第3章

确定性推理

第2章研究的知识表示方法是问题求解所必需的。表示问题是为了进一步解决问题。从问题表示到问题的解决，有个求解的过程，也就是搜索过程。在这一过程中，采用适当的搜索技术，包括各种规则、过程和算法等推理技术，力求找到问题的解答。本章首先讨论一些早期的搜索技术或用于解决比较简单问题的搜索原理，然后研究一些比较新的、能够求解复杂问题的推理技术。此外，将在第4章专门讨论不确定性推理问题。

3.1 图搜索策略

在2.1节的状态空间表示法中已经看到了，状态空间法用图结构来描述问题的所有可能状态，其问题的求解过程转化为在状态空间图中寻找一条从初始节点到目标节点的路径。从本节起，将要研究如何通过网络寻找路径，进而求解问题。首先研究图搜索的一般策略，它给出图搜索过程的一般步骤，并可从中看出无信息搜索和启发式搜索的区别。

可把图搜索控制策略看成一种在图中寻找路径的方法。初始节点和目标节点分别代表初始数据库和满足终止条件的目标数据库。求得把一个数据库变换为另一数据库的规则序列问题就等价于求得图中的一条路径问题。在图搜索过程中涉及的数据结构除了图本身之外，还需要两个辅助的数据结构，即存放已访问但未扩展节点的OPEN表，以及存放已扩展节点的CLOSED表。搜索的过程实际是从隐式的状态空间图中不断生成显示的搜索图和搜索树，最终找到路径的过程。为实现这一过程，图中每个节点除了自身的状态信息外，还需存储诸如父节点是谁，由其父节点是通过什么操作可到达该节点，以及节点位于搜索树的深度、从起始节点到该节点的路径代价等信息。每个节点的数据结构参考如图3.1所示，图中一个节点的数据结构包含5个域，即STATE——节点所表示状态的基本信息；PARENT NODE——指针域，指向当前节点的父节点；ACTION——从父节点表示的状态转换为当前节点状态所使用的操作；DEPTH——当前节点在搜索树中的深度；PATH COST——从起始节点到当前节点的路径代价。

图搜索(GRAPHSEARCH)的一般过程如下：

(1) 建立一个只含有起始节点 S 的搜索图 G，把 S 放到一个OPEN表中。

(2) 初始化CLOSED表为空表。

(3) LOOP：若OPEN表是空表，则失败退出。

(4) 选择OPEN表上的第一个节点，把它从OPEN表移出并放进CLOSED表中。称此节点为节点 n。

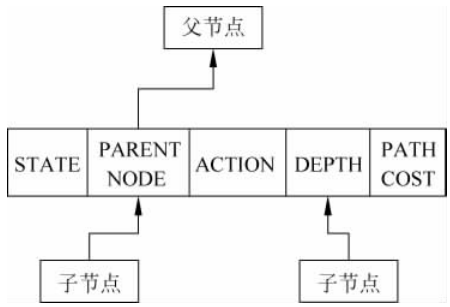

图 3.1　节点数据结构图

(5) 若 n 为一目标节点，则有解并成功退出，此解是追踪图 G 中沿着指针从 n 到 S 这条路径而得到的(指针将在第(7)步中设置)。

(6) 扩展节点 n，生成后继节点集合 M。

(7) 对那些未曾在 G 中出现过的(既未曾在 OPEN 表上，也未在 CLOSED 表上出现过的)M 成员设置其父节点指针指向 n 并加入 OPEN 表。对已经在 OPEN 或 CLOSED 表中出现过的每一个 M 成员，确定是否需要将其原来的父节点更改为 n。对已在 CLOSED 表上的每个 M 成员，若修改了其父节点，则将该节点从 CLOSED 表中移出，重新加入 OPEN 表中。

