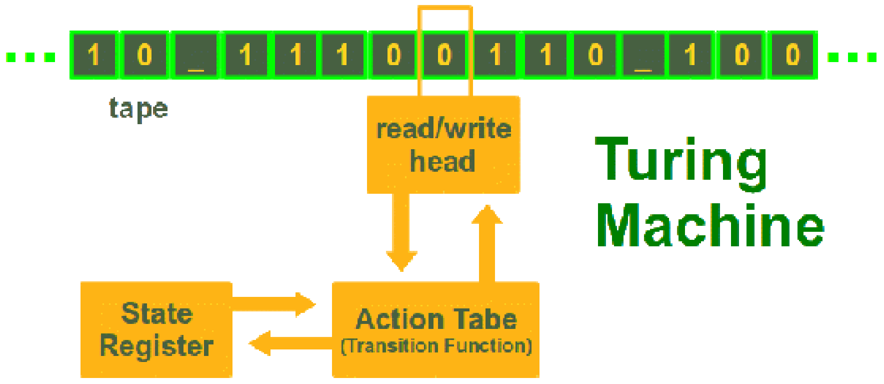
TURING MACHINES

Explanation (from Wikipedia)

A Turing machine consists of:



1. A **tape** divided into cells, one next to the other. Each cell contains a symbol from some finite alphabet. The alphabet contains a special blank symbol and one or more other symbols. The tape is assumed to be arbitrarily extendable to the left and to the right, i.e., the Turing machine is always supplied with as much tape as it needs for its computation. Cells that have not been written before are assumed to be filled with the blank symbol. In some models the tape has a left end marked with a special symbol; the tape extends or is indefinitely extensible to the right.
2. A **head** that can read and write symbols on the tape and move the tape left and right one (and only one) cell at a time. In some models the head moves and the tape is stationary.
3. A **state register** that stores the state of the Turing machine, one of finitely many. Among these is the special start state with which the state register is initialized. These states, writes Turing, replace the "state of mind" a person performing computations would ordinarily be in.
4. A finite **table of instructions** that, given the state(q_i) the machine is currently in and the symbol(a_j) it is reading on the tape (symbol currently under the head), tells the machine to do the following in sequence:
   - Either erase or write a symbol (replacing a_j with a_{j1}),
   - Move the head ('L' for one step left or 'R' for one step right or 'N' to stay in the same place),
   - Assume the same or a new state as prescribed (go to state q_{i1}).

Note that every part of the machine (i.e. its state, symbol-collections, and used tape at any given time) and its actions (such as printing, erasing and tape motion) is finite, discrete and distinguishable; it is the unlimited amount of tape and runtime that gives it an unbounded amount of storage space

Formal Definitions

- A **Deterministic Turing Machine (DTM)** is a tuple M=(Q, q_0,F,T,I, b, δ) s.t.
  - Q is a finite non-empty set of states
  - q_0 is the initial state; $q_0 \in Q$
  - F is a set of final (or accepting) states; $F \subseteq Q$
  - T is a finite non-empty set of tape alphabet symbols
  - b is the blank symbol: the only symbol which can occur on the tape infinitely; $b \in T$
  - I is the set of input symbols; $I \subseteq T-\{b\}$
  - δ is the next-move (transition) function: $(Q-F)\times T \rightarrow Q\times T\times\{L,R,N\}$
    $\delta(q_i,a_i) = (q_j,a_j,m)$ means that
    If the head is over a_i while the TM is in state q_i, then
    - the head will replace a_i by a_j
    - the head will move in the direction indicated by m (L= left, R= right, N=no move)
    - the TM will now be in state q_j

    Note: not all $\delta(q_i,a_i)$ are defined
- A **Non-Deterministic Turing Machine (NTM)** is such that.
  - Q, q_0, F, T, I, b are defined exactly as for DTMs
  - Instead of $\delta(q_i,a_i)$ yielding 0 or 1 element of $Q\times T\times\{L,R,N\}$
    $\delta(q_i,a_i) \subseteq Q\times T\times\{L,R,N\}$
- Variations:
  - TMs may have multiple tapes
  - TMs may have a single final state
  - TMs may allow only L and R movements
- An string of input symbols is **accepted** by a Turing Machine iff the TM starts in q_0 with the tape head at the left and makes a sequence of moves which makes it enter one of the accepting states.
- The **language L(M) accepted** by a Turing Machine M is the set of strings of input symbols accepted by that Turing machine.

