# Foundations of Artificial Intelligence

## 4. Informed Search Methods

Heuristics, Local Search Methods, Genetic Algorithms

Wolfram Burgard, Bernhard Nebel, and Martin Riedmiller

Albert-Ludwigs-Universität Freiburg

Mai 17, 2011

# Contents

# Best-First Search

Search procedures differ in the way they determine the next node to expand.

**Uninformed Search:** Rigid procedure with no knowledge of the cost of a given node to the goal.

**Informed Search:** Knowledge of the worth of expanding a node $n$ is given in the form of an *evaluation function* $f(n)$, which assigns a real number to each node. Mostly, $f(n)$ includes as a component a *heuristic function* $h(n)$, which estimates the costs of the cheapest path from $n$ to the goal.

**Best-First Search:** Informed search procedure that expands the node with the "best" $f$-value first.

# General Algorithm

**function** TREE-SEARCH(*problem*) **returns** a solution, or failure
  initialize the frontier using the initial state of *problem*
  **loop do**
    **if** the frontier is empty **then return** failure
    choose a leaf node and remove it from the frontier
    **if** the node contains a goal state **then return** the corresponding solution
    expand the chosen node, adding the resulting nodes to the frontier

Best-first search is an instance of the general TREE-SEARCH algorithm in which $frontier$ is a priority queue ordered by an evaluation function $f$.

When $f$ is always correct, we do not need to search!

# Greedy Search

A possible way to judge the "worth" of a node is to estimate its path-costs to the goal.

$$h(n) = \textit{estimated path-costs from } n \textit{ to the goal}$$

The only real restriction is that $h(n) = 0$ if $n$ is a goal.

A best-first search using $h(n)$ as the evaluation function, i.e. $f(n) = g(n)$ is called a *greedy search*.

Example: Route-finding problem:
$h(n) =$ straight-line distance from $n$ to the goal

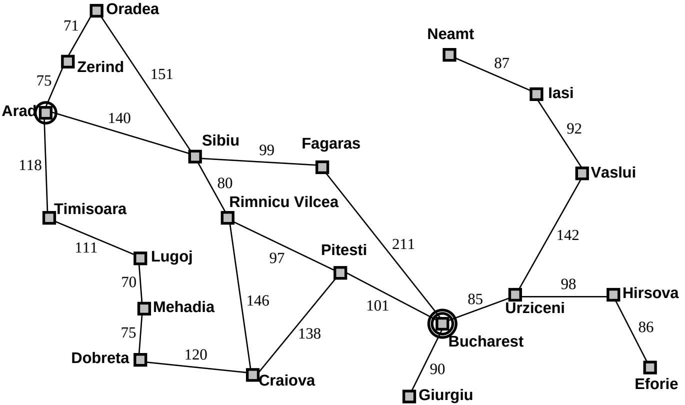
# Heuristics

The evaluation function $h$ in greedy searches is also called a *heuristic* function or simply a *heuristic*.

- The word *heuristic* is derived from the Greek word $\varepsilon\upsilon\rho\iota\sigma\kappa\varepsilon\iota\nu$ (note also: $\varepsilon\upsilon\rho\eta\kappa\alpha$!)
- The mathematician Polya introduced the word in the context of problem solving techniques.
- In AI it has two meanings:
  - Heuristics are fast but in certain situations incomplete methods for problem-solving [Newell, Shaw, Simon 1963] (The greedy search is actually generally incomplete).
  - Heuristics are methods that improve the search in the average-case.

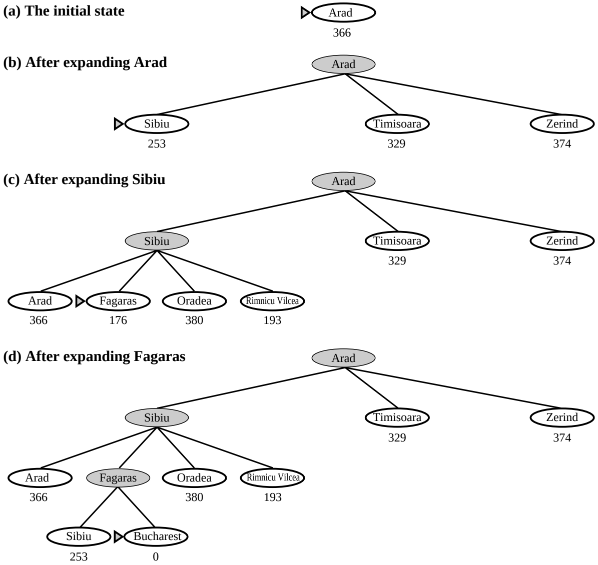
$\rightarrow$ In all cases, the heuristic is *problem-specific* and *focuses* the search!

