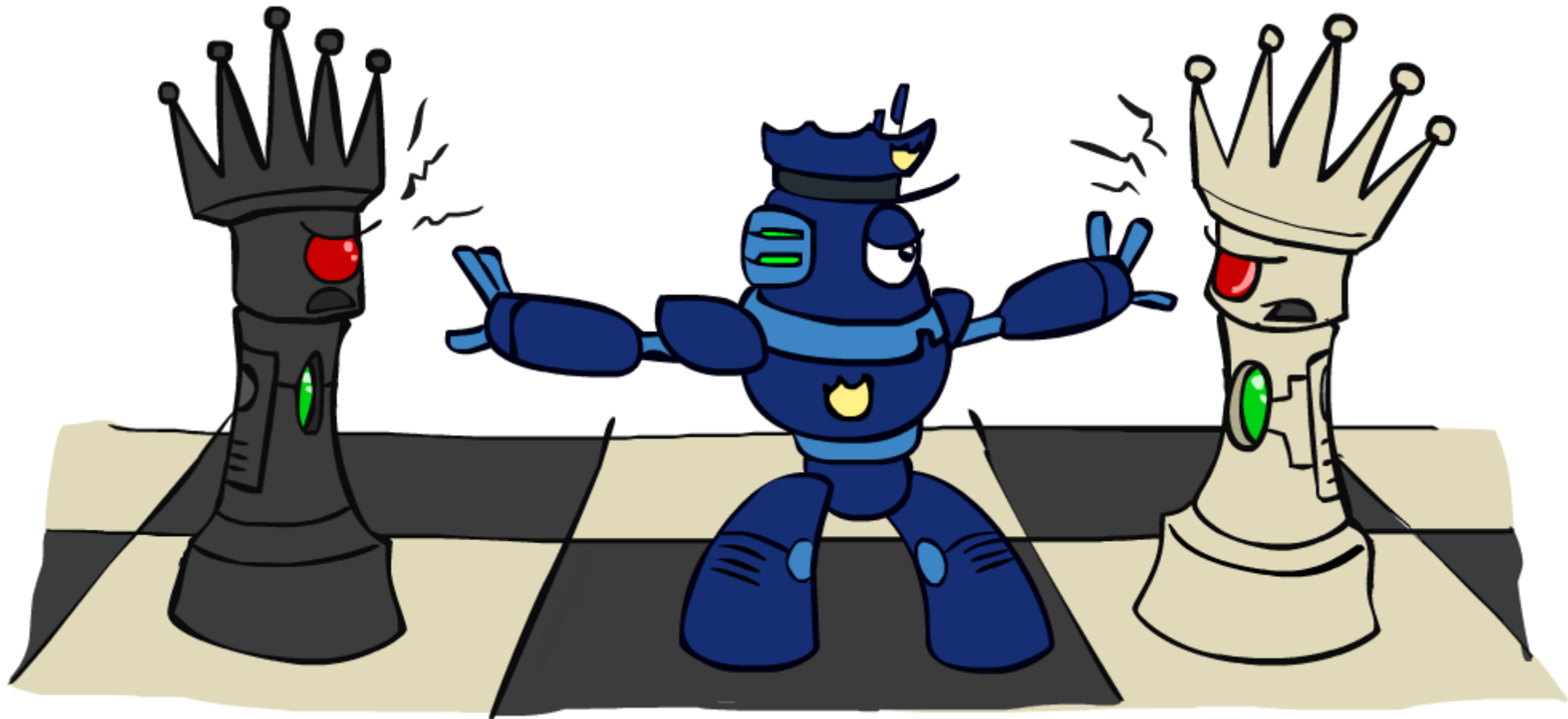


Local Search

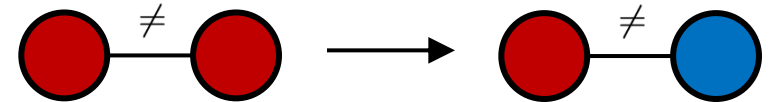


Iterative Improvement

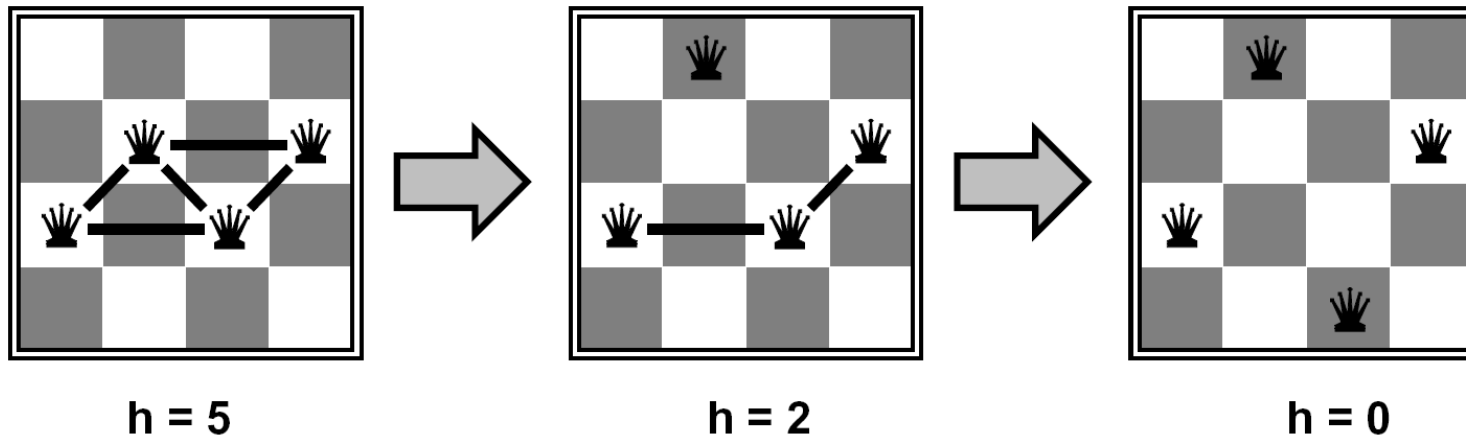


Iterative Algorithms for CSPs

- Local search methods typically work with “complete” states, i.e., all variables assigned
- To apply to CSPs:
 - Take an assignment with unsatisfied constraints
 - Operators *reassign* variable values
 - No fringe! Live on the edge.
- Algorithm: While not solved,
 - Variable selection: randomly select any conflicted variable
 - Value selection: min-conflicts heuristic:
 - Choose a value that violates the fewest constraints
 - I.e., hill climb with $h(n)$ = total number of violated constraints



