

# Artificial Intelligence

## CE-417, Group 1

### Computer Eng. Department

### Sharif University of Technology

Fall 2020

By Mohammad Hossein Rohban, Ph.D.

Courtesy: Most slides are adopted from CSE-573 (Washington U.), original  
slides for the textbook, and CS-188 (UC. Berkeley).

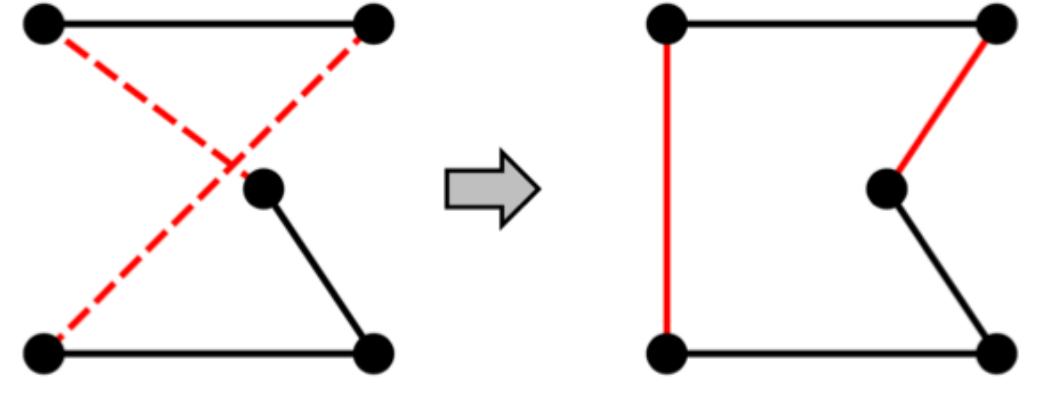
# Local Search

# Iterative improvement algorithms

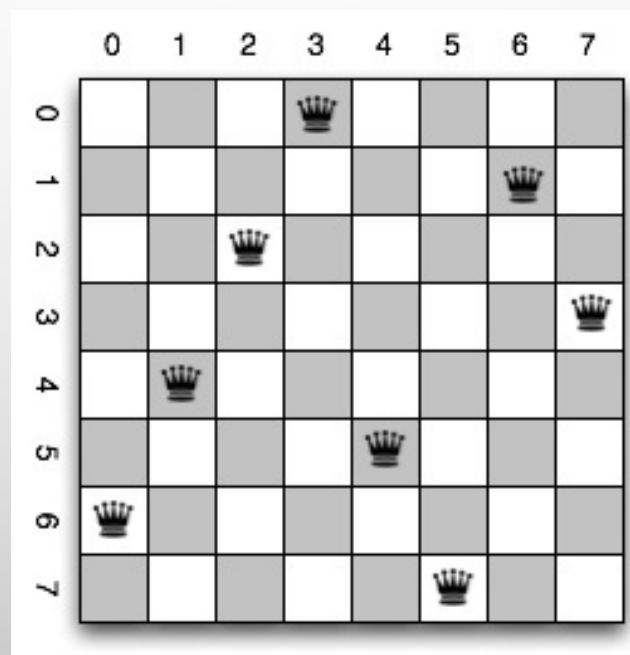
- Previously: Search to find best path to goal
  - Systematic exploration of search space.
- Today: a state is solution to problem
  - For some problems path is **irrelevant**.
  - e.g., 8-queens
- In such cases, can use iterative improvement algorithms;
  - **keep a single “current” state, try to improve it**

# Examples

- TSP



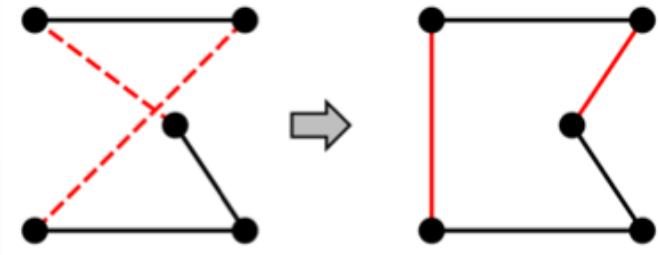
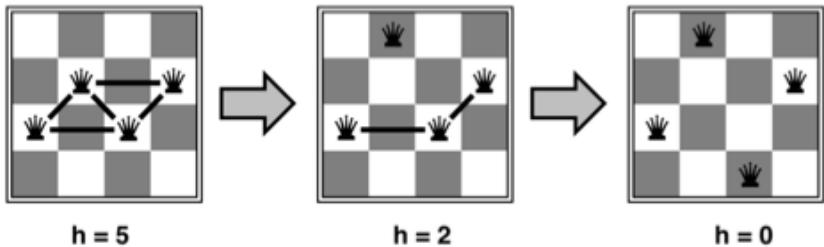
- n-queens



# Local search algorithms

- State space = set of "complete" configurations
- Find configuration satisfying constraints,
  - e.g., all n-queens on board, no attacks
- In such cases, we can use **local search algorithms**
- Keep a single "current" state, try to improve it.
- Very memory efficient
  - *duh* - only remember current state

