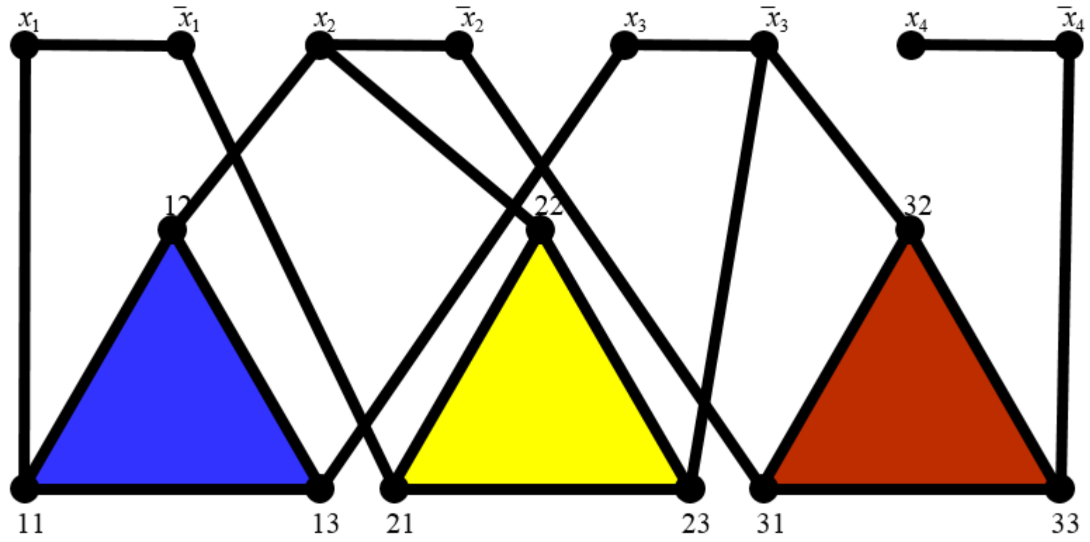
Complexity Class and Approximation Algorithms

Algorithms & Data Structures
ITCS 6114/8114

Dr. Dewan Tanvir Ahmed
Department of Computer Science
University of North Carolina at Charlotte

NP-Complete Problems



Outline

- We discuss some hard problems:
  - how hard? (computational complexity)
  - what makes them hard?
  - any solutions?

Definitions

- Optimization problems:
  - An optimization problem takes input and find the optimal solutions.
- Example
  - Given G, determine the minimal number of colors (and the coloring) such that no adjacent vertices are assigned the same color.

Definitions (cont.)

- Decision Problems:
  - A decision problem takes input and yields two possible answers, “yes” and “no”.
- Example
  - Given graph G=(V,E) and a positive integer k, is there a coloring of G using at most k colors (and no adjacent vertices are assigned the same color)?

Definitions (cont.)

- For classification, problems are defined in terms of yes-or-no (decision) version.
- We can do so because these two versions are "equally" hard, while optimization problems are seemingly harder.

Computational complexity

Polynomially bounded:

- An algorithm is polynomially bounded if its worst-case complexity is bounded by a polynomial function of the input size.
- A problem is polynomially bounded if there is a polynomially bounded algorithm for it.

Computational complexity

The class P is the class of decision problems that are polynomially bounded.

- While class P may seem to be too broad,
  - It is still a useful classification because problems not in P would be intractable.
- The class P is closed under compositions
  - Sequential blocks: $p1(n) + p2(n)$
  - Nested subroutines $p1(p2(n))$

Computational complexity

The class NP is the class of decision problems for which there is a polynomially bounded nondeterministic algorithm.

- NP stands for “Nondeterministic Polynomial”, and should not be mistaken as “non-polynomial”.

