CS11313 - Fall 2022

Design & Analysis of Algorithms

NP Completeness

Ibrahim Albluwi

Reductions

A *reduction* from problem X to problem Y:
An algorithm for solving problem X that includes a solver of problem Y as a subroutine.

Problem X *reduces* to problem Y (denoted as $X \leqslant Y$): An algorithm for solving Y can be used to solve X.

Problem X *polytime-reduces* to problem Y ($X \leqslant_p Y$): An algorithm for solving Y can be used to solve X in addition to a polynomial-time amount of work.

Total cost for solving X = Cost of solving Y + Cost of reduction

Y might be called multiple time (typically 1 call)

Typically less than the cost of solving Y

Reductions (Examples)

LINEAR
Given b and c, solve $bx + c = 0$

QUADRATIC
Given a, b and c, solve $ax^2 + bx + c = 0$

LINEAR reduces to QUADRATIC

Reductions (Examples)

LINEAR
Given b and c, solve $bx + c = 0$

QUADRATIC
Given a, b and c, solve $ax^2 + bx + c = 0$

LINEAR reduces to QUADRATIC

SELECT
Given a list of elements, find the kth largest element.

SORT
Given a list of elements, order the elements in non-decreasing order.

SELECT reduces to SORT

Use SORT to sort the elements and then report the element of rank k.

Running Time. $\underbrace{O(N \log N)}_{\text{SORT}} + \underbrace{O(1)}_{\text{reduction}}$

SORT reduces to SELECT

Sort the elements by repeatedly using SELECT to find the next largest element.

Running Time. $\underbrace{O(N)}_{\text{SELECT}} \times \underbrace{O(N)}_{\text{reduction}}$

Reductions (Examples)

SSSP (Single Source Shortest Paths)

Given a graph G and a source vertex s, find the shortest path from s to every vertex in G.

SDSP (Single Destination Shortest Paths)

Given a graph G and a destination vertex d, find the shortest path from every vertex in G to d.

SDSP reduces to SSSP

- Create G^T, a transpose of G.
- Set s to d and run SSSP on G^T.
- Transpose the shortest paths tree.

Running Time. $\underbrace{O(E \log V)}_{\text{SSSP}} + \underbrace{O(E + V)}_{\text{reduction}}$

SSSP (using Dijkstra's algorithm, assuming the graph is cyclic and has non-negative weights)

reduction (transposing)

Quiz # 1

Suppose there is a proof that no computer can solve problem X.

How can we prove that a problem Y is also impossible to solve?

A. Show that X reduces to Y.

B. Show that Y reduces to X.

C. Computers can solve any problem. It is only that *we* might not be clever enough to come up with an algorithm!

D. It depends.

Quiz # 1

Suppose there is a proof that no computer can solve problem X.

How can we prove that a problem Y is also impossible to solve?

A. Show that X reduces to Y.

B. Show that Y reduces to X.

C. Computers can solve any problem. It is only that *we* might not be clever enough to come up with an algorithm!

D. It depends.

X reduces to Y

We can use Y to solve X.

If Y is solvable:
X is also solvable (contradiction!)

Y reduces to X

We can use X to solve Y.

While X is unsolvable, there might be another way for solving Y not using X.

Reductions (Examples)

optional

`TOTALITY`

Does a given program P terminate on all possible inputs?
(never enters an infinite loop!)

`EQUIVALENCE`

Given two programs P_1 and P_2. Do these two programs produce the same output for every input?
(i.e. are they equivalent?)

We know that `TOTALITY` is an impossible problem to solve. ← see the theory of computation course!

How can we show that `EQUIVALENCE` is also impossible to solve?

Reductions (Examples)

optional

TOTALITY

Does a given program P terminate on all possible inputs?
(never enters an infinite loop!)

EQUIVALENCE

Given two programs P_1 and P_2. Do these two programs produce the same output for every input?
(i.e. are they equivalent?)

We know that `TOTALITY` is an impossible problem to solve. ← see the theory of computation course!