# Constraint Satisfaction vs. Constraint Optimization



## Goal Satisfaction

Constraint satisfaction  
reach the goal node  
guided by heuristic fn

## Optimization

Constraint Optimization  
optimize(objective fn)

You can go back and forth between the two problems. Typically in the same complexity class

# Local Search and Optimization

- **Local search:**

- Keep track of single current state
- Move only to “neighboring” state (defined by operators)
- Ignore previous states, path taken

- **Advantages:**

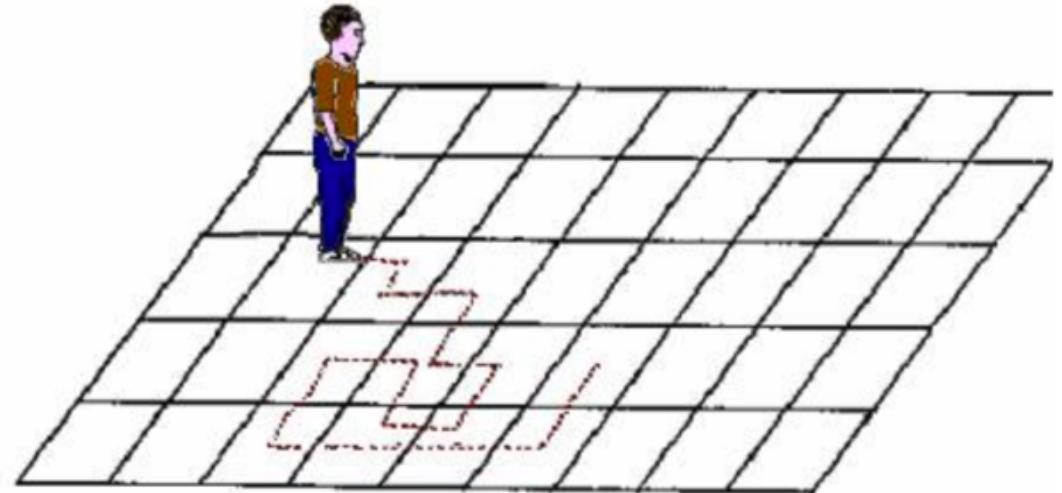
- Use very little memory
- Can often find reasonable solutions in large or infinite (continuous) state spaces.

- **“Pure optimization” problems**

- All states have an objective function
- Goal is to find state with max (or min) objective value
- Does not quite fit into path-cost/goal-state formulation
- Local search can do quite well on these problems.

# Trivial Algorithms

- Random Sampling
  - Generate a state randomly
- Random Walk
  - Randomly pick a neighbor of the current state
- Why even mention these?
  - Both algorithms asymptotically complete.
    - If the state space is finite, each state is visited at a fixed rate asymptotically.

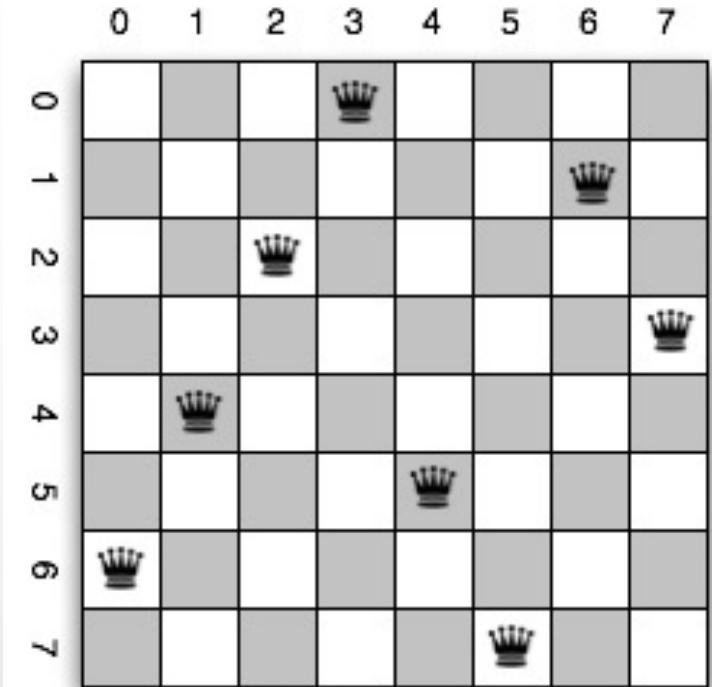


# Hill-climbing search

- “a loop that continuously moves towards increasing value”
  - terminates when a peak is reached
  - Aka greedy local search
- Value can be either
  - Objective function value
  - Heuristic function value (minimized)
- Hill climbing does not look ahead of the immediate neighbors
- Can randomly choose among the set of best successors
  - if multiple have the best value
- **“climbing Mount Everest in a thick fog with amnesia”**

# Example: $n$ -Queens

- State
  - All  $n$  queens on the board in some configuration
  - But each in a different column
- Successor function
  - Move single queen to another square in same column.
- How to convert this into an optimization problem?

