COMP3308/COMP3608, Lecture 3b

ARTIFICIAL INTELLIGENCE

Local Search Algorithms

Reference: Russell and Norvig, ch. 4

Outline

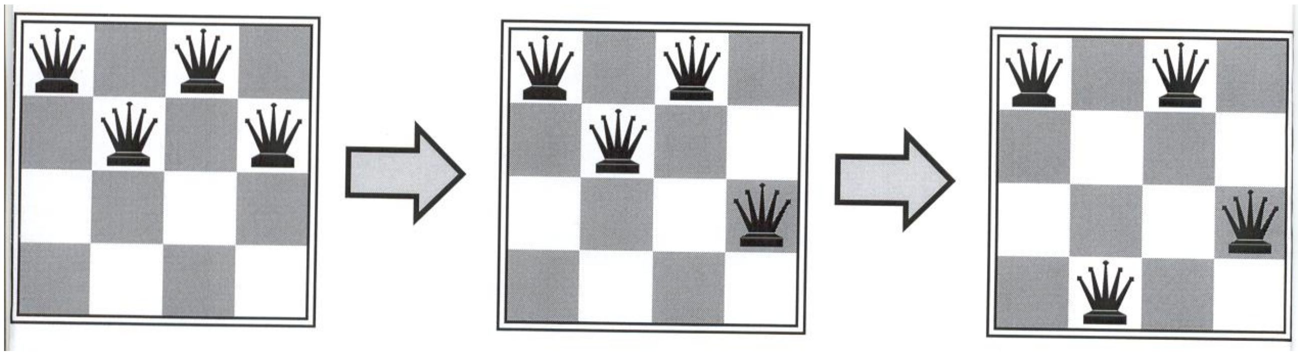
- **Optimisation problems**
- **Local search algorithms**
  - **Hill-climbing**
  - **Beam search**
  - **Simulated annealing**
  - **Genetic algorithms**

Optimisation Problems

- **Problem setting so far: path finding**
  - **Goal: find a path from S to G**
  - **Solution: the path**
  - **Optimal solution: least cost path**
  - **Search algorithms:**
    - **Uninformed: BFS, UCS, DFS, IDS**
    - **Informed: greedy, A***
- **Now a new problem setting: optimisation problem**
  - **Each state has a value *v***
  - **Goal: find the optimal state**
    - **= the state with the highest or lowest *v* score (depending on what is desirable, maximum or minimum)**
  - **Solution: the state; the path is not important**
- **A large number of states => can't be enumerated**
  - **=> We can't apply the previous algorithms – too expensive**

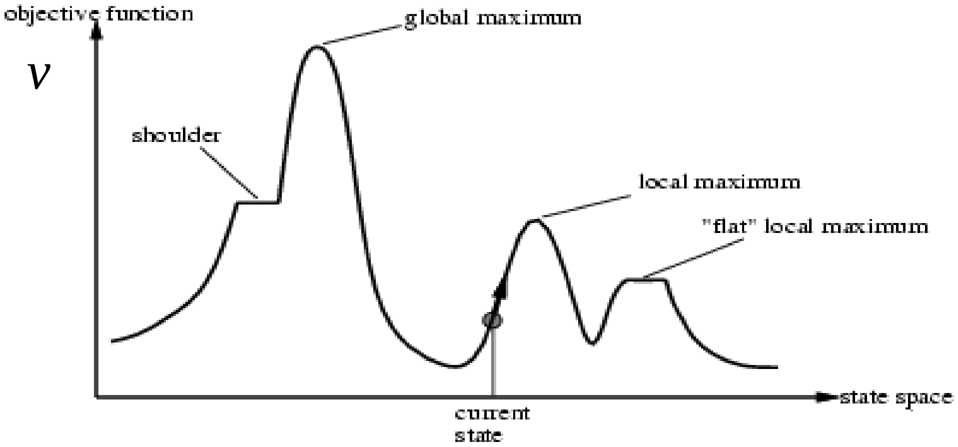
Optimisation Problems - Example

- **n-queens problem**
  - **The solution is the goal configuration, not the path to it**
- **Non-incremental formulation**
  - ***Initial state:*** **n-queens on the board (given or randomly chosen)**
  - ***States:*** **any configuration with n-queens on the board**
  - ***Goal:*** **no queen is attacking each other**
  - **Operators: "move a queen" or "move a queen to reduce the number of attacks"**



