- Uninformed search无信息的搜索：除了问题中提供的定义之外没有任何关于状态的附加信息。

- Informed search有信息的搜索：在问题本身的定义之外还可利用问题的特定知识。

Review: Last Class


```
function TREE-SEARCH( problem, fringe) returns a solution, or failure
    fringe ← INSERT(MAKE-NODE(INTIAL-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
- Variety of uninformed search strategies
  - Iterative deepening search uses only linear space and not much more time than other uninformed algorithms

Uninformed Search Strategies

Uninformed search strategies use only the information available in the problem definition

- Breadth-first search （广度优先搜索）
- Uniform-cost search （代价一致搜索）
- Depth-first search （深度优先搜索）
- Depth-limited search（深度有限搜索）
- Iterative deepening search （迭代深入深度优先搜索）

Uninformed Search Strategies

- Summary of uninformed tree (problem-solving) search:
  - Algorithms differ by method of *expansion, or choice of state-action* sequence for offline evaluation
  - Complexity tradeoffs, but poor (worst case or typical case) performance from all when state space is large

Informed Search Algorithms

Chapter 4

http://staff.ustc.edu.cn/~linlixu/ai2018spring/

Outline

- Best-first search（最佳优先搜索）
  - Greedy best-first search
  - A* search
- Heuristics
- Local search algorithms
  - Hill-climbing search
  - Simulated annealing search
  - Local beam search
  - Genetic algorithms

Best-First Search

Idea: use an evaluation function（评价函数） $f(n)$ for each node

— estimate of "desirability"

$\Longrightarrow$ Expand most desirable unexpanded node

Implementation:

fringe is a queue sorted in decreasing order of desirability

— priority queue （优先级队列）

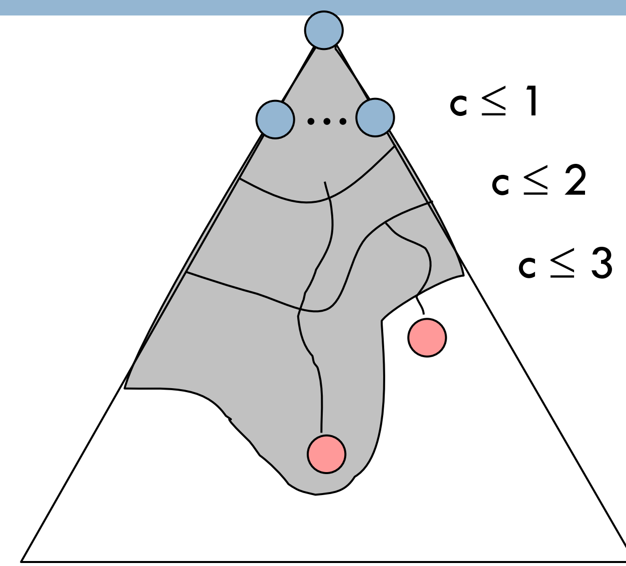
Special cases:

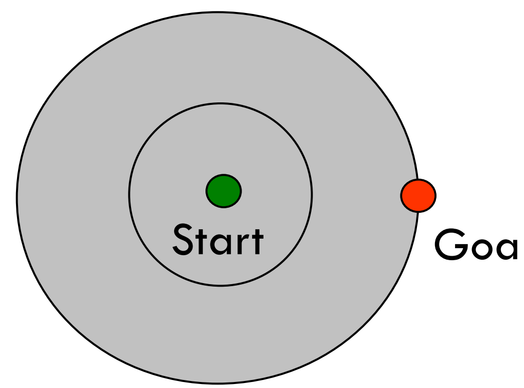
greedy search

A* search

Uniform Cost

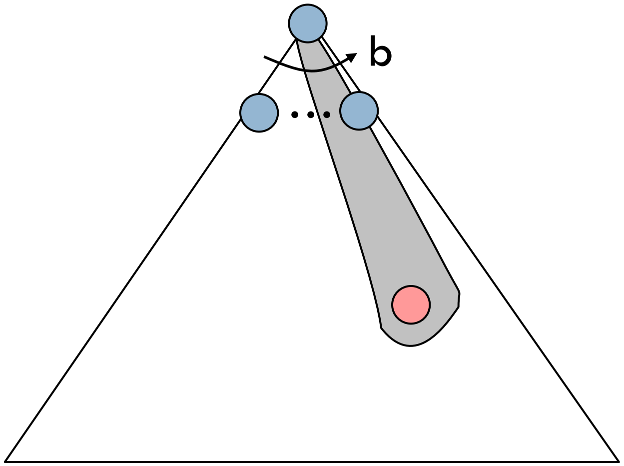
- Strategy: expand lowest path cost
- The good: UCS is complete and optimal!
- The bad:
  - Explores options in every "direction"
  - No information about goal location



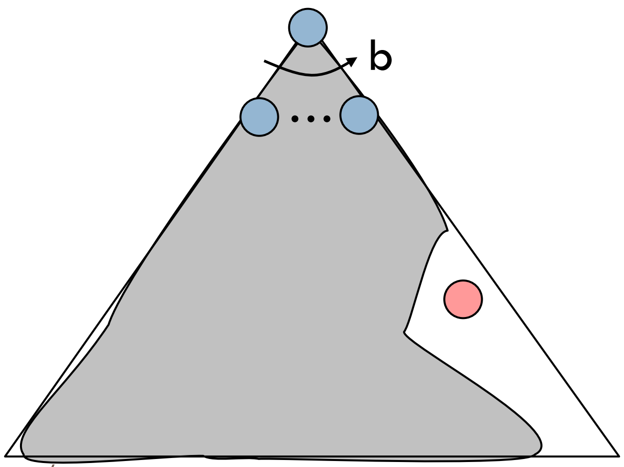


Best First

- Strategy: expand nodes which appear closest to goal
  - Heuristic（启发函数）: function which maps states to distance

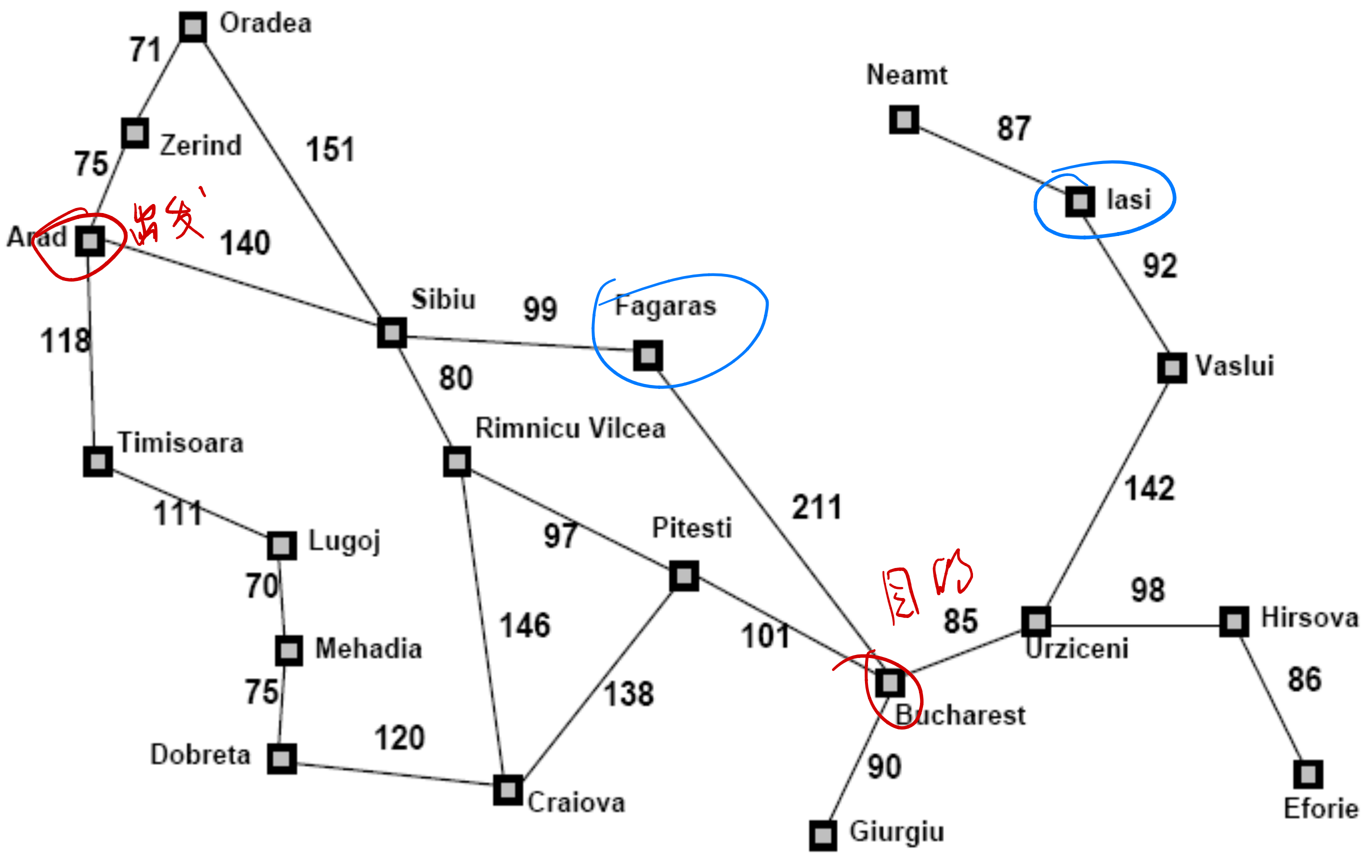


- A common case:
  - Best-first takes you straight to the (wrong) goal



- Worst-case: like a badly-guided DFS

Romania with Step Costs in Km



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

Evaluation function $f(n) = h(n)$ (heuristic function 启发函数)

= estimate of cost from *n* to the closest goal

（节点n到目标节点的最低耗散路径的耗散估计值）