(8) 按某一任意方式或按某个探试值，重排 OPEN 表。

(9) GO LOOP。

以上搜索过程可用图 3.2 的程序框图来表示。

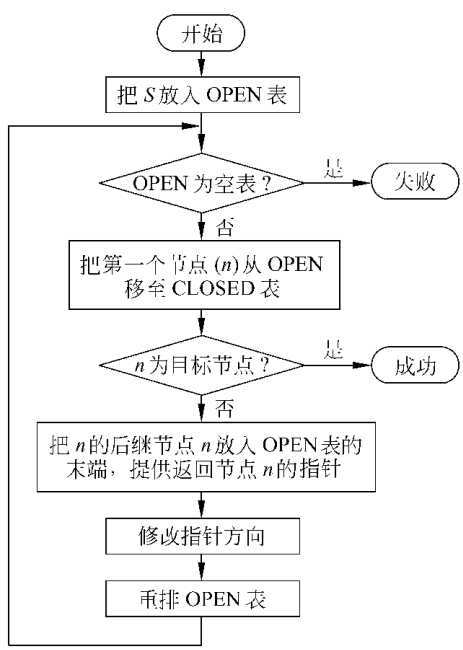

图 3.2　图搜索过程框图

这个过程一般包括各种各样的具体的图搜索算法。此过程生成一个显式的图 G(称为搜索图)和 G 的一个子集 T(称为搜索树),树 T 上的每个节点也在图 G 中。搜索树是由第(7)步中设置的指针来确定的。由于在搜索过程中每次都会根据需要来确定是否修改当前节点指向其父节点的指针,所以已经被扩展出来的 G 中的每个节点(除 S 外)都有且仅有唯一一个父节点,即形成了一棵树,也就是搜索树 T。由于在树结构中,任意两点间只存在唯一一条路径,所以可以从 T 中找到到达任意节点的唯一路径。搜索过程中使用的 OPEN 表存储的都是当前搜索树的叶子节点,因此也被称为 Fronge 表,即前沿表。较确切地说,在过程的第(3)步,OPEN 表上的节点都是搜索树上未被扩展的那些节点;在 CLOSED 表上的节点,或者是几个已被扩展但是在搜索树中没有生成后继节点的叶子节点,或者是搜索树的非叶子节点。

过程的第(8)步对 OPEN 表上的节点进行排序,以便能够从中选出一个“最好”的节点作为第(4)步扩展使用。这种排序可以是任意的(即盲目的,属于盲目搜索),也可以用以后要讨论的各种启发思想或其他准则为依据(属于启发式搜索)。每当被选作扩展的节点为目标节点时,这一过程就宣告成功结束。这时,能够重现从起始节点到目标节点的这条成功路径,其办法是从目标节点按指针向 S 返回追溯。当搜索树不再剩有未被扩展的叶子节点时,过程就以失败告终(某些节点最终可能没有后继节点,所以 OPEN 表可能最后变成空表)。在失败终止的情况下,从起始节点出发,一定达不到目标节点。

GRAPHSEARCH 算法同时生成一个节点的所有后继节点。为了说明图搜索过程的某些通用性质,将继续使用同时生成所有后继节点的算法,而不采用修正算法。在修正算法中,一次只生成一个后继节点。

从图搜索过程可以看出,是否重新安排 OPEN 表,即是否按照某个试探值(或准则、启发信息等)重新对未扩展节点进行排序,将决定该图搜索过程是无信息搜索或启发式搜索。本章后续各节,将依次讨论无信息搜索和启发式搜索策略。

3.2 盲目搜索

不需要重新安排 OPEN 表的搜索叫做无信息搜索或盲目搜索,它包括宽度优先搜索、深度优先搜索和等代价搜索等。盲目搜索只适用于求解比较简单的问题。

3.2.1 宽度优先搜索

如果搜索是以接近起始节点的程度依次扩展节点的,那么这种搜索就叫做宽度优先搜索(breadth-first search),如图 3.3 所示。从图 3.3 可见,这种搜索是逐层进行的;在对下一层的任一节点进行搜索之前,必须搜索完本层的所有节点。

宽度优先搜索算法如下:

