CS 188 人工智能导论
Spring 2024 Note 3

Author (all other notes): Nikhil Sharma

作者（贝叶斯网络笔记）：Josh Hug 和 Jacky Liang，由 Regina Wang 编辑

作者（逻辑笔记）：Henry Zhu，由 Peyrin Kao 编辑

学分（机器学习和逻辑笔记）：部分章节改编自教材人工智能：现代方法。

最后更新：2023年8月26日

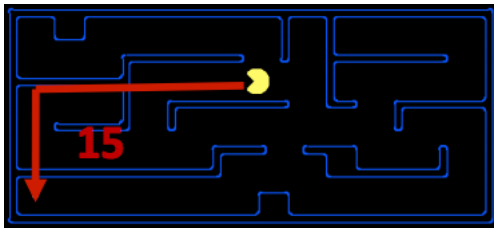
Informed Search

均匀成本搜索因为它既完整又最优而好处多多，但它可能相当慢，因为它在寻找目标时从起始状态向每个方向扩展。如果我们对应该集中搜索的方向有一些概念，就可以显著提高性能，更快地**锁定**一个目标。这正是**启发式搜索**的重点。

Heuristics

启发式算法是允许估计到目标状态距离的驱动力——它们是接受一个状态作为输入并输出相应估计值的函数。这种函数所执行的计算是特定于正在解决的搜索问题的。出于以下A*搜索中将会看到的原因，我们通常希望启发式函数是到目标剩余距离的下限，因此启发式算法通常是**放松问题**的解决方案（其中已去除了一些原始问题的约束）。转到我们的吃豆人示例，让我们考虑之前描述的路径问题。解决这个问题的一种常用启发式算法是**曼哈顿距离**，对于两点(x_1, y_1)和(x_2, y_2)定义如下：

$$Manhattan(x_1, y_1, x_2, y_2) = |x_1 - x_2| + |y_1 - y_2|$$



上面的可视化展示了曼哈顿距离帮助解决的放松问题——假设吃豆人想要到达迷宫的左下角，它计算从吃豆人当前位置到目标位置的距离假设迷宫中没有墙。这个距离是放松搜索问题中的确切目标距离，相应地也是实际搜索问题中的估计目标距离。有了启发式算法，我们的智能体就非常容易实现逻辑，使其在决定执行哪个动作时“倾向于”扩展那些被估计为更接近目标状态的状态。

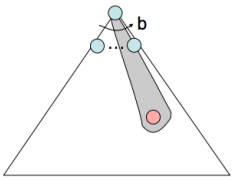
这种偏好概念非常强大，以下两种实现启发式函数的搜索算法：贪婪搜索和A*，都利用了这一点。

Greedy Search

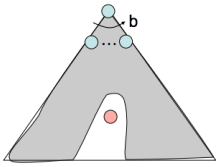
· 描述 - 贪婪搜索是一种总是选择具有最低启发值 的前沿节点进行扩展的探索策略，这对应于它认为最接近目标的状态。

· 前沿节点表示 - 贪婪搜索与UCS一样，使用优先级队列表示前沿节点。不同的是，贪婪搜索使用 估计的前进成本（启发值的形式）来分配优先级，而不是使用 计算的后退成本（到状态路径中的边权重之和）。

· 完整性和最优性 - 即使目标状态存在，贪婪搜索也不能保证找到它，而且在选择一个非常糟糕的启发函数时，它也不是最优的。它通常在不同场景下表现得非常不可预测，可能直接到达目标状态，也可能像一个被错误引导的DFS一样，遍历所有错误的区域。



(a) 好时候的贪婪搜索 :)



(b) 贪婪搜索在糟糕的一天 :(

A* Search

· 描述 - A* 搜索是一种总是选择具有最低估计总成本 的前沿节点进行扩展的探索策略，其中总成本是从起始节点到目标节点的整个成本。

· 前沿节点表示 - 与贪婪搜索和UCS一样，A*搜索也使用优先级队列表示前沿节点。唯一的不同是优先级选择的方法。A* 结合了UCS使用的总后退成本（到状态路径中的边权重之和）和贪婪搜索使用的估计前进成本（启发值），通过将这两个值相加，有效地得出从起点到目标的 估计总成本。鉴于我们希望最小化从起点到目标的总成本，这是一个很好的选择。

· 完整性和最优性 - A* 搜索在给定一个适当的启发式时既是完整的也是最优的（我们稍后会讨论）。它结合了迄今为止我们介绍的所有其他搜索策略的优点，结合了贪婪搜索的普遍高速度和UCS的最优性和完整性！

Admissibility and Consistency

既然我们已经讨论了启发式及其在贪婪搜索和 A* 搜索中的应用，让我们花些时间讨论什么构成了一个好的启发式。为此，我们首先以以下定义重新制定UCS、贪婪搜索和 A* 中用于确定优先级队列排序的方法：

· g(n) - 表示UCS计算的总代价的函数。

· h(n) - 表示贪婪搜索使用的 启发值 函数，即估计的前进成本。

· f(n) - 表示 A* 搜索使用的估计总成本的函数。 f(n) = g(n) + h(n).

