# Foundations of AI

# 4. Informed Search Methods

## Heuristics, Local Search Methods, Genetic Algorithms

***Wolfram Burgard and Bernhard Nebel***

# Contents

- Best-First Search
- A* and IDA*
- Local Search Methods
- Genetic Algorithms

# 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 cost of a given node to the goal is in the form of an *evaluation function* *f* or *h*, which assigns a real number to each node.

**Best-First Search:** Search procedure that expands the node with the "best" *f*- or *h*-value.

# General Algorithm

```
function BEST-FIRST-SEARCH(problem, EVAL-FN) returns a solution sequence
  inputs: problem, a problem
          Eval-Fn, an evaluation function

  Queueing-Fn ← a function that orders nodes by EVAL-FN
  return GENERAL-SEARCH(problem, Queueing-Fn)
```

When *h* 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 distance to the goal.

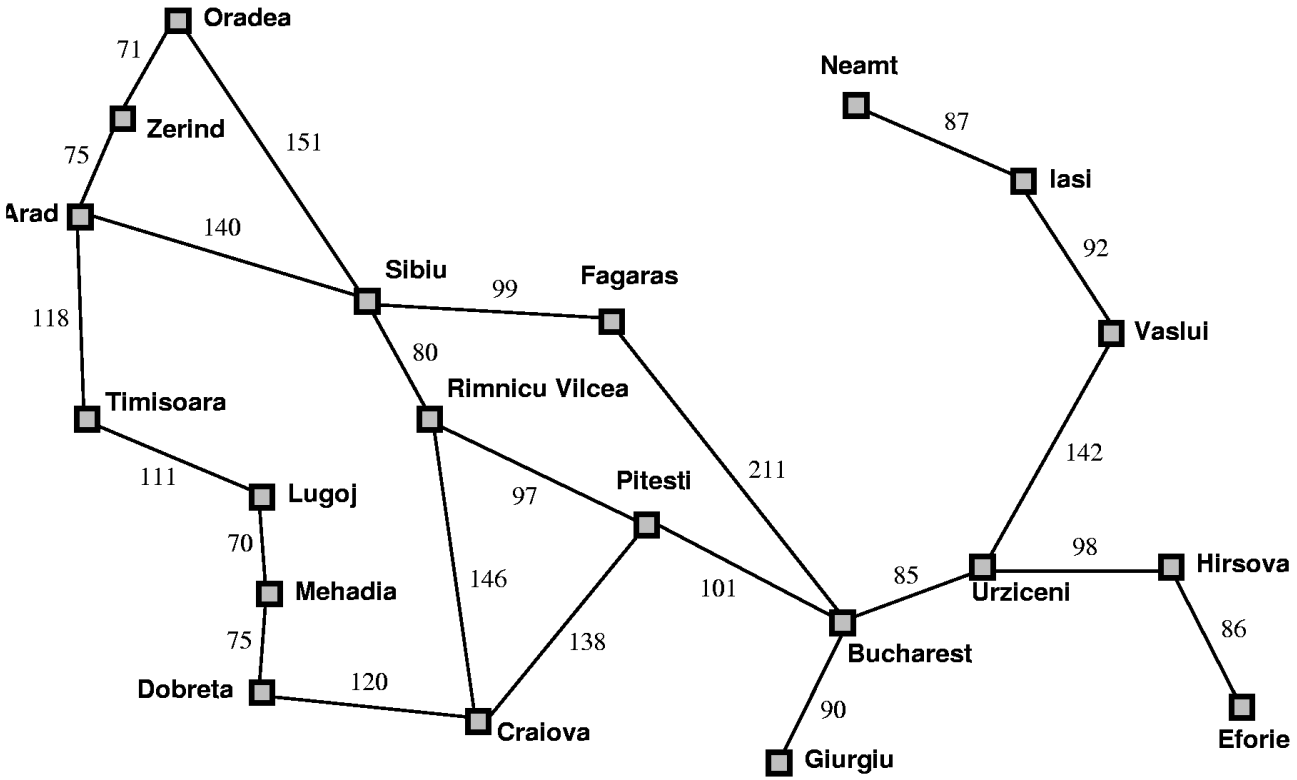
*h(n) = estimated distance from n to the goal*

The only real condition is that *h(n) = 0* if *n* is a goal.

