Reductions

https://cs.pomona.edu/classes/cs140/

https://adriann.github.io/npc/npc.html

Outline

Topics and Learning Objectives

- Discuss the process of reducing one problem to another

Exercise

- None

Quick check: does $\lg(n) \in P$?

Reduction

- Instead of taking the time to mathematically prove that some algorithm/problem belongs to a certain class, we can take a shortcut.
- We can put a problem in a specific class by looking at its relative difficulty.
- [Some Problem] is as hard as [Some Other Problem].
- “The decision TSP Problem is as hard as the Hamiltonian Cycle Problem, which is NP-Hard. Therefore, decision TSP is also NP-Hard (or NP-Complete in this case since we can verify it with a polynomial time algorithm)”

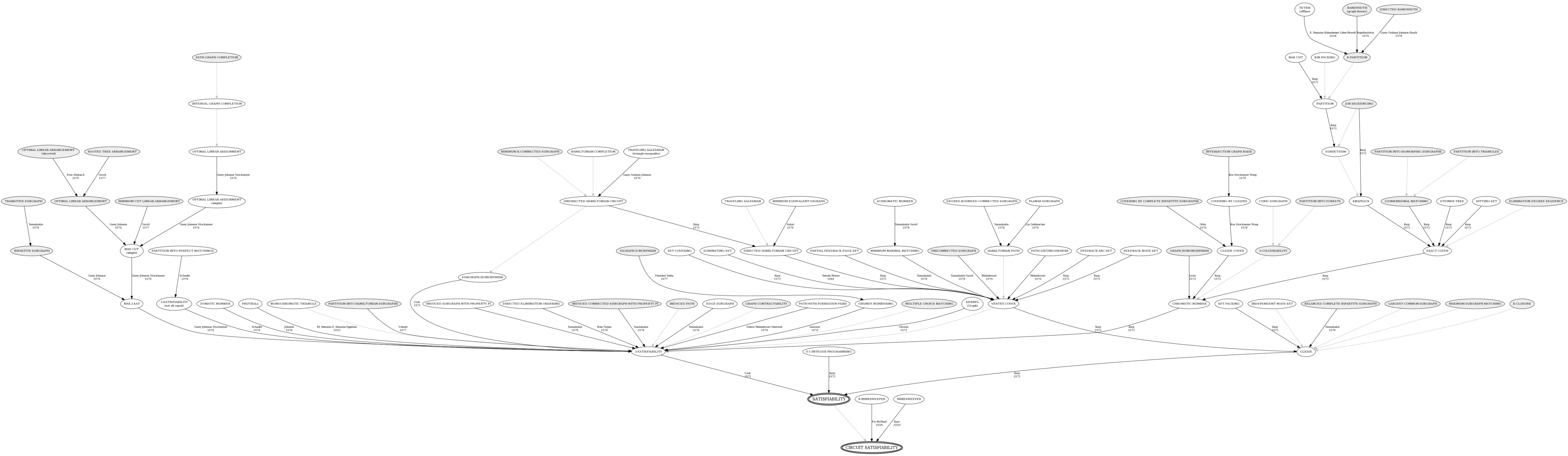
Complexity Comparisons

If you want to show that problem A is “easy”, then…
you show how to solve it by turning it into a known “easy” problem B.

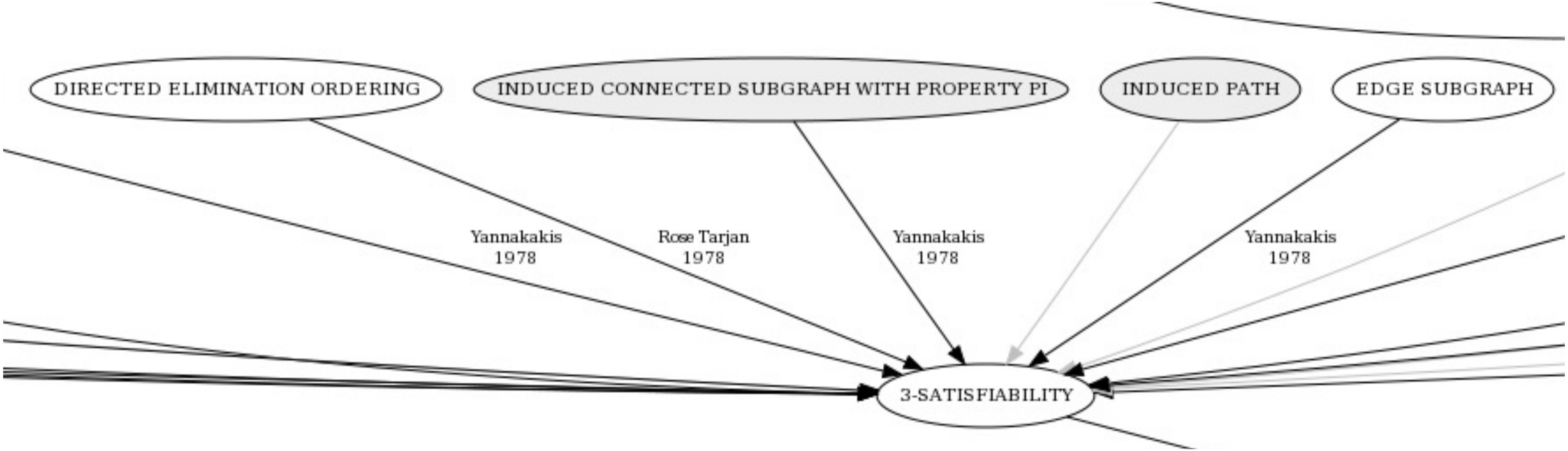
If you want to show that problem A is “hard”, then…
you show how it can be used to solve a known “hard” problem B.

These are called reductions.

