Review
Previously

In other words, efficiency depended on listing a combinatorial object.

✎ In each case, an optimal solution can be found after exhausting all solution possibilities that grow combinatorially with input size.

Selection sort performs all possible (distinct) pair-wise key comparisons, in a sense, a brute-force approach to sorting.

✎ Selection sort does not check possible orderings like a combinatorial view of sorting may suggest.

✎ The Assignment Problem is different than *KP* and *TSP* in that, like sorting, it has polynomial-bounded efficiency solutions.

Next
1. *Answer 2 questions of theoretical interest*
2. *Useful tools:*
   - ✎ *Lower bounds*
   - ✎ *Nondeterministic algorithms*
   - ✎ *Decision problems*
   - ✎ *Polynomial reductions*

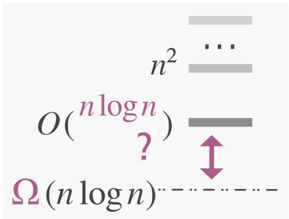
Useful Tools
Lower Bounds

For example, is 2^n the best we can do for the Knapsack Problem? for any algorithm, ever?

Argue about a class of algorithms, some perhaps unknown, via shared characteristics.

➪ Interesting question

What's the best possible efficiency for any algorithm to solve a given problem?



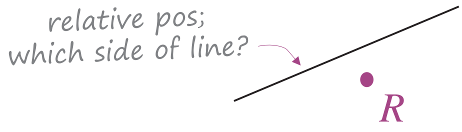
Sometimes trivially obvious, others need to be proven.

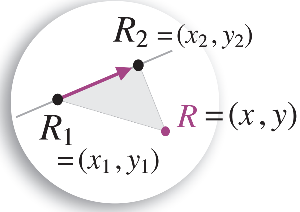
➪ Tight bounds = no room for improvement

- ✎ **Definition** an alg at proven lower bound is known
- ✎ **Examples** list permutations of n distinct items?

Sign of determinant depends on which side of line point *R* lies.

➪ Revisit problem reduction





$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x & y & 1 \end{vmatrix} = x_1 y_2 + x y_1 + x_2 y - x y_2 - x_2 y_1 - x_1 y$$

compute a determinant;
what's the sign?

$P \leq_m Q$

Reduction involves a <u>function</u> that **maps** instances of *P* to instances in *Q* for all instances (i.e., get same result from either).

Id original problem *P* and its question, **reduced problem** *Q* and its <u>equivalent</u> question.

➪ Reduction as solution startegy

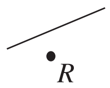
- ✎ Thinking map
- ✎ Reduction pattern? typical places to look for answers?

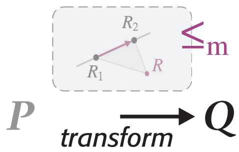
Lower Bounds
Problem Reduction

P Unknown problem
Q Known problem

$P \le_m Q$

➪ **Reduce to *Q* ≡ use *Q* to solve *P***





Sign $\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x & y & 1 \end{vmatrix}$

Reverse direction to argue limits

$Q \le_m P$

➪ **Reduce known *Q* to *P***

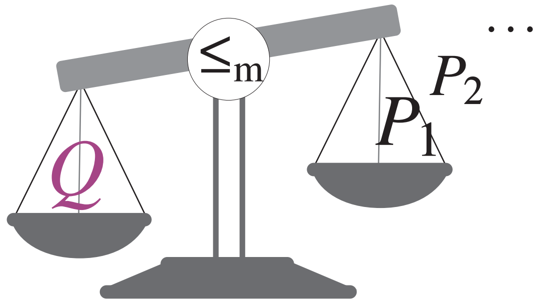
- ✎ *Q* has a credible lower bound
- ✎ Try to use *P* to solve *Q* devise a transformation
- ✎ Examples… *KP* $\le_m$ *TSP* ? (easily, actually), implication?
- ✎ Other problems like *P*? next

If better solution could be obtained through *P*, then a proven tight lower bound for *Q* must be wrong!