A best-first search with this function is called a *greedy search.*

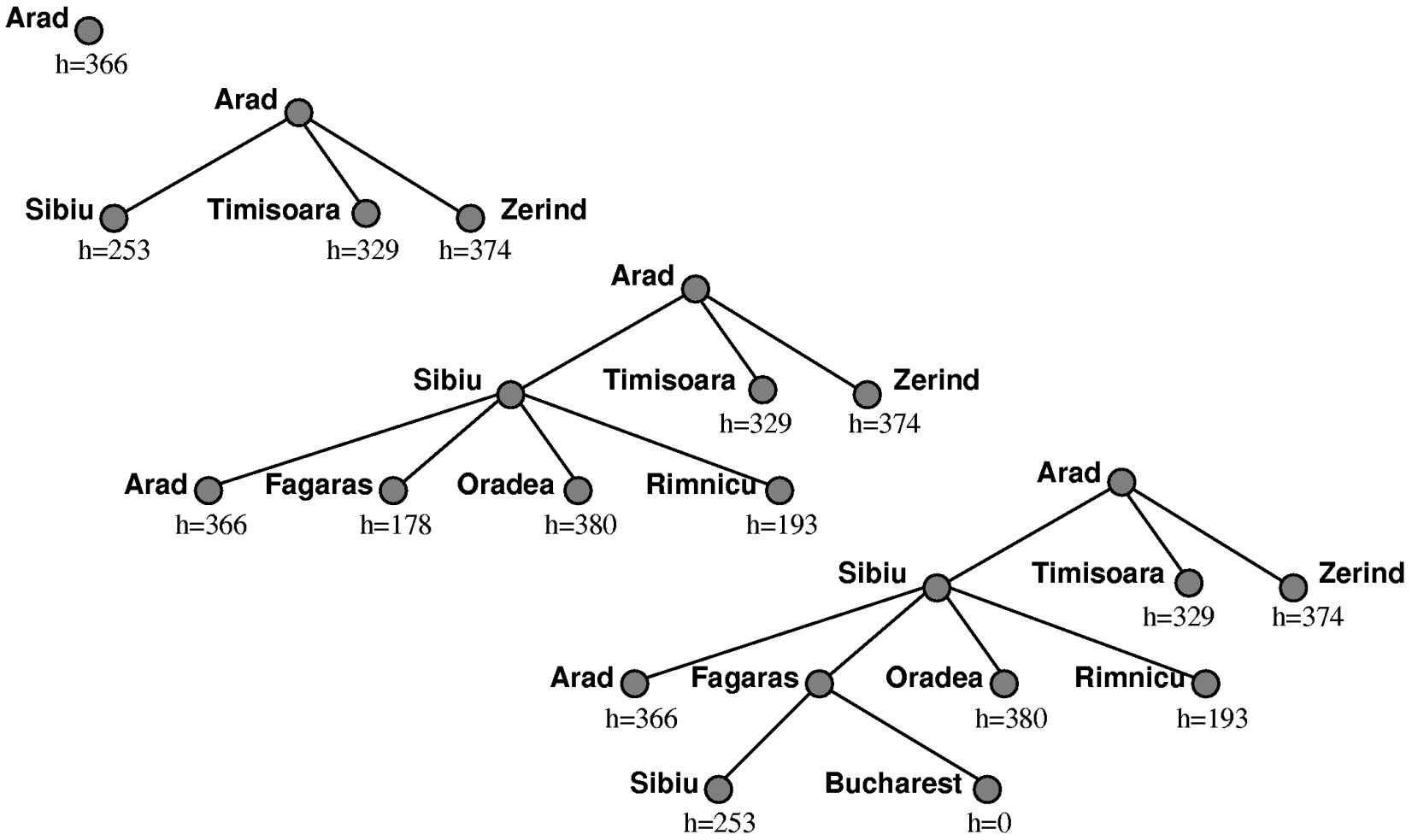
Route-finding problem: *h* = straight-line distance between two locations.

# Greedy Search Example




| Straight–line distance to Bucharest | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# Greedy Search from *Arad* to *Bucharest*



# 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 ευρισκειν (note also: ευρηκα!)
- 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.

→ In all cases, the heuristic is *problem-specific* and *focuses* the search!

# A*: Minimization of the estimated path costs

A* combines the greedy search with the uniform-search strategy.

*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*.

