CSE 5311 Notes 18: NP-Completeness

(Last updated 7/21/13 8:23 PM)

Elementary Concepts

Satisfiability: $(p \vee q) \wedge (p \vee \overline{q}) \wedge (\overline{p} \vee q) \wedge (\overline{p} \vee \overline{q})$

Is there an assignment? (Decision Problem)

Similar to debugging a logic circuit - Is there an input case that turns on the output LED?

Aside: Evaluating one input setting for a circuit is *P-complete* $\Rightarrow$ hard to massively parallelize.

NP-complete means (informally):

1. The problem may be computed ("decided") in nondeterministic polynomial time.

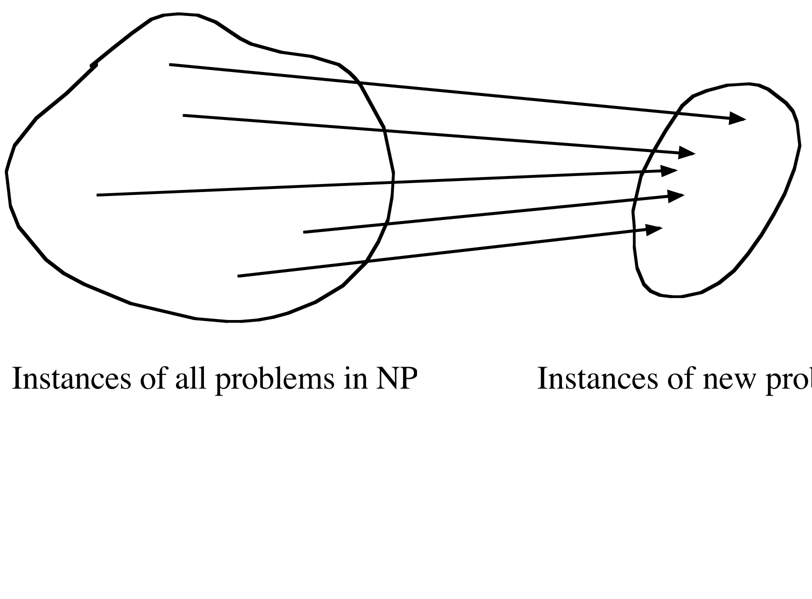
   a. Guess a solution (polynomial time - easy to get)

   b. Check the solution in polynomial time (deterministic).

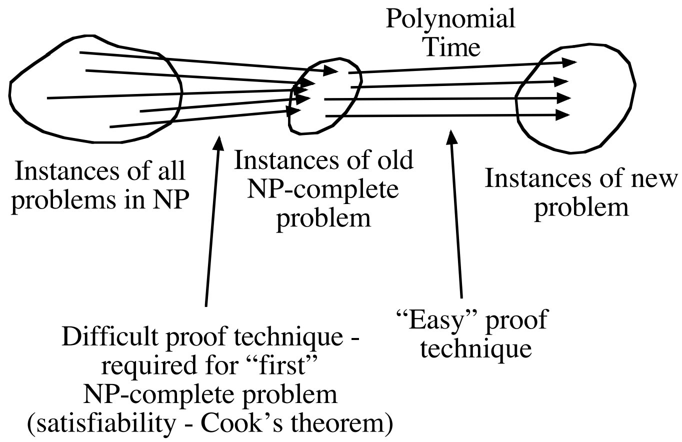
   Checking ("verification") is easier than computing.

2. All problems in NP may be transformed ("reduced") to this problem in polynomial time.

   If an instance of one problem is transformed to an instance of another, the new problem has a solution iff the old problem has a solution.



Showing that all NP problems reduce to new problem is unnecessary. Instead, find another NP-complete problem:



Property 1 without property 2 - problem is just in NP. Example: Is a table sorted?

Property 2 without property 1 - problem is said to be “NP-hard” (at least as difficult as all other problems in NP). Note: property 1 is usually trivial to establish and is often omitted in proofs.

Significance of a problem being NP-complete

No polynomial-time algorithm is known for any NP-complete problem. (Only exponential time)

If a polynomial-time algorithm is known for one NP-complete problem, then there is a polynomial-time algorithm for every NP-complete problem.

Exponential lower bound has never been shown.

If difficult instances of an NP-complete problem arise in practice, then approximation schemes with bounds on the quality of the solution are needed.

What about playing chess?

Example Problems:

Satisfiability

Graph (Vertex) Coloring

Job Scheduling with Penalties - durations, deadlines, penalties (single processor)

Bin Packing - how many fixed-sized bins are needed to hold variable-sized objects?

Knapsack - how many objects with different profits and sizes should go into a knapsack?

Subset Sums - is there a subset whose sum is a particular value?

Hamiltonian Path - does a graph (or digraph) have a path including each vertex exactly once?

Hamiltonian Circuit - is there a cycle including each vertex exactly once?

Traveling Salesperson - minimize distance for Hamiltonian circuit

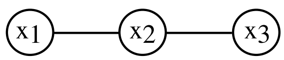
Steiner subgraph - is there a connected subgraph (tree) that includes designated *terminal* vertices and whose total weight does not exceed a given value? (Euclidean version is NP-hard)

REDUCTIONS

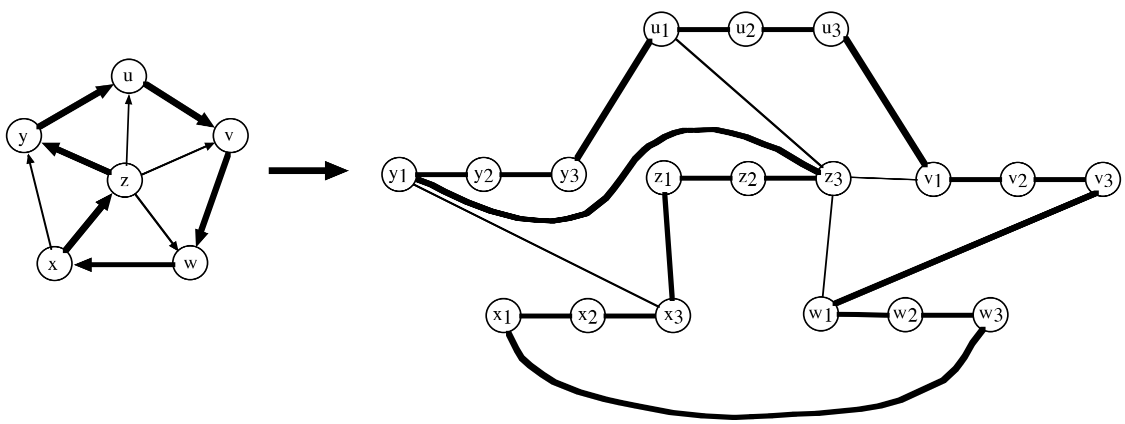
Important resource - M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, 1979.