How can we show that `EQUIVALENCE` is also impossible to solve?

Answer. Show that `TOTALITY` reduces to `EQUIVALENCE`.

`TOTALITY` reduces to `EQUIVALENCE`

- Create P_1 as a copy of P, except that it outputs `TRUE` instead of its original output.
- Create a program P_2 that outputs `TRUE` and does nothing else.
- Use `EQUIVALENCE` to check if P_1 and P_2 are equivalent.
  If they are equivalent, P terminates on all input. If they are not, the only possibility is that P does not terminate on some input (since the output of P_1 and P_2 is always the same).

Since `TOTALITY` can be solved using `EQUIVALENCE` and `TOTALITY` is known to be impossible, `EQUIVALENCE` must also be impossible.

Reductions (Examples)

optional

PAIR

Given lists L_1 and L_2 of size N, pair the min in L_1 with the min in L_2, the next min in L_1 with the next min in L_2, etc.

SORT

Given a list of elements, sort them in non-decreasing order.

Example.

L_1 = [13, 7, 3, 1, 11, 2]

L_2 = [2, 8, 6, 4, 10, 0]

PAIR = [1-0, 2-2, 3-4, 7-6, 11-8, 13-10]

Reductions (Examples)

optional

PAIR

Given lists L_1 and L_2 of size N, pair the min in L_1 with the min in L_2, the next min in L_1 with the next min in L_2, etc.

SORT

Given a list of elements, sort them in non-decreasing order.

Example.

L_1 = `[13, 7, 3, 1, 11, 2]`

L_2 = `[2, 8, 6, 4, 10, 0]`

PAIR = `[1-0, 2-2, 3-4, 7-6, 11-8, 13-10]`

PAIR reduces to SORT

- Use SORT to sort L_1 and L_2.
- Pair L_1`[0]` with L_2`[0]`,
  L_1`[1]` with L_2`[1]`,
  etc.

SORT reduces to PAIR

- Let L_1 be the list to be sorted.
- Create L_2 containing the numbers 1 to N.
- Extract the sorted version of L_1 from the result of applying PAIR on L_1 and L_2.

Reductions (Examples)

optional

Implication.

- We already know that any comparison based algorithm for SORT performs $\Omega(N \log N)$ compares in the worst case.
- The reduction from SORT to PAIR requires only $\Theta(N)$ amount of work (creating L_2 and extracting the result)
- PAIR must require $\Omega(N \log N)$ compares in the worst case. Otherwise, the $\Omega(N \log N)$ lower bound for SORT is not correct (contradiction!)

PAIR reduces to SORT

- Use SORT to sort L_1 and L_2.
- Pair L_1[0] with L_2[0],
  L_1[1] with L_2[1],
  etc.

SORT reduces to PAIR

- Let L_1 be the list to be sorted.
- Create L_2 containing the numbers 1 to N.
- Extract the sorted version of L_1 from the result of applying PAIR on L_1 and L_2.

NEVER FORGET

If A is hard to solve and
A easily reduces to B $(A \leqslant_p B)$,
Then B is also hard to solve!

What does it mean for a problem to be hard anyway?

A fine line Between *Hard* and *Easy* Problems

- Shortest Paths on unweighted graphs — O(E+V) using BFS
- Shortest Paths on weighted DAGs — O(E+V) using Topological sort
- Longest Paths on weighted DAGs — O(E+V) using Topological Sort
- Shortest Paths on weighted graphs (no negative weights) O(ELogV) using Dijkstra's
- Longest Paths on weighted graphs NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

- Fractional Knapsack Problem — has an efficient greedy algorithm
- 0-1 Knapsack Problem — NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

- Change Making for canonical coin systems — has an efficient greedy algorithm
- Change Making for arbitrary coin systems

NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

A fine line Between *Hard* and *Easy* Problems

Does a graph G contain an Eulerian Cycle?
(a cycle that visits all the *edges* in G exactly once)

Direct solution: True if and only if each vertex has an even degree!

