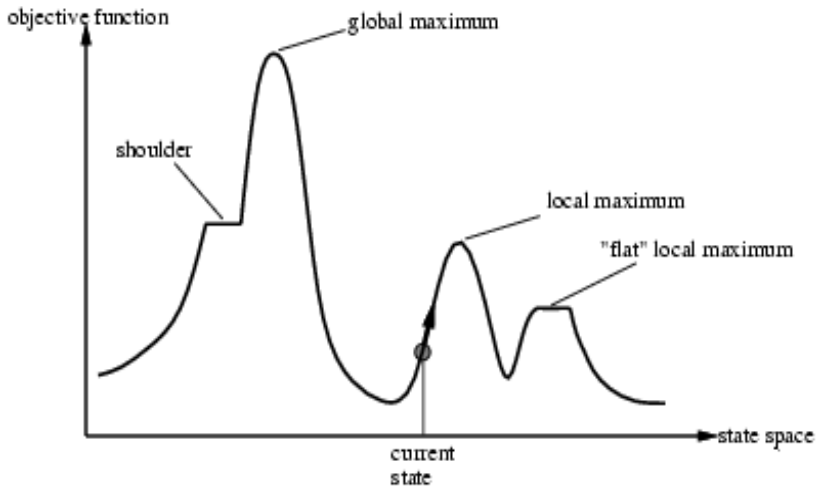
Local search algorithms(iterative improvement)

The general idea is to start with a complete configuration and to make modifications to improve its quality.



Local search (Optimization) algorithms:

- Hill-climbing search
- Simulated annealing
- Local beam search
- Genetic algorithms

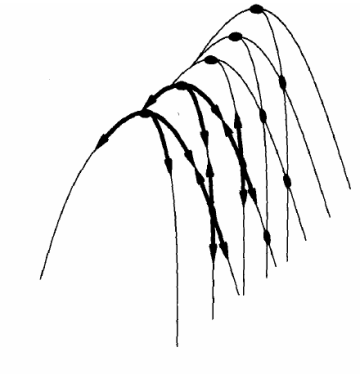
1. Hill-climbing search

Here is the algorithm :

```
function HILL-CLIMBING( problem) return a local maximum state
    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
```

This simple policy has three well-known drawbacks:

- **Local maxima:** a local maximum, as opposed to a global maximum, *is a peak that is lower than the highest peak* in the state space..
- **Plateaux:** a plateau is an area of the state space where the *evaluation function is essentially flat*. The search will conduct a random walk.
- **Ridges:** a ridge may have steeply sloping sides, so that the search reaches the top of the ridge with ease, but the top may slope only very gently toward a peak. Unless there happen to be operators that move directly along the top of the ridge, the search may oscillate from side to side, making little progress.



(Ridges)

Hill-climbing variations

1-**Stochastic hill-climbing**: Random selection among the uphill moves.
2- First-choice hill-climbing : stochastic hill climbing by generating successors randomly until a better one is found.
3- Random-restart hill-climbing: ➔ Tries to avoid getting stuck in local maxima.

2. Simulated annealing

function SIMULATED-ANNEALING(*problem, schedule*)
returns a solution state
input: *problem,* a problem
schedule, a mapping from time to temperature
local variables: *current***,** a node.
*next***,** a node.
*T***,** a " temperature" controlling probability of steps
current ← MAKE-NODE(INITIAL-STATE[*problem*])
for time 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 $\Delta E > 0$ **then** *current* ← *next*
else *current* ← *next* only with probability $e^{\Delta E/T}$

$e^{\Delta E/T}$ is **Boltzman's constant**.

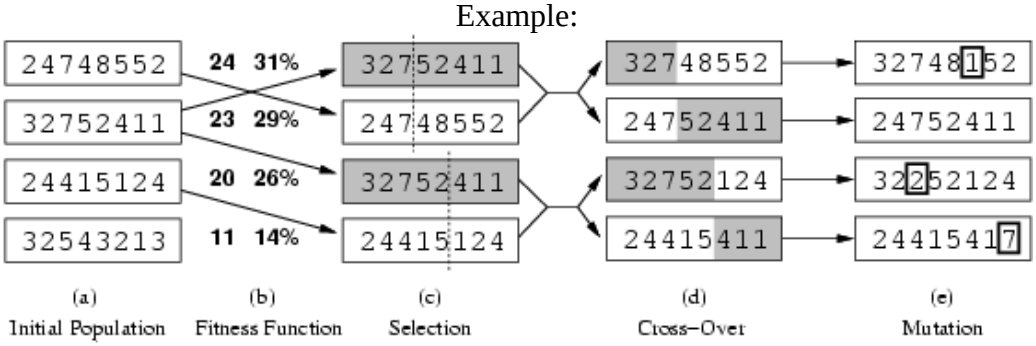
3.Genetic algorithms

A successor state is generated by combining two parent states.
Start with *k* randomly generated states (population)
A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
Evaluation function is called fitness function, with higher values for better states.
Produce the next generation of states by **selection**, **crossover**, and **mutation**