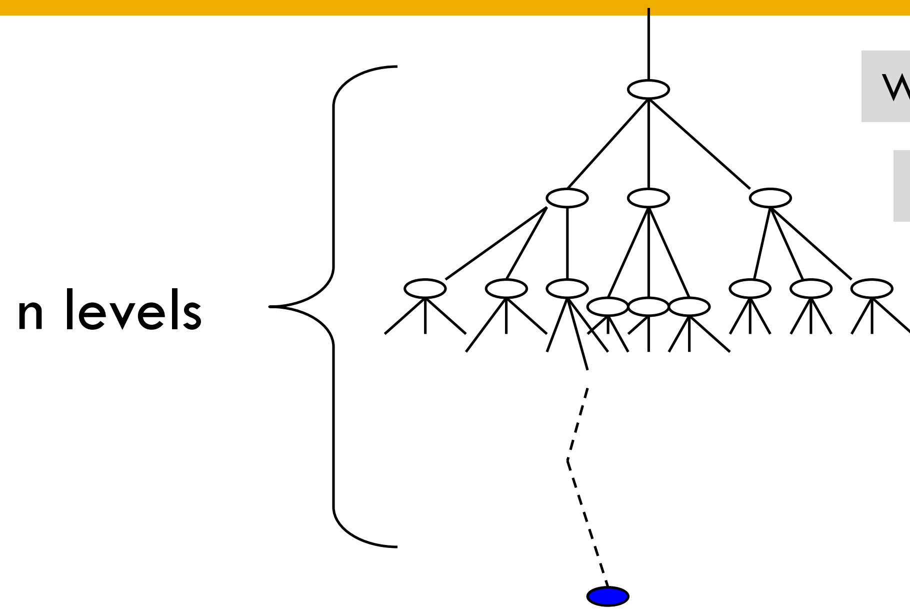
Computational complexity

A non-deterministic algorithm

- The non-deterministic "guessing" phase.
  - Some completely arbitrary string s, "proposed solution"
  - Each time the algorithm is run the string may differ
- The deterministic "verifying" phase.
  - A deterministic algorithm takes the input of the problem and the proposed solution s
  - And a return value true or false
- The output step.
  - If the verifying phase returned true, the algorithm outputs yes. Otherwise, there is no output.

- In practice, a problem is in NP if its solutions can be verified by a polynomial algorithm.

Example: Where is the gas station?



Which route to take? Take the leftmost.

Which route to take? Take the rightmost

Which route to take? Take the middle.

A nondeterministic algorithm always makes the right guess among multi options. For the problem above, a nondeterministic algorithm takes n guesses and n checkings to find the gas station. In contrast, a deterministic algorithm using a breath-first search would have to take $O(3^n)$ steps.

Classes beyond NP

- PSPACE problems
  - e.g. Quantified Boolean formula problem
  - can be solved by using reasonable amount of memory (bounded as a polynomial of the input size), without regard to time the solution takes. Or
  - PSPACE is the set of all decision problems that can be solved by a Turing machine using a polynomial amount of space.
- EXPTIME problems
  - e.g. Chees, Checkers, Go
  - that can be solved in exponential time.

$P \subseteq NP$

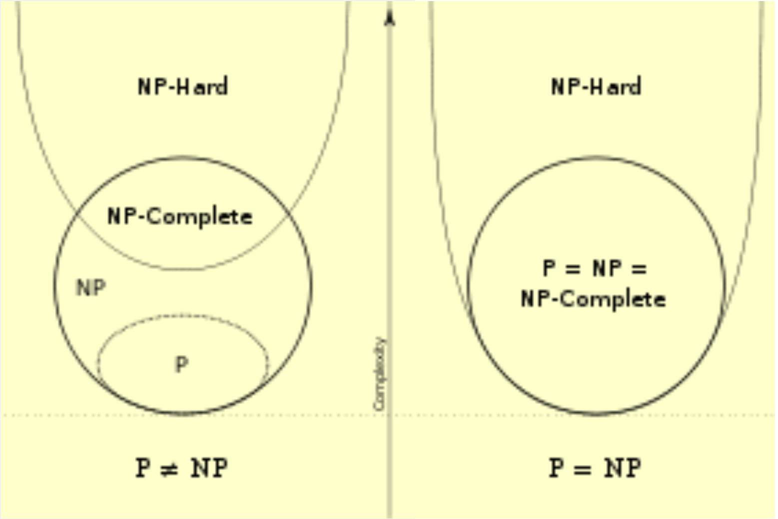
Theorem: $P \subseteq NP$.

Proof

- If a problem is in P, there is a polynomial algorithm A.
- With minor modification, we can have a nondeterministic polynomial algorithm A': the guessing phase is trivial, i.e., do nothing; and the verifying phase is just A.
- Therefore, any problem in P is also in NP. ■

NP-Complete Problems

- NP-Complete Problems are hardest ones in the class NP.
  - If an NPC problem could be solved in polynomial time, so could be all problems in NP.
  - How do we know a problem is NPC?
  - Two steps:
    - Prove it is in NP.
    - Prove it is NP-hard.
  - A problem is NP-hard if it is as hard as or even harder than any problem in NP.



