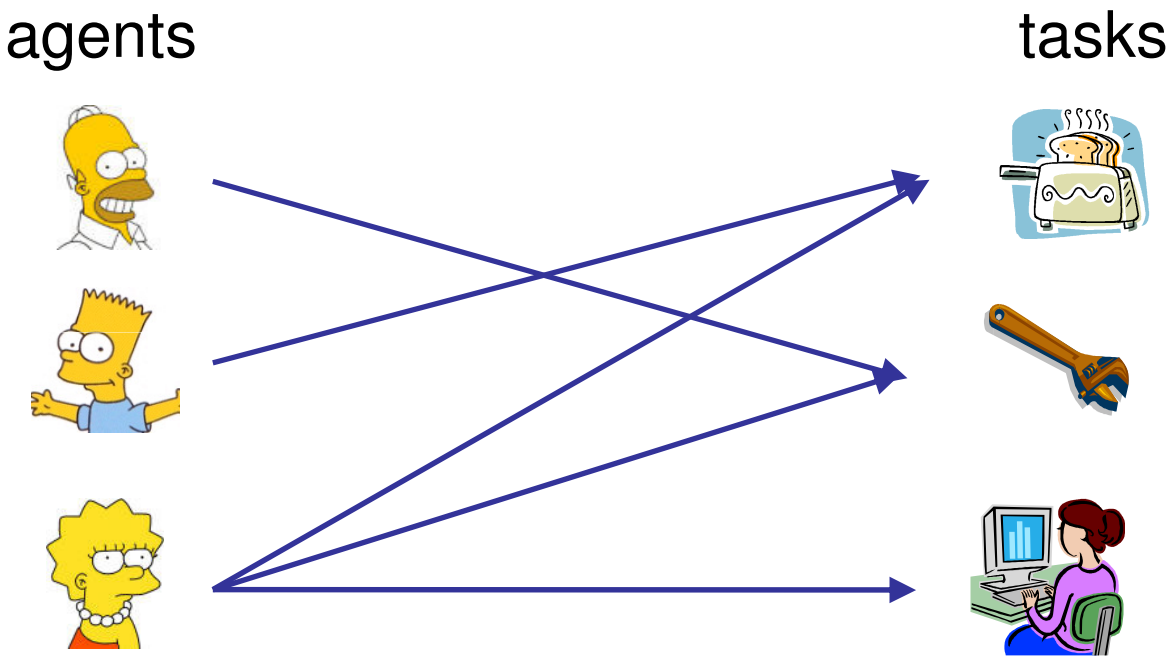
Computational Complexity in Multiagent Systems

Edith Elkind
University of Southampton, UK

Evangelos Markakis
CWI Amsterdam, the Netherlands

Example: perfect matching

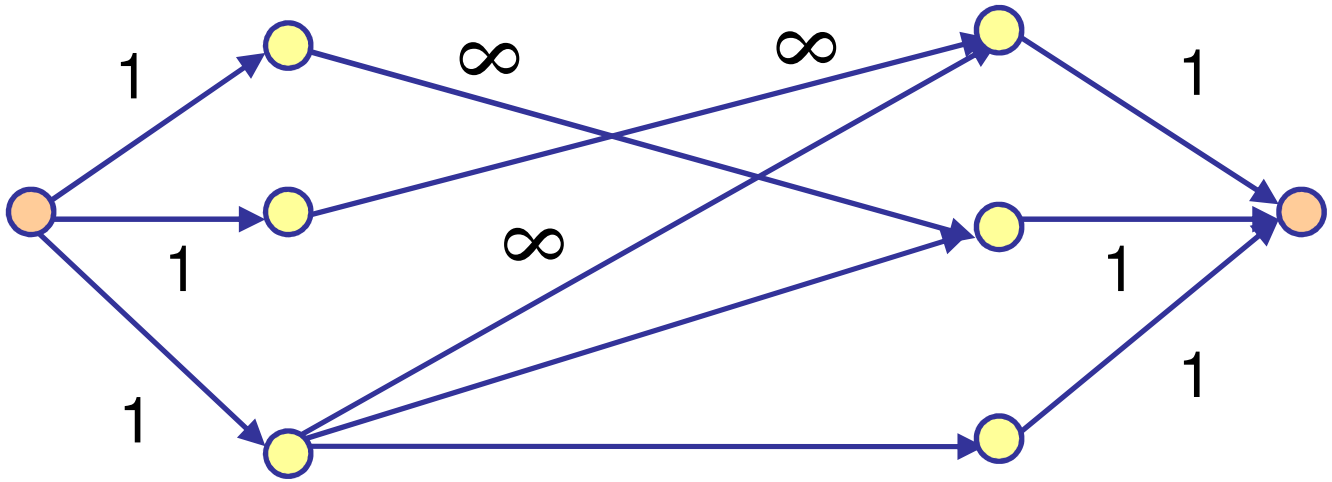


- Need to match agents to tasks one-to-one
- Each agent can perform some of the tasks

Straightforward solution

- Consider all orderings of the tasks
- Try to match i^{th} agent to i^{th} task
- If fails (i^{th} agent can't perform i^{th} task), move on to the next ordering
- Running time: $n!$

Clever solution

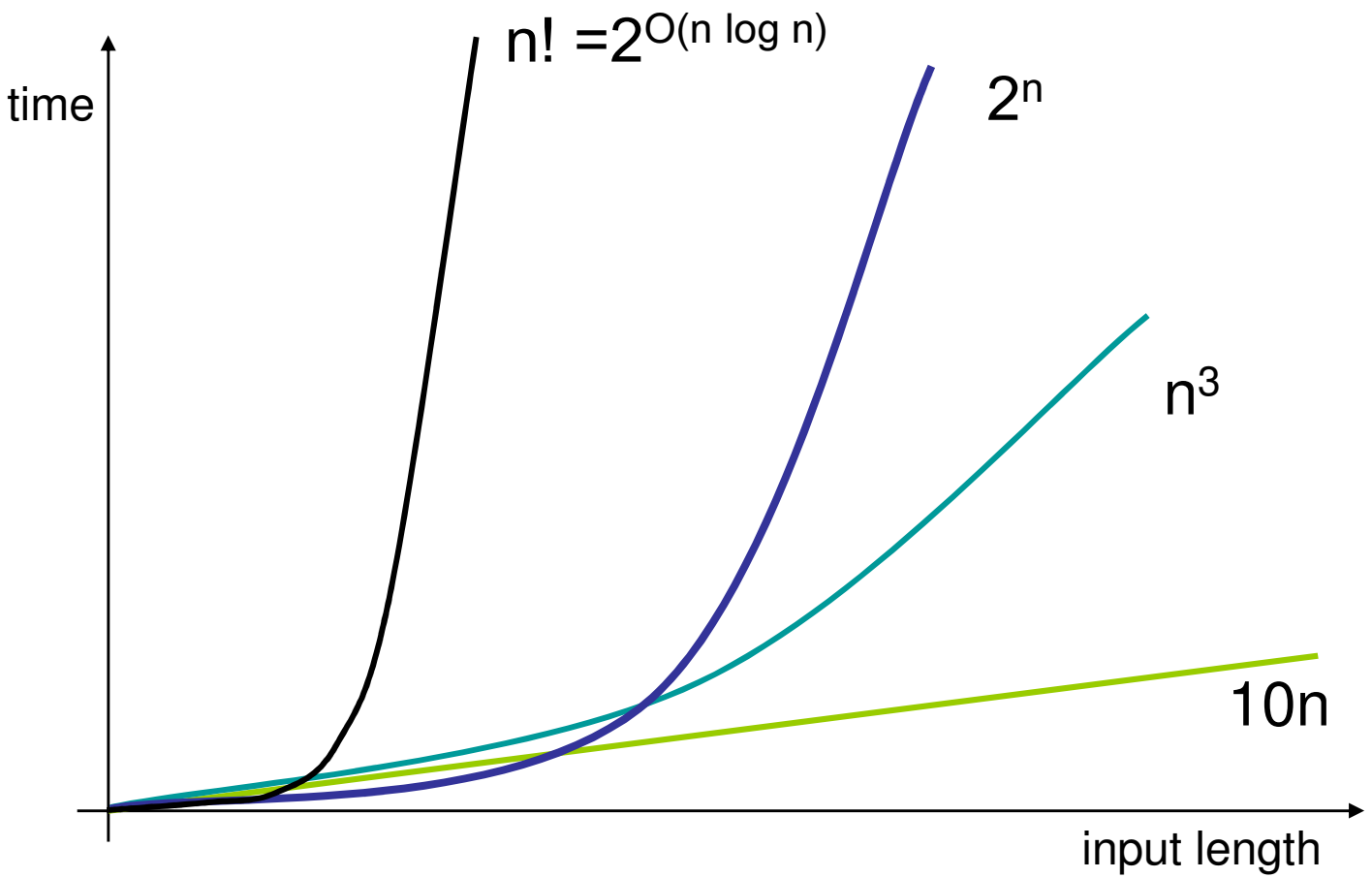


- Reduce to network flow problem
- Network flow can be found in time $O(n^3)$

Which algorithm is better?

