

---

## Informed Search Algorithms

---

1

---

## Outline

---

- Best-first search
  - A\* search
  - Heuristics
- 

2

---

## Review

---

```
function TREE-SEARCH(problem, fringe) returns a solution, or failure
  fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
  loop do
    if fringe is empty then return failure
    node ← REMOVE-FRONT(fringe)
    if GOAL-TEST[problem] applied to STATE(node) succeeds return node
    fringe ← INSERTALL(EXPAND(node, problem), fringe)
```

A strategy is defined by picking the order of node expansion

---

3

---

## Best-first search

---

- **Idea:** use an evaluation function for each node
    - estimate of “desirability”
    - Expand most desirable unexpanded node
  - **Implementation:**
    - *fringe* is a queue sorted in decreasing order of desirability
  - **Special cases:**
    - greedy search
    - A\* search
- 

4

## Algorithm Best-first

1. Start with OPEN containing just the initial state.
2. Until a goal is found or there are no nodes left on OPEN do:
  - (a) Pick the best node on OPEN.
  - (b) Generate its successors.
  - (c) For each successor do:
    - i. If it has not been generated before, evaluate it, add it to OPEN, and record its parent.
    - ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case, update the cost of getting to this node and to any successors that this node may already have.

5

## Greedy search

- Evaluation function  $h(n)$  (heuristic) = estimate of cost from  $n$  to goal
  - e.g.,  $h_{SLD}(n)$  = straight-line distance from  $n$  to Bucharest
- Greedy search expands the node that appears to be closest to goal

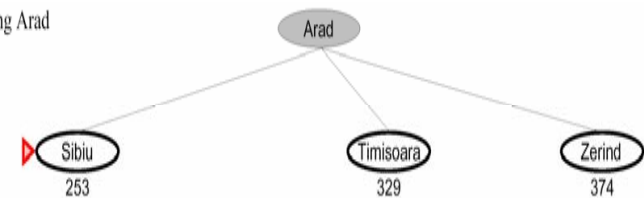
6

(a) The initial state



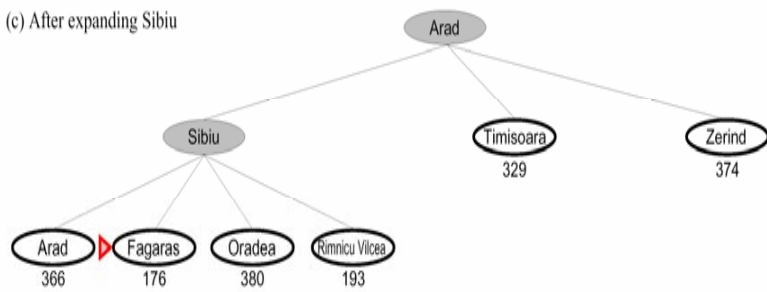
7

(b) After expanding Arad



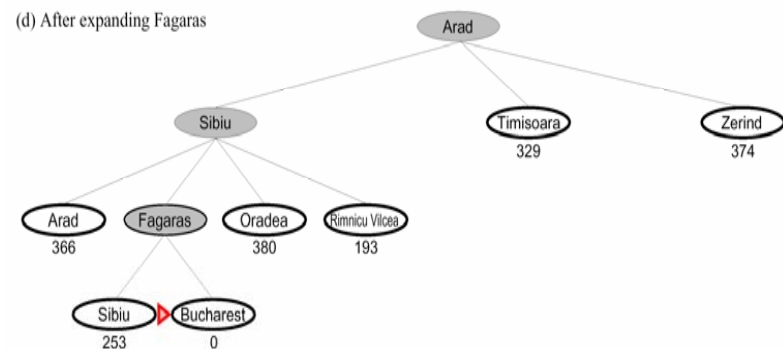
8

(c) After expanding Sibiu



9

(d) After expanding Fagaras



10

## Properties of greedy search

- **Completeness:** No. Can get stuck in loops, e.g.,  
Iasi → Neamt → Iasi → Neamt !  
Complete in finite space with repeated-state checking
- **Time Complexity:**  $O(b^m)$ , but a good heuristic can give dramatic improvement
- **Space Complexity:**  $O(b^m)$  keeps all nodes in memory
- **Optimality:** No

11

## A\* search

- **Idea:** avoid expanding paths that are already expensive
- **Evaluation function**  $f'(n) = g(n) + h'(n)$ 
  - $g(n)$  = cost so far to reach  $n$
  - $h'(n)$  = estimated cost to goal from  $n$
  - $f'(n)$  = estimated total cost of path through  $n$  to goal

12

## Admissible Heuristics

A\* search uses an admissible heuristic i.e.,  
 $h(n) \leq h^*(n)$  where  $h^*(n)$  is the true cost from  $n$ .  
e.g.,  $h_{SLD}(n)$  never overestimates the actual road distance.