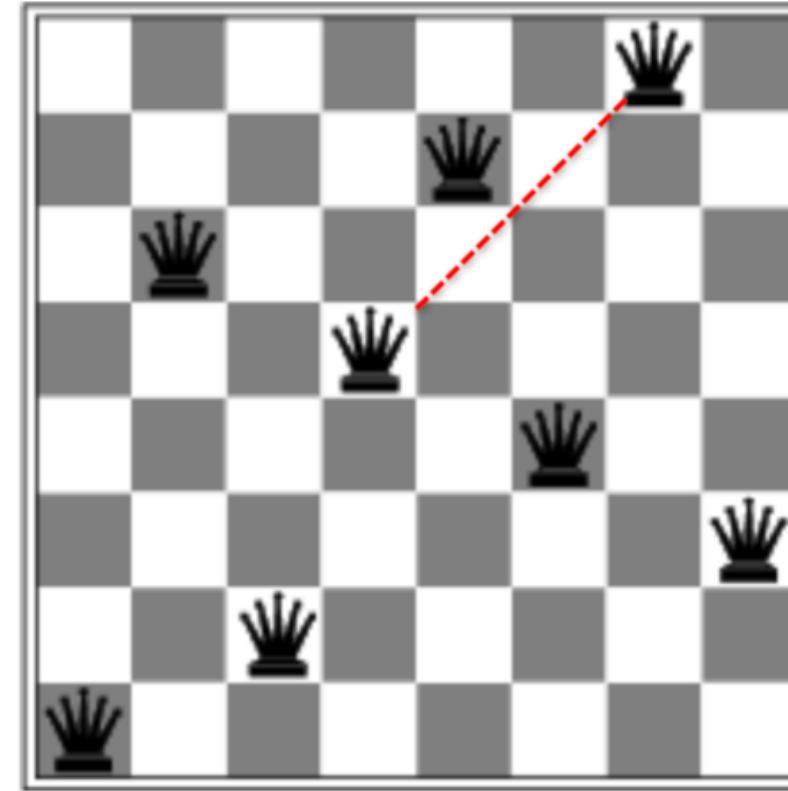


# Hill-climbing search: 8-queens

- Result of hill-climbing in this case...

Bummer

A local minimum with  $h = 1$

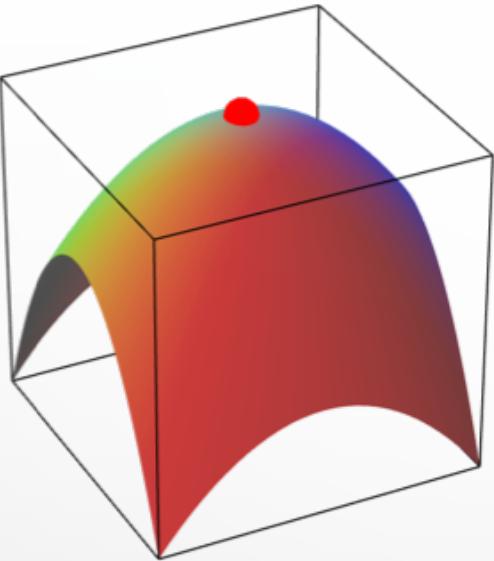


# Hill-climbing performance on n-queens

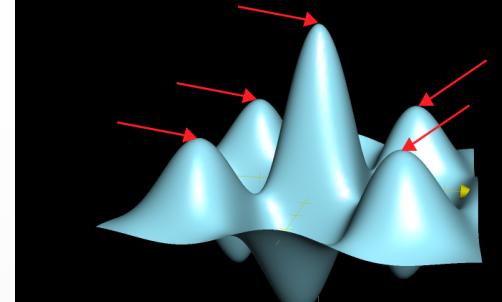
- Hill-climbing can solve large instances of  $n$ -queens ( $n = 106$ ) in a few (ms)seconds
- 8 queens statistics:
  - State space of size  $\approx 17$  million
  - Starting from random state, steepest-ascent hill climbing solves 14% of problem instances
  - It takes 4 steps on average when it succeeds, 3 when it gets stuck
  - When “**sideways**” moves are allowed, performance improves ...
  - When multiple restarts are allowed, performance improves even more