- $h(n)$ a problem-specific function
- $h(n)$ = *0 if n is goal node*

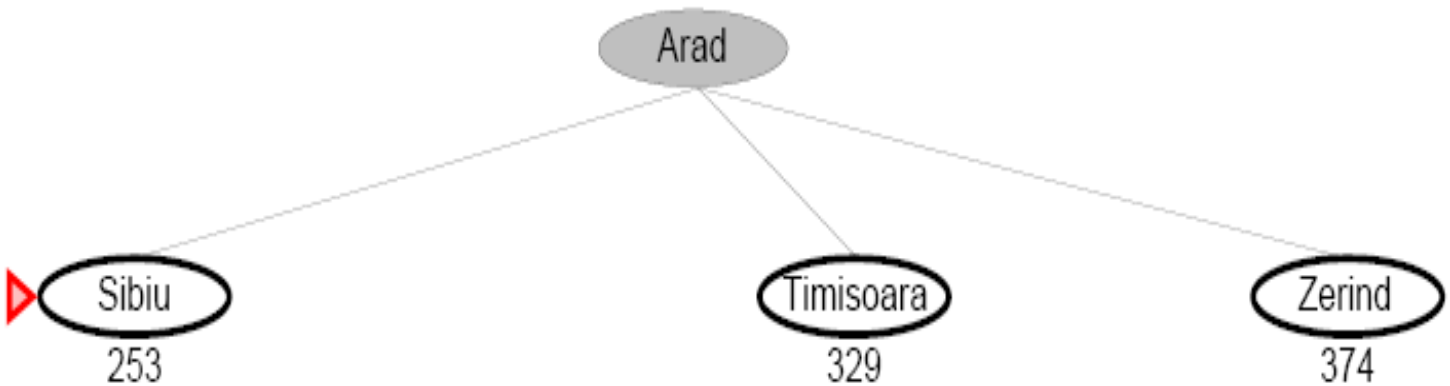
E.g., $h_{\mathrm{SLD}}(n)$ = straight-line distance from *n* to Bucharest

Greedy search expands the node that appears to be closest to goal（试图扩展离目标节点最近的点）

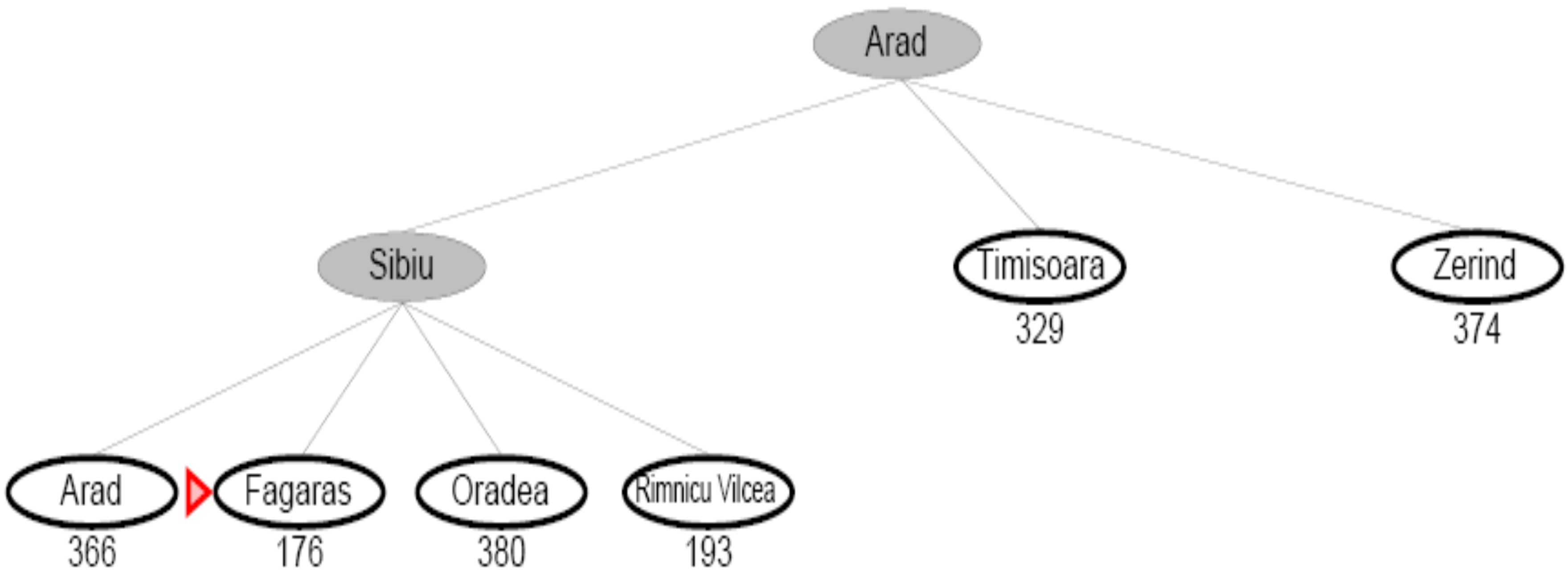
Greedy Search Example

http://staff.ustc.edu.cn/~linlixu/ai2018spring/

Greedy Search Example

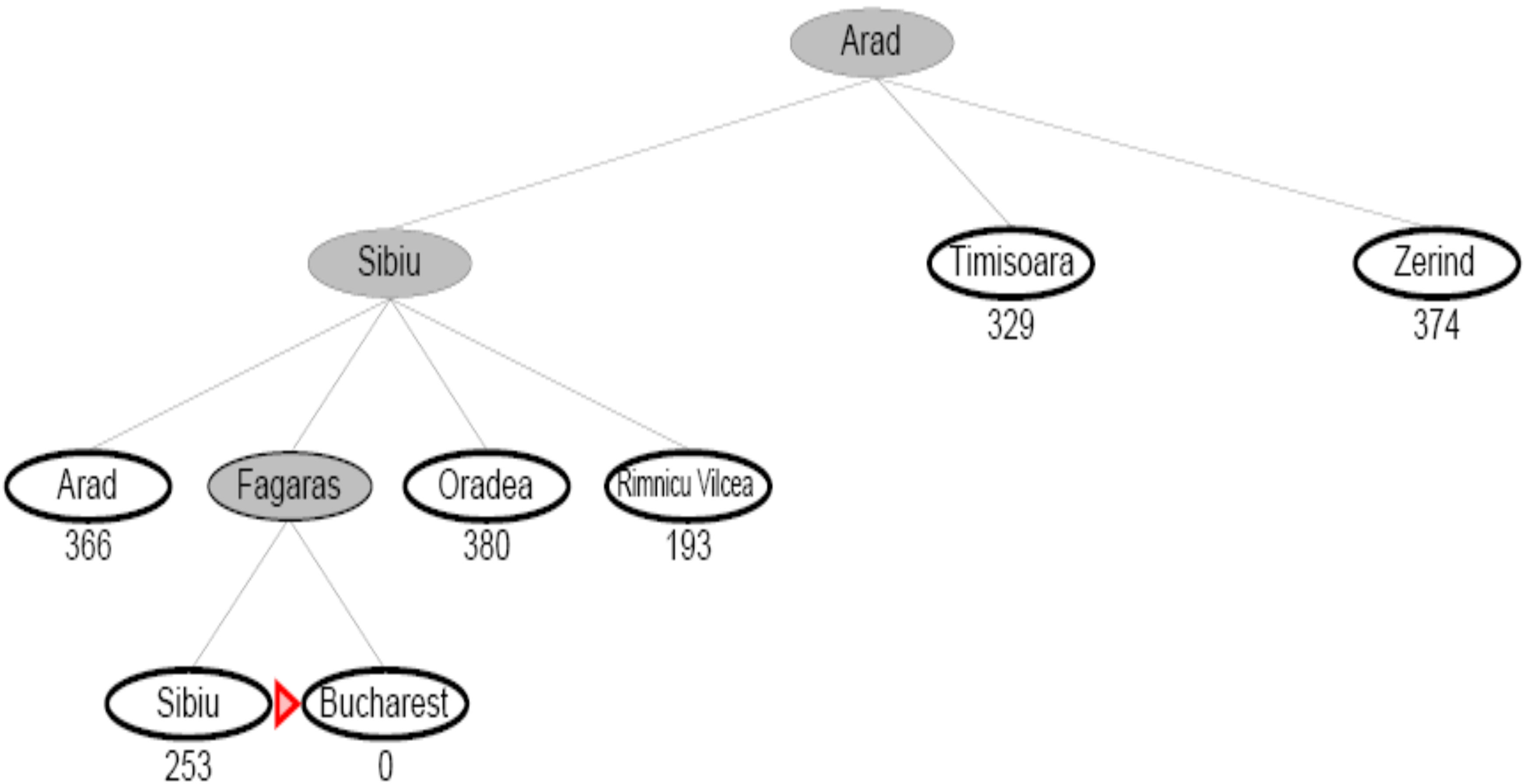


Greedy Search Example



http://staff.ustc.edu.cn/~linlixu/ai2018spring/

Greedy Search Example



http://staff.ustc.edu.cn/~linlixu/ai2018spring/

Properties of Greedy Search

Complete??

Properties of Greedy Search

Complete?? No — can get stuck in loops, e.g., with Oradea as goal,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

Time??

Properties of Greedy Search

Complete?? No — can get stuck in loops, e.g., with Oradea as goal,

Iasi $\rightarrow$ Neamt $\rightarrow$ Iasi $\rightarrow$ Neamt $\rightarrow$

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space??

b: Branching factor
d: Solution depth
m: Maximum depth

Properties of Greedy Search

Complete?? No — can get stuck in loops, e.g., with Oradea as goal,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$ — keeps all nodes in memory

Optimal??

b: Branching factor
d: Solution depth
m: Maximum depth

Properties of Greedy Search

Complete?? No — can get stuck in loops, e.g., with Oradea as goal,

Iasi → Neamt → Iasi → Neamt →

Complete in finite space with repeated-state checking

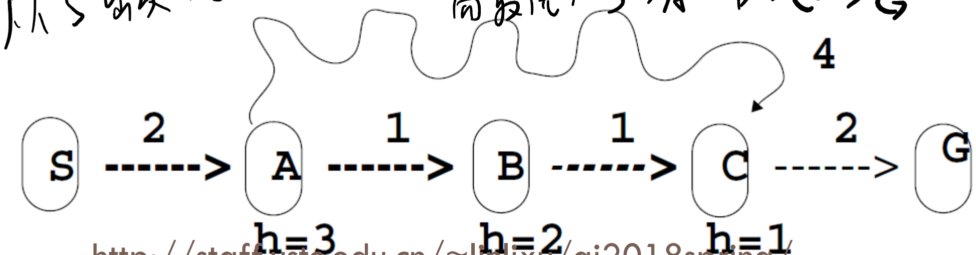
Time?? $O(b^m)$, but a good heuristic can give dramatic improvement

Space?? $O(b^m)$ — keeps all nodes in memory

b: Branching factor
d: Solution depth
m: Maximum depth

Optimal?? No

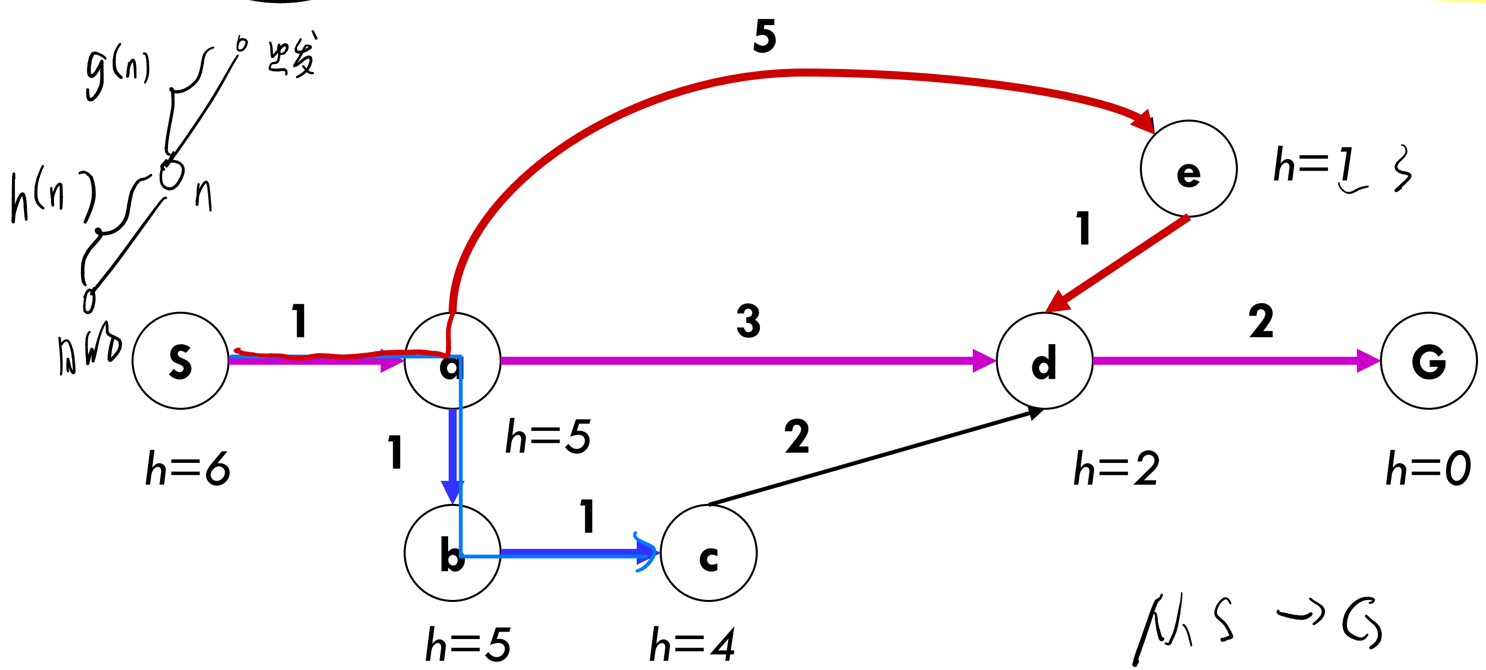
从S出发到G：由贪心：从 S→A→C→G
而最优：S→A→B→C→G



http://staff.ustc.edu.cn/~linlixu/ai2018spring/

Combining UCS and Greedy

- Uniform-cost orders by path cost, or *backward cost* g(n)
- Greedy-search orders by goal proximity, or *forward cost* h(n)



- A* Search orders by the sum: f(n) = g(n) + h(n)

Example: Teg Grenager

A* Search

Evaluation function $f(n) = g(n) + h(n)$

$g(n)$ = cost so far to reach n —到达节点n的耗散

$h(n)$ = estimated cost to goal from n

—启发函数：从节点n到目标节点的最低耗散路径的耗散估计值

$f(n)$ = estimated total cost of path through n to goal

—经过节点n的最低耗散的估计函数

Theorem: A* search is optimal

A* search uses an admissible heuristic 可采纳启发式

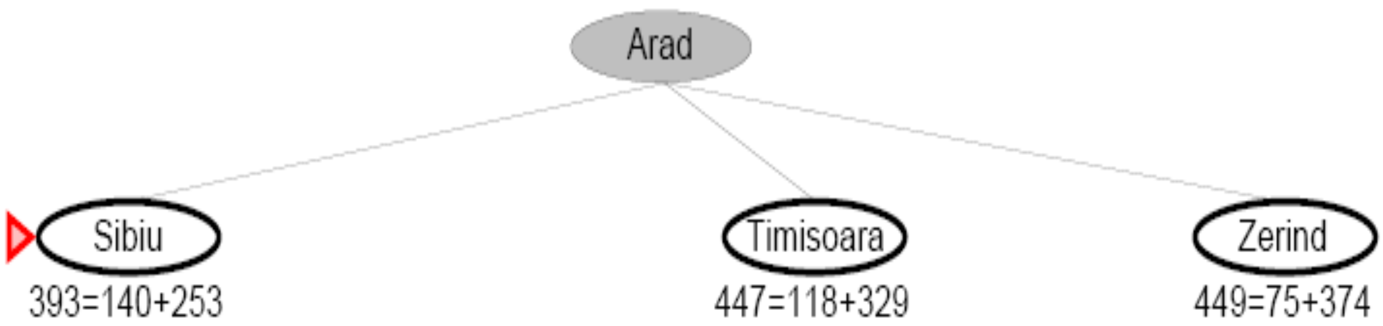
i.e., $h(n) \leqslant h^*(n)$ where $h^*(n)$ is the true cost from n.

