Beyond Classical Search[1]

Lecture 5

[1] The slides have been prepared using the textbook material available on the web, and the slides of the previous editions of the course by Prof. Luigia Carlucci Aiello

Summary

- Russell &Norvig Chapter 4 Sec. 1
- Hill-climbing
- Simulated annealing
- Local beam search
- Genetic Algorithms

Iterative improvement algorithms

In many optimization problems, *path* is irrelevant;
the goal state itself is the solution

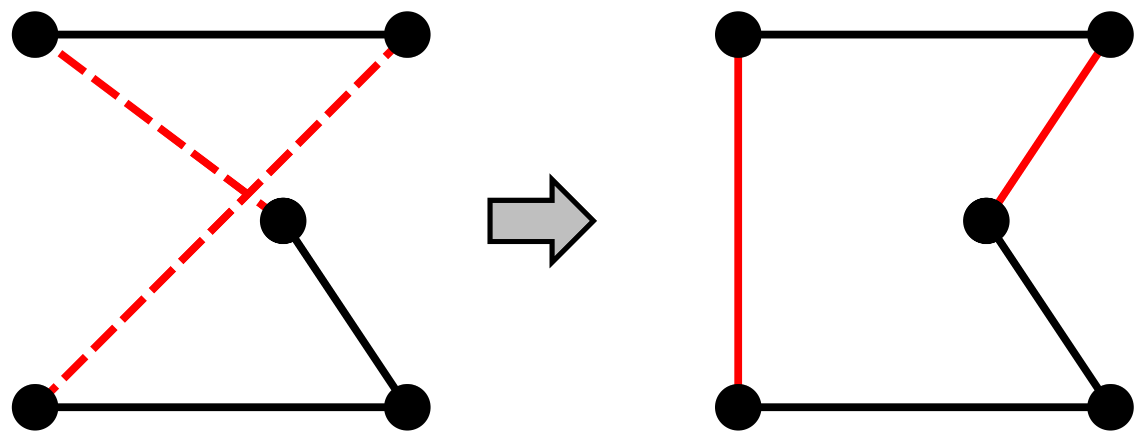
Then state space = set of "complete" configurations;
find *optimal* configuration, e.g., TSP
or, find configuration satisfying constraints, e.g., timetable

In such cases, can use *iterative improvement* algorithms;
keep a single "current" state, try to improve it

Constant space, suitable for online as well as offline search

Example: Travelling Salesperson Problem

Start with any complete tour, perform pairwise exchanges

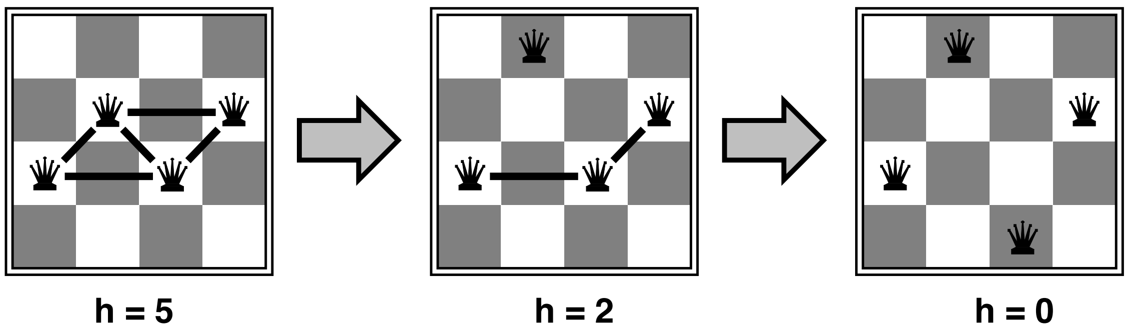


Variants of this approach get within 1% of optimal very quickly with thousands of cities

Example: n-queens

Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal

Move a queen to reduce number of conflicts



Almost always solves n-queens problems almost instantaneously for very large n, e.g., $n = 1 million$

Example: Fitness function

fitness: # attacking queens.
There maybe multiple states that have the same fitness.