# Hill Climbing Drawbacks

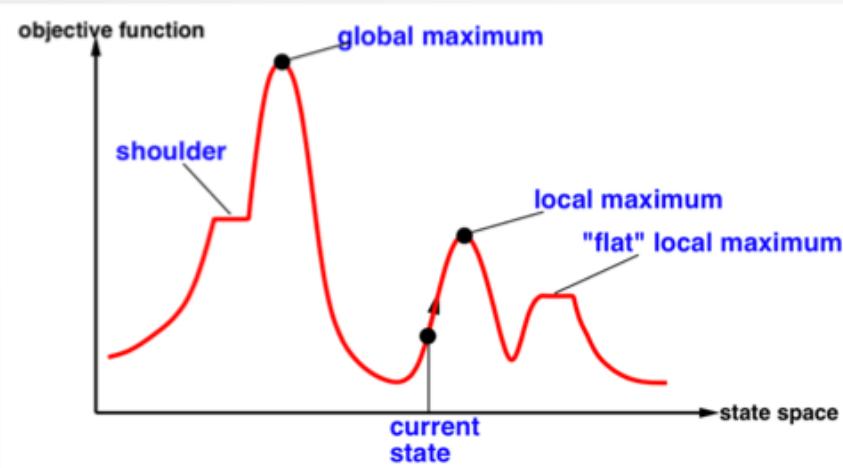
Local maxima



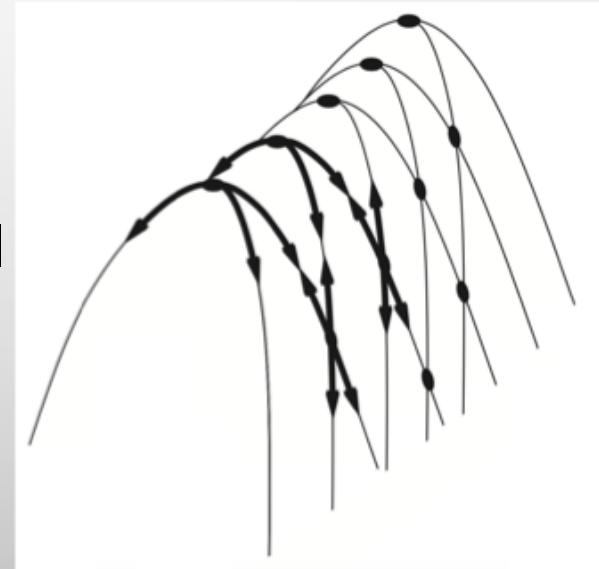
These are all local maxima



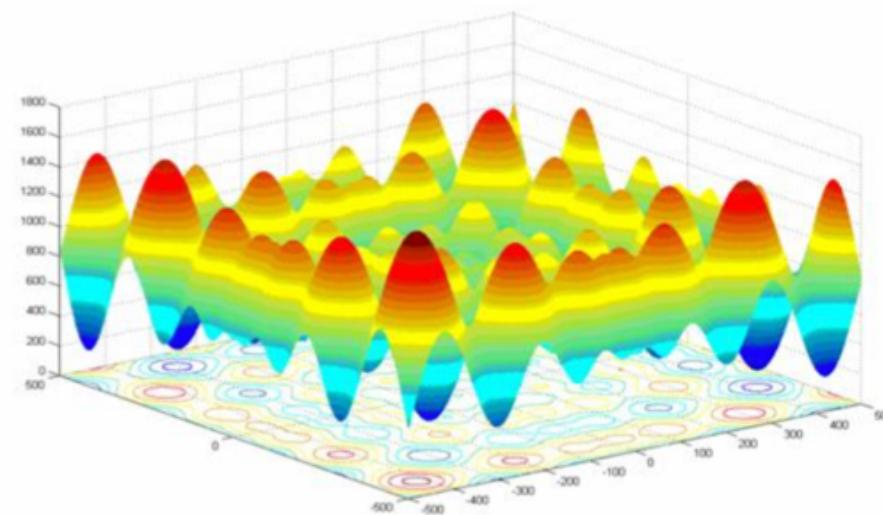
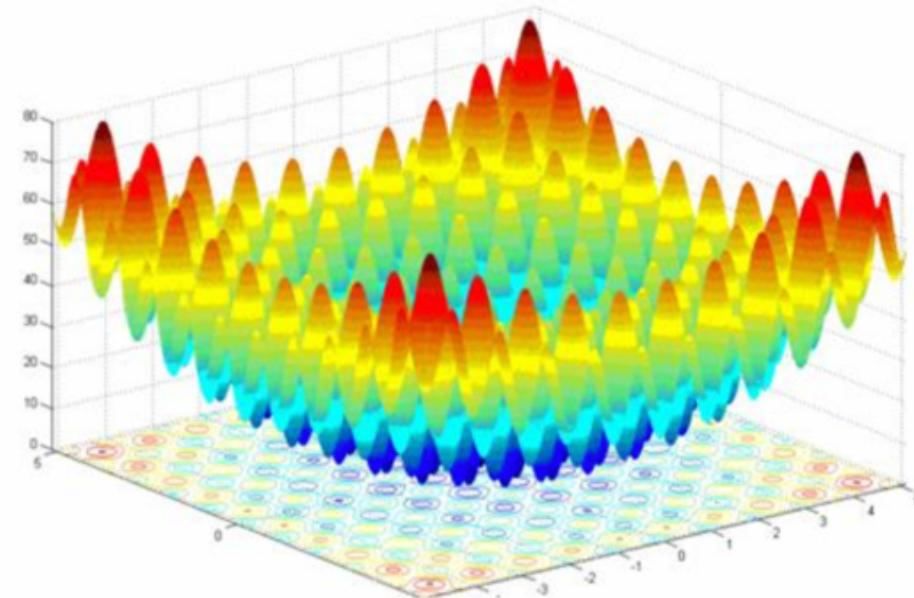
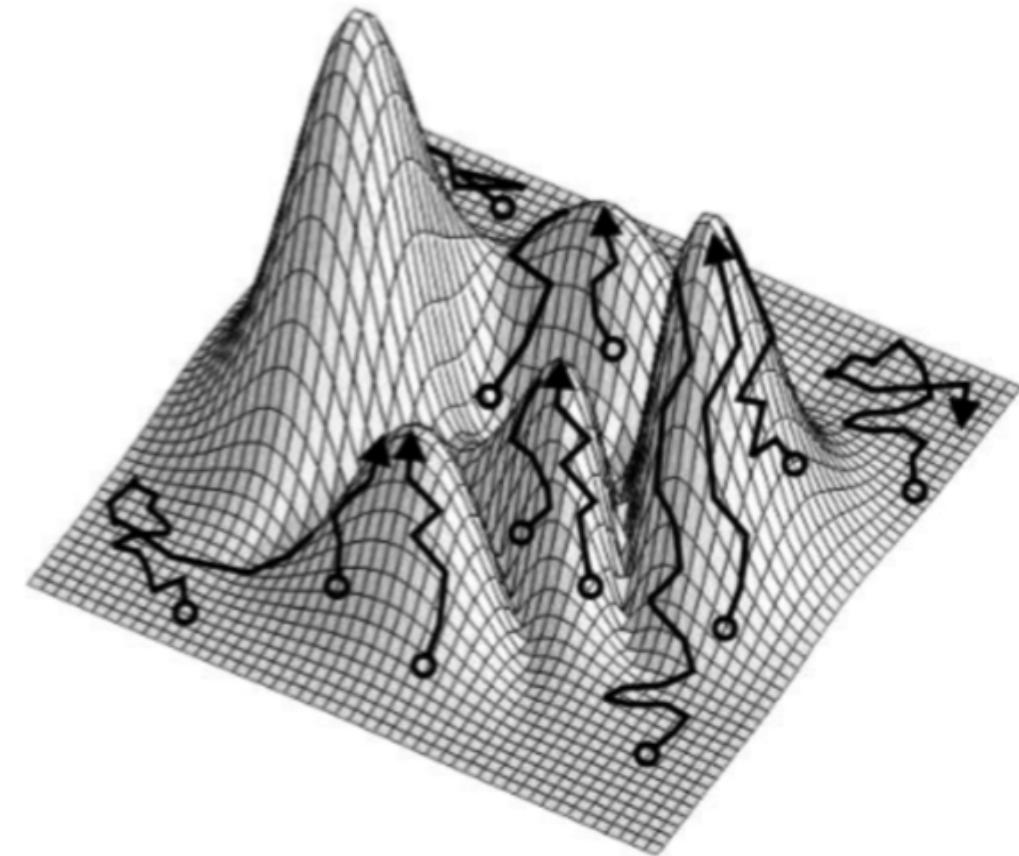
Plateaus



