1.1.4 启发式搜索

1. 问题的提出

2. 启发性信息

按其用途划分, 启发性信息可分为以下三类：

(1) 用于扩展节点的选择, 即用于决定应先扩展哪一个节点, 以免盲目扩展。

(2) 用于生成节点的选择,即用于决定应生成哪些后续节点,以免盲目地生成过多无用节点。

(3) 用于删除节点的选择,即用于决定应删除哪些无用节点, 以免造成进一步的时空浪费。

3.启发函数

启发函数是用来估计搜索树上节点x与目标节点S_g接近程度的一种函数, 通常记为$h(x)$。

4.启发式搜索算法

1) 全局择优搜索

2) 局部择优搜索

全局择优搜索算法：

步1 把初始节点S_0放入$OPEN$表中,计算$h(S_0)$。

步2 若$OPEN$表为空,则搜索失败, 退出。

步3 移出$OPEN$表中第一个节点N放入CLOSED表中, 并冠以序号n。

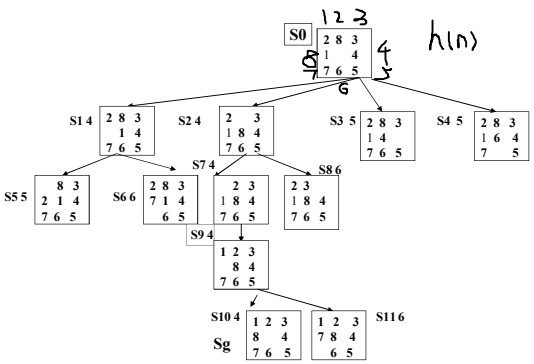
步4 若目标节点$S_g=N$, 则搜索成功, 结束。

步5 若N不可扩展, 则转步2。

步6 扩展N, 计算每个子节点x的函数值$h(x)$, 并将所有子节点配以指向N的返回指针后放入$OPEN$表中, 再对$OPEN$表中的所有子节点按其函数值大小以升序排序,转步2。

例 1.5 用全局择优搜索法解八数码难题。初始棋局和目标棋局同例3。

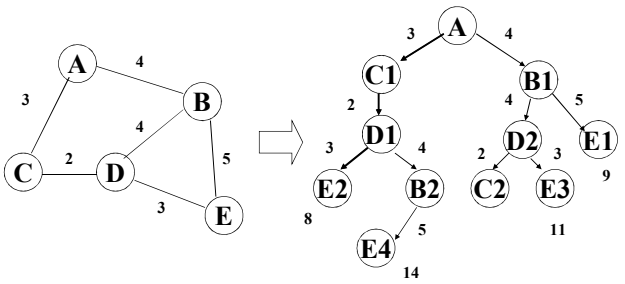
解 设启发函数$h(x)$为节点x的格局与目标格局相比数码不同的位置个数。以这个函数指导的搜索树如下图所示。此八数问题的解为：S_0, S_1, S_2, S_3, S_g。



代价函数g(x)

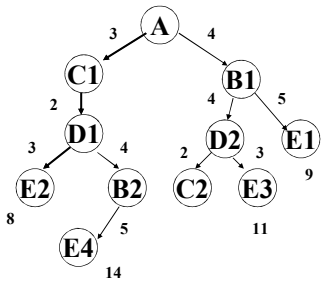
节点的代价：g(xj) = g(xj)+c(xi,xj)

g(S0)=0



分支界限法（最小代价优先法）

最近择优法（瞎子爬山法）



A→C1→D1→E2

1.1.5 A算法和A*算法

估价函数（ evaluation function ）

总是希望：找到最有希望通向目标节点的待扩展节点优先扩展

估价函数

用于度量节点的“希望”的量度。f(n)

此节点在通向目标节点的最佳路径上的“希望”

