

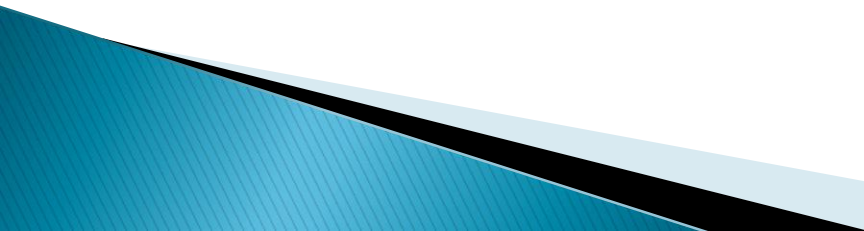
# Artificial Intelligence

## Informed search

More about

Textbook, Chapter 4, Informed Search and Exploration

# Outline

- To be informed = to use problem-specific knowledge about the direction to the goal
  - New search strategies?
    - Best-first search and its variants (Best of what?)
  - Direction to the goal = heuristic functions
    - What they are and how to invent them?
  - Optimization with a local search  
(No memory, time problems! Solutions?)
    - Hill climbing, local beam search, genetic algorithms,...
- 

# What do we have until now?

- **Completeness** (i.e. sure solution) can be coupled with linear space complexity (ID), but not with linear time complexity.
- Time complexity is **exponential** at best (exponent can be halved in bidirectional search, but under conditions).
- **Optimal cost paths** (the shortest, the cheapest, ...) are computable, if:
  - we introduce step cost  $c(n)$  , and
  - are patient enough to wait for the solution (Uniform Cost, aka Dijkstra)
  - then a path with  
can be found

$$\text{Path cost} = g(\text{goal}) = \sum_{n=\text{start}}^{\text{goal}} c(n) = \min$$

Price to pay?

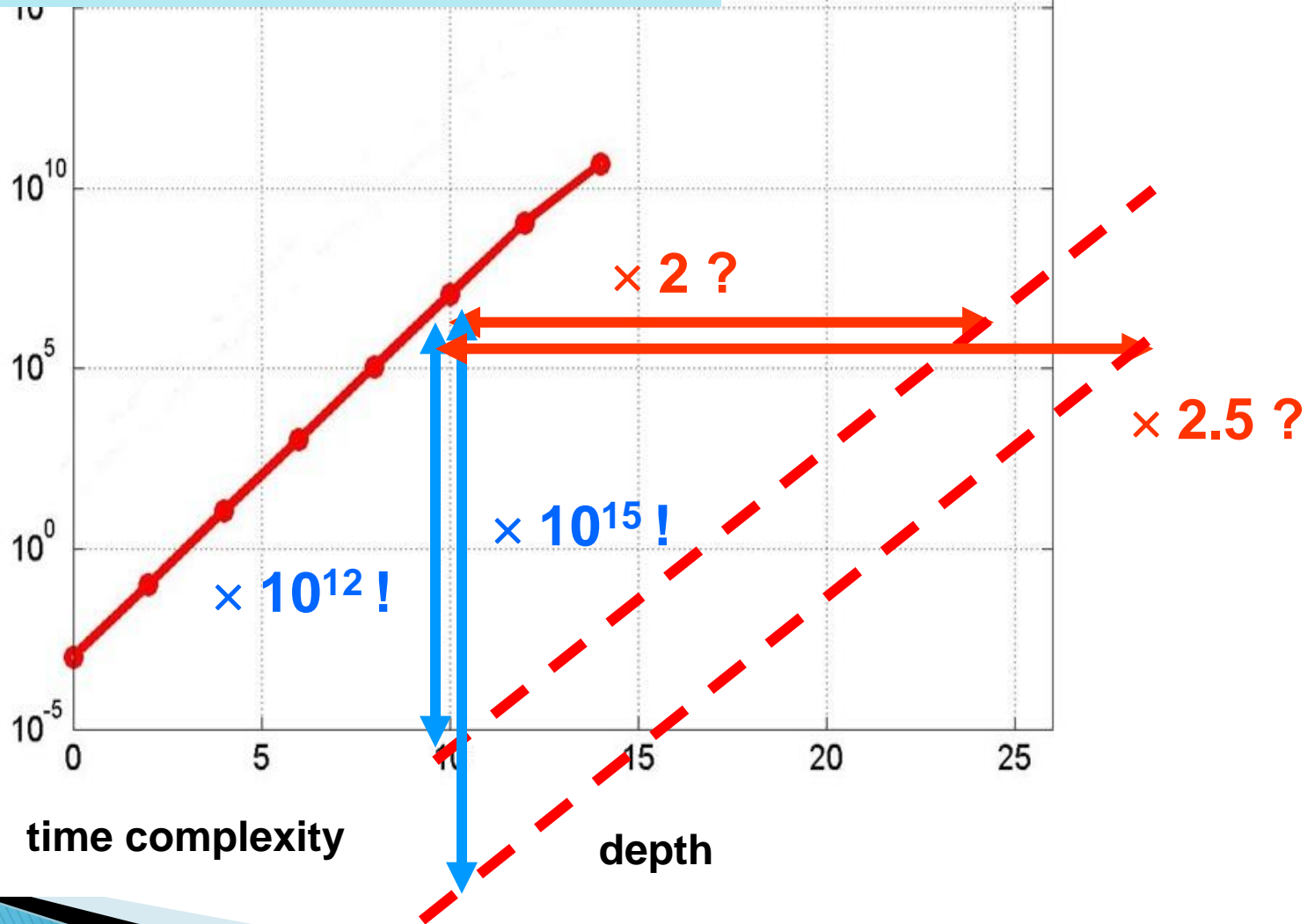
# Supercomputers: from 2000 to 2018 years

IBM Summit (2018)

