Exhaustive Generation: Backtracking and Branch-and-bound

Lucia Moura

Winter 2021

Backtracking begins...

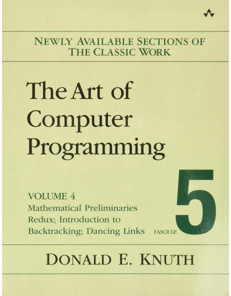
Nowhere to go but out,
Nowhere to come but back.
— BEN KING, in *The Sum of Life* (c. 1893)

Lewis back-tracked the original route up the Missouri.
— LEWIS R. FREEMAN, in *National Geographic Magazine* (1928)

When you come to one legal road that's blocked,
you back up and try another.
— PERRY MASON, in *The Case of the Black-Eyed Blonde* (1944)

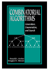
7.2.2. Backtrack Programming

Now that we know how to generate simple combinatorial patterns such as tuples, permutations, combinations, partitions, and trees, we're ready to tackle more exotic patterns that have subtler and less uniform structure. Instances of almost *any* desired pattern can be generated systematically, at least in principle, if we organize the search carefully. Such a method was christened "backtrack" by R. J. Walker in the 1950s, because it is basically a way to examine all fruitful possibilities while exiting gracefully from situations that have been fully explored.



Donald Knuth, The art of computer programming, Volume 4, Fascicle 5 (page 28 of 384)

Where are we on the textbook ?



Contents

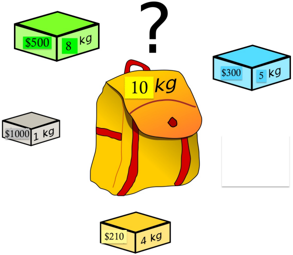
Knapsack Problem

Knapsack (Optimization) Problem

Instance: Profits $p_0, p_1, \ldots, p_{n-1}$
Weights $w_0, w_1, \ldots, w_{n-1}$
Knapsack capacity M

Find: and n-tuple $[x_0, x_1, \ldots, x_{n-1}] \in \{0,1\}^n$
such that $P = \sum_{i=0}^{n-1} p_i x_i$ is maximized,
subject to $\sum_{i=0}^{n-1} w_i x_i \leq M$.

Example



Objects:	1	2	3	4
weight (kg)	8	1	5	4
profit	$500	$1,000	$ 300	$ 210

Knapsack capacity: $M = 10$ kg.

Examples of feasible solutions and their profit:

x_1	x_2	x_3	x_4	profit
1	1	0	0	$ 1,500
0	1	1	1	$ 1,510

This problem is NP-hard.

Naive Backtracking Algorithm for Knapsack

Examine all 2^n tuples and keep the ones with maximum profit.

Global Variables $X, OptP, OptX$.
Algorithm $\textsc{Knapsack1}\ (l)$

    if $(l = n)$ then
      if $\sum_{i=0}^{n-1} w_i x_i \leq M$ then $CurP \leftarrow \sum_{i=0}^{n-1} p_i x_i$;
        if $(CurP > OptP)$ then
          $OptP \leftarrow CurP$;
          $OptX \leftarrow [x_0, x_1, \ldots, x_{n-1}]$;
    else $x_l \leftarrow 1$; $\textsc{Knapsack1}\ (l+1)$;
        $x_l \leftarrow 0$; $\textsc{Knapsack1}\ (l+1)$;

First call: $OptP \leftarrow -1$; $\textsc{Knapsack1}\ (0)$.

Running time: 2^n n-tuples are checked, and it takes $\Theta(n)$ to check each solution. The total running time is $\Theta(n2^n)$.

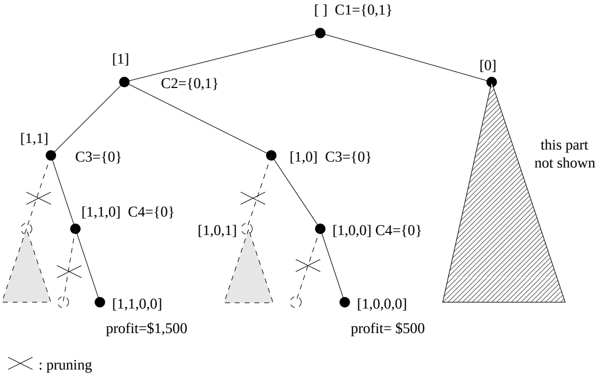
A General Backtracking Algorithm

- Represent a solution as a list: $X = [x_0, x_1, x_2, \ldots]$.
- Each $x_i \in P_i$ (possibility set)
- Given a partial solution: $X = [x_0, x_1, \ldots, x_{l-1}]$, we can use constraints of the problem to limit the choice of x_l to $\mathcal{C}_l \subseteq P_l$ (choice set).
- By computing $\mathcal{C}_l$ we prune the search tree, since for all $y \in P_l \setminus \mathcal{C}_l$ the subtree rooted on $[x_0, x_1, \ldots, x_{l-1}, y]$ is not considered.

