Analysis of Algorithms

Coping With NP Completeness

Hikmat Farhat

April 21, 2020

Coping with NP-complete problems

- It **seems** impossible to solve NP-complete problems in polynomial time
- We can try to find an "efficient" non-polynomial time algorithm
- Basically find a solution without using brute force (i.e. trying all possibilities)
- Brute force for 3SAT
  - Given a 3SAT instance with n variables
  - try all possible 2^n assignments
  - if formula not satisfiable then we have to go through all 2^n possibilities
  - Complexity $O(n^3 \cdot 2^n)$. Why ?
  - Because evaluating each clause takes constant time
  - There are at most n choose 3 clauses $\binom{n}{3} = \frac{n!}{(n-3)!3!} = O(n^3)$

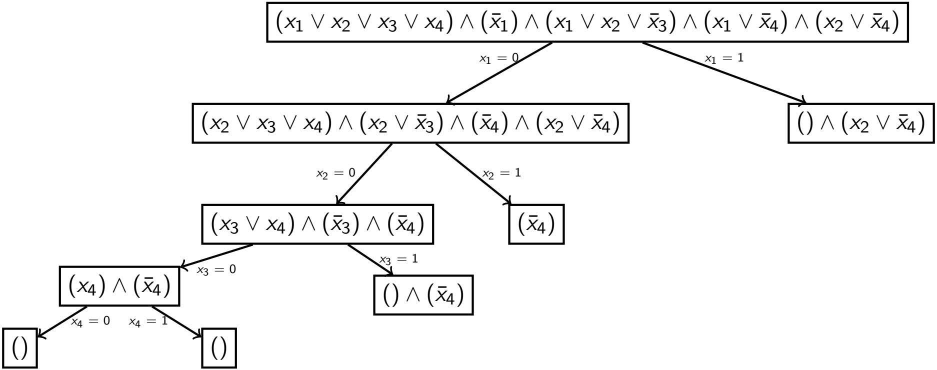
SAT

- Can we do better?
- Construct a solution, if possible, step by step
- If current partial solution cannot be extended to a valid solution: backtrack
- Consider the formula below: if we assign the true value to variable x:
    1. Remove all clauses containing x since they are satisfied
    2. Remove $\bar{x}$ from all clauses
- Note that an empty clause is **unsatisfiable**

$$(x_1 \vee x_2 \vee x_3 \vee x_4) \wedge (\bar{x}_1) \wedge (x_1 \vee x_2 \vee \bar{x}_3) \wedge (x_1 \vee \bar{x}_4) \wedge (x_2 \vee \bar{x}_4)$$

Backtracking example

- As can be seen from the backtracking below the formula is satisfiable with
- $x_1 = False, x_2 = True, x_4 = False$ and x_3 can be either True or False



Solving 3SAT using backtracking

- Let $\phi = C_1 \wedge C_2 \wedge \ldots \wedge C_k$ be a 3SAT formula with n variables and k clauses.
- If ϕ is empty, i.e. has no clauses, then it is trivially satisfiable.
- If ϕ has at least one clause we can write

$$\begin{aligned}\phi &= (x \vee y \vee z) \wedge \phi' \\ &= (x \wedge \phi') \vee (y \wedge \phi') \vee (z \wedge \phi')\end{aligned}$$

- So we reduce the 3SAT with n variables to three probems with $n-1$ variables.
- We obtain the recurrence

$$T(n) = 3T(n-1) + O(n^3)$$

- With a solution of $T(n) = O(n^3 3^n)$. Which is worse than before !

- We are doing unnecessary work.
- Now $x \wedge \phi$ is true if both terms are true which is equivalent to $\phi'|x$ (i.e. ϕ' is sat given that x is true)
- If $\phi'|x$ is not satisfiable then $y \wedge \phi'$ is satisfiable iff $\phi'|\bar{x}y$
- Finally, if both $\phi'|x$ and $\phi'|\bar{x}y$ are not satisfiable then $z \wedge \phi'$ is satisfiable iff $\phi'|\bar{x}\bar{y}z$ is.
- The above suggests the following algorithm for solving a 3SAT formula

```
SolveSAT (φ)
if φ = ∅ then
    return True
if φ contains an empty clause then
    return False
Decompose φ = (x ∨ y ∨ z) ∧ φ'
if SolveSAT (φ|x) then
    return True
if SolveSAT (φ|x̄y) then
    return True
return SolveSAT (φ|x̄ȳz)
```

