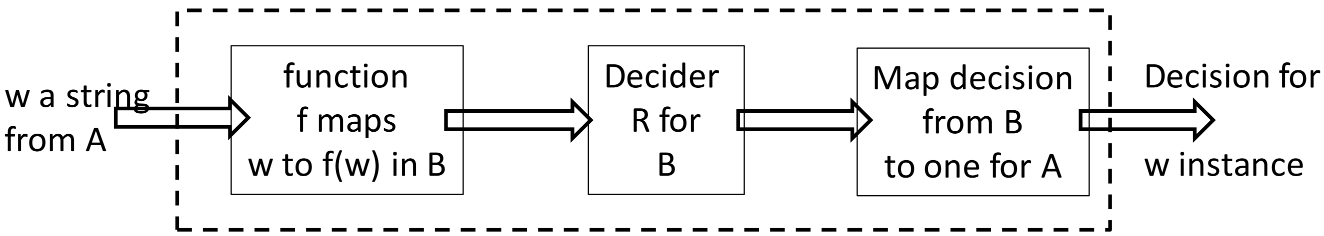
pp. 292-311. **The Class NP-Complete** (Sec. 7.4)

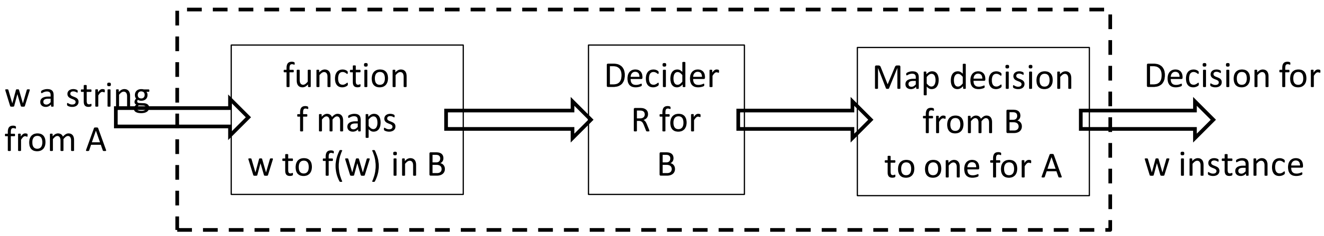
- P = {L|L decidable in poly time}
- NP = {L|L verifiable in poly time}
- Certainly all P is in NP
- ***Unknown*** if NP is bigger than P
- (p. 299) **NP-Complete = subset of NP where if any one is solvable in poly time, then all in NP-Complete are**
  - No one has found polynomial algorithms for any in it
  - If someone finds such an algorithm for any problem in NP-Complete, then NP moves to P
  - ***Unknown*** if NP-complete = NP
- (p 300) **Theorem 7.27 SAT is in P iff P=NP**
  - 1^{st} NP complete problem
  - Will prove any NP problem convertible into SAT
  - Needs several intermediate theorems first

- (p. 261) Definition: **Language A is Turing-Reducible to B, written $A \leq_T B$, if A is decidable relative to B using some function f:A->B that transforms instances**
  - i.e. any w_A from A can be mapped/reduced to a w_B in B such that B's decision on w_B can be converted into decision on w_A



  - If B decidable, then so is A.
- (p. 300) Definition 7.28: **f:∑* -> ∑* is a polynomial time computable function** if
  - Some polynomial time TM exists
  - which when started with w on tape,
  - halts with just f(w) on its tape,

