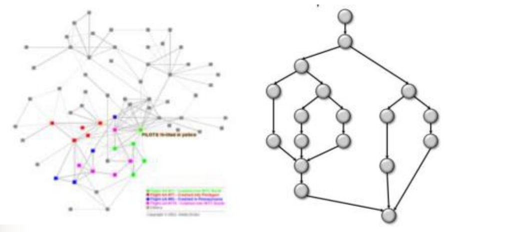
第三章 搜索与推理

主要内容：

■ 什么是搜索问题

已知初始状态和目标状态，求解一个行动序列使智能体从初始状态到目标状态。如果所求序列可以使得总耗散最低，则问题称为最优搜索问题。



■ 什么是搜索问题

起始状态：Arad

目标状态：Bucharest

状态空间的离散性：

--城市是离散的

环境的静态性：

--城市的相对位置不变

路径的耗散函数的确定性：

城市之间的距离是已知的

搜索问题：从Arad到Bucharest的路径

最优化搜索问题：从Arad到Bucharest的最短路径

起始状态

7	2	4
5		6
8	3	1

Start State

目标状态

1	2	3
4	5	6
7	8	

Goal State

华容道是不是一个搜索问题？

状态空间的离散性：
--8个格子排列方式是离散的

环境的静态性：
--九宫格大小和形状在格子移动过程中不变

合法行动与后继的确定性：
--只有空格四周的格子可动

路径的耗散函数的确定性：
--相邻两状态间所需步为1

搜索问题：从起始状态到目标状态的移动方法

最优化搜索问题：从起始状态到目标状态步骤最少的移动方法

起始状态：空的棋盘

目标状态：棋盘上摆了八个皇后，任意两个皇后都不能互相攻击。目标状态不确定，但是当前状态是否为目标状态是可以检测的。

状态空间的离散性：
--8个皇后在棋盘上摆放方式

环境的静态性：
--棋盘的格局和大小不变

合法行动与后继的确定性：
--满足棋盘上所有皇后不能互相攻击的后继才是合法的

路径的耗散函数的确定性：
--相邻两状态间所需步为1

搜索问题：求出（所有）合法的目标状态

§3.1 图搜索策略

■ 问题求解（Problem-solving）

➢ 表示（Problem formulation）

状态空间表示（State space representation）

➢ 搜索（Searching）

无信息搜索（Uninformed search）

有信息搜索（Informed search）

State Space Graph

State Space Graph

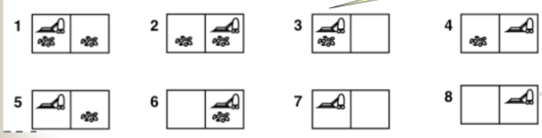
- State space graph: A mathematical representation of a search problem
  - Nodes are (abstracted) world configurations
  - Arcs represent transitions resulting from actions
  - The goal test is a set of goal nodes (maybe only one)
- In a state space graph, each state occurs only once!
- We can rarely build this full graph in memory (it's too big), but it's a useful idea

Example

例：真空吸尘器的世界

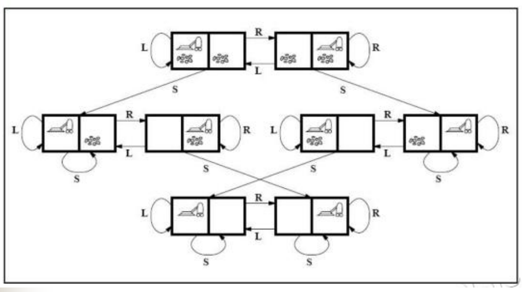
- 假设：吸尘器的世界只有两块地毯大小，地毯或者是脏的，或者是干净的
- 吸尘器能做的动作只有三个
  {向左 (*Left*)，向右 (*Right*)，吸尘 (*Suck*)}
- 一共有多少种可能的情况？

状态



Example

状态空间图



一般搜索策略

- 状态空间法的求解过程*转化为*在状态空间图中*搜索*一条从初始节点到目标节点的路径问题
- 图的搜索
  - 无信息搜索（盲目搜索）
    - 宽度优先搜索
    - 深度优先搜索
  - 有信息搜索（启发式搜索）
    - 贪婪算法，A算法
    - A*算法

图的一般搜索策略

搜索策略的分类

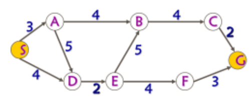
不同搜索策略的区别在于扩展节点的顺序

- 无信息搜索：无法知道当前状态离目标状态的"远近"或者不利用类似的先验信息来进行搜索的策略。
- 有信息搜索：利用启发式信息来进行搜索的策略。

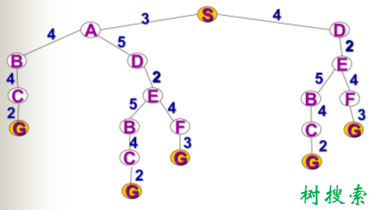
树搜索与图搜索

树搜索与图搜索

- 树是*无圈*连通图，每个节点只有一个父节点
- 树搜索不检查重复状态
- 图搜索大多比树搜索高效



图搜索



树搜索

无信息搜索

§3.2 无信息式搜索

不同搜索策略的搜索效率是不同的

- 无信息式搜索：不使用与问题有关的经验信息

  宽度优先搜索，深度优先搜索，有界深度优先搜索，等代价搜索，迭代加深搜索

- 特点：搜索过程中不使用与问题有关的经验信息

  搜索效率低，不适合大空间的实际问题求解

图的搜索过程

图的搜索过程

- 记住下一步还可以走哪些点

  OPEN表(记录还没有扩展的点)

- 记住哪些点走过了

  CLOSED表(记录已经扩展的点)

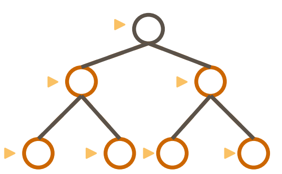
- 记住从目标返回的路径

每个表示状态的节点结构中必须有指向父节点的指针

BF-search

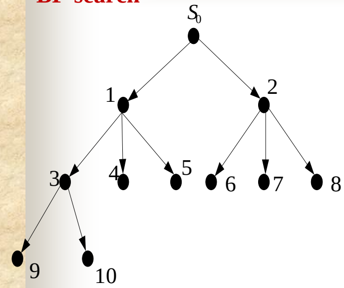
BF-search

- Expand *shallowest* unexpanded node
- Implementation:
  - *open* is a FIFO queue



BF-search

BF-search



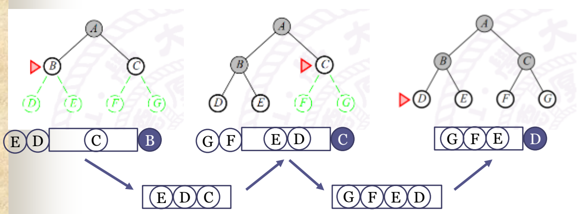
- Open表：已经生成出来但其子状态未被搜索的状态。
- Close表：记录了已被生成扩展过的状态。

宽度优先搜索法中状态的搜索次序

宽度优先搜索

宽度优先搜索

- 先被访问的节点先进行扩展
- 每次扩展深度最浅的节点
- 用一个先进先出的数据结构来保存待扩展节点序列



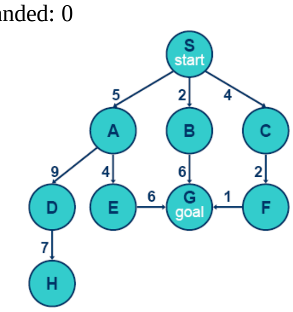
Example

Example

General Search (Problem, Queue)

# of nodes tested: 0, expanded: 0

Expnd. node	Open list
	{S}



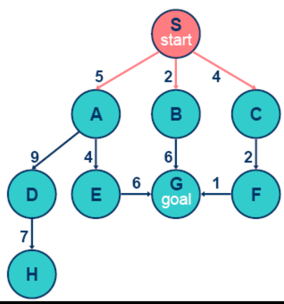
Example

Example

General Search (Problem, Queue)

# of nodes tested: 1, expanded: 1

Expnd. node	Open list
	{S}
S not goal	{A,B,C}



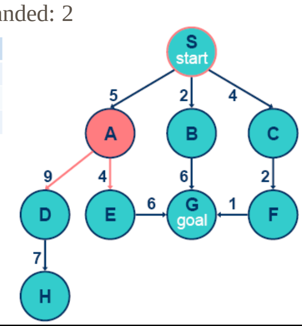
Example

Example

General Search (Problem, Queue)

# of nodes tested: 2, expanded: 2

Expnd. node	Open list
	{S}
S	{A,B,C}
A not goal	{B,C,D,E}



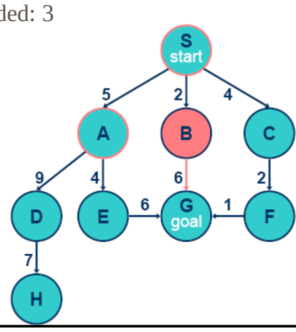
Example

Example

General Search (Problem, Queue)