13

## A\* Search Example

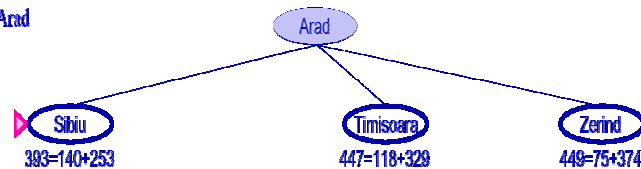
(a) The initial state



14

## A\* Search Example

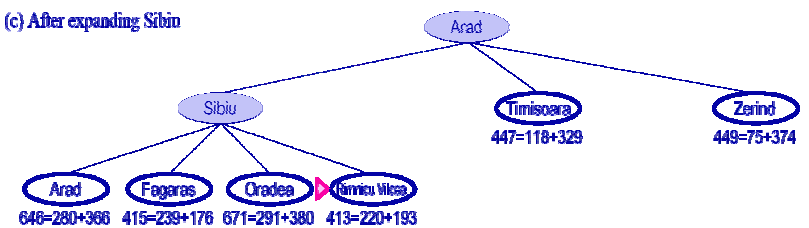
(b) After expanding Arad



15

## A\* Search Example

(c) After expanding Sibiu

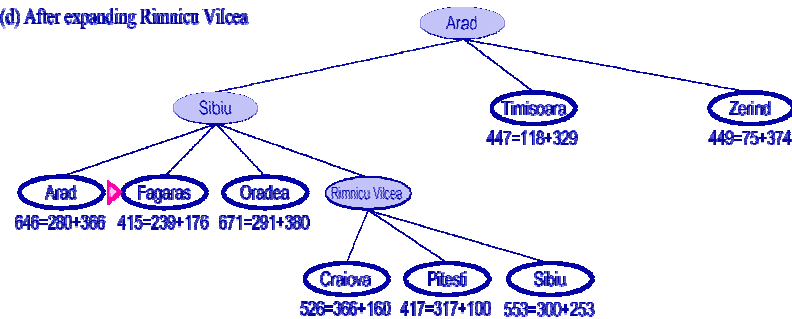


16



## A\* Search Example

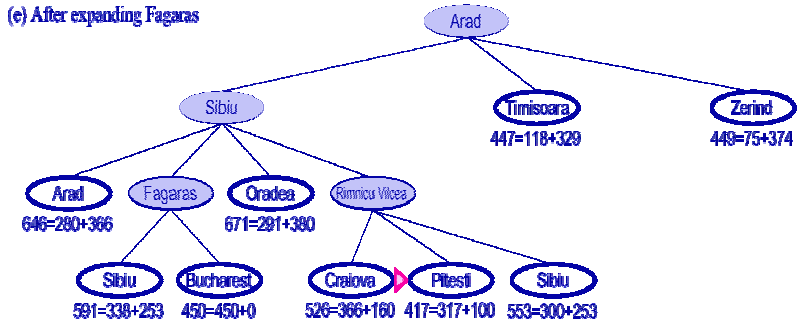
(d) After expanding Rimnicu Vilcea



17

## A\* Search Example

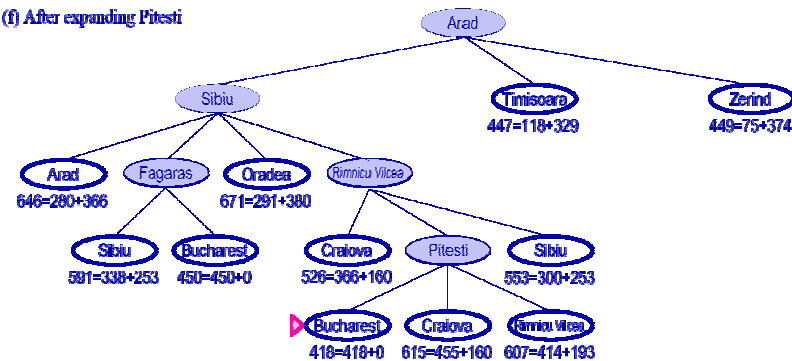
(e) After expanding Fagaras



18

## A\* Search Example

(f) After expanding Pitesti

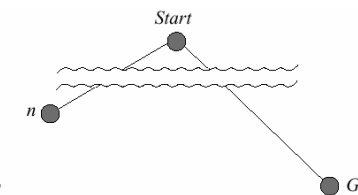


19

## Optimality of A\* (standard proof)

Suppose some suboptimal goal  $G_2$  has been generated and is in the queue.

Let  $n$  be an unexpanded node on a shortest path to an optimal goal  $G_1$ .



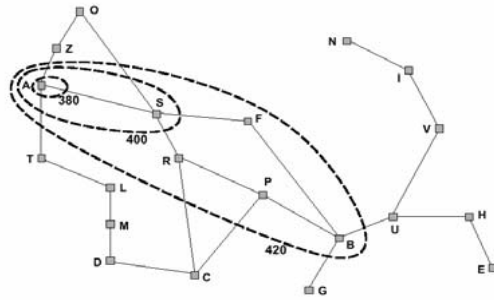
$$\begin{aligned}
 f(G_2) &= g(G_2) && \text{since } h(G_2) = 0 \\
 &> g(G_1) && \text{since } G_2 \text{ is suboptimal} \\
 &\geq f(n) && \text{since } h \text{ is admissible}
 \end{aligned}$$

Since  $f(G_2) > f(n)$ , A\* will never select  $G_2$  for expansion

20

## Optimality of A\* (more useful)

**Lemma:** A\* expands nodes in order of increasing  $f$  value  
 Gradually adds "f-contours" of nodes (cf. breadth-first adds layers)  
 Contour  $i$  has all nodes with  $f = f_i$ , where  $f_i < f_{i+1}$



1

## Properties of A\*

**Complete??** Yes, unless there are infinitely many nodes with  $f \leq f(G)$

**Time ??** Exponential in [relative error in  $h \times$  length of solution.]

**Space ??** Keeps all nodes in memory

**Optimal ??** Yes. cannot expand  $f_{i+1}$  until  $f_i$  is finished.

2

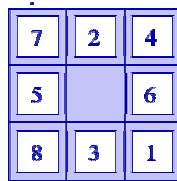
## Admissible heuristics

e.g., for the 8-puzzle:

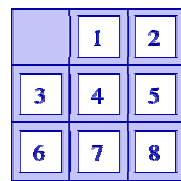
$h_1(n)$  = number of misplaced tiles

$h_2(n)$  = total Manhattan distance

(i.e., no. of squares from desired location of each tile)



Start State



Goal State

$h_1(S) = ?? 8$

$h_2(S) = ?? 2+3+3+2+4+2+0+2 = 18$

3

## Dominance

If  $h_2(n) \geq h_1(n)$  for all  $n$  (both admissible) then  $h_2$  dominates  $h_1$  and is better for search

Typical search costs:

$d = 14$  IDS = 3,473,941 nodes

$A^*(h_1) = 539$  nodes

$A^*(h_2) = 113$  nodes

$d = 24$  IDS = 54,000,000,000 nodes

$A^*(h_1) = 39,135$  nodes

$A^*(h_2) = 1,641$  nodes

Given any admissible heuristics  $h_a, h_b$ ,

$h(n) = \max(h_a(n), h_b(n))$

is also admissible and dominates  $h_a, h_b$ ,

4

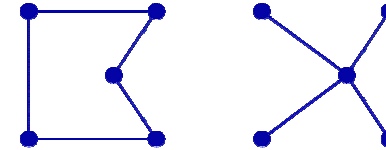
## Relaxed problems

- Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem
- If the rules of the 8-puzzle are relaxed so that a tile can move **anywhere**, then
  - $h_1(n)$  gives the shortest solution
- If the rules are relaxed so that a tile can move to **any adjacent** square, then
  - $h_2(n)$  gives the shortest solution
- **Key point:** the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

5

## Relaxed problems (Cont.)

- Well-known example: travelling salesperson problem (TSP)
- Find the shortest tour visiting all cities exactly once



- Minimum spanning tree can be computed in  $O(n^2)$  and is a lower bound on the shortest (open) tour.

