Artificial Intelligence

Module 3: Search Strategies

PART 3.4: Local Search

Dr. Chandra Prakash

Assistant Professor

Department of Computer Science and Engineering

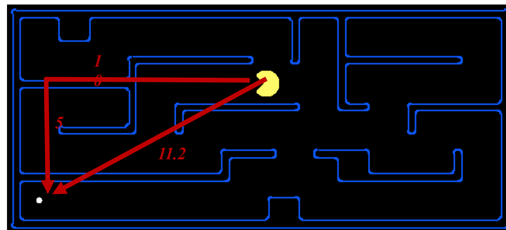
(Slides adapted from StuartJ. Russell, B Ravindran, Mausam, Prof. Pallab Dasgupta, Prof. Partha Pratim Chakrabarti, Saikishor Jangiti

Module 3: Search Strategies

- PART 3.1: Search
- PART 3.2: Uninformed Search
- PART 3.3: Informed/Heuristic Search
- PART 3.4: Beyond Classical Search
  - Local Search
    - Generate-and-test
    - Hill climbing
    - Simulated Analing
  - Problem reduction
- PART 3.5: Constraint Satisfaction Problems
- PART 3.6: Adversarial Search

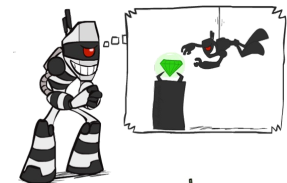
Recap: Search

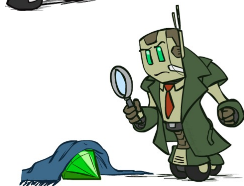
- Search problem:
  - States (configurations of the world)
  - Actions and costs
  - Successor function (world dynamics)
  - Start state and goal test
- Search tree:
  - Nodes: represent plans for reaching states
  - Plans have costs (sum of action costs)
- Search algorithm:
  - Systematically builds a search tree
  - Chooses an ordering of the fringe (unexplored nodes)
  - Optimal: finds least-cost plans
- A heuristic is:
  - A function that estimates how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing



What is Search For?

- Assumptions about the world: a single agent, deterministic actions, fully observed state, discrete state space
- Planning: sequences of actions
  - The path to the goal is the important thing
  - Paths have various costs, depths
  - Heuristics give problem-specific guidance
- Identification: assignments to variables
  - The goal itself is important, not the path
  - All paths at the same depth (for some formulations)
  - CSPs are specialized for identification problems





Path vs. State Optimization

- **So far:** Problems were observable, determinsitic, known evnironment
  - Path to goal is solution to problem
  - Systematic exploration of search space.
- For many problems, the solution path is irrelevant,
  - The goal state is itself the solution to the problem
  - e.g 8-queens
    - what matter is the final configuration, not the order in which they are added.
  - relax the assumption of determinism and observability
- Constant space

Path vs. State Optimization

- State space = set of "complete" configurations
  - Goal is to find an "optimal/close to optimal" configuration satisfying constraints
- Iterative improvement algorithms
  - Keep a single "current" state and try to improve it
- Applications
  - integrated circuit design
  - factory-floor layout
  - job-shop scheduling
  - Exam time table
  - automatic programming
  - telecommunications
  - network optmization
  - vehicle routing, etc.

Satisfaction vs. Optimization

Goal Satisfaction	**Optimization**
• reach the goal node	• optimize (objective fn)
• Constraint satisfaction	• Constraint Optimization

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

More Search Methods

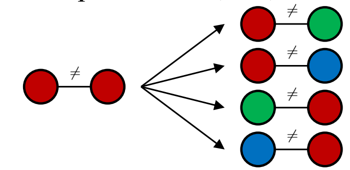
- Complete state formulation
  - For example, for the 8 queens problem, all 8 queens are on the board and need to be moved around to get to a goal state
- Equivalent to optimization problems often found in science and engineering
- Start somewhere and try to get to the solution from there
- Local search around the current state to decide where to go next
- **Local Search**
  - Hill Climbing
  - Simulated Annealing
  - Beam Search
  - Genetic Search

Local Search and Optimization

- **Local search**
  - Keep track of single current state
  - Move only to neighboring states
  - Ignore paths
- **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.

Local Search – Iterative Improvement Algorithms

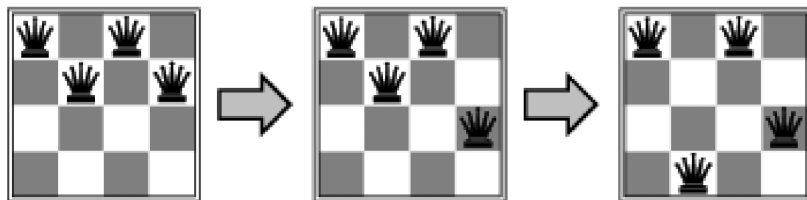
- Tree search keeps unexplored alternatives on the fringe (ensures completeness)
- Local search:
  - improve a single option until you can't make it better (no fringe!)
- Local search methods typically work with "complete" states, i.e., all variables assigned
- New successor function:
  - local changes



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

Example: n-queens

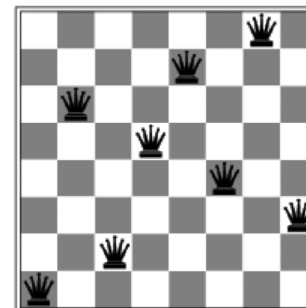
- Put n queens on an n x n board with no two queens on the same row, column, or diagonal



- Is it a satisfaction problem or optimization

8-queens problem

- Is this a solution?
- What is h?



