Design and Analysis of Algorithms

Mohammad GANJTABESH

mgtabesh@ut.ac.ir

School of Mathematics, Statistics and Computer Science,
University of Tehran,
Tehran, Iran.

Techniques for the design of Algorithms

The classical techniques are as follows:

1. Divide and Conquer
2. Dynamic Programming
3. Greedy Algorithms
4. Backtracking Algorithms
5. Branch and Bound Algorithms

0/1-Knapsack

Review

Suppose that we have n objects, say o_i ($i = 1, 2, \cdots, n$), each with corresponding weight (w_i) and profit (p_i), and a weight bound b. The goal of this problem is to find an $X = (x_1, x_2, \cdots, x_n)$ that maximize $\sum_{i=1}^{n} x_i p_i$ with respect to $\sum_{i=1}^{n} x_i w_i \leq b$.

- if $x_i \in \{0, 1\}$ the this problem is called 0/1-Knapsack.
- if $x_i \in [0, 1]$ the this problem is called fractional-Knapsack.

0/1-Knapsack

Review

Suppose that we have n objects, say o_i ($i = 1, 2, \cdots, n$), each with corresponding weight (w_i) and profit (p_i), and a weight bound b. The goal of this problem is to find an $X = (x_1, x_2, \cdots, x_n)$ that maximize $\sum_{i=1}^{n} x_i p_i$ with respect to $\sum_{i=1}^{n} x_i w_i \leq b$.

- if $x_i \in \{0, 1\}$ the this problem is called 0/1-Knapsack.
- if $x_i \in [0, 1]$ the this problem is called fractional-Knapsack.

Review

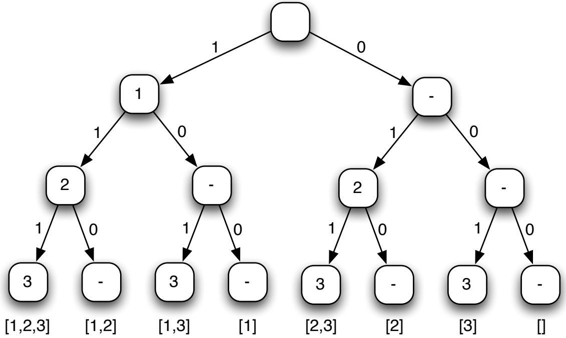
Fractional-Knapsack problem can be solved efficiently by using Greedy approach and the answer is optimal.

0/1-Knapsack

To design a backtracking algorithm for this problem, we should generate all subsets of $\{0, 1\}^n$ and check which one is optimal.

0/1-Knapsack

To design a backtracking algorithm for this problem, we should generate all subsets of $\{0, 1\}^n$ and check which one is optimal.



0/1-Knapsack: Backtracking Algorithms

```
Backtrack-Knapsack($X, optX, optP, \ell$){
    if $\ell = n + 1$ then{
        if $\sum_{i=1}^{n} x_i w_i \leq b$ then{
            $curP \leftarrow \sum_{i=1}^{n} x_i p_i$;
            if $curP \geq optP$ then{
                $optP \leftarrow curP$;
                $optX \leftarrow [x_1, x_2, \cdots, x_n]$;
            }
        }
    }
    else{
        $x_l \leftarrow 1$;
        Backtrack-Knapsack($X, optX, optP, \ell + 1$);
        $x_l \leftarrow 0$;
        Backtrack-Knapsack($X, optX, optP, \ell + 1$);
    }
}
```

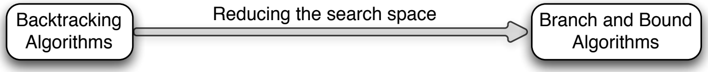
Techniques for the design of Algorithms

The classical techniques are as follows:

1. Divide and Conquer
2. Dynamic Programming
3. Greedy Algorithms
4. Backtracking Algorithms
5. Branch and Bound Algorithms

Basic Concepts

- Branch-and-Bound is based on backtracking, which is an exhaustive searching technique in the space of all feasible solutions.



- The cardinality of the sets of feasible solutions are typically as large as 2^n, $n!$, or even n^n for inputs of size n.
- The idea of the branch-and-bound technique is to speed up backtracking by omitting the search in some parts of the space of feasible solutions, because one is already able to recognize that these parts do not contain any optimal solution in the moment when the exhaustive search would start to search in these parts.
- The branch-and-bound is based on some pre-computation of a bound on the cost of an optimal solution (a lower bound for maximization problems and an upper bound for minimization problems).