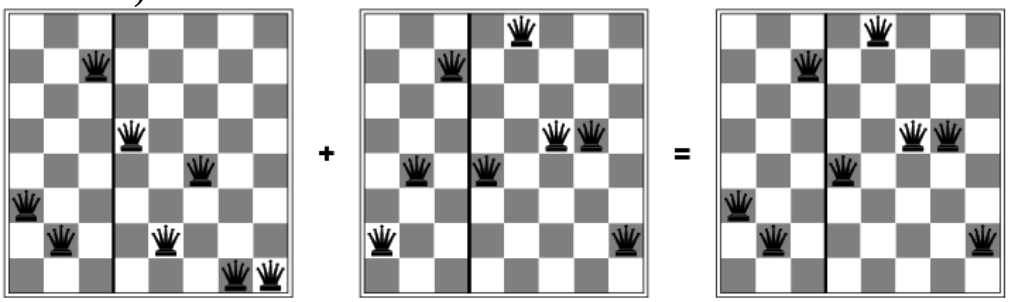
```
function GENETIC_ALGORITHM(population, FITNESS-FN)
                                                    return an individual
    input: population, a set of individuals
           FITNESS-FN, a function telling  the quality of an individual

    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)
               MUTATE(child) if needed
               add child to new_population
           population ← new_population
    until some individual is fit enough or enough time has elapsed
    return the best individual
```

Example:



Fitness function:
number of non-attacking pairs of queens (min=0, max = 8×7/2= 28)
24/(24+23+20+11) = 31%
23/(24+23+20+11) = 29% etc

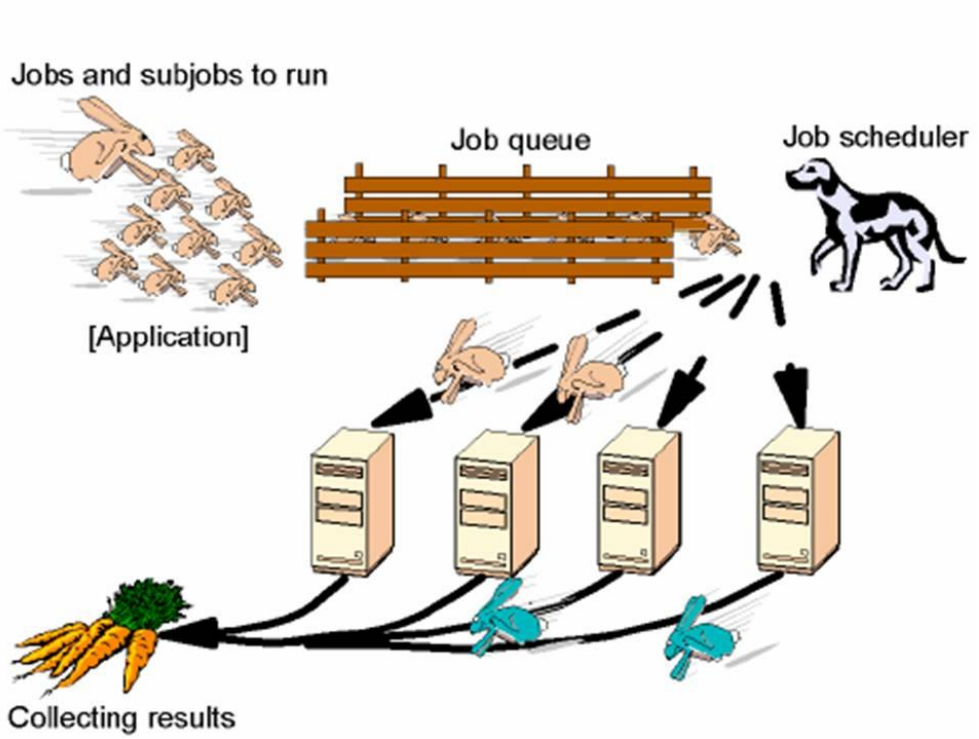


4. Local beam search (LBS)

Keep track of *k* states rather than just one. Start with *k* randomly generated states. At each iteration, all the successors of all *k* states are generated . If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

Local beam search algorithm resembles **Grid computing**.

During 2007 the term **cloud computing** came into popularity.


Jobs and subjobs to run
Job queue
Job scheduler
[Application]
Collecting results