Solving Hard Problems

We need to solve NP-hard problems, too

We can apply heuristics, which may allow some solve relatively large instances to be solved, or efficiently computable approximate algorithms.

First consider heuristics called "Branch-and-bound"

"Branch-and-bound" uses a bound of estimated cost for pruning the search space.

In a minimization problem we search for a solution $v$ whose cost $c(v)$ is minimal.

There B&B applies a lower bound $b()$, which assigns to each partial solution $u$ an estimate such that

$$b(u) \leq c(v)$$

for each complete solution $v$ that contains $u$.

If $c$ is the cost of some candidate solution, there is no need to continue to complete a partial solution $u$ if $b(u) \geq c$.

"Branch-and-bound"

- Heuristics for solving difficult optimization problems.

Example: TSP

- Variation of breadth-first search, to prune the search space, using a bound on the objective function.

Consider optimization problems whose solution $s$ has a positive cost $c(s) > 0$.

In hard problems, the search space of (partial) solutions is super-polynomial wrt input size.

Example ("Branch and bound" for TSP)

The length of a TSP route is at least $\lceil S/2 \rceil$, where $S$ is the sum length of two shortest edges adjacent to each vertex:

$$\left\lceil (1 + 3) + (3 + 6) + (1 + 2) + (3 + 4) + (2 + 3) \right\rceil / 2 = 14$$

Similar lower bound for routes that contain edges $(a, b), (b, d)$ and $(b, d)$:

$$\left\lceil (1 + 3) + (3 + 7) + (1 + 2) + (3 + 7) + (2 + 3) \right\rceil / 2 = 16$$
Can fix the start (say, a) and direction (say, b before c) \( \sim \frac{41}{2} = 12 \) candidate solutions

Generate them, applying the lower bound to prune suboptimal ones (and to extend partial solutions in a "best first" order):

The optimal route \( (c(a, b, d, e, c, a) = 16) \) is found by examining 1/3 of candidates

Maximization \( \rightarrow \) prune using an upper bound

Upper bound:

Process items in decreasing order of their unit value \( a_i/t_i \)

Let \( u \) be a selection of items \( \{1, \ldots, i\} \) with total volume \( t \) and value \( a \). Then its extensions can have value at most

\[
b(u) = a + (T - t)(a_{i+1}/t_{i+1})
\]

**Example** (Knapsack with B&B)

Task: Pack maximally valuable load of given items \( 1, \ldots, n \), in volume \( T \)

Candidate solutions:
Subsets of items, whose total volume \( \leq T \)

Search space:
Item subsets, generated in increasing order

Can arrange as a binary tree:

Root \( \sim \emptyset \)

Left (right) child of a node at level \( i \)
\( \sim \) include (exclude) item \( i + 1 \)

**Example** \( T = 10; \) item vol value \( a_i/t_i \)

\[
\begin{array}{cccc}
1 & 4 & 40 & 10 \\
2 & 7 & 42 & 6 \\
3 & 5 & 25 & 5 \\
4 & 3 & 12 & 4 \\
\end{array}
\]

B&B search (with "best-first" heuristic):
Approximation algorithms

Approximate solutions can be computed efficiently for some optimization problems — but for some other problems approximation seems essentially as difficult (NP-hard) as finding the exact optimum.

Example (Approximated vertex cover)

A vertex cover for a graph is a subset of its vertices which includes at least one endpoint of each edge.

Finding a minimal vertex cover is NP-hard.

An approximative solution:

\[ S := \emptyset; \]
\[ \textbf{while } E \neq \emptyset \textbf{ do} \]
\[ \text{Select some edge } \{u,v\} \in E; \]
\[ S := S \cup \{u,v\}; \]
\[ \text{Delete from } E \text{ any edges that are adjacent to } u \text{ or } v; \]
\[ \textbf{return } S; \]

Algorithm finds a vertex cover in linear time.

How far is \( S \) from an optimal cover \( S^* \)?

\( S^* \) contains at least one endpoint of each edge \( \{u,v\} \) selected by the algorithm \( \sim \)
\[ |S|/2 \leq |S^*| \]

Accuracy of Approximation

The relative error of an approximated solution \( s_a \) wrt an optimal solution \( s^* \):
\[ |c(s_a) - c(s^*)|/c(s^*) \]

in minimization \( c(s_a) \geq c(s^*) \)
in maximization \( c(s_a) \leq c(s^*) \)

An approximation algorithm is an \( \epsilon \)-approximation, if its solutions \( s_a \) satisfy
\[ |c(s_a) - c(s^*)|/c(s^*) \leq \epsilon \]

For example, 0.1-approximation

in minimization:

in maximization:

TSP can be approximated efficiently in metric graphs, where each \( u, v \) and \( r \) satisfy
\[ d(u, v) \leq d(u, r) + d(r, v) \]

1. \( T \leftarrow \) minimal spanning tree;
2. \( P \leftarrow \) path traversing edges of \( T \) twice;
3. \( C \leftarrow P \) bypassing nodes already visited;

Accuracy vs. optimal path length \( l(C^*) \)?
\[ l(C) \leq l(P) = 2l(T) \leq 2l(C^*) \]
Approximation of TSP in unrestricted graphs is hard:

