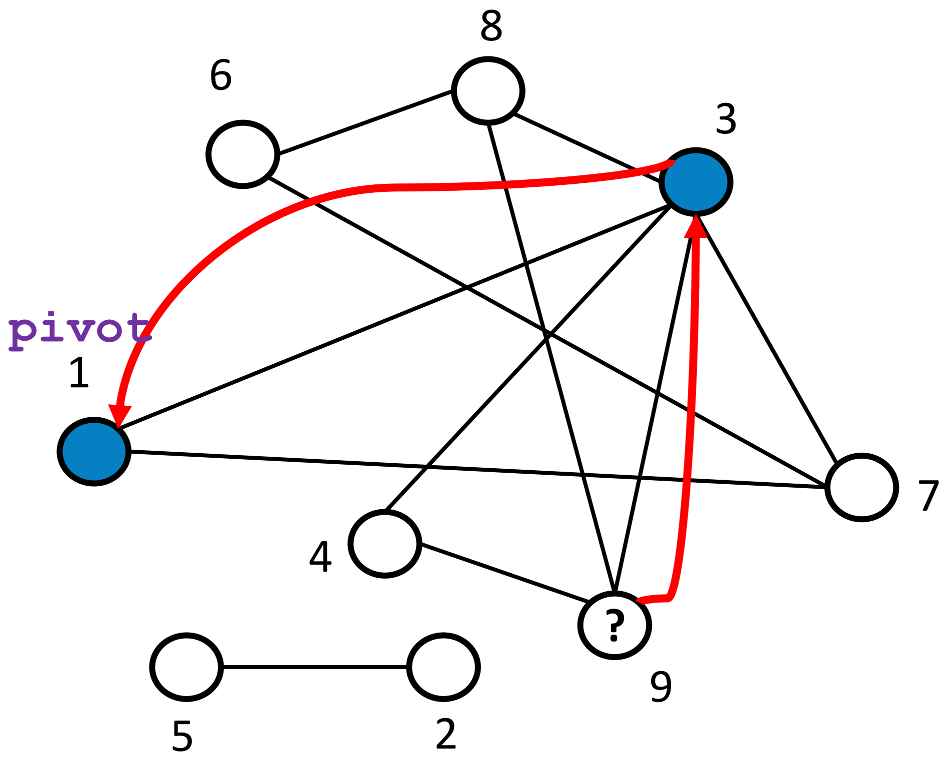
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

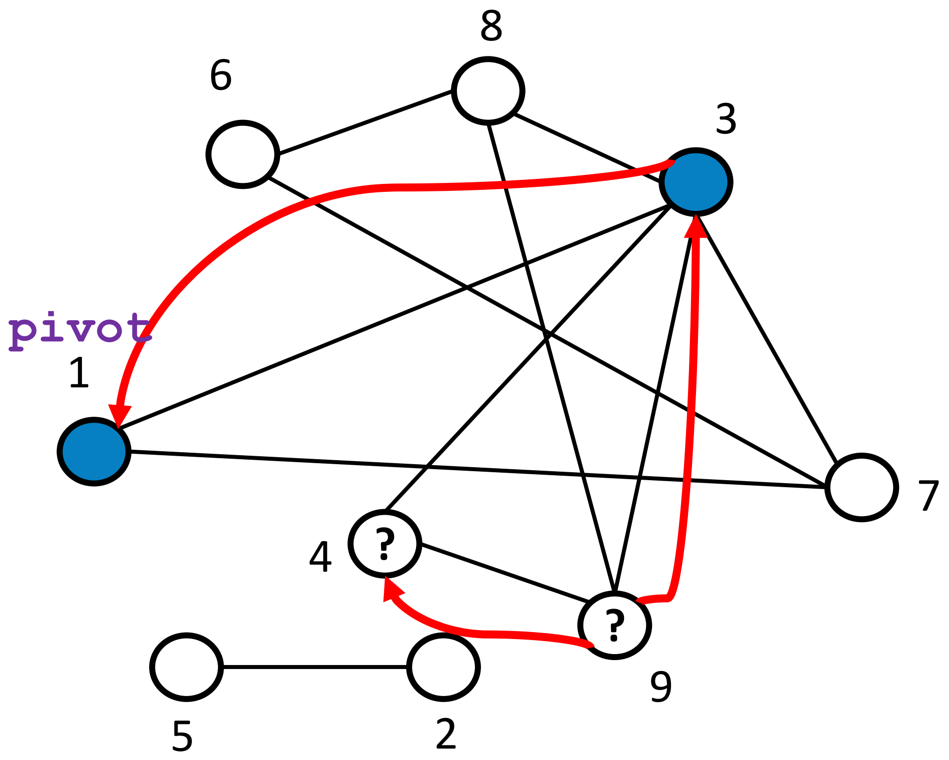
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

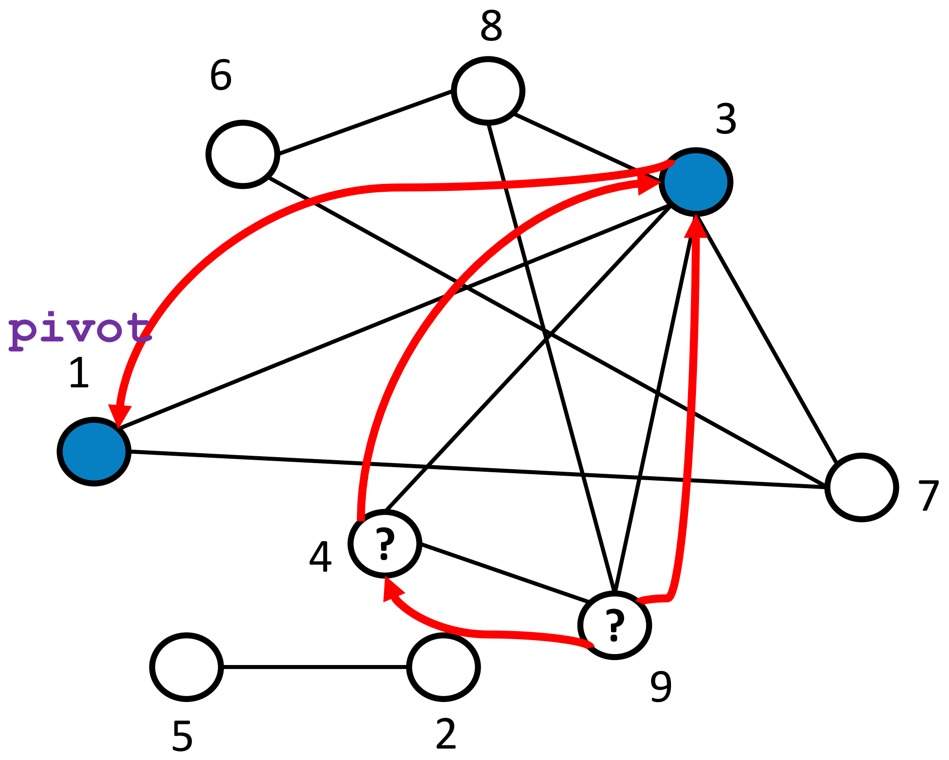
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

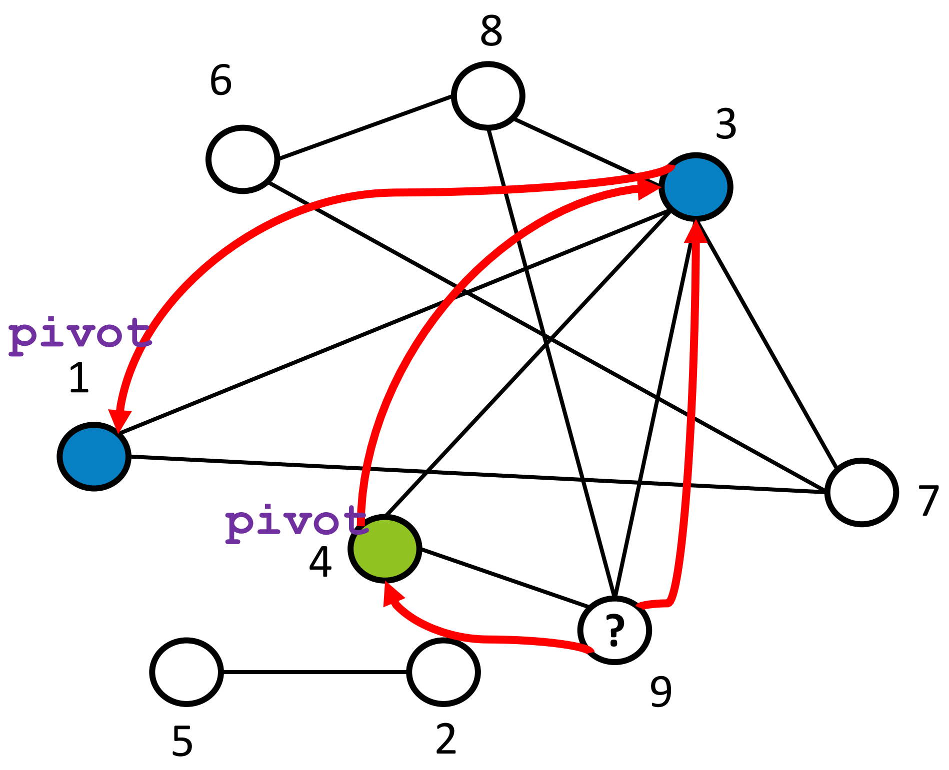
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

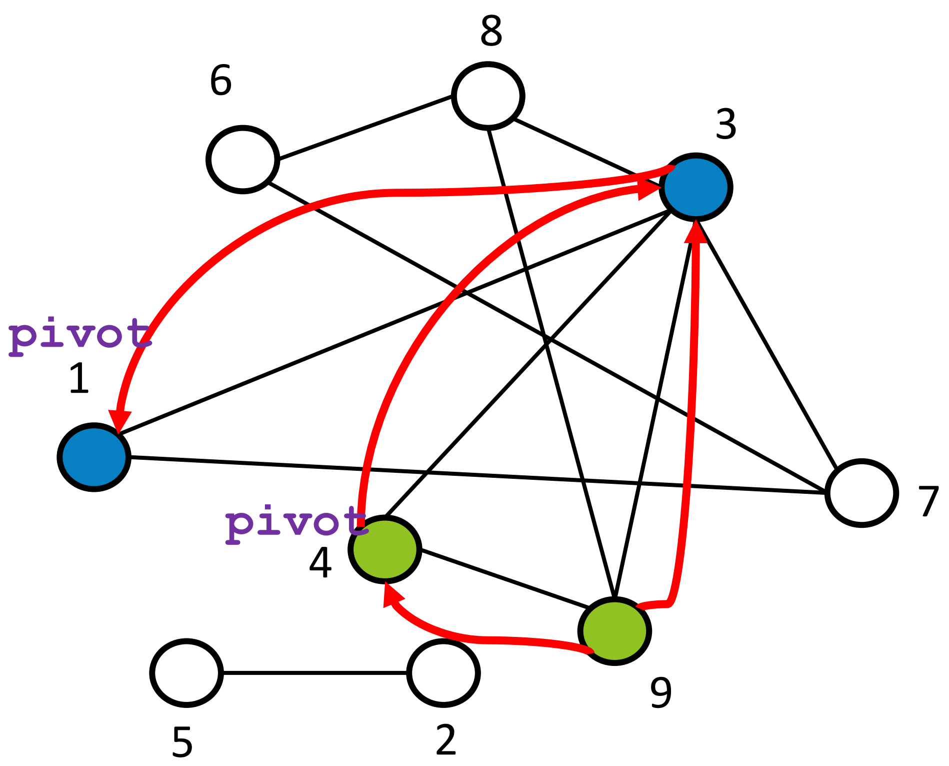
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

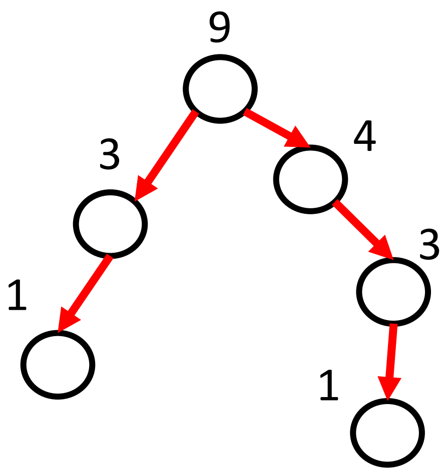
Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.

