NP-COMPLETE PROBLEMS

David Kauchak – Spring 2024

1

Admin

Assignment 10

Assignment 11

2

Final exam timing

Time-limited take-home exam

Can take whenever you want and you'll have 3 hours to take it

Will be available Monday, April 29

Must be submitted by
- Seniors: 12pm (noon) on Thursday, May 2
- Everyone else: 11:59pm on Wednesday, May 8

3

Run-time analysis

We've spent a lot of time in this class putting algorithms into specific run-time categories:

- O(log n)
- O(n)
- O(n log n)
- $O(n^2)$
- O(n log log n)
- $O(n^{1.67})$
- …

When I say an algorithm is O(f(n)), what does that mean?

4

Tractable vs. intractable problems

trac·ta·ble (trăk'tə-bəl)
adj.
1. Easily managed or controlled; governable.
2. Easily handled or worked; malleable.

What is a "tractable" problem?

5

Tractable vs. intractable problems

trac·ta·ble (trăk'tə-bəl)
adj.
1. Easily managed or controlled; governable.
2. Easily handled or worked; malleable.

Tractable problems can be solved in O(f(n)) where f(n) is a polynomial

We'll call **P,** the set of tractable problems

6

Tractable vs. intractable problems

trac·ta·ble (trăk'tə-bəl)
adj.
1. Easily managed or controlled; governable.
2. Easily handled or worked; malleable.

What about…

$O(n^{100})$?

$O(n^{\log \log \log \log n})$?

7

Tractable vs. intractable problems

trac·ta·ble (trăk'tə-bəl)
adj.
1. Easily managed or controlled; governable.
2. Easily handled or worked; malleable.

Technically $O(n^{100})$ is tractable by our definition

Why don't we worry about problems like this?

8

Tractable vs. intractable problems

trac·ta·ble (trăk'tə-bəl)
adj.
1. Easily managed or controlled; governable.
2. Easily handled or worked; malleable.

Technically $O(n^{100})$ is tractable by our definition

- Few practical problems result in solutions like this
- Once a polynomial time algorithm exists, more efficient algorithms are usually found
- Polynomial algorithms are amenable to parallel computation

9

Solvable vs. unsolvable problems

solv·a·ble (sŏl'və-bəl, sôl'-)
adj.
Possible to solve: *solvable problems; a solvable riddle.*

What is a "solvable" problem?

10

Solvable vs. unsolvable problems

solv·a·ble (sŏl'və-bəl, sôl'-)
adj.
Possible to solve: *solvable problems; a solvable riddle.*

A problem is solvable if given enough (i.e. finite) time you could solve it

11

Sorting

Given n integers, sort them from smallest to largest.

Tractable/intractable?

Solvable/unsolvable?

12

Sorting

Given n integers, sort them from smallest to largest.

Solvable and tractable:
Mergesort: $\Theta(n \log n)$

13

Enumerating all subsets

Given a set of n items, enumerate all possible subsets.

Tractable/intractable?

Solvable/unsolvable?

14

Enumerating all subsets

Given a set of n items, enumerate all possible subsets.

Solvable, but intractable: $\Theta(2^n)$ subsets

For large n this will take a very, very long time

15

Halting problem

Given an arbitrary algorithm/program and a particular input, will the program terminate?

Tractable/intractable?

Solvable/unsolvable?

16

Halting problem

Given an arbitrary algorithm/program and a particular input, will the program terminate?

Unsolvable ☹

17

Integer solution?

Given a polynomial equation, are there *integer* values of the variables such that the equation is true?

$$x^3yz + 2y^4z^2 - 7xy^5z = 6$$

Tractable/intractable?

Solvable/unsolvable?

18

Integer solution?

Given a polynomial equation, are there *integer* values of the variables such that the equation is true?

