Generalized Sorting with Predictions

Pinyan Lu
SUFE

Xuandi Ren
PKU

Enze Sun
SJTU

Yubo Zhang
PKU

January, 2021

Sorting → Generalized Sorting

- one of the most basic computational tasks
- $\Theta(n \log n)$ comparisons are necessary and sufficient to sort n elements
- What if only a subset of comparisons is allowed?
- ⇒ generalized sorting problem (sorting with forbidden pairs), introduced in [HKK11]
  - only a subset of comparisons allowed
  - each of the same cost
  - $\tilde{O}(n^{1.5})$ comparisons are sufficient [HKK11]

Graph Model

- The problem can be viewed as a graph problem for convenience

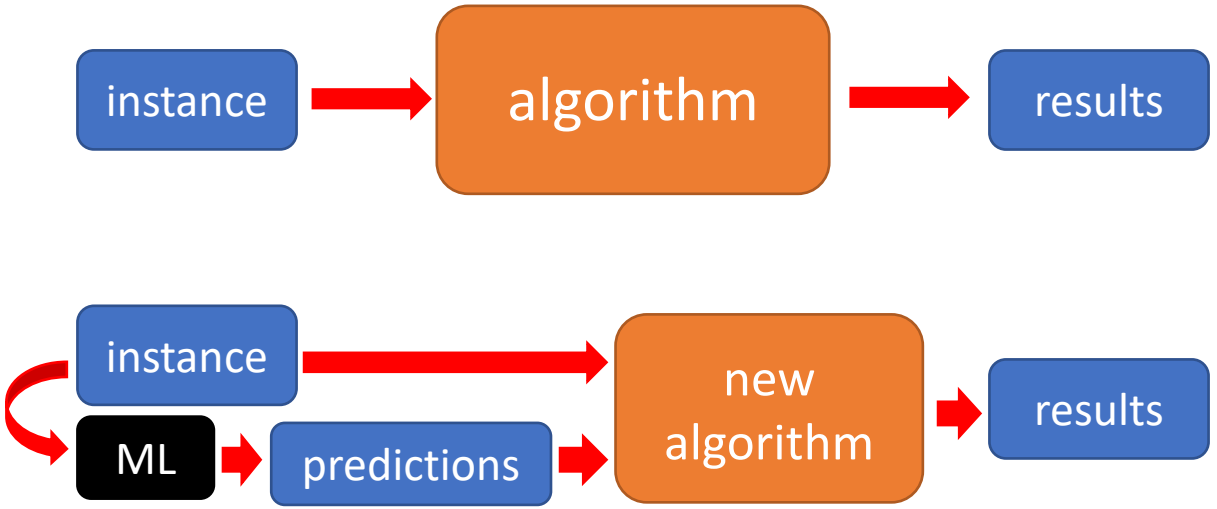
Input: an undirected graph $G = (V, E)$
each $v \in V$ represents an element,
each $(u, v) \in E$ represents an allowed comparison

Iteratively:
probe $(u, v) \in E$,
receive $u < v$ or $v < u$

Output: a total order of all elements
i.e. a sequence $v_1 < v_2 < \cdots < v_n$
$v_i \in V$

- It is guaranteed:
  - There is a orientation $\vec{G} = (V, \vec{E})$ of G representing the underlying total order
  - $\vec{G}$ is acyclic and there is a Hamiltonian path in $\vec{G}$
  - $\vec{G}$ is fixed at the beginning
- The goal is to find out the underlying Hamiltonian path using the smallest number of queries!

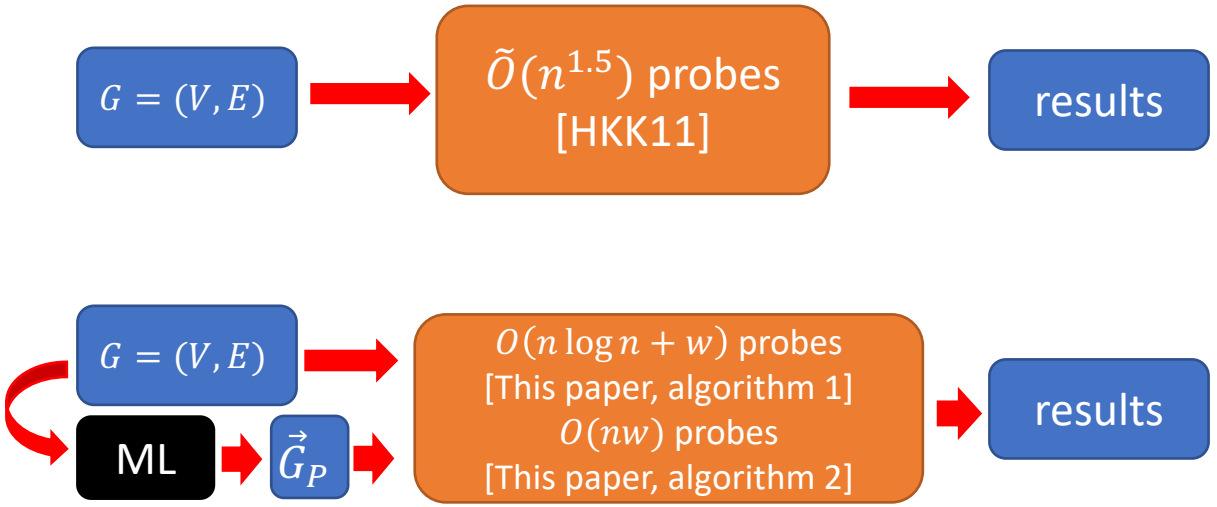
Algorithm with Predictions



- ***Consistency:*** has near optimal performance when the predictions are good
- ***Robustness:*** is no worse than the prediction-less case when the predictions have large errors

The quality of predictions is measured by an amount w
We want: the better the prediction, the better the performance

