Weighting Schemes and the NL vs UL Problem

Anant Dhayal Jayalal Sarma Saurabh Sawlani

IIT Madras, Chennai.

Indo-UK Workshop on Computational Complexity
IMSc, Chennai, Jan 9, 2015.

The Problem

$$\textsc{Reach} = \{(G, s, t) \mid \exists \text{ a directed path from } s \text{ to } t \text{ in } G\}$$

The Problem

$\textsc{Reach} = \{(G, s, t) \mid \exists$ a directed path from s to t in $G\}$

- Undirected Reachability is in $\mathtt{L}$ (Reingold 2004)

The Problem

$\textsc{Reach} = \{(G, s, t) \mid \exists$ a directed path from s to t in $G\}$

- Undirected Reachability is in $\mathtt{L}$ (Reingold 2004)
- Directed Reachability is $\mathtt{NL}$-complete. Even for layered DAGs.

The Problem

$$\textsc{Reach} = \{(G, s, t) \mid \exists \text{ a directed path from } s \text{ to } t \text{ in } G\}$$

- Undirected Reachability is in $\mathtt{L}$ (Reingold 2004)
- Directed Reachability is $\mathtt{NL}$-complete. Even for layered DAGs.
- $\mathtt{UL}$ - Problems solvable by logspace NTM having at most one accepting path for each input.

The Problem

$\textsc{Reach} = \{(G, s, t) \mid \exists$ a directed path from s to t in $G\}$

- Undirected Reachability is in $\mathtt{L}$ (Reingold 2004)
- Directed Reachability is $\mathtt{NL}$-complete. Even for layered DAGs.
- $\mathtt{UL}$ - Problems solvable by logspace NTM having at most one accepting path for each input.

Structural question : Can space bounded non-determinism be made unambiguous?

Weighting Schemes

- For each graph $G(V, E)$, weighing scheme defines a function $w : E \to \mathbb{N}$.
- Polynomially bounded and log-space computable.
- Weight of a path is the sum of the weights in edges in it.

Weighting Schemes

- For each graph $G(V, E)$, weighing scheme defines a function $w : E \to \mathbb{N}$.
- Polynomially bounded and log-space computable.
- Weight of a path is the sum of the weights in edges in it.
- A weighting scheme that maps $(w : E \to \mathbb{N})$ such that there is a unique minimum-weight path from s to any vertex v in the graph is called a MIN-UNIQUE weighting scheme.

Weighting Schemes

- For each graph $G(V, E)$, weighing scheme defines a function $w : E \to \mathbb{N}$.
- Polynomially bounded and log-space computable.
- Weight of a path is the sum of the weights in edges in it.
- A weighting scheme that maps $(w : E \to \mathbb{N})$ such that there is a unique minimum-weight path from s to any vertex v in the graph is called a $\textsc{Min-unique}$ weighting scheme.
- Testing reachability in a graph G augmented with a $\textsc{Min-unique}$ weighting scheme is in $\mathtt{UL}$ (Allender and Reinhardt - 2000).

Implications?

Implications?

- For showing $\mathtt{NL} = \mathtt{UL}$, it suffices to come up with a min-unique weighting scheme that is computable in log-space.

Implications?

- For showing $\mathtt{NL} = \mathtt{UL}$, it suffices to come up with a min-unique weighting scheme that is computable in log-space.
- With a polynomial sized advice, we can produce a set of n^2 graphs preserving reachability and with the guarantee that at least one of them is a min-unique graph. (Allender and Reinhardt, 2000)

Implications?

- For showing $\mathtt{NL} = \mathtt{UL}$, it suffices to come up with a min-unique weighting scheme that is computable in log-space.
- With a polynomial sized advice, we can produce a set of n^2 graphs preserving reachability and with the guarantee that at least one of them is a min-unique graph. (Allender and Reinhardt, 2000)

$$\mathtt{NL}/\mathsf{poly} = \mathtt{UL}/\mathsf{poly}$$

Implications?

- For showing $\mathtt{NL} = \mathtt{UL}$, it suffices to come up with a min-unique weighting scheme that is computable in log-space.
- With a polynomial sized advice, we can produce a set of n^2 graphs preserving reachability and with the guarantee that at least one of them is a min-unique graph. (Allender and Reinhardt, 2000)

$$\mathtt{NL}/\mathsf{poly} = \mathtt{UL}/\mathsf{poly}$$

- If deterministic linear space has functions that are not computable by circuits of size $2^{\epsilon n}$, then $\mathtt{NL} = \mathtt{UL}$.

Implications?

- For showing $\mathtt{NL} = \mathtt{UL}$, it suffices to come up with a min-unique weighting scheme that is computable in log-space.
- With a polynomial sized advice, we can produce a set of n^2 graphs preserving reachability and with the guarantee that at least one of them is a min-unique graph. (Allender and Reinhardt, 2000)

$$\mathtt{NL}/\mathrm{poly} = \mathtt{UL}/\mathrm{poly}$$

- If deterministic linear space has functions that are not computable by circuits of size $2^{\epsilon n}$, then $\mathtt{NL} = \mathtt{UL}$.
- A natural question : Can we design such weighing schemes for restricted classes of graphs?

Implications?

- For showing $\mathtt{NL} = \mathtt{UL}$, it suffices to come up with a min-unique weighting scheme that is computable in log-space.
- With a polynomial sized advice, we can produce a set of n^2 graphs preserving reachability and with the guarantee that at least one of them is a min-unique graph. (Allender and Reinhardt, 2000)

$$\mathtt{NL}/\mathsf{poly} = \mathtt{UL}/\mathsf{poly}$$

- If deterministic linear space has functions that are not computable by circuits of size $2^{\epsilon n}$, then $\mathtt{NL} = \mathtt{UL}$.
- A natural question : Can we design such weighing schemes for restricted classes of graphs?
- Yes, for planar grid graphs (Bourke, Tewari and Vinodchandran - 2007).

Implications?

- For showing $\mathtt{NL} = \mathtt{UL}$, it suffices to come up with a min-unique weighting scheme that is computable in log-space.
- With a polynomial sized advice, we can produce a set of n^2 graphs preserving reachability and with the guarantee that at least one of them is a min-unique graph. (Allender and Reinhardt, 2000)

$$\mathtt{NL}/\mathrm{poly} = \mathtt{UL}/\mathrm{poly}$$

- If deterministic linear space has functions that are not computable by circuits of size $2^{\epsilon n}$, then $\mathtt{NL} = \mathtt{UL}$.
- A natural question : Can we design such weighing schemes for restricted classes of graphs?
- Yes, for planar grid graphs (Bourke, Tewari and Vinodchandran - 2007).
- Planar reachability problem reduces (in log-space) to Grid Graph Reachability (Allender *et al* 2006). Thus, Planar Reach is in $\mathtt{UL}$.

Is Allender-Reinhardt result tight?

- $\mathtt{NL} = \mathtt{UL} \Longleftarrow$ L-computable $\textsc{Min-unique}$ Weighing schemes.
  Is the converse true?

Is Allender-Reinhardt result tight?

- $\mathtt{NL} = \mathtt{UL} \Longleftarrow$ L-computable $\textsc{Min-unique}$ Weighing schemes.
  Is the converse true?
- $\mathtt{NL} = \mathtt{UL} \iff \mathtt{UL}$-computable $\textsc{Min-unique}$ weighing schemes.(Pavan, Tewari, Vinodchandran, 2012).

Is Allender-Reinhardt result tight?

- $\mathtt{NL} = \mathtt{UL} \Longleftarrow$ L-computable $\textsc{Min-unique}$ Weighing schemes.
  Is the converse true?
- $\mathtt{NL} = \mathtt{UL} \iff \mathtt{UL}$-computable $\textsc{Min-unique}$ weighing schemes.(Pavan, Tewari, Vinodchandran, 2012).
- A weighting scheme that maps ($w : E \rightarrow \mathbb{N}$) such that there are at most n^c (c is known) minimum-weights path from s to any vertex v in the graph is called a $\textsc{Min-poly}$ weighting scheme.

Is Allender-Reinhardt result tight?