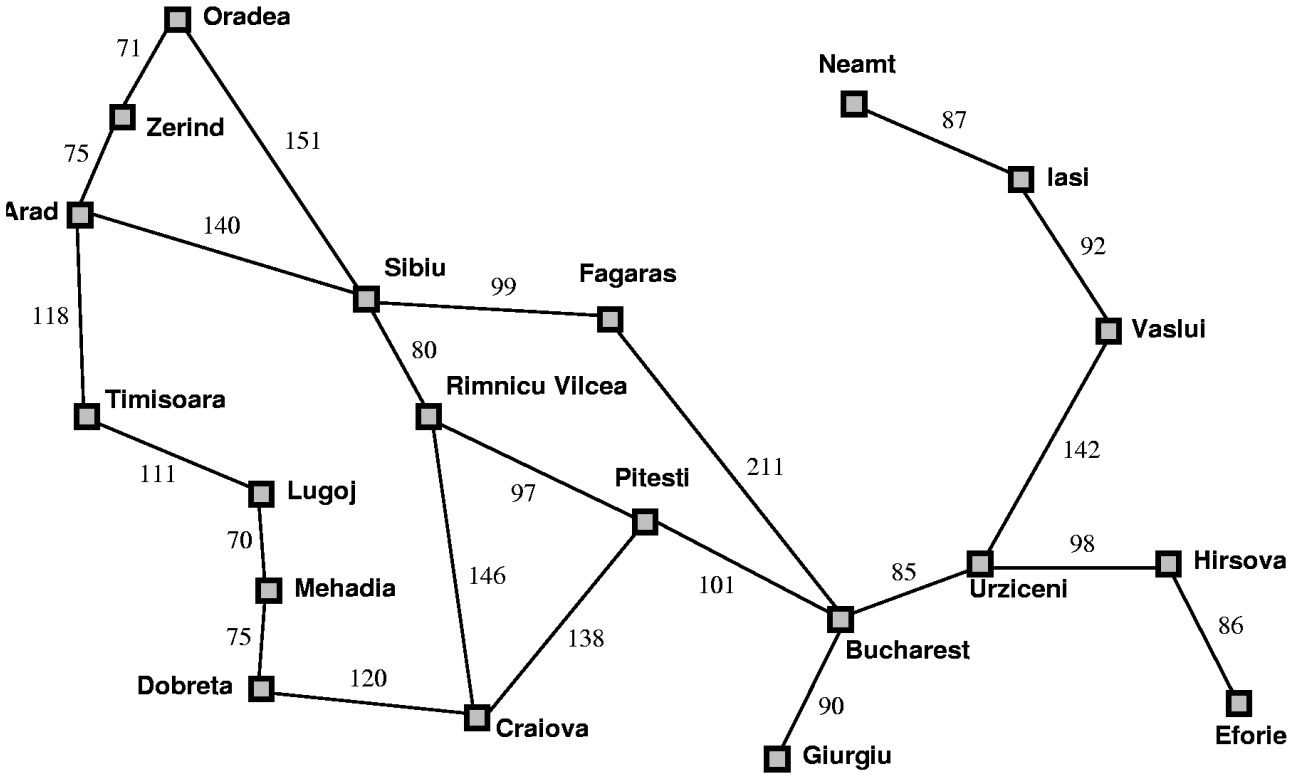
Let h*(n) be the actual cost of the optimal path from *n* to the next goal.

*h* is *admissible* if the following holds for all *n* :

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

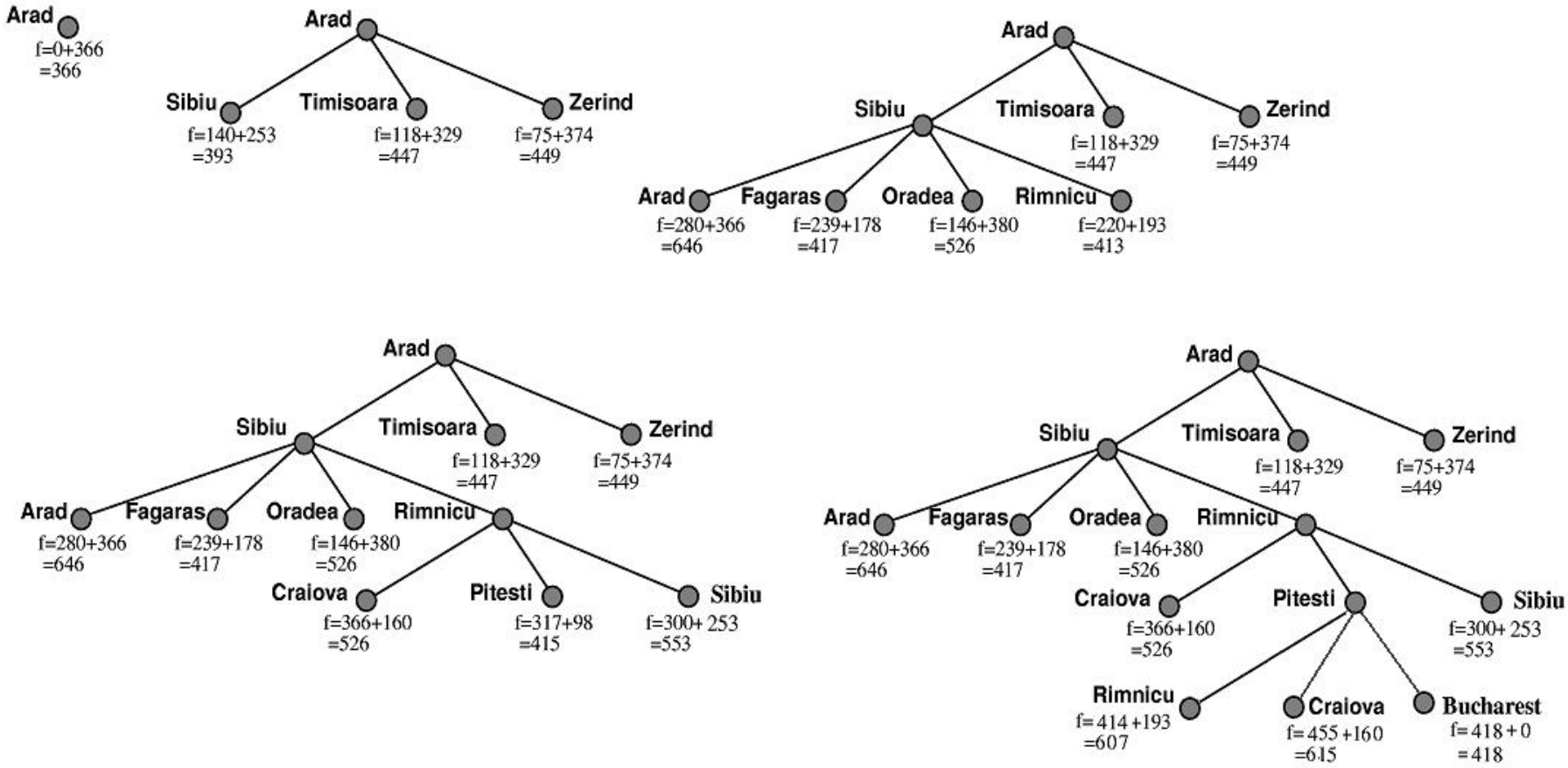
We require that for A*, *h* is admissible (straight-line distance is admissible).