Does a graph G contain a Hamiltonian Cycle?
(a cycle that visits all the *vertices* in G exactly once)

NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

Traveling Salesman Problem (TSP)
Given a complete weighted graph, what is the *shortest Hamiltonian Cycle*?

NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

A fine line Between *Hard* and *Easy* Problems

Is a graph 2-Colorable?
(can the vertices be colored using 2 colors, such that no two adjacent vertices have the same color?)

Direct solution: True if there is no cycle of odd length (can be checked using BFT)

Is a graph k-Colorable?
(can the vertices be colored using k colors or less, such that no two adjacent vertices have the same color?)

NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

More Hard Problems

Bin Packing

Given an unlimited number of bins (each with capacity C), and n objects with sizes $s_1, \ldots, s_n$ where $0 < s_i \leq C$, find the *minimum* number of bins needed to pack all objects.

NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

More Hard Problems

Subset Sum

Given a multiset S of integers and an integer k, find a *minimum* subset of S whose elements sum up to exactly k.

Example. S = {1, 1, 1, 4, 4, 5, 6}, k = 8

Possible Subsets: {1, 1, 1, 5}, {1, 1, 6}, {4, 4} ← min subset

NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

Subset Partition

Given a multiset S of integers, can S be partitioned into 2 subsets of the same sum?

Example. S = {1, 2, 3, 4}

YES: {1, 4} and {2, 3}

S = {1, 2, 3, 4, 5}

No

NO KNOWN POLYNOMIAL TIME ALGORITHM EXISTS!

A Hard Problem?

What is common between finding longest paths in cyclic graphs, 0-1 Knapsack, Subset Sum, Subset Partition, Bin-Packing, TSP and Checking if a Hamiltonian cycle exists?
(+ many others ...)

(1) No one until now found a polynomial time algorithm to solve any of them.

(2) No one proved that no polynomial time algorithm can be found for any of them.

(3) Each of them poly-time reduces to all the other problems!
I.e. Finding a polynomial time solution to any of them means that all of them have polynomial time solutions!

(4) You will get $1,000,000 from the Clay Mathematics Institute if you find a polynomial time solution for any of them or prove that any of them can't have a polynomial time solution!

Welcome to the

P *vs* NP

Problem

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples | Traveling Salesman Problem

Optimization problem:
Given a complete weighted graph G, find a simple circuit C that visits each node in G exactly once such that the total cost of the edges in C is *minimum*.

Decision problem:
Given a complete weighted graph G, does G contain a simple circuit C that visits each node exactly once such that the total cost of the edges in C is *less than or equal to some threshold T*?

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples

Bin-Packing

Optimization problem:
Given an unlimited number of bins (each with capacity C), and n objects with sizes $s_1, \ldots, s_n$ where $0 < s_i \leq C$, find the *minimum* number of bins needed to pack all objects

Decision problem:
Can the objects fit in *less than k bins* ?

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples Graph Coloring

Optimization problem:
Find the *minimum* number of colors such that adjacent vertices are not assigned the same color.

Decision problem:
Can the vertices be properly colored *in K or fewer* colors such that adjacent vertices are not assigned the same color?

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples Subset Sum

Optimization problem:
Given a multi-set S of integers and an integer k, find a *minimum* subset of S whose elements sum up to exactly k.

Decision problem:
Does S contain a subset whose elements sum up to exactly k?

Example.
S = {1, 1, 1, 4, 4, 5, 6}, k = 8

Possible Subsets: {1, 1, 1, 5}
{1, 1, 6}
{4, 4} ⟵ *minimum*

Definitions

Optimization problem:
Find the ***best*** solution among a set of feasible solutions.

Decision problem:
Requires a **yes/no** answer.

Examples Hamiltonian Cycle

Decision problem:
Is there a cycle that visits each vertex in the graph once?

Examples Subset Partition

Decision problem:
Given a set S of integers, Can we partition S into two subsets of exactly the same size?

