# Lecture 5: NP-Completeness and Reductions

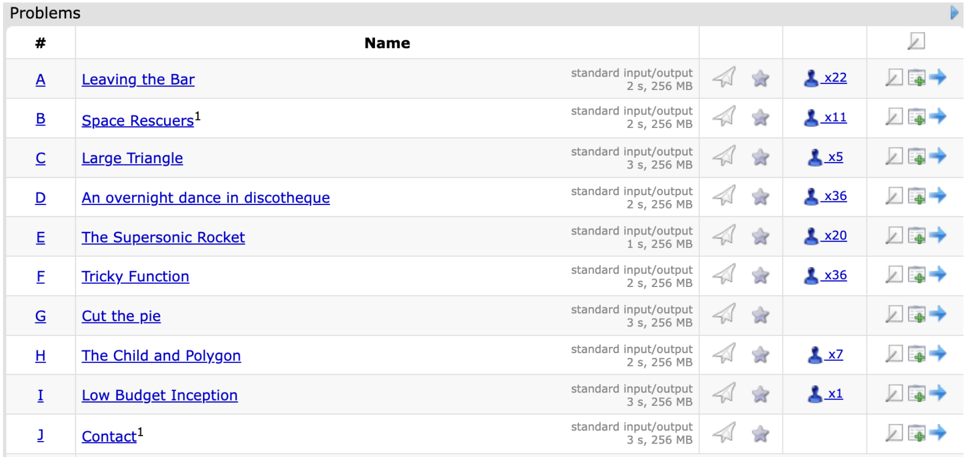
**Steven Skiena**

Department of Computer Science
State University of New York
Stony Brook, NY 11794–4400

http://www.cs.stonybrook.edu/~skiena

## Contest Results

Winner: Balneario Camboriu (7 problems, 740 minutes, 53 attempts at problem G)



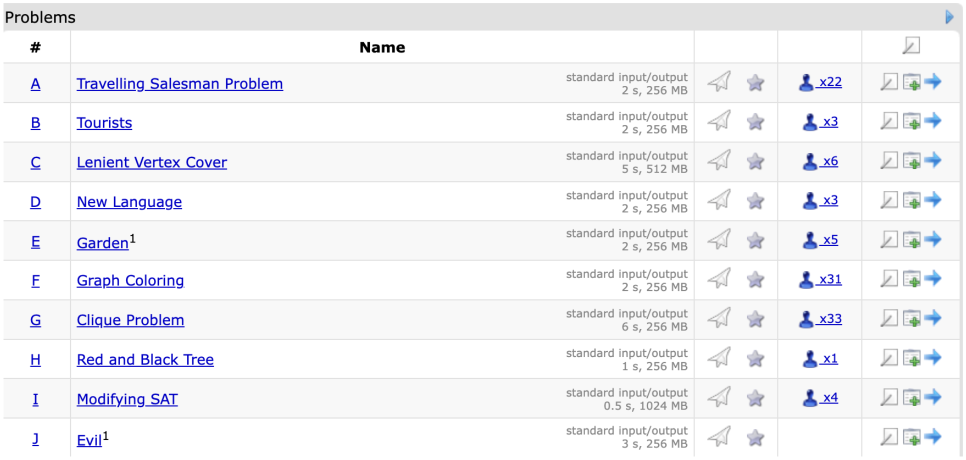
Problems

| # | Name | | | | |
|---|---|---|---|---|---|
| A | Leaving the Bar | standard input/output 2 s, 256 MB | | x22 | |
| B | Space Rescuers[1] | standard input/output 2 s, 256 MB | | x11 | |
| C | Large Triangle | standard input/output 3 s, 256 MB | | x5 | |
| D | An overnight dance in discotheque | standard input/output 2 s, 256 MB | | x36 | |
| E | The Supersonic Rocket | standard input/output 1 s, 256 MB | | x20 | |
| F | Tricky Function | standard input/output 2 s, 256 MB | | x36 | |
| G | Cut the pie | standard input/output 3 s, 256 MB | | | |
| H | The Child and Polygon | standard input/output 2 s, 256 MB | | x7 | |
| I | Low Budget Inception | standard input/output 3 s, 256 MB | | x1 | |
| J | Contact[1] | standard input/output 3 s, 256 MB | | | |

Give a big thanks to Tanzir for setting up the contests!

## Final Contest Results

Winner: InChaVola (7 problems, 1019 minutes)



Problems

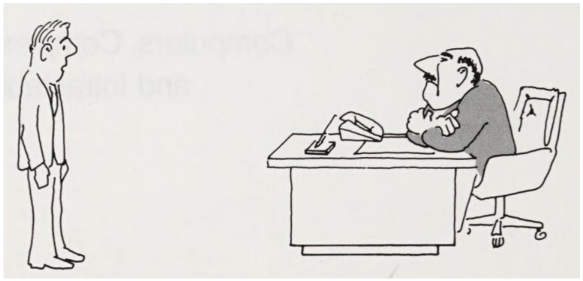
| # | Name | | | |
|---|---|---|---|---|
| A | Travelling Salesman Problem | standard input/output 2 s, 256 MB | | x22 |
| B | Tourists | standard input/output 2 s, 256 MB | | x3 |
| C | Lenient Vertex Cover | standard input/output 5 s, 512 MB | | x6 |
| D | New Language | standard input/output 2 s, 256 MB | | x3 |
| E | Garden[1] | standard input/output 2 s, 256 MB | | x5 |
| F | Graph Coloring | standard input/output 2 s, 256 MB | | x31 |
| G | Clique Problem | standard input/output 6 s, 256 MB | | x33 |
| H | Red and Black Tree | standard input/output 1 s, 256 MB | | x1 |
| I | Modifying SAT | standard input/output 0.5 s, 1024 MB | | x4 |
| J | Evil[1] | standard input/output 3 s, 256 MB | | |

