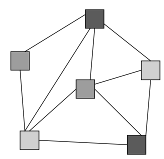
CSE 589
Applied Algorithms
Spring 1999

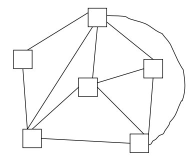
3-Colorability
Branch and Bound

3-Colorability

- Input: Graph G = (V,E) and a number k.
- Output: Determine if all vertices can be colored with 3 colors such that no two adjacent vertices have the same color.



3-colorable



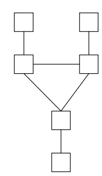
Not 3-colorable

3-CNF-Sat $\leq_P$ 3-Color

- Given a 3-CNF formula *F* we have to show how to construct in polynomial time a graph *G* such that:
  - *F* is satisfiable implies *G* is 3-colorable
  - *G* is 3-colorable implies *F* is satisfiable

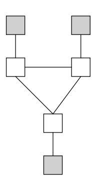
The Gadget

- This is a classic reduction that uses a “gadget”.
- Assume the outer vertices are colored at most two colors. The gadget is 3-colorable if and only if the outer vertices are not all the same color.

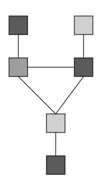


Properties of the Gadget

- Three colorable if and only if outer vertices not all the same color.



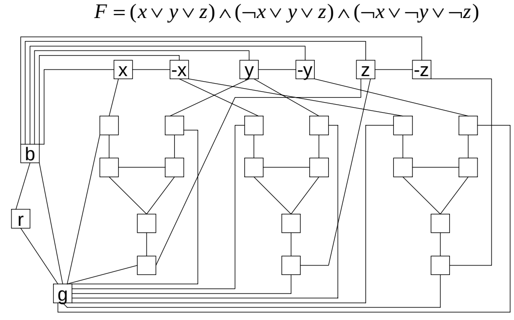
Not 3 colorable



Is 3 colorable

Reduction by Example

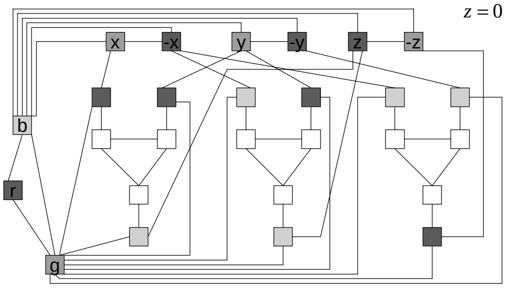
$$F = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$$



Satisfaction Example

$F = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$

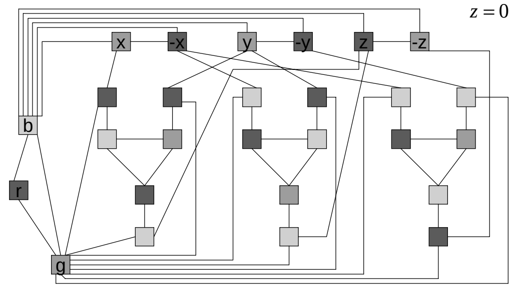
$x = 1$
$y = 1$
$z = 0$



Satisfaction Example

$F = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$

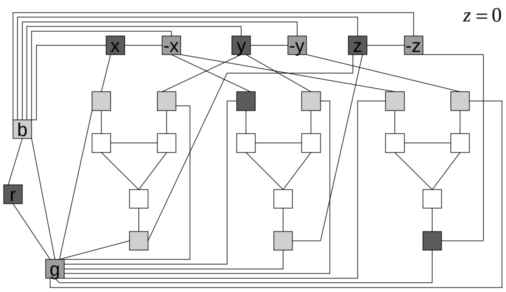
$x = 1$
$y = 1$
$z = 0$



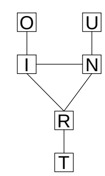
Non-Satisfaction Example

$F = (x \vee y \vee z) \wedge (\neg x \vee y \vee z) \wedge (\neg x \vee \neg y \vee \neg z)$

$x = 0$
$y = 0$
$z = 0$



Naming the Gadget



General Construction

$F = \bigcap_{i=1}^{k} (a_{i1} \vee a_{i2} \vee a_{i3})$ where $a_{ij} \in \{x_1, \neg x_1, \ldots, x_n, \neg x_n\}$

$G = (V, E)$ where

$V = \{r, g, b\} \cup \{x_1, \neg x_1, \ldots, x_n, \neg x_n\} \cup \{O_i, U_i, T_i, I_i, N_i, R_i : 1 \le i \le k\}$

