Reductions

Abhiram Ranade

March 22, 2016

The central questions in algorithm design

The central questions in algorithm design

Given a problem Q, can we design a fast algorithm for it? Can we prove that nothing better is possible?

The central questions in algorithm design

Given a problem Q, can we design a fast algorithm for it? Can we prove that nothing better is possible?

We know good algorithms for many problems, but even after 50 years of computer science research, we know little about proving "nothing better is possible".

The central questions in algorithm design

Given a problem Q, can we design a fast algorithm for it? Can we prove that nothing better is possible?

We know good algorithms for many problems, but even after 50 years of computer science research, we know little about proving "nothing better is possible".

Status for many combinatorial optimization problems, e.g. MIS, VC, ILP and others:

The central questions in algorithm design

Given a problem Q, can we design a fast algorithm for it? Can we prove that nothing better is possible?

We know good algorithms for many problems, but even after 50 years of computer science research, we know little about proving "nothing better is possible".

Status for many combinatorial optimization problems, e.g. MIS, VC, ILP and others:

- We do not know the "best algorithm",

The central questions in algorithm design

Given a problem Q, can we design a fast algorithm for it? Can we prove that nothing better is possible?

We know good algorithms for many problems, but even after 50 years of computer science research, we know little about proving "nothing better is possible".

Status for many combinatorial optimization problems, e.g. MIS, VC, ILP and others:

- We do not know the "best algorithm",
- We dont even know if a polynomial time algorithm exists, of however high degree.

The central questions in algorithm design

Given a problem Q, can we design a fast algorithm for it? Can we prove that nothing better is possible?

We know good algorithms for many problems, but even after 50 years of computer science research, we know little about proving "nothing better is possible".

Status for many combinatorial optimization problems, e.g. MIS, VC, ILP and others:

- We do not know the "best algorithm",
- We dont even know if a polynomial time algorithm exists, of however high degree.
- We have no proof that polytime algorithms do not exist.

The central questions in algorithm design

Given a problem Q, can we design a fast algorithm for it? Can we prove that nothing better is possible?

We know good algorithms for many problems, but even after 50 years of computer science research, we know little about proving "nothing better is possible".

Status for many combinatorial optimization problems, e.g. MIS, VC, ILP and others:

- We do not know the "best algorithm",
- We dont even know if a polynomial time algorithm exists, of however high degree.
- We have no proof that polytime algorithms do not exist.

However we can say something very interesting about these problems.

The central questions in algorithm design

Given a problem Q, can we design a fast algorithm for it? Can we prove that nothing better is possible?

We know good algorithms for many problems, but even after 50 years of computer science research, we know little about proving "nothing better is possible".

Status for many combinatorial optimization problems, e.g. MIS, VC, ILP and others:

- We do not know the "best algorithm",
- We dont even know if a polynomial time algorithm exists, of however high degree.
- We have no proof that polytime algorithms do not exist.

However we can say something very interesting about these problems.

The starting point for this is the notion of *reduction*.

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.

We state this process more formally.

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.
We state this process more formally.

Definition (Karp): Problem R is said to reduce to problem Q in polytime ($R \leq_K Q$) if $\exists$ polynomial time functions IM, SM such that for any instance x of R, $y = IM(x)$ is an instance of Q such that if z is a solution to y then $SM(z)$ is a solution to instance x.

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.
We state this process more formally.

Definition (Karp): Problem R is said to reduce to problem Q in polytime ($R \leq_K Q$) if $\exists$ polynomial time functions IM, SM such that for any instance x of R, $y = IM(x)$ is an instance of Q such that if z is a solution to y then $SM(z)$ is a solution to instance x.

IM : instance map, SM : solution map.

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.
We state this process more formally.

Definition (Karp): Problem R is said to reduce to problem Q in polytime ($R \leq_K Q$) if $\exists$ polynomial time functions IM, SM such that for any instance x of R, $y = IM(x)$ is an instance of Q such that if z is a solution to y then $SM(z)$ is a solution to instance x.

IM : instance map, SM : solution map.

Reduction is like translation.

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.
We state this process more formally.

Definition (Karp): Problem R is said to reduce to problem Q in polytime ($R \leq_K Q$) if $\exists$ polynomial time functions IM, SM such that for any instance x of R, $y = IM(x)$ is an instance of Q such that if z is a solution to y then $SM(z)$ is a solution to instance x.

IM : instance map, SM : solution map.

Reduction is like translation.

- A problem is a language in which you can ask questions (instances).

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.
We state this process more formally.

Definition (Karp): Problem R is said to reduce to problem Q in polytime ($R \leq_K Q$) if $\exists$ polynomial time functions IM, SM such that for any instance x of R, $y = IM(x)$ is an instance of Q such that if z is a solution to y then $SM(z)$ is a solution to instance x.

IM : instance map, SM : solution map.

Reduction is like translation.

- A problem is a language in which you can ask questions (instances).
- The function IM translates a question in one language into a question in another.

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.
We state this process more formally.