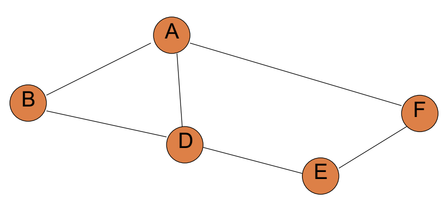
$$x^3yz + 2y^4z^2 - 7xy^5z = 6$$

Unsolvable ☹

19

Hamiltonian cycle

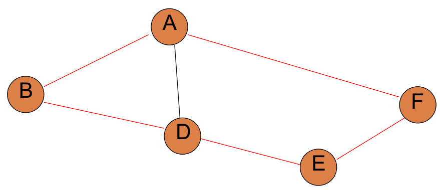
Given an undirected graph G=(V, E), a hamiltonian cycle is a cycle that visits every vertex V exactly once



20

Hamiltonian cycle

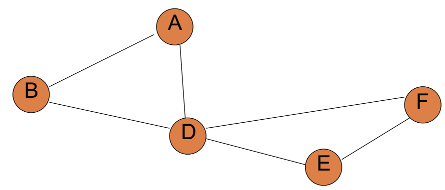
Given an undirected graph G=(V, E), a hamiltonian cycle is a cycle that visits every vertex V exactly once



21

Hamiltonian cycle

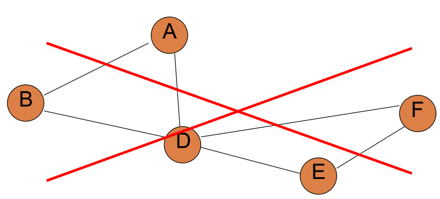
Given an undirected graph G=(V, E), a hamiltonian cycle is a cycle that visits every vertex V exactly once



22

Hamiltonian cycle

Given an undirected graph G=(V, E), a hamiltonian cycle is a cycle that visits every vertex V exactly once



23

Hamiltonian cycle

Given an undirected graph, does it contain a hamiltonian cycle?

Tractable/intractable?

Solvable/unsolvable?

24

Hamiltonian cycle

Given an undirected graph, does it contain a hamiltonian cycle?

Solvable: Enumerate all possible paths (i.e. include an edge or don't) check if it's a hamiltonian cycle

How would we do this check exactly, specifically given a graph and a path?

25

Checking hamiltonian cycles

```
HAM-CYCLE-VERIFY(G, p)
for i ← 1 to |V|
        visited[i] ← false
n ← length[p]
if p_1 ≠ p_n or n ≠ |V| + 1
        return false
visited[p_1] ← true
for i ← 1 to n − 1
        if visited[p_i]
                return false
        if (p_i, p_{i+1}) ∉ E
                return false
        visited[p_i] ← true
for i ← 1 to |V|
        if !visited[i]
                return false
return true
```

26

Checking hamiltonian cycles

```
HAM-CYCLE-VERIFY(G, p)
for i ← 1 to |V|
        visited[i] ← false
n ← length[p]
if p_1 ≠ p_n or n ≠ |V| + 1
        return false
visited[p_1] ← true
for i ← 1 to n − 1
        if visited[p_i]
                return false
        if (p_i, p_{i+1}) ∉ E
                return false
        visited[p_i] ← true
for i ← 1 to |V|
        if !visited[i]
                return false
return true
```

Make sure the path starts and ends at the same vertex and is the right length

Can't revisit a vertex

Edge has to be in the graph

Check if we visited all the vertices

27

NP problems

NP is the set of problems that can be *verified* in polynomial time

A problem can be verified in polynomial time if you can check that a given solution is correct in polynomial time

(NP is an abbreviation for non-deterministic polynomial time)

28

Checking hamiltonian cycles

```
HAM-CYCLE-VERIFY(G, p)
for i ← 1 to |V|
        visited[i] ← false
n ← length[p]
if p_1 ≠ p_n or n ≠ |V| + 1
        return false
visited[p_1] ← true
for i ← 1 to n − 1
        if visited[p_i]
                return false
        if (p_i, p_{i+1}) ∉ E
                return false
        visited[p_i] ← true
for i ← 1 to |V|
        if !visited[i]
                return false
return true
```

Running time?

O(V) adjacency matrix
O(V+E) adjacency list

What does that say about the hamilonian cycle problem?

It belongs to NP

29

NP problems

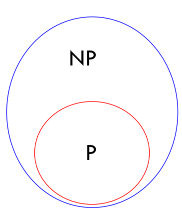
Why might we care about NP problems?