# A* Search Example



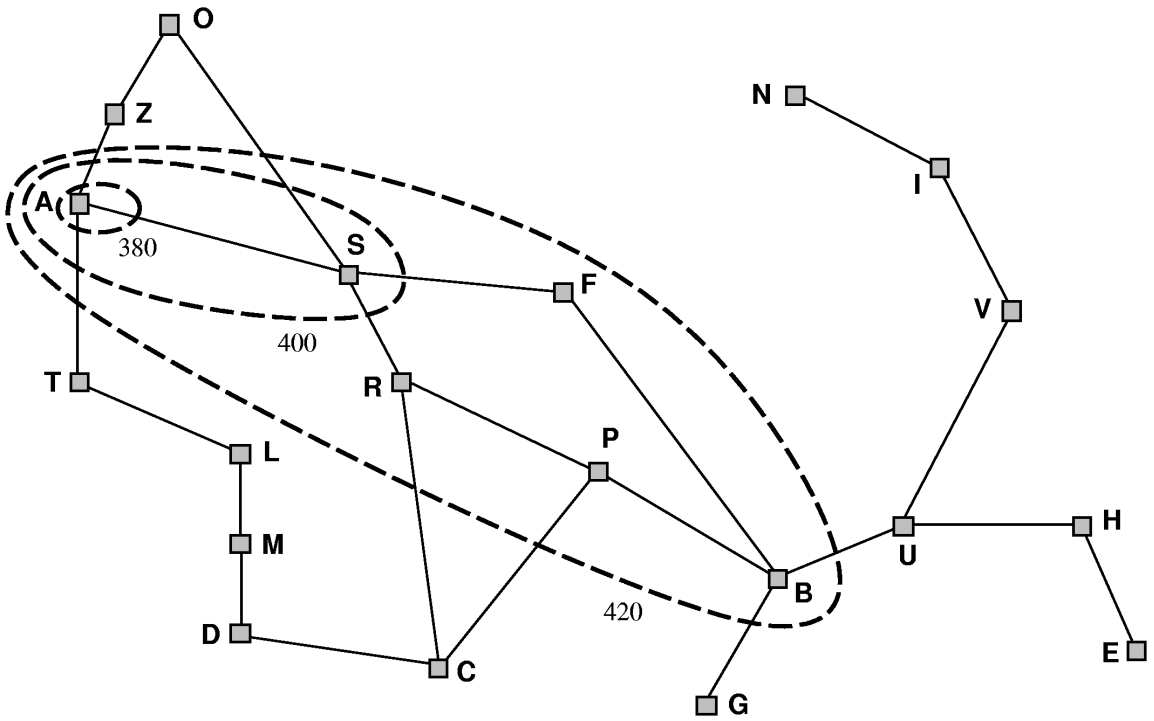

| Straight–line distance to Bucharest | |
|---|---|
| **Arad** | 366 |
| **Bucharest** | 0 |
| **Craiova** | 160 |
| **Dobreta** | 242 |
| **Eforie** | 161 |
| **Fagaras** | 178 |
| **Giurgiu** | 77 |
| **Hirsova** | 151 |
| **Iasi** | 226 |
| **Lugoj** | 244 |
| **Mehadia** | 241 |
| **Neamt** | 234 |
| **Oradea** | 380 |
| **Pitesti** | 98 |
| **Rimnicu Vilcea** | 193 |
| **Sibiu** | 253 |
| **Timisoara** | 329 |
| **Urziceni** | 80 |
| **Vaslui** | 199 |
| **Zerind** | 374 |

