目录

第一章

略

第二章

1. 产生式系统的定义与组成部分

产生式系统是人工智能系统常用的一种程序结构，通常由以下三部分组成：综合数据库、产生式规则集和控制系统。

综合数据库是由问题的状态描述所构成的集合。

产生式规则具有 IF<前提条件>THEN<操作>的形式。当规则的前提条件被某一状态描述满足时，就对该状态施行规则所指出的操作。

控制系统决定在这些适用的规则中选出哪一条来使用。控制系统的第二个任务是检验状态描述是否满足终止条件，如果满足终止条件，则终止产生式系统的运行，并用使用过的规则序列构造出问题的解。

2. 产生式系统的基本过程（必考）

Procedure PRODUCTION

1. DATA←初始状态描述
2. until DATA 满足终止条件，do：
3. begin
4. 在规则集合中，选出一条可用于 DATA 的规则 R
5. DATA←把 R 应用于 DATA 所得的结果
6. End

步骤 4 是不确定的，只要求选出一条可用的规则 R，至于这条规则如何选取，却没有具体说明。

选取规则所依据的原则称为控制策略。

多数人工智能系统控制策略的信息并不足以在第 4 步选出最合用的规则，因此控制策略便成了一个搜索过程。

高效的系统需要被求解问题足够的知识，以便在步骤 4 选出一条最合用的规则。

3. 产生式系统的特点

1、模块性强。综合数据库、规则集和控制系统相对独立，程序的修改更加容易。

2、各产生式规则相互独立，不能互相调用，增加一些或删去一些产生式规则都十分方便。

3、产生式规则的形式与人们推理所用的逻辑形式十分接近，人们具有的知识转换成产生式规则很容易，产生式规则也容易被人们读懂。

4. 产生式系统的控制策略

产生式系统的控制策略可以分为两类，一类是不可撤回的，另一类是试探性的。试探性控制策略进一步又可分为回溯方式和图搜索方式。

5. 不可撤回式控制解八数码难题

设爬山函数 CF(S) ：不在目标位的数码个数的负值。

初始状态 S0 ：

2	8	3
1	6	4
7		5

目标状态 Sg ：

1	2	3
8		4
7	6	5

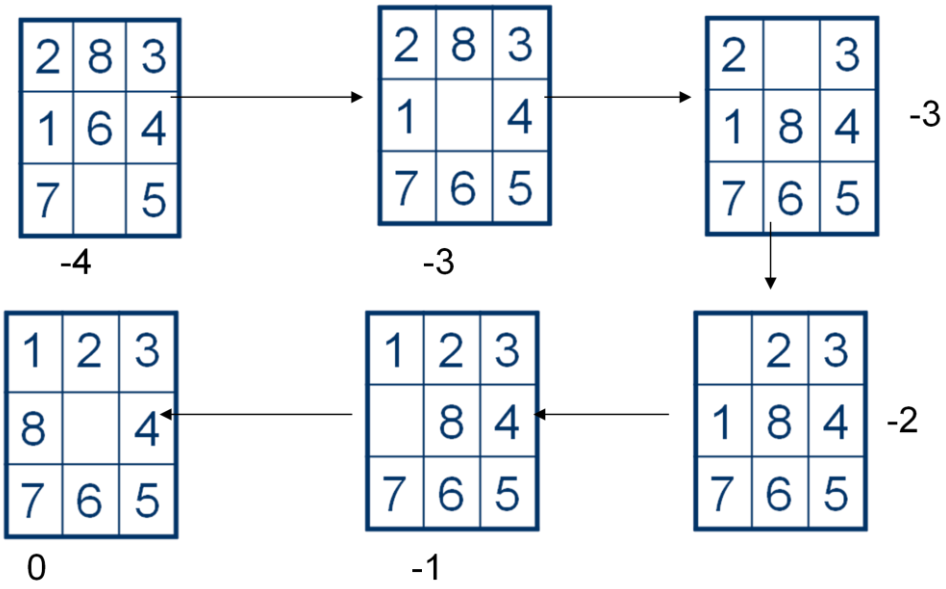
CF(S0) = - 4　　CF(Sg) = 0

状态描述 S：3 阶方阵

4 条产生式规则使用顺序：空格左、上、右、下移。

控制策略：处于任一状态 S，系统根据爬山函数选择一条规则使得这条规则作用于 S 时，获得的下一状态爬山函数不减少且最大（亦即距离目标状态最少）。

解过程：



6. 回溯控制策略的定义及产生回溯的条件

回溯方式是一种试探性的控制策略。（类似深度优先）

1．　当产生了一个上溯到初始状态的路径上已经出现过的状态时；

2．　当已应用了一定数量的规则仍未得到一个解时，这个数量是事先确定的，叫做深度限制；

3．　当已没有了可用的规则时。

7. 图搜索控制策略的定义

图搜索控制方式可以同时记录多个规则序列的执行效果，作为一种特殊情形，我们可以用树结构记录规则的应用和所产生的状态，这种树结构称作搜索树。树的根表示初始状态，有向弧表示规则的使用，除根以外的其它各节点：规则应用的结果图搜索控制方式不断地扩展搜索树，直到它包括了满足终止条件的节点为止。

8. 产生式系统的工作方式

1． 正向产生式系统（数据驱动控制）：

综合数据库：用状态描述

产生式规则：F 规则--状态产生新状态

从初始状态出发，不断地应用 F 规则，直到产生目标状态为止。

适用条件：初始节点数≤目标节点数

2． 反（逆）向产生式系统（目标驱动控制）：

综合数据库：用目标描述

产生式规则：B 规则 目标产生子目标

从目标状态出发，利用反向的产生式规则（ B 规则 ）不断地产生子目标，直到产生出与初始状态相同的子目标为止。

适用条件：初始节点数≥目标节点数

3． 双向产生式系统：正向产生式系统和反向产生式系统结合.