Example. S = {1, 2, 3, 4}
YES: {1, 4} and {2, 3}

S = {1, 2, 3, 4, 5}
No

Quiz # 2

Given a solver for the optimization version of TSP, how can we solve the decision version?

Answer. If we know the length of the shortest tour L, we can very easily answer the question *Is there a tour of length less than* T as follows:

If $L \geq T$: There is no tour of length less than T.
If $L < T$: There is a tour of length less than T.

Quiz # 2

Given a solver for the optimization version of TSP, how can we solve the decision version?

Answer. If we know the length of the shortest tour L, we can very easily answer the question *Is there a tour of length less than T* as follows:

If $L \geq T$: There is no tour of length less than T.
If $L < T$: There is a tour of length less than T.

Given a solver for the decision version of TSP, how can we solve the optimization version?

Answer.

- Compute a bound B for the length of the shortest tour (e.g. the sum of the edge weights in the graph, or $V \times$ the largest weight)
- Use binary search to find the length of the shortest tour:

  Use the solver of the decision problem to answer the question:
  Is there a tour of length less than $B/2$?

  Eliminate the left or right half based on the answer and repeat.

Quiz # 3

If the decision version of a problem is hard, does this imply that the optimization version is also hard?

Quiz # 3

If the decision version of a problem is hard, does this imply that the optimization version is also hard?

Answer. Yes.

The decision version is no harder (as hard or easier) than the optimization version.

To discuss and prove hardness,
we will consider only *decision problems*!

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time.

Examples

- Given a list of integers L and an integer K:
  - is K in L?
  - Is there an integer in L that is greater than K ?
  - Do any two numbers in L sum to K ?
- Given a permutation of elements P:
  - is P sorted in ascending order?
  - is P a palindrome?
- Given a graph G:
  - Is there a spanning tree whose sum of edge weights is less than T ?
  - Is there a path between v and w in a graph G less than T ?
  - Is there a cycle in the graph?
  - Is the graph connected?
- Given a set of activities, can we schedule X activities without overlap?

etc.

Quiz # 4

Which of the following problems are *not* in **P** ?

A. Traveling Salesman Problem.

B. 0-1 Knapsack.

C. Bin-Packing.

D. All of the above.

I don't know.

Quiz # 4

Which of the following problems are *not* in **P** ?

A. Traveling Salesman Problem.

B. 0-1 Knapsack.

C. Bin-Packing.

D. All of the above.

We don't know.

A problem is in **P** if it has a polynomial time solution.

A problem is *not* in **P** if there is a proof that it does not have a polynomial time solution.

While we don't have polynomial time solutions for these problems, no one proved that these problems do not have polynomial time solutions!

Definitions (Complexity Classes)

Class P.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class NP.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I for a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Definitions (Complexity Classes)

Class P.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class NP.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I for a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example Is there A `HAMILTONIAN` Cycle?

Given a graph G, and a path C (a witness), can we verify in polynomial time if C is a hamiltonian cycle?

Yes!

1. Check that the first and last vertices are the same.
2. Check that no vertex repeats.
3. Check that the path has exactly V edges.

Definitions (Complexity Classes)

Class P.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class NP.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example TSP is in NP

Given a graph G, a length L, and a path C (a witness), can we verify in polynomial time if C is a hamiltonian cycle of length less than L?

Yes!

1. Check that C is a Hamiltonian cycle.
2. Check that the sum of the edge weights is less than L.

Definitions (Complexity Classes)

Class P.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class NP.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example SUBSET-SUM is in NP

Given a multi-set S, two integers K and L, and a subset H of S (a witness), can we verify in polynomial time if $|H| \leq K$ and that its elements sum to L?

Yes!

Definitions (Complexity Classes)

Class P.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class NP.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Example `SUBSET-SUM`

Given a multi-set S, two integers K and L, and a subset H of S (a witness), can we verify in polynomial time if $|H| \leq K$ and that its elements sum to L?

Yes!

