Search in Complex Environments

Jacopo Mauro

Slide Based on Slides of the Artificial Intelligence: A Modern Approach book

Local Search and Optimization Problems

Trees and paths are not the answer to all the problems

- TSP, configuration satisfying constraints (timetable), …

Idea in this case: one can use iterative improvement algorithms

- keep a single “current” state
- try to improve it

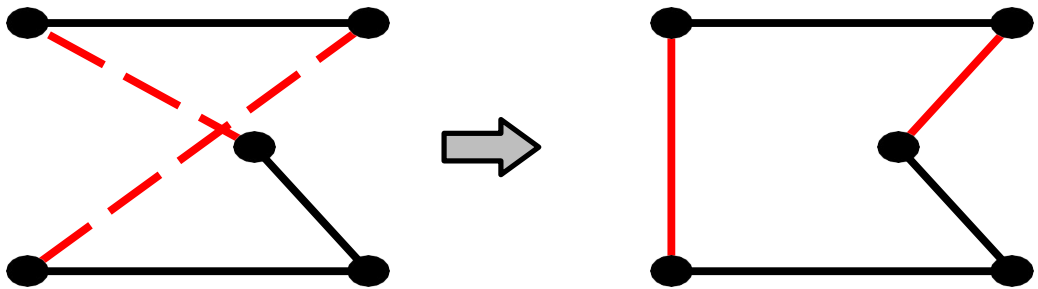
Local search algorithms searching from a start state to neighboring states, without keeping track of the paths

Not systematic → might never explore a portion of the search space

Example: Travelling Salesperson Problem

Idea: start with any complete tour, perform pairwise exchanges.

Variants of this approach get within 1% of optimal very quickly (with thousands of cities)

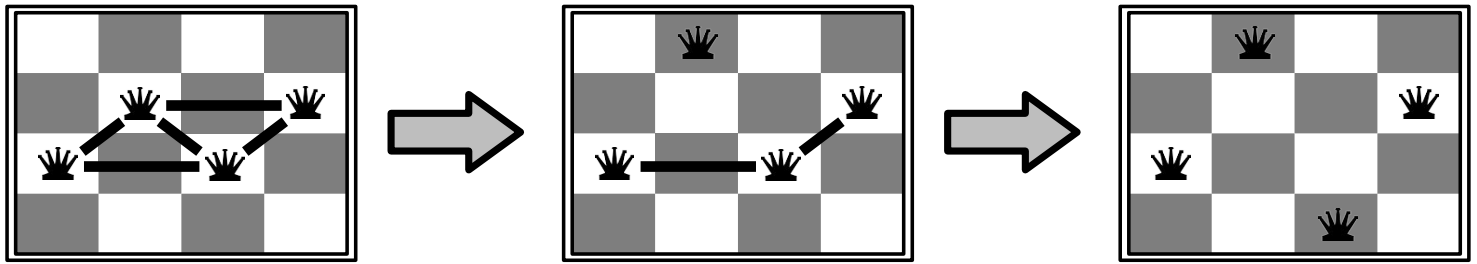


Example: n-queens

Goal: Put n queens on an n×n board with no two queens on the same row, column, or diagonal

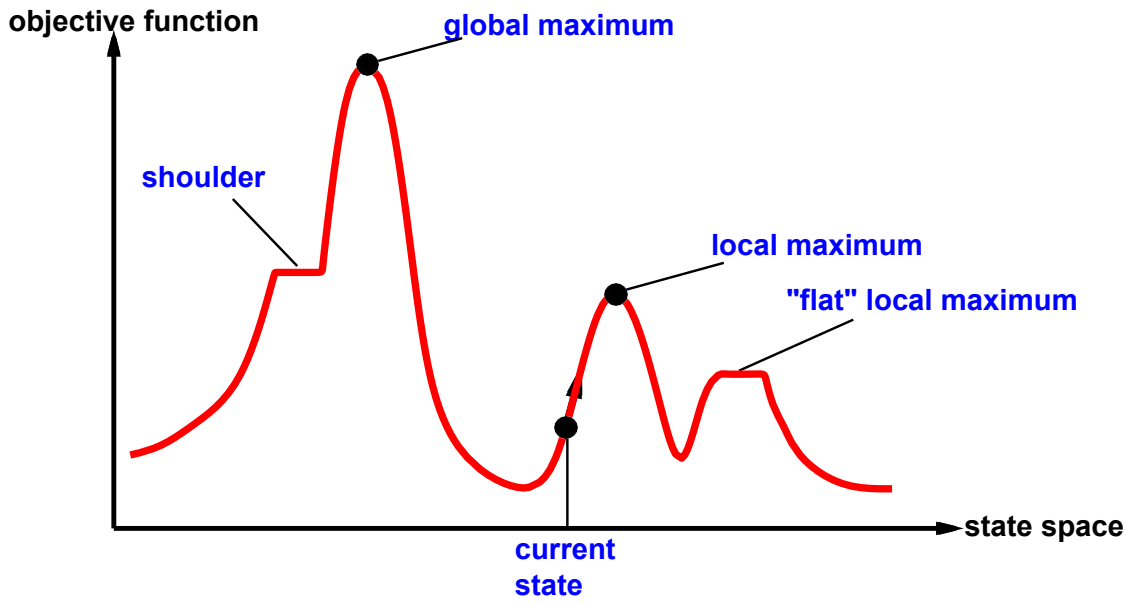
Idea: Move a queen to reduce number of conflicts