Suppose you know *directed* Hamiltonian circuit is NP-complete. Show that *undirected* Hamiltonian circuit is NP-complete:

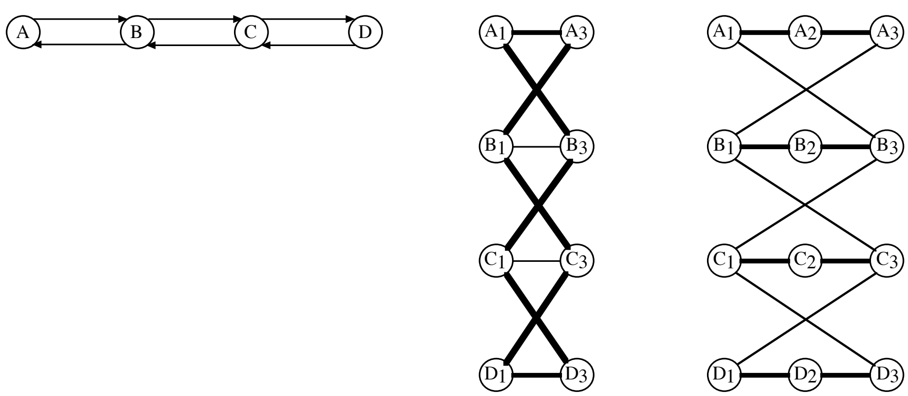
1. Replace each vertex x by three vertices x_1, x_2, x_3 connected as:



2. Include an edge $\{u_3, v_1\}$ for each edge (u, v) in the directed graph.



Leaving out x_2 will not work - allows going in wrong direction:



Show 3-satisfiability is NP-complete by reduction from conjunctive normal form satisfiability.

In CNF an expression is a conjunction of several *clauses* (disjunctions).

Each clause has several *literals* which may be asserted or negated.

The reduction is based on replacing each clause with $k > 3$ literals by k - 2 clauses for 3-satisfiability and introducing k - 3 <u>new</u> variables:

$A \vee B \vee C \vee D \vee E \vee F \vee G$
$\overline{A}, \overline{B}, \overline{C}, \overline{D}, \overline{E}, \overline{F}, \overline{G}$

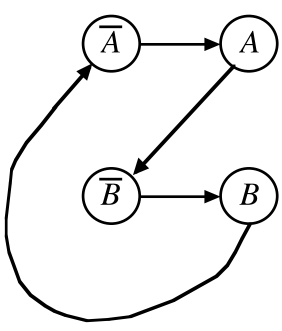
$A \vee B \vee X_1$
$\overline{X}_1 \vee C \vee X_2$
$\overline{X}_2 \vee D \vee X_3$
$\overline{X}_3 \vee E \vee X_4$
$\overline{X}_4 \vee F \vee G$
$\overline{A}, \overline{B}, \overline{C}, \overline{D}, \overline{E}, \overline{F}, \overline{G}$

Note, however, that 2-satisfiability is in P. Convert to a graph problem by replacing each $P \vee Q$ by $\overline{P} \longrightarrow Q$ and $\overline{Q} \longrightarrow P$ based on $A \rightarrow B \equiv \overline{A} \vee B$.

If there is a path from $\overline{X}$ to X, then X is true. If there is a path from X to $\overline{X}$, then X is false.

If X and $\overline{X}$ are in a cycle, then the expression is unsatisfiable.

Consider A, B, $\overline{A} \vee \overline{B}$:



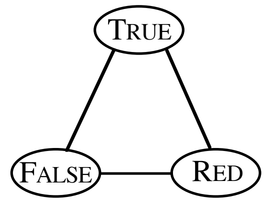
Show that graph 3-colorability is NP-complete by a reduction from 3-sat.

This reduction is fairly difficult. Others are much worse.

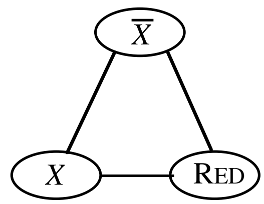
Conceptually, we will call the 3 colors TRUE, FALSE, and RED.

Since coloring is usually viewed as assigning the numbers 0, 1, 2 to the vertices, for any successful coloring there are five renamings based on permutations.

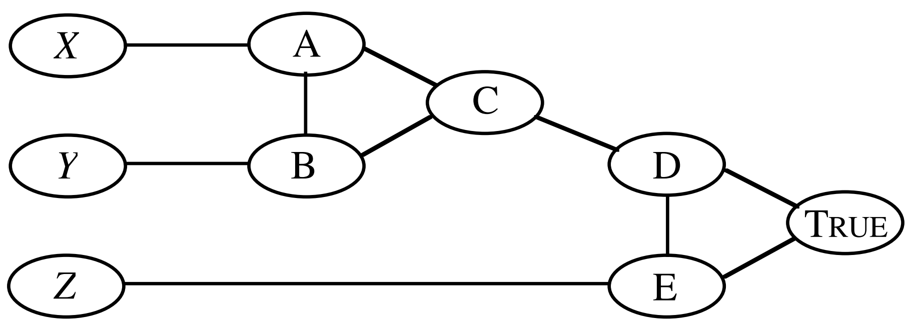
The reduction starts with a triangle to establish which number has which color:

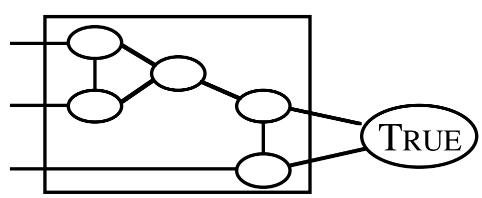


For each variable *X*, another triangle is needed to constrain the value:



For each clause $X \vee Y \vee Z$ the following pattern is used. At least one of *X*, *Y*, and Z is forced to be true.





Widget

Observe:

1. *X*, *Y*, and *Z* must have the same color as TRUE or FALSE.

2. One of D and E has the same color as FALSE, the other the same color as RED.

3. If E has the same color as FALSE, then Z has the same color as TRUE.

4. If E has the same color as RED, then D has the same color as FALSE.

Summary:

A	B	C	D	E	X	Y	Z
TRUE	RED	???	FALSE	RED	FALSE	FALSE	FALSE
TRUE	RED	FALSE	RED	FALSE	FALSE	FALSE	TRUE
TRUE	FALSE	RED	FALSE	RED	FALSE	TRUE	FALSE
TRUE	FALSE	RED	FALSE	RED	FALSE	TRUE	TRUE
FALSE	TRUE	RED	FALSE	RED	TRUE	FALSE	FALSE
FALSE	TRUE	RED	FALSE	RED	TRUE	FALSE	TRUE
FALSE	RED	TRUE	FALSE	RED	TRUE	TRUE	FALSE
FALSE	RED	TRUE	FALSE	RED	TRUE	TRUE	TRUE

