第8章 盲目搜索

8.1 用公式表示状态空间

很多实际问题的搜索空间是如此之大，以致它们不能通过显式图来表示。本章需要上一章所讲到的基本搜索过程的详细细节。首先，我们非常关心如何用公式来表示这些搜索问题；第二，我们必须找到隐式表示大的搜索图的方法；第三，我们需要用高效的算法来对这些大图进行搜索。

在一些规划的问题中，如堆积木，想像代表各种环境状态的数据结构和改变它们的动作是不难的，然而，找到可管理状态空间图的表示是很难的。为了做到这一点，必须仔细分析问题——考虑对称性，忽略不相关的细节，发现适当的抽象等。不幸的是，建立一个搜索问题的任务仍然是一个需要人工参与的大问题。

除了agent堆积木等问题外，数码问题也常被用来演示如何在状态空间中生成动作序列。考虑到多样化，将在本章和下一章用到这种问题。一个典型的例子是15数码问题，它由放在一个4×4的方阵中的15个数码构成，其中的一个单元是空的，它的周边单元中的数码可以移到该单元中。此问题的任务是找到一个数码移动序列使初始的无序数码转变为一些特殊的排列。8数码问题是一个简化版本，在一个3×3的方阵中有8个数码。假定该问题的目标是把数码从开始状态移动到目标状态，如图8-1所示。

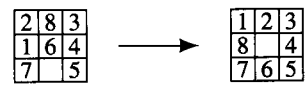


图8-1 8数码问题的开始和目标状态

在这个问题中，一个明显的图标状态描述是一个3×3的方阵，方阵的每个单元中包含1~8之间的数字或一个代表空格的符号。目标状态是图8-1中右边的方阵。状态间的移动就是把一个数码移到空的单元中。一般地讲，在构造一个问题的状态空间时我们有一些代表性的选择。在8数码问题中，我们可以想像有8×4个不同的移动，它们是：1上移、1下移、1左移、1右移、2上移、……，等等（当然在一个给定的状态中，并不是所有的这些移动都是可能的）。一个更精练的公式只有4个移动，即：空格左移、空格上移、空格右移和空格下移。一个给定的开始状态和一组可能的移动隐式地定义了从开始状态可到达的一个状态图。在8数码表示的状态空间中节点数是9！=362 880个（8数码状态空间刚好可以分成两个独立的图；一个图中的数码不能从其他图中的状态到达）。

8.2 隐式状态空间图的组成

从开始状态可以到达的状态空间图部分是通过开始状态描述和可能在任何状态下采用的动作结果描述来隐式表示的，因此，从理论上讲，可以把一个图的隐式表示转换为显式表示。为此，可以产生开始节点的所有后继节点（通过那个节点应用所有可能的算子），然后再生成所有后继节点的所有后继节点，等等。对那些太大以至不能显式表示的图，搜索过程只需要生成要求的状态空间，以发现到达目标的路径。过程在发现了一个可以接受的目标路径时终止。

有三个基本部分参与表示隐式状态空间图：

1) 一个标识开始节点的描述。这个描述是对环境初始状态建模的一些数据结构。

2) 把代表环境状态的状态描述转换成代表动作后状态描述的转换函数。这些函数也常被叫做算子。在agent问题中，它们是动作结果的模型。当一个算子应用到一个节点时，它产生该节点的一个后继节点。

3) 目标状态，可以是状态描述中的一个真假值函数，或者是和目标状态一致的状态描述的真实实例列表。

我们将学习两类主要的搜索过程。其中之一，我们没有指定问题的任何推理信息，例如要搜索这一部分而不是另一部分，就像到目前为止的只要发现一条到目标的路径即可。这种过程被称为是盲目的。另一种，我们指定了要解决问题的信息以帮助集中搜索。这个过程叫启发式(*heuristic*)搜索[⊖]。本章将讨论盲目搜索过程，下一章再讨论启发式搜索过程。