定义估价函数的原则：

一个节点处在最佳路径上的概率；

求出任意一个节点与目标节点集之间的距离度量或差异度量；

根据格局（博弈问题）或状态的特点来打分。

1 A算法

基本思想：

定义一个估价函数f，对当前的搜索状态进行评估，找出一个最有希望的节点来扩展

估价函数：

$f(n) = g(n) + h(n)$，n为被评价节点

$g^*(n)$：从s到n的最优路径的实际代价

$h^*(n)$：从n到g的最优路径的实际代价

$f^*(n)=g^*(n)+h^*(n)$：从s经过n到g的最优路径的实际代价

$g(n)$、$h(n)$、$f(n)$分别是$g^*(n)$、$h^*(n)$、$f^*(n)$的估计值

$g(n)$通常为从S到到n这段路径的实际代价，

则有$g(n) \geq g^*(n)$

$h(n)$:是从节点n到目标节点Sg的最优路径的估计代价. 它的选择依赖于

有关问题领域的启发信息，叫做启发函数

A算法：在图搜索的一般算法中，在搜索的每一步都利用估价函数$f(n)= g(n)+h(n)$对Open表中的节点进行排序表中的节点进行排序, 找出一个最有希望的节点作为下一次扩展的节点

(1) 把起始节点*S*放到**OPEN**表中，计算*f*(*S*)，并把其值与节点*S*联系起来.

(2) 如果**OPEN**表是个空表，则失败退出，无解.

(3) 从**OPEN**表中选择一个*f*值最小的节点*i*。 结果有几个节点合格，当其中有一个为目标节点时，则选择此目标节点，否则就选择其中任一个节点作为节点*i*.

(4) 把节点*i* 从**OPEN**表中移出, 并把它放入**CLOSED**的扩展节点表中.

(5) 如果*i*是个目标节点, 则成功退出, 求得一个解.

(6) 扩展节点*i*, 生成其全部后继节点. 对于*i*的每一个后继节点*j*:

a) 计算*f*(*j*).

b) 如果*j*既不在**OPEN**表中, 也不在**CLOSED**表中, 则用估价函数*f* 把它添入**OPEN**表. 从*j*加一指向父辈节点*i*的指针。

c) 如果*j*已在**OPEN**表或**CLOSED**表上, 则比较刚刚对*j*计算过的*f*值和前面计算过的该节点在表中的*f*值. 如果新的*f*值较小, 则

I. 以此新值取代旧值.

II. 从*j*指向*i*, 而不是指向它的父辈节点

III. 如果节点*j*在**CLOSED**表中, 则把它移回**OPEN**表

(7) 转向**(2)**,即**GOTO(2);**

2 最佳图搜索算法A*（A*算法）

在A算法中，如果：h(n)≤h*(n)，则A算法称为A*算法。

当定义的启发函数：h（n）是h*（n）的下界

例如在八数码难题中，定义估价函数为:

$$f(n)=d(n)+W(n)$$

其中, $d(n)$表示节点n在搜索树中的深度;

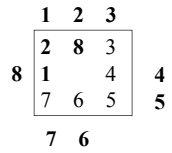
$W(n)$ 表示节点n中”**不在位**”的数码个数.

举例

8数码难题

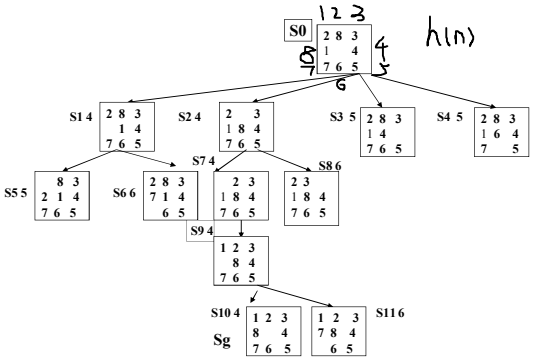
算法1：h1(n) = “不在位”的牌数

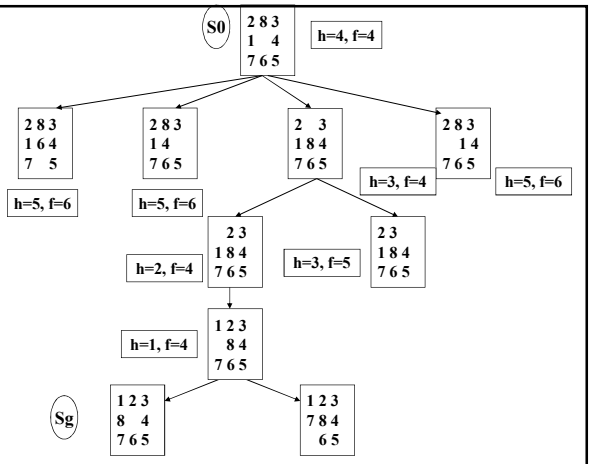
算法2：h2(n) = 牌“不在位”的距离和



牌1：1
牌2：1
牌8：2

h1(n) ≤h2(n)