(1) 把起始节点放到 OPEN 表中(如果该起始节点为一目标节点,则求得一个解答)。

(2) 如果 OPEN 是个空表,则没有解,失败退出;否则继续。

(3) 把第一个节点(节点 n)从 OPEN 表移出,并把它放入 CLOSED 的扩展节点表中。

(4) 扩展节点 n。如果没有后继节点,则转向上述第(2)步。

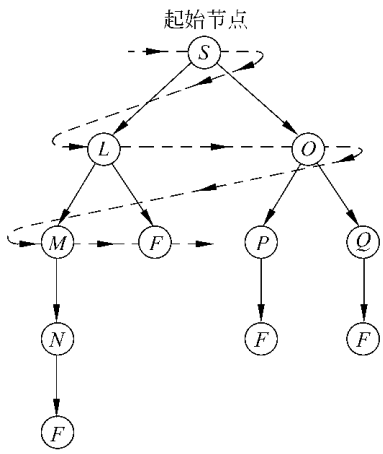

图 3.3　宽度优先搜索示意图

(5) 把 n 的所有后继节点放到 OPEN 表的末端，并提供从这些后继节点回到 n 的指针。

(6) 如果 n 的任一个后继节点是个目标节点，则找到一个解答，成功退出；否则转向第(2)步。

上述宽度优先算法如图 3.4 所示。

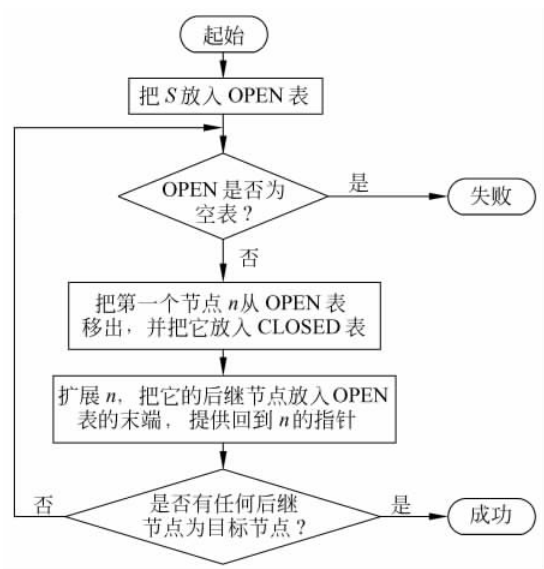

图 3.4　宽度优先算法框图

这一算法假定起始节点本身不是目标节点。要检验起始节点是目标节点的可能性，只要在第(1)步的最后，加上一句“如果起始节点为一目标节点，则求得一个解答”即可做到，正如第(1)步括号内所写的。

显而易见，宽度优先搜索方法在假定每一次操作的代价都相等的情况下，能够保证在搜索树中找到一条通向目标节点的最短途径。在宽度优先搜索中，节点进出 OPEN 表的顺序是先进先出，因此其 OPEN 表是一个队列结构。