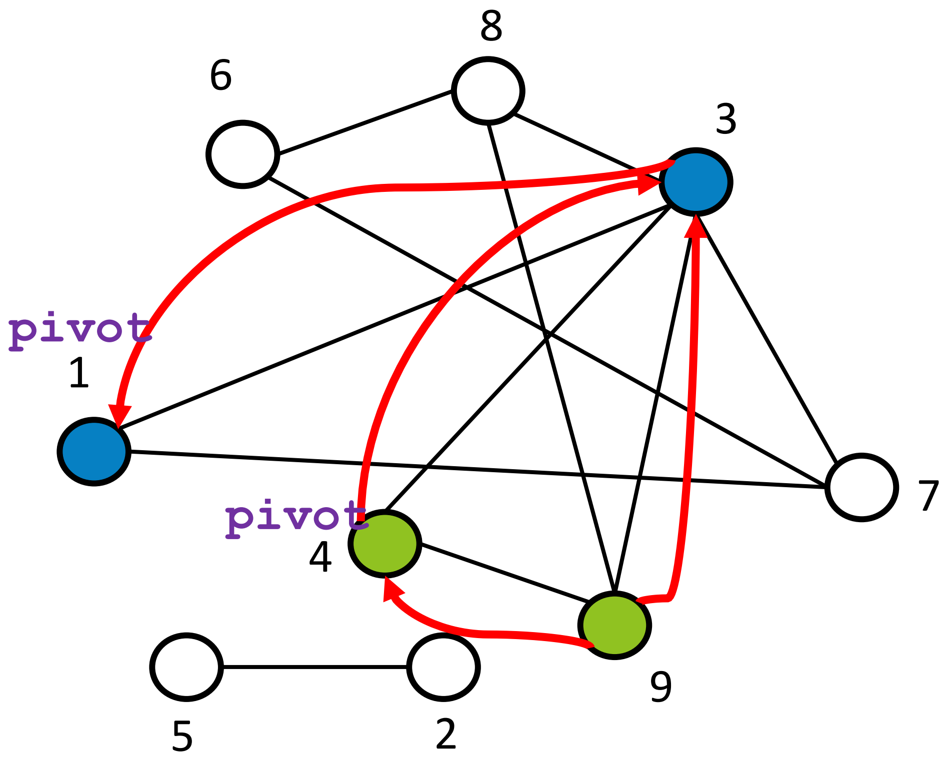


Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.

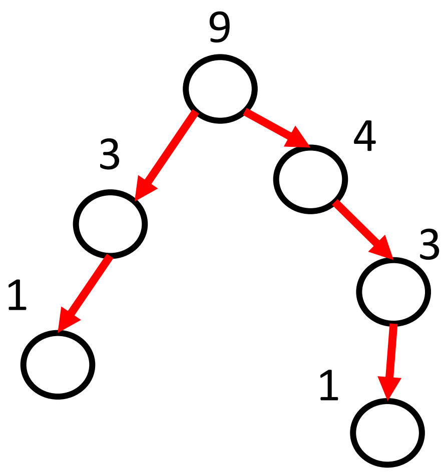


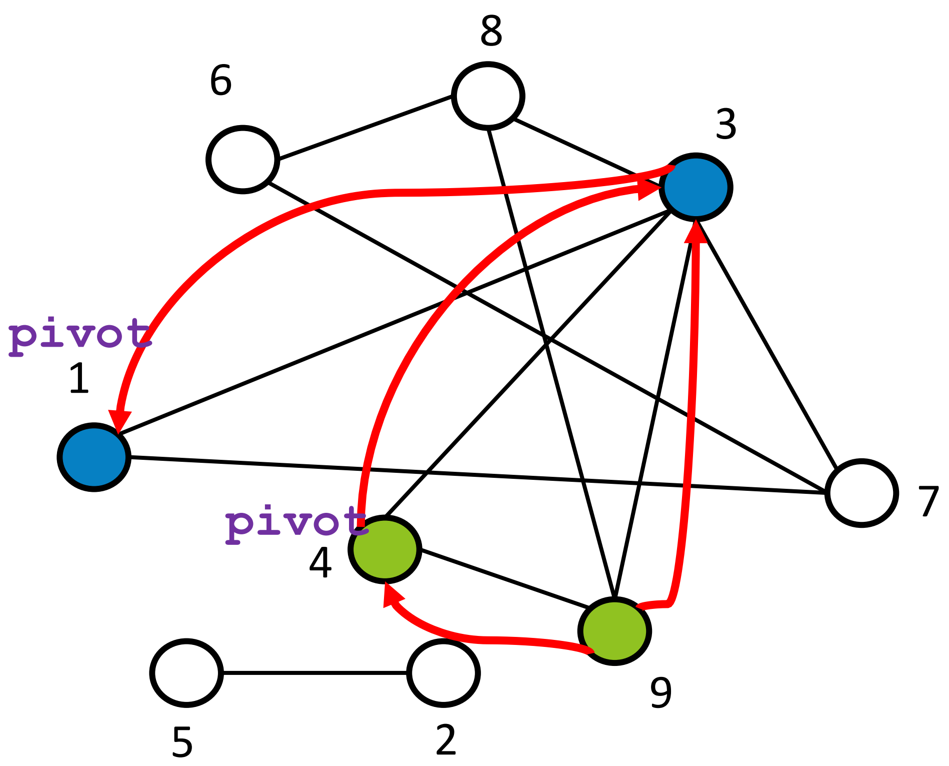


Pivot – Recursive Query View

edge = similar
no-edge = dissimilar

Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



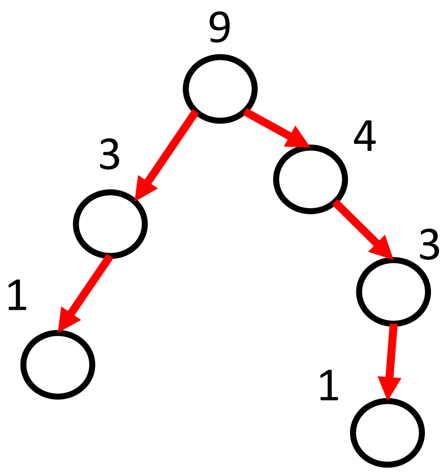


Our approach (Pruned Pivot)

If the query tree for v has size greater than 1/ε, make v a singleton cluster.
Gives a 3+O(ε) approximation.

`Pivot` – Recursive Query View

Vertex v is a pivot **iff**
none of its smaller-π-value neighbors is a pivot.



[Behnezhad, Charikar, Ma, Tan, 2022]
Tree-depth = O(1/ε)
+
[Chakrabarty, Makarychev, 2023]
Vertex-width = O(1/ε)

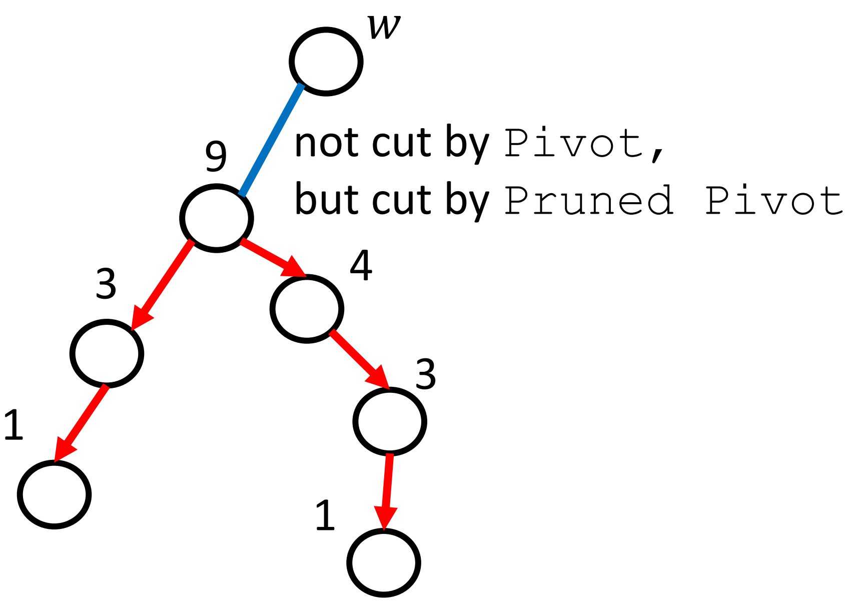
Tree-size = $1/\varepsilon^{O(1/\varepsilon)}$

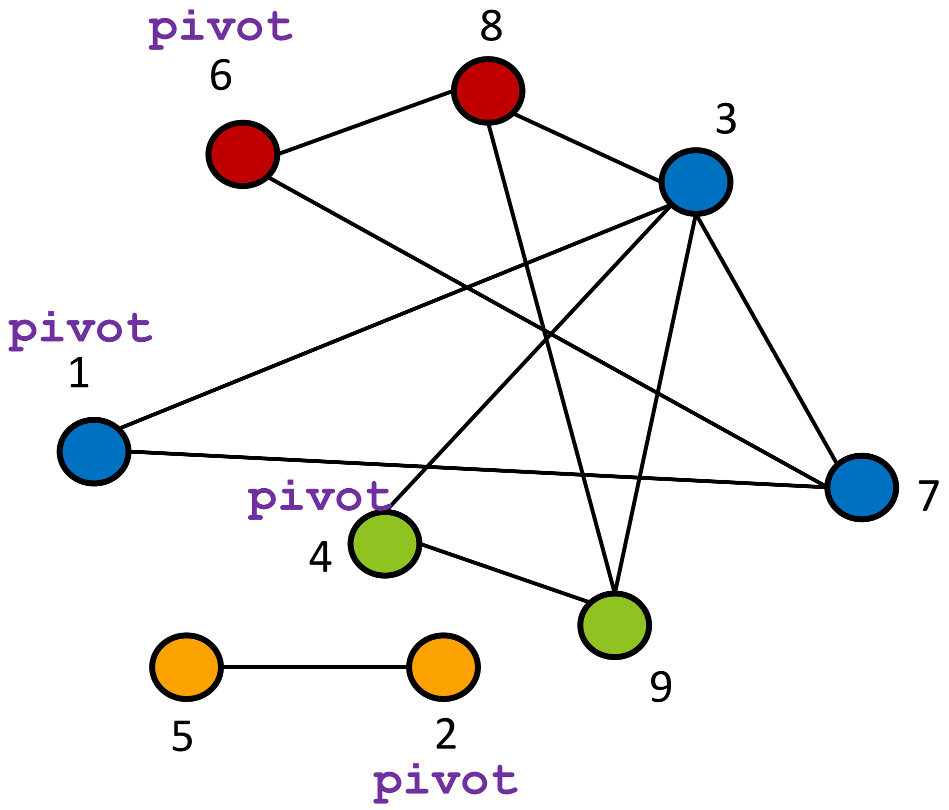
Our approach (`Pruned Pivot`)

If the query tree for v has size greater than 1/ε, make v a singleton cluster.
Gives a 3+O(ε) approximation.

Pruned Pivot:

Why to expect it works?