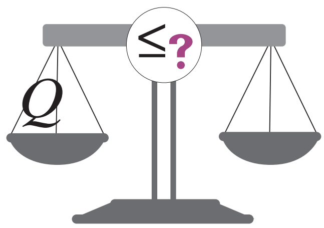
For sorting as *Q*, a reliably tight $\Omega(n \log n)$ applies to *P* as well, i.e., it can't be solved more efficiently.

Useful Tools
Reducibility

Practically, best known efficiency of *Q* applies to all possible ***reduction targets*** (a class of problems?).



An efficient reduction can serve as a basis for an equivalence in complexity. Either both sides are efficient or not.



cs681fig11-14.cdr
Thursday, May 9, 2024 11:29:04 AM
Color profile: Disabled
Composite Default screen

Cost of Computation

➪ [Computational] complexity

An existence question or, in a more typical context, to evaluate known algorithms; either way, must specify the term *efficient* (otherwise, efficiency is relative).

Interesting question

Is there an “efficient” algorithm to solve given problem?

Polynomial-bounded algorithms solve majority of interesting problems in a reasonable time, in most cases for all (at least most) interesting instances.

A useful approach to answer

Arbitrary but reasonable, based on computation cost and usefulness:
run time behavior ≡ cost

cs681fig11-14.cdr
Thursday, May 9, 2024 11:29:04 AM
Color profile: Disabled

Composite Default screen

Algorithm Types

We know the steps leading to correct result.

Quiz
Give 3 examples of problems solved by a **deterministic** algorithm (name it).

Deterministic

Have a valid procedure to get result

Nondeterministic: 2 stages

No steps need be specified to obtain a no-cost *guessed* result.

We must know steps to verify that a result is correct, i.e., deterministically.

Exercise
Describe a **nondeterministic** algorithm for the assignment problem.

① Guessing, suggest a result

② Verification, a valid procedure to check result

From Quiz, any problem with polynomial-bounded solutions can provide an algorithm, a procedure for a self-verified result, for use in step 2.

cs681fig11-14.cdr
Thursday, May 9, 2024 11:29:04 AM
Color profile: Disabled

Composite Default screen

Algorithm Efficiency

➪ **Nondeterminsitic polynomial**

➪ **Tractable [problem]**

$O(p(n))$

Advocated by Jack Edmonds in famous 1965 paper *Paths, Trees, and Flowers*.

Quiz
Can the *Johnson-Trotter* algorithm be considered efficient?

Polynomial runtime ≡ efficient

Worst-case run time grows like a polynomial of input size (n), or better

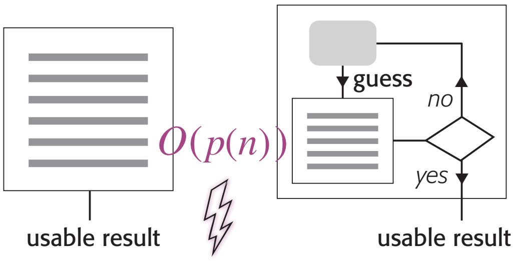
Exercise
Compare solutions of assignment problem via exhaustive search or the Hungarian method. Can we verify an answer efficiently?

If we can only check solution in polynomial time, the algorithm is **non-deterministic polynomial**.

When is an algorithm *efficient*?

- Deterministic, polynomial time ☺
- Non-deterministic, but to check result is deterministic polynomial 🤔

Conclusion



Polynomials place a bound on reasonable growth of runtime or the number of steps (same since each step must finish in finite time), which always yields a useful result.

Practically, a problem may not be solved efficiently if no known deterministic polynomial or no non-deterministic polynomial algorithms exist, i.e., problem intractable.

A Useful Tool
Decision Problems

Some problems naturally arise as decision questions: is a given integer prime? Can a key be found in a list? ...

➪ Definition, examples (simple)

Problems with yes/no (1/0) solution

or accept/reject

➪ Significance

Quiz
How can sorting be formulated as a decision problem? (2 versions.)

- ✎ Interesting problems can always be formulated as decision (decision versions)

Simple to formalize with no loss of generality, decision problem form is convenient as a tool to study power of computing machines.

