COMS W4701: Artificial Intelligence

Lecture 6: Local Search

Tony Dear, Ph.D.
Department of Computer Science
School of Engineering and Applied Sciences

Today

- Local search in discrete spaces
- Hill-climbing
- Simulated annealing
- Evolutionary algorithms

Local Search Algorithms

- Search algorithms covered so far may traverse all possible states
- If space is finite and solution exists, (most) algorithms guaranteed to find it

- Systematic search do not work well in large, infinite, or continuous spaces
- In some problems, we care more for final state than the actual trajectory

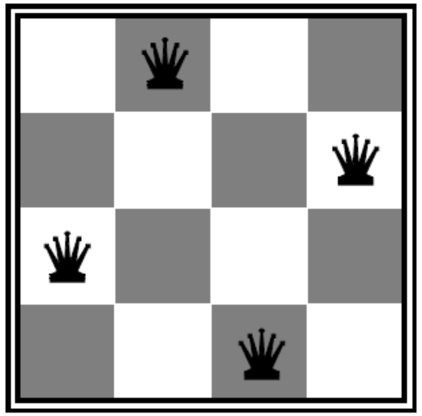
- **Local search** approaches only keep track of current state
- Use very little (constant) memory and can often find solutions in large spaces in practice, though with fewer guarantees

Example: n-Queens

- CSP solvers incrementally generate consistent assignments until complete
- Now consider a state space of all complete assignments of n queens, both consistent and inconsistent

- **Random sampling**: Randomly generate states until a valid one is found

- **Random walk**: From the current state, randomly generate a *successor state* by modifying a single queen assignment
- Repeat until valid state is found

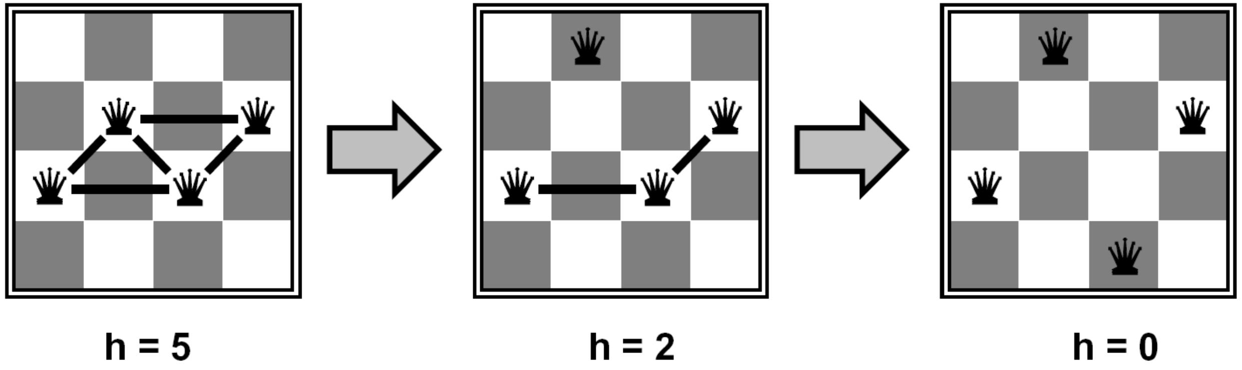


Example: n-Queens

- Both random sampling and random walks are *complete* with infinite time
- No upper limit on time needed, may be very slow in practice

- **Iterative best improvement** selects successor states by minimizing (greedy descent) or maximizing (hill climbing, greedy ascent) an *evaluation/objective function*

- E.g., number of *conflicts* h in current state with valid solution $h = 0$



Iterative Best Improvement

```
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
```

- Hill climbing and other iterative methods may get stuck at **local optima**, states at which which no neighbor is better than the current one