- For small n, can be easier to try all permutations:
  - $3!=6$, $3^3=27$
  - $4!=24$, $4^3=64$
  - $5!=120$, $5^3=125$
- For large n, $O(n^3)$ is clearly better:
  - $20!=2.43290201 \times 10^{18}$, $20^3=8000$

Growth rate



Complexity Theory

- Goals:
  - classify problems into those that can be solved efficiently and those that (probably) cannot
  - provide tools for proving that some problems are (likely to be) hard
- Why useful to know that a problem is hard?
  - can stop trying solve it for the general case
  - focus on heuristics and special cases

Outline

- 1st part
  - classes **P** and **NP**
  - classic **NP**-complete problems
  - examples of **NP**-complete problems in multiagent systems
- 2nd part
  - pseudopolynomial algorithms
  - classes **PSPACE** and **#P**
  - a glimpse beyond:
    - polynomial hierarchy, approximation algorithms, randomized algorithms, ...

Model of computation

- How do we formalize the notion of running time?
- What is a step of the computation?
- Classical approach: Turing machines
- This tutorial: common sense
  essentially, number of operations of a straight-line program (without goto's)
  - assume that numerical values are given in binary, and all operations on them are bit-by-bit

Polynomial time: class **P**

- Decision problem:
  a problem where answer is “yes” or “no”
  - e.g., is there a matching of size n?
- Definition: a decision problem T is polynomial-time solvable if there is

  a polynomial $f(n)$ and an algorithm A such that given an instance I of T, $|I|=n$, A runs in time $f(n)$ and outputs the correct answer (“yes” or “no”)
- **P**: class of all poly-time solvable decision problems

Class **P** and efficient computation

- polynomial-time solvable = efficiently computable?
- $100n^{30}$ vs n! - which one is better?
- Some answers:
  - usually after the first poly-time algorithm is found, practical algorithms appear soon
  - convenient to work with
    - (some) independence of input representation
    - closed under composition

Problems in **P**: arithmetics

- Arithmetics:
  - adding two n-bit numbers: $O(n)$
  - subtracting two n-bit numbers: $O(n)$
  - multiplication: $O(n^2)$
  - division: $O(n^2)$
- Yes-no version:
  is the i^{th} bit of output equal to j?

Problems in **P**: search

- Input: array $A[1..n]$, number x
- Question: does x appear in A?
- Time: $O(n \log \max A[i])$
  - comparisons take non-unit time!
- Suppose A is sorted?
  - Time: $O(\log n \log \max A[i])$

Problems in **P**: shortest path

- Input: Graph $G=(V, E)$, source s, sink t, target k, edge lengths $\ell_1, \ldots, \ell_{|E|}$
- Question: is there a path from s to t of length at most k?
- Time: $O(|E|^2 \log \max \ell_i)$
  - Dijkstra's algorithm

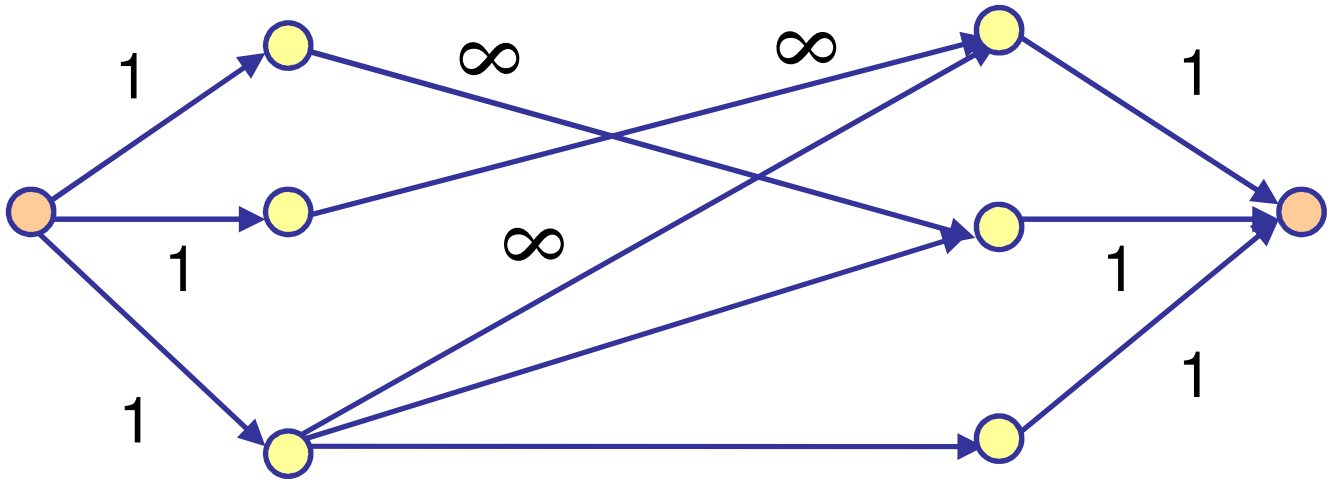
Problems in **P**: network flow

- Input: graph $G=(V, E)$, source s, sink t, target k, edge capacities $c_1, \ldots, c_{|E|}$
- Question: is there a flow of size at least k from s to t?
- Time: $O(|V||E|^2 \log \max c_i)$
  - Edmonds-Karp algorithm

Problems in **P**: bipartite matching

- Input: bipartite graph $G=(L, R, E)$, $|L|=|R|=n$
- Question:
  is there a matching between L and R in G?
- Time: $O(n|E|^2)$
  - proof: transform an instance of
    Bipartite Matching
    into an instance of Network Flow

Bipartite matching → network flow



- Network flow can be found in time $O(n^3)$

Reductions

- Theorem:
  If problem A is polynomial-time reducible to problem B and B is poly-time solvable, then A is poly-time solvable as well.
  - suppose A can be reduced to B in time f(n)
  - B can be solved in time g(n)
  - if f and g are polynomials, then g(f(n)) is polynomial, too

Problems not known to be in **P**: longest path

- Input: graph $G=(V, E)$, source s, sink t, target k, edge lengths $\ell_1, \ldots, \ell_{|E|}$
- Question: is there a loop-free path from s to t of length at least k?

Problems not known to be in **P**: traveling salesman

- Input: n cities, $n(n-1)/2$ distances between cities, target k
- Question: is there a route that visits each city exactly once and has length at most k?

Problems not known to be in **P**: vertex cover

- Input: graph $G=(V, E)$, target k
- Question: is there a vertex cover of size at most k?
  - i.e., a set of vertices such that every edge in E has at least one endpoint in this set

What do LP, TSP and VC have in common?

- For all 3 problems, if an instance is a "yes"-instance, this is easy to prove:
  - LP: exhibit a path of length at least k
  - TSP: exhibit a tour of length at most k
  - VC: exhibit a vertex cover of size at most k
- Not all problems are like that:
  - e.g., find a minimal circuit equivalent to a given one
    - how do you prove there is no smaller circuit?

