CSE 412: Artificial Intelligence

Topic – 4: Informed Search and Exploration

Department of Computer Science and Engineering
Daffodil International University

Topic Contents

- Informed (Heuristic) Search Strategies
- Heuristic Functions
- Local Search Algorithms and Optimization Problems
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms

Best-First Search

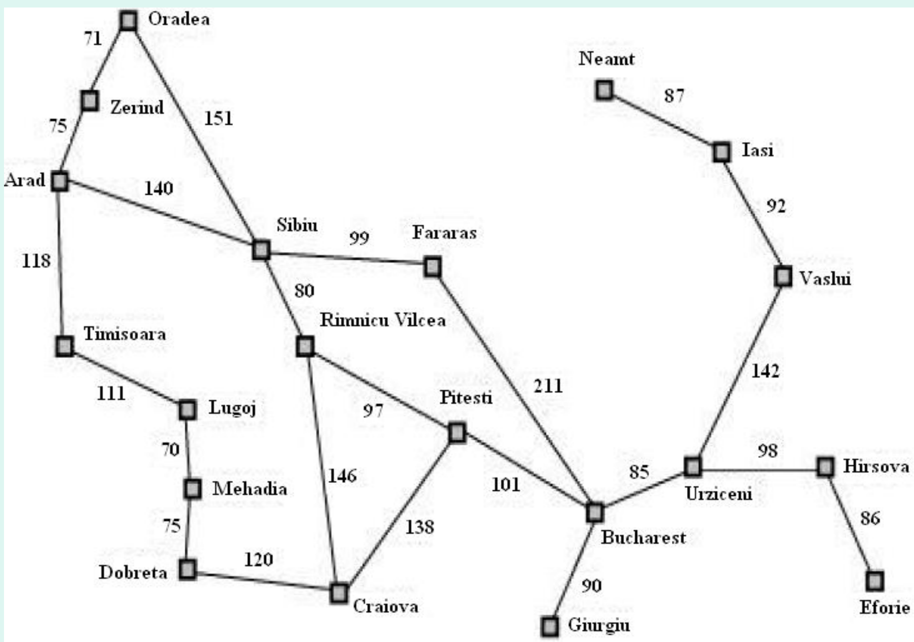
- Node is selected for expansion based on an *evaluation function f(n)*
- Evaluation function estimates distance to the goal
- Choose node which *appears* best
- Implementation:
  – fringe is a priority queue sorted in ascending order of *f*-values

A Heuristic Function *h*(*n*)

- Dictionary definition: *"A rule of thumb, simplification, or educated guess that reduces or limits the search for solutions in domains that are difficult and poorly understood"*
- For best-first search:
  - *h(n)* = estimated cost of the cheapest path from node *n* to goal node

Romania with Step Costs in Km

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnico Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

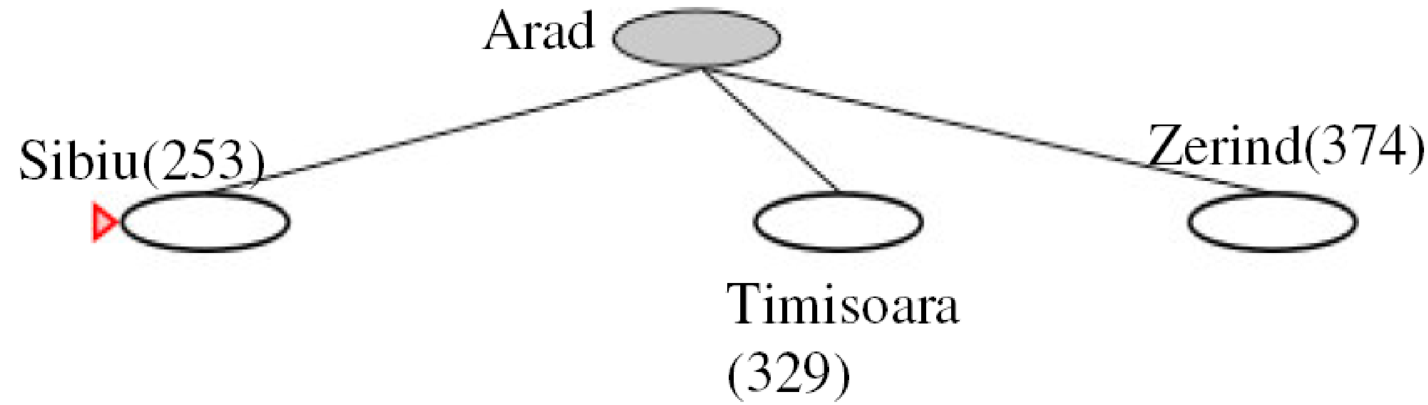


- h_{SLD} = straight-line distance heuristic
- h_{SLD} cannot be computed from the problem description itself
- In **greedy best-first search** *f(n)=h(n)*
  - Expand node that is closest to goal

Greedy Search: Example

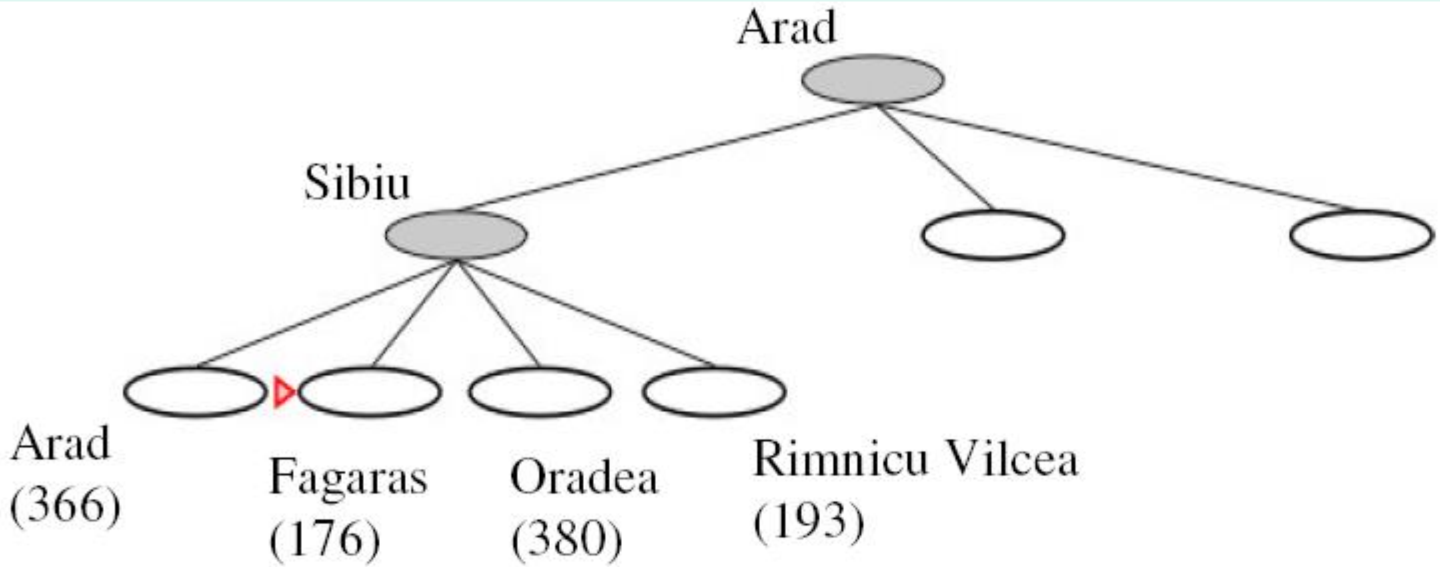
- Greedy search to solve the Arad to Bucharest problem