Definition (Karp): Problem R is said to reduce to problem Q in polytime ($R \leq_K Q$) if $\exists$ polynomial time functions IM, SM such that for any instance x of R, $y = IM(x)$ is an instance of Q such that if z is a solution to y then $SM(z)$ is a solution to instance x.

IM : instance map, SM : solution map.

Reduction is like translation.

- A problem is a language in which you can ask questions (instances).
- The function IM translates a question in one language into a question in another.
- The function SM translates the solution.

Reduction

We saw how problems such as Maximum Independent Set can be expressed using Integer Linear Programming.
We state this process more formally.

Definition (Karp): Problem R is said to reduce to problem Q in polytime ($R \leq_K Q$) if $\exists$ polynomial time functions IM, SM such that for any instance x of R, $y = IM(x)$ is an instance of Q such that if z is a solution to y then $SM(z)$ is a solution to instance x.

IM : instance map, SM : solution map.

Reduction is like translation.

- A problem is a language in which you can ask questions (instances).
- The function IM translates a question in one language into a question in another.
- The function SM translates the solution.
- We want translation to happen fast, so we demand polynomial time for f, g.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof:

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.
Algorithm for R, where instance $= x$ is

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$

Time for step 1: $O(|x|^k)$ for some constant k.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$
Time for step 1: $O(|x|^k)$ for some constant k.
Time for step 2: $O(|y|^{k'})$ for some constant k'.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$
Time for step 1: $O(|x|^k)$ for some constant k.
Time for step 2: $O(|y|^{k'})$ for some constant k'.
Time for step 3: $O(|z|^{k''})$ for some constant k''.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$
Time for step 1: $O(|x|^k)$ for some constant k.
Time for step 2: $O(|y|^{k'})$ for some constant k'.
Time for step 3: $O(|z|^{k''})$ for some constant k''.

An algorithm takes time $t \Rightarrow$, the length of its output $\leq t$.

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.
Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$
Time for step 1: $O(|x|^k)$ for some constant k.
Time for step 2: $O(|y|^{k'})$ for some constant k'.
Time for step 3: $O(|z|^{k''})$ for some constant k''.

An algorithm takes time $t \Rightarrow$, the length of its output $\leq t$.
Thus $|y| = O(|x|^k)$,

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$
Time for step 1: $O(|x|^k)$ for some constant k.
Time for step 2: $O(|y|^{k'})$ for some constant k'.
Time for step 3: $O(|z|^{k''})$ for some constant k''.

An algorithm takes time $t \Rightarrow$, the length of its output $\leq t$.
Thus $|y| = O(|x|^k)$, $|z| = O(|y|^{k'}) = O(|x|^{kk'})$,

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$
Time for step 1: $O(|x|^k)$ for some constant k.
Time for step 2: $O(|y|^{k'})$ for some constant k'.
Time for step 3: $O(|z|^{k''})$ for some constant k''.

An algorithm takes time $t \Rightarrow$, the length of its output $\leq t$.
Thus $|y| = O(|x|^k)$, $|z| = O(|y|^{k'}) = O(|x|^{kk'})$,
Time $= O(|x|^k) + O(|y|^{k'}) + O(|z|^{k''})$

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.
Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$
Time for step 1: $O(|x|^k)$ for some constant k.
Time for step 2: $O(|y|^{k'})$ for some constant k'.
Time for step 3: $O(|z|^{k''})$ for some constant k''.

An algorithm takes time $t \Rightarrow$, the length of its output $\leq t$.
Thus $|y| = O(|x|^k)$, $|z| = O(|y|^{k'}) = O(|x|^{kk'})$,
Time $= O(|x|^k) + O(|y|^{k'}) + O(|z|^{k''}) = O(|x|^k + |x|^{kk'} + |x|^{kk'k''})$

Key Property of $\leq_K$

Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Proof: Let IM, SM be the instance map, solution map of the reduction.

Algorithm for R, where instance $= x$ is

1. Compute $y = IM(x)$.
2. Compute $z =$ solution to y using A.
3. Compute and return $SM(z)$.

IM, SM, A take time polynomial in lengths of their arguments $\Rightarrow$
Time for step 1: $O(|x|^k)$ for some constant k.
Time for step 2: $O(|y|^{k'})$ for some constant k'.
Time for step 3: $O(|z|^{k''})$ for some constant k''.

An algorithm takes time $t \Rightarrow$, the length of its output $\leq t$.
Thus $|y| = O(|x|^k)$, $|z| = O(|y|^{k'}) = O(|x|^{kk'})$,
Time $= O(|x|^k) + O(|y|^{k'}) + O(|z|^{k''}) = O(|x|^k + |x|^{kk'} + |x|^{kk'k''})$
Thus the total time is polynomial in $|x|$.

Implications of the key property

Implications of the key property

Key Property: Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Implications of the key property

Key Property: Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Contrapositive: If $R \leq_K Q$, then if no polytime algorithm exists for R then no polytime algorithm exists for Q.

Implications of the key property

Key Property: Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Contrapositive: If $R \leq_K Q$, then if no polytime algorithm exists for R then no polytime algorithm exists for Q.