V-value Landscape

- Each state has a value *v* that we can compute
- This value is defined by a heuristic *evaluation function* (also called *objective function*)
- Goal - 2 variations depending on the task:
  - find the state with the highest value (global maximum) or
  - find the state with the lowest value (global minimum)
- Complete local search – finds a goal state if one exists
- Optimal local search – finds the best goal state – the state associated with the global maximum/minimum



Hill-Climbing Algorithm - Idea

- **Also called *iterative improvement* algorithm**
- **Idea: Keep only a single state in memory, try to improve it**

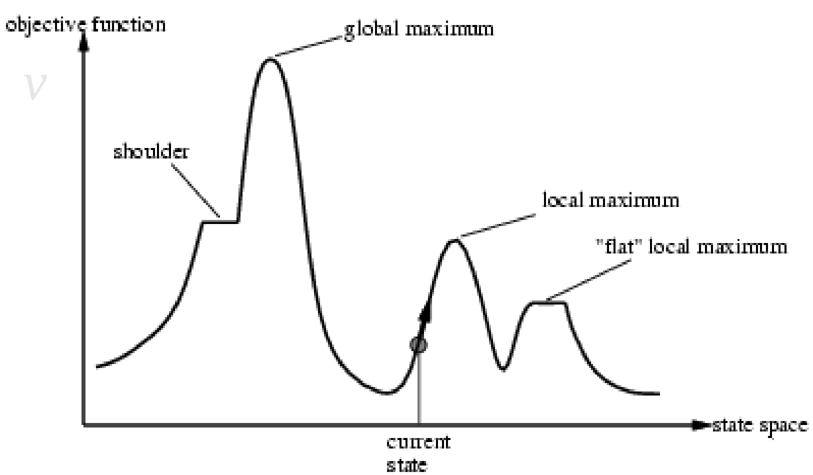
- **Two variations:**
  - **Steepest *ascent* – the goal is the *maximum* value**
  - **Steepest *descent* – the goal is the *minimum* value**

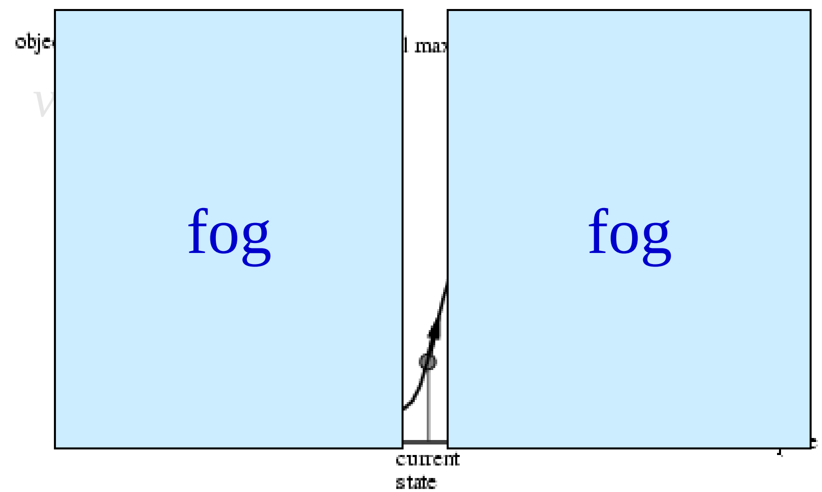
Hill-climbing Search

- **Assume that we are looking for a maximum value (i.e. hill-climbing <u>ascend</u>)**
- **Idea: move around trying to find the highest peak**
  - **Store only the current state**
  - **Do not look ahead beyond the immediate neighbors of the current state**
  - **If a neighboring state is better, move to it and continue, otherwise stop**
  - **"Like climbing Everest in <u>thick fog</u> <u>with amnesia</u>"**

we can't see the whole landscape, only the neighboring states

keeps only 1 state in memory, no backtracking to previous states





Hill-climbing Algorithm

Hill-climbing descent

1) Set current node *n* to the initial state *s*

(The initial state can be given or can be randomly selected)

2) Generate the successors of *n*. Select the best successor n_{best} ; *it is* the successor with the best *v* score, *v(best)* (i.e. the lowest score for descent)