综合数据库：既有初始状态描述，又有目标状态描述。

产生式规则集：既有 F 规则，又有 B 规则。

正向产生式规则用在初始方向的状态描述上，反向产生式规则用在目标描述上。

控制系统:判断选 F 规则还是 B 规则；判断已经产生的状态和目标是否能以某种方式匹配，从而满足结束条件。

结束条件：中间汇合时的状态。

适用条件：初始节点数与目标节点数都很多。

特点：效率高；复杂。

9. 可交换的产生式系统的定义与性质（必考）

在某些产生式系统中。规则应用的次序对产生的状态无影响，即从初始状态到目标状态不依赖规则次序，因此可应用不可撤回式控制策略，从而提高了产生式系统的效率，这类产生式系统就是可交换的产生式系统。

一个产生式系统称为可交换的，如果对任意状态描述 D 具有如下性质：

（a）设 RD 是可应用于 D 的规则集，任取 r ∈RD， r 作用于 D 得 D′，设为 D′= r(D)，则 r 对 D′ 可用(即：设 D′的可用规则集为 RD′，则 r∈ RD′ ，即 RDt RD′)；（每一条对 D 可应用的规则，对于对 D 应用一条可应用的规则后所产生的状态描述仍是可应用的）（可应用性）。

（b）设 D 满足目标条件，D 的可用规则集为 RD，任取 r∈ RD ，用 r 作用到 D 产生状态 D′ ，D′满足目标条件。（如果 D 满足目标条件，则对 D 应用任何一条可应用的规则所产生的状态描述也满足目标条件）（可满足性）。

（c）设 D 的可用规则集为 RD，任取 r1, r2, …, rn ∈RD，依据（a）将 r1, r2, …, rn 依次作用到 D 及产生的状态上，得状态 Dn；设 r1′, r2′, …, rn′ 是 r1, r2, …, rn 的任意一个排列，用 r1′, r2′, …, rn′依次作用到 D 及产生的状态上，得状态 Dn′，则 Dn′= Dn （对 D 应用一个由可应用于 D 的规则所构成的规则序列所产生的状态描述不因序列的次序不同而改变）(无次序性）。

10. 可分解的产生式系统的定义

能够把产生式系统综合数据库的状态描述分解为若干组成部分，产生式规则可以分别用在各组成部分上，并且整个系统的终止条件可以用在各组成部分的终止条件表示出来的产生式系统，称为可分解的产生式系统。

11. 可分解的产生式系统的基本过程（必考）

Procedure SPLIT

1. DATA ← 初始状态描述
2. {Di} ← DATA 的分解结果；每个 Di 看成是独立的状态描述
3. until 对所有的{Di}， Di 都满足终止条件，do:

4. begin
5. 在{Di}中选择一个不满足终止条件的 D*
6. 从{Di}中删除 D*
7. 从规则集合中选出一个可应用于 D*的规则 R
8. D ← 把 R 应用于 D*的结果
9. {di} ← D 的分解结果
10. 把{di}加入{Di}中
11. end

12. 可分解的产生式系统重写问题的解序列

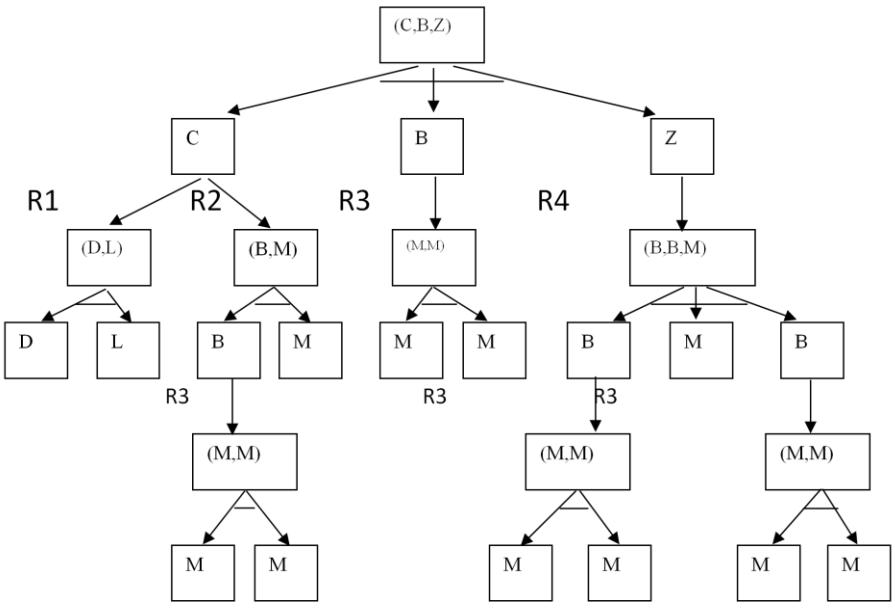
设产生式系统的初始状态描述为（C、B、Z），它的产生式规则是以下重写规则：

R1：C –> (D , L)　　R2：C –> (B , M)

R3：B –> (M , M)　　R4：Z –> (B , B , M)

终止条件是状态描述仅包含 M。

重写问题的 AND/OR 树：



第三章

1. 算法中用到的部分变量、常量、谓词、函数

- 变量：

  DATA，RDATA：状态变量

  RULES, PATH ： 表变量

- 常量：

  NIL: 空表 ----LISP 语言中的常量，也可用（ ）

- 谓词：

  TERM(DATA): DATA 满足结束条件时，为真

  DEADEND(DATA):DATA 不在解路上，为真（往下到达目标的可能性来定义这个谓词：若从 DATA 当前状态往下走到达目标的可能性很小时，则放弃这个状态 ）