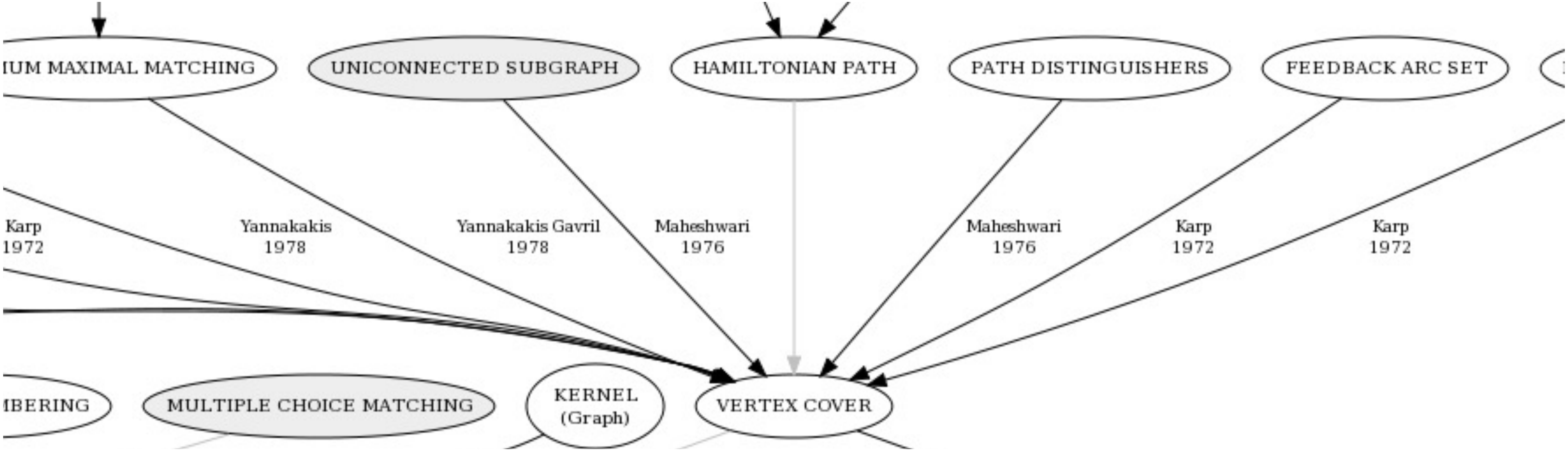
https://adriann.github.io/npc/npc.html



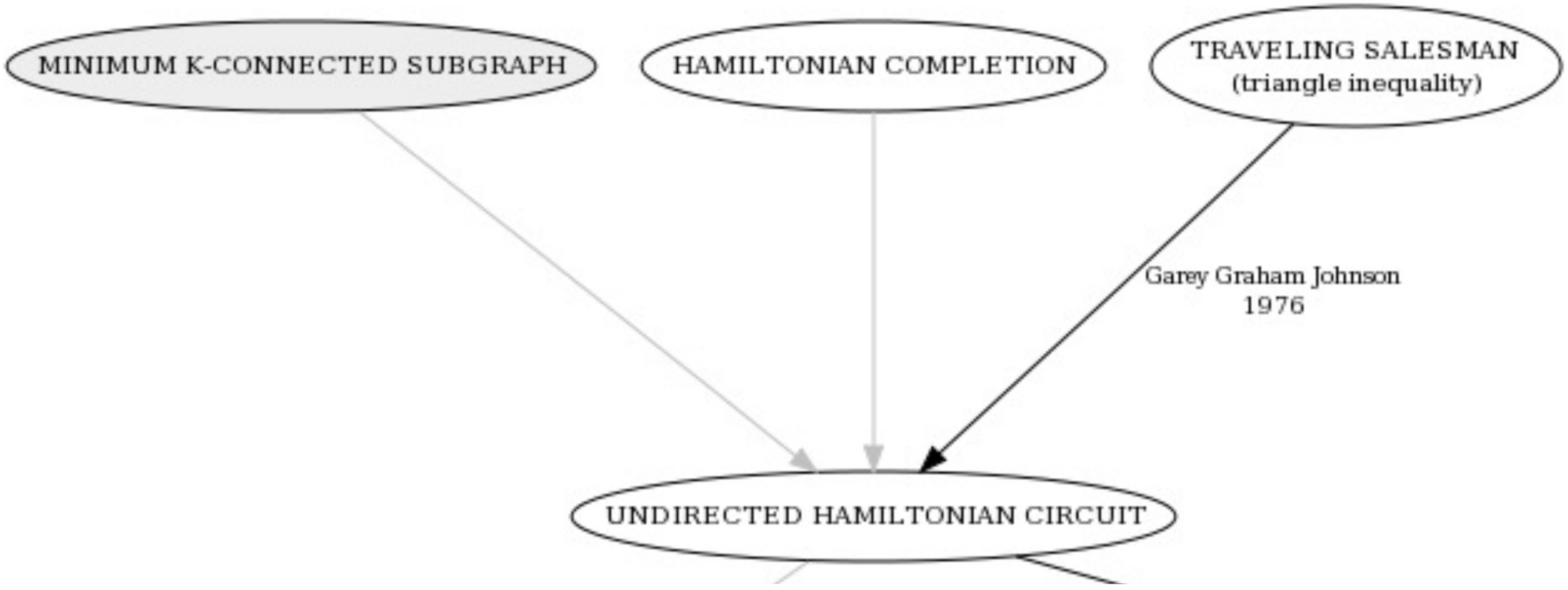
https://adriann.github.io/npc/npc.html



https://adriann.github.io/npc/npc.html



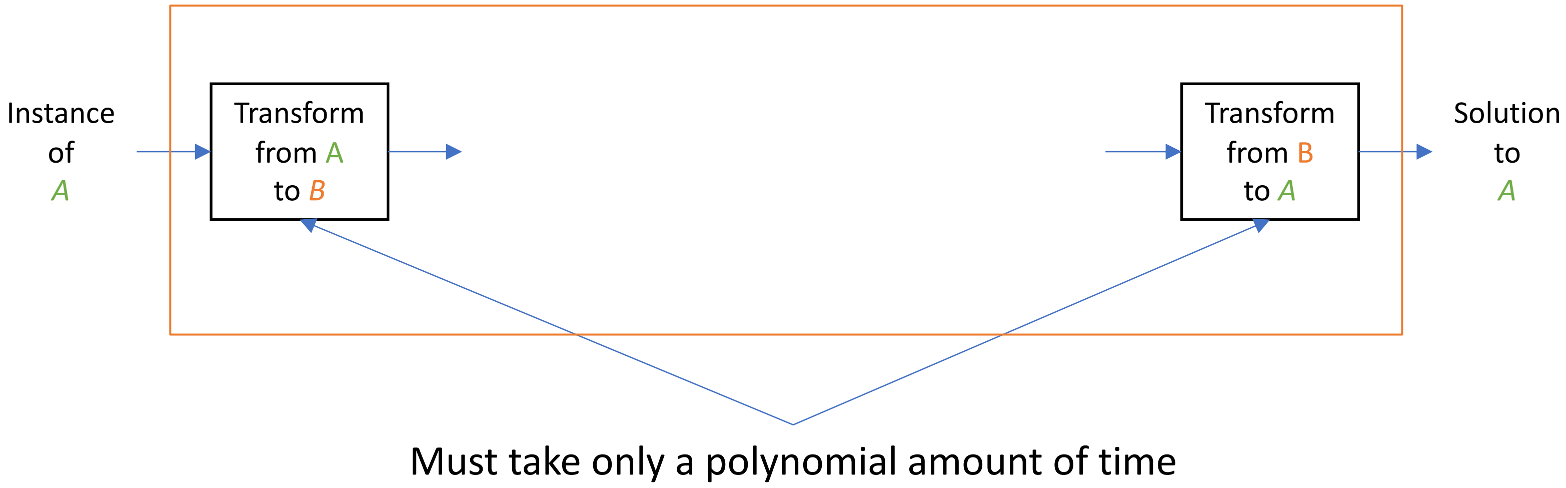
https://adriann.github.io/npc/npc.html



A reduction involves two different problems

We can reduce problem A to problem B if

- We have a polynomial time algorithm for converting an input to problem A into an equivalent input for problem B **and**
- We have a polynomial time algorithm for converting an output of problem B into an output of problem A



A reduction involves two different problems

We can reduce problem A to problem B if

- We have a polynomial time algorithm for converting an input to problem A into an equivalent input for problem B **and**
- We have a polynomial time algorithm for converting an output of problem B into an output of problem A

If we can perform a reduction, then we can say things like

- If B is in P then A is in P
- If B is in NP-Complete then A is in NP-Complete
- B is at least as hard as A (though B might be much harder—you can always convert a problem into something that takes way more work)

Reduction Example

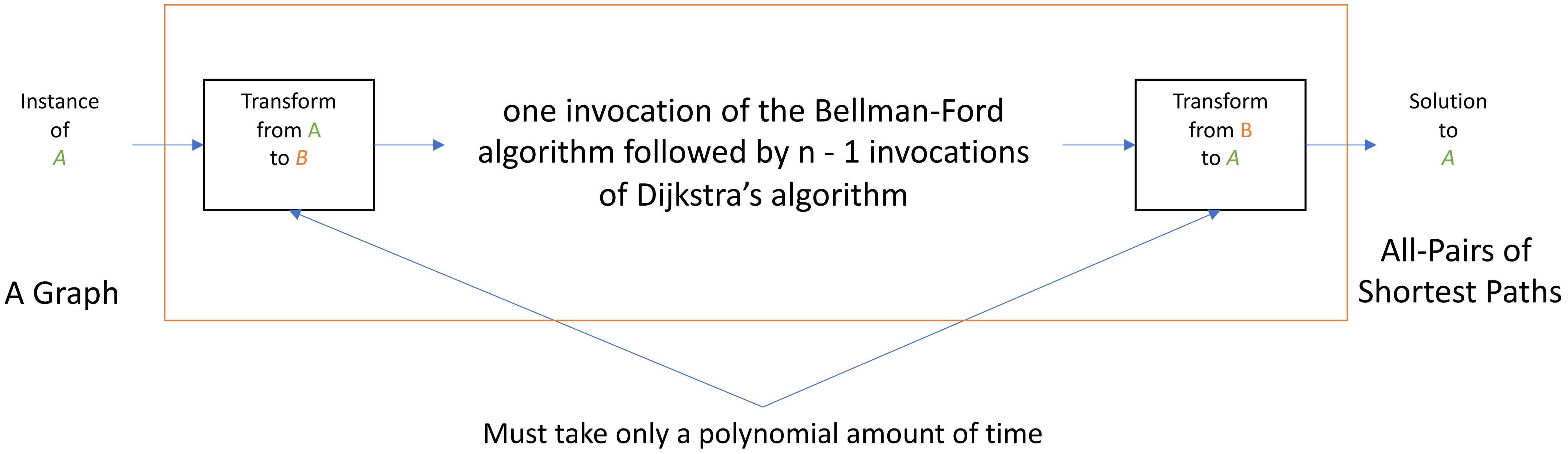
- We can do better than the Floyd-Warshall algorithm $O(n^3)$ for sparse graphs (even with negative edges).

- For example, a clever trick reduces the all-pairs shortest path problem to one invocation of the Bellman-Ford algorithm followed by n - 1 invocations of Dijkstra's algorithm.

- This reduction, which is called Johnson's algorithm, runs in $O(mn) + (n - 1) \cdot O(m \log n) = O(mn \log n)$.

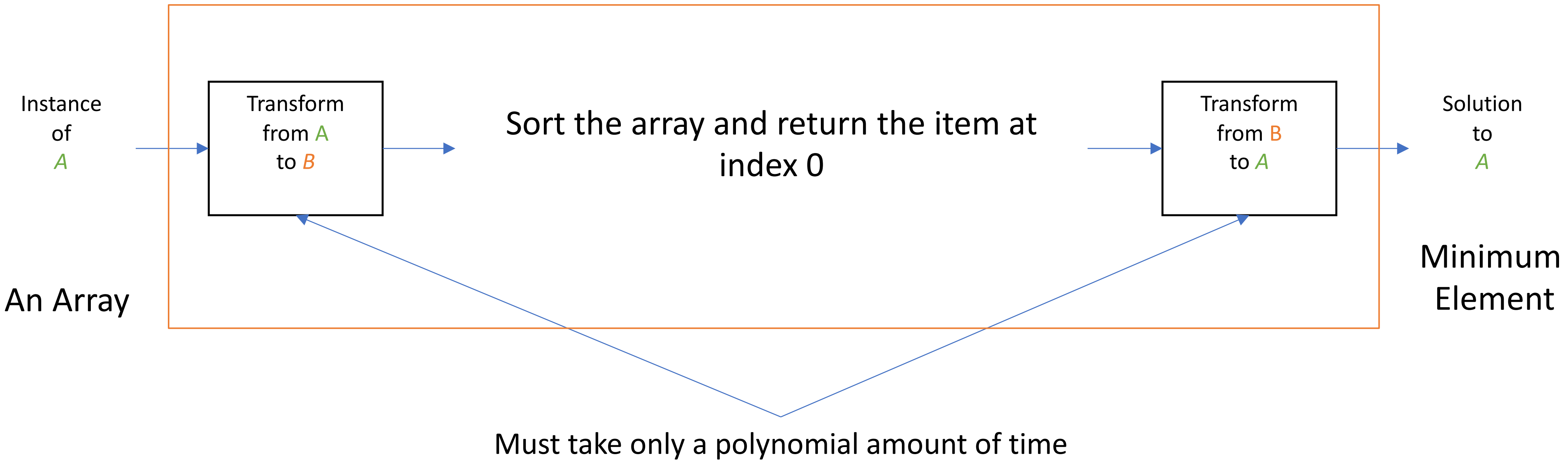
- This is subcubic in n except when m is very close to quadratic in n.