Generalized Sorting with Predictions

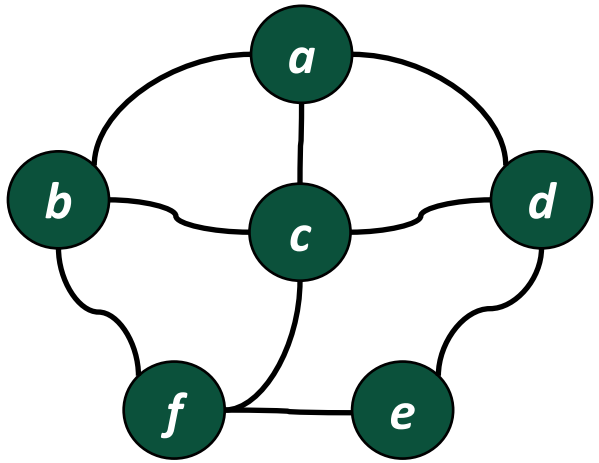


Predictions: $\vec{G}_P = (V, \vec{P}) =$ an orientation of G

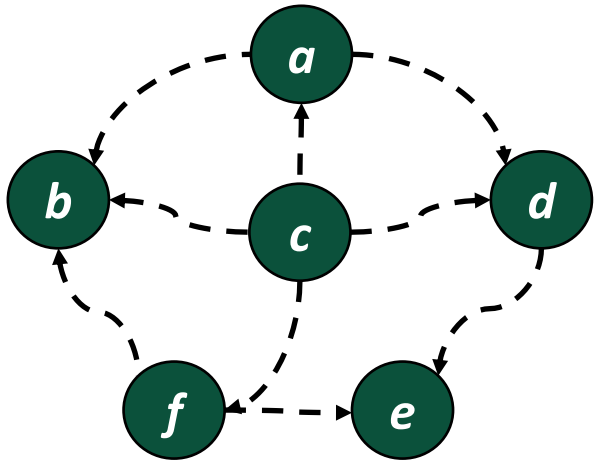
Measurement: $w = |\vec{P} \backslash \vec{E}| =$ #mis-predicted edges

- ***Consistency:*** when w is very small, almost reaches the best possible
- ***Robustness:*** in the query model, combining an $O(f(n))$ algorithm $\mathcal{A}$ and an $O(g(n))$ algorithm $\mathcal{B}$ leads to an $O(\min(f(n), g(n)))$ algorithm $\mathcal{C}$, thus robustness is trivially satisfied

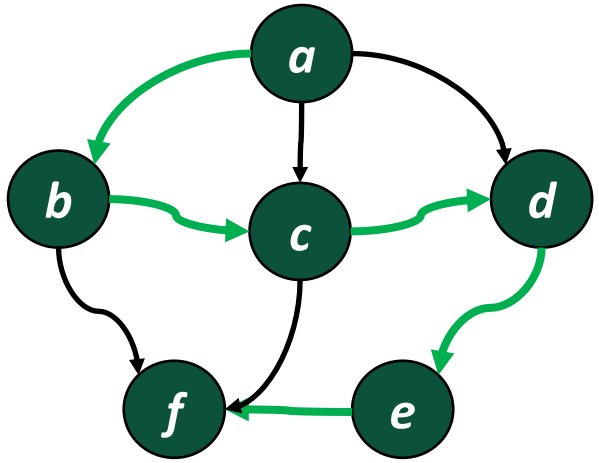
An Example



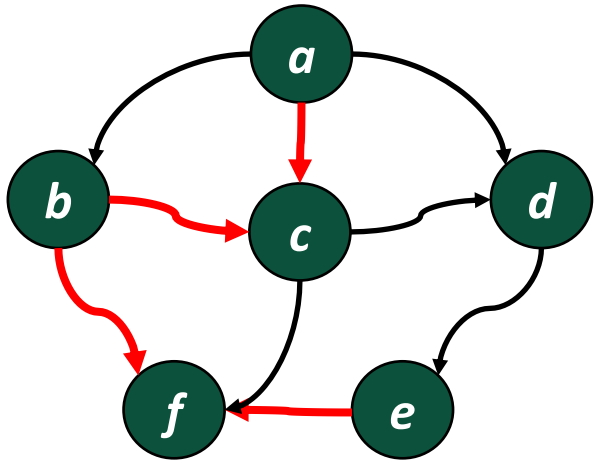
G : the undirected graph (input)



$\vec{G}_P$: the predicted digraph (input)



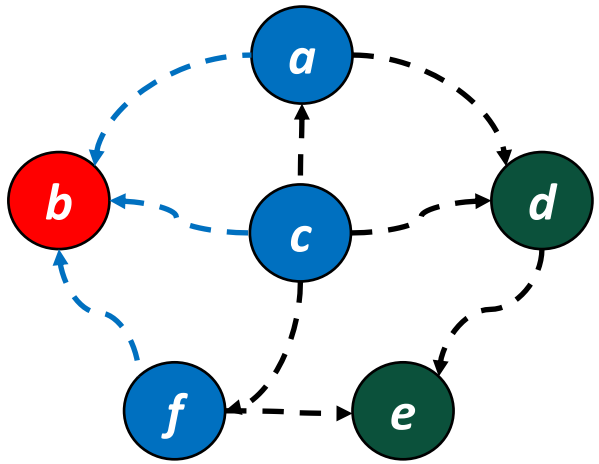
$\vec{G}$: the correct digraph



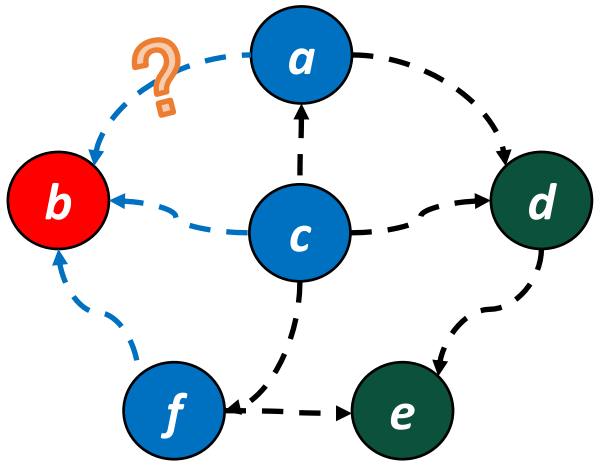
w = #mis-predicted edges = 4

: allowed comparisons
: the predicted directions
: correct directions
: edges in the Hamiltonian path
: mis-predicted edges