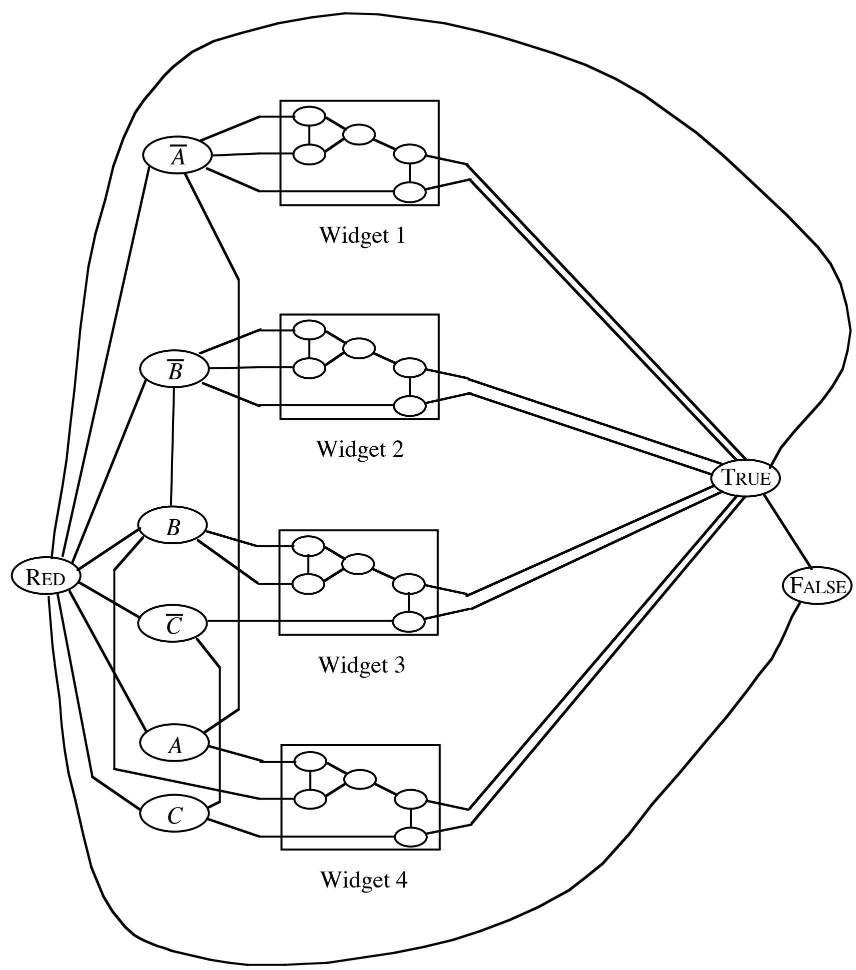
Unsatisfiable instance - graph will require 4 colors

$\overline{A}$

$\overline{B}$

$B \vee \overline{C}$

$A \vee B \vee C$



If $\overline{A}$ is removed, 3 coloring is possible.

k-clique - complete subgraph with *k* vertices

Show *k*-clique is NP-complete by a reduction from 3-sat.

Each literal becomes a vertex.

Connect each vertex to the vertices for all other clauses, except for X — $\overline{X}$

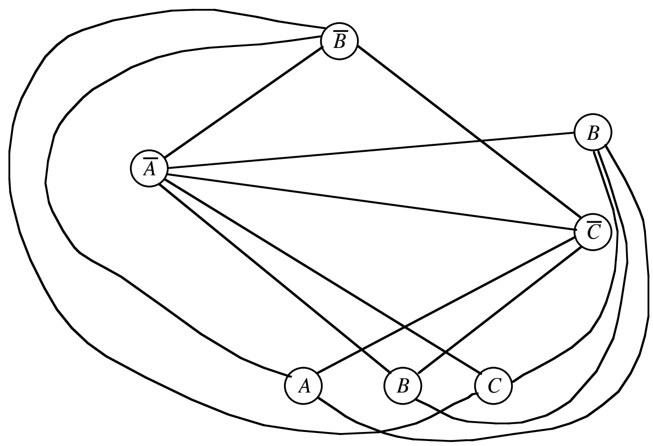
Is there a clique with one vertex per clause (i.e. *k* is the number of clauses)?

$\overline{A}$

$\overline{B}$

$B \vee \overline{C}$

$A \vee B \vee C$



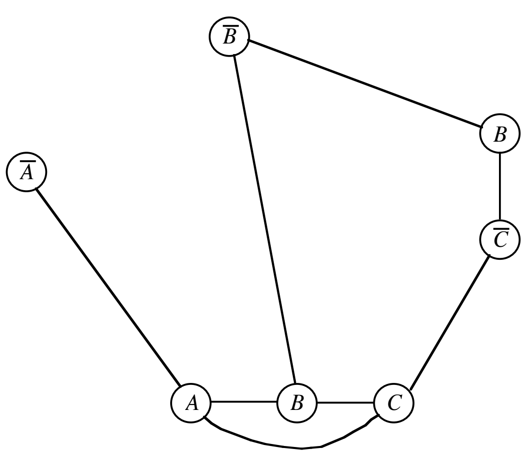
Vertex Cover = set of vertices such that every edge has at least one incident vertex in cover.

Is there a vertex cover with no more than *p* vertices?

Reduce from *k*-clique:

1. Take complement of graph. (Edge is in complement iff edge is not in graph.)
2. Is there a |V| - *k* cover? (Choose vertices not in the *k*-clique.)

From *k*-clique example - Is there a 3-cover?

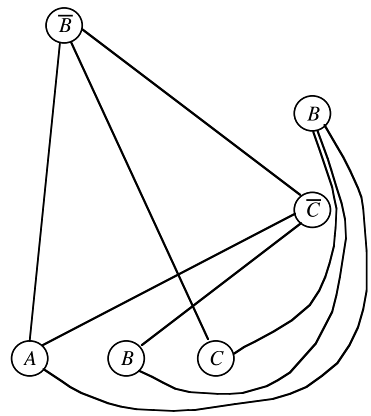


Consider clique when first clause is removed.

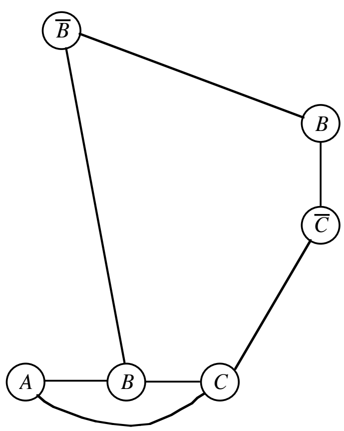
$\overline{B}$

$B \vee \overline{C}$

$A \vee B \vee C$



3-vertex cover



Show Steiner subgraph is NP-complete by a reduction from 3-sat.

Steiner vertex for each possible literal on n propositions.

Terminal vertex for each of m clauses, u, and v.

Unit-weight edges in subgraph with u, v, and literal vertices (for choosing assignment).

Edges with weight $2n + 1$ between each clause vertex and vertices for its literals

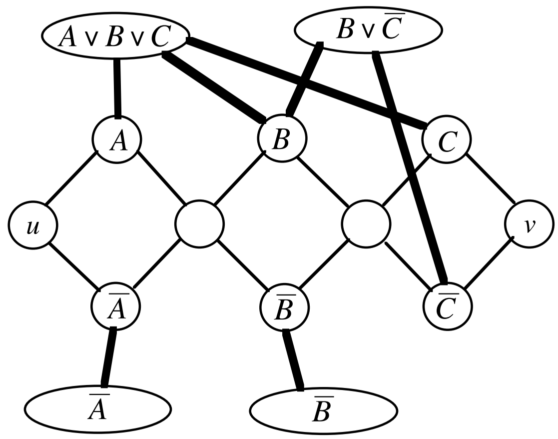
Is there a subgraph with total weight *not exceeding* $2n + m(2n + 1)$?