Example `SUBSET-PARTITION`

Given a multi-set S, two subsets H_1 *and* H_2 of S (a witness), can we verify in polynomial time if $|H_1| + |H_2| = |S|$ and that the sum of the elements in H_1 = the sum of the elements in H_2?

Yes!

Quiz # 5

Every problem that is in **P** is also in **NP**.

A. True.

B. False.

We don't know.

Quiz # 5

Every problem that is in **P** is also in **NP**.

A. True.

B. False.

We don't know.

If a problem is solvable in polynomial time, it is also verifiable in polynomial time.

We can always solve the problem to verify a given witness!

Quiz # 6

Every problem that is in **NP** is also in **P**.

A. True.

B. False.

We don't know.

Quiz # 6

Every problem that is in **NP** is also in **P**.

A. True.

B. False.

We don't know.

Does easy verification imply that finding a solution is also easy?

- No one knows!
- No one yet found a problem that is in NP but is not in P !
- This is a $1,000,000 question!

Two Possible World Views

No one knows which is true!

Quiz # 7

What are examples of problems that are *not* in **NP**?

Quiz # 7

What are examples of problems that are *not* in **NP**?

Example. Given a chessboard, is there a move that guarantees black to win?

Given a chessboard and a certain move (a solution), we can't verify in polynomial time if the move will guarantee black to win

(we need to do exponential amount of work to check all possible black and white moves to see if black will win!)

What is in a name?

What does **NP** stand for?

A. **N**ot **P**olynomial.

B. **N**o **P**akeup Exam.

C. **N**o **P**roblem.

D. **N**one of the a**P**ove.

What is in a name?

What does **NP** stand for?

A. **N**ot **P**olynomial.

B. **N**o **P**akeup Exam.

C. **N**o **P**roblem.

D. **N**one of the a**P**ove.

NP stands for: **N**on-deterministically **P**olynomial.
I.e. Can be solved using a non-deterministic machine in polynomial time.

Assume that `TM` is a machine that can guess and verify an infinite number of solutions all at the same time (call `TM` a *non-deterministic* machine).

If a problem is verifiable in polynomial time, `TM` can solve the problem by guessing all the possible solutions and verifying them at once (in polynomial time!)

Definitions (Complexity Classes)

Class P.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class NP.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Class NP-Complete.

A decision problem is NP-Complete if:

- It is in NP.
- All problems in NP reduce to it in polynomial time.

How do we show that *all* problems in **NP** reduce to a certain problem???

Cook-Levin Theorem (1971)

Steve Cook
1982 Turing Award

Leonid Levin

All problems in **NP** poly-time reduce to SAT.

What is **SAT**?

slide by Kevin Wayne

Boolean Satisfiability (SAT)

Literal. A Boolean variable or its negation. x_i or $\overline{x_i}$

Clause. A disjunction of literals. $C_j = x_1 \vee \overline{x_2} \vee x_3$

Conjunctive normal form (CNF). A propositional formula Φ that is a conjunction of clauses. $\Phi = C_1 \wedge C_2 \wedge C_3 \wedge C_4$

SAT. Given a CNF formula Φ, does it have a satisfying truth assignment?

3-SAT. SAT where each clause contains exactly 3 literals (and each literal corresponds to a different variable).

Example What values for x_1, x_2, x_3 and x_4 satisfy the following formula?

$$\Phi = (\overline{x_1} \vee x_2 \vee x_3) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

Answer. x_1 = TRUE, x_2 = TRUE, x_3 = FALSE, x_4 = FALSE

adapted from a slide by Kevin Wayne

Boolean Satisfiability (SAT)

Key Facts.

- SAT is in NP.
  Given a formula and boolean values for the variables, it is easy to verify if these values satisfy the formula!

- It is not clear if SAT is also in P.
  - We can try all possible 2^N boolean assignments.
  - We don't know if a polynomial time solution exists.