- Need to convert to an optimization problem

  h = number of pairs of queens that are attacking each other

  h = 17 for the above state

18	12	14	13	13	12	14	14
14	16	13	15	12	14	12	16
14	12	18	13	15	12	14	14
15	14	14	♛	13	16	13	16
♛	14	17	15	♛	14	16	16
17	♛	16	18	15	♛	15	♛
18	14	♛	15	15	14	♛	16
14	14	13	17	12	14	12	18

Search Space

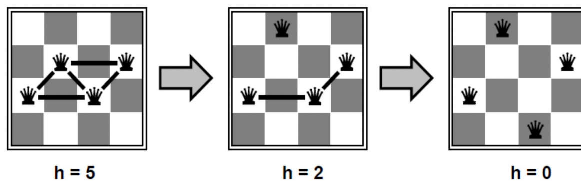
- State
  - All 8 queens on the board in some configuration
- Successor function
  - move a single queen to another square in the same column.
- Example of a heuristic function h(n):
  - the number of pairs of queens that are attacking each other
  - (so we want to minimize this)

Trivial Algorithms

- Random Sampling
  - Generate a state randomly
- Random Walk
  - Randomly pick a neighbor of the current state
- Both algorithms asymptotically complete.

Example: 4-Queens

- Move a queen to reduce number of conflicts

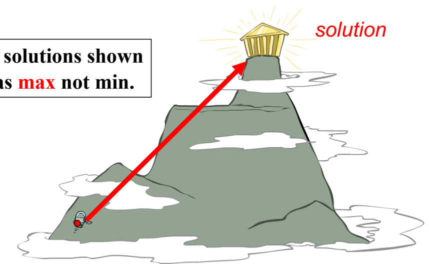


- States: 4 queens in 4 columns (4^4 = 256 states)
- Operators: move queen in column
- Goal test: no attacks
- Evaluation: c(n) = number of attacks

Hill Climbing

- Searching for a goal state = Climbing to the top of a hill
- Generate-and-test + direction to move (feedback from test procedure).
- Test function + heuristic function = Hill Climbing
- Heuristic function (objective function) to estimate how close a given state is to a goal state.
- Often used when a good heuristic function is available for evaluating states but when no other useful knowledge is available.

Note: solutions shown here as max not min.



- Often used for numerical optimization problems.
- How does it work?

Loacl Search

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"

Hill-climbing (Greedy Local Search) max version

```
function HILL-CLIMBING( problem) return a state that is a local maximum
input: problem, a problem
local variables: current, a node.
                 neighbor, a node.
current ←MAKE-NODE(INITIAL-STATE[problem])
loop do
neighbor ←  a highest valued successor of current
if VALUE [neighbor] ≤ VALUE[current] then return STATE[current]
current ←   neighbor
```

- min version will reverse inequalities and look for lowest valued successor
- **What's bad about this approach?**
- **What's good about it?**

Simple Hill Climbing

- Evaluation function as a way to inject task-specific knowledge into the control process.
- Key difference between Simple Hill climbing and Generate-and-test is the use of evaluation function as a way to inject task specific knowledge into the control process.
- Is on state better than another ? For this algorithm to work, precise definition of better must be provided.
- Better :
  - higher value of heuristic function
  - Lower value

Simple Hill Climbing

- **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

Example: coloured blocks

Heuristic function: the sum of the number of different colours on each of the four sides (solution = 16).

Steepest-Ascent Hill Climbing

- Considers all the moves from the current state.
- Selects the best one as the next state.
- Also known as Gradient Search.

- Example: coloured blocks
  - Consider all perturbation of initial state and choose best one.
- Tread-off between time required to select a move or no of the move required to get a solution

Algorithm
1. Evaluate the initial state.
2. Loop until a solution is found or a complete iteration produces no change to current state:
   - X = a state such that any possible successor of the current state will be better than SUCC (the worst state).
   - For each operator that applies to the current state, evaluate the new state:
     goal → quit
     better than X → set X to this state
   - X is better than the current state → set the current state to X.

Steepest ascent Hill climbing vs Best first search

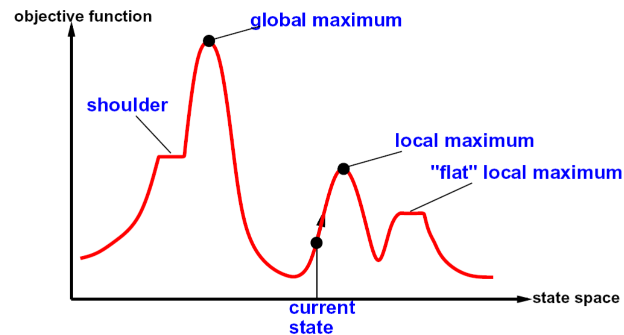
Similar to Steepest ascent hill climbing with two exceptions:

1. In hill climbing, one move is selected and all the others are rejected, never to be reconsidered. This produces the straight-line behaviour that is characteristic of hill climbing.
   - In BFS, one move is selected, but the others are kept around so that they can be revisited later if the selected path becomes less promising
2. The best available state is selected in the BFS, even if that state has a value that is lower than the value of the state that was just explored.
   - Whereas in hill climbing the progress stop if there are no better successor nodes.

Hill-climbing on 8-queens

- Randomly generated 8-queens starting states…
- 14% the time it solves the problem
- 86% of the time it get stuck at a local minimum
- However…
  - Takes only 4 steps on average when it succeeds
  - And 3 on average when it gets stuck
  - (for a state space with 8^8 =~17 million states)

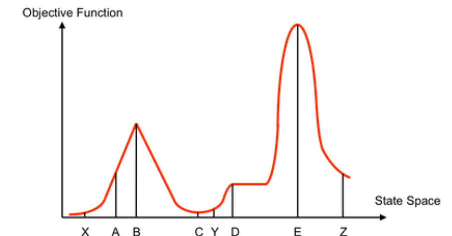
“Landscape” of search



Hill Climbing gets stuck in local minima
depending on?