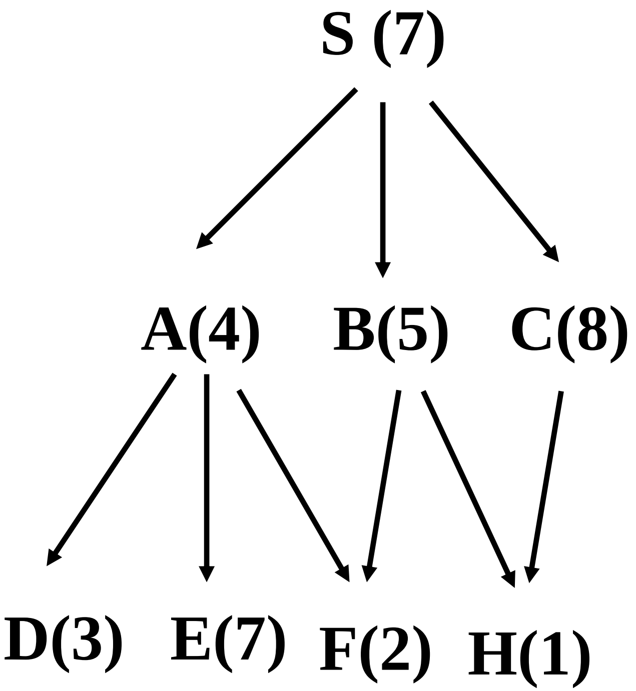
3) If *v(best)* > *v(n)*, return *n* //compare the child with the parent; if child not //better – stop; local or global minimum found

Else set *n* to n_{best} . Go to step 2 //if better - accept the child and keep // searching

- **Summary: Always expands the best successor, no backtracking**

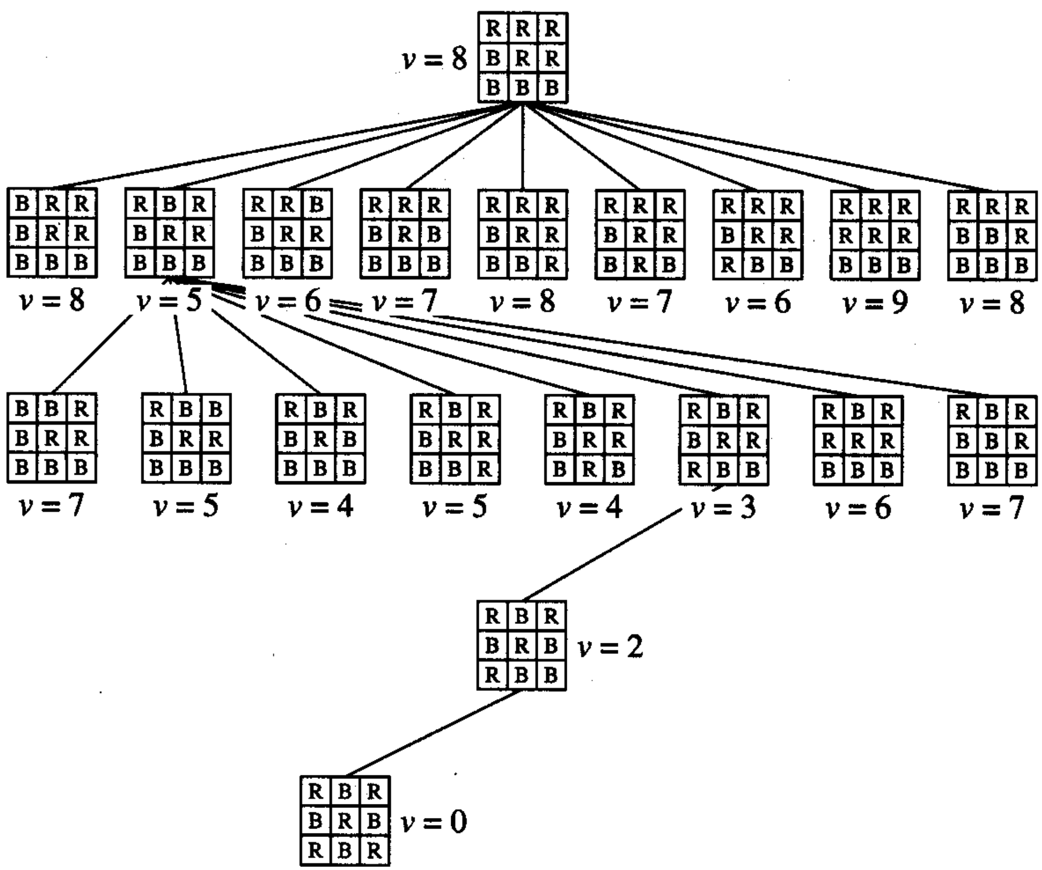
Hill-climbing – Example 1

- *v* - value is in brackets; the lower, the better (i.e. descent)
- Expanded nodes: SAF