Informal Implications:

Implications of the key property

Key Property: Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Contrapositive: If $R \leq_K Q$, then if no polytime algorithm exists for R then no polytime algorithm exists for Q.

Informal Implications:

- Difficulty of designing polytime algorithm for $R \leq$ Difficulty of designing polytime algorithm for Q.

Implications of the key property

Key Property: Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Contrapositive: If $R \leq_K Q$, then if no polytime algorithm exists for R then no polytime algorithm exists for Q.

Informal Implications:

- Difficulty of designing polytime algorithm for $R \leq$ Difficulty of designing polytime algorithm for Q.
- Difficulty of proving impossibility of designing polytime algorithms for $R \geq$ Difficulty of proving impossibility of designing polytime algorithms for Q.

Implications of the key property

Key Property: Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Contrapositive: If $R \leq_K Q$, then if no polytime algorithm exists for R then no polytime algorithm exists for Q.

Informal Implications:

- Difficulty of designing polytime algorithm for $R \leq$ Difficulty of designing polytime algorithm for Q.
- Difficulty of proving impossibility of designing polytime algorithms for $R \geq$ Difficulty of proving impossibility of designing polytime algorithms for Q.

Reductions help us in solving hard problems practically, e.g. reducing MIS, VC, timetabling to ILP.

Implications of the key property

Key Property: Suppose $R \leq_K Q$, and suppose we have an algorithm A for solving Q in polytime. Then there exists a polytime algorithm for R.

Contrapositive: If $R \leq_K Q$, then if no polytime algorithm exists for R then no polytime algorithm exists for Q.

Informal Implications:

- Difficulty of designing polytime algorithm for $R \leq$ Difficulty of designing polytime algorithm for Q.
- Difficulty of proving impossibility of designing polytime algorithms for $R \geq$ Difficulty of proving impossibility of designing polytime algorithms for Q.

Reductions help us in solving hard problems practically, e.g. reducing MIS, VC, timetabling to ILP.

Reductions help us compare problems from the point of view of designing polytime algorithms.

Exercises

1. Review what we did for expressing MIS as ILP, and show that this indeed establishes that MIS $\leq_K$ ILP. What are IM, SM?
2. Show that Knapsack $\leq_K$ ILP.
3. Suppose the instance map IM is used in the reduction from problem R to Problem Q. Suppose instances x, x' of R have different answers. Show that $IM(x) \neq IM(x')$. In other words, the instance map can be many-to-one but it should only map instances with the same solution in R to the same instance of Q.

Agenda for next few weeks

Agenda for next few weeks

Goal: Compare the relative difficulty of different problems from the point of view of designing polytime algorithms.

Agenda for next few weeks

Goal: Compare the relative difficulty of different problems from the point of view of designing polytime algorithms.

Method: Prove reductions between different problems.

Agenda for next few weeks

Goal: Compare the relative difficulty of different problems from the point of view of designing polytime algorithms.

Method: Prove reductions between different problems.

Transitivity: $S \leq_K R, R \leq_K Q \Rightarrow S \leq_K Q$

Agenda for next few weeks

Goal: Compare the relative difficulty of different problems from the point of view of designing polytime algorithms.

Method: Prove reductions between different problems.

Transitivity: $S \leq_K R, R \leq_K Q \Rightarrow S \leq_K Q$

Note: If $R \leq_K Q$ and $Q \leq_K R$ then R, Q are equally difficult, or equivalent from the point of view of designing polytime algorithms.

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

- Maximum Independent Set, Minimum Vertex Cover

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

- Maximum Independent Set, Minimum Vertex Cover
- Integer Linear Programming

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

- Maximum Independent Set, Minimum Vertex Cover
- Integer Linear Programming
  We have already proved $MIS, VC \leq_K ILP$. Converse soon.

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

- Maximum Independent Set, Minimum Vertex Cover
- Integer Linear Programming
  We have already proved $MIS, VC \leq_K ILP$. Converse soon.
- Knapsack

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

- Maximum Independent Set, Minimum Vertex Cover
- Integer Linear Programming
  We have already proved $MIS, VC \leq_K ILP$. Converse soon.
- Knapsack
- Determining whether a circuit design is correct.

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

- Maximum Independent Set, Minimum Vertex Cover
- Integer Linear Programming
  We have already proved $MIS, VC \leq_K ILP$. Converse soon.
- Knapsack
- Determining whether a circuit design is correct.
- Graph colouring, "Travelling Salesman Problem", ...

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

- Maximum Independent Set, Minimum Vertex Cover
- Integer Linear Programming
  We have already proved $MIS, VC \leq_K ILP$. Converse soon.
- Knapsack
- Determining whether a circuit design is correct.
- Graph colouring, "Travelling Salesman Problem", ...

Either all have polytime algorithms, or none do.

Looking ahead:

We will show that several apparently very different problems such as the following are equivalent from the point of view of designing polytime algorithms

- Maximum Independent Set, Minimum Vertex Cover
- Integer Linear Programming
  We have already proved $MIS, VC \leq_K ILP$. Converse soon.
- Knapsack
- Determining whether a circuit design is correct.
- Graph colouring, "Travelling Salesman Problem", ...