Part of the search tree for the previous Knapsack example:

w_i	8	1	5	4
p_i	\$500	\$1,000	\$ 300	\$ 210

$M = 10.$



General Backtracking Algorithm with Pruning

Global Variables $X = [x_0, x_1, \ldots]$, $\mathcal{C}_l$, for $l = 0, 1, \ldots)$.

Algorithm $\textsc{Backtrack}\ (l)$
- if ($X = [x_0, x_1, \ldots, x_{l-1}]$ is a feasible solution) then
  - "Process it"
- Compute $\mathcal{C}_l$;
- for each $x \in \mathcal{C}_l$ do
  - $x_l \leftarrow x$;
  - $\textsc{Backtrack}(l + 1)$;

Backtracking with Pruning for Knapsack

Global Variables $X, OptP, OptX$.

Algorithm $\text{KNAPSACK2}\ (l, CurW)$

  if $(l = n)$ then if $(\sum_{i=0}^{n-1} p_i x_i > OptP)$ then

    $OptP \leftarrow \sum_{i=0}^{n-1} p_i x_i$;

    $OptX \leftarrow [x_0, x_1, \ldots, x_{n-1}]$;

  if $(l = n)$ then $\mathcal{C}_l \leftarrow \emptyset$

  else if $(CurW + w_l \leq M)$ then $\mathcal{C}_l \leftarrow \{0, 1\}$;

    else $\mathcal{C}_l \leftarrow \{0\}$;

  for each $x \in \mathcal{C}_l$ do

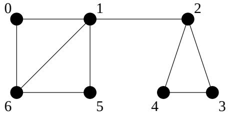
    $x_l \leftarrow x$;

    $\text{KNAPSACK2}\ (l + 1, CurW + w_l x_l)$;

First call: $\text{KNAPSACK2}\ (0, 0)$.

Backtracking: Generating all Cliques

PROBLEM: All Cliques
INSTANCE: a graph $G = (V, E)$.
FIND: all cliques of G without repetition



Cliques (and maximal cliques): $\emptyset, \{0\}, \{1\}, \ldots, \{6\}$,
$\{0,1\}, \{0,6\}, \underline{\{1,2\}}, \{1,5\}, \{1,6\}, \{2,3\}, \{2,4\}, \{3,4\}, \{5,6\}$,
$\underline{\{0,1,6\}}, \underline{\{1,5,6\}}, \underline{\{2,3,4\}}$.

Definition

Clique in $G(V, E)$: $C \subseteq V$ such that for all $x, y \in C$, $x \neq y$, $\{x, y\} \in E$.
Maximal clique: a clique not properly contained into another clique.

Many combinatorial problems can be reduced to finding cliques (or the largest clique):

- Largest independent set in G (stable set): is the same as largest clique in $\overline{G}$.

Many combinatorial problems can be reduced to finding cliques (or the largest clique):

- Largest independent set in G (stable set): is the same as largest clique in $\overline{G}$.
- Exact cover of sets by subsets: find clique with special property.

Many combinatorial problems can be reduced to finding cliques (or the largest clique):

- Largest independent set in G (stable set): is the same as largest clique in $\overline{G}$.
- Exact cover of sets by subsets: find clique with special property.
- Find a Steiner triple system of order v: find a largest clique in a special graph.

Many combinatorial problems can be reduced to finding cliques (or the largest clique):

- Largest independent set in G (stable set): is the same as largest clique in $\overline{G}$.
- Exact cover of sets by subsets: find clique with special property.
- Find a Steiner triple system of order v: find a largest clique in a special graph.
- Find a code with minimum distance d with maximum number of codewords.

Many combinatorial problems can be reduced to finding cliques (or the largest clique):

- Largest independent set in G (stable set): is the same as largest clique in $\overline{G}$.
- Exact cover of sets by subsets: find clique with special property.
- Find a Steiner triple system of order v: find a largest clique in a special graph.
- Find a code with minimum distance d with maximum number of codewords.
- Find all intersecting set systems: find all cliques in a special graph.

Many combinatorial problems can be reduced to finding cliques (or the largest clique):

- Largest independent set in G (stable set): is the same as largest clique in $\overline{G}$.
- Exact cover of sets by subsets: find clique with special property.
- Find a Steiner triple system of order v: find a largest clique in a special graph.
- Find a code with minimum distance d with maximum number of codewords.
- Find all intersecting set systems: find all cliques in a special graph.
- Etc.

In a Backtracking algorithm, $X = [x_0, x_1, \ldots, x_{l-1}]$ is a partial solution $\iff \{x_0, x_1, \ldots, x_{l-1}\}$ is a clique.
But we don't want ot get the same k-clique $k!$ times:
$[0, 1]$ extends to $[0, 1, 6]$
$[0, 6]$ extends to $[0, 6, 1]$

So we require partial solutions for be in sorted order:
$x_0 < x_1 < x_2 < \ldots < x_{l-1}$.