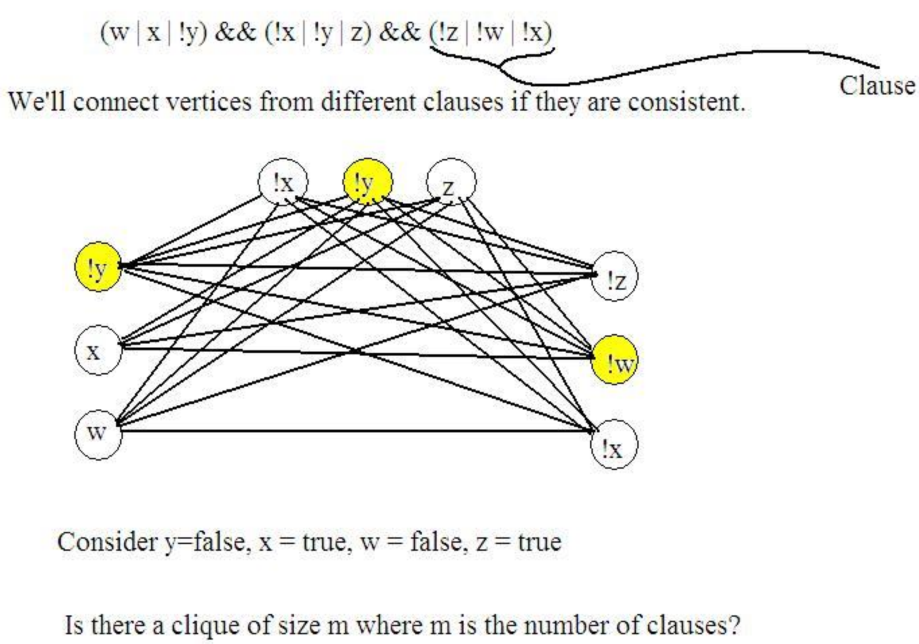
- (p. 300) Def. 7.29: Language A is **polynomial time reducible to** language to B (Written **$A \leq_P B$**) if
  - There is some polynomial time computable function f
  - Where w is in A iff f(w) is in B
  - See Fig. 7.30, p.301
  - Thus for every string w in A there is a string f(w) in B
  - And if w not in A, then f(w) not in B
- If you can write a polynomial time decider for B
  - then, using f, can write a polynomial time solver for A



- (p. 301) Theorem 7.3.1. **If $A \leq_P B$ and B in P, then A in P**
- Given any w in A
  - Compute w' = f(w) – poly time
  - Run Decider for B and output result – poly time
  - Sum of two poly time functions is still poly

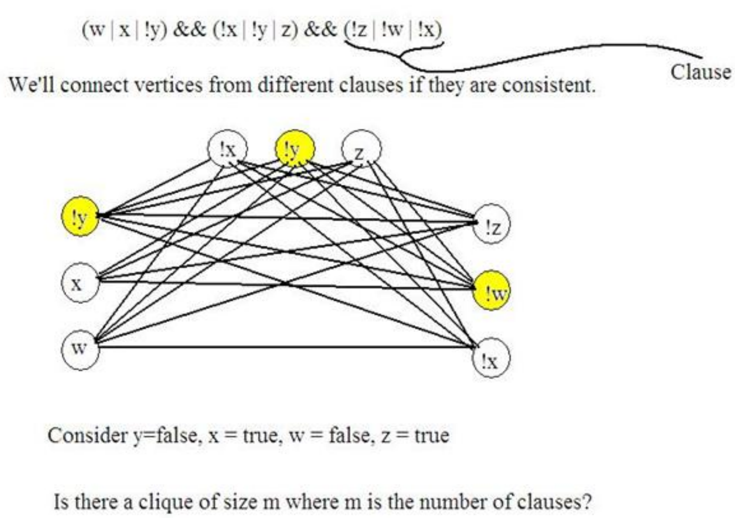
- Two sample problems
- (p. 299) **SAT: The Satisfiability Problem**
  - SAT = {wff| wff is satisfiable}
  - Wff = Well-formed-Formula, made up of
    - Boolean Variables (may take on only 0 or 1)
    - Expressions built from AND, OR, NOT
  - (p. 302) **CNF**: a wff is in **conjunctive normal form:**
    - The AND of a set of **clauses** (called a **conjunction**)
      - Where each clause is the OR of a set of **literals** called a **disjunction**
        - Where each literal is a variable or its complement
- **3SAT = {wff| wff in CNF has exactly 3 literals}**
  - E.g. $(a_1 \vee b_1 \vee c_1)\&\&(a_2 \vee b_2 \vee c_2)\&\&\ldots(a_k \vee b_k \vee c_k)$
- Also: **CLIQUE ={<G,k>| G includes a k-clique}**
  - Where a k-clique has k vertices with edges to each other
  - CLIQUE known to be in NP (p. 296)