Hill-climbing – Example 2

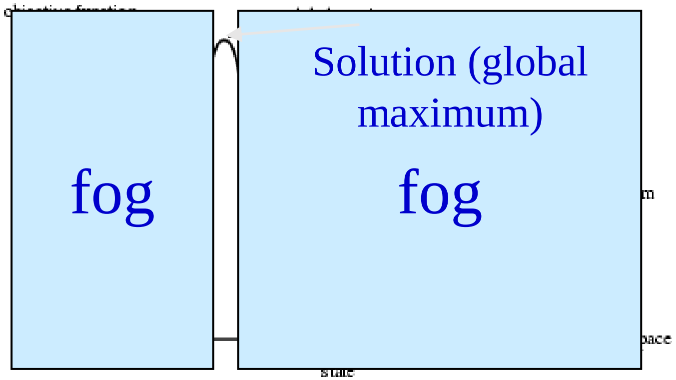
- Given: 3x3 grid, each cell is colored either red (R) or blue (B)
- Aim: find the coloring with minimum number of pairs of adjacent cells with the same color
- Ascending or descending?

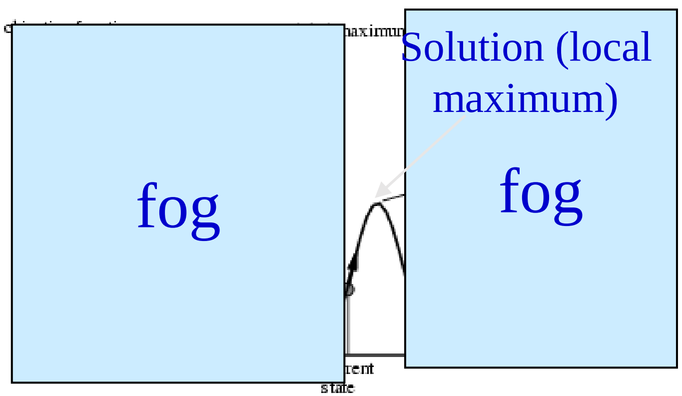


***v* – # pairs of adjacent cells with the same color**

Picture from N. Nielsen, AI, 1998

Hill-climbing Search





Weaknesses:

- **Not a very clever algorithm – can easily get stuck in a local optimum (maximum/minimum)**
- **However, not all local maxima/minima are bad – some may be reasonably good even though not optimal**

Advantages: good choice for hard, practical problems

- **Uses very little memory**
- **Finds reasonable solutions in large spaces where systematic algorithms are not useful**

Not complete, not optimal but keeps just one node in memory!

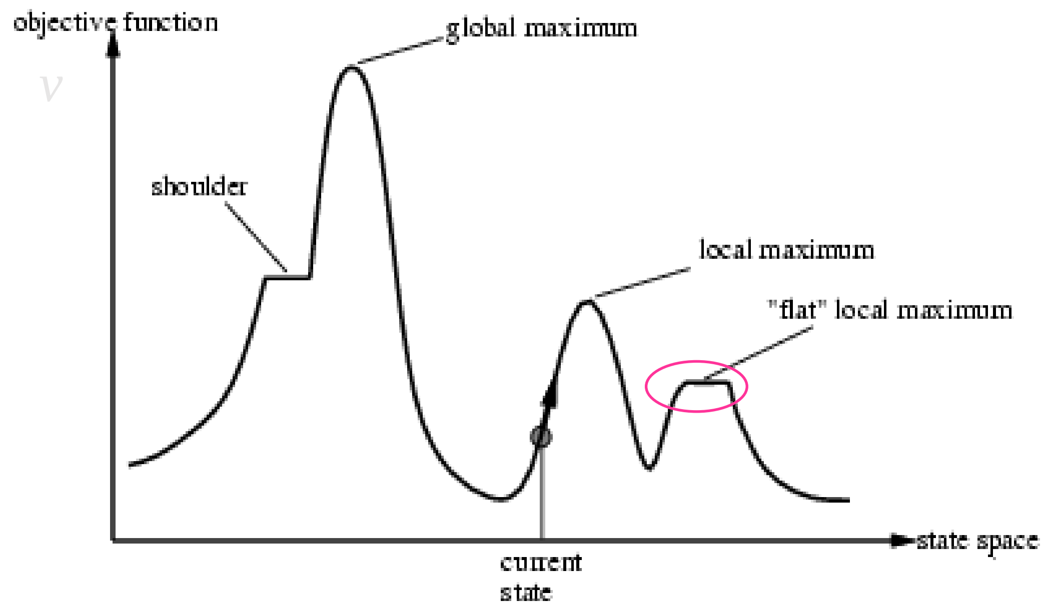
Hill-climbing – Escaping Bad Local Optima