- ✎ Consistent basis for theory to answer questions about algorithm limitations

Decision Problems
Decidability
➪ Intractability

Exercise
Write a formal statement of the **Halting Problem**.

➪ **Undecidable = no algorithm!**
Generally rare, famous example: the halting problem by Alan Turing

➪ **Decidable problems**

✎ Tractable (≡ easy), or known hard

✎ Unknown (intractable? so far)

Decision Problems
Examples

➪ Conjunctive (maxterm)

Exercise
Give examples of inputs (instances). **Hint**: see description below.

➪ Satisfiability (SAT) i.e., truth

{*Boolean* var}$_n$, {*Bool.* clause}$_r$; is there a set of true/false values such that <u>all</u> clauses are true?

A generic instance consists of *n* variables + conjunctive (joined via AND) *r* clauses, composed of subsets of the vars and their complements combined via OR (disjunctively).

- ✎ CNF (conjunctive normal form) = *Boolean* product-of-sums
- ✎ Sample instances

$\{x_1, x_2, x_3, x_4, x_5\}$,
$\{x_1+x_3+x_4,\ x_2,\ \bar{x}_2+\bar{x}_4+x_5\}$
$E=(x_1+x_3+x_4)\cdot x_2\cdot(\bar{x}_2+\bar{x}_4+x_5)$
Is there an assignment where *E* is true (satisfiable)?
Ans. Yes, e.g., x1 = x2 = x5 = true, rest don't care.

➪ Hamiltonian circuit

Exercise
State the **chromatic number** graph coloring problem as a decision problem.

➪ Knapsack (?) decision version, template

Decision Problems
A Note on Utility

> “Decision problems are too limited. Some computational problems are not easily expressed as decision problems. Indeed, we will introduce several classes in the book to capture tasks such as computing non-Boolean functions, solving search problems, approximating optimization problems, interaction, and more. Yet the framework of decision problems turn out to be surprisingly expressive, and we will often use it in this book.”

Sanjeev Arora and Boaz Barak

ISBN-13 : 978-0521424264

Decidable Problems
A Classification

➪ **Complexity class**

Tractable problems are "easy" since we know how to solve them efficiently (i.e., feasible for <u>all</u> <u>instances</u>).

➪ **Set of (deterministic) <u>polynomial</u> decision problems, P**

$P \subseteq NP$

A valid procedure must always halt with a result (either computed or, if given, verified at least) in reasonable time.

➪ **Set of <u>nondeterministic polynomial</u> decision problems, NP**

?

Arise naturally in practice, some seem simple, even similar to ones known in **P**, e.g., HC vs **Eulerian Circuit** (EC).

➪ **Neither clearly in P nor outside NP, but similarly hard (next)**

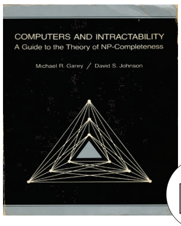
NP-Complete Problems

➪ Polynomial reducibility

No proof one is not possible either (i.e., credible lower bounds).

➪ No polynomial algorithm (?)

Deterministic (but not **P**), polynomially verifiable (so **NP**), easy reduction target



➪ Examples

$P \in NPC \Leftrightarrow$

All problems in **NP** are **polynomially reducible** to *P*.

➪ ❶ $P \in \mathbf{NP}$, ❷ $\forall Q \in \mathbf{NP} \leq_p P$

$Q \leq_m P$

Polynomially reducible
A function to map yes/no instances in polynomial time

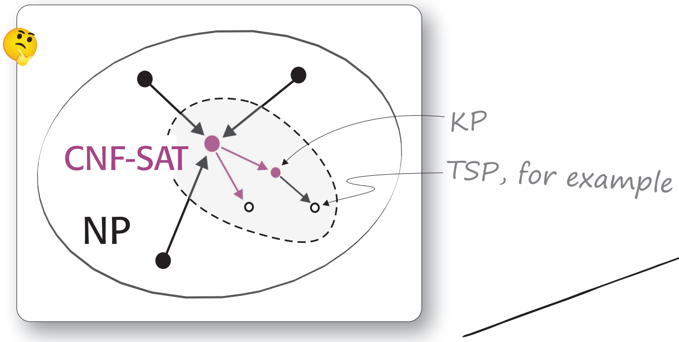
$q_{\text{YES}} \mapsto p_{\text{YES}}$

$q_{\text{NO}} \mapsto p_{\text{NO}}$

➪ CNF-SAT Cook 1971 (Levin 73)