Let $S_{l-1} = \{x_0, x_1, \ldots, x_{l-1}\}$ for $X = [x_0, x_1, \ldots, x_{l-1}]$.
The **choice set** of this point is:
if $l = 0$ then $\mathcal{C}_0 = V$
if $l > 0$ then

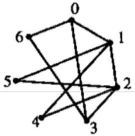
$$\begin{aligned}\mathcal{C}_l &= \{v \in V \setminus S_{l-1} : v > x_{l-1} \text{ and } \{v, x\} \in E \text{ for all } x \in S_{l-1}\} \\ &= \{v \in \mathcal{C}_{l-1} \setminus \{x_{l-1}\} : \{v, x_{l-1}\} \in E \text{ and } v > x_{l-1}\}\end{aligned}$$

So,
$\mathcal{C}_0 = V$
$\mathcal{C}_l = \{v \in \mathcal{C}_{l-1} \setminus \{x_{l-1}\} : \{v, x_{l-1}\} \in E \text{ and } v > x_{l-1}\}$, for $l > 0$

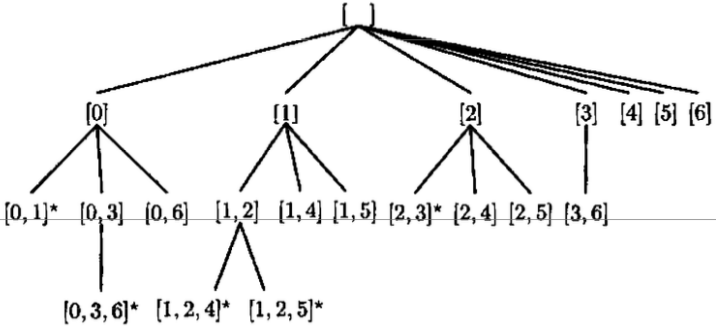
To compute $\mathcal{C}_l$, define:
$A_v = \{u \in V : \{u, v\} \in E\}$ (vertices adjacent to v)
$B_v = \{v+1, v+2, \ldots, n-1\}$ (vertices larger than v)
$\mathcal{C}_l = A_{x_{l-1}} \cap B_{x_{l-1}} \cap \mathcal{C}_{l-1}$.

To **detect if a clique is maximal** (set inclusionwise):
Calculate N_l, the set of vertices that can extend S_{l-1}:
$N_0 = V$
$N_l = N_{l-1} \cap A_{x_{l-1}}$.
S_{l-1} is maximal $\iff N_l = \emptyset$.

Generating all cliques



v	A_v	B_v
0	1, 3, 6	1, 2, 3, 4, 5, 6
1	0, 2, 4, 5	2, 3, 4, 5, 6
2	1, 3, 4, 5	3, 4, 5, 6
3	0, 2, 6	4, 5, 6
4	1, 2	5, 6
5	1, 2	6
6	0, 3	



Algorithm $\textsc{AllCliques}(l)$
Global: X, $\mathcal{C}_l(l = 0, \ldots, n-1)$, A_l, B_l pre-computed.

if $(l = 0)$ then output ([ ]);
    else output $([x_0, x_1, \ldots, x_{l-1}])$;
if $(l = 0)$ then $N_l \leftarrow V$;
    else $N_l \leftarrow A_{x_{l-1}} \cap N_{l-1}$;
if $(N_l = \emptyset)$ then output ("maximal");
if $(l = 0)$ then $\mathcal{C}_l \leftarrow V$;
    else $\mathcal{C}_l \leftarrow A_{x_{l-1}} \cap B_{x_{l-1}} \cap \mathcal{C}_{l-1}$;
for each $(x \in \mathcal{C}_l)$ do
    $x_l \leftarrow x$;
    $\textsc{AllCliques}(l+1)$;

First call: $\textsc{AllCliques}(0)$.

Average Case Analysis of AllCliques

Let G be a graph with n vertices and
let $c(G)$ be the number of cliques in G.

The running time for AllCliques for G is in $O(nc(G))$,
since $O(n)$ is an upper bound for the running time at a node,
and $c(G)$ is the number of nodes visited.

Let $\mathcal{G}_n$ be the set of all graphs on n vertices.
$|\mathcal{G}_n| = 2^{\binom{n}{2}}$ (bijection between $\mathcal{G}_n$ and all subsets of the set of unordered pairs of $\{1, 2, \ldots, n\}$).

Assume the graphs in $\mathcal{G}_n$ are equally likely inputs for the algorithm (that is, assume uniform probability distribution on $\mathcal{G}_n$).
Let $T(n)$ be the average running time of AllCliques for graphs in $\mathcal{G}_n$.
We will calculate $T(n)$.

$T(n) =$ the average running time of AllCliques for graphs in $\mathcal{G}_n$.
Let $\overline{c}(n)$ be the average number of cliques in a graph in $\mathcal{G}_n$.

Then, $T(n) \in O(n\overline{c}(n))$.