Hill Climbing: Quiz

- Fail to find a solution
- Either Algo may terminate not by finding a goal state but by getting to a state from which no better state can be generated.
- This happen if program reached
  - Local maximum,
  - Plateau,
  - Ridge.



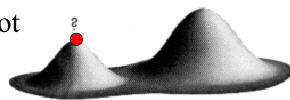
Starting from X, where do you end up ?

Starting from Y, where do you end up ?

Starting from Z, where do you end up ?

Hill Climbing: Disadvantages

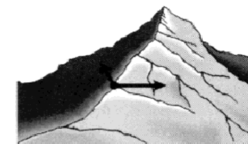
Local maximum
A state that is better than all of its neighbours, but not better than some other states far away.



Plateau
A flat area of the search space in which all neighbouring states have the same value.
- requiring random walk

Hill Climbing: Disadvantages

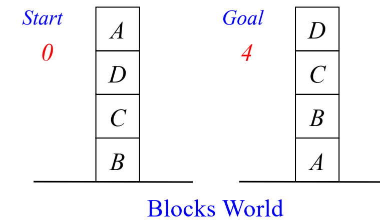
Ridge
- Special kind of local maximum.
- The orientation of the high region, compared to the set of available moves, makes it impossible to climb up.
- Many moves executed serially may increase the height.



Hill Climbing: Disadvantages

- Hill climbing is a local method:
  Decides what to do next by looking only at the "immediate" consequences of its choices rather than by exhaustively exploring all the consequences.
- Global information might be encoded in heuristic functions.

Hill Climbing: Blocks World

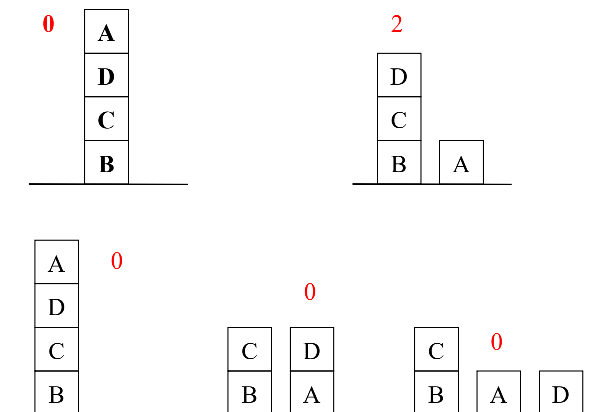


Blocks World

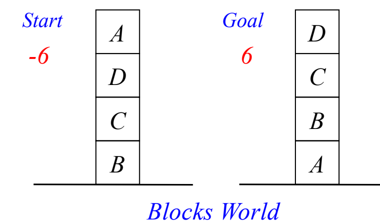
Local heuristic:
+1 for each block that is resting on the thing it is supposed to be resting on.

-1 for each block that is resting on a wrong thing.

Hill Climbing:



Hill Climbing:

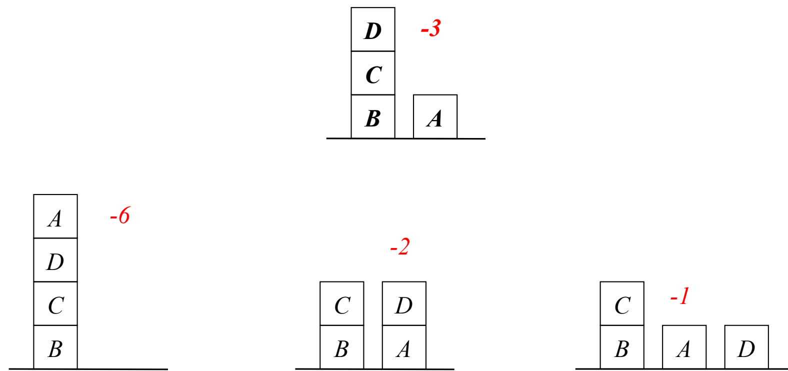


Blocks World

Global heuristic:

- For each block that has the correct support structure: +1 to every block in the support structure.
- For each block that has a wrong support structure: -1 to every block in the support structure.

Hill Climbing:



Hill Climbing

- Can be very inefficient in a large, rough problem space.
- Global heuristic may have to pay for computational complexity.
- Often useful when combined with other methods, getting it started right in the general neighbourhood.

Escaping Shoulders: Sideways Move

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

Hill Climbing: Ways Out

Ways Out

- Backtrack to some earlier node and try going in a different direction. (good way in dealing with local maxima)
- Make a big jump to try to get in a new section. (good way in dealing with plateaus)
- Moving in several directions at once. (good strategy for dealing with ridges)

Tabu Search

- prevent returning quickly to the same state
- Keep fixed length queue (“tabu list”)
- add most recent state to queue; drop oldest
- Never make the step that is currently tabu’ed
- 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

Escaping Shoulders / Local Optima Enforced Hill Climbing

- Perform breadth first search from a local optima
  - to find the next state with better h function
- Typically,
  - prolonged periods of exhaustive search
  - bridged by relatively quick periods of hill-climbing
- Middle ground b/w local and systematic search

Hill-climbing: Stochastic Variations

- Stochastic hill-climbing
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.
- To avoid getting stuck in local minima
  - Random-walk hill-climbing
  - Random-restart hill-climbing
  - Hill-climbing with both

Hill Climbing with random walk

- 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: Put random walk into greedy hill-climbing
- At each step do one of the two
  - 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?
- **If you want to pick one local search algorithm, Which one you will change**

Hill-climbing with both

- 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: Resample a new current state

Simulated Annealing

Simulated Annealing

- A variation of hill climbing in which, at the beginning of the process, some downhill moves may be made.
- To do enough exploration of the whole space early on, so that the final solution is relatively insensitive to the starting state.
- Lowering the chances of getting caught at a local maximum, or plateau, or a ridge.