Greedy Search: Example



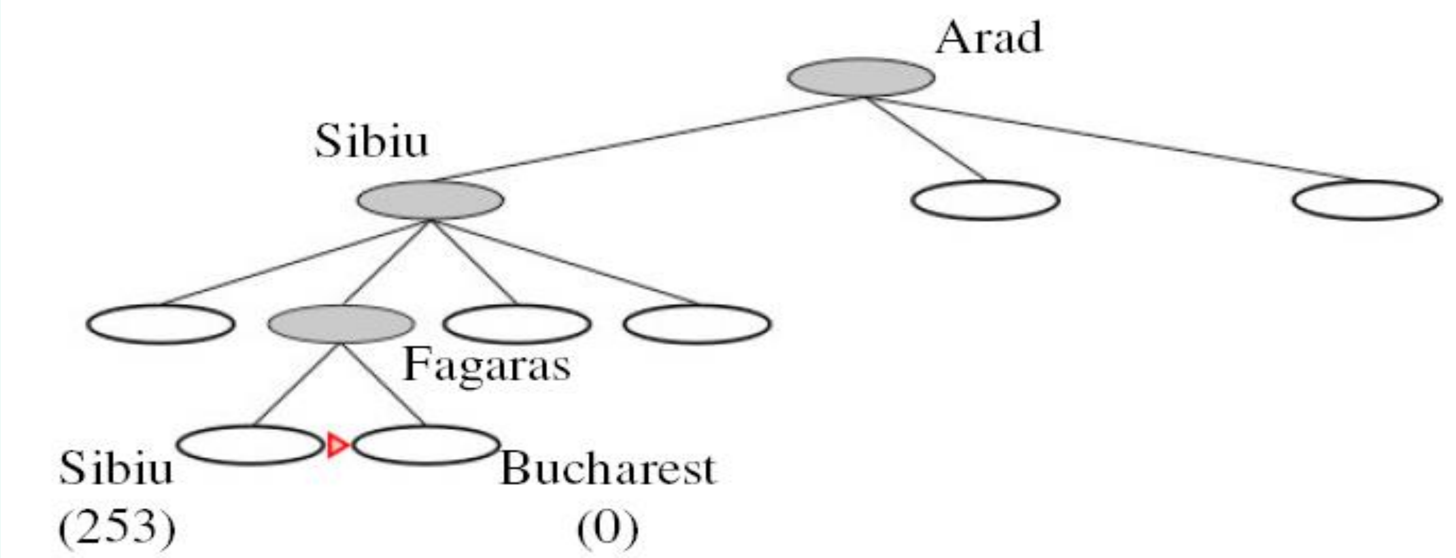
- Greedy best-first search will select Sibiu

Greedy Search: Example



- Greedy best-first search will select Fagaras

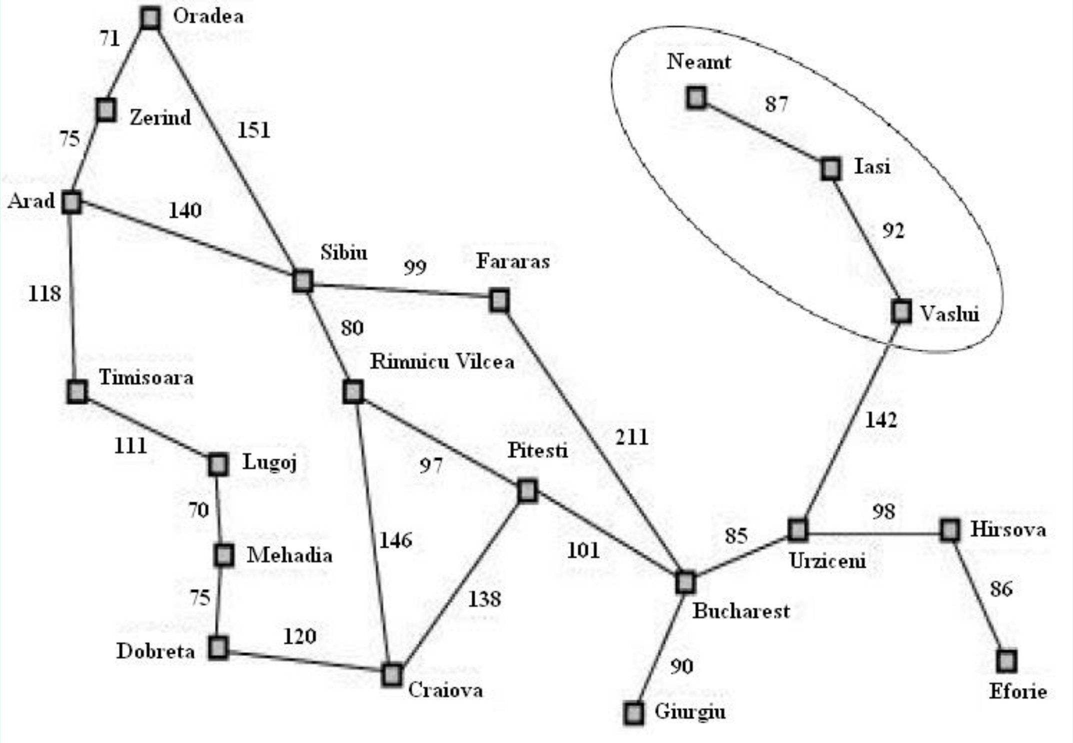
Greedy Search: Example



- Goal reached
  - For this example no node is expanded that is not on the solution path
  - But not optimal (see Arad, Sibiu, Rimnicu Vilcea, Pitesti)

Greedy Search: Evaluation

- Complete or optimal: no
  - Minimizing *h*(*n*) can result in false starts, e.g. Iasi to Fagaras
  - Check on repeated states



Greedy Search: Evaluation

- Time and space complexity:
  - In the worst case all the nodes in the search tree are generated: $O(b^m)$

    (*m* is maximum depth of search tree and *b* is branching factor)
  - But: choice of a good heuristic can give dramatic improvement

A* Search

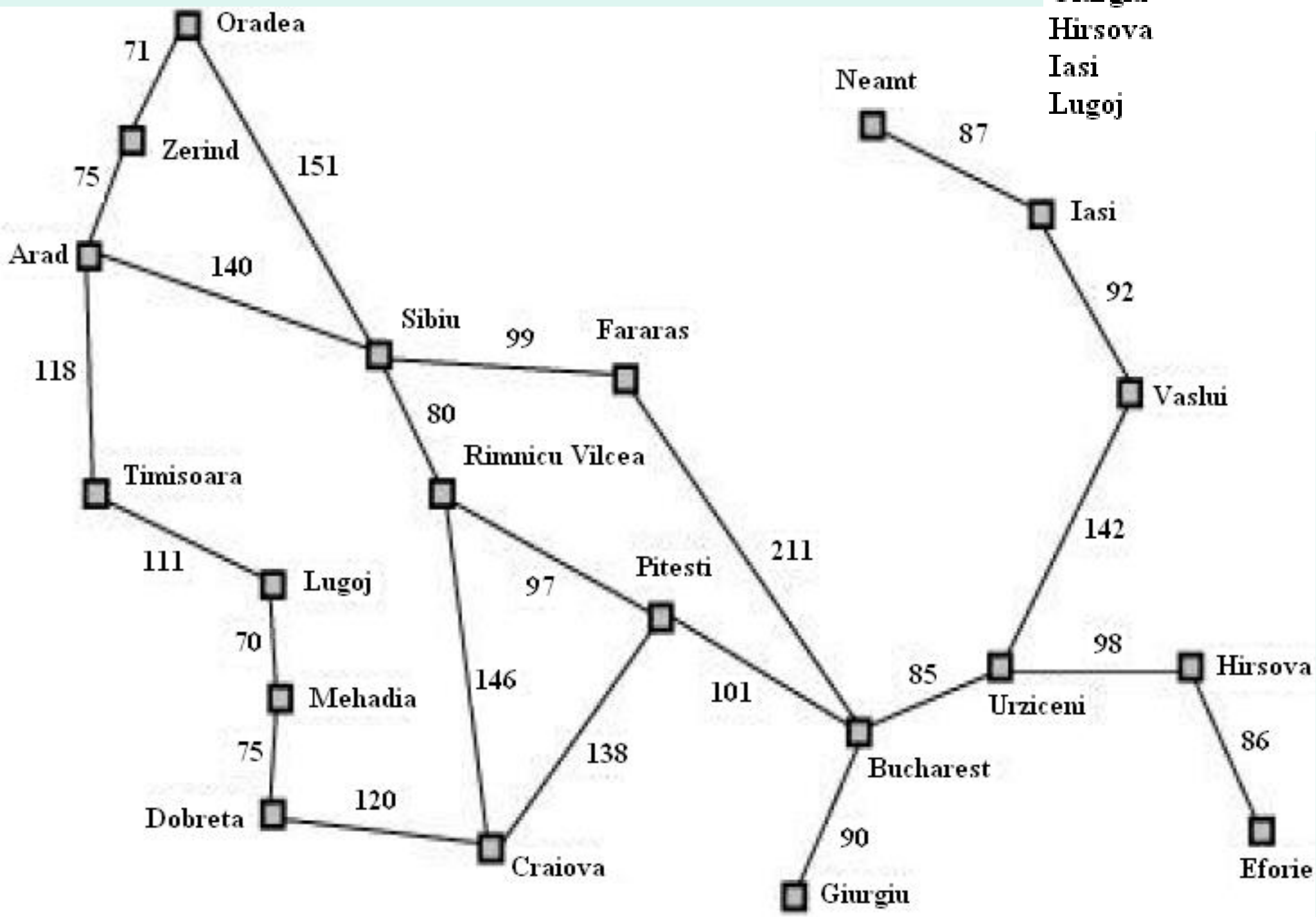
- Best-known form of best-first search
- Idea: avoid expanding paths that are already expensive
- Evaluation function $f(n) = g(n) + h(n)$
  - $g(n)$: the cost (so far) to reach the node
  - $h(n)$: estimated cost to get from the node to the goal
  - $f(n)$: estimated total cost of path through n to goal
- A* search is both complete and optimal if $h(n)$ satisfies certain conditions