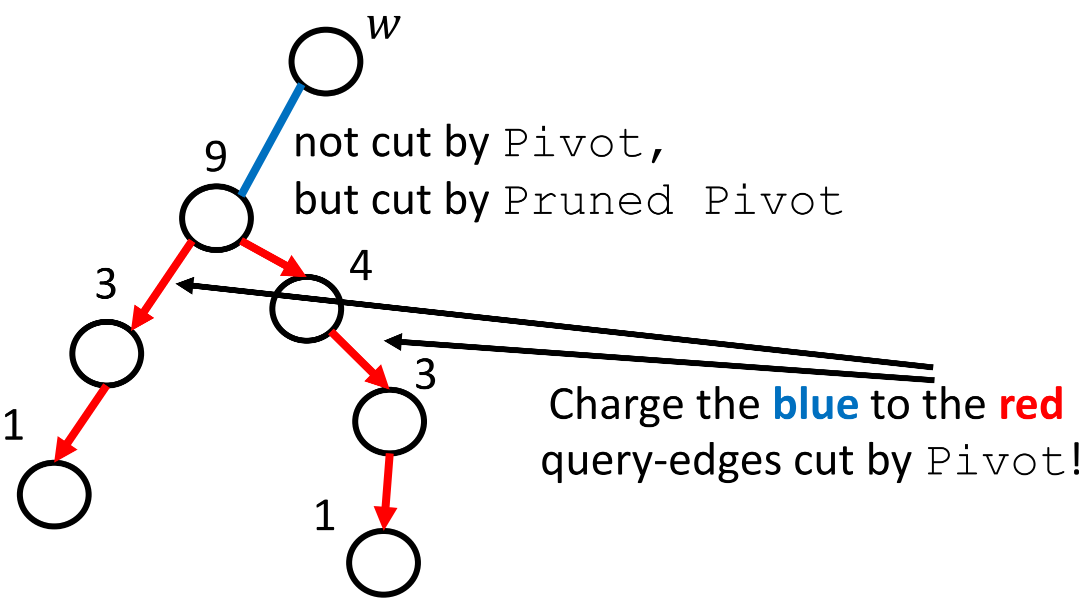
Our approach (Pruned Pivot)

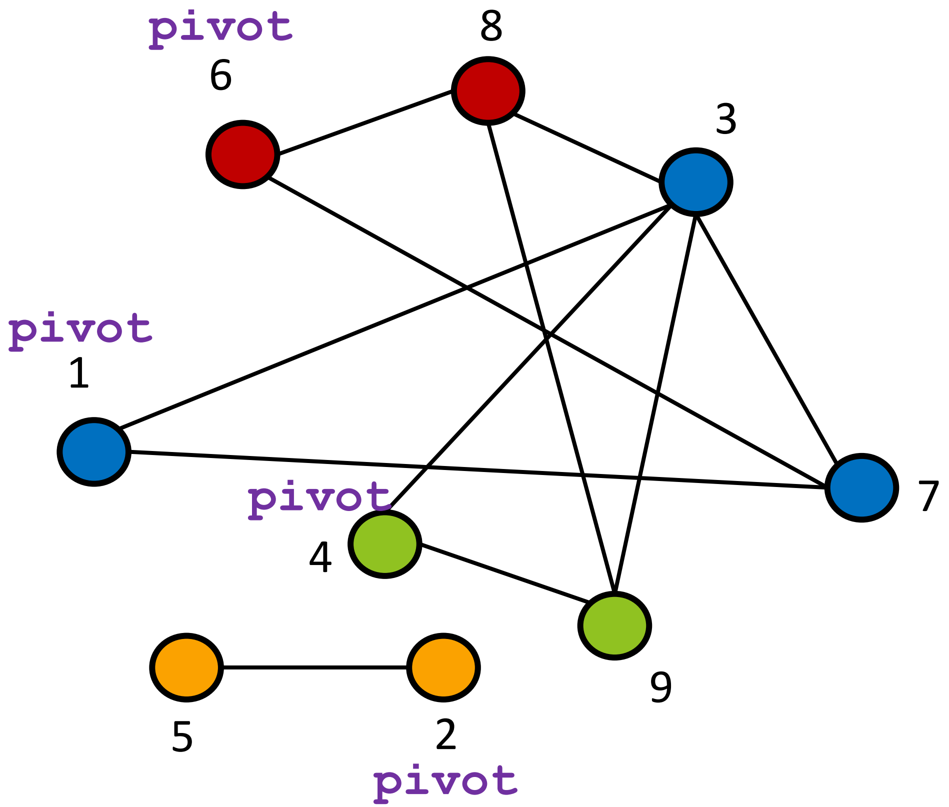
If the query tree for v has size greater than 1/ε, make v a singleton cluster.

Gives a 3+O(ε) approximation.

Pruned Pivot:

Why to expect it works?





Our approach (Pruned Pivot)

If the query tree for v has size greater than 1/ε, make v a singleton cluster.

Gives a 3+O(ε) approximation.

Outline

- Pivot [Ailon, Charikar, Newman, 2005]
- Our approach (Pruned Pivot)
- Implementations
- Implications on Maximal Independent Set
- Analysis

LCA(v)

1. Perform LCA queries from v.
2. If the number of queries exceeds $1/\varepsilon$, make v singleton.

MPC

1. Reduce the degree of each vertex to (at most) $1/\varepsilon$.
2. Collect $1/\varepsilon$-hop neighborhood of each vertex.
3. Simulate the algorithm for each vertex locally.

Dynamic

1. An edge is in expectation in O(1) many query trees.
2. k-highest-ranked neighbors can be accessed in expected O(k) time.

Dynamic

1. An edge is in expectation in O(1) many query trees.
2. k-highest-ranked neighbors can be accessed in expected O(k) time.

TBC (To Be Convinced)

Bucket 1	Bucket 2	…	Bucket $d(v)$
$\left[1, \frac{n}{d(v)}\right)$	$\left[\frac{n}{d(v)}, 2\frac{n}{d(v)}\right)$		$\left[(d(v)-1)\frac{n}{d(v)}, n\right]$

Outline

- Pivot [Ailon, Charikar, Newman, 2005]
- Our approach (Pruned Pivot)
- Implementations
- Implications on Maximal Independent Set
- Analysis

Maximal Independent Set (MIS)

Pivot = Randomized greedy algorithm for MIS

Maximal Independent Set (MIS)

Pivot = Randomized greedy algorithm for MIS

How many **LCA queries** needed to decide whether v is a pivot, i.e., whether v is in MIS?

Maximal Independent Set (MIS)

Pivot = Randomized greedy algorithm for MIS

How many **LCA queries** needed to decide whether v is a pivot, i.e., whether v is in MIS?

$\boldsymbol{\Delta}$ = maximum degree
$\overline{\boldsymbol{\Delta}}$ = average degree

Complexity	Comment	References
$2^{O(\Delta)}$	Expectation from any vertex	[Nguyen, Onak, 2008]
$O(\overline{\Delta})$	Expectation from a *random* vertex	[Yoshida, Yamamoto, Ito, 2009]

Maximal Independent Set (MIS)

Pivot = Randomized greedy algorithm for MIS

How many **LCA queries** needed to decide whether v is a pivot, i.e., whether v is in MIS?

Δ = maximum degree
$\bar{\Delta}$ = average degree

Complexity	Comment	References
$2^{O(\Delta)}$	Expectation from any vertex	[Nguyen, Onak, 2008]
$O(\bar{\Delta})$	Expectation from a *random* vertex	[Yoshida, Yamamoto, Ito, 2009]
$\Delta^{O(\Delta \log \Delta)}\, poly \log n$	Any vertex, whp	[Rubinfeld, Tamir, Vardi, Xie, 2011]
$\Delta^{O(\log^2 \Delta)} poly \log n$	Any vertex, whp	[Levi, Rubinfeld, Yodpinyanee, 2017]
$\Delta^{O(\Delta)} \log^* n$	Any vertex, whp	[Levi, Medina, 2017]
$\Delta^{O(\log \Delta)} poly \log n$	Any vertex, whp	[Ghaffari, 2016]
$\Delta^{O(\log \log \Delta)} poly \log n$	Any vertex, whp	[Ghaffari, Uitto, 2019]
$poly(\Delta) \log n$	Any vertex, whp	[Ghaffari, 2022]

Maximal Independent Set (MIS)

Pivot = Randomized greedy algorithm for MIS

How many **LCA queries** needed to decide whether v is a pivot, i.e., whether v is in MIS?

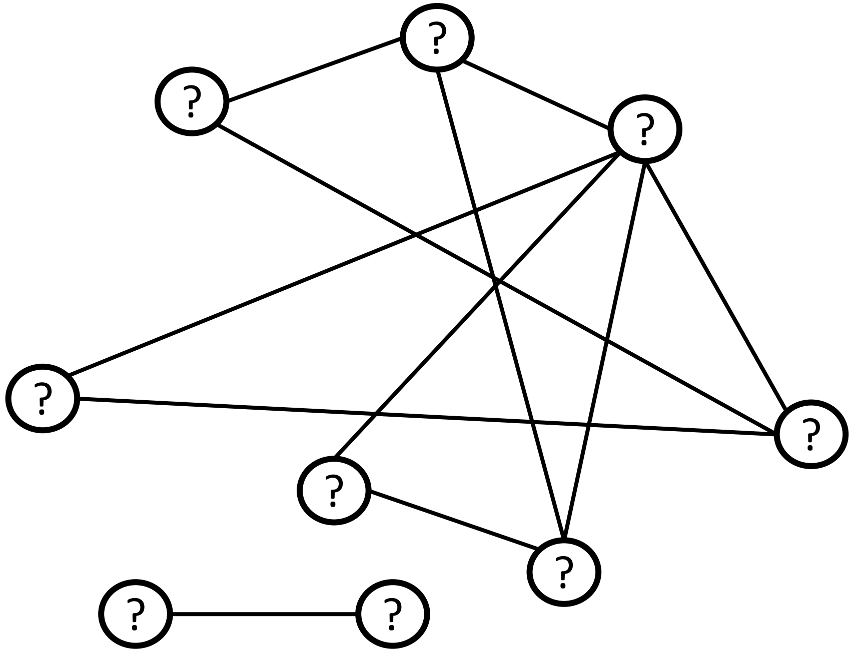
Δ = maximum degree
$\overline{\Delta}$ = average degree

Complexity	Comment	References
$2^{O(\Delta)}$	Expectation from any vertex	[Nguyen, Onak, 2008]
$O(\overline{\Delta})$	Expectation from a *random* vertex	[Yoshida, Yamamoto, Ito, 2009]
$\Delta^{O(\Delta \log \Delta)}\, poly \log n$	Any vertex, whp	[Rubinfeld, Tamir, Vardi, Xie, 2011]
$\Delta^{O(\log^2 \Delta)} poly \log n$	Any vertex, whp	[Levi, Rubinfeld, Yodpinyanee, 2017]
$\Delta^{O(\Delta)} \log^* n$	Any vertex, whp	[Levi, Medina, 2017]
$\Delta^{O(\log \Delta)} poly \log n$	Any vertex, whp	[Ghaffari, 2016]
$\Delta^{O(\log \log \Delta)} poly \log n$	Any vertex, whp	[Ghaffari, Uitto, 2019]
$poly(\Delta) \log n$	Any vertex, whp	[Ghaffari, 2022]
$O(\overline{\Delta})$	Expectation from a *random* vertex	**this work**

Outline

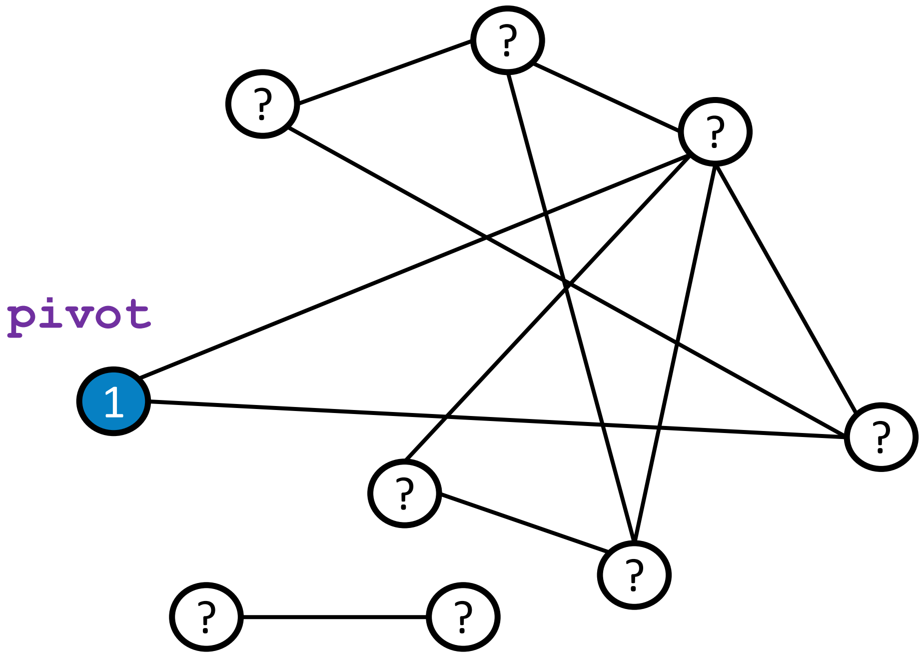
- Pivot [Ailon, Charikar, Newman, 2005]
- Our approach (Pruned Pivot)
- Implementations
- Implications on Maximal Independent Set
- Analysis