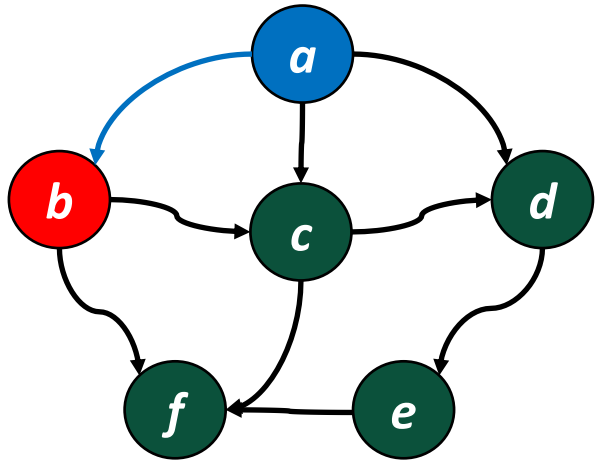
Some Notations



$N_b = \mathcal{N}_{in}(\vec{G}_P, b) = \{a, c, f\}$



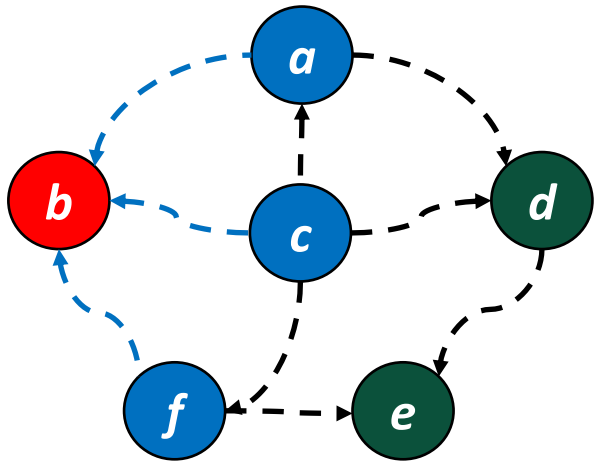
$T_b = \{a, c, f\}$



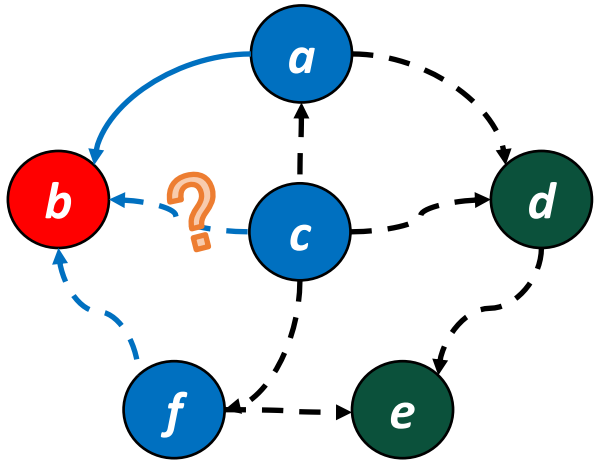
$S_b = \{a\}$

- $N_u \triangleq \mathcal{N}_{in}(\vec{G}_P, u) = u$'s in-neighbors in the predicted graph
- $S_u \triangleq u$'s *real* in-neighbors among N_u
- $T_u \triangleq u$'s in-neighbors among N_u which are *not yet wrong*
  - *either correct or unprobed*