Class **NP**: informal definition

- **NP**: nondeterministic polynomial time
- Idea: once you guess a solution, you can check in polynomial time if it is correct
- Longest Path, Traveling Salesman, Vertex Cover are all in this class

Class **NP**: formal definition

A decision problem A is in **NP** if
there is a polynomial f
and a decision problem B in **P** such that
an instance I of A is a “yes”-instance

if and only if

there exists a string W(I) of length at most f(|I|)
such that (I, W(I)) is a “yes”-instance of B

– W(I): the witness for I

Example: vertex cover

- I(A): graph G=(V, E), target k
- I(B): pairs of the form (G, k, V'), where V' is a vertex cover for G of size at most k
- B is in **P**: it is easy to check if a triple (G, k, V') is a "yes"-instance
- size of V' is polynomial in size of (G, k)
- Hence, Vertex Cover is in **NP**

P vs. **NP**

- Every problem in **P** is also in **NP**
  - is the converse also true?
- I.e., is finding a solution more difficult than verifying one?
- This is the famous **P** = **NP**? problem
  - $1 000 000 problem (www.claymath.org)
- E.g., we do not know how to prove that VC has no poly-time algorithms!

Reductions: reminder

- Theorem:
  If problem A is polynomial-time reducible to problem B and B is poly-time solvable, then A is poly-time solvable as well.

Hardness proofs

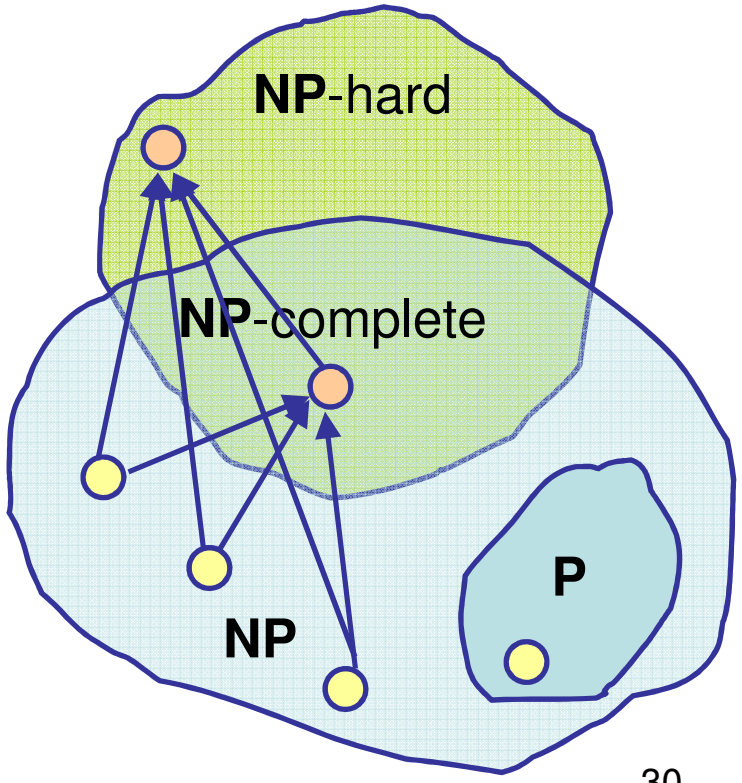
- How can we prove that a problem X is hard?
- By finding a poly-time reduction f from another hard problem Y to X!
- Suppose some A solves $x \in X$ in poly time.
- Can solve $y \in Y$ by computing A(f(y)) - contradiction!
- Direction: from a problem known to be hard to the problem that we want to prove hard

Hard problems: where to start?

- If we have one hard problem,
  we can show that other problems are hard
  by reducing this problem to them
- However, no problem in **NP** is known
  to be hard
- A different approach: find a problem X in **NP**
  s.t. all other problems in **NP** reduce to it
  - X is as hard as anything in **NP**!

NP-hardness and **NP**-completeness

- A problem is **NP**-hard if all problems in **NP** reduce to it
- **NP**-complete: **NP**-hard and in **NP**



NP-complete problems: 3-SAT

- Input:
  - n boolean variables $x_1, \ldots, x_n$
  - m clauses $c_1, \ldots, c_m$
  - each clause is a conjunction of 3 literals: $z_1 \vee z_2 \vee z_3$
  - each literal: variable or its negation
- Question: is there a list of values for $x_1, \ldots, x_n$ such that all clauses are satisfied?
- Example: n=3, m=2: $x_1 \vee \neg x_2 \vee x_3$, $\neg x_1 \vee x_2 \vee \neg x_3$
  - satisfied by $x_1=1$, $x_2=0$, $x_3=0$

3-SAT: **NP**-completeness proof sketch

1. 3-SAT is in **NP**

   given an assignment, can check if all clauses are satisfied in linear time

2. 3-SAT is **NP**-hard

   proof idea: for any problem in **NP**, computing the solution can be simulated by a non-deterministic circuit (guess a solution and verify it)

   circuit can be encoded by a huge (but still poly-sized) boolean formula

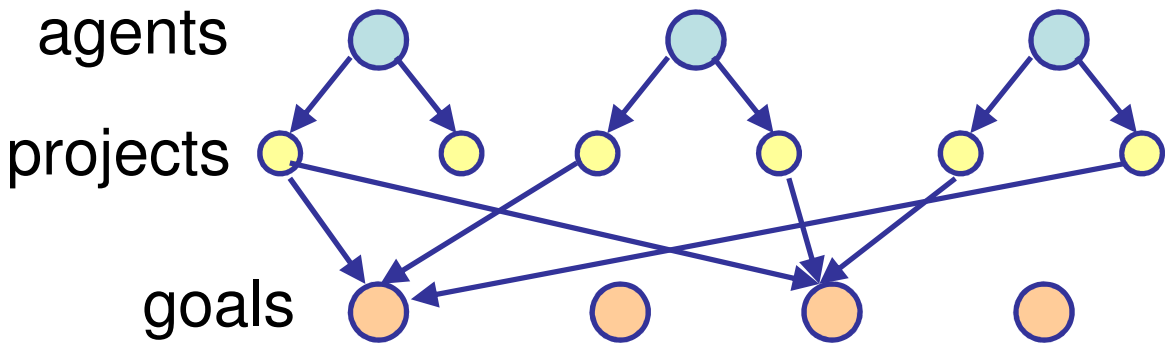
   satisfying assignment corresponds to guesses of the circuit

3-SAT and multiagent systems

- multi-agent task allocation:
  - N agents, M projects, K goals
  - for each agent i, there is a set M_i of projects he can complete; he has time to finish exactly one of them
  - for each goal k, there is a set M^k of projects that fulfill it; any of the projects in M^k will fulfill k
- Question: can we fulfill all goals?
- **NP**-hard: reduction from 3-SAT

NP-hardness Proof (1/2)

- Recall: need to reduce 3-SAT to our problem (not vice versa)
- For an instance X of 3-SAT, construct an instance of MTA:
  - one agent for each variable, one goal for each clause
  - 2 projects p_i and q_i for agent i
  - if clause j contains x_i, then goal j can be fulfilled by p_i
  - if clause j contains $\neg x_i$, then goal j can be fulfilled by q_i



$c_1 = x_1 \vee x_2 \vee x_3$

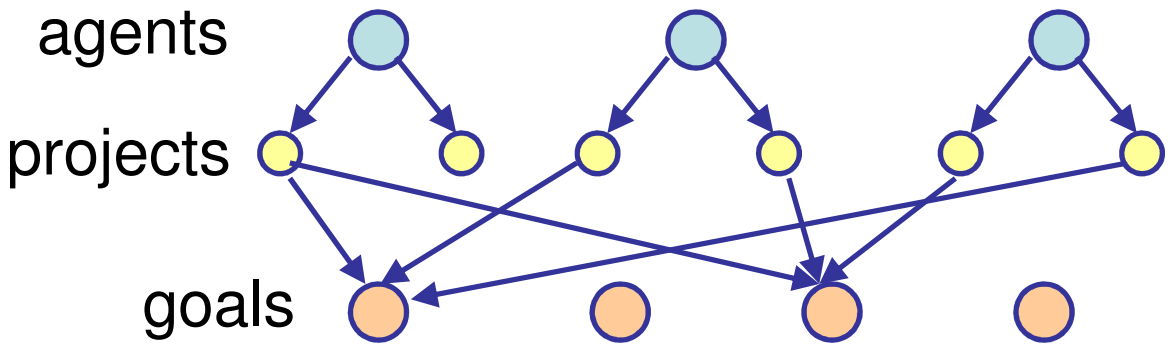
$c_3 = x_1 \vee \neg x_2 \vee \neg x_3$

NP-hardness proof (2/2)