0/1-Knapsack: Branch and Bound Algorithms

```
B&B-Knapsack1$(X, optX, optP, \ell, curW)\{$
    if $\ell = n+1$ then$\{$
        if $\sum_{i=1}^{n} x_i p_i \geq optP$ then$\{$
            $optP \leftarrow \sum_{i=1}^{n} x_i p_i$;
            $optX \leftarrow [x_1, x_2, \cdots, x_n]$;
        $\}$
    $\}$
    else$\{$
        if $curW + w_\ell \leq b$ then $C_\ell \leftarrow \{1, 0\}$;
        else $C_\ell \leftarrow \{0\}$;
    $\}$
    for each $x \in C_\ell$ do $\{$
        $x_l \leftarrow x$;
        Backtrack-Knapsack$(X, optX, optP, \ell + 1, curW + x_\ell w_\ell)$;
    $\}$
$\}$
```

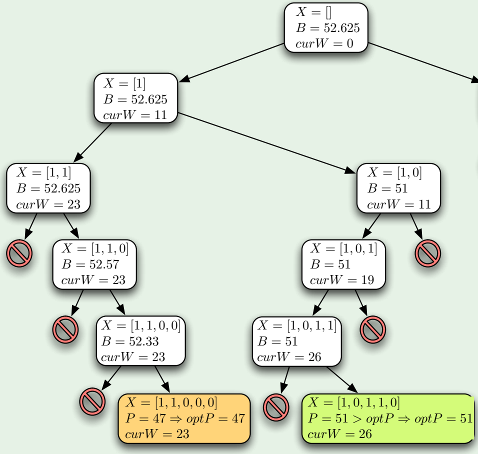
0/1-Knapsack: Branch and Bound Algorithms (more pruning)

```
B&B-Knapsack2($X, optX, optP, \ell, curW$){
    if $\ell = n+1$ then{
        if $\sum_{i=1}^{n} x_i p_i \geq optP$ then{
            $optP \leftarrow \sum_{i=1}^{n} x_i p_i$;
            $optX \leftarrow [x_1, x_2, \cdots, x_n]$;
        }
    }
    else{
        if $curW + w_\ell \leq b$ then $C_\ell \leftarrow \{1,0\}$;
        else $C_\ell \leftarrow \{0\}$;
    }
    $B \leftarrow \sum_{i=1}^{l-1} x_i p_i + GFK(p_\ell, p_{\ell+1}, \cdots, p_n, w_\ell, w_{\ell+1}, \cdots, w_n, b - curW)$;
    if $B \leq optP$ then return;
    for each $x \in C_\ell$ do {
        $x_l \leftarrow x$;
        Backtrack-Knapsack($X, optX, optP, \ell+1, curW + x_\ell w_\ell$);
    }
}
```

Example for B&B-Knapsack2 algorithm

Example

Suppose that $P = [23, 24, 15, 13, 16]$, $W = [11, 12, 8, 7, 9]$, and $b = 26$.
The algorithm B&B-Knapsack2 works as follows:



$X = []$ $B = 52.625$ $curW = 0$
$X = [1]$ $B = 52.625$ $curW = 11$
$X = [0]$ $B = 50.14$ $curW = 0$
$X = [1,1]$ $B = 52.625$ $curW = 23$
$X = [1,0]$ $B = 51$ $curW = 11$
$X = [1,1,0]$ $B = 52.57$ $curW = 23$
$X = [1,0,1]$ $B = 51$ $curW = 19$
$X = [1,1,0,0]$ $B = 52.33$ $curW = 23$
$X = [1,0,1,1]$ $B = 51$ $curW = 26$
$X = [1,1,0,0,0]$ $P = 47 \Rightarrow optP = 47$ $curW = 23$
$X = [1,0,1,1,0]$ $P = 51 > optP \Rightarrow optP = 51$ $curW = 26$

Comparision of previous algorithms

The following table represents the worse case size of search space of random instances executed for 5 times.