6

## Iterative Deepening A\*

A\*- fixed- cost( max)

1. Set  $N$  to be the set of initial nodes and set  $lub$  to infinity.
  2. If  $N$  is empty, then signal failure and exit.
  3. Set  $n$  to be the first node in  $N$  and remove  $n$  from  $N$ .
  4. If  $n$  is a goal node, then signal success and exit.
  5. If the value of  $n$  is equal to  $max$ , then go to step 2.
  6. For each child:
    - Check whether the value is larger than  $max$ .
    - If yes then if the value is smaller than  $lub$ , set  $lub$  to this value.
    - If no then add child to  $N$ .
  7. Sort  $N$  and go to step 2.
- Call A\*- fixed- cost with increasing  $max$  (using  $lub$ ).  
 $lub$  = lower upper bound

7

## IDA\* (Iterative Deepening A\*)

- Memory- bounded Search
  - iterative deepening
  - bounded f- cost
  - most cases: required space = bd
  - if too long time -> epsilon- admissible variant

8

## SMA\* (Simplified Memory- Bounded A\*)

- Memory- bounded Search
  - uses whatever memory is possible
  - avoids repeated states as far as memory allows
  - complete if memory is sufficient to store shallowest solution path
  - optimal if memory is sufficient to store shallowest solution path (otherwise best solution with available memory is returned)
  - makes use of 'forgotten nodes'

9

## Iterative improvement algorithms

- In many optimization problems, path is irrelevant; the goal state itself is the solution
- Then state space = set of "complete" configurations (complete-state formulation vs. incremental formulation)
- 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
- Often want to find optimal configuration, e.g., TSP,
- but also works for constraint satisfaction problems, e.g. n-queens, timetabling

10

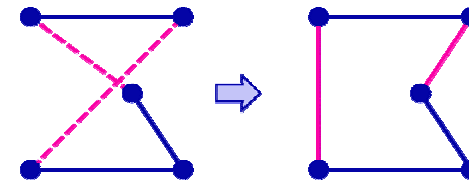
## Local Search

- Local search algorithms work by keeping in memory just one current state (or perhaps a few), moving around the state space based on purely local information.
- The paths followed by the search are not retained.
- Local search algorithms are not systematic,
- They have two advantages:
  - (1) they use very little memory—usually a constant amount; and
  - (2) they can often find reasonable solutions in large or infinite (continuous) state spaces for which systematic algorithms are unsuitable.

11

## Example: Travelling Salesperson Problem

- Start with any complete tour, perform pairwise exchanges

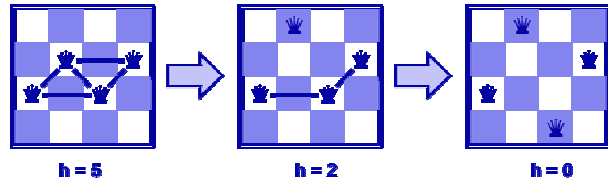


- Variants of this approach get within 1% of optimal very quickly with thousands of cities

12

## Example: $n$ -queens

- Put  $n$  queens on an  $n \times n$  board with no two queens on the same row, column, or diagonal
- Local search: start with all  $n$ , move a queen to reduce conflicts



- Almost always solves  $n$ -queens problems almost instantaneously for very large  $n$ , e.g.,  $n=1$  million
- (Why? Perhaps because choices for queen  $k$  are made wrt all others)

13

## Generate and Test

1. Generate a possible solution.
2. Test to see if this is actually a solution by comparing the chosen point or the endpoint of the path to the set of acceptable goal states
3. If a solution has been found, quit. Otherwise, return to step 1.

British Museum Algorithm

14

## Simple Hill Climbing

1. Evaluate the initial state. If it is also a goal state then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until there are no operators left to be applied in the current state.
  - a) Select an operator that has not yet been applied to the current state and apply it to produce a new state.
  - b) Evaluate the new state
    - i. If it is a goal state, then return it and quit.
    - ii. If it is not a goal state but it is better than the current state, then make it the current state.
    - iii. If it is not better than the current state, then continue in the loop.

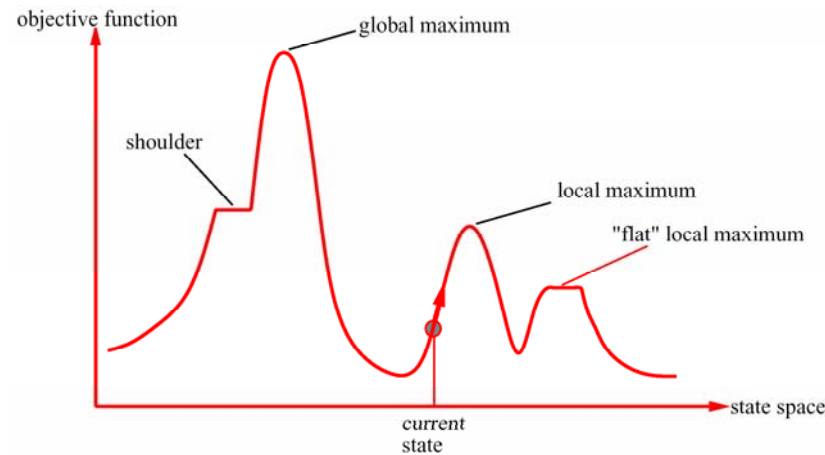
15

## Steepest- Ascent Hill Climbing

1. Evaluate the initial state. If it is also a goal state then return it and quit. Otherwise, continue with the initial state as the current state.
2. Loop until a solution is found or until a complete iteration produces no change to the current state:
  - a) Let SUCC be a state such that any possible successor of the current state will be better than SUCC.
  - b) For each operator that applies to the current state do:
    - i. Apply the operator and generate a new state.
    - ii. Evaluate the new state. If it is a goal state, then return it and quit. If not, compare it to SUCC. If it is better then set SUCC to this state. If it is not better, leave SUCC alone.
    - iii. If SUCC is better than current state, then set current state to SUCC.