A* Search

- A* search is optimal if h(n) is an **admissible heuristic**
- A heuristic is admissible if it *never overestimates* the cost to reach the goal
  - $h(n) \leq h^*(n)$ where $h^*(n)$ is the true cost from n
- Admissible heuristics are optimistic about the cost of solving the problem
- e.g. $h_{SLD}(n)$ never overestimates the actual road distance

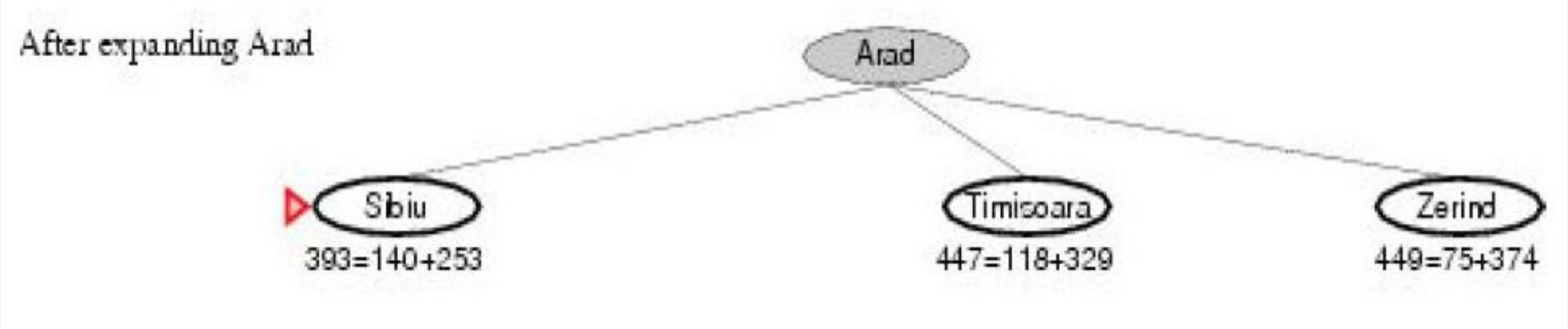
Romania Example

Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimnico Vilcea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urziceni	80
Iasi	226	Vaslui	199
Lugoj	244	Zerind	374