Either all have polytime algorithms, or none do.

Unfortunately, we do not know which!

Some more looking ahead..

Some more looking ahead..

These problems will constitute the class of NP-complete problems.

Some more looking ahead..

These problems will constitute the class of NP-complete problems.
Formal definition, explanation of the name, soon

Some more looking ahead..

These problems will constitute the class of NP-complete problems. Formal definition, explanation of the name, soon

Note: In spite of 30-40 years of research we have not been able to find polytime algorithms for these problems, nor prove that such algorithms do not exist.

Some more looking ahead..

These problems will constitute the class of NP-complete problems.
Formal definition, explanation of the name, soon

Note: In spite of 30-40 years of research we have not been able to find polytime algorithms for these problems, nor prove that such algorithms do not exist.

Current belief:

Some more looking ahead..

These problems will constitute the class of NP-complete problems.
Formal definition, explanation of the name, soon

Note: In spite of 30-40 years of research we have not been able to find polytime algorithms for these problems, nor prove that such algorithms do not exist.

Current belief: "So many people have tried to design polytime algorithms for ILP, so many for MIS, so many for TSP, and none have succeeded so far, so very likely no algorithm exists."

Some more looking ahead..

These problems will constitute the class of NP-complete problems. Formal definition, explanation of the name, soon

Note: In spite of 30-40 years of research we have not been able to find polytime algorithms for these problems, nor prove that such algorithms do not exist.

Current belief: "So many people have tried to design polytime algorithms for ILP, so many for MIS, so many for TSP, and none have succeeded so far, so very likely no algorithm exists."

Practical implication: If you are called upon to design a polytime algorithm for a problem Q, check if $Q \in NPC$. If so, you are probably wasting your time, attempting something on which many famous people have failed.

Some more looking ahead..

These problems will constitute the class of NP-complete problems. Formal definition, explanation of the name, soon

Note: In spite of 30-40 years of research we have not been able to find polytime algorithms for these problems, nor prove that such algorithms do not exist.

Current belief: "So many people have tried to design polytime algorithms for ILP, so many for MIS, so many for TSP, and none have succeeded so far, so very likely no algorithm exists."

Practical implication: If you are called upon to design a polytime algorithm for a problem Q, check if $Q \in NPC$. If so, you are probably wasting your time, attempting something on which many famous people have failed.

Perhaps better to go for exponential time algorithms, approximation algorithms...

Some more looking ahead..

These problems will constitute the class of NP-complete problems.
Formal definition, explanation of the name, soon

Note: In spite of 30-40 years of research we have not been able to find polytime algorithms for these problems, nor prove that such algorithms do not exist.

Current belief: "So many people have tried to design polytime algorithms for ILP, so many for MIS, so many for TSP, and none have succeeded so far, so very likely no algorithm exists."

Practical implication: If you are called upon to design a polytime algorithm for a problem Q, check if $Q \in NPC$. If so, you are probably wasting your time, attempting something on which many famous people have failed.

Perhaps better to go for exponential time algorithms, approximation algorithms...
Hence know NP-completeness!

Technicalities 0: Some notation

If R is a problem, and x an instance of R, then we will write $R(x)$ to denote the solution of x in R.

Technicalities 0: Some notation

If R is a problem, and x an instance of R, then we will write $R(x)$ to denote the solution of x in R.

We might occasionally problem $R(x)$ to mean "R is a problem and x is its instance".

Technicalities 0: Some notation

If R is a problem, and x an instance of R, then we will write $R(x)$ to denote the solution of x in R.

We might occasionally problem $R(x)$ to mean "R is a problem and x is its instance".

This is similar to how we declare functions in programming $f(x, y, z)$ might mean the signature of the function, as well as the value returned when arguments are x, y, z.

Technicalities 1: The class P

$P =$ class of problems that can be solved in polynomial time.

Technicalities 1: The class P

$P =$ class of problems that can be solved in polynomial time.

"Problem Q has a polytime algorithm" $\equiv Q \in P$

Technicalities 1: The class P

$P =$ class of problems that can be solved in polynomial time.

"Problem Q has a polytime algorithm" $\equiv Q \in P$

Sorting $\in P$

Technicalities 1: The class P

$P =$ class of problems that can be solved in polynomial time.

"Problem Q has a polytime algorithm" $\equiv Q \in P$

Sorting $\in P$

We wish to know if $MIS, ILP, \ldots \in P$

Technicalities 2: Other definitions of reduction

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.
Note: "Additional work" includes copying arguments to and results back from the Algorithm for Q.

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.
Note: "Additional work" includes copying arguments to and results back from the Algorithm for Q.

Note: Cook reducibility has key property

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.
Note: "Additional work" includes copying arguments to and results back from the Algorithm for Q.

Note: Cook reducibility has key property
Proof: Let $x =$ instance of R. Additional work $=$ poly($|x|$).

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.
Note: "Additional work" includes copying arguments to and results back from the Algorithm for Q.