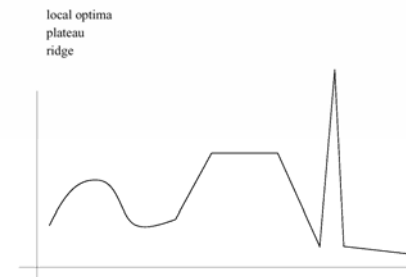
16

## Local Search Space



17

## Hill Climbing Problems



- **Local maxima:** a local maximum is a peak that is higher than each of its neighboring states, but lower than the global maximum.
- **Ridges:** Ridges result in a sequence of local maxima that is very difficult for greedy algorithms to navigate.
- **Plateaux:** a plateau is an area of the state space landscape where the evaluation function is flat. It can be a flat local maximum, from which no uphill exit exists, or a **shoulder**, from which it is possible to make progress

18

## Local Optima (Foothill Problem)

- **Problem Definition**
  - Point reached by hill-climbing may be maximal but not maximum
  - Maximal
    - Definition: not dominated by any neighboring point (with respect to criterion measure)
  - Maximum
    - Definition: dominates all neighboring points (wrt criterion measure)
- **Ramifications**
  - Steepest ascent hill-climbing will become trapped (why?)
  - Need some way to break out of trap state
    - Accept transition (i.e., search move) to dominated neighbor
    - Start over: random restarts

19

## Lack of Gradient (Plateau Problem)

- **Problem Definition**
  - Function space may contain points whose neighbors are indistinguishable (with respect to criterion measure)
  - Effect: "flat" search landscape
- **Ramifications**
  - Steepest ascent hill-climbing will become trapped
  - Need some way to break out of zero gradient
    - Accept transition (i.e., search move) to random neighbor
    - Random restarts
    - Take bigger steps

20

## Single-Step Traps (Ridge Problem)

### ■ Problem Definition

- Function space may contain points such that single move in any "direction" leads to suboptimal neighbor
- Effect
  - There exists steepest gradient to goal
  - None of allowed steps moves along that gradient
  - Thin "knife edge" in search landscape, hard to navigate

### ■ Ramifications

- Steepest ascent hill-climbing will become trapped (why?)
- Need some way to break out of ridge-walking
  - Formulate composite transition (multi-dimension step) – how?
  - Accept multi-step transition (at least one to worse state) – how?
  - Random restarts

1

## Ridge Problem Solution: Multi-Step Trajectories

### ■ Intuitive Idea: Take More than One Step in Moving along Ridge

#### ■ Analogy: Tacking in Sailing

- Need to move against wind direction
- Have to compose move from multiple small steps
  - *Combined move*: in (or more toward) direction of steepest gradient
  - Another view: *decompose* problem into self-contained *subproblems*

### ■ Multi-Step Trajectories: Macro Operators

- Macros: (inductively) generalize from 2 to > 2 steps
- Example: Rubik's Cube
  - Can solve 3 x 3 x 3 cube by solving, interchanging 2 x 2 x 2 cubies
  - *Knowledge* used to formulate subcube (cubie) as macro operator
- *Treat operator as single step* (multiple primitive steps)

2

## Plateau, Local Optimum, Ridge Solution: Global Optimization

### ■ Intuitive Idea

- Allow search algorithm to take some "bad" steps to escape from trap states
- *Decrease probability of taking such steps gradually to prevent return to traps*

### ■ Analogy: Marble(s) on Rubber Sheet

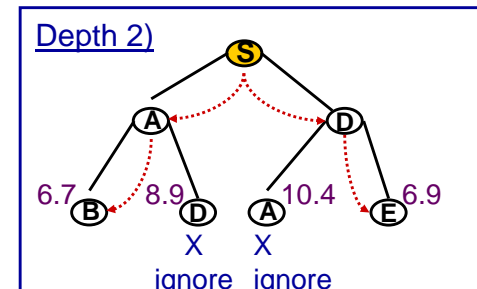
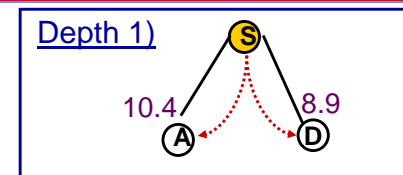
- Goal: move marble(s) into global minimum from any starting position
- Shake system: hard at first, gradually decreasing vibration
- Marbles tend to break out of local minima but have less chance of re-entering

### ■ Analogy: Annealing

- Ideas from metallurgy, statistical thermodynamics
- Cooling molten substance: slow as opposed to rapid (quenching)
- Goal: maximize material strength of solidified substance

3

## Beam search



- Assume a pre-fixed WIDTH (example : 2 )

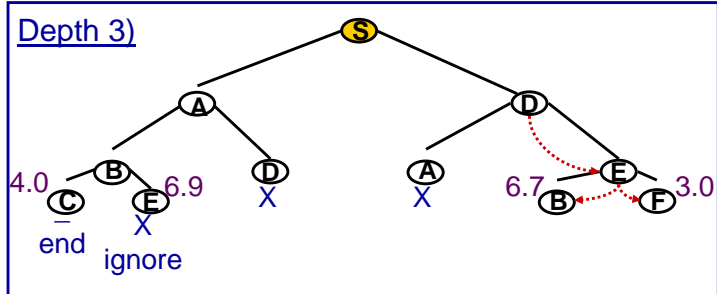
- Perform breadth-first, BUT:

- Only keep the WIDTH best new nodes

depending on heuristic at each new level.