Properties

- All DTMs are NTMs
- It is possible to turn any NTM into a DTM: i.e. any language accepted by an NTM is accepted by some DTM
- Church-Turing Thesis: anything that can be "computed" can be computed on a Turing Machine

NTM EXAMPLE

Partition Problem:

- Accept strings of the form $10^{i1}10^{i2}\ldots10^{ik}$ such that
- There is some set $I \subseteq \{1, 2, \ldots, k\}$ for which
- $\sum_{j\in I} i_j = \sum_{j\notin I} i_j$

NTM

- NTM has 3 tapes:
  - Tape1 contains input string S
  - Tapes 2 and 3 are output tapes
- NTM scans tape1:
  - After reading a '1' it picks tape 2 or 3 (call is TAPE)
  - After reading a '0' it stores it in TAPE
  - At the end of tape1, it counts the # of 0s in tapes 2 and 3 and accepts S if the same number of 0s is in both tapes.
- Formally:
  $M=(\{q_0,..,q_5\}, q_0,\{q_5\},\{0,1,B,\$\},\{0,1\}, B, \delta)$

  δ is defined as:

	Current Symbol			(New Symbol, Head move)			New State	Explanation
State	Tape1	Tape2	Tape3	Tape1	Tape2	Tape3		
q_0	1	B	B	1,N	\$,R	\$R	q_1	Mark end of tapes 2 and 3
q_1	1	B	B	1,R	B,N	B,N	q_2	Random state
	1	B	B	1,R	B,N	B,N	q_3	
q_2	0	B	B	0,R	0,R	B,N	q_2	Write 0s on tape 2
	1	B	B	1,R	B,N	B,N	q_1	
	B	B	B	B,N	B,L	B,L	q_4	
q_3	0	B	B	0,R	B,N	0,R	q_3	Write 0s on tape 3
	1	B	B	1,R	B,N	B,N	q_1	
	B	B	B	B,N	B,L	B,L	q_4	
q_4	B	0	0	B,N	0,L	0,L	q_4	Match 0s in tapes2,3
	B	\$	\$	B,N	\$,N	\$,N	q_5	
q_5								Accept

COMPLEXITIES

Note: These definitions are simplified and not completely accurate.

Algorithms

- An algorithm A is said to be of **polynomial** time if there is a polynomial p(n) s.t. Cost(A) ∈ O(p)
- An algorithm is said to take **superpolynomial** time if there is **no** polynomial p(n) s.t. Cost(A)∈ O(p)

Turing Machines

- A DTM M is said to be of **time complexity** T(n) if every accepted input string of length n is accepted in at most T(n) moves.
- A NTM M is said to be of **time complexity** T(n) if for every accepted input string of length n there is a sequence of at most T(n) moves leading to an accepting state.
- A DTM M is said to be of **space complexity** S(n) if at most S(n) different cells are scanned on any tape to accept any input string of length n.
- A NTM M is said to be of **space complexity** S(n) if for every accepted input string of length n there is a sequence of moves leading to an accepting state in which at most S(n) different cells are scanned on any tape.

Decision Problems

- A **decision problem** is a problem in a **formal system** (system of abstract thought based on the model of mathematics) with a yes-no answer

  Examples:
  - Is this graph connected?
  - Is this number prime?
  - Is this polynomial nonnegative?
  - Is this optimization problem feasible?

P AND NP CLASSES

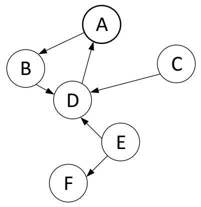
P Class

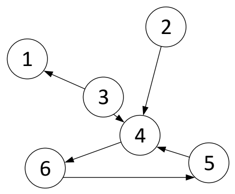
- P = the complexity class of decision problems which can be **solved in polynomial time** (i.e with a polynomial algorithm)
- Examples:
  - Connectedness: Is the graph G connected?
  - Element uniqueness: Does an array have duplicate elements?
  - Primality: is a number prime? (AKS primality testing algorithms developed in 2002 is $O(\log^6 n)$)
- P = the complexity class of decision problems that can be **solved on a DTM** of polynomial complexity