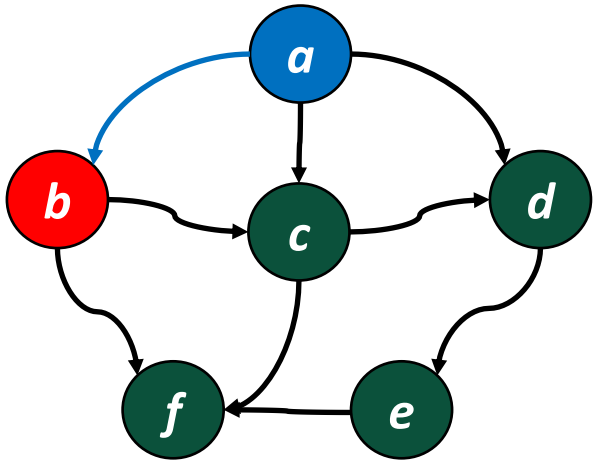
Some Notations



$N_b = \mathcal{N}_{in}(\vec{G}_P, b) = \{a, c, f\}$



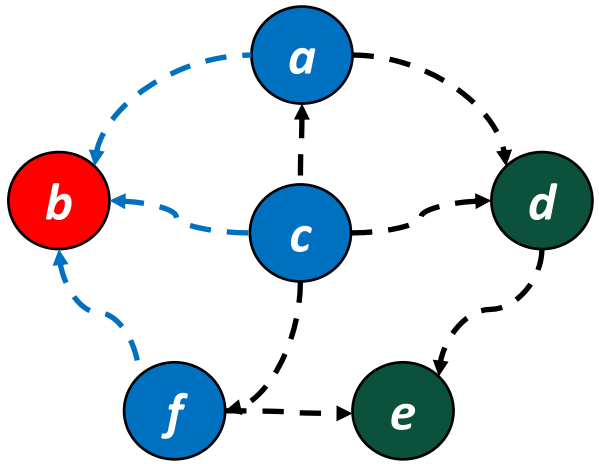
$T_b = \{a, c, f\}$



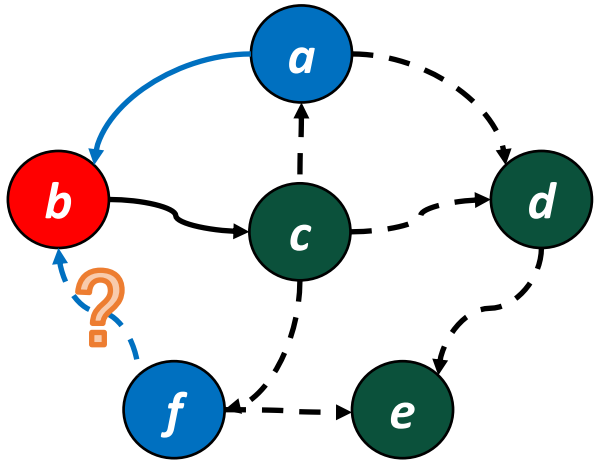
$S_b = \{a\}$

- $N_u \triangleq \mathcal{N}_{in}(\vec{G}_P, u) = u$'s in-neighbors in the predicted graph
- $S_u \triangleq u$'s *real* in-neighbors among N_u
- $T_u \triangleq u$'s in-neighbors among N_u which are *not yet wrong*
  - *either correct or unprobed*

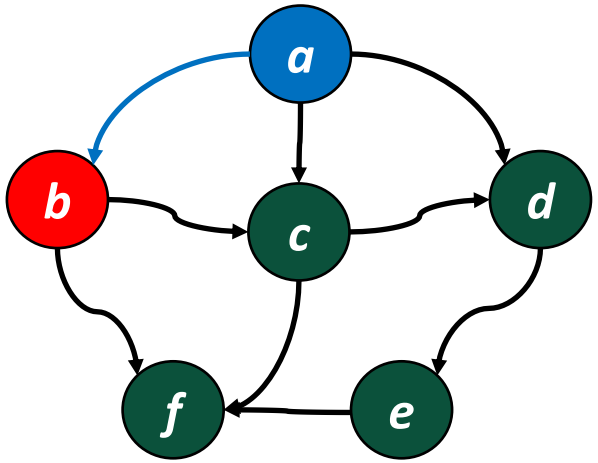
Some Notations



$N_b = \mathcal{N}_{in}(\vec{G}_P, b) = \{a, c, f\}$



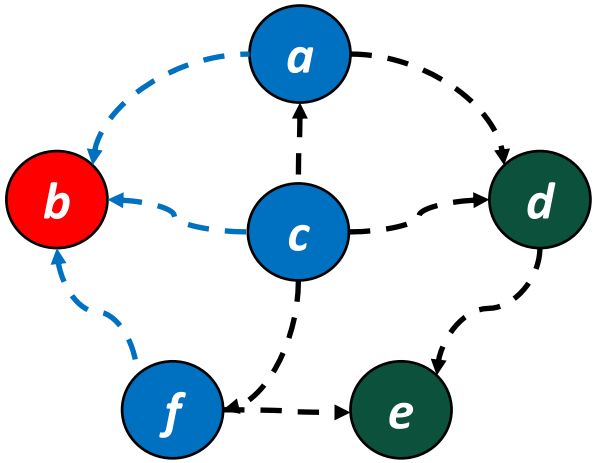
$T_b = \{a, f\}$



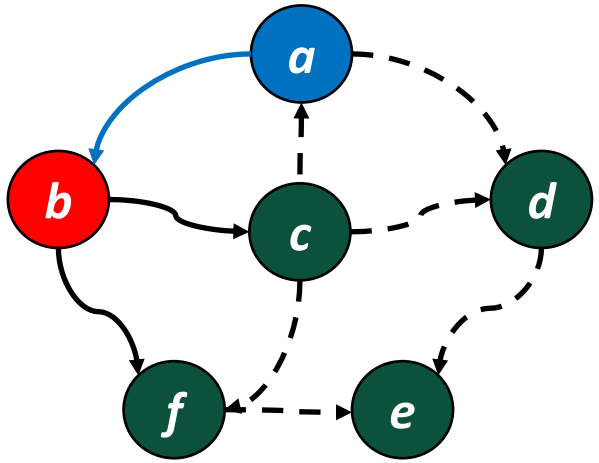
$S_b = \{a\}$

- $N_u \triangleq \mathcal{N}_{in}(\vec{G}_P, u) = u$'s in-neighbors in the predicted graph
- $S_u \triangleq u$'s *real* in-neighbors among N_u
- $T_u \triangleq u$'s in-neighbors among N_u which are *not yet wrong*
  - *either correct or unprobed*