Almost always solves n-queens problems almost instantaneously for very large n, e.g., 1 million



h = 5 **h = 2** **h = 0**

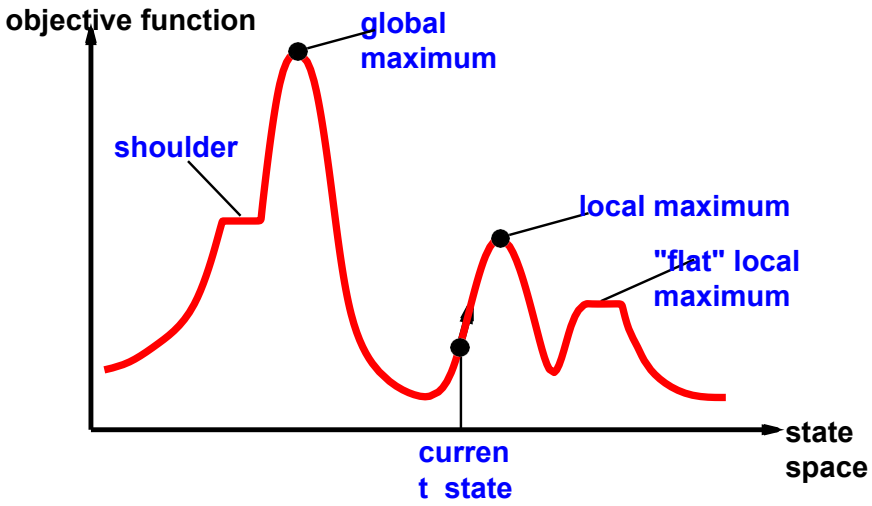
Hill-climbing



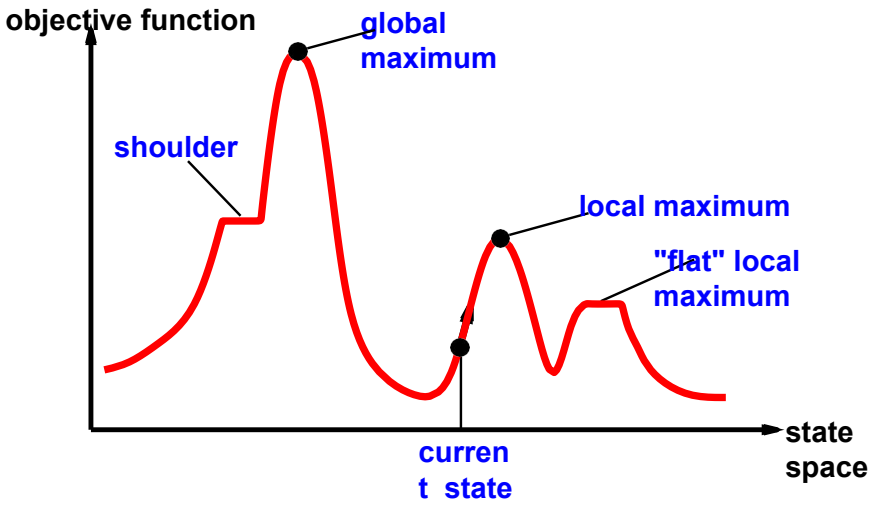
Hill-climbing

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
  current ← problem.INITIAL
  while true do
    neighbor ← a highest-valued successor state of current
    if VALUE(neighbor) ≤ VALUE(current) then return current
    current ← neighbor
```

Problems of Hill-climbing?



Problems of Hill-climbing?



Random-restart hill climbing overcomes local maxima → trivially complete

Random sideways moves escape from shoulders → loop on flat maxima

Simulated annealing

Idea: escape local maxima by allowing some “bad” moves + gradually decrease their size and frequency (aka decrease the temperature)

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  current ← problem.INITIAL
  for t = 1 to ∞ do
      T ← schedule(t)
      if T = 0 then return current
      next ← a randomly selected successor of current
      ΔE ← VALUE(current) – VALUE(next)
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(−ΔE/T)
```

Properties of simulated annealing

Exploration vs. Exploitation:

- High temperature: Encourages exploration by accepting worse solutions.
- Low temperature: Focuses on exploitation by refining towards the best solution.