- A alternative to a random-restart hill-climbing when stuck on a local maximum is to do a 'reverse walk' to escape the local maximum.
- This is the idea of simulated annealing.
- The term simulated annealing derives from the roughly analogous physical process of heating and then slowly cooling a substance to obtain a strong crystalline structure.
- The simulated annealing process lowers the temperature by slow stages until the system ``freezes" and no further changes occur.

Simulated Annealing

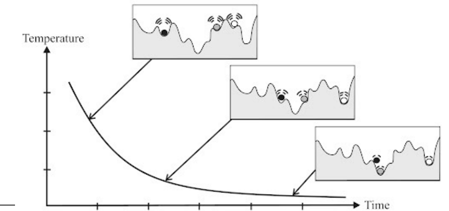
- Simulated Annealing = physics inspired twist on random walk
- Basic ideas:
  - like hill-climbing identify the quality of the local improvements
  - instead of picking the best move, pick one randomly
  - say the change in objective function is d
  - if d is positive, then move to that state
  - otherwise:
    - move to this state with probability proportional to d
    - thus: worse moves (very large negative d) are executed less often
  - however, there is always a chance of escaping from local maxima
  - over time, make it less likely to accept locally bad moves
  - (Can also make the size of the move random as well, i.e., allow "large"steps in state space)

Simulated Annealing

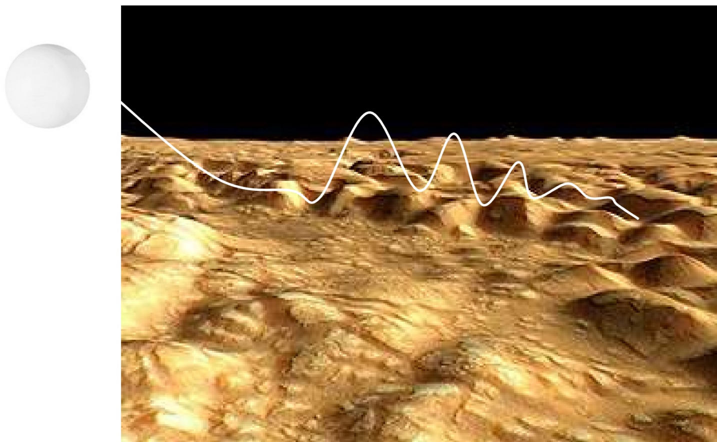
- Idea:
  - Escape local maxima by allowing downhill moves
  - But make them rarer as time goes on

```
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
       ΔE ← VALUE[next] – VALUE[current]
       if ΔE > 0 then current ← next
       else current ← next only with probability e^(ΔE/T)
```



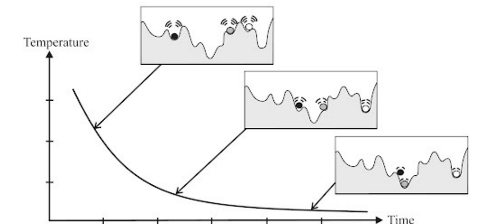
Ping-Pong Ball Example



Simulated Annealing and Temperature T

- Probability of transition to higher energy state is given by function:
  - $P = e^{-\Delta E/kT}$

  Where
  - ΔE is the positive change in the energy level
  - T is the temperature
  - K is Boltzmann constant.



- **Annealing schedule:** if the temperature is lowered sufficiently slowly, then the goal will be attained.
  - 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"

Physical Interpretation of Simulated Annealing

- A Physical Analogy:
  - imagine letting a ball roll downhill on the function surface
    - this is like hill-climbing (for minimization)
  - now imagine shaking the surface, while the ball rolls, gradually reducing the amount of shaking
    - this is like simulated annealing
- Annealing = physical process of cooling a liquid or metal until particles achieve a certain frozen crystal state
  - simulated annealing:
    - free variables are like particles
    - seek "low energy" (high quality) configuration
    - slowly reducing temp. T with particles moving around randomly

Simulate Annealing: Implementation

- It is necessary to select an annealing schedule which has three components:
  - Initial value to be used for temperature
  - Criteria that will be used to decide when the temperature will be reduced
  - Amount by which the temperature will be reduced.

Simulated Annealing in Practice

method proposed in 1983 by IBM researchers for solving VLSI layout problems (Kirkpatrick et al, *Science*, 220:671-680, 1983).

- theoretically will always find the global optimum

– Other applications:
- Traveling salesman, Graph partitioning, Graph coloring, Scheduling, Facility Layout, Image Processing, …

– useful for some problems, but can be very slow
- slowness comes about because T must be decreased very gradually to retain optimality

Local Search with Multiple Present States

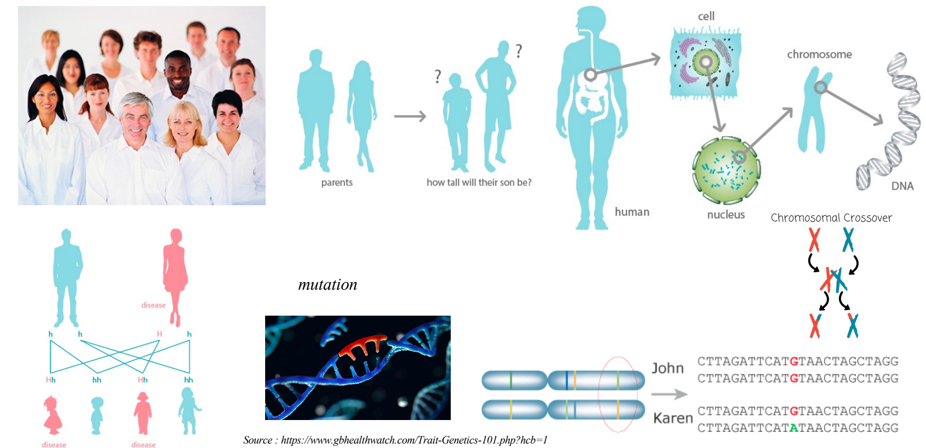
- Instead of working on only one configuration/ neighbor at any time, we could work on multiple promising configurations concurrently

