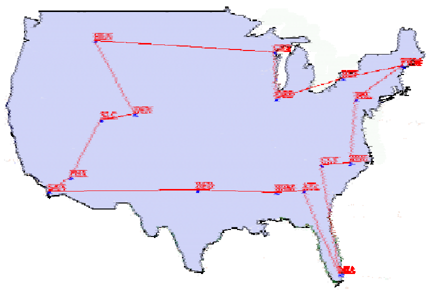
Lecture T6: NP-completeness



Is there a tour of length at most 1570?

Overview

Lecture T4:

- **What is an algorithm?**
  - **Turing machine**
- **Which problems can be solved on a computer?**
  - **not the halting problem**

Lecture T5:

- **Which algorithms will be useful in practice?**
  - **polynomial vs. exponential algorithms**

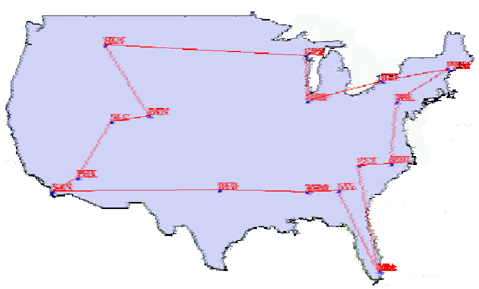
This lecture:

- **Which problems can be solved on a computer in a reasonable amount of time?**
  - **probably not the travelling salesperson problem (TSP)**

Some Hard Problems

TSP

- **A travelling salesperson needs to visit N cities. Is there a route of length at most D?**



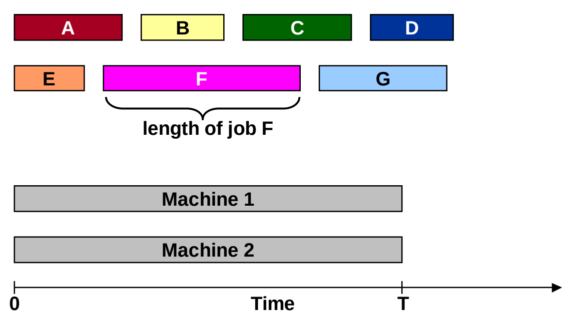
Is there a tour of length at most 1570? Yes, red tour = 1565.

Some Hard Problems

TSP

SCHEDULE

- **A set of jobs of varying length need to be processed on two identical machines before a certain deadline T. Can the jobs be arranged so that the deadline is met?**



Some Hard Problems

TSP
SCHEDULE
CLIQUE

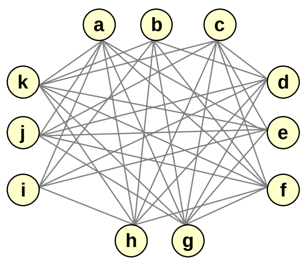
- **Given N people and their pairwise relationships. Is there a group of S people such that every pair in the group knows each other.**

Friendship Graph

People: a, b, c, d, e, . . ., k

Friendships: (a, e), (a, f), (a, g), . . ., (h, k)

Clique size: S = 4?



Some Hard Problems

TSP
SCHEDULE
CLIQUE
SAT

- **Is there a way to assign truth values to a given Boolean formula that makes it true?**

Boolean formula: (x' + y + z) (x + y' + z) (y + z) (x' + y' + z')

Yes, x = true, y = true, z = false.

Some Hard Problems

TSP
SCHEDULE
CLIQUE
SAT
FACTOR

- **Given two positive integers X and L, is there a nontrivial factor of X that is less than L?**
- **Factoring is at the heart of RSA encryption.**

Input: X = 23,536,481,273, L = 110,000

Yes, since X = 224,737 * 104,729.

Some Hard Problems

TSP
SCHEDULE
CLIQUE
SAT
FACTOR

These problems are intimately related!

Richard Karp (1960's)

Properties of Algorithms

What is an algorithm?

