第9章　启发式搜索

9.1　使用评估函数

除了搜索过程不是从开始节点统一向外扩展外，本章描述的搜索过程有点像广度优先搜索，不同的是，它会优先顺着有启发性和具有特定信息的节点搜索下去，这些节点可能是到达目标的最好路径。我们称这个过程为最优（*best-first*）或启发式搜索。下面是其基本思想：

1) 假定有一个启发式(评估)函数 $\hat{f}$ ，可以帮助确定下一个要扩展的最优节点（给*f*加一个“帽子”的原因后面就会清楚，称 $\hat{f}$ 为“f-hat”）。我们采用一个约定，即 $\hat{f}$ 的值小表示找到了好的节点。这个函数基于指定问题域的信息，它是状态描述的一个实数值函数。

2) 下一个要扩展的节点*n*是 $\hat{f}$ (*n*)值最小的节点(本章假定节点扩展产生一个节点的所有后继)。

3) 当下一个要扩展的节点是目标节点时过程终止。

我们常常可以为最优搜索指定好的评估函数。如在8数码问题中，可以用不正确位置的数字个数作为状态描述好坏的一个度量：

$\hat{f}$ (*n*) = 位置不正确的数字个数(和目标相比)

在搜索过程中采用这个启发式函数将产生图9-1所示的图，每个节点的数值是该节点的 $\hat{f}$ 值。

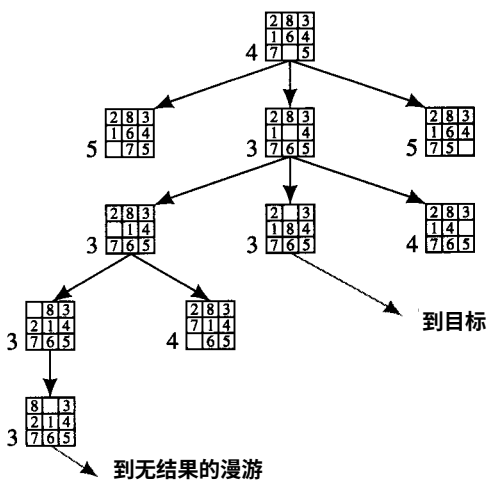


图9-1　一个启发式搜索过程的一个可能结果

这个例子表明，在搜索过程中我们需要偏向有利于回溯到早期路径的搜索(为了避免由于过分的优化试探而陷入“花园小径”)。因此我们加了一个“深度因子”给 $\hat{f}$ ： $\hat{f}(n)=\hat{g}(n)+\hat{h}(n)$ ， $\hat{g}$(n)是对图中节点*n*的“深度”估计(即从开始节点到*n*的最短路径长度)， $\hat{h}$ (*n*)是对节点*n*的启发或评估。像前面一样，如果 $\hat{h}$(n) = 不正确位置的数字个数(和目标相比)， $\hat{g}$(n) = 搜索图中节点*n*的深度，我们可以得到图9-2。在这个图中，把 $\hat{g}$(n)和 $\hat{h}$(n) 的值写在每个节点的旁边，在这种情况

下，可以看到搜索相当直接地朝着目标进行(除了用圆圈标注的节点外)。

这些例子提出了两个重要的问题。第一，我们如何为最优搜索决定评估函数？第二，最优搜索的特性是什么？它能找到到达目标节点的好路径吗？本章的结尾将会提到关于评估函数选择的一些指导，第10章将会解释关于自动学习评估函数的一些方法。本章主要讨论最优搜索的形式表示。作为特殊的例子，我开发了一个包括最优搜索版本的一般图搜索算法（为了更详细地了解启发式搜索，可以参考引用的文章和Pearl写的书[pearl 1984]）。

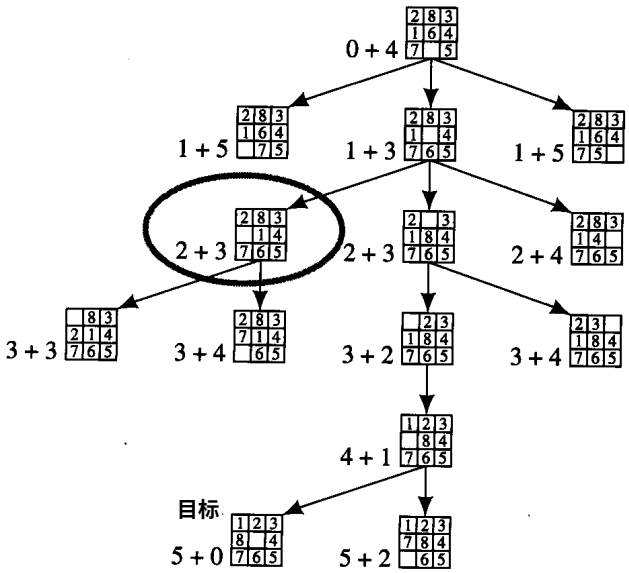


图9-2 使用 $\hat{f}(n)=\hat{g}(n)+\hat{h}(n)$ 的启发式搜索

9.2 一个通用的图搜索算法

为了更准确地解释本章的启发式搜索过程，这里提出一个通用的图搜索算法，它允许各种用户——偏爱启发式的或盲目的，进行定制。我把这个算法叫做图搜索（GRAPHSEARCH）。下面是（第一个版本）它的定义。

GRAPHSEACH

1) 生成一个仅包含开始节点n_0的搜索树Tr。把n_0放在一个称为*OPEN*的有序列表中。
2) 生成一个初始值为空的列表*CLOSED*。
3) 如果*OPEN*为空，则失败并退出。
4) 选出*OPEN*中的第一个节点，并将它从*OPEN*中移出，放入*CLOSED*中。称该节点为n。
5) 如果n是目标节点，顺着*Tr*中的弧从n回溯到n_0找到一条路径，获得解决方案，则成功退出（弧在第6步产生）。
6) 扩展节点n，生成n的后继节点集M。通过在*Tr*中建立从n到M中每个成员的弧生成n的后继。
7) 按照任意的模式或启发式方式对列表*OPEN*重新排序。
8) 返回步骤3 。

这个算法可用来执行最优搜索、广度优先搜索或深度优先搜索。在广度优先搜索中，新节点只要放在*OPEN*的尾部即可（先进先出，FIFO），节点不用重排。在深度优先搜索中，新节

点放在*OPEN*的开始（后进先出，LIFO）。在最优（也叫启发式）搜索中，按照节点的启发式方式来重排*OPEN*。

9.2.1 算法A*

用最优搜索算法详细说明GRAPHSEARCH。最优搜索算法根据函数 $\hat{f}$ 的增加值，(在第7步)重排*OPEN*中的节点，如8数码问题。把GRAPHSEACH的这种算法称为算法A*。下面将会看到，定义使A*执行广度搜索或相同代价搜索的函数 $\hat{f}$ 是可行的。为了确定要用的函数 $\hat{f}$ 族，必须先介绍一些其他符号。