$\overline{A}$

$\overline{B}$

$B \vee \overline{C}$

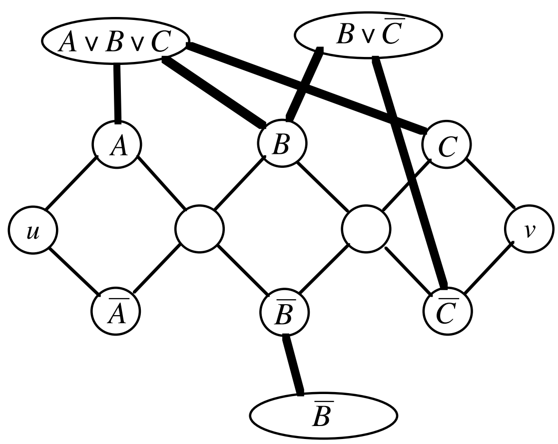
$A \vee B \vee C$



$\overline{B}$

$B \vee \overline{C}$

$A \vee B \vee C$



APPROXIMATION

Goal: Performance guarantees for optimization (NP-hard) problems corresponding to NP-complete problems.

1. How fast?

2. Approach:

   Greedy
   Online
   Preprocessing (e.g. MST, DFS)
   Randomization
   Restricted cases (e.g. spare or dense graphs)
   (Parallelism)

3. Quality of solution

   max ratio $= \dfrac{Optimal}{Solution} \geq 1$ (e.g. knapsack)

   min ratio $= \dfrac{Solution}{Optimal} \geq 1$ (e.g. TSP)

4. Generality

Approximation Algorithm - achieve max/min ratio in $O\left(n^k\right)$ time (k fixed)

Approximation Scheme - flexible ratio 1 + ε in $O(f(n,\varepsilon))$

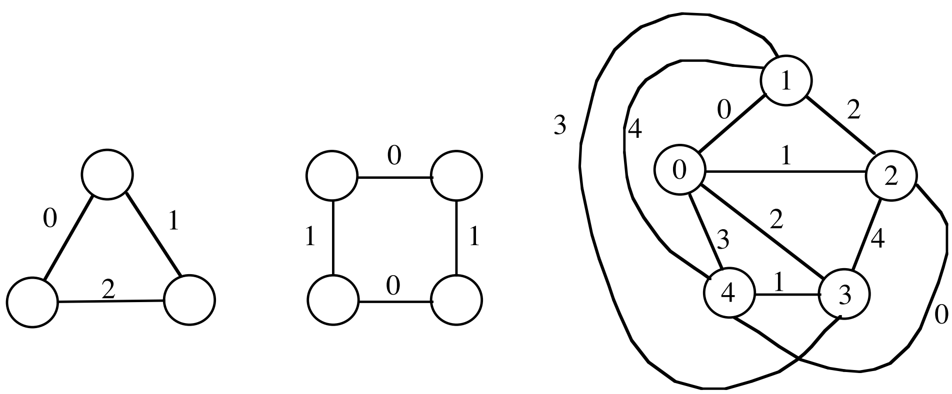
Polynomial-time Approximation Scheme - $O\left(n^{f(\varepsilon)}\right)$

Fully PTAS - $O\left(n^k\left(\frac{1}{\varepsilon}\right)^l\right)$ time

Examples are presented in ascending order of min/max ratio

Edge Coloring (`http://ranger.uta.edu/~weems/NOTES5311/misraGriesNew.c`)

An unusually optimistic situation . . .

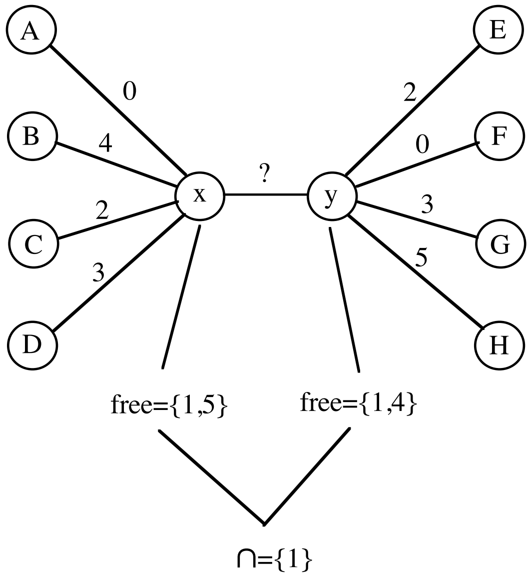


Vizing's Theorem

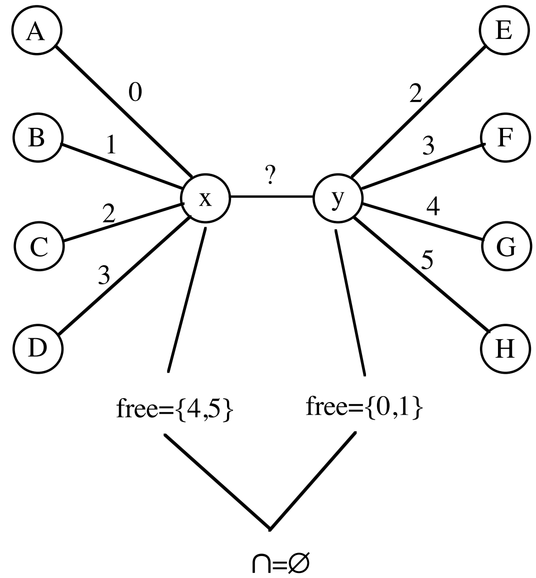
$\Delta(G) \le X'(G) \le \Delta(G)+1$ (Required number of colors is either degree ("Class 1") or degree + 1 ("Class 2"). For bipartite graphs, $\Delta(G) = X'(G)$.)

NP-complete to test if $\Delta(G) = X'(G)$, but takes only O(VE) to color with $\Delta(G)+1$ colors ($\Delta(G)$ for bipartite) in an incremental fashion. Thus:

$$\text{min ratio} \le \frac{\Delta(G)+1}{X'(G)} \le \frac{\Delta(G)+1}{\Delta(G)}$$



What if?



So, steal a color from another edge and give that edge a new color.

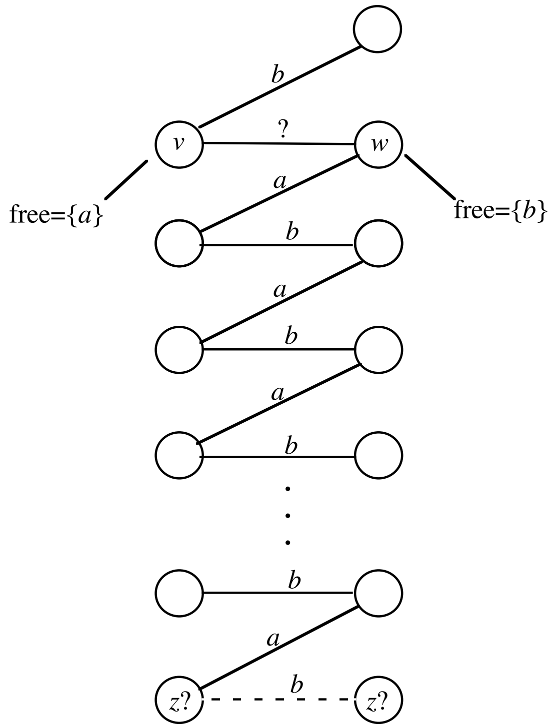
HOW?