图 3.5 绘出把宽度优先搜索应用于八数码难题时所生成的搜索树。这个问题就是要把初始棋局变为如下目标棋局的问题：

1 2 3
8 4
7 6 5

搜索树上的所有节点都标记它们所对应的状态描述，每个节点旁边的数字表示节点扩展的顺序(按顺时针方向移动空格)。图 3.5 中的第 27 个节点是目标节点。

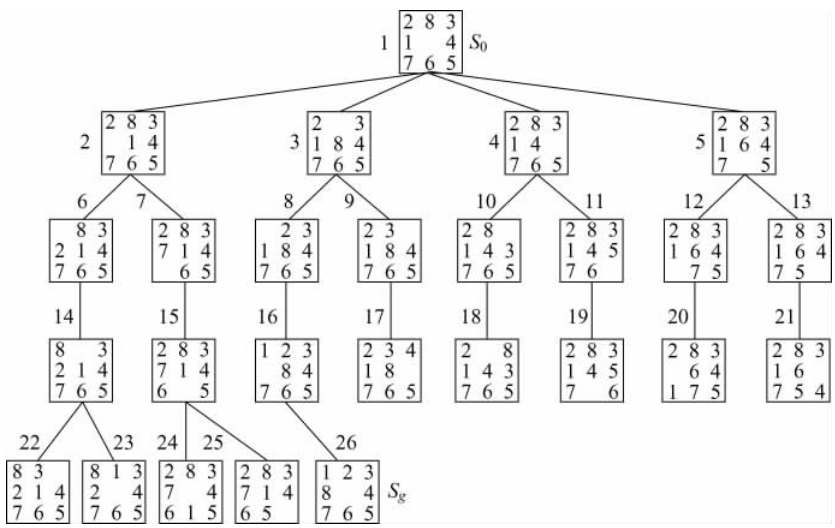

图 3.5 八数码难题的宽度优先搜索树

3.2.2 深度优先搜索

另一种盲目(无信息)搜索叫做深度优先搜索(depth-first search)。在深度优先搜索中，首先扩展最新产生的(即最深的)节点，如图 3.6 所示。深度相等的节点可以任意排列。定义节点的深度如下：

(1) 起始节点(即根节点)的深度为 0。

(2) 任何其他节点的深度等于其父节点深度加 1。

首先，扩展最深的节点的结果使得搜索沿着状态空间某条单一的路径从起始节点向下进行；只有当搜索到达一个没有后裔的状态时，它才考虑另一条替代的路径。替代路径与前面已经试过的路径的不同之处仅仅在于改变最后 n 步，而且保持 n 尽可能小。

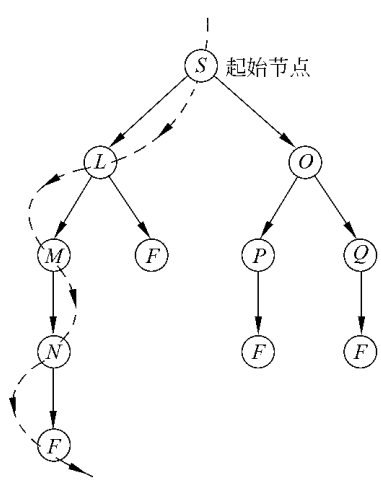

图 3.6 深度优先搜索示意图

对于许多问题，其状态空间搜索树的深度可能

为无限深，或者可能至少要比某个可接受的解答序列的已知深度上限还要深。为了避免考虑太长的路径（防止搜索过程沿着无益的路径扩展下去），往往给出一个节点扩展的最大深度——深度界限。任何节点如果达到了深度界限，那么都将把它们作为没有后继节点处理。值得说明的是，即使应用了深度界限的规定，所求得的解答路径并不一定就是最短的路径。

