Computational Complexity

The Class P

- The class P contains all decision problems that are decidable by an algorithm that runs in polynomial time in the size of the input.
  - Does this define a class of languages?
  - Yes:
    - Each language is a specific decision problem whose encodings of "yes instances" (the language) is decided by a deterministic TM, M, where M decides that particular language in polynomial time.
  - Recall: These languages are decidable

The Class P

- Are there problems not in P:
  - Yes, we know that HALT (and others) are not in P
    - They are not even solvable (decidable)
- Are there decidable problems not in P?
  - Yes – there are problems that provably need exponential time
  - There are other problems for which we don't know whether they are in P or not
    - Let's consider these…

Satisfiability (SAT)

- Instance
  - A logical expression containing
    - variables x_i
    - logical connectors &, |, and !
    - In conjunctive normal form (C_1 & C_2 & C_3 … & C_n)
- Question
  - Is there an assignment of truth values to each of the variables such that the expression will evaluate to true.

Satisfiability (SAT)

- Example of an instance
  - $(x_1 \mid x_2 \mid x_3) \;\&\; (x_4 \mid !x_2) \;\&\; !x_4 \;\&\; (!x_1 \mid !x_3)$
  - One solution:
    - x_1 = true
    - x_2 = false
    - x_3 = false
    - x_4 = false

Satisfiability (SAT)

- Is it decidable?
  - Naïve algorithm to solve
    - Systematically consider all combinations of assignments of True and False values for all variables and test the expression
- What's the worst case time complexity?
  - In the worst case, for n variables
    - $T(n) = O(2^n)$

Satisfiability (SAT)

- Can we do better?
  - No known polynomial algorithm
    - Either one doesn't exist… or
    - One exists and we haven't found it yet.

Hamiltonian Cycle

- A Hamiltonian cycle in a graph is a cycle that goes through each vertex exactly once (and returns to the start)
  - UHAMCYCLE = {<G> | G is an undirected graph with a Hamiltonian cycle}
  - Is UHAMCYCLE decidable?
    - Yes – brute force search takes exponential time
  - Nobody knows if UHAMCYCLE is solvable in polynomial time

(U stands for undirected)

Clique

- A clique C in a graph is a subset of vertices such that every two vertices in C are adjacent (meaning they have an edge between them).
  - CLIQUE = {<G,k> | G is a graph, k is an integer, and G has a clique of size k}
    - Another way of saying this is that there is a complete subgraph of size k.
  - Is CLIQUE decidable?
    - Yes, there are fixed number of subsets of size k of a graph with finite vertices. Brute force search.

Polynomially Verifiable

- A verifier for a language A is an algorithm V, where A = { w | V accepts <w,c> for some string c}.
- A polynomial time verifier runs in polynomial time in the length of w.
- The string c is called the certificate, or proof, that w is in A.

NP

- NP is the class of languages that have polynomial-time verifiers

- So, it may be that the algorithm can't run in polynomial time, but at least a specific certificate (or proof) that shows that the string is in the language can be checked in polynomial time
  - (that's all polynomial verifiability means – given a certificate, you can use that to confirm in polynomial time that the string should be accepted)

NP

- Is SAT polynomially verifiable?
  - Sure, given the instance, and a supposed solution for that instance (assignment of truth values to the variables), just apply it and check.
- Is UHAMCYCLE polynomially verifiable?
  - Sure, given the graph and a supposed cycle (in order), check to make sure each vertex is included and the edges exist to form the cycle
    - Polynomial time in the size of the graph

NP

- Is CLIQUE polynomially verifiable?
  - Sure, given the graph and a supposed clique of size k, confirm that each pair of vertices has an edge between them
    - Polynomial time in the size of the graph
      - For graph with |V| vertices, there are at most ${}_{|V|}C_2 < |V|^2$ edges
        - Run through this list of edges at most ${}_kC_2 < k^2$ times looking for an edge in the necessary collection, where k is the size of the clique