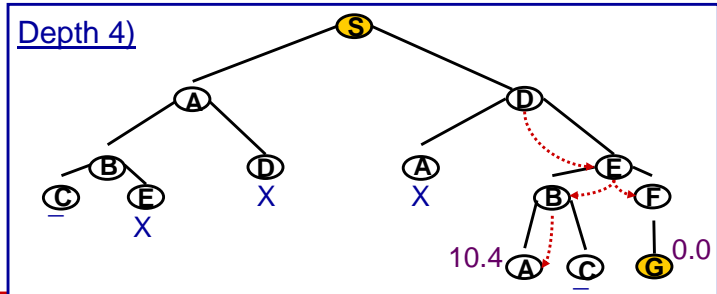
4

### Depth 3)



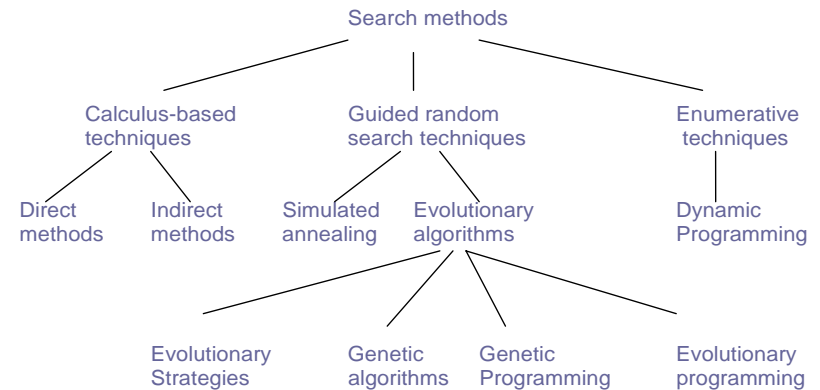
- Optimization: ignore leafs that are not goal nodes (see C)

### Depth 4)



5

## Taxonomy of Search Methods



6

## Simulated Annealing

- Introduced by Metropolis et al. in 1953 and adapted by Kirkpatrick (1983) in order to find an optimum solution to large-scale combinatorial optimization problems
- It is a derivative-free optimization method
- SA was derived from physical characteristics of spin glasses

7

## Simulated Annealing

- The principle behind SA is similar to what happens when metals are cooled at a controlled rate
- The slowly decrease of temperature allows the atoms in the molten metal to line themselves up an form a regular crystalline structure that possesses a low density and a low energy

8



## Simulated Annealing

---

- The value of an objective function that we intend to minimize corresponds to the energy in a thermodynamic system
- High temperatures corresponds to the case where high-mobility atoms can orient themselves with other non-local atoms and the energy can increase: function evaluation accepts new points with higher energy

9

## Simulated Annealing

---

- Low temperatures corresponds to the case where the low-mobility atoms can only orient themselves with local atoms & the energy state is not likely to increase: Function evaluation is performed **only** locally and points with higher energy are more and more refused
- The most important element of SA is the so-called **annealing schedule** (or **cooling schedule**) which express how fast the temperature is being lowered from high to low

10

## Simulated Annealing Terminology:

---

- **Objective function  $E(x)$** : function to be optimized
- **Move set**: set of next points to explore

$\Delta E = x_{\text{new}} - x$  is a random variable with pdf =  $g(.,.)$

- **Generating function**: pdf for selecting next point

$$g(\Delta x, T) = (2\pi T)^{-n/2} \exp[-||\Delta x||^2 / 2T]$$

(n is the explored space dimension)

11

## Simulated Annealing Terminology

---

- **Acceptance function  $h(\Delta E, T)$** : to determine if the selected point should be accepted or not. Usually
$$h(\Delta E, T) = 1/(1+\exp(\Delta E/(cT))).$$
- **Annealing (cooling) schedule**: schedule for reducing the temperature  $T$

12

## Basic steps involved in a general SA method

---

- **Step 1 [Initialization]:** Choose an initial point  $x$  and a high temperature  $T$  and set the iteration count  $k$  to 1
- **Step 2 [Evaluation]:** Evaluate the objective function  $E = f(x)$
- **Step 3 [Exploration]:** Select  $\Delta x$  with probability  $g(\Delta x, T)$ , and set  $x_{\text{new}} = x + \Delta x$
- **Step 4 [Reevaluation]:** Compute the new value of the objective function  $E_{\text{new}} = f(x_{\text{new}})$

13

## Basic step involved in a general SA method (cont.)

---

- **Step 5 [Acceptance]:** Test if  $x_{\text{new}}$  is accepted or not by computing  $h(\Delta E, T)$  where  $\Delta E = E_{\text{new}} - E$
- **Step 6:** Lower the temperature according to the annealing schedule ( $T = \eta T$ ;  $0 < \eta < 1$ )
- **Step 7:**  $k = k + 1$ , if  $k$  reaches the maximum iteration count, then stop otherwise go to step 3

14

## Tabu Search

---

- Tabu: prevent returning quickly to same state.
- Implementation: Keep fixed length queue ("tabu list"): add most recent step to queue; drop "oldest" step.
- Never make step that's currently on the tabu list.
- Quite powerful; competitive with simulated annealing

15

## Basic Evolutionary Algorithm

---

```
t:=0;           {initialize time}
InitPopulation(P,t); {initialize random population of individuals}
EvaluateFitness(P,t); {evaluate fitness of all initial individuals}
                  {in the population}
while not terminate(P,t) do {test for termination criterion, e.g.
                           { # of generations,satisfactory fitness, etc.}
begin
  t:=t+1;           {increase time}
  SelectParents(P,Ps); {select subpopulation for reproduction}
  Recombine(Ps);     {recombine the genes of selected
                    {parents}
  Mutate(Ps);        {mutate (perturb randomly) the}
                    {mated population}
  EvaluateFitness(Ps,t); {evaluate new fitness}
  Survive(P,Ps);      {select the survivors using actual}
                    {fitnesses}
end;
```

16

## Genetic Algorithms

- A genetic algorithm has the following components:
- a mechanism to encode solutions to problems as strings,
- a population of solutions represented as strings,
- a problem dependent fitness function,
- a selection mechanism,
- crossover and mutation operators.

