Local Search

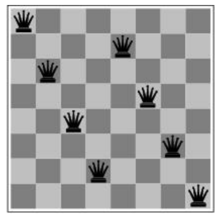
Outline

- Hill climbing
- Simulated annealing
- Local beam search
- Genetic algorithms (briefly)

2

Local search and optimization

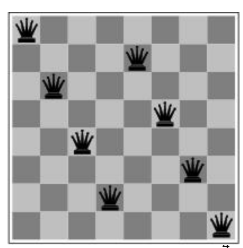
- Previously: systematic exploration of search space.
  - solution to problem is path to goal
- Local search suitable for problems in which *path is irrelevant;* the goal state itself is the solution
- E.g 8-queens



3

Local search and optimization

- Local search suitable for **Optimization problems.**
  - State space = set of "complete" configurations
  - Find best state according to some *objective function h(s).*
  - E.g., h(s)= # of conflicts
- E.g 8-queens using complete-state formulation
  - States: 8 queens on the board, one per column
  - Actions: Move a queen to a square in the same column



4

Local search and optimization

- Local search= keep a single current state and move to neighboring states to improve it

- Advantages:
  - Use very little memory
  - Often find reasonable solutions in large or infinite state spaces unsuitable for systematic algorithms
    - solve million-queens quickly

5

Hill-climbing search

- Keep a single current node and move to the best neighboring states to improve it

- “a loop that continuously moves in the direction of increasing value”
  - chooses randomly to break ties
  - It terminates when a peak is reached where no neighbor has a higher value

- Hill-climbing a.k.a. *greedy local search, steepest ascent/descent*

6

Hill-climbing search

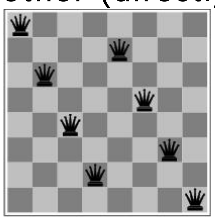
function HILL-CLIMBING(*problem*) **return** a state that is a local maximum

current ← MAKE-NODE(problem.INITIAL-STATE)
loop do
neighbor ← a highest valued successor of *current*
if *neighbor*.VALUE ≤ *current.*VALUE
then return *current*.STATE
current ← *neighbor*

Hill-climbing example

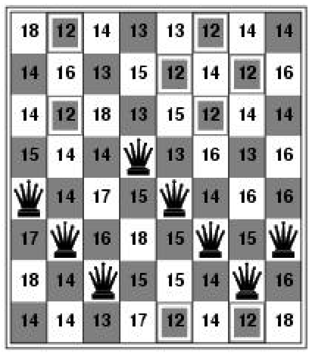
- 8-queens problem (complete-state formulation).
  - States: 8 queens on the board, one per column
  - Action: move a single queen to another square in the same column.

- Heuristic function *h(n)*: the number of pairs of queens that are attacking each other (directly or indirectly).

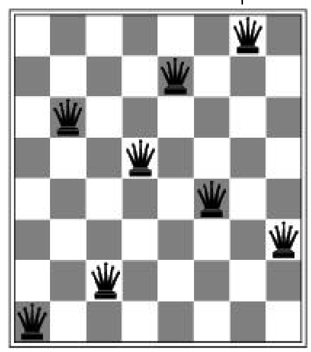


Hill-climbing example

a)



b)

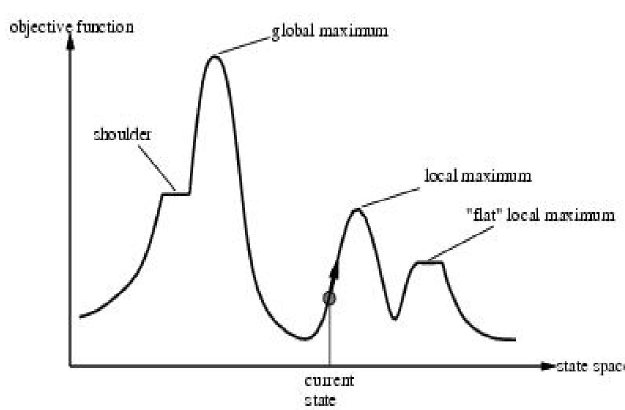


a) shows a state of h=17 and the h-value for each possible successor.

b) A local minimum in the 8-queens state space (h=1).