Setup



1. Reveal ranks one at the time.

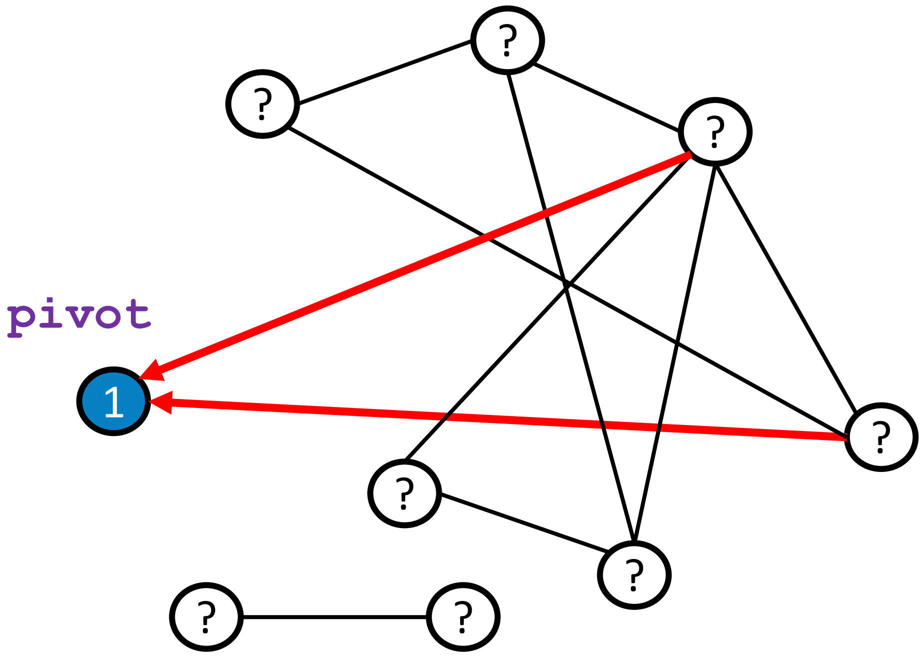
Setup



1. Reveal ranks one at the time.

pivot = in MIS

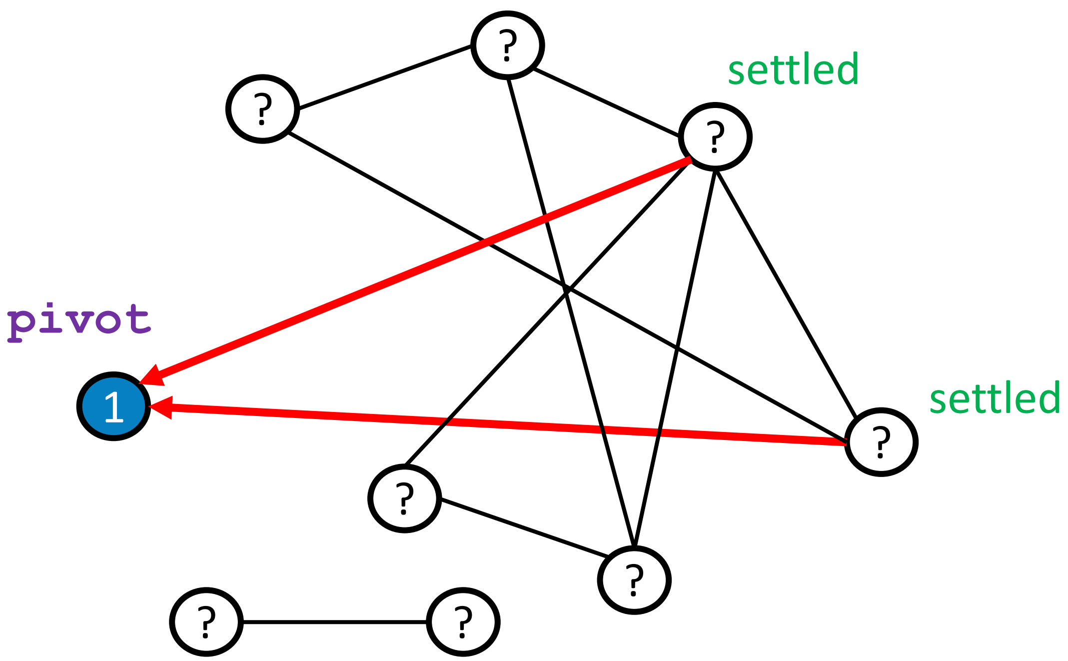
Setup



1. Reveal ranks one at the time.

pivot = in MIS

Setup

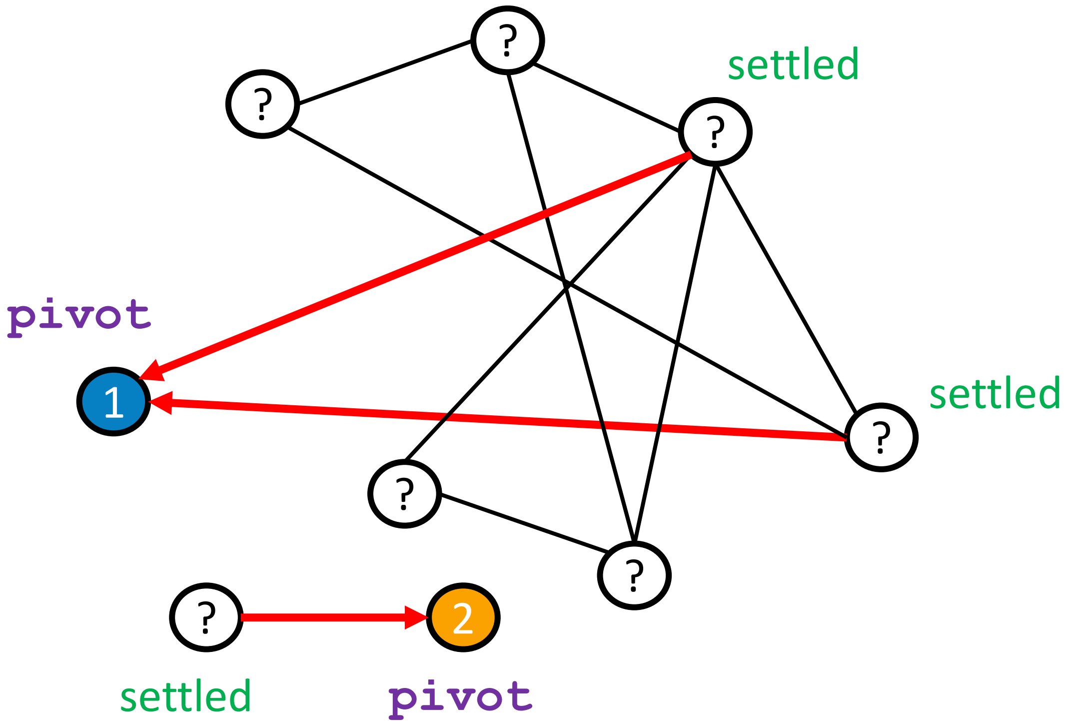


1. Reveal ranks one at the time.

`pivot` = in MIS

v can be settled **before** its rank is revealed.

Setup

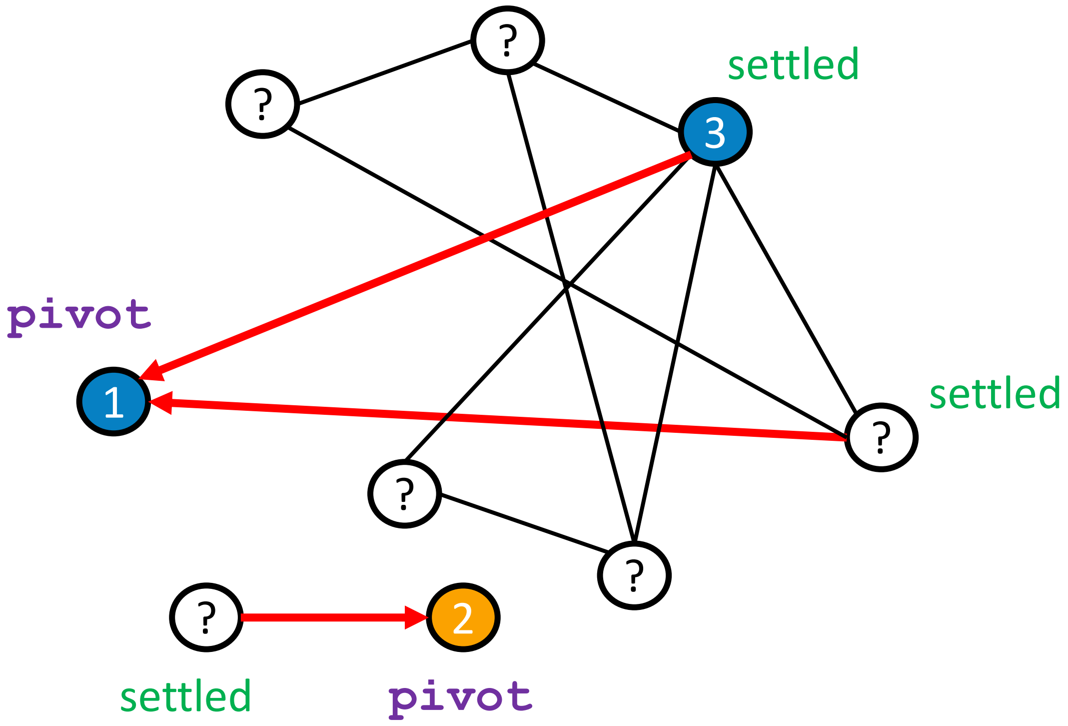


1. Reveal ranks one at the time.

pivot = in MIS

v can be settled **before** its rank is revealed.

Setup

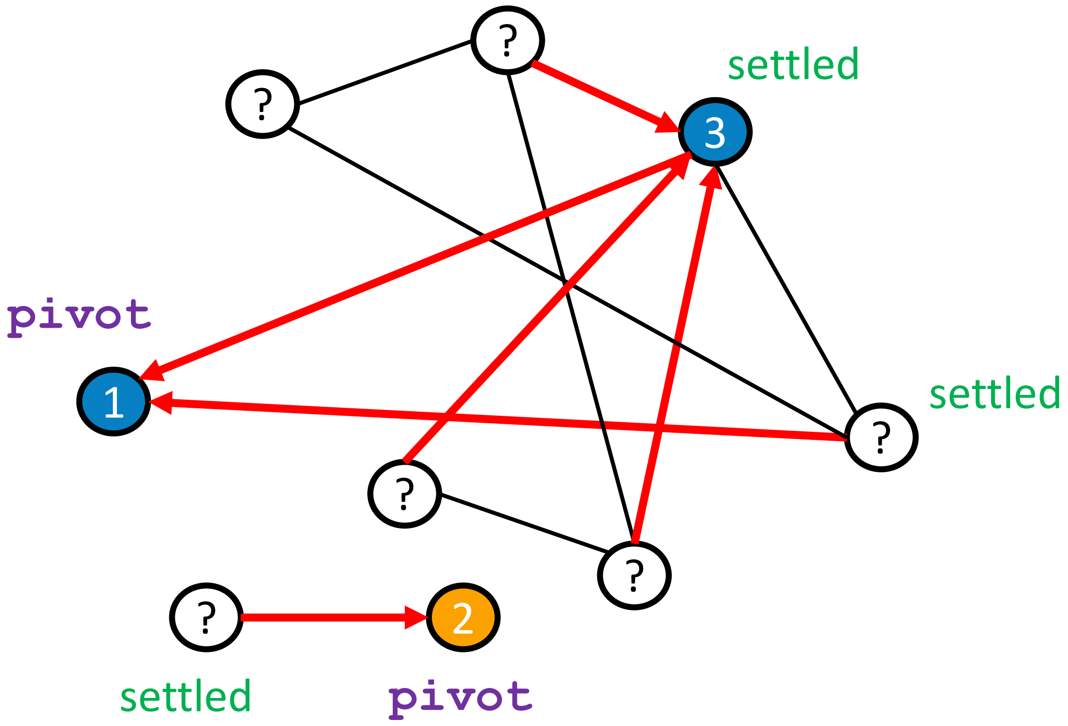


1. Reveal ranks one at the time.

pivot = in MIS

v can be settled **before** its rank is revealed.

