Dealing with NP

1 Backtracking

Features:

1. Constructs solutions component by component (grows a partial solution)
2. This processing is often implemented by constructing a tree of choices being made, called the state-space tree.
3. Root: initial state before the search for a solution begins.
4. Nodes: component of the solutions.
5. Leaves: dead end or solution found.

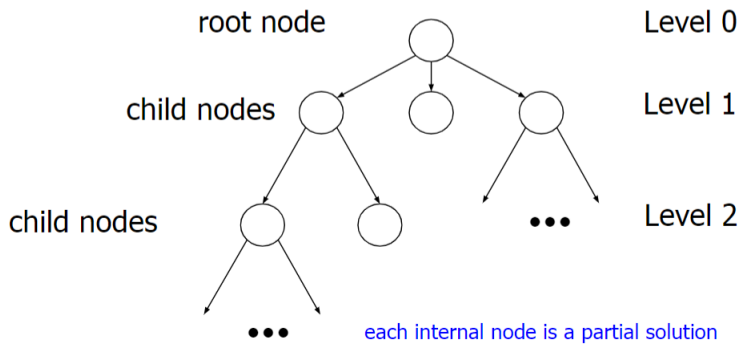

Figure 1: Backtracking

2 Branch-and-Bound

The main idea is to find optimal solution to optimize problem, by exploring the whole tree of solutions. Assume it is a minimization problem:

1. **Upper bound (UB):** keep track of BEST solution found so far.
2. **Lower bound (LB):** for each node (partial solution), computes a LB on the value of the objective function for all descendants of the node (extensions of the partial solution). We use LB for:
   - Ruling out certain nodes as "nonpromising" to prune the tree – if a node's bound is not better than the best solution seen so far.
   - Guiding the search through state-space as a measure of "promise".

3 Local Search Algorithms

Local search starts from initial position, iteratively moves from current position to one of neighboring positions (neighborhood relationship). Then it use evaluation function to choose among neighboring positions.

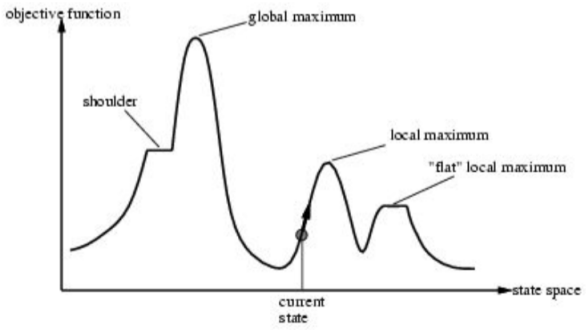

Figure 2: Local Search

There are multiple LS techniques:

1. **Hill climbing**: First-improvement or best-improvement neighbor as the next state.
2. **Stochastic local search**: Randomize initialization step, and Randomize search steps such that suboptimal/worsening steps are allowed.
3. **Simulated annealing**: Temperature decreasing with time, more worsening steps with high temperature.
4. **Tabu search:** Short-term memory to avoid revisiting previous states
5. **Iterated local search:** Perturbations on initial state s, several local searches

4 Approximation Algorithms

4.1 Definition

An approximation algorithm is a type of algorithm designed for solving optimization problems, particularly those that are NP-hard, where finding an exact solution is computationally infeasible for large instances. The goal of an approximation algorithm is to quickly find a solution that is close to the optimal one within a guaranteed bound.

Features:

1. **Polynomial time:** Approximation algorithms run in polynomial time, making them efficient and suitable for large problem instances.
2. **Guarantees:** Approximation algorithms provide a performance guarantee on how close the solution will be to the optimal. This is particularly useful when exact solutions are impractical due to time constraints.

4.2 Approximation Ratio

The quality of an approximation algorithm is often measured by its approximation ratio. For a **minimization** problem, this is defined as:

$$\rho(n) = \frac{A(I)}{OPT(I)} \tag{1}$$

Here:

- $A(I)$: the value of the solution produced by the algorithm for instance I
- $OPT(I)$: the value of the optimal solution
- $\rho(n)$: approximation ratio

For **maximization problem**:

$$\rho(n) = \frac{OPT(I)}{A(I)} \tag{2}$$

In general, $\rho(n) \geq 1$. If $\rho(n) = 1$, we have an optimal solution. Higher ratios indicate worse performance. If ρ is independent of n, then we have constant factor approximation.

4.3 Proof

Assume a minimization problem, and we want to prove an approximation ratio ρ, we want to show:

$$A(I) \leq \rho \cdot OPT(I) \tag{3}$$

To do this, usually we derive a lower bound for OPT:

$$OPT \geq x \tag{4}$$

and an upper bound for A:

$$A \leq y \tag{5}$$

After some magic math, we have:

$$A \leq \rho \cdots OPT \tag{6}$$