Dealing with Hard Problems

- What to do when we find a problem that looks hard…

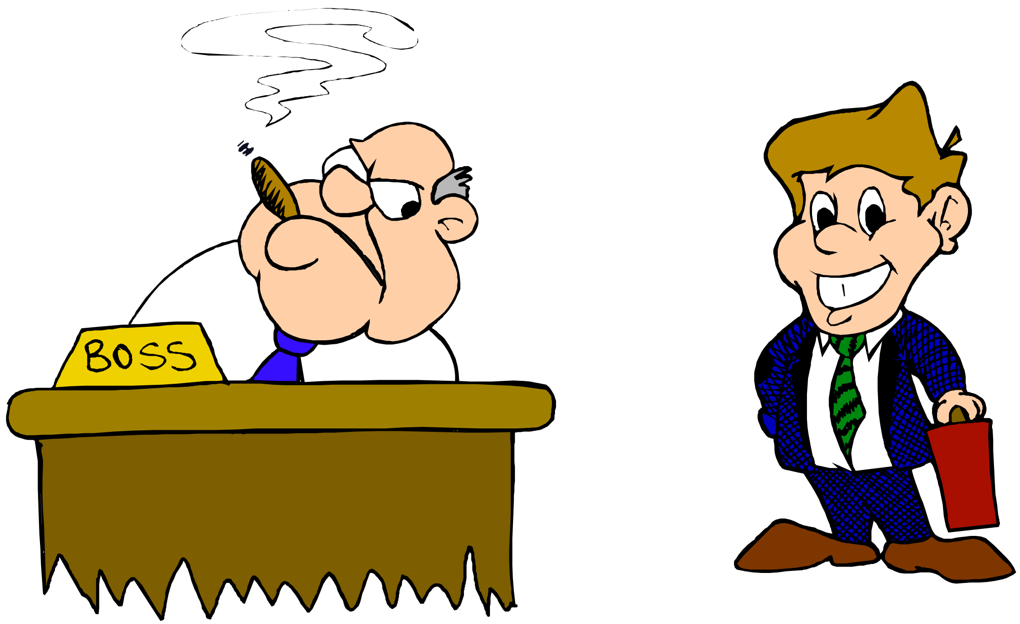
I couldn't find a polynomial-time algorithm;

I guess I'm too dumb.

(cartoon inspired by [Garey-Johnson, 79])

Dealing with Hard Problems

- Sometimes we can prove a strong lower bound... (but not usually)