在探讨什么构成一个“好”的启发式之前，我们必须先回答一个问题，即不论我们使用什么启发函数，A* 是否保持其完整性和最优性。这其实很容易找到破坏这两个特性的启发式函数。例如，考虑启发函数 h(n) = 1 − g(n)。无论搜索问题是什么，使用这个启发式会产生

$$\begin{aligned} f(n) &= g(n)+h(n) \\ &= g(n)+(1-g(n)) \\ &= 1 \end{aligned}$$

因此，这样的启发式将A*搜索简化为BFS，其中所有边的代价都是等同的。正如我们已经展示的那样，BFS在一般边权不恒定的情况下不保证是最优的。

使用A*树搜索的最优性所需的条件称为**可接受性**。可接受性约束指出，可接受启发式估计的值既不负面也不过高估计。定义$h^*(n)$为从给定节点n到达目标状态的真实最优前向代价，我们可以将可接受性约束数学表述如下：

$$\forall n,\ 0 \le h(n) \le h^*(n)$$

定理：对于给定的搜索问题，如果一个启发式函数h满足可接受性约束，使用A^*树搜索带有h的搜索问题将获得最优解。

证明：假设在给定搜索问题的搜索树中有两个可到达的目标状态，分别是最优目标A和次优目标B。一些n的祖先（可能包括A本身）必须当前在边界上，因为A是从起始状态可达的。我们声称n将优先于B进行扩展，使用以下三条陈述：

1.g(A) < g(B)。因为A被认为是最优的，而B是次优的，我们可以得出结论A具有比B更低的回到起始状态的代价。

2.h(A) = h(B) = 0，因为我们假设我们的启发式满足可接受性约束。由于A和B均为目标状态，从A或B到达目标状态的真实最优代价为$h^*(n) = 0$;因此$0 \leqslant h(n) \leqslant 0$.

3.$f(n) \leqslant f(A)$，因为通过可接受性$h, f(n) = g(n) + h(n) \leqslant g(n) + h^*(n) = g(A) = f(A)$。通过节点n的总代价最多是A的真实后向代价，这也是A的总代价。

我们可以结合陈述1和2得出f(A) < f(B)，如下：

$$f(A) = g(A) + h(A) = g(A) < g(B) = g(B) + h(B) = f(B)$$

结合上述不等式和陈述3的一个简单结果如下：

$$f(n) \leq f(A) \wedge f(A) < f(B) \Longrightarrow f(n) < f(B)$$

因此，我们可以得出结论 n 在 B之前扩展。因为我们已经证明了任意 n都是如此，我们可以得出结论 A的所有祖先（包括 A本身）在 B之前扩展。 2

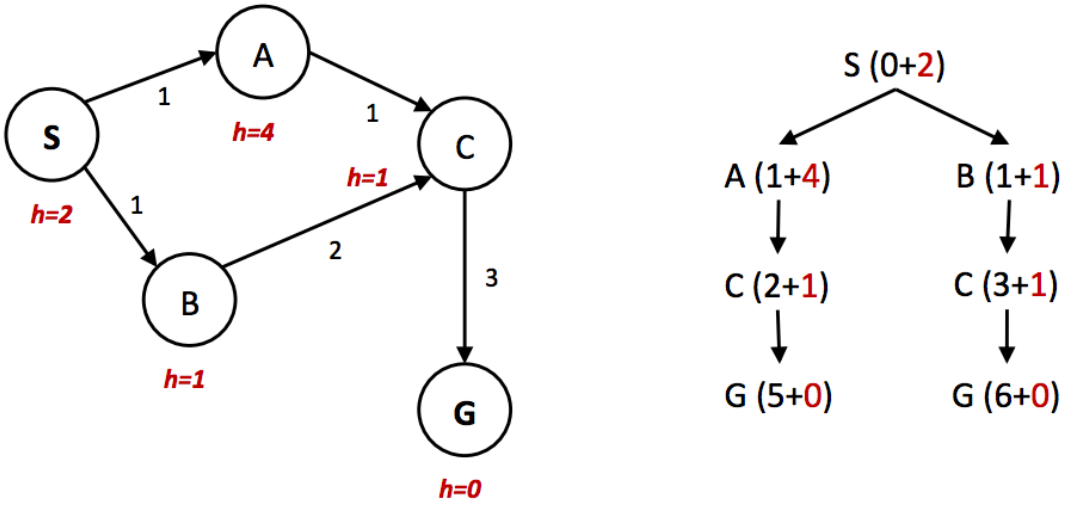
我们发现树搜索的一个问题是在某些情况下它可能无法找到解决方案，陷入在状态空间图中无限循环搜索相同的循环。即使在我们的搜索技术不涉及这种无限循环的情况下，由于有多种方法可以达到相同的节点，我们经常会多次访问相同的节点。这导致工作量呈指数增长，自然的解决方案是跟踪已经扩展的状态，并且永不再扩展它们。更明确地说，在使用您选择的搜索方法时，维护一个“已达到”集合的扩展节点集。然后，确保在扩展之前每个节点不在集合中，并在扩展后添加到集合中。如果没有，带有此优化的树搜索称为图搜索[1]，**其伪代码如下所示：**