- **Hill climbing finds the closest local optimum (minimum or maximum, depending on the version – descent or ascent)**
    - **Which may or may not be the global optimum**

- **The solution that is found depends on the initial state**
- **When the solution found is not good enough - random restart:**
    - **run the algorithm several times starting from different points; select the best solution found (i.e. the best local optimum)**
    - ***If at first you don't succeed, try, try again!***
    - **This is applicable for tasks without a fixed initial state**

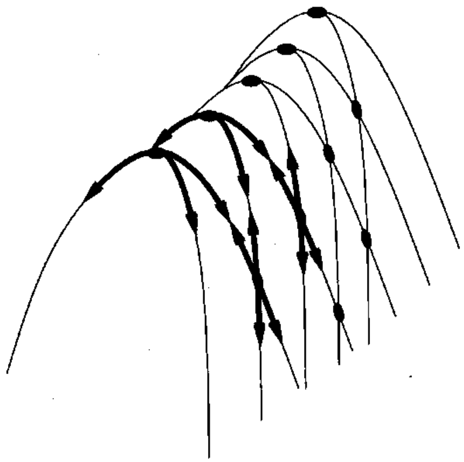
Hill-climbing – Escaping Bad Local Optima (2)

- **Plateaus (flat areas): no change or very small change in *v***
- **Our version of hill-climbing does not allow visiting states with the same *v* as it terminates if the best child's *v* is the same as the parent's**
- **But other versions keep searching if the values are the same and this may result in visiting the same state more than once and walking endlessly**
  - **Solution: keep track of the number of times *v* is the same and do not allow revisiting of nodes with the same *v***



Hill-climbing – Escaping Bad Local Optima (3)

- **Ridges – the current local maximum is not good enough; we would like to move up but all moves go down**
  - **Example:**



- Dark circles = states

- **A sequence of local maxima that are <u>not</u> connected with each other**
- **- From each of them all available actions point downhill.**

- **Possible solutions: combine 2 or more moves in a macro move that can increase the height or allow a limited number of look-ahead search**

Beam Search

- **It keeps track of *k* states rather than just 1**

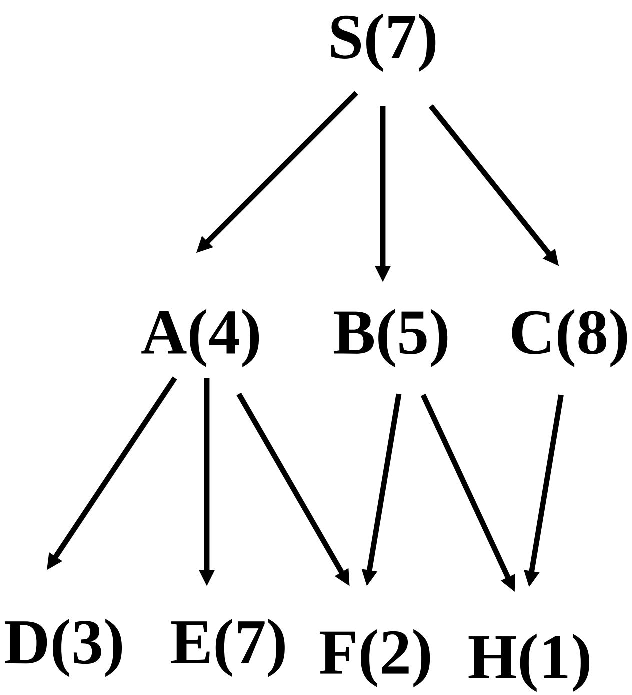
- **Version 1: Starts with <u>1 given state</u>**
- **At each level: generate all successors of the given state**
- **If any one is a goal state, stop; else select the *k* best successors from the list and go to the next level**

- **Version 2: Starts with <u>*k* randomly generated states</u>**
- **At each level: generate all successors of all *k* states**
- **If any one is a goal state, stop; else select the *k* best successors from the list and go to the next level**

- **In nutshell: keeps only *k* best states**

Beam Search - Example

- Consider the version that starts with 1 given state
- Starting from S, run beam search with k=2 using the values in brackets as evaluation function (the smaller, the better)
- Expanded nodes = ?