  NULL(X): 表 X 为空表时，为真

- 函数：

  APPRULES(DATA): 将 DATA 所有可用规则进行排序所得到的表

  FIRST(X): 取表 X 的头

  TAIL(X): 取表 X 的尾

  CONS(E, X):将 E 加入表 X 前

2. 回溯算法（BACKTRACK）的基本过程（必考）

```
Recursive Procedure BACKTRACK ( DATA )
    1 . if TERM ( DATA ) , return NIL ;
    2 . if DEADEND ( DATA ) , return FAIL ;
    3 . RULES←APPRULES ( DATA ) ;
    4 . LOOP : if NULL ( RULES ) , return FAIL ;
    5 . R←FIRST ( RULES ) ;
    6 . RULES←TAIL ( RULES ) ;
    7 . RDATA←R ( DATA ) ;
    8 . PATH←BACKTRACK ( RDATA ) ;
    9 . if PATH = FAIL , go LOOP ;
    10. return CONS ( R , PATH ) .
```

3. 回溯算法（BACKTRACK）算法的修改与补充（必考）

```
Recursive Procedure BACKTRACK1 ( DATALIST )
    1. DATA←FIRST(DATALIST) ;
    2. if MEMBER(DATA,TAIL(DATALIST)) ,
           return FAIL;
    3. if TERM ( DATA ) , return NIL ;
    4. if DEADEND ( DATA ) , return FAIL ;
    5. if LENGTH(DATALIST)>BOUND,
           return FAIL ;
    6. RULES←APPRULES ( DATA ) ;
    7. LOOP : if NULL ( RULES ) , return FAIL ;
```

```
8.  R←FIRST（RULES）；
9.  RULES←TAIL（RULES）；
10. RDATA←R（DATA）；
11. RDATALIST←CONS(RDATA,DATALIST）；
12. PATH←BACKTRACK1（RDATALIST）
13. if PATH = FAIL，go LOOP；
14. return CONS（R，PATH）
```

4. 回溯策略的四皇后问题

四皇后问题：4 枚皇后放在 4X4 的国际象棋棋盘上，如何放置使得它们不能相互俘获。

俘获：同行；同列；同对角线

综合数据库：以状态为节点的有向图

状态：4X4 矩阵

初始状态： 空矩阵

规则：Rij:if i=1 时，矩阵中无皇后标志，或 4 ≥ i >1 时，矩阵的 i-1 行有一个皇后标志，then 在矩阵的第 i 行第 j 列放一个皇后标记

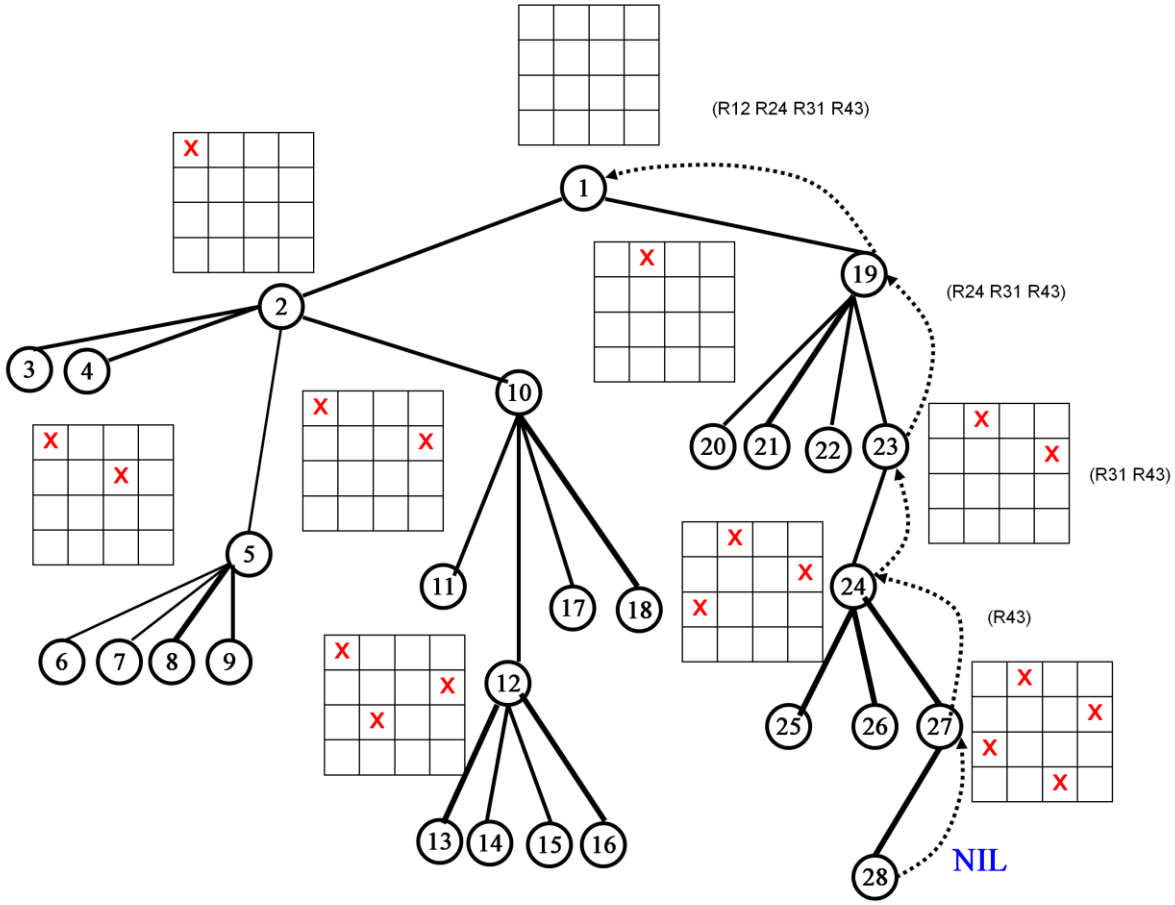
结束条件：TERM 为真，矩阵中有 4 个皇后标志，且不能相互俘获

控制策略：回溯

DEADEND(DATA): DATA 中存在 1 对皇后相互俘获，为真

APPRULES(RULES): j<k 时 Rij 排在 Rik 之前

回溯策略解过程：



5. 图搜索策略相关概念

有向图 ：G=(P,A)，P:点集 A:弧集

弧:两点间有方向的线。

如果有一条弧从节点 ni 出发指向 nj；则节点 nj 称为节点 ni 的子节点，节点 ni 称为节点 nj 的父亲节点 .