# of nodes tested: 3, expanded: 3

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B not goal	{C,D,E,G}



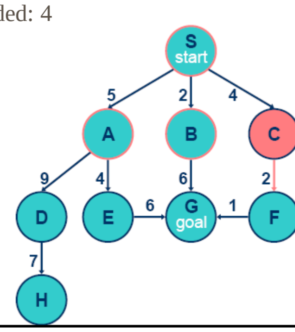
Example

Example

General Search (Problem, Queue)

# of nodes tested: 4, expanded: 4

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C not goal	{D,E,G,F}



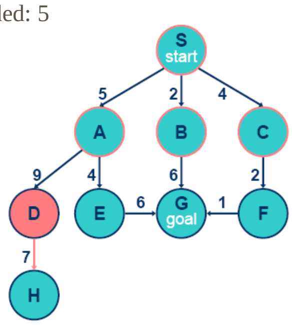
Example

Example

General Search (Problem, Queue)

# of nodes tested: 5, expanded: 5

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D not goal	{E,G,F,H }



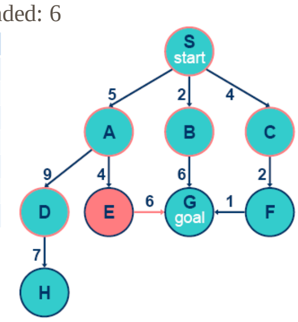
Example

Example

General Search (Problem, Queue)

# of nodes tested: 6, expanded: 6

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H }
E not goal	{G,F,H,G }

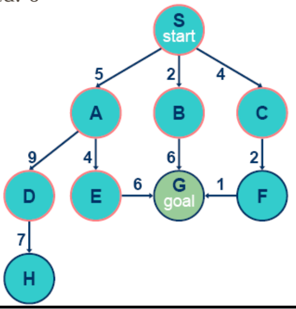


Example

General Search (Problem, Queue)

# of nodes tested: 7, expanded: 6

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H }
E	{G,F,H,G }
G goal	{F,H,G} no expand

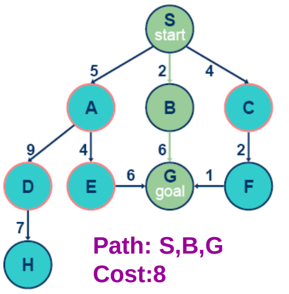


Example

General Search (Problem, Queue)

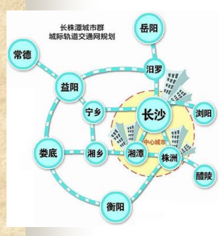
# of nodes tested: 7, expanded: 6

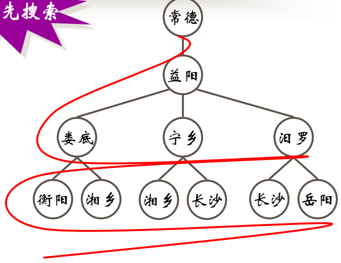
Expnd. node	Open list
	{S}
S	{A,B,C}
A	{B,C,D,E}
B	{C,D,E,G}
C	{D,E,G,F}
D	{E,G,F,H }
E	{G,F,H,G }
G	{F,H,G}



Path: S,B,G
Cost:8

宽度优先搜索





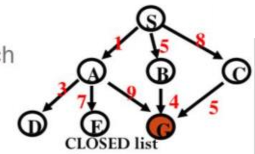
OPEN: 常德 益阳 娄底 宁乡 汨罗 衡阳 湘乡 长沙 岳阳

CLOSED:

OPEN 表中节点先进先出 (FIFO)—— 队列

Example

Breadth-First Search



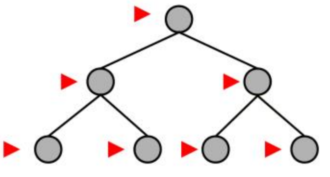
exp. node	OPEN list	CLOSED list
	{ S }	{}
S	{ A B C }	{S}
A	{ B C D E G }	{S A}
B	{ C D E G G' }	{S A B}
C	{ D E G G' G" }	{S A B C}
D	{ E G G' G" }	{S A B C D}
E	{ G G' G" }	{S A B C D E}
G	{ G' G" }	{S A B C D E}

Solution path found is S A G <-- this G also has cost 10

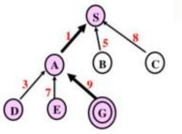
Number of nodes expanded (including goal node) = 7

DF-search

- Expand *deepest* unexpanded node
- Implementation: *open*
  - is a LIFO queue (=stack)

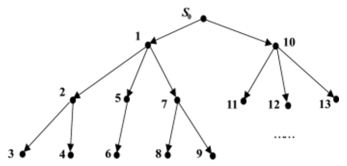


- 深度优先搜索
  - 后被访问的节点先进行扩展
  - 每次扩展深度最深的节点
  - “一条路走到黑”，对于无边界搜索问题无法保证完备性
  - 用后进先出的数据结构来保存待扩展节点序列



DF-search

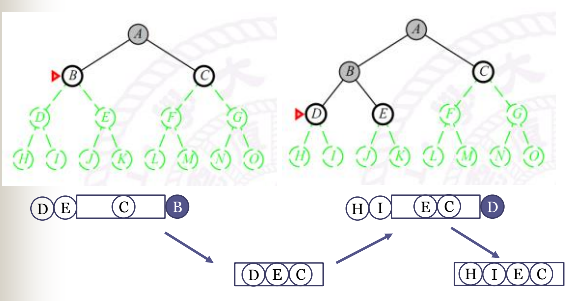
- 当搜索到某一个状态时，它所有的子状态以及子状态的后裔状态都须先于该状态的兄弟状态被搜索。
- 应选择合适的深度限制值，或采取不断加大深度限制值的办法，反复搜索，直到找到解。



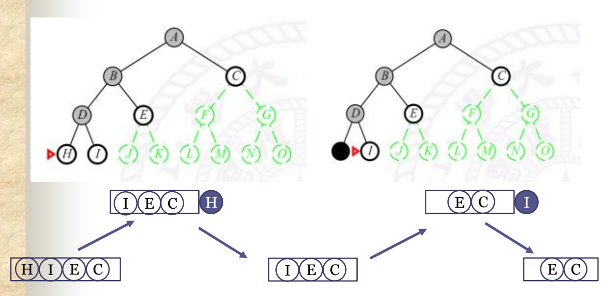
DF-search

- 不能保证第一次搜索到的某个状态时的路径是到这个状态的最短路径。
- 对任何状态而言，以后的搜索有可能找到另一条通向它的路径。如果路径的长度对解题很关键的话，当算法多次搜索到同一个状态时，它应该保留最短路径。

■ 深度优先搜索



■ 深度优先搜索

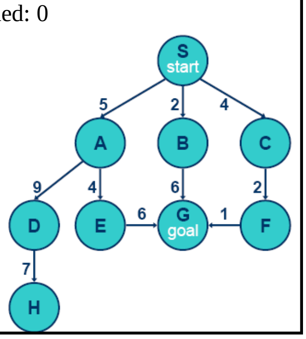


Example

General Search (problem, Stack)

# of nodes tested: 0, expanded: 0

Expnd. node	Open list
	{S}

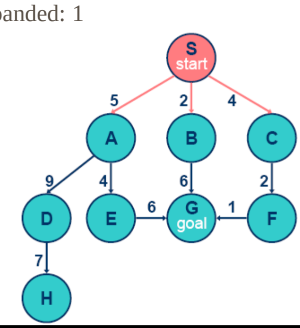


Example

General Search (problem, Stack)

# of nodes tested: 1, expanded: 1

Expnd. node	Open list
	{S}
S not goal	{A,B,C}



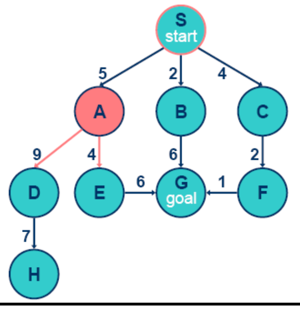
Example

Example

General Search (problem, Stack)

# of nodes tested: 2, expanded: 2

Expnd. node	Open list
	{S}
S	{A,B,C}
A not goal	{D,E,B,C}



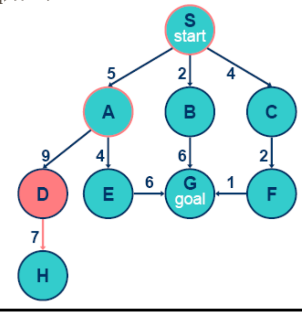
Example

Example

General Search (problem, Stack)

# of nodes tested: 3, expanded: 3

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D not goal	{H,D,E,B,C}



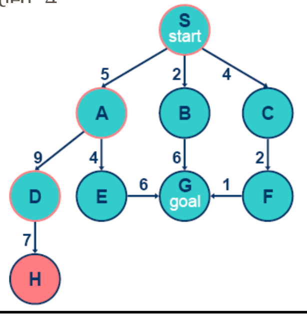
Example

Example

General Search (problem, Stack)

# of nodes tested: 4, expanded: 4

Expnd. node	Open list
	{S:0}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H not goal	{E,B,C}