Oak Ridge Lab, 1223 Pflop

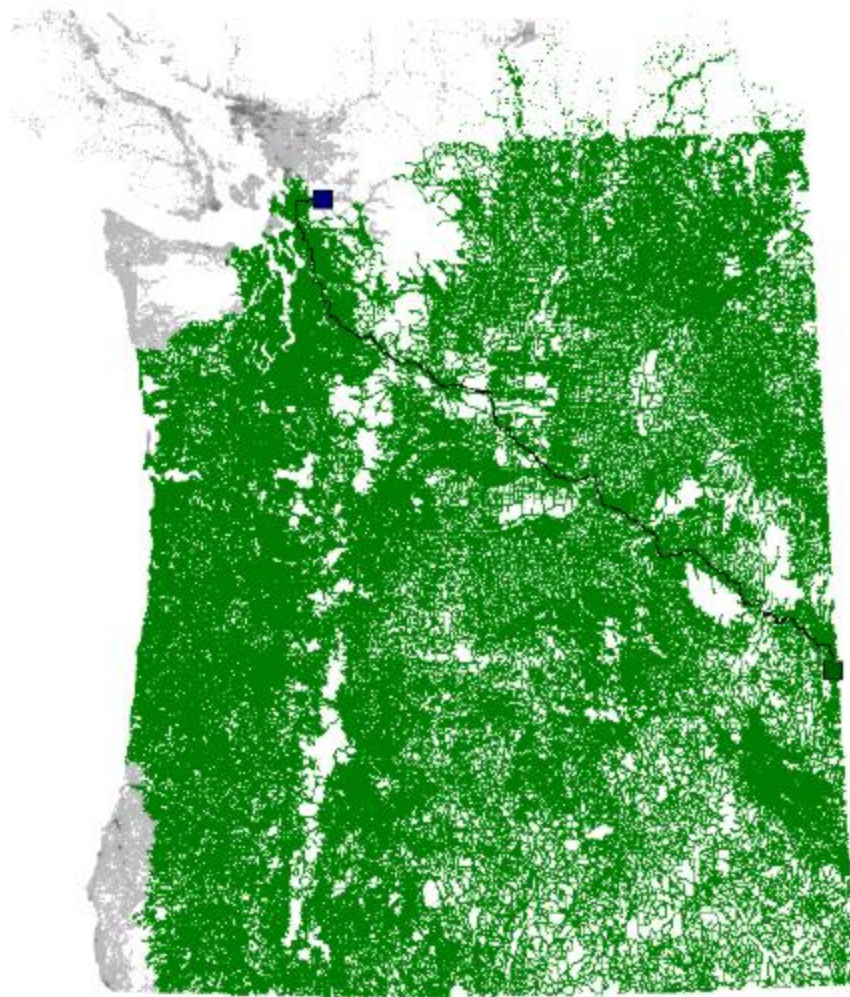
13 MW

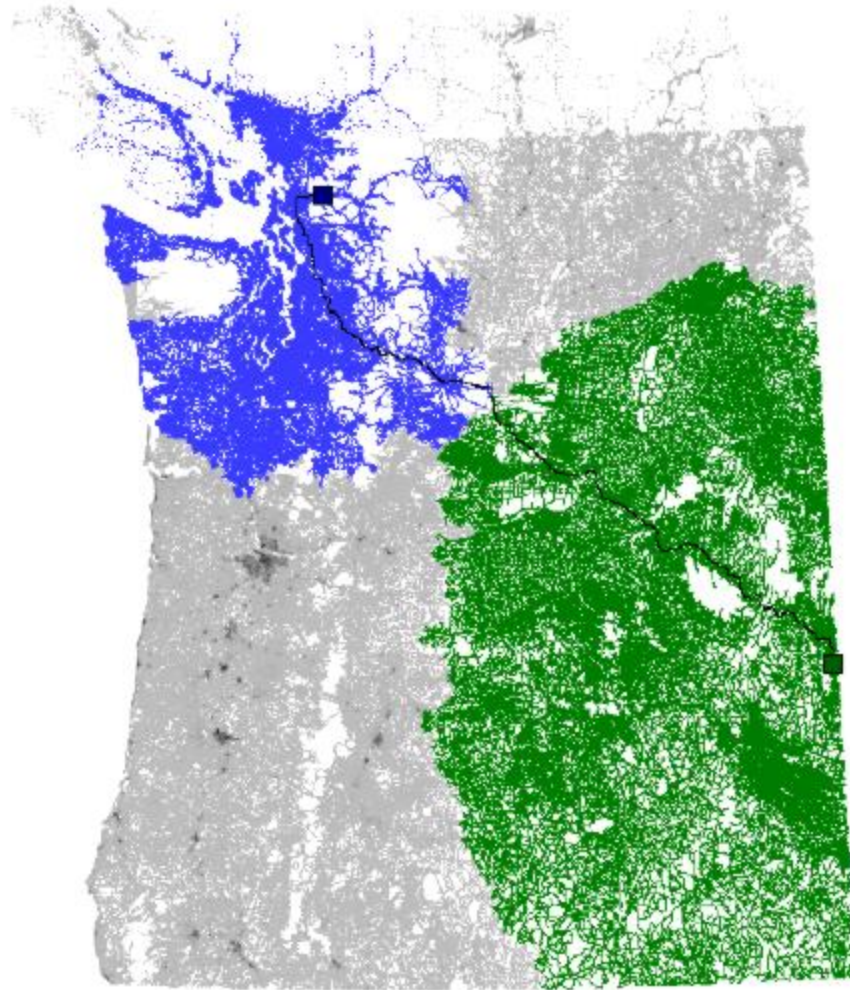
Curse of noninformed search  
i.e. exponential time complexity

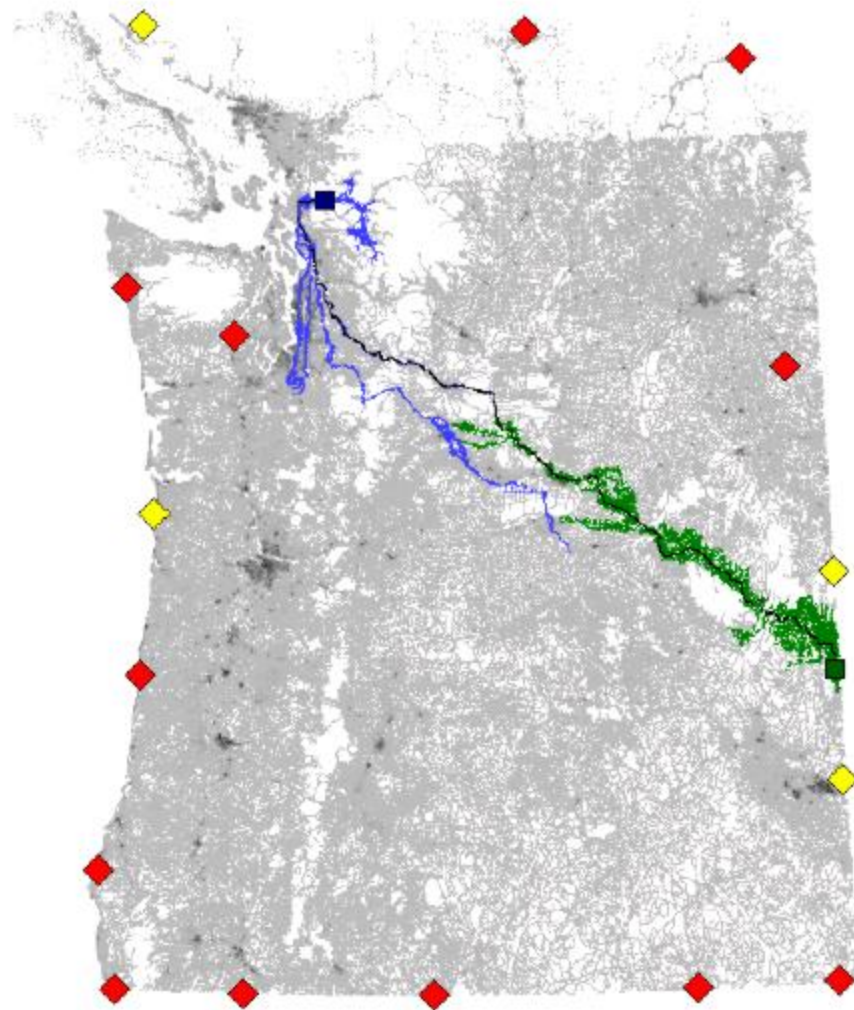


1.6M vertices  
3.8M edges  
(US Northwest  
road network)

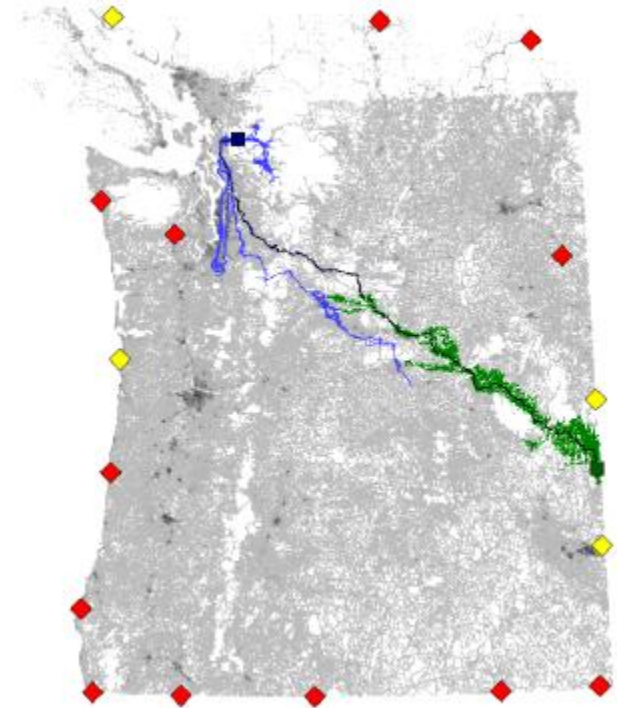
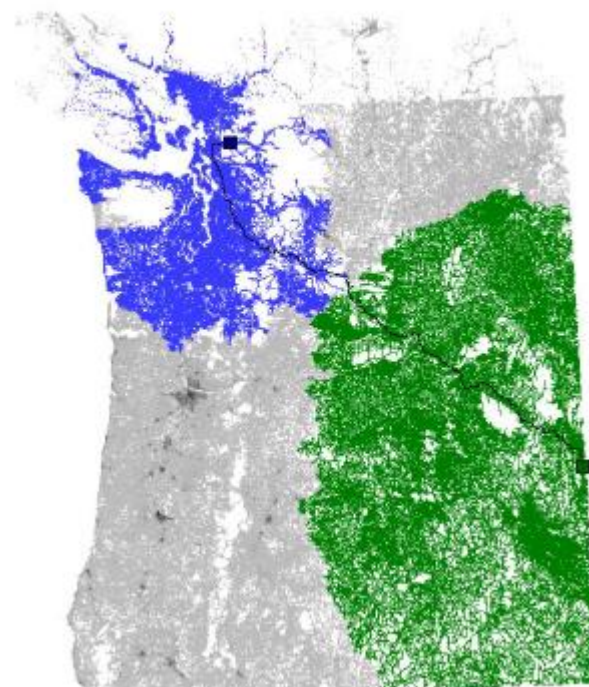












1.6M vertices  
3.8M edges  
(US Northwest)

USA, travel times, random pairs

1.6M vertices  
3.8M edges

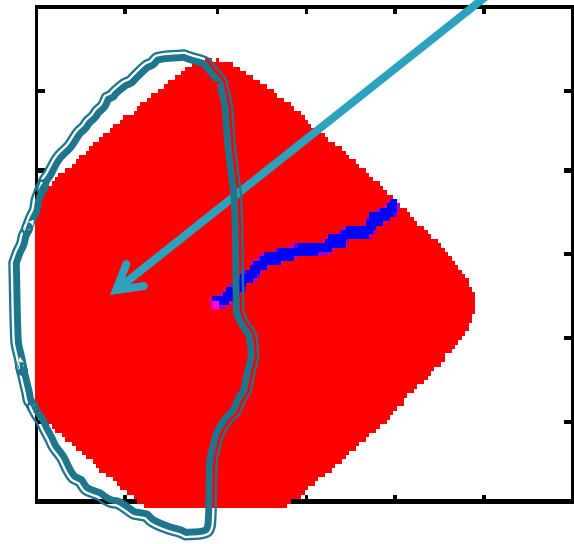
Full USA network  
24M vertices, 58M edges

METHOD	PREPROCESSING		QUERY	
	minutes	MB	scans	ms
Dijkstra	—	536	11 808 864	5440.49
ALT(16)	18	2563	187 968	295.44
RE	28	893	2 405	1.77
REAL(16)	45	3032	592	0.80
REAL(64,16)	114	1579	538	0.86