- $\mathtt{NL} = \mathtt{UL} \Longleftarrow$ L-computable $\textsc{Min-unique}$ Weighing schemes.
  Is the converse true?
- $\mathtt{NL} = \mathtt{UL} \iff$ UL-computable $\textsc{Min-unique}$ weighing schemes.(Pavan, Tewari, Vinodchandran, 2012).
- A weighting scheme that maps $(w : E \rightarrow \mathbb{N})$ such that there are at most n^c (c is known) minimum-weights path from s to any vertex v in the graph is called a $\textsc{Min-poly}$ weighting scheme.

Questions:

- Can $\textsc{Min-Poly}$ Weighted Reachability be done in $\mathtt{UL}$?
- Does this help in showing $\mathtt{NL} = \mathtt{UL}$?

Result 1 : Relaxing MIN-UNIQUE to MIN-POLY.

Theorem (1)

Testing reachability in a layered DAG G augmented with a MIN-POLY *weighting scheme is in* UL.

Result 1 : Relaxing $\textsc{Min-Unique}$ to $\textsc{Min-Poly}$.

Theorem (1)

Testing reachability in a layered DAG G augmented with a $\textsc{Min-poly}$ *weighting scheme is in* $\texttt{UL}$*.*

Comparison: ReachFewL = ReachUL (Garvin, Stolee, Tewari, Vinodchandran - 2011)

The above result talks about graphs with unique/polynomially many paths from s to any vertex v. Our result talks about graphs with unique/polynomially many minimum-weight paths from s to any vertex v. Total $s \rightsquigarrow v$ paths could be exponential in number.

Result 2 : MAX-UNIQUE Weighting Schemes

- A weighting scheme that maps ($w : E \to \mathbb{N}$) such that there is a unique maximum-weight path from s to any vertex v in the graph is called a MAX-UNIQUE weighting scheme.

Result 2 : Max-Unique Weighting Schemes

- A weighting scheme that maps ($w : E \to \mathbb{N}$) such that there is a unique maximum-weight path from s to any vertex v in the graph is called a Max-unique weighting scheme.
- Studied in a related context :
  - LongPath $= \{(G, s, t, j) \mid$ a simple directed path from s to t in G of length at least $j\}$.

Result 2 : MAX-UNIQUE Weighting Schemes

- A weighting scheme that maps ($w : E \to \mathbb{N}$) such that there is a unique maximum-weight path from s to any vertex v in the graph is called a MAX-UNIQUE weighting scheme.
- Studied in a related context :
  - LONGPATH $= \{(G, s, t, j) \mid$ a simple directed path from s to t in G of length at least $j\}$.
  - Testing LONGPATH in a DAG G with unique source s augmented with a MAX-UNIQUE weighting scheme is in UL (Limaye, Mahajan, and Nimbhorkar - 2009).

Result 2 : Max-Unique Weighting Schemes

- A weighting scheme that maps ($w : E \rightarrow \mathbb{N}$) such that there is a unique maximum-weight path from s to any vertex v in the graph is called a Max-unique weighting scheme.
- Studied in a related context :
  - $\textsc{LongPath} = \{(G, s, t, j) \mid$ a simple directed path from s to t in G of length at least $j\}$.
  - Testing LongPath in a DAG G with unique source s augmented with a Max-unique weighting scheme is in UL (Limaye, Mahajan, and Nimbhorkar - 2009).
  - They use this, along with the weighing schemes for planar grid graphs, to show that the longest path in planar graphs is in UL.

Result 2 : MAX-UNIQUE Weighting Schemes

- A weighting scheme that maps ($w : E \to \mathbb{N}$) such that there is a unique maximum-weight path from s to any vertex v in the graph is called a MAX-UNIQUE weighting scheme.
- Studied in a related context :
  - LONGPATH $= \{(G, s, t, j) \mid$ a simple directed path from s to t in G of length at least $j\}$.
  - Testing LONGPATH in a DAG G with unique source s augmented with a MAX-UNIQUE weighting scheme is in UL (Limaye, Mahajan, and Nimbhorkar - 2009).
  - They use this, along with the weighing schemes for planar grid graphs, to show that the longest path in planar graphs is in UL.
- **Lemma:** REACH on Layered DAGs logspace reduces to LONGPATH on single source Layered DAGs. In addition, it preserves the max-unique and max-poly property of the graph.
- MAX-UNIQUE weighted REACH is in UL.

Result 3: Max-Poly Weighting Schemes

- A weighting scheme that maps ($w : E \rightarrow \mathbb{N}$) such that there are at most n^c (c is known) maximum-weight paths from s to any vertex v in the graph is called a Max-poly weighting scheme.

Theorem (2)

Testing Reachability in a layered DAG G augmented with a Max-poly *weighting scheme can be done by a non-deterministic log-space algorithm unambiguously and hence is in the complexity class* UL.

The final algorithm is designed for Long Path problem.

Consequences

The following statements are equivalent :

- $\mathtt{NL} = \mathtt{UL}$

Consequences

The following statements are equivalent :

- $\mathtt{NL} = \mathtt{UL}$
- There is a polynomially bounded $\mathtt{UL}$-computable $\textsc{Min-unique}$ weighting scheme for any layered DAG. (Pavan, Tewari, Vinodchandran - 2012).

Consequences

The following statements are equivalent :

- $\mathtt{NL} = \mathtt{UL}$
- There is a polynomially bounded $\mathtt{UL}$-computable $\textsc{Min-unique}$ weighting scheme for any layered DAG. (Pavan, Tewari, Vinodchandran - 2012).
- There is a polynomially bounded $\mathtt{UL}$-computable $\textsc{Max-unique}$ weighting scheme for any layered DAG.
- There is a polynomially bounded $\mathtt{UL}$-computable $\textsc{Min-poly}$ weighting scheme for any layered DAG.
- There is a polynomially bounded $\mathtt{UL}$-computable $\textsc{Max-poly}$ weighting scheme for any layered DAG.

The rest of the talk ...

We will present :

- Outline Allender-Reinhardt Algorithm.
- Modification to get a special NL algorithm for MIN-POLY case.
- UL Algorithm for MIN-POLY case and proof sketch.
- Reduction from REACH to LONGPATH.

We will not present :

- UL algorithm for MAX-POLY case.

Notations

- Replace weights with paths of the corresponding length. Now, shortest paths from s to any vertex v in G is unique. All edges go from a lower numbered vertex to a higher numbered vertex.
- $d(v)$: Length of the shortest $s \rightsquigarrow v$ path.

Notations

- Replace weights with paths of the corresponding length. Now, shortest paths from s to any vertex v in G is unique. All edges go from a lower numbered vertex to a higher numbered vertex.

- $d(v)$: Length of the shortest $s \rightsquigarrow v$ path.
- c_k: Number of vertices within level-k.
- Σ_k: Sum of $d(v)$s of vertices within level-k.

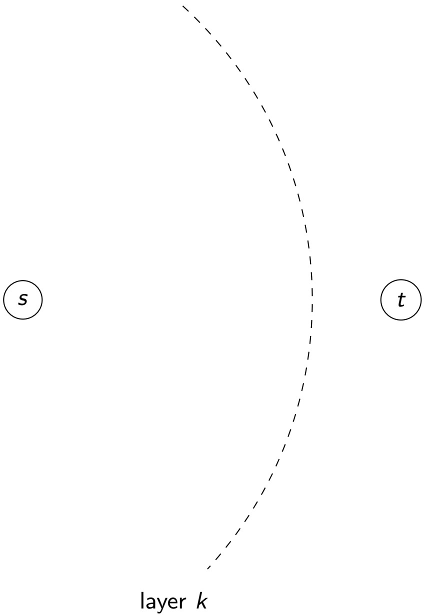
Idea (Allender, Reinheardt - 2000) : Inductively for $k = 0$ to n

- A $\mathtt{UL}$ algorithm to check if $d(v) \leq k$ assuming correct values of c_k, Σ_k are available.
- Use this to compute c_{k+1}, Σ_{k+1} from c_k and Σ_k

Routine to check if $d(v) \leq k$ unambiguously (**Min-unique case**)

[Reinhardt and Allender 2000]