Example: 4-Queens

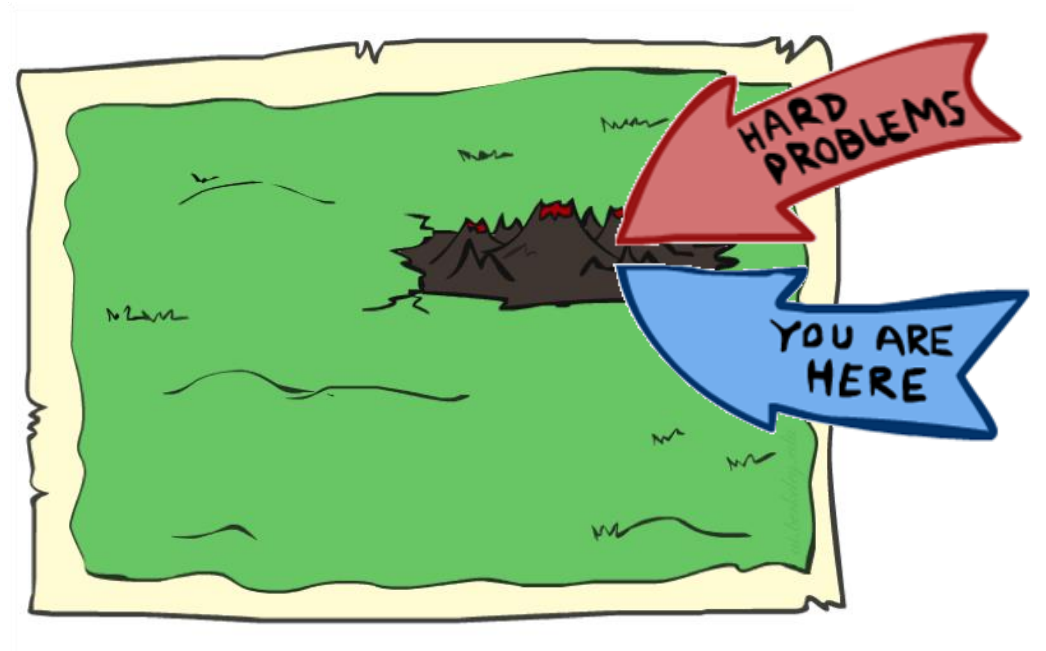
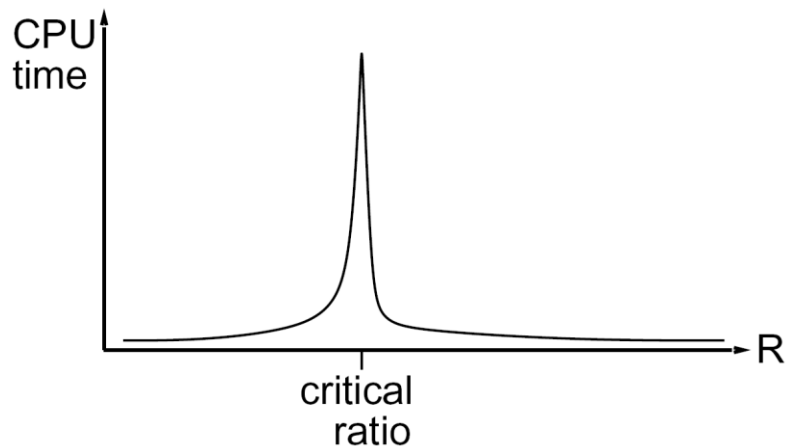


- States: 4 queens in 4 columns ($4^4 = 256$ states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: $c(n) =$ number of attacks

Performance of Min-Conflicts

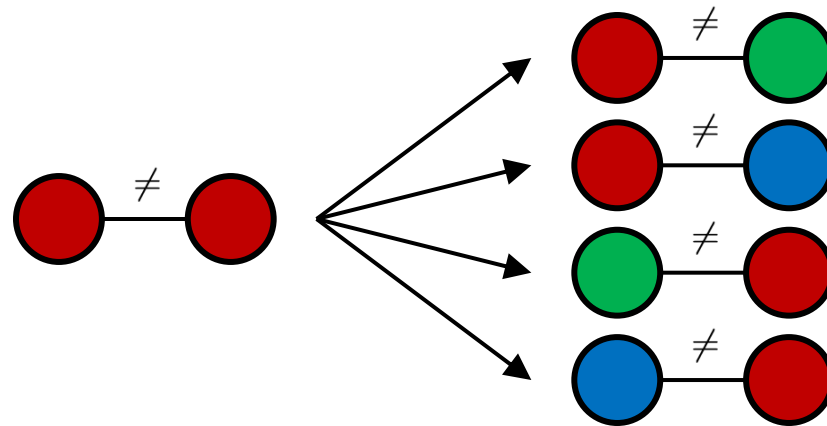
- Given random initial state, can solve n-queens in almost constant time for arbitrary n with high probability (e.g., n = 10,000,000)!
- The same appears to be true for any randomly-generated CSP *except* in a narrow range of the ratio

$$R = \frac{\text{number of constraints}}{\text{number of variables}}$$



Local Search

- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search: improve a single option until you can't make it better (no fringe!)
- New successor function: local changes