1.2 与或图搜索

1.2.1 与或图

例 如下图所示,设有四边形$ABCD$和$A'B'C'D'$, 要求证明它们全等。

四边形$ABCD$和$A'B'C'D'$

分析：分别连接B、D和B'、D', 则原问题可分解为两个子问题：

$Q1$：证明

$\triangle ABD \cong \triangle A'B'D'$

$Q2$：证明

$\triangle BCD \cong \triangle B'C'D'$

于是, 原问题的解决可归结为这两个子问题的解决。换句话说,原问题被解决当且仅当这两个子问题都被解决。

进一步,问题Q_1还可再被分解为

Q_{11}：证明$AB = A'B'$

Q_{12}：证明$AD = A'D'$

Q_{13}：证明$\angle A = \angle A'$

或

Q_{11}'： 证明$AB = A'B'$

Q_{12}'： 证明$AD = A'D'$

Q_{13}'： 证明 $BD = B'D'$

问题Q_2还可再被分解为

Q_{21}：证明 $BC = B'C'$

Q_{22}：证明 $CD = C'D'$

Q_{23}：证明 $\angle C = \angle C'$

或

Q_{21}'：证明 $BC = B'C'$

Q_{22}'：证明 $CD = C'D'$

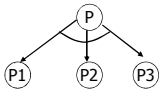
Q_{23}'：证明 $BD = B'D'$

问题的分解与变换

1. 与图: 把一个原问题**分解**为若干个子问题，P1，P2，P3，…可用“与图”表示；

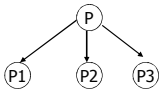
P1，P2，P3，…对应的子问题节点称为“与节点”。

例3：设P可分解为三个子问题P1，P2，P3的与，则P和P1，P2，P3之间的关系可用“与图”表示：



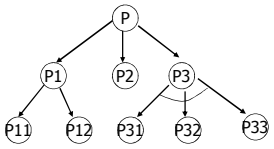
2. 或图: 把一个原问题**变换**为若干个子问题，P1，P2，P3，…
可用“或图”表示；
P1，P2，P3，…对应的子问题节点称为“或节点”。

例3：设P可变换为三个子问题P1，P2，P3的或，则P和P1，P2，P3之间的关系可用“或图”表示：



52

3. 与或图: 如果一个原问题即**需要通过分解又需要通过变换**才能得到其本原问题，则其归约过程可用一个“与或图”来表示。



53

4. 端节点和终叶节点

在与或树中，没有子节点的节点称为**端节点**；

本原问题所对应的节点称为**终叶节点**。(终止节点)

终叶节点一定是端节点，但端节点却不一定是终叶节点。

54

5. 可解节点和不可解节点

可解节点：

在与或树中，满足以下三个条件之一的节点为可解节点：

（1）任何终叶节点都是可解节点

（2）对“或”节点，当其子节点中至少有一个为可解节点时，则该“或”节点就是可解节点。

（3）对“与”节点，只有当其子节点全部为可解节点时，该“与”节点才是可解节点。

55

5. 可解节点和不可解节点

不可解节点：

在与或树中，满足以下三个条件之一的节点为不可解节点：

（1）不为终叶节点的端节点是不可解节点。

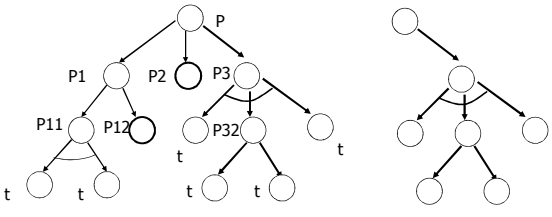
（2）对“或”节点，当其全部子节点中都为不可解节点时，则该“或”节点就是不可解节点。