$\boxed{\text{Values of } c_k \text{ and } \Sigma_k \text{ are known}}$



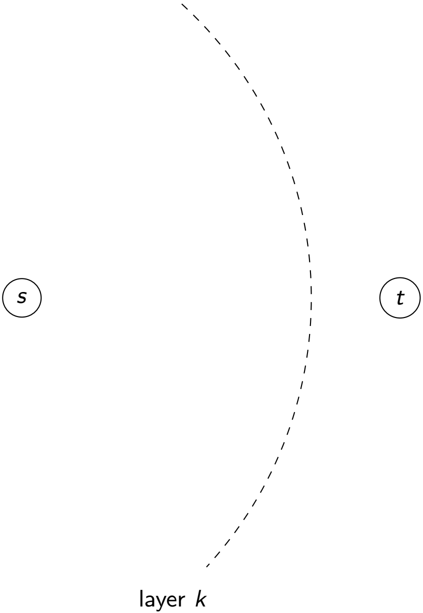
Routine to check if $d(v) \leq k$ unambiguously (**Min-unique case**)

[Reinhardt and Allender 2000]

$\boxed{\text{Values of } c_k \text{ and } \Sigma_k \text{ are known}}$

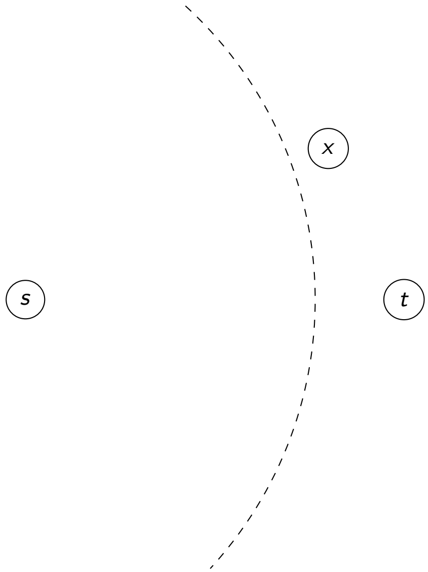
For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$



Routine to check if $d(v) \leq k$ unambiguously (**Min-unique case**)

[Reinhardt and Allender 2000]



layer k

$\boxed{\text{Values of } c_k \text{ and } \Sigma_k \text{ are known}}$

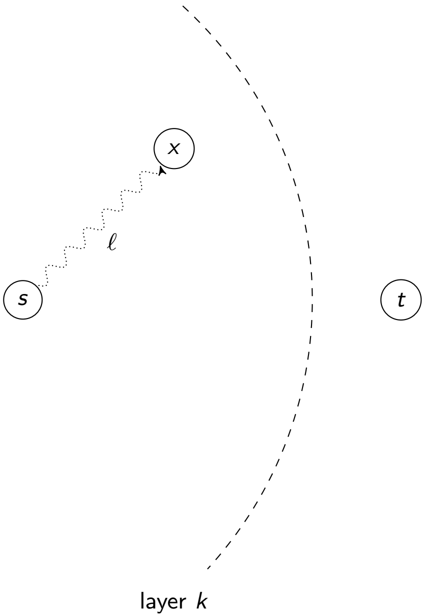
For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

If the guess is NO, move to the next x

Routine to check if $d(v) \leq k$ unambiguously (**Min-unique case**)

[Reinhardt and Allender 2000]



$\boxed{\text{Values of } c_k \text{ and } \Sigma_k \text{ are known}}$

For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

If the guess is YES,

$\rightarrow$ Guess an integer $1 \leq \ell \leq k$,

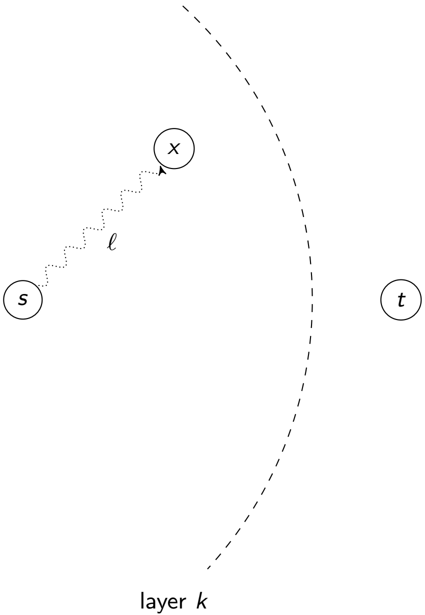
and an $s \rightsquigarrow x$ path of length ℓ

$\rightarrow$ If path is found,

$count := count + 1, sum := sum + \ell$

Routine to check if $d(v) \leq k$ unambiguously (**Min-unique case**)

[Reinhardt and Allender 2000]



$\boxed{\text{Values of } c_k \text{ and } \Sigma_k \text{ are known}}$

For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

If the guess is YES,

$\rightarrow$ Guess an integer $1 \leq \ell \leq k$,

and an $s \rightsquigarrow x$ path of length ℓ

$\rightarrow$ If path is found,

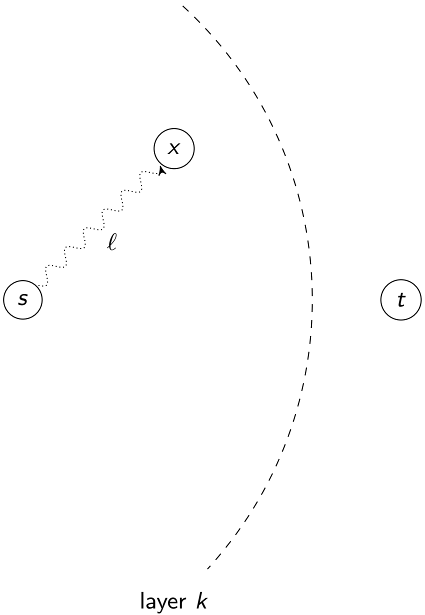
$count := count + 1, sum := sum + \ell$

Final Check:

$count = c_k$ and $sum = \Sigma_k$

Routine to check if $d(v) \leq k$ unambiguously (**Min-unique case**)

[Reinhardt and Allender 2000]



Values of c_k and Σ_k are known

For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

If the guess is YES,

$\rightarrow$ Guess an integer $1 \leq \ell \leq k$,

and an $s \rightsquigarrow x$ path of length ℓ

$\rightarrow$ If path is found,

$count := count + 1, sum := sum + \ell$

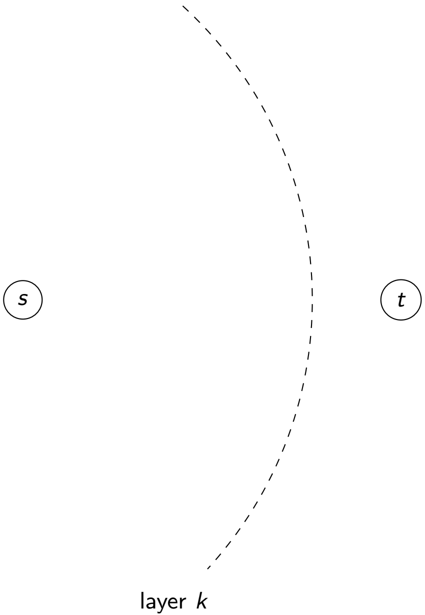
Final Check:

$count = c_k$ and $sum = \Sigma_k$

Return YES iff v was guessed within level k

Algorithm to calculate c_{k+1} and Σ_{k+1} (**Min-unique case**)

[Reinhardt and Allender 2000]

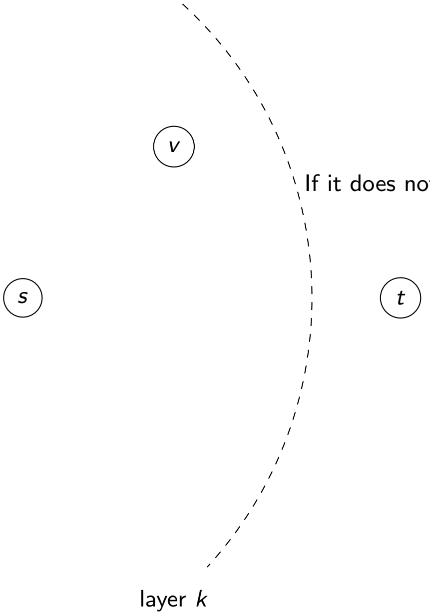


$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}) = (c_k, \Sigma_k)}$

Call the routine to check if $d(v) \leq k$

Algorithm to calculate c_{k+1} and Σ_{k+1} (**Min-unique case**)