# A* Search from *Arad* to *Bucharest*



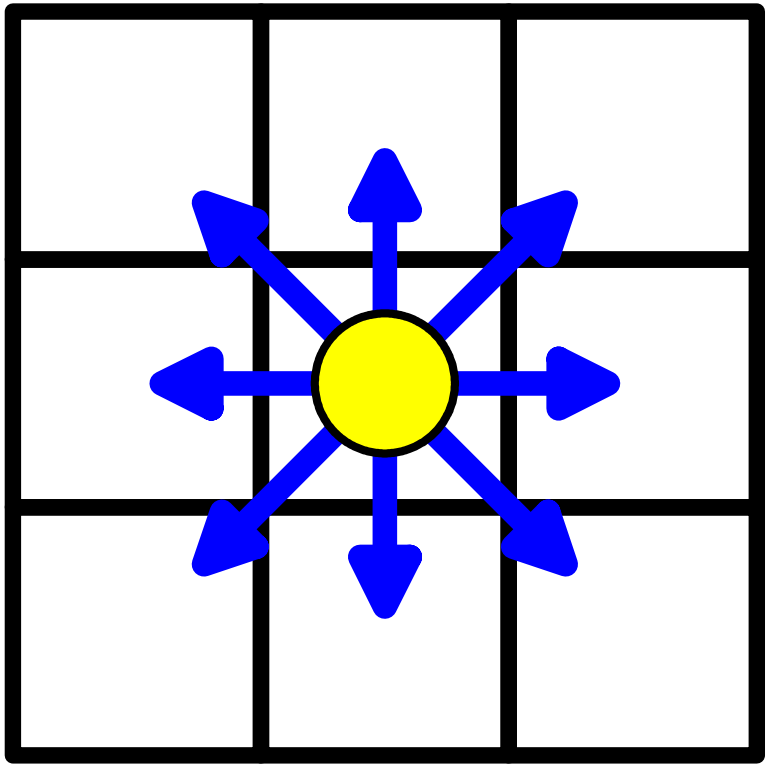
# Contours in A*

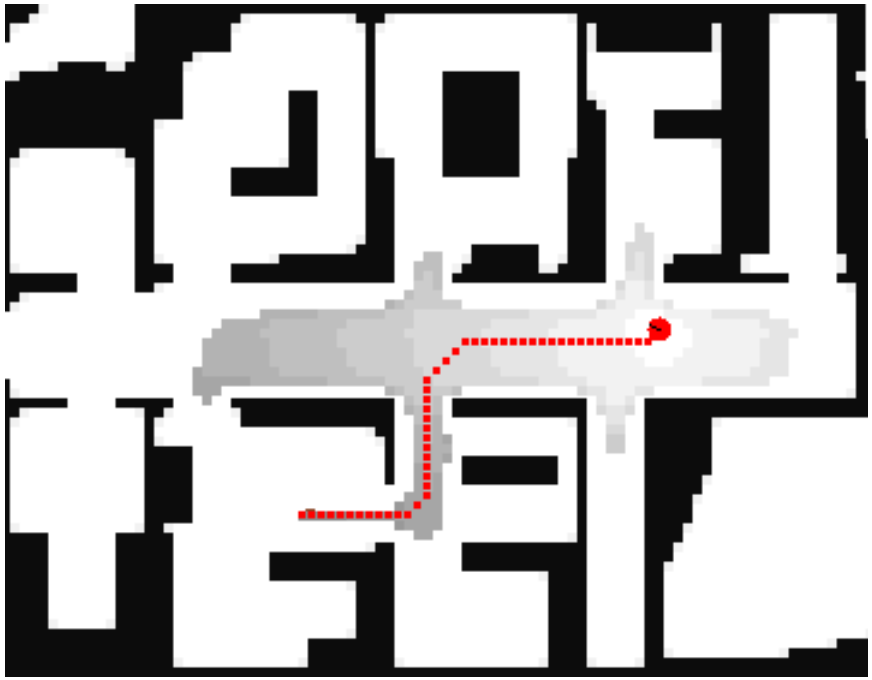
Within the search space, contours arise in which for the given *f*-value all nodes are expanded.