## Topic: Introduction to NP-Completeness

- Introduction to NP-Completeness
- Reductions for Algorithms
- NP-Completeness and Graph Algorithms
- Satisfiability and Other Problems
- The Art of Proving Hardness
- P vs. NP
- Concluding Remarks

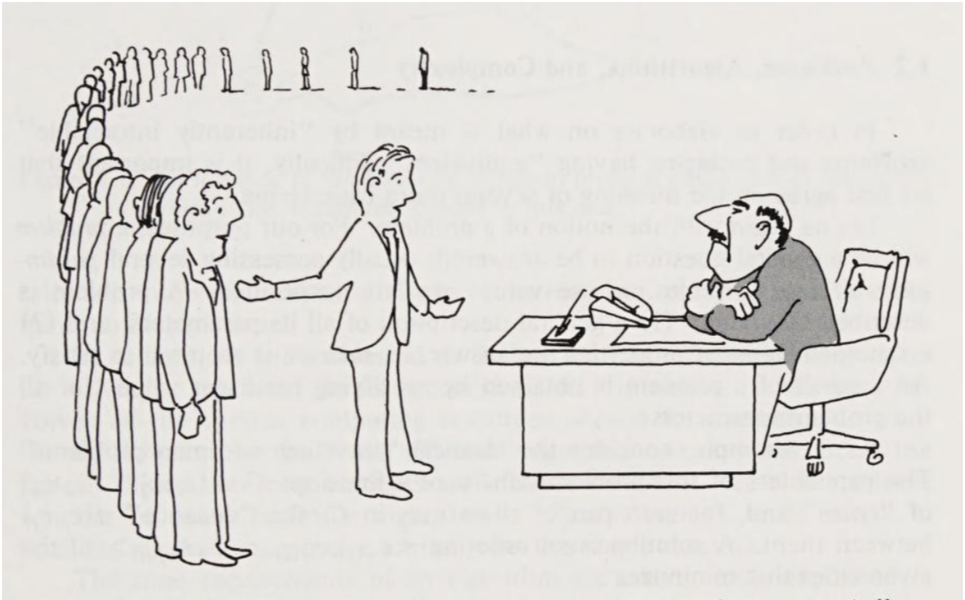
## Reporting to the Boss

Suppose you fail to find a fast algorithm. What can you tell your boss?



- "I guess I'm too dumb…" (dangerous confession)
- "There is no fast algorithm!" (lower bound proof)
- "I can't solve it, but no one else in the world can, either…" (NP-completeness reduction)

## The NP-Completeness Guarantee



“I can’t find an efficient algorithm, but neither can all these famous people.”

## The Theory of NP-Completeness

For many problems we can not find efficient algorithms, such as the traveling salesman problem.
We also cannot prove exponential-time lower bounds for these problems.
The theory of NP-completeness, developed by Stephen Cook and Richard Karp, provides the tools to show that all of these problems were really the same problem.

## The Main Idea

Suppose I gave you the following algorithm to solve the *bandersnatch* problem:

```
Bandersnatch($G$)
        Convert $G$ to an instance of the Bo-billy problem $Y$.
        Call the subroutine Bo-billy on $Y$ to solve this instance.
        Return the answer of Bo-billy($Y$) as the answer to $G$.
```

Such a translation from instances of one type of problem to instances of another type such that answers are preserved is called a *reduction*.

## What Does this Imply?

Now suppose my reduction translates $G$ to $Y$ in $O(P(n))$:

1. If my Bo-billy subroutine ran in $O(P'(n))$ I can solve the Bandersnatch problem in $O(P(n) + P'(n'))$
2. If I know that $\Omega(P'(n))$ is a lower-bound to compute Bandersnatch, then $\Omega(P'(n) - P(n'))$ must be a lower-bound to compute Bo-billy.

The second argument is the idea we use to prove problems hard!

## My Most Profound Tweet

An NP-completeness proof ensures that a dumb algorithm that is slow isn't a slow algorithm that is dumb.

But the ideas of NP-completeness and reductions are an important part of how I think about and design algorithms.

## Questions?

## Topic: Reductions for Algorithms

- Introduction to NP-Completeness
- Reductions for Algorithms
- NP-Completeness and Graph Algorithms
- Satisfiability and Other Problems
- The Art of Proving Hardness
- P vs. NP
- Concluding Remarks

## Reductions

Reducing (tranforming) one algorithm problem $A$ to another problem $B$ is an argument that if you can figure out how to solve $B$ then you can solve $A$.