[Reinhardt and Allender 2000]



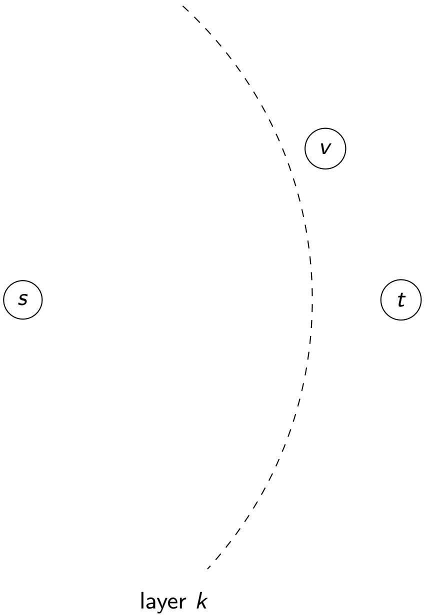
$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}) = (c_k, \Sigma_k)}$

Call the routine to check if $d(v) \leq k$

If it does not return 0, move on to the next choice of v

Algorithm to calculate c_{k+1} and Σ_{k+1} (**Min-unique case**)

[Reinhardt and Allender 2000]



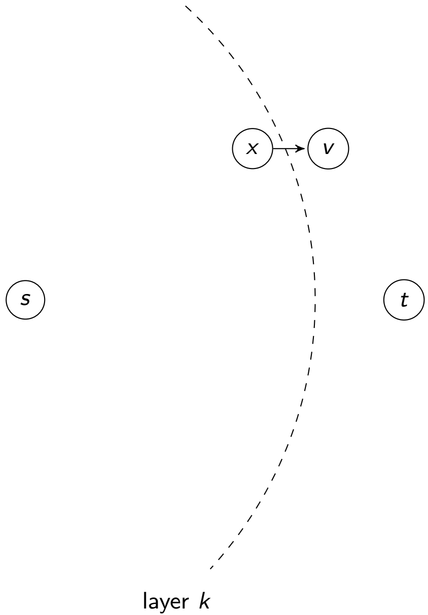
$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}) = (c_k, \Sigma_k)}$

Call the routine to check if $d(v) \leq k$

If it returns 0,

Algorithm to calculate c_{k+1} and Σ_{k+1} (**Min-unique case**)

[Reinhardt and Allender 2000]



$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}) = (c_k, \Sigma_k)}$

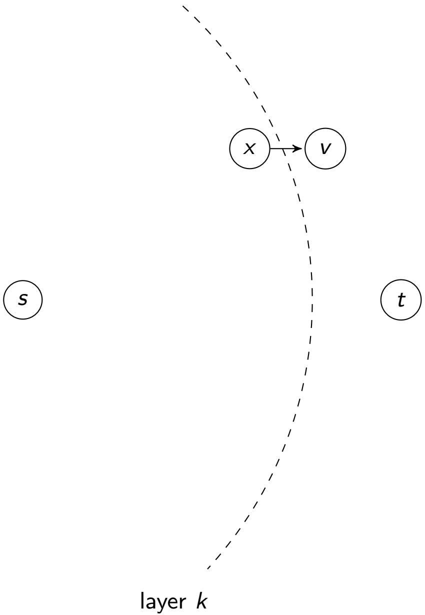
Call the routine to check if $d(v) \leq k$

If it returns 0,

$\forall x \mid (x, v) \in E$, Check $d(x) \leq k$

Algorithm to calculate c_{k+1} and Σ_{k+1} (**Min-unique case**)

[Reinhardt and Allender 2000]



$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}) = (c_k, \Sigma_k)}$

Call the routine to check if $d(v) \leq k$

If it returns 0,

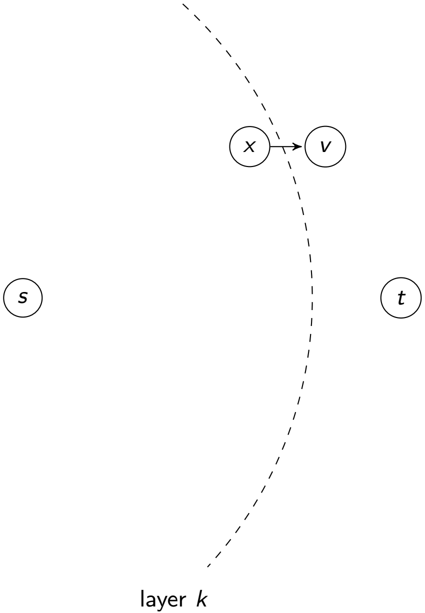
$\forall x \mid (x, v) \in E$, Check $d(x) \leq k$

If all checks output 0

$\rightarrow$ Move to the next v

Algorithm to calculate c_{k+1} and Σ_{k+1} (**Min-unique case**)

[Reinhardt and Allender 2000]



$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}) = (c_k, \Sigma_k)}$

Call the routine to check if $d(v) \leq k$

If it returns 0,

$\forall x \mid (x, v) \in E$, Check $d(x) \leq k$

If all checks output 0

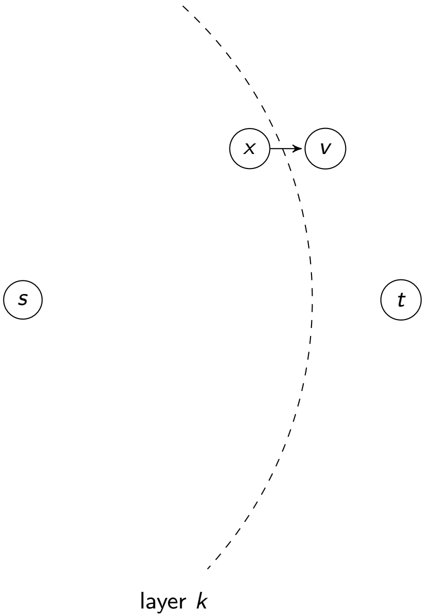
$\rightarrow$ Move to the next v

If $\exists\ x' \neq x$ with $d(x') \leq k$ and $(x', v) \in E$

$\boxed{\rightarrow \text{Not Min-unique}}$

Algorithm to calculate c_{k+1} and Σ_{k+1} (**Min-unique case**)

[Reinhardt and Allender 2000]



$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}) = (c_k, \Sigma_k)}$

Call the routine to check if $d(v) \leq k$

If it returns 0,

$\forall x \mid (x, v) \in E$, Check $d(x) \leq k$

If all checks output 0

$\rightarrow$ Move to the next v

If $\exists\ x' \neq x$ with $d(x') \leq k$ and $(x', v) \in E$

$\boxed{\rightarrow \text{Not Min-unique}}$

Else

$$c_{k+1} := c_{k+1} + 1$$
$$\Sigma_{k+1} := \Sigma_{k+1} + k + 1$$

In the MIN-POLY case :

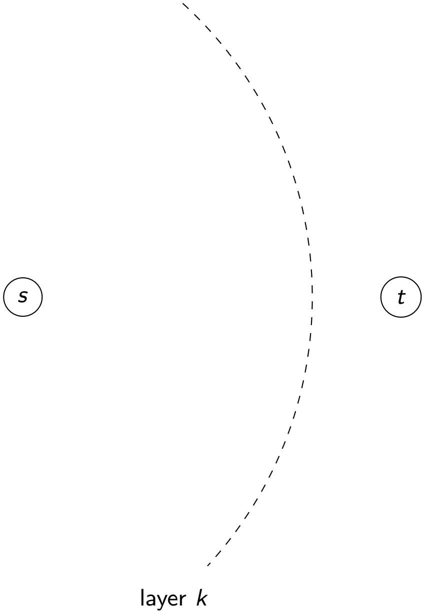
- Mindblock : $d(v) \leq k$ test is not Unambiguous anymore.
- Solution : Guess the paths too. Keep track of total number of paths that we have seen to v.