So, all we need to do is estimating $\overline{c}(n)$.

$$\overline{c}(n) = \frac{\sum_{G \in \mathcal{G}_n} c(G)}{|\mathcal{G}_n|} = \frac{1}{2^{\binom{n}{2}}} \sum_{G \in \mathcal{G}_n} c(G).$$

We will show that:

$$\overline{c}(n) \leq (n+1)n^{\log_2 n}, \text{ for } n \geq 4.$$

Skeetch of the Proof:

Define the indicator function, for each sunset $W \subseteq V$:

$$\mathcal{X}(G,W) = \begin{cases} 1, & \text{if } W \text{ is a clique of } G \\ 0, & \text{otherwise} \end{cases}$$

Then,

$$\begin{aligned} \overline{c}(n) &= \frac{1}{2^{\binom{n}{2}}} \sum_{G \in \mathcal{G}_n} c(G) \\ &= \frac{1}{2^{\binom{n}{2}}} \sum_{G \in \mathcal{G}_n} \left(\sum_{W \subseteq V} \mathcal{X}(G,W) \right) \\ &= \frac{1}{2^{\binom{n}{2}}} \sum_{W \subseteq V} \sum_{G \in \mathcal{G}_n} \mathcal{X}(G,W) \end{aligned}$$

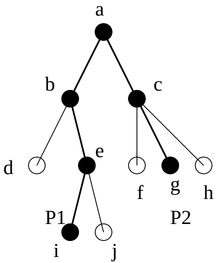
Now, for fixed W, $\sum_{G \in \mathcal{G}_n} \mathcal{X}(G, W) = 2^{\binom{n}{2} - \binom{|W|}{2}}$.
(Number of subsets of $\binom{V}{2}$ containing edges of W)

$$\begin{aligned} \overline{c}(n) &= \frac{1}{2^{\binom{n}{2}}} \sum_{W \subseteq V} 2^{\binom{n}{2} - \binom{|W|}{2}} \\ &= \frac{1}{2^{\binom{n}{2}}} \sum_{k=0}^{n} \binom{n}{k} 2^{\binom{n}{2} - \binom{k}{2}} = \sum_{k=0}^{n} \frac{\binom{n}{k}}{2^{\binom{k}{2}}}. \end{aligned}$$

So, $\overline{c}(n) = \sum_{k=0}^{n} t_k$, where $t_k = \frac{\binom{n}{k}}{2^{\binom{k}{2}}}$.
A technical part of the proof bounds t_k as follows: $t_k \leq n^{\log_2 n}$
(see the textbook for details)
So, $\overline{c}(n) = \sum_{k=0}^{n} t_k \leq \sum_{k=0}^{n} n^{\log_2 n} = (n+1) n^{\log_2 n} \in O(n^{\log_2 n + 1})$.
Thus, $T(n) \in O(n \overline{c}(n)) \subseteq O(n^{\log_2 n + 2})$.

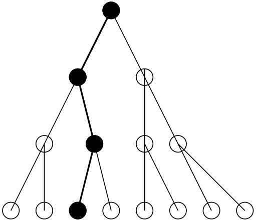
Estimating the size of a Backtrack tree

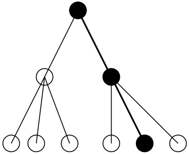
State Space Tree: tree size $= 10$



Probing path P_1:
Estimated tree size: $N(P_1) = 15$

Probing path P_2:
Estimated tree size: $N(P_2) = 9$





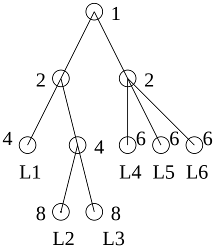
Probing path P_1:
Estimated tree size: $N(P_1) = 15$

Probing path P_2:
Estimated tree size: $N(P_2) = 9$

Game for chosing a path (probing):
At each node of the tree, pick a child node uniformly at random.
For each leaf L, calculate $P(L)$, the probability that L is reached.
We will prove later that the expected value of $\overline{N}$ of $N(L)$ turns out to be the size of the space state tree. Of course,

$$\overline{N} = \sum_{L \text{ leaf}} P(L)N(L) \qquad \text{(by definition)}$$

In the previous example, consider T (number is estimated number of nodes at this level)



$P(L_1) = 1/4$, $P(L_2) = P(L_3) = 1/8$, $P(L_4) = P(L_5) = P(L_6) = 1/6$
$N(L_1) = 1 + 2 + 4 = 7$ $N(L_2) = N(L_3) = 1 + 2 + 4 + 8 = 15$
$N(L_4) = N(L_5) = N(L_6) = 1 + 2 + 6 = 9$

$$\overline{N} = \sum_{i=1}^{6} P(L_i)N(L_i) = \frac{1}{4} \times 7 + 2 \times (\frac{1}{8} \times 15) + 3 \times (\frac{1}{6} \times 9) = 10 = |T|$$