**Theorem** If \( \mathcal{P} \neq \mathcal{NP} \), then TSP has no polynomial \( k \)-approximation for any \( k > 0 \)

**Proof.** Assume that TSP has a poly-time \( k \)-approximation algorithm \( A \)

We show that using \( A \), we could solve the NP-complete Hamiltonian circle (HC) problem in polynomial time (\( \rightarrow \mathcal{P} = \mathcal{NP} \))

Now \( A \) finds TSP routes \( r_a \) with

\[
c(r_a) \leq (1 + k)c(r^*)
\]

Transform a given HC-instance \( G = (V, E) \) into a TSP graph \( G' \) with

\[
d(u, v) = \begin{cases} 
1 & \text{if } \{u, v\} \in E \\
(k + 1)|V| & \text{if } \{u, v\} \not\in E
\end{cases}
\]

If \( G \) has a Hamiltonian circle, then

\( G' \) has a TSP route of length \( |V| \) \( \Rightarrow \)

\[c(r_a) \leq (1 + k)|V|\]

If \( G \) has no Hamiltonian circle, then any TSP route in \( G' \) has length \( > (k + 1)|V| \) \( \Rightarrow \)

\[c(r_a) > (1 + k)|V|\]

Thus \( G \in HC \iff c(r_a) \leq (1 + k)|V| \)

Full polynomial \( \epsilon \)-approximation for Knapsack

Knapsack is much easier to solve approximatively, even at requested accuracy

Next:

An algorithm that finds \( \epsilon \)-approximations for Knapsack in time \( O(n^2/\epsilon) \)

That is, Knapsack has a full polynomial \( \epsilon \)-approximation, that is, algorithm that works in time \( O(p(n, 1/\epsilon)) \), where \( p() \) is a polynomial

Good approximations can be found by an interesting application of an exact pseudo-polynomial algorithm

Consider Knapsack instance

\[K = (c_1, \ldots, c_n; t_1, \ldots, t_n; T)\]

Here \( c_i \) and \( t_i \) are integer values and volumes of the items. We search for an optimal load \( S^* \subseteq \{1, \ldots, n\} \) such that

\[\sum_{i \in S^*} t_i \leq T\]

and the value

\[c(S^*) = \sum_{i \in S^*} c_i\]

is maximal
Exact optimum can be found using Dynamic Programming, to compute feasible solutions $(S,c,t)$, where $S$ is a load with value $c$ and volume $t$:

**proc** Knapsack($c[1..n]$, $t[1..n]$, $T$) **returns** Set $L := \{ (0,0,0) \}$;  
for $i := 1$ to $n$ do  
for $(S, c, t) \in L$ do  
if $t + t_i \leq T$ then  
$L := L \cup \{ (S \cup \{ t \}, c + c_i, t + t_i) \}$;  
for $(S, c, t), (S', c', t') \in L$ with $c = c'$ do  
if $t > t'$ then $L := L \setminus \{ (S, c, t) \}$;  
else $L := L \setminus \{ (S', c', t') \}$;  
Choose $(S, c, t) \in L$ with largest $c$;  
return $S$;  
$L$ stores of equal-value loads only those whose volume is smallest; Thus $|L| \leq c^* + 1$  
The algorithm can be implemented to run in time $O(n \times |L|) = O(nc^*)$

Time can be reduced by scaling the values of the problem down →  

dependent redacted...

Let $S_d$ be an optimal solution for $K_d$  
How good is $S_d$, as a solution to the original instance $K$ (wrt its optimal solution $S^*$)?  
c($S_d$) = $\sum_{i \in S_d} c_i = d \sum_{i \in S_d} (c_i/d)$  
$\geq d \sum_{i \in S_d} [c_i/d]$  
$\geq d \sum_{i \in S^*} [c_i/d] \quad (S_d$ is optimal for $K_d)$  
$\geq d \sum_{i \in S^*} (c_i/d - 1) = \sum_{i \in S^*} (c_i - d)$  
$v^* = v - \sum_{i \in S^*} d \geq v^* - nd$

Thus $v^* - nd \leq c(S_d) \leq v^*$ and so  

c($S_d$)/$c^* \leq nd/c^*$  
$\sim S_d$ is an $nd/c^*$-approximation for $K$

Choice of $d$ controls accuracy and time:  
c($S_d$) $\leq c^*/d \rightarrow$ proc. Knapsack finds an $nd/c^*$-approximation, in time $O(nc^*/d)$

With choice $d = nd/c^*$ we get an $c^*$-approximation in time $O(n^2/c)$

But we don’t know the optimum $c^*$!

Compute a $0.5$-approximation $c'$, using greedy heuristics in time $O(n \log n)$;  
Then $c^*/2 \leq c' \leq c^*$

Choice $d = nd/c'$ gives, using Knapsack procedure, an  

$n(\epsilon c'/n)/c^*$-approximation  
which is now also an  
$
\epsilon$-approximation

With these choices $c' \geq c^*/2$ and $d = \epsilon c'/n$, the time complexity of Knapsack is  

$O(nc^*/d) = O(n^2 c^*/(\epsilon^2))$  
$= O(n^2/\epsilon)$

But what if $\epsilon d/n < 1$?

Should we scale the values bigger?

No: Then $\epsilon' < n/\epsilon$, and procedure Knapsack finds even an exact solution in time  

$O(nc^*) = O(nc^*) = O(n^2/\epsilon)$