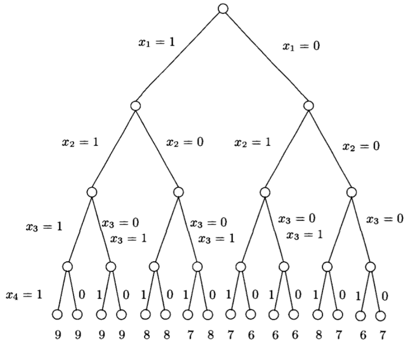
n	Backtrack-Knapsack	B&B-Knapsack1	B&B-Knapsack2
8	511	333	78
12	8191	4988	195
16	131071	78716	601
20	2097151	1257745	480
24	33554431	19814875	755

Backtracking algorithm for Max-SAT

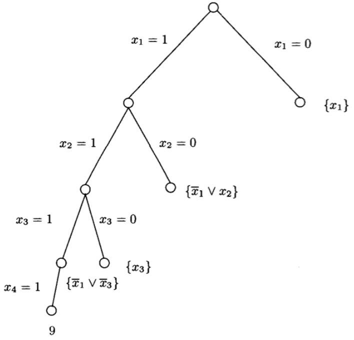
Example

$$\begin{aligned}\varphi(x_1, x_2, x_3, x_4) \quad = \quad & (x_1 \vee \overline{x}_2) \wedge (x_1 \vee x_3 \vee \overline{x}_4) \wedge (\overline{x}_1 \vee x_2) \\ & \wedge (x_1 \vee \overline{x}_3 \vee x_4) \wedge (x_2 \vee x_3 \vee \overline{x}_4) \wedge (x_1 \vee \overline{x}_3 \vee \overline{x}_4) \\ & \wedge x_3 \wedge (x_1 \vee x_4) \wedge (\overline{x}_1 \vee \overline{x}_3) \wedge x_1 .\end{aligned}$$

Backtrack:

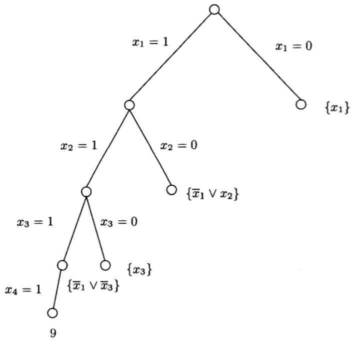


Branch-and-Bound (left child first):

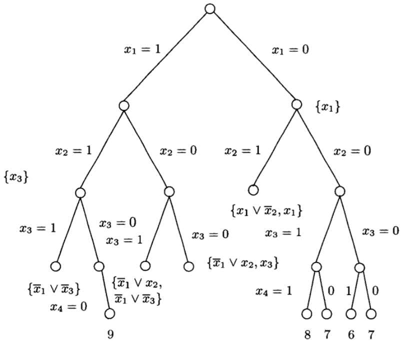


Backtracking algorithm for Max-SAT

Branch-and-Bound (left child first):



Branch-and-Bound (right child first):



$x_1 = 1$
$x_1 = 0$
$\{x_1\}$
$x_2 = 1$
$x_2 = 0$
$x_2 = 1$
$x_2 = 0$
$\{x_3\}$
$\{x_1 \vee \overline{x}_2, x_1\}$
$x_3 = 1$
$x_3 = 0$
$x_3 = 1$
$x_3 = 0$
$x_3 = 1$
$x_3 = 0$
$\{\overline{x}_1 \vee x_2, x_3\}$
$\{\overline{x}_1 \vee \overline{x}_3\}$
$\{\overline{x}_1 \vee x_2, \overline{x}_1 \vee \overline{x}_3\}$
$x_4 = 0$
$x_4 = 1$
0
1
0
9
8
7
6
7

The efficiency of the branch-and-bound may essentially depend on

- the search strategy in the tree,
- the kind of building of tree by backtracking.

Suggestion: use priority over the generated partial solutions (Heap, Deap, etc.)

Exercises

1. (Set Cover Problem) Suppose that $S = \{1, 2, \cdots, n\}$ and $C \subset Powerset(S)$ is a collection of subsets of S. Write a Branch and Bound algorithm to find a $C' \subset C$ such that:

   $$\bigcup_{c \in C'} c = S$$

   , where $|C'|$ is minimum.
2. Suppose that $G = (V, E)$ is a graph. A Clique is a complete subgraph of G. A Max-Clique is a clique containing maximum vertices.
   a. Write a Backtracking algorithm to find a Max-Clique of G.
   b. Change the Backtracking algorithm to Branch and Bound algorithm to find a Max-Clique of G.