Diagonal ridges

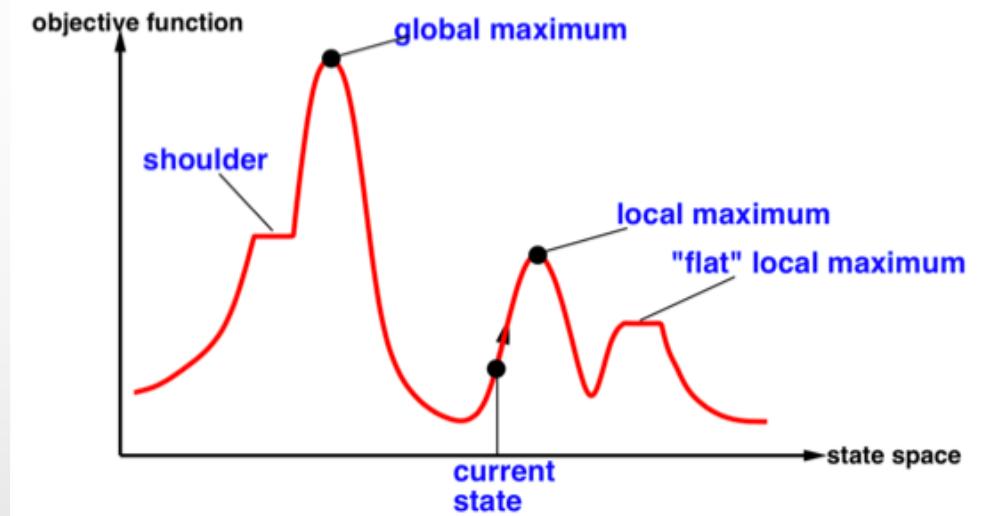


# Trajectories, difficulties



# Escaping Shoulders: Sideways Move

- If no downhill (uphill) moves, allow sideways moves in hope that algorithm can escape
  - Must limit the number of possible sideways moves to avoid infinite loops
- For 8-queens
  - Allow sideways moves with limit of 100
  - Raises percentage of problems solved from 14 to 94%
  - However....
    - 21 steps for every successful solution
    - 64 for each failure



# Hill Climbing Properties

- Not complete. Why?
- Terrible worst case running time.
- Simple,  $O(1)$  space, and often very fast.