- S
- generate ABC
- select AB (the best 2 children)
- generate DEFH
- select FH (the best 2 children)
- expanded nodes: SABFH

Beam Search and Hill-Climbing Search

- **Compare beam search with 1 initial state and hill climbing with 1 initial state**
  - **Beam – 1 start node, at each step keeps *k* best nodes**
  - **Hill climbing – 1 start node, at each step keeps 1 best node**

- **Compare beam search with *k* random initial states and hill-climbing with *k* random initial states**
  - **Beam – *k* starting positions, *k* threads run in parallel, passing of useful information among them as at each step the best *k* children are selected**
  - **Hill climbing – *k* starting positions, *k* threads run individually, no passing of information among them**

Beam Search with A*

- Recall that memory was a big problem for A*
- Idea: keep only the best *k* nodes in the fringe, i.e. use a priority queue of size *k*

- Advantage: memory efficient
- Disadvantage: neither complete, nor optimal

Simulated Annealing

- **What is annealing in metallurgy?**
  - **a material's temperature is gradually decreased (very slowly) allowing its crystal structure to reach a minimum energy state**
- **Similar to hill-climbing but selects a random successor instead of the best successor (step 2 below)**

1) Set current node *n* to the initial state *s*.

Randomly select *m*, one of *n*'s successors

2) If *v(m)* is better than *v(n)*, *n=m* //accept the child *m*

Else *n=m* with a probability *p* //accept the child *m* with probability *p*

3) Go to step 2 until a predefined number of iterations is reached or the state reached (i.e. the solution found) is good enough

The Probability *p*

parent *n*

$\downarrow$

child *m*

- **Assume that we are looking for a <u>minimum</u>**
- **There are different ways to define *p*, e.g.**

$$p = e^{\frac{v(n)-v(m)}{T}}$$

- **Main ideas:**
  - **1) *p* decreases exponentially with the badness of the child (move) and**
  - **2) bad children (moves) are more likely to be allowed at the beginning than at the end**

- **nominator – shows how good the child *m* is**
  - **Bad move (the child is worse than the parent): v(n)<v(m), e.g.**
  - **case1: v(n)=10, v(m)=20, p1=e^-10/T**
  - **case2: v(n)=10, v(m)=11, p2=e^-1/T**
  - **m (the child) in case1 is worse than in case2**
  - **p1<p2 as T is positive => p exponentially decreases with the badness of the move**

The Probability *p* (2)

$$p = e^{\frac{v(n)-v(m)}{T}}$$

- **denominator: parameter *T* that decreases (anneals) over time based on a *schedule*, e.g. *T=T*0.8***
  - **high *T* – bad moves are more likely to be allowed**
  - **low *T* – more unlikely; becomes more like hill-climbing**
  - ***T* decreases with time and depends on the number of iterations completed, i.e. until "bored"**
- **Some versions have an additional step (Metropolis criterion for accepting the child):**
  - ***p* is compared with *p'*, a randomly generated number [0,1]**
  - **If *p* > *p'*, accept the child, otherwise reject it**
- **In summary, simulated annealing combines a hill-climbing step (accepting the best child) with a *random walk step* (accepting bad children with some probability). The random walk step can help escape bad local minima.**

Simulated Annealing - Theorem

- **What is the correspondence?**
  - ***v*** **– total energy of the atoms in the material**
  - ***T*** **– temperature**
  - ***schedule*** **– the rate at which *T* is lowered**