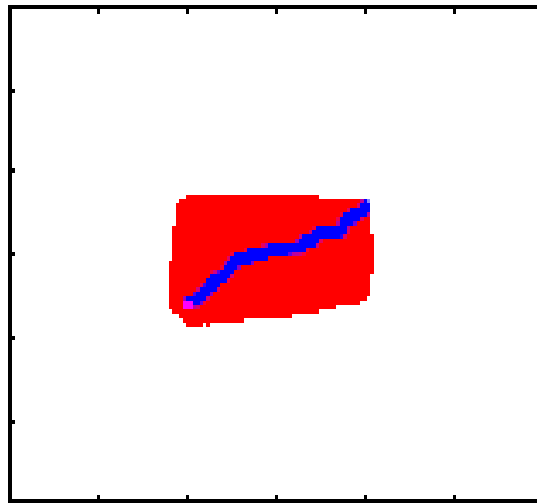
22000 x 6000 x

# The essence of the problem

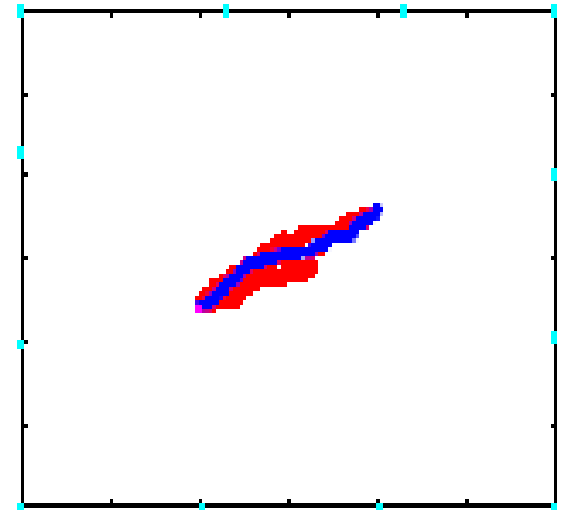
We spend time on searching also *AWAY* from the goal!  
Then we must stop, change direction and backtrack!



Uniform cost  
(Dijkstra)

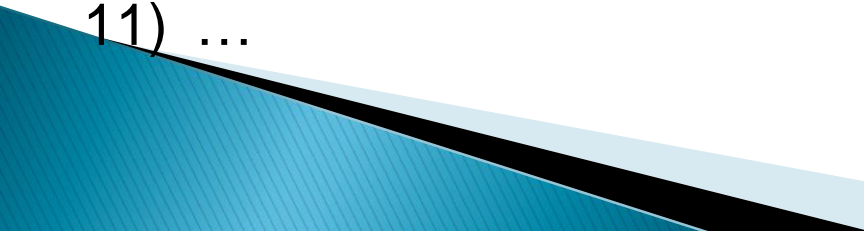


Another idea



Yet another idea

# Ideas

- 1) Penalty for backtracking. ( $A^*$ , etc.)
  - 2) Search from both directions. (bidirectional)
  - 3) Smart usage of depth-first search. (iterative)
  - 4) Pre-process the graph. (ALT landmarks)
  - 5) Fully giving up backtracking. (hill-climbing)
  - 6) Use lower quality directions also. (beam search)
  - 7) Permit bad directions also. (simulated annealing)
  - 8) Prohibit probably bad directions. (tabu search)
  - 9) Cont. improve based on partial results (learning  $A^*$ , anytime  $A^*$ , ...)
  - 10) Randomize and repeat.
  - 11) ...
- 

# Penalty for backtracking!

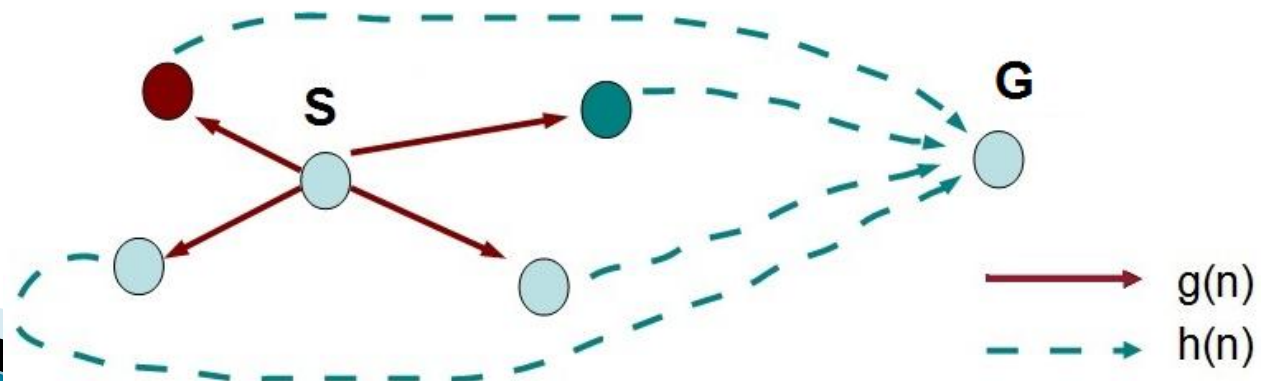
= i.e. awarding going forward, **toward the goal**.

What do we need? (1) In what „direction” is the goal expected?  
(2) How „far” is the goal expected in this direction?.

This information is called **heuristics**, **heuristic function  $h(n)$** :

- Must be computable in every state in the search space
- Must estimate the expected cost of going in a given direction
- If exact, no search needed (if very uncertain, no help at all)
- At the goal should be:  $h(goal) = 0$

Search using heuristics is a **heuristic (informed) search**.

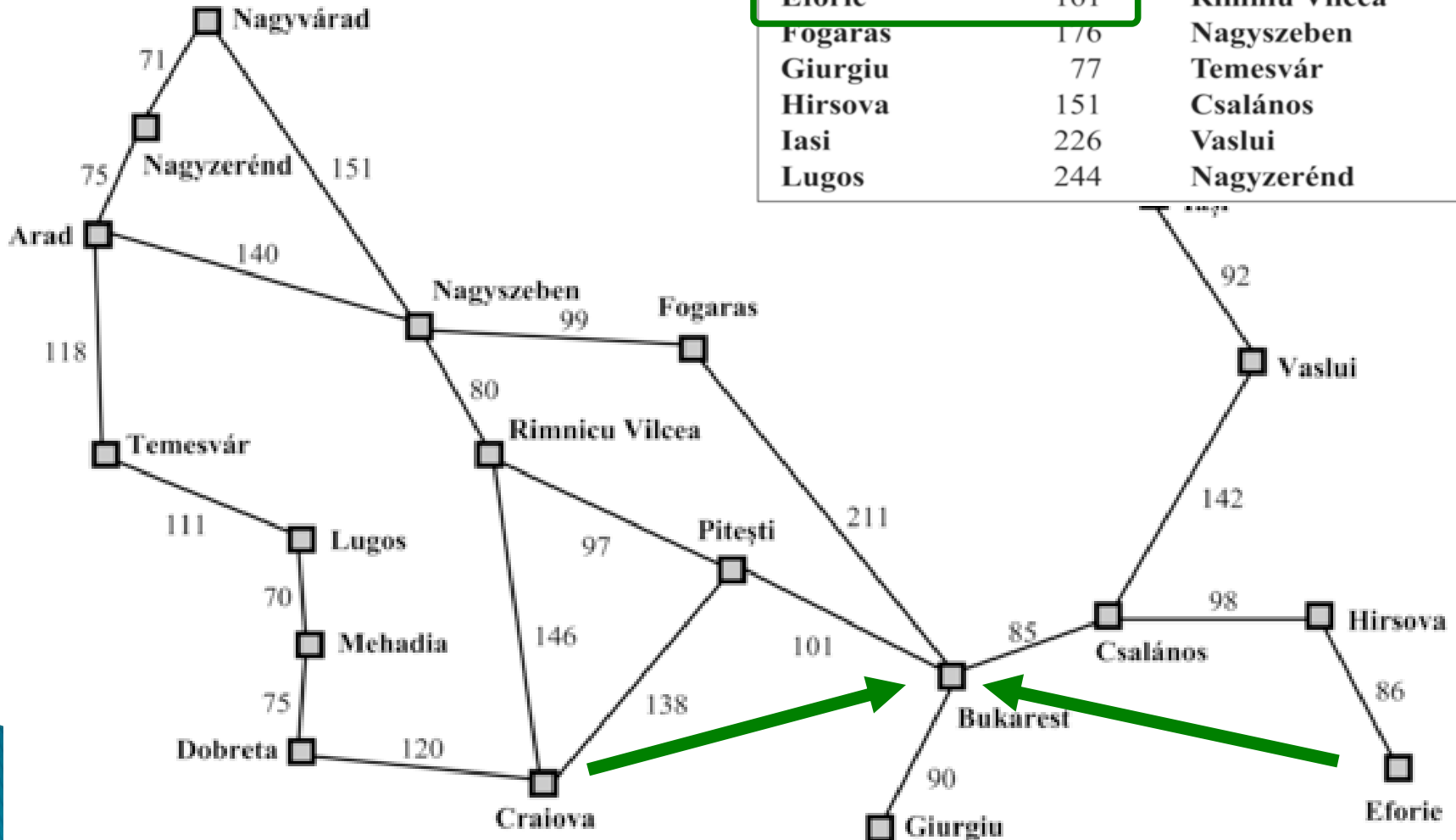


Let the heuristics – straightline distance ( $h_{SLD}$ )

Conditions fulfilled?