8.3 广度优先搜索

盲目搜索过程只把算子应用到节点，它没有使用问题领域的任何特殊知识（除了关于什么

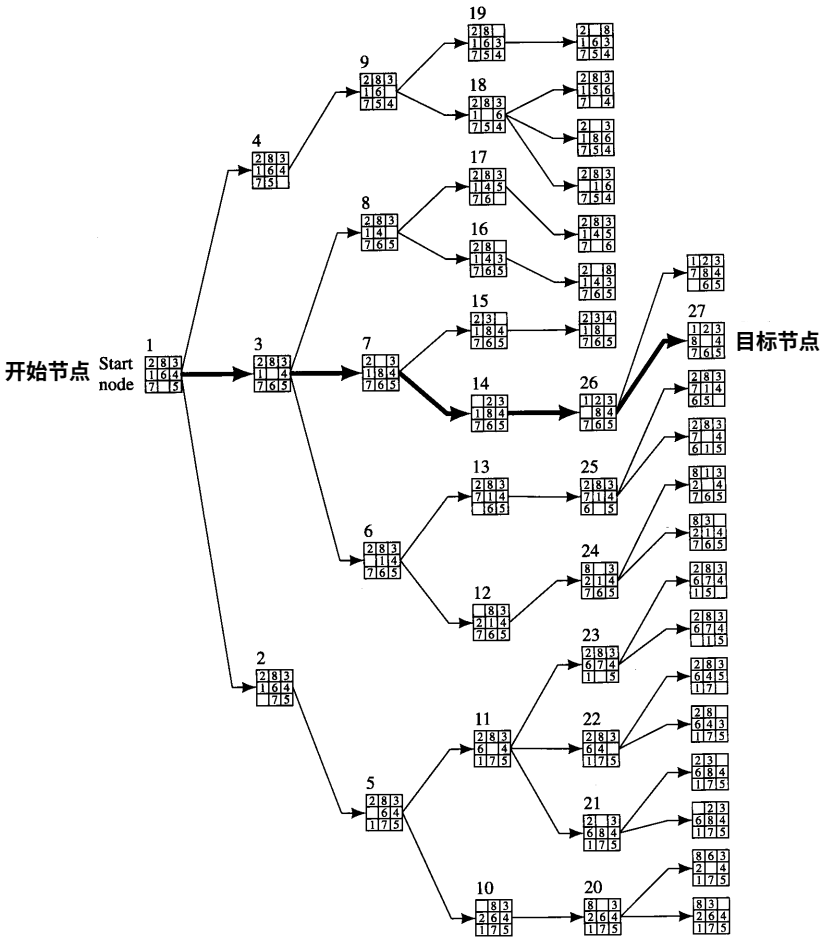


图8-2 8数码问题的广度优先搜索

⊖ heuristic 一词来源于希腊语heuriskein，意思是“发现”。外来语Eureka! (“我已发现了它!”) 和heuristic同源。

动作是合法的知识外)。最简单的盲目搜索过程就是广度优先搜索 (*breadth-first search*)。该过程把所有的算子应用到开始节点以产生一个显式的状态空间图，再把所有可能的算子应用到开始节点的所有直接后继，再到后继的后继，等等。搜索过程一律从开始节点向外扩展。由于每一步将所有可能的算子应用到一个节点，因此可把它们组成一个叫后继函数 (*successor function*)的函数。当把后继函数应用到一个节点时，产生一个节点集，该节点集就是把所有能应用到那个节点的算子应用到该节点而产生的。一个节点的后继函数的每一次应用称为节点的扩展(*expanding*)。

图8-2显示了8数码问题经广度优先搜索所产生的节点集。标出开始和目标节点，节点扩展顺序由一个数字表示在节点的旁边。相同深度的节点按照一些固定的顺序扩展。在扩展一个节点时，按空格左移、上移、右移和下移的顺序应用算子。尽管每个移动都是可逆的，但删去了从后继到双亲的弧。解答路径在图中用黑线表示。如我们将看到的一样，广度优先搜索的一个特征是：当发现目标节点时，我们已经找到了到达目标的一条最短路径。然而广度优先搜索的一个缺点是它要求产生和存储一个大小是最浅目标节点深度的指数的树。