Many algorithm problems are reducible to sorting (e.g. element uniqueness, mode, etc.).

## A Computer Scientist and an Engineer Wanted Some Tea…

## Make Graphs, Not Algorithms

Designing novel graph algorithms is very hard, so don’t do it. Instead, try to design graphs that enable you to use classical algorithms to model your problem.

This approach is consistent with the idea of a reduction between two problems, which is important in the theory of NP-completeness.

## Shortest $k$-Link Path

Given a weighted graph $G$, find the shortest (lowest weight) path using exactly $k$ links from $s$ to $t$.

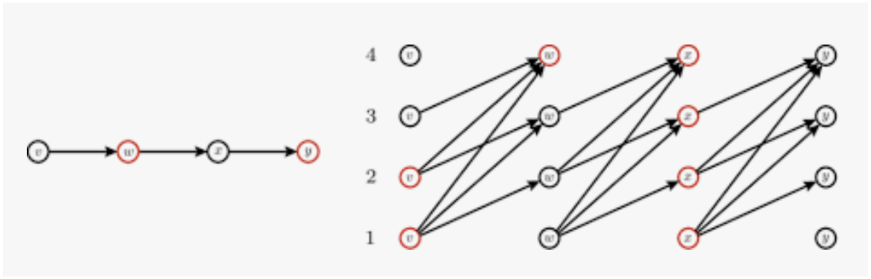
Yes, there is a dynamic programming solution:

$$Cost[s,t,k] = \min_j w(s,j) + Cost[j,t,k-1]$$

But is there another way?

## Reduction to Shortest Path

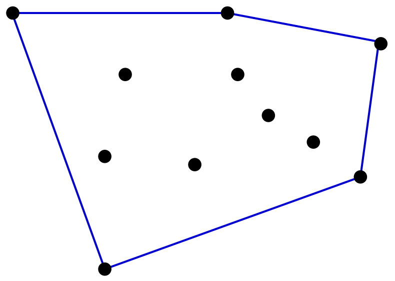
Build a DAG with $k$ copies of $G$, such that all edges go from the $i$th to $(i+1)$ copy.



Dijkstra's algorithm (or something simpler) can now find the shortest weighted path from $s_1$ to $t_k$.

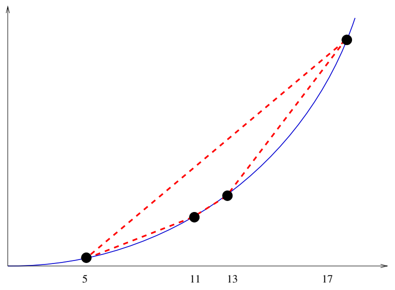
## Convex Hull and Sorting

Many algorithmic problems are reduciable to sorting (e.g. element uniqueness, mode, etc.)
A nice example of a reduction goes from sorting numbers to the convex hull problem:



We must translate each number to a point. We can map $x$ to $(x, x^2)$.

## Why the Parabola?



Each integer is mapped to a point on the parabola $y = x^2$. Since this parabola is convex, every point is on the convex hull. Further values, the convex hull returns the points sorted by $x$-coordinate, ie. the original numbers.

## Convex Hull to Sorting Reduction

```
Sort($S$)
    For each $i \in S$, create point $(i, i^2)$.
    Call subroutine convex-hull on this point set.
    From the leftmost point in the hull,
        read off the points from left to right.
```

Recall the sorting lower bound of $\Omega(n \lg n)$. If we could do convex hull in better than $n \lg n$, we could sort faster than $\Omega(n \lg n)$ – which violates our lower bound.
*Thus convex hull must take* $\Omega(n \lg n)$ *as well!!!*
Observe that any $O(n \lg n)$ convex hull algorithm also gives us a complicated but correct $O(n \lg n)$ sorting algorithm.

## Shortest Common Superstring

Find the most compressed string $T$ that contains each string from a set of strings $S$ as a substring of $T$?

```
A B R A C A
R A C A D A
A C A D A B
C A D A B R
A D A B R A
```

```
A B R A C A D A B R A
A B R A C A
    R A C A D A
      A C A D A B
        C A D A B R
          A D A B R A
```

How can you find a superstring?

How can you find a short superstring?

How can you find the shortest superstring?

## Reduction to Traveling Salesman

Build a directed graph where $w(i, j)$ is max string length - longest suffix of $S_i$—prefix of $S_j$ overlap.
The minimum cost tour visiting all vertices (strings) defines the shortest common superstring.

## 3SUM-Hardness

Consider the problem of testing whether a set of integers $S$ contain three elements $a$, $b$ and $c$ such that $a + b + c = 0$.
This can be solved in $O(n^2)$, and smart people think that there is no significantly faster algorithm.
Many other problems have been reduced to 3SUM, including testing whether three lines in the plane meet in a single point.
Thus a faster algorithm for 3SUM would mean a faster algorithm for degeneracy testing.