设$h(n)$ =节点n和目标节点（遍及所有可能的目标节点以及从 n到它们的所有可能路径）之间的最小代价路径的实际代价。

设$g(n)$ = 从开始节点n_0到节点n的一个最小代价路径的代价。

那么$f(n) = g(n) + h(n)$就是从n_0到目标节点并且经过节点n的最小代价路径的代价。注意$f(n_0) = h(n_0)$是从n_0到目标节点的一个（不受限的）最小代价路径的代价。

对每个节点n，设 $\hat{h}(n)$（启发因子）是$h(n)$的某个估计，$\hat{g}(n)$（深度因子）是由A*发现的到节点n的最小代价路径的代价。在算法A*中，我们用 $\hat{f} = \hat{h} + \hat{g}$。注意，如果算法A*中的 $\hat{h}$ 恒等于0，就成为相同代价搜索。这些定义示例在图9-3中。

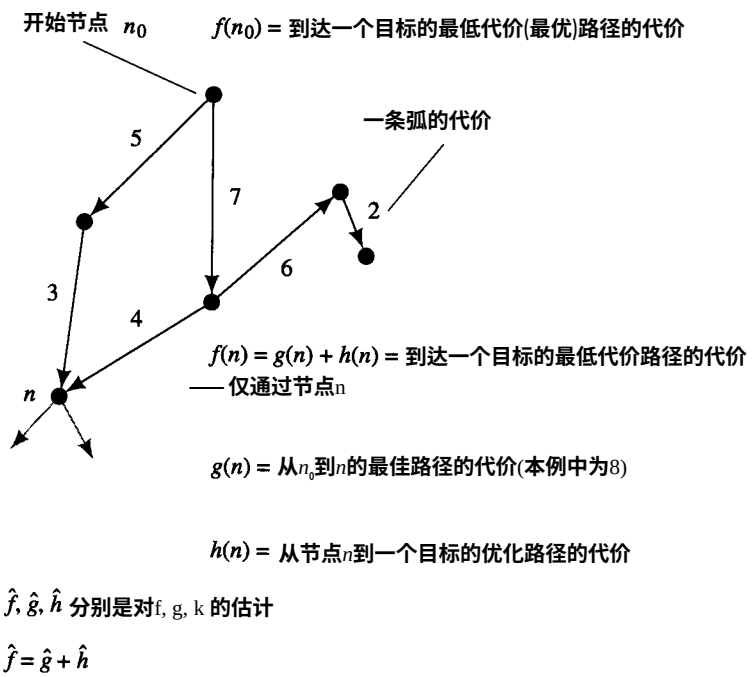


图9-3 后发式搜索符号

图9-2所示的8数码搜索过程是A*应用的一个例子。在这里，我们假定了单位弧代价，因此$g(n)$就是图中节点n的深度。尽管稍微有一点复杂，但在算法A*的定义中忽略了它们。如果要搜索的隐式图不是一棵树会怎样呢？也就是说，假如有超过一个动作序列能从开始状态到达相同的环境状态。例如，8数码隐式图显然就不是一棵树，因为动作是可逆的 —— 即任何节点n的每一个后继都可以使n作为它的一个后继。在建立8数码搜索树中忽略了这些循环。在那种情况下，

它们容易被忽略，只要在节点的后继中不包括它的双亲就行了。把GRAPHSEARCH 中的第6步改为：

6) 扩展节点n，生成后继集合 $\mathcal{M}$，n的双亲不能在 $\mathcal{M}$中。通过在Tr中建立从n到 $\mathcal{M}$中每个成员的弧生成n的后继。

考虑到更长的循环，我们把6改为：

6) 扩展节点n，生成后继集合 $\mathcal{M}$，n的祖先不能在 $\mathcal{M}$中。通过在Tr中建立从n到 $\mathcal{M}$中每个成员的弧生成n的后继。

当然，为了检查这些更长的循环，我们要检查标识节点 n的祖先的数据结构。对复杂的数据结构，这一步增加了算法的复杂度。

修改第6步的算法在搜索目标路径时避免了再循环，但是仍有可能通过不同的路径到达相同的环境状态。解决这种问题的一个方法就是简单地忽略它。也就是说，算法不检查 $\mathcal{M}$中的一个节点是否已经在$OPEN$或$CLOSED$中。因此算法对那种可能性就健忘了，以致它可能通过不同的路径到达相同的节点。这个“相同节点”在 Tr中重复多次，重复的次数是算法发现到达该节点的不同路径数目。如果Tr中的两个节点被同样的数据结构标识，那么在它们下面有相同的子树。也就是说，算法会重复搜索结果。后面我们将看到，对 $\hat{f}$ 施加适当的条件，当A*首次扩展Tr中的节点n时，它已经找到了到达节点n并且有最小值 $\hat{f}$ 的一条路径。

为了防止在前述条件不成立下 $\hat{f}$ 的重复搜索，需要对算法 A*进行一些修改。因为搜索可以顺着不同的路径到达相同的节点，因此算法 A*会产生一个搜索图 G。G是A*扩展开始节点、开始节点的后继等等而产生的节点和弧的结构。 A*也包含一个搜索树 Tr。Tr是G的一个子图，它是到达搜索图中所有节点的最优（最小代价）路径生成树。图 9-4显示了一个单位弧代价的图。图9-4a表示一个早期搜索阶段，搜索图的搜索树部分用黑色的弧表示，灰色的弧不在搜索树中。注意图9-4a中的节点4有两条路径可以到达；两条路径都在搜索图中，但只有一条在搜索树中。我们保留搜索图是因为后面的搜索过程可能会发现更短的路径，这些路径使用了在早期搜索图中、但不在早期搜索树中的弧。例如图 9-4b，扩展节点1会发现到达节点2和它的子孙（包括节点4）的一条更短路径，因此搜索树会相应地改变。

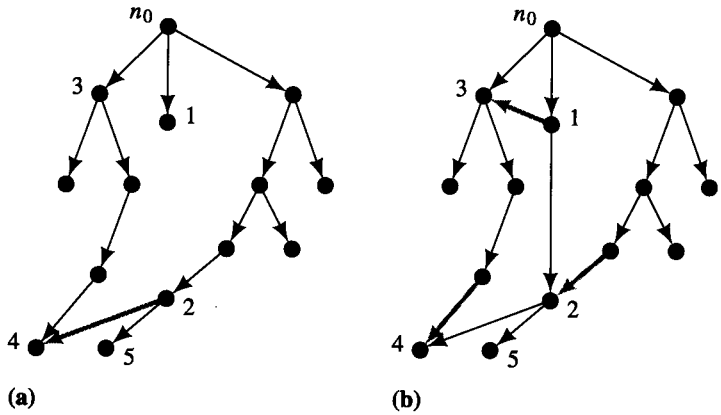


图9-4 搜索图和搜索过程产生的树

为了完整起见，阐述一下包含搜索图的 A*算法。然而，应该注意，很少需要这个算法，因为我们常常对 $\hat{f}$ 施加一定的条件以保证当算法 A*扩展一个节点时，它已经发现了到达该节