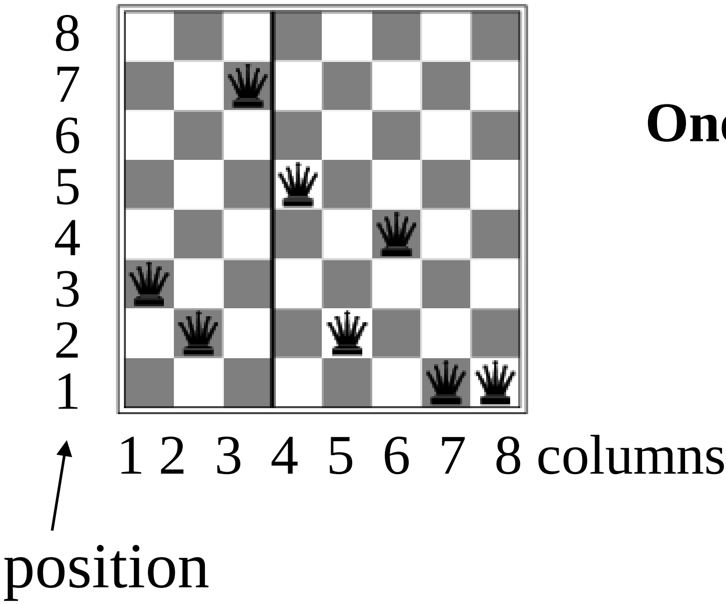
- **Theorem: If the schedule lowers *T* slowly enough, the algorithm will find global optimum**
  - **i.e. is complete and optimal given a long enough cooling schedule => annealing schedule is very important**
- **Simulated annealing has been widely used to solve VLSI layout problems, factory scheduling and other large-scale optimizations**
- **It is easy to implement but a "slow enough" schedule may be difficult to set**

Genetic Algorithms

- **Inspired by mechanisms used in evolutionary biology, e.g. selection, crossover, mutation**
- **Similar to beam search, in fact a variant of stochastic beam search**

- **Each state is called an *individual*. It is coded as a string.**
- **Each state *n* has a fitness score *f(n)* (evaluation function). The higher the value, the better the state.**
- **Goal: starting with *k* randomly generated individuals, find the optimal state**
- **Successors are produced by selection, crossover and mutation**
- **At any time keep a fixed number of states (the population)**

Example – 8-queens Problem



One possible encoding is (3 2 7 5 2 4 1 1)

column 1: a queen at position 3

column 2: a queen at position 2

…

Example – 8-queens Problem (2)

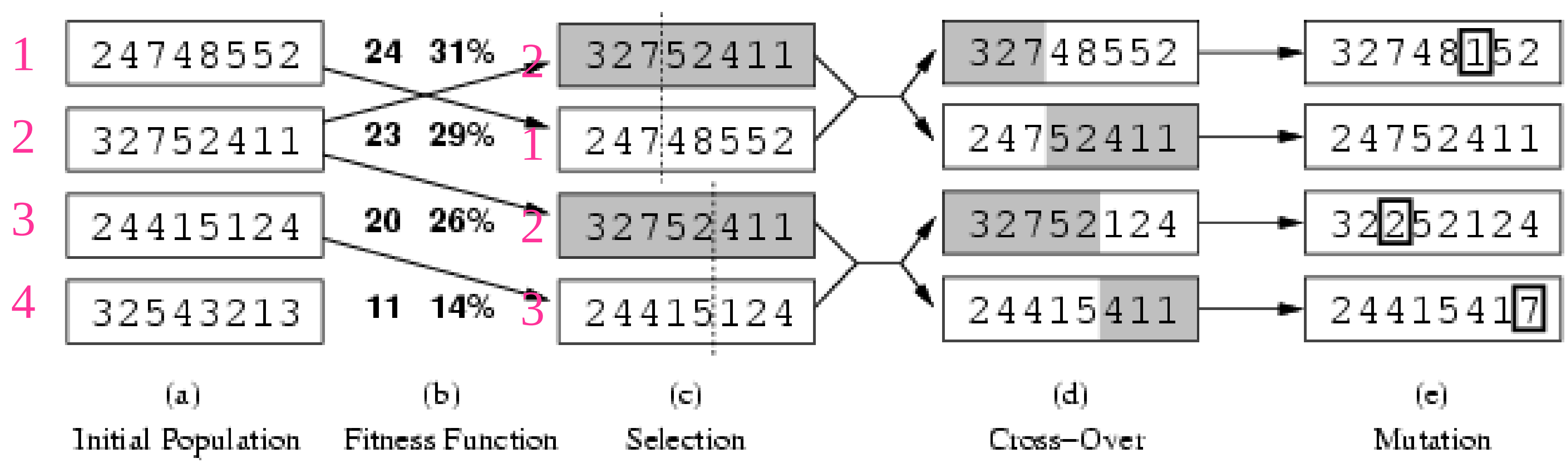
- Suppose that we are given 4 individuals (initial population) with their fitness values
  - Fitness values = number of non-attacking pairs of queens (given 8 queens, how many different pairs of queens are there? 28 => max value is 28)
  - Let the probability for reproduction is proportional to the fitness (expressed is %)