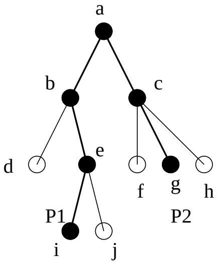
In practice, to **estimate** $\overline{N}$, do k probes $L_1, L_2, \ldots, L_k$, and calculate the average of $N(L_i)$:

$$N_{est} = \frac{\sum_{i=1}^{k} N(L_i)}{k}$$

Algorithm $\textsc{EstimateBacktrackSize}()$

$s \leftarrow 1$; $N \leftarrow 1$; $l \leftarrow 0$;
Compute $\mathcal{C}_0$;
while $\mathcal{C}_l \neq \emptyset)$ do
  $c \leftarrow |\mathcal{C}_l|$;
  $s \leftarrow c * s$;
  $N \leftarrow N + s$;
  $x_l \leftarrow$ a random element of $\mathcal{C}_l$;
  Compute $\mathcal{C}_{l+1}$ for $[x_0, x_1, \ldots, x_l]$;
  $l \leftarrow l + 1$;
return N;

In the example below, doing only 2 probes:



P_1:	l	$\mathcal{C}_l$	c	x_l	s	N
					1	1
	0	b, c	2	b	2	3
	1	d, e	2	e	4	7
	2	i, j	2	i	8	$\underline{15}$
	3	$\emptyset$				

P_1:	l	$\mathcal{C}_l$	c	x_l	s	N
					1	1
	0	b, c	2	c	2	3
	1	f, g, h	3	g	6	$\underline{9}$
	2	$\emptyset$				

Theorem

For a state space tree T, let P be the path probed by the algorithm ESTIMATEBACKTRACKSIZE.
If $N = N(P)$ is the value returned by the algorithm, then the expected value of N is $|T|$.

Proof.
Define the following function on the nodes of T:

$$S([x_0, x_1, \ldots, x_{l-1}]) = \begin{cases} 1, & \text{if } l = 0 \\ |\mathcal{C}_{l-1}| \times S([x_0, x_1, \ldots, x_{l-2}]) & \end{cases}$$

($s \leftarrow c * s$ in the algorithm)
The algorithm computes: $N(P) = \sum_{Y \in P} S(Y)$.

$P = P(X)$ is a path in T from root to leaf X, say $X = [x_0, x_1, \ldots, x_{l-1}]$.
Call $X_i = [x_0, x_1, \ldots, x_i]$.
The probability that $P(X)$ chosen is:

$$\frac{1}{|\mathcal{C}_0(x_0)|} \times \frac{1}{|\mathcal{C}_1(x_1)|} \times \ldots \times \frac{1}{|\mathcal{C}_{l-1}(x_{l-1})|} = \frac{1}{S(X)}.$$

So,

$$\begin{aligned}
\overline{N} &= \sum_{X \in \mathcal{L}(T)} prob(P(X)) \times N(P(X)) \\
&= \sum_{X \in \mathcal{L}(T)} \frac{1}{S(X)} \sum_{Y \in P(X)} S(Y) \\
&= \sum_{Y \in T} \sum_{\{X \in \mathcal{L}(T) : Y \in P(X)\}} \frac{S(Y)}{S(X)} \\
&= \sum_{Y \in T} S(Y) \sum_{\{X \in \mathcal{L}(T) : Y \in P(X)\}} \frac{1}{S(X)}
\end{aligned}$$

We claim that: $\sum_{\{X \in \mathcal{L}(T): Y \in P(X)\}} \frac{1}{S(X)} = \frac{1}{S(Y)}$.

Proof of the claim:
Let Y be a non-leaf. If Z is a child of Y and Y has c children, then $S(Z) = c \times S(Y)$.
So,

$$\sum_{\{Z:Z \text{ is a child of } Y\}} \frac{1}{S(Z)} = c \times \frac{1}{c \times S(Y)} = \frac{1}{S(Y)}$$

Iterating this equation until all Z's are leafs:

$$\frac{1}{S(Y)} = \sum_{\{X:X \text{ is a leaf descendant of } Y\}} \frac{1}{S(X)}$$

So the claim is proved!

Thus,

$$\begin{aligned} \overline{N} &= \sum_{Y \in T} S(Y) \sum_{\{X \in \mathcal{L}(T): Y \in P(X)\}} \frac{1}{S(X)} \\ &= \sum_{Y \in T} S(Y) \frac{1}{S(Y)} \\ &= \sum_{Y \in T} 1 = |T|. \end{aligned}$$

The theorem is thus proved!

Exact Cover

PROBLEM: Exact Cover

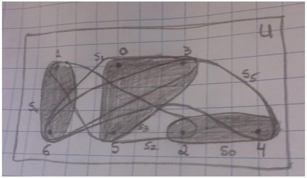
INSTANCE: a collection $\mathcal{S}$ of subsets of $\mathcal{U} = \{0, 1, \ldots, n-1\}$.

QUESTION: Does $\mathcal{S}$ contain an exact cover of $\mathcal{U}$

Rephrasing the question: Does there exist $\mathcal{S}' = \{S_{x_0}, S_{x_1}, \ldots, S_{x_{l-1}}\} \subseteq \mathcal{S}$ such that every element of $\mathcal{U}$ is contained in exactly one set of $\mathcal{S}'$?