Aside: The simpler problem of coloring a bipartite graph using Δ colors.

From p. 347 of H.N. Gabow, "Using Euler Partitions to Edge Color Bipartite Multigraphs", *Int'l J. of Computer and Information Sciences 5(4)*, 1976, 345-355:

Now we describe a method for coloring due to Vizing. Originally each edge of G is uncolored; it must be assigned one of Δ possible colors. An uncolored edge (v, w) is colored as follows. At most Δ - 1 edges incident to v are colored, so some color a is missing at v; similarly, some color b is missing at w. Construct an "alternating (a, b) path" starting at w, as follows. The path begins with the edge incident to w that is colored a (if it exists). Consecutive edges in the path are alternately colored a and b. The path ends at the vertex z where the next color is missing. It is easy to see that $z \neq$ v, w if the graph is bipartite.

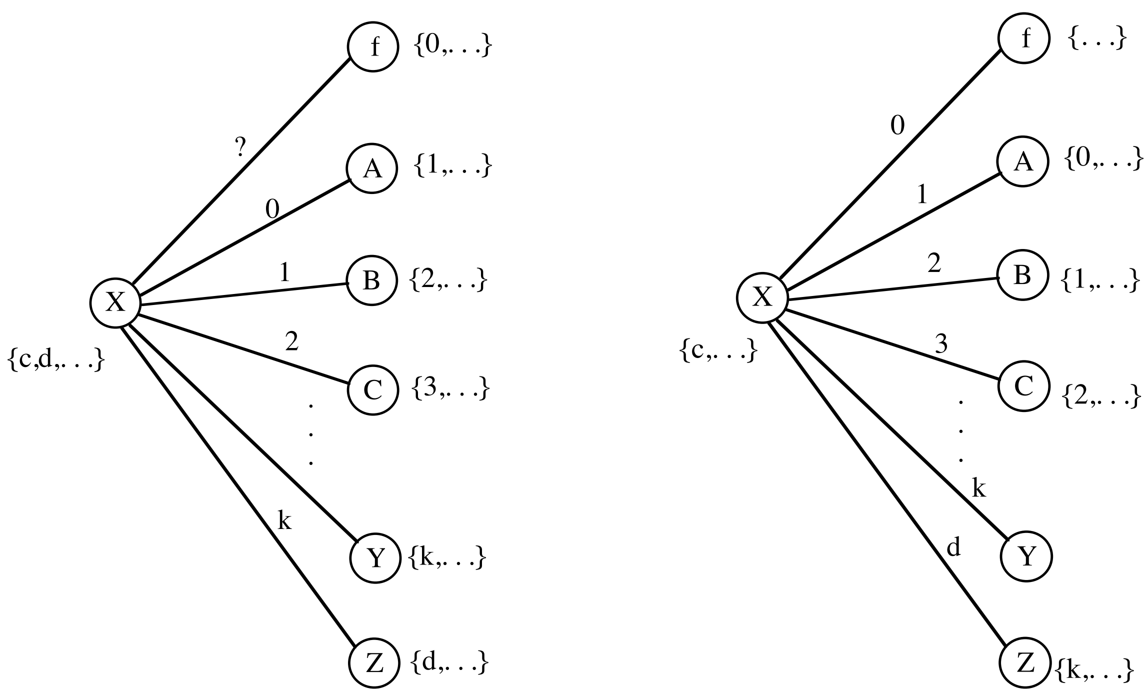
Interchange colors along the path, switching a to b and b to a. This makes color a missing at both v and w, since $z \neq v, w$. Edge (v, w) can now be colored a.



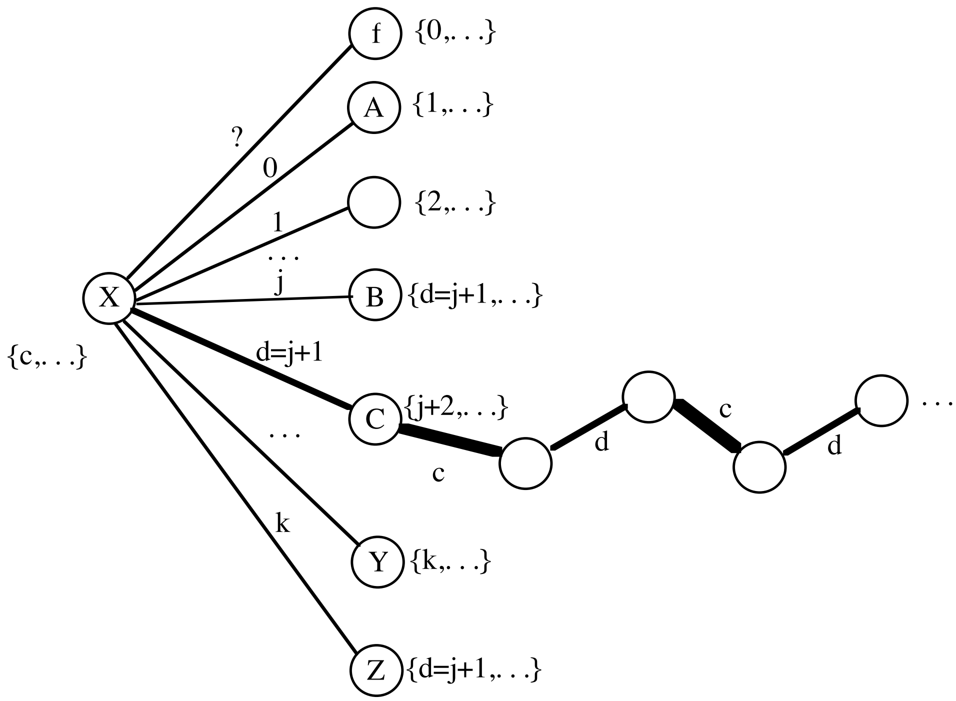
Back to non-bipartite . . .

Most general case - *maximal fan* with free(X) ∩ free(f) = ∅.

If no edge incident to X is colored with last free color d, immediately “rotate” fan:



While maximizing the fan, suppose the next color (d) is already on an earlier fan edge:

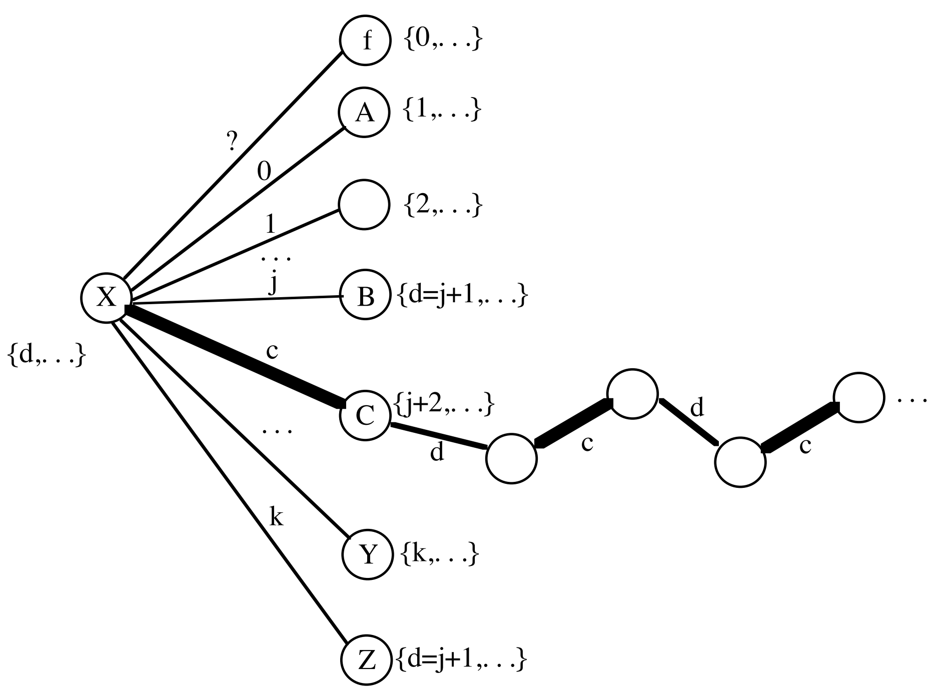


1. Find (alternating) dc-path starting with X-C.

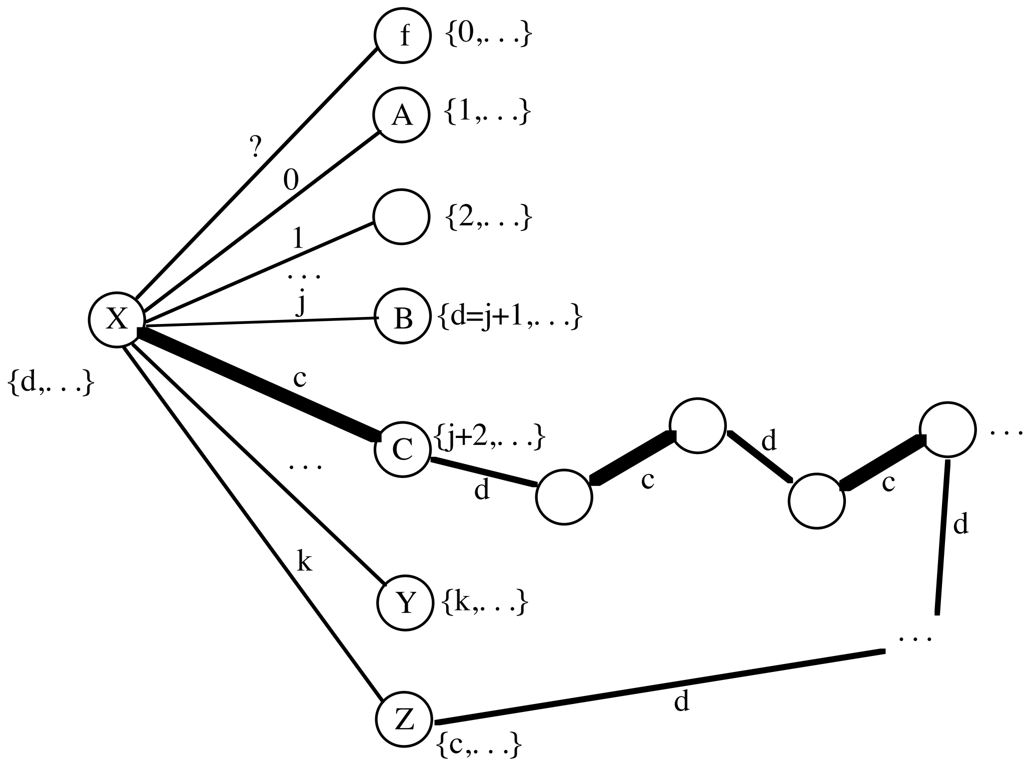
   dc-path (above) reaches no more than one of B or Z. (Why?)

2. Invert colors (c ↔ d) along entire dc-path.

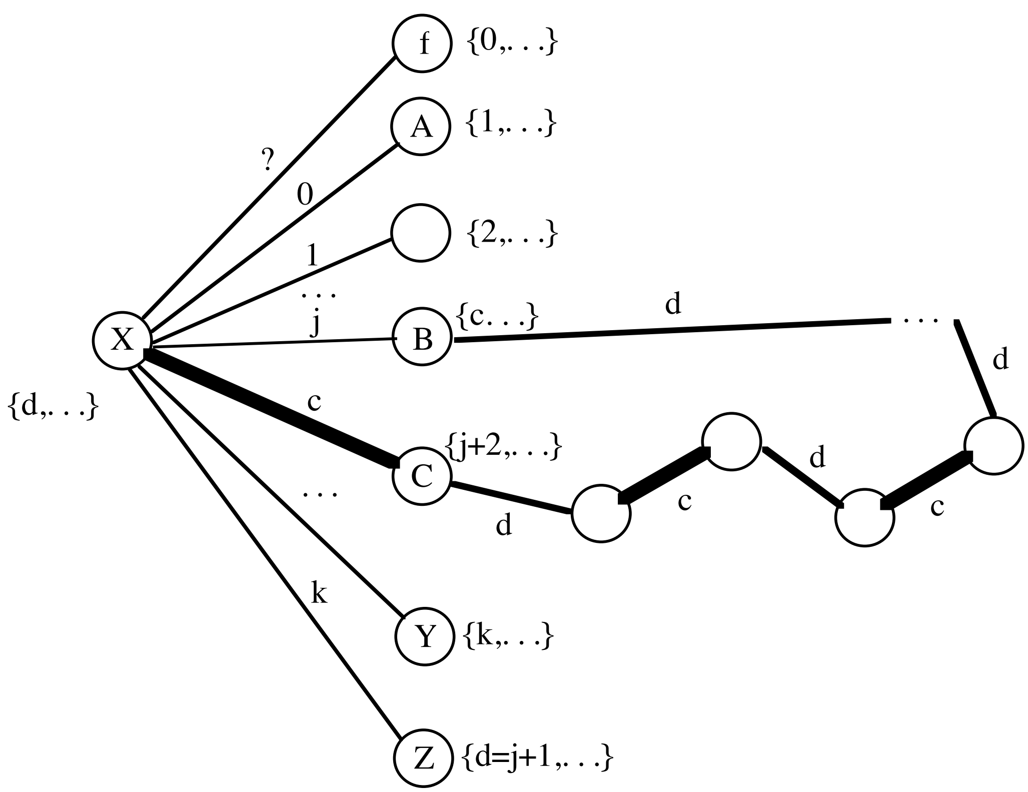
   a. Neither B or Z reached - fan stops at B.



b. Z reached - fan stops at B.



c. B reached - fan keeps all vertices.



3. Rotate fan.

Vertex Cover - Approximation Algorithm

```
VC := ∅
for each edge {u, v}   // arbitrary order
        if u ∉ VC and v ∉ VC
                VC := VC ∪ {u, v}
```

1. At termination, VC is a vertex cover.

2. Polynomial time - obvious.

3. a. VC_{OPT} must cover the set of edges processed based on the "if".

   b. VC_{OPT} must include at least one of {u, v} for each of these edges, so:

$$\frac{1}{2}|VC| \le |VC_{OPT}| \qquad \text{min ratio} \le \frac{|VC|}{|VC_{OPT}|} \le 2$$

Aggressive strategy of choosing one vertex from an uncovered edge is vulnerable to "stars".