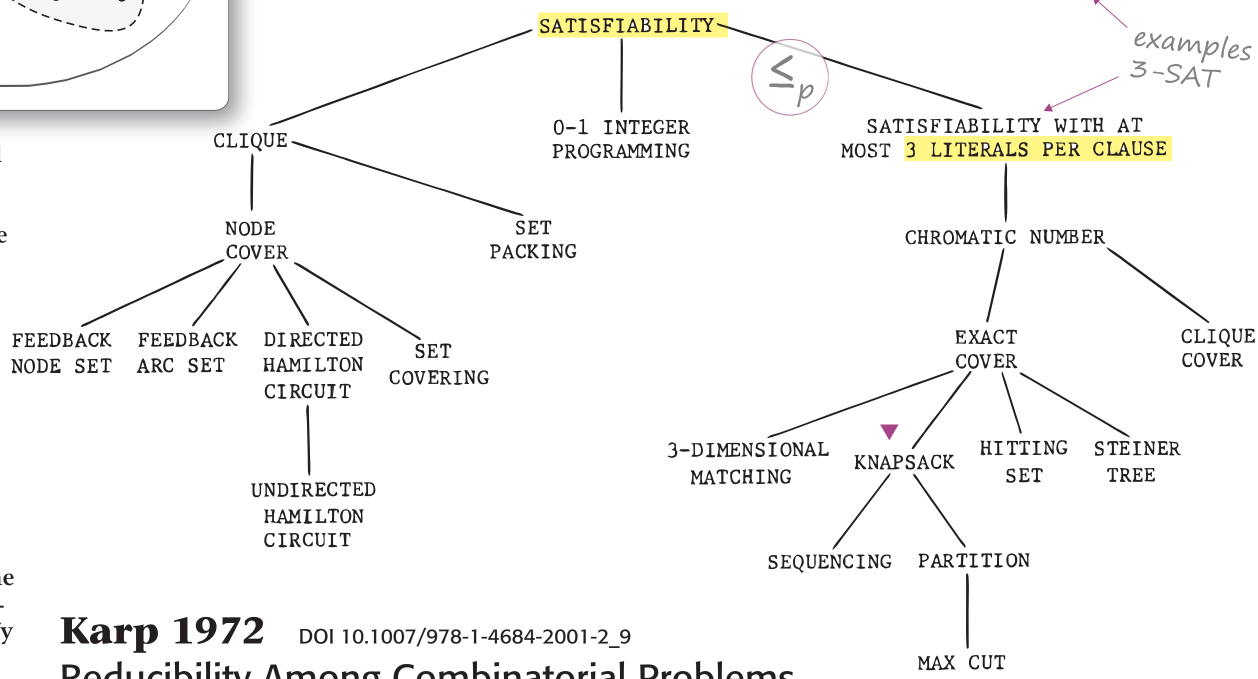
Proven NPC independently

NP-Complete Problems
2nd Generation



$(a \lor b \lor c) \land (\neg a \lor c)$

$(a+b+c) \cdot (\overline{a}+c)$



<u>Transitive</u> polynomial reductions lead back to CNF-SAT (*figure*), whatever proved there applies.

Exercise
In which class does the 2-SAT fall? Cite appropriate reference, justify answer.

Karp 1972 DOI 10.1007/978-1-4684-2001-2_9
Reducibility Among Combinatorial Problems.

NP-Complete Problems
Practical Implications

Karp 1972, ***Reducibility Among Combinatorial Problems***.

- **2-Step proof (Karp 21)**

Similarly hard to any in NP including each other.

- **Solving one is enough**

Better manage time and effort.