(Also require $h(n) \geqslant 0$, so $h(G) = 0$ for any goal G.)

E.g., $h_{\mathrm{SLD}}(n)$ never overestimates the actual road distance (SLD: Straight-Line Distance)

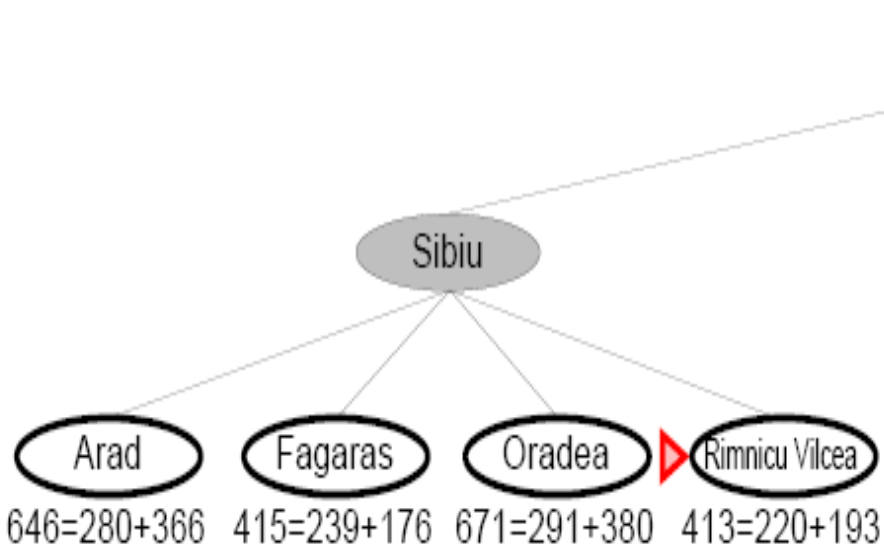
A* Search Example

A* Search Example



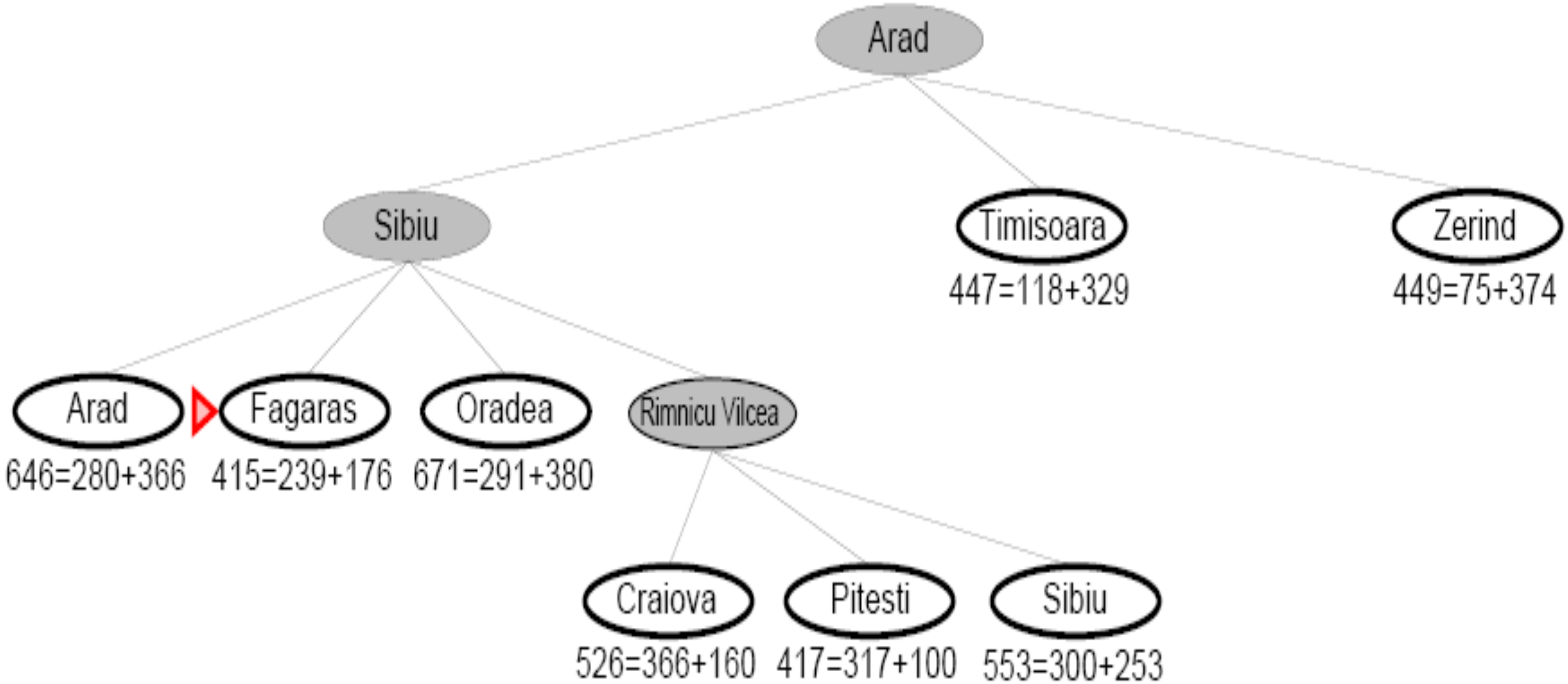
http://staff.ustc.edu.cn/~linlixu/ai2018spring/

A* Search Example



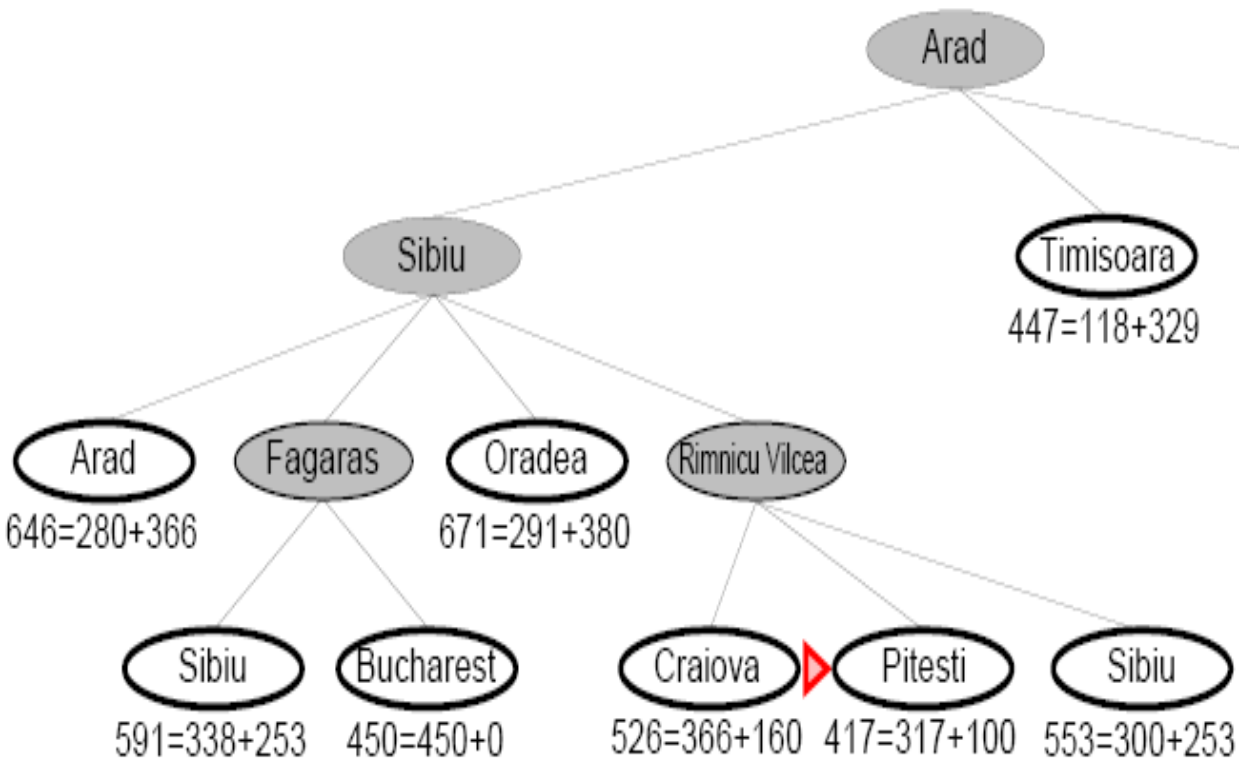
http://staff.ustc.edu.cn/~linlixu/ai2018spring/

A* Search Example



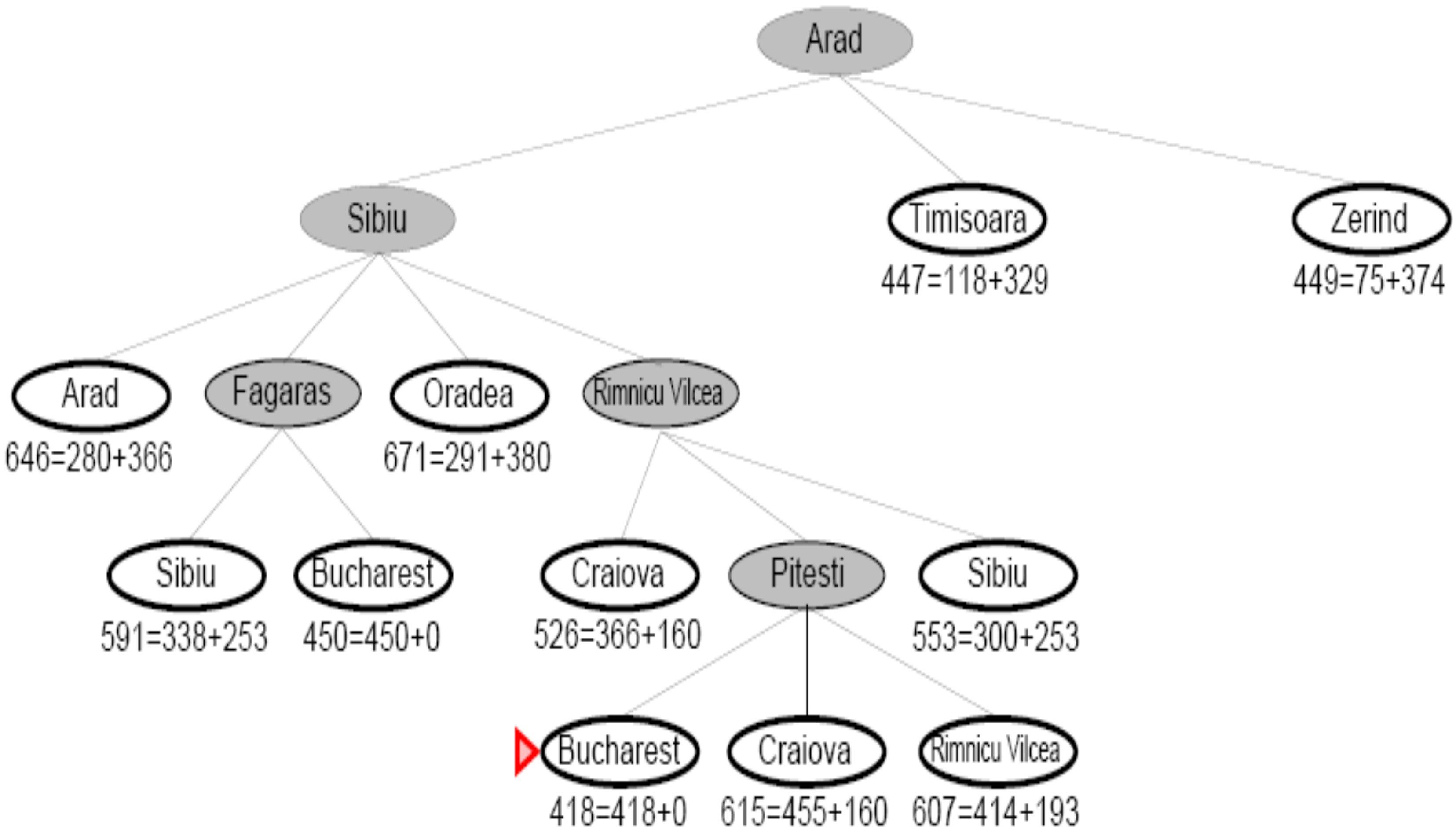
http://staff.ustc.edu.cn/~linlixu/ai2018spring/

A* Search Example



http://staff.ustc.edu.cn/~linlixu/ai2018spring/

A* Search Example



http://staff.ustc.edu.cn/~linlixu/ai2018spring/

Admissible Heuristics

- A* heuristic $h(n)$ is admissible（可采纳的） if for every node n,
  $h(n) \leq h^*(n)$, where $h^*(n)$ is the true cost to reach the goal state from n.

- An admissible heuristic never overestimates the cost to reach the goal（从不会过高估计到达目标的耗散）, i.e., it is optimistic（乐观的）

- Example: $h_{SLD}(n)$ (never overestimates the actual road distance)

- Theorem: If $h(n)$ is admissible, A^* using `TREE-SEARCH` is optimal

Coming up with admissible heuristics is most of what's involved in using A* in practice

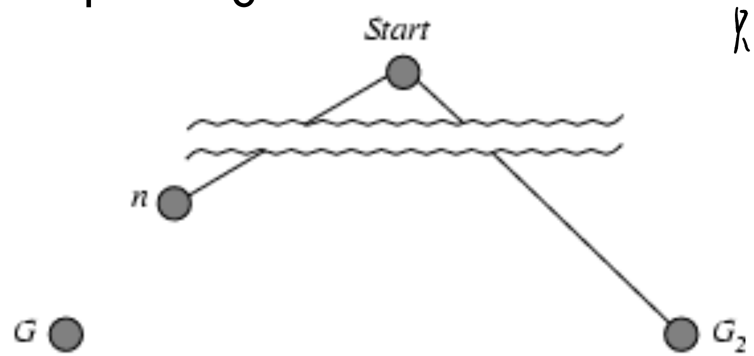
Optimality of A^* (proof)

Suppose 队列 some suboptimal（非最优） goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal G.

只需证明 $f(G_2) > f(n)$

则可保证算法会选择扩展n而非G_2

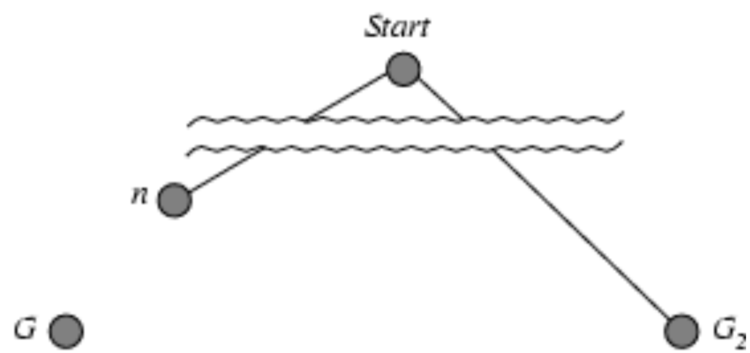
∵ 算法会每次选 f()最小的结点来扩展.



- $f(G_2) = g(G_2)$ since $h(G_2) = 0$
- $g(G_2) > g(G)$ since G_2 is suboptimal
- $f(G) = g(G)$ since $h(G) = 0$
- $f(G_2) > f(G)$ from above

Optimality of A* (proof)

Suppose some suboptimal goal G_2 has been generated and is in the fringe. Let *n* be an unexpanded node in the fringe such that *n* is on a shortest path to an optimal goal *G*.



- $f(G_2) > f(G)$ from above
- $h^*(\,)$为实际路径
- $h(n) \le h^*(n)$ since h is admissible 可采纳
- $g(n) + h(n) \le g(n) + h^*(n) = g(G) = f(G)$ *n* is on a shortest path to an optimal goal *G*
- $f(n) \le f(G)$

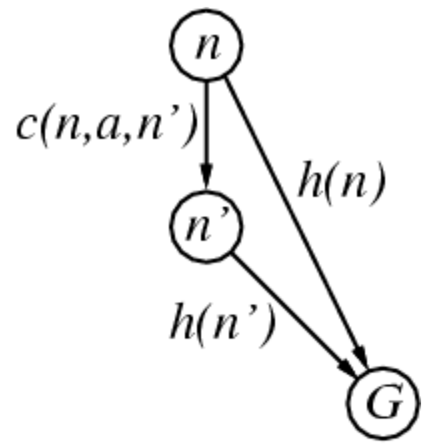
Hence $f(G_2) > f(n)$, and A* will never select G_2 for expansion

Consistent Heuristics

图搜索 课后看.

- A* heuristic is consistent（一致） if for every node *n*, every successor *n'* of *n* generated by any action *a*,

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



- If *h* is consistent, we have

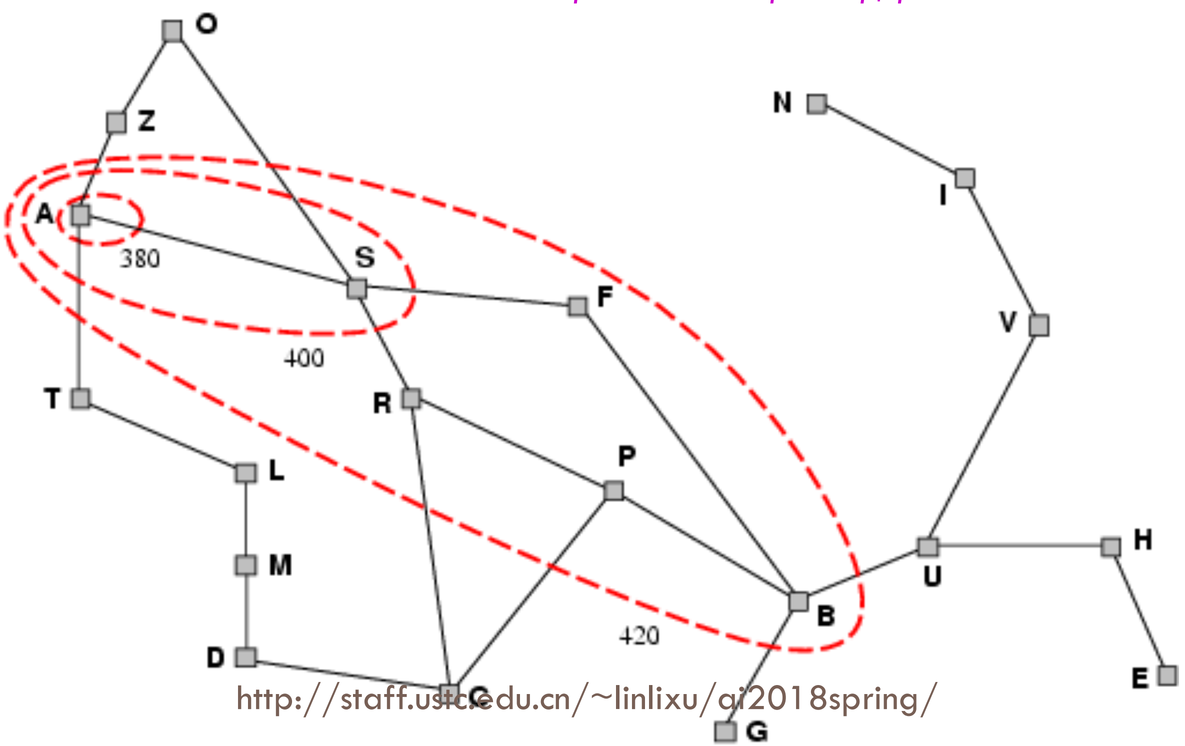
$$
\begin{aligned}
f(n') &= g(n') + h(n') \\