  = {L | ∃ a DTM M and a polynomial p(n) s.t. M is of time complexity p(n) and L=L(M)}

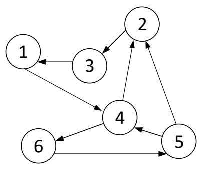
NP Class

- NP = the complexity class of decision problems which can be **verified in polynomial time** (i.e a solution can be verified with a polynomial algorithm)
- Examples:
  - Is there a subset of {-8, -5, 0, 1, 2, 6, 10} s.t. the sum of the elements will be 0?
  - Are these two graphs isomorphic:





  - Does this graph have a Hamiltonian circuit:



- NP = the complexity class of decision problems that can be **solved on a NTM** of polynomial complexity

  = {L | ∃ an NTM M and a polynomial p(n) s.t. M is of time complexity p(n) and L=L(M)}

Properties:

- $P \subseteq NP$
- **Open Problem**: Is $P = NP$ or is $P \subset NP$?

POLYNOMIAL TRANSFORMATIONS

Definition

- A language L is **polynomially transformable** to a language L_0 if there is a polynomial DTM M which will convert every string w in the alphabet of L into a string w_0 in the alphabet of L_0 s.t. $w \in L$ if and only if $w_0 \in L_0$

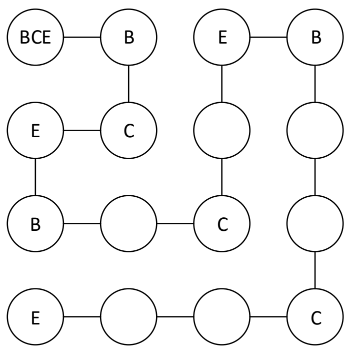
Example

- Languages:
  - $L = \{0^n \text{ for } n \in \mathbb{N}\} = \{0,00,000,0000, \text{etc.}\}$
  - $L_0 = \{0^{n^2} \text{ for } n \in \mathbb{N}\} = \{0,0000, 000000000, 0000000000000000, \text{etc.}\}$
- Deterministic Turing Machine:

3 tapes:	Algorithm:
T1=input string w	Copy w from T1 to T2
T2=scratchpad	For each element of T2, copy w
T3=output string w_0	(from T1) onto T3

State	Current Symbol T1	T2	T3	(NewSymbol, move) T1	T2	T3	New State	Explanation
q_0	0	B	B	0,R	0,R	B,N	q_0	Copy T1 to T2
	B	B	B	B,L	B,L	B,N	q_1	
q_1	0	0	B	0,L	0,N	0,R	q_1	Copy T1 to T3 R→L
	B	0	B	B,R	B,L	B,N	q_2	
	0	B	B	0,N	B,N	B,N	accept	
q_2	0	0	B	0,R	0,N	0,R	q_2	Copy T1 to T3 L→R
	B	0	B	B,L	B,L	B,N	q_1	
	0	B	B	0,N	B,N	B,N	accept	

- This TM is of time complexity $O(n^2)$
- Therefore L is polynomially transformable to L_0
- Note that it is harder to prove that L_0 is polynomially transformable to L →



Theorem:

- If L is polynomially transformable to L_0 and $L_0 \in P$ then $L \in P$

PROBLEM REDUCTION REVISITED

Definitions

- Problem X is **reducible** to problem Y if an algorithm for solving Y efficiently (if it existed) could be used as a subroutine to solve problem X efficiently
- If that is the case, then solving X cannot be harder than solving Y (as long as the rest of the algorithm for X is at least as efficient as the Y algorithm). So a lower bound on the cost of X becomes a lower bound on the cost of Y.
- If problem X is reducible to problem Y, the algorithm that solves X using the subroutine that solves Y is called a **Turing reduction** from X to Y
- A problem X is **polynomially reducible** to a problem Y (notation $\mathbf{X \leq_p Y}$) if there is a Turing reduction from X to Y which
  – runs in polynomial time excluding the time spent in the Y algorithm.
  – has a polynomial number of calls to the Y algorithm.