Example: $\mathcal{U} = \{0, 1, 2, 3, 4, 5, 6\}$
$\mathcal{S} = \{S_0 = \{2, 4\}, S_1 = \{0, 3, 6\}, S_2 = \{1, 2, 5\}, S_3 = \{0, 3, 5\}, S_4 = \{1, 6\}, S_5 = \{3, 4, 6\}\}$

Solution: yes, $x = [0, 3, 4]$



matrix form representation:

	0	1	2	3	4	5	6
S_0	0	0	1	0	1	0	0
S_1	1	0	0	1	0	0	1
S_2	0	1	1	0	0	1	0
S_3	1	0	0	1	0	1	0
S_4	0	1	0	0	0	0	1
S_5	0	0	0	1	1	0	1

Exact Cover

PROBLEM: Exact Cover
INSTANCE: a collection $\mathcal{S}$ of subsets of $\mathcal{U} = \{0, 1, \ldots, n-1\}$.
QUESTION: Does $\mathcal{S}$ contain an exact cover of $\mathcal{U}$

Rephrasing the question: Does there exist $\mathcal{S}' = \{S_{x_0}, S_{x_1}, \ldots, S_{x_{l-1}}\} \subseteq \mathcal{S}$ such that every element of $\mathcal{U}$ is contained in exactly one set of $\mathcal{S}'$?

Transforming into a clique problem:

$\mathcal{S} = \{S_0, S_1, \ldots, S_{m-1}\}$

Define: $G(V, E)$ in the following way: $V = \{0, 1, , \ldots, m-1\}$

$\{i, j\} \in E \iff S_i \cap S_j = \emptyset$

An exact cover of $\mathcal{U}$ is a clique of G that covers $\mathcal{U}$.

$\mathcal{U} = \{0, 1, 2, 3, 4, 5, 6\}$
$\mathcal{S} = \{S_0 = \{2, 4\}, S_1 = \{0, 3, 6\}, S_2 = \{1, 2, 5\}, S_3 = \{0, 3, 5\}, S_4 = \{1, 6\}, S_5 = \{3, 4, 6\}\}$

$G = (V, E)$, where $V = \{0, 1, 2, 3, 4, 5\}$ $E = \{\{0, 1\}, \{0, 3\}, \{0, 4\}, \{1, 2\}, \{2, 5\}, \{3, 4\}\}$

Example of exact cover problems

Good ordering on $\mathcal{S}$ for prunning:
$\mathcal{S}$ sorted in decreasing lexicographical ordering.
Choice set:

$$
\begin{aligned}
\mathcal{C}'_0 &= V \\
\mathcal{C}'_l &= A_{x_{l-1}} \cap B_{x_{l-1}} \cap \mathcal{C}'_{l-1}, \text{ if } l > 0,
\end{aligned}
$$

where

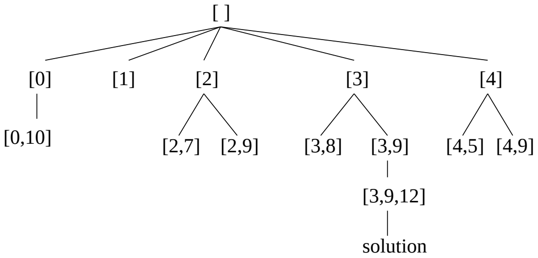
$$
\begin{aligned}
A_x &= \{y \in V : S_y \cap S_x = \emptyset\} \quad \text{(vertices adjacent to } x\text{)} \\
B_x &= \{y \in V : S_x >_{lex} S_y\}
\end{aligned}
$$

Further pruning will be used to reduce $\mathcal{C}'_l$ by removing H_r's, which will be defined later.

Example: (corrected from book page 121)

j	S_j	rank(S_j)	$A_j \cap B_j$	corrected?
0	0,1,3,	104	10	Y
1	0,1,5	98	12	
2	0,2,4	84	7,9	Y
3	0,2,5	82	8,9,12	Y
4	0,3,6	73	5,9	Y
5	1,2,4	52	$\emptyset$	
6	1,2,6	49	11	Y
7	1,3,5	42	$\emptyset$	Y
8	1,4,6	37	$\emptyset$	
9	1	32	10,11,12	
10	2,5,6	19	$\emptyset$	
11	3,4,5	14	$\emptyset$	
12	3,4,6	13	$\emptyset$	

i	0	1	2	3	4	5	6
H_i	0,1,2,3,4	5,6,7,8,9	10	11,12	$\emptyset$	$\emptyset$	$\emptyset$



$\textsc{ExactCover}\ (n, \mathcal{S})$

Global X, $\mathcal{C}_l$, $l = (0, 1, \ldots)$
Procedure $\textsc{ExactCoverBT}(l, r')$
if $(l = 0)$ then $U_0 \leftarrow \{0, 1, \ldots, n-1\}$;
$r \leftarrow 0$;
else $U_l \leftarrow U_{l-1} \setminus S_{x_{l-1}}$;
$r \leftarrow r'$;
while $(r \notin U_l)$ and $(r < n)$ do $r \leftarrow r + 1$;
if $(r = n)$ then output $([x_0, x_1, \ldots, x_{l-1}])$.
if $(l = 0)$ then $\mathcal{C}'_0 \leftarrow \{0, 1, \ldots, m-1\}$;
else $\mathcal{C}'_l \leftarrow A_{x_{l-1}} \cap B_{x_{l-1}} \cap \mathcal{C}'_{l-1}$;
$\mathcal{C}_l \leftarrow \mathcal{C}'_l \cap H_r$;
for each $(x \in \mathcal{C}_l)$ do
$x_l \leftarrow x$;
$\textsc{ExactCoverBT}(l + 1, r)$;