# Tabu Search

- Prevent returning quickly to the same state
- Keep fixed length queue (“tabu list”)
- Add most recent state to queue; drop oldest
- Never move to a tabu state
- Properties:
  - As the size of the tabu list grows, hill-climbing will asymptotically become “non-redundant” (won’t look at the same state twice)
  - In practice, a reasonable sized tabu list (say 100 or so) improves the performance of hill climbing in many problems

# Hill Climbing: Stochastic Variations

- When the state-space landscape has local minima, any search that moves only in the greedy direction cannot be complete
- Random walk, on the other hand, is asymptotically complete
- **Idea:** Combine random walk & greedy hill-climbing
- At each step do one of the following:
  - **Greedy:** With prob.  $p$  move to the neighbor with largest value
  - **Random:** With prob.  $1-p$  move to a random neighbor

# Hill-climbing with random restarts

- If at first you don't succeed, try, try again!
- Different variations
  - For each restart: run until termination vs. run for a fixed time
  - Run a fixed number of restarts or run indefinitely
- Analysis
  - Say each search has probability  $p$  of success
  - e.g., for 8-queens,  $p = 0.14$  with no sideways moves
- Expected number of restarts?
- Expected number of steps taken?



# Hill-Climbing with Both Random Walk & Random Sampling

- At each step do one of the three
  - **Greedy**: move to the neighbor with largest value
  - **Random Walk**: move to a random neighbor
  - **Random Restart**: Start over from a new, random state

# Simulated Annealing

- Idea: escape local maxima by allowing some “bad” moves
  - **but gradually decrease their size and frequency**
  - method proposed in 1983 by IBM researchers for solving VLSI layout problems
- A Physical Analogy:
  - Imagine letting a ball roll downhill on the function surface
  - Now shake the surface, while the ball rolls,
  - Gradually reducing the amount of **shaking**



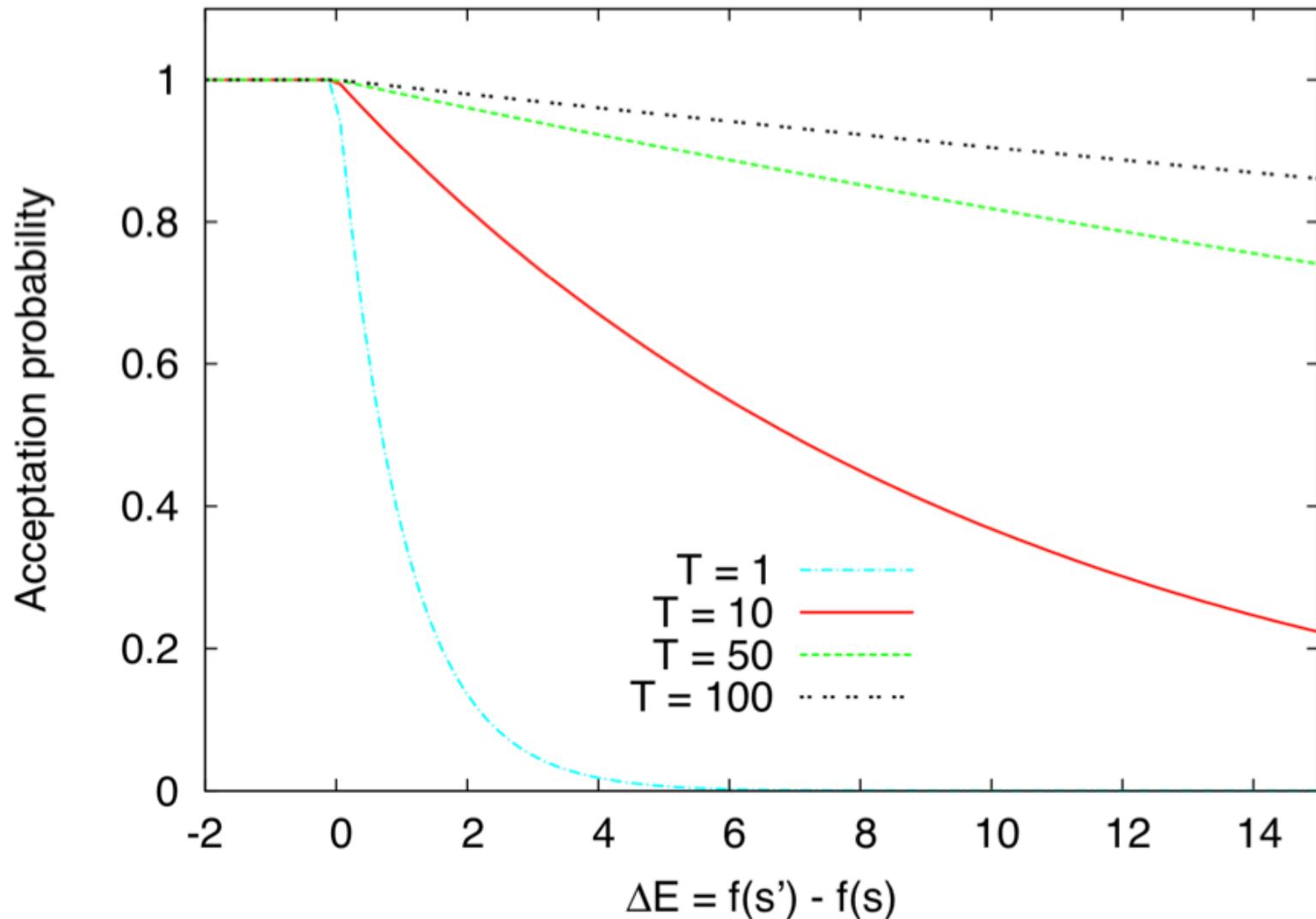
# Simulated Annealing (cont.)