- LOCAL BEAM SEARCH
  - Maintain k states rather than just one. Begin with k randomly generated states
  - In each iteration, generate all the successors of all k states
  - Stop if a goal state is found; otherwise Select the k best successors from the complete list and repeat
- GENETIC ALGORITHMS
  - States are strings over a finite alphabet (genes). Begin with k randomly generated states (population).
  - Select individuals for next generation based on a fitness function.
  - Two types of operators for creating the next states:
    - Crossover: Fit parents to yield next generation (offspring)
    - Mutation: Mutate a parent to create an offspring randomly with some low probability

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
- Equivlent to k random-start searches ?
  - 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!

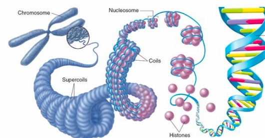
Genetic algorithms: We and Genetics



Source : https://www.gbhealthwatch.com/Trait-Genetics-101.php?hcb=1

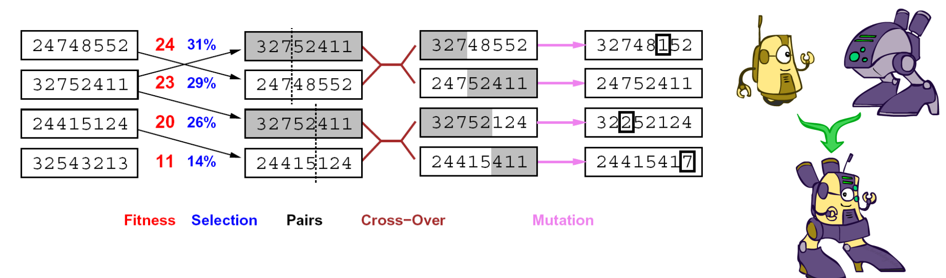
Genetic algorithms

- Twist on Local Search: successor is generated by combining two parent states
- A state is represented as a string over a finite alphabet (e.g. binary)
  - 8-queens
    - State = position of 8 queens each in a column



- Start with k randomly generated states (population)
- Evaluation function (fitness function):
  - Higher values for better states.
  - Opposite to heuristic function, e.g., # non-attacking pairs in 8-queens
- Produce the next generation of states by “simulated evolution”
  - Random selection
  - Crossover
  - Random mutation

Genetic Algorithms

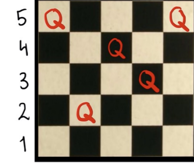


- Genetic algorithms use a natural selection metaphor
  - Keep best N hypotheses at each step (selection) based on a fitness function
  - Also have pairwise crossover operators, with optional mutation to give variety

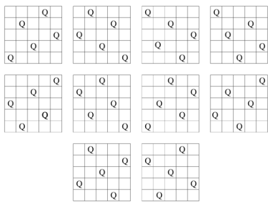
Genetic Algo

1) Chromosome Design
2) Initialization
3) Fitness evaluation
4) Selection
5) Crossover
6) Mutation
7) Update generation
8) Go back to 3)

1) Chromosome Design

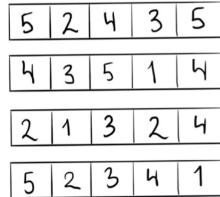


Chromosome: 5 2 4 3 5

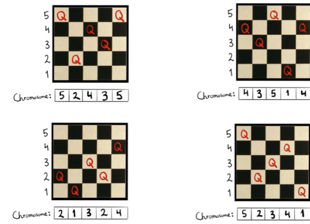


2) Initialization:

Lets take our Initial population:



In particularly, chromosomes represented as the following on board:



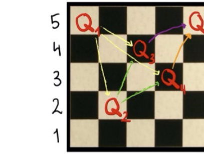
3. Fitness Evaluation

Fitness function = F1+F2+F3+F4+F5

where:

F1 = number of pairs of nonattacking queens with queen Q1.
F2 = number of pairs of nonattacking queens with queen Q2.
F3 = number of pairs of nonattacking queens with queen Q3.
F4 = number of pairs of nonattacking queens with queen Q4.
F5 = number of pairs of nonattacking queens with queen Q5.

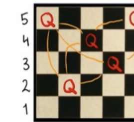
Fitness evaluation:



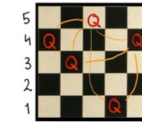
Here, I assigned queens as Q_1, Q_2, Q_3, Q_4, Q_5.
=>
Fitness Function = $F_1+F_2+F_3+F_4+F_5$

where:
F_1 = number of pairs of nonattacking queens with queen Q_1.
F_2 = number of pairs of nonattacking queens with queen Q_2.
F_3 = number of pairs of nonattacking queens with queen Q_3.
F_4 = number of pairs of nonattacking queens with queen Q_4.
F_5 = number of pairs of nonattacking queens with queen Q_5.

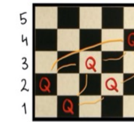
Fitness function = 3+2+1+1+0 = 7



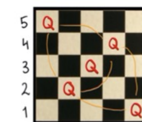
Fitness function = 7



Fitness function = 6



Fitness function = 6



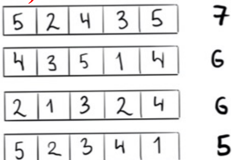
Fitness function = 5

=>

Chromosome	Fitness
5 2 4 3 5	7
4 3 5 1 4	6
2 1 3 2 4	6
5 2 3 4 1	5

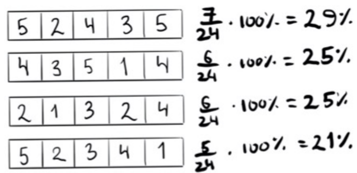
All Correct ????

