Relations between Problems

- some problems can be solved using an algorithm for another problem
- reduction
  - e.g. problem B can be efficiently solved if there is an efficient algorithm for problem A
  - B cannot be fundamentally harder than A $\iff$ A cannot be fundamentally easier than B
  - $B \leq_x A$
    - B can be reduced to A
    - solving B with the help of A

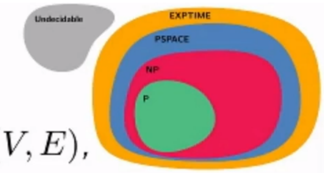
Reducing City Tour to Dinner Party

- similarity
  - view both problems as graphs
  - "differently reachable from …" $\iff$ "liked by …"
- difference
  - cycle must be closed for the dinner party
- conclusions
  - if there is a tour $\Rightarrow$ there is a possible seating order
  - if there is a seating $\Rightarrow$ finding a tour path is easy
    - no tour $\Rightarrow$ no seating
  - $\Rightarrow$ City Tour $\leq_x$ Dinner Party:
    The seating problem is at least as hard as the tour problem.
- the dinner party problem can be also reduced to the city tour problem

Complexity Classes

- P
  - decision problems solvable in polynomial time
- NP
  - decision problems solvable in nondeterministic polynomial time
  - solution is verifiable in polynomial time given a certificate (e.g. a solution)
- PSPACE
  - decision problems solvable using a polynomial amount of memory
  - disregards time complexity
- EXPTIME
  - decision problems solvable in exponential time
- Undecidable
  - problems which cannot be solved no matter how much time or space is allowed
  - must be proven that no such algorithm exists

- ## Some Example Problems

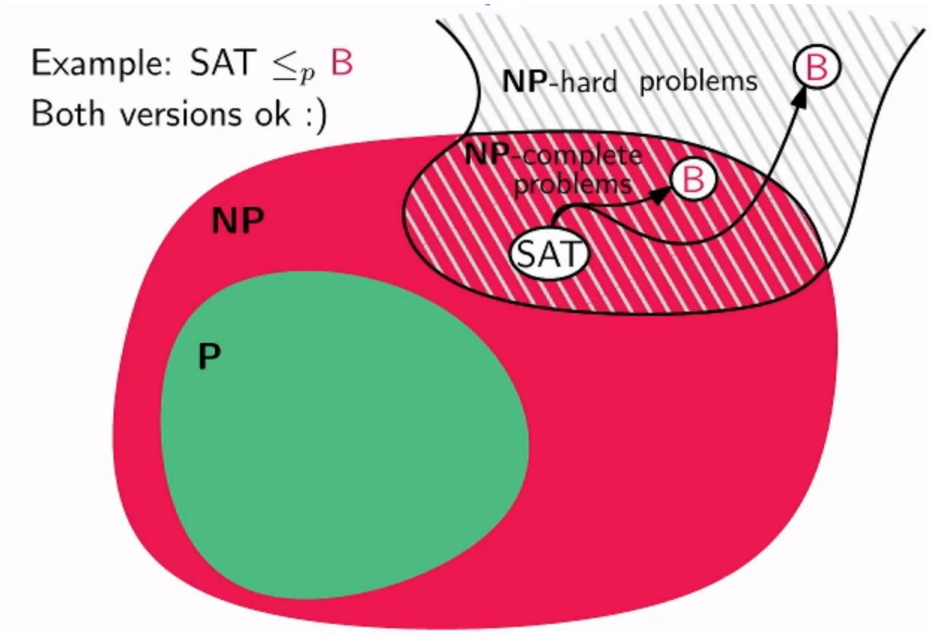
  - Paths in graphs: *Given a graph $G = (V, E)$, is there a . . .*
    - *path from vertex u to vertex v with at most k edges?*
    - *simple path from u to v with at least k edges?*
    - *simple path through all vertices (with $n-1$ edges)?*
  - Integer factorization: *Given two integers n and k with $1 < k < n$, does n have a factor d with $1 < d \leq k$?*
  - Halting problem: *Given a program P and an input I,*
    - *does P halt on I after finitely many steps?*
    - *does P halt on I after exponentially many steps?*
  - Checkers/Hex: *Given an $n \times n$ board and a game situation, is there a winning strategy for the first player?*