Contours at f = 380, 400, 420

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

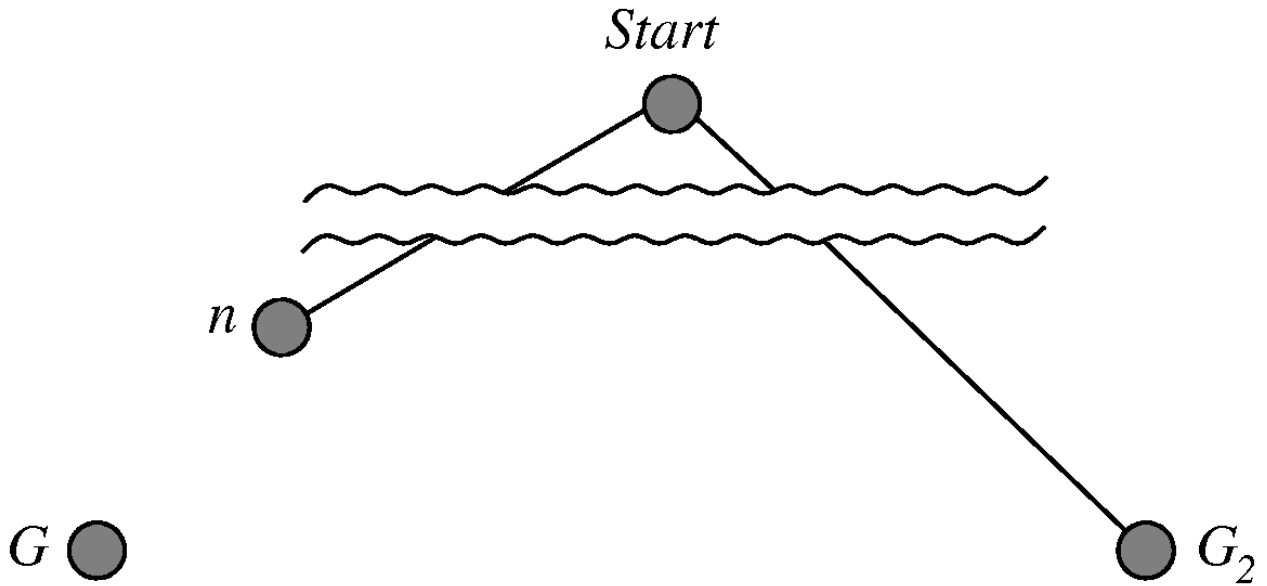




# Optimality of A*

**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 $g(G_2) > f^*$.



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*

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

→ 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 $\delta$ such that every operator has at least cost $\delta$.

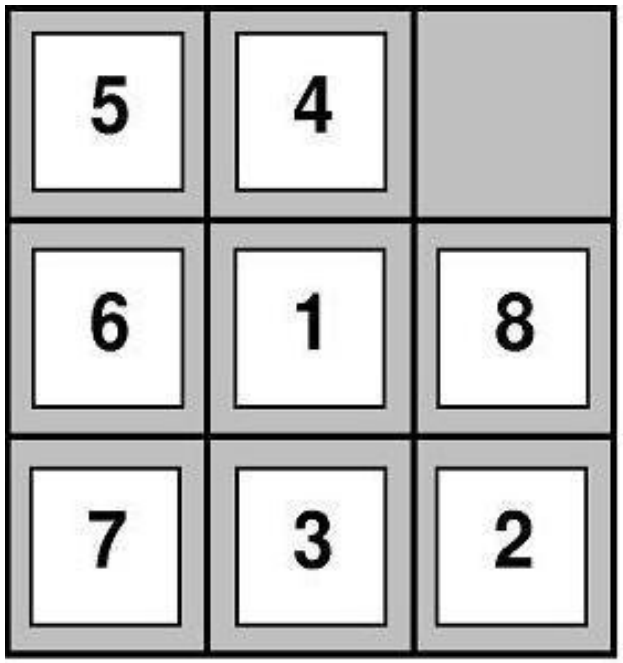
→ Only a finite number of nodes $n$ with $f(n) \leq f^*$.

## Complexity:

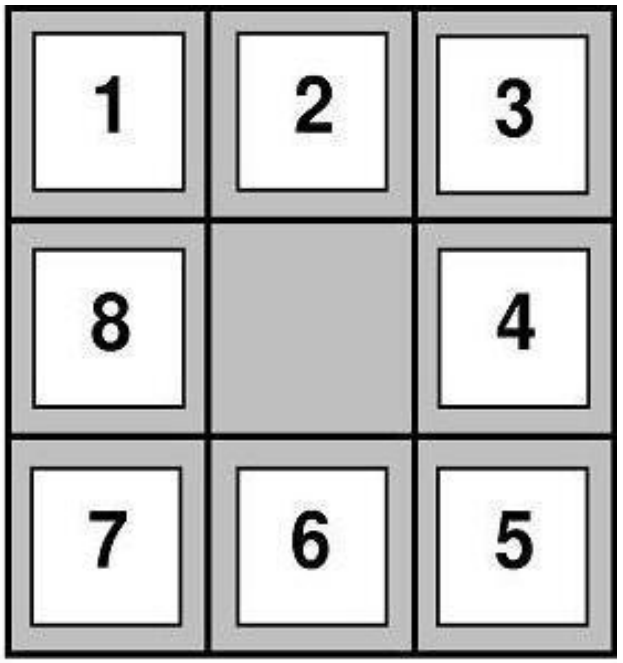
In the case where $|h^*(n) - h(n)| \leq O(\log(h^*(n))$, only a sub-exponential number of nodes will be expanded.

Normally, growth is exponential because the error is proportional to the path costs.

# Heuristic Function Example




Start State




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 (*Manhatten distance)*

# Empirical Evaluation

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

| | Search Cost | | | 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 | 364404 | 227 | 73 | 2.78 | 1.42 | 1.24 |
| 14 | 3473941 | 539 | 113 | 2.83 | 1.44 | 1.23 |
| 16 | – | 1301 | 211 | – | 1.45 | 1.25 |
| 18 | – | 3056 | 363 | – | 1.46 | 1.26 |
| 20 | – | 7276 | 676 | – | 1.47 | 1.27 |
| 22 | – | 18094 | 1219 | – | 1.48 | 1.28 |
| 24 | – | 39135 | 1641 | – | 1.48 | 1.26 |