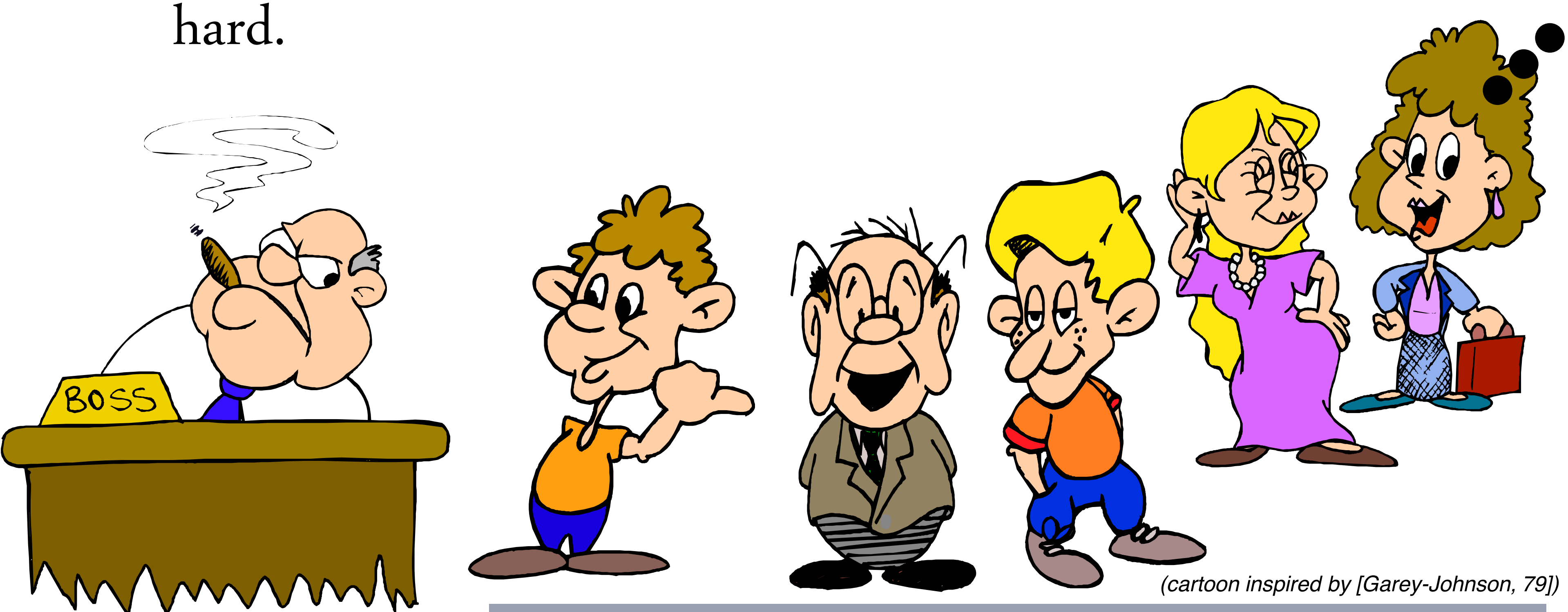


I couldn't find a polynomial-time algorithm,
because no such algorithm exists!

(cartoon inspired by [Garey-Johnson, 79])

Dealing with Hard Problems

- NP-completeness let's us show collectively that a problem is hard.



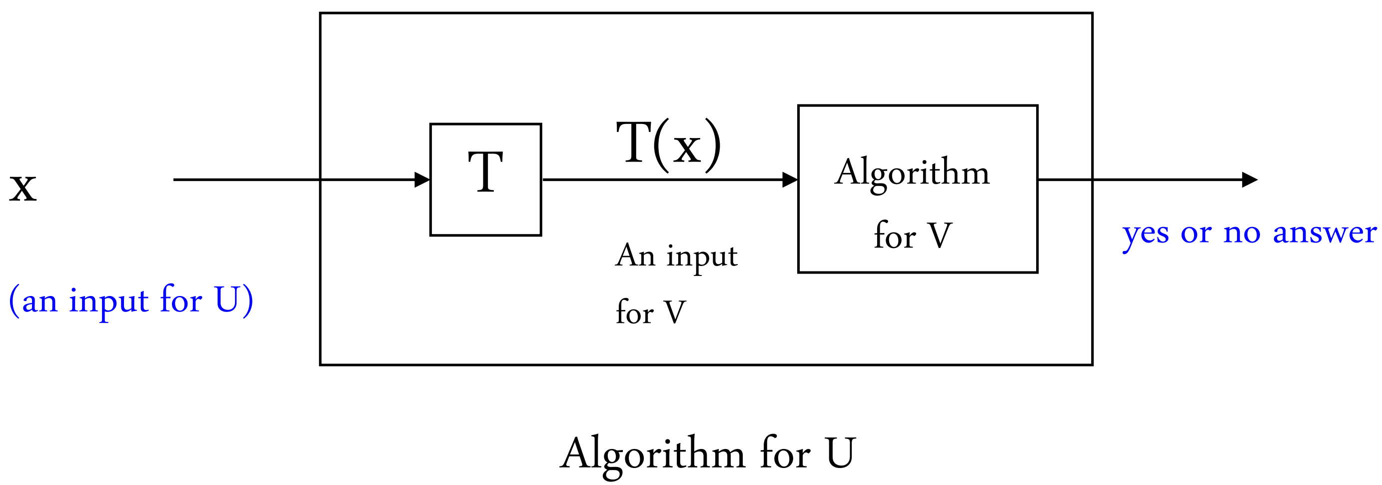
(cartoon inspired by [Garey-Johnson, 79])

I couldn't find a polynomial-time algorithm, but neither could all these other smart people.

Polynomial Reductions

- A formal way to say "as hard as".
  - Reduction is a transformation from one problem to another.
  - Formally, let T be a function from input set for a decision problem U into the input set for a decision problem V, such that
    - For every string x, if x is a yes input for U, then $T(x)$ is a yes input for V.
    - For every string x, if x is a no input for U, then $T(x)$ is a no input for V. (Or equivalently, if $T(x)$ is a yes input for V, then x is a yes input for U).
  - T is a polynomial reduction when it can be computed in polynomially bounded time.
  - Problem U is polynomially reducible to V, denoted as $U \leq_p V$, if there exists a polynomial reduction from U to V.