&= g(n) + c(n,a,n') + h(n') \\
&\geq g(n) + h(n) \\
&= f(n)
\end{aligned}
$$

- i.e., f(n) is non-decreasing along any path.

- Theorem: If h(n) is consistent, A* using GRAPH−SEARCH is optimal

Optimality of A*

- A* expands nodes in order of increasing *f* value
- Gradually adds "*f*-contours（等值线）" of nodes
- Contour *i* has all nodes with $f=f_i$, where $f_i < f_{i+1}$



Properties of A*

- Complete? Yes (unless there are infinitely many nodes with f $\le f(G)$)
- Time? A*算法对于任何给定的启发函数都是效率最优的
  But still exponential
- Space? Keeps all nodes in memory
- Optimal? Yes

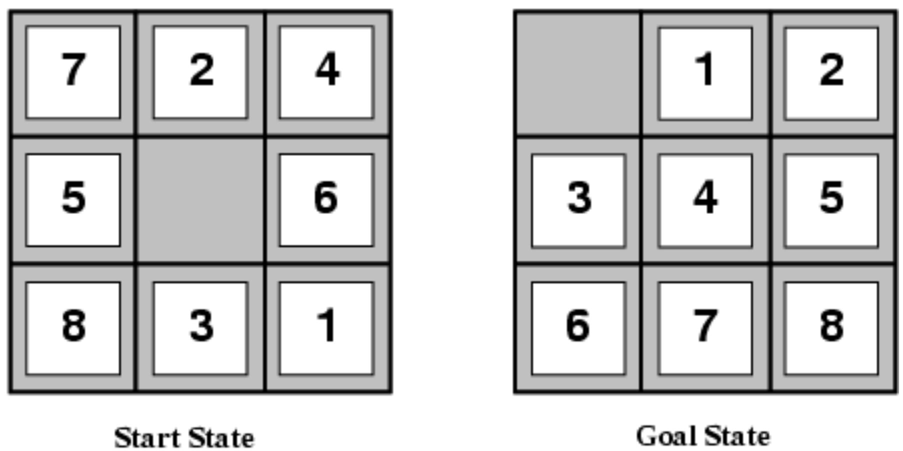
A^* expands all nodes with $f(n) < C^*$

A^* expands some nodes with $f(n) = C^*$

A^* expands no nodes with $f(n) > C^*$

8-puzzle Revisited

- 8-puzzle — 把棋子水平或者竖直地滑动到空格中，直到目标局面：



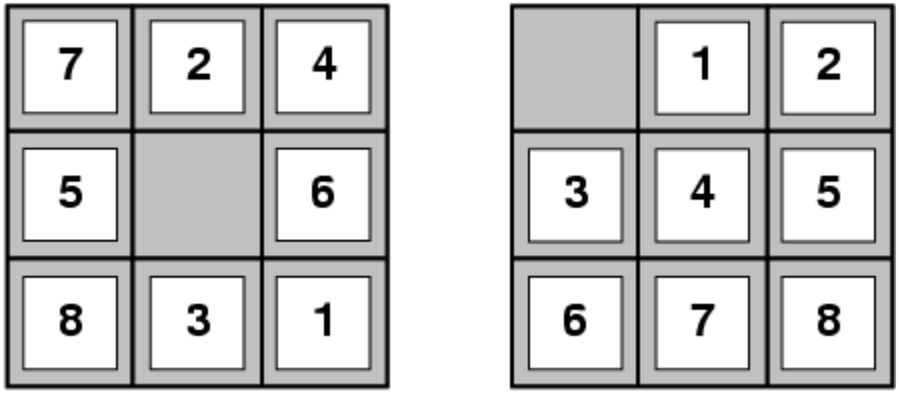
- 平均解步数是22步。分支因子约为3
  - 到达深度为22的穷举搜索将考虑约$3^{22} \approx 3.1 \times 10^{10}$
  - 状态个数O((n+1)!)，NP完全问题

Admissible Heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles（错位的棋子数）
- $h_2(n)$ = total Manhattan distance（所有棋子到其目标位置的水平竖直距离和）

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



Start State　　Goal State

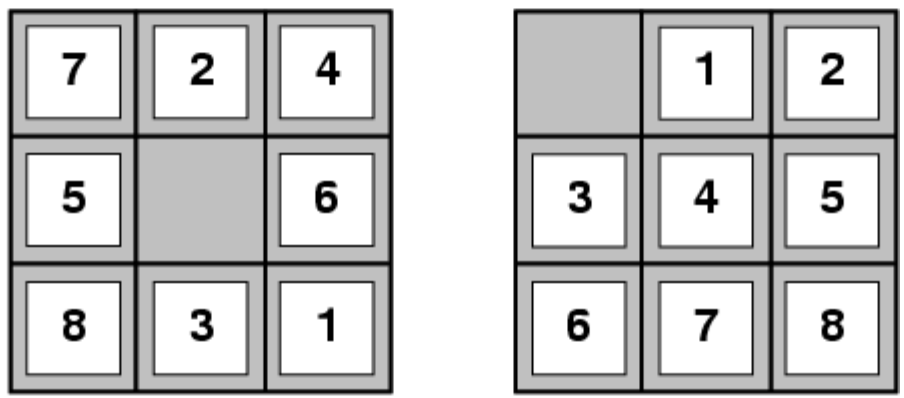
- $h_1(S)$ = ?
- $h_2(S)$ = ?

Admissible Heuristics

E.g., for the 8-puzzle:

- $h_1(n)$ = number of misplaced tiles（错位的棋子数）
- $h_2(n)$ = total Manhattan distance（所有棋子到其目标位置的水平竖直距离和）

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



Start State Goal State

- $h_1(S)$ = ? 8
- $h_2(S)$ = ? 3+1+2+2+2+3+3+2 = 18

Dominance

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)
- then h_2 **dominates** h_1 (dominate 统治、占优)
- h_2 is better for search

- Typical search costs (average number of nodes expanded):

设置启发函数本身也有代价

- $d=12$ IDS = 3,644,035 nodes
  $A^*(h_1)$ = 227 nodes
  $A^*(h_2)$ = 73 nodes
- $d=24$ IDS = too many nodes
  $A^*(h_1)$ = 39,135 nodes
  $A^*(h_2)$ = 1,641 nodes

With A*: a trade-off between quality of estimate and work per node!

Given any admissible heuristics h_a, h_b,

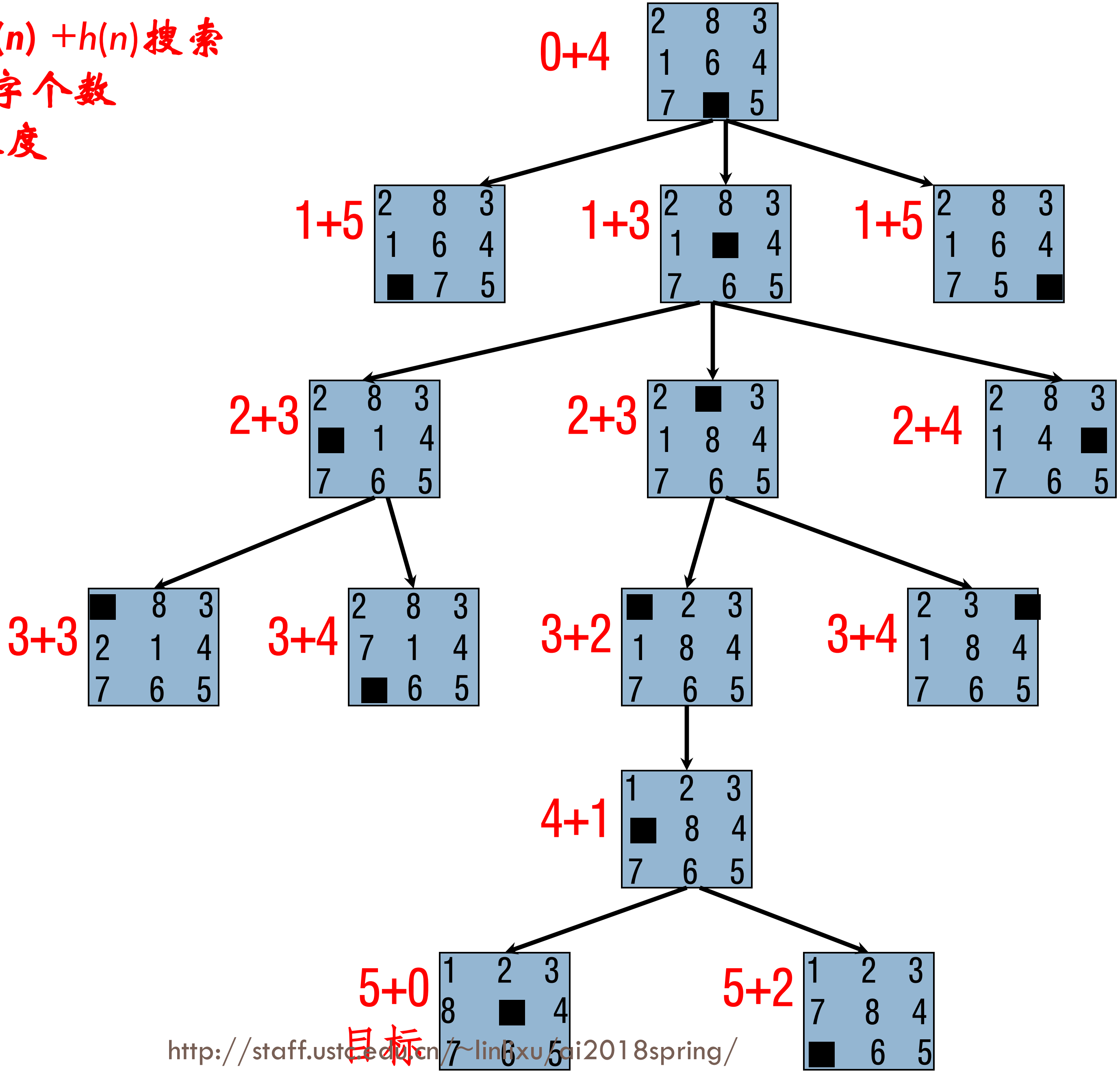
$$h(n) = max(h_a(n); h_b(n))$$

is also admissible and dominates h_a, h_b

使用 $f(n) = g(n) + h(n)$搜索

$h(n)$=错放数字个数

$g(n)$ = 节点深度



Most of the work is in coming up with admissible heuristics

How do we get good heuristics?

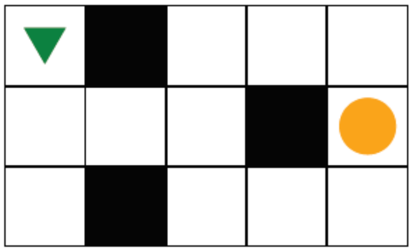
Just relax…

http://staff.ustc.edu.cn/~linlixu/ai2018spring/

Relaxation

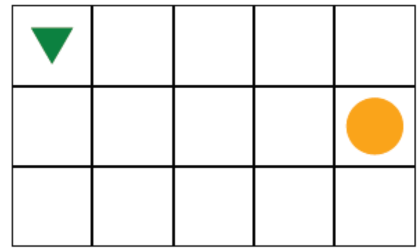
迷宫问题

Goal: move from triangle to circle



Hard

去掉障碍物