（3）对“与”节点，只要其子节点中有一个为不可解节点时，该“与”节点是不可解节点。

56

6. 解图（解树）

由可解节点构成的，并且由这些可解节点可以推出初始节点为可解节点的子图为解图。



57

1. 2. 2 与或图搜索

1. 搜索方式

- 与或树的盲目搜索
  - 与或树的广度优先搜索
  - 与或树的深度优先搜索
- 与或树的启发式搜索
  - 与或树的有序搜索
  - 博弈树的启发式搜索

2. 一般搜索过程：

（1）把原始问题作为初始节点S0，并把它作为当前节点；

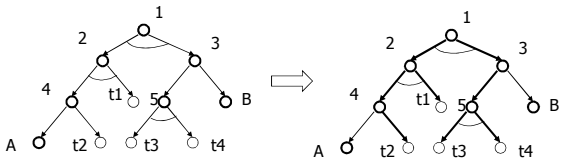
（2）应用**分解**或**等价变换**操作对当前节点进行扩展；

（3）为每个子节点设置指向父节点的**指针**；

（4）选择合适的子节点作为当前节点，反复执行第（2）步和第（3）步，在此期间需要多次调用**可解标记过程**或**不可解标记过程**，直到初始节点被标记为**可解节点或不可解节点**为止。

3. 搜索树，解树　　搜索目标：寻找解树

设有如下图所示的与/或树，节点按图中所标注的顺序号进行扩展，其中标有t1、t2、t3、t4的节点是终止节点，A 、 B 为不可解的端节点。



1. 2. 3 与/或树的启发式搜索

1. 最优解树: 代价最小的那棵解树。

2．节点的代价

（1）若n为终止节点，h(n)=0。

（2）若n为或节点，且子节点为n1，n2，…，nk，

$$h(n)= \min \{c(n，ni)+ h(ni)\} \quad (1\le i\le k)$$

（3）若n为与节点，且子节点为n1，n2，…，nk，

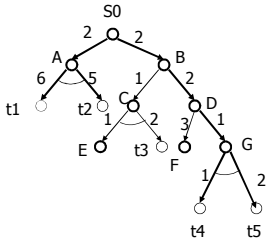
和代价法： $h(n)= \sum((c(n，ni)+ h(ni)) \quad i=1,2,\ldots\ldots,k$

最大代价法： $h(n)= \max\{c(n，ni)+h(ni)\} \quad (1\le i\le k)$

（4）若n是端节点，但又不是终止节点，则n不可扩展, $h(n)= \infty$

3. 解树的代价：初始节点S0的代价

设右图是一棵与/或树，其中包括两棵解树，左边的解树由S0、A 、 t1及t2组成；右边的解树由S0、 B、D、G、t4、t5 组成。在此与/或树中，t1、t2、t3、t4 、t5为终止节点； E、 F是端节点；边上的数字是该边的代价。请计算解树的代价。



解：左边解树，

按和代价：h(A)=11

h(S0)=13;

按最大代价：h(A)=6

h(S0)=8;

右边解树，

按和代价：h(G)=3, h(D)=4,

h(B)=6, h(S0)=8;