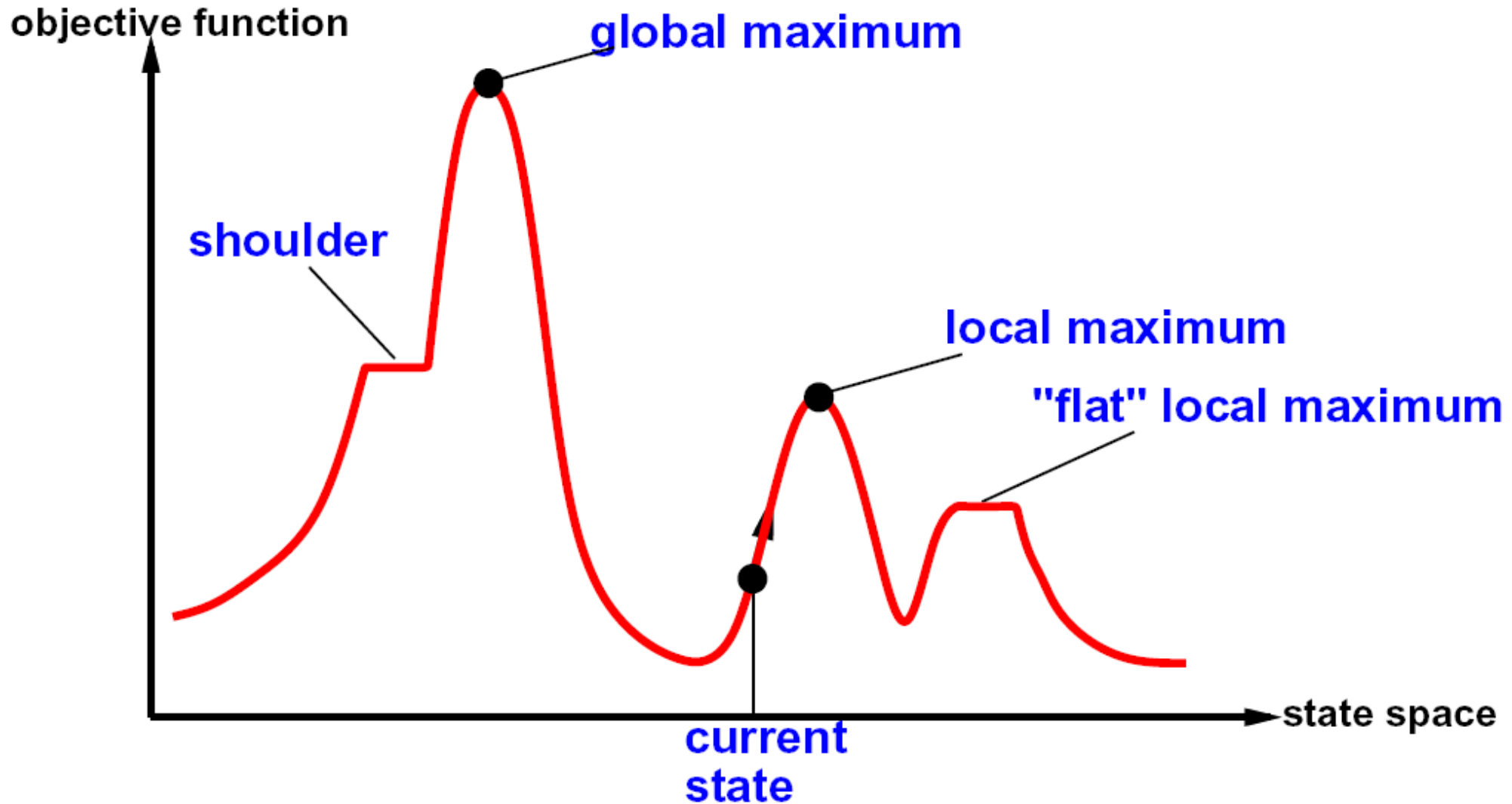
- Generally much faster and more memory efficient (but incomplete and suboptimal)

