Intelligent Systems/ Sistemas Inteligentes

Búsqueda local en grafos/ Local search

Antonio Bahamonde
Departamento de informática

Universidad de Oviedo en Gijón

Contents

Local Search

In many problems, the path to the goal is irrelevant.

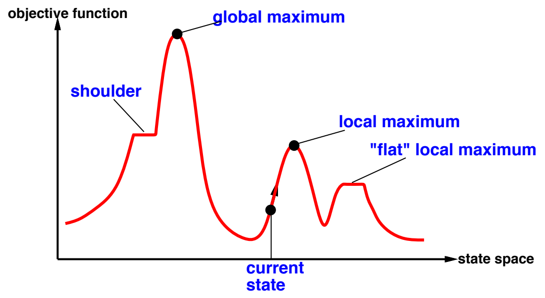
- Configuration problems: queens, integrated-circuit design, factory-floor layout, job-shop scheduling, …
- Pure **optimization problems**, in which the aim is to find the best state according to an **objective function**

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\ f(\boldsymbol{w})$$

Local search algorithms operate using a single **current node** and try to improve it (iterative)

Local Search

State-space landscape.
Example. A one-dimensional state-space landscape in which elevation corresponds to the objective function. The aim is to find the global maximum



Local Search Idea

Local search means:

- Use single current state and move to neighboring states.

Idea: start with an initial guess at a solution and incrementally improve it until it is one

Advantages:

- Use very little memory
- Find often reasonable solutions in large or infinite state spaces.

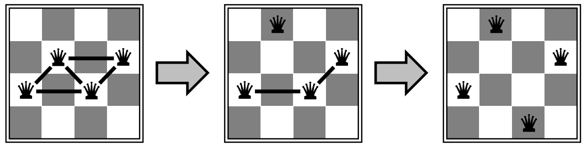
Local Search. Example, N-queens

Algorithm scheme

Put n queens on an n × n board, with no two queens on the same column

Move a queen to reduce the number of conflicts; repeat until we cannot move any queen anymore

– then we are at a local maximum, hopefully it is global too



This almost always solves n-queens problems almost instantaneously for very large n (e.g., n = 1 million)

Local Search: Hill-climbing

The most basic local search. Steepest ascent.

```
function Hill-Climbing(problem) returns 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
```

Local Search: Hill-climbing

It terminates when it reaches a “peak” where no neighbor has a higher value.

The algorithm does not maintain a search tree, so the data structure for the current node need only record the state and the value of the objective function.

Hill climbing does not look ahead beyond the immediate neighbors of the current state. This resembles trying to find the top of Mount Everest in a thick fog while suffering from amnesia

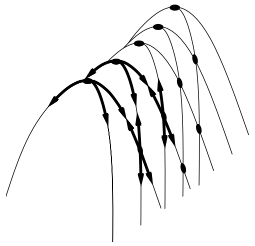
Hill climbing is sometimes called **greedy local search** because it grabs a good neighbor state without thinking ahead about where to go next

Step by step

Local Search: Hill-climbing

Unfortunately, hill climbing often gets stuck for the following reasons:

- Local maxima
- Ridges
- Plateaux



the algorithm reaches a point at which no progress is being made

9

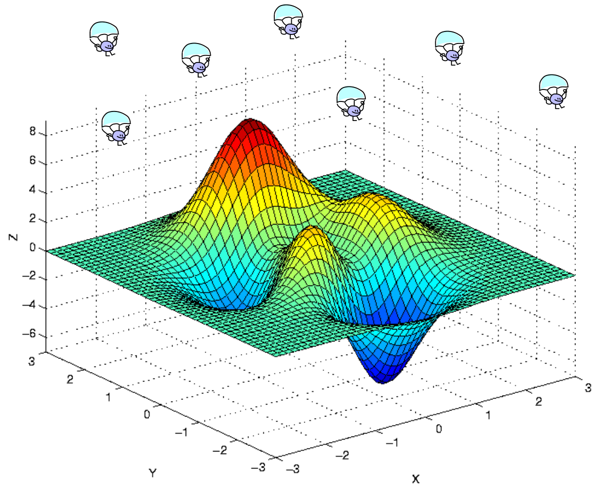
Local Search: Hill-climbing

The hill-climbing algorithm described so far is incomplete

- It often fail to find a goal when one exists because they can get stuck on local maxima.
- **Random-restart hill climbing** adopts the well-known adage, “If at first you don’t succeed, try, try again.” This strategy is trivially complete, given enough time

10

Hill-climbing Random-restart



11

Genetic Algorithms

Genetic Algorithms

GAs begin with a set of k randomly generated states, called the **population**.

Each state, or **individual**, is represented as a string over a finite alphabet (**chromosomes**)

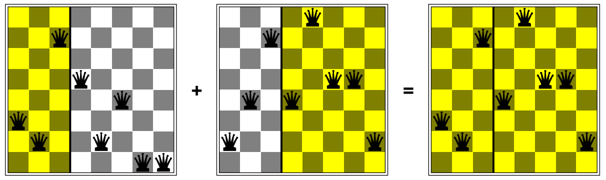
—most commonly, a string of 0s and 1s

To produce the next generation of states, each state is rated by the objective function or (in GA terminology) the **fitness function**.

$$\boldsymbol{w}^* = \underset{\boldsymbol{w}}{\operatorname{argmax}}\ f(\boldsymbol{w})$$

Genetic Algorithms

For each pair to be mated, a **crossover** point is chosen randomly from the positions in the string.