Polynomial Reductions



If $U \leq_p V$ and V is in P, then U is in P.

Theorem: If U $\leq_{\mathrm{p}}$ V and V is in P, then U is in P.

Proof:

- Let p be a polynomial bound on the computation of T, and q a polynomial bound on an algorithm A for V.
- Let x be an input for U of size n.
- The size of $T(x)$ is at most $p(n)$, and algorithm A on $T(x)$ takes at most $q(p(n))$ steps.
- The total amount of work to transform x to $T(x)$ and then use V's algorithm to get the correct answer for U on x is $p(n) + q(p(n))$, a polynomial in n. ■

Definition: NP-hard

- A problem is NP-hard if an algorithm for solving it can be translated into one for solving any NP-problem (nondeterministic polynomial time) problem.
- NP-hard therefore means "at least as hard as any NP-problem," although it might, in fact, be harder.
- A NP-hard problem needs not to be in NP, but if it does, it becomes NP-Complete, i.e., hardest one in NP.
- NP-Complete problems form an equivalent class under polynomial reductions: if $U, V \in NPC$, then $U \leq pV$ and $V \leq pU$.

Cook's Theorem

- Theorem: The satisfiability problem is NP-Complete.
  - Conjunctive normal form (CNF) of a propositional formula consists a sequence of clauses separated by Boolean AND operator ($\wedge$), where a clause is a sequence of literals separated by Boolean OR operators ($\vee$).
- For example:
  - $(p \vee q \vee s) \wedge (\neg q \vee r) \wedge (\neg p \vee r) \wedge (\neg r \vee s) \wedge (\neg p \vee \neg s \vee \neg q)$

    where p, q, r, and s are propositional variables.

Cook's Theorem (cont.)

- Truth assignment: an assignment of true or false to each variable.
  - A truth assignment is said to satisfy a formula if it makes the value of the entire formula true.
  - e.g., $(r = true, s = true, p = false, q = false)$ is a truth assignment that satisfies the CNF above.
- Decision Problem: Given a CNF formula, is there a truth assignment that satisfy it?

More NP-Complete problems

- Traveling Salesman Problem (TSP)
  - **Optimization problem**: Given a complete, weighted graph, find a minimum-weight Hamiltonian cycle
  - **Decision Problem:** Given a complete, weighted graph and an integer k, is there a Hamiltonian cycle with total weight at most k?
- Clique problem - A clique in graph G is a subset of vertices W such that each pair of vertices in W is connected by an edge in G.
  - **Optimization problem:** Given a graph G, find the largest clique in G.
  - **Decision problem:** Given a graph G and integer k, does G have a clique of size at least k?

Dealing NP-Complete problems

- What to do in case of NP-Complete problems?
  - Use a heuristic
  - Find an approximate algorithm
  - Use exponential time algorithm anyway

Approximation algorithm

- Approximation algorithm:
  - Fast algorithms (i.e., polynomially bounded) that are not guaranteed to give the best solution but will give one that is close to the optimal.
- Measurement of performance
  - For an approximate algorithm A and input I, the performance can be measured by the ratio
    - minimization problem: $rA(I)$ = value return by A for I / opt(I)
    - maximization problem: $rA(I)$ = opt(I) / value return by A for I
  - Note that $rA(I) \geq 1$.

Approximation algorithm (cont.)

Traveling Salesman Problem (TSP)

- TSP asks the following question: Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? [Source wiki]
- Optimization problem: Given a complete, weighted graph, find a minimum-weight Hamiltonian cycle
- Decision Problem: Given a complete, weighted graph and an integer k, is there a Hamiltonian cycle with total weight at most k?

Approximation algorithm (cont.)