# Iterative Deepening A* Search (IDA*)

Idea: A combination of IDS and A*. All nodes inside a contour are searched.

```
function IDA*(problem) returns a solution sequence
  inputs: problem, a problem
  static: f-limit, the current f- COST limit
          root, a node

  root ← MAKE-NODE(INITIAL-STATE[problem])
  f-limit ← f- COST(root)
  loop do
      solution, f-limit ← DFS-CONTOUR(root, f-limit)
      if solution is non-null then return solution
      if f-limit = ∞ then return failure; end
```

---

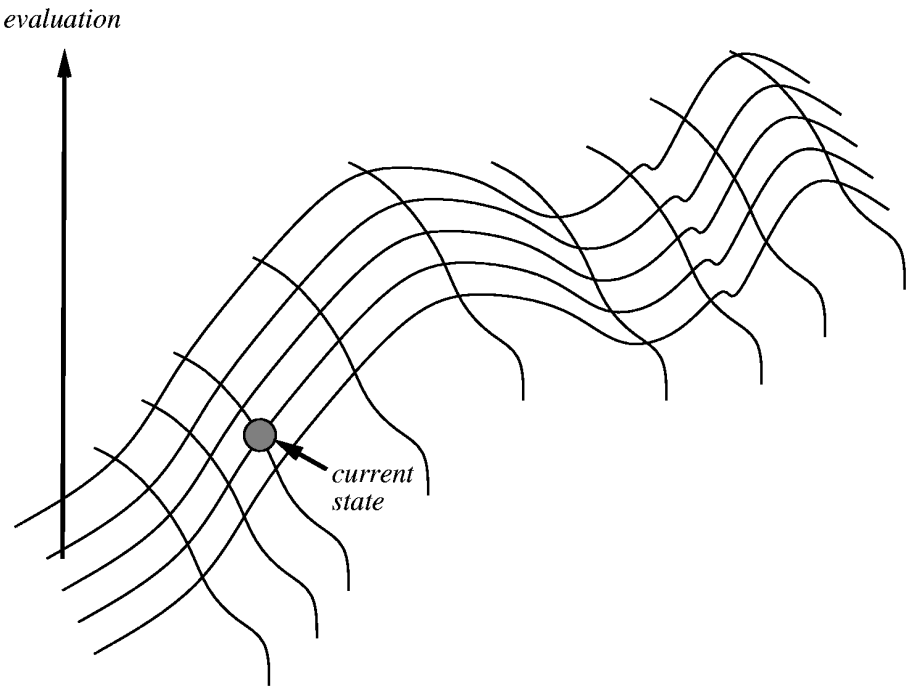
```
function DFS-CONTOUR(node, f-limit) returns a solution sequence and a new f- COST limit
  inputs: node, a node
          f-limit, the current f- COST limit
  static: next-f, the f- COST limit for the next contour, initially ∞

  if f- COST[node] > f-limit then return null, f- COST[node]
  if GOAL-TEST[problem](STATE[node]) then return node, f-limit
  for each node s in SUCCESSORS(node) do
      solution, new-f ← DFS-CONTOUR(s, f-limit)
      if solution is non-null then return solution, f-limit
      next-f ← MIN(next-f, new-f); end
  return null, next-f
```

# 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.

Idea: Begin with a randomly-chosen configuration and improve on it stepwise → **Hill Climbing**.