- (p.302) **3SAT is polynomial time reducible to CLIQUE**
  - Proof: convert wffs to graphs
    - Wff C = C_1 ^ C_2 … ^ C_k (i.e. k clauses)
    - C_i = a_i v b_i v c_i where a_i, b_i, c_i all literals
  - f converts wff C to string <G,k>
    - G has k groups of 3 vertices (each group from a clause)
    - Each vertex in a triple corresponds to a literal
      - And named to match
    - All vertices in G have edges to all other vertices <u>except</u>
      - **No edges between vertices in same triple**
      - **No edge between vertices with opposite labels** (i.e. same variable, different signs)
    - See page 303 for example



http://cs.nmu.edu/~mkowalcz/cs422w09/36/reduction2.jpg

- (p. 303) Wff C is satisfiable iff G has a k-clique
  - =>: If wff has a satisfying assignment, then each clause has at least one literal that is true
    - Choose just one of these in each triple
      - By construction there must be an edge between all selected vertices & thus must be a k-clique
  - <=: If the graph has a k clique
    - Cannot include vertices in same triple (not permitted by construction)
    - Cannot include literals with opposite sides (not permitted by construction)
    - Assign value to variables to make each literal in k-clique true
    - Result is a satisfying assignment
- If CLIQUE is solvable in poly time, so is 3SAT and vv



- (p. 304) Def 7.34. **B is NP-complete if both B in NP and *__every__* A in NP is polynomial time reducible to B**
- (p. 304) Theorem 7.35. **If B is in NP-complete and B in P, then P = NP**
  - Any member can be converted to any other by series of polynomial time f
- (p. 304) Theorem 7.36. **If B in NP-complete, and $B \leq_P C$ for some C in NP, then C is also NP-complete**
  - Since B is NP-complete, every language in NP is polynomial time reducible to B,
  - But B is polynomial time reducible to C
  - Can compose the functions, so every language in NP is also polynomial time reducible to C
  - Thus C also in NP-Complete

- (p. 304) **COOK-LEVIN Theorem. SAT is NP-complete**!
  - First show SAT is in NP
    - A nondeterministic TM N can guess an assignment and then verify in polynomial time. Thus in NP
  - Now *show any A in NP is polynomial time reducible to SAT*
    - n = |w|, w in A
    - N an NTM that decides A in $O(n^k)$ for some k
      - Tape used is thus at most n^k cells in length
    - Construct **tableau** (table) of size n^k x n^k (p. 305)
      - n^k rows (one for each step of NTM)
      - Each row is a configuration
        - 1st row is **starting configuration** of N on w
        - Each configuration at most n^k symbols long (columns – max tape length)
        - For convenience, each config starts & ends with #
        - Each entry in table called a **cell**
          - A state or a symbol
          - Let C = Q U Γ U {#} = state set + tape chars
    - Tableau is **accepting** if some row an accepting config
      - And row i+1 follows row i via valid transition
    - Now to show N accepts w is eqvt to question “does an accepting tableau exist?”

- Conversion f from A to SAT: Each cell in tableau has a symbol from C
  - Define a set of $2^k x 2^k x|C|$ Boolean variables $x_{i,j,s}$
    - i, j between 1 and 2^k
    - s over all symbols in C
    - $x_{i,j,s}$ = 1 iff cell[i,j] contains symbol s
  - (p. 306) Define a wff made up of AND of 4 sets of clauses
    - Wff_{cell} = clauses ensure 1 variable is true for each i,j
    - Wff_{start} = clause that forces variables with i=1 to have initial config of N
    - Wff_{accept} = clauses that guarantees an accepting configuration appears as some row
    - Wff_{move} = clauses that guarantee that a move from the config for row i to row i+1 is valid
      - See 6 “windows” on p. 308 for rows I and i+1
      - Centered around state symbol in row i
  - This conversion can be done in poly time
  - Thus any problem in NP can have its decider (if it exists) converted into a SAT problem in poly time
  - Solving the SAT problem finds answer for A

- Sample tableau (for deterministic TM accepting $(aa)^n$)

TM: decide {(aa)*}							
state	**tape**	**new state**	**new tape**	**dir**			
q0	a	q1	a	R			
q1	a	q0	a	R			
q0	blank	q2	blank	L			
Tableau for aa		n^ (+2)					
Steps (each a configuration)		**1**	**2**	**3**	**4**	**5**	**6**
	1	#	q0	a	a	bl	#
	2	#	a	q1	a	bl	#
	3	#	a	a	q0	bl	#
	4	#	a	q2	a	bl	#
3 cells =	4x6x6	144	variables				