Hill-climbing (or gradient ascent/descent)

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

inputs: *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*

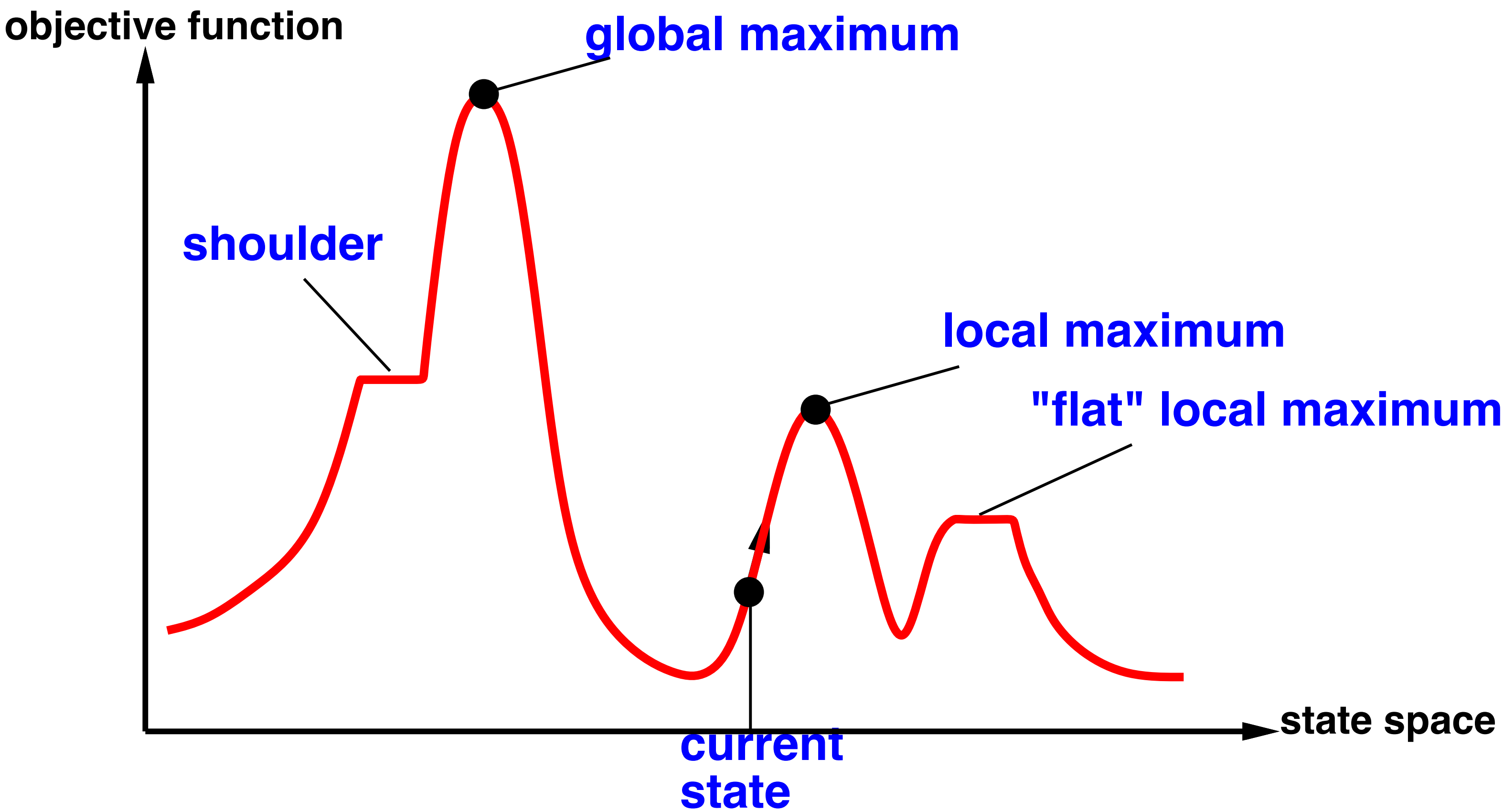
if VALUE[neighbor] < VALUE[current]

then return STATE[*current*]

current ← *neighbor*

end

Problems



objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

Possible Solutions

◊ **side moves** with a limit on the maximum number #

◊ **Stochastic Hill Climbing**: random moves (choosing among the uphill successors)

◊ **First choice**: random generation of successors taking the first one uphill

◊ **random restart**

Success is strongly related to the “shape” of the state space

Some remarks about performance

8-queens problem (8^8states)

- hill-climbing 14% with 4 (3) steps
- 100 side moves 94% with 22(64) steps
- random restart 7 iterations $\frac{1}{p}$

- random restart HC (1-p)/p * 3 $+$ 4 $=$ 22 moves
- HC with side moves (1-p)/p * 64 $+$ 21 $=$ 25

Simulated annealing

Idea: avoid local maxima allowing some "bad" moves
gradually decreasing their effect and frequency

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to temp
  local variables: *current*, a node
    next, a node
    T, a temp (prob downward steps)

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

Properties of simulated annealing

At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

T decreased slowly enough $\Longrightarrow$ always reach best state

Is this necessarily an interesting guarantee?

Devised by Metropolis et al., 1953, for physical process modelling

Widely used in VLSI layout, airline scheduling, etc.

“Local beam” search

◇ Keeps k states.