- Informally, a step-by-step set of instructions that can be applied in the same way to all instances of a problem.
- Formally, a deterministic Turing machine. [Recall Lectures T3, T4.]
  - always produces the same answer given the same input

Properties of Algorithms

A given problem can be solved by many different algorithms.

- Which ones are useful in practice?

A working definition: (Jack Edmonds, 1962)

- Efficient: polynomial time for ALL inputs.
  - mergesort requires N $\log_2 N$ steps
- Inefficient: "exponential time" for SOME inputs.
  - brute force TSP takes $N! > 2^N$ steps

Robust definition has led to explosion of useful algorithms for wide spectrum of problems.

Properties of Computers

Modern computers have varying characteristics:

- 1970's mainframe.
- 1980's personal computer.
- 1990's microprocessor.
- Supercomputer.
- Network of computers.

From a theoretical standpoint, they're all the same.

- 1930's Turing machine.

For example, none of these machine can solve general 1,000 city TSP problems. . . .

Exponential Growth

Exponential growth dwarfs technological change.

- Suppose each electron in the universe had power of today's supercomputers.
- And each works for the life of the universe in an effort to solve TSP problem using N! algorithm from Lecture P6.

Some Numbers

quantity	number
Home PC instructions/second	10^9
Supercomputer instructions per second	10^{12}
Seconds per year	10^9
Age of universe in years (estimated)	10^{13}
Electrons in universe (estimated)	10^{79}

- Will not succeed!

$1000! >> 10^{1000} >> 10^{79} * 10^{13} * 10^9 * 10^{12}$

Complexity Class P

Definition of P:

- Set of all decision problems solvable in polynomial time on a deterministic Turing machine.
- Definition important because of Strong Church-Turing thesis.

Strong Church-Turing thesis:

- P is the set of all decision problems solvable in polynomial time on real computers.

Evidence supporting thesis:

- True for all physical computers.
  - can create deterministic TM that simulates TOY machine in polynomial time (and vice versa)
  - can create deterministic TM that simulates any physical machine in polynomial time (and vice versa)
- Possible exception:
  - quantum computers – no conventional gates

Complexity Class NP

Definition of NP:

- Set of all decision problems solvable in polynomial time on a nondeterministic Turing machine.
- Definition important because it links many fundamental problems.

Equivalent definition:

- Set of all decision problems that can be verified in polynomial time on a deterministic Turing machine.

FACTOR: Is there a nontrivial factor of X = 23,536,481,273 that is less than L = 110,000?

- Witness: 104,729 (a factor of X).
- Can efficiently verify that X / 104,729 = 224,737
  $\Rightarrow$ X is a yes-instance.
- Conclusion: FACTOR is in NP.

Complexity Class NP

Definition of NP:

- Set of all decision problems solvable in polynomial time on a nondeterministic Turing machine.
- Definition important because it links many fundamental problems.

Equivalent definition:

- Set of all decision problems that can be verified in polynomial time on a deterministic Turing machine.

SAT: is the formula (x' + y + z) (x + y' + z) (y + z) (x' + y' + z') satisfiable?

- Witness: (x,y,z) = (true, true, false) .
- Easy to verify that input is a yes-instance given witness.
- Conclusion: SAT is in NP.

Complexity Class NP

Definition of NP:

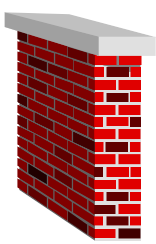
- Set of all decision problems solvable in polynomial time on a nondeterministic Turing machine.
- Definition important because it links many fundamental problems.

Equivalent definition:

- Set of all decision problems that can be verified in polynomial time on a deterministic Turing machine.

BIG PROBLEM: need to know solution ahead of time.

- Real computers can simulate by guessing all possibilities.
- Simulation takes exponential time unless you get "lucky."

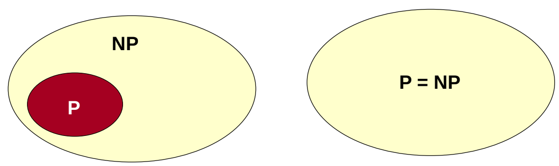


The Main Question

Does P = NP?