17

## Encoding Mechanism

- Fixed length binary string are used to represent individuals
- Simple Binary Coding
  - Example:
    - {000, 001, 010, 011, 100, 101, 110, 111}
- Gray Coding
  - Example:
    - {000, 001, 011, 010, 110, 111, 101, 100}

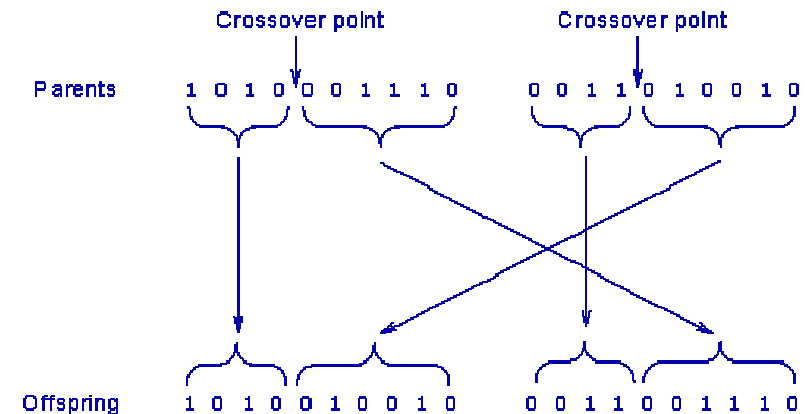
18

## Selection

- Selection is based on the survival of the fittest mechanism of nature.
- Common Selection Schemes:
  - Proportionate selection scheme
    - Number of offsprings= $f_s/f_a$
    - where  $f_s$  is the fitness value
    - $f_a$  is the average fitness value of the population
  - Tournament selection scheme
  - Rank based selection
  - Elitist strategies

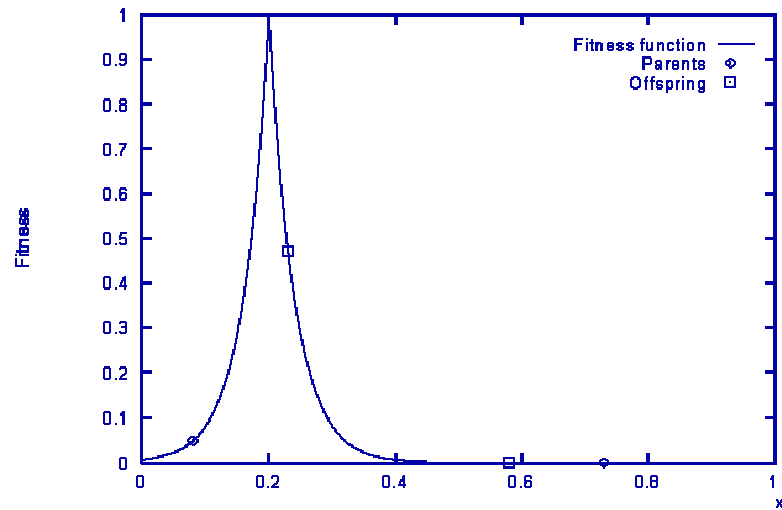
19

## Crossover



20

## Example of Crossover



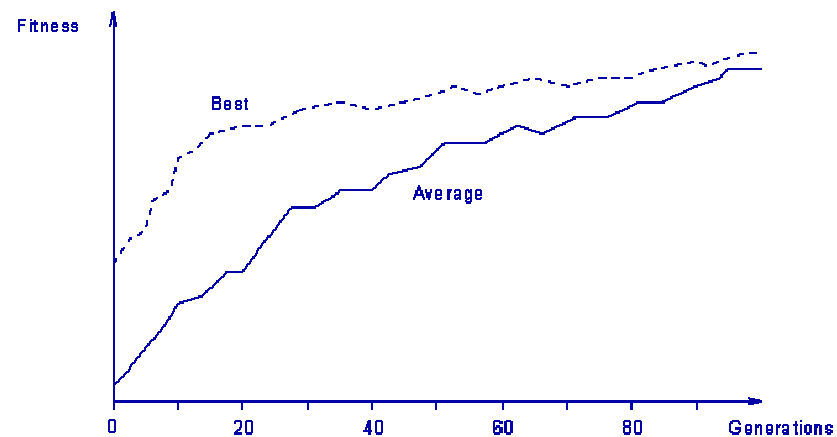
21

## Mutation

Mutation point  
↓  
Offspring    1 0 1 0 0 1 0 0 1 0  
Mutated Offspring    1 0 1 0 1 1 0 0 1 0

22

## Typical GA Convergence Behavior



23

## Control Parameters of GA

- Population size
  - Typically 30-200
- Crossover rate
  - Typically 0.5-1.0
- Mutation rate
  - Typically 0.001-0.05
- Stopping Criteria
  - Reaching a fixed number of iterations,
  - Evolving a string with a high fitness value,
  - Creation of a certain degree of homogeneity within the population.

24

## A GA Example

- **Problem:** Assume that we want to find the maximum of the following function  $f(x,y)$  :

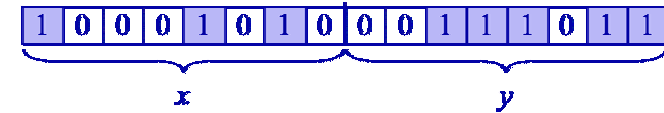
$$f(x,y) = (1-x)^2 e^{-x^2-(y+1)^2} - (x-x^3-y^3) e^{-x^2-y^2}$$

- where independent variables  $x$  and  $y$  vary between -3 and 3.

25

## Encoding:

- The problem variables are represented as a chromosome - variables  $x$  and  $y$  as a concatenated binary string.