4) Selection



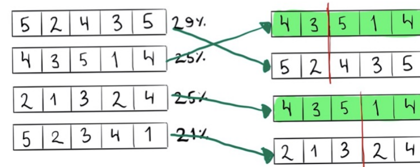
Chromosome	Fitness
5 2 4 3 5	7
4 3 5 1 4	6
2 1 3 2 4	6
5 2 3 4 1	5

- first, we need to add all fitness functions => 7+6+6+5 = 24
- then we need to compute probability of being chosen from fitness function.



Chromosome	Probability
5 2 4 3 5	$\frac{7}{24} \cdot 100\% = 29\%$
4 3 5 1 4	$\frac{6}{24} \cdot 100\% = 25\%$
2 1 3 2 4	$\frac{6}{24} \cdot 100\% = 25\%$
5 2 3 4 1	$\frac{5}{24} \cdot 100\% = 21\%$

Selection



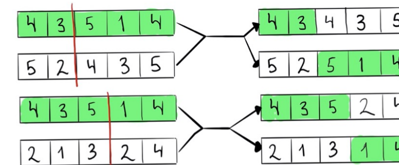
5. Crossover :

Randomly choose the two pair to reproduce based on probabilities
[4 3 5 1 4]
[5 2 4 3 5]
[4 3 5 1 4]
[2 1 3 2 4]

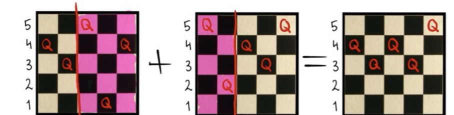
For the first pair
The crossover point will be picked after two genes.

In the case of the second pair
The crossover point will be picked after three genes.

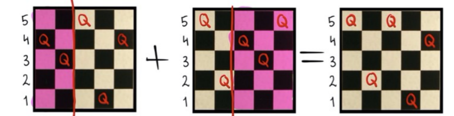
Crossover



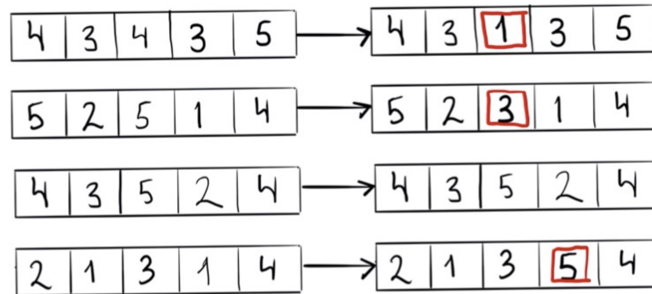
Creation of first child:

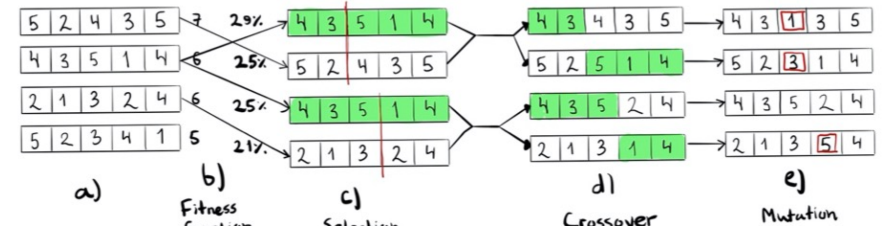


Creation of second child:

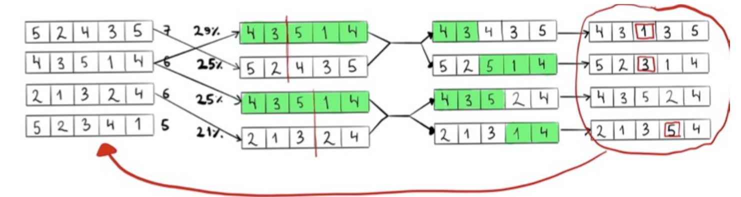


6. Mutation





7) Update generation



8) Go back to 3)

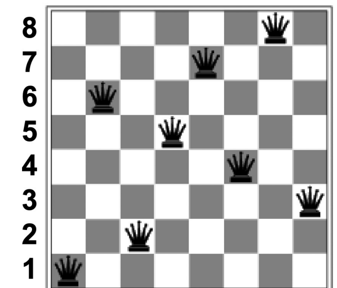
3) Fitness evaluation

Go back to step 3 (finding fitness function)

Chromosome	Fitness	Percentage
4 3 1 3 5	8	25.8%
5 2 3 1 4	8	25.8%
4 3 5 2 4	8	25.8%
2 1 3 5 4	7	22.6%

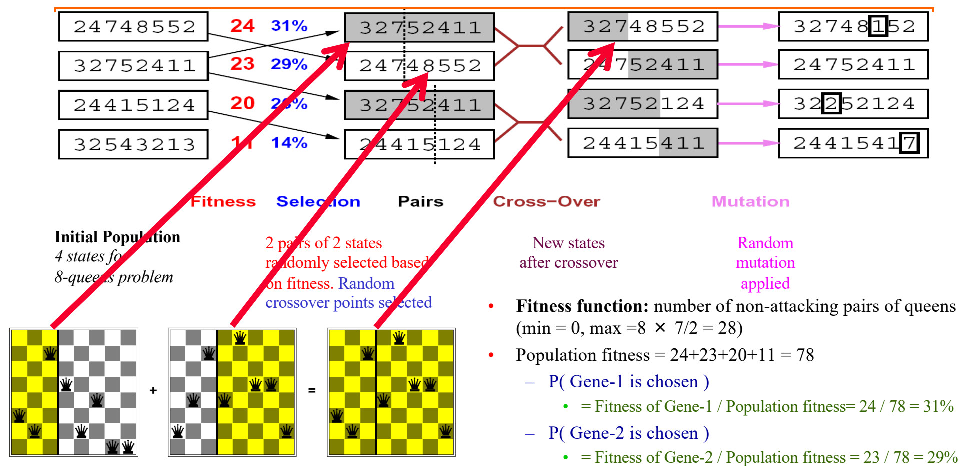
Can we evolve 8-queens through genetic algorithms?

