CS630 Graduate Algorithms

September 17, 2024

by Dora Erdos and Jeffrey Considine

- Polynomial reductions

slides by Dora Erdos and Jeffrey Considine, based on Kevin Wayne, Adam Smith

Polynomial Reduction to Problems (informal)

Think of the max-flow solving algorithm (in our case FF - there are others) as a call to an API.

In the applications we used this API to solve other problems.

- max bipartite matching
- disjoint paths
- baseball elimination
- blood donation, dance competition, spies, etc.

Can we do this API-style trick with other algorithms/solvers too?

Polynomial-time reductions

Suppose there is an algorithm to solve problem Y.

- the algorithm is unknown to us, we treat it as a *black-box* or *API*

Polynomial Reduction. Problem X is polynomial-time reducible to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle/black box that solves problem Y.

Notation. $X \leq_P Y$.

Bipartite Matching $\leq_p$ Max Flow

- define flow graph corresponding to the bipartite graph
- find max flow (using FF)
- iterate over middle edges and assign pairs with non-zero flow to the matching

Longest Path and Hamiltonian Path problems

Longest Path: Given a weighted graph *G* and an integer *k*, is there a simple path of lengths at least *k*?

- if G does have a path of length k, it is easy to “prove” by returning the path itself
  - Proof by example - the path is an easy certificate for Longest Path

Longest Path and Hamiltonian Path problems

Longest Path: Given a weighted graph *G* and an integer *k*, is there a simple path of lengths at least *k*?

- if G does have a path of length k, it is easy to ”prove” by returning the path itself
  - Proof by example - the path is an easy certificate for Longest Path

Hamiltonian-path problem: Given an *un*weighted graph G(V,E) is there a simple path that contains every node exactly once?

  - Proof by example - the path is another easy certificate

Hamiltonian Path - TopHat

Hamiltonian path problem: Given an *un*weighted graph G, is there a simple path in G that contains *all* vertices?

Suppose that there is an algorithm Long( ) that solves the Longest path problem.

Question: we can use Long( ) to solve the Ham-path problem. Which of these best describes this approach?

A. Pick a random source s. Run DFS on G from s to find the “deepest” path. If this contains every node, then we found the Hamiltonian path.

B. Assign weight 1 to each edge in G then call Long(G, n-1). If it returns n-1 as the longest simple path length, then G contains a Hamiltonian path.

C. Find whether there is a Hamiltonian path in G. If yes, then return that Long(G) is n-1.

Longest Path and Hamiltonian Path problems

Longest Path: Given a weighted graph *G* and an integer *k*, is there a simple path of lengths at least *k*?

- we don't know of any polynomial algorithm for it
  - No super-polynomial lower bound either
  - Belongs to a class called NP-Complete → next lecture's topic
- if G does have a path of length k, it is easy to "prove" by returning the path itself

Longest Path and Hamiltonian Path problems

Longest Path: Given a weighted graph *G* and an integer *k*, is there a simple path of lengths at least *k*?

- we don't know of any polynomial algorithm for it
  - No super-polynomial lower bound either
  - Belongs to a class called NP-Complete → next lecture's topic
- if G does have a path of length k, it is easy to "prove" by returning the path itself

Hamiltonian-path problem: Given an unweighted graph G(V,E) is there a simple path that contains every node exactly once?

There is a polynomial-time reduction from Hamiltonian-path to Longest Path. Which means that any algorithm solving LP can also be used to solve Ham-Path with polynomial number of extra steps.