含有深度界限的深度优先搜索算法如下：

（1）把起始节点 S 放到未扩展节点 OPEN 表中。如果此节点为一目标节点，则得到一个解。

（2）如果 OPEN 为一空表，则失败退出。

（3）把第一个节点（节点 n）从 OPEN 表移到 CLOSED 表。

（4）如果节点 n 的深度等于最大深度，则转向（2）。

（5）扩展节点 n，产生其全部后裔，并把它们放入 OPEN 表的前头。如果没有后裔，则转向（2）。

（6）如果后继节点中有任一个为目标节点，则求得一个解，成功退出；否则，转向（2）。

有界深度优先搜索算法的程序框图如图 3.7 所示。很显然，深度优先算法中节点进出 OPEN 表的顺序是后进先出，OPEN 表是一个栈。

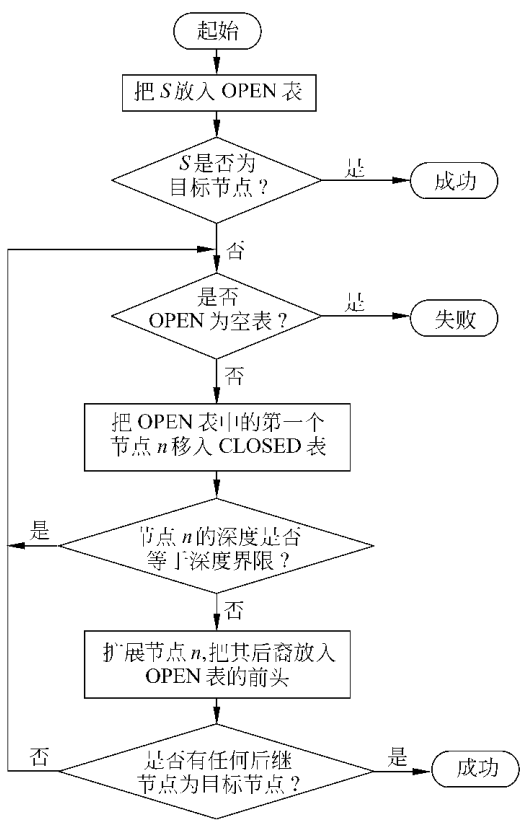

图 3.7　有界深度优先搜索算法框图

图 3.8 绘出按深度优先搜索生成的八数码难题搜索树，其中，设置深度界限为 5。粗线条的路径表明含有 5 条应用规则的一个解。从图可见，深度优先搜索过程是沿着一条路径进行下去，直到深度界限为止，然后再考虑只有最后一步有差别的相同深度或较浅深度可供选择的路径，接着再考虑最后两步有差别的那些路径，等等。

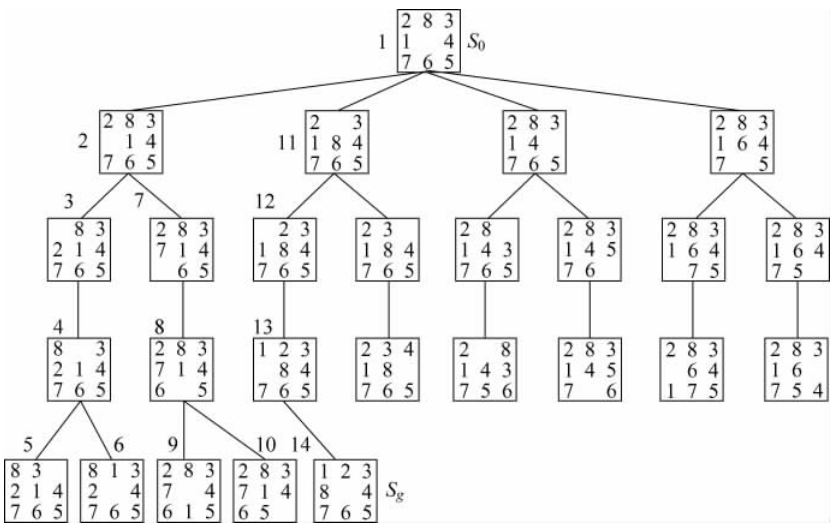

图 3.8 八数码难题的有界深度优先搜索树

3.2.3 等代价搜索

在宽度优先搜索中，假定每一步操作的代价都相同的情况下，它能找到最短路径，这条路径实际就是一条包含最少操作次数或应用算符的解。但对于很多问题来说，应用算符序列最少的解往往并不是想要的解，也不等同于最优解，通常人们希望找的是问题具有某些特性的解，尤其是最小代价解。搜索树中每条连接弧线上的有关代价以及随之而求得的具有最小代价的解答路径，与许多这样的广义准则相符合。宽度优先搜索可被推广用来解决这种寻找从起始状态至目标状态的具有最小代价的路径问题，这种推广了的宽度优先搜索算法叫做等代价搜索算法。如果所有的连接弧线具有相等的代价，那么等代价算法就简化为宽度优先搜索算法。在等代价搜索算法中，不是描述沿着等长度路径断层进行的扩展，而是描述沿着等代价路径断层进行的扩展。

在等代价搜索算法中，把从节点 i 到它的后继节点 j 的连接弧线代价记为 $c(i,j)$，把从起始节点 S 到任一节点 i 的路径代价记为 $g(i)$。在搜索树上，假设 $g(i)$也是从起始节点 S 到节点 i 的最少代价路径上的代价，因为它是唯一的路径。等代价搜索方法以 $g(i)$的递增顺序扩展其节点，其算法如下：