What about its error?

Arad	366	Mehadia	241
Bukarest	0	Neamt	234
Craiova	160	Nagyvárad	380
Dobreta	242	Pitești	100
Eforie	161	Rimnicu Vilcea	193
Fogaras	176	Nagyszeben	253
Giurgiu	77	Temesvár	329
Hirsova	151	Csalános	80
Iasi	226	Vaslui	199
Lugos	244	Nagyzerénd	374



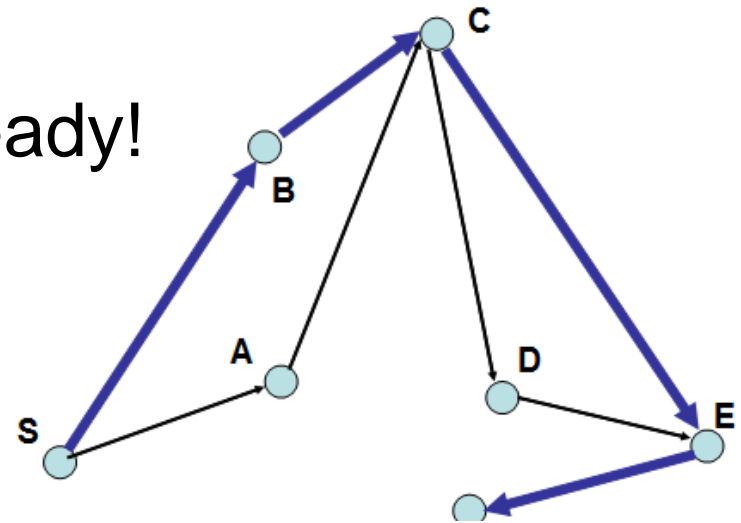
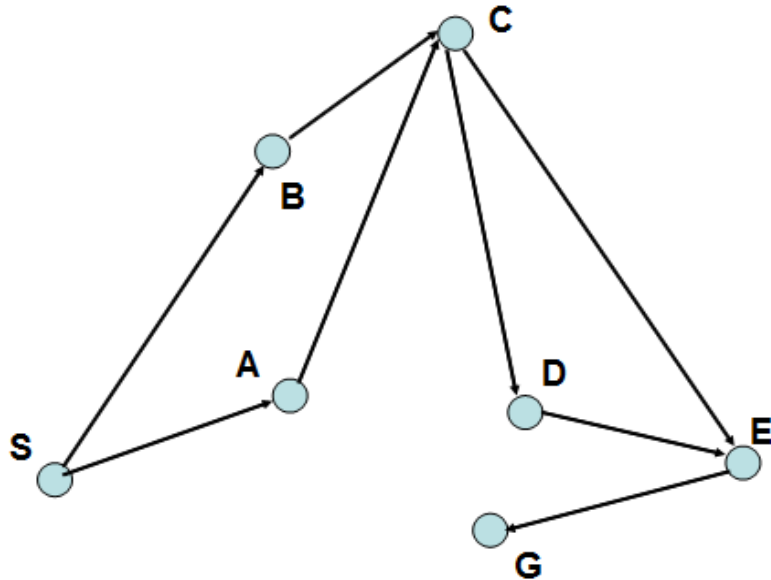
# Best-first search

- ▶ General approach of informed search:
  - Best-first search: a node is selected for expansion based on an *evaluation function*  $f(n)$  (*heuristics*) in TREE-SEARCH().
- ▶ Idea: the evaluation function measures distance to the goal.
  - Choose the node which *appears* best
- ▶ Implementation:
  - *fringe* is a queue sorted in decreasing order of desirability (i.e.  $f(n)$ ).
  - Special cases: greedy search, A\* search

(Ro-MohoK)

# Best-First with only

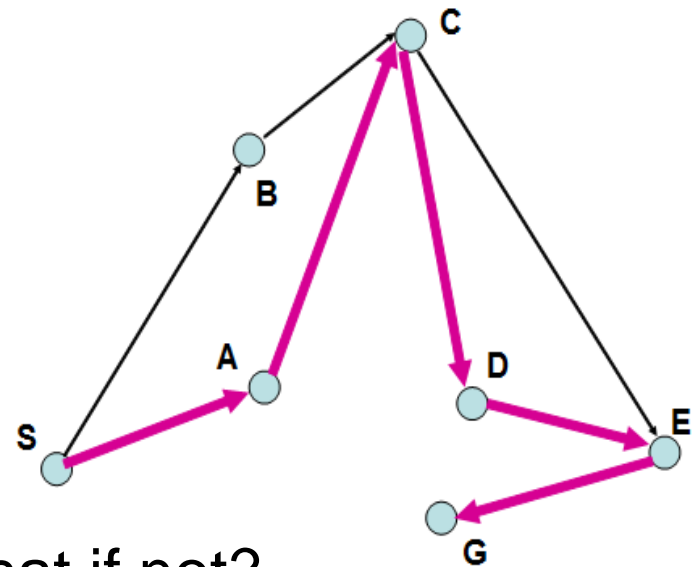
$f(n) = h(n)$  may be greedy!



Complexity: time = memory

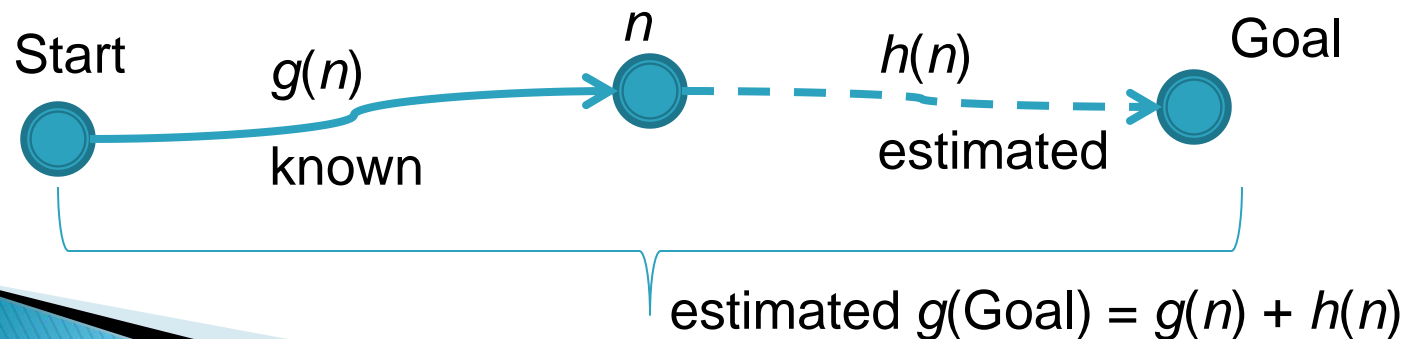
If exact  $h(n)$ :

linear time and memory, but what if not?



# 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.
  - (i.e. the sum of action costs along the path)
  - $h(n)$  estimates cost to get from the node to the closest goal.
  - $f(n)$  estimates the total cost of a path through  $n$  to the goal.





# A\* search

- ▶ A\* search uses an admissible heuristic
  - A heuristic is *admissible* if it *never overestimates* the cost to reach the goal (~optimistic).

Formally:

1.  $h(n) \leq h^*(n)$  where  $h^*(n)$  is the true cost from  $n$
2.  $h(n) \geq 0$ , and  $h(G) = 0$  for any goal  $G$ .

(e.g.  $h_{SLD}(n)$  never overestimates the actual road distance)

## Theorem:

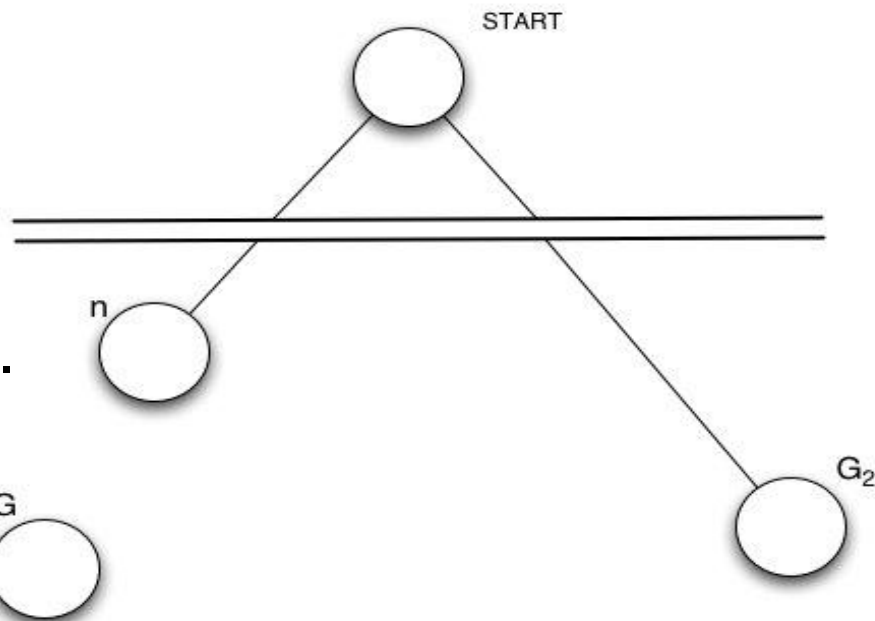
If  $h(n)$  is admissible, then

A\* using BEST-FIRST-SEARCH() with selector function  $f(n)$  is optimal (with respect to the path cost).