In the MIN-POLY case :

- Mindblock : $d(v) \leq k$ test is not Unambiguous anymore.
- Solution : Guess the paths too. Keep track of total number of paths that we have seen to v.
- $p(v)$: Number of shortest $s \rightsquigarrow v$ paths.
- p_k: Sum of $p(v)$s of vertices within level-k.

Routine to check if $d(v) \leq k$ unambiguously (**Min-poly case**)

$\boxed{\text{Values of } c_k, \Sigma_k \text{ and } p_k \text{ are known}}$

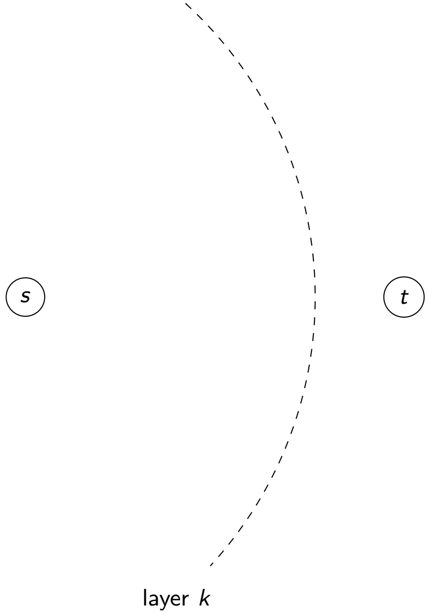


Routine to check if $d(v) \leq k$ unambiguously (**Min-poly case**)

$\boxed{\text{Values of } c_k, \Sigma_k \text{ and } p_k \text{ are known}}$

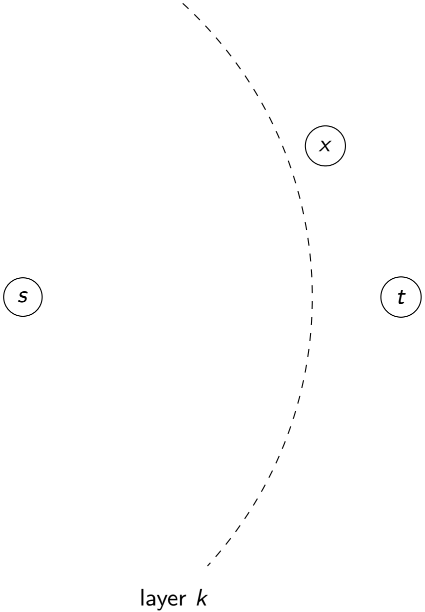
For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$



Routine to check if $d(v) \leq k$ unambiguously (**Min-poly case**)

Values of c_k, Σ_k and p_k are known

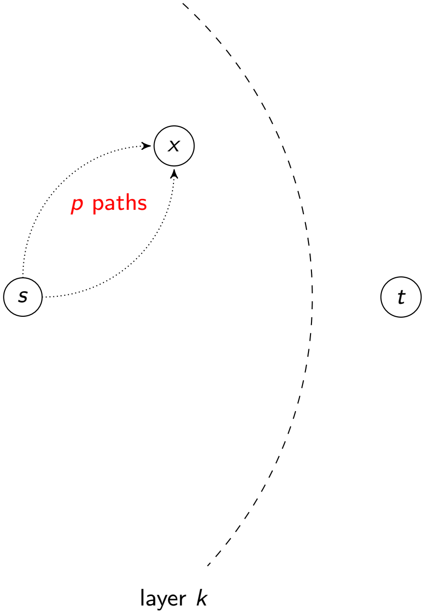


For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

If the guess is NO, move to the next x

Routine to check if $d(v) \leq k$ unambiguously (**Min-poly case**)



$\boxed{\text{Values of } c_k, \Sigma_k \text{ and } p_k \text{ are known}}$

For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

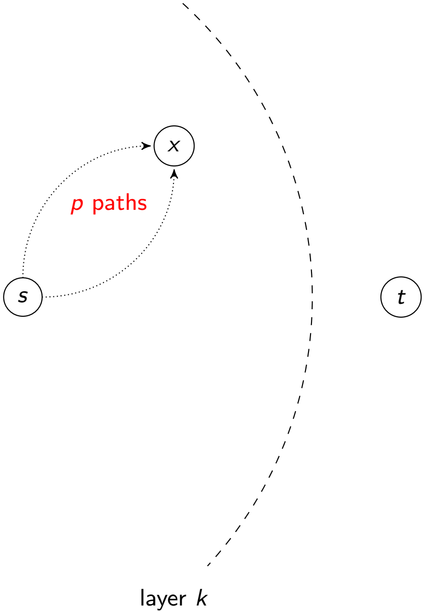
If the guess is YES,

$\rightarrow$ Guess an integer $1 \leq \ell \leq k$,

and an integer $1 \leq p \leq n^c$

$\rightarrow$ Guess p $s \rightsquigarrow x$ paths of length ℓ

Routine to check if $d(v) \leq k$ unambiguously (**Min-poly case**)



$\boxed{\text{Values of } c_k, \Sigma_k \text{ and } p_k \text{ are known}}$

For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

If the guess is YES,

$\rightarrow$ Guess an integer $1 \leq \ell \leq k$,

and an integer $1 \leq p \leq n^c$

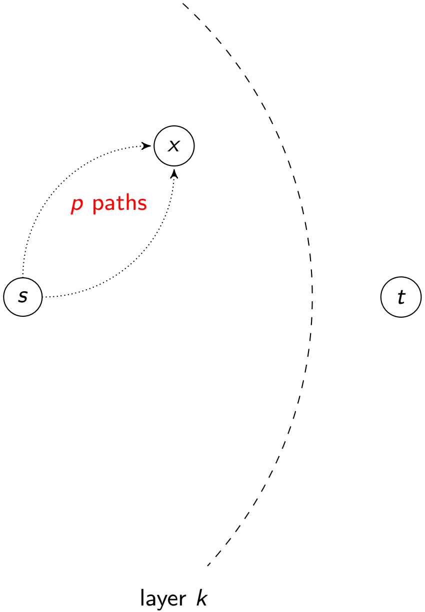
$\rightarrow$ Guess p $s \rightsquigarrow x$ paths of length ℓ

$\rightarrow$ If paths are found and in order,

$count := count + 1, sum := sum + \ell$

$paths := paths + p$

Routine to check if $d(v) \leq k$ unambiguously (**Min-poly case**)



$\boxed{\text{Values of } c_k, \Sigma_k \text{ and } p_k \text{ are known}}$

For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

If the guess is YES,

$\rightarrow$ Guess an integer $1 \leq \ell \leq k$,

and an integer $1 \leq p \leq n^c$

$\rightarrow$ Guess p $s \rightsquigarrow x$ paths of length ℓ

$\rightarrow$ If paths are found and in order,

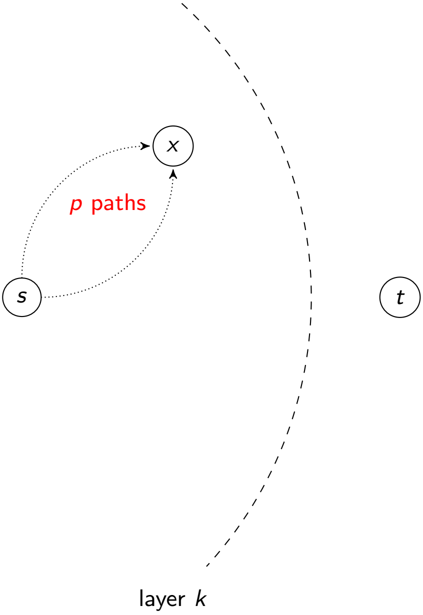
$count := count + 1, sum := sum + \ell$

$paths := paths + p$

Final Check:

$count = c_k$, $sum = \Sigma_k$ and $paths = p_k$

Routine to check if $d(v) \leq k$ unambiguously (**Min-poly case**)



$\boxed{\text{Values of } c_k, \Sigma_k \text{ and } p_k \text{ are known}}$

For each $x \in V$

$\rightarrow$ Non-deterministically guess if $d(x) \leq k$

If the guess is YES,

$\rightarrow$ Guess an integer $1 \leq \ell \leq k$,

and an integer $1 \leq p \leq n^c$

$\rightarrow$ Guess p $s \rightsquigarrow x$ paths of length ℓ

$\rightarrow$ If paths are found and in order,

$count := count + 1, sum := sum + \ell$

$paths := paths + p$

Final Check:

$count = c_k$, $sum = \Sigma_k$ and $paths = p_k$

Return $p(v)$ iff v was guessed within level k

Guessing n^c paths in log-space

- In a layered DAG, a path can be represented by a subset of vertices.