## Questions?

## Topic: NP-Completeness and Graph Algorithms

- Introduction to NP-Completeness
- Reductions for Algorithms
- NP-Completeness and Graph Algorithms
- Satisfiability and Other Problems
- The Art of Proving Hardness
- P vs. NP
- Concluding Remarks

## The Main Idea

Suppose I gave you the following algorithm to solve the *bandersnatch* problem:

```
Bandersnatch($G$)
        Convert $G$ to an instance of the Bo-billy problem $Y$.
        Call the subroutine Bo-billy on $Y$ to solve this instance.
        Return the answer of Bo-billy($Y$) as the answer to $G$.
```

Such a translation from instances of one type of problem to instances of another type such that answers are preserved is called a *reduction*.

## What Does this Imply?

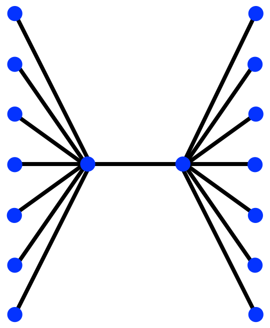
Now suppose my reduction translates $G$ to $Y$ in $O(P(n))$:

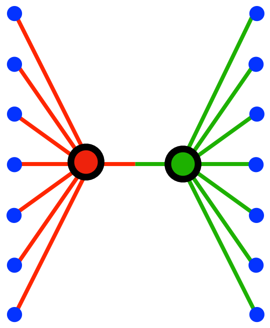
1. If my Bo-billy subroutine ran in $O(P'(n))$ I can solve the Bandersnatch problem in $O(P(n) + P'(n'))$
2. If I know that $\Omega(P'(n))$ is a lower-bound to compute Bandersnatch, then $\Omega(P'(n) - P(n'))$ must be a lower-bound to compute Bo-billy.

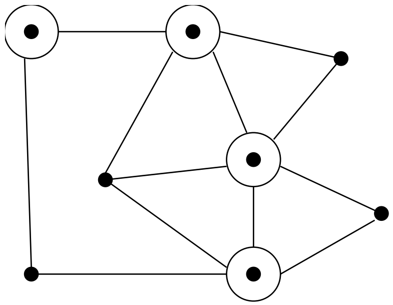
The second argument is the idea we use to prove problems hard!

## Vertex Cover

Instance: A graph $G = (V, E)$, and integer $k \leq V$
Question: Is there a subset of at most $k$ vertices such that every $e \in E$ has at least one vertex in the subset?







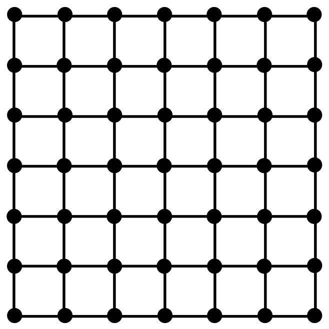
## Starting from the Right Problem

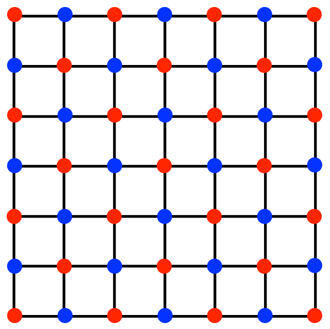
While theoretically any $NP$-complete problem can be reduced to any other one, choosing the correct one makes finding a reduction much easier.

$$3 - Sat \propto VC$$

## Maximum Independent Set

Instance: A graph $G = (V, E)$ and integer $j \leq v$.
Question: Does the graph contain an independent of $j$ vertices, ie. is there a subset of $v$ of size $j$ such that no pair of vertices in the subset defines an edge of $G$?

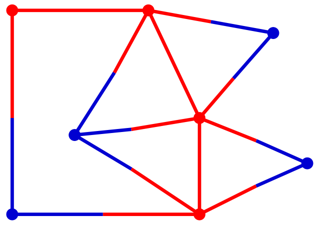




## Vertex Cover and Independent Set

When talking about graph problems, it is most natural to reducefrom a graph problem.
If you take a graph and find its vertex cover, the remaining vertices form an independent set, meaning there are no edges between any two vertices in the independent set.
Why? If there were such an edge the rest of the vertices could not have been a vertex cover.

## Maximum Independent Set is NP-Complete

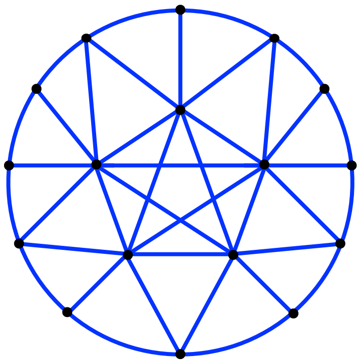
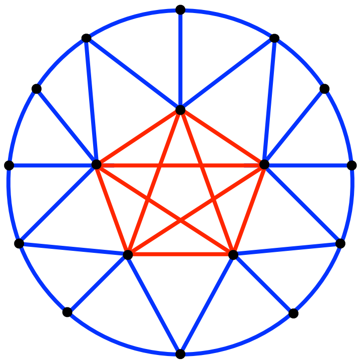


The smallest vertex cover gives the biggest independent set, and so the problems are equivallent: delete the subset of vertices in one from $V$ to get the other!

Thus finding the maximum independent set is NP-complete!