- The SolveSAT function obeys the recurrence

$$T(n) = T(n-1) + T(n-2) + T(n-3) + O(n^k)$$

- Which has a solution (see later) $O(1.839^n)$

Annihilator method

- An operator takes functions as input to produce different functions.
- We will use the following operators
  1. Sum of two functions $(f+g)(n) = f(n) + g(n)$
  2. Scale of a function $(\alpha \cdot f)(n) = \alpha \cdot f(n)$
  3. Shift (right) $(E \cdot f)(n) = f(n+1)$.
- An annihilator is an **operator** that transforms a function into a null function.
- For example, let $f(n) = 2^n$ then
  $(E-2)f(n) = (E-2)2^n = 2^{n+1} - 2 \cdot 2^n = 0$

- In general $(E-c)\alpha \cdot a^n = \alpha \cdot a^{n+1} - \alpha \cdot c \cdot a^n = \alpha \cdot (a-c) \cdot a^n$. This is null **iff** $c = a$.
- Also, if $c \neq a$ then $(E-c)a^n = constant \cdot a^n$
- How does that help us in solving recurrences?
- Consider the following recurrence

$$\begin{aligned} & T(n) = 3T(n-1) \\ \Rightarrow & T(n+1) - 3T(n) = 0 \\ \Rightarrow & ET(n) - 3T(n) = 0 \\ \rightarrow & (E-3)T(n) = 0 \\ \Rightarrow & T(n) = \alpha \cdot 3^n \text{ for some constant } \alpha \end{aligned}$$

- We saw from the above that for the shift operator

$$(E-c)\alpha \cdot a^n = \alpha \cdot (E-c)a^n$$

- Then if we have $\alpha a^n + \beta b^n$ it follows that

$$(E-b)(E-a)(\alpha a^n + \beta b^n) = (E-b)\gamma b^n = \gamma(E-b)b^n = 0$$

- In general, for **distinct** $a_0, \ldots, a_k$

$$(E-a_0)(E-a_1)\ldots(E-a_k)\left[\sum_{i=0}^{k} \alpha_i a_i^n\right] = 0$$

Polynomials

- Consider the function $\alpha \cdot n + \beta$
- $(E-1)\alpha n = \alpha(n+1) + \beta - \alpha n - \beta = \alpha$
- So $(E-1)^2(\alpha n + \beta) = 0$
- In general if $(E-1)^k F(n) = 0$ then

$$F(n) = \sum_{i=0}^{k-1} \alpha_i n^i$$

- Also $(E-a)^k F(n) = 0$ then

$$F(n) = \left(\sum_{i=0}^{k-1} \alpha_i n^i\right) \cdot a^n$$

Fibonacci

- Recall that the Fibonacci sequence obeys the following recurrence

$$\begin{aligned} F(n+2) &= F(n+1) + F(n) \\ \Rightarrow & E^2F(n) - EF(n) - F(n) = 0 \\ \Rightarrow & (E^2 - E - 1)F(n) = 0 \end{aligned}$$

- The above can be factored as

$$(E-\phi)(E-\hat{\phi})F(n) = 0$$

- Where $\phi = \frac{1+\sqrt{5}}{2}$ and $\hat{\phi} = 1 - \phi$.
- Therefore

$$F(n) = \alpha\phi^n + \beta\hat{\phi}^n$$

- The values of α and β are determined from the boundary conditions $F(0) = 0$ and $F(1) = 1$ thus $\alpha = -\beta = \frac{1}{\sqrt{5}}$ and we get

$$F(n) = \frac{1}{\sqrt{5}}\phi^n - \frac{1}{\sqrt{5}}(1-\phi)^n$$

Tower of Hanoi

- Remember for the Tower of Hanoi the recurrence was

$$T(n) = 2T(n-1) + 1$$

- using E

$$(E-2)T - 1 = 0$$

- This is the first time we see a non homogeneous

AVL Tree

- Recall an AVL tree has the property that the left and right subtrees cannot have a height difference more than 1.
- Consider the **smallest** (in terms of number of nodes) AVL tree of height h.
- Since the tree is the smallest then the left tree has height $h-1$ and the right has height $h-2$ (or vice versa).
- Therefore the smallest AVL of height h obeys the recurrence

$$\begin{aligned} N(h) &= N(h-1) + N(h-2) + 1 \\ \Rightarrow N(h+2) - N(h+1) - N(h) = 1 & \\ \Rightarrow (E^2 - E - 1)N(h) = 1 & \end{aligned}$$