John’s All-Pairs Shortest Path Algorithm



Finding the Minimum Element

This is making the problem take more work than needed... But the reduction is still possible.



Reduction for NP-Complete

- Given a new problem (and algorithm) called P_{new}
- Let's say we have an algorithm (potentially sub-optimal) to solve it, but we don't know to what class it belongs.

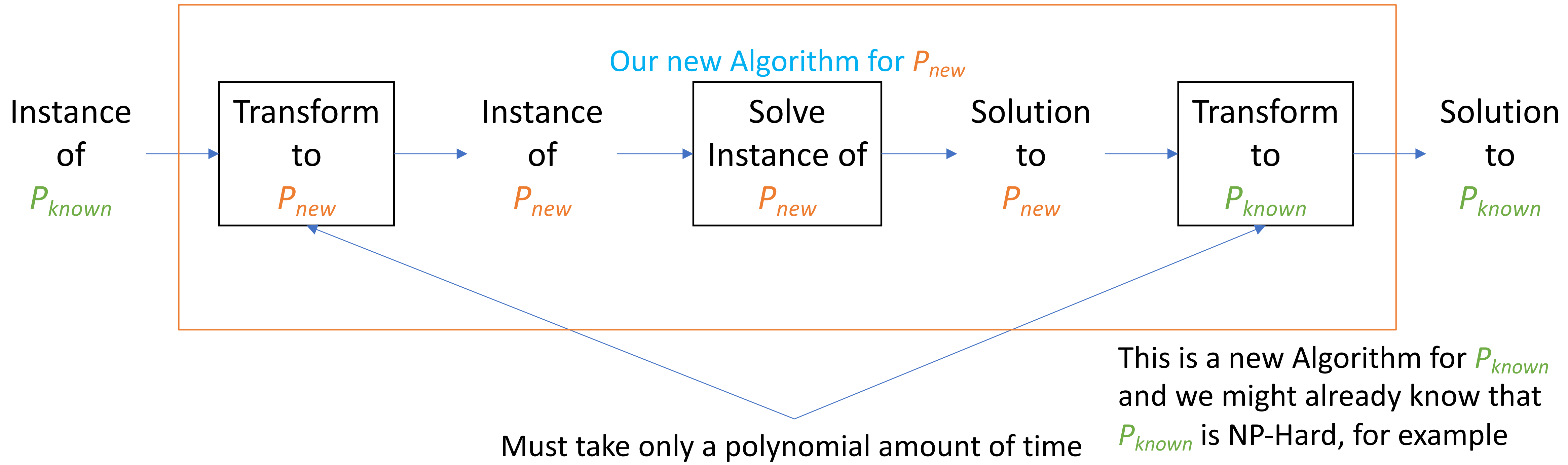
- We guess that (our Theorem)

new problem — already proven to be in NP-Complete

Problem P_{new} is at least as hard as problem P_{known}

- Reduce P_{known} to P_{new} ($\mathrm{P_{KNOWN}} \leq_p \mathrm{P_{NEW}}$)
  - Solve P_{known} using a polynomial number of calls to the algorithm for P_{new}
  - Reduce the **harder/known** known problem to our new problem
  - In doing say we can say that we've either found a more efficient solution to P_{known}, or we've proved that P_{new} is also hard

Example Reduction (Reduce P_{known} to P_{new})



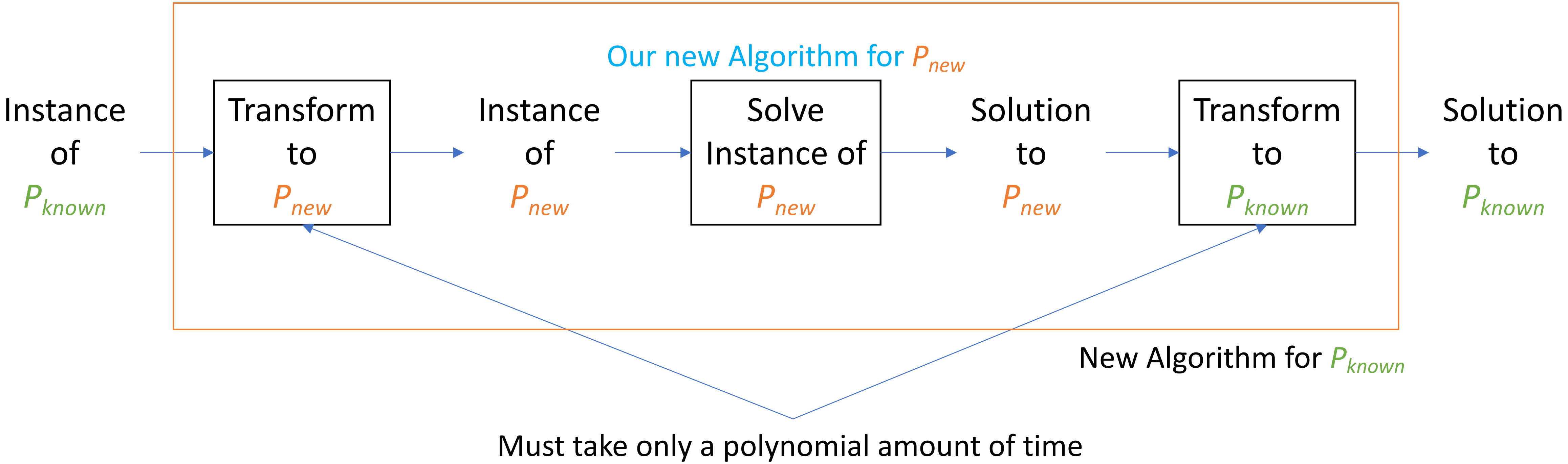
WE ALREADY PROVED THE CHARACTERISTICS OF P_{known} SO, WE MUST HAVE FOUND A NEW WAY TO IMPLEMENT THE SAME THING USING P_{new}

Prove two algorithms belong to the same class

P_{known} is the all-pairs shortest path problem

P_{new} is a new method for computing the shortest path from a start vertex to all other vertices

Reduce P_{known} to P_{new}



Examples of Reductions

Reduce median selection to sorting.

- Finding the median value of an array of numbers is as hard as sorting the number and sorting the number can be solved in polynomial-time.
- Note: finding the median turns our to be easier than comparison-based sorting (O(n))

Reduce cycle detection to DFS

- Detecting a cycle in a graph is as hard as performing a depth first search and DFS can be done in polynomial-time.
- This is related to Kruskal's minimum spanning tree algorithm and the union-find data structure

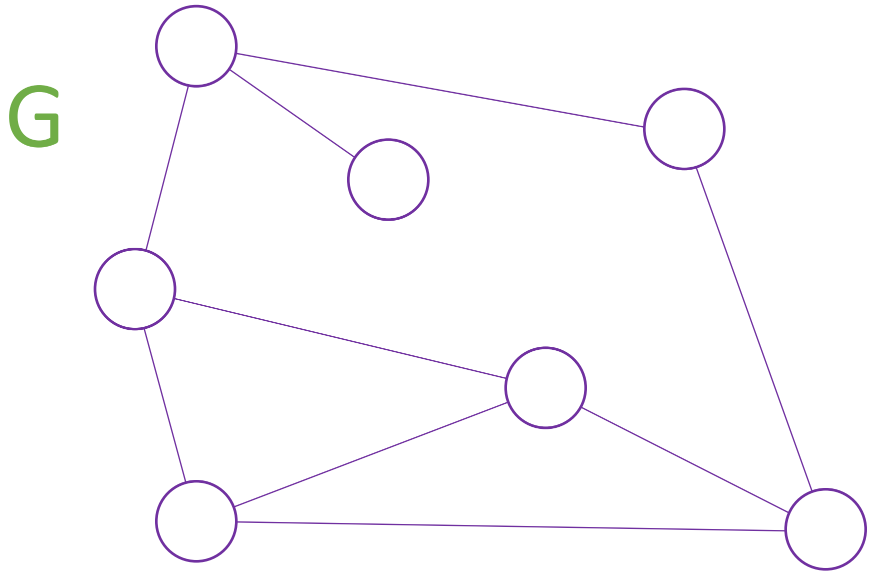
Reduce all pairs shortest path to single source shortest path

- Computing all pairs shortest paths is as hard as computing the shortest path from one node to every other node n times, which can be done in polynomial time.
- Invoke polynomial time algorithm "n times" is still polynomial time (just increase exponent by 1).

Full Reduction Example

The S-Independent Set Problem

- Given a graph G and a number S, is there a set of nodes of size S in G such that no two nodes in the set are directly connected in G (they are independent of each other)?