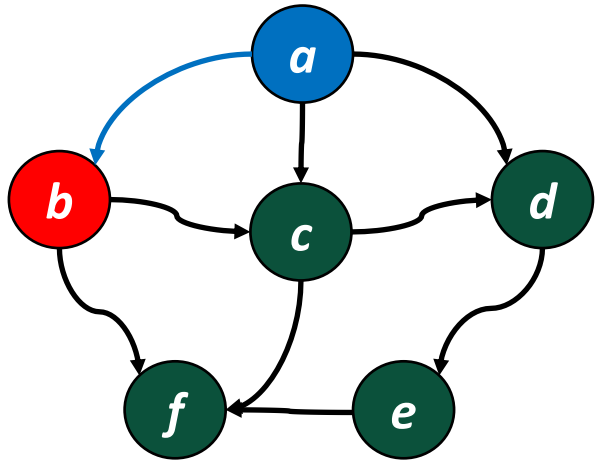
Some Notations



$N_b = \mathcal{N}_{in}(\vec{G}_P, b) = \{a, c, f\}$



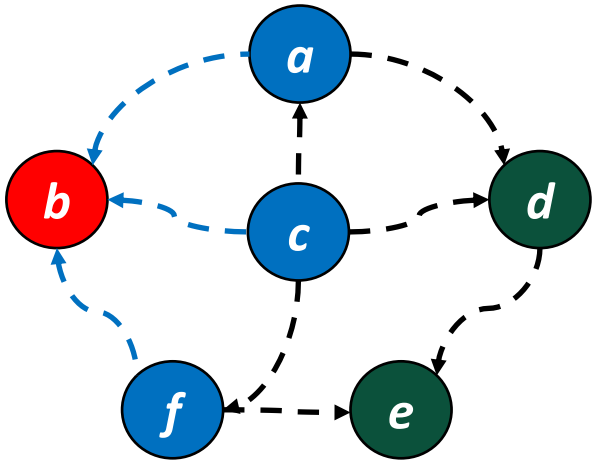
$T_b = \{a\}$



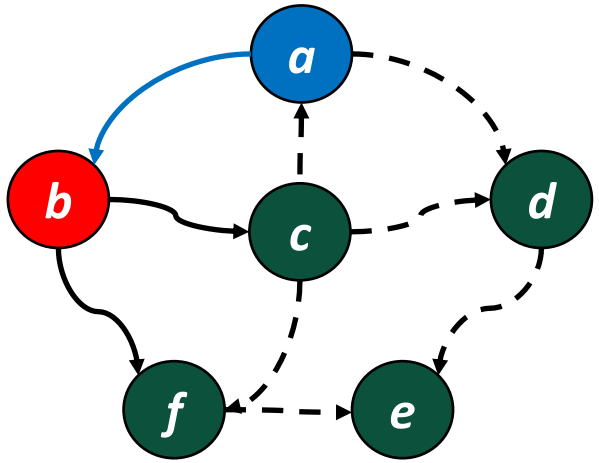
$S_b = \{a\}$

- $N_u \triangleq \mathcal{N}_{in}(\vec{G}_P, u) = u$'s in-neighbors in the predicted graph
- $S_u \triangleq u$'s *real* in-neighbors among N_u
- $T_u \triangleq u$'s in-neighbors among N_u which are *not yet wrong*
  - *either correct or unprobed*

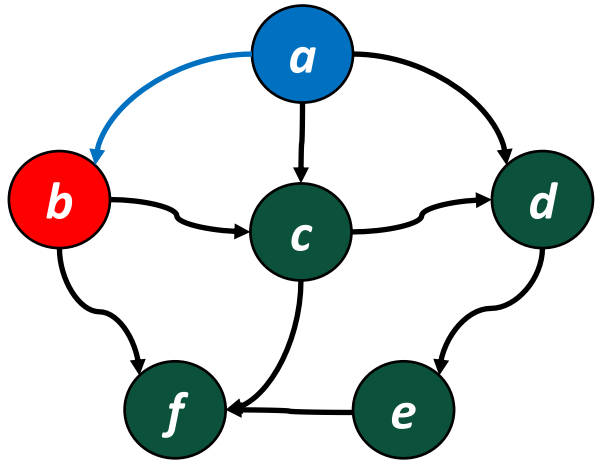
Overall Idea



$N_b = \mathcal{N}_{in}(\vec{G}_P, b) = \{a, c, f\}$



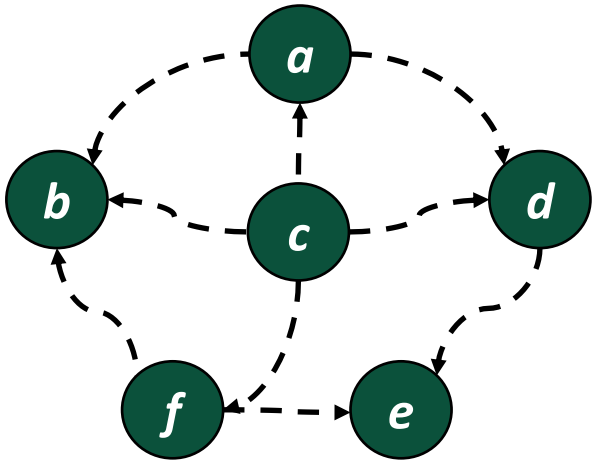
$T_b = \{a\}$



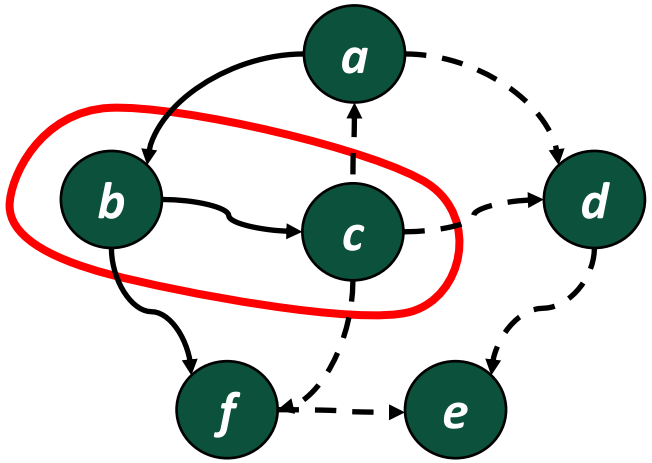
$S_b = \{a\}$

- $N_u \supseteq T_u \supseteq S_u$ all the time
  - $T_u = N_u$ initially
  - T_u shrinks to S_u finally
- We want to determine each S_u for all $u \in V$, using the smallest number of probes