# Greedy Search Example




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

# Greedy Search from Arad to Bucharest



# Greedy Search - Properties

- a good heuristic might reduce search time drastically
- non-optimal
- incomplete
- graph-search version is complete only in finite spaces

Can we do better?

# A$^*$: Minimization of the Estimated Path Costs

A$^*$ combines the greedy search with the uniform-search strategy:
Always expand node with lowest $f(n)$ first, where

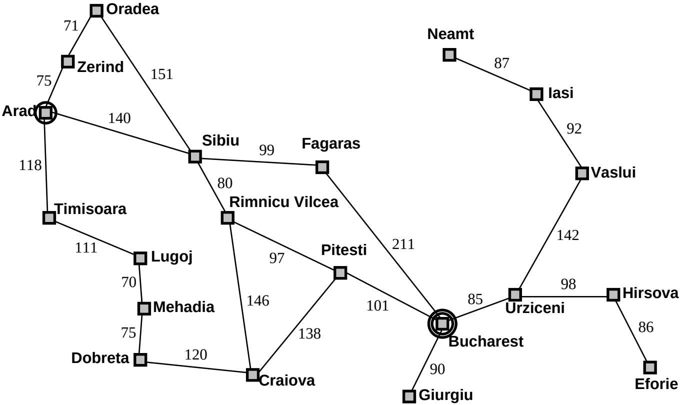
$g(n) =$ actual cost from the initial state to $n$.
$h(n) =$ estimated cost from $n$ to the next goal.
$f(n) = g(n) + h(n)$,
the estimated cost of the cheapest solution through $n$.

Let $h^*(n)$ be the actual cost of the optimal path from $n$ to the next goal.
$h$ is *admissible* if the following holds for all $n$:

$$h(n) \leq h^*(n)$$

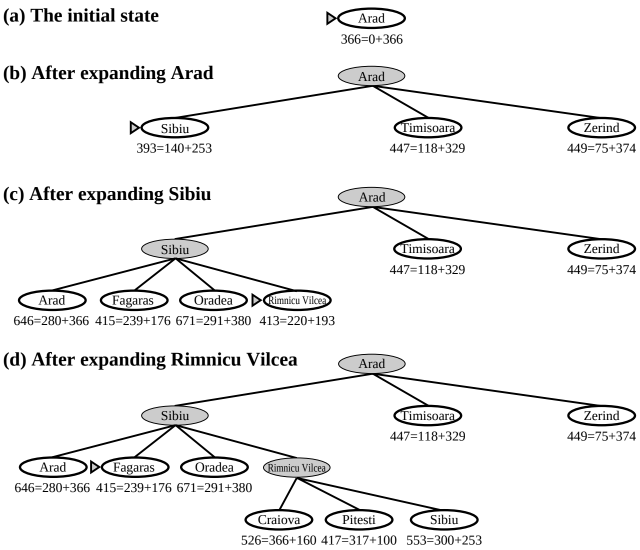
We require that for A$^*$, $h$ is admissible (example: straight-line distance is admissible).
In other words, $h$ is an *optimistic* estimate of the costs that actually occur.

# $A^*$ Search Example

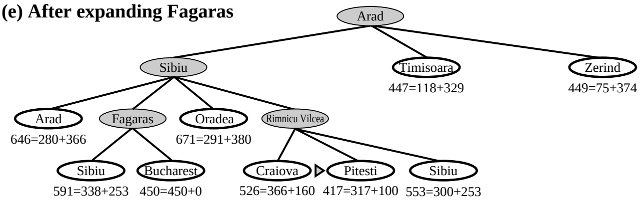


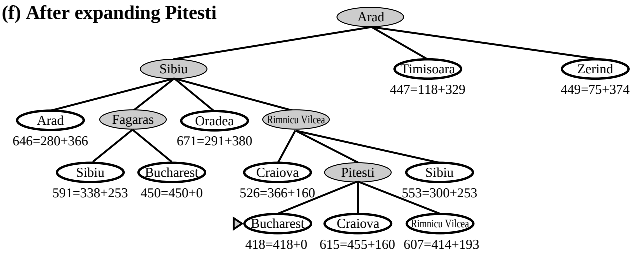

| | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Drobeta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 176 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti**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