Hill Climbing

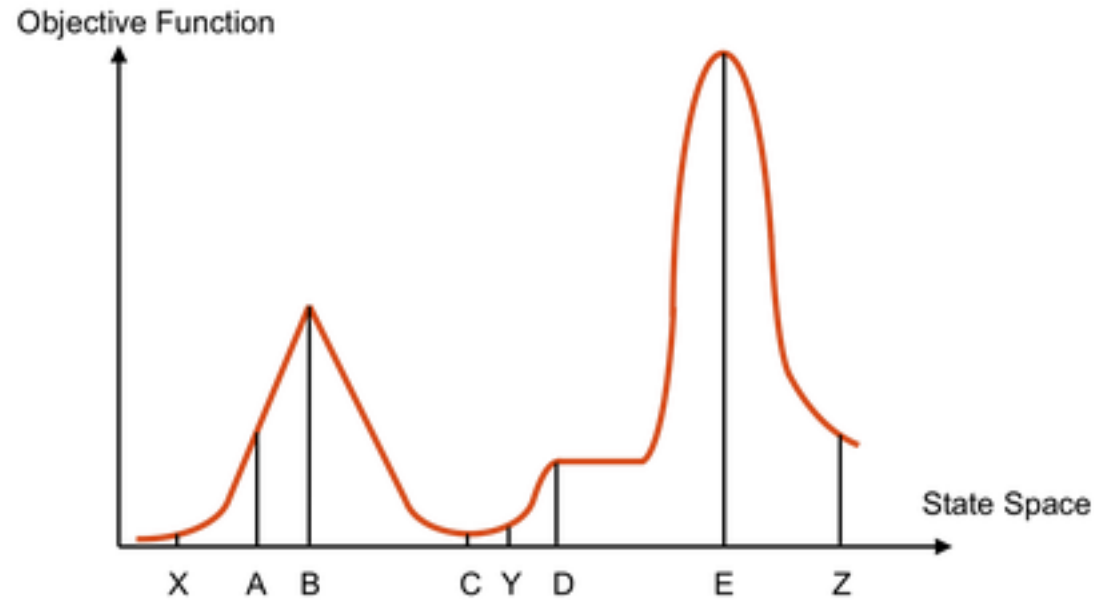
- Simple, general idea:
 - Start wherever
 - Repeat: move to the best neighboring state
 - If no neighbors better than current, quit
- What's bad about this approach?
 - Complete?
 - Optimal?
- What's good about it?



Hill Climbing Diagram



Hill Climbing Quiz



Starting from X, where do you end up ?

Starting from Y, where do you end up ?

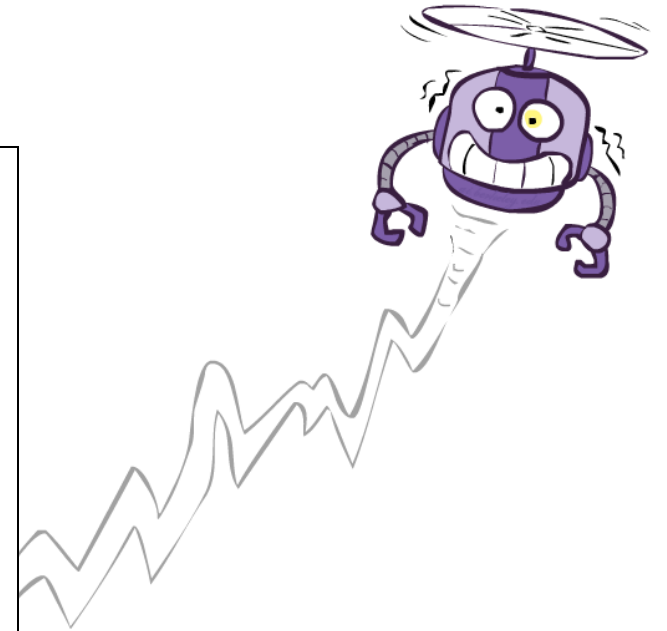
Starting from Z, where do you end up ?

Simulated Annealing

- Idea: Escape local maxima by allowing downhill moves
 - But make them rarer as time goes on