- The solution is

$$N(h) = \alpha\phi^h + \beta\hat{\phi}^h + \gamma$$

- We can determine the constants α, β, γ by using the boundary conditions $N(0) = 1, N(1) = 2, N(2) = 4$
- We need the asymptotic behavior, for large h

$$N(h) = \Omega(\phi^h)$$

- Therefore

$$h = O(\log n)$$

Subset sum

- Given a set of integers S and a target value t the subset sum asks if there is a subset $M \subseteq S$ such that $sum(M) = t$.
- We already have seen a solution using dynamic programming.
- Here we will solve it using backtracking.
- Let $S = \{x_1, \ldots, x_n\}$. At each step i the backtracking algorithm tries two choices: including/excluding x_i in/from the solution.

Subset Sum

- At each step keep two variables: *sum* and *remainder*
- The *sum* denotes the current sum and the *remainder* denotes the sum of all items that have not been used yet.
- Let t be the target. Given a choice between adding/not adding element with value v
  1. if $sum + v = t$ then solution found, return solution.
  2. if $sum + v > t$ backtrack, do not include v
  3. if $sum + remainder - v < t$ backtrack .

Branch and Bound: Knapsack

- We compute an upper bound using the greedy solution for the fractional knapsack

$$S = v_1 + v_2 + \ldots + v_k + \frac{W - w_1 - \ldots - w_k}{w_{k+1}} v_{k+1}$$

- But since

$$\frac{v_1}{w_1} > \frac{v_2}{w_2} > \ldots > \frac{v_k}{w_k}$$

- Then we can write

$$S \leq \frac{w_1}{w_1} v_1 + \frac{w_2}{w_1} v_1 + \ldots + \frac{w_k}{w_1} v_1 + \frac{W - w_1 - \ldots - w_k}{w_1} v_1$$
$$S \leq W \cdot \frac{v_1}{w_1}$$

- Assuming that items are sorted by descending $\frac{v}{w}$ we use backtracking with the same order of selection for the items
- At each step, given a knapsack of size W and a list of unused items
- We have an upper bound for the entire branch
- As an example assume we have a knapsack of capacity $W = 11$ and the following items