- Both of them are allocated 8 bits, consequently the chromosome is 16 bits long.
- Here for simplicity we use binary to decimal conversion.

$$(10001010)_2 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 0 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = (138)_{10}$$

and

$$(00111011)_2 = 0 \times 2^7 + 0 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = (59)_{10}$$

26

## Normalization and Mapping

- The range of integers that can be handled by 8-bits, that is the range from 0 to  $(2^8 - 1)$ , is mapped to the actual range of parameters  $x$  and  $y$ , that is the range from -3 to 3:

$$\frac{6}{256-1} = 0.0235294$$

- To obtain the actual values of  $x$  and  $y$ , we multiply their decimal values by 0.0235294 and subtract 3 from the results:

$$x = (138)_{10} \times 0.0235294 - 3 = 0.2470588$$

and

$$y = (59)_{10} \times 0.0235294 - 3 = -1.6117647$$

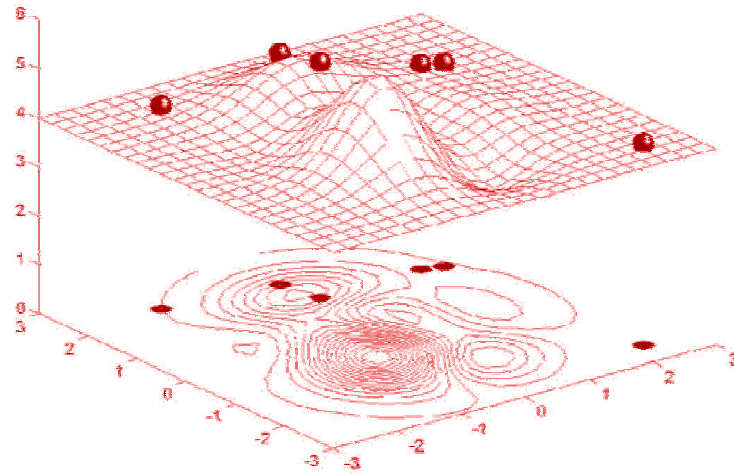
27

## GA Parameters

- The following parameters are used:
  - Population size = 6
  - Crossover probability = 0.7
  - Mutation probability = 0.001.
  - Maximum number of generations = 100