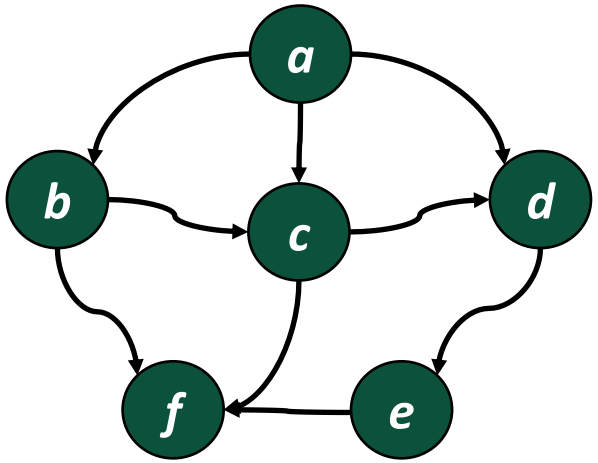
Overall Idea



$\vec{G}_P$



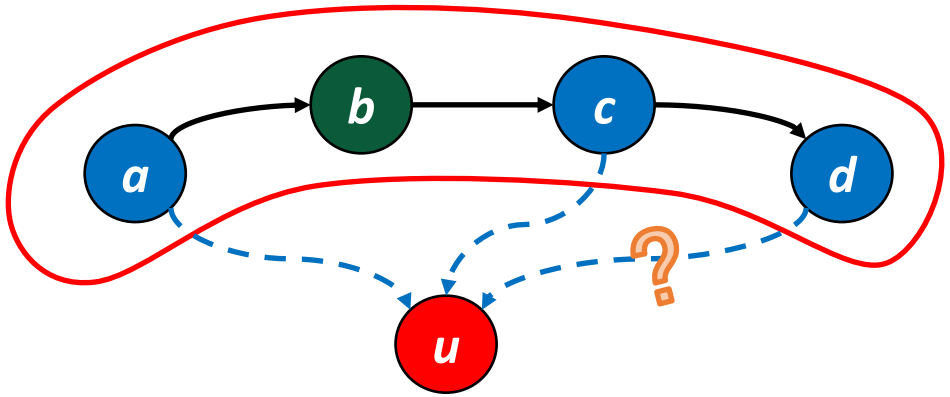
$A = \{b, c\}$
(an example)



$\vec{G}$

- We maintain a vertex set A s.t.
  - $\forall u \in A$, direction of edges between N_u and u are all known to us
- Initially $A = \emptyset$
- When $A = V$ we succeed
- $A \approx$ *currently sorted elements*

Ideal Vertex



: edges that have been probed

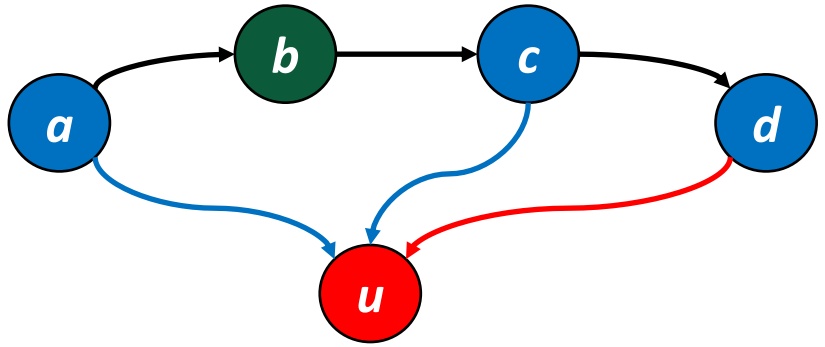
: edges in the prediction graph

When the total order of $T_u = \{a, c, d\}$ is known to us…

Conditions for u to be an ideal vertex (simplified):

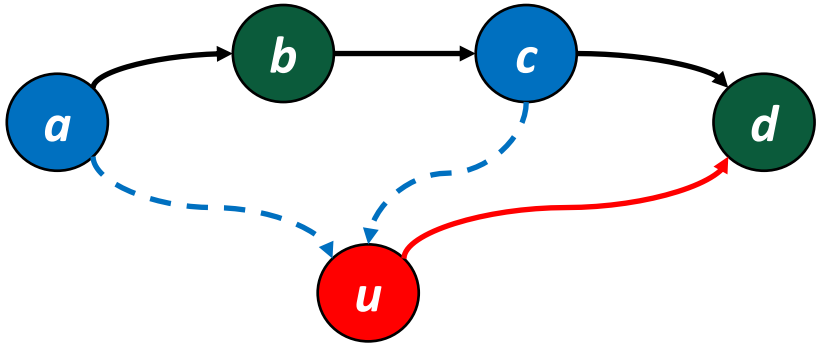
- $T_u \subseteq A$
- The total order of T_u is known to us

$d < e$



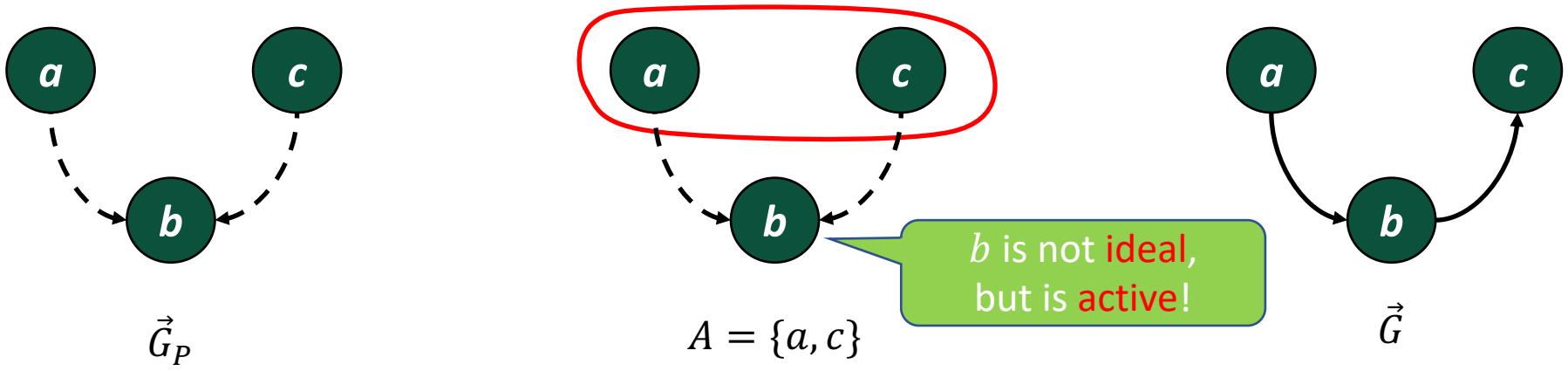
can conclude $a < u$ and $c < u$ by transitivity

$e < d$



find a mis-predicted edge (d, u), this probe is charged to the term w

Ideal Vertex → Active Vertex



There may be no ideal vertices at some time!