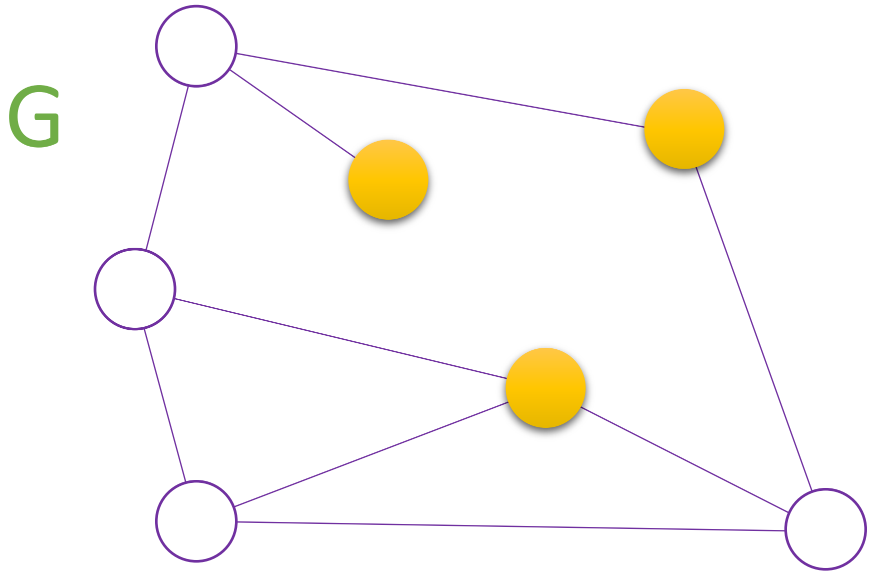


S = 3

Full Reduction Example

The S-Independent Set Problem

- Given a graph G and a number S, is there a set of nodes of size S in G such that no two nodes in the set are directly connected in G (they are independent of each other)?

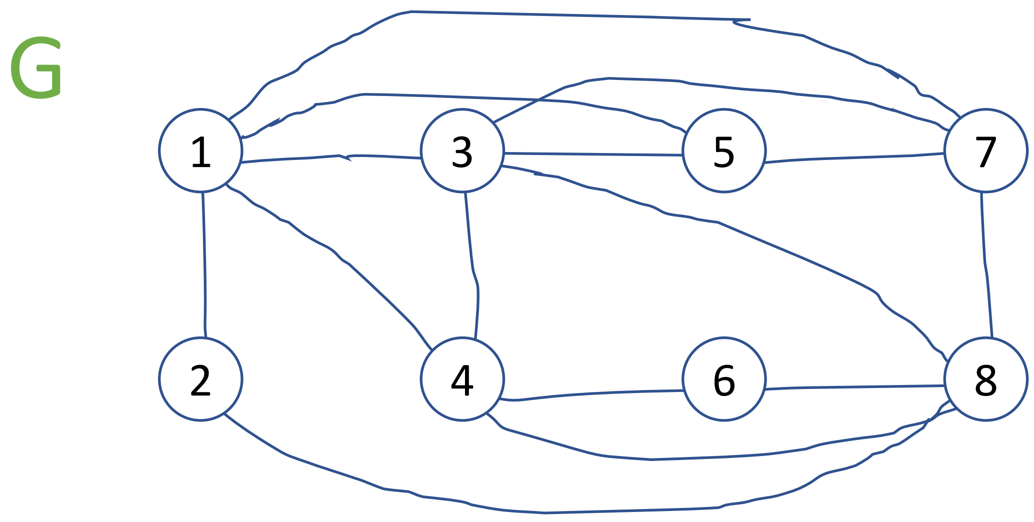


S = 3

Full Reduction Example

The k-Clique Problem

- Given a graph G and a number k, is there a set of nodes of size k in G such that all nodes are directly connected with one another?



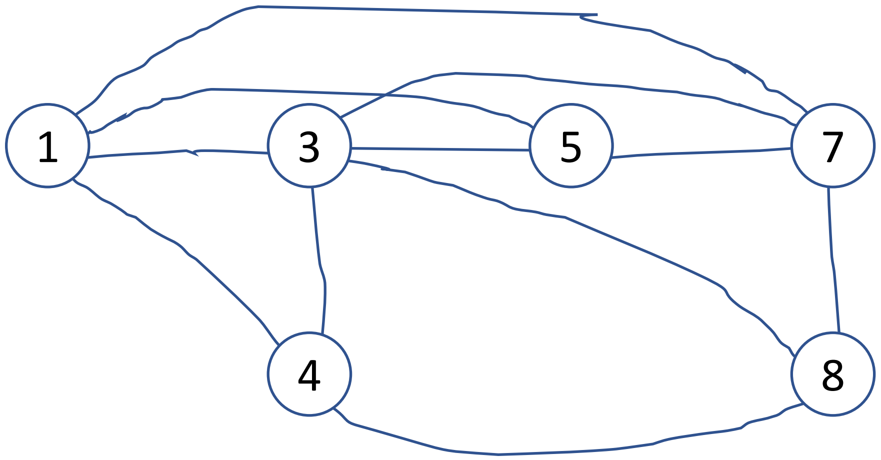
k = 4

Full Reduction Example

The k-Clique Problem

- Given a graph G and a number k, is there a set of nodes of size k in G such that all nodes are directly connected with one another?

G



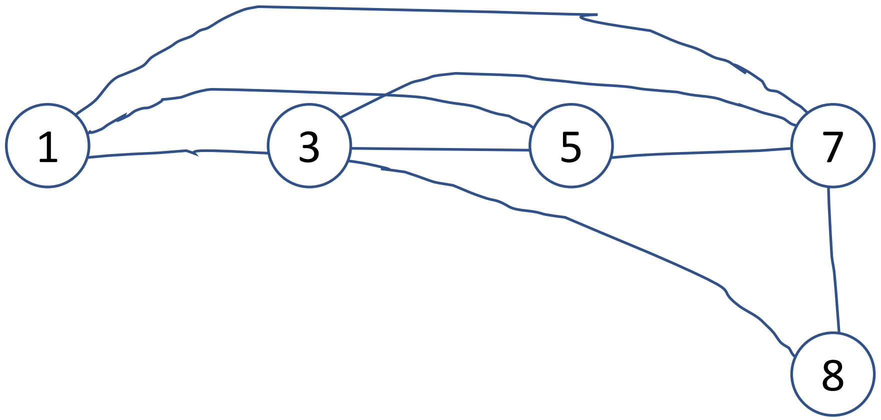
k = 4

Full Reduction Example

The k-Clique Problem

- Given a graph G and a number k, is there a set of nodes of size k in G such that all nodes are directly connected with one another?

G



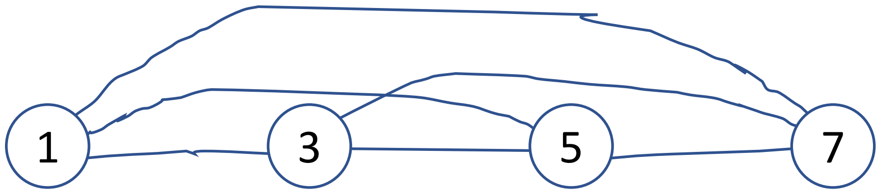
k = 4

Full Reduction Example

The k-Clique Problem

- Given a graph G and a number k, is there a set of nodes of size k in G such that all nodes are directly connected with one another?

G



k = 4

Full Reduction Example

The S-Independent Set Problem

- Given a graph G and a number S, is there a set of nodes of size S in G such that no two nodes in the set are directly connected in G (they are independent of each other)?

The k-Clique Problem

- Given a graph G and a number k, is there a set of nodes of size k in G such that all nodes are directly connected with one another?

Reduce S-Independent Set to k-Clique

Known

New

The S-Independent Set Problem

Given a graph G and a number S, is there a set of nodes of size S in G such that no two nodes in the set are directly connected in G?

The k-Clique Problem

Given a graph G and a number k, is there a set of nodes of size k in G such that all nodes are directly connected with one another?

We don’t know the computational classification of k-Clique.

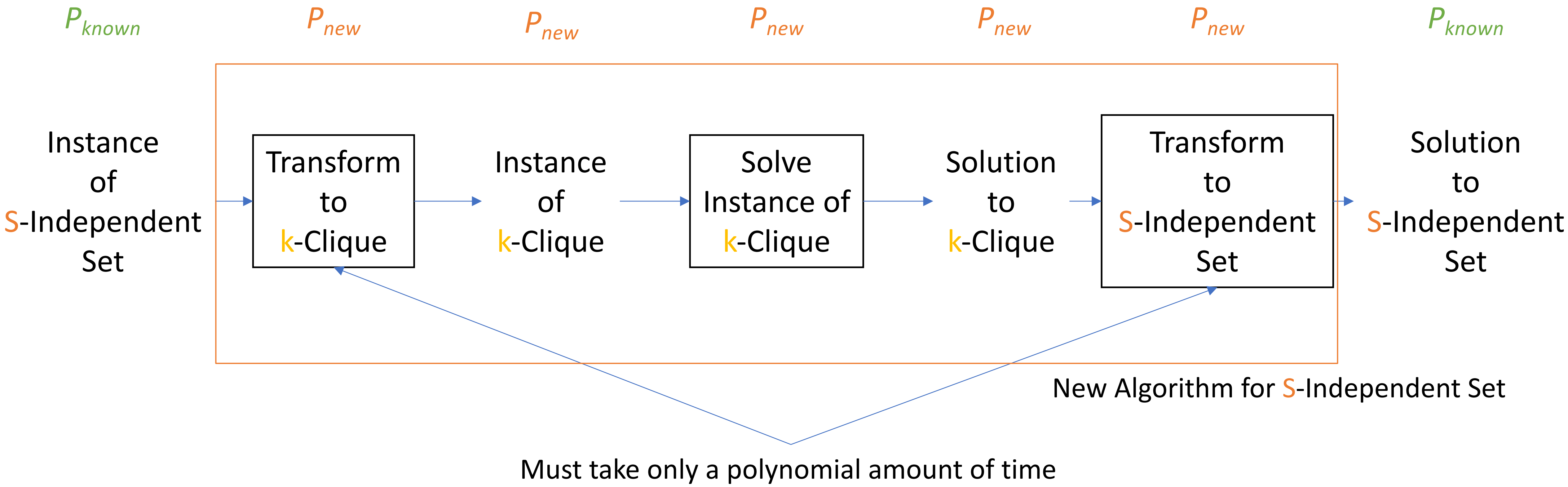
We **do** know the computational classification of S-Independent Set (NP-Complete).