Guessing n^c paths in log-space

- In a layered DAG, a path can be represented by a subset of vertices.
- For a path $p : (s = v_1, v_i, \ldots, v_j)$,

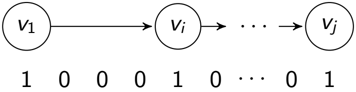
Guessing n^c paths in log-space

- In a layered DAG, a path can be represented by a subset of vertices.
- For a path $p : (s = v_1, v_i, \ldots, v_j)$,

$$v_1 \longrightarrow v_i \to \cdots \to v_j$$

Guessing n^c paths in log-space

- In a layered DAG, a path can be represented by a subset of vertices.
- For a path $p : (s = v_1, v_i, \ldots, v_j)$,



Guessing n^c paths in log-space

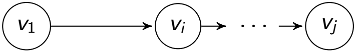
- In a layered DAG, a path can be represented by a subset of vertices.
- For a path $p : (s = v_1, v_i, \ldots, v_j)$,

$$v_1 \longrightarrow v_i \longrightarrow \cdots \longrightarrow v_j$$

$$1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad \cdots \quad 0 \quad 1 \qquad \boxed{\rightarrow \phi(p) = 2^1 + 2^i + \ldots + 2^j}$$

Guessing n^c paths in log-space

- In a layered DAG, a path can be represented by a subset of vertices.
- For a path $p : (s = v_1, v_i, \ldots, v_j)$,



1 0 0 0 1 0 $\cdots$ 0 1

$$\boxed{\rightarrow \phi(p) = 2^1 + 2^i + \ldots + 2^j}$$

- $\phi(p)$ is unique for each path.
- However it cannot be represented by logarithmic number of bits.
- So, we need a polynomially bounded m such that $\phi_m(p) = \phi(p) \mod m$ also remains unique.

Guessing n^c paths in log-space

- In a layered DAG, a path can be represented by a subset of vertices.
- For a path $p : (s = v_1, v_i, \ldots, v_j)$,

$$v_1 \longrightarrow v_i \rightarrow \cdots \rightarrow v_j$$

$$1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \cdots 0 \quad 1 \qquad \boxed{\rightarrow \phi(p) = 2^1 + 2^i + \ldots + 2^j}$$

- $\phi(p)$ is unique for each path.
- However it cannot be represented by logarithmic number of bits.
- So, we need a polynomially bounded m such that $\phi_m(p) = \phi(p) \mod m$ also remains unique.

Theorem (Fredman, Komlos, Szemeredi - 1984)

For every constant c there is a constant c' so that for every set S of n-bit integers with $|S| \leq n^c$ there is a $c' \log n$-bit prime number m so that for all $x, y \in S$, $x \neq y \implies x \not\equiv y \mod m$.

Guessing n^c paths in log-space

- In a layered DAG, a path can be represented by a subset of vertices.
- For a path $p : (s = v_1, v_i, \ldots, v_j)$,

$$v_1 \longrightarrow v_i \to \cdots \to v_j$$

$$1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \ \cdots \ 0 \quad 1 \qquad \boxed{\to \phi(p) = 2^1 + 2^i + \ldots + 2^j}$$

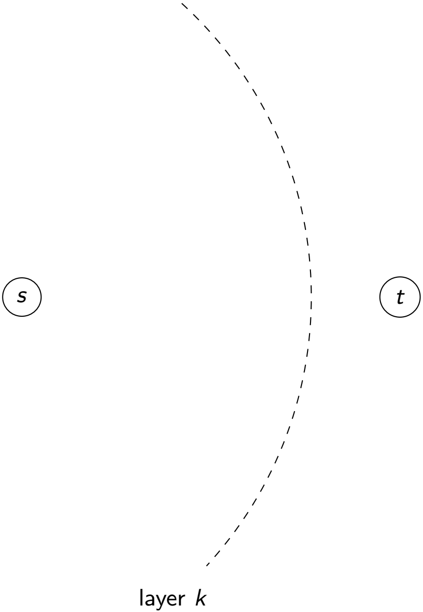
- $\phi(p)$ is unique for each path.
- However it cannot be represented by logarithmic number of bits.
- So, we need a polynomially bounded m such that $\phi_m(p) = \phi(p) \mod m$ also remains unique.

Theorem (Fredman, Komlos, Szemeredi - 1984)

For every constant c there is a constant c' so that for every set S of n-bit integers with $|S| \leq n^c$ there is a $c' \log n$-bit prime number m so that for all $x, y \in S$, $x \neq y \implies x \not\equiv y \mod m$.

- ReachFewL = ReachUL, Garvin, Stolee, Tewari, Vinodchandran [2011] - used a similar ϕ to give weights to edges.

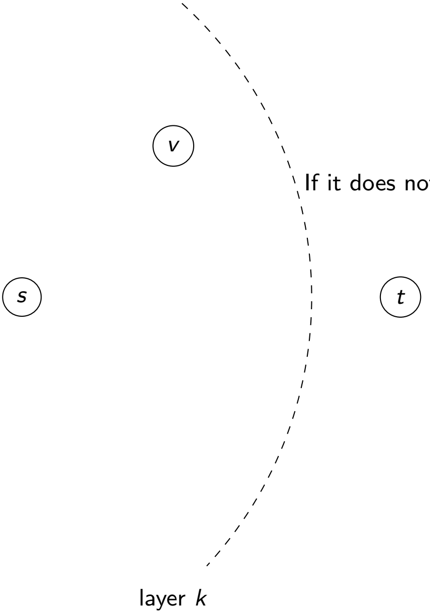
Algorithm to calculate c_{k+1},Σ_{k+1} and p_{k+1} (**Min-poly case**)



$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}, p_{k+1}) = (c_k, \Sigma_k, p_k)}$

Call the routine to check if $d(v) \leq k$

Algorithm to calculate c_{k+1},Σ_{k+1} and p_{k+1} (**Min-poly case**)

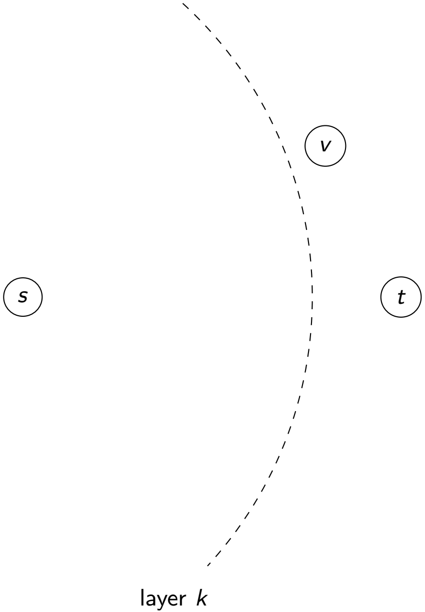


$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}, p_{k+1}) = (c_k, \Sigma_k, p_k)}$

Call the routine to check if $d(v) \leq k$

If it does not return 0, move on to the next choice of v

Algorithm to calculate c_{k+1},Σ_{k+1} and p_{k+1} (**Min-poly case**)

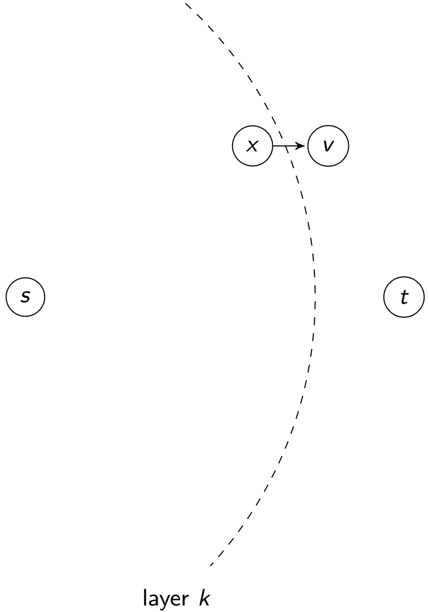


$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}, p_{k+1}) = (c_k, \Sigma_k, p_k)}$