Note: Cook reducibility has key property
Proof: Let $x =$ instance of R. Additional work $=$ poly($|x|$).
$Q \in P \Rightarrow$ Time(Algorithm for Q) $=$ poly in argument size.

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.
Note: "Additional work" includes copying arguments to and results back from the Algorithm for Q.

Note: Cook reducibility has key property
Proof: Let $x =$ instance of R. Additional work $=$ poly($|x|$).
$Q \in P \Rightarrow$ Time(Algorithm for Q) $=$ poly in argument size.
Arguments passed to Algo for Q : x or what is generated in "additional work".

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.
Note: "Additional work" includes copying arguments to and results back from the Algorithm for Q.

Note: Cook reducibility has key property
Proof: Let $x =$ instance of R. Additional work = poly($|x|$).
$Q \in P \Rightarrow$ Time(Algorithm for Q) = poly in argument size.
Arguments passed to Algo for Q : x or what is generated in "additional work". $\Rightarrow$ Argument Length = poly($|x|$).

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.
Note: "Additional work" includes copying arguments to and results back from the Algorithm for Q.

Note: Cook reducibility has key property
Proof: Let $x =$ instance of R. Additional work $=$ poly($|x|$).
$Q \in P \Rightarrow$ Time(Algorithm for Q) $=$ poly in argument size.
Arguments passed to Algo for Q : x or what is generated in "additional work". $\Rightarrow$ Argument Length $=$ poly($|x|$).
Hence total time $=$ poly($|x|$).

Technicalities 2: Other definitions of reduction

Our main motivation for defining reduction is its key property:
"$R \leq_K Q \Rightarrow$ If $Q \in P$ then $R \in P$"

There may be other notions of reducibility that have this property.

Cook Reducibility: $R \leq_C Q$ iff R can be solved by calling an algorithm for Q polynomial number of times, and doing polytime additional work.
Note: "Additional work" includes copying arguments to and results back from the Algorithm for Q.

Note: Cook reducibility has key property
Proof: Let x = instance of R. Additional work = poly($|x|$).
$Q \in P \Rightarrow$ Time(Algorithm for Q) = poly in argument size.
Arguments passed to Algo for Q : x or what is generated in "additional work". $\Rightarrow$ Argument Length = poly($|x|$).
Hence total time = poly($|x|$).

Cook reduction is also transitive.

Technicality 3: Decision problems

Decision problems are problems which have an answer YES or NO.

Technicality 3: Decision problems

Decision problems are problems which have an answer YES or NO.

The theory of NP-completeness focusses on decision problems because they are simple to discuss.

Technicality 3: Decision problems

Decision problems are problems which have an answer YES or NO.

The theory of NP-completeness focusses on decision problems because they are simple to discuss.

Decision problems are still hard enough!

Technicality 3: Decision problems

Decision problems are problems which have an answer YES or NO.

The theory of NP-completeness focusses on decision problems because they are simple to discuss.

Decision problems are still hard enough!
Next.

Optimization, existence, construction problems

Optimization, existence, construction problems

MIS:
Optimization: Find maximum independent set in G

Optimization, existence, construction problems

MIS:
Optimization: Find maximum independent set in G
Construction: Find an independent set of size k (if any).

k: additional input

Optimization, existence, construction problems

MIS:

Optimization: Find maximum independent set in G

Construction: Find an independent set of size k (if any).

k: additional input

Existence: Does there exist an independent set of size k?

Optimization, existence, construction problems

MIS:
Optimization: Find maximum independent set in G
Construction: Find an independent set of size k (if any).

k: additional input

Existence: Does there exist an independent set of size k?

Existence problem is a decision problem. Is it easier for designing polytime algorithms?

Optimization, existence, construction problems

MIS:
Optimization: Find maximum independent set in G
Construction: Find an independent set of size k (if any).
k: additional input
Existence: Does there exist an independent set of size k?

Existence problem is a decision problem. Is it easier for designing polytime algorithms?

Thm: MIS-optimization $\leq_C$ MIS-existence

Optimization, existence, construction problems

MIS:
Optimization: Find maximum independent set in G
Construction: Find an independent set of size k (if any).

k: additional input

Existence: Does there exist an independent set of size k?

Existence problem is a decision problem. Is it easier for designing polytime algorithms?

Thm: MIS-optimization $\leq_C$ MIS-existence

Next

Optimization, existence, construction problems

MIS:

Optimization: Find maximum independent set in G

Construction: Find an independent set of size k (if any).

k: additional input

Existence: Does there exist an independent set of size k?

Existence problem is a decision problem. Is it easier for designing polytime algorithms?

Thm: MIS-optimization $\leq_C$ MIS-existence Next

Note:

Optimization, existence, construction problems

MIS:
Optimization: Find maximum independent set in G
Construction: Find an independent set of size k (if any).

k: additional input

Existence: Does there exist an independent set of size k?

Existence problem is a decision problem. Is it easier for designing polytime algorithms?