Need to prove: given instance of 3-SAT is satisfiable iff we can fulfill all goals:

if there is a satisfying assignment $(b_1, \ldots, b_n)$,
assign i to p_i if $b_i=1$ and to q_i if $b_i=0$

if there is an assignment of agents to projects that fulfills all goals,
set $b_i=1$ if i is assigned to p_i and $b_i=0$ if i is assigned to q_i



NP-hardness proof: remarks

- Our instance of MTA has a very special structure:
  - no two agents share a project
  - each agent has a choice of exactly 2 projects
  - exactly 3 projects per goal
- This special case is **NP**-hard, so the general case is hard, too

Useful **NP**-complete problems: graph theory

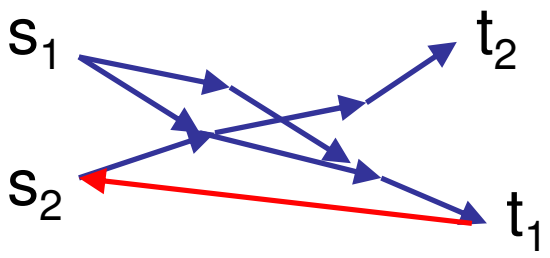
- Longest Path
- Traleling Salesman
- Vertex Cover
- 2 Disjoint Paths:
  - Input: graph $G=(V, E)$, two source-sink pairs (s_1, t_1), (s_2, t_2)
  - Question: is there a vertex-disjoint pair of paths from s_1 to t_1 and from s_2 to t_2?

Essential edge

- Input: a graph G=(V, E), source s, sink t, an edge e
- Question: is e essential, i.e., is there a loop-free path from s to t that goes through e?
- Applications:
  - suppose edges are owned by agents
  - most classical payment schemes pay 0 to non-essential agents
  - can we determine if a given agent shoud get 0?

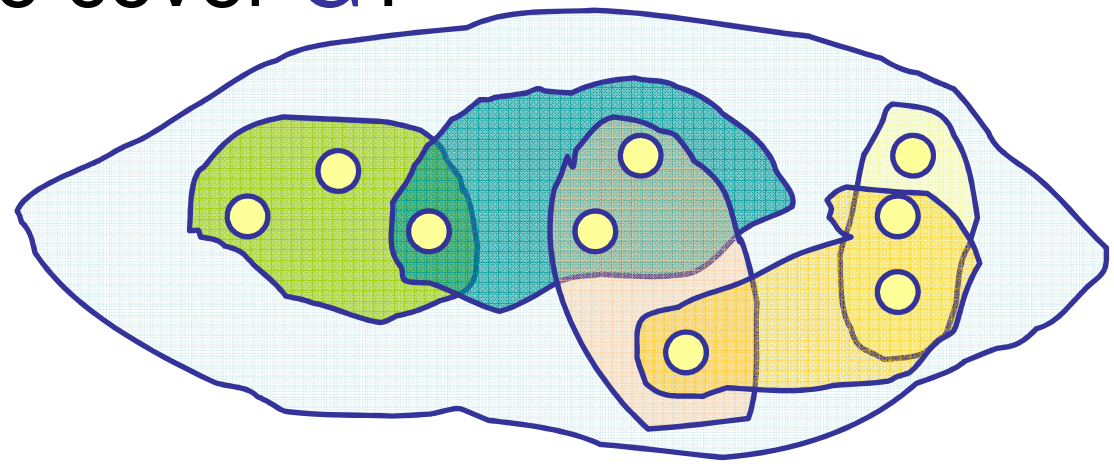
Essential edge is NP-complete

- In **NP**: exhibit a loop-free path through e
- **NP**-hard: reduction from 2 Disjoint Paths
  - need to map instance of 2DP to instance of EE
  - instance of DP: G, s_1, t_1, s_2, t_2
  - instance of EE: G', s, t, e
  - reduction: set G' = G, s = s_1, t = t_2, e = (t_1, s_2)



Useful **NP**-complete problems: Exact cover by 3-sets (X3C)

- Input: a ground set G, $|G|=3n$, a collection F of subsets of G: $F = \{S_1, \ldots, S_m\}$, $|S_i|=3$ for all i
- Question: can we choose n sets from F to cover G?



Combinatorial auctions with single-minded bidders

- n bidders, m items, target K
- each bidder has a value for some subset of items
- Question: can we allocate the items so that the social welfare is at least K?
- **NP**-complete: reduction from X3C
  - in **NP**: guess an allocation, verifying is easy
  - **NP**-hard: next slide

CA with S-M bidders are **NP**-hard

- Need to map X3C to CA/S-M
  - instance of X3C: set G, $|G|=3n$, subset $S_1, \ldots, S_m$
  - instance of CA/S-M: pair (bundle, value) for each bidder, target welfare K
- Mapping: bundle $B_i=S_i$, unit values, $K=n$
  - if there is a cover, there is an allocation of value K
  - if there is an allocation of value K, n bidders get their bundles, so there is a cover

Voting

- n voters, m candidates
- each voter has a preference list
  - total ordering of candidates
- Voting protocol: collects votes (total orderings), decides a winner
  - e.g., plurality: candidate with the largest number of first-place votes wins

Other voting protocols

- Borda: each candidate gets
  - m-1 point from each voter that ranks him 1st
  - m-2 points from each voter that ranks him 2nd …
- STV:
  - if there is a plurality winner, return him
  - else, select a candidate with the lowest number of 1st-place votes, and delete him from all ballots
  - repeat till there is a plurality winner

Voting manipulation

- Suppose
  10 voters have preferences A>B>C,
  10 voters have preferences C>A>B,
  your preferences are B>A>C
- Voting rule is plurality
- You are better off voting A>B>C!
- There are such scenarios for every voting protocol (Gibbard-Satterthwaite theorem)

Voting manipulation: computational aspects

- Problem: given everyone else's votes and your own preferences, is there a non-truthful vote that makes you better off?
  - if you vote truthfully, A is elected
  - if you lie, B is elected, and you prefer B to A
- In P for plurality and Borda, but **NP**-hard for STV!
  - complexity is a good thing: barrier against manipulation

NP-hardness: end of story?

- **NP**-hardness is a worst case analysis
  - shows that there exist instances of the problem that are hard to solve
- Can be used to identify limitations of solving an *arbitrary* instance of a problem
  - does your problem have additional structure?
  - if so, maybe it is poly-time solvable?
  - poly-time solvable (interesting) special cases?

Summary of 1st part

So far we have seen:

- definitions of the classes **P** and **NP**
- some useful **NP**-complete problems
- examples of **NP**-complete problems in multiagent systems

Computational Complexity in Multiagent Systems

Part 2 (Vangelis Markakis)

Summary of 1st part

So far we have seen:

- definitions of the classes **P** and **NP**
- some useful **NP**-complete problems
- examples of **NP**-complete problems in multiagent systems

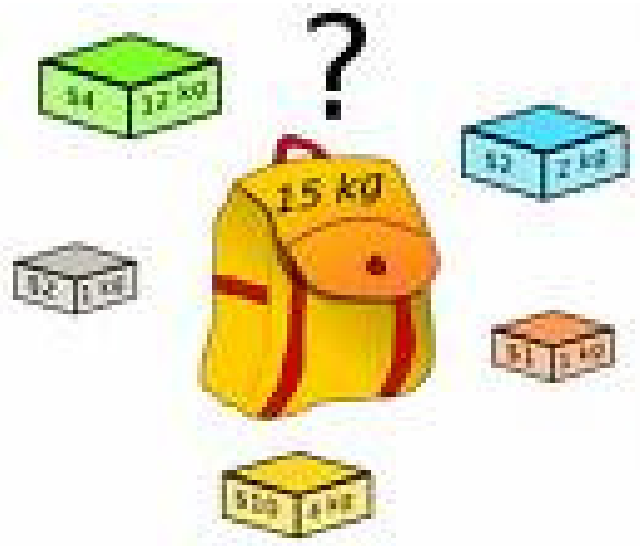
Can we hope to solve special cases of **NP**-hard problems in polynomial time?