Conditions for u to be an ideal vertex (simplified):

- $T_u \subseteq A$
- The total order of T_u is known to us

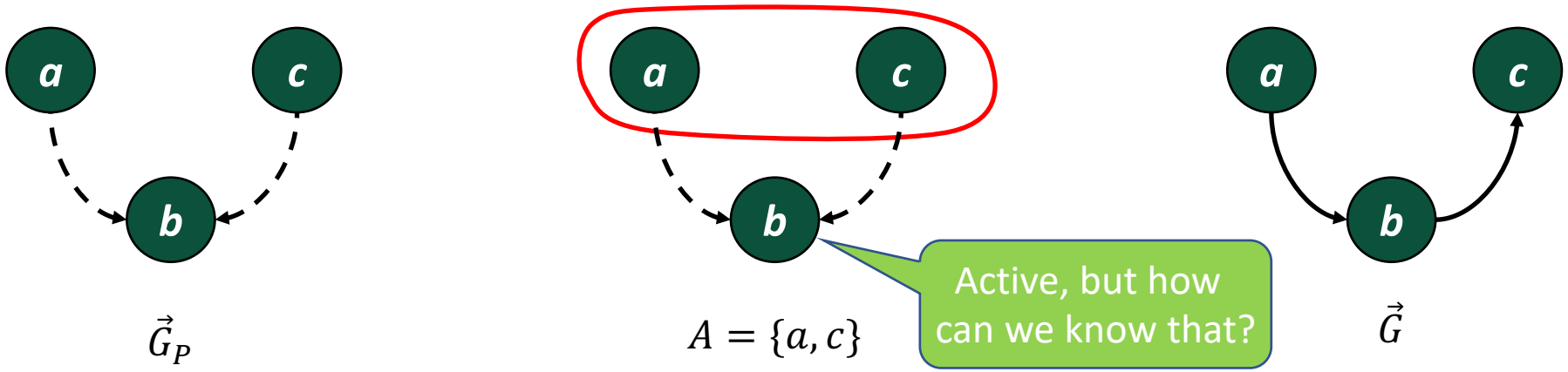
weaken →

Conditions for u to be an active vertex (simplified):

- $S_u \subseteq A$
- The total order of S_u is known to us

An ideal vertex must be an active one since $S_u \subseteq T_u$.
At any time there must be at least an active vertex.

Active Vertex

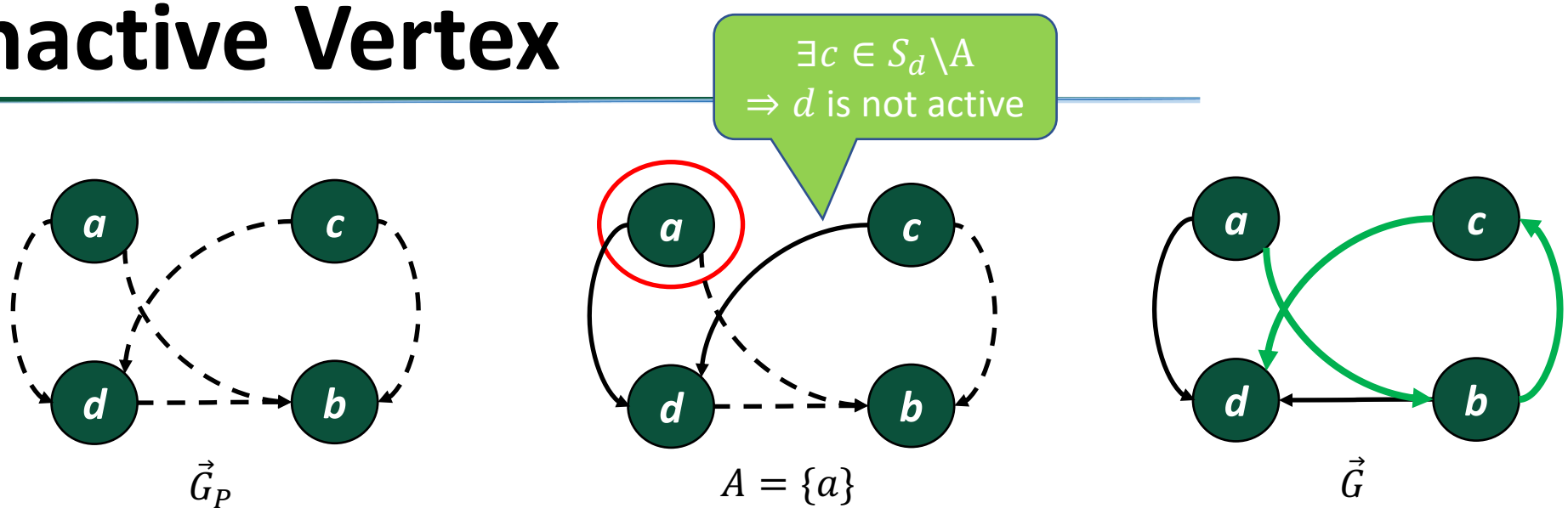


S_u is not known to us…

Therefore whether a vertex is active is not known to us…

However we can identify some vertices that are not active!