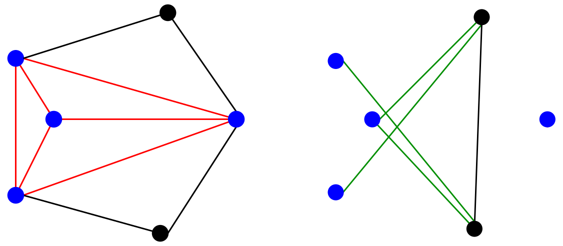
## Maximum Clique

Instance: A graph $G = (V, E)$ and integer $j \leq v$.
Question: Does the graph contain a clique of $j$ vertices, ie. is there a subset of $v$ of size $j$ such that every pair of vertices in the subset defines an edge of $G$?

## From Independent Set

In an independent set, there are no edges between two vertices. In a clique, there are always between two vertices. Thus if we complement a graph (have an edge iff there was no edge in the original graph), a clique becomes an independent set and an independent set becomes a clique!



## Densest Subgraph

Show that the *dense subgraph* problem is NP-complete:
Input: A graph $G$, and integers $k$ and $y$.
Question: Does $G$ contain a subgraph with exactly $k$ vertices and at least $y$ edges?

## Questions?

## Topic: Satisfiability and Other NP-Complete Problems

- Introduction to NP-Completeness
- Reductions for Algorithms
- NP-Completeness and Graph Algorithms
- Satisfiability and Other Problems
- The Art of Proving Hardness
- P vs. NP
- Concluding Remarks

## Satisfiability

Instance: A set $V$ of variables and a set of clauses $C$ over $V$.
Question: Does $C$ have a satisfying truth assignment?

$$(\, x_1 \text{ or } x_2 \text{ or } \overline{x_3} \,) \qquad (\, x_1 \text{ or } x_2 \text{ or } \overline{x_3} \,)$$
$$(\, x_1 \text{ or } \overline{x_2} \text{ or } x_3 \,) \qquad (\, x_1 \text{ or } \overline{x_2} \text{ or } x_3 \,)$$
$$(\, \overline{x_1} \text{ or } \overline{x_2} \text{ or } \overline{x_3} \,) \qquad (\, \overline{x_1} \text{ or } \overline{x_2} \text{ or } \overline{x_3} \,)$$
$$(\, \overline{x_1} \text{ or } x_2 \text{ or } x_3 \,) \qquad (\, \overline{x_1} \text{ or } x_2 \text{ or } x_3 \,)$$

Example 1: $V = v_1, v_2$ and $C = \{\{v_1, \overline{v}_2\}, \{\overline{v}_1, v_2\}\}$
A clause is satisfied when at least one literal in it is true. $C$ is satisfied when $v_1 = v_2 =$true.

## Not Satisfiable

Example 2: $V = v_1, v_2,$

$$C = \{\{v_1, v_2\}, \{v_1, \overline{v}_2\}, \{\overline{v}_1\}\}$$

Although you try, and you try, and you try and you try, you can get no satisfaction.
There is no satisfying assigment since $v_1$ must be false (third clause), so $v_2$ must be false (second clause), but then the first clause is unsatisfiable!

## Satisfiability is Hard

Satisfiability is known/assumed to be a hard problem.
Every top-notch algorithm expert in the world has tried and failed to come up with a fast algorithm to test whether a given set of clauses is satisfiable.
Further, many strange and impossible-to-believe things have been shown to be true if someone in fact did find a fast satisfiability algorithm.

## Integer Partition (Subset Sum)

Instance: A set of integers $S$ and a target integer $t$.
Problem: Is there a subset of $S$ which adds up exactly to $t$?
Example: $S = \{1, 4, 16, 64, 256, 1040, 1041, 1093, 1284, 1344\}$ and $T = 3754$
Answer: $1 + 16 + 64 + 256 + 1040 + 1093 + 1284 = T$
Observe that integer partition is a number problem, as opposed to the graph and logic problems we have seen to date.

## Integer Programming

Instance: A set $v$ of integer variables, a set of inequalities over these variables, a function $f(v)$ to maximize, and integer $B$.
Question: Does there exist an assignment of integers to $v$ such that all inequalities are true and $f(v) \geq B$?
Example:

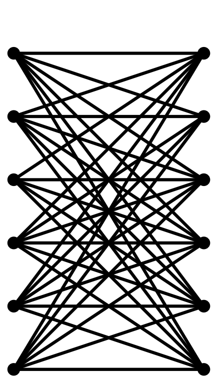
$$v_1 \geq 1, \quad v_2 \geq 0$$

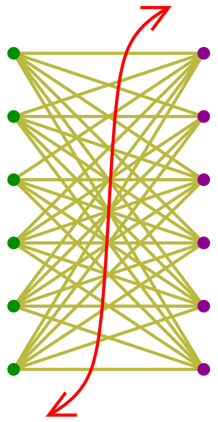
$$v_1 + v_2 \leq 3$$

$$f(v) : 2v_2, \quad B = 3$$

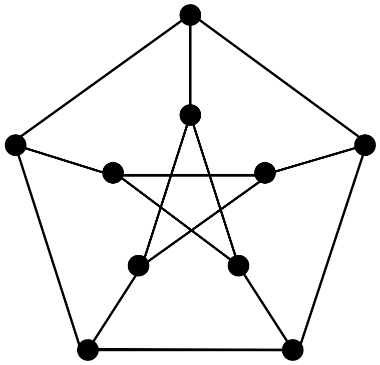
A solution to this is $v_1 = 1$, $v_2 = 2$.