# Optimality of A\*(standard proof)

Suppose a suboptimal goal  $G_2$  is in the queue.

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



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

Since  $f(G_2) > f(n)$ , A\* will never select  $G_2$  for expansion (i.e. for checking, but note that  $G_2$  can be inside the queue).

An example:

Handling of the:

best node

open list (fringe)

closed list

Searching for optimal (lowest cost) path in Romania  
(Ro-AcsillagK.ppt)

# Consistency

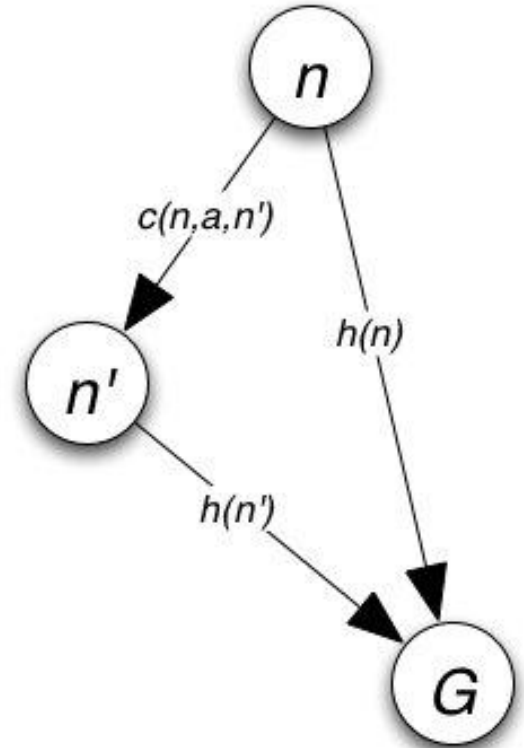
- ▶ A heuristic is *consistent* if

$$h(n) \leq c(n, a, n') + h(n')$$

- ▶ If  $h$  is consistent, we have

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + c(n, a, n') + h(n') \\ &\geq g(n) + h(n) \\ &\geq f(n) \end{aligned}$$

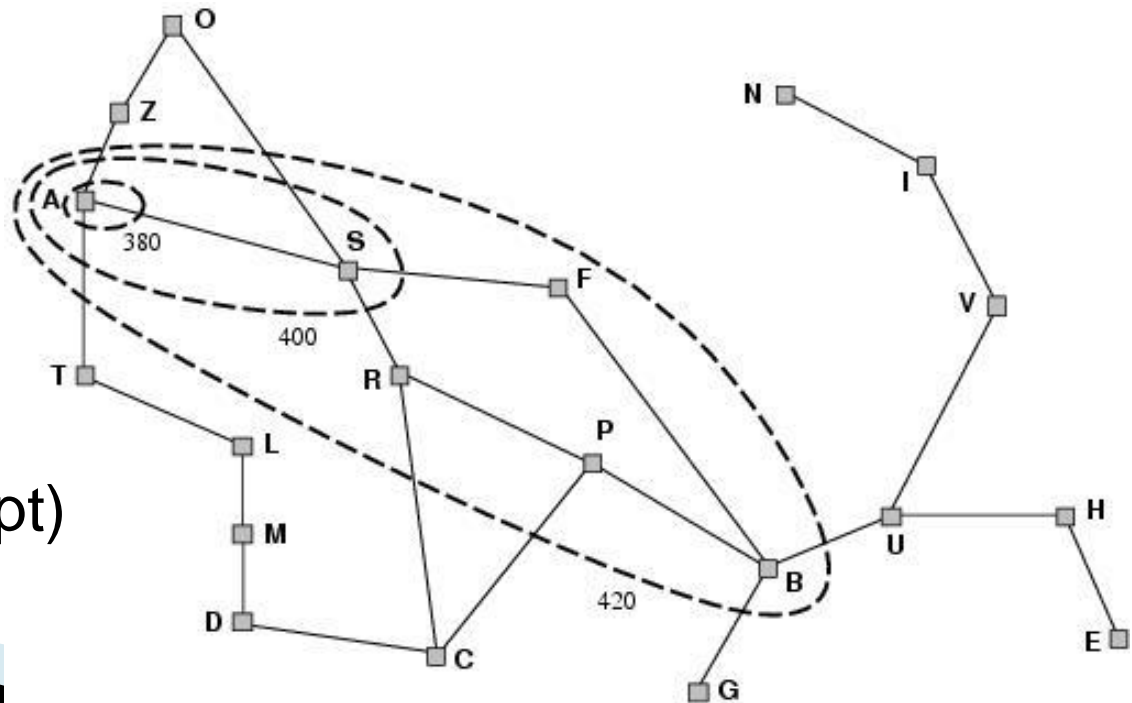
i.e.  $f(n)$  is non-decreasing along any path to the goal.



# Consistency and Optimality – f-contours

- ▶ A\* expands nodes in order of increasing  $f$  values
- ▶ f-contours can be drawn in state space
  - Uniform-cost search works in circular contours.
  - A\* f-contours are elongated toward the goal
  - A\* expands all nodes within f-contours with  $f(n) < C^*$  and some nodes on the goal contour  $f(n)=C^*$ .

Contour  $i$  encloses all nodes with  $f=f_i$ , where  $f_i < f_{i+1}$ .



(A-star-USA-konturok.ppt)

# A\* search, evaluation

- ▶ Completeness: YES
  - Since bands of increasing  $f$  are added
  - Unless there are infinitely many nodes with  $f < f(G)$

**Locally finite graphs** - finite branching factor  
- bounded action cost  $> \varepsilon > 0$ .

# A\* search, evaluation

- ▶ Completeness: YES
- ▶ Time complexity:
  - Number of nodes expanded is still exponential in the length of the solution (but significantly less than for a noninformed search).

# A\* search, evaluation

- ▶ Completeness: YES
- ▶ Time complexity: (exponential with path length)
- ▶ Space complexity:
  - It keeps all generated nodes in memory
  - Hence space is the major problem, not time



# A\* search, evaluation

- ▶ Completeness: YES
- ▶ Time complexity: (exponential with path length)
- ▶ Space complexity: (exponential, all nodes are stored)
- ▶ Optimality: YES
  - Cannot expand  $f_{i+1}$  until  $f_i$  is finished.
  - A\* expands all nodes with  $f(n) < C^*$
  - A\* expands some nodes with  $f(n) = C^*$
  - A\* expands no nodes with  $f(n) > C^*$

Also *optimally efficient* (details see textbook)

# Finding heuristic functions

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- ▶ E.g In the 8-puzzle
  - An average solution cost is about 22 steps (branching factor +/- 3)
  - Exhaustive search to depth 22 means  $3.1 \times 10^{10}$  states.
  - A good heuristic function can reduce the search process.

# Finding heuristic functions

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

Goal State

- ▶ E.g In the 8-puzzle the two commonly used heuristics are:
- ▶  $h_1$  = the number of misplaced tiles
  - $h_1(s)=8$
- ▶  $h_2$  = the sum of the distances of the tiles from their goal positions (Manhattan distance).
  - $h_2(s)=3+1+2+2+2+3+3+2=18$

- ▶  $h_1$  = the number of misplaced tiles
- ▶  $h_2$  = the sum of the distances of the tiles from their goal positions (Manhattan distance)

$d$	Search Cost			Effective Branching Factor		
	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	1.27
22	–	18094	1219	–	1.48	1.28
24	–	39135	1641	–	1.48	1.26

# Heuristics – comparing accuracy and efficiency

$b^*$  **effective branching factor**: if the number of all nodes expanded by search is  $N$ , the depth of the solution is  $d$ , then  $b^*$  is the branching factor of that  $d$ -deep balanced tree, which contains exactly  $N$  nodes:

$$N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$$

(e.g. if for  $d = 5$  and  $N = 52$ , then the effective branching factor is ca. 1.91)

Branching factor generated by a given heuristic function is generally a constant for the majority of problem examples belonging to a given problem class.

Measuring  $b^*$  on a small number of problems usually yields a good estimate.

A well designed heuristic function has the effective branching factor close to 1.