Example

- The algorithm we used to solve the element uniqueness problem Q using a solution to the minimum spanning tree problem P was a Turing reduction from Q to P.
- Q runs in $\theta(n)$ time excluding the time spent in P
- Q calls P once
- Therefore the element uniqueness problem is polynomially reducible to the minimum spanning tree problem.

Theorem

- If X is polynomially reducible to Y and $Y \in P$ then $X \in P$

NP-COMPLETENESS

- A problem $Y \in NP$ is **nondeterministic polynomial-time complete** (**NP-complete**) if **all** problems in NP can be polynomially reduced to Y

Satisfiability Problem

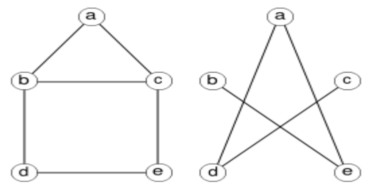
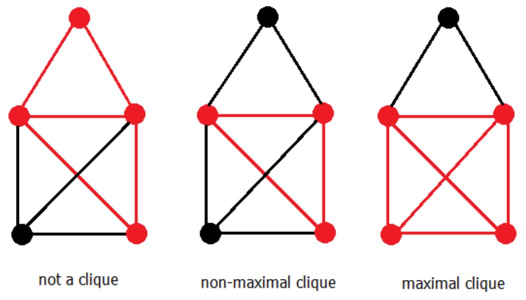
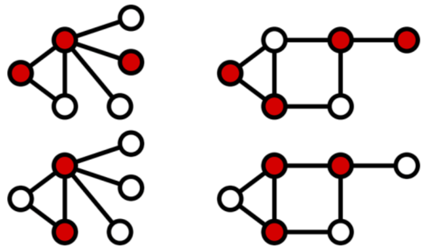
- Can a boolean formula evaluate to true? (i.e. is satisfiable)
- Examples:
  – "x and not x" is not satisfiable because it is false for all boolean value of x
  – "x and not y" is satisfiable because it is true when x=true and y=false
- Cook's Theorem: The satisfiability problem is NP-complete

Theorem

- If X∈NP is NP-complete and X is polynomially reducible to Y∈NP then Y is NP-Complete

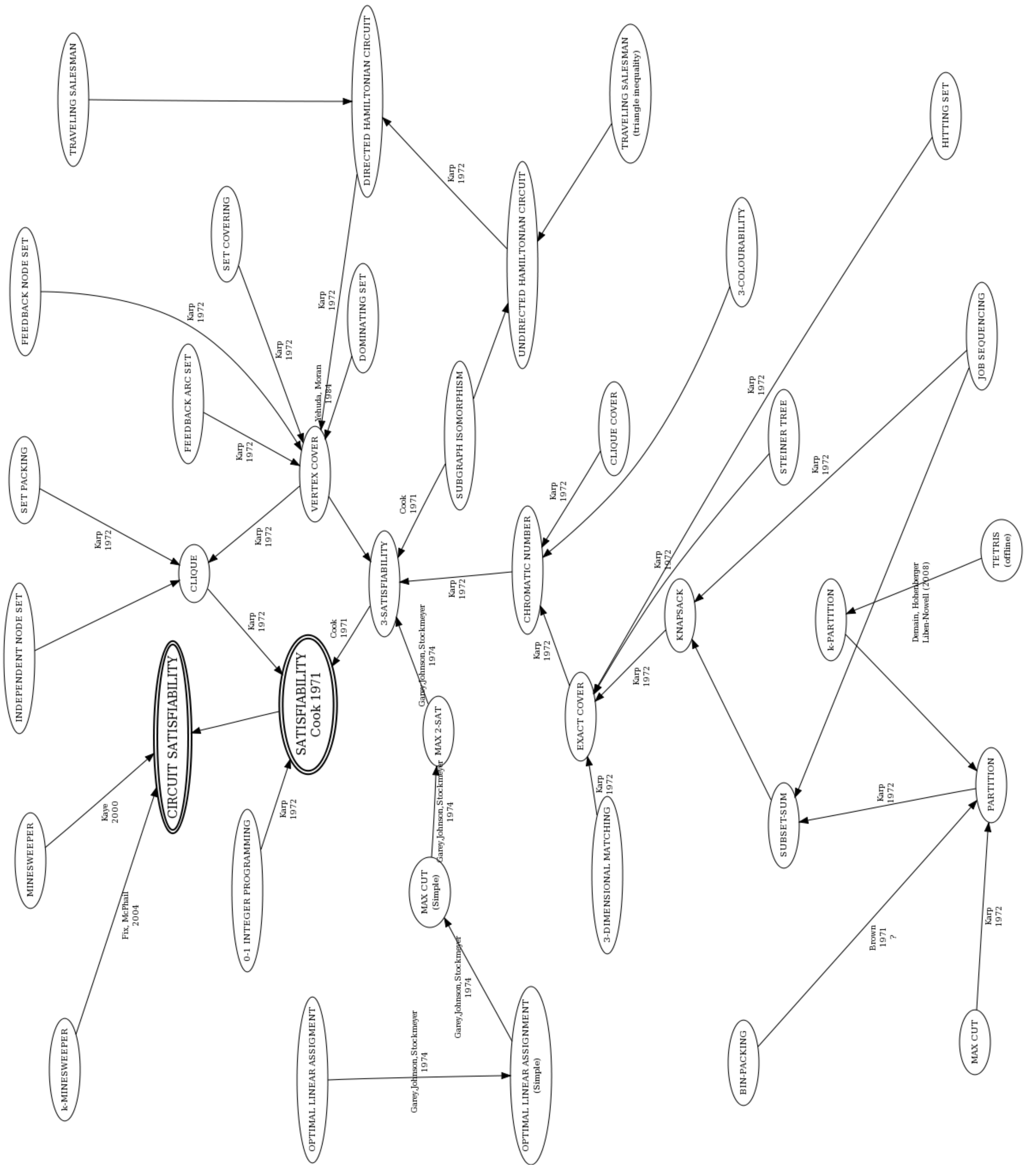
Other NP-complete Problems:

- Let G =(V, E) be an undirected graph

• The complement graph $G^C = (V,E^C)$ where E^C = {possible edges not in G}	
• a k-clique of G is a complete subgraph of G with k vertices (every pair of vertices is connected by an edge)	
• A vertex cover of G is a subset S⊆V s.t. each edge of G is incident upon some vertex in S	

- The clique problem: "does an undirected graph have a k-clique?" is NP-complete
- Is the vertex cover problem: "does an undirected graph have a vertex cover of size k?" NP-complete?
- The clique problem is polynomially reducible to the vertex cover problem
  - S is a clique of G iff V-S is a vertex cover of G^C:
  - S is a clique in G,
    - iff Every pair of vertices in S must be in G
    - iff no edge in G^C connects two vertices in S
    - iff every edge in G^C is incident upon some vertex in V-S
    - iff V-S is a vertex cover of G^C
- Therefore the vertex cover problem is NP-complete

Some proved NP-complete problems


TRAVELING SALESMAN
DIRECTED HAMILTONIAN CIRCUIT
TRAVELING SALESMAN (triangle inequality)
HITTING SET
UNDIRECTED HAMILTONIAN CIRCUIT
Karp 1972
SET COVERING
FEEDBACK NODE SET
DOMINATING SET
3-COLOURABILITY
JOB SEQUENCING
FEEDBACK ARC SET
Yehuda, Moran 1984
SUBGRAPH ISOMORPHISM
CLIQUE COVER
STEINER TREE
SET PACKING
VERTEX COVER
Cook 1971
CHROMATIC NUMBER
TETRIS (offline)
CLIQUE
3-SATISFIABILITY
KNAPSACK
k-PARTITION
Demain, Hohenberger Liben-Nowell (2008)
INDEPENDENT NODE SET
CIRCUIT SATISFIABILITY
SATISFIABILITY Cook 1971
Garey,Johnson,Stockmeyer 1974
MAX 2-SAT
EXACT COVER
MINESWEEPER
Kaye 2000
0-1 INTEGER PROGRAMMING
MAX CUT (Simple)
3-DIMENSIONAL MATCHING
SUBSET-SUM
PARTITION
Fix, McPhail 2004
Brown 1971 ?
k-MINESWEEPER
OPTIMAL LINEAR ASSIGMENT
OPTIMAL LINEAR ASSIGNMENT (Simple)
BIN-PACKING
MAX CUT