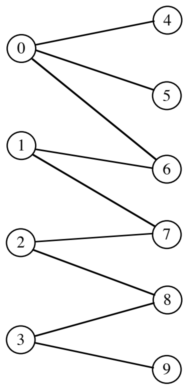
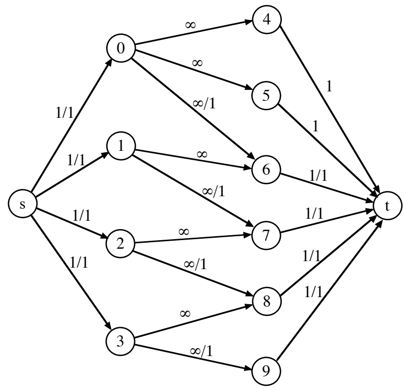
Minimum Bipartite Vertex Cover - Exact Solution (not NP-complete . . . unless P = NP)

Instance of max-flow is isomorphic to bipartite matching max-flow, except capacities from V_1 to V_2 are ∞.

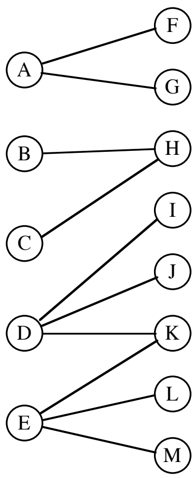
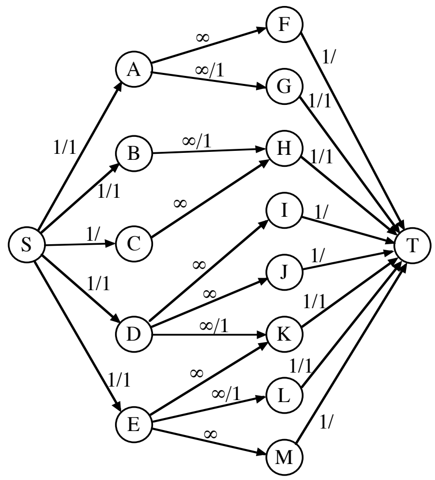
Set of edges from V_1 to V_2 with (unit) flow after max-flow is found is a maximum matching. The size of a maximum matching gives an (obvious) lower bound on the size of a minimum bipartite vertex cover. (Showing that the size of a maximum matching is also an upper bound is more involved and omitted).

Theorem: If a minimum *S*-*T* cut is known, then $(V_1 \cap T) \cup (V_2 \cap S)$ is a minimum bipartite vertex cover.

Proof: Suppose the bipartite graph has an edge $\{v_1, v_2\}$ with $v_1 \in V_1$ and $v_2 \in V_2$ and $v_1 \in S$. Since the capacity of $\{v_1, v_2\}$ is ∞, $\{v_1, v_2\}$ is an edge in the residual network and $v_2 \in S$ to prevent $\{v_1, v_2\}$ from being uncovered. ***

$S = \{s\} \quad T = \{t, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\} \quad V_1 \cap T = \{0,1,2,3\} \qquad V_2 \cap S = \varnothing$

$S = \{s, B, C, H\} \quad T = \{t, A, D, E, F, G, I, J, K, L, M\} \quad V_1 \cap T = \{A,D,E\} \qquad V_2 \cap S = \{H\}$

Bin Packing (one dimensional)

Minimize number of unit size bins to hold objects with sizes $0 < s_i \le 1$.

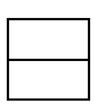
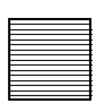
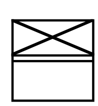
Next Fit

Online

Use one bin, then seal when next item doesn't fit.

Worst case sequence

$\frac{1}{2}, \frac{1}{2N}, \frac{1}{2}, \frac{1}{2N}, \frac{1}{2}, \frac{1}{2N}, \frac{1}{2}, \frac{1}{2N}, \cdots$ Repeated to get $4N$ elements

OPT (offline)   Next Fit  $\frac{NF}{OPT} \le 2$

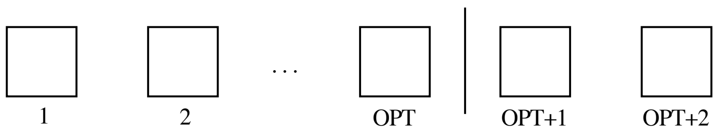
First-fit Decreasing

Sort n sizes descending

For each object, go through bins "left-to-right" to find first bin that object fits in.

Achieves $\frac{FF}{OPT} \le 1.5$ (not hard to improve ratio to 4/3, difficult to get 11/9)

Claim: Objects placed in extra bins have size ≤ 1/2



Proof: Suppose otherwise.

Claim: Number of objects in extra bins ≤ OPT(S) - 1

Proof: Suppose OPT(S) extra objects are used.

1. Waste in every OPT bin < size of every object in extra bins.

2. Consider $\sum_{i=1}^{OPT} \left(OPT_i \ sum + Extra\ Object_i \right)$

   But since each $OPT_i \ sum + Extra\ Object_i > 1$, this sum exceeds OPT, a contradiction.

Based on the two claims, the number of extra bins ≤ OPT/2 and the ratio is 1.5.

Set Cover

Input: Set S and subsets such that $S = \bigcup_{i=1}^{n} S_i$

Output: Small set of subsets covering S.

Greedy Technique:

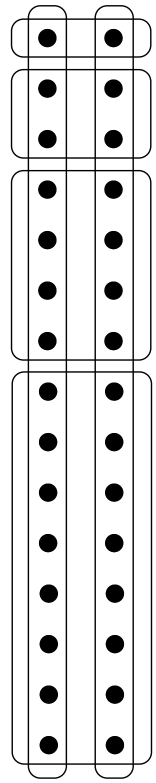
Choose subset with largest number of uncovered elements.

(Implementation: Doubly-linked list for each element in S. Doubly-linked list for each subset. Ordered table for priority queue.)

Achieves: $\frac{Greedy}{OPT} \le \ln(|\text{largest subset}|) + 1$

See CLRS, p. 1119-1121 for detailed proof.

Example to motivate logarithmic approximation ratio:



Traveling Salesperson (complete graph)

No ρ-approximation in P time (unless P = NP).

Suppose a graph is to be tested for a Hamiltonian cycle:

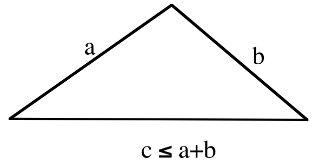
Weight each "real" edge with 1.

"Imaginary" edges are weighted with $\rho|V| + 1$.

If ρ-approximation gives TSP with length $|V|$, then performance is better than guaranteed and have a Hamiltonian cycle.

If ρ-approximation gives TSP with length $> \rho|V|$, then performance guarantee has not been met.

If edge weights obey triangle inequality, the scale-up problem is avoided.