对于产生式系统，

节点：用状态描述标记

弧：用规则标记

假定图中的每一个节点只有有限个子节点。

路：节点序列(ni0, ni1,…，nik)称为从节点 ni0 到节点 nik 的一条长度为 k 的路径，其中，对于 j=1，…，k，每一个 nij 都是 nij-1 的子节点。

如果存在一条从节点 ni 到节点 nj 的路径，则节点 nj 称做是从节点 ni 出发可达到的，节点 nj 称做节点 ni 的后裔，节点 ni 称做是节点 nj 的祖先。

解路径： 从初始节点到一个满足终止条件节点的路径。

图搜索策略把寻找从初始状态描述到目标描述的规则序列问题转化成寻找有向图的解路径问题．

有向树：每一个节点最多只有一个父亲的有向图．

根节点：有向树中没有父节点的节点

叶节点：有向树中没有子节点的节点

有向树中节点的深度:

1) 根节点的深度是 0，

2)其它节点的深度等于它父节点的深度加 1。

加权有向图（权图）：

每条弧线上都有使用费用（正数）的图。

例：从节点 ni 到节点 nj 的有向孤的费用：C(ni, nj)

路径的费用：路径上所有弧费用的和．

两节点间具有最小费用的路径：是此两节点间所有弧费用的费用总和最小的一条路。

最佳解路径：费用最小的解路径。

隐含图：由部分节点和一组其它节点生成规则所确定的图．

图搜索控制策略的目标：从隐含图出发生成一个部分明确的图，使该明确图中含有目标节点．

图搜索非形式定义

假定：所有隐含图中有向弧的费用大于某一个小的正数 ε

问题：求隐含图中初始节点 s 到目标节点集{ti}中任意成员的一条具有最小费用的解路径。

6. 图搜索算法（GRAPHSEARCH）的基本过程（必考）

- OPEN 表：未扩展的节点
- CLOSED 表：已扩展或正在扩展的节点
- G：搜索图，动态变化，部分明确的图

Procedure GRAPHSEARCH

1．G←{s}， OPEN ←（s）．

2．CLOSED ←NIL．

3．LOOP：IF OPEN=NIL,THEN FAIL．

4． n ← FIRST(OPEN)，OPEN ←TAIL(OPEN),CONS(n, CLOSED) ．

5． IF TERM(n)，THEN 成功结束

（解路径可通过追溯 G 中从 n 到 s 的指针获得）。

6． 扩展节点 n，

令 M={m｜ m 是 n 的子节点，且 m 不是 n 的祖先} ，

G ←G ∪M

7．（设置指针，调整指针）对于 m∈M,

(1)若 m∉CLOSED, m∉OPEN, 建立 m 到 n 的指针，并 CONS(m, OPEN).

(2)(a)m∈OPEN, 考虑是否修改 m 的指针.

(b)m∈CLOSED,考虑是否修改 m 及在 G 中后裔的指针。

8． 重排 OPEN 表中的节点（按某一任意确定的方式或者根据探索信息）。

9． GO LOOP

7. 无信息的图搜索过程

一般有两种无信息的图搜索过程：深度优先搜索与宽度优先搜索。

深度优先搜索：排列 OPEN 表中的节点时按它们在搜索树中的深度递减排序 。深度最大的节点放在表的前面，深度相等的节点以任意方式排序。

宽度优先搜索：在排列 OPEN 表中节点时按它们在搜索图中的深度递增顺序，深度最小的节点放在表的前面 。

8. 启发式图搜索过程

启发式信息：用于帮助减少搜索量的与问题有关的信息或知识。

启发式搜索：使用启发信息指导的搜索过程叫做启发式搜索。

启发信息强，可以降低搜索的工作量，但可能导致找不到最优解；

启发信息弱，一般会导致搜索的工作量加大，极端情况下演变为盲目搜索，但有可能找到最优解。

- 使用启发信息的一种重要方法是采用估价函数

估价函数：定义在状态空间上的实值函数。

用 f(n)表示节点 n 的估价函数：

1. f(n)表示从起点到目标，经由节点 n 最小费用路径上费用的估计。

（在搜索图中，接近解路径的节点有较低的函数值）

2. 以估价函数 f 的递增次序排列 OPEN 表中的节点：

估价函数低的排在前；

具有相等函数值的节点以任意次序排序。

- 八码难题估价函数：f(n)=d(n)+W(n)

d(n)：节点 n 在搜索树中的深度

W(n)：与 n 对应的状态描述中偏离目标的棋子的个数。

9. A 算法和 A*算法的定义

A 算法：使用估价函数 f(n)=g(n)+h(n) 排列 OPEN 表中节点顺序的 GRAPHSEARCH 算法。

g(n)：对 g*(n)的一个估计是当前的搜索图 G 中 s 到 n 的最优路径费用 g(n)≥g*(n)

h(n)：对 h*(n)的估计，称为启发函数。

Note：若 h(n)=0，g(n)=d，则 f(n)=d，为宽度优先。

A*算法：对任何节点 n 都有 h(n)≤h*(n)的 A 算法。

10. 算法可采纳的定义

定义：如果一个搜索算法对于任何具有解路径的图都能找到一条最佳路径，则称此算法为可采纳的。

11. A*算法的可采纳性（必考）

可以证明：A*算法是可采纳的（如果解路径存在，A*一定由于找到最佳解路径而结束）

定理 1 GRAPHSEARCH 对有限图必然终止。

定理 2 若存在 s 到目标的解路径，则算法 A*终止前的任何时刻，OPEN 表中总存在一个节点 n′， n′在从 s 到目标的最佳解路径上，且满足 f(n′) ≤f*(s)

定理 3 若存在解路径，则 A*算法必终止。

定理 4 算法 A*是可采纳的。（若解路径存在,A*一定找到最佳解路径而终止）.