Polynomial-time reductions

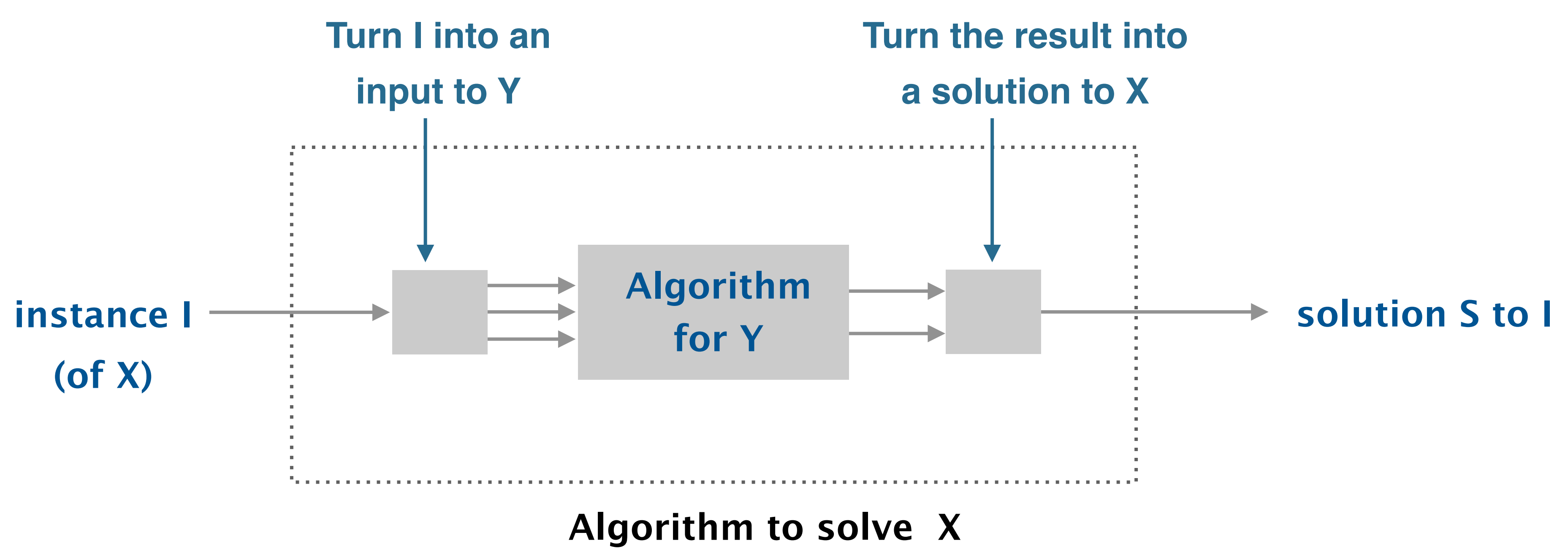
Suppose there is an algorithm to solve problem Y.

- the algorithm is unknown to us, we treat it as a *black-box* or *API*

Polynomial Reduction. Problem X is polynomial-time reducible to problem Y if arbitrary instances of problem X can be solved using:

- Polynomial number of standard computational steps, plus
- Polynomial number of calls to an oracle/black box that solves problem Y.

Notation. $X \leq_P Y$.



Hamiltonian path $\leq_P$ Longest Path

Same result, new notation.

Circuit Value Problem

Given a description of a Boolean circuit and its input values, calculate the output value of the circuit.

A Boolean circuit is usually describe with a directed acyclic graph with vertices labeled as inputs, outputs, or internal gates (of type NOT/AND/OR…)