Setup

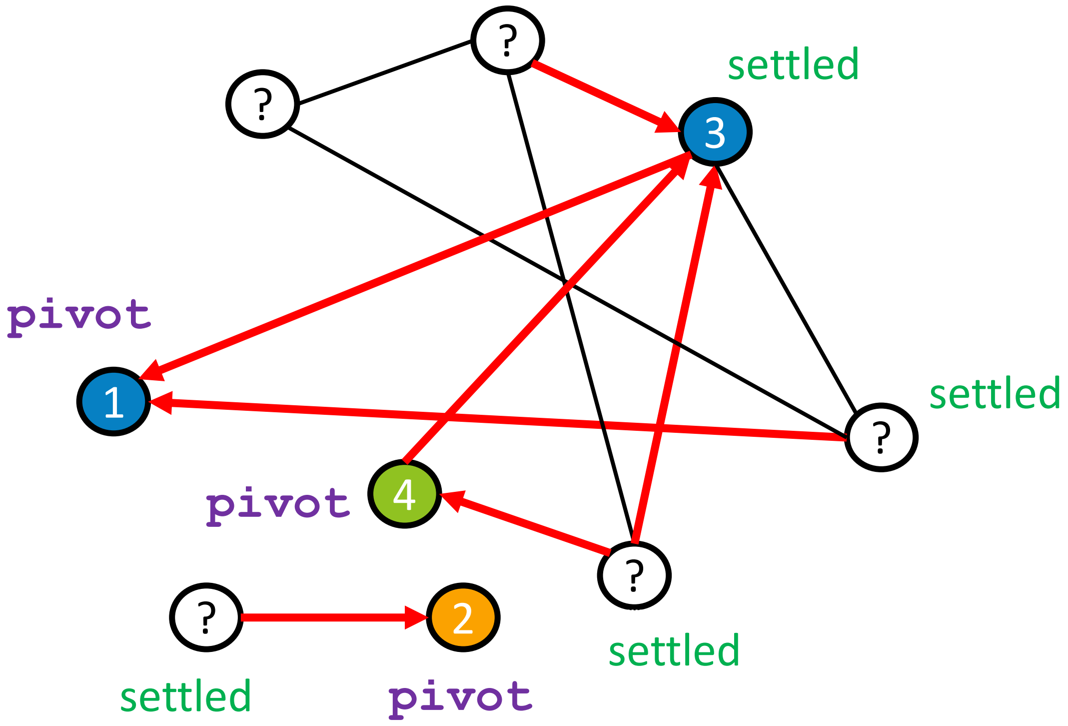


1. Reveal ranks one at the time.

pivot = in MIS

v can be settled **before** its rank is revealed.

Setup

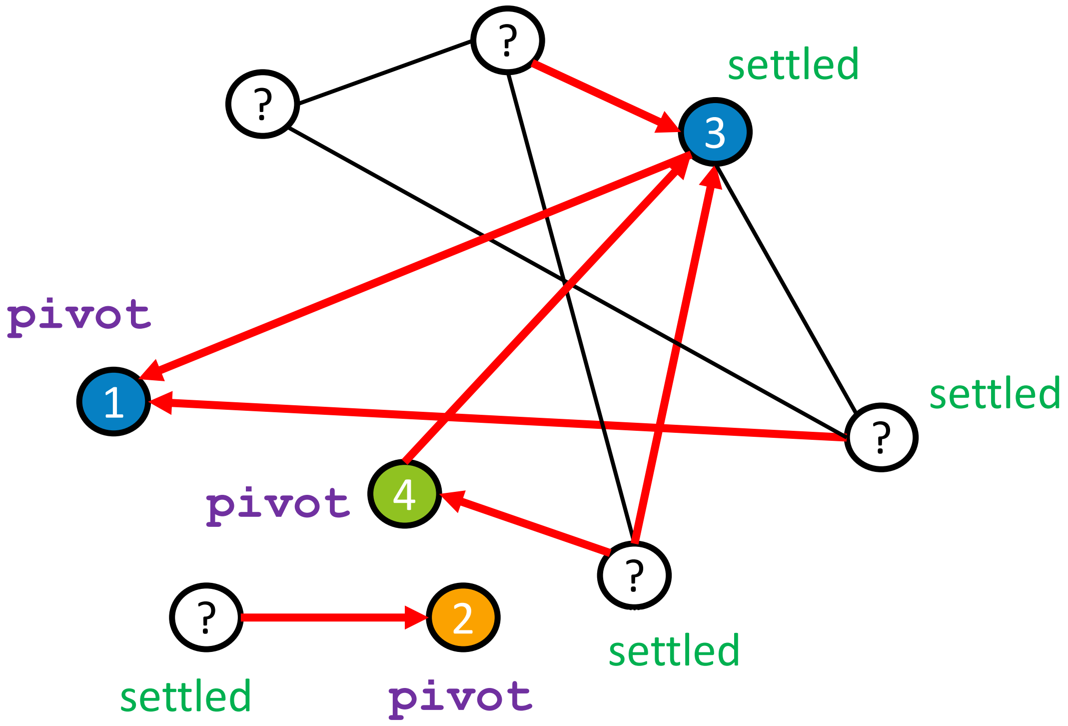


1. Reveal ranks one at the time.

`pivot` = in MIS

v can be settled **before** its rank is revealed.

Setup



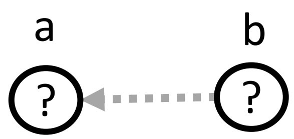
1. Reveal ranks one at the time.
2. How large is the **in-tree** to a vertex?
   How many times **an edge** is queried?

pivot = in MIS

v can be settled **before** its rank is revealed.

An attempt

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

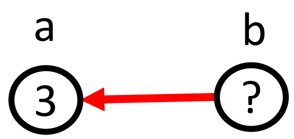


t = 0

v can be settled **before** its rank is revealed.

An attempt

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

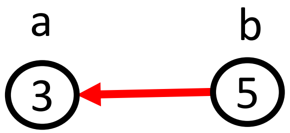


t = 3

v can be settled **before** its rank is revealed.

An attempt

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

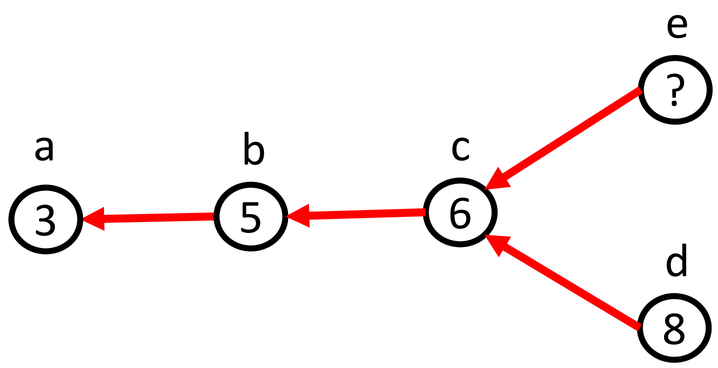


t = 5

v can be settled **before** its rank is revealed.

An attempt

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?



t = 9

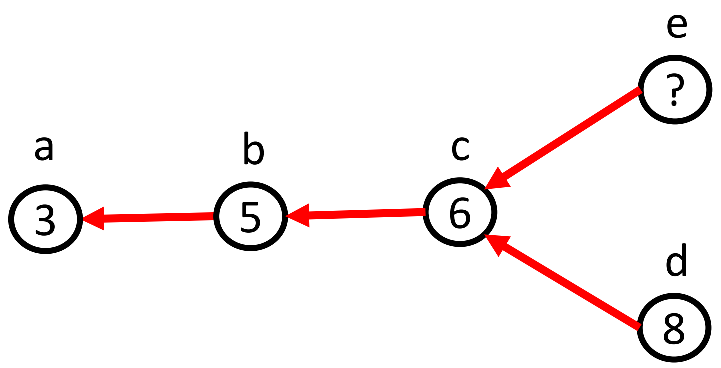
v can be settled **before** its rank is revealed.

An attempt

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

Idea:

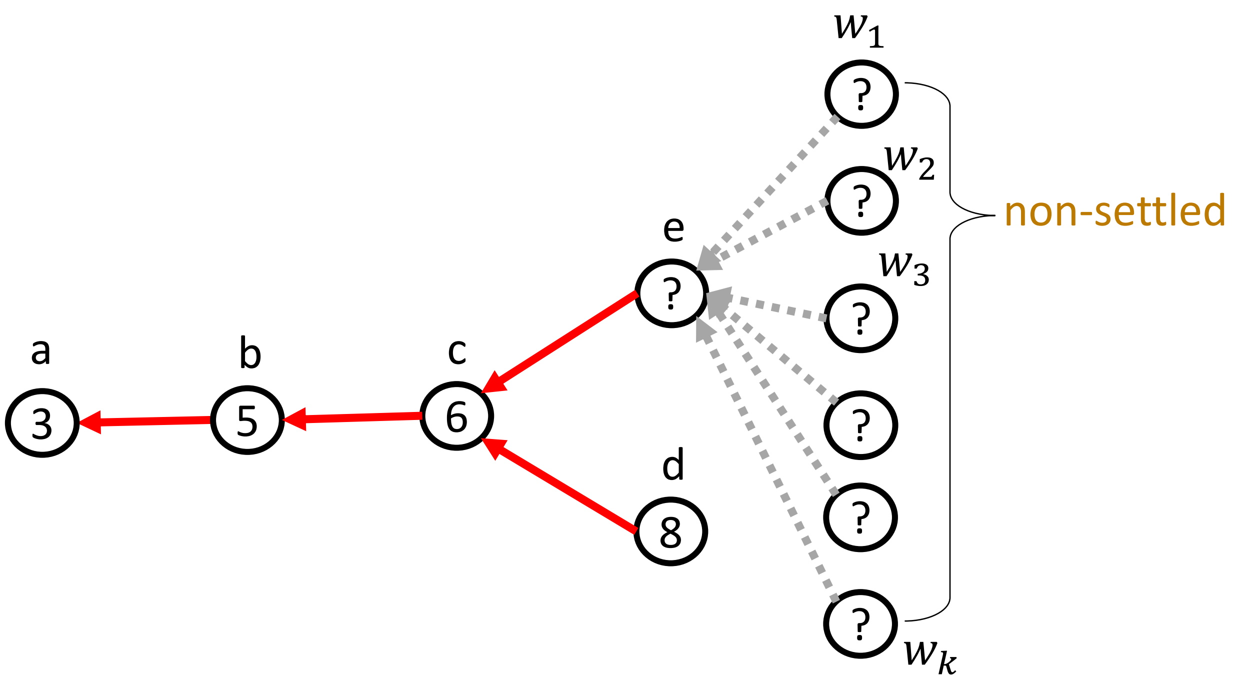
- Condition on this state
- Ask how many e's (non-settled) neighbors will query e at time t = 10, 11, …, n.



t = 9

v can be settled **before** its rank is revealed.