◇ At each step the successors of the k states are generated and the best k are selected among them, unless goal is reached.

◇ Non just a parallel execution: at each step the best nodes are chosen among all the successors (**come here the grass is greener**)

◇ Problem: too quick convergence in the same region of the search space; the **stochastic beam search** randomly chooses k successors weighting more the most promising ones.

Genetic Algorithms 1

Idea: organisms evolve; those adaptable to the environment survive and reproduce, others die (Darwin)

initial population: individuals or *cromosomes*
selection: by *fitness function*
reproduction: *crossover*
reproduction: *mutation*

Search in the space of individuals
Steepest ascent hill-climbing, since little genetic alterations are performed on selected individuals.

Genetic Algorithms 2

To deploy Genetic Algorithms we must define:

1. individual representation?
2. fitness function?
3. selection?
4. reproduction?

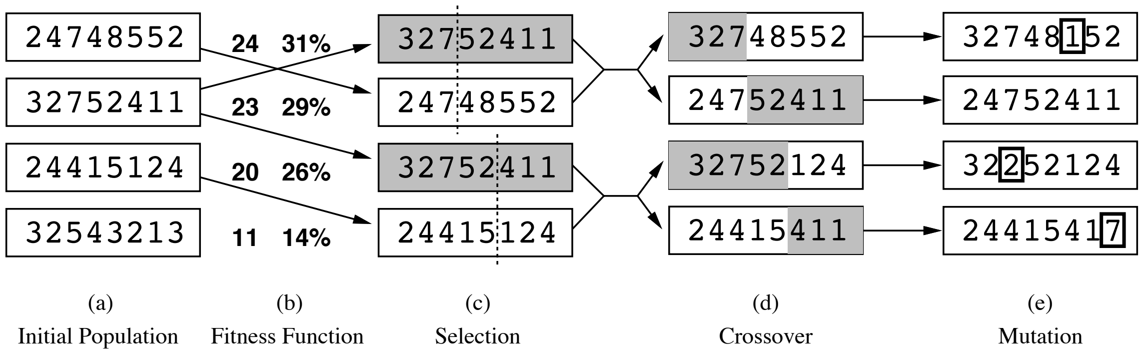
1. string of charatcters (*genes*) (often 0/1)
2. function mapping individuals into real numbers
3. generally selection is stochastic
4. Crossover + mutation

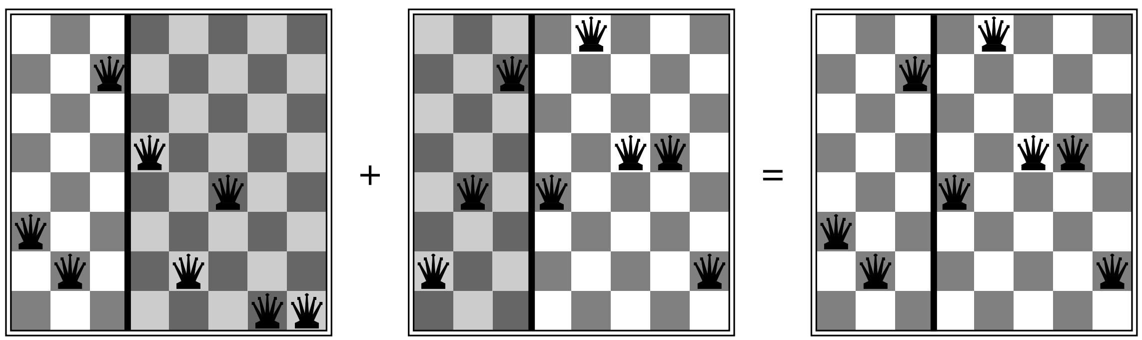
Genetic Algorithms: implementation

function GENETIC-ALGORITHM(*population*, FITN)
returns individual
inputs: *population*, set of individuals
FITN, measuring fitness of individuals

repeat
parents ← SELECTION(*population*,FITN)
population ← REPRODUCE(*parents*)
until some individual is fit enough
return best individual in *population*, for FITN

Genetic Algorithms 3





Generalization to Evolutionary Algorithms

- **evolution strategies** individuals are sequences of reals
- **genetic programming** individual is a computer program
- **mixing number** $\rho > 2$
- **elitism** (include top scoring parents)
- **culling**

Local search in continuous spaces

◇ Branching factor is infinite!

- **discretization**
- **empirical gradient** (steepest ascent hill-climbing)
- use the **gradient** i.e. solve $\bigtriangledown f = 0$ to find the maximum (typically through approximation Newton-Raphson)
- **constraint optimization**: Linear programming

Summary

♢ Local Search:

- solves large problem
- statistically optimal
- Hill Climbing, Local Beam Search, Simulated Annealing, Genetic Algorithms