- **Important to recognize**

Solve, in theory, means for all instances (*quicksort* solves sorting efficiently for all finite lists).

- **Interesting instances?**

Branch-and-bound is one such method (later).

There are ways to deal with complexity when we focus on some instances

Summary

Term often used informally to describe problems whose decision version is NPC.

➪ NP-hard [problem]

Exercise
Give at least 3 examples for each case.

➪ Polynomial time solvable

Deterministic algorithm solves problem in polynomial runtime

Quiz
Can a **Hamiltonian circuit** problem be polynomially reduced to a **Eulerian circuit** question?

➪ Polynomial time verifiable

Verify a solution in polynomial time

Polynom reducibility to any NPC is useful!

➪ Polynomial time reducible

Reduce to problem in polynomial time

Conclusions

➪ Polynomial efficiency

Currently being challenged by quantum computers.

$KP \in \mathbf{P} \Leftrightarrow \mathbf{P} = \mathbf{NP}$

$\mathbf{P} \subset \mathbf{NP}$ iff $\mathbf{P} \subseteq \mathbf{NP}$ and $\mathbf{P} \neq \mathbf{NP}$

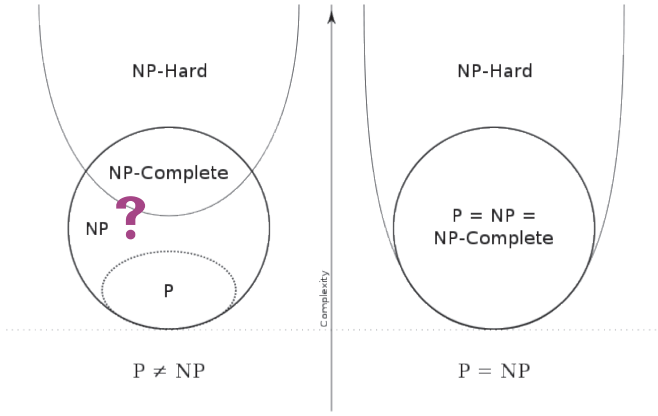
Open question: $\mathbf{P} \overset{?}{=} \mathbf{NP}$

Is **P** a proper subset of **NP**?

The **Halting Problem** is the most notable example of a problem outside **NP**.

Exercise
What's the efficiency of a sorting algorithm in which permutations of input items are checked until the right (sorted) one is found?

Oddly, this is all we seem to be able to do for some interesting problems!



By Behnam Esfahbod, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=3532181

cs681fig11-14.cdr
Thursday, May 9, 2024 11:29:08 AM
Color profile: Disabled

Composite Default screen

Exercise

Check *KP* $\leq_p$ *TSP*

Step 1: Outline a procedure.

- ✎ Show *TSP* in **NP** (polynomially verifiable)

Step 2: use the pseudocode to show reduction to be polynomial.

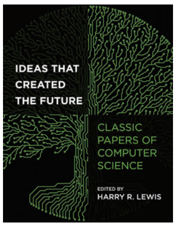
- ✎ Is *KP* reducible to *TSP*? write a pseudocode
- ✎ Show *TSP* NPC based on the *KP*

Reducibility Among Combinatorial Problems.
DOI 10.1007/978-1-4684-2001-2_9

Read Karp 1972

Specify *TSP* as decision problem

References



ISBN-13: 978-0262045308

Reprinted #36 (Cook'71 #34)

- ✏ Richard M. Karp, Reducibility Among Combinatorial Problems, 1972
- ✏ JACK EDMONDS, PATHS, TREES, AND FLOWERS, 1965
- ✏ Garey and Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences), 1979
- ✏ Sanjeev Arora and Boaz Barak, Computational Complexity: A Modern Approach 1st Edition, 2009 ISBN-13: 978-0521424264
- ✏ Erik D. Demaine, William Gasarch, and Mohammad Hajiaghayi, Computational Intractability: A Guide to Algorithmic Lower Bounds, DRAFT October 15, 2023
- ✏ Anany Levitin, Introduction to the Design and Analysis of Algorithms 3rd Edition, 2011 ISBN-13 : 978-0132316811