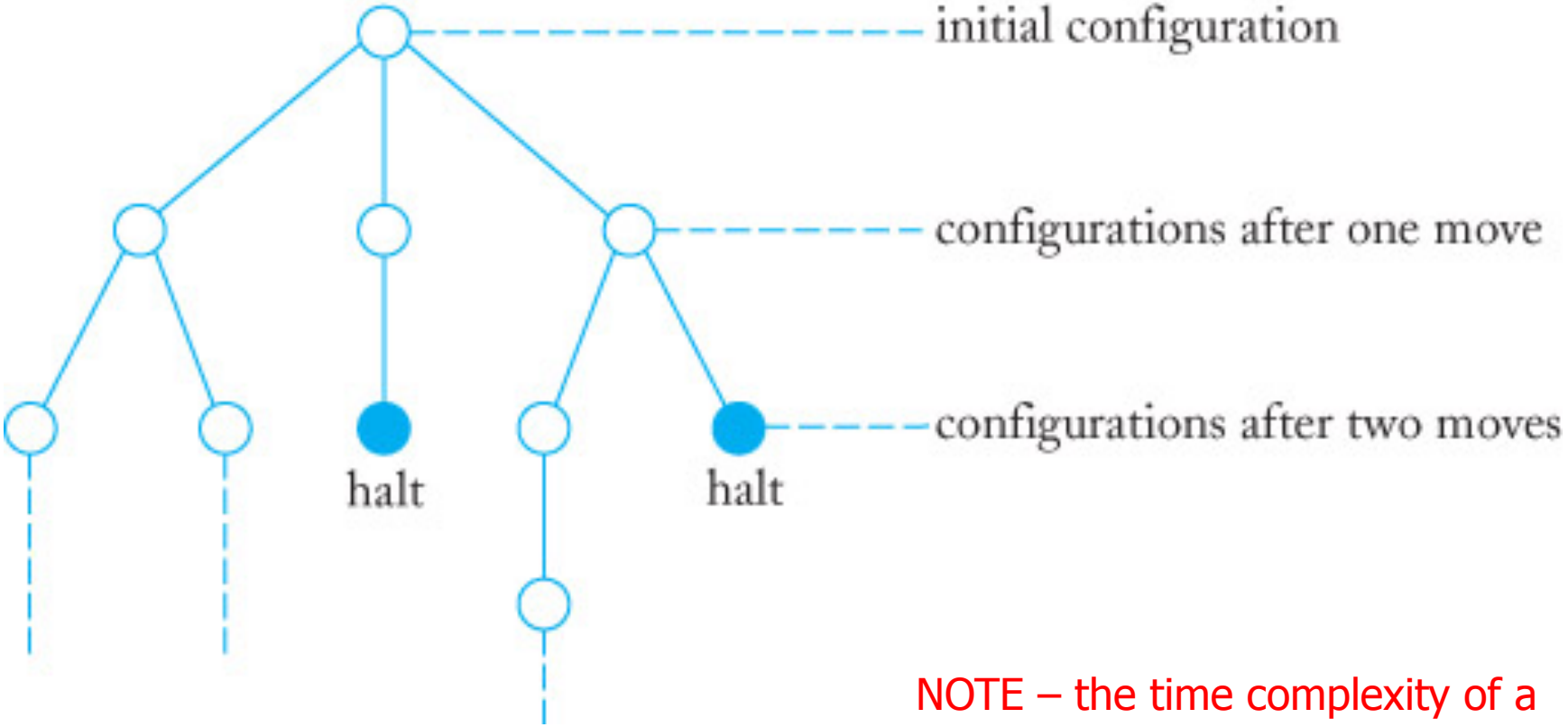
Non-Deterministic TM

- Let's reconsider the NDTM
  - Same as the ordinary TM except:
    - The transition function will return a **set** of triplets
      - (q, x, D)
    - For each state / symbol combination, 0 or more transitions can be defined.
    - The machine can "choose" which transition to take.
  - δ: Q x $\Gamma \rightarrow$ P (Q x Γ x {R, L})
  - A NDTM will accept as long as one branch accepts

Non-Deterministic TM

- Simulating a NDTM on a TM
  - Consider all paths through the TM.
  - One way to think about it…
    - A TM “replicates” itself whenever there is a choice.
    - Each “replication” continues computation along a given path

Non-Deterministic TM



NOTE – the time complexity of a non-deterministic TM is defined to be the time associated with its longest computing branch

Non-Deterministic TM

- If we map all paths that can possibly be taken:
  - Number of paths may grow exponentially as a function of the number of moves considered in any path
  - Example:
    - Suppose at each configuration there are 2 possible moves.
    - Number of paths: 2^n paths of length n

Another View of the Class NP

- Theorem 7.20:
  - A language is in NP if and only if it is decided by some non-deterministic polynomial time Turing Machine

Remember that P requires the language to be decidable in polynomial time by a deterministic TM. So NP allows extra flexibility. There may be an exponential number of paths, but as long as they all compute in polynomial time, it's in NP.

The Class NP

- The class NP contains all decision problems that are decidable by a non-deterministic Turing machine that runs in polynomial time.
- For each w accepted, there is at least one accepting branch of computation.
  - All branches must run in polynomial time
- Note that NP does not stand for “not” polynomial
  - Instead it refers to nondeterministic polynomial

Recall: P is Robust

- Recall that the class P is robust
  - It is the same class for all reasonable deterministic computational models
    - All reasonable deterministic computational models are polynomially equivalent
- A non-deterministic TM is NOT a "reasonable deterministic computational model"
- A non-deterministic TM may run exponentially faster than an equivalent deterministic TM