相同代价搜索(*uniform-cost search*)[Dijkstra 1959]是广度优先搜索的一种变体，在该方法中，节点从开始节点顺着代价等高点向外扩展，而不是顺着相同深度等高线。如果图中所有弧的代价相同（比如说等于 1），那么相同代价搜索就和广度优先搜索一致。反过来，相同代价搜索可以看作是下一章要讲的启发式搜索的一个特殊情况。广度优先和相同代价搜索方法的简要描述只给出了它们的主要思想，但是要解决其他复杂的情况则需要技术改进。比如一个节点扩展产生的节点在前面的搜索过程中已经到达过。将在下一章介绍了更一般的算法后再讨论这些方法。

8.4 深度优先或回溯搜索

深度优先搜索一次对节点应用一个算子以产生该节点的一个后继。每一个节点都留下一个标记，用来指示如果需要时所必需的附加算子。对每一个节点，必须有一个决策来决定哪个算子先用，哪个次之等等。只要一个后继产生，它的下一个后继就会被生成，一直向下传下去，等等。为了防止从开始节点的搜索过程深度太深，需要一个深度约束 (*depth bound*)。超过这个深度约束时不再产生后继（假定没有任何目标节点超过这个深度约束）。这种限制允许我们忽略搜索图的一部分，这些部分已经确定不能高效地到达目标节点。

用8数码和深度约束为5为例说明这个过程。我们再次以下列顺序：空格左移、上移、右移和下移来应用算子，忽略从后继到双亲的弧。在图 8-3a中给出了开始的几个节点，每个节点左边的数字是该节点产生的顺序。也留下了一些弧指示那些还没有完全展开的节点。在节点 5，我们到达了深度约束点但还没有到达目标，因此，考虑下一个最近产生的但还没有完全展开的节点4，生成它的另一个后继节点6（见图8-3b)。这时，可以抛弃节点5，因为我们不会再在它下面产生节点。节点6也在深度约束点且不是目标节点，因此我们考虑下一个最近产生、但没有完全展开的节点2，所产生的节点7 —— 抛弃节点3和它的所有后继，因为我们不再产生它们下面的任何节点。返回节点2是年历回溯(*chronological backtracking*)的一个例子。当再次到达深度约束时，产生了图 8-3c。这时没有到达目标节点，我们就生成节点 8的另一个后继，它也不是目标节点。因此，我们抛弃节点8和它的后继，回溯产生节点7的另一个后继。继续这个过程，最终产生图8-4。