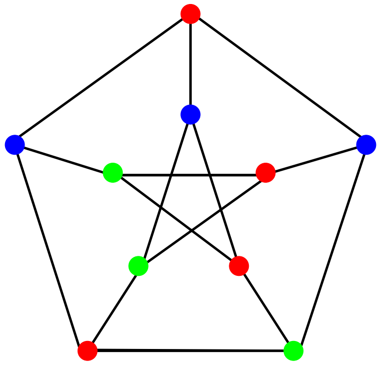
## Maximum Cut



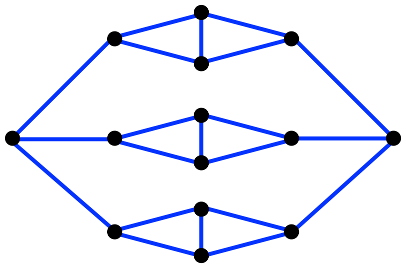


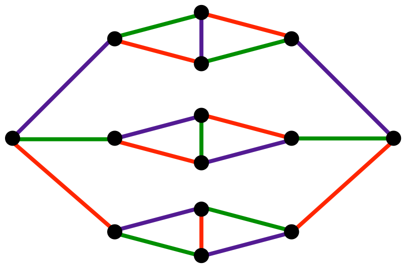
## Vertex Coloring



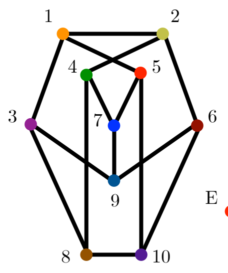


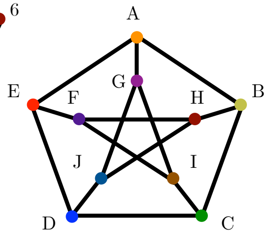
## Edge Coloring

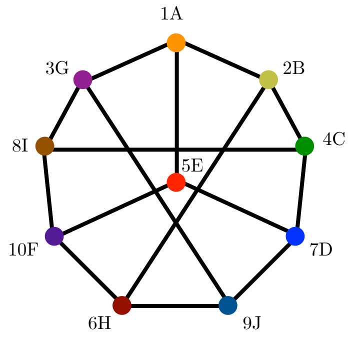




## Graph Isomorphism

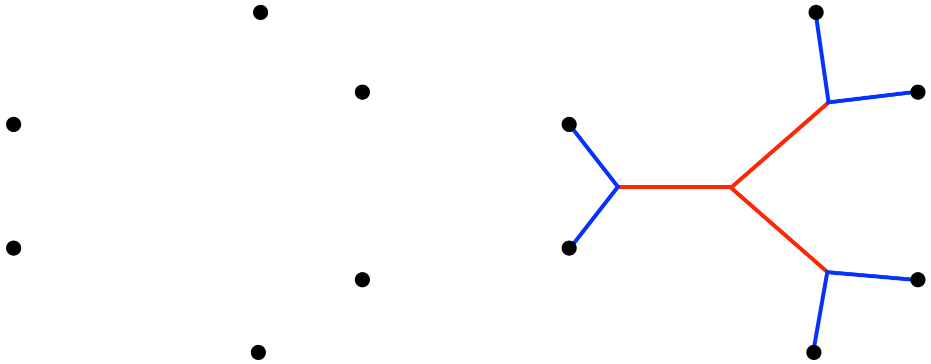






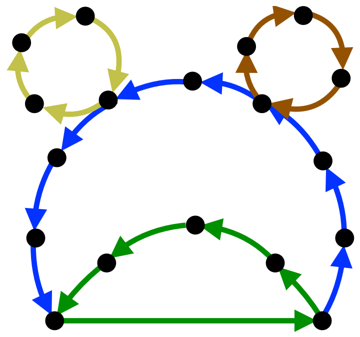
Not NP-complete, but Isomorphism-Hard

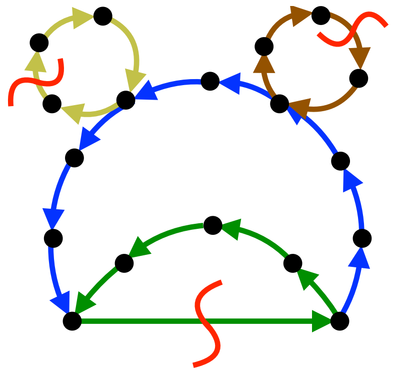
## Steiner Tree



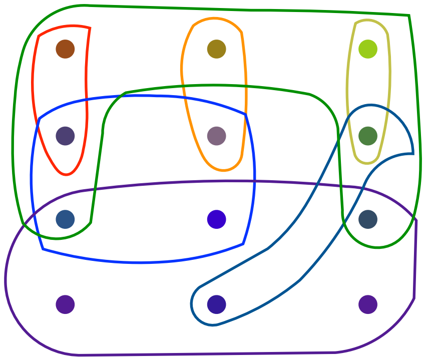
MST is hard in graphs if only a subset of nodes must be in the tree.