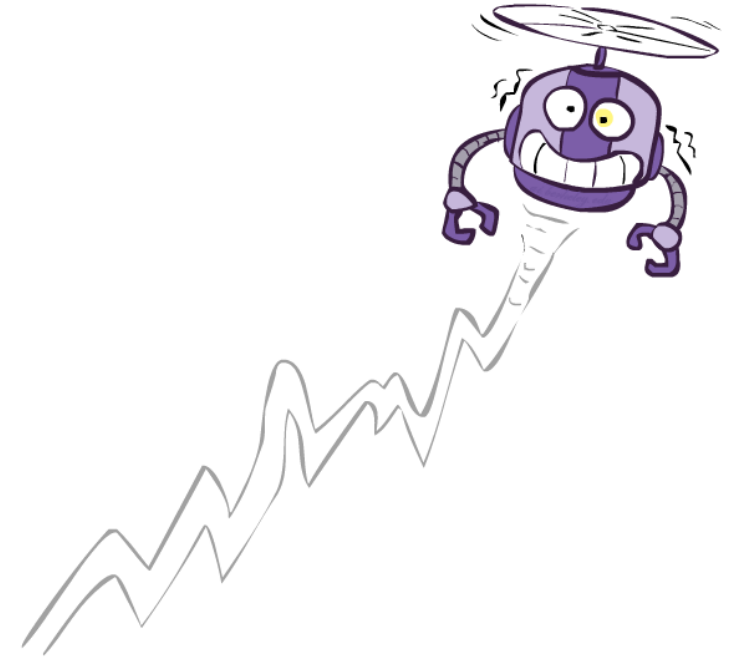
```
function SIMULATED-ANNEALING(problem, schedule) returns a solution state
  inputs: problem, a problem
           schedule, a mapping from time to "temperature"
  local variables: current, a node
                   next, a node
                   T, a "temperature" controlling prob. of downward steps

  current ← MAKE-NODE(INITIAL-STATE[problem])
  for t ← 1 to ∞ do
    T ← schedule[t]
    if T = 0 then return current
    next ← a randomly selected successor of current
     $\Delta E$  ← VALUE[next] - VALUE[current]
    if  $\Delta E > 0$  then current ← next
    else current ← next only with probability  $e^{-\Delta E/T}$ 
```

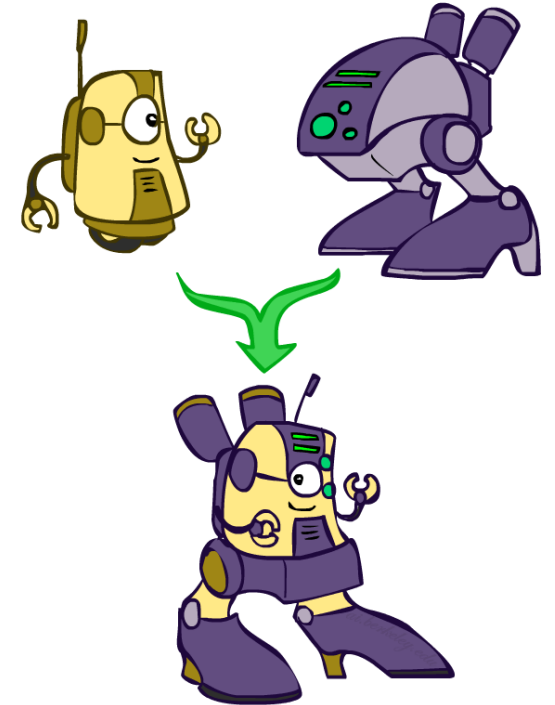
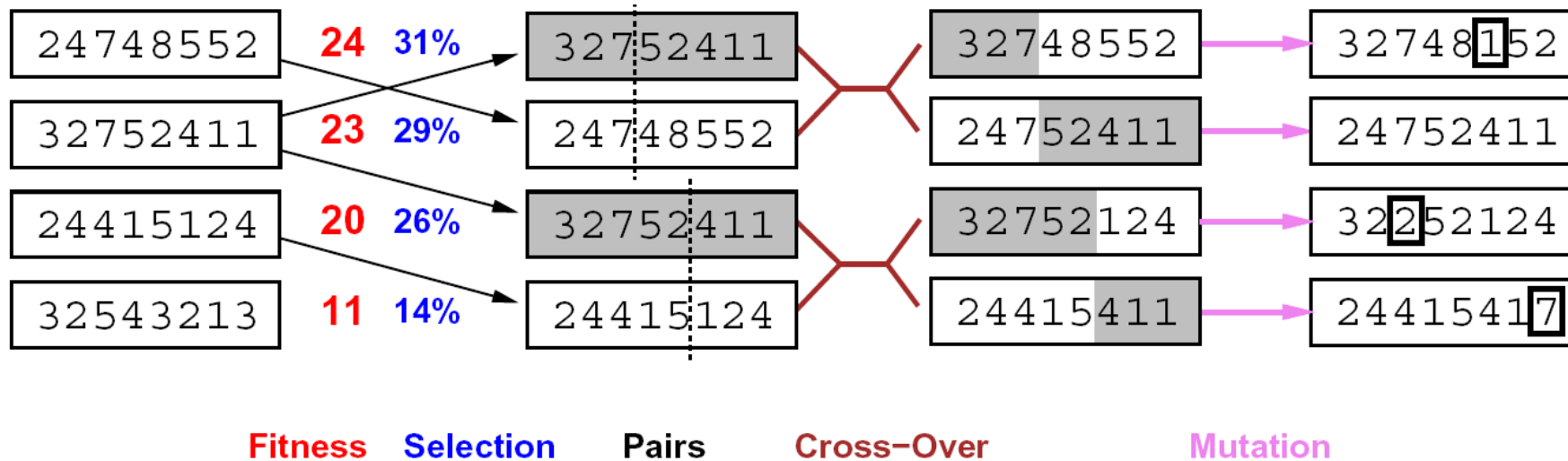