Thm: MIS-optimization $\leq_C$ MIS-existence Next

Note:

- MIS-construction $\leq_K$ MIS-optimization

Optimization, existence, construction problems

MIS:

Optimization: Find maximum independent set in G

Construction: Find an independent set of size k (if any).

k: additional input

Existence: Does there exist an independent set of size k?

Existence problem is a decision problem. Is it easier for designing polytime algorithms?

Thm: MIS-optimization $\leq_C$ MIS-existence Next

Note:

- MIS-construction $\leq_K$ MIS-optimization

  Construct optimal, return only k of the vertices

Optimization, existence, construction problems

MIS:
Optimization: Find maximum independent set in G
Construction: Find an independent set of size k (if any).
k: additional input
Existence: Does there exist an independent set of size k?

Existence problem is a decision problem. Is it easier for designing polytime algorithms?

Thm: MIS-optimization $\leq_C$ MIS-existence Next

Note:

- MIS-construction $\leq_K$ MIS-optimization
  Construct optimal, return only k of the vertices
- MIS-existence $\leq_K$ MIS-construction

Optimization, existence, construction problems

MIS:
Optimization: Find maximum independent set in G
Construction: Find an independent set of size k (if any).

k: additional input

Existence: Does there exist an independent set of size k?

Existence problem is a decision problem. Is it easier for designing polytime algorithms?

Thm: MIS-optimization $\leq_C$ MIS-existence Next

Note:

- MIS-construction $\leq_K$ MIS-optimization
  Construct optimal, return only k of the vertices
- MIS-existence $\leq_K$ MIS-construction
  Say YES only if construction algorithm returns a solution.

MIS-optimization $\leq_C$ MIS-construction

MIS-optimization $\leq_C$ MIS-construction

Proof:

Algorithm MIS-optimization(G){

MIS-optimization $\leq_C$ MIS-construction

Proof:

Algorithm MIS-optimization(G){

For $k = 1$ to number of vertices call MIS-construction(G,k).

MIS-optimization $\leq_C$ MIS-construction

Proof:
Algorithm MIS-optimization(G){
For $k = 1$ to number of vertices call MIS-construction(G,k).

Report largest value of k for which construction succeeds.
}

MIS-optimization $\leq_C$ MIS-construction

Proof:

Algorithm MIS-optimization(G){
For $k = 1$ to number of vertices call MIS-construction(G,k).

Report largest value of k for which construction succeeds.
}

Polynomial number of calls + polytime additional work.

MIS-construction $\leq_C$ MIS-existence

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) $=$ NO, then return NO.

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) $=$ NO, then return NO.

$u =$ any vertex in G.

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) $=$ NO, then return NO.

$u =$ any vertex in G.
$G' =$ graph after removing u from G.

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) $=$ NO, then return NO.

$u =$ any vertex in G.
$G' =$ graph after removing u from G.
$G'' =$ graph after removing u and its neighbours from G.

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) $=$ NO, then return NO.

$u =$ any vertex in G.
$G' =$ graph after removing u from G.
$G'' =$ graph after removing u and its neighbours from G.

If MIS-existence(G'', $k-1$) return $u||$ MIS-construction(G'', $k-1$)

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) $=$ NO, then return NO.

$u =$ any vertex in G.
$G' =$ graph after removing u from G.
$G'' =$ graph after removing u and its neighbours from G.

If MIS-existence(G'', $k-1$) return $u||$ MIS-construction(G'', $k-1$)
else return MIS-construction(G', k).
}

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) = NO, then return NO.

u = any vertex in G.
G' = graph after removing u from G.
G'' = graph after removing u and its neighbours from G.

If MIS-existence(G'', $k-1$) return $u||$ MIS-construction(G'', $k-1$)
else return MIS-construction(G', k).
}

At most n recursive calls to MIS-construction.

n = number of vertices.

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) $=$ NO, then return NO.

$u =$ any vertex in G.
$G' =$ graph after removing u from G.
$G'' =$ graph after removing u and its neighbours from G.

If MIS-existence$(G'', k-1)$ return $u||$ MIS-construction$(G'', k-1)$
else return MIS-construction(G', k).
}

At most n recursive calls to MIS-construction.

$n =$ number of vertices.

Thus at most n calls to MIS-existence.

MIS-construction $\leq_C$ MIS-existence

Proof:

Algorithm MIS-construction(G,k){

If MIS-existence(G,k) $=$ NO, then return NO.

$u =$ any vertex in G.
$G' =$ graph after removing u from G.
$G'' =$ graph after removing u and its neighbours from G.

If MIS-existence($G'', k-1$) return $u||$ MIS-construction($G'', k-1$)
else return MIS-construction(G', k).
}

At most n recursive calls to MIS-construction.

$n =$ number of vertices.

Thus at most n calls to MIS-existence.
Additional work besides calls to MIS-existence: polynomial in size of G.

Technicality 4: Reducibility amongst decision problems

Technicality 4: Reducibility amongst decision problems

General Karp reduction: $R \leq_K Q$ iff there exist polytime instance map IM and solution map SM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $SM(Q(IM(r))) = R(r)$.