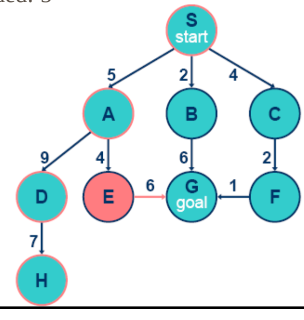
Example

Example

General Search (problem, Stack)

# of nodes tested: 5, expanded: 5

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E not goal	{G, B,C }



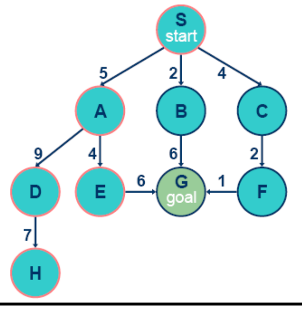
Example

Example

General Search (problem, Stack)

# of nodes tested: 6, expanded: 5

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G, B,C }
G goal	{B,C } no expand



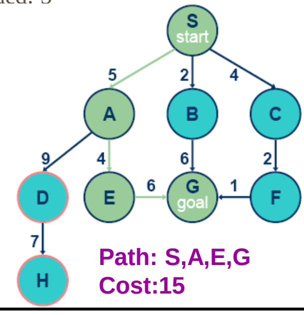
Example

Example

General Search (problem, Stack)

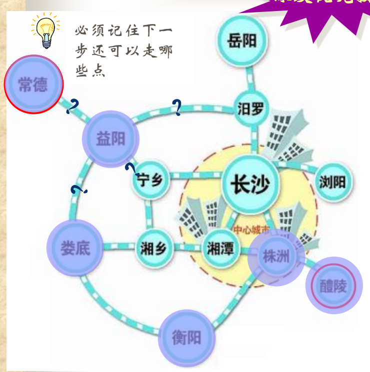
# of nodes tested: 6, expanded: 5

Expnd. node	Open list
	{S}
S	{A,B,C}
A	{D,E,B,C}
D	{H,E,B,C}
H	{E,B,C}
E	{G, B,C }
G	{B,C }



Path: S,A,E,G
Cost:15

深度优先搜索



状态：（城市名）

算子：常德→益阳

益阳→常德

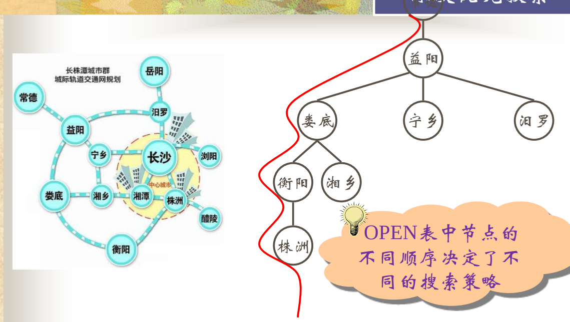
益阳↔汨罗

益阳↔宁乡

益阳↔娄底

…

深度优先搜索



OPEN: 常德 益阳 娄底 衡阳 株洲 湘乡 宁乡 汨罗

CLOSED:

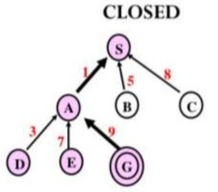
OPEN 表中节点后进先出 (LIFO)—— 栈 (Stack)

Depth-First Search

Depth-First Search

return GENERAL-SEARCH(problem, ENQUEUE-AT-FRONT)

exp. node	OPEN list	CLOSED list
	{ S }	
S	{ A B C }	
A	{ D E G B C }	
D	{ E G B C }	
E	{ G B C }	
G	{ B C }	



Solution path found is S A G <-- this G has cost 10

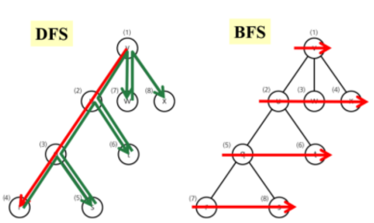
Number of nodes expanded (including goal node) = 5

深度优先搜索

- 深度优先搜索特点
  - 可能会选择一条分支并且沿着一条很长的（甚至是无限的）路径一直走下去
  - 对于无边界的搜索问题，可以通过对深度优先搜索提供一个预先设定的深度限制m来防止深度优先搜索进入死循环
  - 如果目标深度$d>$深度限制m，深度优先搜索可能无法得到解，因此完备性也无法保证

DFS & BFS

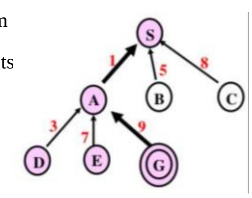
- DFS & BFS
  - When will BFS outperform DFS?
  - When will DFS outperform BFS?



Depth-limited search

Depth-limited search

- Is DF-search with depth limit l
  - i.e. nodes at depth l have no successors
  - Problem knowledge can be used
- Solves the infinite-path problem
- If $l<d$ then incompleteness results
- If $l>d$ then not optimal.



Example

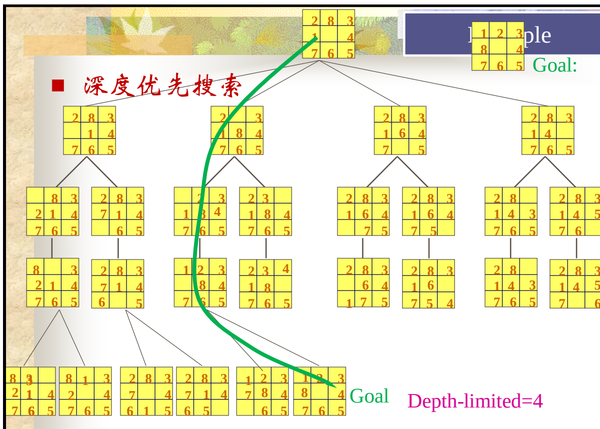
八数码难题

（初始状态） （目标状态）

操作：空格上移，空格下移，空格左移，空格右移

Example

■ 深度优先搜索



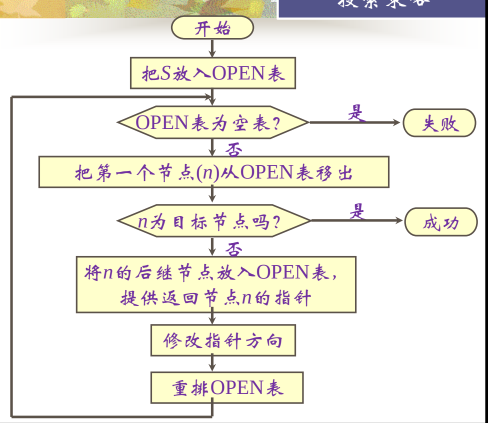
Uniform-cost search

Uniform-cost search

- Extension of BF-search:
  - Expand node with lowest path cost
- Implementation: *open*=queue ordered by path cost.
- UC-search is the same as BF-search when all step-costs are equal
- Dijkstra's algorithm, can be regarded as a variant of uniform-cost search, where there is no goal state and processing continues until all nodes have been removed from the priority queue.

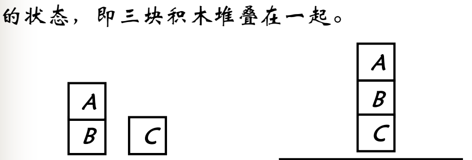
搜索策略

图的一般搜索策略（树搜索）



Example

举例：通过搬动积木块，希望从初始状态达到一个目的状态，即三块积木堆叠在一起。



(a) 初始状态 (b) 目的状态

积木问题

Example

- 操作算子为MOVE（X，Y）：把积木X搬到Y（积木或桌面）上面。

MOVE（A，Table）：“搬动积木A到桌面上”。

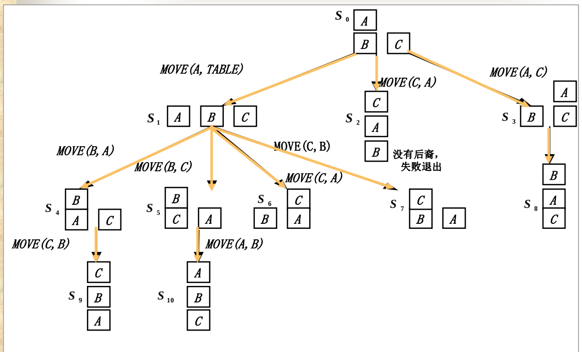
- 操作算子可运用的约束条件：

（1）被搬动积木的顶部必须为空。

（2）如果 Y 是积木，则积木 Y 的顶部也必须为空。

（3）同一状态下，运用操作算子的次数不得多于一次。

Example

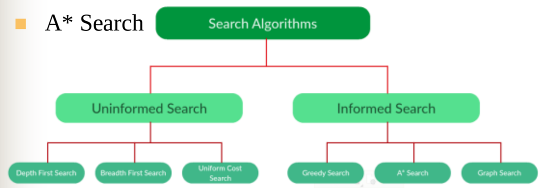


积木问题的宽度优先搜索树

启发式搜索

§3.3 有信息式搜索

- Outline
  - Heuristics
  - Greedy Search, A Search
  - A* Search



Tree Search

Review: Tree Search

```
function TREE-SEARCH(problem,fringe) return a solution or failure
    fringe ← INSERT(MAKE-NODE(INITIAL-STATE[problem]), fringe)
    loop do
        if EMPTY?(fringe) then return failure
        node ← REMOVE-FIRST(fringe)
        if GOAL-TEST[problem] applied to STATE[node] succeeds
            then return SOLUTION(node)
        fringe ← INSERT-ALL(EXPAND(node, problem), fringe)
```