Simulated Annealing

- Theoretical guarantee:
 - Stationary distribution: $p(x) \propto e^{-\frac{E(x)}{kT}}$
 - If T decreased slowly enough, will converge to optimal state!
- Is this an interesting guarantee?
- Sounds like magic, but reality is reality:
 - The more downhill steps you need to escape a local optimum, the less likely you are to ever make them all in a row
 - People think hard about *ridge operators* which let you jump around the space in better ways

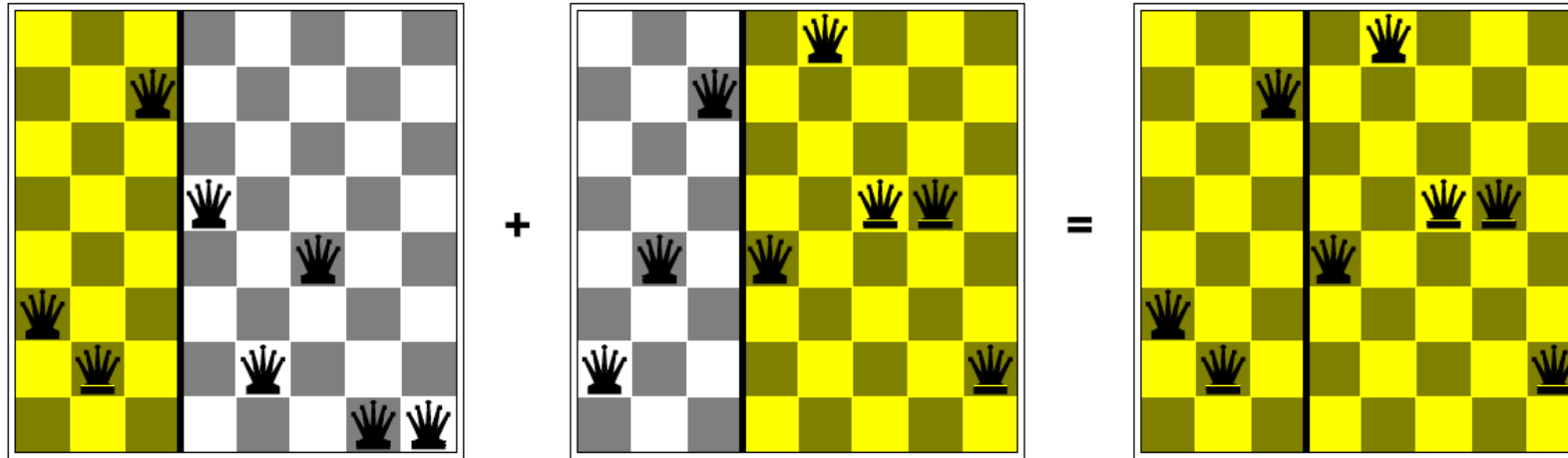


Genetic Algorithms



- Genetic algorithms use a natural selection metaphor
 - Keep best N hypotheses at each step (selection) based on a fitness function
 - Also have pairwise crossover operators, with optional mutation to give variety
- Possibly the most misunderstood, misapplied (and even maligned) technique around

Example: N-Queens



- Why does crossover make sense here?
- When wouldn't it make sense?
- What would mutation be?
- What would a good fitness function be?