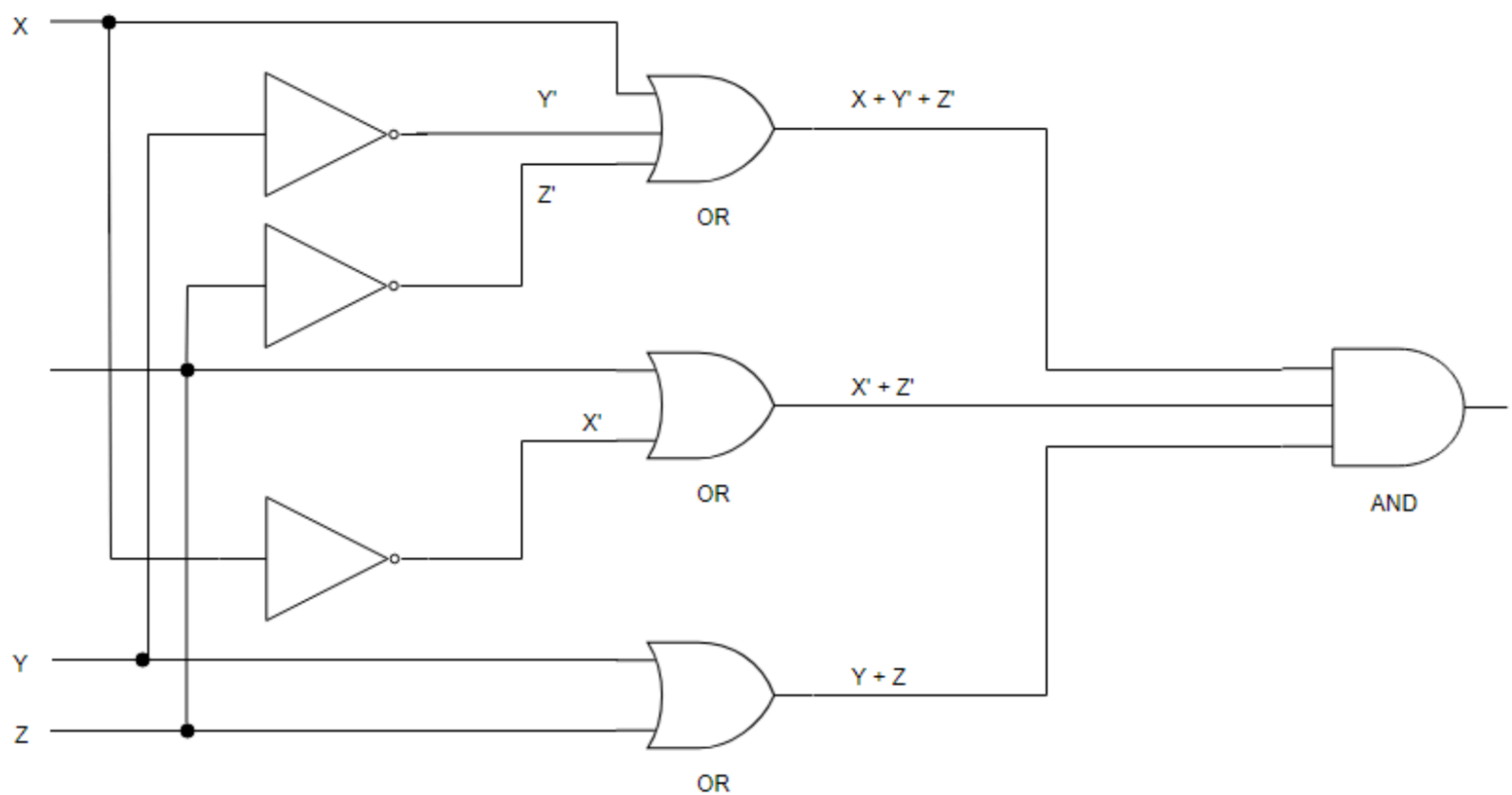
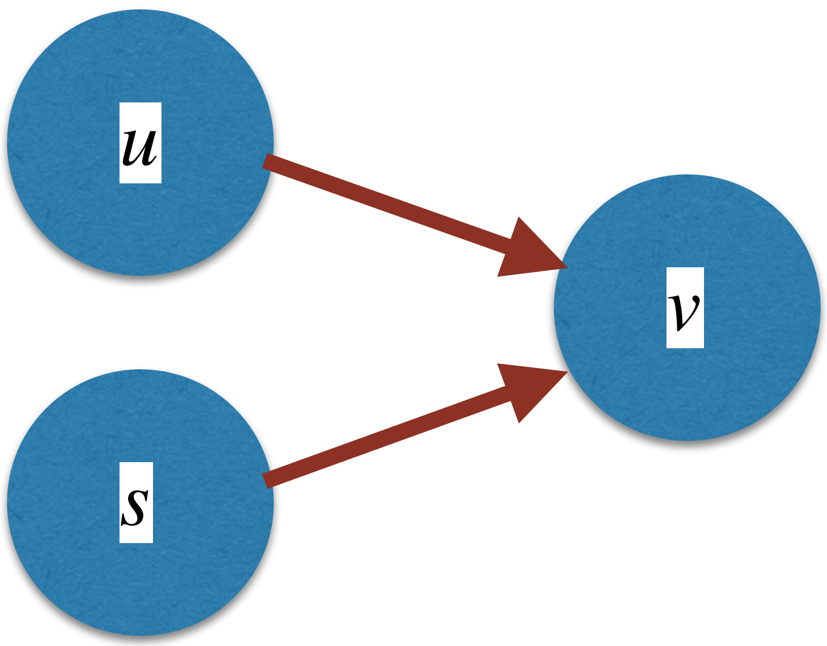


