Exhaustive Search Techniques

There are many practical problems that no one has efficient (= poly.time) algorithms for.

- e.g. 0-1 knapsack, TSP, independent set in graph, shortest path in a graph with negative weights (no repeated nodes)

Options:

- heuristics - run quickly but no guarantee on quality of solution
- approximation algorithms - guarantee quality of solution e.g. length of TSP ≤ 2 • min length of TSP
- exact solutions - take exponential time

Note that to experiment with heuristics and approx. algs, we need exact algs.

Backtracking - a systematic way to try all possible solutions like searching in an implicit graph of partial solutions used for decision problems (we'll deal with optimization later.)

Example: Subset Sum (knapsack with value = weight)

Given elements 1...n with weights w₁...wₙ and target weight W

Is there a subset S ⊆ {1,...,n} st. \( \sum_{i \in S} w_i = W \)?
Fact: this problem is $NP$-complete (pf. later).

No one knows a poly. time alg.

Best we can do is explore all subsets.

\[ S = \emptyset \quad R = \{1, \ldots, n\} \]
\[ S = \{1, \ldots, i\} \quad R = \{i+1, \ldots, n\} \]

we'll say what to do with weights soon!

General configuration: \[ C = S, R \]
\[ S = \{1, \ldots, i\} \quad R = \{i+1, \ldots, n\} \]

Two children - put $i$ in or out

General Backtracking Algorithm:

1. $C$ - set of active configurations
2. initially - original config. (e.g. $S = \emptyset, R = 1 \ldots n$)
3. while $C \neq \emptyset$
   1. $C \leftarrow$ remove config. from $C$.
   2. \text{//explore $C$}
   3. if $C$ solves problem - DONE \textbf{SUCCESS}
   4. if $C$ is a dead end - discard it
   5. else expand $C$ to $C_1 \ldots C_t$ by making additional choices and add each $C_i$ to $C$.}

end
Store $A$ as stack — DFS of config. space
size of $A$ = height of tree
Store $Q$ as queue — BFS of config. space
size of $Q$ = width of tree
To reduce space, use DFS
eg. for subset sum, width is $2^n$, height is $n$

Note: might also explore “most promising” config. first — use priority queue.

Back to Subset Sum
How to explore a config. $S, R$?
Keep $w = \sum_{i \in S} w_i$ just update these as we go.
$\quad r = \sum_{i \in R} w_i$

Then
If $w = W$ — success (solved problem)
if $w > W$ — dead end (don’t expand this)
if $r + w < W$ — dead end

Running time $O(2^n)$

This is also a dynamic prog alg. with run time $O(n \cdot W)$
Which is better? Depends! If $W$ has $n$ bits then backtracking is better.
Above we used backtracking to explore all subsets. Can also explore all permutations of $1 \ldots n$.

$P = \emptyset$ \hspace{1cm} $R = \{1, \ldots, n\}$

$P = \langle 1 \rangle$ \hspace{1cm} $R = \{2, \ldots, n\}$ \hspace{1cm} $n$ children

$P = \langle 1, 2 \rangle$ \hspace{1cm} $P = \langle 1, n \rangle$ \hspace{1cm} $P = \langle n, 1 \rangle$ \hspace{1cm} $P = \langle n, n-1 \rangle$ \hspace{1cm} $n-1$ children

$P = \langle 1, 2, \ldots, n \rangle$ \hspace{1cm} $P = \langle n, n-1, \ldots, 1 \rangle$

There are $n!$ leaves

Config. $C = \{P \mid$ permutation of length $i$ \hspace{1cm} $R \mid$ remaining elements $\}$
Branch and Bound

- for optimization problems (say minimize)
  (backtracking was for decision problems)
- not DFS, but explore most promising config. first
- keep min so far
- "branch" - generate children
- "bound" - compute lower bound \( l_c \) on objective and prune a config, \( C \) if \( l_c > \) current min.

General Branch and Bound Algorithm

\( A \) - set of active configurations
initially the original config.
best-soln, best-cost - best so far.
while \( A \neq \emptyset \)

\[ C \leftarrow \text{remove "most promising" config. from } A \]
\[ \text{expand } C \text{ to } C_1 \ldots C_t \text{ by making additional choices } \text{BRANCH} \]
for \( i = 1 \ldots t \)
if \( C_i \) solves whole problem
then if cost \( (C_i) < \) best-cost then
  update best
else if \( C_i \) is dead end then discard it
else if lower bound \( (C_i) < \) best-cost \text{BOUND}
  then add \( C_i \) to \( A \)
end
Note that we test $G_i$ when it is generated (rather than when it is removed from $A$) could have done this for backtracking too.

**Example:** Travelling Salesman Problem.

Given graph $G = (V, E)$ (undirected) and non-neg. weights $w : E \rightarrow \mathbb{R}^{\geq 0}$ find a cycle $C$ that goes through every vertex exactly once and has min. weight $\sum_{e \in C} w(e)$ called a TSP tour.

This is a famous NP-complete problem. There is a book about it. An expert is Prof. Cook, C&O.

There are contests to solve big instances.

2004 - 24,978 cities in Sweden
2006 - 85,900 VLSI input

---

**Branch and bound algorithm**

* based on enumerating all subsets of edges

* Configuration $C$: $I \subseteq E$ - edges included in tour
  $X \subseteq E$ - edges excluded from tour
  with $I \cap X = \emptyset$

  e.g. $\begin{array}{c} \boxed{a} \boxed{b} \\ \boxed{c} \boxed{d} \end{array}$ if $X = \overline{(a, b)}$ then only TSP tour is $a \rightarrow b \rightarrow c \rightarrow d$

  if $X = \overline{(a, b)}$ and $I = \overline{(c, d)}$ there is no solution

**Necessary conditions** - used to detect dead ends

* $E - X$ must be connected, actually biconnected

* $I$ must have $\geq 2$ edges incident to each vertex

* $I$ contains no cycle (except on all vertices)

http://www.math.uwaterloo.ca/tsp/index.html
How to branch

\[ C = (I, X) \]

choose \( e \in E \setminus (I \cup X) \)

How to bound - want to compute lower bound efficiently

One idea based on MST:

A relaxed problem:

\[ \text{1-tree} = \text{spanning tree on vertices 2\ldots n plus} \]
\[ 2 \text{ edges from vertex 1} \]

Claim Any TSP tour is a 1-tree

Thus \( w(\min \text{TSP}) \geq w(\min \text{1-tree}) \)

so this provides a lower bound
We can find a min 1-tree efficiently

- even given X (excluded edges) — throw them out
- I (included edges) — give them weight 0
- just find MST on \( \frac{n}{2}, \ldots, n \)
- and add 2 min weight edges incident to 1.

Can now use general branch and bound algorithm

lower bound for config. \( C \) = min 1-tree for \( C \)

Enhancements

- choose "most promising" config. — the one with
  - the min. weight 1-tree
- branch wisely
  
  e.g. find vertex \( i \) in min 1-tree of deg. \( \geq 2 \)
  and let \( e = \max \) weight edge \((i,j)\) in
  1-tree but not in I U X.

\[
\begin{align*}
  i &\quad j \\
  e &\quad (i, j)
\end{align*}
\]

This plus further enhancements lead to competitive
TSP algorithms.