- **Is every problem that is solvable in poly time on a nondeterministic TM also solvable in poly time on a deterministic TM?**
- **Is the verification problem as hard as the original decision problem?**

Most important open problem in theoretical computer science. Also ranked #3 in all of mathematics. (Smale, 1999)



If P ≠ NP **If P = NP**

The Main Question

Does P = NP?

- **Is every problem that is solvable in poly time on a nondeterministic TM also solvable in poly time on a deterministic TM?**
- **Is the verification problem as hard as the original decision problem?**

If yes, then:

- **Efficient algorithms for TSP and factoring.**
- **Cryptography is impossible (except for one-time pads) on conventional machines.**
- **Modern banking system will collapse.**

If no, then:

- **Can't hope to write efficient algorithm for TSP.**
- **But maybe efficient algorithm still exists for factoring???**

The Main Question

Does P = NP?

- **Is every problem that is solvable in poly time on a nondeterministic TM also solvable in poly time on a deterministic TM?**
- **Is the verification problem as hard as the original decision problem?**

Probably no, since:

- **Thousands of researchers have spent four decades in search of polynomial algorithms for many fundamental NP problems without success.**
- **Consensus opinion: P ≠ NP.**

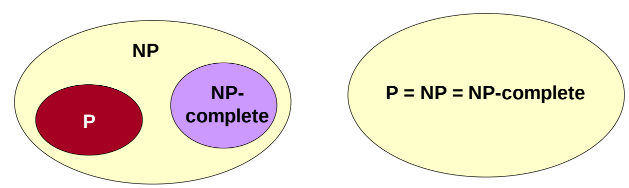
But maybe yes, since:

- **No success in proving P ≠ NP either.**

NP-Complete

Definition of NP-complete:

- **A problem with the property that if it can be solved in poly time, then so can every other problem in NP (hardest problems in NP).**



If P ≠ NP **If P = NP**

NP-Complete

Definition of NP-complete:

- A problem with the property that if it can be solved in poly time, then so can every other problem in NP (hardest problems in NP).

Links together a huge number of fundamental problems:

- TSP, SCHEDULE, SAT, CLIQUE, thousands more.
- Note: FACTOR is in NP but not known to be NP-complete.
- Given an efficient algorithm for TSP, can efficiently solve SCHEDULE, SAT, CLIQUE, FACTOR, etc.

Notorious complexity class.

- Only exponential algorithms known for these problems.
- Called intractable - unlikely that they can be solved given limited computing resources.

Reduction

Reduction is a general technique for showing that one problem is harder (easier) than another.

- For problems A and B, we can often show: if A can be solved efficiently, then so can B.
- In this case, we say B reduces to A. (B is "easier" than A).

Warmup: PRIMALITY reduces to FACTOR.

- Given any instance of PRIMALITY (i.e., positive integer p), we can determine the yes-no answer by using X = L = p as input to FACTOR and returning opposite answer.
  - original instance: Is p = 23,536,481,273 prime?
  - transformed instance: Does X = 23,536,481,273 have a nontrivial factor less than L = 23,536,481,273?
  - if answer to transformed instance is no, then answer to original instance is yes
  - if answer to transformed instances is yes, then answer to original instance is no

Reduction

Reduction is a general technique for showing that one problem is harder (easier) than another.

- For problems A and B, we can often show: if A can be solved efficiently, then so can B.
- In this case, we say B reduces to A. (B is "easier" than A).

SAT reduces to CLIQUE

- Given any input to SAT, we create a corresponding input to CLIQUE that will help us solve the original SAT problem.
- Specifically, for a SAT formula with K clauses, we construct a CLIQUE input that has a clique of size K if and only if the original Boolean formula is satisfiable.
- If we had an efficient algorithm for CLIQUE, we could apply our transformation, solve the associated CLIQUE problem, and obtain the yes-no answer for the original SAT problem.

