Algorithms: Design and Analysis, Part II

Local Search

Principles of Local Search

Neighborhoods

Let X = set of candidate solutions to a problem.

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

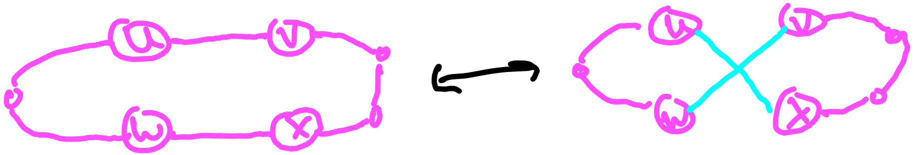
Key ingredient: neighborhoods

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

Examples: x, y are neighboring cuts $\Leftrightarrow$ differ by moving one vertex

x, y are neighboring variable assignments $\Leftrightarrow$ differ in the value of a single variable

x, y are neighboring TSP tours $\Leftrightarrow$ differ by 2 edges



Tim Roughgarden

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 soln 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 y at random ② biggest improvement ③ more complex heuristics

Question #3: how to define neighborhoods?

Answer: find "sweet spot" between solution quality and efficient searchability

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

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]