Call the routine to check if $d(v) \leq k$

If it returns 0,

Algorithm to calculate c_{k+1},Σ_{k+1} and p_{k+1} (**Min-poly case**)



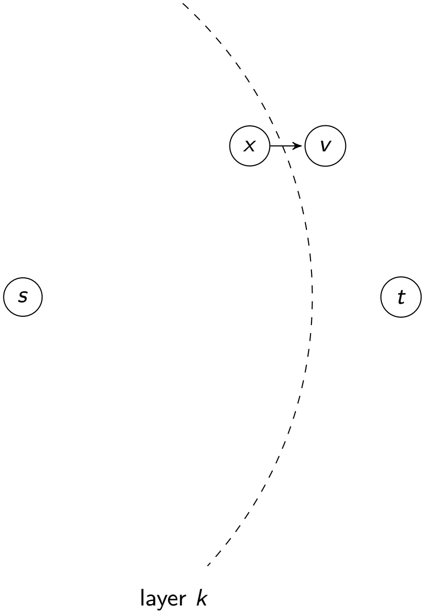
$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}, p_{k+1}) = (c_k, \Sigma_k, p_k)}$

Call the routine to check if $d(v) \leq k$

If it returns 0,

$\forall x \mid (x, v) \in E$, Call $d(x) \leq k$

Algorithm to calculate c_{k+1},Σ_{k+1} and p_{k+1} (**Min-poly case**)



$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}, p_{k+1}) = (c_k, \Sigma_k, p_k)}$

Call the routine to check if $d(v) \leq k$

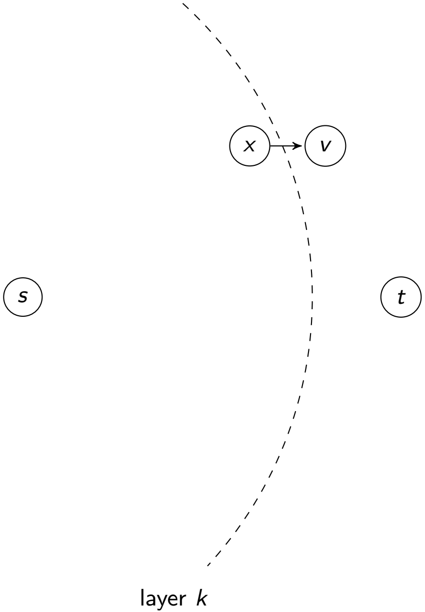
If it returns 0,

$\forall x \mid (x, v) \in E$, Call $d(x) \leq k$

If all calls output 0

$\rightarrow$ Move to the next v

Algorithm to calculate c_{k+1},Σ_{k+1} and p_{k+1} (**Min-poly case**)



$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}, p_{k+1}) = (c_k, \Sigma_k, p_k)}$

Call the routine to check if $d(v) \leq k$

If it returns 0,

$\forall x \mid (x, v) \in E$, Call $d(x) \leq k$

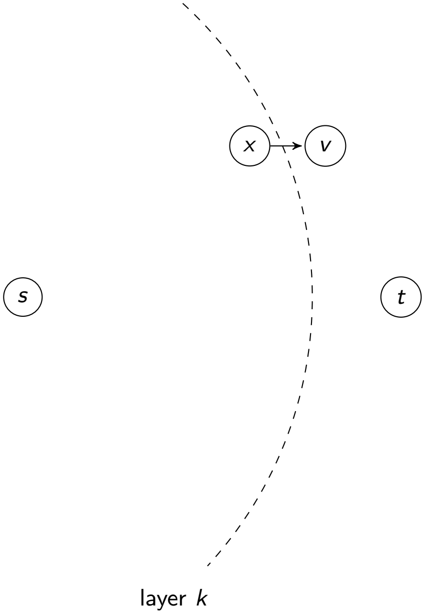
If all calls output 0

$\rightarrow$ Move to the next v

If $p(v) > n^c$

$\boxed{\rightarrow \text{Not Min-poly}}$

Algorithm to calculate c_{k+1},Σ_{k+1} and p_{k+1} (**Min-poly case**)



$\boxed{\text{Intitialize } (c_{k+1}, \Sigma_{k+1}, p_{k+1}) = (c_k, \Sigma_k, p_k)}$

Call the routine to check if $d(v) \leq k$

If it returns 0,

$\forall x \mid (x, v) \in E$, Call $d(x) \leq k$

If all calls output 0

$\rightarrow$ Move to the next v

If $p(v) > n^c$

$\boxed{\rightarrow \text{Not Min-poly}}$

Else

$$c_{k+1} := c_{k+1} + 1$$
$$\Sigma_{k+1} := \Sigma_{k+1} + k + 1$$
$$p_{k+1} := p_{k+1} + p(v)$$

Main

- non-deterministically guess m
- $c_0 = 1, \Sigma_0 = 0, p_0 = 1$
- for (k $=$ 1 to n) compute $[c_k, \Sigma_k, p_k]$ from $[c_{k-1}, \Sigma_{k-1}, p_{k-1}]$
- if t was covered, ACCEPT-m

Main

- non-deterministically guess m
- $c_0 = 1, \Sigma_0 = 0, p_0 = 1$
- for (k = 1 to n) compute $[c_k, \Sigma_k, p_k]$ from $[c_{k-1}, \Sigma_{k-1}, p_{k-1}]$
- if t was covered, ACCEPT-m

Analysis:

- If m does not hash appropriately even for one vertex v, the algorithm will fail while guessing $s \rightsquigarrow v$ paths.

Main

- non-deterministically guess m
- $c_0 = 1, \Sigma_0 = 0, p_0 = 1$
- for (k = 1 to n) compute $[c_k, \Sigma_k, p_k]$ from $[c_{k-1}, \Sigma_{k-1}, p_{k-1}]$
- if t was covered, ACCEPT-m

Analysis:

- If m does not hash appropriately even for one vertex v, the algorithm will fail while guessing $s \rightsquigarrow v$ paths.
- If m hashes correctly for all vertices, the algorithm will unambiguously reach the state ACCEPT
  (or the configuration [ACCEPT,m])

Main

- non-deterministically guess m
- $c_0 = 1, \Sigma_0 = 0, p_0 = 1$
- for (k = 1 to n) compute $[c_k, \Sigma_k, p_k]$ from $[c_{k-1}, \Sigma_{k-1}, p_{k-1}]$
- if t was covered, ACCEPT-m

Analysis:

- If m does not hash appropriately even for one vertex v, the algorithm will fail while guessing $s \rightsquigarrow v$ paths.
- If m hashes correctly for all vertices, the algorithm will unambiguously reach the state ACCEPT
  (or the configuration [ACCEPT,m])
  iff t is reachable from s and the graph is MIN-POLY.

Main

- non-deterministically guess m
- $c_0 = 1, \Sigma_0 = 0, p_0 = 1$
- for (k $=$ 1 to n) compute $[c_k, \Sigma_k, p_k]$ from $[c_{k-1}, \Sigma_{k-1}, p_{k-1}]$
- if t was covered, ACCEPT-m

Analysis:

- If m does not hash appropriately even for one vertex v, the algorithm will fail while guessing $s \rightsquigarrow v$ paths.
- If m hashes correctly for all vertices, the algorithm will unambiguously reach the state ACCEPT
  (or the configuration [ACCEPT,m])
  iff t is reachable from s and the graph is Min-Poly.
- Each accept configuration has at most one computational path (`FewUL`).

Making the algorithm Unambiguous

- Idea : Guess the <u>least</u> m which hashes all the paths distinctly (Call the guessed value as f' and the actual value as f).