点的最小代价路径。

算法A*：

1) 生成一个只包含开始节点n_0的搜索图G，把n_0放在一个叫$OPEN$的列表上。
2) 生成一个列表$CLOSED$，它的初始值为空。
3) 如果$OPEN$为空，则失败退出。
4) 选择$OPEN$上的第一个节点，把它从$OPEN$中移入$CLOSED$，称该节点为n。
5) 如果n是目标节点，顺着G中，从n到n_0的指针找到一条路径，获得解决方案，成功退出（该指针定义了一个搜索树，在第7步建立）。
6) 扩展节点n，生成其后继节点集$\mathcal{M}$，在G中，n的祖先不能在$\mathcal{M}$中。在G中安置$\mathcal{M}$的成员，使它们成为n的后继。
7) 从$\mathcal{M}$的每一个不在G中的成员建立一个指向n的指针（例如，既不在$OPEN$中，也不在$CLOSED$中）。把$\mathcal{M}$的这些成员加到$OPEN$中。对$\mathcal{M}$的每一个已在$OPEN$中或$CLOSED$中的成员m，如果到目前为止找到的到达m的最好路径通过n，就把它的指针指向n。对已在$CLOSE$中的$\mathcal{M}$的每一个成员，重定向它在G中的每一个后继，以使它们顺着到目前为止发现的最好路径指向它们的祖先。
8) 按递增$\hat{f}$值，重排$OPEN$（相同最小$\hat{f}$值可根据搜索树中的最深节点来解决）。
9) 返回第3步。

在第7步中，如果搜索过程发现一条路径到达一个节点的代价比现存的路径代价低，我们就要重定向指向该节点的指针。已经在$CLOSED$中的节点子孙的重定向保存了后面的搜索结果，但是可能需要指数级的计算代价。因此，第7步常常不会实现。随着搜索的向前推进，其中有些指针最终将会被重定向。

9.2.2 A*的可接纳性

对图和$\hat{h}$施加一些条件可以保证应用到图的算法A*总能找到最小代价路径。图的条件是：

1) 图中每个节点的后继是有限的（如果有的话）；
2) 图中所有弧的代价都大于某个正数 。

$\hat{h}$的条件是：

3) 对搜索图中的所有节点n，$\hat{h}(n) \le h(n)$。也就是说，$\hat{h}$决不会超过实际值h的估计。（这样的$\hat{h}$函数有时被称为优化概算机）

一般来说，找到满足这个低约束条件的$\hat{h}$并不困难。例如，在城市路由问题中，从城市n到目标的直线距离就是从n到目标节点的最佳路径距离的一个最低约束。在8数码问题中，不在正确位置的数码个数就是到达目标步数的一个最小约束。

用这三个约束条件，只要存在到达目标的路径，算法A*可以保证找到一条到达目标的最佳路径。把这个结果作为一个定理（下面给出该定理的一个新版证明，原始证明参见[Hart，Nilsson，& Raphael 1968]。

定理9.1 如果图和$\hat{h}$具有上述的稳定条件，而且从开始节点n_0到目标节点有一条有限代价的路径，那么算法A*保证终止于到达目标的一条最小代价路径。

证明：证明的主要部分是下面的重要引理。为了获得关于A*为什么能保证发现最优路径

的直觉知识，完全理解这个引理是重要的。

引理9.1 在A*终止前的每一步，总是有一个节点n^*，它$OPEN$上有下面的特性：

1) n^*在到达目标的一条最佳路径上。

2) A*已经发现了到达n^*的一条最佳路径。

3) $\hat{f}(n^*) \leq f(n_0)$

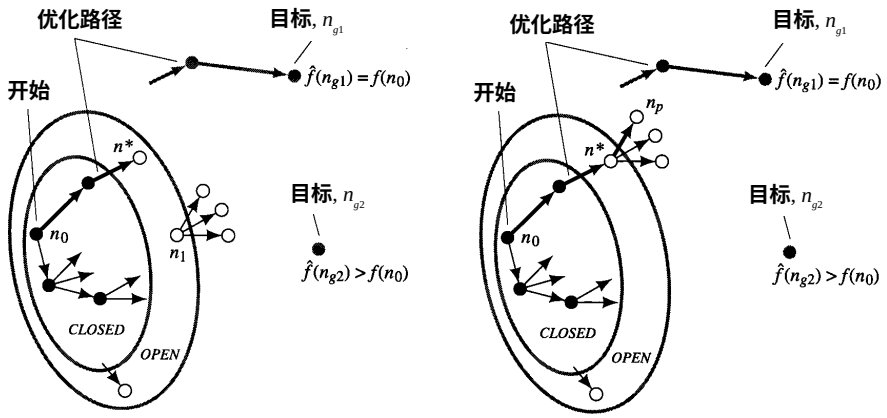
引理证明：为了证明A*每一步保证引理结论，只要证明（1）在算法开始时结论正确；（2）如果一个节点扩展前结论正确，那么节点扩展后结论继续正确。我们称这种证明方法为数学归纳法。下面是证明过程：

基本条件：在搜索开始时（当节点n_0准备扩展时），n_0在$OPEN$上，它是到达目标的一条最佳路径，A*已经发现了这条路径，而且，因为 $\hat{f}(n_0) = \hat{h}(n_0) \leq f(n_0)$，故 $\hat{f}(n_0) \leq f(n_0)$。

因此，在该阶段，节点n_0可以是引理中的节点n^*。

归纳步骤：假设在节点$m(m \geq 0)$扩展时引理的结论是正确的，（用这个假设）证明在节点$m+1$扩展时引理是正确的。参考图9-5将有助于我们证明归纳步骤。