The Knapsack problem

Input: A knapsack of capacity B, a set of goods $\{1,\ldots,n\}$, each with size s_i and profit p_i, and an integer k

Goal: Can we fill the knapsack with goods of total weight at most B and total profit at least k?

Knapsack is an **NP**-hard problem



Pseudopolynomial time algorithms

Let $p = max_i\ p_i$

Theorem: There exists an algorithm for Knapsack that runs in time $O(n^2\ p\ log\ B)$

- This is NOT a polynomial time algorithm
- We need at most $log\ p$ bits to represent each p_i

$$O(n^2\ p) = O(n^2\ 2^{log\ p}\ )$$

- Algorithms with such running times are called *pseudopolynomial*
- Corollary: We can solve Knapsack efficiently when the p_i's are not too big

An application in voting theory

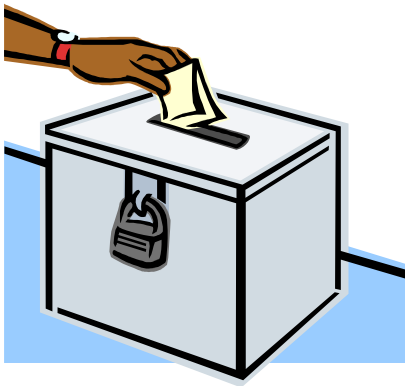
Suppose the parliament is voting for passing a bill

- Each party has w_i members in the parliament
- We need at least q votes for the bill to pass
- A coalition of parties has the power to make the bill pass if $\Sigma_{\text{i in S}}\, w_i \geq q$

More formally: weighted voting games

- Set of agents (parties) N
- Each agent i has a weight w_i
- A game has a *quota* q (typically $q \geq w_i$)
- Each coalition S $\subseteq$ N has a value $v(S)$

  (this case: 1 if $\Sigma_{\text{i in S}}\, w_i \geq q$, 0 otherwise)
- Winning coalitions: coalitions with value 1



Weighted voting games

- Consider $q = 51$, $w_1 = 48$, $w_2 = 28$, $w_3 = 24$
  - No single agent wins, every coalition of 2 agents wins, and the grand coalition wins
  - No single agent has more *power* than any other

- Voting *power* is not proportional to voting *weight*
  - How do we *measure voting power*?
    - Your ability to change the outcome of the game with your vote

Power Indices

- Measuring the probability of affecting the outcome
  - There exist various models to capture this intuition in the literature
- Two prominent indices
  - Banzhaf Power Index
  - Shapley-Shubik Power Index
    - Essentially the Shapley value, but for games with 0/1 values

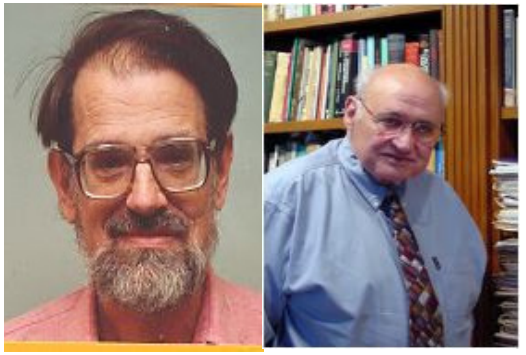
The Banzhaf Power Index

- *Pivotal agent* (of a winning coalition): an agent that causes a winning coalition to lose when removed from it

- The *Banzhaf Power Index* of an agent is the portion of all coalitions where the agent is pivotal

$$\beta_i(v) = \frac{1}{2^{n-1}} \sum_{S \subset N | i \in S} [v(S) - v(S \setminus \{i\})]$$

The Shapley-Shubik Index



- The portion of all **permutations** where the agent is pivotal

- Direct application of the Shapley value for simple coalitional games

$$sh_i(v) = \frac{1}{n!} \sum_{\pi \in \Pi} [v(S_\pi(i) \cup \{i\}) - v(S_\pi(i))]$$

$S_\pi(i) = \{j : j \text{ precedes } i \text{ in } \pi\}$

Computing power indices

No poly-time algorithm is known for power indices

BUT:

Theorem: For weighted voting games with n agents and quota q, there is an algorithm that computes the Banzhaf and the Shapley-Shubik index in time $O(n^2 q)$

Corollary: We can compute the power indices in polynomial time if q is bounded by some polynomial in n

The complexity of **counting**

The class **#P**

Counting problems

- Some computational problems are inherently counting problems
- Recall the Banzhaf index:

$$\beta_i(v) = \frac{1}{2^{n-1}} \sum_{S \subset N | i \in S} [v(S) - v(S \setminus \{i\})]$$

- Need to count *coalitions* where *i* is pivotal
- The Shapley-Shubik index:

$$sh_i(v) = \frac{1}{n!} \sum_{\pi \in \Pi} [v(S_\pi(i) \cup \{i\}) - v(S_\pi(i))]$$

- Need to count *permutations* where *i* is pivotal

Other examples of counting problems

- #SAT (counting version of SAT): given boolean formula φ how many satisfying assignments does it have?

- #VERTEX COVER: given (G, k) how many covers of size at most k are there?

What do these problems have in common?

For each of the objects that you want to count (e.g. assignments) we can verify efficiently that it is a valid solution

#P: A class for counting problems

- **#P** is the class of *function problems f*, for which there is a decision problem B$\in$ **P** s.t.:

  Input x:

  $f(x)$ = # witnesses y s.t. (x, y) is a *yes* instance of B

- compare to **NP** (*decision problems*)

  Input x:

  $\exists$ witness y s.t. (x,y) is a *yes* instance of B

Example

For #SAT: $f(\varphi)$ = # assignments that satisfy φ

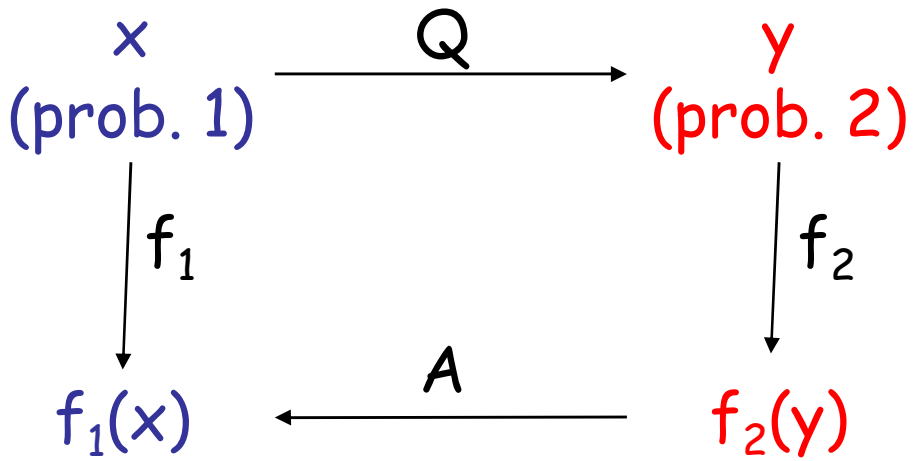
- Decision problem B(φ, y): given a boolean formula φ and an assignment y, does y satisfy φ?

- B can be solved in polynomial time

- $f(\varphi) = \#y$ s.t. B(φ, y) = yes

Reductions

- Reduction from function problem f_1 to function problem f_2
  - Described by two efficiently computable functions Q, A



$$f_1(x) = A(f_2(Q(x)))$$

Completeness

A problem f is **#P**-complete if

- f is in **#P**
- every problem in **#P** reduces to f

Theorem: #SAT is **#P**-complete

(analog of **NP**-completeness of SAT)

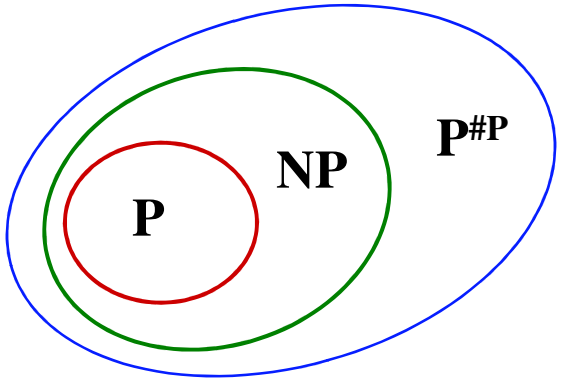
- The counting version of many **NP**-complete problems is **#P**-complete
- BUT not all **#P**-complete problems arise form **NP**-complete decision problems

Relationship to other classes - How difficult is **#P** ?