- All problems in in NP reduce to SAT in polynomial time.
  - This is the Cook-Levin Theorem.
  - The details of the proof are beyond the scope of this course.
  - In a nutshell, Cook and Levin showed how any decision problem that is in NP can be converted (in polynomial time) to the problem of satisfying a boolean formula.
    (i.e. a digital circuit can be designed for it that has a polynomial number of gates)

adapted from a slide by Kevin Wayne

Reduction Example

optional

Graph Coloring reduces to SAT in polynomial time.

Assume that the problem is to check if the graph is 2-colorable.

1. Create the boolean variables:
   A_{red} , A_{blue} , B_{red} , B_{blue} , C_{red} , C_{blue}

2. Enforce that each vertex has one color:
   $(A_{red} \lor A_{blue}) \land \neg(A_{red} \land A_{blue})$ = TRUE
   $(B_{red} \lor B_{blue}) \land \neg(B_{red} \land B_{blue})$ = TRUE
   $(C_{red} \lor C_{blue}) \land \neg(C_{red} \land C_{blue})$ = TRUE

3. Enforce that no adjacent vertices have the same color:
   $\neg(A_{red} \land B_{red}) \land \neg(A_{blue} \land B_{blue})$ = TRUE
   $\neg(A_{red} \land C_{red}) \land \neg(A_{blue} \land C_{blue})$ = TRUE
   $\neg(B_{red} \land C_{red}) \land \neg(B_{blue} \land C_{blue})$ = TRUE

Can be easily converted to CNF.

The graph is 2-colorable if the above boolean expressions are satisfiable!

Quiz # 9

How do we show that a problem other than SAT is NP-Complete?

A. Be as clever as Cook and Levin and show how all problems in NP reduce to this new problem.

B. No need! SAT is the only NP-Complete Problem!

C. None of the above.

Quiz # 9

How do we show that a problem other than SAT is NP-Complete?

A. Be as clever as Cook and Levin and show how all problems in NP reduce to this new problem.

B. No need! SAT is the only NP-Complete Problem!

C. None of the above.

To show that a problem is NP-Complete:

1. Show that it is in NP.
2. Show that an NP-Complete problem reduces to it in polynomial time!

If all problems in NP poly-time reduce to A and A poly-time reduces to B, then all problems in NP poly-time reduce to B!

SAT is not The Only NP-Complete Problem!

Dick Karp
1985 Turing Award

Key Finding. SAT poly-time reduces to many problems!

Implication. All of these problems are NP-Complete!

slide by Kevin Wayne

SAT is not The Only NP-Complete Problem!

adapted from a slide by Kevin Wayne

Steve Cook

Leonid Levin

Dick Karp

All of these problems are NP-complete.

A provably efficient algorithm for *any one* of them would yield a provably efficient algorithm for *all* of them

World View if P != NP

Again ... Two Possible World Views

vs

If P $\neq$ NP

If P = NP

NP-Completeness (Proof Examples)

`ILP` (*binary* Integer Linear Programming)

Given a system of inequalities, find a `0-1` solution.

$$
\begin{array}{rcrcrcr}
 & & x_1 & + & x_2 & \geq & 1 \\
x_0 & & & + & x_2 & \geq & 1 \\
x_0 & + & x_1 & + & x_2 & \leq & 2
\end{array}
$$

Example. A solution for the above is:

$$x_0 = 1, \quad x_1 = 1, \quad x_2 = 0$$

Task. Show that `ILP` is NP-Complete.

1. `ILP` is in NP.

Given values for the variables, we can verify in polynomial time if the inequalities are true.

2. SAT poly-time reduces to `ILP`.

$$
\begin{array}{ccccccccl}
\bar{x}_1 & \vee & x_2 & \vee & x_3 & & & & = \text{TRUE} \\
x_1 & \vee & \bar{x}_2 & \vee & x_3 & & & & = \text{TRUE} \\
\bar{x}_1 & \vee & \bar{x}_2 & \vee & \bar{x}_3 & & & & = \text{TRUE} \\
\bar{x}_1 & \vee & \bar{x}_2 & & & \vee & x_4 & & = \text{TRUE} \\
 & & \bar{x}_2 & \vee & x_3 & \vee & x_4 & & = \text{TRUE}
\end{array}
$$