An attempt



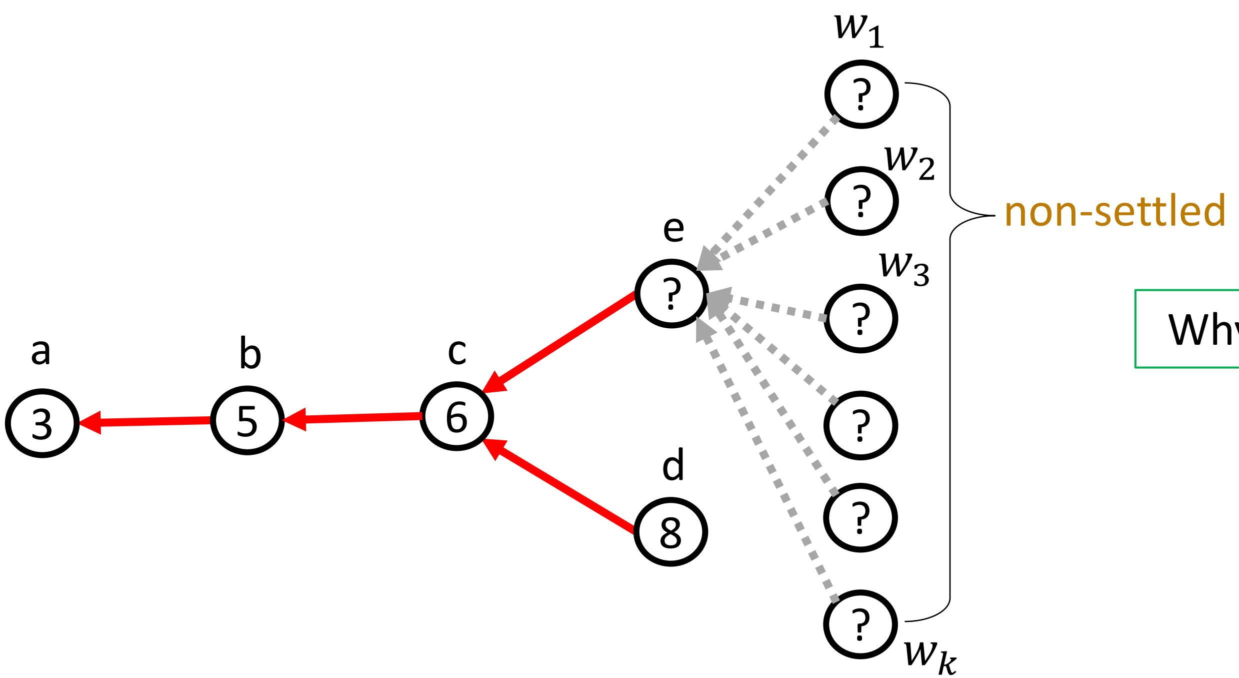
t = 9

v can be settled **before** its rank is revealed.

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

Idea:
- Condition on this state
- Ask how many e's (non-settled) neighbors will query e at time t = 10, 11, …, n.

An attempt



t = 9

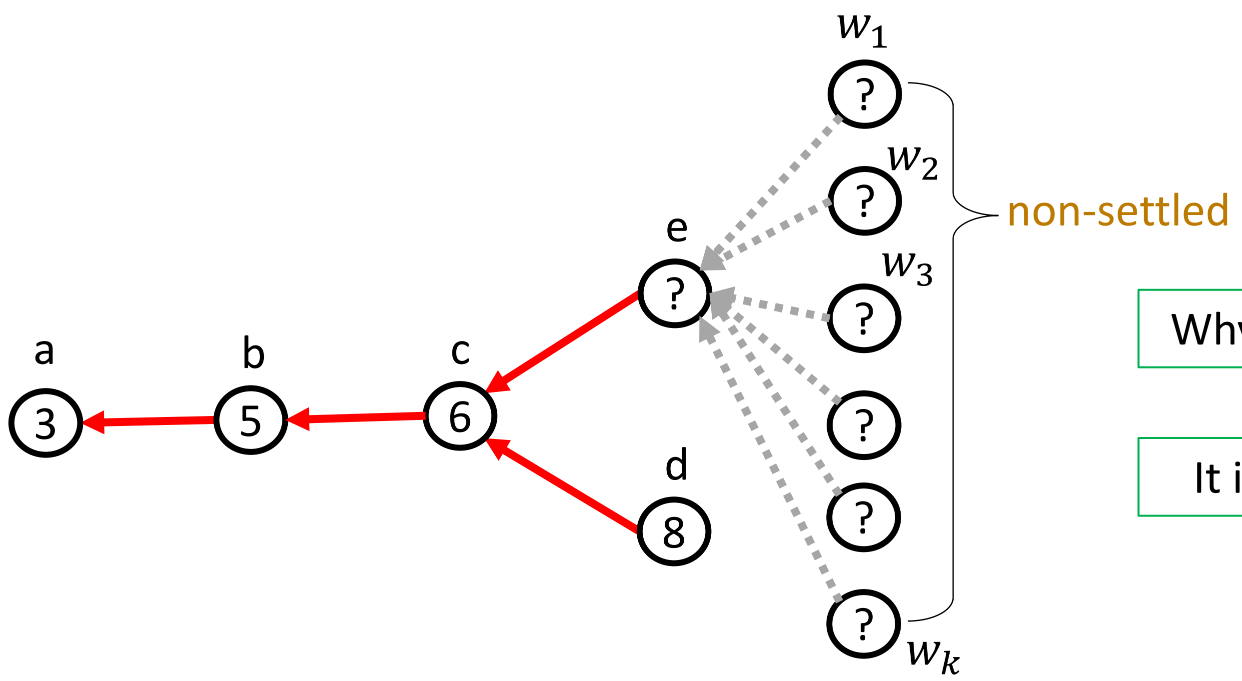
v can be settled **before** its rank is revealed.

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

Idea:
- Condition on this state
- Ask how many e's (non-settled) neighbors will query e at time t = 10, 11, …, n.

Why is this not an issue in the entire process?

An attempt



t = 9

v can be settled **before** its rank is revealed.

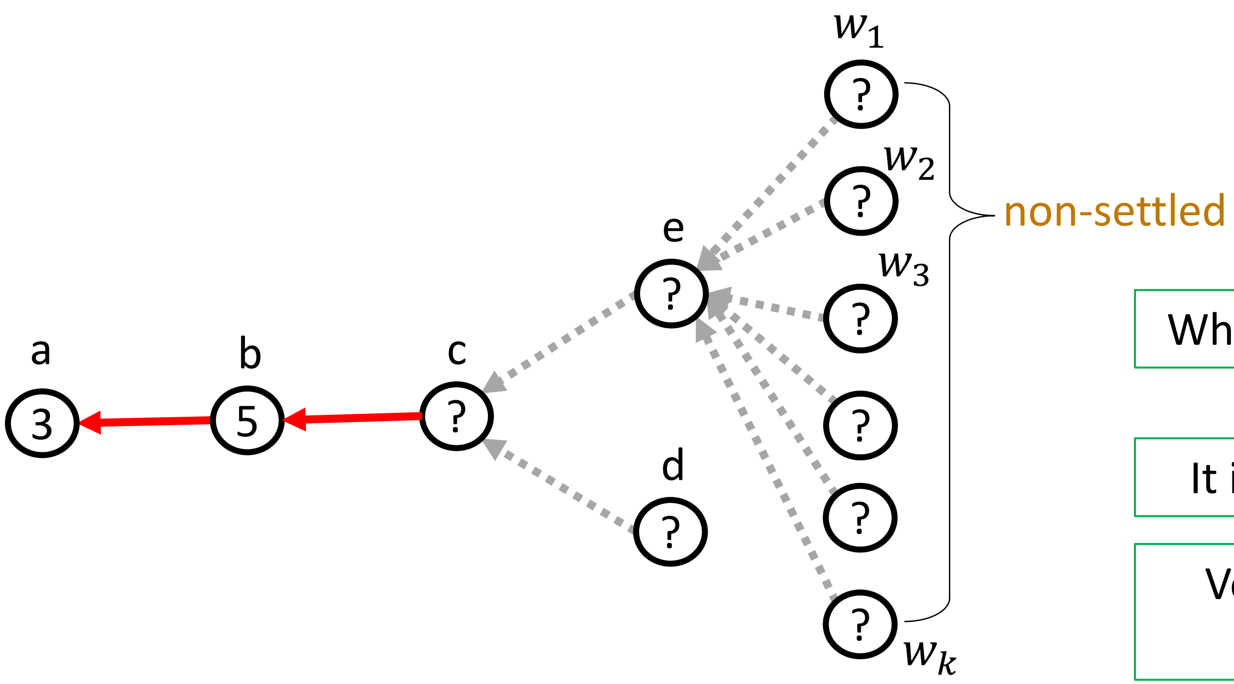
1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

Idea:
- Condition on this state
- Ask how many e's (non-settled) neighbors will query e at time t = 10, 11, …, n.

Why is this not an issue in the entire process?

It is **unlikely** e will query c in the first place!

An attempt (and an idea)



1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

Idea:
- Condition on this state
- Ask how many e's (non-settled) neighbors will query e at time t = 10, 11, …, n.

Why is this not an issue in the entire process?

It is **unlikely** e will query c in the first place!

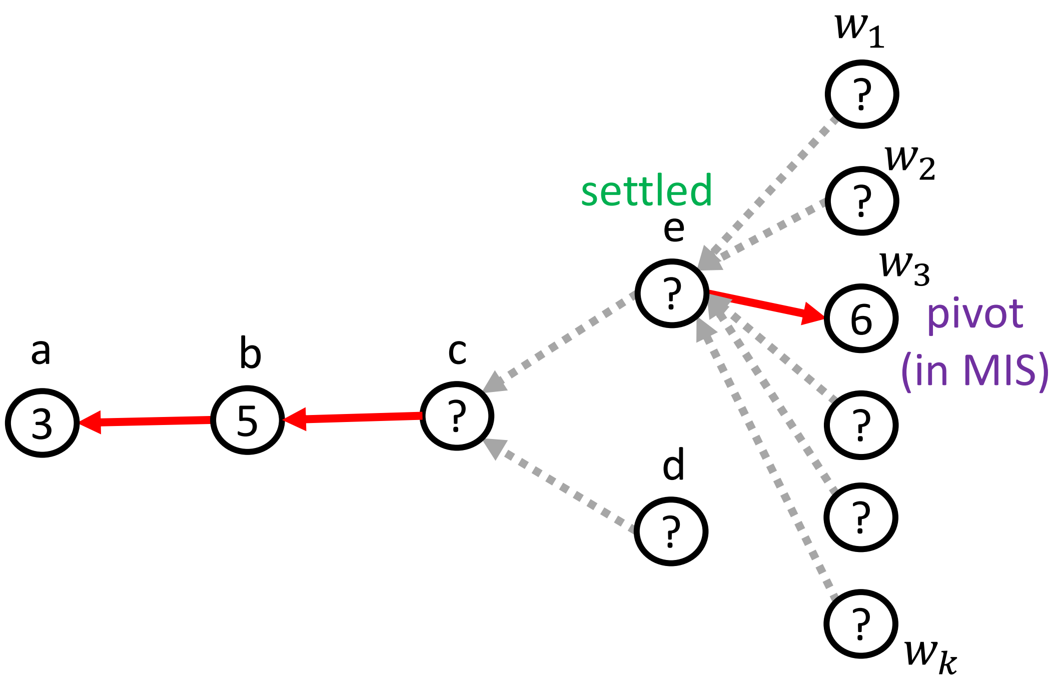
Very likely that w_1 or w_2 or w_3 or … or w_k will have rank **higher** than c.

t = 5

v can be settled **before** its rank is revealed.

An attempt (and an idea)

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?



t = 6

v can be settled **before** its rank is revealed.

Idea:
- Condition on this state
- Ask how many e's (non-settled) neighbors will query e at time t = 10, 11, …, n.

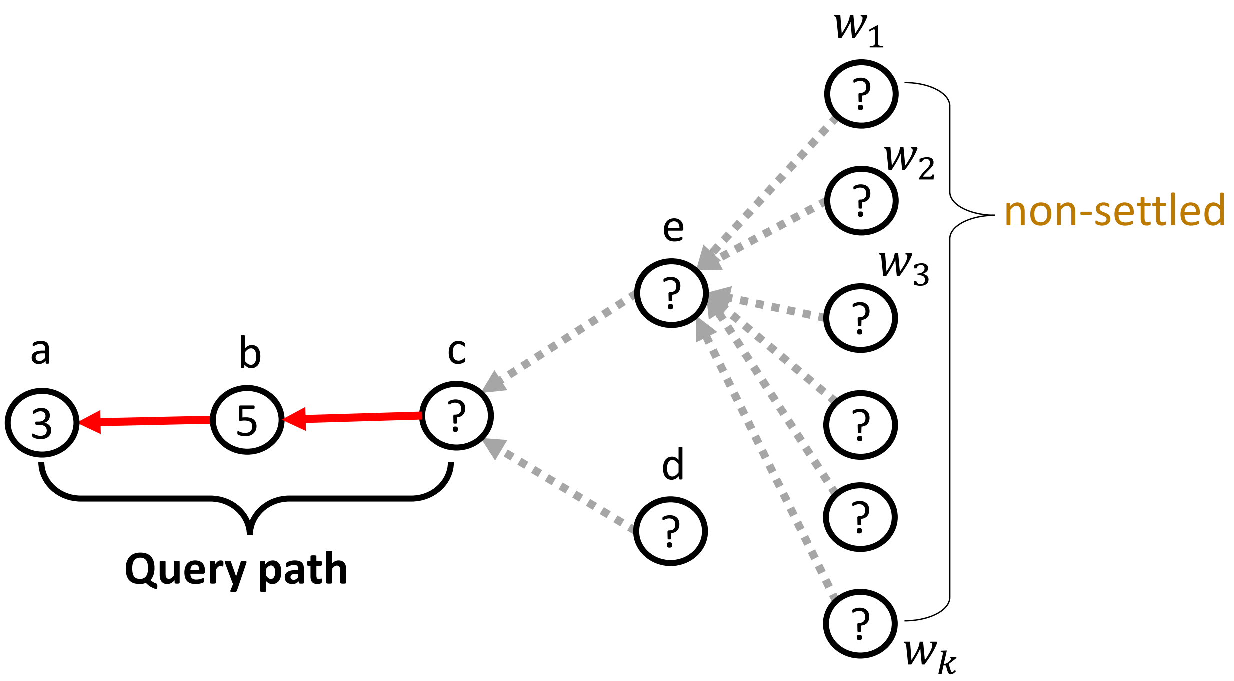
Why is this not an issue in the entire process?