$E = \{\{r, g\}, \{g, b\}, \{b, r\}\}$

$\quad \cup \{\{x_1, \neg x_1\}, \ldots, \{x_n, \neg x_n\}\}$

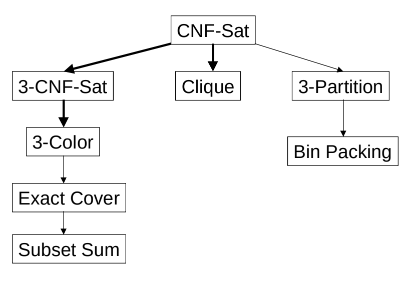
$\quad \cup \{\{x_1, b\}, \{\neg x_1, b\}, \ldots, \{x_n, b\}, \{\neg x_n, b\}\}$

$\quad \cup \{\{O_i, I_i\}, \{U_i, N_i\}, \{T_i, R_i\}, \{I_i, N_i\}, \{N_i, R_i\}, \{R_i, I_i\} : 1 \le i \le k\}$

$\quad \cup \{\{a_{i1}, O_i\}, \{a_{i2}, U_i\}, \{a_{i3}, T_i\} : 1 \le i \le k\}$

$\quad \cup \{\{O_i, g\}, \{U_i, g\}, \{T_i, g\} : 1 \le i \le k\}$

Reductions



Exact Cover

- Input: A set $U = \{u_1, u_2, \ldots, u_n\}$ and subsets $S_1, S_2, \ldots, S_m \subseteq U$
- Output: Determine if there is set of pairwise disjoint set that union to U, that is, a set X such that:

$$X \subseteq \{1,2,\ldots,m\}$$
$$i, j \in X \text{ and } i \neq j \text{ implies } S_i \cap S_j = \phi$$
$$\bigcup_{i \in X} S_i = U$$

Example of Exact Cover

$$U = \{a,b,c,d,e,f,g,h,i\}$$

$$\{a,c,e\},\{a,f,g\},\{b,d\},\{b,f,h\},\{e,h,i\},\{f,h,i\},\{d,g,i\}$$

Exact Cover

$$\{a,c,e\},\{b,f,h\},\{d,g,i\}$$

3-Partition

- Input: A set of numbers $A = \{a_1, a_2, \ldots, a_{3m}\}$ and number B with the properties that $B/4 < a_i < B/2$ and $$\sum_{i=1}^{3m} a_i = mB.$$
- Output: Determine if A can be partitioned into S_1, S_2,…, S_m such that for all i $$\sum_{j \in S_i} a_j = B.$$

Note: each S_i must contain exactly 3 elements.

Example of 3-Partition

- A = {26, 29, 33, 33, 33, 34, 35, 36, 41}
- B = 100, m = 3
- 3-Partition
  - 26, 33, 41
  - 29, 36, 35
  - 33, 33, 34

Bin Packing

- Input: A set of numbers $A = \{a_1, a_2, \ldots, a_{3m}\}$ and numbers B (capacity) and K (number of bins).
- Output: Determine if A can be partitioned into S_1, S_2,…, S_K such that for all i

$$\sum_{j \in S_i} a_j \leq B.$$

Bin Packing Example

- A = {2, 2, 3, 3, 3, 4, 4, 4, 5, 5, 5}
- B = 10, K = 4
- Bin Packing
  - 3, 3, 4
  - 2, 3, 5
  - 5, 5
  - 2, 4, 4

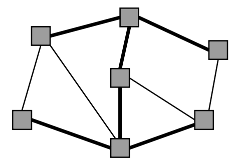
Perfect fit!

Coping with NP-Completeness

- Given a problem appears to be hard what do you do?
  - Try to find a good algorithm for it.
  - Try to show its decision version is NP-complete or NP-hard.
  - Failing both, the problem probably is a hard one.
  - For a hard problem there are many things to try.
    - Branch-and-bound algorithm - for exact solution
    - Approximate algorithm - heuristic

Load Balanced Spanning Tree Cost Criteria

- Given a graph G = (V,E) and a spanning tree T.
  - d(T) = max degree of any vertex of T
  - c(T) = sum of the squares of the degrees



d(T) = 3
c(T) = 4*1 + 1*4 + 2*9 = 26

Advantage of c(T) is that it has finer gradations.