24748552	24	31%
32752411	23	29%
24415124	20	26%
32543213	11	14%

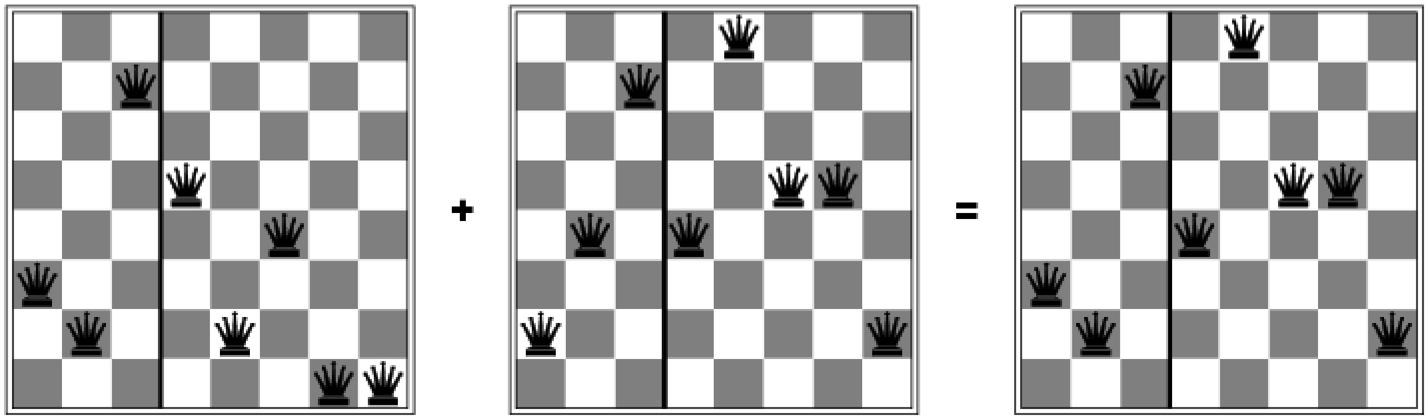
(a)
Initial Population

Example – 8-queens Problem (3)

- ***Select*** **4 individuals for reproduction based on the fitness function**
  - **The higher the fitness function, the higher the probability to be selected**
  - **Let individuals 2, 1, 2 and 3 be selected, i.e. individual 2 is selected twice while 4 is not selected**
- ***Crossover*** **– random selection of crossover point; crossing over the parents strings**
- ***Mutation*** **– random change of bits (in this case 1 bit was changed in each individual)**



A Closer Look at the Crossover



(3 2 7 5 2 4 1 1) + (2 4 7 4 8 5 5 2) = (3 2 7 4 8 5 5 2)

- **When the 2 states are different, crossover produces a state which is a long way from either parents**
- **Given that the population is diverse at the beginning of the search, crossover takes big steps in the state space early in the process and smaller later, when more individuals are similar**

Genetic Algorithm – Pseudo Code (1 variant)

from http://pages.cs.wisc.edu/~jerryzhu/cs540.html

1. Let $s_1, \ldots, s_N$ be the current population
2. Let $p_i = f(s_i) / \Sigma_j f(s_j)$ be the reproduction probs
3. FOR $k = 1$; $k<N$; k+=2
   - parent1 = randomly pick s with probs p
   - parent2 = randomly pick another s with probs p
   - randomly select a crossover point, swap strings of parents 1, 2 to generate children $t[k]$, $t[k+1]$
4. FOR $k = 1$; $k<=N$; k++
   - Randomly mutate each position in $t[k]$ with a small probability
5. The new generation replaces the old: $\{ s \} \leftarrow \{ t \}$. Repeat **until some individual is fit enough or a predefined maximum number of iterations has been reached**

Genetic Algorithms - Discussion

- **Combine:**
    - **uphill tendency (based on the fitness function)**
    - **random exploration (based on crossover and mutation)**
- **Exchange information among parallel threads - the population consists of several individuals**
- **The main advantage comes from crossover**
- **Success depends on the representation (encoding)**
- **Easy to implement**
- **Not complete, not optimal**

Links

Simulated annealing as a training algorithm for backpropagation neural networks:

- **R.S. Sexton, R.E. Dorsey, J.D. Johnson, *Beyond backpropagation: using simulated annealing for training neural networks,* people.missouristate.edu/randallsexton/sabp.pdf**