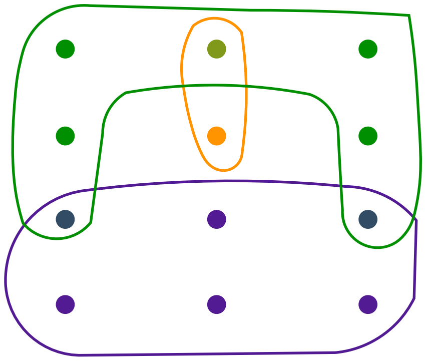
## Feedback Edge/Vertex Set



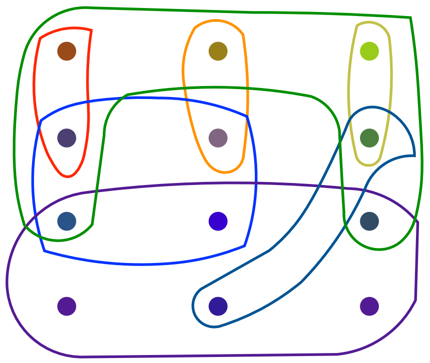
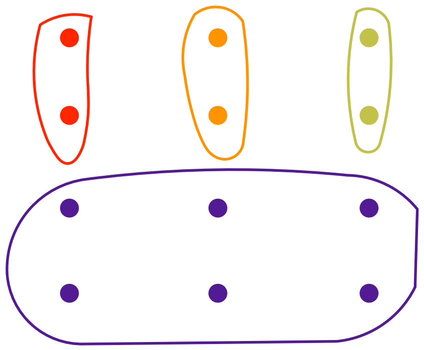


## Set Cover

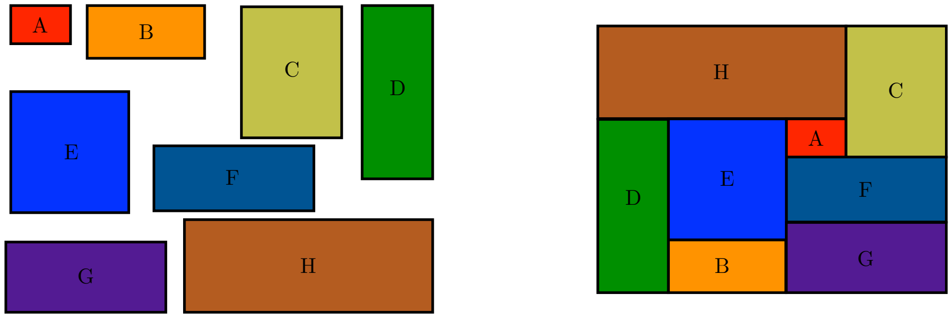




## Set Packing

# Bin Packing



## Questions?

## Topic: The Art of Proving Hardness

- Introduction to NP-Completeness
- Reductions for Algorithms
- NP-Completeness and Graph Algorithms
- Satisfiability and Other Problems
- The Art of Proving Hardness
- P vs. NP
- Concluding Remarks

## The Art of Proving Hardness

Proving that problems are hard is an skill. Once you get the hang of it, it is surprisingly straightforward and pleasurable to do.

The dirty little secret of NP-completeness proofs is that they are usually easier to recreate than explain, in the same way that it is usually easier to rewrite old code than the try to understand it.

I offer the following advice to those needing to prove the hardness of a given problem…

## Make your source problem as simple (i.e. restricted) as possible

Never use TSP as a source problem (Bandersnatch):

- Better is *TSP on instances restricted to the triangle inequality*.
- Even better, use *Hamiltonian cycle*, where all the weights are 1 or $\infty$.
- Even better, use *Hamiltonian path* instead of cycle.
- Even better, use *Hamiltonian path on directed, planar graphs where each vertex has total degree 3*.

All are equally hard, so the more you can restrict Bandersnatch, the less work your reduction has to do.

## Make your target problem as hard as possible

Don’t be afraid to add extra constraints or weights or freedoms to the Bo-billy problem in order to make your problem more general (at least temporarily).

## Select the right source problem for the right reason

Selecting the right source problem makes a big difference is how difficult it is to prove a problem hard. This is the first and easiest place to go wrong.
I usually consider four and only four problems as candidates for my hard source problem. Limiting them to four means that I know a lot about these problems:

- 3-Sat – that old reliable… When none of the three problems below seem appropriate, I go back to the source.
- Integer partition – the one and only choice for problems whose hardness seems to require using large numbers.

- Vertex cover – for any graph problems whose hardness depends upon *selection*. Chromatic number, clique, and independent set all involve trying to select the correct subset of vertices or edges.
- Hamiltonian path – for any graph problems whose hardness depends upon *ordering*, like when you are trying to route or schedule something.

## Amplify the penalties for making the undesired transition

You are trying to translate one problem into another, while making them stay the same as much as possible.