- **1) Chromosome Design**
  2) Initialization
  3) Fitness evaluation
  4) Selection
  5) Crossover
  6) Mutation
  7) Update generation
  8) Go back to 3)



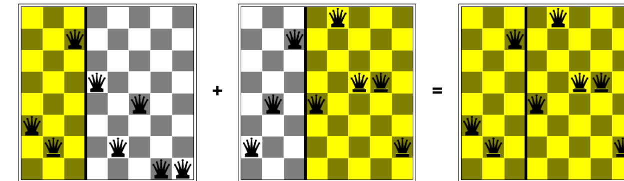
- **Chromosome Design**
  String representation : 16257483

Genetic Algorithms



- **Fitness function:** number of non-attacking pairs of queens (min = 0, max =8 × 7/2 = 28)
- Population fitness = 24+23+20+11 = 78
  - P(Gene-1 is chosen)
    - = Fitness of Gene-1 / Population fitness= 24 / 78 = 31%
  - P(Gene-2 is chosen)
    - = Fitness of Gene-2 / Population fitness = 23 / 78 = 29%

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?

Comments on Genetic Algorithms

- Genetic algorithm is a variant of "stochastic beam search"
- Pros
  - 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!
- Cons
  - 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

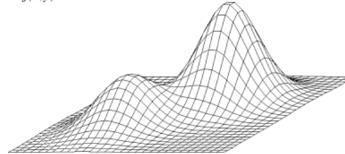
Question
  - are GAs really optimizing the individual fitness function? Mixability?

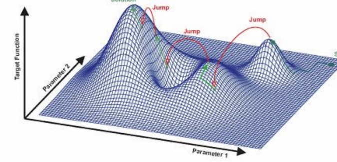
More in Local Search

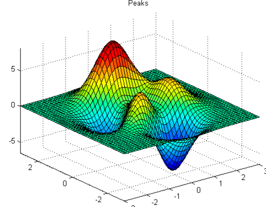
- Loacl search in continuous space
  - non of the algo expect first-choice hill climbing and simulated anneling, can handle continous state and action space because they have infinite branching factors.

Optimization of Continuous Functions

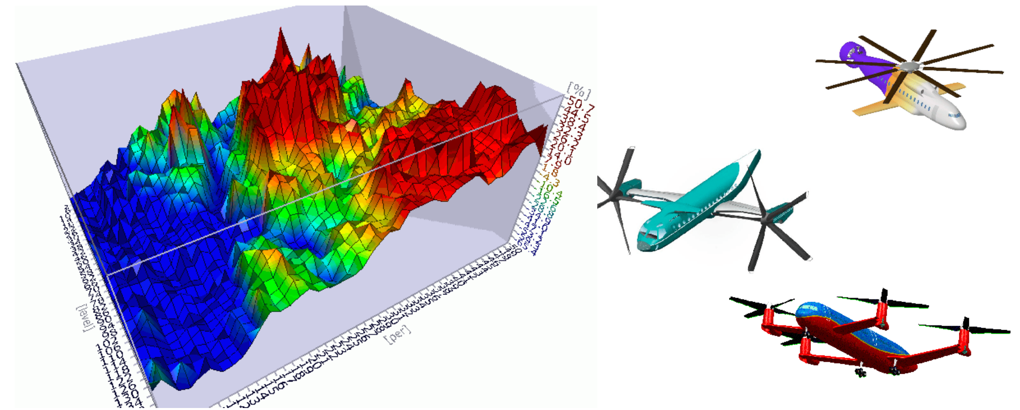
- Discretization
  - use hill-climbing
- Gradient descent
  - Assume we have a continuous function:
  - make a move in the direction of the gradient
    - gradients: closed form or empirical







Real world problem and Applicatons

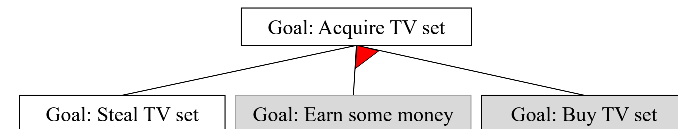


Partially Observable / Nondeterministic Environment

- When the environment is either partially observable or nondeterministic (or both), percepts become useful.
  - Partially observable environment:
    - every percept helps narrow down the set of possible states the agent might be in, thus making it easier for the agent to achieve its goals.
  - Nondeterministic environment:
    - percepts tell the agent which of the possible outcomes of its actions has actually occurred.
  - In both cases, the future percepts cannot be determined in advance and the agent's future actions will depend on those future percepts.
- CONTINGENCY PLAN
  - the solution to a problem is not a sequence but a contingency plan
  - also known as a STRATEGY
    - specifies what to do depending on what percepts are received.

Solution to nondeterministic problem

- Problem Reduction
  - AND-OR graph (or tree) is useful for representing the solution of problems that can be solved by decomposing them into a set of smaller problems, all of which must then be solved.
  - One AND arc may point to any number of successor nodes, all of which must be solved in order for the arc to point to a solution.