Easy

⇒用简化问题的最优解作为启发式函数，

Heuristic:

$$h(s) = \text{ManhattanDistance}(s, (2, 5))$$

- Interpretation: relax → removing constraints

Relaxed Problems

- A problem with fewer restrictions on the actions is called a relaxed problem （松弛问题）

- The cost of an optimal solution to a relaxed problem is an admissible heuristic for the original problem
  一个松弛问题的最优解的耗散是原问题的一个可采纳的启发式

- If the rules of the 8-puzzle are relaxed so that a tile can move anywhere, then $h_1(n)$ gives the shortest solution
  如果棋子可以任意移动，则h_1给出最短的确切步数

- If the rules are relaxed so that a tile can move to any adjacent square, then $h_2(n)$ gives the shortest solution
  如果棋子可以移动到任意相邻的位置，则h_2给出最短的确切步数

Key point: the optimal solution cost of a relaxed problem is no greater than the optimal solution cost of the real problem

Relaxed Problems

- 构造松弛问题
  - 原问题：一个棋子可以从方格A移动到方格B，如果A与B水平或者垂直相邻**而且**B是空的
  - 松弛1：一个棋子可以从方格A移动到方格B，如果A与B相邻 — h_2
  - 松弛2：一个棋子可以从方格A移动到方格B，如果B是空的
  - 松弛3：一个棋子可以从方格A移动到方格B —h_1

- 如果有一个可采纳启发式的集合$\{h_1, \cdots, h_m\}$

  $h(n) = max(h_1(n), \cdots, h_m(n))$

  可采纳并比成员启发式更有优势

Evaluation Function f(n)

h(n) — heuristic, estimate of cost from n to the closest goal
（节点n到目标节点的最低耗散路径的耗散估计值）

g(n) — path cost to n （初始节点到这个节点的路径损耗的总和）

Possible evaluation functions:

- f(n) = g(n) ≡ **Uniform Cost**
- f(n) = h(n) ≡ **Greedy**
- f(n) = g(n) + h(n) ≡ **A***

estimates the total cost of a solution path which goes through node n

Informed (Heuristic) Search

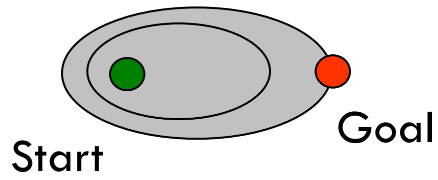
- Summary of uninformed tree (problem-solving) search:
  - Algorithms differ by method of *expansion, or choice of state-action* sequence for offline evaluation
  - Complexity tradeoffs, but poor (worst case or typical case) performance from all when state space is large
- Improvement:
  - Exploit knowledge about which successor states are preferable
  - "Best-First" search: expand nodes based on evaluation function $f(n)$
  - $f(n) \sim$ estimate of distance to goal
  - expand node n with lowest evaluation
  - "Best-First" algorithms are characterized by choice of evaluation function

Outline

- Best-first search（最佳优先搜索）
  - Greedy best-first search
  - A* search
- Heuristics
- Local search algorithms （局部搜索算法）
  - Hill-climbing search （爬山法搜索）
  - Simulated annealing search （模拟退火搜索）
  - Local beam search （局部剪枝搜索）
  - Genetic algorithms （遗传算法）

Large Scale Problems with A*

- What states get expanded?
  - All states with f-cost less than optimal goal cost



- In huge problems, often A* isn't enough