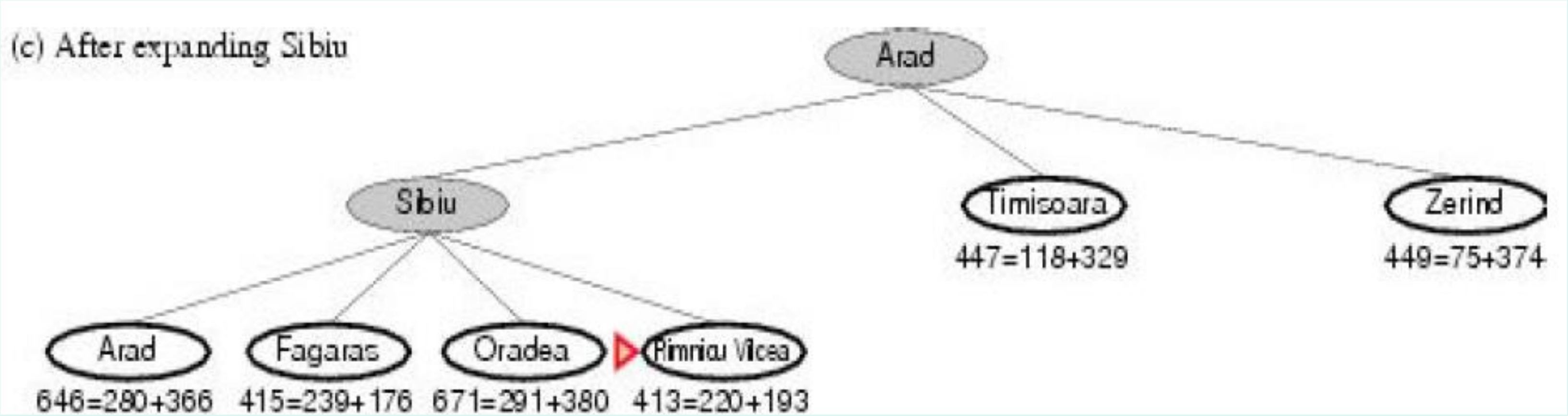


A* Search: Example

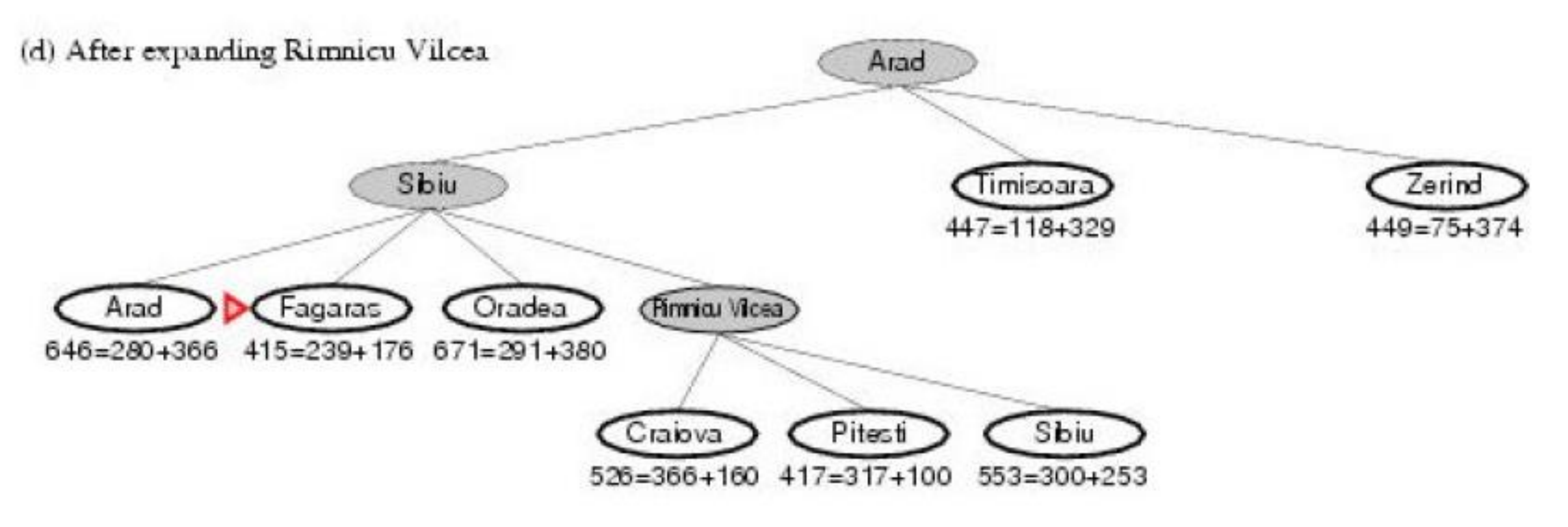
A* Search: Example



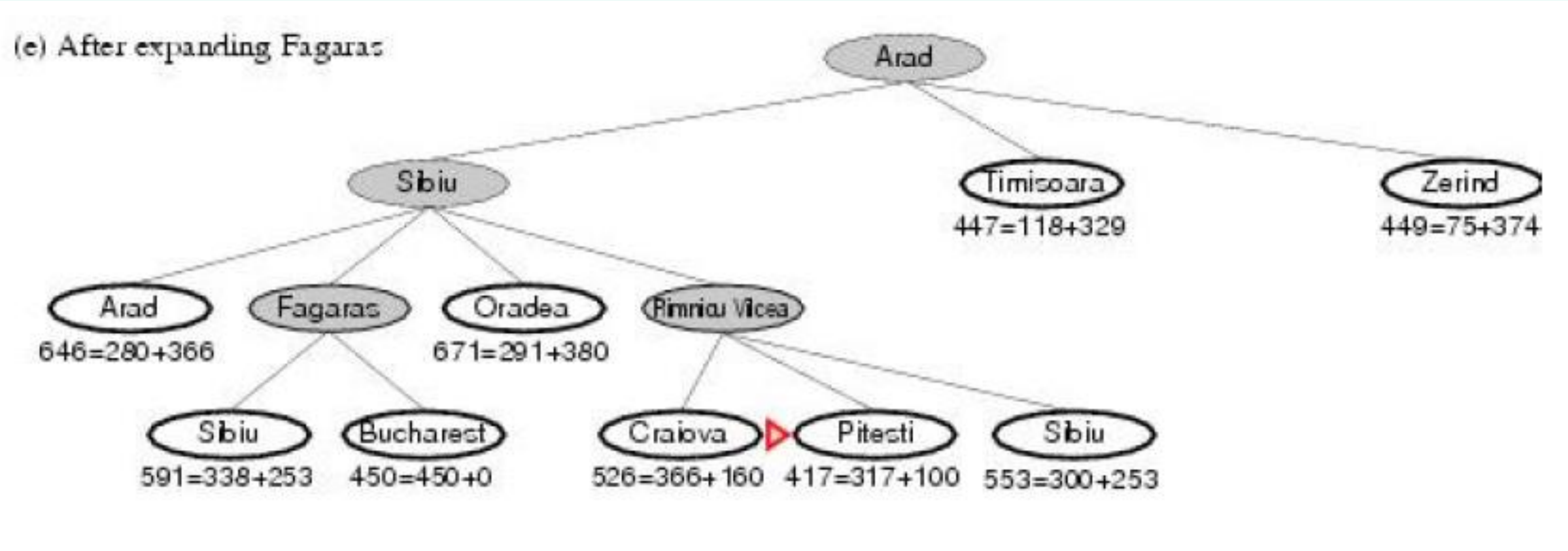
A* Search: Example



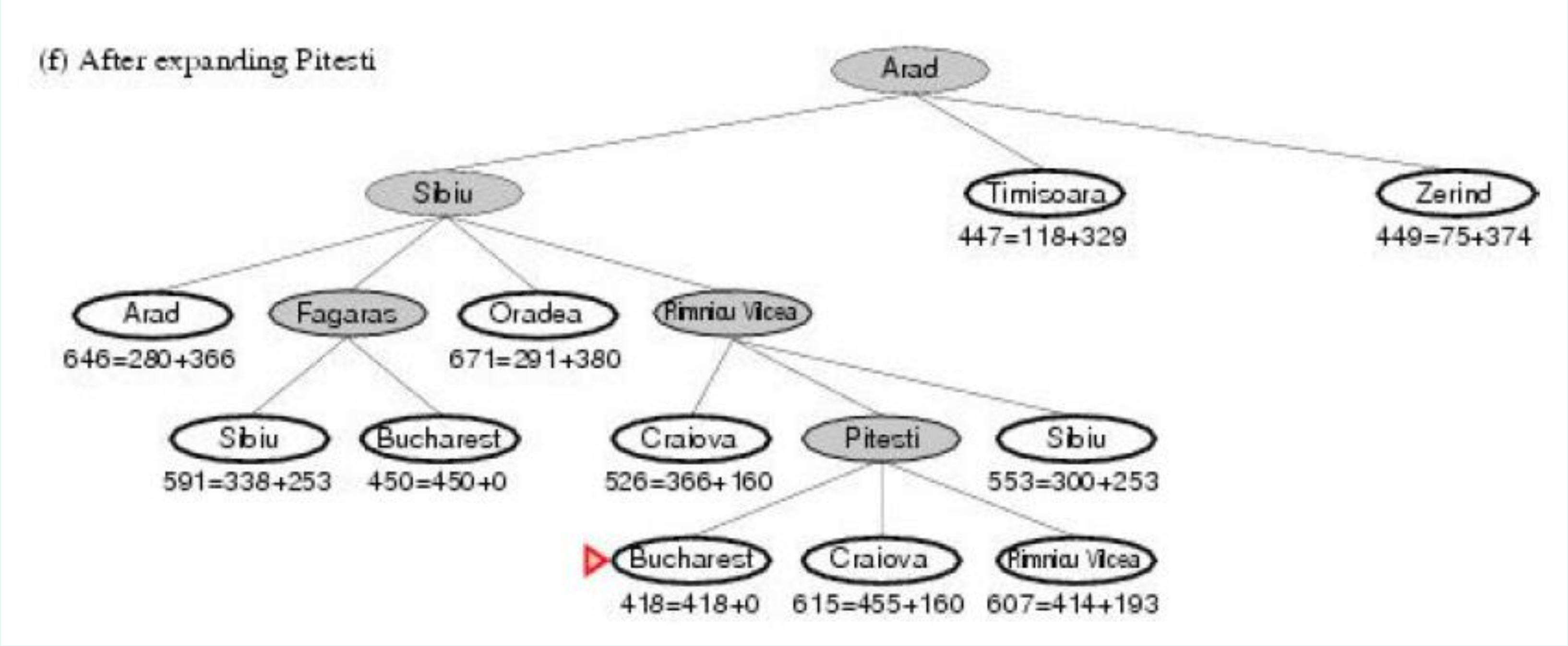
A* Search: Example



A* Search: Example



A* Search: Example



Optimality of A*

- Suppose a suboptimal goal node G_2 appears on the fringe and let the cost of the optimal solution be C^*.
- Since G_2 is suboptimal and

  $h(G_2) = 0$ (true for any goal node),

  we know that $f(G_2) = g(G_2) + h(G_2) = g(G_2) > C^*$
- Now consider a fringe node n that is on an optimal solution path.
- If $h(n)$ does not overestimate the cost of completing the solution path,

  then we know that $f(n) = g(n) + h(n) \leq C^*$
- Since $f(n) \leq C^* < f(G_2)$, G_2 will not be expanded and A* search must return an optimal solution.

A* Search: Evaluation

- Complete: yes
  - Unless there are infinitely many nodes with $f < f(G)$
- Optimal: yes
  - A* is also **optimally efficient** for any given $h(n)$. That is, no other optimal algorithm is guaranteed to expand fewer nodes than A*.