A strategy is defined by picking *the order of node expansion.*

Best-First Search

- Best-First Search
  - General approach of informed search:

    -Best-first search: node is selected for expansion based on an *evaluation function*.
  - Idea: evaluation function measures distance to the goal

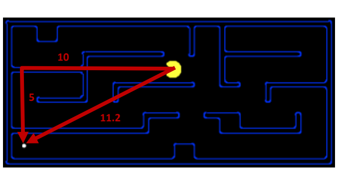
    -Choose node which *appears* best
  - Implementation:

    -Fringe is queue sorted in decreasing order of desirability

    -Special cases: Greedy search , A* search

Heuristics Function

- Heuristics Function
  - A function that *estimates* how close a state is to a goal
  - Designed for a particular search problem
  - Examples: Manhattan distance, Euclidean distance for pathing

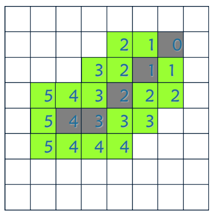


Heuristics Function

- Heuristics Function
  - It is denoted by $h(n)$.
  - $h(n)$ = estimated cost of the *cheapest* path from node n to a goal node
  - If n is goal then $h(n)=0$

Example

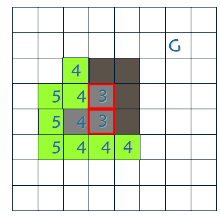
Example: Cost as True Distance



优先选择距离目标最近的点

Example

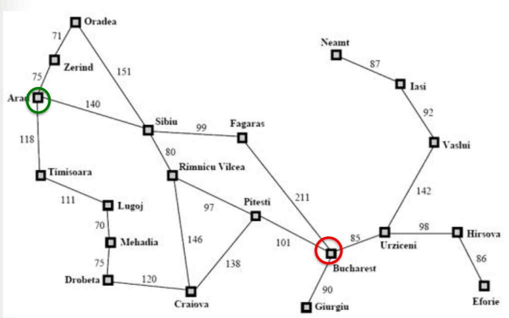
Example: Cost as True Distance (little change)



优先选择距离目标最近的点

Example

Example: Romania with Step Costs in km



Example

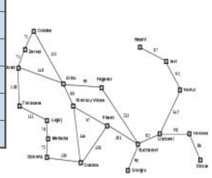
- On holiday in Romania; currently in Arad
  - Flight leaves tomorrow from Arad
- Formulate goal
  - Be in Bucharest
- Formulate problem
  - States: various cities
  - Actions: drive between cities
- Find solution
  - Sequence of cities; e.g. Arad, Sibiu, Fagaras, Bucharest, …

Example

Romania with Step Costs in km

City	SLD	City	SLD
Arad	366	Mehadia	241
Bucharest	0	Neamt	234
Craiova	160	Oradea	380
Dobreta	242	Pitesti	100
Eforie	161	Rimmicu vikea	193
Fagaras	176	Sibiu	253
Giurgiu	77	Timisoara	329
Hirsova	151	Urzicemi	80
Iasi	226	Vaslui	199
Lugoj	244	zerind	374

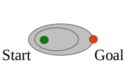
- h_{SLD}=Straight-line distance heuristic.
- h_{SLD} can **NOT** be computed from the problem description itself



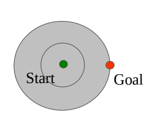
Effect of Heuristics

Effect of Heuristics

Guide search ***towards the goal*** instead of ***all over the place***



Informed



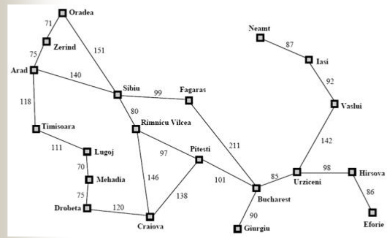
Uninformed

Greedy Search

Greedy Search

- We use a heuristic function
  - $f(n) = h(n)$
  - $h(n)$ estimates the distance remaining to a goal
- In greedy search, the idea is to expand the node with the *smallest* estimated cost to reach the goal.

Greedy search example



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

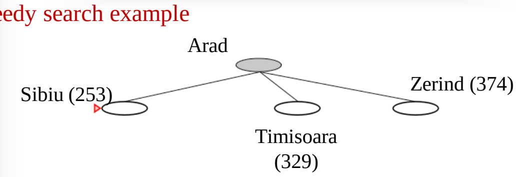
贪婪搜索

Greedy search example

- Assume that we want to use *greedy search* to solve the problem of travelling from Arad to Bucharest.
  - $f(n)=h(n)$
- The initial state=Arad

贪婪搜索

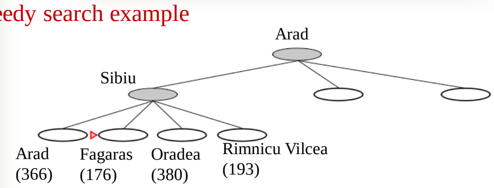
Greedy search example



- The first expansion step produces:
  - Sibiu, Timisoara and Zerind
- Greedy best-first will select *Sibiu.*

贪婪搜索

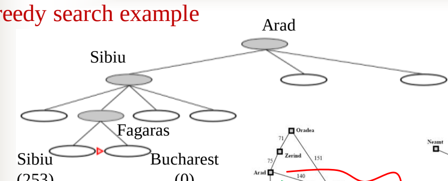
Greedy search example

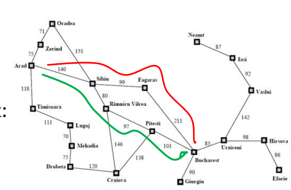


- If Sibiu is expanded we get:
  - Arad, Fagaras, Oradea and Rimnicu Vilcea
- Greedy best-first search will select *Fagaras*

贪婪搜索

Greedy search example





- If Fagaras is expanded we get:
  - Sibiu and *Bucharest*
- Goal reached !!
  - Yet not optimal (see Arad, Sibiu, Rimnicu Vilcea, Pitesti)

    450 vs 418

贪婪搜索

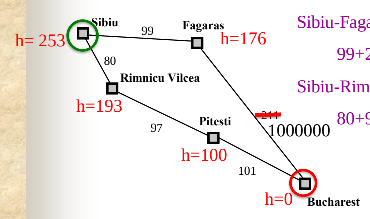
Greedy search, evaluation

- Often perform very well
- They tend to find good solutions quickly
- Not always optimal ones.

Greedy Search

Greedy Search

- Expand the node that seems closest…(order frontier by h)
- What can possibly go wrong?



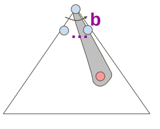
Sibiu-Fagaras-Bucharest =
99+211 = 310

Sibiu-Rimnicu Vilcea-Pitesti-Bucharest =
80+97+101 = 278

Greedy Search

Greedy Search

- Strategy: expand a node that ***seems*** closest to a goal state, according to h
- Problem 1: it chooses a node even if it's at the end of a very long and winding road
- Problem 2: it takes h literally even if it's completely wrong



A算法

A算法

Nils Nilsson 尼尔逊

Bertram Raphael 拉斐尔

- 尼尔逊提出一种算法以提高最短路径搜索的效率，被称为A1算法
- 拉斐尔改进了A1算法，称为A2算法

A算法

A算法

估价函数

$$f(x) = g(x) + h(x)$$

$g(x)$：从起始状态到当前状态x的代价

$h(x)$：从当前状态x到目标状态的估计代价（启发函数）

$f(x) = g(x)$ —UCS

$f(x) = h(x)$—贪婪算法

虽提高了算法效率，但不能保证找到最优解

A*算法

A*算法

Peter Hart

- 哈特对A算法进行修改，并证明当估价函数满足一定的限制条件时，算法一定可以找到最优解
- 估价函数满足一定限制条件的算法称为A*算法

A*算法的限制条件

$$f(x) = g(x) + h(x)$$

$g(x)$：大于0

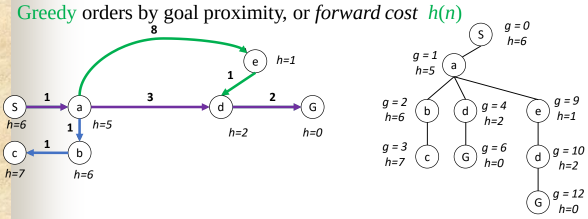
$h(x)$：不大于x到目标的实际代价

A*算法

■ Combining UCS and Greedy

Uniform-cost orders by path cost, or *backward cost* $g(n)$

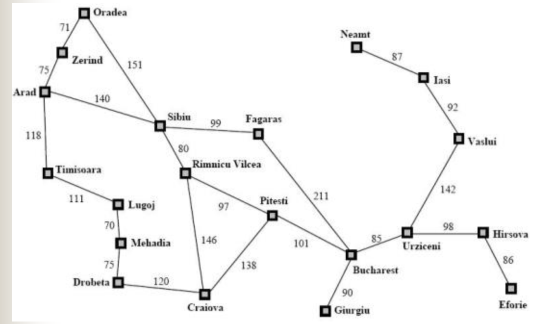
Greedy orders by goal proximity, or *forward cost* $h(n)$