- Annealing = physical process of cooling a liquid → frozen
- simulated annealing:
  - free variables are like particles
  - seek “low energy” (high quality) configuration
  - slowly reducing temp.  $T$  with particles moving around randomly
- high  $T$ : probability of “locally bad” move is higher
- low  $T$ : probability of “locally bad” move is lower
- typically,  $T$  is decreased as the algorithm runs longer
  - i.e., there is a “temperature schedule”

# Simulated Annealing (cont.)

```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
          schedule, a mapping from time to “temperature”
  local variables: current, a node
                    next, a node
                    T, a “temperature” controlling prob. of downward steps
  current  $\leftarrow$  MAKE-NODE(INITIAL-STATE[problem])
  for t  $\leftarrow$  1 to  $\infty$  do
    T  $\leftarrow$  schedule[t]
    if T = 0 then return current
    next  $\leftarrow$  a randomly selected successor of current
     $\Delta E \leftarrow$  VALUE[next] – VALUE[current]
    if  $\Delta E > 0$  then current  $\leftarrow$  next
    else current  $\leftarrow$  next only with probability  $e^{\Delta E/T}$ 
```

# Effect of temperature



# Simulated Annealing in practice

- Other applications:
  - Traveling salesman, Graph partitioning, Graph coloring, Scheduling, Facility Layout, Image Processing, ...
- Optimal, given that  $T$  is decreased **sufficiently slow**.
  - Is this a useful guarantee?
- Convergence can be guaranteed if at each step,  $T$  drops no more quickly than  **$C/\log n$** ,  $C=\text{constant}$ ,  $n = \# \text{ of steps so far}$ .

# Local beam search

- Idea: Keeping only one node in memory is an extreme reaction to memory problems.
- Keep track of  $k$  states instead of one
  - Initially:  $k$  randomly selected states
  - Next: determine all successors of  $k$  states
  - If any of successors is goal → finished
  - Else select  $k$  best from successors and repeat