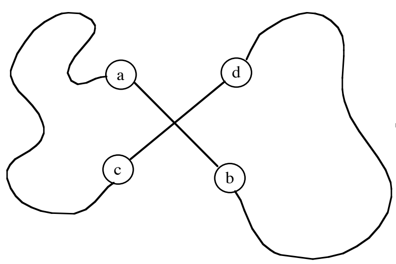
2-approximate

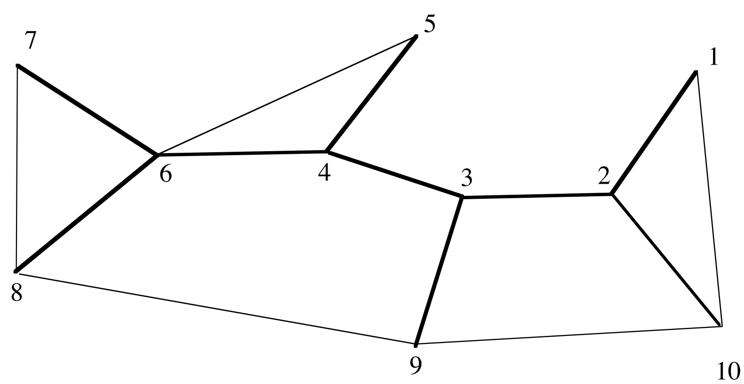
1. Find minimum spanning tree.

2. Depth-first search - order vertices by discovery time.

3. Return to start vertex.

4. Remove edge crossings - optional





2-approximate proof:

1. $|MST| \le |T_{OPT}|$

   Best case - removing largest edge in OPT (a cycle) gives MST.

2. $|T_\Delta| \le 2|MST|$

   Since MST short cuts are no longer than subpath skipped.

   $\frac{1}{2}|T_\Delta| \le |MST| \le |T_{OPT}|$, so $|T_\Delta| \le 2|T_{OPT}|$

(Aside: see `http://en.wikipedia.org/wiki/Christofides_algorithm` for a 3/2-approximate technique.)

Subset Sums - exact solution by dynamic programming (`ssNew.c`)

Given n numbers $S_1, \ldots S_n$. find a subset (e.g. an index set chosen from 1 . . . n) such that sum is an exact value C.

1. Use table with C entries

   A[i] = j, where j is the *smallest* index such that i = S_j + values w/index < j

2. Initialize table and then go forward

1	2	3	4	5	6	C=36
2	3	6	11	15	25	

A:	1	2	3	4	5	6	7	8	9	10
		1	2			3				
	11	12	13	14	15	16	17	18	19	20
	4				5					
	21	22	23	24	25	26	27	28	29	30
					6					
	31	32	33	34	35	36				

Now suppose each S_i is multiplied by 1000. D.P. table grows by factor of 1000.

Ordinary Subset Sum (CLRS, p. 1128-1129)

Saves space by maintaining lists of reachable sums.

1	2	3	4	5	6
2	3	6	11	15	25

```
Target=36

L0={0/0}
    2


L1={0/0, 2/1}
    3    5


L2={0/0, 2/1, 3/2, 5/2}
    6    8    9   11


L3={0/0,  2/1,  3/2,  5/2,  6/3,  8/3,  9/3,  11/3}
   11     13    14    16    17    19    20     22


L4={0/0,  2/1,  3/2,  5/2,  6/3,  8/3,  9/3,  11/3,  13/4,
   15     17    18    20    21    23    24     26     28

   14/4,  16/4,  17/4,  19/4,  20/4,  22/4}
   29     31     32     34     35     37?


L5={0/0,  2/1,  3/2,  5/2,  6/3,  8/3,  9/3,  11/3,  13/4,
   25     27    28    30    31    33    34     36     38?

   14/4, 15/5, 16/4, 17/4, 18/5, 19/4, 20/4, 21/5, 22/4,

   23/5, 24/5, 26/5, 28/5, 29/5, 31/5, 32/5, 34/5, 35/5}


L6={0/0,  2/1,  3/2,  5/2,  6/3,  8/3,  9/3,  11/3,  13/4,

   14/4, 15/5, 16/4, 17/4, 18/5, 19/4, 20/4, 21/5, 22/4,

   23/5, 24/5, 25/6, 26/5, 27/6, 28/5, 29/5, 30/6, 31/5,

   32/5, 33/6, 34/5, 35/5, 36/6}
```

Subset Sum with Range of Target Values

Uses intervals to achieve approximation to within ε of desired value.

CLRS, p. 1130-1133, gives fully PTAS based on ε.

1	2	3	4	5	6
2	3	6	11	15	25

```
Target=33..36

L0={0..3/0}
    2..5

L1={0..3/0, 3..5/1}
    3..6    6..8


L2={0..3/0, 3..5/1, 5..8/2}
    6..9    9..11  11..14


L3={0..3/0, 3..5/1, 5..8/2, 8..14/3}
   11..14  14..16  16..19  19..25


L4={0..3/0, 3..5/1, 5..8/2, 8..14/3, 14..25/4}
   15..18  18..20  20..23  23..29    29..40?


L5={0..3/0, 3..5/1, 5..8/2, 8..14/3, 14..25/4, 25..36/5}
   25..28  28..30  30..33  33..39?


L6={0..3/0, 3..5/1, 5..8/2, 8..14/3, 14..25/4, 25..36/5}
```

Graph (Vertex) Coloring - no efficient heuristic (unless P = NP)

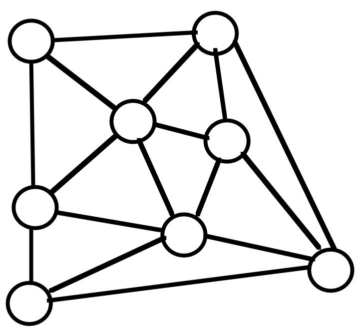
Suppose a P time algorithm exists to color every graph G with $X(G) \geq k$ using *fewer* than $\frac{4}{3}X(G)$ colors, then 3-colorability would be in P.

Proof:

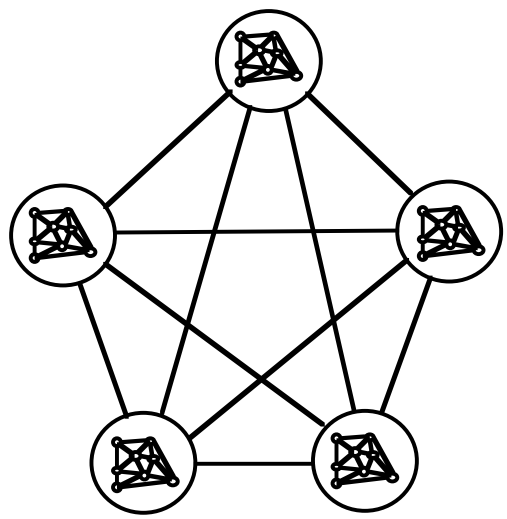
1. C_k is the complete graph with k vertices.

2. G is an instance of 3-colorability.

3. Graph $H = C_k[G]$ (composition of graphs).



G



C_k

Each vertex in a copy of G is connected to all vertices in all other copies of G.

4. If $X(G) = 3$, each copy of G requires 3 colors, so $X(H) = 3k$.

   Algorithm must use fewer than $\frac{4}{3}(3k) = 4k$ colors for G to be 3-colorable.

   Otherwise, $X(G) > 3$ so each copy of G needs at least 4 colors. $X(H) \geq 4k$

   Algorithm must then use at least $4k$ colors to color H.

Thus, such an algorithm is “unlikely”.