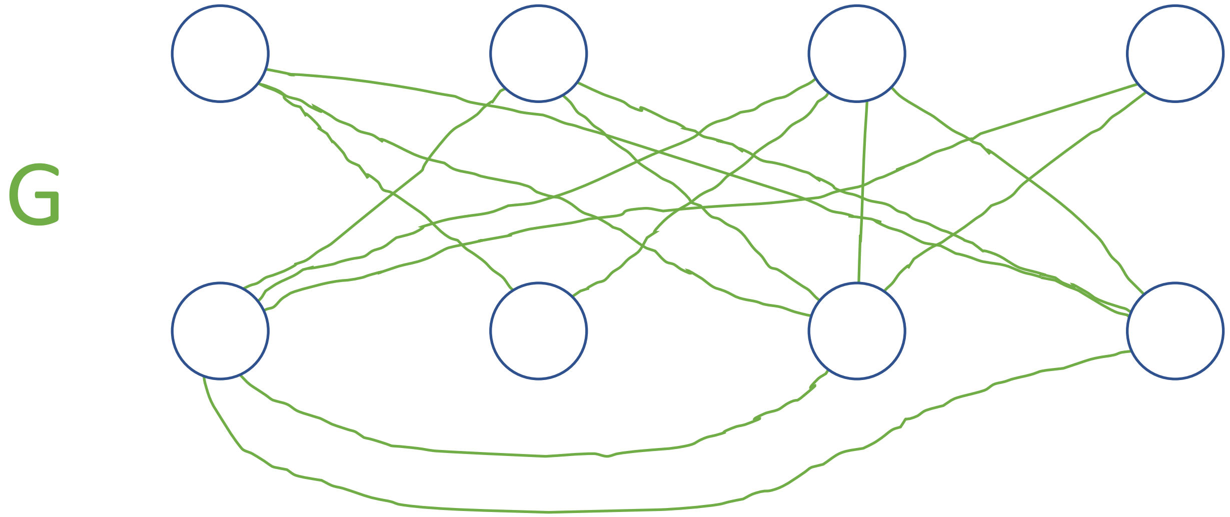
How do we use S-Independent Set to find the computational classification of k-Clique?

Reduce S-Independent Set to k-Clique.

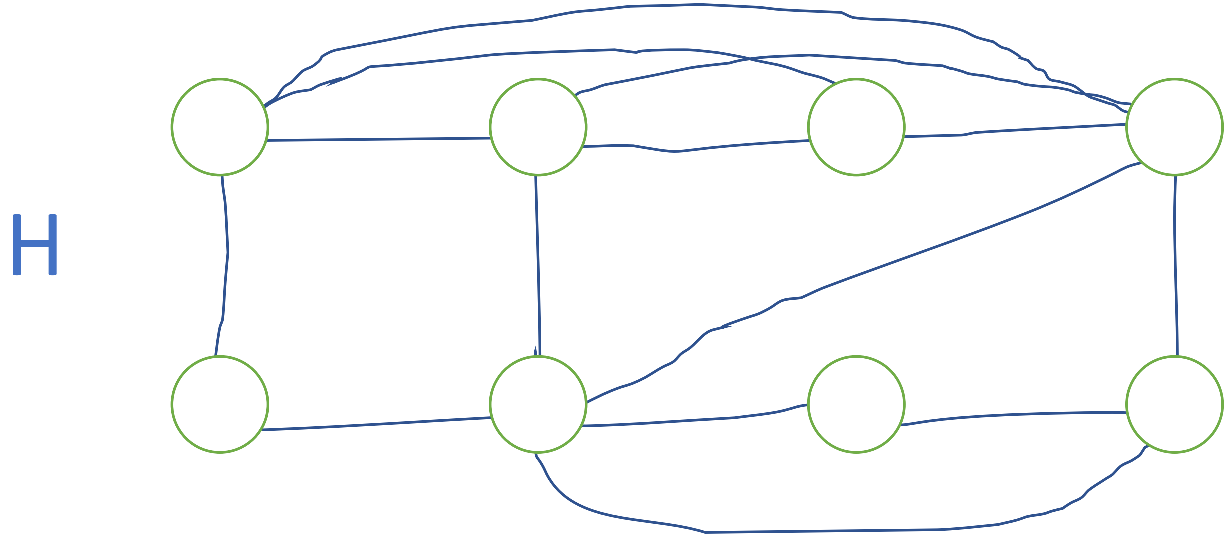
If we can perform the reduction, then k-Clique must be as hard as S-Independent Set.

Reduce S-Independent Set to k-Clique





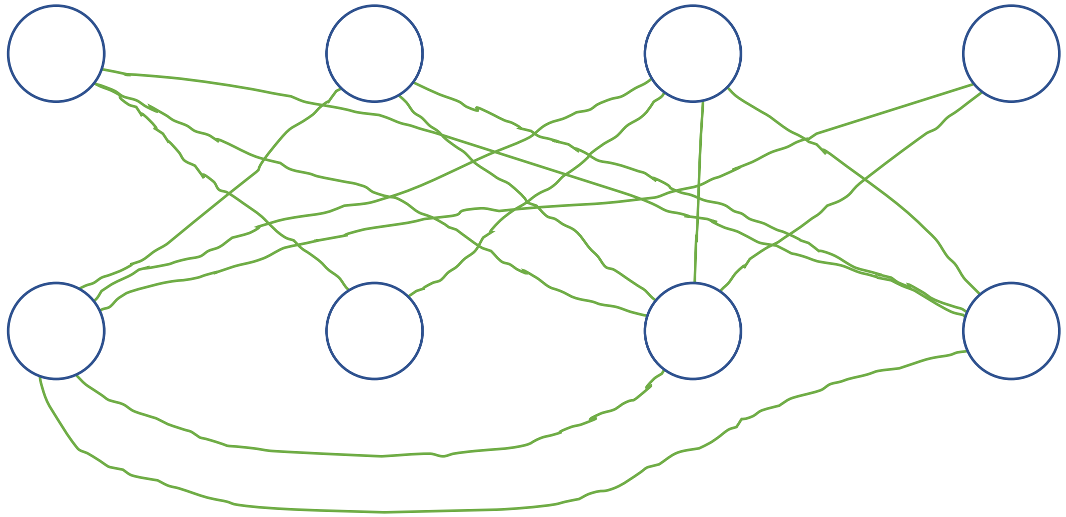
We want to find the S-Independent set of G



Let’s instead find the k-Clique of H. (k = S)

Where H is the complement of G.

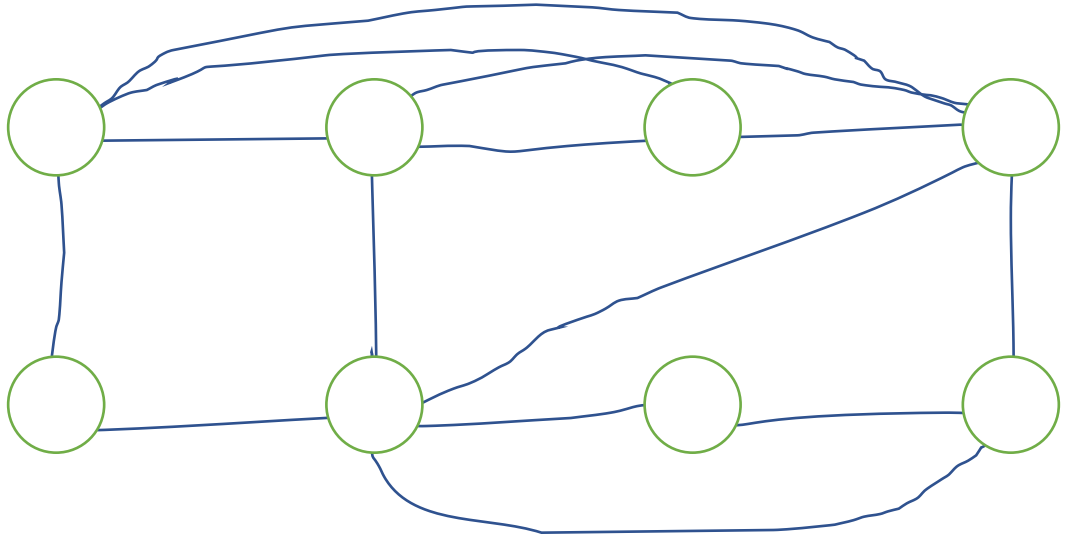
G



We want to find the S-Independent set of G

G has an S-Independent set if and only if H has a k-Clique (we're not going to prove this)