  - State space just too big
  - Can't visit all states with f less than optimal
  - Often, can't even store the entire fringe

∴ 解决方法则只考虑存储部分结点，而非完整路径

- Solutions
  - Better heuristics
  - Greedy hill-climbing (fringe size = 1)
  - Beam search (limited fringe size)

损失一定最优性，但能处理大规模问题

Limited Memory Options

- Bottleneck: not enough memory to store entire fringe
- Hill-Climbing Search:
  - Only "best" node kept around, no fringe! 只存一个结点 →但非常短视，只能看到当前的局部信息
  - Usually prioritize successor choice by h (greedy hill climbing)
  - Compare to greedy backtracking, which still has fringe

  启发式函数：反应当前状态离目标的好坏
- Beam Search (Limited Memory Search)
  - In between: keep K nodes in fringe 也由启发式函数选择留下哪k个结点
  - Dump lowest priority nodes as needed
  - Can prioritize by *h* alone (greedy beam search), or *h*+*g* (limited memory A*)

    按优先顺序列出
- No guarantees once you limit the fringe size!

Local Search Algorithms

很多问题只关心的，而不关心如何找到的（路径）： $\max_s f(s)$ s.t. $c(s) \geq d$

路径不重要

- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution

- State space = set of "complete" configurations（完全状态）
  - Find configuration satisfying constraints, e.g., n-queens

- In such cases, we can use local search algorithms
- keep a single "current" state, try to improve it
  - until you can't make it better

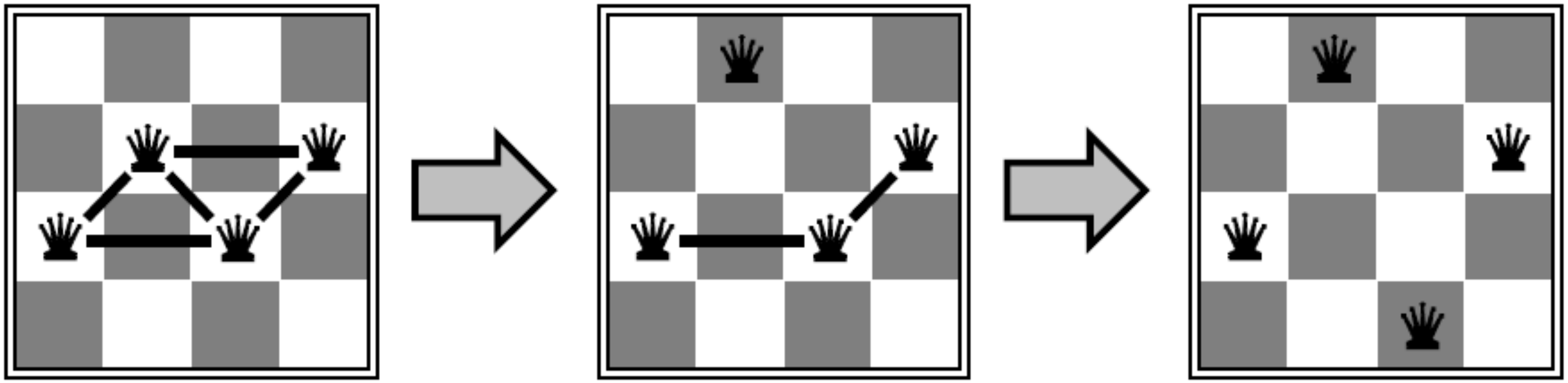
优点：

- Constant space, suitable for online as well as offline search

- Generally much more efficient (but incomplete)

Example: *n*-Queens

- Put n queens on an $n \times n$ board with no two queens on the same row, column, or diagonal



h = 5 h = 2 h = 0

把 h (存在冲突的边数) 作为代价，目标是 min 代价.

Optimization Problems

- Now a different setting:
  - Each state s has a score or cost, f(s), that we can compute
  - The goal is to find the state with the highest (or lowest) score, or a reasonably high (low) score
  - We do not care about the path
  - This is an optimization problem
  - 枚举所有状态
    Enumerating the states is intractable 棘手的
  - Previous search algorithms are too expensive
  - No known algorithm for finding optimal solution efficiently

Hill-Climbing Search

- Simple, general idea:
  - Start wherever
  - Always choose the best neighbor
  - If no neighbors have better scores than current, quit
- Why can this be a terrible idea?
  - Complete?
  - Optimal?
- What's good about it?

Hill-Climbing Search

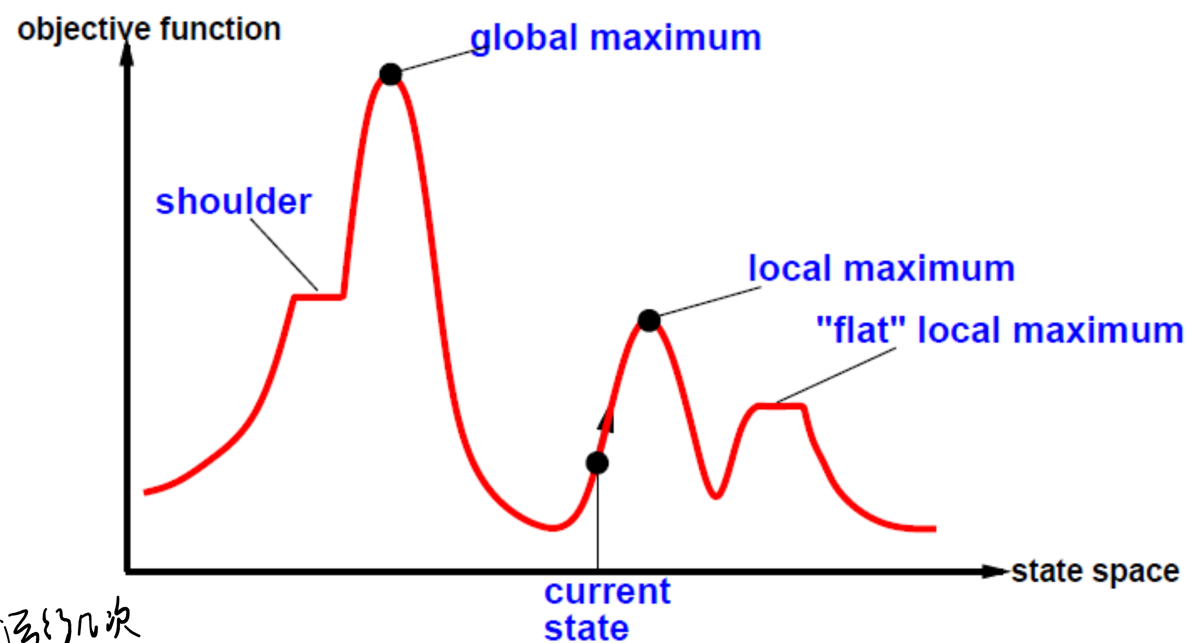
- “Like climbing Everest in thick fog with amnesia（健忘症）"
  - Can’t see the entire landscape all at once