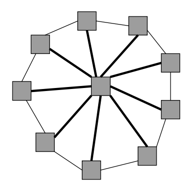
Deriving c(T)

- Every spanning tree on n vertices has $n-1$ edges. Hence, the average number of edges per vertex is $d = 2(n-1)/n$, about 2.
- Let d_i be the degree of vertex i. The variance in degree is

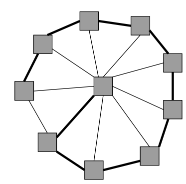
$$\sum_{i=1}^{n}(d_i - d)^2 / n = (\sum_{i=1}^{n} d_i^{\,2} - d^2)/n$$

- Minimizing the variance is equivalent to minimizing $\sum_{i=1}^{n} d_i^{\,2}$

Examples of c(T)

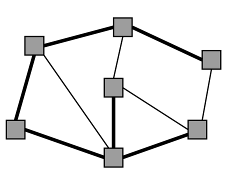


c(T) = 9* 1^2 + 1*9^2 = 90

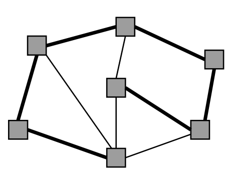


c(T) = 2*1^2 + 8*2^2 = 34

Another Example



c(T) = 3*1 + 3*4 + 1*9 = 24



c(T) = 2*1 + 5*4 = 22

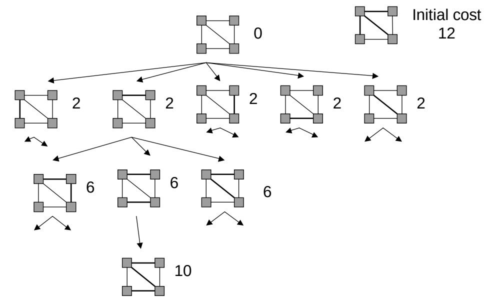
Load Balanced Spanning Tree with Minimum Variance

- Input: Undirected graph G = (V,E).
- Ouput: A spanning tree that minimizes the sum of the squares of the degrees of the vertices in the tree.

Branch and Bound

- Start with an initial tree T with cost c(T).
- Systematically search through all forests by recursively (branching) adding new edges to the current forest.
- Discontinue a search if the forest cannot be contained in a spanning tree of smaller cost. (This is the bounding step).
- This is better than exhaustive search, but it is still only valuable on very small problems.

Example of Branch and Bound



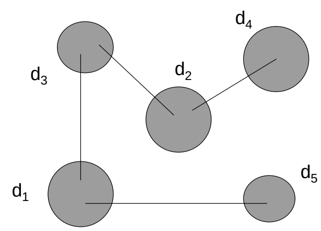
Bounding Condition

- Let *c(F)* be the cost of the current forest of *k* trees where tree T_i had minimum degree vertex d_i sorted smallest to largest. Let *B* be the best cost of any tree so far.
- The lowest possible cost of any tree containing *F* is

$$m(F) = c(F) + 2\sum_{i=1}^{k-2}(d_i+1)^2 - 2\sum_{i=1}^{k-2} d_i^2 + \sum_{i=k-1}^{k}(d_i+1)^2 - \sum_{i=k-1}^{k} d_i^2$$

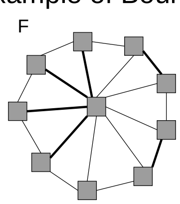
- If $m(F) \geq B$ then do not continue searching from *F*.

Graphic of Bounding Condition



$d_1 \leq d_2 \leq d_3 \leq d_4 \leq d_5$

Example of Bounding



d_i = 0,1,1,1
c(F) = 1*0 + 8*1 + 1*16 = 24
m(F) = 24 + 2(1*1 + 1*4) - 2(1*0 + 1*1)
+ (1*1 +1*4) - (1*0 + 1*1)
= 36

Branch and Bound Control

```
The edges of G are in an array E[1..m]
F is a set of indices of edges, initially empty
There is an initial Best-Tree with Best-Cost

LBST-Search(F)
   if F is a tree then
      if c(F) < Best-Cost then
         Best-Tree := F;
         Best-Cost := c(F);
   else {F is not a tree}
      for i = last-index-in(F) + 1 to m do
         if not(cycle(F,i)) and m(F,i) < Best-Cost then
            F := union(F,i);
            LBST-Search(F);
```

Notes on Branch and Bound

- Branch and bound is still an exponential search. To make it work well many efficiencies should be made.
  - Eliminate copy of the partial solution F on the recursive call.
  - Maintain cost of partial solution F and its sequence of minimum degrees to make computation of m(F,i) fast.
  - Use up tree for cycle checking.
  - Reduce use of expensive bounding checks when possible.
  - Add more bounding checks