# Heuristics – comparing accuracy and efficiency

Testing  $h_1$ ,  $h_2$  heuristic functions:

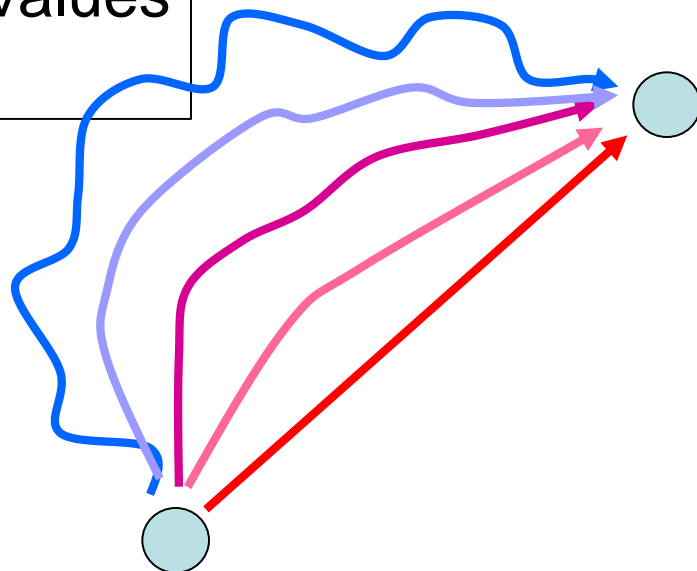
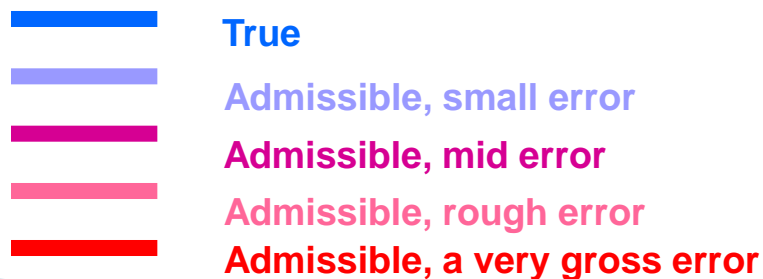
Is  $h_2$  always better than  $h_1$ ? Yes (why?).

for all nodes  $h_2(n) \geq h_1(n)$ :

**$h_2$  dominates  $h_1$ , dominance = efficiency**

A\* using  $h_2$  will expand less nodes than that using  $h_1$

Always use heuristics yielding higher values as long as it remains admissible.



# Inventing admissible heuristics

- ▶ Admissible heuristics can be derived from the exact solution cost of a relaxed version of the problem:
  - Relaxed 8-puzzle for  $h_1$  : a tile can move anywhere  
As a result,  $h_1(n)$  gives the shortest solution
  - Relaxed 8-puzzle for  $h_2$  : a tile can move to any adjacent square.  
As a result,  $h_2(n)$  gives the shortest solution.

The optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem (why?).

**The optimal solution of a relaxed problem is always an admissible heuristics for the real problem.**

# Inventing admissible heuristics

- ▶  $h(n) = \max\{ h_1(n), \dots, h_m(n) \}$
- ▶ Another way to find an admissible heuristic is through learning from experience:
  - Experience = solving lots of 8-puzzles
  - An inductive learning algorithm can be used to predict costs for other states that arise during search.
- Cost of computation?



# Iterative Deepening A\* (IDA\*)

In every iteration DF search with f-cost limit as the depth limit. Every iteration expands all nodes within the given f-cost contour and looks over the contour to find the next contour value.

IDA\* is **complete** and **optimal**, as the A\*, but its memory complexity is linear. Time complexity depends on how many values the f-cost can take.

(see demo)

## Bidirectional A\*

Same heuristics in both directions?

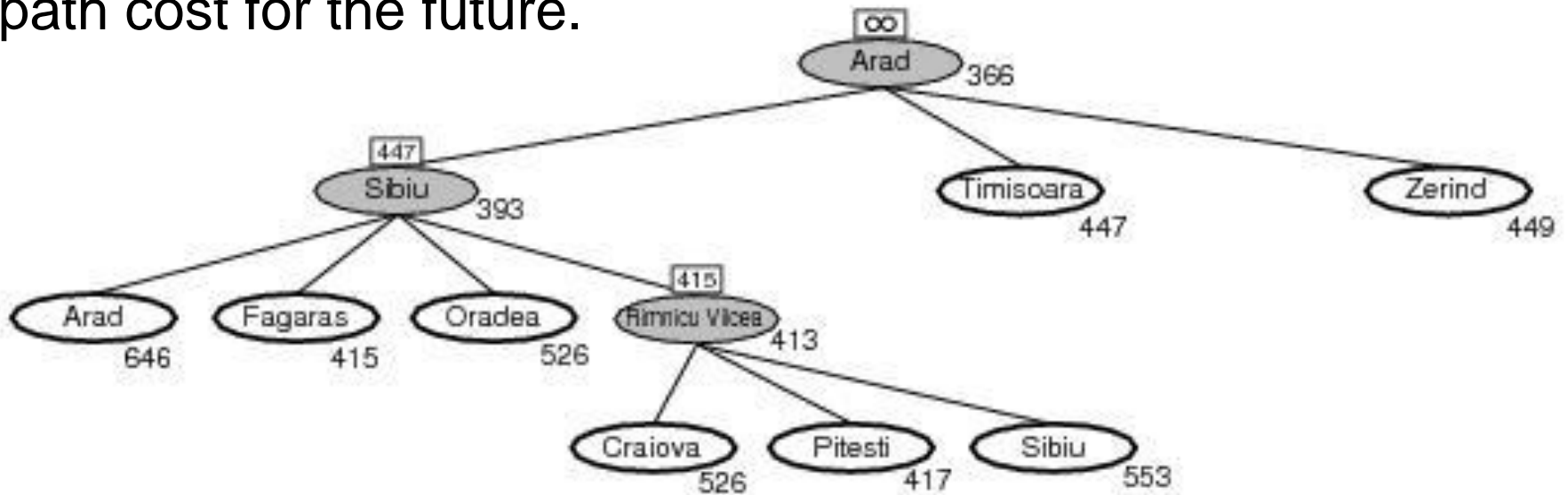
How to find matching nodes in two systems of f-contours?

# Recursive Best-First search

(Ro-RLEK.ppt)

It does BF search, acc. to f-cost, until a better alternative appears in left unexpanded branches.

Search switches over, freeing memory, but stores the actual path cost for the future.

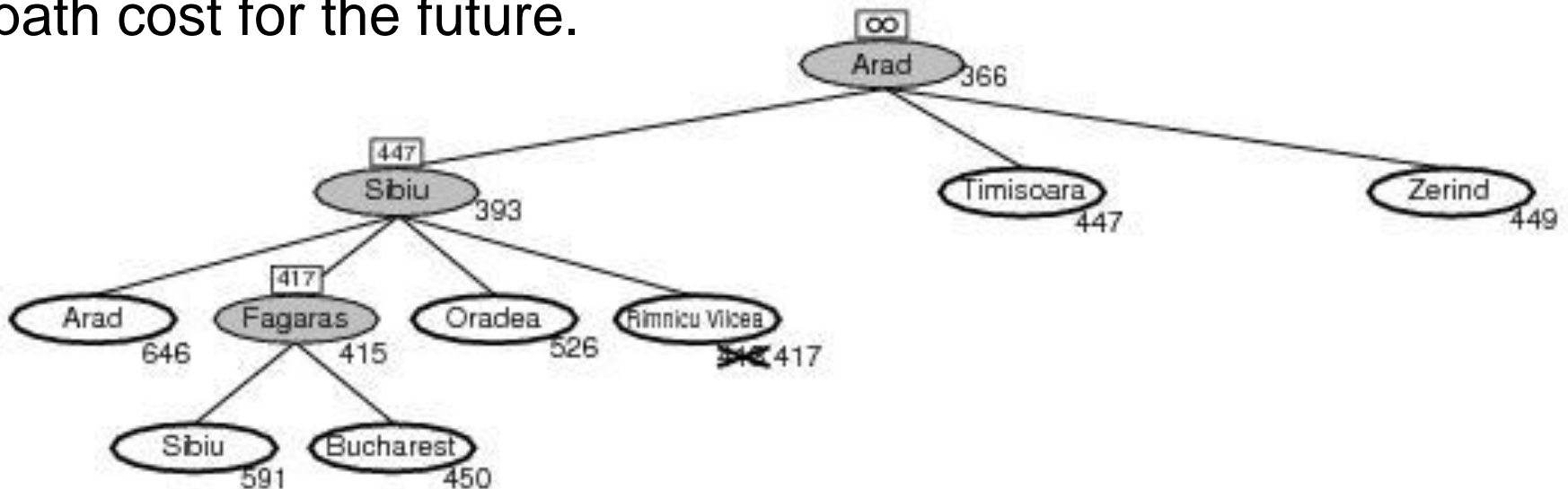


# Recursive Best-First search

(Ro-RLEK.ppt)

Does BF search, acc. to f-cost, until a better alternative appears in left unexpanded branches.

Search switches over, freeing memory, but stores the actual path cost for the future.