Image source: https://online.visual-paradigm.com/

Topological Sort

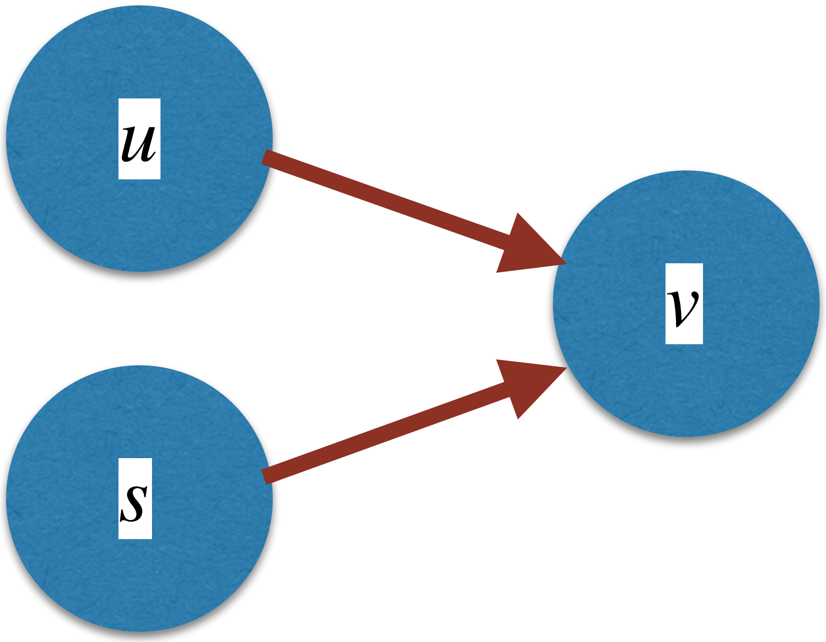
Given a directed acyclic graph, generate an order of the vertices such that if there is an edge $u \to v$, then u comes before v in the new order.



What are the topological sorts of this graph?

Topological Sort

Given a directed acyclic graph, generate an order of the vertices such that if there is an edge $u \to v$, then u comes before v in the new order.

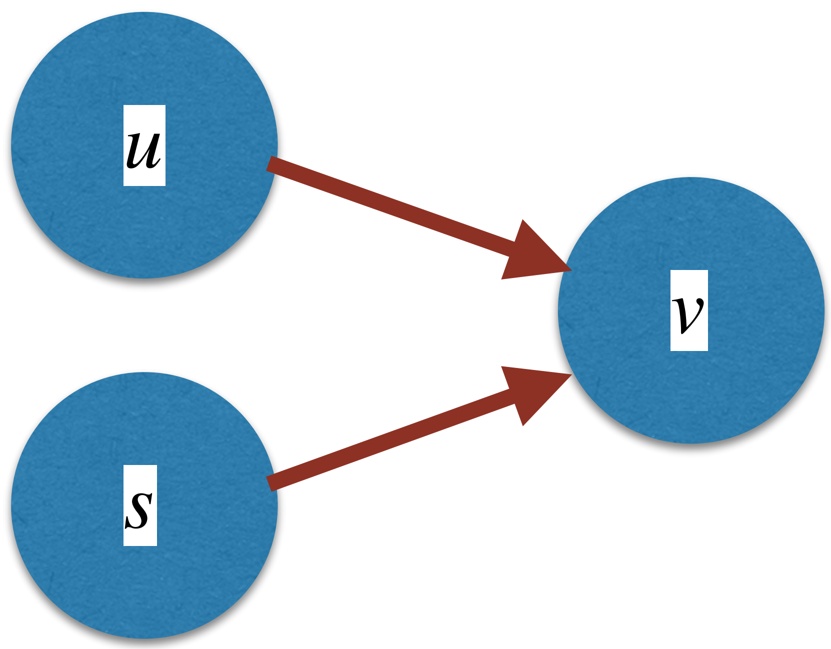


What are the topological sorts of this graph?

1. $u \to s \to v$
2. $s \to u \to v$

Topological Sort

Given a directed acyclic graph, generate an order of the vertices such that if there is an edge $u \to v$, then u comes before v in the new order.



What are the topological sorts of this graph?

1. $u \to s \to v$
2. $s \to u \to v$

How does this relate to the previous problem?

Circuit Value Problem $\leq_P$ Topological Sort

Loosely following "left-to-right" evaluation strategy…

Circuit Value Problem $\leq_P$ Topological Sort

Loosely following "left-to-right" evaluation strategy…

- Perform a topological sort of the circuit DAG.
  - Input → output
- Traverse this order and calculate all the node values in one pass.
  - When we get to a node, all its inputs are already calculated.

Circuit Value Problem $\leq_P$ Topological Sort

Loosely following "left-to-right" evaluation strategy…

- Perform a topological sort of the circuit DAG.
    - Input → output
- Traverse this order and calculate all the node values in one pass.
    - When we get to a node, all its inputs are already calculated.

No recursion after topological sort.

(Topological sort usually uses modified DFS.)