- If we can't verify the solution in polynomial time then an algorithm cannot exist that determines the solution in this time (why not?)
- All algorithms with polynomial time solutions are in NP

The NP problems that are currently not solvable in polynomial time *could in theory be solved in polynomial time*

30

P and NP



Big-O allowed us to group algorithms by run-time

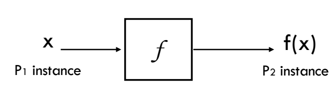
Today, we're talking about sets of problems grouped by how easy they are to solve

31

Reduction function

Given two problems P_1 and P_2 a *reduction function*, $f(x)$, is a function that transforms a problem instance x of type P_1 to a problem instance of type P_2

such that: a solution to x exists for P_1 iff a solution for $f(x)$ exists for P_2



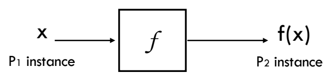
32

Reduction function

Where have we seen reductions before?

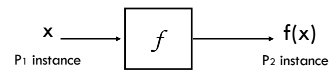
- Bipartite matching reduced to flow problem
- All pairs shortest path *through a particular vertex* reduced to single source shortest path

Why are they useful?

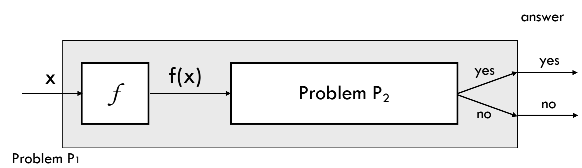


33

Reduction function

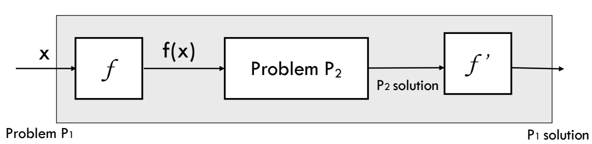


Allow us to solve P_1 problems if we have a solver for P_2



34

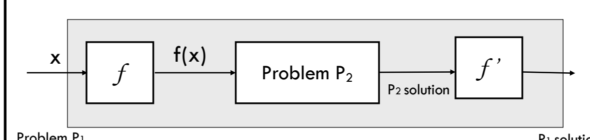
Reduction function



Most of the time we'll worry about yes no question, however, if we have more complicated answers we often just have to do a little work to the solution to the problem of P_2 to get the answer

35

Reduction function: Example



P1 = Bipartite matching

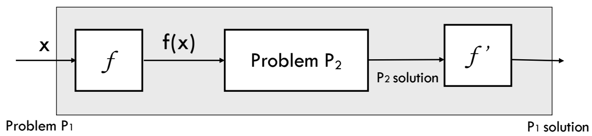
P2 = Network flow

Reduction function (f): Given *any* bipartite matching problem turn it into a network flow problem

What is *f* and what is *f'*?

36

Reduction function: Example



P1 = Bipartite matching
P2 = Network flow

Reduction function (f): Given *any* bipartite matching problem turn it into a network flow problem

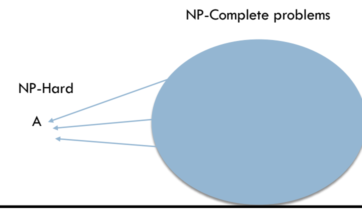
A reduction function reduces problems instances

37

NP-Complete

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)



38

NP-Complete

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

A $\le_p$B: A is polynomial time reducible to B

39

NP-Complete

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

Problem A is NP-complete if:
1. A ∈ NP
2. A ∈ NP-Hard: X $\le_p$A for all X ∈ NP-Complete

40

NP-Complete

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

The hamiltonian cycle problem is NP-complete

What are the implications of this?
What does this say about how hard the hamiltonian cycle problem is compared to other NP-complete problems?

41

NP-Complete

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

The hamiltonian cycle problem is NP-complete

It's *at least as hard* as *any* of the other NP-complete problems

42

NP-Complete

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

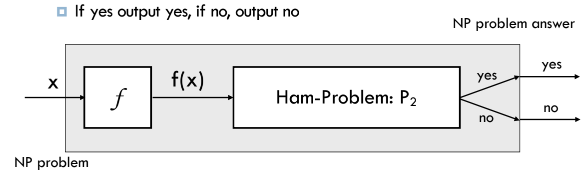
If I found a polynomial-time solution to the hamiltonian cycle problem, what would this mean for the other NP-complete problems?