# A* Search from Arad to Bucharest

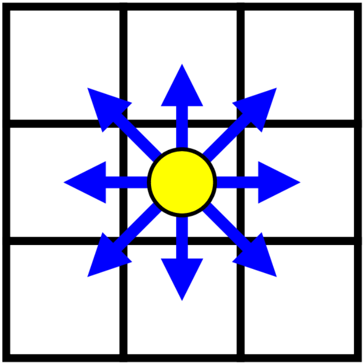


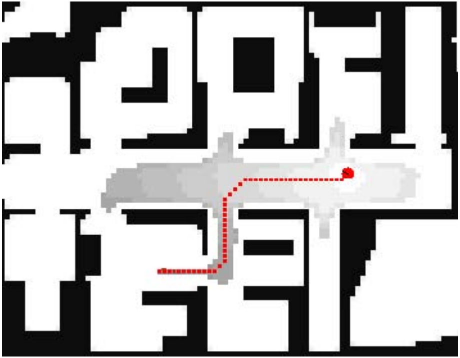
# A* Search from Arad to Bucharest



# Example: Path Planning for Robots in a Grid-World

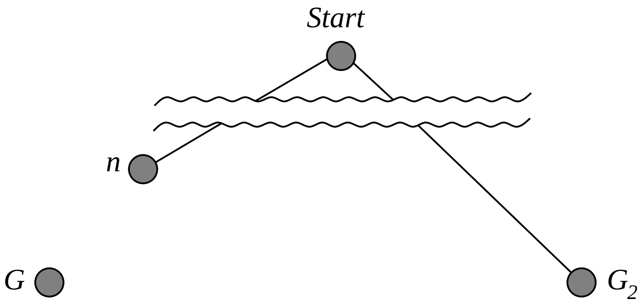




# Optimality of A*

**Claim:** The first solution found has the minimum path cost.

**Proof:** Suppose there exists a goal node $G$ with optimal path cost $f^*$, but A* has found another node $G_2$ with $g(G_2) > f^*$.



# Optimality of A*

Let $n$ be a node on the path from the start to $G$ that has not yet been expanded. Since $h$ is admissible, we have

$$f(n) \leq f^*.$$

Since $n$ was not expanded before $G_2$, the following must hold:

$$f(G_2) \leq f(n)$$

and

$$f(G_2) \leq f^*.$$

It follows from $h(G_2) = 0$ that

$$g(G_2) \leq f^*.$$

$\rightarrow$ Contradicts the assumption!

# Completeness and Complexity

**Completeness:**

If a solution exists, A$^*$ will find it provided that (1) every node has a finite number of successor nodes, and (2) there exists a positive constant $\delta > 0$ such that every step has at least cost $\delta$.

$\rightarrow$ there exists only a finite number of nodes $n$ with $f(n) \leq f^*$.

**Complexity:**

In general, still exponential in the path length of the solution (space, time)

More refined complexity results depend on the assumptions made, e.g. on the quality of the heuristic function. Example:

In the case in which $|h^*(n) - h(n)| \leq O(\log(h^*(n)))$, only one goal state exists, and the search graph is a tree, a sub-exponential number of nodes will be expanded [Gaschnig, 1977, Helmert & Roeger, 2008].

# Heuristic Function Example

| | | |
|---|---|---|
| 7 | 2 | 4 |
| 5 | | 6 |
| 8 | 3 | 1 |

Start State

| | | |
|---|---|---|
| | 1 | 2 |
| 3 | 4 | 5 |
| 6 | 7 | 8 |

Goal State

$h_1$ = the number of tiles in the wrong position

$h_2$ = the sum of the distances of the tiles from their goal positions (*Manhattan distance*)