- To compare to classes of decision problems, consider

  $\mathbf{P^{\#P}}$**:** decision problems, decidable by an algorithm that
  - can make "calls" to **#P** problems
  - apart from the calls, it runs in poly-time

e.g. does a formula φ have more than *k* satisfying assignments?



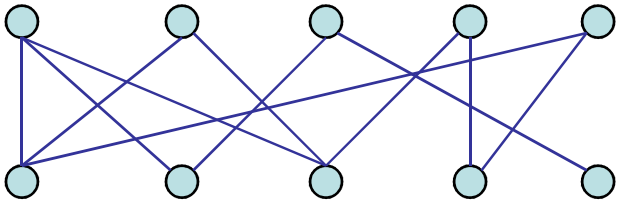
Relationship to other classes - How difficult is **#P** ?

Question: is **#P** hard because it entails *finding* **NP** witnesses?

…or is *counting* difficult by itself?

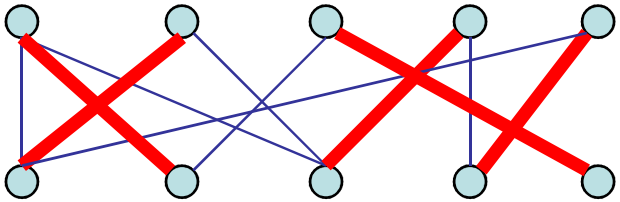
Recall bipartite matchings

- G = (U, V, E) bipartite graph with |U| = |V|
- a **perfect matching** in G is a subset M ⊂ E that touches every node, and no two edges in M share an endpoint



Recall bipartite matchings

- G = (U, V, E) bipartite graph with |U| = |V|
- a **perfect matching** in G is a subset M ⊂ E that touches every node, and no two edges in M share an endpoint



Bipartite matchings

- #MATCHING: given a bipartite graph G = (U, V, E) how many perfect matchings does it have?

Theorem: #MATCHING is **#P**-complete.

- But… can *find* a perfect matching in polynomial time!
  - counting itself must be difficult

Back to voting

Theorem: Computing the Banzhaf and the Shapley-Shubik indices is **#P**-complete.

- Easy to see it is in **#P:**
  - Decision problem B(*i*, S): Given a coalition S, containing *i*, *is i pivotal for S?*
  - #witnesses = # coalitions in which *i* is pivotal
- **#P**-hardness can be shown by a reduction from #MATCHING

More **#P**-complete problems

- #TSP: Given (G, k), how many traveling salesman tours of cost at most k are there?

- #KNAPSACK: In how many ways can we fill the knapsack s.t. profit is at least k?

- Counting Nash Equilibria in Noncooperative Games

Space complexity

Space complexity

- So far we have only addressed **time** complexity

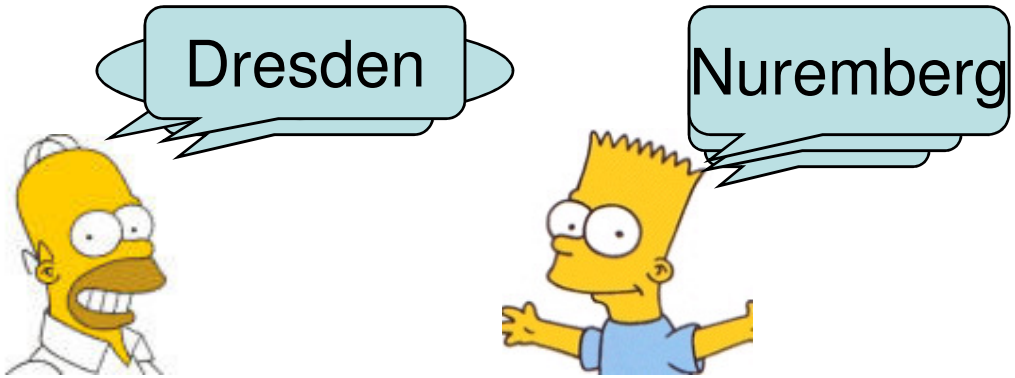
- What about **space** (memory) requirements ?

Perfect information games with alternating moves

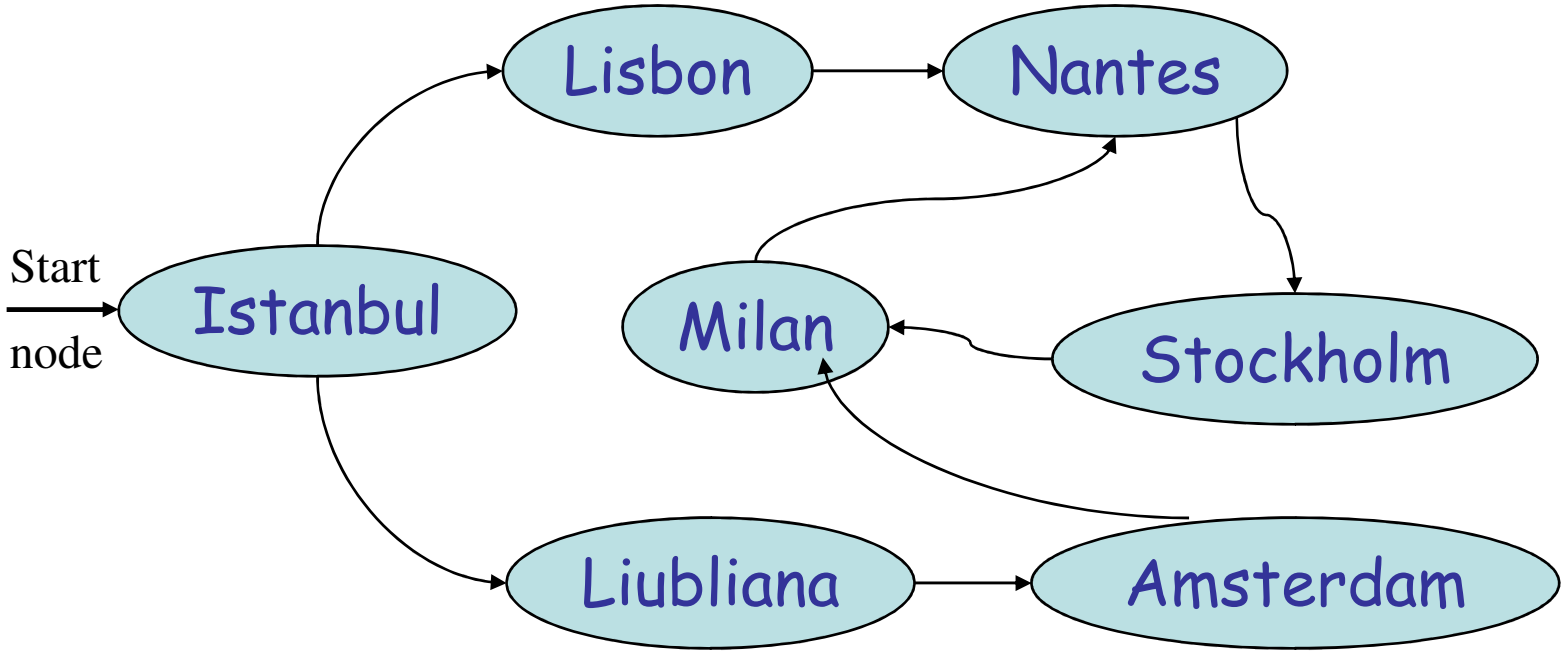
- Games with (typically) 2 players
- The game proceeds in rounds. In each round one of the players makes a move
- Players alternate in making moves
- Players observe and remember the history of the moves made by the other players during the game

Example: The geography game

- Each player picks a city, which begins with the letter that ended the previous word
- Cannot use a city that has already been used
- The player that cannot move any more loses



A Graph-theoretic view



More formally:

Suppose Player 1 moves first

The Generalized Geography Problem: Given a directed graph on a set of n nodes, and a designated start node, does Player 1 have a winning strategy?