item	Value	Weight
1	1	1
2	2	3
3	3	5
4	4	7

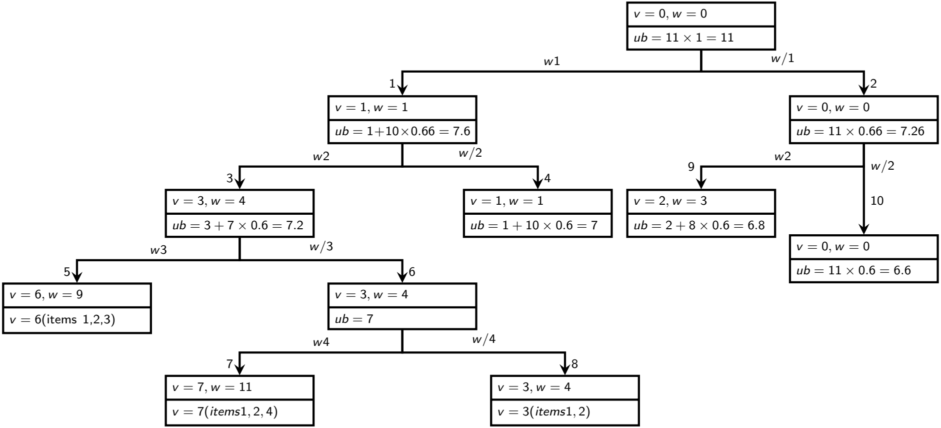


Figure: Branch and Bound for Knapsack instance

Example2

- Consider the following instance with the knapsack weight $W = 7$

item	Value	Weight	density
1	8	5	1.6
2	6	4	1.5
3	2	2	1
4	1	1	1

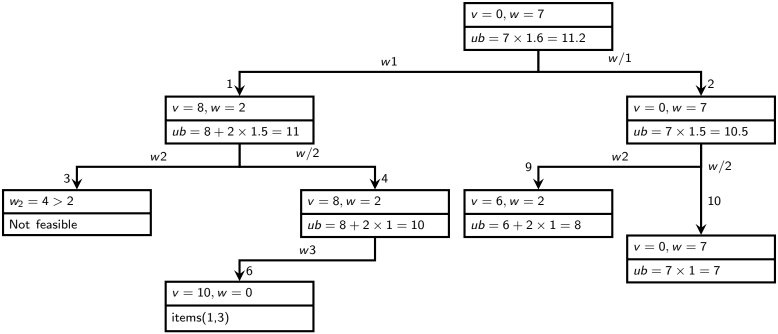


Figure: Branch and Bound for Knapsack instance

Clique

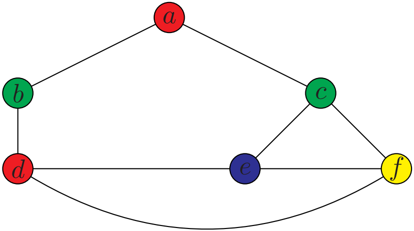
- Recall that given a graph $G = \langle V, E \rangle$ a subset $S \subseteq V$ is said to be a **clique** iff for all $u, v \in S$ then $(u, v) \in E$.
- We also saw that if G has a **clique** of size k then $\bar{G} = \langle V, \bar{E} \rangle$ has an **independent se**t of size k, where $(u, v) \in E$ iff $(u, v) \notin E$
- We will use branch and bound to find a clique of maximum size in a given graph
- First we develop lower and upper bounds for the clique problem.

Upper bound: greedy k-coloring

- Given a graph $G = \langle V, E \rangle$ we need to color it using k-colors.
- First assign a number to each color: $1, 2, \ldots, k$.
- Choose a vertex v and assign to it the **lowest** number that is not used by its neighbors.

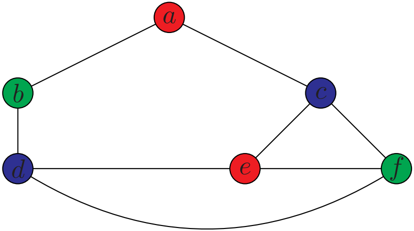
Example

- We use the greedy coloring to the graph shown below were nodes are considered in alphabetical order.
- The colors are numbered as $\{Red = 1, Green = 2, Blue = 3, Yellow = 4, Magenta = 5, \ldots\}$



Greedy coloring not optimal, used as upper bound

- The greedy coloring in the previous example used 4 colors
- An optimal coloring, as shown below, uses only 3 colors



- But greedy coloring gives an **upper bound**:
- For any graph optimal coloring $\leq$ greedy coloring

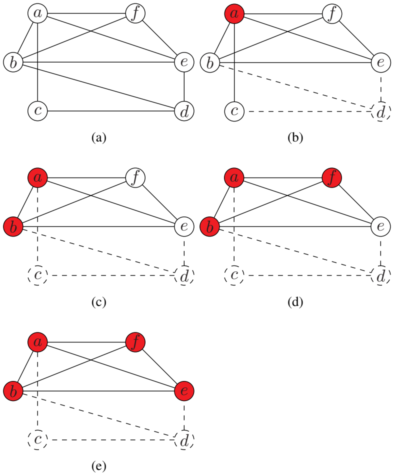
How is that used for Clique?

- Suppose a graph $G = \langle V, E \rangle$ has a **Clique of size** k.
- Then G **cannot be colored by less** than k colors
- **Conversely**, if a graph has a k coloring then
- **The size of maximal Clique** $\leq k$.
- Let k_g be the number of colors obtained by greedy coloring
- Since k_g is an upper bound for coloring
- **Then the size of maximal Clique** $\leq k_g$

Lower Bound: greedy Clique

- Given a graph $G = \langle V, E \rangle$ we can obtain a Clique using a greedy strategy as follows
  1. $C \leftarrow \emptyset$
  2. Select the highest degree vertex v from $V - C$ and added to C
  3. Remove all nodes that are not connected to v from V
  4. repeat until V is empty

Greedy Clique Example



Clique Branch-and-Bound

- As an example for using branch-and-bound for clique using the already discussed upper and lower bounds
- See the paper by Dawn M. Strickland on blackboard.