Variable Assignments		j					
i	**s**	**1**	**2**	**3**	**4**	**5**	**6**
1	#	1	0	0	0	0	1
1	a	0	0	1	1	0	0
1	bl	0	0	0	0	1	0
1	q0	0	1	0	0	0	0
1	q1	0	0	0	0	0	0
1	q2	0	0	0	0	0	0
2	#	1	0	0	0	0	1
2	a	0	1	0	1	0	0
2	bl	0	0	0	0	1	0
2	q0	0	0	0	0	0	0
2	q1	0	0	1	0	0	0
2	q2	0	0	0	0	0	0
3	#	1	0	0	0	0	1
3	a	0	1	1	0	0	0
3	bl	0	0	0	0	1	0
3	q0	0	0	0	1	0	0
3	q1	0	0	0	0	0	0
3	q2	0	0	0	0	0	0
4	#	1	0	0	0	0	1
4	a	0	1	0	1	0	0
4	bl	0	0	0	0	1	0
4	q0	0	0	0	0	0	0
4	q1	0	0	0	0	0	0
4	q2	0	0	1	0	0	0

- Remember: showing a problem is NP-Complete
  - Show its in NP (i.e. NTM to create certificate & poly verifier)
  - Show some/any NP-Complete problem polynomially maps to it
    - Not always 3SAT!
- Other NP-Complete problems
  - (p. 310) **3SAT**
    - Do logic conversions from any SAT wff to 3 var clauses
  - (p. 311) **CLIQUE**
    - 3SAT reduces to it via Theorem 7.32 (p. 302)
      - 3 vertices for each clause
        - Labelled with literal name
      - Edges between all vertices, except:
        - Between vertices of a clause
        - Any vertex with any other labelled with the vertex's literal complement
    - P. 303 addresses match of satisfying solution and k-clique

- (p. 312) **VERTEX-COVER** = {<G,k>| G a graph with a subset of k vertices that has every edge in G touching at least one of the subset}
  - 3SAT reduces to (G,k) k=m+2l, m=# variables, l=# clauses
    - For each variable x create *pair* of 2 vertices (labelled x and ~x) with an edge between them
    - Each clause maps to a *triangle* labelled with variables
      - With edges to matching vertices from 1st set
    - Total of 2m + 3l vertices
  - Assume satisfying assignment, show k-cover:
    - Include m vertices from pairs that match assignment
      - Covers edges to clause triangles and other of pair
    - Each triangle has at least 1 vertex in assignment, choose other 2 (2l)
  - Assume G has a k-cover, show satisfying assignment
    - Cover must have at least one vertex in each pair
      - Otherwise edge between pair not covered
    - Cover must have at least 2 vertices in each triangle
      - Otherwise cannot get edge in triangle covered
    - For k=m+2l, above must be exact
    - M from pair must be satisfying (p. 313)

- (p. 314) **HAMPATH**: {<G,s,t>| there is a path from s to t that goes thru all vertices exactly once.}
  - 3SAT of l variables & k clauses reduces to HAMPATH.
  - For *each* variable in 3SAT construct *diamond* as Fig. 7.47
    - 3k+3 vertices in center row
      - 2-vertex pair for each clause + 1 border per clause
      - Lefthand vertex for "true" assignment
      - Righthand for "False"
    - Multiple paths from top to bottom
      - Left or right from top to center
      - Optionally across the center, in either direction
      - Left or right to lower vertex
  - Diamonds stacked on top of each other (Fig. 7.49)
    - Vertex s is topmost; vertex t is bottommost
  - Additionally, add separate vertex for each clause in 3SAT
    - K of them
    - If literal xi appears in clause cj (p. 316, Fig. 7.51)
      - Add edge from left vertex of j'th pair in center of diamond for xi to vertex for cj
      - Add edge from cj to right vertex of j'th pair
    - If literal ~xi appears in clause cj, add edges in opposite

- If 3SAT is satisfiable, then Hamiltonian path from s to t
  - Starts at top, go left if x1 is true, right if false (Fig. 7.53)
  - Go across center, then down to top of next diamond
  - Repeat
  - Exception: for each clause cj pick one satisfying literal
    - Follow the breakout from the appropriate center row
  - Result: all vertices touched exactly once
- If HAMPATH exists in graph
  - If "normal": top-down and thru center, with bypass, then can read out satisfying assignment
  - Fig. 7.54 (p. 318) cannot occur