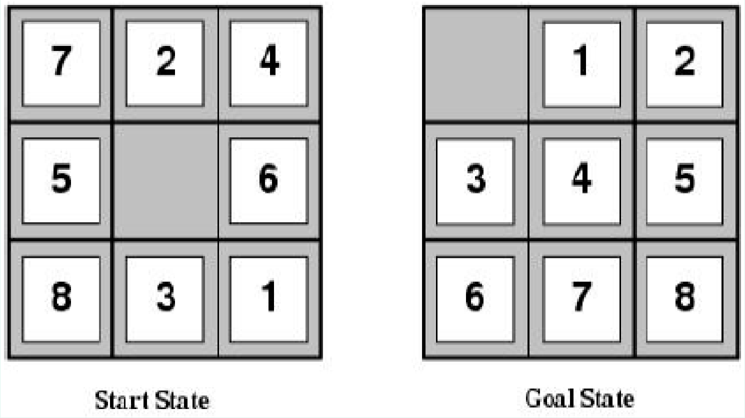
A* Search: Evaluation

- Time complexity:
  - number of nodes expanded is still exponential in length of solution
- Space complexity:
  - All generated nodes are kept in memory
  - A* usually runs out of space before running out of time

Heuristic Functions

- Let us see two heuristics for a problem.

Example: 8-puzzle

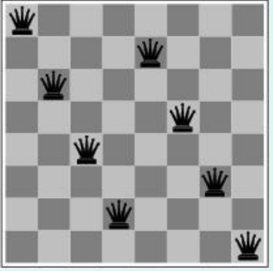


- **States:** location of each tile plus blank
- **Initial state:** Any state can be initial
- **Actions:** Move blank {*Left*, *Right*, *Up*, *Down*}
- **Goal test:** Check whether goal configuration is reached
- **Path cost:** Number of actions to reach goal

Example: 8-puzzle

- For 8-puzzle problem:
  - h_1 = number of tiles out of place. In the example h_1= 8
  - h_2 = total Manhattan distance of errant pieces. In the example

$$h_2 = 3+1+2+2+2+3+3+2 = 18$$



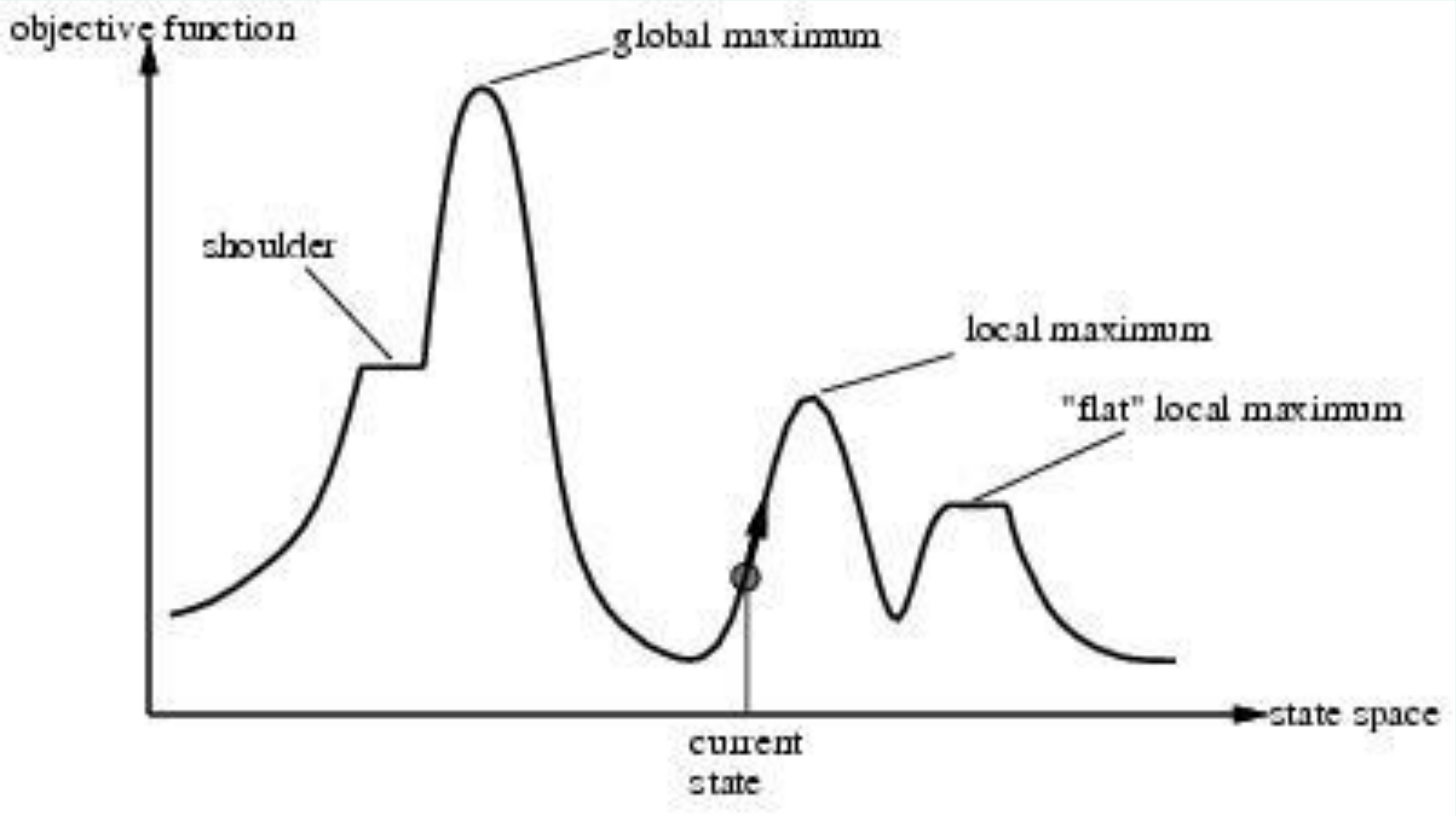
Local Search Algorithms and Optimization Problems

- Previously: systematic exploration of search space
  - Path to goal is solution to problem
- for some problem classes, it is sufficient to find a solution
  - the path to the solution is not relevant, e.g. 8-queens
- Different algorithms can be used
  - Local search
- memory requirements can be dramatically relaxed by modifying the current state
  - all previous states can be discarded
  - since only information about the current state is kept, such methods are called ***local***

Local Search Algorithms and Optimization Problems

- Local search = use single current state and move to neighboring states.
- Advantages:
  - Use very little memory
  - Find often reasonable solutions in large or infinite state spaces.
- Are also useful for pure optimization problems.
  - Find best state according to some *objective function*.
  - e.g. survival of the fittest as a metaphor for optimization.

Local Search Algorithms and Optimization Problems



Hill-Climbing Search

- continually moves uphill
  - increasing value of the evaluation function
  - ***gradient descent search*** is a variation that moves downhill
- is a loop that continuously moves in the direction of increasing value - that is, uphill.
  - It terminates when a peak is reached.

Hill-Climbing Search

- Hill-climbing does not look ahead of the immediate neighbors of the current state.
- Hill-climbing chooses randomly among the set of best successors, if there is more than one.
- Hill-climbing is also called *greedy local search*
- Hill-climbing is a very simple strategy with low space requirements
  - stores only the state and its evaluation, no search tree
-

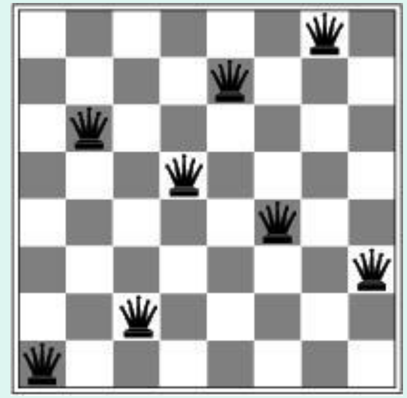
Hill-Climbing Example

- 8-queens problem (complete-state formulation).
- Successor function: 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