What is the space complexity of deciding if Player 1 has a winning strategy ? (as a function of n)

Space complexity classes

SPACE(f(n))**:** the class of problems decidable by an algorithm that uses O(f(n)) space

PSPACE: the class of problems decidable by an algorithm that uses polynomial amount of space

NPSPACE: the class of problems *verifiable* by an algorithm that uses polynomial amount of space

Relations between space classes

Savitch's theorem: **NSPACE**(f(n)) $\subseteq$ **SPACE**($f(n)^2$)

Corollary: **PSPACE = NPSPACE**

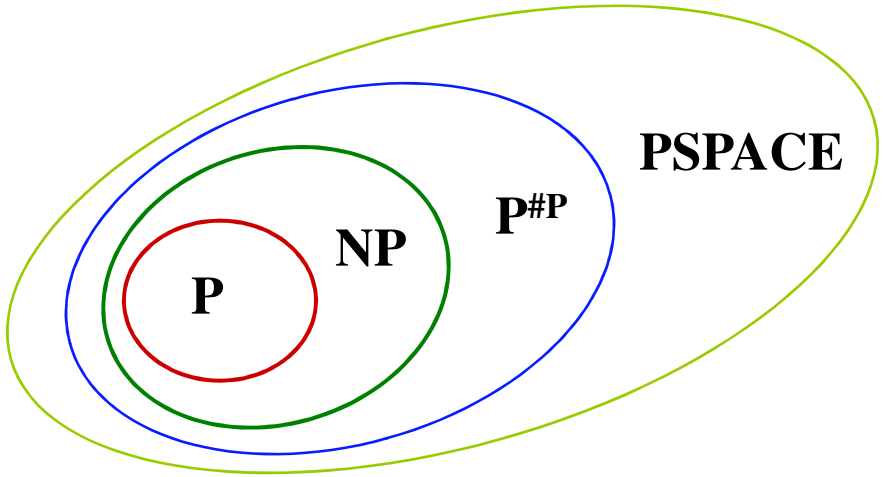
For space complexity,

verification $\Rightarrow$ computation with almost same complexity (compare with time complexity)

Relations with time classes

PSPACE is the most difficult class we have seen so far:

$$\mathbf{P \subseteq NP \subseteq P^{\#P} \subseteq PSPACE}$$



Completeness

A problem L is **PSPACE**-complete if

- L $\in$ **PSPACE**
- All problems in PSPACE are *polynomial time* reducible to L

Complete problems for **PSPACE**?

A classic completeness result

The TQBF problem (a quantified version of SAT):

Input: A fully quantified Boolean formula (all variables fall under some quantifier)

e.g. $\varphi = \forall x \exists y \forall z\ [(x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)]$

Decide if φ is true

Theorem: TQBF is **PSPACE**-complete

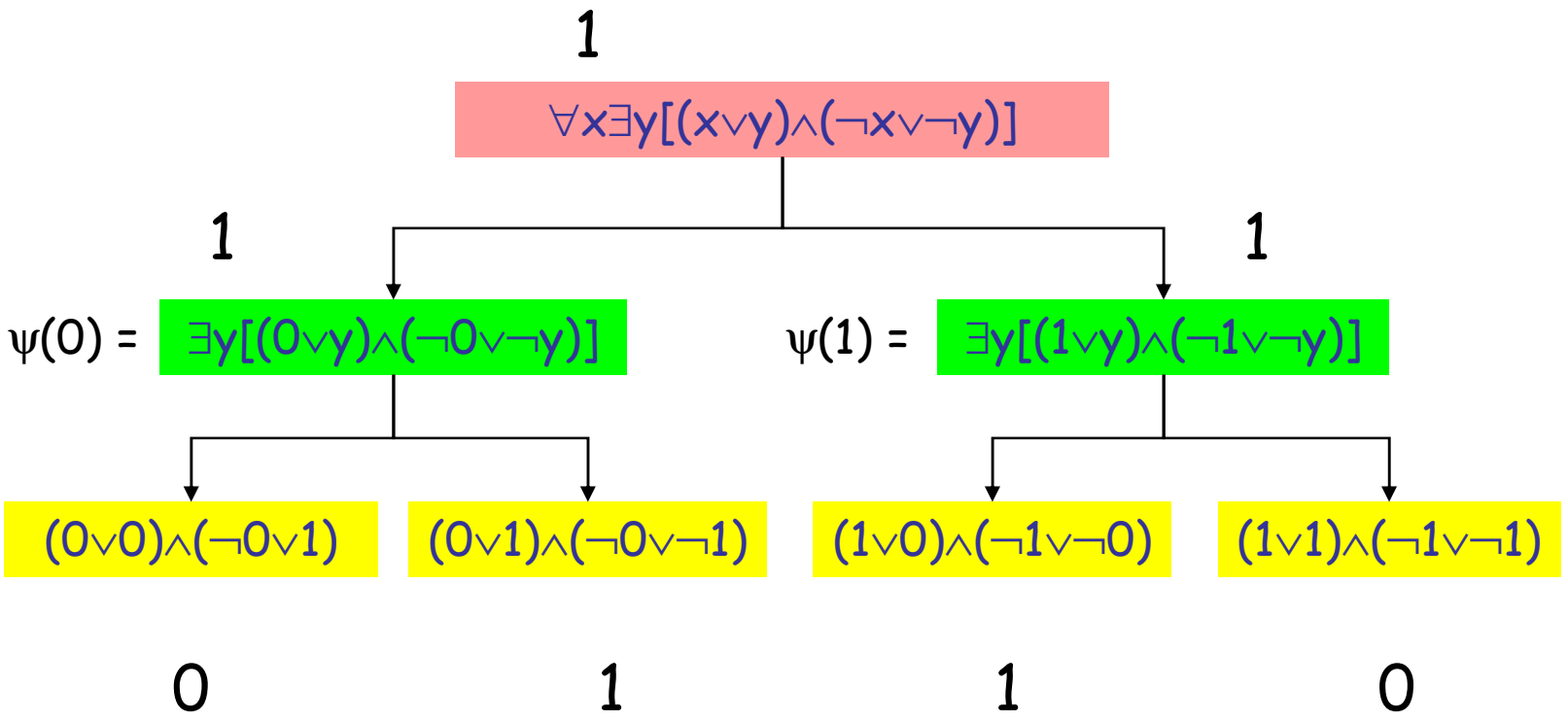
A classic completeness result

Proof: Here, we will only show it belongs to **PSPACE**.

Poly-space algorithm A for evaluating φ:

- If φ has no quantifiers: evaluate it
- If $\varphi = \forall x(\psi(x))$ call A on $\psi(0)$ and on $\psi(1)$; Accept if *both* are true.
- If $\varphi = \exists x(\psi(x))$ call A on $\psi(0)$ and on $\psi(1)$; Accept if *either* is true.

Algorithm A for TQBF



Analysis of the algorithm

Main idea: space can be reused (unlike time)

- Both recursive calls can use the same space,
- Total space needed is polynomial in the number of variables (the depth of the recursion)

$\Rightarrow$ TQBF is polynomial-space decidable

The complexity of the geography problem

Theorem: The Generalized Geography Problem is **PSPACE**-complete

Reduction from TQBF. A winning strategy can be described as a quantified expression over non-boolean variables

(∃ move for Player 1) (∀ move of player 2) (∃ move for Player 1) ….