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

Example

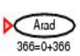
Example



A*算法

A* Search Example

(a) The initial state



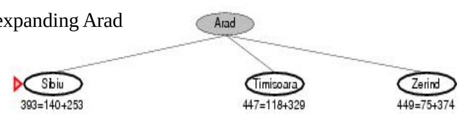
■ Find Bucharest starting at Arad

➢ f(Arad) = g(Arad)+h(Arad)=0+366=366

A*算法

A* Search Example

(b) After expanding Arad



■ Expand Arrad and determine *f(n)* for each node

➢ f(Sibiu)=g(Arad)+c(Arad, Sibiu)+h(Sibiu)=140+253=393

➢ f(Timisoara)= g(Arad)+ c(Arad, Timisoara)+h(Timisoara)=118+329=447

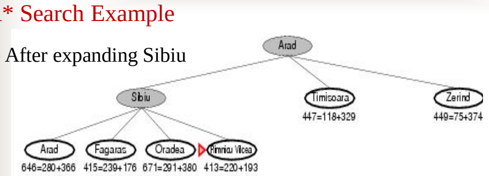
➢ f(Zerind)= g(Arad)+ c(Arad, Zerind)+h(Zerind)=75+374=449

■ Best choice is Sibiu

A*算法

A* Search Example

(c) After expanding Sibiu



■ Expand Sibiu and determine *f(n)* for each node

➢ f(Arad)=g(Sibiu)+c(Sibiu, Arad)+h(Arad)=280+366=646

➢ f(Fagaras)= g(Sibiu)+ c(Sibiu, Fagaras)+h(Fagaras)=239+176=415

➢ f(Oradea)= g(Sibiu)+ c(Sibiu,Oradea)+h(Oradea)=291+380=671

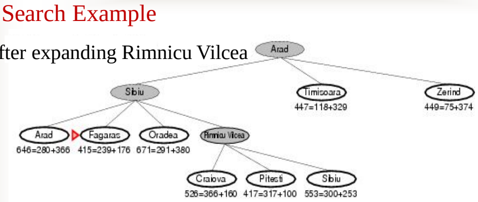
➢ f(Rimnicu Vilcea)= g(Sibiu)+ c(Sibiu,Rimnicu Vilcea)+ h(Rimnicu Vilcea)=220+192=413

■ Best choice is Rimnicu Vilcea

A*算法

A* Search Example

(d) After expanding Rimnicu Vilcea



■ Expand Rimnicu Vilcea and determine *f(n)* for each node

➢ f(Craiova)= g(Rimnicu Vilcea) +c(Rimnicu Vilcea, Craiova) +h(Craiova) =366 +160=526

➢ f(Pitesti)= g(Rimnicu Vilcea) +c(Rimnicu Vilcea, Pitesti)+ h(Pitesti) =317 +100 =417

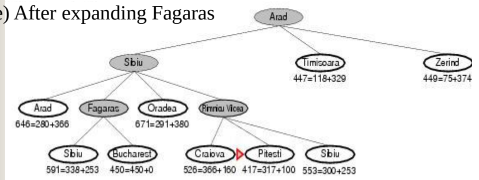
➢ f(Sibiu)= g(Rimnicu Vilcea) +c(Rimnicu Vilcea,Sibiu)+h(Sibiu)=300+253=553

■ Best choice is Pitesti

A*算法

A* Search Example

(e) After expanding Fagaras

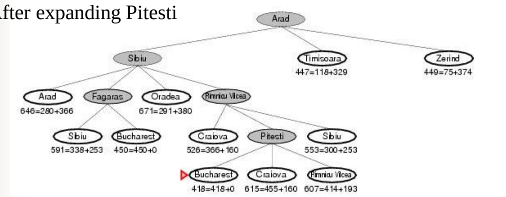


- Expand Fagaras and determine *f(n)* for each node
  - *f*(Sibiu)= *g*(Fagaras)+ *c*(Fagaras, Sibiu)+*h*(Sibiu)=338+253=591
  - *f*(Bucharest)= *g*(Fagaras)+ *c*(Fagaras,Bucharest)+*h*(Bucharest)=450+0=450
- Best choice is Pitesti !!!

A*算法

A* Search Example

(f) After expanding Pitesti



- Expand Pitesti and determine *f(n)* for each node
  - *f*(Bucharest)= *g*(Pitesti)+ *c*(Pitesti,Bucharest)+*h*(Bucharest) = 418+0=418
- Best choice is Bucharest !!!
  - Optimal solution (only if *h*(*n*) is admissible)

Example

Example

- 八数码问题
- 估价函数的定义
  - $f(x) = g(x) + h(x)$
  - $g(x)$:从初始状态到x需要进行的移动操作的次数
  - $h(x)$: x状态下错放的棋子数满足限制条件

谁更接近目标状态？

错放的棋子越少越好！

2	8	3
7	1	4
	6	5

$h(x)$=4

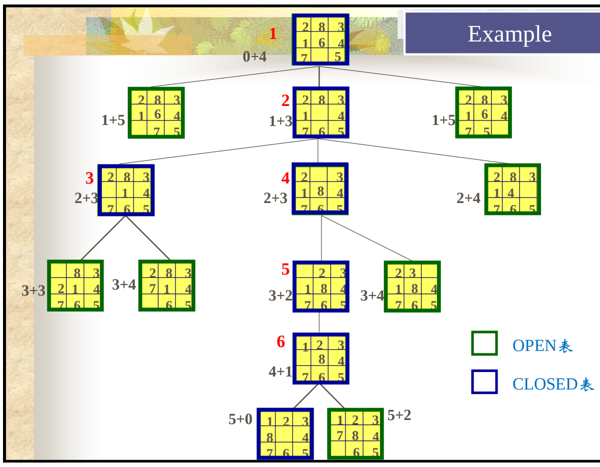
1	2	3
7	8	4
	6	5

$h(x)$=2

1	2	3
	8	4
7	6	5

$h(x)$=1

Example



估价函数

有信息搜索-估价函数

- 不同的估价函数定义对应于不同的算法

  $f(x) = g(x) + h(x)$

  - $h(x)$=0：UCS，非启发式算法
  - $g(x)$=0：贪婪搜索，无法保证找到解
  - 即使采用同样的形式 $f(x) = g(x) + h(x)$，不同的定义和不同的值也会影响搜索过程

估价函数

有信息搜索-估价函数对算法的影响

举例：八数码问题

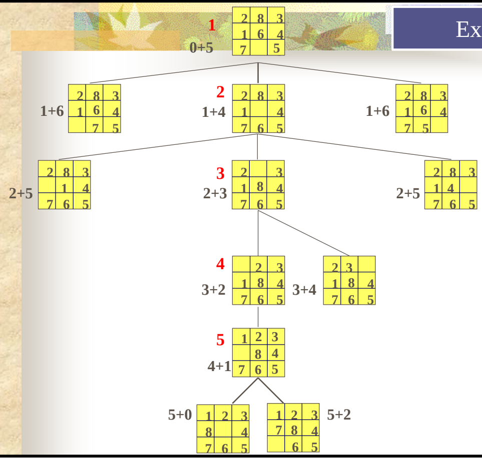
- $f(x) = g(x) + h(x)$
- $g(x)$：从初始状态到x需要进行的移动操作的次数
- $h(x)$：所有棋子与目标位置的曼哈顿距离之和

曼哈顿距离：两点之间水平距离和垂直距离之和仍满足估价函数的限制条件

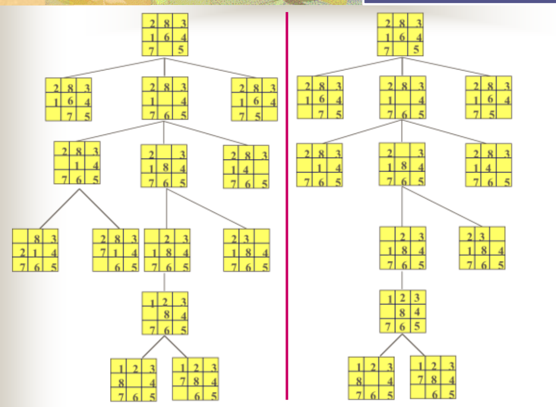
2	8	3
7	1	4
	6	5

$h(x)$=2+1+1+2=6

Example



Example



有信息搜索

有信息搜索- 占优

- If $h_2(n) \geq h_1(n)$ for all n (both admissible)

  then h_2 dominates h_1
- h_2 is better for search
- Typical search costs (average number of nodes expanded):
  - d=12

    $A^*(h_1)$ = 227 nodes，$A^*(h_2)$ = 73 nodes
  - d=24

    $A^*(h_1)$ = 39,135 nodes，$A^*(h_2)$ = 1,641 nodes

总结

总结

- 问题求解可分为问题表示和解的搜索两部分
  - 问题表示-状态空间图
  - 解搜索-图搜索策略
- 搜索类型
  - 无信息搜索，有信息搜索

    -通过启发函数引入启发式信息

    -通过估价函数预测下一步应选择的节点

消解原理

§ 3.4 消解原理

- What is reasoning?（什么是推理）
- Resolution（消解）

What is Reasoning

What is reasoning

- Ability and Process of making decision based on facts and knowledge.
  - *Mechanism*

    How to do reasoning theoretically?
  - *Control Strategy*

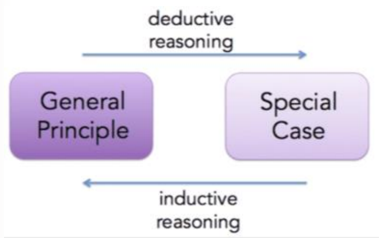
    How to realize the mechanism?