Inactive Vertex



Conditions for u to be an active vertex (simplified):

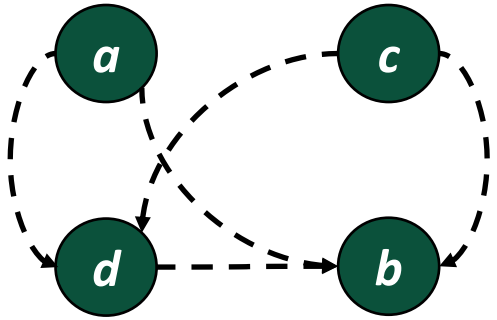
- $S_u \subseteq A$
- The total order of S_u is known to us

Conditions for u to be an inactive vertex (simplified):

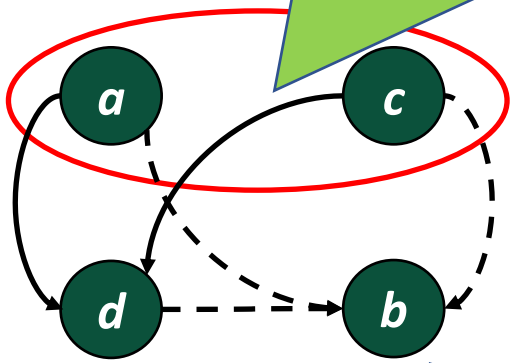
- $\exists v \in S_u \backslash A$

Inactive Vertex

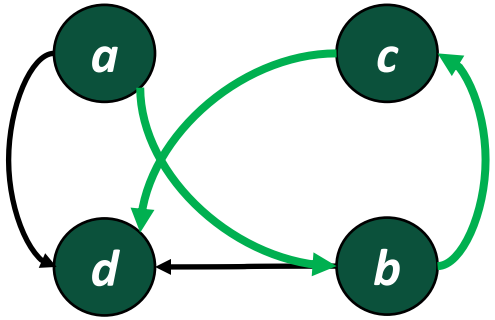
$\exists a, c \in S_d$ s.t. $(a \prec c) \wedge (c \prec a)$
$\Rightarrow d$ is not active



$\vec{G}_P$



$A = \{a, c\}$



$\vec{G}$

On looking for certificates for an active vertex (vertex b for example), we will correct its mis-predicted in-edges!

Conditions for u to be an active vertex (simplified):

- $S_u \subseteq A$
- The total order of S_u is known to us

Conditions for u to be an inactive vertex (simplified):

- $\exists v \in S_u \setminus A$
- (or) $\exists v_1, v_2 \in S_u$ s.t. $(v_1 \prec v_2) \wedge (v_2 \prec v_1)$

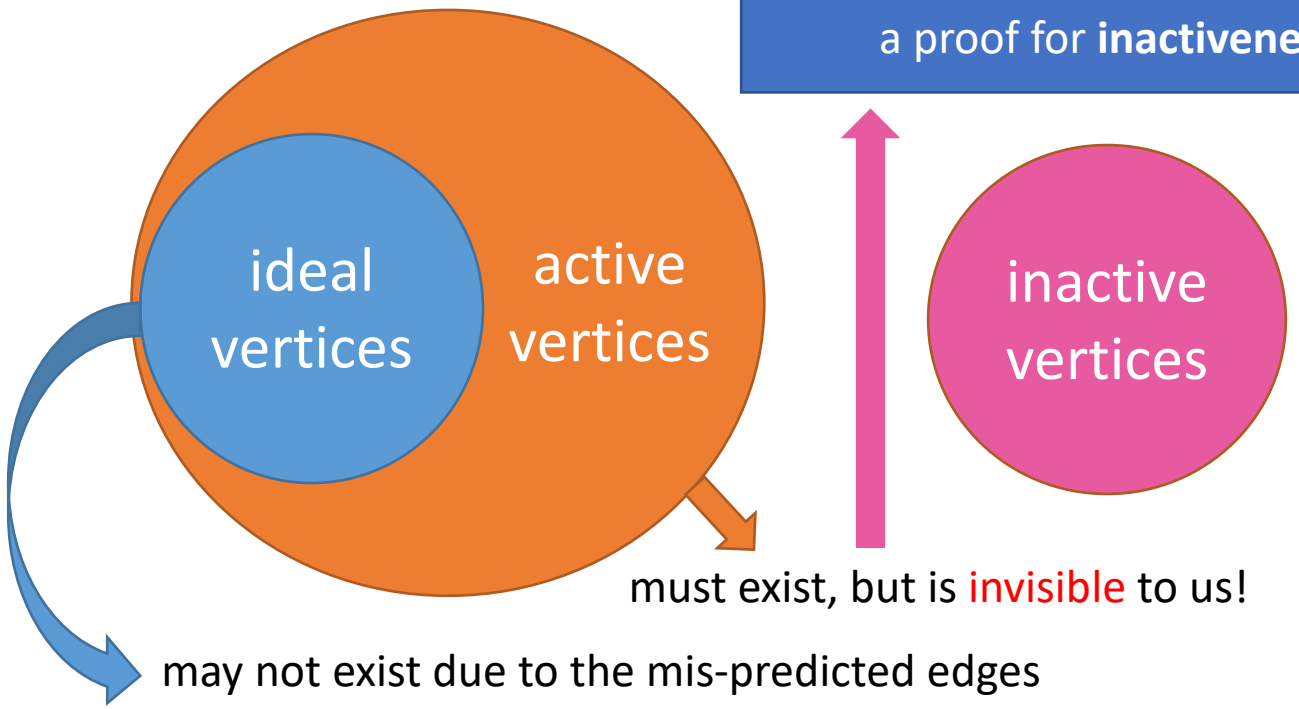
Such a v or such a pair of (v_1, v_2) is called a **certificate** for u,
which indicates that u is not active and is not the vertex we are looking for!

Re-searching for Certificates

- As the set A extends, the certificate for some vertex u may becomes invalid…
  - need to find a new certificate for u
  - in the worst case this may happen again and again, and we may make too many probes…
  - randomly pick certificates!

  - By carefully handling the correlation and analyzing the random process, we prove the number of probes needed is w.h.p. $O(n \log n + w)$.

Sketch



certificate:
a proof for **inactiveness**

must exist, but is invisible to us!

may not exist due to the mis-predicted edges

on looking for certificates for an active vertex u, its mis-predicted incoming edges can be found out, and it will finally fall into the ideal case

re-searching certificates again and again?
introduce randomness to help

need to carefully handle the correlation and analyze the process

re-searching for certificates, w.h.p.

mis-predicted edges

$$O(n \log n + w)$$

Open Questions

- Can this problem be solved in $O(n + w)$ probes or even better?

- Can the prediction-less version of generalized sorting problem be solved in $O(n^{1.5-\epsilon})$ probes?

The End

- Thanks!