Main

$m \leftarrow |\mathcal{S}|$;
Sort $\mathcal{S}$ in decreasing lexico order
for $i \leftarrow 0$ to $m-1$ do
  $A_i \leftarrow \{j : S_i \cap S_j = \emptyset\}$;
  $B_i \leftarrow \{i+1, i+2, \ldots, m-1\}$;
for $i \leftarrow 0$ to $n-1$ do
  $H_i \leftarrow \{j : S_j \cap \{0, 1, \ldots, i\} = \{i\}\}$;
$H_n \leftarrow \emptyset$;
$\textsc{ExactCoverBT}(0, 0)$;

(U_i contains the uncovered elements at level i.
r is the smallest uncovered in U_i.)

Backtracking with bounding

When applying backtracking for an **optimization** problem, we use **bounding** for prunning the tree.
Let us consider a **maximization** problem.
Let $\text{profit}(X)$ = profit for a feasible solution X.
For a partial soluion $X = [x_0, x_1, \ldots, x_{l-1}]$, define

$$P(X) = \max \quad \{ \quad \text{profit}(X') : \text{ for all feasible solutions } X' = [x_0, x_1, \ldots, x_{l-1}, x'_l, \ldots, x'_{n-1}] \}.$$

A **bounding function** B is a real valued function defined on the nodes of the space state tree, such that for any feasible solution X, $B(X) \geq P(X)$.
$B(X)$ is an upper boud on the profit of any feasible solution that is descendant of X in the state space tree.
If the current best solution found has value $OptP$, then we can prune nodes X with $B(X) \leq OptP$, since $P(X) \leq B(X) \leq OptP$, that is, no descendant of X will improve on the current best solution.

General Backtracking with Bounding

Algorithm $\textsc{BackTrackBounding}(l)$

Global X, $OptP$, $OptX$, $\mathcal{C}_l$, $l = (0, 1, \ldots)$

if ($[x_0, x_1, \ldots, x_{l-1}]$ is a feasible solution) then

  $P \leftarrow \mathrm{profit}([x_0, x_1, \ldots, x_{l-1}])$;

  if ($P > OptP$) then

    $OptP \leftarrow P$;

    $OptX \leftarrow [x_0, x_1, \ldots, x_{l-1}]$;

Compute $\mathcal{C}_l$;

$B \leftarrow B([x_0, x_1, \ldots, x_{l-1}])$;

for each $(x \in \mathcal{C}_l)$ do

    if $B \leq OptP$ then return;

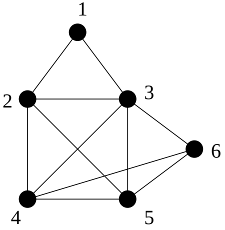
    $x_l \leftarrow x$;

    $\textsc{BackTrackBounding}(l + 1)$

Maximum Clique Problem

PROBLEM:	Maximum Clique (optimization)
INSTANCE:	a graph $G = (V, E)$.
FIND:	a maximum clique of G.

This problem is NP-complete.



Maximum cliques:

{2,3,4,5}, {3,4,5,6}

Modification of AllCliques to find a maximum clique (no bounding).
Blue adds **bounding** to this algorithm.

Algorithm $\textsc{MaxClique}(l)$
Global: X, $\mathcal{C}_l (l = 0, \ldots, n-1)$, A_l, B_l pre-computed.

if $(l > OptSize)$ then
  $OptSize \leftarrow l$;
  $OptClique \leftarrow [x_0, x_1, \ldots, x_{l-1}]$;
if $(l = 0)$ then $\mathcal{C}_l \leftarrow V$;
  else $\mathcal{C}_l \leftarrow A_{x_{l-1}} \cap B_{x_{l-1}} \cap \mathcal{C}_{l-1}$;
$\mathbf{M} \leftarrow \mathbf{B}([\mathbf{x_0}, \mathbf{x_1}, \ldots, \mathbf{x_{l-1}}])$;
for each $(x \in \mathcal{C}_l)$ do
  if $(\mathbf{M} \leq \mathbf{OptSize})$ **then return;**
  $x_l \leftarrow x$; $\textsc{MaxClique}(l+1)$;

Main
  $OptSize \leftarrow 0$; $\textsc{MaxClique}(0)$;
  output $OptClique$;

Bounding Functions for MaxClique

Definition

Induced Subgraph

Let $G = (V, E)$ and $W \subseteq V$. The subgraph induced by W, $G[W]$, has vertex set W and edgeset: $\{\{u, v\} \in E : u, v \in W\}$.