[1]在其他课程中，比如CS70和CS170，你可能已经接触过图论中的“树”和“图”。具体来说，树是一种满足某些约束（连通且无环）的图。这并不是本课程中我们区分树搜索和图搜索的方式。

```
function GRAPH-SEARCH(problem, frontier) return 解决方案或失败
    reached ←一个空集合
            ←  INSERT(MAKE − NODE(INITIAL − STATE[problem]),  frontier)  while  not  IS-
EMPTY(frontier) do
      end
      node ← POP(frontier) 如果 problem.IS-GOAL(node.STATE) then
      end
      return node
    if node状态不在 reached then
      end
      添加 节点.状态于 已访问集合
      对于 每个 子节点 in 展开(问题, 节点)执行
      end
      frontier ← INSERT(child-node, frontier)


return failure
```

请注意，在实现中，至关重要的是将已达到的集合存储为不相交的集合而不是列表。将其存储为列表需要O(n)操作来检查会员资格，这消除了图搜索旨在提供的性能改进。图搜索的另一个警告是，即使在可采纳的启发式下，它也倾向于破坏A*的最优性。考虑下面的简单状态空间图和相应的搜索树，并注明权重和启发值：



在上述示例中，很明显，最佳路径是遵循S → A → C → G,，总路径成本为f1 + 1 + 3 = 5。到目标的唯一其他路径，S → B → C → G路径成本为1 + 2 + 3 = 6。然而，由于节点 A的启发值比节点 B的启发值大得多，节点 C首先沿着第二条次优路径作为节点的子节点 B进行扩展。然后将其放入“已达到”集，因此A*图搜索在访问它作为节点的子节点时未能重新扩展 A，因此从未找到最佳解决方案。因此，为了在A*图搜索下保持最优性，我们需要比可采纳性更强的属性，

一致性。一致性的核心思想是，我们不仅要让启发式低估任何给定节点到目标的总距离，还要低估图中每条边的成本/权重。用启发式函数来度量一条边的成本，简单来说就是两个连接节点的启发式值之差。数学上，一致性约束可以表达如下：

$$\forall A, C \quad h(A) - h(C) \leq cost(A, C)$$

定理。对于一个给定的搜索问题，如果一致性约束得到满足，使用启发式函数 h的 A^* 图搜索将在该搜索问题上得到最优解。

证明。为了证明上述定理，我们首先证明，当使用一致的启发式进行 A^* 图搜索时，每当我们移除一个节点进行扩展时，已经找到该节点的最优路径。

利用一致性约束，我们可以证明沿任何路径的节点的 f(n)值是非递减的。定义两个节点，n和n'，其中n'是n的子节点。那么：

$$\begin{aligned} f(n') &= g(n') + h(n') \\ &= g(n) + cost(n, n') + h(n') \\ &\geq g(n) + h(n) \\ &= f(n) \end{aligned}$$

如果沿路径的每对父子节点(n, n')，都有$f(n') \geqslant f(n)$，那么沿该路径的f(n)值必须是非递减的。我们可以检查上述图在f(A)和f(C)之间违反了这个规则。根据这些信息，我们现在可以证明，每当一个节点n被移除进行扩展时，其最优路径已经被找到。假设这是错误的——即当n从前沿被移除时，找到的路径为非最优路径。这意味着必须存在某个祖先节点n，该节点n''在前沿但从未扩展过，但在通往 n的最优路径上。矛盾！我们已经证明沿路径的f值是非递减的，因此n''会在n之前被移除进行扩展。