Be bold with your penalties, to punish anyone trying to deviate from your proposed solution.

“If you pick this, then you have to pick up this huge set which dooms you to lose.”

The sharper the consequences for doing what is undesired, the easier it is to prove if and only if.

## Think strategically at a high level, then build gadgets to enforce tactics.

You should be asking these kinds of questions:

- How can I force that either A or B but not both are chosen?
- How can I force that A is taken before B?
- How can I clean up the things I did not select?

## Alternate between looking for an algorithm or a reduction if you get stuck

Sometimes the reason you cannot prove hardness is that there is an efficient algorithm to solve your problem!

When you can’t prove hardness, it likely pays to change your thinking at least for a little while to keep you honest.

## Now watch me try it!

To demonstrate how one goes about proving a problem hard, I accept the challenge of showing how a proof can be built on the fly.

I need a volunteer to pick a random problem from the 400+ hard problems in the back of Garey and Johnson.

`https://www.cs.stonybrook.edu/~skiena/373/hard.txt`

## The Problem

## The Solution

## Questions?

## Topic: P vs. NP

- Introduction to NP-Completeness
- Reductions for Algorithms
- NP-Completeness and Graph Algorithms
- Satisfiability and Other Problems
- The Art of Proving Hardness
- P vs. NP
- Concluding Remarks

## P versus NP

- A problem is in $NP$ if a given answer can be checked in polynomial time.
- A problem is in $P$ if it can be solve in time polynomial in the size of the input.

Satisfiability is in $NP$, since we can guess an assignment of (true, false) to the literals and check it in polynomial time.
The precise distinction between $P$ or $NP$ is somewhat technical, requiring formal language theory and Turing machines to state correctly.
But the real issue is the difference between finding solutions or verifying them.

## Classifying Example Problems

- In $P$ – Is there a path from $s$ to $t$ in $G$ of length less than $k$.
- In $NP$ – Is there a TSP tour in $G$ of length less than $k$. Given the tour, it is easy to add up the costs and convince me it is correct.
- *Not* in $NP$ – How many TSP tours are there in $G$ of length less than $k$. Since there can be an exponential number of them, we cannot count them all in polynomial time.

Don't let this issue confuse you – the important idea here is of reductions as a way of proving hardness.

## Polynomial or Exponential?

Just changing a problem a little can make the difference between it being in $P$ or $NP$-complete:

| | $P$ | $NP$-complete |
|---|---|---|
| | Shortest Path | Longest Path |
| | Eulerian Circuit | Hamiltonian Circuit |
| | Edge Cover | Vertex Cover |

The first thing you should do when you suspect a problem might be NP-complete is look in Garey and Johnson, *Computers and Intractability*.

## Is $P = NP$?

This remains the greatest open problem in Computer Science.

Some will say it is true for $N = 1$ or $P = 0$. :-)

## Questions?

## Topic: Concluding Remarks

- Introduction to NP-Completeness
- Reductions for Algorithms
- NP-Completeness and Graph Algorithms
- Satisfiability and Other Problems
- The Art of Proving Hardness
- P vs. NP
- Concluding Remarks

## This Was Fun

Thanks for your attention and enthusiasm.
I greatly enjoyed teaching and meeting you all.
Thanks also to the organizers: Rafael and Carlos, and all the assistants.

## How Should You Spend the Rest of Your Life?

You are all bright people and will achieve success however you define it.
Industry needs smart people every bit as much as academia.
The best place to live depends upon many things: home, family, economic prospects, lifestyle preferences, political situations etc.

## Should I Go To Grad School?

People with skills like should be able to get good jobs in industry with your undergraduate degree.

But technical skills atrophy with time. With time you get more expert on your companies projects and less so with what is changing in the world.

More interesting jobs, at higher pay, are usually available to those with gradate degrees.

When you are young is the right time to go to graduate school, before you have important family obligations.

## Should I Go to a Different Country?

The number and quality of graduate programs in Brazil and other countries is increasing, but the strongest programs in the world are abroad.

I always tell my Stony Brook undersgraduates that it is better to shift to a different university for a PhD (and usually for an MS) because you will learn from different people who taught you undergrad.

For a PhD degree, I encourage students to find the best university that they can get accepted to: it does make a difference.

## Should I Go to the US?

We used to get students from Brazil at Stony Brook, but less so now.

Financially, any decent US university will give complete financial support to all of our PhD students: free tuition plus a $36,000/year stipend: enough to live securely.

Our PhD students often get summer jobs at Google, Facebook, etc. while they are students, and permanent job offers on graduation.

The PhD model in the US is somewhat different than Europe and other places: a year or so longer, a little more course work, and research funding from faculty advisors instead of grants or teaching.

I personally think university research at top US schools offer more exciting research prospects (e.g. AI and quantum) than at other place, due to a mix of industrial activity/funding, international faculty, and entrepeneurial university culture.

## Should I go to Stony Brook?

This is 100% up to you.
But feel free to write me with any questions if you have them:
skiena@cs.stonybrook.edu
Be sure to say at the top that we met in Brazil at the IOI or ICPC class I taught so I know who you are.
Good luck at the World Finals! I will be rooting for you!

## Questions?