Nearest-Neighbor Strategy

select an arbitrary vertex s to start the cycle C

$v = s$

while there are vertices not yet in C

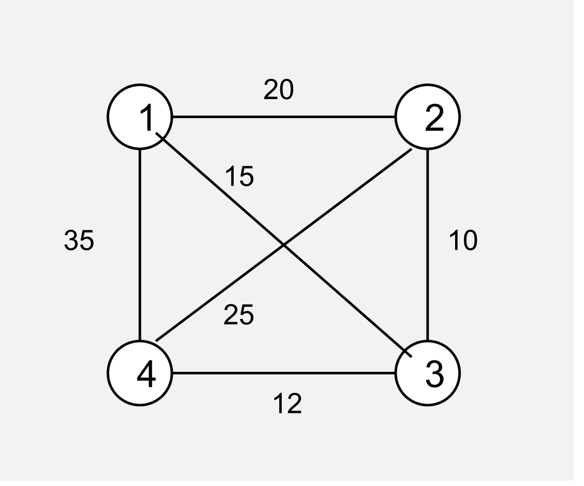
select an edge(v, w) of minimum weight, where w is not in C.

add edge (v, w) to C

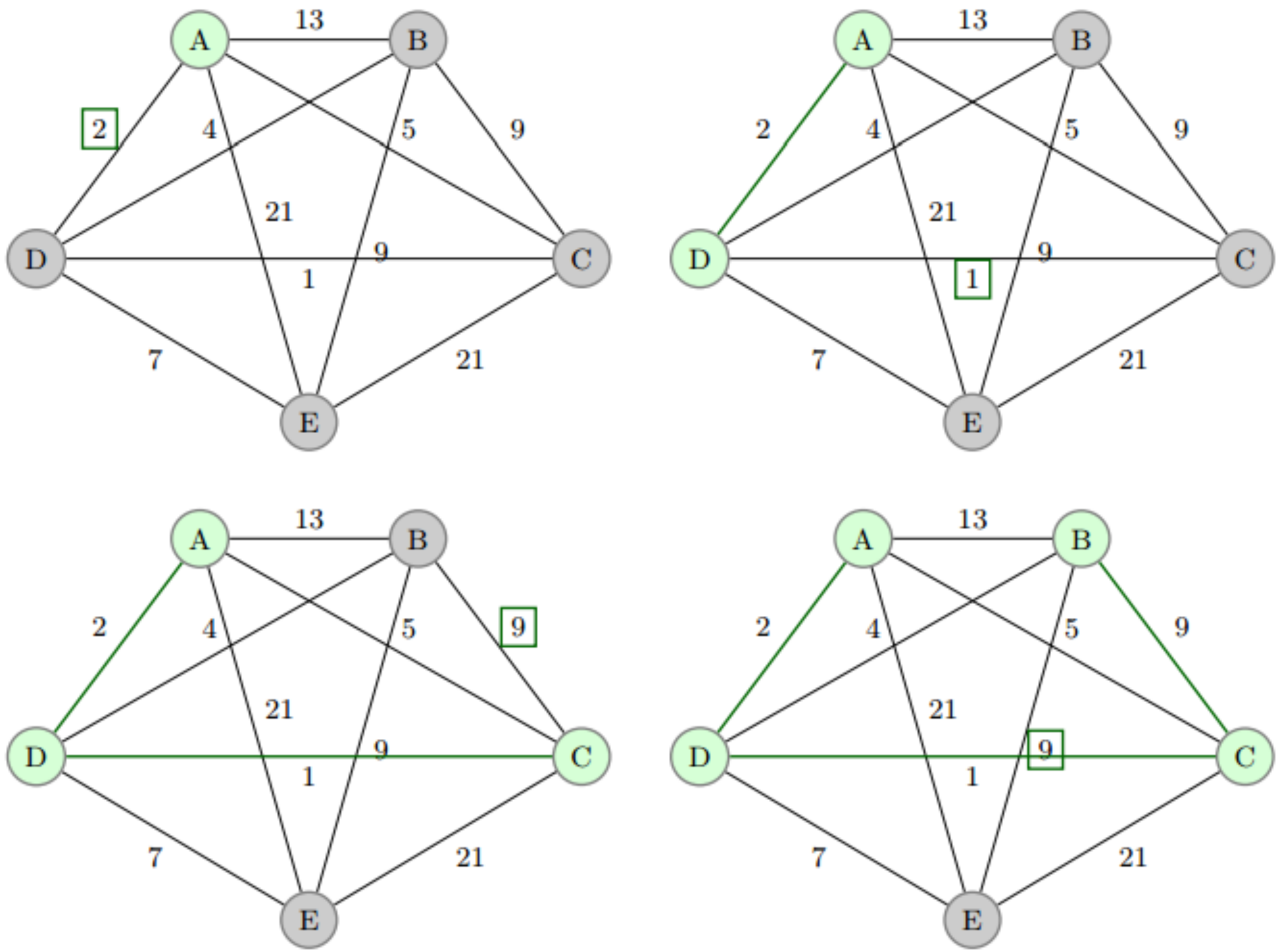
$v = w$

Add the edge (v, s) to C

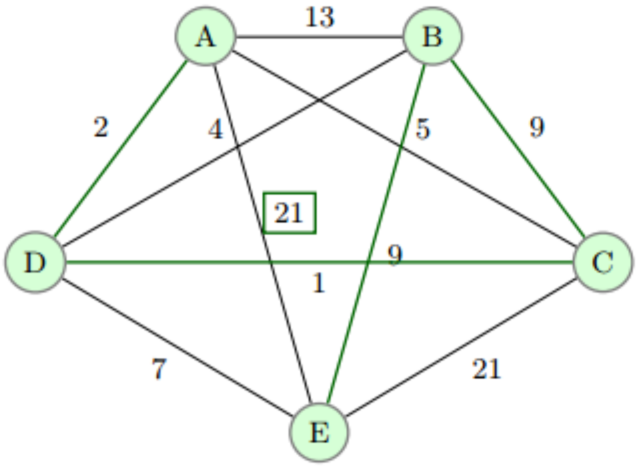
return C.

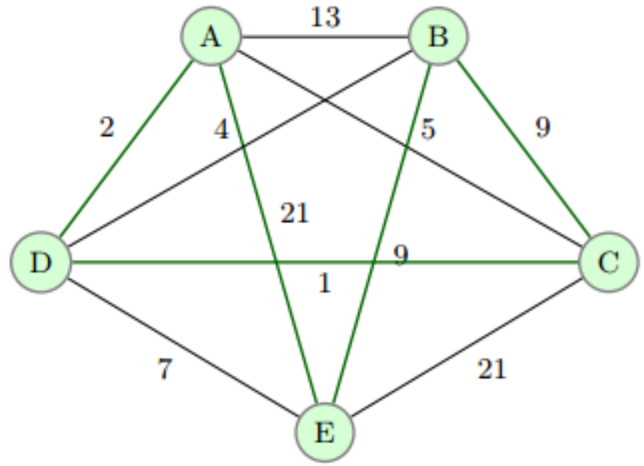


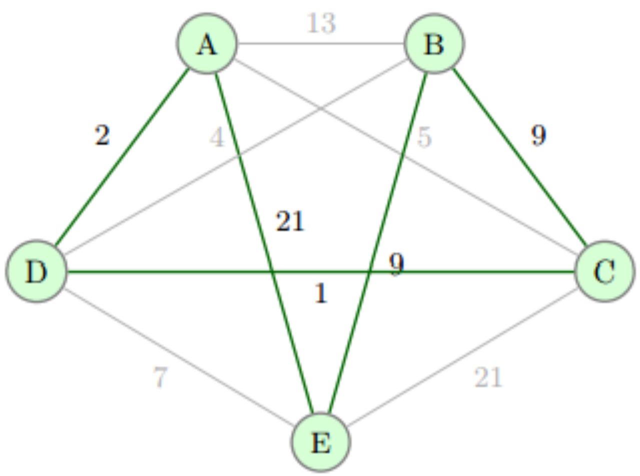
Example: Nearest-Neighbor Strategy



Example: Nearest-Neighbor Strategy







$\boxed{A \to D \to C \to B \to E \to A}$

$2 + 1 + 9 + 9 + 21 = 42.$

Approximation algorithm (cont.)

Shortest-Link Strategy

```
shortestlinkTSP(V,E,W)
   R = E; // R is remaining edges
   C = empty; // C is cycle edges
   while R is not empty
       Remove the lightest edge, vw, from R
       If  (vw does not make a cycle with edges in C
            and vw would not be the third edge in C
            incident on v or w)
         Add vw to C
   Add the edge connecting the endpoints of the path in C
   return C.
```

Performance evaluation

Theorem: Let A be any approximation algorithm for the TSP. If there is any constant c such that $rA(I) \le c$ for all instances I, then P = NP.

Problems and class

- Negative weight cycle detection $\in P$
- $n \times n$ Chess $\in EXP$ and $\notin P$
- Tetrix $\in EXP$ and don't know whether $\in P$

Reference

- Algorithm Design. Jon Kleinberg; Éva Tardos. Addison-Wesley.
- Algorithm Design: Foundations, Analysis, and Internet Examples. Michael T. Goodrich and Roberto Tamassia. John Wiley & Sons.
- Introduction to Algorithms. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein.

Thank you!