定理 5 算法 A*选择的任意扩展点 n 都有 f(n)≤f*(s)

定理 6 如果 A1 和 A2 是两个 A*算法，算法 A2 比 A1 有较多的信息，且它们搜索同一个隐含图。若该图存在解路径，当这两个算法终止时，A2 所扩展的每一节点也必被 A1 所扩展，即：A2 扩展的节点数≤A1 扩展的节点数

定理 7　如果 A*算法的启发函数 h 满足单调限制，则当算法 A*选择节点 n 扩展时，就已经发现了通向节点 n 的最佳路径，即 g(n)=g*(n)．

定理 8　如果 A*算法启发式函数 h 满足单调限制，则 A*所扩展的节点序列的估价函数值是非递减的．

12. 单调限制的定义

定义　如果启发函数 h 对任何节点 ni 和 nj，只要 nj 是 ni 的后继，都有

h(ni) - h(nj)≤c(ni, nj)

h(t)=0　（t 是目标节点）

则称启发函数 h 满足单调限制．

13. 算法 A 的启发能力的定义及影响因素

定义　设 A1 和 A2 是两个启发式算法，它们分别使用估价函数 f1 和 f2，如果在寻找解路径的过程中，A1 所用的计算费用比 A2 少，则称 A1 比 A2 有较强的启发能力，也可以称估价函数 f1 比 f2 有较强的启发能力．

影响算法 A 启发能力的三个重要因素：

（1）算法 A 所找到的解路径的费用。

（2）算法 A 在寻找这条解路径的过程中所需要扩展的节点数。

（3）计算启发函数所需要的计算量。

14. 算法 A 的启发能力补充

可控制 g 和 h 在搜索中所起的作用：

有时，把估价函数写成：f=g+wh(w 为正数)的形式

w 很小时，强调宽度优先

w 很大时，强调启发分量

经验表明，让 w 的值与搜索树中节点的深度成相反方向变化时，搜索效率往往会有所提高．在深度较小时，搜索主要依靠启发部分，随着深度的增加，宽度优先部分的作用逐渐增大，以保证最终能找到一条解路径．一般 w 与深度成反比。

15. 渗透度的定义

渗透度是对一个搜索算法的搜索性能的度量，表示搜索集中指向某个目标的程度，而不是在无关的方向上徘徊。

定义为：

$$P = L / T$$

其中，L 是算法发现的解路径的长度，T 是算法在寻找这条解路径期间所产生的节点数（不包括初始节点，包括目标节点）

16. 有效分枝系数的定义

有效分枝系数就是一棵搜索树的平均分枝数．

设搜索树的深度是 L，算法所产生的总节点数为 T，有效分枝系数是 B，则有

$$B + B^2 十 ... + B^L = T$$

或

$$B (B^L - 1) / (B - 1) = T$$

第四章

1. 与/或图（AND/OR 图）的定义与相关概念

定义： 与/或图是一种超图．在超图中父亲节点和一组后继节点用超弧连接．超弧又叫 k-连接符．

k-连接符：一个父节点指向一组 k 个有与关系的后继节点， 这样一组弧线称为一个 k-连接符．k>1 时，用一圆弧标记此连接符。

注意：若所有的连接符都是 1-连接符，则得到的就是与/或图的特例--普通有向图。

与/或树：每一个节点最多只有一个父亲的与/或图．

根节点： 在 AND/OR 树或 AND / OR 图中没有父节点的节点.