Other **PSPACE**-complete problems

- Many other alternating games (GO, mahjong, solitaire games)
- Deadlock reachability in certain systems of communicating processes
- Existence of Nash equilibria in stochastic games

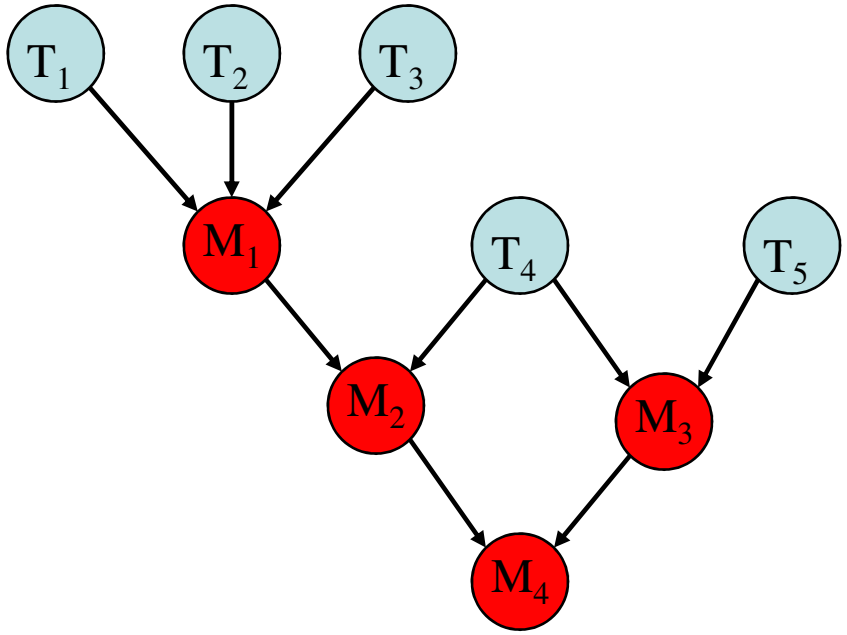
And more: decision making under uncertainty

Stochastic Task Scheduling

- Two identical processors
- A set of tasks N
- A set of *mandatory* tasks $M \subseteq N$
- A task dependency graph
  - A task can start if one of its predecessors has finished
- Execution time of each task: Poisson random variable

Goal: Minimize *expected* elapsed time till completion of last mandatory task

Stochastic scheduling



Theorem: Stochastic Scheduling is **PSPACE**-complete (and so are many other problems involving decision making in stochastic environments)

Between P and PSPACE

$\sum_i$: problems reducible to deciding the satisfiability of a formula of the form:

$$\underbrace{\exists x_1 \forall x_2 \exists x_3 \ldots}_{\text{i alternating quantifiers}} R(x_1, x_2, x_3, \ldots)$$

$\prod_i$: problems reducible to deciding the satisfiability of a formula of the form:

$$\underbrace{\forall x_1 \exists x_2 \forall x_3 \ldots}_{\text{i alternating quantifiers}} R(x_1, x_2, x_3, \ldots)$$

The Polynomial Time Hierarchy

- $\Sigma_1 = \mathbf{NP}$
- $\Sigma_i \subseteq \Sigma_{i+1}, \Pi_i \subseteq \Pi_{i+1}$
- $\mathbf{PH} := \cup \Sigma_i = \cup \Pi_i$
- $\mathbf{PH} \subseteq \mathbf{PSPACE}$
- In fact: $\mathbf{PH} \subseteq \mathbf{P}^{\#\mathbf{P}}$ (Toda's theorem)

An example

The Minimum Equivalent DNF Problem: Given a Boolean formula φ and an integer k, is there an equivalent formula with size at most k ?

Theorem: Min. Equivalent DNF problem belongs to $\sum_2$

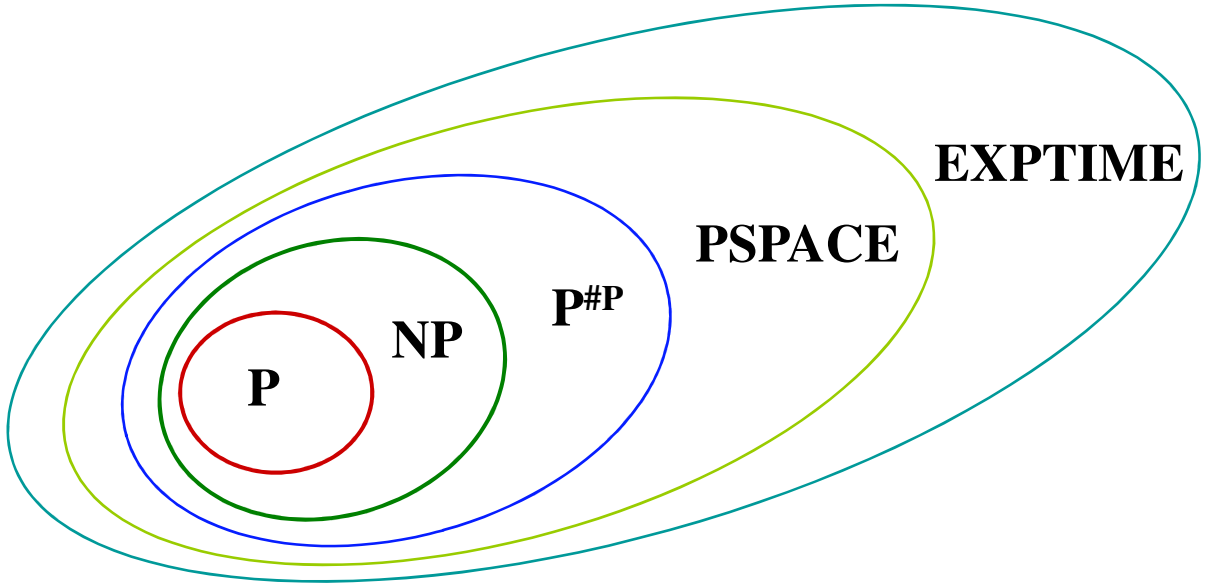
Informal proof: The problem can be stated as:

($\exists$ formula ψ of size $\leq k$) ($\forall$ truth assignment) ($\varphi \leftrightarrow \psi$)

- Has also been shown to be $\sum_2$ - complete

Beyond **PSPACE**

- **EXPTIME**: problems decided by an exponential time algorithm



- We know that **P ≠ EXPTIME**
- Conjecture: **P ≠ NP ≠ P^#P**, and **PSPACE ≠ EXPTIME**
- **P^#P = PSPACE?**

Coping with hardness results: approximation algorithms

- Some hard problems may still admit efficient approximation algorithms

- Definition: For a minimization problem, an algorithm achieves a γ-approximation, if $\forall$ instance I, the solution returned satisfies:

$$SOL(I) \leq \gamma\, OPT(I)$$

- Polynomial Time Approximation Scheme (PTAS): A family of algorithms $\{A_\varepsilon\}$ s.t. $\forall\ \varepsilon > 0$ and for every instance I, A_ε returns a solution in polynomial time with

$$SOL(I) \leq (1 + \varepsilon)OPT(I)$$

- Analogous definitions exist for maximization problems

Examples of approximation algorithms

Theorem 1: The Knapsack problem admits a PTAS (based on the pseudopolynomial time algorithm for Knapsack)

Theorem 2: The Vertex Cover problem admits a 2-approximation

Proof: Output the vertices of a *maximal* matching M (one that cannot be improved upon by adding more edges)

- Maximal matching is a cover (otherwise can add an edge)
- OPT $\geq |M|$ (need at least one vertex from each matched edge)
- SOL $= 2|M| \leq 2$ OPT

A different approach: randomized algorithms

- What if an algorithm is allowed to make errors with small probability?
- **BPP**: problems for which there is an efficient algorithm that outputs the correct solution with prob. at least 2/3
- We can lower the probability of error by repetition
- $\mathbf{P} \subseteq \mathbf{BPP} \subseteq \Sigma_2$

Summary

- Complexity theory: language for talking about the difficulty of a problem

- Derives upper and lower bounds on the resources required for solving a problem
  - Time
  - Space

- Worst-case analysis:
  good heuristics still possible

Some standard references

- M. Garey, D. Johnson. *Computers and Intractability - A Guide to the Theory of **NP**-completeness*. W. H. Freeman, 1979.

- C. Papadimitriou. *Computational Complexity*. Addison-Wesley, 1994.

- M. Sipser. *Introduction to the Theory of Computation*. PWS, 1996