Cooling Schedule: rate of temperature decrease (cooling schedule) is important

- Too fast: May miss the global optimum.
- Too slow: Computationally expensive.
- Common schedules: $T \propto 1/\log(t)$, where t is the iteration step
- Global Optimality Guarantee: if the temperature is decreased at an appropriate rate simulated annealing will always reach the global best state (based on properties of the Boltzmann distribution → temperature must decrease asymptotically towards 0)

Local beam search

Idea: keep k states instead of 1; choose top k of all their successors

Not the same as k searches run in parallel!

Searches that find good states recruit other searches to join them

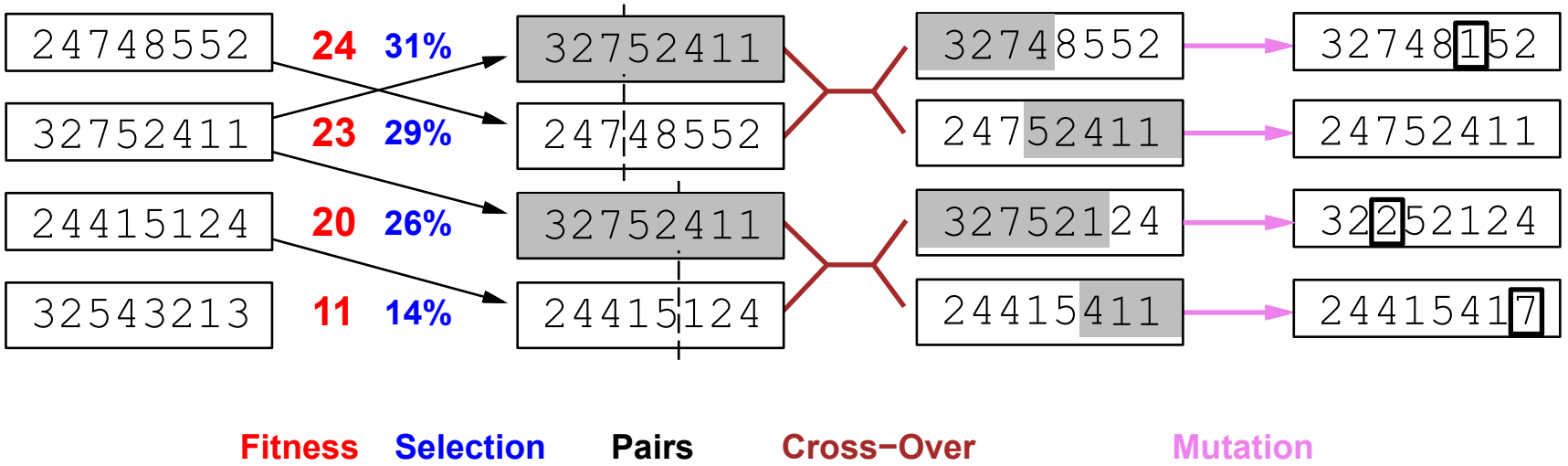
Problem: quite often, all k states end up on same local hill

- Idea: choose k successors randomly, biased towards good ones

Genetic algorithms

Idea: Simulate natural selection

= Stochastic local beam search + generate successors from pairs of states



Genetic Algorithms

Fitness Function

- Evaluates the quality of each individual.
- Avoid overly complex fitness functions to reduce computation time.

Genetic Operators:

- Selection: Favors fitter individuals for reproduction.
- Crossover: Combines features from two parents to create offspring.
- Mutation: Introduces random changes to maintain diversity.

Population Size:

- Small Population: Faster convergence but risks premature optimization.
- Large Population: Better exploration but computationally expensive.

Genetic Algorithms

Crossover Rate (Recombination):

- Typically set between 0.6–0.9 (60–90% of the population undergoes crossover)
- Higher rates encourage more exploration and combining of good solutions.

Mutation Rate:

- Usually kept low (0.001–0.01) to avoid excessive randomness.
- Helps maintain diversity and escape local optima.