设n^*是m个节点扩展后，A*发现的一个最佳路径上的假设节点，它在$OPEN$上。如果在第$(m+1)$步，n^*没有被选择扩展（图9-5a），n^*的属性和以前相同，因此证明了归纳步骤。如果n^*被选为扩展点（图9-5b），它的所有新后继将被放在$OPEN$上，它们中至少有一个n_p，将会在到达目标的最优路径上（由于假定一个最优路径通过 n^*，它必须继续通过它的一个后继）。故A*找到了到达n_p的一条最佳路径，因为如果有到达n_p的一条更好路径，那么这条路径也是到达目标的更好路径（这和我们的假设没有比通过 n^*到达目标的路径更好的路径相矛盾）。这样，让n_p成为第$(m+1)$步的新n^*，除了 $\hat{f}(n^*) \leq f(n_0)$外，我们已经证明了归纳步骤。



a) A* 选择节点n_1作为要扩展的第$(m+1)$个节点　　b) A* 选择节点n^*作为要扩展的第$(m+1)$个节点

图9-5 当第$(m+1)$个节点要扩展时的情况

现在证明 $\hat{f}(n^*) \leq f(n_0)$。对在一条最佳路径上的节点n^*，A*已经找到了一条到达n^*的最佳路径，我们有：

$\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$，根据定义

$\leq g(n^*) + h(n^*)$　　因为 $\hat{g}(n^*) = g(n^*)$，$\hat{h}(n^*) \leq h(n^*)$

$\leq f(n^*)$　　根据定义$g(n^*) + h(n^*) = f(n^*)$

$\leq f(n_0)$　　由于n^*在一条最佳路径上，故$f(n^*) = f(n_0)$

这就完成了引理的证明。

继续证明定理，我们首先证明如果存在可以到达的目标， A*必须终止，然后证明A*终止于找到一条到达目标的最佳路径。

- A*必须终止：假如它不会终止。在这种情况下， A*始终不断地扩展*OPEN*上的节点，最终它将在搜索树上扩展比任何预设的深度约束更深的节点，因为我们已经假设正在搜索的图有有限个分枝因子。由于每个弧的代价都比 >0大，故在*OPEN*中的所有节点的 $\hat{g}$ (因此 $\hat{f}$)值将最终超过$f(n_0)$。这将和引理产生矛盾。
- A*终止于一条最优路径：A*只能在第3步（*OPEN*为空）或第5步（在一个目标节点）终止。一个第3步的终止只能出现在不包含任何目标节点的有限图中， 只要有一个目标节点，定理都声称找到到达目标的一条最佳路径。因此， A*终止于找到一个目标节点。假如它终止于发现了一个非最佳目标n_{g2}，$f(n_{g2}) > f(n_0)$， 当有一个最佳目标n_{g1}时，$n_{g1} \neq n_{g2}$，$f(n_{g1}) = f(n_0)$（见图9-5）。在n_{g2}终止时， $\hat{f}(n_{g2}) \geq f(n_{g2}) > f(n_0)$。但是在A*选择$n_{g2}$进行扩展之前，根据引理在*OPEN*上有一个节点$n^*$，它在最优路径上，且 $\hat{f}(n^*) \leq f(n_0)$。因此，A*不可能选择n_{g2}进行扩展，因为A*总是选择有最小 $\hat{f}$ 值的节点进行扩展，而 $\hat{f}(n^*) \leq f(n_0) < f(n_{g2})$。

定理到此完全证明完毕。我们称任何保证能找到一条到达目标的最佳路径的算法是可接纳的。因此，在定理的三个条件下， A*是可接纳的。延伸开来，任何没有高过 h的函数 $\hat{h}$ 都是可接纳的。从现在起，当谈到关于A*的应用时，都假设定理的三个条件是满足的。

如果A*的两个版本A_1*和A_2*，其差别在于对所有的非目标节点， $\hat{h}_1 < \hat{h}_2$，那么我们就说A_1*比A_2*更灵通(*informed*)。在叙述下面的结果时没有进行证明（参见 [Hart，Nilsson，& Raphael 1968， Hart，Nilsson，& Raphael 1972， Nilsson 1980]）。

定理9.2 如果A_2*比A_1*更灵通，那么对任何有从n_0到目标节点的一条路径的图，在搜索终止时，被A_2*扩展过的每一个节点也被A_1*扩展过。

可以得出A_1*扩展的节点至少和A_2*的一样多，这个更灵通的算法A_2*也就更有效。因此，我们要寻找尽可能接近真实函数 h的函数 $\hat{h}$（为了搜索效率），同时又不能超过它们（为了可接纳性）。当然，在衡量总的搜索效率时，也应当考虑计算 $\hat{h}$ 的代价。最灵通的算法是 $\hat{h} \equiv h$，但是典型地讲，这样的 $\hat{f}$ 只能在很高的代价下，通过完成我们正在尝试的每一个搜索才能得到。

图9-6概括了我们已经讨论过的一些搜索算法的关系。当对所有的节点 $\hat{h} \equiv 0$时，我们得到相同代价算法（搜索顺着相同代价的边沿向外扩展）。当 $\hat{f}(n) = \hat{g}(n)$=深度（n）时，我们得到广度优先搜索法，它顺着相同深度的边沿向外扩展。相同代价和广度优先算法都是A* ($\hat{h} \equiv 0$)的特殊情况，故它们也都是可接纳的。

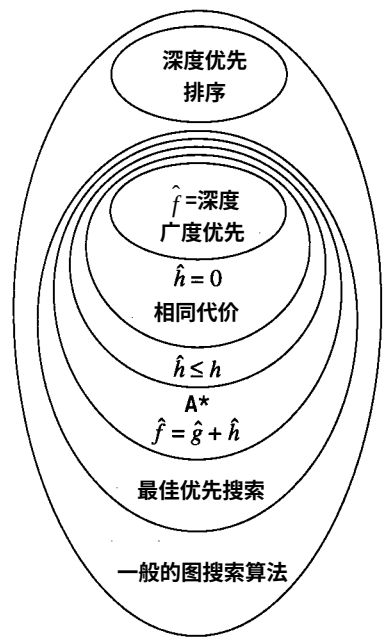


图9-6 搜索算法之间的关系

9.2.3 一致性（或单调）条件

考虑一对节点（n_i、n_j），n_j是n_i的一个后继。如果在搜索图中所有的这种节点对都满足下述条件：

$$\hat{h}(n_i) - \hat{h}(n_j) \leq c(n_i, n_j)$$

其中$c(n_i, n_j)$是从n_i到n_j的代价。

上式也写作：$\hat{h}(n_i) \leq \hat{h}(n_j) + c(n_i, n_j)$

和

$$\hat{h}(n_j) \geq \hat{h}(n_i) - c(n_i, n_j)$$

那么我们就说h服从一致性条件。这个条件陈述了顺着搜索图中的任何路径，到达目标的最优（剩余）代价的估价的减少不会大于该路径弧代价。也就是说，在考虑了一个弧的已知代价后，启发式函数在局部是一致的[⊖]。这种一致性条件可以表示为如图9-7所示的三角不等式。

一致性函数也暗示了当搜索树中结点的值远离开始节点时，它是单调非递减的。设n_i和n_j是由A*在搜索树上产生的两个节点，n_j是n_i的后继。如果满足一致性条件，就有$\hat{f}(n_j) \geq \hat{f}(n_i)$。为了证明这个事实，我们从一致性条件开始：

$$\hat{h}(n_j) \geq \hat{h}(n_i) - c(n_i, n_j)$$

给上式两边都加上$\hat{g}(n_j)$，有：

$$\hat{h}(n_j) + \hat{g}(n_j) \geq \hat{h}(n_i) + \hat{g}(n_j) - c(n_i, n_j)$$

但是$\hat{g}(n_j) = \hat{g}(n_i) + c(n_i, n_j)$(我们可能会担心$\hat{g}(n_j)$会比这个值小，因为我们可能会顺着其他的比通过$n_i$点代价更低的路径到达$n_j$。但这样一来，在搜索树上$n_j$就不是$n_i$的后继了)。