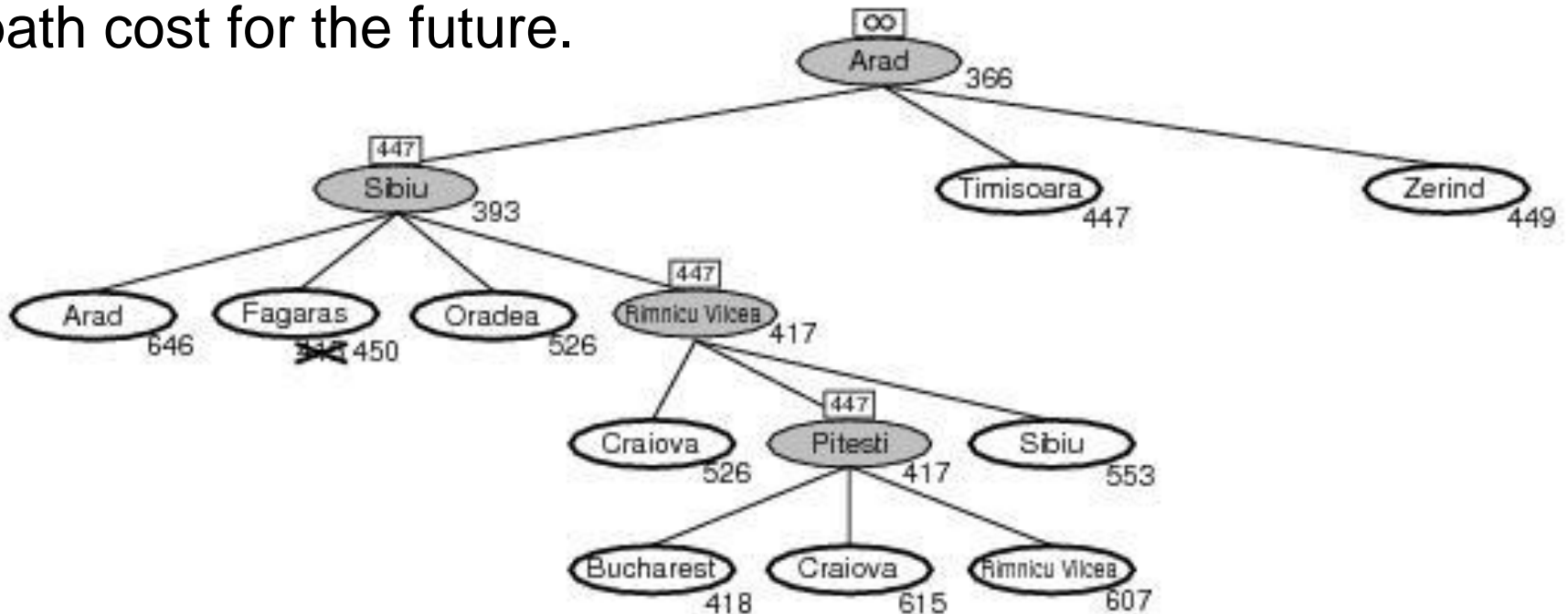


# Recursive Best-First search

(Ro-RLEK.ppt)

Does BF search, acc. to f-cost, until a better alternative appears in left unexpanded branches.

Search switches over, freeing memory, but stores the actual path cost for the future.



# Weighted A\* - variants

$$f(n) = w_1 g(n) + w_2 h(n)$$

$w_1$	$w_2$	
0	1	greedy (nonoptimal)
1	0	uniform cost (optimal)
1	1	A* (optimal, optimally efficient)
$(2-w)$	$w, w \geq 0$	weighted (suboptimal, depending on $w$ )
1	$w, w \gg 1$	almost greedy (optimal, nonefficient)
$w$	$1, w > 1$	$w$ -suboptimal
...		

A\* - other variants (D\*, any-time, learning, ...)

A\* - with non-admissible heuristics

# Local Search

Looking for optimum state = description of the optimal solution.  
Generally only the actual state is stored.

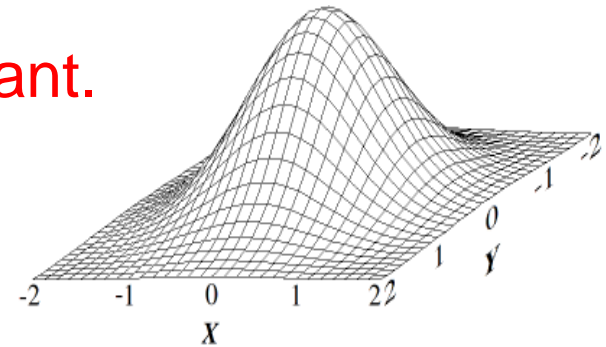
Looking for alternatives only in the vicinity of the actual state.

Special features of the search space ...

No backtracking, memory complexity constant.

Time complexity - linear.

Solvability warranty? (complete?, optimal?)



Hill-Climbing (discrete gradient search)

Simulated Annealing

Local beam search

Random start ... ..

Genetic algorithms

...

# Hill-Climbing

- Always tends toward better alternative
- Does not manage search tree

3 main problems

**local maximum:** stops in the first local optimum

**plateau:** no difference in direction of bettering, random choice

**ridge:** side slopes of the ridge steep, ridge slope can be very mild.

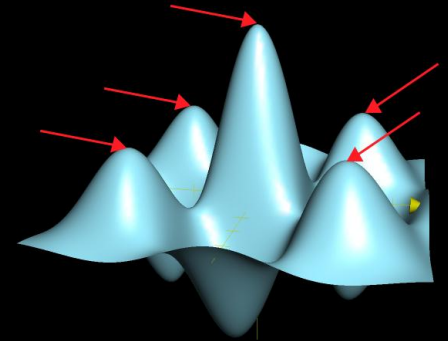
(Ro-HegyMaszo-K.ppt)

## Randomly started hill-climbing

HC is fast, can be frequently restarted

- time spent
- no improvement achieved

These are all local maxima



# Simulated Annealing

(SimAnneal-anim.ppt)

Instead of random restart

- It is permissible to go some steps back (downwards), to get out from the local optimum.

SA:

- instead of the best action a **random step** is chosen.
- if the step is an **improvement**, it is always done.
- if not, it is accepted only with a **probability** (Boltzmann-distribution).

$$P \approx \exp(- \Delta E / T)$$

The probability is exponentially decreasing with the “corrupting” capability of the step and the cooling:

- $\Delta E$  = deterioration in the cost function
- $T$  = “temperature”, cooling as time (search) is going on.



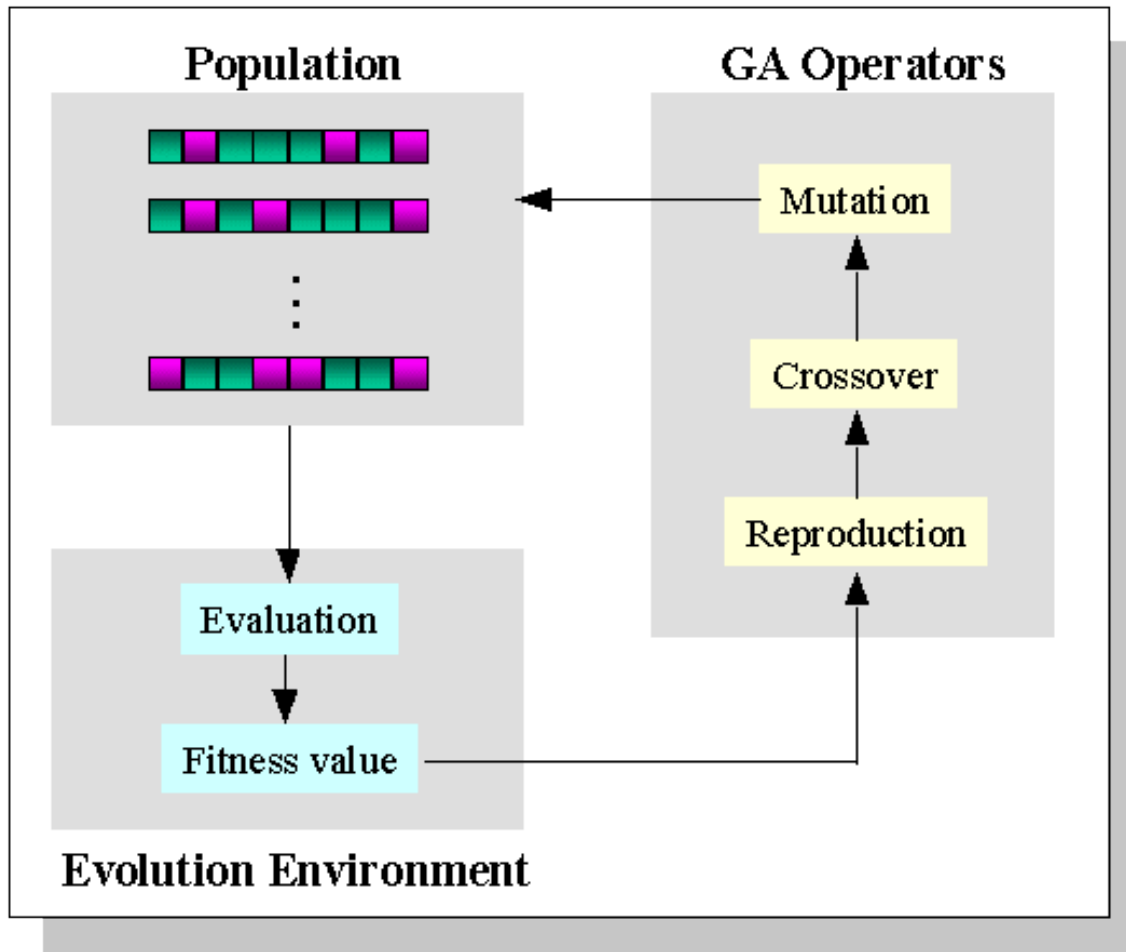
# Genetic algorithms

k randomly generated states = a population of individuals  
every state (individual) = defined over a finite alphabet  
(typically 0/1 string) = problem coding

- population
- fitness-function
- cross-over (selection, pairing) depending on fitness
- mutation
- *elitism* (transferring the best individuals of the current generation to the next generation)
- *Darwin-/ Lamarck-inheritance* (only genetic code, or the life experience also)
- shaping new population

Like Hill-Climbing, etc. (difference is in the reproduction)