Genetic Algorithms

Selection Method strategies

- Roulette Wheel: Probabilistic, favors fitter individuals.
- Tournament Selection: Picks the best among (random) subsets.
- Elitism: Directly carries over top solutions to the next generation. Used to preserve the best solutions.

Number of Generations

- Stop condition: Based on either fixed generations or convergence (no improvement over time).

Continuous state spaces

Discretization methods: turn continuous space into discrete space, e.g., empirical gradient considers ±δ change in each coordinate

Gradient-Based Methods:

- Gradient Descent: Moves in the direction of steepest descent
  - Initialize: Start with an initial guess x_0.
  - Compute Gradient: $\nabla f(x)$ at the current point.
  - Update Rule: move in the direction of the negative gradient: $x_{t+1}=x_t-\eta\nabla f(x_t)$ where η is the learning rate (step size)
  - Repeat: Continue until convergence (e.g., gradient is close to 0 or changes in f(x) are small).
- Newton's Method: Incorporates second derivatives for faster convergence.
- Linear Programming: Solve linear equations

Search with Nondeterministic Actions

Agent doesn't know the state its transitioned to after action, but all the possible states it will be ("belief state")

Example: the erratic vacuum world where

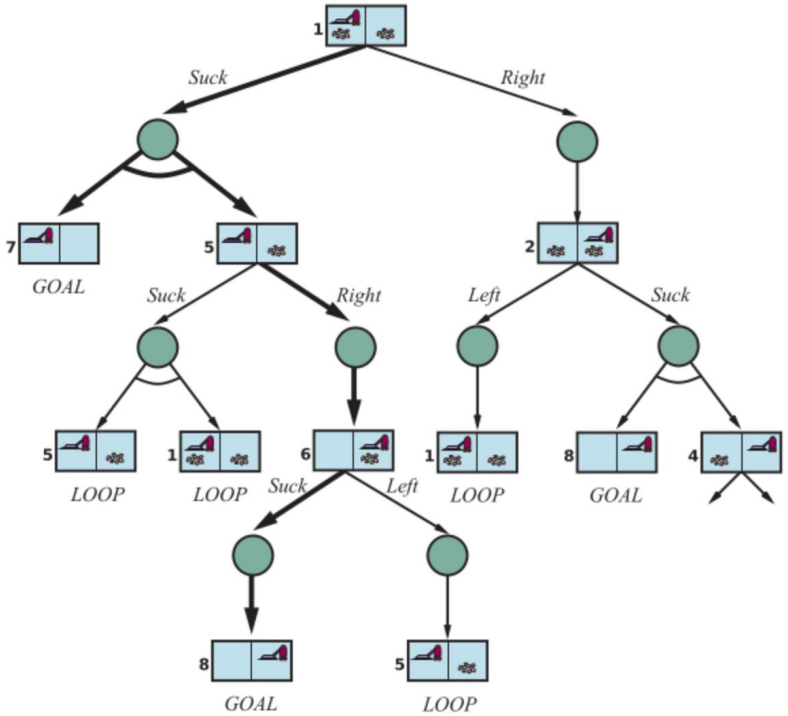
- When is dirty sucks cleans the square and sometimes cleans up the adjacent square
- When applied to a clean square sometimes deposits dirt on the carpet

Idea: AND-OR search trees

- OR nodes to explore all the transitioned state
- AND nodes to have plan for all the states

Solution: conformant tree with goal node at every leaf + one action at each of its OR nodes + outcome for all actions for AND nodes