因此，$\hat{f}(n_j) \geq \hat{f}(n_i)$;

因为这个原因，一致性条件（对$\hat{h}$）也常被称为单调条件（对$\hat{f}$）。下面是一个涉及到一致性条件的重要定理（[Hart，Nilsson & Raphael 1968]）。

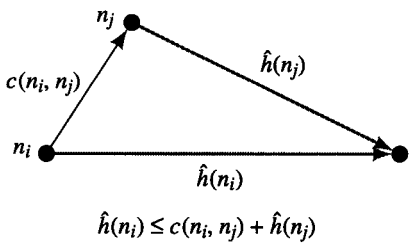


图9-7 一致性条件

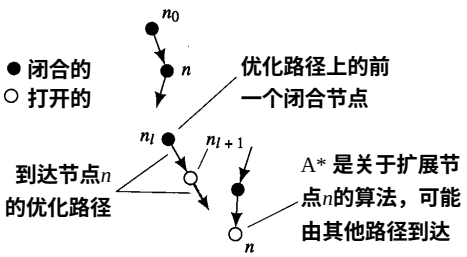


图9-8 用于证明定理9.3的图

定理9.3 如果$\hat{h}$上的一致性条件被满足，那么当A*扩展节点n时，它已经找到了到达节点n的一条最优路径。

证明：假设A*在隐式图G中正在搜索从开始节点n_0到目标节点的一条最佳路径，它准备扩展一个打开的节点n。设$\xi = (n_0, n_1, \ldots, n_l, n_{l+1}, \ldots, n = n_k)$是图$G$中从$n_0$到$n$的一条最佳路径的节点序列，$n_1$是$\xi$上被A*扩展的最后一个节点（参见图9-8）。因为$n_1$是$\xi$上最后一个没打开的

[⊖] Pearl [Pearl 1984, pp.82-83]定义了一个启发式函数的全局一致性概念，并证明它等同于我们关于局部一致性的定义。

节点，因此n_{l+1}在*OPEN*上是一个扩展候选者。

对任何节点n_i和它的后继节点n_{i+1}，在ξ中（到达n的一条最优路径），我们有

$$g(n_{i+1}) + \hat{h}(n_{i+1}) = g(n_i) + c(n_i, n_{i+1}) + \hat{h}(n_{i+1}) \geq g(n_i) + \hat{h}(n_i);$$

当一致性条件满足时，由≥关系的传递性可以得到

$$g(n_j) + \hat{h}(n_j) \geq g(n_i) + \hat{h}(n_i) \quad \text{（对ξ上的任何} n_i \text{和} n_j\ (i<j)\text{）}$$

在特殊情况下，

$$g(n) + \hat{h}(n) \geq g(n_{l+1}) + \hat{h}(n_{l+1}) = \hat{f}(n_{l+1})$$

因为A*已经发现了到达n_{l+1}的一条最佳路径，使 $\hat{g}(n_{l+1}) = g(n_{l+1})$

但是由于A*将要扩展节点n而不是n_{l+1}，故必然有

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n) \leq \hat{f}(n_{l+1})$$

但是我们已经有

$$\hat{f}(n_{l+1}) \leq g(n) + \hat{h}(n);$$

因此

$$\hat{g}(n) \leq g(n);$$

但是由于我们计算 $\hat{g}$ 的方法暗示了 $\hat{g}(n) \geq g(n)$，故必然有 $\hat{g}(n) = g(n)$;

这表示了或者$n_{l+1} = n$，或者我们必然已经找到了到达节点n的一些其他最佳路径。证明完毕。

一致性条件是很重要的，因为当它被满足时， A*不再需要重定向指针（第7步），搜索一个图与搜索一个树就没有什么差别了。

满足一致性条件时，可以为A*的可接纳性给出一个简单直观的论证。它是这样的：

1) $\hat{f}$ 的单调性暗示了搜索顺着 $\hat{f}$ 值增大的边缘向外扩展。

2) 因此，被选择的第一个目标节点就是有最小 $\hat{f}$ 值的一个目标节点。

3) 对任何目标节点n_g， $\hat{f}(n_g) = \hat{g}(n_g)$（这里，我们用了这样的事实，如果 $\hat{h}$ 函数是一致的，它也将绝不会比真正的h函数大[Pearl 1984，p. 83]）。

4) 因此，第一个被选择的目标节点将是有最小 $\hat{g}$ 值的目标节点。

5) 作为定理9.3的一个结论，无论何时（特别地）当一个目标节点n_g被选择扩展时，我们已找到了到达那个目标节点的一个最佳路径。即 $\hat{g}(n_g) = g(n_g)$。

6) 这样，第一个被选择的目标节点将是算法发现的最佳路径的目标节点。

很多启发式函数满足一致性条件。例如， 8数码问题中的“不在正确位置的数码个数”函数就是一个例子。当一个启发式函数不满足一致性条件，但其他方面是可以接受的条件时，那么（用Mérõ[Merõ 1984]提出的思想）我们能（在搜索期间）调整该函数使它满足一致性条件。假如，在A*的每一步，我们检查刚刚扩展的节点n的后继的 $\hat{h}$ 值。任何 $\hat{h}$ 值小于 $\hat{h}(n)$值的节点减去从n到这个节点的弧代价就会得到它们调整后（在搜索过程中）的 $\hat{h}$ 值，这样它们就刚好等于 $\hat{h}(n)$值减去那段弧代价的所得值（但是*CLOSED*中的某个节点必须移回到*OPEN*中。这种情况可能会发生，该节点的 $\hat{h}$ 值是用这种方式调整过的）。读者可以去验证一下这种调整方法，它将使算法是可以接纳的。

9.2.4 迭代加深的A*

在第8章，讲过广度优先搜索的存储需求会随着搜索空间中目标深度的增加呈指数递增。

尽管好的启发式搜索减少了分枝因子，但启发式搜索还是有如上一样的缺点。在第 8章介绍的迭代加深搜索，不但允许我们找到最小代价路径，而且存储需求随着深度增加呈线性增长。由[Korf 1985]提出的迭代加深A*(IDA*)方法能获得同启发式搜索相似的好处。通过使用 IDA*的并行实现能获得更高的效率（见[Powley，Ferguson，& Korf 1993]）。