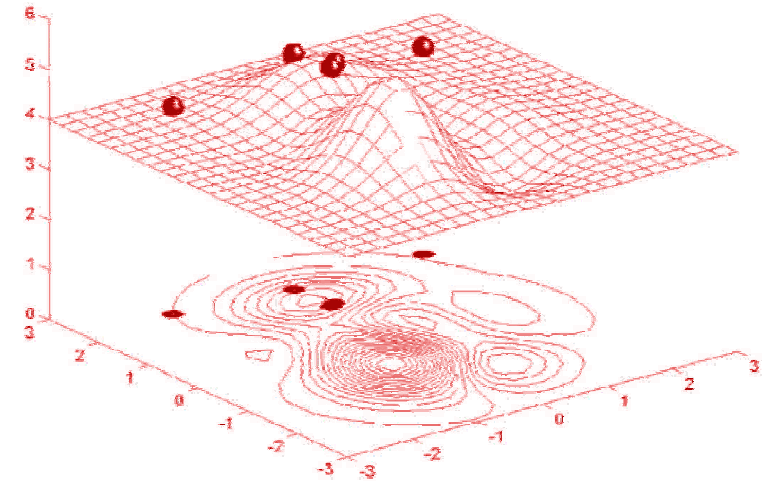
28

## Initial Population



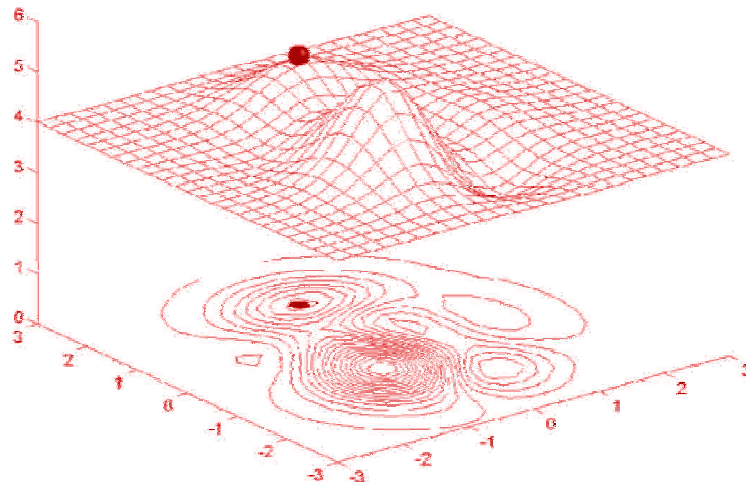
29

## First generation



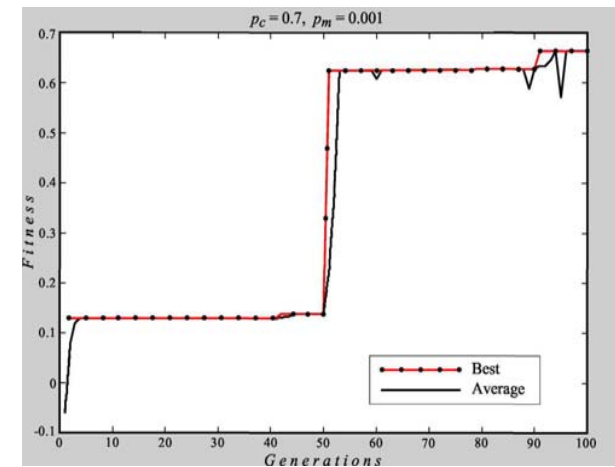
30

## Local maximum



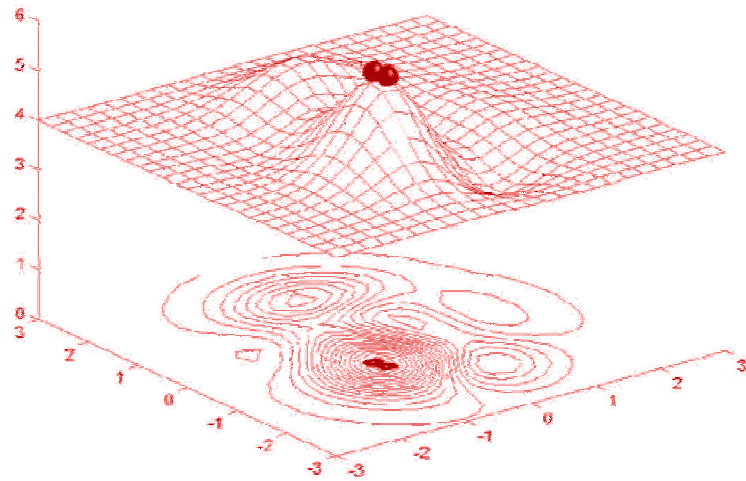
31

## Fitness vs Generations: Local Maximum Case



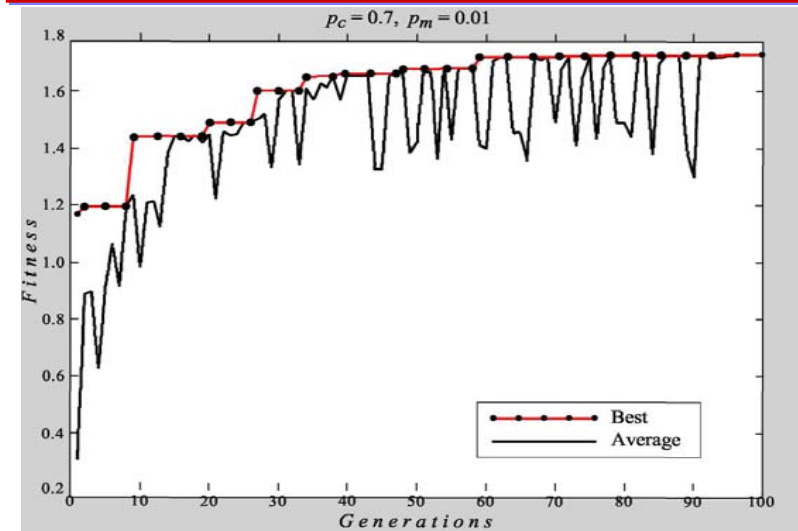
32

## Global Maximum



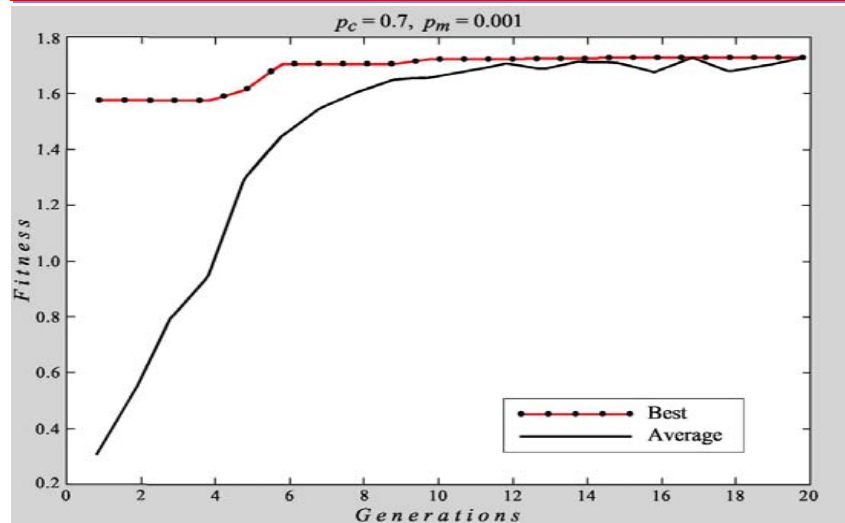
33

## Fitness vs Generations: Global Maximum Case (6 chromosomes)



34

## Fitness vs Generations: Global Maximum Case (60 chromosomes)



35

## Evolutionary Programming

- Real valued representation
- No recombination

36

## Evolutionary Strategies

- Real valued representation
  - Object Variables
  - Strategy Variables
- Main operator
  - Mutation
- Secondary operator
  - Recombination

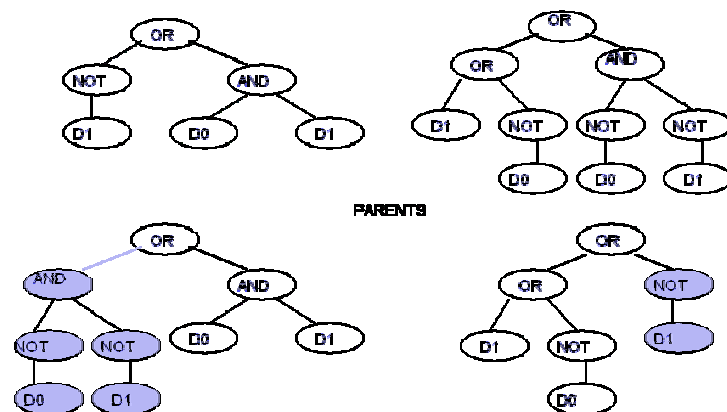
37

## Genetic Programming

- Individuals
  - Hierarchically structured computer programs, i.e., the set of all possible combinations of functions that can be composed recursively from the available set of  $n$  functions  $F=\{f_1, f_2, \dots, f_n\}$  and the available set of  $m$  terminals  $T=\{a_1, a_2, \dots, a_n\}$ .
- Main operators
  - Fitness proportionate selection
  - Recombination
- Secondary operators
  - Mutation
  - Permutation
  - Editing
  - Define Building Block

38

## Recombination Example for GP



39

## Main Characteristics of Evolutionary Algorithms

	ES	EP	GA	GP
<b>Representation</b>	Real-valued	Real-valued	Binary-Valued	Lisp S-expressions
<b>Self-Adaptation</b>	Standard deviations and covariances	Variance	None	None
<b>Fitness</b>	Objective function values	Scaled objective function value	Scaled objective function value	Scaled objective function value
<b>Mutation</b>	Main operator	Only operator	Background operator	Background operator
<b>Recombination</b>	Different variants, important for self-adaptation	None	Main Operator	Main Operator
<b>Selection</b>	Deterministic, extinctive	Probabilistic, extinctive	Probabilistic, preservative	Probabilistic, preservative

40



## Applications of Evolutionary Algorithms

---

- Some interesting examples of EA applications:
- Time-tabling
- Job-shop scheduling:
- Game playing:
- Computer Aided Design (CAD)
- Face Recognition
- Financial Time-Series Prediction