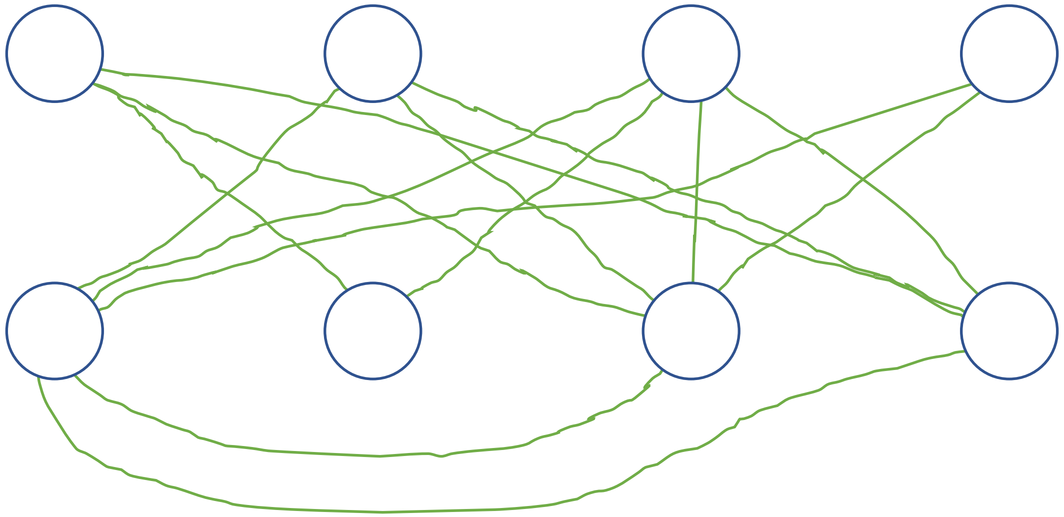
H



Let's instead find the k-Clique of H. (k = S)

Where H is the complement of G.

G

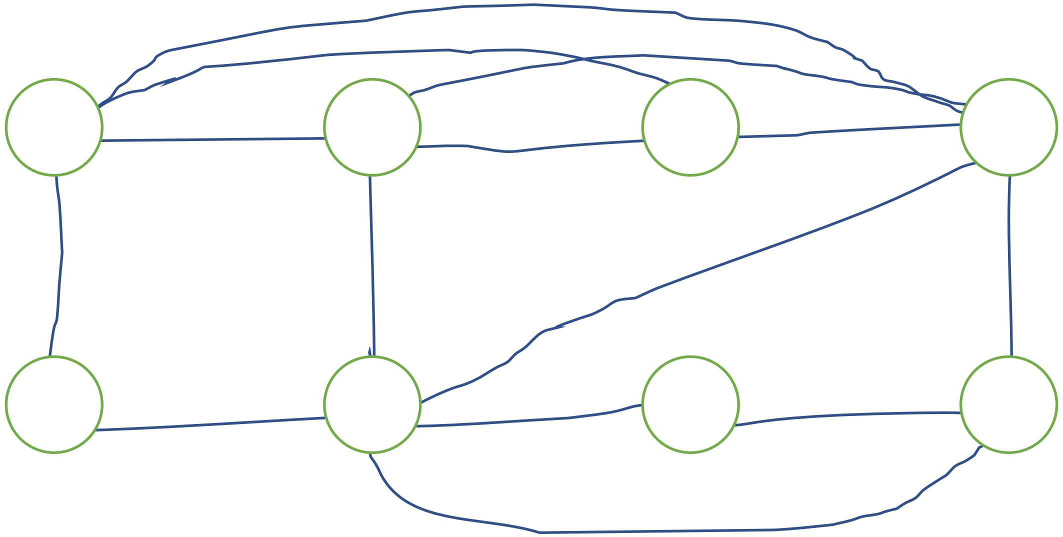


We want to find the S-Independent set of G

G has an S-Independent set if and only if H has a k-Clique (we’re not going to prove this)

Let S = 4, and thus k = 4

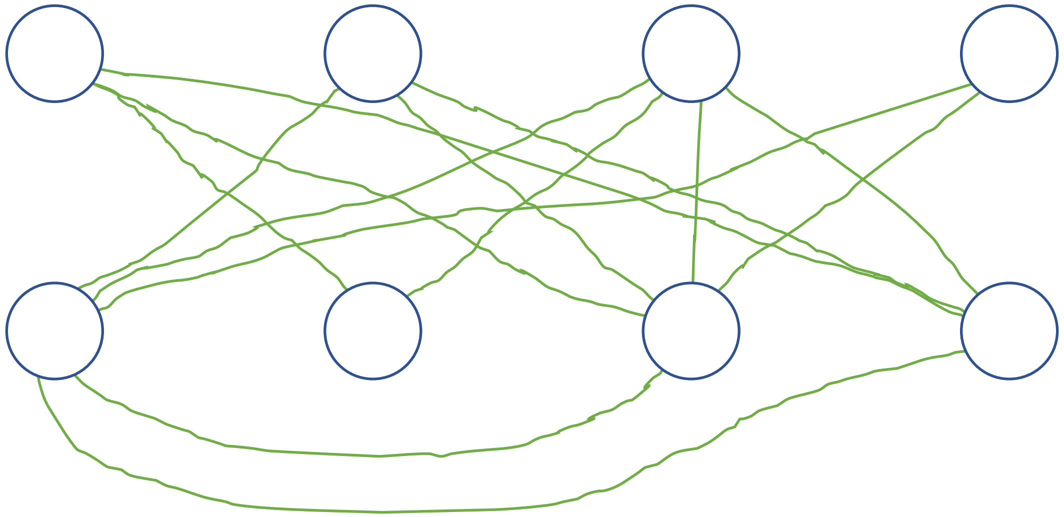
H



Let’s instead find the k-Clique of H. (k = S)

Where H is the complement of G.

G

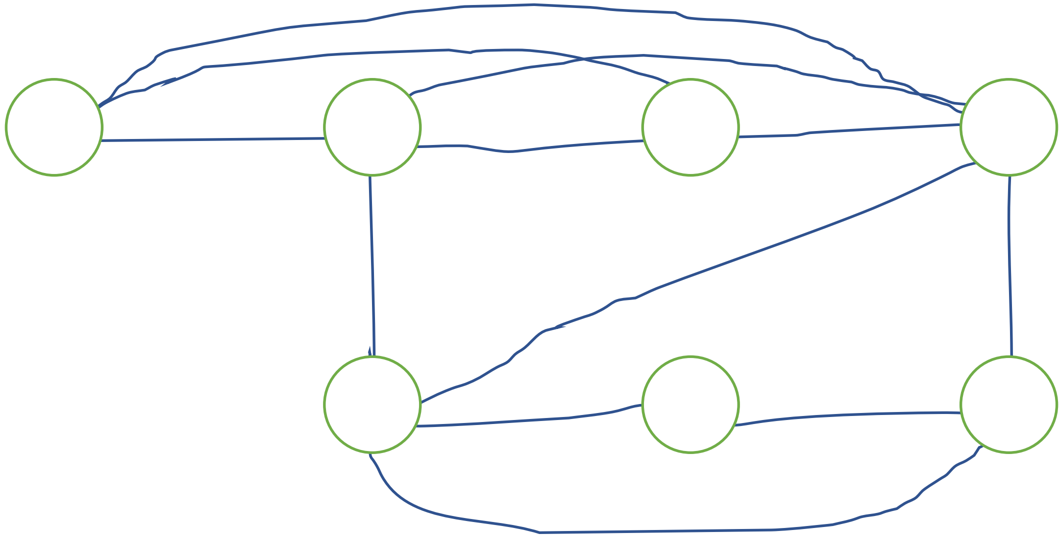


We want to find the S-Independent set of G

G has an S-Independent set if and only if H has a k-Clique (we're not going to prove this)

Let S = 4, and thus k = 4

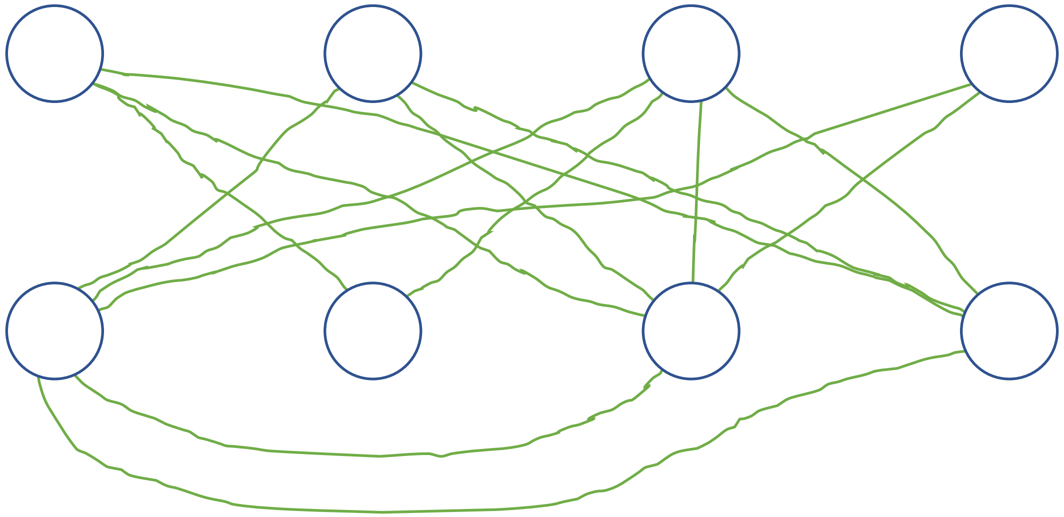
H



Let's instead find the k-Clique of H. (k = S)

Where H is the complement of G.