该方法同普通迭代加深方法工作相似，我们执行一系列深度优先搜索。在第一次搜索中，建立一个“截止代价”，它等于 $\hat{f}(n_0) = \hat{h}(n_0) + \hat{g}(n_0) = \hat{h}(n_0)$，$n_0$是开始节点。我们所知道的就是到达目标的最佳路径代价可能等于这个截断值（如果 $h(n_0) = \hat{h}(n_0)$，上述语句就是肯定的；因为$h(n_0) \geq \hat{h}(n_0)$，它不可能较小）。我们在深度优先方式下扩展节点，无论何时，只要扩展节点的一个后继的 $\hat{f}$ 值超过截断值，就进行回溯。如果该深度优先搜索终止在一个目标节点，显然它已经找到了到达目标的一个最小代价路径。否则，一个最优路径的代价一定比这个截断值要大。因此，我们增大截断值，开始另一次深度优先搜索。一个最优路径的下一个可能值会是什么？它可能和上一次深度优先搜索中遍历过（但没有扩展）的节点的最小 $\hat{f}$ 值一样小。有这个最小 $\hat{f}$ 值的节点可能是在一条最优路径上（因为我们知道没有任何最佳路径的代价等于前一次的截断值）。这个最小 $\hat{f}$ 值可以作为下次深度优先搜索的新的截断值，等等，如此重复进行。显然，可以容易地看到， IDA*保证会找到一个到达目标的最小代价路径（ [Korf 1985]提出了这个结果的一个证据， [Korf 1993]再次提供证明。后面还讨论了当单调条件不满足时迭代加深方法的一些限制，并且提出了一个称为递归最优搜索的变更算法）。

IDA* 确实重复节点扩展，但与普通迭代加深一样，存储需求和深度优先搜索实现效率之间存有折衷。当搜索空间的每个节点的 $\hat{f}$ 值都不同，迭代次数可能等于其 $\hat{f}$ 值比最优路径代价小的节点的数目，情况又怎样呢？（有些方法放弃可接纳性来处理这个问题）。

9.2.5 递归最优搜索

由Korf提出的递归最优搜索，RBFS([Korf 1993]方法比IDA*用到的存储稍微多一点，但是比IDA*产生的节点少。当扩展一个节点n时，RBFS计算n的后继的 $\hat{f}$ 值，并且重新计算搜索树上n和n的祖先的 $\hat{f}$ 的值。这个重新计算的过程叫做 $\hat{f}$ 值的回溯(*backing up*)。回溯是这样进行的：刚刚扩展的节点的后继的回溯值就是该后继的 $\hat{f}$ 值。搜索树上带有后继m_i的节点m的回溯值是

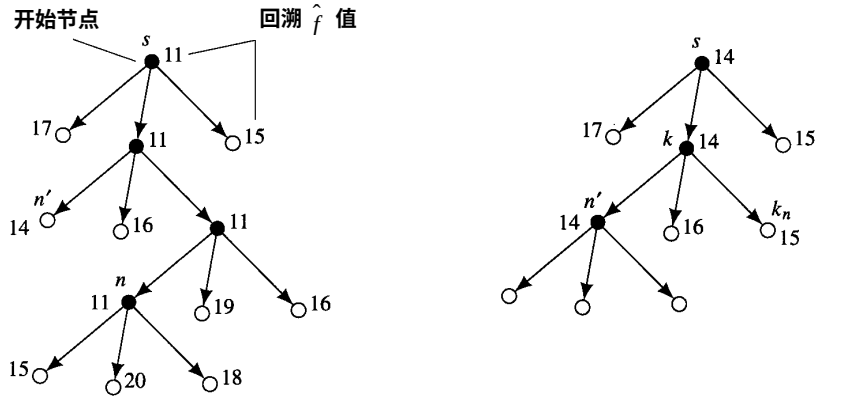
$$\hat{f}(m) = \min_{m_i} \hat{f}(m_i)$$

其中 $\hat{f}(m_i)$是节点m_i的回溯值。

如果节点n(它刚被扩展)的后继之一在所有的*OPEN*节点中有最小 $\hat{f}$ 值，它被依次扩展一直进行下去。但是如果 *OPEN* 中的其他节点n'—— 它不是n的后继，有最小的 $\hat{f}$ 值。在这种情况下，算法回朔到节点 n'和n的最低共同祖先节点 k。让节点k_n是到节点n的路径上节点k的后继。RBFS移去以k_n为根的子树，于是k_n变成了其 $\hat{f}$ 值等于它的回溯值的一个*OPEN* 节点，搜索继续在有最低 $\hat{f}$ 值的*OPEN*节点n'下进行。

图9-9表达了该算法的主要思想。搜索由一条节点路径和该路径上所有节点的兄弟构成。因此，存储要求仍是到目前为止所探索的最优路径长度的线性函数。顶部节点都在 *OPEN*上，在搜索树上的所有节点都和回溯的 $\hat{f}$ 值有关。

若需了解其他有关空间效率的搜索算法，可以参考 [korf 1996]。



a) RBFS 仅扩展节点n，并未回溯出它的后继的 $\hat{f}$ 值

b) 已回溯出 $\hat{f}$ 值，节点k_n的子树已丢弃，搜索从节点n'以下的节点继续

图9-9 递归最优搜索

9.3 启发式函数和搜索效率

在决定A*效率时，启发式函数的选择是至关重要的。用 $\hat{h} \equiv 0$保证了可接纳性，但生成了相同代价搜索，因而效率不高。 $\hat{h}$ 等于h上较低约束的最高可能值，不但可以扩展较少的节点，还能维持可接纳性。如8数码问题中，函数 $\hat{h}(n) = W(n)$ (其中$W(n)$是不在正确位置的数码个数)就是$h(n)$上的一个较低约束，但是它没有提供对数码状态难度（按照到达目标的步数）的一个很好的估计。一个更好的估计是函数 $\hat{h}(n) = P(n)$，$P(n)$是每个数码离开“初始点”（忽略插入数码）的距离总和。

在AI历史的早期，[Newell，Shaw & Simon 1959，Newell 1964]建议用一个简化的模型来构成指导搜索的“计划”。同样的思想被应用于找到一个好的启发式函数的问题中，[Gaschnig 1979]和[Pearl 1984，第4章]描述过这种思想。例如，我们可以通过允许较少约束的数码移动来放宽8数码的约束条件。如果我们允许一个数码直接移到 (一次)它的目标区中，那么到达目标的步数就是在错误位置的数码的个数之和 $W(n)$。一个约束稍强(因此更好)的模型允许数码移动到相邻的位置，即使已经有一个数码在那里。在这种模型中，到达目标的步数就是每个数码离它的目标位置之和 $P(n)$。这些模型表示了一个 agent如何自动计算我们直觉猜想的 $\hat{f}$ 函数。Pearl指出，我们可能已经计算了一个函数 $\hat{h}$ ，它比$W(n)$稍微好一些，在$W(n)$中，任何数码在一步移动中可以和空格交换。在另一个约束稍强的模型中，数码只能顺着相邻单元 (顺着该路径计算每一个位置)的一条路径移到空格的位置。