推理的分类

推理的分类

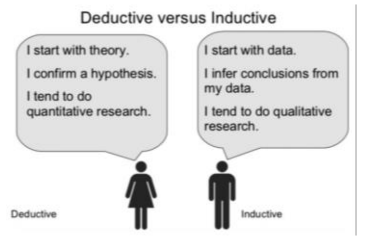
■ 根据逻辑基础

Deduction（演绎）vs. Induction（归纳）



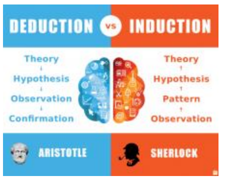
推理的分类

Deductive reasoning starts with a statement or hypothesis and then tests to see if it’s true through observation, where *inductive reasoning* starts with observations and moves backward towards generalizations and theories



Deduction

Deduction has theories that predict an outcome, which are tested by experiments.



Examples

> If A = B and B = C, then A = C.
>
> Deduction

> Since all squares are rectangles, and all rectangles have four sides, so all squares have four sides.
>
> Deduction

Induction

Induction makes observations that lead to generalizations for how that thing works.

Examples

> I have a bag of many coins, and I’ve pulled 10 at random and they’ve all been pennies, therefore this is probably a bag full of pennies.
>
> Induction

课堂练习

课堂练习

三段论式（三段论法）

- 足球运动员的身体都是强壮的；
- 高波是一名足球运动员；
- 所以，高波的身体是强壮的。

演绎推理：一般 → 个别

课堂练习

课堂练习

检查全部产品合格 —完全归纳推理→ 该厂产品合格

检查全部样品合格 —不完全归纳推理→ 该厂产品合格

归纳推理：个别 → 一般

- 完全归纳推理（必然性推理）
- 不完全归纳推理（非必然性推理）

确定性推理

推理的分类

- 根据知识的确定性

Reasoning under certainty（确定）vs. uncertainty（不确定）

确定性推理：推理时所用的知识与证据都是确定的，推出的结论也是确定的，其真值或者为真或者为假

不确定性推理

不确定性推理：推理时所用的知识与证据不都是确定的，推出的结论也是不确定的。

不确定性推理
- 似然推理（概率论）
- 近似推理或模糊推理（模糊逻辑）

Uncertainty

- Hard for an agent to handle
- Usually handled statistically
- Statistical reasoning requires knowledge of probability

Control Strategies

Control Strategies in Reasoning

- Inference Direction

Facts → Conclusions (Forward chain, Data-driven)
Facts ← Conclusions (Backward chain, Goal-driven)
Facts ↔ Conclusions (Bi-directional)

→ Forward Chaining	← Backward Chaining
• Data driven • When: If all facts available up front. • If → then	• Goal Driven • When: there are many attributes employed in many rules (e.g diagnostic problems) • Then → If

Example

推理是如何进行的？

- 推理过程多种多样
- Example
  - 如果今天不下雨，我就去你家
  - 今天没有下雨
- Example
  - 小王说他下午或者去图书馆或者在家休息
  - 小王没去图书馆
- 计算机如何选择？

消解原理

消解原理

- 美国数学家鲁滨逊提出消解原理
- 基本原理：要证明一个命题为真都可以通过证明其否命题为假来得到
- 将多样的推理规则简化为一个——消解

鲁滨逊

Example

Example 1

- 小王说他下午或者去图书馆或者在家休息
- 小王没去图书馆

R—小王下午去图书馆
S—小王下午在家休息

$$\left.\begin{array}{l} R \vee S \\ \sim R \end{array}\right\} \Rightarrow S$$

Example 2

- 如果今天不下雨，我就去你家 $\sim P \rightarrow Q \Leftrightarrow P \vee Q$
- 今天没有下雨 $\sim P$

消解原理

P ∨ Q　　~ P ∨ R　　亲本子句

Q ∨ R　　消解式

消解式是亲本子句的逻辑结论

- 消解只在仅含否定和析取联接词的公式（子句）间进行
- 先把公式化成规范的形式（范式，子句集）

含变量消解

含变量的消解

Example：苏格拉底论断

凡人都会死 $(\forall x)$ (Man $(x) \Rightarrow$ Mortal (x))

苏格拉底是人 Man (Socrates)

如何得到结论：苏格拉底会死 Mortal (Socrates)

含变量消解

含变量的消解

- 要完成消解还面临几个问题
  - “⇒” 和 “∀” 必须去掉

    Man $(x) \Rightarrow$ Mortal (x) $\Leftrightarrow$ ~ Man $(x) \vee$ Mortal

    “∀”怎么办？

    化为子句集
  - 如果能去掉“ ”~ Man (x) 和Man(Socrates)也不能构成互补对，形式不一样，怎么办？ 置换与合一

文字 & 子句

Terminology

- Literal（文字）
  - Atomic sentences and their negation
  - E.g. P , ~P, Q(x, y), ~R(f(x, y), z)
- Clause（子句）
  - Disjunction of literals (文字的析取)
  - E.g. P(x)∨Q(x) , ~ R(x，f(y))∨S(x,g(x))

文字 & 子句

Terminology

- Conjunctive Normal Form（CNF, 合取范式）
  - Conjunction of clauses
- Clause Set（子句集）--合取关系
  - { P(x)∨Q(x) , ~R(x, f(y))∨S(x, g(x))}
  - Equals: (P(x)∨Q(x)) ∧ (~ R(x, f(y))∨S(x, g(x)))

消解过程

消解过程

- 当消解可使用时，消解过程被应用于母体子句对，以便产生一个导出子句。
- 如果存在某个公理$E_1 \vee E_2$和另一公理$\sim E_2 \vee E_3$，那么$E_1 \vee E_3$在逻辑上成立，这就是*消解*，而称$E_1 \vee E_3$为$E_1 \vee E_2$和$\sim E_2 \vee E_3$的消解式。

子句集的步骤

（1）消去蕴涵符号

$A \Rightarrow B \rightarrow \sim A \vee B$

（2）减少否定符号的辖域

$\sim(A \wedge B) \rightarrow \sim A \vee \sim B$

$\sim(A \vee B) \rightarrow \sim A \wedge \sim B$

$\sim(\sim A) \rightarrow A$

$\exists x(\sim A) \rightarrow \sim \forall x(A)$

$\forall x(\sim A) \rightarrow \sim \exists x(A)$

子句集的步骤

（3）对变量标准化

$\forall x\{P(x)(\exists x)Q(x)\} \rightarrow \forall x\{P(x)(\exists y)Q(y)\}$

（4）消去存在量词

$\forall y[(\exists x)P(x,y)] \rightarrow \forall y[P(g(y),y)]$

规则：以一个Skolem函数代替每个出现的存在量词的量化变量。如果消去的存在量词不在任何一个全称量词的辖域内，那么就用不含变量的Skolem函数替换。

子句集的步骤

（5）化为前束型

$$\text{前束型}=\underbrace{\text{（前缀）}}_{\text{全称量词串}}\quad\underbrace{\text{（母式）}}_{\text{无量词公式}}$$

（6）把母式化为合取范式

$A \vee (B \wedge C) \rightarrow \{A \vee B\} \wedge \{A \vee C\}$

子句集的步骤

（7）消去全称量词

（8）消去连词符号∧

（9）更换变量名称

子句集的求取（共9步）

将下列谓词公式化为一个子句集

$$(\forall x)\{P(x) \Rightarrow \{(\forall y)[\underline{P(y) \Rightarrow P(f(x,y))}] \wedge \sim(\forall y)[\underline{Q(x,y) \Rightarrow P(y)}]\}\}$$

(1) 消去蕴涵符号

只应用∨和～符号，以～A∨B替换A ⇒ B。

(1) $(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \wedge \sim(\forall y)[\sim Q(x, y) \vee P(y)]\}\}$

(2) 减少否定的辖域范围

每个否定符号只用到一个谓词符号上。

(1) $(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \wedge \sim(\forall y)[\sim Q(x, y) \vee P(y)]\}\}$

⇩

(2) $(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \wedge (\exists y)[Q(x, y) \wedge \sim P(y)]\}\}$

⇩

(3) $(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge (\exists \mathbf{w})[Q(x,\mathbf{w}) \wedge \sim P(\mathbf{w})]\}\}$