G

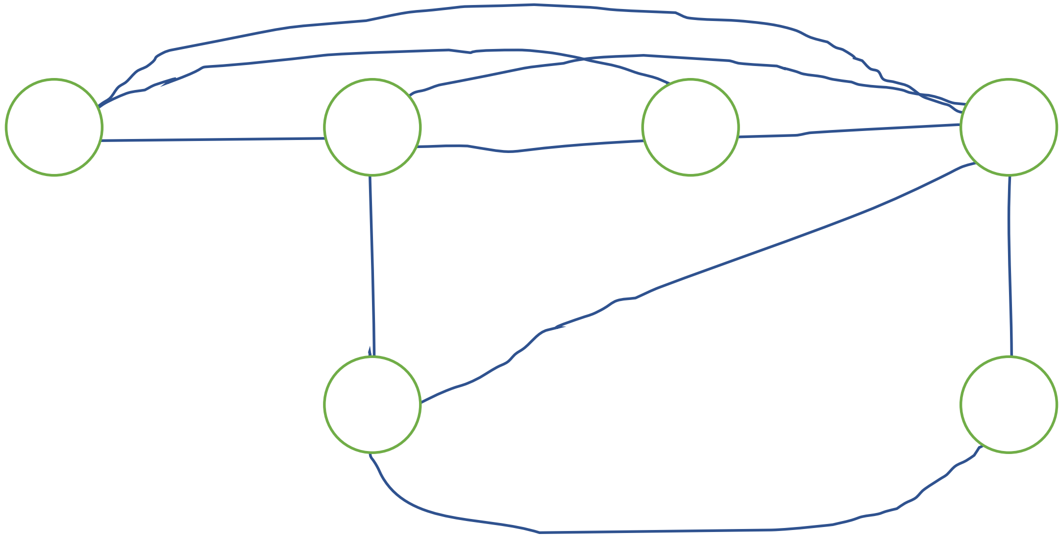


We want to find the S-Independent set of G

G has an S-Independent set if and only if H has a k-Clique (we’re not going to prove this)

Let S = 4, and thus k = 4

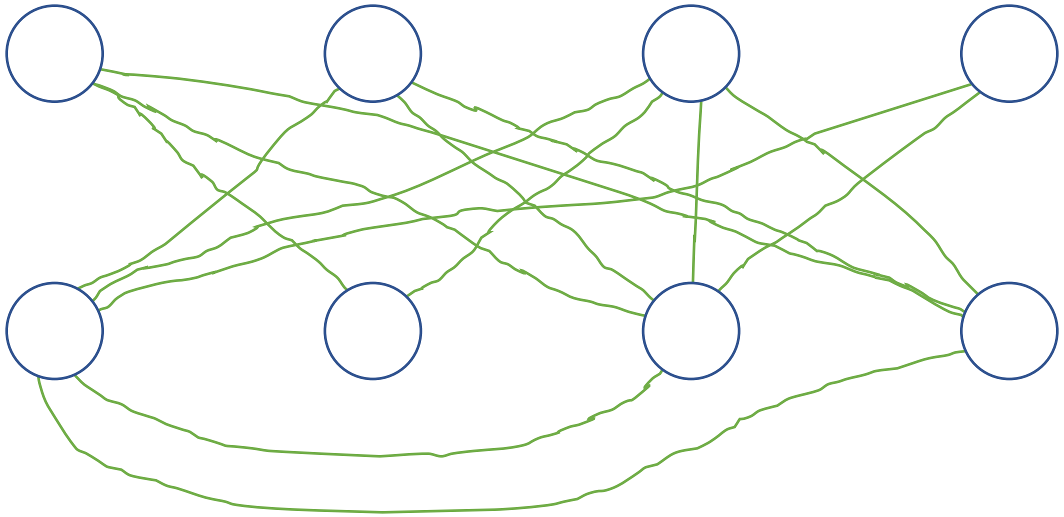
H



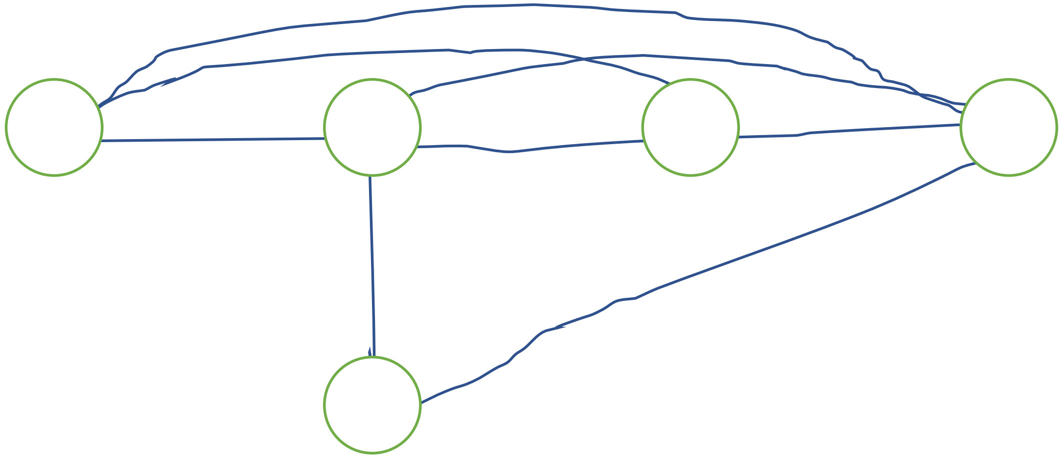
Let’s instead find the k-Clique of H. (k = S)

Where H is the complement of G.

G



Let S = 4, and thus k = 4



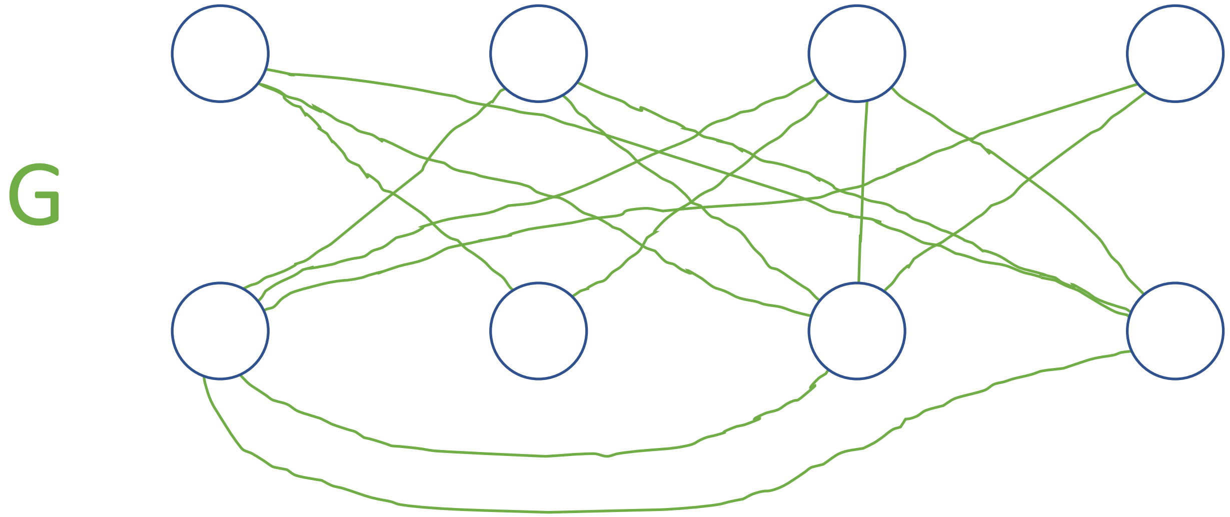
H

We want to find the S-Independent set of G

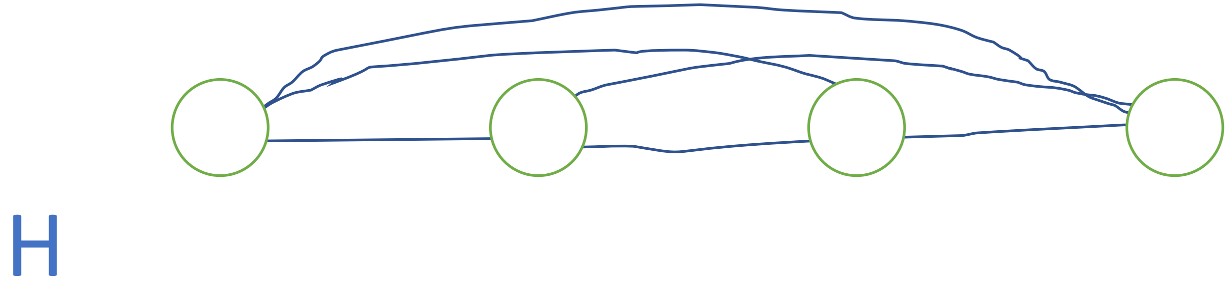
G has an S-Independent set if and only if H has a k-Clique (we're not going to prove this)

Let's instead find the k-Clique of H. (k = S)

Where H is the complement of G.