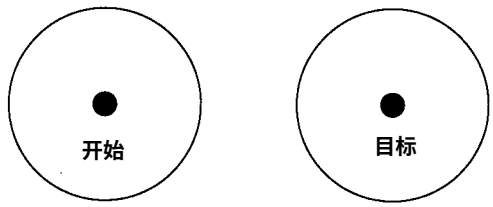
在road-map问题中，能够用不严格约束的模型来解释作为 $\hat{h}$ 函数使用的欧几里德(直线)距离。在一个非严格约束的模型中，不必穿过每一条现存的路，一个旅行者能“电话传输”，用直线距离直接到达任何城市。由于在非严格约束的模型中的方案决不会超过原始问题的代价，故所选择的 $\hat{h}$ 函数总是可接纳的！结果是它们也是一致的。因此使用它们时，算法不必再次遍历以前扩展过的节点。

在选择 $\hat{h}$ 函数时，我们必须考虑计算 $\hat{h}$ 本身的计算量。模型的非严格约束越少，启发式函数就越好——当然计算会越困难。如果一个人用的启发式函数等于 h，那么可以达到扩展节点的绝对最小值，但是这个 $\hat{h}$ 的计算将要求解决最初的问题。我们常常是在精确 $\hat{h}$ 函数和其计算代价之间取折衷值。

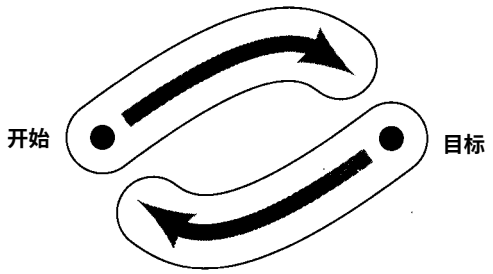
经常，通过使用一些非低约束的函数 $\hat{h}$ 以可接纳性为代价来换取搜索效率。也就是说，一个可能不是最佳的路径比最佳路径更容易找到，一个非较低约束的 $\hat{h}$ 函数可能比一个较低约束的函数容易计算。在这些情况下，效率可能会成倍地增加——因为被扩展的节点会被减少（虽然以可接纳性为代价）并且计算量也被减少。

另一种可能性是修改评估函数中 $\hat{g}$ 和 $\hat{h}$ 的权值，即用 $\hat{f}=\hat{g}+w\hat{h}$，w是一个正数。非常大的w值会过分强调启发式部分，而非常小的 w会突出搜索的广度优先特性。实验结果表明如果w值与搜索树上节点深度成反比，常会提高搜索效率。在浅深度时，搜索主要依赖启发式部分，然而随着深度的增加，为了确保最终会发现一些到达目标的路径，搜索会逐渐以广度优先为主。

可能有人认为通过从开始点和目标点两边同时搜索，可以改善搜索效率。典型地讲，这种改善只能通过广度优先搜索获得。当广度优先搜索用于双向搜索时，我们可以保证搜索的前沿会在开始点和目标点之间相遇（见图9-10a），且由于已经建立终止条件，双向广度优先搜索可以保证找到一条最佳路径[Pohl 1971]。但是如果启发式搜索由双向开始，如果启发式倾向于在不适当的方向进行搜索，那么两个搜索前沿可能不会相遇，从而不能找到一条最佳路径（见图9-10b）。



a) 广度优先搜索



b) 启发式搜索

图9-10 双向搜索

搜索效率的一个度量是有效分枝因子 B，它描述了一个搜索过程朝着目标前进的激烈程度。假设搜索找到了一个长为 d的路径，生成了N个节点，那么B就是有下列属性的树上的每个节点的后继个数：

- 树中每个非树叶节点都有B个后继。
- 树中的树叶节点的深度均为d。
- 树中的节点总数是N。

因此，B和路径长度d以及生成的总节点数N之间有下列关系：

$$B + B^2 + \cdots + B^d = N$$

$$\frac{(B^d - 1)B}{(B - 1)} = N$$

虽然B不能明确地表达为d和N的函数，但是B相对于N在不同d值下的图表被显示在图9-11中。一个朝着目标高聚集的搜索，其B值在其他方向是非常小的；另一方面，一个“灌木丛式”的搜索图将会有高的B值。

在有效搜索因子适当地独立于路径长度的范围内，可以用它来评价在各种长度的搜索中产生的节点数。例如，用评估函数 $\hat{f}(n) = \hat{g}(n) + W(n)$，对于图9-2中的8数码问题，我们可以用图9-11计算它产生的节点个数，B值等于1.2。假如我们想用同样的评估函数来解决一个更难的8数码问题，比如说要求30 步，来估计会产生多少个节点。从图 9-11中，假定有效分枝因子保持常量，我们看到，30步问题产生大约2000个节点。