```
function HILL-CLIMBING(problem) returns a state that is a local maximum
   inputs: 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 Search

- Problem: depending on initial state, can get stuck in local maxima（局部最大值）



多运行几次

非常完全

Random-restart hill climbing overcomes local maxima — trivially complete

Random sideways moves escape from shoulders ☺ loop on at maxima ☹

Hill-Climbing with Random Restarts

- Very simple modification:

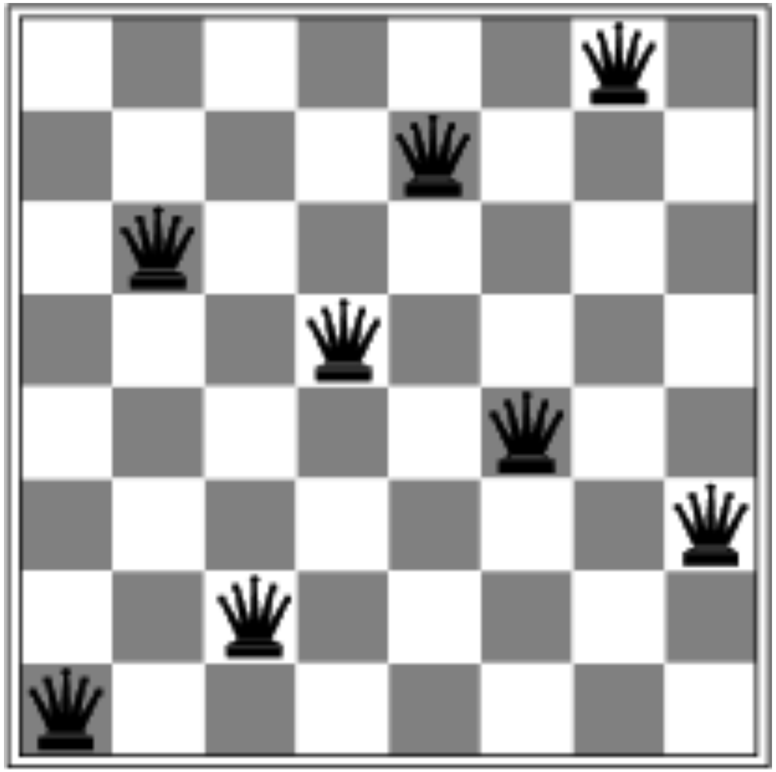
1. When stuck, pick a random new starting state and re-run hill-climbing from there
2. Repeat this *k* times
3. Return the best of the *k* local optima

- Can be very effective
- Should be tried whenever hill-climbing is used
- Fast, easy to implement; works well for many applications where the solution space surface is not too “bumpy” (i.e., not too many local maxima)

Hill-Climbing Search: 8-Queens Problem

- *h* = number of pairs of queens that are attacking each other, either directly or indirectly
- *h = 17* for the above state