# Empirical Evaluation

- $d$ = distance from goal
- Average over 100 instances

| | Search Cost (nodes generated) | | | Effective Branching Factor | | |
|---|---|---|---|---|---|---|
| $d$ | IDS | A$^*(h_1)$ | A$^*(h_2)$ | IDS | A$^*(h_1)$ | A$^*(h_2)$ |
| 2 | 10 | 6 | 6 | 2.45 | 1.79 | 1.79 |
| 4 | 112 | 13 | 12 | 2.87 | 1.48 | 1.45 |
| 6 | 680 | 20 | 18 | 2.73 | 1.34 | 1.30 |
| 8 | 6384 | 39 | 25 | 2.80 | 1.33 | 1.24 |
| 10 | 47127 | 93 | 39 | 2.79 | 1.38 | 1.22 |
| 12 | 3644035 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | - | 539 | 113 | - | 1.44 | 1.23 |
| 16 | - | 1301 | 211 | - | 1.45 | 1.25 |
| 18 | - | 3056 | 363 | - | 1.46 | 1.26 |
| 20 | - | 7276 | 676 | - | 1.47 | 1.47 |
| 22 | - | 18094 | 1219 | - | 1.48 | 1.28 |
| 24 | - | 39135 | 1641 | - | 1.48 | 1.26 |

# Variants of A*

A* in general still suffers from exponential memory growth. Therefore, several refinements have been suggested:

- iterative-deepening A*, where the f-costs are used to define the cutoff (rather than the depth of the search tree): **IDA***
- Recursive Best First Search (RBFS): introduces a variable $f\_limit$ to keep track of the best alternative path available from any ancestor of the current node. If current node exceeds this limit, recursion unwinds back to the alternative path.
- other alternatives (not discussed here) memory-bounded A* (MA*), and simplified MA*, SMA*.

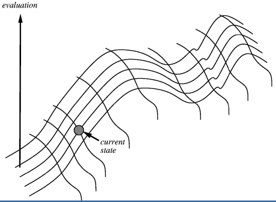
# Recursive Best First Search

```
function RECURSIVE-BEST-FIRST-SEARCH(problem) returns a solution, or failure
   return RBFS(problem, MAKE-NODE(problem.INITIAL-STATE), ∞)

function RBFS(problem, node, f_limit) returns a solution, or failure and a new f-cost limit
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  successors ← [ ]
  for each action in problem.ACTIONS(node.STATE) do
      add CHILD-NODE(problem, node, action) into successors
  if successors is empty then return failure, ∞
  for each s in successors do  /* update f with value from previous search, if any */
      s.f ← max(s.g + s.h, node.f))
  loop do
      best ← the lowest f-value node in successors
      if best.f > f_limit then return failure, best.f
      alternative ← the second-lowest f-value among successors
      result, best.f ← RBFS(problem, best, min(f_limit, alternative))
      if result ≠ failure then return result
```

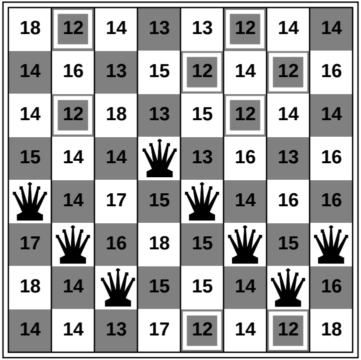
# Local Search Methods

- In many problems, it is unimportant how the goal is reached - only the goal itself matters (8-queens problem, VLSI Layout, TSP).
- If in addition a quality measure for states is given, a **local search** can be used to find solutions.
- operates using a single current node (rather than multiple paths)
- use very little memory
- Idea: Begin with a randomly-chosen configuration and improve on it stepwise → **Hill Climbing**.
- note: can be used for maximisation or minimisation respectively (see 8 queens example)



# Example: 8-Queens Problem (1)

Example state with heuristic cost estimate $h = 17$ (counts the number of pairs threatening each other directly or indirectly).

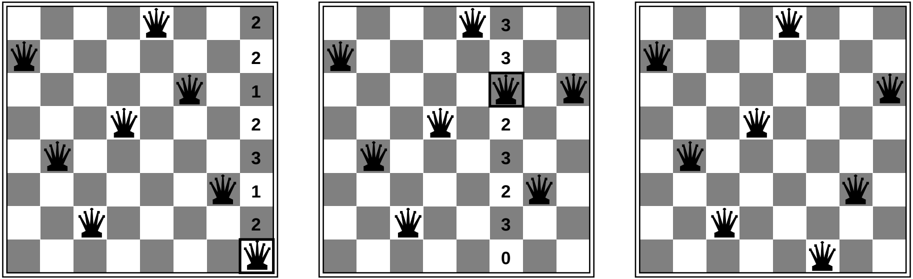


# Hill Climbing

```
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
```

Possible realisation of a hill-climbing algorithm:
Select a column and move the queen to the square with the fewest conflicts.



# Problems with Local Search Methods

- *Local maxima:* The algorithm finds a sub-optimal solution.
- *Plateaus:* Here, the algorithm can only explore at random.
- Ridges: Similar to plateaus.