Cosine Similarity

Old measure of vector similarity.

$$u \cdot v = u^T v = \cos\ \theta\ \|u\|\ \|v\|$$

$$\frac{u \cdot v}{\|u\|\ \|v\|} = \cos\ \theta$$

Cosine Similarity

Old measure of vector similarity.

$$u \cdot v = u^T v = \cos\,\theta \;\|u\|\;\|v\|$$

$$\frac{u \cdot v}{\|u\|\;\|v\|} = \cos\,\theta$$

Information Retrieval

- Map documents of interest to vectors of features.
- Use cosine similarity to compare document vectors and identify related or similar documents.
- Higher cosine similarity → better match.

Cosine Similarity

Old measure of vector similarity.

$$u \cdot v = u^T v = \cos\,\theta\ \|u\|\ \|v\|$$

$$\frac{u \cdot v}{\|u\|\ \|v\|} = \cos\,\theta$$

Information Retrieval

- Map documents of interest to vectors of features.
- Use cosine similarity to compare document vectors and identify related or similar documents.
- Higher cosine similarity → better match.

Popular again with new machine learning developments

- Turns out many large language model make really good vectors.
- Revival of cosine similarity to retrieve related documents, and drive Retrieval Augmented Generation (RAG)

Cosine Similarity

How to find vectors with highest cosine similarity to v?

Cosine Similarity

How to find vectors with highest cosine similarity to v?

$\theta \rightarrow 0$ to get $\cos\,\theta \rightarrow 1$ (max)

Cosine Similarity $\leq_P$ Nearest Neighbors

How to find vectors with highest cosine similarity to v?

$\theta \rightarrow 0$ to get $\cos\,\theta \rightarrow 1$ (max)

1. Normalize all unit vectors to be unit vectors.
   (Divide by length, so resulting vectors have length 1.)
2. Find the nearest neighbors of v.

Cosine Similarity ≤P Nearest Neighbors

How to find vectors with highest cosine similarity to v?

$\theta \to 0$ to get $\cos\,\theta \to 1$ (max)

1. Normalize all unit vectors to be unit vectors.
   (Divide by length, so resulting vectors have length 1.)
2. Find the nearest neighbors of v.
   - Now all the vectors are on a unit hypersphere.
   - Smaller θ corresponds to smaller distance.

Cosine Similarity $\leq_P$ Nearest Neighbors

How to find vectors with highest cosine similarity to v?

$\theta \to 0$ to get $\cos\,\theta \to 1$ (max)

1. Normalize all unit vectors to be unit vectors.
   (Divide by length, so resulting vectors have length 1.)
2. Find the nearest neighbors of v.
   - Now all the vectors are on a unit hypersphere.
   - Smaller θ corresponds to smaller distance.

Isn't reducing cosine similarity to nearest neighbors reducing to the same problem?

Cosine Similarity ≤$_P$ Nearest Neighbors

How to find vectors with highest cosine similarity to v?

$\theta \to 0$ to get $\cos\,\theta \to 1$ (max)

1. Normalize all unit vectors to be unit vectors.
   (Divide by length, so resulting vectors have length 1.)
2. Find the nearest neighbors of v.
   - Now all the vectors are on a unit hypersphere.
   - Smaller θ corresponds to smaller distance.

Isn't reducing cosine similarity to nearest neighbors reducing to the same problem?

- Very similar, but nearest neighbors is an "off the shelf" problem.
- Vector databases are specialized for this.

Cosine Similarity ≤$_P$ Nearest Neighbors

How to find vectors with highest cosine similarity to v?

$\theta \to 0$ to get $\cos\,\theta \to 1$ (max)

1. Normalize all unit vectors to be unit vectors.
   (Divide by length, so resulting vectors have length 1.)
2. Find the nearest neighbors of v.
   - Now all the vectors are on a unit hypersphere.
   - Smaller θ corresponds to smaller distance.

Isn't reducing cosine similarity to nearest neighbors reducing to the same problem?