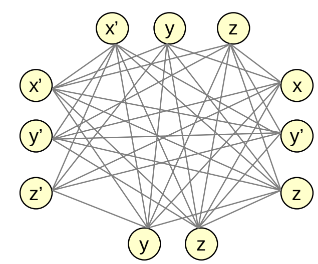
SAT reduces to CLIQUE

SAT reduces to CLIQUE

- Associate a person to each variable occurrence in each clause.
- Two people know each other except if:
  - they come from the same clause
  - they represent t and t' for some variable t

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')



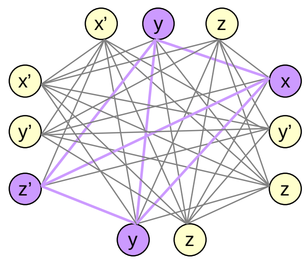
SAT reduces to CLIQUE

SAT reduces to CLIQUE

- Associate a person to each variable occurrence in each clause.
- Two people know each other except if:
  - they come from the same clause
  - they represent t and t' for some variable t
- Clique of size 4 ⇒ satisfiable assignment.
  - set variable in clique to true
  - (x, y, z) = (true, true, false)

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')



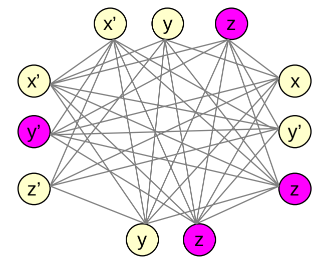
SAT reduces to CLIQUE

SAT reduces to CLIQUE

- Associate a person to each variable occurrence in each clause.
- Two people know each other except if:
  - they come from the same clause
  - they represent t and t' for some variable t
- Clique of size 4 ⇒ satisfiable assignment.
- Satisfiable assignment ⇒ clique of size 4
  - (x, y, z) = (false, false, true)
  - choose one true literal from each clause

Boolean formula:

(x' + y + z) (x + y' + z) (y + z) (x' + y' + z')



CLIQUE is NP-Complete

CLIQUE is NP-complete.

- CLIQUE is in NP.
- SAT is NP-complete.
- SAT reduces to CLIQUE.

Thousands of problems shown to be NP-complete in this way.

But, how was the first problem shown to be NP-complete?

The "World's First" NP-Complete Problem

SAT is NP-complete. (Cook-Levin, 1960's)

Idea of proof:

- By definition, nondeterministic TM can solve problem in NP in polynomial time.
- Polynomial-size Boolean formula can describe (nondeterministic) TM.
- Given any problem in NP, establish a correspondence with some instance of SAT.
- SAT solution gives simulation of TM solving the corresponding problem.
- IF SAT can be solved in polynomial time, then so can any problem in NP (e.g., TSP).

Stephen Cook

Coping With NP-Completeness

Hope that worst case doesn't occur.

- Complexity theory deals with worst case behavior. The instance(s) you want to solve may be "easy."
  - TSP where all points are on a line or circle
  - 13,509 US city TSP problem solved (Cook et. al., 1998)

Bill Cook

Coping With NP-Completeness

Hope that worst case doesn't occur.

Change the problem.

- Develop a heuristic, and hope it produces a good solution.
  - TSP assignment.
- Design an approximation algorithm: algorithm that is guaranteed to find a high-quality solution in polynomial time.
  - active area of research, but not always possible
  - Euclidean TSP tour within 1% of optimal (Arora, 1997)

Sanjeev Arora

Coping With NP-Completeness

Hope that worst case doesn't occur.

Change the problem.

Exploit NP-completeness.

Keep trying to prove P = NP.

Summary

Many fundamental problems are NP-complete.

- TSP, SAT, SCHEDULE.

Theory says we probably won't be able to design efficient algorithms for NP-complete problems.

- You will likely run into these problems in your scientific life.
- If you know about NP-completeness, you can identify them and avoid wasting time.

Theorem: if P = NP then cryptography is essentially impossible on conventional machines.