It is **unlikely** e will query c in the first place!

Very likely that w_1 or w_2 or w_3 or … or w_k will have rank **higher** than c.

Formalizing

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

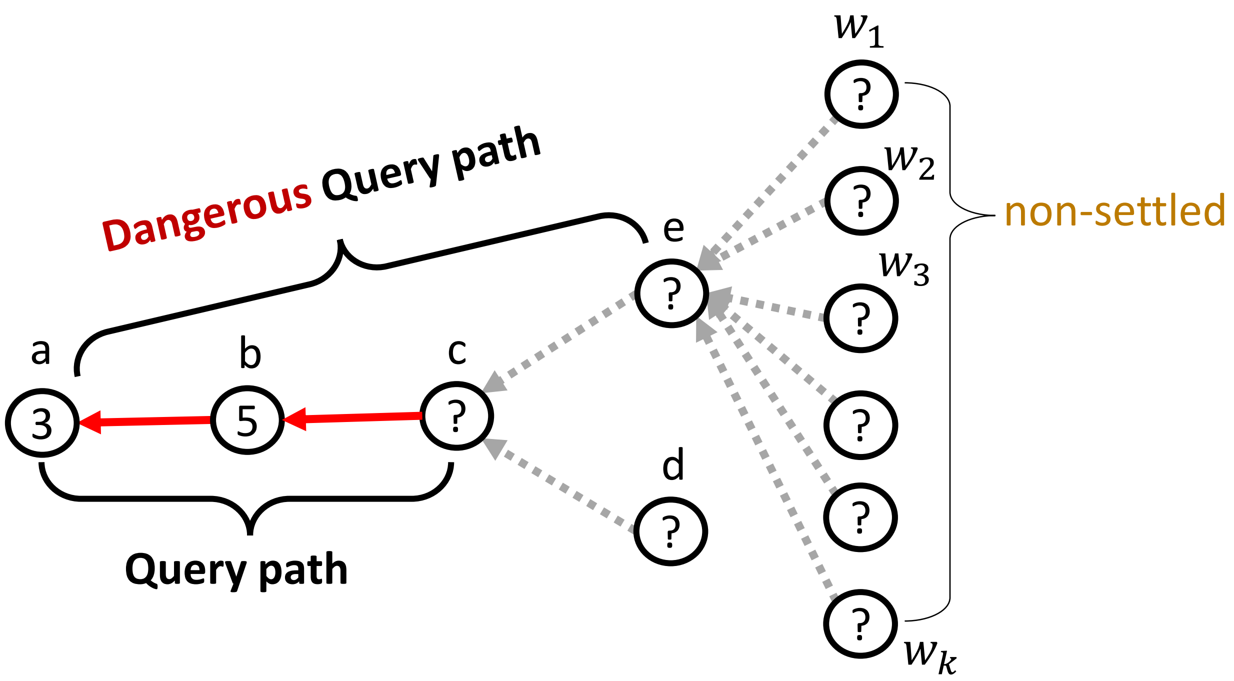


t = 5

v can be settled **before** its rank is revealed.

Formalizing

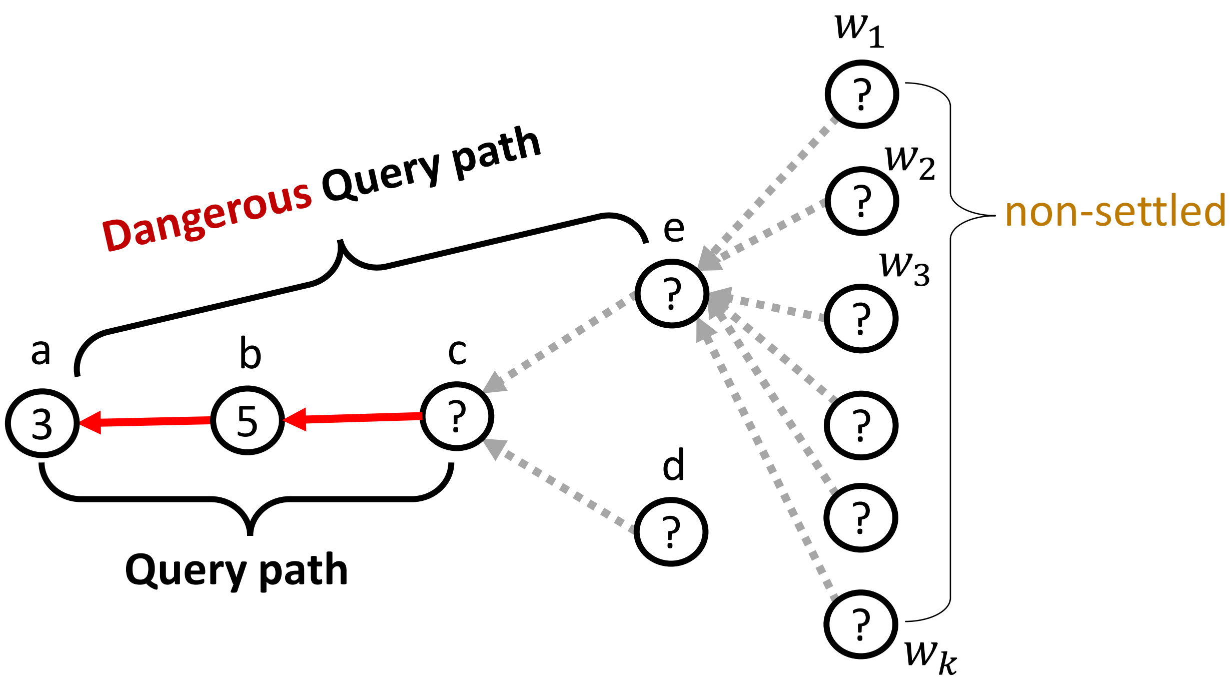
1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?



t = 5

v can be settled **before** its rank is revealed.

Formalizing



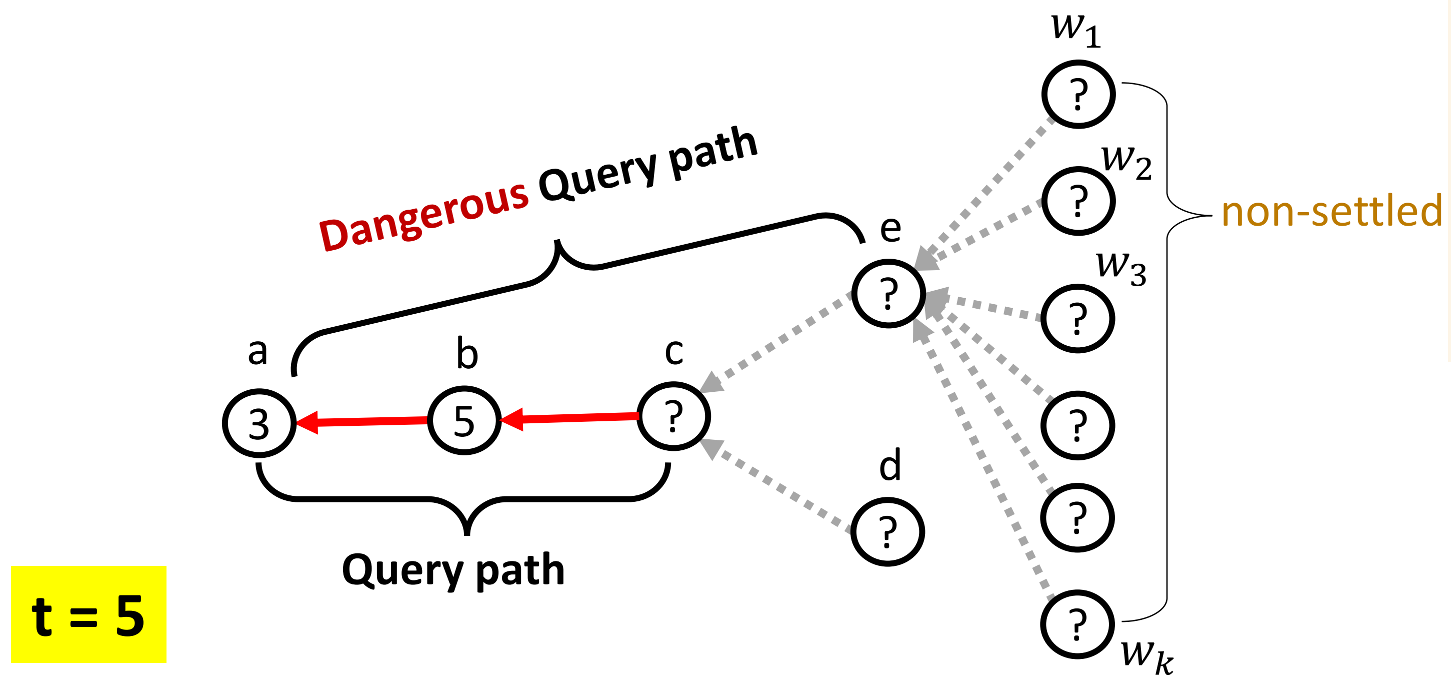
t = 5

v can be settled **before** its rank is revealed.

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

1. If among $c, e, w_1, w_2, \ldots, w_k$, the rank of c is set next, then:
   - new Query path $e \to c \to b \to a$
   - Potentially many w_i-es query $e \to c \to b \to a$; new Dangerous query paths $w_i \to e \to c \to b \to a$ (BAD)
2.

Formalizing

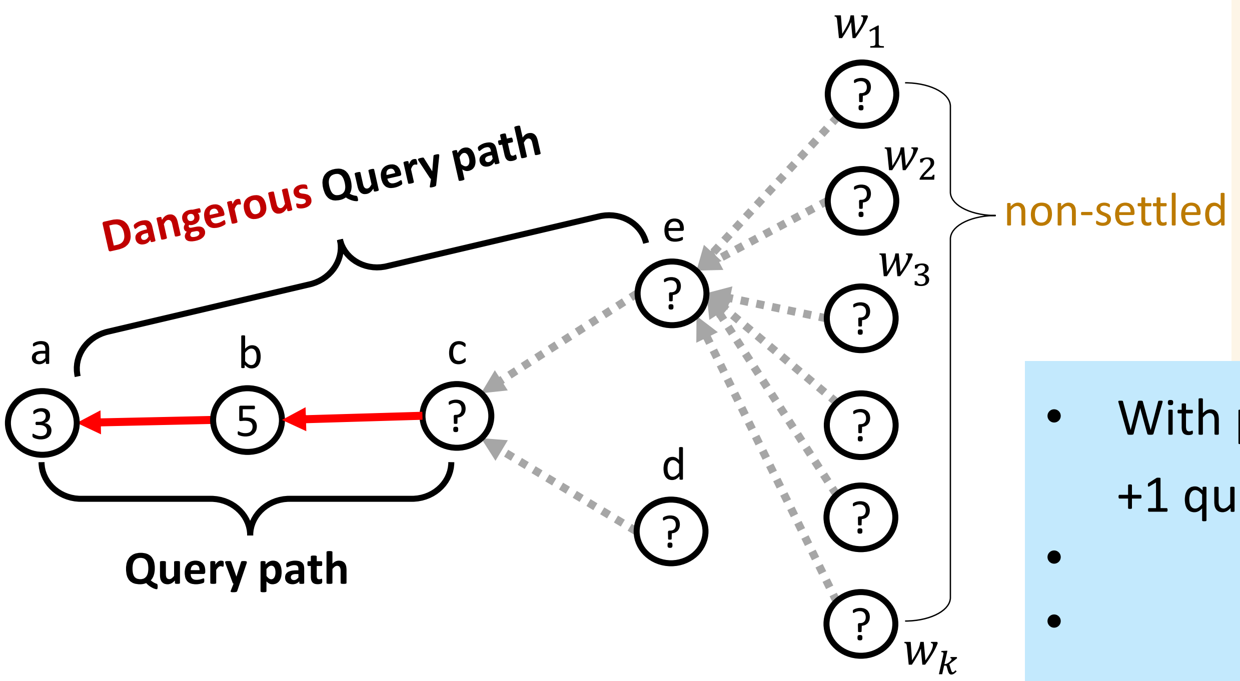


v can be settled **before** its rank is revealed.