If we have:
partial solution: $X = [x_0, x_1, \ldots, x_{l-1}]$ with choice set $\mathcal{C}_l$,
extension solution $X = [x_0, x_1, \ldots, x_{l-1}, x_l, \ldots, x_j]$,
Then $\{x_l, \ldots, x_j\}$ must be a clique in $G[\mathcal{C}_l]$.
Let $mc(l)$ denote the size of a maximum clique in $G[\mathcal{C}_l]$, and let $ub(l)$ be an upper bound on $mc(l)$.
Then, a general bounding function is $B(X) = l + ub[l]$.

Bound based on size of subgraph

General bounding function: $B(X) = l + ub[l]$.

Since $mc(l) \leq |\mathcal{C}_l|$, we derive the bound:

$$B_1(X) = l + |\mathcal{C}_l|.$$

Bounds based on colouring

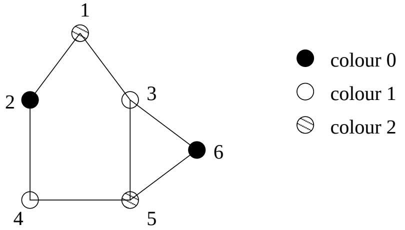
Definition (Vertex Colouring)

Let $G = (V, E)$ and k a positive integer. A (vertex) k-colouring of G is a function

$$\text{COLOR}: V \to \{0, 1, \ldots, k-1\}$$

such that, for all $\{x, y\} \in E$, $\text{COLOR}(x) \neq \text{COLOR}(y)$.

Example: a 3-colouring of a graph:



Lemma

If G has a k-colouring, then the maximum clique of G has size at most k.

Proof. Let C be a clique. Each $x \in C$ must have a distinct colour. So, $|C| \leq k$. This is true for any clique, in particular for the maximum clique.

Finding the minimum colouring gives the best upper bound, but it is a hard problem. We will use a **greedy heuristic** for finding a small colouring.
Define $\text{ColourClass}[h] = \{i \in V : \text{Colour}[i] = h\}$.

```
GreedyColour(G = (V, E))
    Global Colour
    k ← 0; // colours used so far
    for i ← 0 to n − 1 do
        h ← 0;
        while (h < k) and (A_i ∩ ColourClass[h] ≠ ∅) do
            h ← h + 1;
        if (h = k) then k ← k + 1;
                        ColourClass[h] ← ∅;
        ColourClass[h] ← ColourClass[h] ∪ {i};
        Colour[i] = h;
    return k;
```

Sampling Bound:
Statically, beforehand, run $\textsc{GreedyColour}(G)$, determining k and $\textsc{Colour}[x]$ for all $x \in V$.

$\textsc{Sampling}$Bound$(X = [x_0, x_1, \ldots, x_{l-1}])$
Global $\mathcal{C}_l$, $\textsc{Colour}$
return $l + |\{\textsc{Colour}[x] : x \in \mathcal{C}_l\}|$;

Greedy Bound:
Call $\textsc{GreedyColour}$ dynamically.

$\textsc{Greedy}$Bound$(X = [x_0, x_1, \ldots, x_{l-1}])$
Global $\mathcal{C}_l$
$k \leftarrow \textsc{GreedyColour}(G[\mathcal{C}_l])$;
return $l + k$;

Number of nodes of the backtracking tree: random graphs with edge density 0.5

# vertices	50	100	150	200	250
# edges	607	2535	5602	9925	15566
max clique size	7	9	10	11	11
bounding function:					
none	8687	257145	1659016	7588328	26182672
size bound	3202	57225	350310	1434006	5008757
sampling bound	2268	44072	266246	1182514	4093535
greedy bound	430	5734	22599	91671	290788

Branch-and-bound

The book presents branch-and-bound as a variation of backtracking in which the choice set is tried in decreasing order of bounds.

However, branch-and-bound is usually a more general scheme.

It often involves keeping all active nodes in a priority queue, and processing nodes with higher priority first (priority is given by upper bound).

Next we look at the book's version of branch-and-bound.

Algorithm $\textsc{BranchAndBound}(l)$

external $B()$, $\textsc{profit}()$; global $\mathcal{C}_l$ $(l = 0, 1, \ldots)$
if $([x_0, x_1, \ldots, x_{l-1}]$ is a feasible solution) then
  $P \leftarrow \textsc{Profit}([x_0, x_1, \ldots, x_{l-1}])$
  if $(P > OptP)$ then $OptP \leftarrow P$;
                $OptX \leftarrow [x_0, x_1, \ldots, x_{l-1}]$;
Compute $\mathcal{C}_l$; $count \leftarrow 0$;
for each $(x \in \mathcal{C}_l)$ do
  $nextchoice[count] \leftarrow x$;
  $nextbound[count] \leftarrow B([x_0, x_1, \ldots, x_{l-1}, x])$;
  $count \leftarrow count + 1$;
Sort $nextchoice$ and $nextbound$ by decreasing order of $nextbound$;
for $i \leftarrow 0$ to $count - 1$ do
  if $(nextbound[i] \leq OptP)$ then return;
  $x_l \leftarrow nextchoice[i]$;
  $\textsc{BranchAndBound}(l + 1)$;