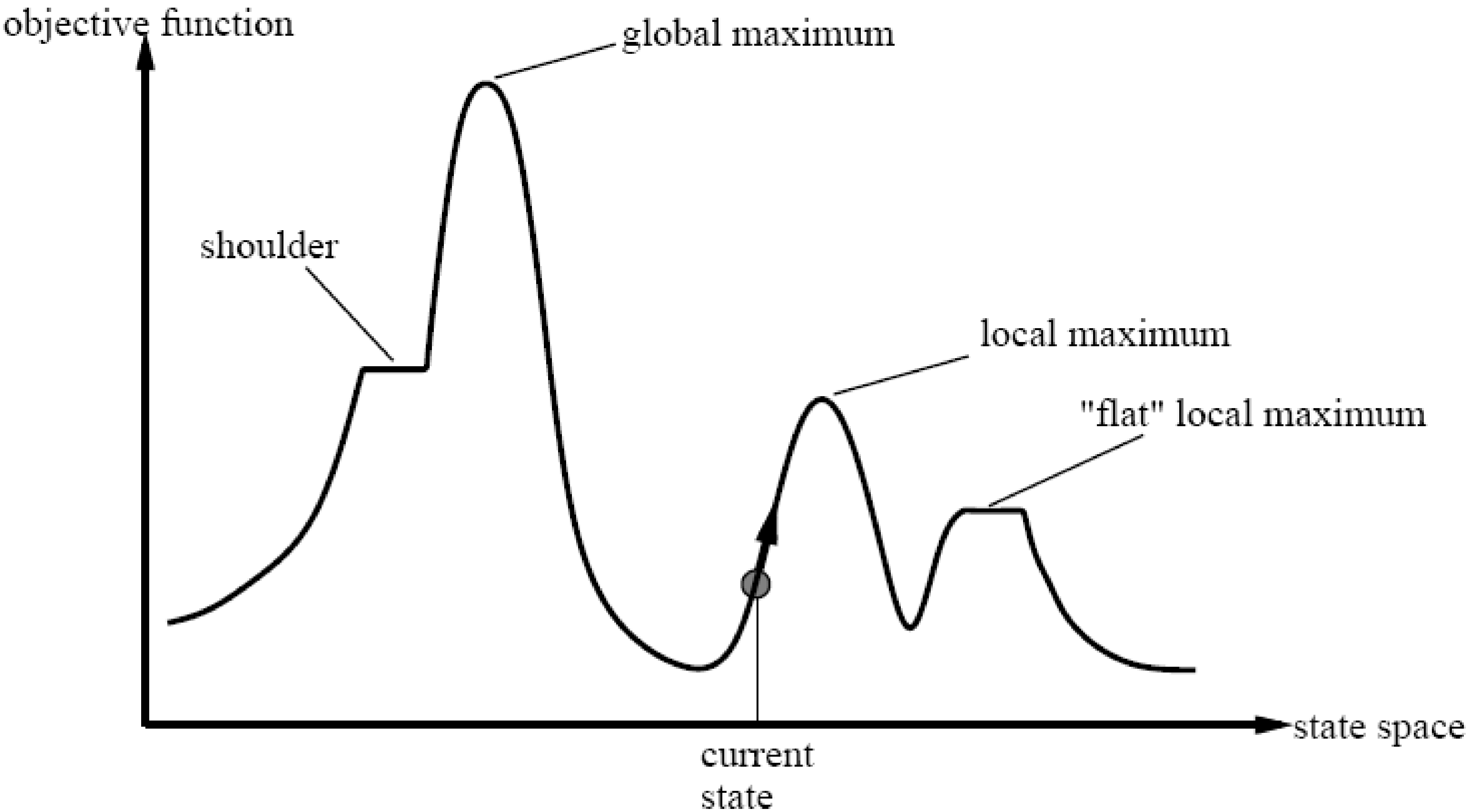
(a)

(b)

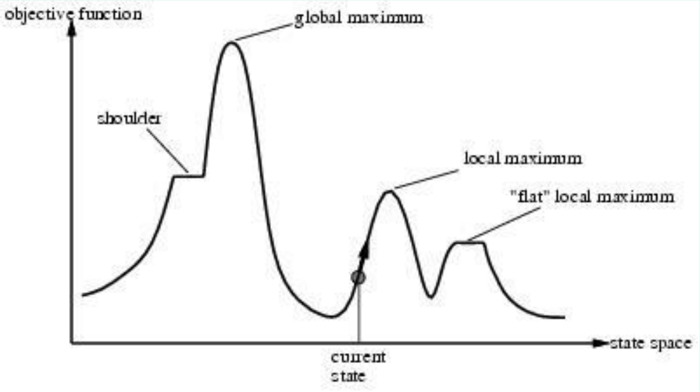


(a) An 8-queens state with heuristic cost estimate *h* = 17, showing the value of *h* for each possible successor obtained by moving a queen within its column. The best moves are marked.

(b) A local minimum in the 8-queens state space; the state has *h* = 1 but every successor has a higher cost.


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

Drawbacks



- Ridge = sequence of local maxima difficult for greedy algorithms to navigate
- Plateaux = an area of the state space where the evaluation function is flat.
- Gets stuck 86% of the time.

Hill-Climbing Search

- problems
  - local maxima
    - algorithm can’t go higher, but is not at a satisfactory solution
  - plateau
    - area where the evaluation function is flat
  - ridges
    - search may oscillate slowly

Escaping Local Optima

* *HC gets stuck at local maxima limiting the quality of the solution found.*

- Two ways to modify HC:
  1. choice of neighbor
  2. criteria for accepting neighbor for current
- For example:
  1. choose neighbor randomly
  2. accept neighbor if it is better or if it isn't, accept with some fixed probability p

Escaping Local Optima

- Modified HC can escape local maxima but:
  - chance of making a bad move is the same at the beginning of the search as at the end
  - magnitude of improvement or lack of is ignored

☞ How can HC address these concerns?
  - fixed probability p that bad move is accepted can be replaced with a probability that decreases as the search proceeds
  - now as the search progresses, the chances of taking a bad move reduces

Hill-Climbing Search

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

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***
  - implements stochastic hill climbing by generating successors randomly until a better one is found.
- ***Random-restart hill-climbing***
  - Tries to avoid getting stuck in local maxima.

Simulated Annealing

- Escape local maxima by allowing “bad” moves.
  - Idea: but gradually decrease their size and frequency.
- Origin; metallurgical annealing
- Bouncing ball analogy:
  - Shaking hard (= high temperature).
  - Shaking less (= lower the temperature).
- If T decreases slowly enough, best state is reached.
- Applied for VLSI layout, airline scheduling, etc.