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

1. If among $c, e, w_1, w_2, \ldots, w_k$, the rank of c is set next, then:
   - new Query path $e \to c \to b \to a$
   - Potentially many w_i-es query $e \to c \to b \to a$; new Dangerous query paths $w_i \to e \to c \to b \to a$ (BAD)
2. Otherwise, e never queries c! (GREAT)

Formalizing



t = 5

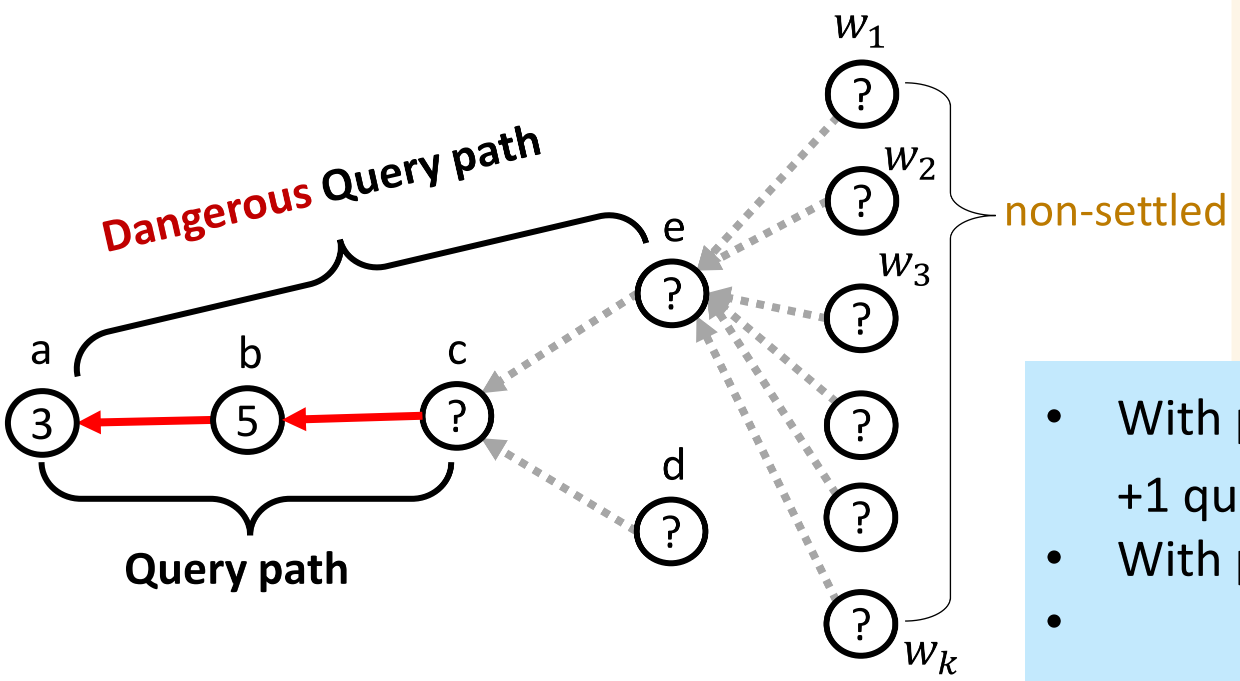
v can be settled **before** its rank is revealed.

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

1. If among $c, e, w_1, w_2, \ldots, w_k$, the rank of c is set next, then:
   - new Query path $e \to c \to b \to a$
   - Potentially many w_i-es query $e \to c \to b \to a$; new Dangerous query paths $w_i \to e \to c \to b \to a$ (BAD)
2. Otherwise, e never queries c! (GREAT)

- With probability $\frac{1}{k+1}$:
  +1 query path **and** k Dangerous query paths
-
-
-

Formalizing



t = 5

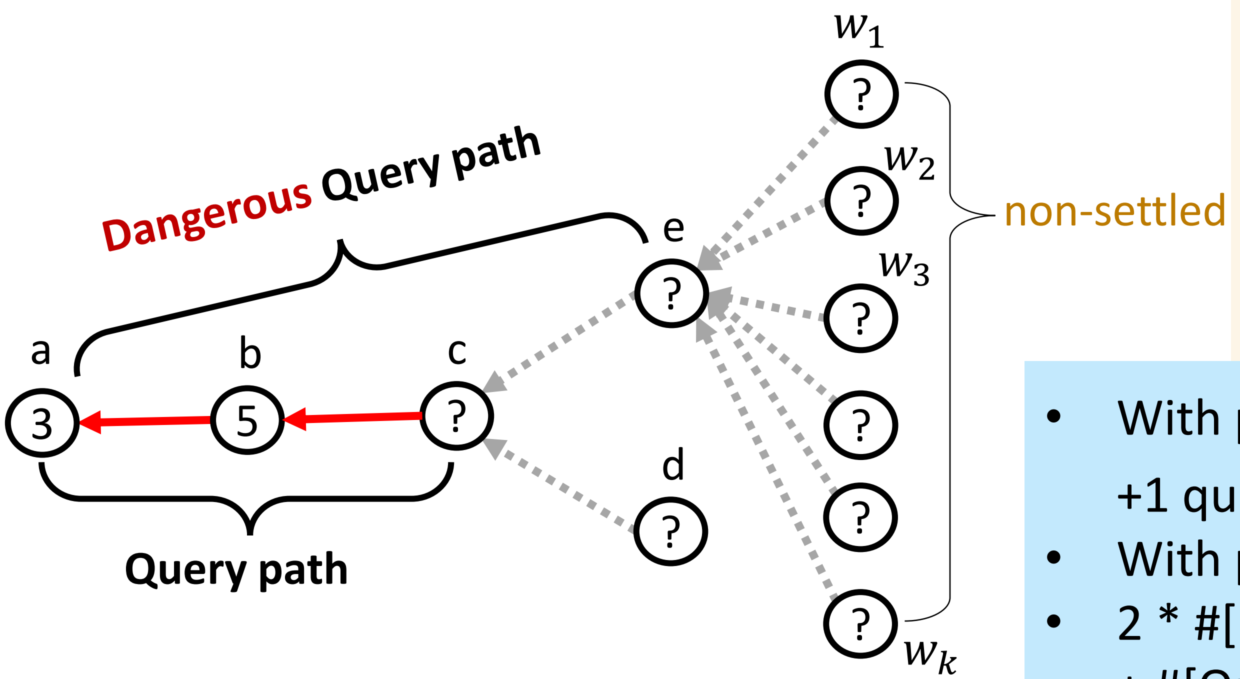
v can be settled **before** its rank is revealed.

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

1. If among $c, e, w_1, w_2, \dots, w_k$, the rank of c is set next, then:
   - new Query path $e \to c \to b \to a$
   - Potentially many w_i-es query $e \to c \to b \to a$; new Dangerous query paths $w_i \to e \to c \to b \to a$ (BAD)
2. Otherwise, e never queries c! (GREAT)

- With probability $\frac{1}{k+1}$:
  +1 query path **and** k Dangerous query paths
- With probability 1: -1 Dangerous query path
-
-

Formalizing



t = 5

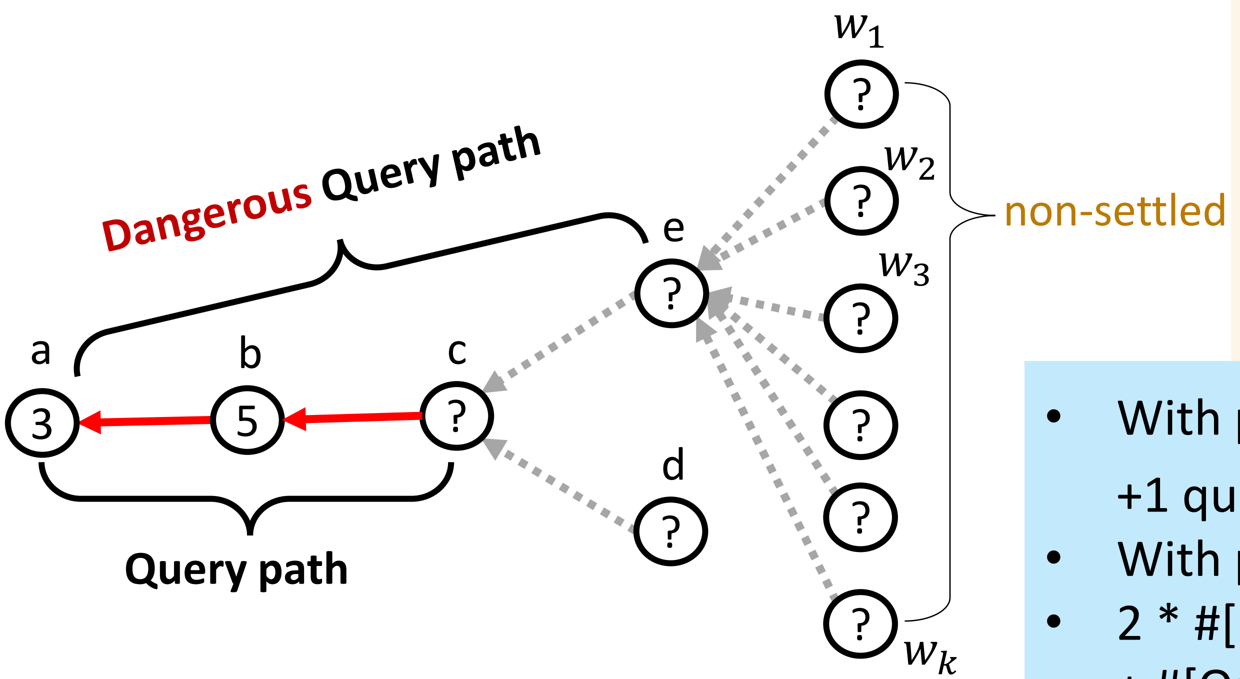
v can be settled **before** its rank is revealed.

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

1. If among $c, e, w_1, w_2, \ldots, w_k$, the rank of c is set next, then:
   - new Query path $e \to c \to b \to a$
   - Potentially many w_i-es query $e \to c \to b \to a$; new Dangerous query paths $w_i \to e \to c \to b \to a$ (BAD)
2. Otherwise, e never queries c! (GREAT)

- With probability $\frac{1}{k+1}$:
  +1 query path **and** k Dangerous query paths
- With probability 1: -1 Dangerous query path
- 2 * #[Dangerous query paths for (a, b)]
  + #[Query paths for (a, b)] is a supermartingale.
-

Formalizing



t = 5

v can be settled **before** its rank is revealed.

1. Reveal ranks one at the time.
2. How many times **edge (a, b)** is queried?

1. If among $c, e, w_1, w_2, \dots, w_k$, the rank of c is set next, then:
   - new Query path $e \to c \to b \to a$
   - Potentially many w_i-es query $e \to c \to b \to a$; new Dangerous query paths $w_i \to e \to c \to b \to a$ (BAD)
2. Otherwise, e never queries c! (GREAT)

- With probability $\frac{1}{k+1}$:
  +1 query path **and** k Dangerous query paths
- With probability 1: -1 Dangerous query path
- 2 * #[Dangerous query paths for (a, b)]
  + #[Query paths for (a, b)] is a supermartingale.
- At t = 0, that sum equals 2 for each edge (a, b).

Open questions

Other applications of this analysis.

MIS in $poly\ \Delta$ LCA queries in expectation from **any** vertex.