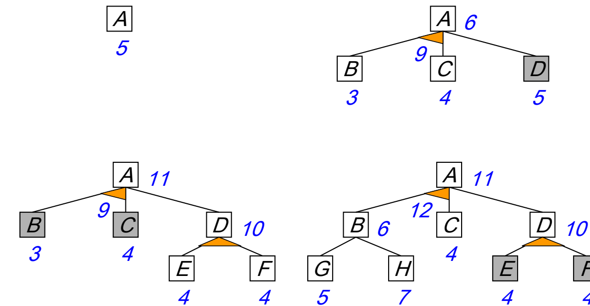


AND-OR Graphs

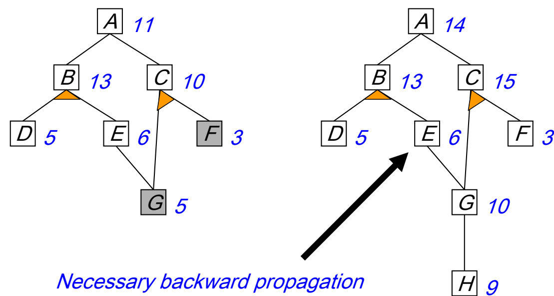
Algorithm AO (Martelli & Montanari 1973, Nilsson 1980)*

- AO* does not explore all the solution paths once it got a solution

Problem Reduction: AO*



Problem Reduction: AO*



Problem Reduction: AO*

- Nilsson calls it the AO* algorithm, combination of DFS and BFS
- Rather than the two lists, OPEN and CLOSED, that were used in the A* algorithm, the AO* algorithm will use a single structure GRAPH, representing the part of the search graph that has been explicitly generated so far.
- Each node in the graph will point both down to its immediate successors and up to its immediate predecessors.
- Each node in the graph will also have associated with it an h' value, an estimate of the cost of a path from itself to a set of solution nodes.
- We will not store g (the cost of getting from the start node to the current node) as did in the A* algorithm.

Problem Reduction: AO*

- It is not possible to compute a single such value since there may be many path to the same state.
- And such a value is not necessary because of the top-down travelling of the best-known path, which guarantees that only nodes that are on the best path will ever be considered for expansion.
- So h’ will serve as the estimate of goodness of a node.

Searching with Partial Observations

- The agent does not always know its state!
- Instead, it maintains a
  - belief state: a set of possible states it might be in.
  - Example: a robot can be used to build a map of a hostile environment. It will have sensors that allow it to “see” the world.
- Searching with no obervation
- sensor-less prob
  - agent's percepts provide no infomation at all

Belief State Space for Sensorless Agent

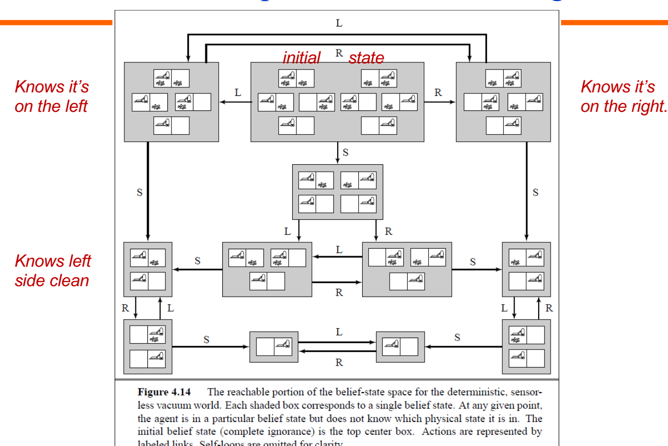


Figure 4.14 The reachable portion of the belief-state space for the deterministic, sensorless vacuum world. Each shaded box corresponds to a single belief state. At any given point, the agent is in a particular belief state but does not know which physical state it is in. The initial belief state (complete ignorance) is the top center box. Actions are represented by labeled links. Self-loops are omitted for clarity.

Online search agents and unkown environment

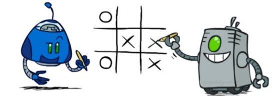
- so far **offline search** algo
  - compute a complete solution before setting foot in the real world and then execute the solution.
- Online search
  - refer to algorithms that must process input data as they received rather than waiting for the entire input dataset to become available.
  - agent interleave computation and action : first it take an action, then it observes the enviorment and computes the next action.
  - good idea in in dynamic or semidynamic domains

Take-away

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
  - Local search: widely used for very big problems
  - Returns good but not optimal solutions in general
  - Memory usage is one of the determining factors for choosing a search algorithm
    - For large state spaces, local search is an attractive practical option
- Learning the concept of problem solving using local improvements
- Learning the concept of getting stuck in local optima
- Learning the ways to get out of local optima
- Local search algorithms

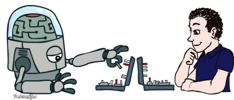
Next :

- Module 3: Search Strategies
  - PART 3.1: Search
  - PART 3.2: Uninformed Search
  - PART 3.3: Informed/Heuristic Search
    - Heuristics
    - Best First Search/ Greedy Search
    - A* Search
  - PART 3.4: Beyond Classical Search
    - Local Search
    - Problem reduction
  - PART 3.5: Adversarial Search
  - PART 3.6: Constraint Satisfaction Problems



References

- ***Artificial Intelligence*** *by Elaine Rich & Kevin Knight*, Third Ed, Tata McGraw Hill
- ***Artificial Intelligence and Expert System*** *by Patterson*
- http://www.cs.rmit.edu.au/AI-Search/Product/
- http://aima.cs.berkeley.edu/demos.html (for more demos)
- ***Artificial Intelligence and Expert System*** *by Patterson*
- Slides adapted from CS188 Instructor: Anca Dragan, University of California, Berkeley
- Slides adapted from CS60045 ARTIFICIAL INTELLIGENCE
- https://towardsai.net/p/computer-science/solving-the-5-queens-problem-using-genetic-algorithm



(some slides adapted from http://aima.cs.berkeley.edu/)