Let S = 4, and thus k = 4



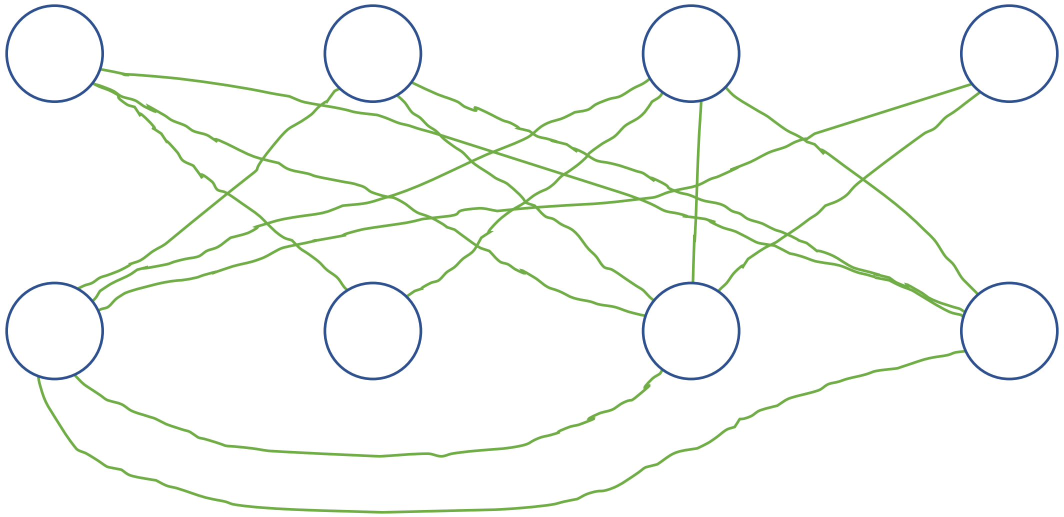
We want to find the S-Independent set of G

G has an S-Independent set if and only if H has a k-Clique (we're not going to prove this)

Let's instead find the k-Clique of H. (k = S)

Where H is the complement of G.

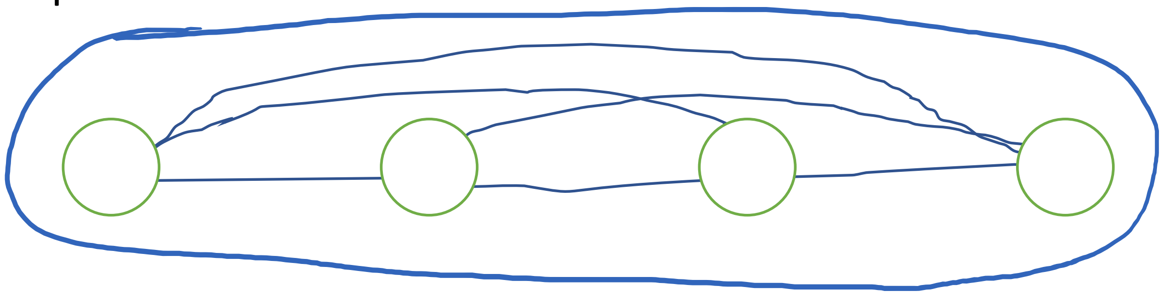
G



We want to find the S-Independent set of G

G has an S-Independent set if and only if H has a k-Clique (we're not going to prove this)

Let S = 4, and thus k = 4



H

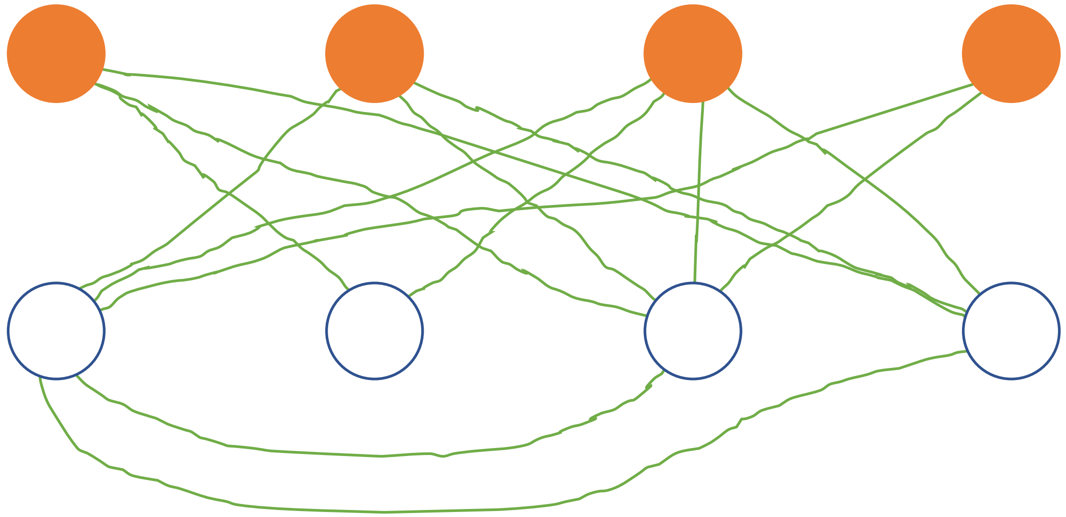
These 4 nodes comprise a size 4 clique of H; return **true**

Let's instead find the k-Clique of H. (k = S)

Where H is the complement of G.

These 4 nodes comprise a size 4 independent set of G; return **true**

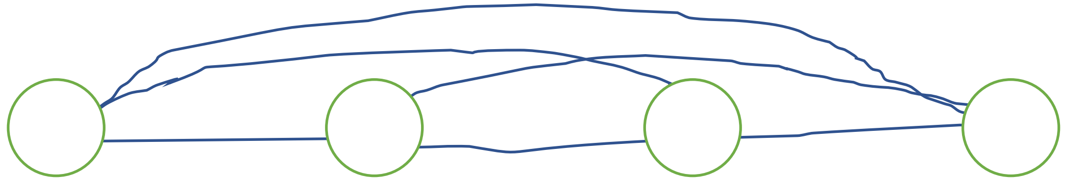
G



We want to find the S-Independent set of G

Let S = 4, and thus k = 4

G has an S-Independent set if and only if H has a k-Clique (we’re not going to prove this)



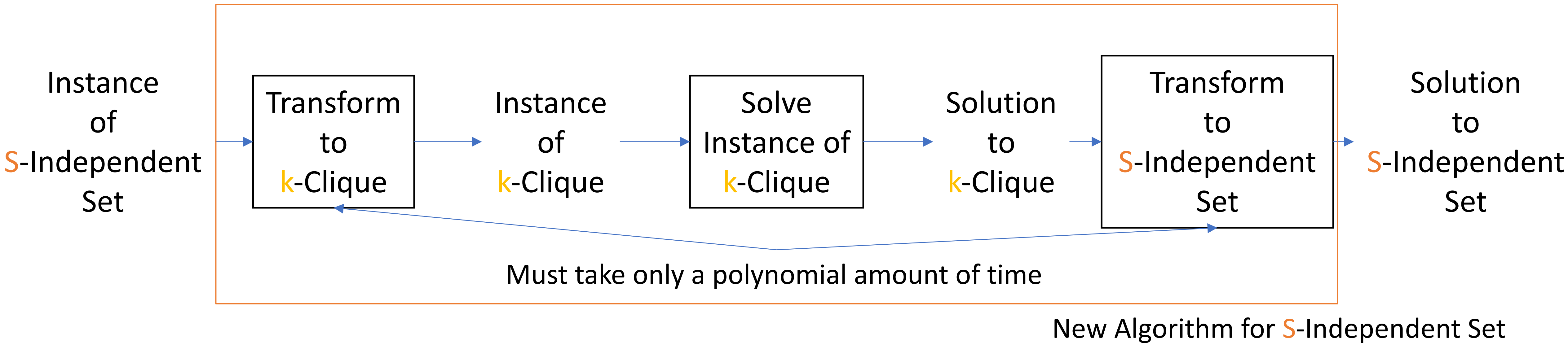
H

Let’s instead find the k-Clique of H. (k = S)

Where H is the complement of G.

These 4 nodes comprise a size 4 clique of H; return **true**

Reduce S-Independent Set to k-Clique



Since the S-Independent Set Problem can be reduced to the The k-Clique Problem, and the S-Independent Set Problem is NP-Complete, then the k-Clique Problem is also NP-Complete.

Reduce S-Independent Set to k-Clique

Known

The S-Independent Set Problem

Given a graph G and a number S, is there a set of nodes of size S in G such that no two nodes in the set are directly connected in G?

New

The k-Clique Problem

Given a graph G and a number k, is there a set of nodes of size k in G such that all nodes are directly connected with one another?

We don't know the computational classification of k-Clique.

We **do** know the computational classification of S-Independent Set (NP-Complete).

How do we use S-Independent Set to find the computational classification of k-Clique?

Reduce S-Independent Set to k-Clique.

If we can perform the reduction, then k-Clique must be as hard as S-Independent Set.

Proving a Problem X is NP-Complete

Effectively we are trying to say that X cannot be solved in $O(n^k)$ by any known process

1. First prove that X is in NP (it can be verified in polynomial time)
2. Next prove that X is NP-Hard
   1. Reduce some known NP-Complete or NP-Hard problem Y to X
   2. This implies that any and all NP-Complete problems can be reduced to X
   3. All NP-Complete problems have been reduced to another in an interconnected web (the original problem is known as 3SAT)

3-SAT Example