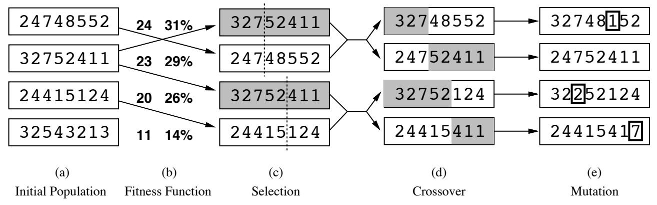


3 2 7 5 2 4 1 1 + 2 4 7 4 8 5 5 2 = 3 2 7 4 8 5 5 2

The crossover helps iff substrings are meaningful components

Finally, each location is subject to random **mutation** with a small independent probability. One digit was mutated from time to time.

Genetic Algorithms



Fitness (for the 8-queens problem, the number of *nonattacking* pairs of queens: 28 for a solution)

In this particular variant of the genetic algorithm, the probability of being chosen for reproducing is directly proportional to the fitness score

Genetic Algorithms

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
inputs:   population, a set of individuals,
              FITNESS-FN, a function that measures the fitness of an individual

    repeat
        new_population ← empty set
        for i = 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, according to FITNESS-FN

function REPRODUCE(x,y) returns an individual
    inputs: x , y , parent individuals

    n ← LENGTH(x ); c ← random number from 1 to n
    return APPEND(SUBSTRING(x,1,c),SUBSTRING(y, c +1,n))
```

Genetic Algorithms syllabus

- **Coding** schema, for example strings of binary digits (0 and 1)
- **Genotype**, **phenotype**
- **Fitness** function to measure the quality of the chromosomes
- Some method to generate the **initial** population:
  random, heuristic, ...
- A set of genetic **operators**: selection, crossover, mutation, replacement, …
- Fitness proportional selection (**roulette wheel**)
- **Elitism**
- **One point** crossover
- GAs require a number of **parameters**: crossover probability, mutation probability, population size, number of generations, ...

Genetic Algorithms syllabus

- Fitness proportional selection (**roulette wheel**)

Example

Ind	i1	i2	i3	i4	i5	i6	i7	i8	i9	i10
fit	5	2	10	2	3	5	6	3	2	10
Pr	10 %	4 %	21 %	4 %	6 %	10 %	12 %	6 %	4 %	21 %
Accu. fit	5	7	17	19	22	27	33	36	38	48

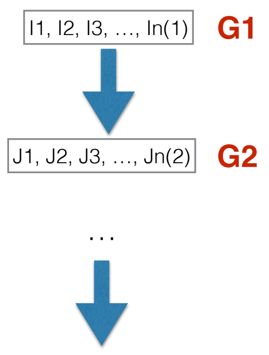
random from 0 to 48

selection	s1	s2	s3	s4
random	1	3	31	16
ind	i1	i1	i7	i3

EDAs: *Estimation of Distribution Algorithms*

Estimation of Distribution Algorithms: EDAs

- Alternative to GA



To build a new generation (G2) from a previous one (G1)

- First estimate the distribution of the best individuals of G1
- Second sample that distribution to generate the next generation G2

Estimation of Distribution Algorithm: EDA

- Heuristic search where the probability distribution is the clue (the heuristic)
- EDA = *Estimation of Distribution Algorithm*
- Motivation: Removing the genetics from the standard genetic algorithms [Baluja y Caruana, 1995].
- [Müehlenbein, Paab, 1996]

EDAs Algorithm

```
D (0) ← generate randomly M individuals (initial population)
Repeat for i = 1 , 2, ...
    D_Se (i-1) ← Select N (<= M) individuals from D (i – 1)
    $p_i$ ( x ) = Pr( x : x ∈ D_Se (I-1)) ←  Estimate the probability
                                               distribution of selection
    D (I) ← Sample M individuals according to $p_i$ (x)
until stop condition

return the best individual ever considered
```

EDAs: Estimation of the distribution of the best individuals

UMDA (Univariate Marginal Distribution Algorithm**,** Mühlenbein'98**):**

$$p_i(x_j) = \frac{\sum(x_j : x \in D\_SE(i-1))}{N}$$

	x1	x2	x3	x4	x5	x6	x7	x8	x9	x10
ind1	0	1	0	0	0	1	1	0	0	0
ind2	1	0	1	1	1	0	1	0	1	1
ind3	1	0	0	0	0	1	1	0	1	0
pi	2/3	1/3	1/3	1/3	1/3	2/3	1	0	2/3	1/3

EDAs: Some applications

- Jose A. Lozano, Pedro Larrañaga, Iñaki Inza and Endika Bengoetxea (Editors, 2006) Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms. Springer
- Pedro Larrañaga and Jose A. Lozano (Editors, 2002) Estimation of Distribution Algorithms. A New Tool for Evolutionary Computation. Kluwer Academic Publishers

Feature selection (Machine Learning)
    (genes that condition diseases, food production, …)

Discrete Optimization