Drawbacks

state space landscape



- Depending on initial state, can get stuck in local maxima, plateaux

Hill-climbing variations

- Stochastic hill-climbing
  - Random selection among the uphill moves.
  - The selection probability can vary with the steepness of the uphill move.
- First-choice hill-climbing
  - generating successors randomly until a better (than the current) one is found.
  - useful when a state has many successors
- Random-restart hill-climbing
  - Restart search from random initial state

11

Simulated annealing

- Escape local maxima by allowing "bad" moves.
  - Idea: but gradually decrease their frequency.
- *T*, a "temperature" controlling the probability of downward steps

12

Simulated annealing

function SIMULATED-ANNEALING(*problem, schedule*) **return** a solution state
input: *problem*, a problem
schedule, a mapping from time to temperature
local variables: *T*, a "temperature" controlling the probability of downward steps

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

13

Simulated annealing

- Escape local maxima by allowing "bad" moves.
- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1
- Commonly used: T ← cT with c constant close to, but smaller than, 1
- Applied for VLSI layout, airline scheduling, etc.

14

Local beam search

- Keep track of *k* states instead of one
  - Initially: *k* random states
  - Next: determine all successors of *k* states
  - If any of successors is goal → finished
  - Else select *k* best from successors and repeat.
- Major difference with random-restart search
  - Information is shared among *k* search threads.
- Can suffer from lack of diversity.
  - Stochastic variant: choose k successors randomly with probability proportional to state value difference.

15

Genetic algorithms

- Inspired by the process of biological evolution
- Start with *k* randomly generated states/individuals (population)
- States are scored by evaluation function (fitness function).
- At each step, the most fit states are selected (*survival of the fittest*) probabilistically: used as seeds for producing the next generation population -- the children or *offspring*, by means of operations such as *crossover* and *mutation*
- The process is repeated until sufficiently fit states are discovered -- the best state has a score exceeding a criterion
- They have been applied successfully to a variety of learning tasks and optimization problems

16

Genetic algorithm

function GENETIC_ALGORITHM(*population,* FITNESS-FN) **return** an individual

input: *population*, a set of individuals

FITNESS-FN, a function which determines the fitness of an individual

repeat

new_population ← empty set

loop for i **from** 1 **to** SIZE(*population*) **do**

x ← RANDOM_SELECTION(*population*, FITNESS_FN)

y ← RANDOM_SELECTION(*population*, FITNESS_FN)

child ← REPRODUCE(*x*,*y*)

if (small random probability) **then** *child* ← MUTATE(*child*)

add *child* to *new_population*

population ← *new_population*

until some individual is fit enough or enough time has elapsed

return the best individual in *population*

17

Genetic algorithms

- In basic GAs, the fundamental representation of each state is a string over a finite alphabet (often a string of 0s and 1s), called a *chromosome*
- The mapping depends on the problem domain, and the designers.

18

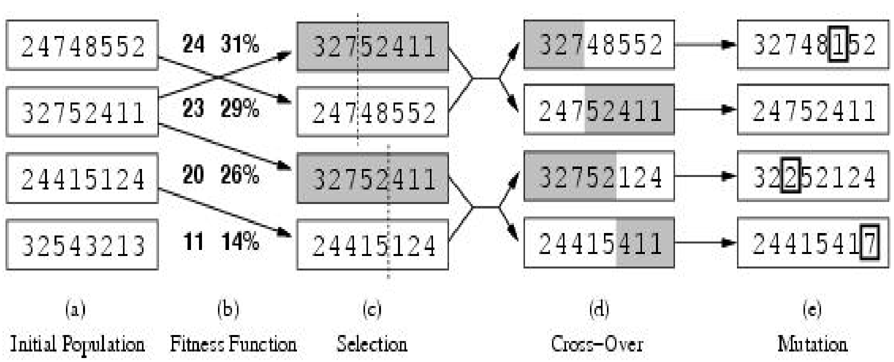
Genetic algorithms

Genetic Operators

- **Replication**: a chromosome is merely reproduced
- **Crossover**: involves the mating of two parent chromosome to yield two new offspring by copying selected bits from each parent
- **Mutation**: creates a single descendant from a single parent by changing the value of a randomly chosen bit (from a 1 to 0 or vice versa)
- Other operators: specialized to the particular representation

19

Genetic algorithms



20

Beyond Simple Environment

- Local Search in Continuous Spaces (Chapter 4.2)
- Searching with Nondeterministic Actions (Chapter 4.3)
- Searching with Partial Observations (Chapter 4.4)
- Online Search (Chapter 4.5)

21