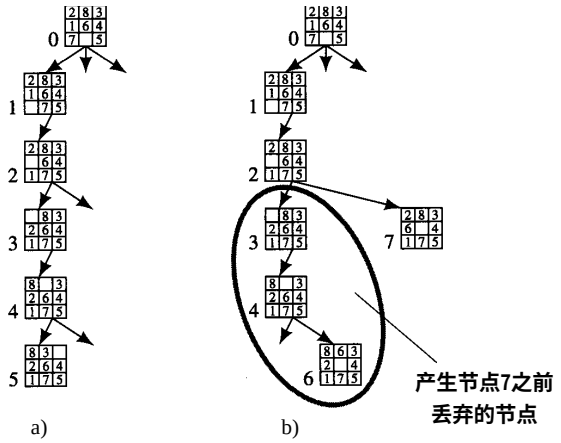


图8-3 深度优先搜索最初几个节点的产生

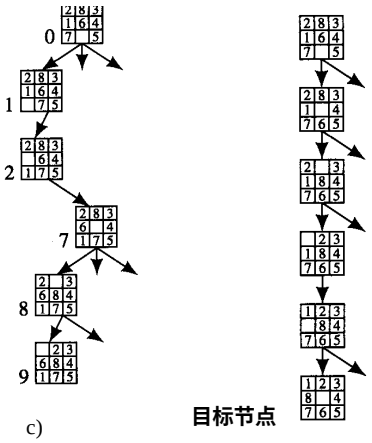


图8-4 深度优先搜索中到达目标节点时的图

深度优先算法使我们只保存搜索树的一部分，它由当前正在搜索的路径和指示该路径上还没有完全展开的节点标志构成。因此，深度优先搜索的存储器要求是深度约束的线性函数。深度优先搜索的一个缺点是当发现目标时，我们不能保证找到的路径是最短长度。另一个缺点是如果只有一个很浅的目标，且该目标位于搜索过程的后部时，也必须浏览大部分搜索空间。

8.5 迭代加深

一种叫做迭代加深(*iterative deepening*)[Korf 1985, Stickel & Tyson 1985]的技术既能满足深度优先搜索的线性存储要求，同时又能保证发现一个最小深度的目标节点（如果一个目标节点能被发现）。在迭代加深方法中，连续的深度优先搜索被引入，每一个的深度约束逐次加 1，直到一个目标节点被发现。图 8-5说明迭代加深是如何在一个简单树上工作的。令人惊奇的是，由迭代加深搜索扩展产生的节点数并不比广度优先搜索产生的多很多。我们可以计算一个有相同分枝因子的树在最坏搜索情况下所生产的节点数，这个树的最浅目标节点在深度 d处，并且是该深度最后一个生产的节点。由广度优先搜索扩展产生的节点数是：

$$N_{\mathrm{bf}} = 1 + b + b^2 + \cdots + b^d = \frac{b^{d+1} - 1}{b - 1}$$

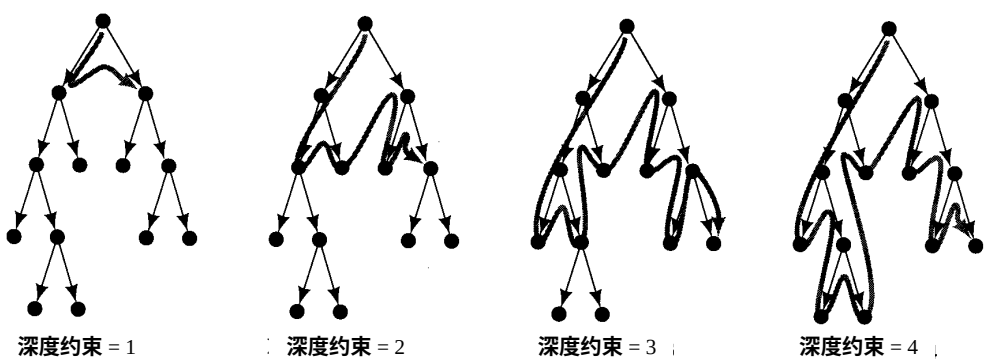


图8-5 迭代加深搜索过程

为了计算由迭代加深扩展来的节点数，我们首先指出，由一个完全的深度优先搜索搜索到第j级而扩展成的节点数是：

$$N_{\mathrm{df}_j} = \frac{b^{j+1} - 1}{b - 1}$$

在最坏的情况下，对深度为d的目标进行迭代加深搜索时必须实施分裂，完全进行深度优先搜索直到深度为d。由所有这些搜索扩展而来的节点数之和是：

$$\begin{aligned} N_{\mathrm{id}} &= \sum_{j=0}^{d} \frac{b^{j+1} - 1}{b - 1} \\ &= \frac{1}{b-1}\left[b\left(\sum_{j=0}^{d} b^j \right) - \sum_{j=0}^{d} 1 \right] \\ &= \frac{1}{b-1}\left[b\left(\frac{b^{d+1} - 1}{b - 1} \right) - (d+1) \right] \end{aligned}$$

简化上式可得到

$$N_{\mathrm{id}} = \frac{b^{d+2} - 2b - bd + d + 1}{(b-1)^2}$$

对大的d而言，$N_{\mathrm{id}}/N_{\mathrm{bf}}$的比率是$b/(b-1)$。对一个分枝因子为10的深度目标，迭代加深搜索只比广度优先搜索多了大约11%的扩展节点。