Example SAT instance

$$
\begin{array}{rcrcrcrcrcl}
(1-x_1) & + & & & x_2 & + & & & x_3 & & \geq 1 \\
x_1 & + & (1-x_2) & + & & & & & x_3 & & \geq 1 \\
(1-x_1) & + & (1-x_2) & + & (1-x_3) & & & & & & \geq 1 \\
(1-x_1) & + & (1-x_2) & & & & & + & x_4 & & \geq 1 \\
 & & (1-x_2) & + & & & x_3 & + & x_4 & & \geq 1
\end{array}
$$

Equivalent `ILP` instance.

Examples by Kevin Wayne

NP-Completeness (Proof Examples)

FIGURE 1 - Complete Problems

RICHARD M. KARP

Dick Karp (1972)
1985 Turing Award

NP-Completeness (Proof Examples)

`TRAVELING SALESMAN PROBLEM (TSP)`

Given a complete weighted graph G, does G contain a simple circuit C that visits each node exactly once of *length* $\leq T$?

Task. Show that `TSP` is `NP`-Complete.

1. Show that `TSP` is in `NP`. ← straight-forward
2. `HAMILTONIAN` poly-time reduces to `TSP`.

G

Input to the `HAMILTONIAN`

G'

Input to `TSP`

G has a hamiltonian cycle iff G' has a tour of length V

Add edge (u, v) with weight **1** if (u, v) is in G.
Add edge (u, v) with weight **2** if (u, v) is not in G.

NP-Completeness (Proof Examples)

optional

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?

Example. Black vertices form an independent set of size 5

Task. Show that `IS` is `NP`-Complete.

1. `IS` is in `NP`.

Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

NP-Completeness (Proof Examples)

optional

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?

Example. Black vertices form an independent set of size 5

Task. Show that `IS` is `NP`-Complete.

1. `IS` is in `NP`.

Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. `SAT` poly-time reduces to `IS`.

Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

optional

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?

Example. Black vertices form an independent set of size 5

Task. Show that `IS` is `NP`-Complete.

1. `IS` is in `NP`.

   Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. `SAT` poly-time reduces to `IS`.

   - Create a node for each literal in each clause.

Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

optional

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?

Example. Black vertices form an independent set of size 5

Task. Show that `IS` is `NP`-Complete.

1. `IS` is in `NP`.

Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. `SAT` poly-time reduces to `IS`.

- Create a node for each literal in each clause.
- Connect each node to the literals in the same clause.

Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

optional

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?

Example. Black vertices form an independent set of size 5

Task. Show that `IS` is `NP`-Complete.

1. `IS` is in `NP`.

Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. `SAT` poly-time reduces to `IS`.

- Create a node for each literal in each clause.
- Connect each node to the literals in the same clause.
- Connect each literal to its negation.

Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

optional

`INDEPENDENT-SET (IS)`

Given a graph and an integer k, is there a subset of k vertices such that no two vertices are adjacent?

Example. Black vertices form an independent set of size 5

Task. Show that `IS` is `NP`-Complete.

1. `IS` is in `NP`.

Given a set S of vertices in G, we can verify in polynomial time if any two are adjacent and if $|S| = k$.

2. `SAT` poly-time reduces to `IS`.

- Create a node for each literal in each clause.
- Connect each node to the literals in the same clause.
- Connect each literal to its negation.
- The expression is satisfiable iff there is an independent set of size = the number of clauses.

Example. $(x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \bar{x}_2 \vee x_3)$

NP-Completeness (Proof Examples)

optional

`VERTEX-COVER (VC)`

Given a graph and an integer k, is there a subset of k vertices such that each edge is incident to at least one vertex in the subset?

Task. Show that VC is NP-Complete.

Example. Black vertices form a vertex cover of size 5

NP-Completeness (Proof Examples)

optional

`VERTEX-COVER (VC)`

Given a graph and an integer k, is there a subset of k vertices such that each edge is incident to at least one vertex in the subset?