Technicality 4: Reducibility amongst decision problems

General Karp reduction: $R \leq_K Q$ iff there exist polytime instance map IM and solution map SM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $SM(Q(IM(r))) = R(r)$.

Q, R decision problems: Solution map SM can only be identity, or complement.

Technicality 4: Reducibility amongst decision problems

General Karp reduction: $R \leq_K Q$ iff there exist polytime instance map IM and solution map SM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $SM(Q(IM(r))) = R(r)$.

Q, R decision problems: Solution map SM can only be identity, or complement.

For simplicity Karp chose identity.

Technicality 4: Reducibility amongst decision problems

General Karp reduction: $R \leq_K Q$ iff there exist polytime instance map IM and solution map SM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $SM(Q(IM(r))) = R(r)$.

Q, R decision problems: Solution map SM can only be identity, or complement.

For simplicity Karp chose identity.

Reduction among decision problems (Karp): $R \leq_K Q$ iff there exist a polytime instance map IM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $Q(IM(r)) = R(r)$.

Technicality 4: Reducibility amongst decision problems

General Karp reduction: $R \leq_K Q$ iff there exist polytime instance map IM and solution map SM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $SM(Q(IM(r))) = R(r)$.

Q, R decision problems: Solution map SM can only be identity, or complement.

For simplicity Karp chose identity.

Reduction among decision problems (Karp): $R \leq_K Q$ iff there exist a polytime instance map IM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $Q(IM(r)) = R(r)$.

Property $R \leq_K Q$, $Q \in P \Rightarrow R \in P$ is true.

Technicality 4: Reducibility amongst decision problems

General Karp reduction: $R \leq_K Q$ iff there exist polytime instance map IM and solution map SM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $SM(Q(IM(r))) = R(r)$.

Q, R decision problems: Solution map SM can only be identity, or complement.

For simplicity Karp chose identity.

Reduction among decision problems (Karp): $R \leq_K Q$ iff there exist a polytime instance map IM s.t. for instance r of R, $IM(r)$ is an instance of Q, and $Q(IM(r)) = R(r)$.

Property $R \leq_K Q$, $Q \in P \Rightarrow R \in P$ is true.

We could have another definition with complement, "Karp-complement", but it isnt commonly useful. But the property will be true for it too.

A simple reduction

A simple reduction

IS(G,k) : Does G have an IS of size k?

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS
Proof $\Rightarrow$:

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS
Proof $\Rightarrow$:
$(u, v) \in E \Rightarrow$ either u, v is in V'.

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS
Proof $\Rightarrow$:
$(u, v) \in E \Rightarrow$ either u, v is in V'.
Both u, v cannot be in $V - V'$.

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS
Proof $\Rightarrow$:
$(u, v) \in E \Rightarrow$ either u, v is in V'.
Both u, v cannot be in $V - V'$.
So $V - V'$ is independent.

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS
Proof $\Rightarrow$:
$(u, v) \in E \Rightarrow$ either u, v is in V'.
Both u, v cannot be in $V - V'$.
So $V - V'$ is independent.

Proof $\Leftarrow$

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS
Proof $\Rightarrow$:
$(u, v) \in E \Rightarrow$ either u, v is in V'.
Both u, v cannot be in $V - V'$.
So $V - V'$ is independent.

Proof $\Leftarrow$
$(u, v) \in E \Rightarrow$ both $u, v \notin V - V'$

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS
Proof $\Rightarrow$:
$(u, v) \in E \Rightarrow$ either u, v is in V'.
Both u, v cannot be in $V - V'$.
So $V - V'$ is independent.

Proof $\Leftarrow$
$(u, v) \in E \Rightarrow$ both $u, v \notin V - V'$
At least one of u, v is in V'.

A simple reduction

IS(G,k) : Does G have an IS of size k?
VC(G,k) : Does G have a VC of size k?

Thm: IS $\leq_K$ VC

From now on: decision problems

Lemma: V' is a VC in $G \Leftrightarrow V - V'$ is an IS
Proof $\Rightarrow$:
$(u, v) \in E \Rightarrow$ either u, v is in V'.
Both u, v cannot be in $V - V'$.
So $V - V'$ is independent.

Proof $\Leftarrow$
$(u, v) \in E \Rightarrow$ both $u, v \notin V - V'$
At least one of u, v is in V'.
V' is a VC.

Proof of IS $\leq_K$ VC

Proof of IS $\leq_K$ VC

Instance map:

Proof of IS $\leq_K$ VC

Instance map:

IM(G,k)$\{$

$n =$ number of vertices in G

Return instance $G, n - k$.

$\}$

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
n = number of vertices in G
Return instance $G, n-k$.
}

Clearly IM is polytime.

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
$n =$ number of vertices in G
Return instance $G, n - k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
n = number of vertices in G
Return instance $G, n - k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Must show (1) IS(G,k) = YES $\Rightarrow$ VC(G,n-k) = YES, and
(2) IS(G,k) = NO $\Rightarrow$ VC(G,n-k) = NO.

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
$n =$ number of vertices in G
Return instance $G, n - k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Must show (1) IS(G,k) = YES $\Rightarrow$ VC(G,n-k) = YES, and
(2) IS(G,k) = NO $\Rightarrow$ VC(G,n-k) = NO.