按最大代价：h(G)=2, h(D)=3,

h(B)=5, h(S0)=7。

4．希望树

选择那些最有希望成为最优解树一部分的节点进行扩展

这些节点及其父节点所构成的与/或树最有可能成为最优解树的一部分

随搜索过程而不断变化

希望解树T

（1）初始节点S0在希望树T中；

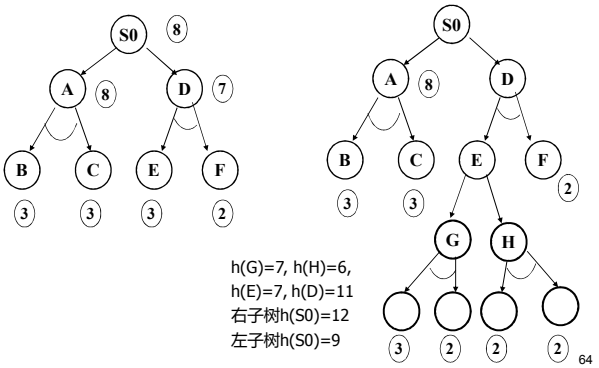
（2）如果n是具有子节点n1，n2，…，nk的或节点，则具有

$$h(n)= \min \{c(n，ni)+ h(ni)\} \qquad 1\le i\le k$$

的某个子节点ni在希望树T中；

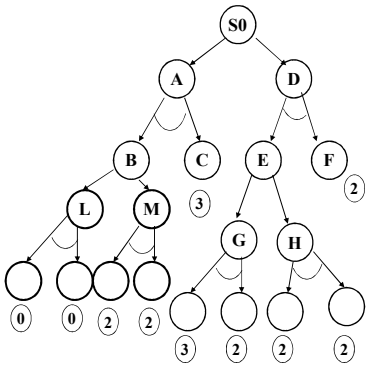
（3）如果n是与节点，则n的全部子节点都在希望树T中。

搜索过程每次扩展节点 时都同时扩展两层，且按一层或节点、一层与节点的间隔方式进行扩展。端节点B、C、E、F下面的数字是用启发函数估算出的h值。按和代价法计算。



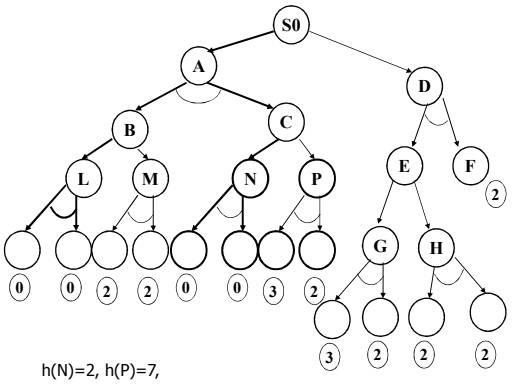
h(G)=7, h(H)=6,
h(E)=7, h(D)=11
右子树h(S0)=12
左子树h(S0)=9

64



h(L)=2, h(M)=6,
h(B)=3, h(A)=8
左子树h(S0)=9

65



h(N)=2, h(P)=7,
h(C)=3, h(A)=8
左子树h(S0)=9

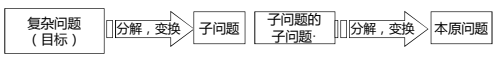
66

1.3 与或图搜索问题求解

67

1.问题的与或图描述

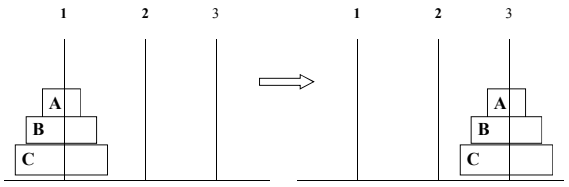
1. 问题归约：



2. 问题的与或图表示包括
(1)一个初始问题描述
(2)一套把问题变换为子问题的操作符
(3)一套本原问题描述

68

2. 三阶梵塔问题



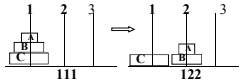
初始配置 **目标配置**

问题描述: 采用三元组表示状态: S=(i,j,k)
其中，i, j, k分别表示金片C,B，A所在的钢针号。
初始状态（1，1，1），目标状态（3，3，3）

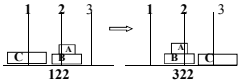
69

求解过程:

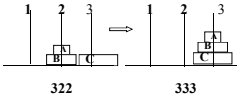
(1) 移动金片A和B至钢针2



(2) 移动金片C至钢针3



(3) 移动金片A和B至钢针3



70

原问题可分解为三个子问题：

（1）把金片A及B移到2号钢针上的双金片移动问题。[(1,1,1)→(1,2,2)]

（2）把金片C移到3号钢针上的单金片移动问题。[(1,2,2)→(3,2,2)]

（3）把金片A及B移到3号钢针上的双金片移动问题。[(3,2,2)→(3,3,3)]

本原问题：是可直接求解或具有已知解答的问题。

问题归约表示由三部分组成：

（1）初始问题描述 [（111），（333）]

（2）把问题变换为子问题的操作符—问题归约算符

移动A、B → 2 等

（3）本原问题描述 如：[（122）→（322）]

71