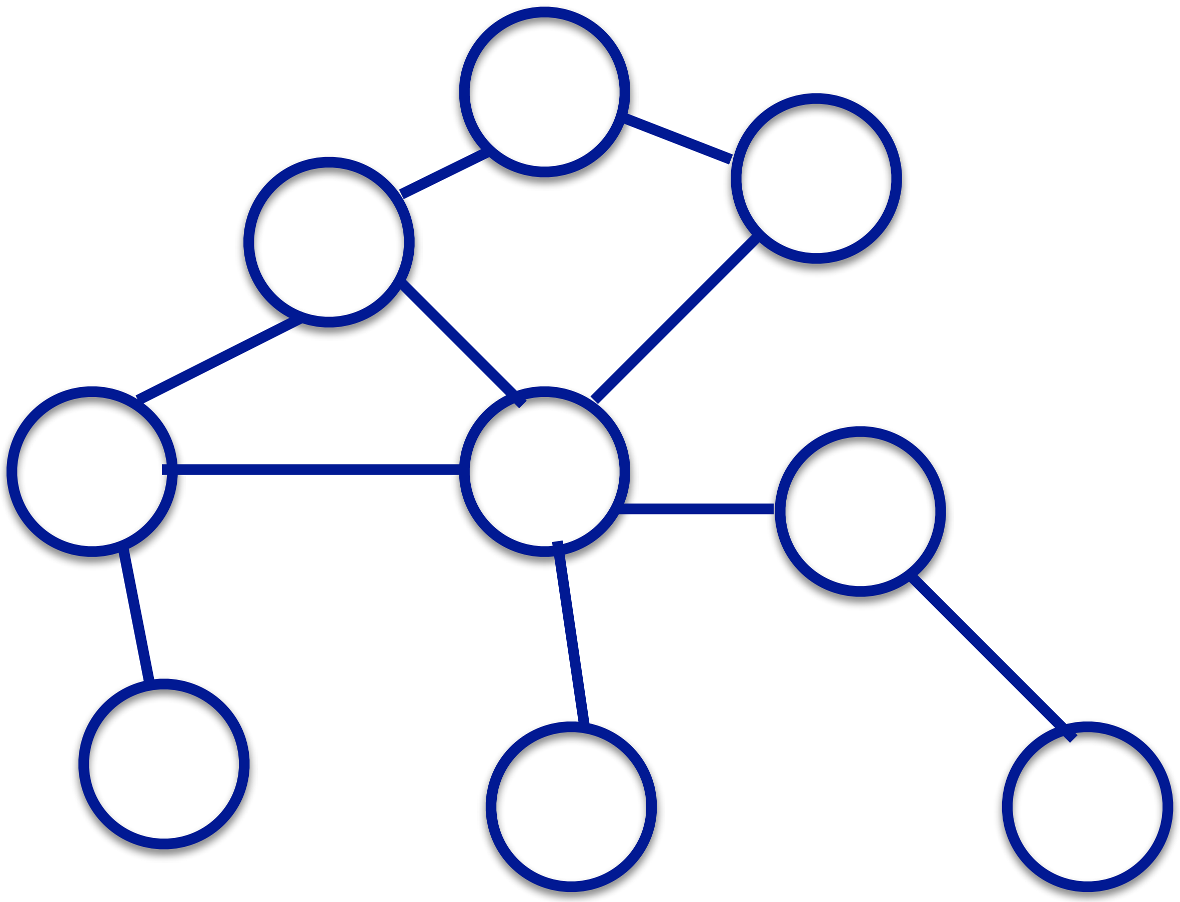
- Very similar, but nearest neighbors is an "off the shelf" problem.
- Vector databases are specialized for this.

TLDR:

- Reduce cosine similarity to nearest neighbors by normalizing lengths.
- Then highest cosine similarity is the same as smallest distance.