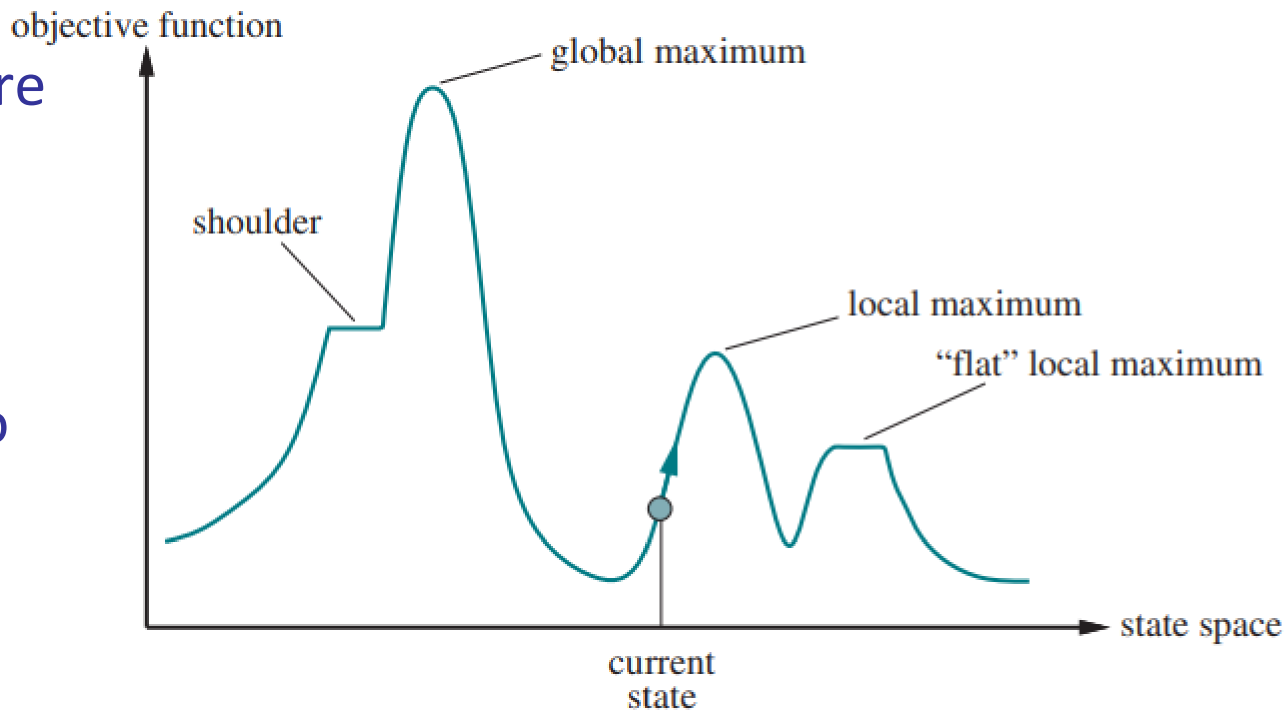
- We prefer **global optima**, but no guarantee that we can find one
- Hill climbing is neither optimal nor complete

State Space Landscapes

- Can think about search space as a "landscape" of different features
- In addition to local/global optima, there may be shoulders or "flat" optima

- We may include *sideways* moves to equally valued successors, but need to avoid getting stuck in endless cycles

- We may include *random* moves to escape local optima

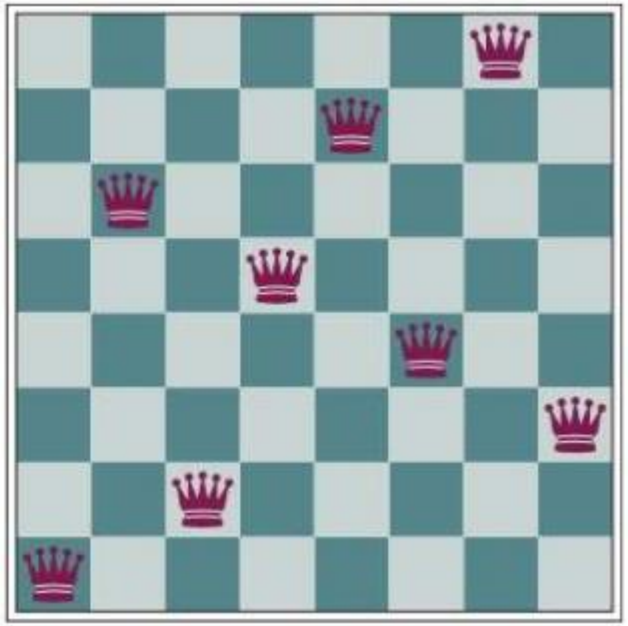


Example: n-Queens

- Current board has $h = 1$ conflict
- All neighboring states are the same or worse