# Genetic algorithms

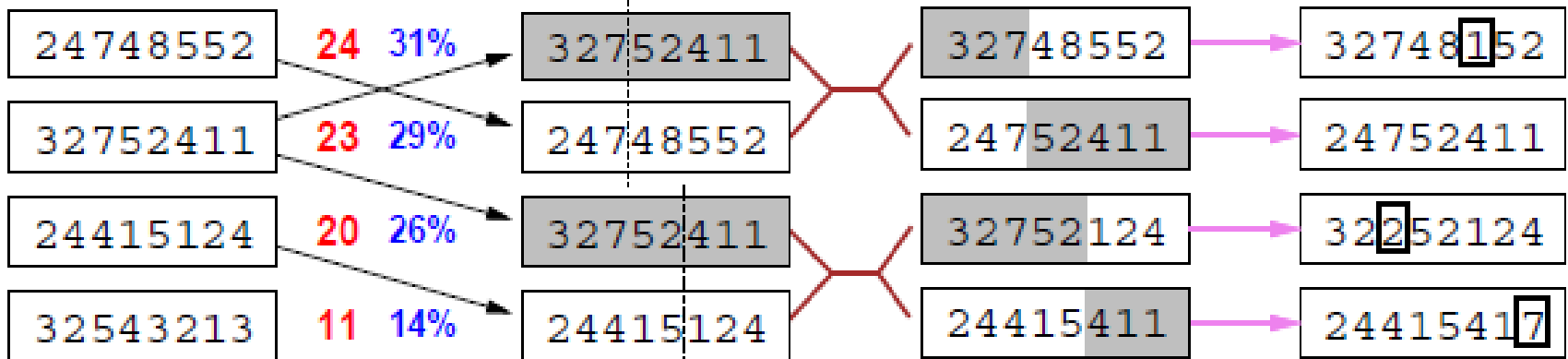
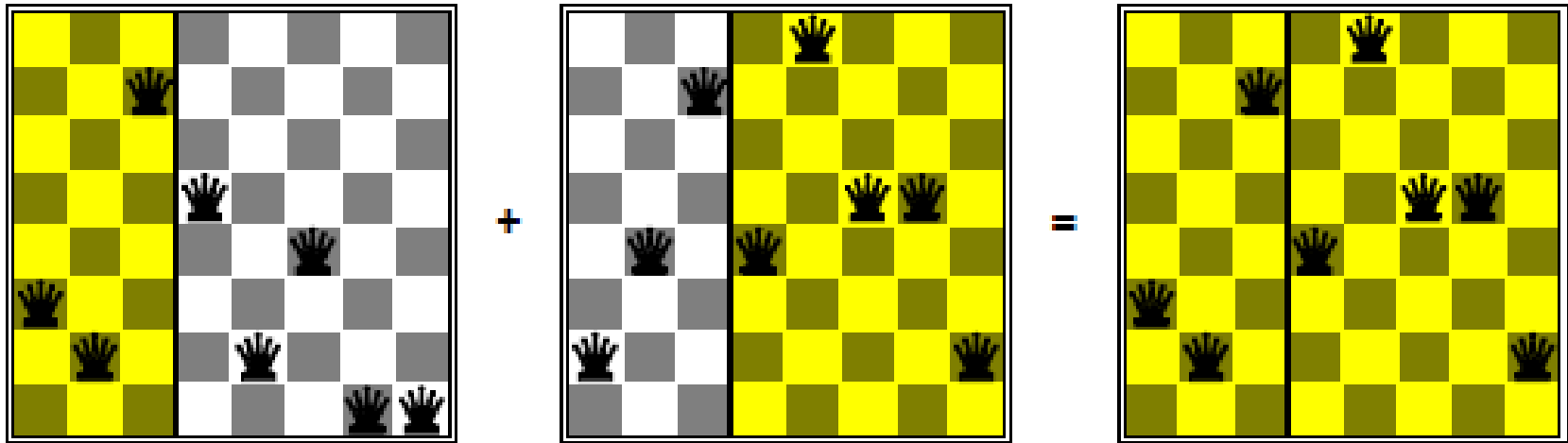


2006 NASA ST5 spacecraft antenna

[https://en.wikipedia.org/wiki/Evolved\\_antenna](https://en.wikipedia.org/wiki/Evolved_antenna)

# Genetic algorithms

8-queen problem  
fitness = 28 - n



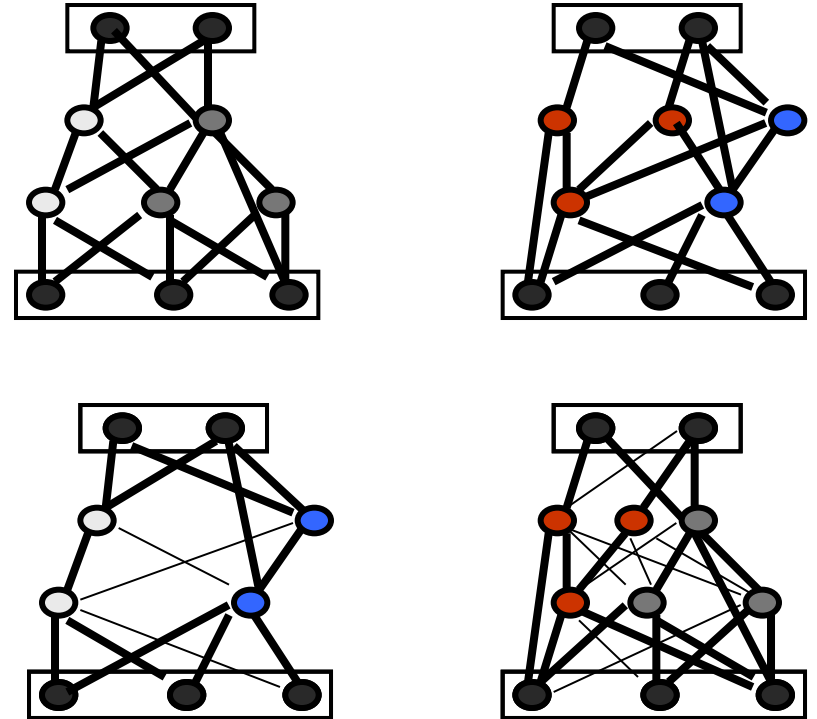
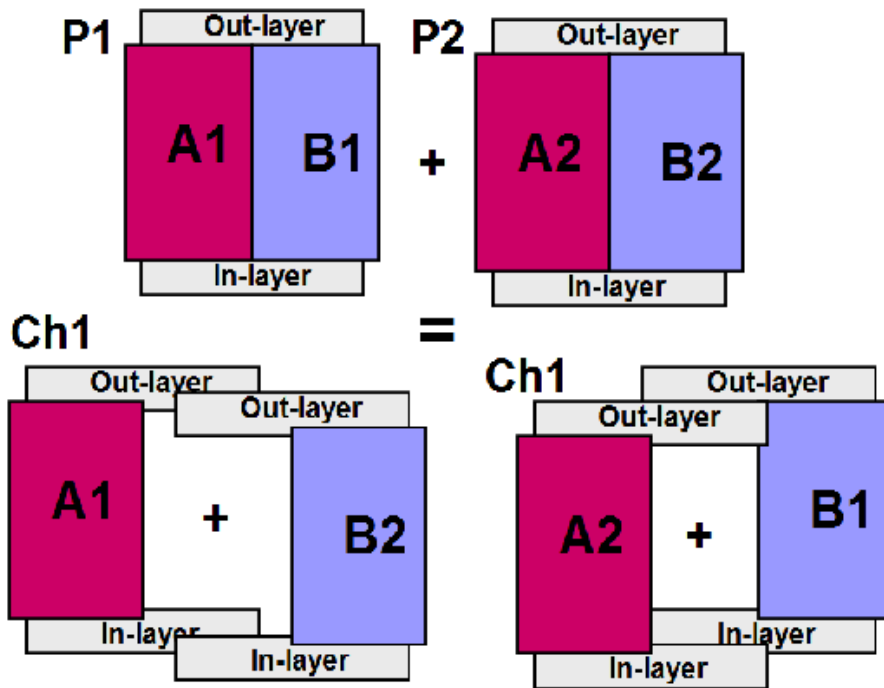
Fitness

Selection

Pairs

Cross-Over

Mutation



Darwin, Lamarck?

# And still more and more search algorithms ...

a. Every search algorithm has plenty of special versions

b. Finding new search algorithms – always a hot AI topic

c. Gradient based procedures in continuous spaces

$$\mathbf{x} \leftarrow \mathbf{x} - \alpha \nabla f(\mathbf{x})$$

$$\mathbf{x} \leftarrow \mathbf{x} - \alpha \mathbf{H}^{-1}(\mathbf{x}) \nabla f(\mathbf{x})$$

Relaxation method

Gradient with a fixed step

Gradient with optimized steps

Newton-method

And many more

d. (Tunnelled hill-climbing (minimizing + tunnelling) ...)

e. On-line search when the information is deficient or changing (exploratory problems), e.g. **learning real-time A\***, etc.

# Summary

- ▶ Heuristic function
- ▶ Admissible heuristics and A\*
- ▶ Local search
- ▶ Suggested reading
  - Prieditis: Machine Discovery of Effective Admissible Heuristics, 1993
- ▶ Demos

<https://qiao.github.io/PathFinding.js/visual/>

<https://www.movingai.com/SAS/>

<https://www.movingai.com/SAS/ASG/>

<https://www.movingai.com/SAS/ASM/>

<http://math.hws.edu/eck/js/genetic-algorithm/GA.html>