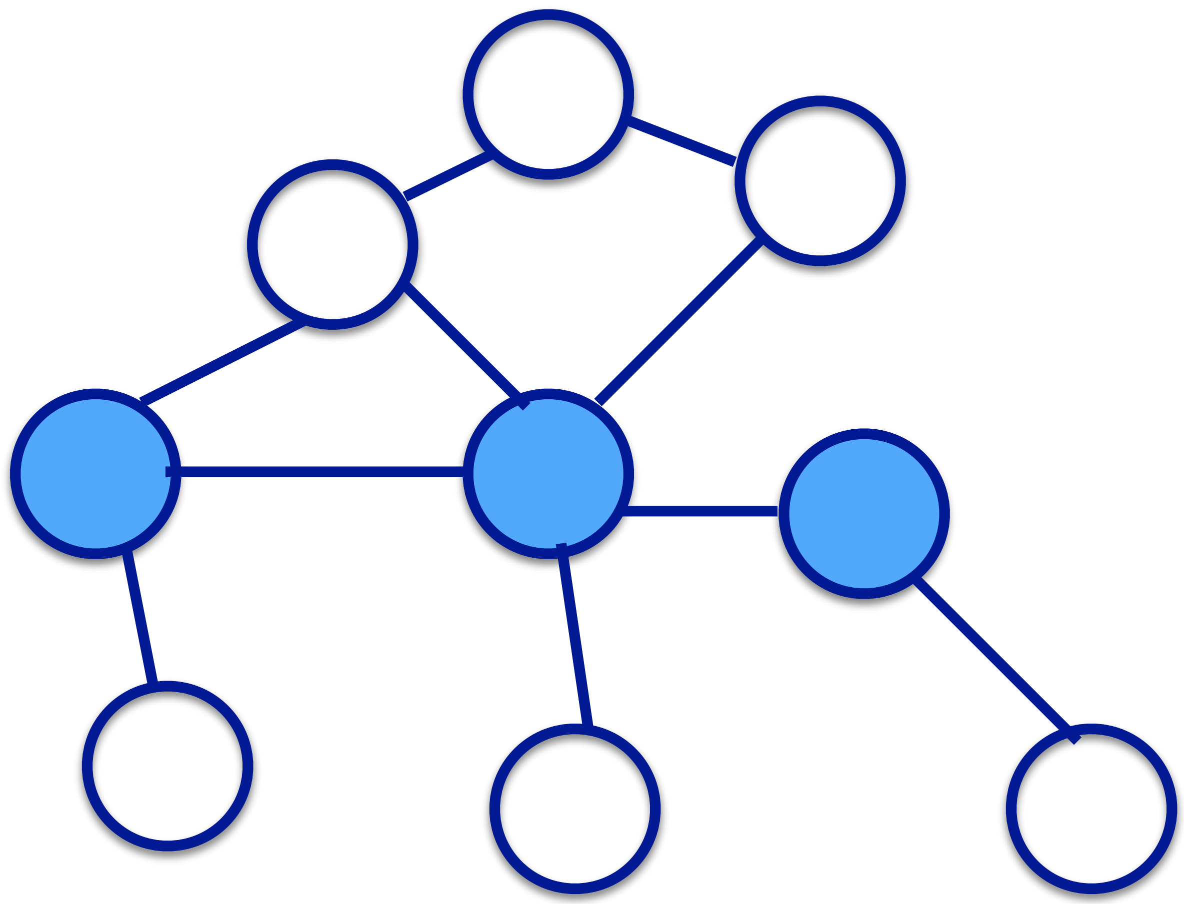
Vertex Cover

Given a graph $G = (V, E)$, find a minimum subset of V such that each edge is covered. That is, find minimum $V' \subseteq V$ such that $\forall (u, v) \in E \; (u \in V' \lor v \in V')$.