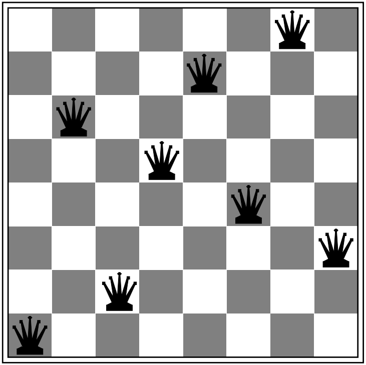
**Solutions:**

- *Start over* when no progress is being made.
- "Inject noise" $\rightarrow$ random walk

Which strategies (with which parameters) are successful (within a problem class) can usually only empirically be determined.

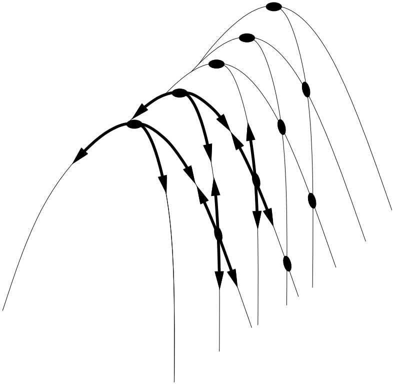
# Example: 8-Queens Problem (Local Minimum)

Local minimum ($h = 1$) of the 8-Queens Problem. Every successor has a higher cost.



# Illustration of the ridge problem

The grid of states (dark circles) is superimposed on a ridge rising from left to right, creating a sequence of local maxima, that are not directly connected to each other. From each local maximum, all the available actions point downhill.



# Performance figures for the 8-Queens Problem

8 queens has about $8^8 \approx 17\ million$ states. Starting from a random initialisation, hill-climbing directly finds a solution in about $14\%$ of the cases. Needs in average only 4 steps!

Better algorithm: allow sideway moves (no improvement), but restrict the number of moves (avoid infinite loops!).

E.g: max. 100 moves: solves $94\%$, number of steps raises to 21 steps for successful instances and 64 for each failure.

# Simulated Annealing

In the simulated annealing algorithm, "noise" is injected systematically: first a lot, then gradually less.

**function** SIMULATED-ANNEALING($problem$, $schedule$) **returns** a solution state
  **inputs**: $problem$, a problem
    $schedule$, a mapping from time to "temperature"

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

Has been used since the early 80's for VSLI layout and other optimization problems.

# Genetic Algorithms

Evolution appears to be very successful at finding good solutions.

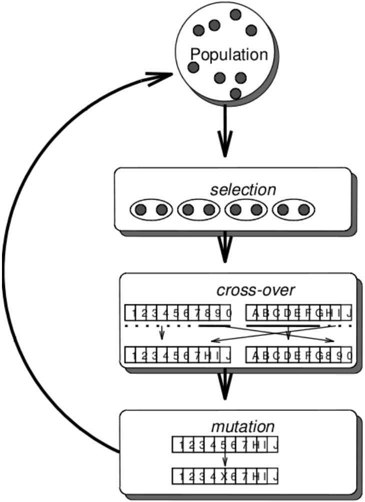
*Idea:* Similar to evolution, we search for solutions by "crossing", "mutating", and "selecting" successful solutions.

*Ingredients:*

- Coding of a solution into a string of symbols or bit-string
- A fitness function to judge the worth of configurations
- A population of configurations

*Example:* 8-queens problem as a chain of 8 numbers. Fitness is judged by the number of non-attacks. The population consists of a set of arrangements of queens.

# Selection, Mutation, and Crossing




Many variations:

how selection will be applied, what type of cross-over operators will be used, etc.

Selection of individuals according to a fitness function and pairing

Calculation of the breaking points and recombination

According to a given probability elements in the string are modified.

# Summary

- Heuristics focus the search
- Best-first search expands the node with the highest worth (defined by any measure) first.
- With the minimization of the evaluated costs to the goal $h$ we obtain a greedy search.
- The minimization of $f(n) = g(n) + h(n)$ combines uniform and greedy searches. When $h(n)$ is admissible, i.e., $h^*$ is never overestimated, we obtain the A* search, which is complete and optimal.
- IDA* is a combination of the iterative-deepening and A* searches.
- Local search methods only ever work on one state, attempting to improve it step-wise.
- Genetic algorithms imitate evolution by combining good solutions.