- (p. 319) **UHAMPATH** – HAMPATH with undirected edges

- (p. 319) **SUBSET-SUM** S = {(S,t)| S = {x1, …) and for some subset Q={q1,… } a subset of S, sum of y’s = t}
  - 3SAT of l variables and k clauses reduces to a Subset-Sum problem with
    - 2l members of S = {y1,…yl,z1,…zl}
      - yi and zi for variable xi
    - 2k members of Q = {g1,…gk,h1…,hk}
    - and t=a # described below
  - Create table as on p. 321
    - Each row of l+k #s:
      - l columns: 1 for each variable
      - and k more columns: 1 for each clause
    - Total of 2l + 2k + 1 rows:
      - 2l of them: variable xi has 2 rows, labelled yi and zi
        - For row yi: all 0’s but a 1 in column for xi and a 1 in each clause column having xi as a literal
        - For row zi: all 0’s but a 1 in column for xi and a 1 in each clause column having ~xi as a literal
      - 2k of them: 2 for each clause, labelled gi and hi
          - Row is all 0s but a single 1 in column for clause i
          - One row for t: All 1s for variable columns; all 3s for clause columns

- Treat each row as digits of a number
- Assume wff is satisfiable, show subset
  - select Q as follows
    - If xi assigned true, select yi for Q
    - If xi assigned false, select zi for Q
  - Add up the selected rows
    - Exactly 1 for each of 1^{st} l digits
    - Each of last k digits between 1 and 3
  - To make last k digits all 3
    - Select enough gs and hs to add up to 3
- Assume subset exists, show assignment
  - All digits in each # is either 0 or 1
  - Each column in table has at most 5 1's
    - At most 3 from literals in clause
    - 2 from gs' and hs'
  - Thus no carries possible
  - Thus for a 1 in each of first l columns, exactly 1 of ys' and zs' must be selected
  - This is assignment

- Summary: from https://people.eecs.berkeley.edu/~vazirani/algorithms/chap8.pdf

Hard problems (NP-complete)	Easy problems (in P)
3SAT	2SAT, HORN SAT
TRAVELING SALESMAN PROBLEM	MINIMUM SPANNING TREE
LONGEST PATH	SHORTEST PATH
3D MATCHING	BIPARTITE MATCHING
KNAPSACK	UNARY KNAPSACK
INDEPENDENT SET	INDEPENDENT SET on trees
INTEGER LINEAR PROGRAMMING	LINEAR PROGRAMMING
RUDRATA PATH	EULER PATH
BALANCED CUT	MINIMUM CUT

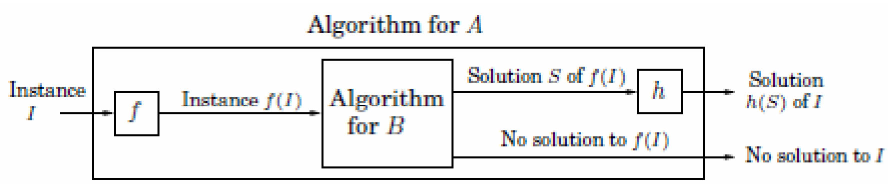
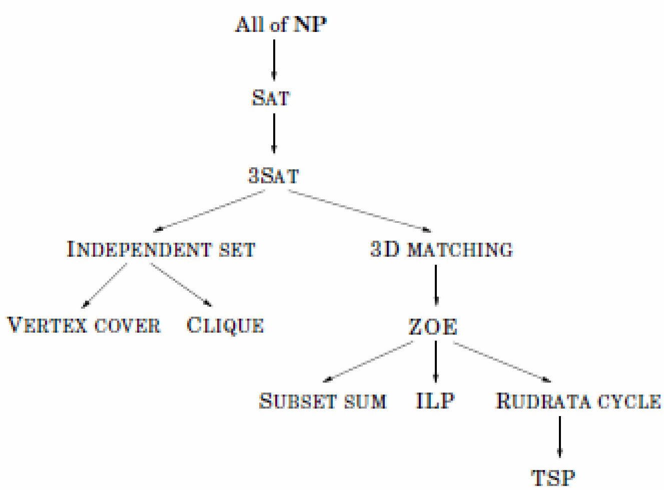


Figure 8.7 Reductions between search problems.



From https://en.wikipedia.org/wiki/NP-completeness