Simulated Annealing

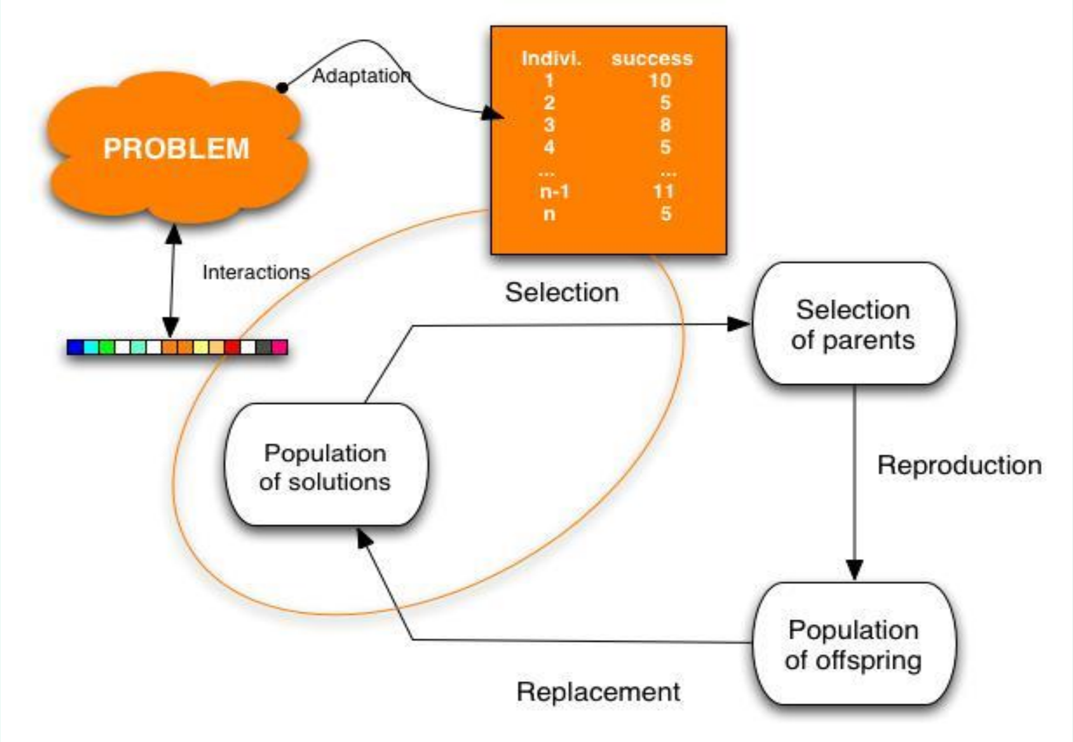
```
function SIMULATED-ANNEALING( problem, schedule) return a solution state
    input: problem, a problem
              schedule, a mapping from time to temperature
    local variables: current, a node.
                              next, a node.
                              T, a “temperature” controlling the probability of downward steps

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

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 $\rightarrow$ 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 at proportionally to state success.

Genetic Algorithms

- Variant of local beam search with *sexual recombination.*



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—most commonly, a string of 0s and 1s.
- For example, an 8-queens state must specify the positions of 8 queens, each in a column of 8 squares, and so requires $8 \times \log_2 8 = 24$ bits.
- Alternatively, the state could be represented as 8 digits, each in the range from 1 to 8.
- Figure 4.6(a) shows a population of four 8-digit strings representing 8-queens states.
-
-

Genetic Algorithms

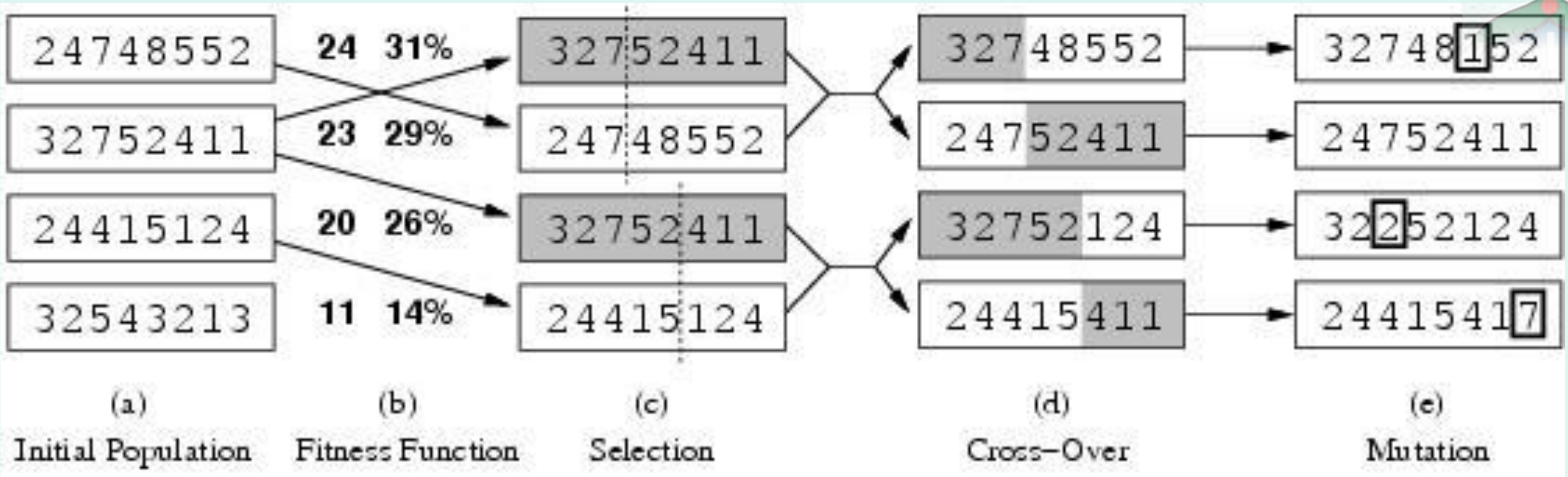


Figure 4.6: The three major operations in genetic algorithm. The initial population in **(a)** is ranked by the fitness function in **(b)**, resulting in pairs for mating in **(c)**. They produce offspring in **(d)**, which are subject to mutation in **(e)**.

Genetic Algorithms

- The 8-queens states involved in this reproduction step are shown in Figure 4.7.

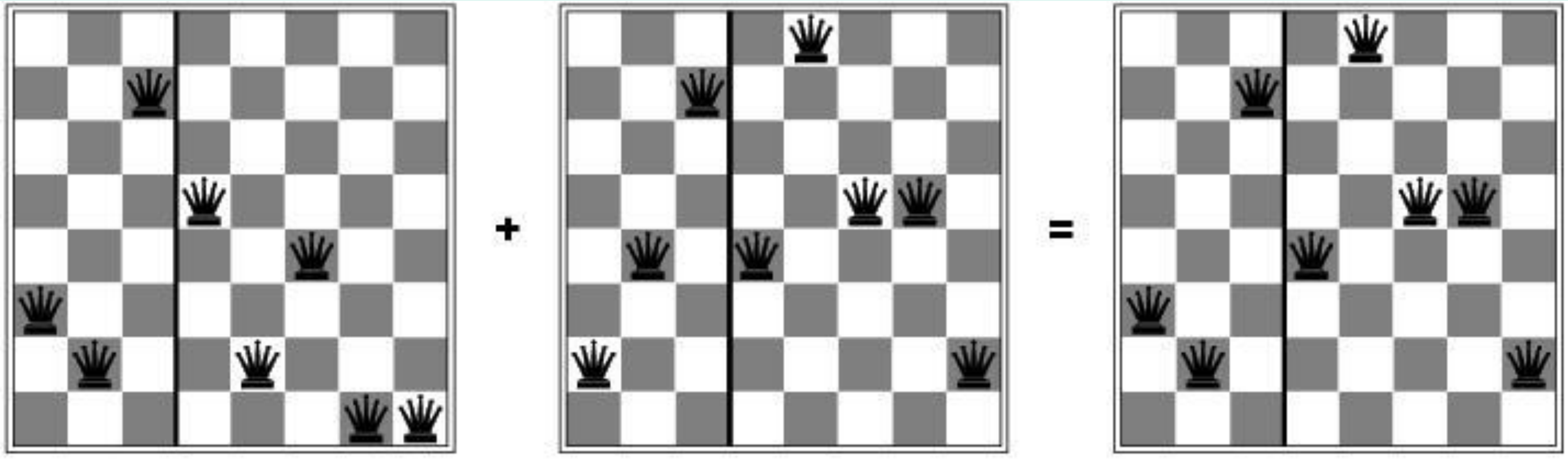


Figure 4.7: The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d).

Genetic Algorithms

- The production of the next generation of states is shown in Figure 4.6(b)–(e).
- In (b), each state is rated by the objective function, or (in GA terminology) the **fitness function**.
- A ***fitness function*** should return higher values for better states, so, for the 8-queens problem we use the number of *nonattacking* pairs of queens, which has a value of 28 for a solution.
- The values of the four states are 24, 23, 20, and 11.
- In this particular variant of the genetic algorithm, the probability of being chosen for reproducing is directly proportional to the fitness score, and the percentages are shown next to the raw scores.