Types of Polynomial Time Reductions

- **Polynomial time Turing (Cook) reduction A $\leq_{PT}$ B:**
  - At most polynomially many calls to the subroutine for B.
  - Everything except the subroutine calls for B needs polynomial time in total.
- **Polynomial time Karp reduction A $\leq_p$ B:**
  Transform inputs for A into inputs for B in polynomial time, in a way that the output from B on the transformed input is the same as the output from A for the original input.
  - special case of Cook reductions
- **Note:** A $\leq_{PT}$ B or A $\leq_p$ B does not imply that an algorithm for A runs faster than one for B. But it implies that
  - if B is in **P**, then A is in **P** as well.
  - if A is not in **P** then B can't be in **P** either.

NP-completeness

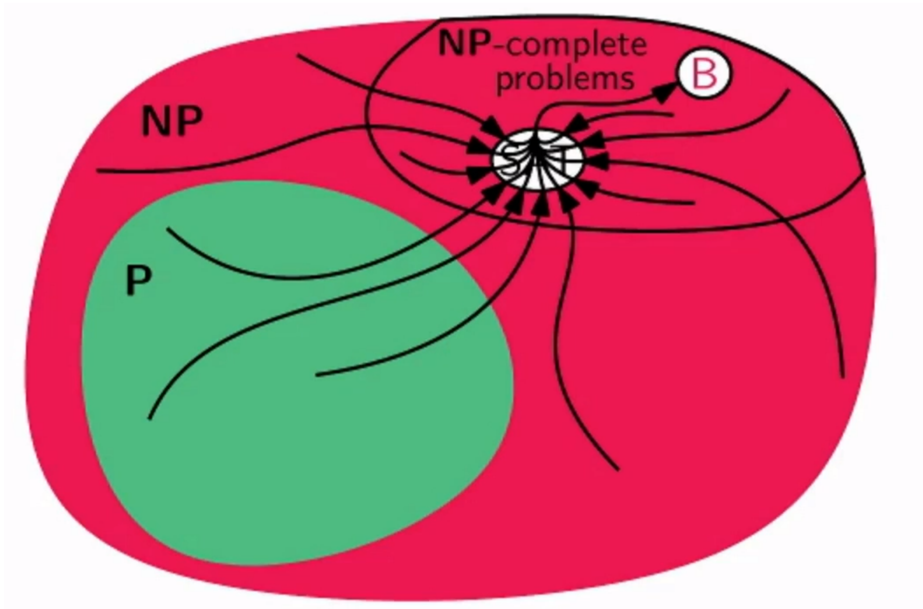
- Problem B is NP-complete if
  - B NP
  - B is NP-hard
    - B is "at least as hard" as all other problems in **NP**, or, more formally:
      A $\leq_p$ B for all problems A in NP.

- 

- ways to show that B is NP-complete
  - Possibility 1:
    Show A $\leq_p$ B for **all problems** A **in NP**.

    Cook's Theorem:
    SAT (satisfiability of boolean formulas) is **NP-complete**.

    Possibility 2:
    Show C $\leq_p$ B for **some NP-complete problem** C:
    * possibility 2 works because
      - As A $\leq_p$ C for all problems A **in NP**,
        and as A $\leq_p$ C and C $\leq_p$ B implies A $\leq_p$ B,
        - all problems in NP can be reduced to C
          - since C is NP-complete
        - C can be reduced to B

- 

  - must also show that B NP
    - otherwise B might be NP-hard but B NP
- ways to show that B is NP-hard
  - reduce a NP-hard problem to B