叶节点：在 AND/OR 树或 AND / OR 图中没有后继的节点．

终止节点： 满足终止条件的节点．

一个可分解的产生式系统定义一个隐含的与/或图．图的根节点表示产生式系统的初始状态描述，连接符表示对一状态描述应用产生式规则或把这一状态描述分解成若干组成部分．

可分解产生式系统的任务：从隐含的与/或图出发找出一个从根节点出发到终止节点集的解图。

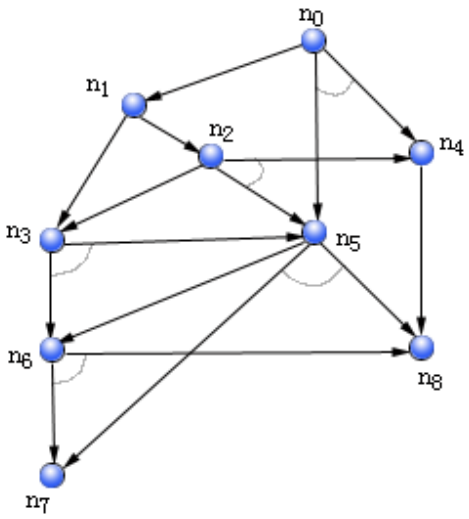
加权与/或图：权加在连接符上。

假定所有连接符的费用均大于某一小的正数 ε。使用连接符的费用可以计算解图的费用．

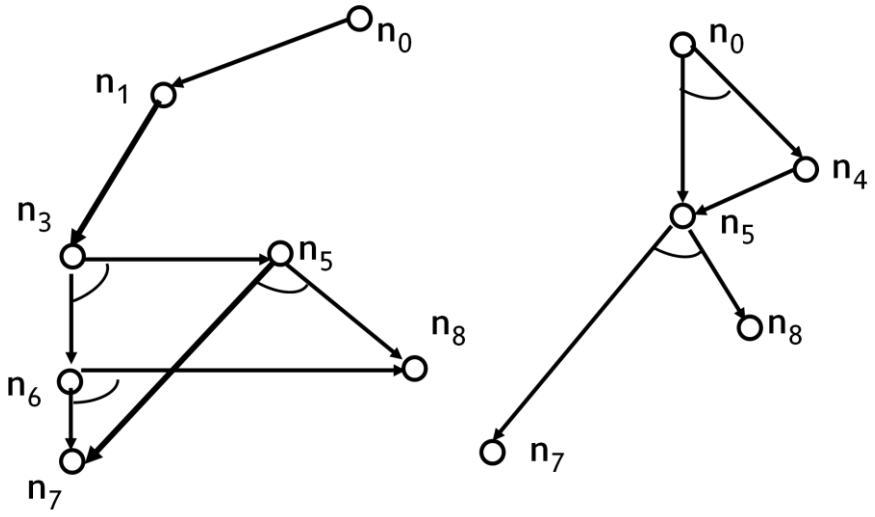
最佳解图：具有最低费用的解图

2. 与/或图（AND/OR 图）的解图及递归定义

AND/OR 图



两个解图：



设 N 是与/或图 G 的终止节点集合，图 G 中无回路，从节点 n 出发到 N 的一个解图是与/或图 G 的一个子图，用 G′表示，递归定义如下：

1. 若 n 是 N 中的一个元素，则 G′只包括节点 n；

2. 若 n 有一个从 n 出发的连接符 k 指向后继节点集合｛n1，…，nk｝，而每一个 ni 都有从 ni 出发的解图，则 G′由节点 n、连接符 k、｛n1，…，nk｝中的每一个节点到 N 的解图所构成；

3. 否则，G 没有从 n 出发到 N 的解图．

设从节点 n 到终止节点集合 N 的解图的费用用 k(n, N)表示，则 k(n, N)递归定义如下：

1. 若 n 是 N 中的元素，则 k(n, N) =0；

2. 若有从 n 出发的一个连接符指向它的解图后继节点 { n_1，…，n_i } ，设此连接符的费用为 C_i，则:

$$k(n, N)= C_i+ k(n_1, N)+...+k(n_i, N)$$

3. 与/或图（AND/OR 图）的启发式搜索过程（必考）

假定 h*(n)是从 n 出发的最佳解图的费用，h(n)是 h*(n)的估计值。利用 h(n)指导对 AND/OR 图的启发式搜索。

在 AND/OR 图中，对任意连接符的单调限制是：

$$h(n)\leq c+h(n_1)+...+h(n_k)$$

其中，n 是任意节点，c 是从 n 出发的连接符的费用， n_1，…，n_k 是 n 的在此连接符下的后继节点。

Note: 若对于所有的终止节点，都有 h(n) = 0，则单调限制还隐含着 h 对所有的节点 n，都有：h(n) ≤h*(n)。

AO*算法的基本过程：

Procedure AO*

1．建立一个只由根节点构成的搜索图 G．s 的费用 q(s) := h(s), G':=G．如果 s 是目标，标记 s 为 SOLVED.

2．Until s 被标记为 SOLVED，do：

3．begin

4． 通过跟踪从 s 出发的有标记的连接符计算部分解图 G'（G 的连接符将在以后的步骤中标记）

5．在 G'中选一个非终止的叶节点 n.

6．扩展节点 n 产生 n 的所有后继，并把它们连到图 G 上，对于每一个不曾在 G 中出现的后继 n_j，$q(n_j)$:=h(n_j)，如果这些后继中某些节点是终止节点，则用 SOLVED 标记。

7．S: = {n}；建立一个只由 n 构成的单元素集合 S。

8．Until S 变空，do：

9．begin

10．从 S 中删除节点 m，满足 m 在 G 中的后裔不出现在 S 中

11． 按以下步骤修改 m 的费用 q(m)：对于每一从 m 出发的指向节点集合{n_{1i}，…，n_{ki}}的连接符，计算 $q_i(m)=c_i+q(n_{1i})+\ldots+q(n_{ki})$，q(m): = min {$q_i(m)$}。

(1)将指针标记加到实现此最小值的连接符上。

(2)如果本次标记与以前的不同，抹去先前的标记。

(3)如果这个连接符指向的所有后继节点都标记了 SOLVED，则把 m 标上 SOLVED．

12． 如果 m 标记了 SOLVED 或者如果 m 的修改费用与以前的费用不同，则把 m 的通过指针标记的连接的所有父节点加到 S 中．

13． end

14． end

4. 能解节点（SOLVED）的定义

① 终止节点是能解节点；

② 若非终止节点有“或”子节点时，其子节点有一能解，则该非终止节点是能解节点；

③ 若非终止节点有“与”子节点时，若其子节点均能解，则该非终止节点是能解节点。

5. AO*算法的两个阶段

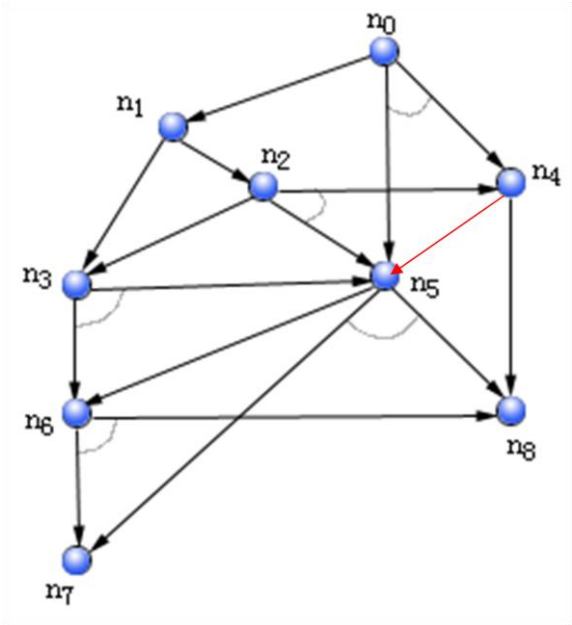
第一阶段：自顶向下的图生成过程。（对于每一个已经扩展了的节点，算法都有一个指针，指向该节点的后继节点中费用值小的那个连接符）