Genetic Algorithms

- In (c), two pairs are selected at random for reproduction, in accordance with the probabilities in (b).
- Notice that one individual is selected twice and one not at all.
- For each pair to be mated, a crossover point is chosen randomly from the positions in the string.
- In Figure 4.6, the crossover points are after the third digit in the first pair and after the fifth digit in the second pair.
- In (d), the offspring themselves are created by crossing over the parent strings at the crossover point.
- For example, the first child of the first pair gets the first three digits from the first parent and the remaining digits from the second parent, whereas the second child gets the first three digits from the second parent and the rest from the first parent.

Genetic Algorithms

- The 8-queens states involved in this reproduction step are shown in Figure 4.7.

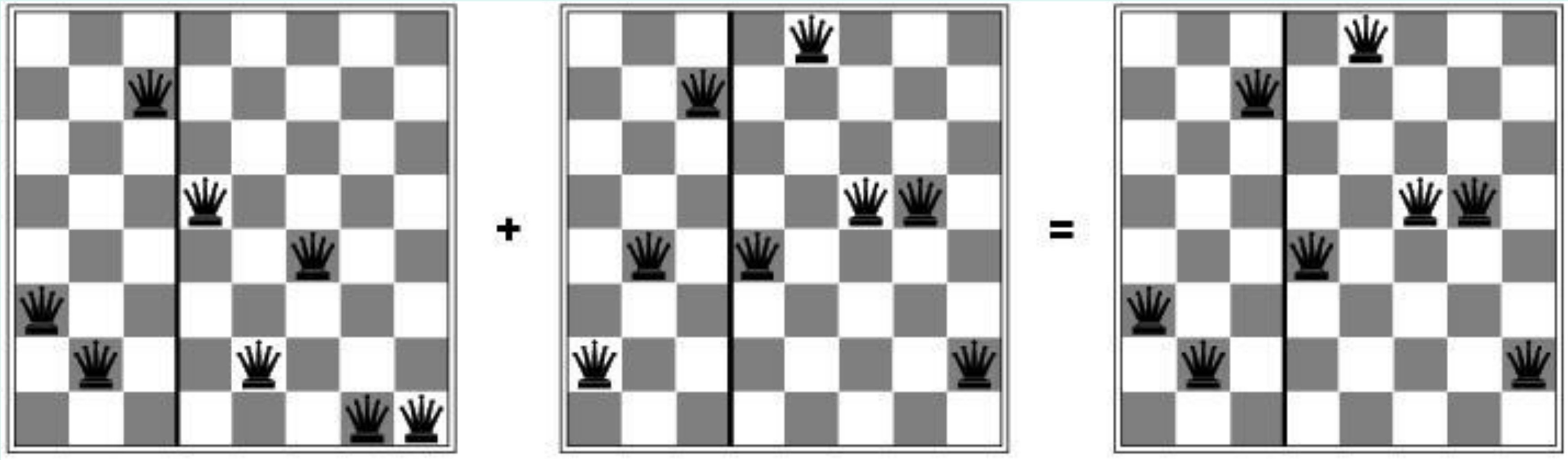


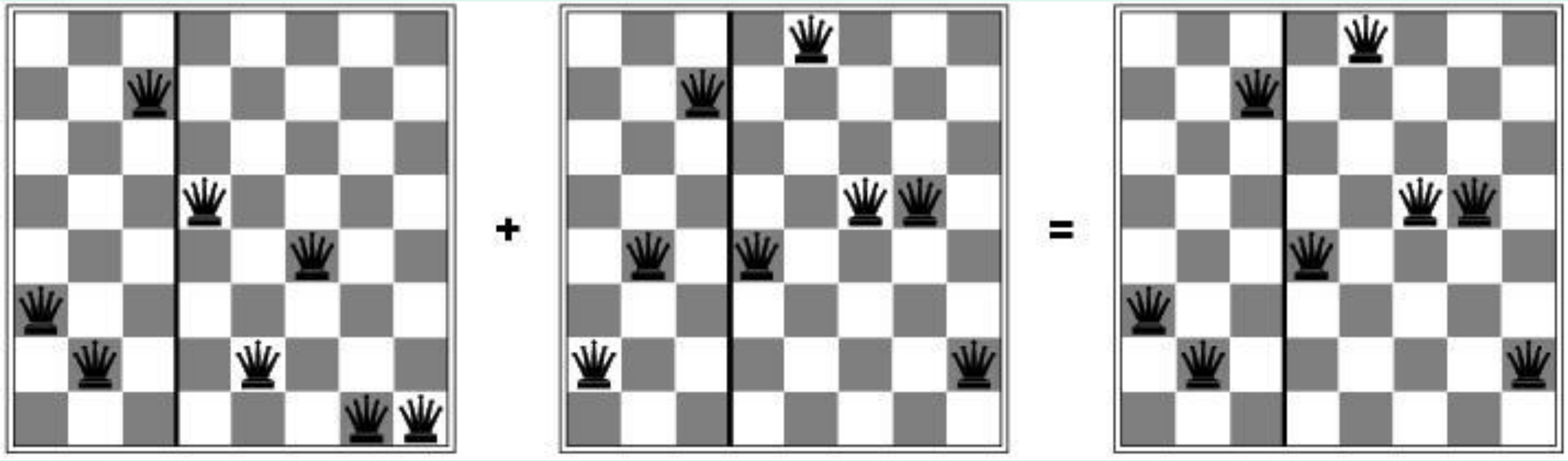
Figure 4.7: The 8-queens states corresponding to the first two parents in Figure 4.6(c) and the first offspring in Figure 4.6(d).

Genetic Algorithms

- The example shows that when two parent states are quite different, the crossover operation can produce a state that is a long way from either parent state.
- It is often the case that the population is quite diverse early on in the process, so crossover (like simulated annealing) frequently takes large steps in the state space early in the search process and smaller steps later on when must individuals are quite similar.

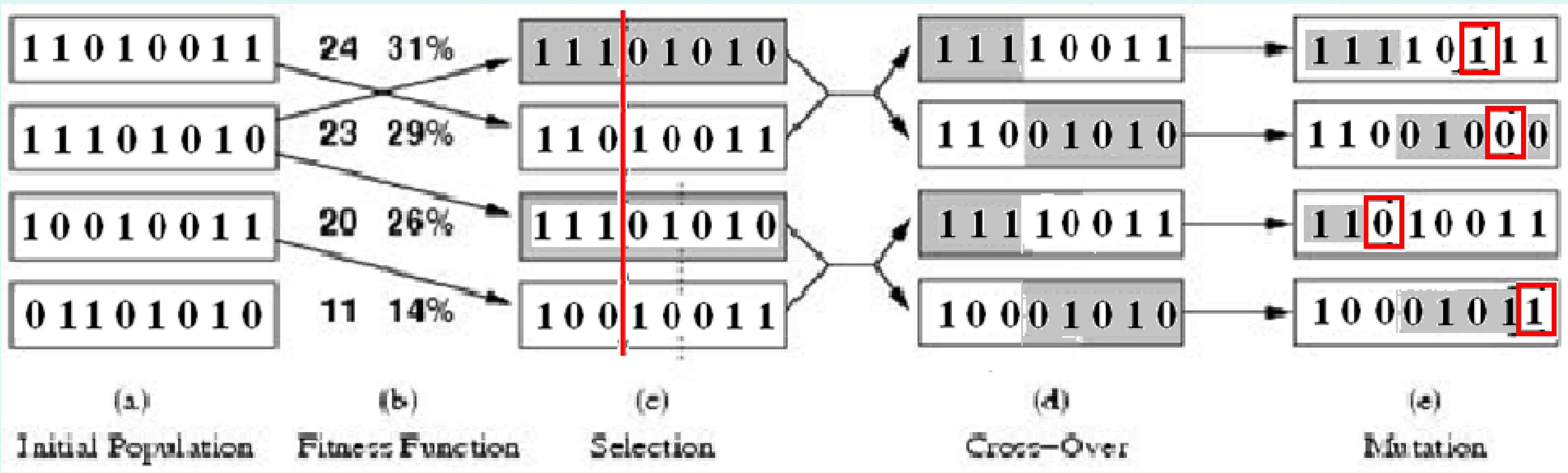
Genetic Algorithms

- Finally, in (e), each location is subject to random mutation with a small independent probability.
- One digit was mutated in the first, third, and fourth offspring.
- In the 8-queens problem, this corresponds to choosing a queen at random and moving it to a random square in its column.



Genetic Algorithms

Figure: A genetic algorithm. The algorithm is the same as the one diagrammed in the figure in slide no. 45, with one variation: in this more popular version, each mating of two parents produces only one offspring, not two.



Genetic Algorithms

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