Making the algorithm Unambiguous

- Idea : Guess the <u>least</u> m which hashes all the paths distinctly (Call the guessed value as f' and the actual value as f).
- Run the algorithm using $m = f'$. Additionally, run the algorithm for all $m = m' < f'$, making sure that algorithm finds "badness" of m' in a unique computational path.

Making the algorithm Unambiguous

- Idea : Guess the least m which hashes all the paths distinctly (Call the guessed value as f' and the actual value as f).
- Run the algorithm using $m = f'$. Additionally, run the algorithm for all $m = m' < f'$, making sure that algorithm finds "badness" of m' in a unique computational path.
- If f' is less than f, then f' is bad anyway and the algorithm will REJECT.
- If f' is more than f, then then in some iteration $m' = f$ will fail to find a "badness" and hence REJECT.
- IF $f' = f$, then attempts to find "badness" of m' will all together succeed in exactly one path. Since f is good and unique, the f' will make the main algorithm work unambiguously.

Find the "badness" of m' unambiguously

For each $m' < f'$,

- Guess the first level where a vertex v has two paths to it which are not hashed correctly. Guess this as k_1' (actual one being k_1) and search for the v in the lex ordering.

Find the "badness" of m' unambiguously

For each $m' < f'$,

- Guess the first level where a vertex v has two paths to it which are not hashed correctly. Guess this as k_1' (actual one being k_1) and search for the v in the lex ordering.
- For any such vertex v, there must exist $a, b \in V$ such that a, b are in-neighbours of v at distance $k_1' - 1$ from s and there must be two paths, p_a through a and p_b through b such that $\phi_m(p_a) = \phi_m(p_b)$. Search through the (a, b) pairs in lex ordering.

Find the "badness" of m' unambiguously

For each $m' < f'$,

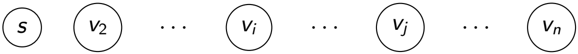
- Guess the first level where a vertex v has two paths to it which are not hashed correctly. Guess this as k'_1 (actual one being k_1) and search for the v in the lex ordering.
- For any such vertex v, there must exist $a, b \in V$ such that a, b are in-neighbours of v at distance $k'_1 - 1$ from s and there must be two paths, p_a through a and p_b through b such that $\phi_m(p_a) = \phi_m(p_b)$. Search through the (a, b) pairs in lex ordering.
- For each (a, b) pair, compute $p(a)$ and $p(b)$ respectively. Guess the paths in the strictly increasing order of ϕ_m hashes and try all the pair of paths among them for witness for "badness" of m.

Reduction from Reach on a DAG to LongPath

$Reach(G, s, t) \to LongPath(G', s', t, 2n + 1)$ (n is the number of vertices in G)

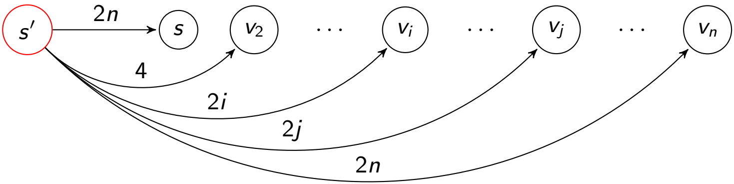
Reduction from REACH on a DAG to LONGPATH

$Reach(G, s, t) \rightarrow LongPath(G', s', t, 2n + 1)$ (n is the number of vertices in G)



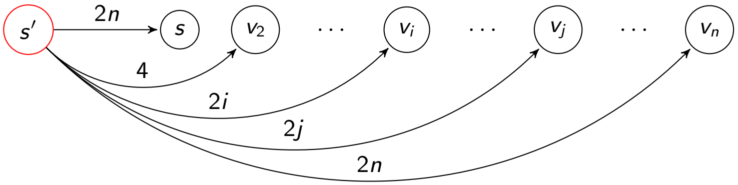
Reduction from REACH on a DAG to LONGPATH

$Reach(G, s, t) \to LongPath(G', s', t, 2n + 1)$ (n is the number of vertices in G)



Reduction from Reach on a DAG to LongPath

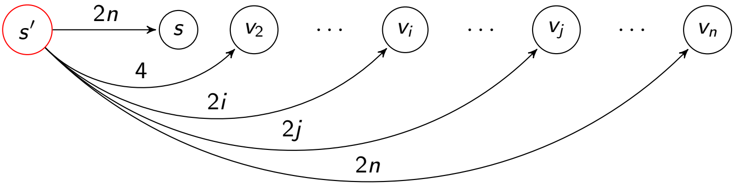
$Reach(G, s, t) \to LongPath(G', s', t, 2n+1)$ (n is the number of vertices in G)



We can see that

Reduction from REACH on a DAG to LONGPATH

$Reach(G, s, t) \to LongPath(G', s', t, 2n + 1)$ (n is the number of vertices in G)

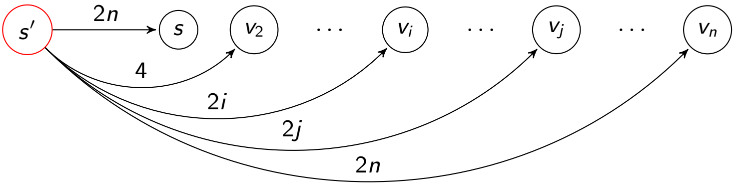


We can see that

- G' is a single-source DAG

Reduction from REACH on a DAG to LONGPATH

$Reach(G, s, t) \to LongPath(G', s', t, 2n+1)$ (n is the number of vertices in G)

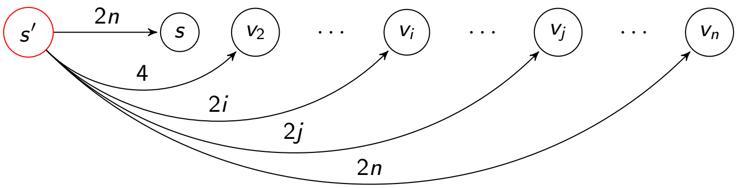


We can see that

- G' is a single-source DAG
- There is a $(s' \rightsquigarrow t)$ path of length at least $2n+1$ in G' if and only if there was a $(s \rightsquigarrow t)$ path in G.

Reduction from Reach on a DAG to LongPath

$Reach(G, s, t) \rightarrow LongPath(G', s', t, 2n + 1)$ (n is the number of vertices in G)



We can see that

- G' is a single-source DAG
- There is a ($s' \rightsquigarrow t$) path of length at least $2n + 1$ in G' if and only if there was a ($s \rightsquigarrow t$) path in G.
- G' is max-unique (max-poly) if and only if G is max-unique (max-poly).

Open Problems

- In this paper, we designed UL algorithms for $\textsc{Reach}$ in directed graphs augmented with min-poly or max-poly weight assignments.

Open Problems

- In this paper, we designed UL algorithms for Reach in directed graphs augmented with min-poly or max-poly weight assignments.

Open Problems:

- Are min-poly (resp. max-poly) log-space computable weighing schemes easier to design than min-unique (resp. max-unique) log-space computable weighing schemes?

Open Problems

- In this paper, we designed $\mathtt{UL}$ algorithms for $\textsc{Reach}$ in directed graphs augmented with min-poly or max-poly weight assignments.

Open Problems:

- Are min-poly (resp. max-poly) log-space computable weighing schemes easier to design than min-unique (resp. max-unique) log-space computable weighing schemes?
- Can we apply any of the above for restricted graph classes? (We know this for grid graphs [Bourke, Tewari, Vinodchandran - 2009]) If we are able to apply this to "Monotone 3D grid graphs", then $\mathtt{NL} = \mathtt{UL}$.

Open Problems

- In this paper, we designed $\mathtt{UL}$ algorithms for $\textsc{Reach}$ in directed graphs augmented with min-poly or max-poly weight assignments.

Open Problems:

- Are min-poly (resp. max-poly) log-space computable weighing schemes easier to design than min-unique (resp. max-unique) log-space computable weighing schemes?
- Can we apply any of the above for restricted graph classes? (We know this for grid graphs [Bourke, Tewari, Vinodchandran - 2009]) If we are able to apply this to "Monotone 3D grid graphs", then $\mathtt{NL} = \mathtt{UL}$.
- Structural study of weighing schemes and their design complexity?

Thank You