Local Search

Principles of Local Search

Algorithms: Design and Analysis, Part II

Neighborhoods

Let X = set of candidate solutions to a problem.

Examples: Cuts of a graph, TSP tours, CSP variable assignments

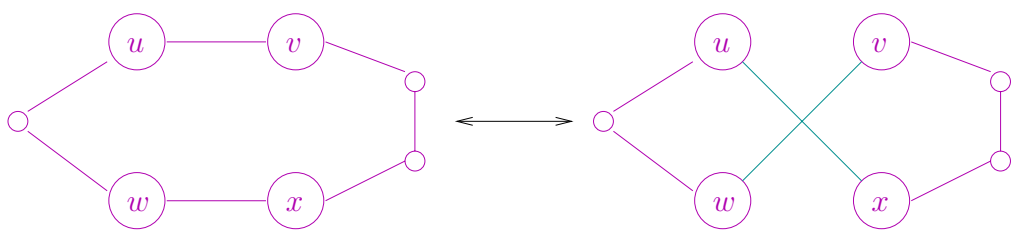
Key ingredient: Neighborhoods

- For each $x \in X$, specify which $y \in X$ are its "neighbors"

Examples: x, y are neighboring cuts $\iff$ Differ by moving one vertex

x, y are neighboring variable assignments $\iff$ Differ in the value of a single variable

x, y are neighboring TSP tours $\iff$ Differ by 2 edges



A Generic Local Search Algorithm

(1) Let x = some initial solution.

(2) While the current solution x has a superior neighboring solution y:

Set $x := y$

(3) Return the final (locally optimal) solution x

FAQ

Question: How to pick initial solution x?

Answer #1: Use a random solution.
$\Rightarrow$ Run many independent trials of local search, return the best locally optimal solution found.

Answer #2: Use your best heuristics
(i.e., use local search as a postprocessing step to make your solution even better).

Question #2: If there are superior neighboring y, which to choose?

Possible answers: (1) Choose at random, (2) biggest improvement, (3) more complex heuristics.

Question #3: How to define neighborhoods?

Note bigger neighborhoods $\Rightarrow$ slower to verify local optimality, but fewer (bad) local optima

Answer: Find "sweet spot" between solution quality and efficient searchability.

FAQ II

Question: Is local search guaranteed to terminate (eventually)?

Answer: If X is finite and every local step improves some objective function, then yes.

Question: Is local search guaranteed to converge quickly?

Answer: Usually not. [though it often does in practice] (see "smoothed analysis")

Question: Are locally optimal solutions generally good approximations to globally optimal ones?

Answer: No. [To mitigate, run randomized local search many times, remember the best locally optimal solution found]