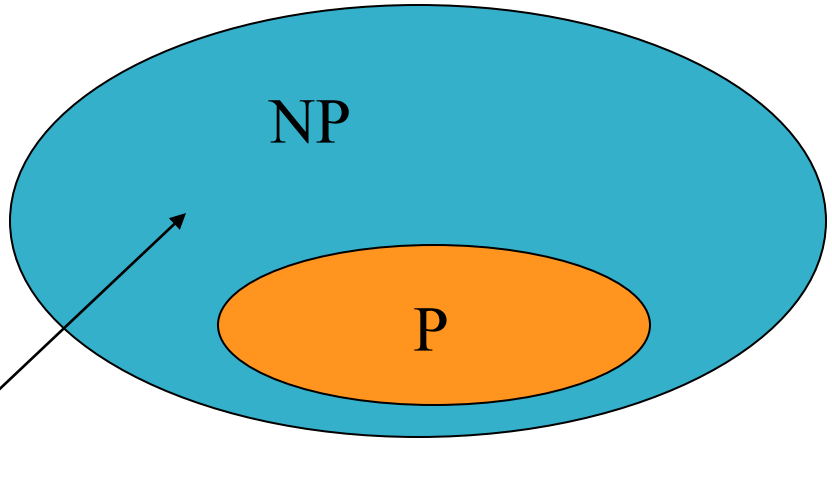
Satisfiability (SAT) using NDTM

- Running SAT on a non-deterministic TM.
  - Step 1: "Guess" a set of boolean assignments to each variable (this is the non-deterministic part – 2^n different choices)
  - Step 2: Evaluate the truth of the entire expression using the guessed assignment (this part is deterministic)
  - Can certainly do each step in polynomial time

The Class NP

- Clearly P is a subset of NP
- Does P=NP?

(a deterministic TM is just a special case of a NDTM that only has one branch of computation, so anything in P is in NP)



Is there something in here?

After 40+ years of research…nobody knows for certain

The Class NP-Complete

- The hardest of the problems in class NP are in the class of NP-complete problems:
- If you could find a polynomial-time algorithm for an NP-complete language, then this would provide an algorithm for all languages in NP
  - P = NP
  - Fame and fortune would follow
- All NP-complete problems are "equally" difficult.

The Class NP-Complete

- To show that a problem is in NP-Complete, you must show two things:
  - The problem is in NP
  - Every other problem in NP can be polynomially reduced to this problem
    - Thus every other problem can be solved without too much extra work if we have a solution to this problem

NP-Complete Problems

- Would seem impossible to show that a specific problem is NP-complete
  - i.e. that none of an infinite number of other NP problems is harder than that specific problem
- Cook-Levin Theorem: SAT is NP-complete
  - Showed that any polynomial-time NDTM can be modeled by a boolean formula
- From there, other problems can be shown NP-complete by comparison to SAT

The Class NP-Complete

- What does it mean for "X not to be harder than Y"
  - It means that there is a polynomial-time function that reduces X to Y
  - X can efficiently be reduced to Y
- What does it mean for X and Y to be "equally" difficult
  - Each can be efficiently reduced to the other

Examples in NP-Complete

- SAT
- CLIQUE
- UHAMCYCLE

- All are NP-complete
  - Once we have SAT, we prove another by showing that SAT can be reduced to the other problem. (Same idea of black box that we use to show a problem is undecidable.)

Examples in NP-Complete

- Multiprocessor Scheduling
- Instance: Set T of tasks, a number $m \in N$ of processors, length $l(t) \in N$ for each $t \in T$, and a deadline $D \in N$

- Question: Is there an m-processor schedule for T that meets the overall deadline D?

Examples in NP-Complete

- MAX-CUT
- Instance: A graph G = (V,E) and an integer k

- Question: Does G have a cut of size k or more? That is, can we partition V into S and T such that there are at least k edges from S to T?

Examples in NP-Complete

- 3 COLOR
- Instance: A graph G

- Question: Can we color the vertices of G with 3 colors such that no two adjacent vertices have the same color

Examples in NP-Complete

- BIN PACKING
- Instance: Finite set U of items, size $s(u) \in N$ for each $u \in U$, a positive integer bin capacity B, and a positive integer K

- Question: Can U be packed in K bins? That is, is there a partition of U into disjoint sets $U_1, \ldots, U_K$ such that the sum of the sizes of the items in each U_i is B or less

Examples in NP-Complete

- CROSSWORD PUZZLE CONSTRUCTION
- Instance: A finite set W of words and an n x n matrix A of 0's and 1's

- Question: Can an n x n crossword puzzle be built up from words in W and blank squares corresponding to the 0's in A

Dealing with NP-Completeness

- If you can prove that your problem is NP-complete, it is probably not worth spending a lot of time searching for an exact polynomial time solution
- Options to make the problem polynomial:
  - Some NP-complete problems have solutions that are, on the average, polynomial
  - Restrict the problem
  - Approximation – good, but not optimal solution
  - Heuristics – may or may not work out