- To escape this flat local minimum, we can allow sideways moves to other states with $h = 1$
- But limit number of sideways moves to avoid cycles

- For 8-queens, success prob without sideways moves is ~14%, with up to 100 sideways moves ~94%



Stochastic Local Search

- We can add randomness to iterative search to expand exploration
- E.g., take a *random step* to a worse successor
- Or do a *random restart* and move to a completely different state

- *Stochastic* hill-climbing: Instead of selecting best successor, select one based on a probability distribution assigned to all successors

- *First-choice* hill-climbing: Instead of evaluating all possible successors, just select a random (e.g., the first) one that improves upon current state

Simulated Annealing

- **Simulated annealing**: Generate random successor and move to it if better
- Otherwise, we *may* move to a worse successor with some probability

- Probability inversely proportional to how "bad" the successor is as well as time
- Time to probability mapping is determined by the *temperature* T

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  $current \leftarrow problem$.INITIAL
  for $t = 1$ **to** ∞ **do**
    $T \leftarrow schedule(t)$
    if $T = 0$ **then return** *current*
    $next \leftarrow$ a randomly selected successor of *current*
    $\Delta E \leftarrow$ VALUE(*current*) – VALUE(*next*)
    if $\Delta E < 0$ **then** $current \leftarrow next$
    else $current \leftarrow next$ only with probability $e^{-\Delta E/T}$

T typically nonnegative, decreases over time

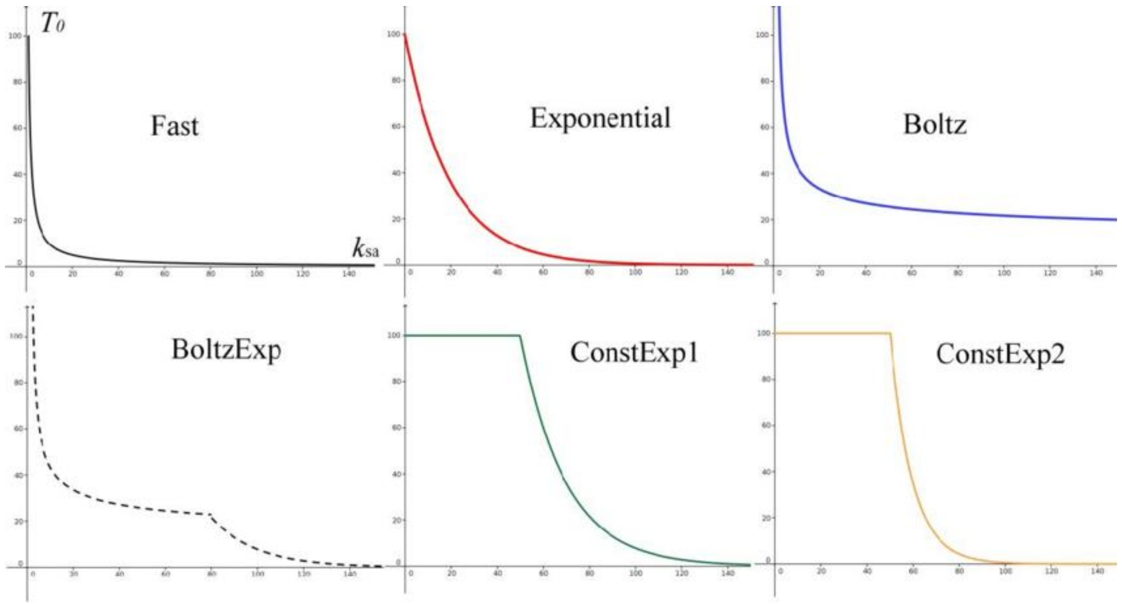
Move to *next* if it is better than *current*, otherwise do so with probability that decreases the worse the *next* state is