另外，当有很多目标节点时，一种称为迭代加宽（*iterative broadening*）的技术也很有用。[Ginsberg & Harvey 1992, Harvey] 1994]描述了这个方法。

8.6 补充读物和讨论

对年代回溯方法有各种改进方法，如[Stallman & Sussman 1977]的Dependency-directed回溯，[Gaschnig 1979] 的backjumping和 [Ginsberg 1993]的dynamic backtracking。[Ginsberg 1993]比较了这些方法，并指出了动态回溯的优越性。这些扩展的回溯技术常用于约束满足问题中（第11章将讲到）。

习题

8.1 在水壶问题中，给定3公升和4公升的水壶各一个，分别叫做3和4 。开始时，3和4都是空的。两个壶都可以从水龙头T中灌水，也可以把水从两个壶中倒到排水沟D中。水可以从一个壶倒入另一个壶中。没有其他的度量器。我们要找到一套方法让4中的水刚好是2公升。[不要害怕！这里就有一个办法：(a) 把3用水龙头灌满；(b)把3中的水倒入4中；(c)把3用水龙头灌满；(d)用3中的水把4倒满；(e)倒掉4中的水；(f)把3中的水倒入4中。]

1) 给水壶问题建一个状态空间搜索公式：
   a) 把初始的图标状态描述作为一个数据结构。
   b) 给定状态图的一个目标条件作为对数据结构的一些测试。
   c) 命名状态算子，准确描述每个算子对状态图的操作。
2) 画出所有不同状态空间节点的图；这些节点在开始节点的三步移动之内，用状态描

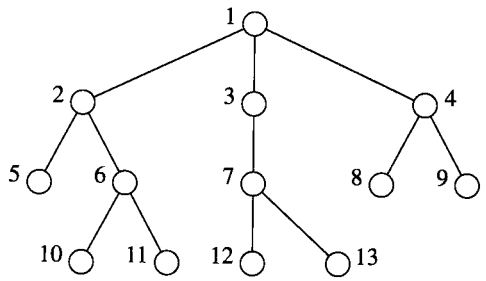
述标出每一个节点，画出图中到每个节点的至少一条路径。用合适的算子标出每一条弧。除了这些节点，画出解决方法中所有的节点和弧（被正确标识）。

8.2 列出下图中树的节点访问序列以满足下面的三个搜索策略 (在所有情况中都选择最左分枝优先访问)：

1) 深度优先搜索；

2) 深度优先迭代加深搜索（每个迭代深度加 1）；

3) 广度优先搜索。



8.3 考虑一棵有限树，深度为 d，分枝因子为 b(一棵只有一个根节点的树， 其深度为 0； 一棵由一个根节点和它的 b 个后继组成的树，其深度为 1；等等)。假设， 最浅目标节点在深度 g ($g \leq d$)。

1) 深度约束为 d、深度优先搜索所产生的，节点的最大和最小数目分别是什么？

2) 广度优先搜索时产生的节点的最大和最小数目是什么？

3) 由深度优先迭代加深搜索产生的节点的最大和最小数目是什么？(假定初始深度限制是 1，如果当前深度限制下没找到目标节点，将深度加 1 继续搜索)。

8.4 假设在广度优先搜索中每秒产生 10 000 个节点，存储每个节点需要 100 个字节。对一个深度为 d、分枝因子为 5 的树进行完全广度优先搜索，需要多少空间和时间？把这些值放在一个表中(你可用一个大概的数字，时间最好表示为小时、天、月、年等等)。

8.5 假定我们正在对一个分枝因子为 b 的树进行搜索，然而我们不知道我们真的在搜索一个树，因此考虑检查每一个产生的状态描述看它是否匹配以前产生的状态描述。在一棵深度为 d 的树中进行搜索，将会有多少个这种检查？