从初始节点出发，先通过有指针标记的连接符，向下搜索，一直到找到未扩展的节点为止（找到目前为止费用值最小的一个局部解图）。然后对其中一个非终止节点进行扩展，并对其后继节点赋费用值和加能解标记。

第二阶段：费用值计算过程。

完成自下向上的费用值修正计算、指针的标记以及节点的能解标记。

6. AO*算法应用举例

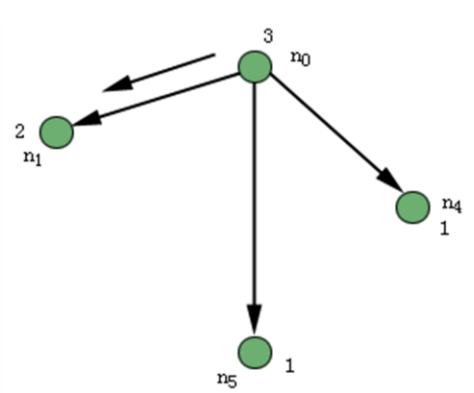


设某个问题的状态空间如图所示。

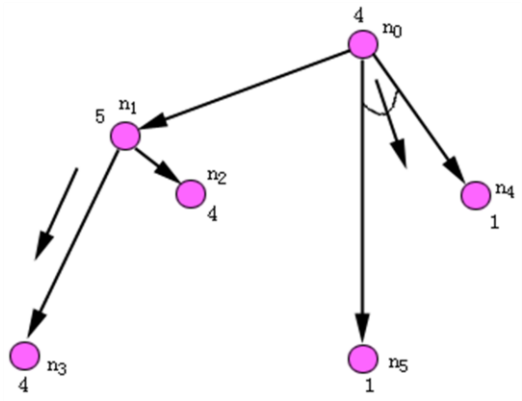
$h(n_0)=0$，$h(n_1)=2$，$h(n_2)=4$，$h(n_3)=4$，$h(n_4)=1$，

$h(n_5)=1$，$h(n_6)=2$，$h(n_7)=h(n_8)=0$（目标节点）。

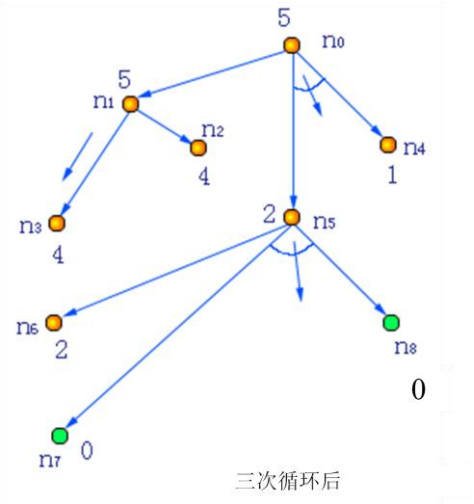
假设 k-连接符的费用值为 k。



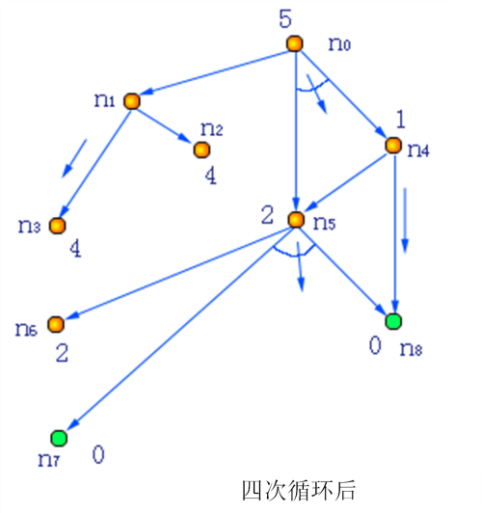
一次循环后



两次循环后



三次循环后



四次循环后

从 n0 开始，沿指向连接符的指针找到的解图即为搜索的结果。n0 给出的修正费用值 q(n0) = 5 就是解图的费用值。

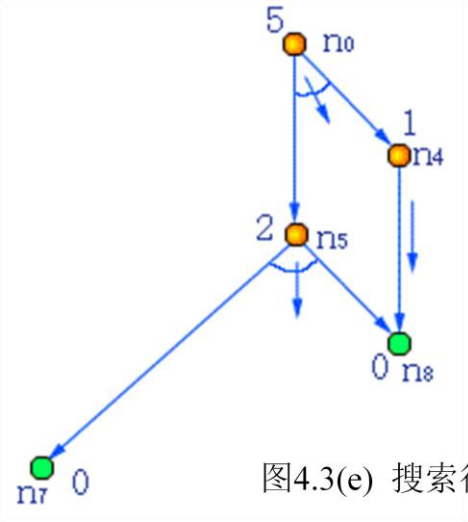


图4.3(e) 搜索得到的解图

Note:如果一个与 / 或 图存在解图，如果对于图中所有的节点 n 都有 h（n）≤h*（n），并且启发函数 h 满足单调限制，则 AO*算法必然终止于找出最佳解图。