Simulated Annealing

- $current \leftarrow next$ only with probability $e^{-\Delta E/T}$
- The worse the successor, the larger the ΔE, and the lower the probability of moving

- T typically follows a *schedule* function
- Schedule choice is problem dependent

- High or constant value when starting, higher probability of exploring worse states
- Temperature decreases to 0 over time and algorithm will only improve current state



Local Beam Search

- Instead of starting with and maintaining one state, start with k states
- Each one generates a successor, and we keep the k best ones

- Unlike (regular) local search with k random restarts, local beam search quickly shares useful information among the parallel search threads
- Disadvantage: Potential lack of diversity in states in the threads

- *Stochastic* beam search: Randomly choose k successors to keep in frontier, with likelihoods proportional to state values

Evolutionary Algorithms

- Local beam search simulates evolution with asexual reproduction
- **Evolutionary algorithms** allow for *crossover* between parent states, allowing for generation of very different successor states!

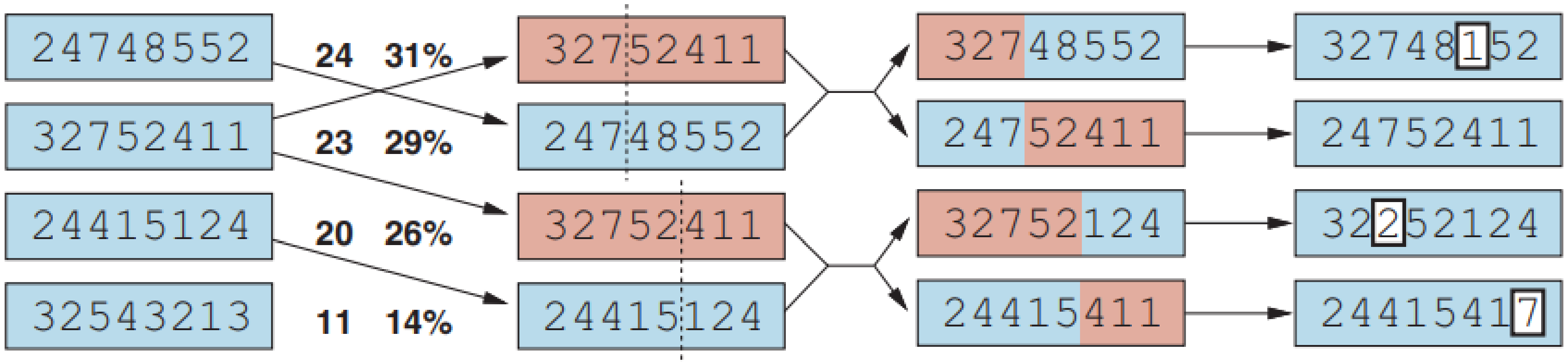
- Idea: The "fittest" among the current states produce "offspring"
  - Selection probabilities are a function of objective function values

- Successors are generated by "combining" two parent states together
- *Mutations* can also occur in children states for more variety

Genetic Algorithms

- In **genetic algorithms**, individuals are represented as bitstrings or other encodings
- A *fitness function* determines probability of being selected to produce offspring

- Successors are generated through *crossover* between two (or more) parents
- Successors may also be subject to random *mutations* in one (or more) of their bits



(a) Initial Population (b) Fitness Function (c) Selection (d) Crossover (e) Mutation

Genetic Algorithms

- If populations are diverse (usually early in the process), crossover can produce an even larger diversity of successors
- Corresponds to large transitions in the state space

- After many generations, higher fitness pushes the population to a smaller region of the state space, and successors are more similar to their parents

- Genetic algorithms can work well if state representations can be factored into functional “blocks” that can be combined in different ways

Summary

- Local search algorithms: Good for problems that only require goal states, not entire paths; very memory-efficient

- No systematic search of state space; fewer optimality and completeness guarantees; tuning of parameters often required

- Hill-climbing: Move toward neighboring states that look better
- Simulated annealing: Occasionally allow moves toward worse states
- Evolutionary algorithms: Combine multiple states to generate new ones