43

NP-complete

If a polynomial-time solution to the hamiltonian cycle problem is found, we would have a polynomial time solution to *any* NP-complete problem

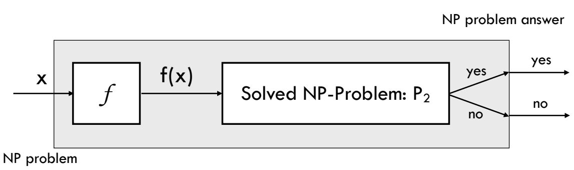
- Take the input of the problem
- Convert it to the hamiltonian cycle problem (by definition, we know we can do this in polynomial time)
- Solve it
- If yes output yes, if no, output no



44

NP-complete

Similarly, if we found a polynomial time solution to *any* NP-complete problem we'd have a solution to *all* NP-complete problems



45

NP-complete problems

Longest path

Given a graph G with nonnegative edge weights does a simple path exist from *s* to *t* with weight at least *g*?

46

NP-complete problems

3D matching

Bipartite matching: given two sets of things and pair constraints, find a matching between the sets

3D matching: given three sets of things and triplet constraints, find a matching between the sets of size at least K

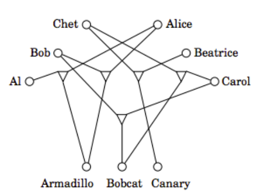


Figure from Dasgupta et. al 2008

48

P vs. NP

Polynomial time solutions exist	NP-complete (and no polynomial time solution currently exists)
Shortest path	Longest path
Bipartite matching	3D matching
Linear programming	Integer linear programming
Minimum cut	Balanced cut
…	…

49

Proving NP-completeness

A problem is *NP-complete* if:

1. it can be verified in polynomial time (i.e. in NP)
2. *any* NP-complete problem can be reduced to the problem in polynomial time (is NP-hard)

Ideas?

50

Proving NP-completeness

Given a problem NEW to show it is NP-Complete

1. Show that NEW is in NP
   a. Provide a verifier
   b. Show that the verifier runs in polynomial time
2. Show that NEW is NP-Hard (i.e., all NP-complete problems are reducible to NEW in polynomial time)
   a. Describe a reduction function *f* from a known NP-Complete problem to NEW
   b. Show that *f* runs in polynomial time
   c. Show that a solution exists to the NP-Complete problem IFF a solution exists *to the NEW problem generate by f*

51

Proving NP-completeness

Show that a solution exists to the NP-Complete problem IFF a solution exists *to the NEW problem generate by f*

- Assume we have an NP-Complete problem instance that has a solution, show that the NEW problem instance generated by *f* has a solution

  yes ----------------------→ yes

- Assume we have a problem instance of NEW *generated by f* that has a solution, show that we can derive a solution to the NP-Complete problem instance

  yes ←---------------------- yes

Other ways of proving the IFF, but this is often the easiest

52

Proving NP-completeness

Show that all NP-complete problems are reducible to NEW in polynomial time

Why is it sufficient to show that one NP-complete problem reduces to the NEW problem?

53

Proving NP-completeness

Show that all NP-complete problems are reducible to NEW in polynomial time

All others can be reduced to NEW by first reducing to the one problem, then reducing to NEW. Two polynomial time reductions is still polynomial time!

54

Proving NP-completeness

Show that all NP-complete problems are reducible to NEW in polynomial time

Show that *any* NP-complete problem is reducible to NEW in polynomial time

BE CAREFUL!

~~Show that NEW is reducible to any NP-complete problem in polynomial time~~

55

NP-complete: 3-SAT

A boolean formula is in *n-conjunctive normal form* (*n*-CNF) if:

- it is expressed as an AND of clauses
- where each clause is an OR of no more than *n* variables

$$(a \vee \neg a \vee \neg b) \wedge (c \vee b \vee d) \wedge (\neg a \vee \neg c \vee \neg d)$$

3-SAT: Given a 3-CNF boolean formula, is it satisfiable?

3-SAT is an NP-complete problem

56

NP-complete: SAT

Given a boolean formula of *n* boolean variables joined by *m* connectives (AND, OR or NOT) is there a setting of the variables such that the boolean formula evaluate to true?

$$(a \wedge b) \vee (\neg a \wedge \neg b)$$

$$((\neg(b \vee \neg c) \wedge a) \vee (a \wedge b \wedge c)) \wedge c \wedge \neg b$$

Is SAT an NP-complete problem?