Search with Nondeterministic Actions



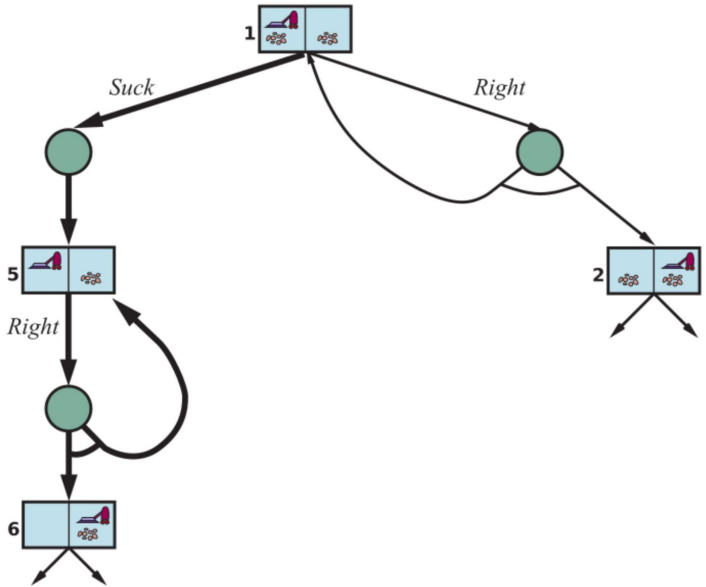
Search with Nondeterministic Actions

```
function AND-OR-SEARCH(problem) returns a conditional plan, or failure
  return OR-SEARCH(problem, problem.INITIAL, [])

function OR-SEARCH(problem, state, path) returns a conditional plan, or failure
  if problem.IS-GOAL(state) then return the empty plan
  if IS-CYCLE(state, path) then return failure
  for each action in problem.ACTIONS(state) do
      plan ← AND-SEARCH(problem, RESULTS(state, action), [state] + path)
      if plan ≠ failure then return [action] + plan
  return failure

function AND-SEARCH(problem, states, path) returns a conditional plan, or failure
  for each s_i in states do
      plan_i ← OR-SEARCH(problem, s_i, path)
      if plan_i = failure then return failure
  return [if s_1 then plan_1 else if s_2 then plan_2 else ... if s_{n-1} then plan_{n-1} else plan_n]
```

Note: failure → means that there is no non-cyclical solution

Some reflections on cyclic solutions

- Consider slippery vacuum
  - Movement of robot can fail
- No solution that are not cyclic
- Minimal solution
  - Leaf is a goal state
  - Leaf reachable for every point of the plan



Search in Partially Observable Environments

The agent has limited or noisy information about the environment.

Must reason about the possible states based on observations and actions.

Applications:

- Robot navigation with limited sensors.
- Decision-making in dynamic, incomplete data environments.

If we can test the environment (sensor) → and or with sensor to restrict the belief state → solution is a conditional plan (have ifs)

- If sensor unreliable → probabilistic reasoning (Chapter 12)

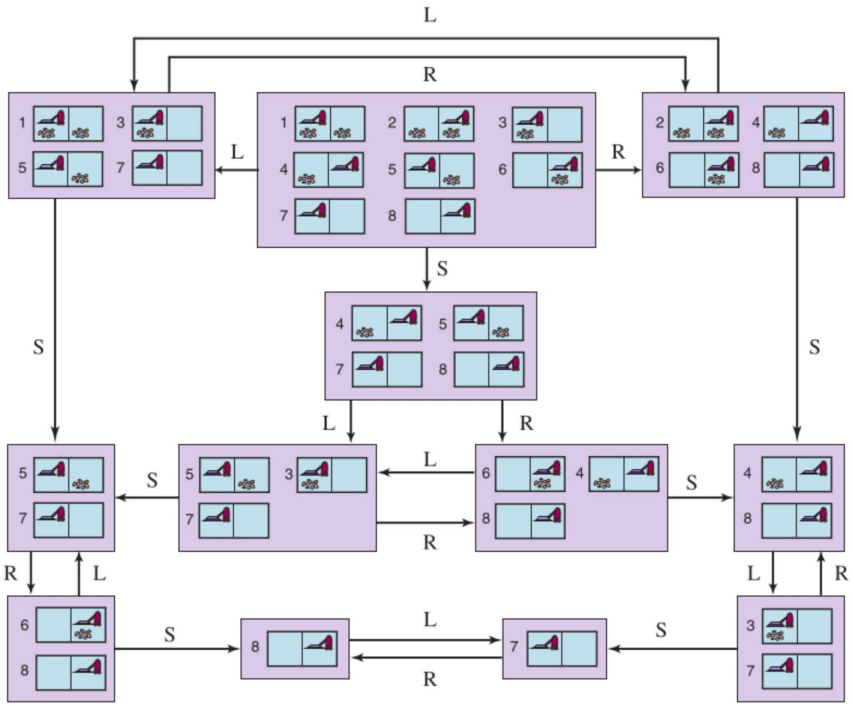
No observations allowed

Seems impossible but sensorless solutions are surprisingly common and useful

- Robots orienting parts using a sequence of actions
- Doctor prescribing generic antibacterial medicine

Conformant Planning: generates a sequence of actions that guarantees success for all states in the initial belief state.

No observations allowed



Online Search

Offline search → complete solution before taking their first action

Online search → first it takes an action, then it observes the environment and computes the next action

- random walk:
  - selects at random one of the available actions
  - eventually find a goal or complete its exploration (space is finite and safely explorable)
- Learning real-time A^*
  - Builds a map of the environment + chooses the "apparent best" move
  - Action not tried in a state are assumed to lead immediately to goal (prioritizes exploration)

Homework

Read Chapter 4