(3) 变量标准化

对变量改名，以保证每个量词有其自己唯一的变量符号。

子句集的求取

(4) 消去存在量词

情况1："∃" 在"∀"的辖域范围内

- 使用Skolem函数替换

$\forall x\, \exists y\, Height(x, y) \Rightarrow \forall x\, Height(x, f(x))$

- Skolem函数$f(x)$表明了y与x之间的依赖或映射关系

$\forall x\, \forall y\, \exists z\, P(x, y, z) \Rightarrow \forall x\, \forall y\, P(x, y, f(x, y))$

情况2："∃" 不在"∀"的辖域范围内

- 使用常量替换

$\exists x\, \forall y\, P(x, y) \Rightarrow \forall y\, P(A, y)$

(3) $(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x,y))] \wedge (\exists w)[Q(x,w) \wedge \sim P(w)]\}\}$

$\Downarrow$ $w = g(x)$为一Skolem函数

(4) $(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \wedge [Q(x, g(x)) \wedge \sim P(g(x))]\}\}$

子句集的求取

(5) 化为前束形

前束形={前缀} {母式}

全称量词串 无量词公式

(6) 母式化合取范式 (7) 消去全程量词

(4) $(\forall x)\{\sim P(x) \vee \{(\forall y)[\sim P(y) \vee P(f(x, y))] \wedge [Q(x, g(x)) \wedge \sim P(g(x))]\}\}$

$\Downarrow$

(5) $(\forall x)(\forall y)\{\sim P(x) \vee \{[\sim P(y) \vee P(f(x, y))] \wedge [Q(x, g(x)) \wedge \sim P(g(x))]\}\}$

$\Downarrow$

(6) $(\forall x)(\forall y)\{[\sim P(x) \vee \sim P(y) \vee P(f(x, y))] \wedge [\sim P(x) \vee Q(x, g(x))] \wedge [\sim P(x) \vee \sim P(g(x))]\}$

$\Downarrow$

(7) $\{[\sim P(x) \vee \sim P(y) \vee P(f(x, y))] \wedge [\sim P(x) \vee Q(x, g(x))] \wedge [\sim P(x) \vee \sim P(g(x))]\}$

子句集的求取

(8) 消去连词符号∧

用{A, B}代替(A∧B)，消去符号∧，得到一个有限集，其中每个公式是文字的析取（子句）

(9) 更换变量名

使一个变量符号不出现在一个以上的子句中。

(7) $\{[\sim P(x) \vee \sim P(y) \vee P(f(x, y))] \wedge [\sim P(x) \vee Q(x, g(x))] \wedge [\sim P(x) \vee \sim P(g(x))]\}$

$\Downarrow$

(8) $\{\sim P(x) \vee \sim P(y) \vee P(f(x, y)),\ \sim P(x) \vee Q(x, g(x)),\ \sim P(x) \vee \sim P(g(x))\}$

$\Rightarrow$

(9) $\{\sim P(x1) \vee \sim P(y) \vee P(f(x1, y)),\ \sim P(x2) \vee Q(x2, g(x2)),\ \sim P(x3) \vee \sim P(g(x3))\}$

子句集的求取

课堂练习：求子句集

"Everyone who loves all animals is loved by someone."

$\forall x\, \{[\forall y\, (Animal(y) \Rightarrow Loves(x, y))] \Rightarrow \exists y\, Loves(y, x)\}$

子句集的求取

Answer

$\forall x\, \{[\forall y\, (Animal(y) \Rightarrow Loves(x,y))] \Rightarrow \exists y\, Loves(y,x)\}$

1. 消去蕴含符合 $\forall x\, (\sim \forall y\, (\sim Animal(y) \vee Loves(x, y)) \vee \exists y\, Loves(y, x))$

2. 减小否定辖域范围 $\forall x\, (\exists y \sim(\sim Animal(y) \vee Loves(x, y)) \vee \exists y\, Loves(y, x))$

$\forall x\, (\exists y\, (Animal(y) \wedge \sim Loves(x,y)) \vee \exists y\, Loves(y, x))$

3. 变量标准化 $\forall x\, (\exists y\, (Animal(y) \wedge \sim Loves(x, y)) \vee \exists \mathbf{z}\, Loves(\mathbf{z}, x))$

4. 消去存在量词 $\forall x\, ((Animal(\mathbf{f(x)}) \wedge \sim Loves(x, \mathbf{f(x)})) \vee Loves(\mathbf{g(x)}, x))$

子句集的求取

$\forall x\, ((Animal(f(x)) \wedge \sim Loves(x, f(x))) \vee Loves(g(x), x))$

5. 化为合取范式 $\forall x\, ((Animal(f(x)) \vee Loves(g(x),x)) \wedge (\sim Loves(x, f(x)) \vee Loves(g(x),x)))$

6. 消去全称量词

$(Animal(f(x)) \vee Loves(g(x), x)) \wedge (\sim Loves(x, f(x)) \vee Loves(g(x), x))$

7. 消去连词符号∧，变子句集

$\{ Animal(f(x)) \vee Loves(g(x), x),\ \sim Loves(x, f(x)) \vee Loves(g(x), x) \}$

8. 变量标准化

(1) $Animal(f(x1)) \vee Loves(g(x1), x1)$

(2) $\sim Loves(x2, f(x2)) \vee Loves(g(x2), x2)$

2.消解推理规则

■ 消解式的定义

- 令L_1, L_2为两任意原子公式；L_1和L_2具有相同的谓词符号，但一般具有不同的变量。
- 已知两子句$L_1 \vee \alpha$和$\sim L_2 \vee \beta$,如果L_1和L_2具有最一般合一者σ，那么通过消解可以从这两个父辈子句推导出一个新子句$(\alpha \vee \beta)\sigma$。这个新子句叫做消解式。

证明

Resolution:

$$\left.\begin{array}{l} C_1 = L \vee \alpha \\ C_2 = \sim L \vee \beta \end{array}\right\} \Rightarrow \quad C_{12} = \alpha \vee \beta$$

Proof:

$\because C_1 = \alpha \vee L \Leftrightarrow \sim \alpha \Rightarrow L$ 且,$C_2 = \sim L \vee \beta \Leftrightarrow L \Rightarrow \beta$

$\therefore C_1 \wedge C_2 \Leftrightarrow (\sim \alpha \Rightarrow L) \wedge (L \Rightarrow \beta)$

$[(\sim \alpha \Rightarrow L) \wedge (L \Rightarrow \beta)] \Rightarrow [\sim \alpha \Rightarrow \beta]$

$\sim \alpha \Rightarrow \beta \quad \Leftrightarrow \quad \alpha \vee \beta = C_{12}$

$\therefore C_{12} = C_1 \wedge C_2$

■ 消解式例子

(a) 空子句 NIL Clause

父辈子句 P ~P → NIL

(b) 假言推理 Modus ponens

P ~P∨Q (P⇒Q) → Q

(c) 合并 Combination

P∨Q ~P∨Q → Q∨Q= Q

(d) 重言式 Tautologies

P∨Q ~P∨~Q → Q∨~Q

P∨Q ~P∨~Q → P∨~P

(e) 链式（三段论） Chain

~P∨Q ~Q∨R → ~P∨R

3.含有变量的消解

■ 含有变量的子句的消解

要把消解推理规则推广到含有变量的子句，必须找到一个作用于父辈子句的置换，使父辈子句含有互补文字。

■ Example

P[x,f(y)]∨Q(x)∨R[f(a),y] ~P[f (f(a)),z]∨R(z,w)

σ ={f(f(a))/x, f(y)/z}

Q [f (f(a))] ∨R(f(a)，y) ∨R(f(y)，w)

基本思想

把要解决的问题作为一个要证明的命题，其目标公式被否定化并化成子句型，然后添加到命题公式集中去，把消解反演推理规则应用于命题公式集，并推导出一个空子句，产生一个矛盾，这说明目标公式的否定式不能成立，即目标公式成立，问题得到解决。

反证法

消解反演求解过程

消解反演 Resolution Refutation

消解反演证明

给出公式集{S}和目标公式L

- 否定L，得～L；
- 把～L添加到S中去；
- 把新产生的集合 T = ｛～L，S｝化成子句集；
- 应用消解原理，力图推导出一个表示矛盾的空子句

Example

Example 1

- 设事实的公式集合 { P,(P∧Q)⇒ R (S∨T) ⇒ Q，T }，

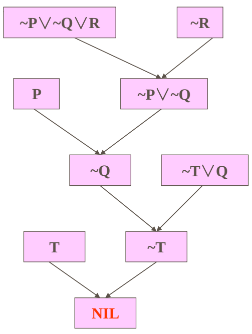
证明： R

- 否定结论，将公式化为子句，

得子句集：

- { P，～P∨～Q∨R，～S∨Q，～T∨Q，T，～R }

消解反演树



Example

Example 2

- "Happy Student"：Everyone who pass the computer test and win the prize is happy. Everyone who wish study or is lucky can pass all tests. Zhang doesn't study, but he is lucky. Every lucky person can win the prize.
- Prove：Zhang is happy

Example

- Solution
  - Step1: first-order logic representation of the problem

Facts or Knowledge:

（∀x）(Pass (x, computer)∧Win (x, prize)) ⇒ Happy (x)

（∀x）（∀y）(Study(x) ∨Lucky(x) ⇒ Pass (x, y))

~Study (zhang)∧ Lucky (zhang)

（∀x）(Lucky(x) ⇒ Win (x, prize))

Negation of the conclusion: ~Happy（zhang）

Example

Step2: Convert the sentence above into clauses

（1）~Pass(x, computer)∨ ~Win(x, prize)∨ Happy(x)

（2）~Study(y)∨ Pass(y, z)

（3）~Lucky(u)∨ Pass(u, v)

（4） ~Study(zhang)

（5） Lucky(zhang)

（6）~Lucky(w)∨Win(w, prize)

（7）~Happy(zhang)

Example

Step3: Resolve these clauses

(1)~Pass(x,computer)∨~Win(x,prize)∨Happy(x)   (7)~Happy(zhang)

{zhang/x}

~Pass(zhang,computer)∨~Win(zhang,prize)   (6)~Lucky(w)∨Win(w,prize)

{zhang/w}

(5) Lucky(zhang)   ~Pass(zhang,computer)∨~Lucky(zhang)

~Pass(zhang,computer)   (3) ~Lucky(u)∨Pass(u,v)

{zhang/u,computer/v}

~Lucky(zhang)   (5) Lucky(zhang)

NIL

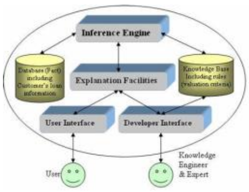
产生式系统

§3.5 产生式系统

- **Definition：**
  - Also called Rule-based System.
  - Proposed by Post in 1943. *DENDRAL*
  - Describes several different things that based on a same basic concept.

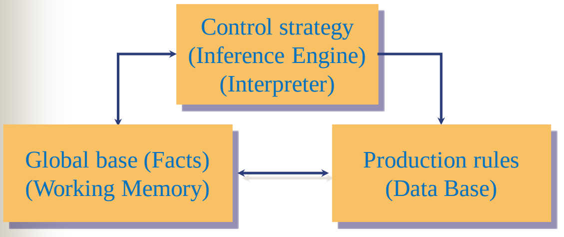
产生式系统

- **Essential：** Knowledge separated 2 parts
  - Facts represented static knowledge, Exp. object, event and relation between them;
  - Production rules represented inference process and action.



Production System

Architecture of Production-rule System



Architecture

- 总数据库
  - 又称综合数据库、上下文、黑板等
  - 存放求解过程中当前信息的数据，如：问题的初始状态、事实或证据、中间推理结论和结果等。
- 产生式规则（规则库）
  - 存放于求解问题相关的知识的规则集合
  - 完整性、一致性、准确性、灵活性和合理性
- 控制策略（推理机）
  - 由一组程序组成，用来控制产生式系统的运行

Architecture

- Matching

Current database is matched with rule condition.

- Conflict resolution

When more than one rule matched with current database, it should decide which rule is used firstly, which is called conflict resolution.

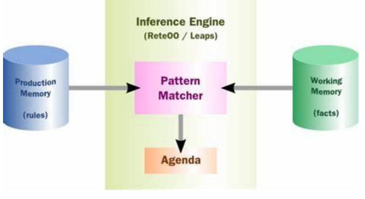
- Operation

Operation means execution of the rule's operation parts

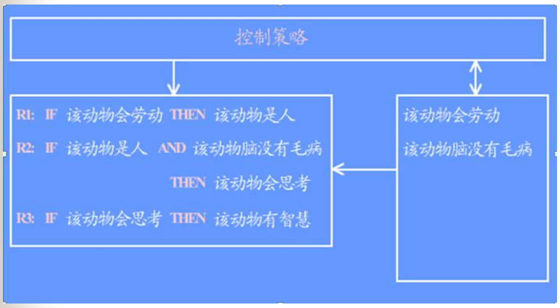
Inference

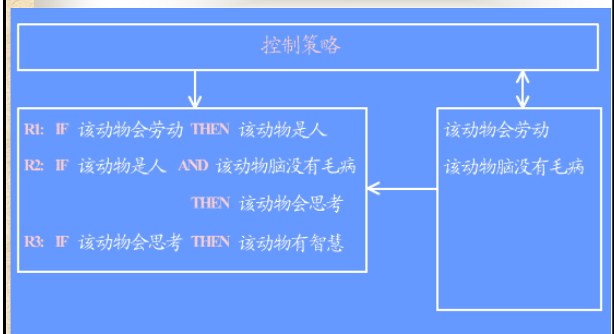
Inference of Production-rule System

- Forward Inference
- Backward Inference
- Bidirectional Inference



Example





An example of production system

- You want a program that can answer questions and make inferences about food items
- Like:
  - What is purple and perishable?
  - What is packed in small containers?
  - What is green and weighs 5 kg?

A production rule system making inferences about food

WORKING MEMORY (WM)
Initially WM = (green, weighs-5-kg)

RULE BASE
R1: **IF** green **THEN** produce
R2: **IF** packed-in-small-container **THEN** delicacy
R3: **IF** [refrigerated **OR** produce] **THEN** perishable
R4: **IF** [weighs-5-kg **AND** inexpensive **AND NOT** perishable] **THEN** staple
R5: **IF** [weighs-5-kg **AND** produce] **THEN** watermelon

INTERPRETER
1. Find all productions whose condition parts are true
2. Deactivate productions that would add a duplicate symbol
3. Execute the lowest numbered production (or quit)
4. Repeat until there is no rule to execute

First cycle of execution

WORKING MEMORY
WM = (green, weighs-5-kg)

CYCLE 1
1. Productions whose condition parts are true: ***R1***
2. No production would add duplicate symbol
3. Execute ***R1***.
   This gives: WM = (***produce,*** green, weighs-5-kg)

RULE BASE
R1: **IF** green **THEN** produce
R2: **IF** packed-in-small-container **THEN** delicacy
R3: **IF** [refrigerated **OR** produce] **THEN** perishable
R4: **IF** [weighs-5-kg **AND** inexpensive **AND NOT** perishable] **THEN** staple
R5: **IF** [weighs-5-kg **AND** produce] **THEN** watermelon

INTERPRETER
1. Find all productions whose condition parts are true
2. Deactivate productions that would add a duplicate symbol
3. Execute the lowest numbered production (or quit)
4. Repeat

Second cycle of execution

WORKING MEMORY
WM = (produce, green, weighs-5-kg)

CYCLE 2
1. Productions whose condition parts are true: ***R1, R3, R5***
2. Production R1 would add duplicate symbol, so ***deactivate R1***
3. Execute ***R3*** because it is the lowest numbered production.
   This gives: WM = (***perishable,*** produce, green, weighs-5-kg)

RULE BASE
R1: **IF** green **THEN** produce
R2: **IF** packed-in-small-container **THEN** delicacy
R3: **IF** [refrigerated **OR** produce] **THEN** perishable
R4: **IF** [weighs-5-kg **AND** inexpensive **AND NOT** perishable] **THEN** staple
R5: **IF** [weighs-5-kg **AND** produce] **THEN** watermelon

INTERPRETER
1. Find all productions whose condition parts are true
2. Deactivate productions that would add a duplicate symbol
3. Execute the lowest numbered production (or quit)
4. Repeat

Third cycle of execution

Example

WORKING MEMORY

WM = (perishable, produce, green, weighs-5-kg)

CYCLE 3

1. Productions whose condition parts are true: ***R1, R3, R5***
2. Productions ***R1, R3*** would add duplicate symbol, so deactivate them
3. Execute ***R5***.

This gives: WM = (***watermelon***, perishable, produce, green, weighs-5-kg)

RULE BASE

R1: **IF** green **THEN** produce
R2: **IF** packed-in-small-container **THEN** delicacy
R3: **IF** [refrigerated **OR** produce] **THEN** perishable
R4: **IF** [weighs-5-kg **AND** inexpensive **AND NOT** perishable] **THEN** staple
R5: **IF** [weighs-5-kg **AND** produce] **THEN** watermelon

INTERPRETER

1. Find all productions whose condition parts are true
2. Deactivate productions that would add a duplicate symbol
3. Execute the lowest numbered production (or quit)
4. Repeat

Fourth cycle of execution

Example

WORKING MEMORY

WM = (watermelon, perishable, produce, green, weighs-5-kg)

CYCLE 4

1. Productions whose condition parts are true: ***R1, R3, R5***
2. Productions ***R1, R3, R5*** would add duplicate symbol, so ***deactivate them***
3. ***Quit***.

What are the conclusions?

RULE BASE

R1: **IF** green **THEN** produce
R2: **IF** packed-in-small-container **THEN** delicacy
R3: **IF** [refrigerated **OR** produce] **THEN** perishable
R4: **IF** [weighs-5-kg **AND** inexpensive **AND NOT** perishable] **THEN** staple
R5: **IF** [weighs-5-kg **AND** produce] **THEN** watermelon

INTERPRETER

1. Find all productions whose condition parts are true
2. Deactivate productions that would add a duplicate symbol
3. Execute the lowest numbered production (or quit)
4. Repeat