Hill-Climbing Search: 8-Queens Problem



- A local minimum with $h = 1$

Life Lesson

Sometimes one needs to temporarily step backward in order to move forward

- Lesson applied to iterative, local search:

–Sometimes one needs to move to an *inferior neighbor* in order to escape a local optimum

Simulated Annealing Search （模拟退火搜索）

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

function SIMULATED-ANNEALING(*problem*, *schedule*) **returns** a solution state
  inputs: *problem*, a problem
    schedule, a mapping from time to "temperature"
  local variables: *current*, a node
    next, a node
    T, a "temperature" controlling prob. of downward steps

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

http://staff.ustc.edu.cn/~linlixu/ai2018spring/

Properties of Simulated Annealing Search

- One can prove: If *T* decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling, etc

Local Beam Search （局部剪枝搜索）

Keep track of *k* states rather than just one

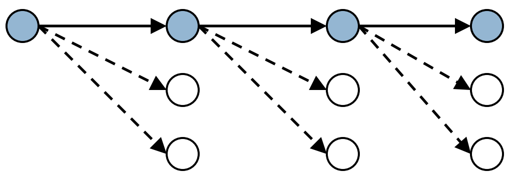
Start with *k* randomly generated states

At each iteration, all the successors of all *k* states are generated

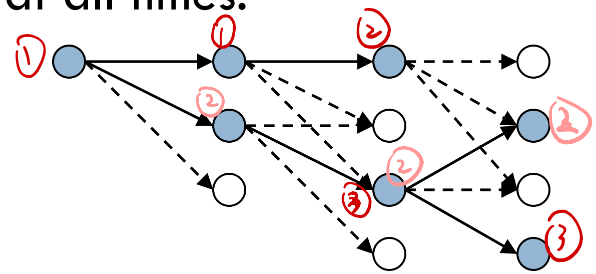
If any one is a goal state, stop; else select the *k* best successors from the complete list and repeat.

Local Beam Search

- Like greedy search, but keep K states at all times:



Greedy Search



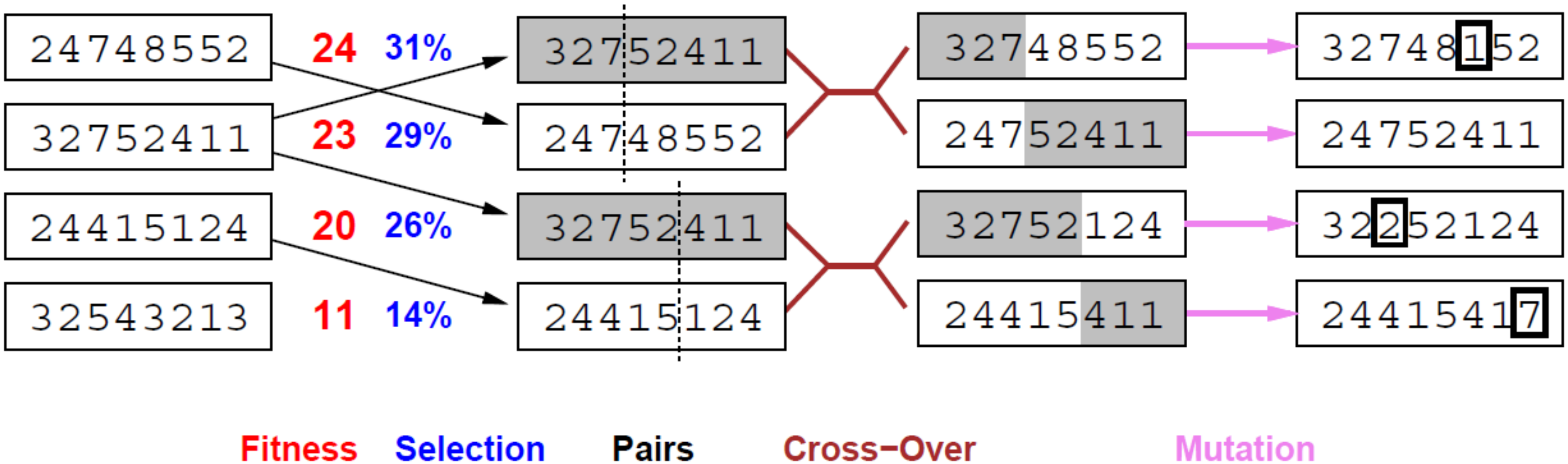
Beam Search

- The best choice in MANY practical settings
- Not the same as *k* searches run in parallel!
- Searches that find good states recruit 招募 other searches to join them
- Variables: beam size, encourage diversity?
- Problem: quite often, all *k* states end up on same local hill
- Idea: choose *k* successors randomly, biased towards good ones

Genetic Algorithms

- Genetic algorithms use a natural selection metaphor
- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population种群)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function适应度函数). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation（选择，杂交，变异）

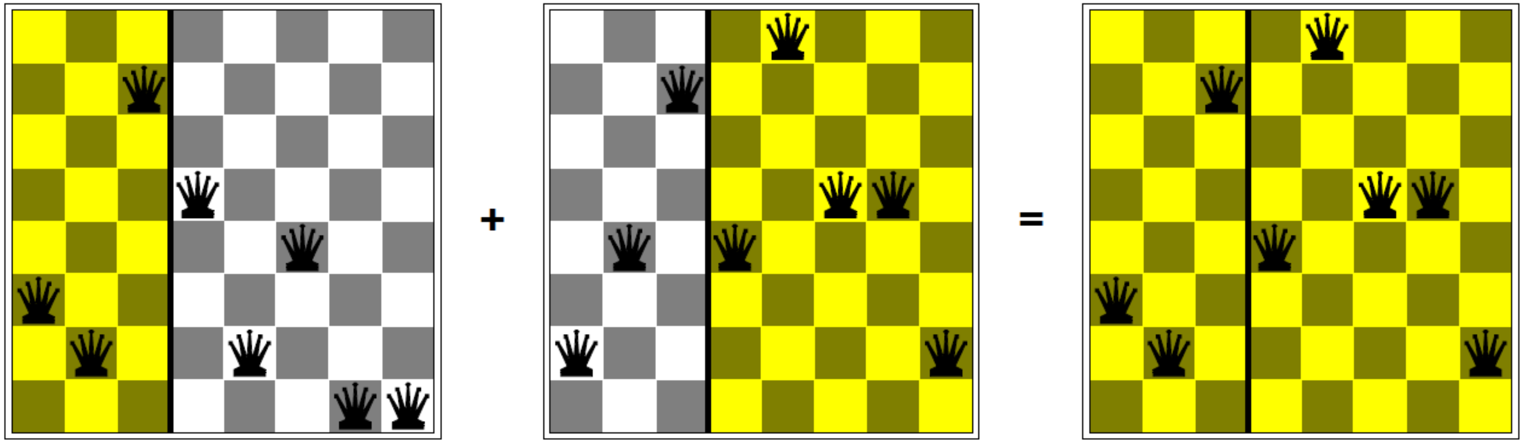
Genetic Algorithms



- Fitness function: number of non-attacking pairs of queens 不互相攻击的皇后对的数目 (min = 0, max = 8 × 7/2 = 28)

24/(24+23+20+11) = 31%
23/(24+23+20+11) = 29% etc

Genetic Algorithms



Summary 1

Heuristic functions estimate costs of shortest paths

Good heuristics can dramatically reduce search cost

Greedy best-first search expands lowest h

— incomplete and not always optimal

A* search expands lowest $g + h$

— complete and optimal

— also optimally efficient (up to tie-breaks, for forward search)

Admissible heuristics can be derived from exact solution of relaxed problems

Summary2

- Local search algorithms

  the path to the goal is irrelevant; the goal state itself is the solution

  keep a single "current" state, try to improve it

  - Hill-climbing search
    - depending on initial state, can get stuck in local maxima
  - Simulated annealing search
    - escape local maxima by allowing some "bad" moves but gradually decrease their frequency
  - Local beam search
    - Keep track of *k* states rather than just one
  - Genetic algorithms

Uninformed/Informed Search — Main points

- 好的启发式搜索能大大提高搜索性能

额外信息：如 从当前结点到目标结点的距离（h函数）

- 但由于启发式搜索需要抽取与问题本身有关的特征信息，而这种特征信息的抽取有时会比较困难，因此盲目搜索仍不失为一种有用的搜索策略。

Uninformed/Informed Search — Main points

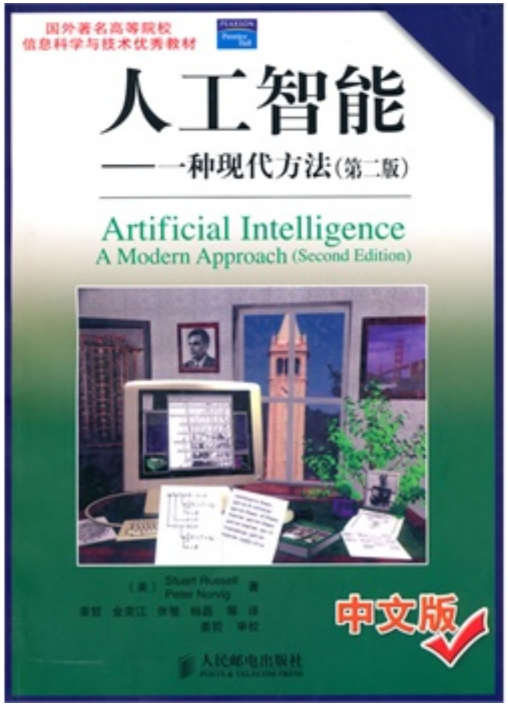
- 好的搜索策略应该
  - 引起运动—避免原地踏步 引发多样性
  - 系统—避免兜圈
  - 运用启发函数—缓解组合爆炸

Uninformed/Informed Search — Main points

- 搜索树 vs 搜索图
  - 搜索树：结点有重复，但登记过程简单
  - 搜索图：结点无重复，但登记过程复杂（每次都要查重）
    - 省空间，费时间。

作业

- 4.1, 4.2, 4.6, 4.7   3.23, 3.15, 3.28, 2.29
  - 习题编号按人民邮电出版社第二版



http://staff.ustc.edu.cn/~linlixu/ai2018spring/