(1) 把起始节点 S 放到未扩展节点表 OPEN 中。如果此起始节点为一目标节点，则求得一个解，否则令 $g(S)=0$。

(2) 如果 OPEN 是个空表，则没有解而失败退出。

(3) 从 OPEN 表中选择一个节点 i,使其 $g(i)$为最小。如果有几个节点都合格,那么就要选择一个目标节点作为节点 i(要是有目标节点的话);否则,就从中选一个作为节点 i。把节点 i 从 OPEN 表移至扩展节点表 CLOSED 中。

(4) 如果节点 i 为目标节点,则求得一个解。

(5) 扩展节点 i。如果没有后继节点,则转向第(2)步。

(6) 对于节点 i 的每个后继节点 j,计算 $g(j)=g(i)+c(i,j)$,并把所有后继节点 j 放进 OPEN 表。提供回到节点 i 的指针。

(7) 转向第(2)步。

等代价搜索算法框图如图 3.9 所示。

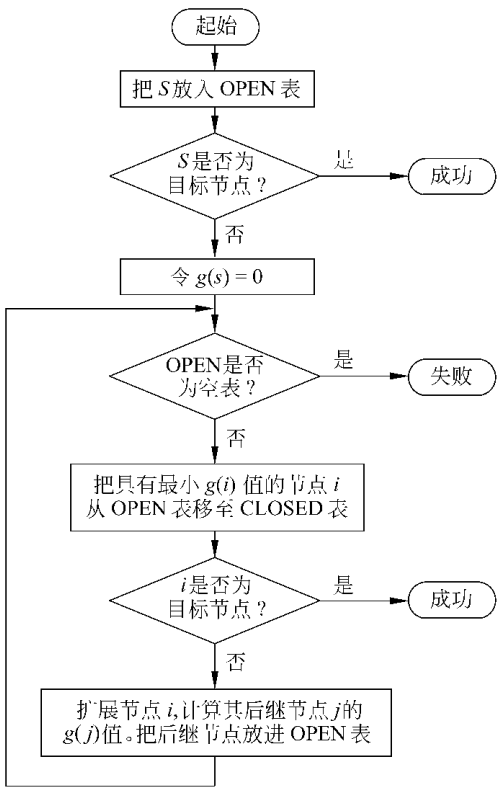

图 3.9　等代价搜索算法框图

3.3　启发式搜索

盲目搜索的效率低,耗费过多的计算空间与时间。如果能够找到一种方法用于排列待扩展节点的顺序,即选择最有希望的节点加以扩展,那么,搜索效率将会大为提高。在许多情况下,能够通过检测来确定合理的顺序。本节所介绍的搜索方法就是优先考虑这类检测。称这类搜索为启发式搜索(heuristic search)或有信息搜索(informed search)。

3.3.1 启发式搜索策略和估价函数

要在盲目搜索中找到一个解，所需要扩展的节点数目可能是很大的，因为这些节点的扩展次序完全是随意的，而且没有利用已解决问题的任何特性。因此，除了那些最简单的问题之外，一般都要占用很多时间或空间(或者两者均有)。这种结果是组合爆炸的一种表现形式。

有关具体问题领域的信息常常可以用来简化搜索。假设初始状态、算符和目标状态的定义都是完全确定的，然后决定一个搜索空间。因此，问题就在于如何有效地搜索这个给定空间。进行这种搜索的技术一般需要某些有关具体问题领域的特性的信息。把此种信息称为启发信息，并把利用启发信息的搜索方法叫做启发式搜索方法。

利用启发信息来决定哪个是下一步要扩展的节点。这种搜索总是选择“最有希望”的节点作为下一个被扩展的节点。这种搜索叫做有序搜索(ordered search)，也称为最佳优先搜索(best-first search)。

通常对于图的搜索问题总希望应使解路径的代价与求得此路径所需要的搜索代价的某些综合指标为最小，一个比较灵活(但代价也较大)的利用启发信息的方法是应用某些准则来重新排列每一步 OPEN 表中所有节点的顺序，然后，搜索就可能沿着某个被认为是最有希望的边缘区段向外扩展。应用这种排序过程，需要某些估算节点“希望”的量度，这种量度叫做估价函数(evaluation function)。估价函数的值越小，意味着该节点位于最优解路径上的“希望”越大，最后找到的最优路径即平均综合指标为最小的路径。