7. 极小极大过程及原则

极小极大过程

1.按宽度优先生成 0 至 L 层所有节点。

2.使用静态估值函数计算第 L 层节点的函数值。

3.按极小极大原则计算各层节点的倒推值，直到求出初始节点的倒推值为止。实现该倒推值的走步就是相对好的走步。

极小极大原则

MAX 节点在其 MIN 子节点的倒推值中选 max；

MIN 节点在其 MAX 子节点的倒推值中选 min

倒推值

在极小极大过程中，第 i 层节点根据第 i+1 层节点的值使用极小极大原则而获得的值。

应用于九宫格的极小极大搜索过程（第三阶段）：

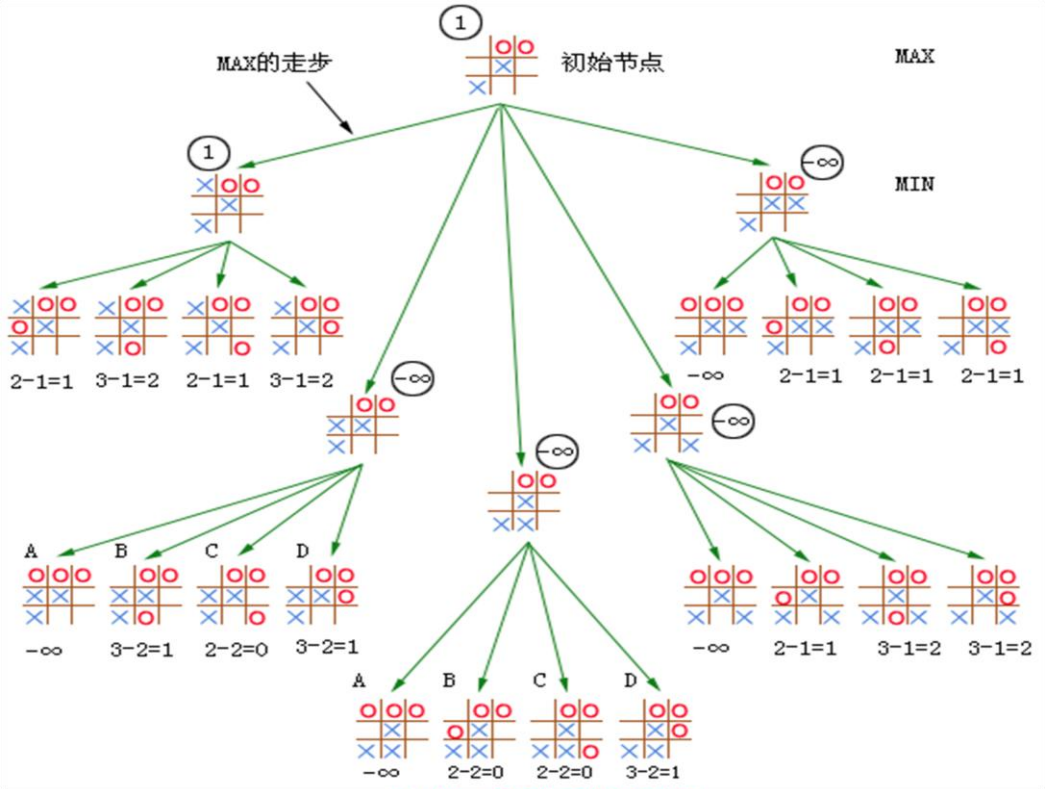


图4.10一字棋第三阶段搜索树

8. 博弈搜索 α-β 过程的定义及剪枝规则（必考）

定义：产生节点与返回值计算同时进行的博弈搜索过程叫 α-β 过程。在 α-β 过程中，计算 MAX 节点的返回值时，不是等待它的所有后继 MIN 节点都取得返回值之后再取它们的极大值，而是在它的第一个后继节点取得返回值之后，确定 MAX 的最终返回值的一个下界，这个下界称为该节点的 α 值；在计算 MIN 节点的返回值，也可以一边产生它的后继节点一边确定返回值的上界，这个上界称作该节点的 β 值。

剪枝规则：

（1） 如果一个 MIN 节点的 β 值小于或等于它的某一个 MAX 祖先节点的 α 值，则剪枝发生在该 MIN 节点之下。此时该 MIN 节点的 β 值可以当做这个节点的最终返回值使用。虽然这个值可能会与使用全部极小极大过程的返回值不同，但对最终结果并无影响。

（2） 如果一个 MAX 节点的 α 值大于或者等于它的某一个 MIN 祖先节点的 β 值，则剪枝发生在 MAX 节点之下。该 MAX 节点的最终返回值可以置成它的 α 值。在搜索过程中，α 和 β 值按如下方式确定：

① 令 α 值等于它的现有的后继节点的返回值中的最大者。

② 令 β 值等于它的现有的后继节点的返回值中的最小值。

9. 博弈搜索 α-β 剪枝的效率

若以最理想的情况进行搜索，即对 MIN 节点先扩展最低估值的节点（若从左向右顺序进行，则设节点估计值从左向右递增排序），MAX 先扩展最高估值的节点（设估计值从左向右递减排序），则当搜索树深度为 D，分枝因数为 B 时，若不使用 α-β 剪枝技术，搜索树的端节点数 B^D；若使用 α-β 剪枝技术．可以证明理想条件下生成的端节点数最少，有

$$N_D=2B^{D/2}-1\text{（D 为偶数）}$$

$$N_D=B^{(D+1)/2}+B^{(D-1)/2}-1\text{（D 为奇数）}$$

比较后得出最佳 α-β 搜索技术所生成深度为 D 处的端节点数约等于不用 α-β 搜索技术所生成深度为 D / 2 处的端节点数。因此，在使用相同存储空间的条件下，α-β 过程能把搜索深度扩大一倍。