57

NP-complete: SAT

Given a boolean formula of *n* boolean variables joined by *m* connectives (AND, OR or NOT) is there a setting of the variables such that the boolean formula evaluate to true?

$$((\neg(b \vee \neg c) \wedge a) \vee (a \wedge b \wedge c)) \wedge c \wedge \neg b$$

1. Show that SAT is in NP
   a. Provide a verifier
   b. Show that the verifier runs in polynomial time
2. Show that NEW is NP-Hard (i.e., all NP-complete problems are reducible to NEW in polynomial time)
   a. Describe a reduction function *f* from a known NP-Complete problem to SAT
   b. Show that *f* runs in polynomial time
   c. Show that a solution exists to the NP-Complete problem IFF a solution exists *to the SAT problem generate by f*

58

NP-Complete: SAT

1. Show that SAT is in NP
   a. Provide a verifier
   b. Show that the verifier runs in polynomial time

Verifier: A solution consists of an assignment of the variables
- If clause is a single variable:
  - return the value of the variable
- otherwise
  - for each clause:
    - call the verifier recursively
    - compute a running solution

polynomial run-time?

59

NP-Complete: SAT

Verifier: A solution consists of an assignment of the variables
- If clause is a single variable:
  - return the value of the variable
- otherwise
  - for each clause:
    - call the verifier recursively
    - compute a running solution

linear time

- at most a linear number of recursive calls (each call makes the problem smaller and no overlap)
- overall polynomial time

60

NP-Complete: SAT

2. Show that all NP-complete problems are reducible to SAT in polynomial time
   a. Describe a reduction function *f* from a known NP-Complete problem to SAT
   b. Show that *f* runs in polynomial time
   c. Show that a solution exists to the NP-Complete problem IFF a solution exists *to the SAT problem generate by f*

Reduce 3-SAT to SAT:
- Given an instance of 3-SAT, turn it into an instance of SAT

Reduction function:
- DONE ☺

- Runs in constant time! (or linear if you have to copy the problem)

61

NP-Complete: SAT

Show that a solution exists to the NP-Complete problem IFF a solution exists *to the NEW problem generate by f*

- Assume we have an NP-Complete problem instance that has a solution, show that the NEW problem instance generated by *f* has a solution
- Assume we have a problem instance of NEW *generated by f* that has a solution, show that we can derive a solution to the NP-Complete problem instance

---

- Assume we have a 3-SAT problem with a solution:
  - Because 3-SAT problems are a subset of SAT problems, then the SAT problem will also have a solution
- Assume we have a problem instance generated by our reduction with a solution:
  - Our reduction function simply does a copy, so it is already a 3-SAT problem
  - Therefore the variable assignment found by our SAT-solver will also be a solution to the original 3-SAT problem

62

NP-Complete problems

Why do we care about showing that a problem is NP-Complete?

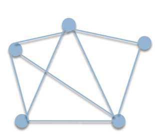
- We know that the problem is hard (and we probably won't find a polynomial time exact solver)
- We may need to compromise:
  - reformulate the problem
  - settle for an approximate solution
- Down the road, if a solution is found for an NP-complete problem, then we'd have one too…

63

CLIQUE

A *clique* in an undirected graph G = (V, E) is a subset V' ⊆ V of vertices that are fully connected, i.e. every vertex in V' is connected to every other vertex in V'

CLIQUE problem: Does G contain a clique of size k?



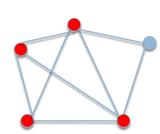
Is there a clique of size 4 in this graph?

64

CLIQUE

A *clique* in an undirected graph G = (V, E) is a subset V' ⊆ V of vertices that are fully connected, i.e. every vertex in V' is connected to every other vertex in V'

CLIQUE problem: Does G contain a clique of size k?



CLIQUE is an NP-Complete problem

65

HALF-CLIQUE

Given a graph G, does the graph contain a clique containing exactly half the vertices?

Is HALF-CLIQUE an NP-complete problem?

66

Is Half-Clique NP-Complete?

1. Show that Half-Clique is in NP
   a. Provide a verifier
   b. Show that the verifier runs in polynomial time
2. Show that Half-Clique is NP-Hard (i.e., all NP-complete problems are reducible to Half-Clique in polynomial time)
   a. Describe a reduction function *f* from a known NP-Complete problem to Half-Clique
   b. Show that *f* runs in polynomial time
   c. Show that a solution exists to the NP-Complete problem IFF a solution exists *to the Half-Clique problem generate by f*

Given a graph G, does the graph contain a clique containing exactly half the vertices?

67