Proof of (1) : IS(G,k) = Yes $\Rightarrow$ IS V' of size k exists.

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
$n =$ number of vertices in G
Return instance $G, n - k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Must show (1) IS(G,k) = YES $\Rightarrow$ VC(G,n-k) = YES, and
(2) IS(G,k) = NO $\Rightarrow$ VC(G,n-k) = NO.

Proof of (1) : IS(G,k) = Yes $\Rightarrow$ IS V' of size k exists.
By Lemma V-V' is VC. Has size n-k.

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
$n =$ number of vertices in G
Return instance $G, n - k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Must show (1) IS(G,k) = YES $\Rightarrow$ VC(G,n-k) = YES, and
(2) IS(G,k) = NO $\Rightarrow$ VC(G,n-k) = NO.

Proof of (1) : IS(G,k) = Yes $\Rightarrow$ IS V' of size k exists.
By Lemma V-V' is VC. Has size n-k. Thus VC(G,n-k) = YES.

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
n = number of vertices in G
Return instance $G, n - k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Must show (1) IS(G,k) = YES $\Rightarrow$ VC(G,n-k) = YES, and
(2) IS(G,k) = NO $\Rightarrow$ VC(G,n-k) = NO.

Proof of (1) : IS(G,k) = Yes $\Rightarrow$ IS V' of size k exists.
By Lemma V-V' is VC. Has size n-k. Thus VC(G,n-k) = YES.

We prove contrapositive of (2): VC(G,n-k) = YES $\Rightarrow$ IS(G,k) = YES.

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
n = number of vertices in G
Return instance $G, n-k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Must show (1) IS(G,k) = YES $\Rightarrow$ VC(G,n-k) = YES, and
(2) IS(G,k) = NO $\Rightarrow$ VC(G,n-k) = NO.

Proof of (1) : IS(G,k) = Yes $\Rightarrow$ IS V' of size k exists.
By Lemma V-V' is VC. Has size n-k. Thus VC(G,n-k) = YES.

We prove contrapositive of (2): VC(G,n-k) = YES $\Rightarrow$ IS(G,k) = YES.
VC(G,n-k) = YES $\Rightarrow$ VC V' of size n-k exists.

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
$n =$ number of vertices in G
Return instance $G, n-k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Must show (1) IS(G,k) = YES $\Rightarrow$ VC(G,n-k) = YES, and
(2) IS(G,k) = NO $\Rightarrow$ VC(G,n-k) = NO.

Proof of (1) : IS(G,k) = Yes $\Rightarrow$ IS V' of size k exists.
By Lemma V-V' is VC. Has size n-k. Thus VC(G,n-k) = YES.

We prove contrapositive of (2): VC(G,n-k) = YES $\Rightarrow$ IS(G,k) = YES.
VC(G,n-k) = YES $\Rightarrow$ VC V' of size n-k exists.
By Lemma V-V' is IS. Has size k.

Proof of IS $\leq_K$ VC

Instance map:
IM(G,k){
n = number of vertices in G
Return instance $G, n - k$.
}

Clearly IM is polytime.

Must show that IS(G,k) has a solution iff VC(G,n-k).

Must show (1) IS(G,k) = YES $\Rightarrow$ VC(G,n-k) = YES, and
(2) IS(G,k) = NO $\Rightarrow$ VC(G,n-k) = NO.

Proof of (1) : IS(G,k) = Yes $\Rightarrow$ IS V' of size k exists.
By Lemma V-V' is VC. Has size n-k. Thus VC(G,n-k) = YES.

We prove contrapositive of (2): VC(G,n-k) = YES $\Rightarrow$ IS(G,k) = YES.
VC(G,n-k) = YES $\Rightarrow$ VC V' of size n-k exists.
By Lemma V-V' is IS. Has size k. Thus IS(G,k) = YES.

Karp reduction restated

Let R, Q be decision problems. $R \leq_K Q$ if there exists function IM from instances of R to instances of Q s.t.

Karp reduction restated

Let R, Q be decision problems. $R \leq_K Q$ if there exists function IM from instances of R to instances of Q s.t.

1. IM runs in polytime.

Karp reduction restated

Let R, Q be decision problems. $R \leq_K Q$ if there exists function IM from instances of R to instances of Q s.t.

1. IM runs in polytime.
2. If R(r) = YES then Q(IM(R)) = YES

Karp reduction restated

Let R, Q be decision problems. $R \leq_K Q$ if there exists function IM from instances of R to instances of Q s.t.

1. IM runs in polytime.
2. If R(r) = YES then Q(IM(R)) = YES
3. If Q(IM(r)) = YES then R(r) = YES

Karp reduction restated

Let R, Q be decision problems. $R \leq_K Q$ if there exists function IM from instances of R to instances of Q s.t.

1. IM runs in polytime.
2. If R(r) = YES then Q(IM(R)) = YES
3. If Q(IM(r)) = YES then R(r) = YES

This will be our working definition.