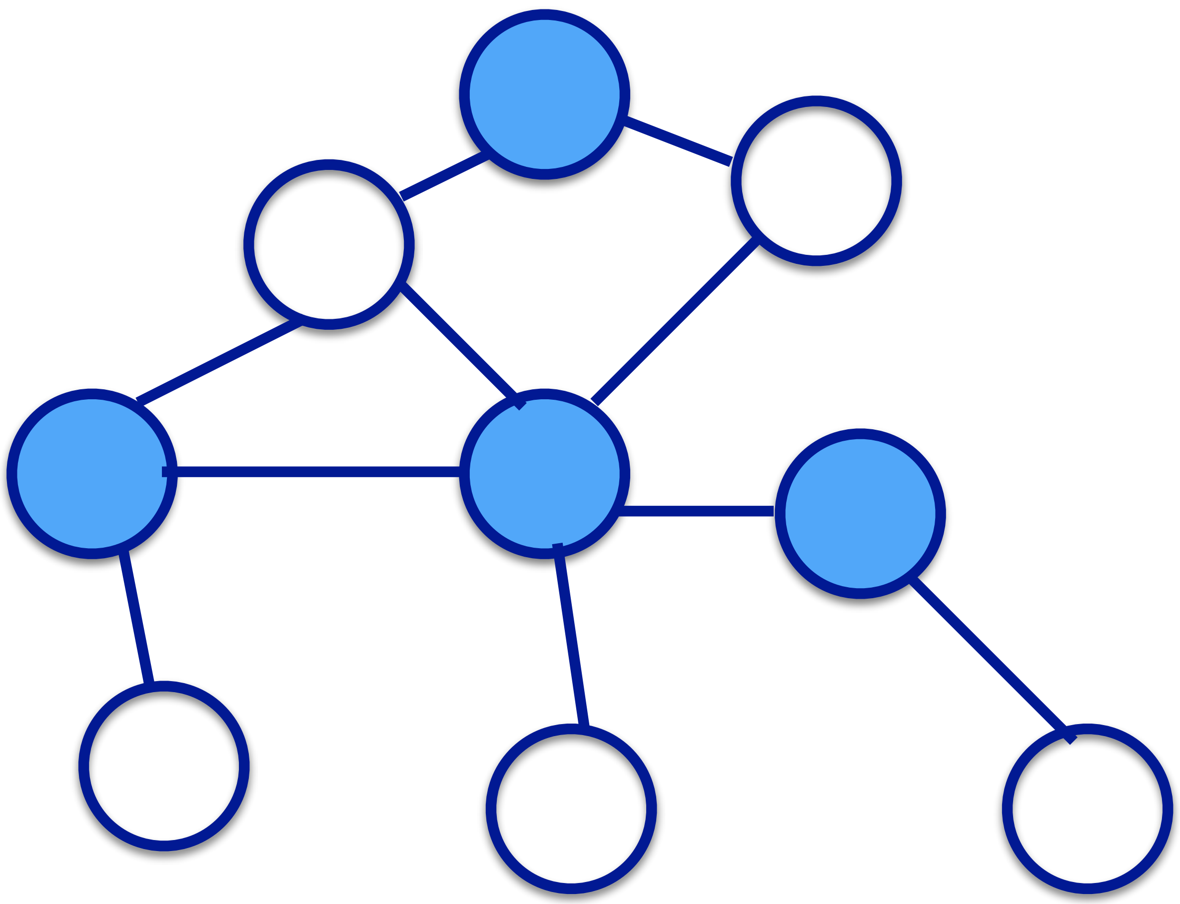
Vertex Cover

Given a graph $G = (V, E)$, find a minimum subset of V such that each edge is covered. That is, find minimum $V' \subseteq V$ such that $\forall (u, v) \in E \ (u \in V' \vee v \in V')$.



Vertex Cover

Given a graph $G = (V, E)$, find a minimum subset of V such that each edge is covered. That is, find minimum $V' \subseteq V$ such that $\forall (u, v) \in E \ (u \in V' \lor v \in V')$.



Set Cover

Finding the minimum set of courses to fulfill your HUB requirements

Algorithm: always pick the course with the largest number of additional HUB units

	CS111	CS112	CS132	AN101	AN103	EE150	BI306	AR307
Quant Reas	X	X	X					X
Crit. think	X	X	X		X			X
Creat/Inno	X	X						
Digi/multi			X					
res and inf				X		X	X	X
Social Inq				X	X			
Indiv in Com					X			
Sci Inq						X	X	X
History						X		

AN101 Intro to Cult Anthropology AN103 Anthropology through Ethnography AS105 EE150 Sustainable Energy BI306 Bio of global change AR307 Archeological Science

Set Cover

Given a set of n items $U = \{u_1, u_2, \ldots, u_n\}$ and m subsets of these items $S_1, S_2, \ldots S_m$ find a minimum number of subsets, such that their union contains every element in U.

Vertex Cover $\leq_P$ Set Cover

Reduction:

Vertex Cover $\leq_P$ Set Cover

Reduction:

1. VC edges $\rightarrow$ SC items

Vertex Cover $\leq_P$ Set Cover

Reduction:

1. VC edges → SC items
2. VC vertices → SC sets of the VC edge SC items

Vertex Cover $\leq_P$ Set Cover

Reduction:

1. VC edges $\rightarrow$ SC items
2. VC vertices $\rightarrow$ SC sets of the VC edge SC items

So,

- Picking an SC set $\leftrightarrow$ picking a VC vertex

Vertex Cover $\leq_P$ Set Cover

Reduction:

1. VC edges $\rightarrow$ SC items
2. VC vertices $\rightarrow$ SC sets of the VC edge SC items

So,

- Picking an SC set $\leftrightarrow$ picking a VC vertex
- Items covered by picked SC set $\leftrightarrow$ VC edges covered by picked VC vertex

Vertex Cover to Set Cover

Reduction:

1. VC edges → SC items
2. VC vertices → SC sets of the VC edge SC items

So,

- Picking an SC set ↔ picking a VC vertex
- Items covered by picked SC set ↔ VC edges covered by picked VC vertex

Spoiler for next time: Set Cover $\leq_P$ Vertex Cover too!