Example. Black vertices form a vertex cover of size 5

Task. Show that VC is NP-Complete.

1. VC is in NP.

Given a set S of vertices in G, we can verify in polynomial time if each edge in the graph is incident to a vertex in S and if $|S| = k$.

NP-Completeness (Proof Examples)

optional

`VERTEX-COVER (VC)`

Given a graph and an integer k, is there a subset of k vertices such that each edge is incident to at least one vertex in the subset?

Example. Black vertices form a vertex cover of size 5

Task. Show that VC is NP-Complete.

1. VC is in NP.

Given a set S of vertices in G, we can verify in polynomial time if each edge in the graph is incident to a vertex in S and if $|S| = k$.

2. `INDEPENDENT-SET` poly-time reduces to `VERTEX-COVER`.

S is an independent set of size k iff $V - S$ is a vertex cover of size $n - k$.

Vertex Cover of size 4

Independent Set of size 5

Quiz # 10

Are there problems that are in **NP** but are not in **P** and are not **NP**-Complete.

A. Yes.

B. No.

C. None of the above.

Quiz # 10

Are there problems that are in **NP** but are not in **P** and are not **NP**-Complete.

A. Yes.

B. No.

C. None of the above.

Yes if **P** $\neq$ **NP**.

No if **P** = **NP**.

There are, however, problems in NP that we could not yet prove to be in P and could not also prove to be NP-Complete!

Examples. Integer Factoring and Graph Isomorphism.

Definitions (Complexity Classes)

Class **P**.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class **NP**.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Class **NP-Complete**.

A decision problem is NP-Complete if:

- It is in NP.
- All problems in NP reduce to it in polynomial time.

Class **NP-Hard**.

A problem is NP-Hard if all problems in NP reduce to it in polynomial time.
(*at least as hard as the hardest problems in* NP)

Definitions (Complexity Classes)

Class P.

A decision problem is in P if it is solvable in polynomial time
(i.e. in $O(n^c)$, where n is the input size and c is a constant)

Class NP.

A decision problem is in NP if it is verifiable in polynomial time.
(Given an instance I or a problem P and a witness W for the solution, can we verify in polynomial time if W proves that the answer for I is yes?)

Class NP-Complete.

A decision problem is NP-Complete if:

- It is in NP.
- All problems in NP reduce to it in polynomial time.

Examples.

- All NP-Complete Problems.
- TSP Optimization.
- Finding the Longest Simple Path.

Class NP-Hard.

A problem is NP-Hard if all problems in NP reduce to it in polynomial time.
(*at least as hard as the hardest problems in* NP)

Two Possible World Views

If P ≠ NP

vs

If P = NP

Living with Intractability

When you encounter an NP-complete problem

- It is safe to assume that it is intractable.
- What to do?

does not have an algorithm that solve all instances in polynomial time.

Four successful approaches

- Don't try to solve intractable problems.
- Try to solve real-world problem instances.
- Look for approximate solutions (not discussed in this lecture).
- Exploit intractability.

slide by Kevin Wayne

Living with Intractability: Don't Try To Solve It!

Knows no theory

I can't find an efficient algorithm.
I guess I'm just to dumb.

Knows computability

I can't find an efficient algorithm,
because no such algorithm is possible!

Knows intractability

I can't find an efficient algorithm,
but neither can all these famous people!

slide by Kevin Wayne

Living with Intractability: Solve Real-World Instances

Observations

- Worst-case inputs may not occur for practical problems.
- Instances that do occur in practice may be easier to solve.

Reasonable approach: relax the condition of *guaranteed* poly-time algorithms.

SAT

- *Chaff* solves real-world instances with 10,000+ variables.
- Princeton senior independent work (!) in 2000.

TSP

- *Concorde* routinely solves large real-world instances.
- 85,900-city instance solved in 2006.

ILP

- *CPLEX* routinely solves large real-world instances.
- Routinely used in scientific and commercial applications.

TSP solution for 13,509 US cities

slide by Kevin Wayne