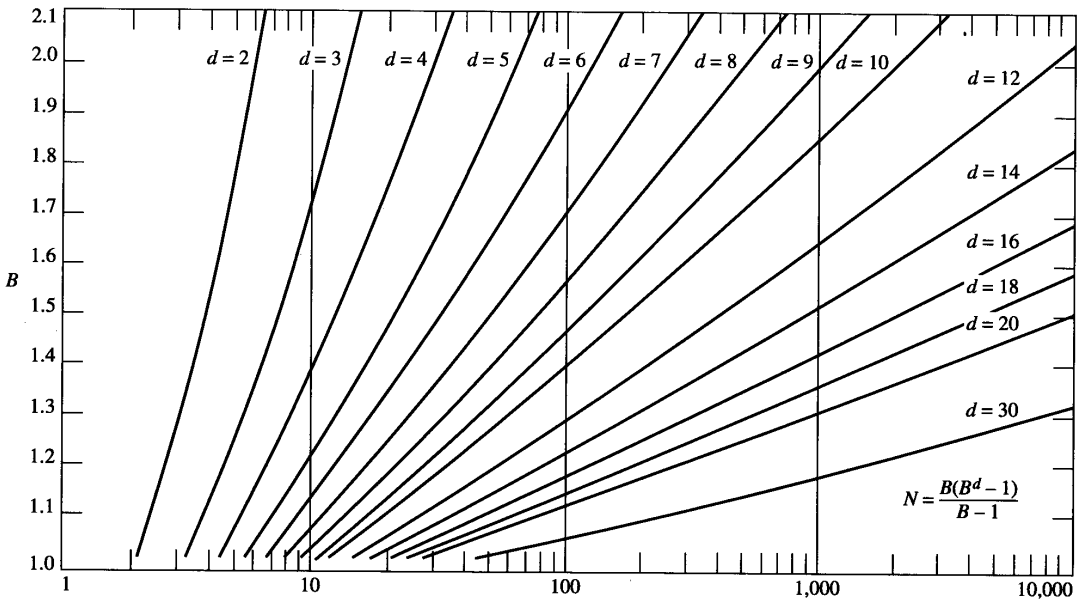


图9-11 不同d值下，NB随N的变化图

归纳一下，有三个重要的因素影响算法A*的效率：

- 被发现路径的代价（长度）；
- 在发现路径中被扩展的节点数；
- 计算 $\hat{h}$ 的计算量。

选择适当的启发式函数可以让我们平衡这些因素以最大化搜索效率。

到目前为止讨论的所有搜索方法，包括启发式搜索，其时间复杂度都是 $O(n)$，n是产生的节点数（假定启发式函数能在常量时间内被计算出）。特殊地，当有效分枝因子是B，弧代价都相等，并且从开始节点到目标的深度为d的情况下，广度优先搜索的复杂度为 $O(B^d)$，对相同代价搜索（$\hat{h} \equiv 0$）和不相等的弧代价，复杂度是$O(B^{C/c})$，其中C是最优方案的代价，c是最小代价弧的代价［korf 1992］。对很多实际的问题，由于d(或C/c)太 大，从而导致不能计算最优方案的搜索（甚至启发式搜索）。例如，我们用搜索算法计算一个目标是 15步远的最佳下一步动作（比如有4个选择），搜索算法可能必须扩展 $4^{15} \approx 10^9$个节点。这么大的计算量对一个要在数

秒内做出决策的agent可能是不实际的。由于所有被扩展的节点必须存在一个树结构中， A*的空间复杂度和时间复杂度的级数相同。

由于agent对它们的计算资源有时间和空间约束，因此不可能在很多任务中找到最佳方案。相反，做出非最优但是可以接受的方案(称为可以满足的方案)或者局部方案倒是必需的。下一章将提出各种能在agent的有限资源下可以使用的方法。

9.4 补充读物和讨论

启发式搜索涉及两种计算。第一是真正的扩展节点和产生路径的目标级计算。第二是决定下一个要扩展的节点的元级计算。目标级是环境中的物理动作；元级是图中的计算动作。在人工智能中，常常会提到目标级/元级的差别。[Russell & Wefald 1991]完整地讨论了它们，在下一章要讨论的简化型搜索方法中，它们扮演了一个特别重要的角色。

8数码问题被很多AI研究者作为启发式搜索方法的试验台。由[Doran & Michie 1966]写的一篇论文使用了8数码问题，开始了在图中寻找路径过程中使用评估函数的历史。

1990年，Korf阐述了“IDA*能解决15数码问题，但更大的数码问题[如24数码]在当前的机器上是很难处理的”[korf 1990，p.191]。然而，随着功能更强的机器(一台Sun Ultra Sparc工作站每秒钟能产生一百万个节点)和更好的(自动发现的)启发式算法的出现，[Korf & taylor 1996]能在2.25小时到1个月的时间范围内找到随机产生的可解决的24数码问题的最优方案。

虽然这些数字问题对改进和测试搜索技术是有用的，但A*和相关的启发式搜索过程也已被成功地应用到了很多实际问题中。

关于更多的使用非约束模型发现启发式函数的内容，参见[Mostow & Prieditis 1989，Prieditis 1993]。[Pohl 1973]用不同的权值对 $\hat{f}$ 函数的启发式部分进行了实验。

关于启发式搜索的经典书籍是[Pearl 1984]。[Kanal & Kumar 1988]是关于搜索算法的论文集。后一本书的第一篇文章提出了一个由AI和运筹学人员分别独立开发的一致搜索方法。

习题

9.1 在“四皇后问题”中，我们把四个皇后放在一个4×4的棋盘上以便四个棋子不能彼此抓到对方(即，只有一个皇后能在棋盘的任何行列或对角线上)。假如我们想用下面的问题空间解决这个问题：开始节点由一个空的4×4数组表示；后继函数产生新的4×4数组，该数组包含一个皇后的附加合法放置，它可在数组的任何位置；预计目标是只要四个皇后在数组中的条件(合法位置)都被满足。

1) 根据皇后到达目标的剩余步数，为这个问题设计一个可以接纳的函数(注意所有的目标节点离开始节点刚好四步!)。

2) 使用你的启发式函数在A*中搜索一个目标节点。画出包含搜索产生的所有4×4数组的搜索树，用 g 和 $\hat{h}$ 值标出每一个数组(注意：考虑到对称性，我们只要产生开始节点的三个后继就可以了)。

9.2 这个习题假定你已经完成了习题7.4中的汉诺塔问题。参考你对那个问题的答案，如果你还没有做它，那么现在解答。提出该问题的一个可接纳的函数 $\hat{h}$，它要比 $\hat{h}\equiv 0$ 更好。

9.3 算法A*直到一个目标节点被选择扩展才会终止。然而，到达目标节点的一条路径可能

在那个节点被选择扩展前早就找到了。一旦目标节点被发现，为什么不终止搜索呢？用一个例子说明你的答案。

9.4 启发式函数中的单调条件要求对所有的节点—后继对 (n_i, n_j)，满足 $\hat{h}(n_i) \leq \hat{h}(n_j)+c(n_i, n_j)$，$c(n_i, n_j)$是从$n_i$到$n_j$的弧代价。如果单调条件不能满足，一种方法建议在搜索过程中可以调整 $\hat{h}$ 以使条件被满足。该思想是指无论何时，当后继为 n_j的节点n_i被扩展时，我们可以给 $\hat{h}(n_j)$增加一个值使单调条件满足。构造一个例子说明即使使用这种方法，当一个节点被扩展时，我们不必找到它的一个最小代价路径。

9.5 说明在算法A*中，如果我们从*OPEN*中移去任何节点n，$\hat{f}(n) > F$，F是$f(n_0)$的一个上限约束，n_0是开始节点，这时的A*仍是可接纳的。

9.6 在这个习题中，我们考虑在两个可接纳的启发式函数中做出选择，其中一个总是“至少像另一个一样精确”。更准确地讲，对一个固定的目标，让 $\hat{h}_1$和 $\hat{h}_2$是两个可接纳的启发式函数，在搜索树中的每一个节点n有 $\hat{h}_1(n) \leq \hat{h}_2(n)$。

回想一下，A*按照 $\hat{f}$ 值扩展节点，但是相同 $\hat{f}$ 值的节点被扩展的次序会根据优先队列的实现发生变化。对一个给定的启发式函数 $\hat{h}$，如果扩展节点的顺序和A*使用 $\hat{h}$ 搜索的顺序一致，我们就说搜索树中的节点顺序是 $A_{\hat{h}}$ —合法。

让N为使用 $\hat{h}$ 的A*搜索扩展的节点集。证明总是有一些搜索顺序是 $A_{\hat{h}_2}$ —合法。它仅仅(不必严格)扩展了N中节点的一个子集。即证明 $\hat{h}_2$比 $\hat{h}_1$产生了一个更小的搜索空间，这种情况总是可能的(注意这个问题叫你找到一些有特殊属性的搜索顺序。在一般情况下，在任意的搜索顺序下拥有这些属性是不可能的，你知道为什么吗？)。