Section 19.4: Algorithmic Strategies for *NP*-Hard Problems

General, Correct, Fast (Pick Two)

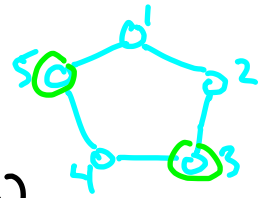
(1) General-purpose. Accommodates all inputs.

(2) Correct: On all inputs.

(3) Fast: Runs in polynomial time (on all inputs).

Compromising on Generality

Examples:



(1) weighted independent set.
- NP-hard in general graphs (as we'll see)
- linear-time solvable by dynamic programming in path graphs (and trees)

(2) knapsack
- solvable in $O(nC)$-time by D.P. (n = # of items, C = knapsack capacity)
  - polynomial time if C = polynomial in n
- NP-hard in general (as we'll see)

Compromising on Correctness

Heuristic algorithm: not always correct.

Ideally: (1) correct on "most" inputs and/or
(2) "almost correct" on every input

Examples (to come):

- greedy heuristics for scheduling, team-hiring, influence maximization
  (with provable "approximate correctness" guarantees)
- local search (often no provable guarantees
  but unreasonably effective in practice)

Compromising on Speed

Exact algorithm: running time is sometimes super-polynomial.

Ideally: (1) typically runs quickly (e.g., in polynomial time) and/or
(2) faster than exhaustive search on every input

Examples: - TSP ($O(n^2 2^n)$ time instead of $O(n!)$)
- finding paths of length k ($O(e^k \cdot m)$ time instead of $O(n^k)$)

Key Take-Aways

Three facts about NP-hard problems.

(1) Practically relevant NP-hard problems are ubiquitous.

(2) If a famous math conjecture is true, no NP-hard problem can be solved by an always-correct and always-polynomial-time algorithm.