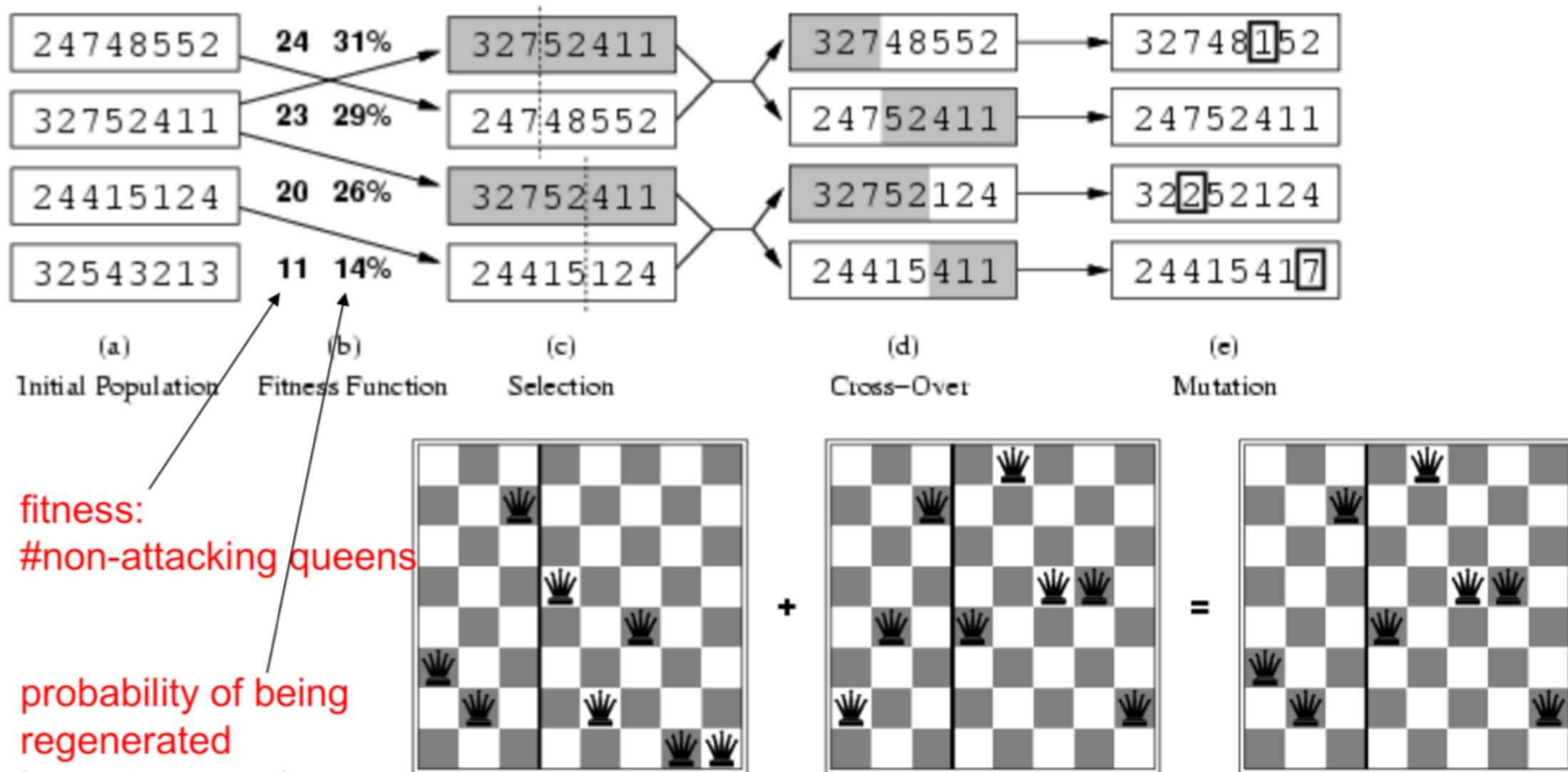
# Local Beam Search

- Not the same as  $k$  random-start 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: Stochastic beam search
  - Choose  $k$  successors randomly, biased towards good ones
- Observe the close analogy to natural selection!

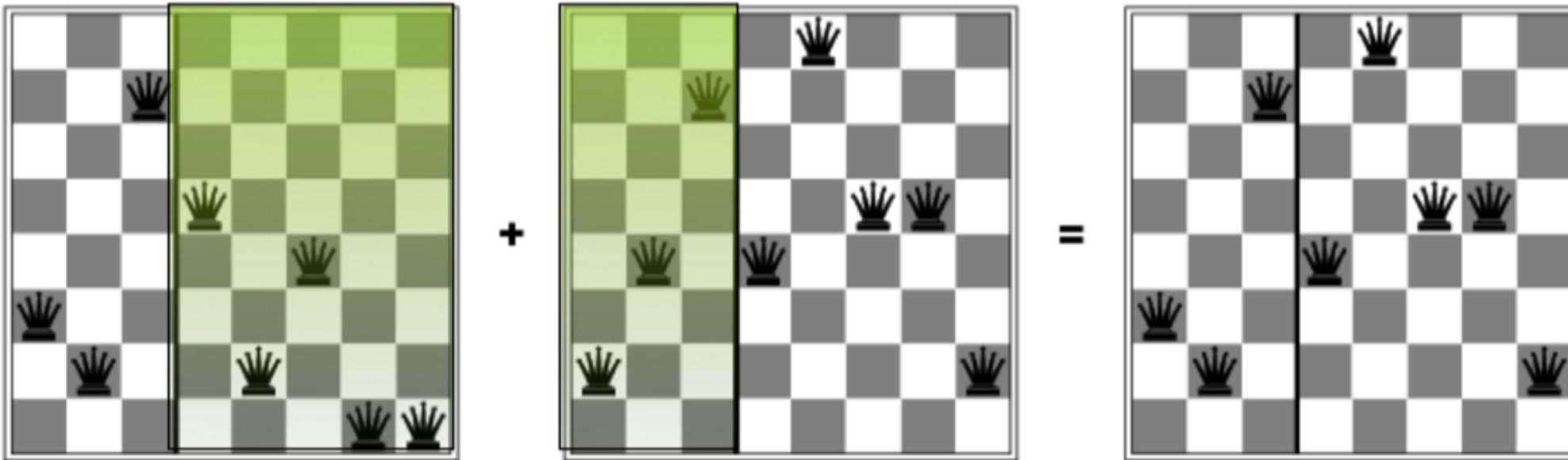
# Genetic algorithms

- Local beam search, but...
  - A successor state is generated by **combining two parent states**
- Start with  $k$  randomly generated states (**population**)
- A state is represented as a **string** over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (**fitness function**). Higher = better
- Produce the next generation of states by **selection, crossover, and mutation**

# n-queens example



## n-queens example (cont.)



Has the effect of “jumping” to a completely different new part of the search space (quite non-local)

# Comments on Genetic Algorithms

- Genetic algorithm is a variant of “stochastic beam search”

- **Positive points**

- Random exploration can find solutions that local search can't
    - (via crossover primarily)
  - Appealing connection to human evolution
    - “neural” networks, and “genetic” algorithms are **metaphors!**

- **Negative points**

- Large number of “tunable” parameters
    - Difficult to replicate performance from one problem to another
  - Lack of good empirical studies comparing to simpler methods
  - Useful on some (small?) set of problems but no convincing evidence that GAs are better than hill-climbing w/random restarts in general