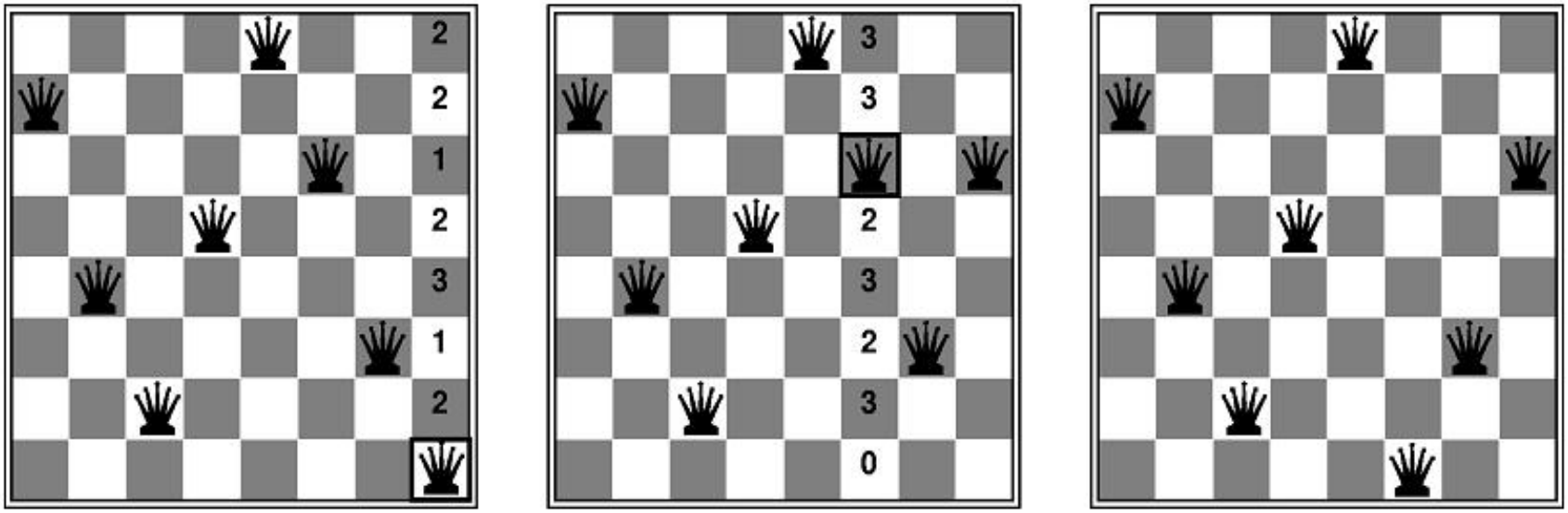


# Hill Climbing

```
function HILL-CLIMBING(problem) returns a solution state
   inputs: problem, a problem
   static: current, a node
           next, a node

   current ← MAKE-NODE(INITIAL-STATE[problem])
   loop do
      next ← a highest-valued successor of current
      if VALUE[next] < VALUE[current] then return current
      current ← next
   end
```

# Example: 8-Queens Problem

Selects a column and moves 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 smoke" → random walk
- Tabu search: Do not apply the last *n* operators.

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

# Simulated Annealing

In the simulated annealing algorithm, "smoke" 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"
  static: 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
      ΔE ← VALUE[next] – VALUE[current]
      if ΔE > 0 then current ← next
      else current ← next only with probability e^(ΔE/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.

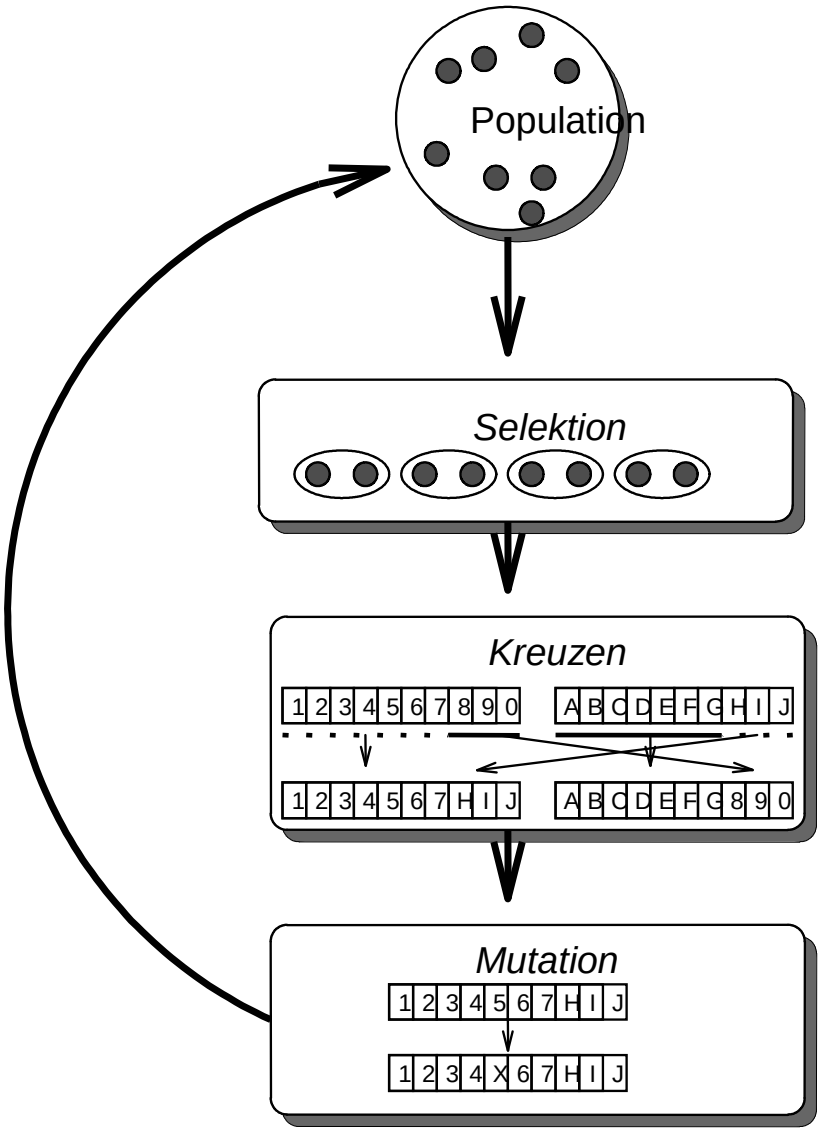
*Idea*: Similar to evolution, we search for solutions by "crossing", "mutating", and "selecting" successful solutions.

*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-overs will be used, etc.

Selektion von Individuen anhand der Fitness-Funktion und Paarung

Festlegung wo aufgebrochen wird und neu zusammenfügen

Mit einer gewissen kleinen Wahrscheinlichkeit wird etwas im String geändert.

# 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.