实际上，确定一种搜索方法是否比另一种搜索方法具有更强的启发能力的问题，往往就变成在实际应用这些方法的经验中获取有关信息的直观知识问题。

估价函数能够提供一个评定候选扩展节点的方法，以确定哪个节点最有可能在通向目标的最佳路径上。启发信息可用在 GRAPHSEARCH 第(8)步中来排列 OPEN 表上的节点，使得搜索沿着那些被认为最有希望的区段扩展。一个估量某个节点“希望”程度的重要方法是对各个节点使用估价函数的实值函数。估价函数的定义方法有很多，比如：试图确定一个处在最佳路径上的节点的概率；提出任意节点与目标集之间的距离量度或差别量度；或者在棋盘式的博弈和难题中根据棋局的某些特点来决定棋局的得分数。这些特点被认为与向目标节点前进一步的希望程度有关。

用符号 f 来标记估价函数，用 $f(n)$ 表示节点 n 的估价函数值。暂时令 f 为任意函数，以后将会提出 f 是从起始节点约束地通过节点 n 而到达目标节点的最小代价路径上的一个估算代价。

用函数 f 来排列 GRAPHSEARCH 第(8)步中 OPEN 表上的节点。根据习惯，OPEN 表上的节点按照它们 f 函数值的递增顺序排列。根据推测，某个具有低的估价值的节点较有可能处在最佳路径上。应用某个算法(例如等代价算法)选择 OPEN 表上具有最小 f 值的节点作为下一个要扩展的节点。这种搜索方法叫做有序搜索或最佳优先搜索，而其算法就叫做有序搜索算法或最佳优先算法。

3.3.2　有序搜索

有序搜索(ordered search)又称为最佳优先搜索(best-first search),它总是选择最有希望的节点作为下一个要扩展的节点。

尼尔逊(Nilsson)曾提出一个有序搜索的基本算法,该算法可以看成是启发式图搜索算法的一般策略。估价函数 f 是这样确定的:一个节点的希望程度越大,其 f 值就越小。被选为扩展的节点,是估价函数最小的节点。

有序状态空间搜索算法如下:

(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 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。如果有几个节点合格,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 是一个目标节点,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,又不在 CLOSED 表中,则用估价函数 f 把它添入 OPEN 表。从 j 加一指向其父节点 i 的指针,以便一旦找到目标节点时记住一个解答路径。

　(c) 如果 j 已在 OPEN 表或 CLOSED 表中,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,而不是指向它的父节点。

　　(iii) 如果节点 j 在 CLOSED 表中,则把它移回 OPEN 表。

(7) 转向(2),即 GO TO(2)。

步骤(6.c)是一般搜索图所需要的,该图中可能有一个以上的父辈节点。具有最小估价函数值 $f(j)$的节点被选为父节点。但是,对于树搜索来说,它最多只有一个父节点,所以步骤(6.c)可以略去。值得指出的是,即使搜索空间是一般的搜索图,其显式子搜索图总是一棵树,因为节点 j 从来没有同时记录过一个以上的父节点。

有序搜索算法框图示于图 3.10。

宽度优先搜索、等代价搜索和深度优先搜索都是有序搜索技术的特例。对于宽度优先搜索,选择 $f(i)$作为节点 i 的深度。对于等代价搜索,$f(i)$是从起始节点至节点 i 这段路径的代价。

当然,与盲目搜索方法比较,有序搜索的目的在于减少被扩展的节点数。有序搜索的有效性直接取决于 f 的选择,这将敏锐地辨别出有希望的节点和没有希望的节点。不过,如果这种辨别不准确,那么有序搜索就可能失去一个最好的解甚至全部的解。如果没有适用的、准确的希望量度,那么 f 的选择将涉及两方面的内容:一方面是一个时间和空间之间的折中方案;另一方面是保证有一个最优的解或任意解。