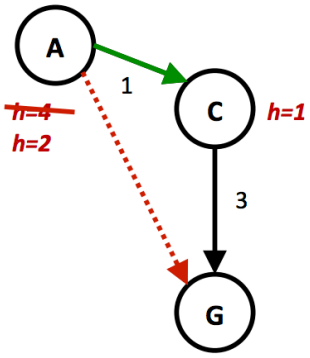
为了完成我们的证明，我们剩下的就是要展示，任何最优目标A总会在任何次优目标B之前被移除进行扩展并返回。这是显而易见的，因为h(A) = h(B) = 0，

$$f(A) = g(A) < g(B) = f(B)$$

正如我们在可容性约束下的 A^*树搜索最优性证明中所做的。因此，我们可以得出 A^*图搜索在一致启发下是最优的结论。2

在我们继续之前，先总结一下上面的讨论，对于有效的启发式要么是可容的，要么是一致的，这意味着对于任何目标状态G，必须h(G) = 0。此外，一致性不仅是比可容性更强的约束，一致性暗示可容性。这仅仅源于这样一个事实：如果没有边缘成本被高估（由一致性保证），那么从任何节点到目标的总预估成本也不会是高估的。

考虑以下三节点网络的一个可容但不一致的启发式示例：



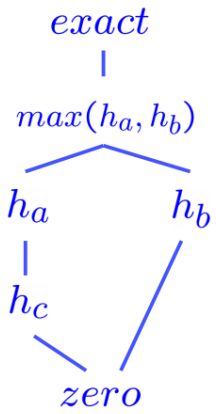
红色虚线表示总的估计目标距离。如果h(A) = 4,则启发式是可接受的，因为从A到目标的距离是4 ⩾ h(A)，同样适用于h(C) = 1 ⩽ 3。然而，从A到C的启发式代价是h(A) − h(C) = 4 − 1 = 3。我们的启发式估计A和C之间边的代价为3，而真实值是cost(A, C) = 1，较小。由于h(A) − h(C) ⩽ cost(A, C)，该启发式不一致。相同计算运行于h(A) = 2，然而，得出h(A) − h(C) = 2 − 1 = 1 ⩽ cost(A, C)。因此，使用h(A) = 2使我们的启发式一致。

dominanc

现在我们已经确定了可接受性和一致性的特性及其在维持A*搜索最优性中的作用，我们可以回到创建“好”启发式的原始问题，以及如何判断一个启发式是否优于另一个。标准度量是**支配**。如果启发式a支配启发式b，那么状态空间图中每个节点的a估计目标距离均大于b的估计目标距离。从数学上讲，

$$\forall n : h_a(n) \geq h_b(n)$$

支配非常直观地捕捉了一个启发式优于另一个的想法——如果一个可接受/一致的启发式支配另一个，它一定更好，因为它总是可以更接近地估计从任何给定状态到目标的距离。另外，**平凡启发式**定义为h(n) = 0，使用它将A*搜索简化为UCS。所有可接受的启发式都支配平凡启发式。平凡启发式通常在搜索问题的**半格**的底部，一个支配层次结构的底部。下图是一个包含了不同启发式h_a, h_b,和h_c的半格，从底部的平凡启发式到顶部的确切目标距离：



一般规则是，对多个可接受启发式应用最大函数也总是可接受的。这是因为任何给定状态的启发式输出的所有值受可接受性条件约束，$0 \leqslant h(n) \leqslant h^*(n)$。该范围内数字的最大值也必须落在同一范围内。对于多个一致的启发式，也可以轻松证明这一点。常见做法是为任何给定的搜索问题生成多个可接受/一致的启发式，并计算它们的输出值的最大值，以生成一个支配（因此优于）所有单个启发式的启发式。

搜索：总结

在这篇笔记中，我们讨论了搜索问题及其组成部分：一个状态空间，一组行为，一个转换函数，一个行为成本，一个初始状态和一个目标状态。智能体通过其传感器和执行器与环境互动。智能体函数描述了在所有情况下智能体的行为。智能体的理性是指智能体寻求最大化其期望效用。最后，我们用PEAS描述来定义我们的任务环境。

关于搜索问题，可以使用多种搜索技术来解决，包括但不限于我们在CS 188中学习的五种：

- *Breadth-first Search*
- *Depth-first Search*
- *Uniform Cost Search*
- *Greedy Search*
- *A* Search*

上述前三种搜索技术是无信息搜索的例子，而后两种是使用启发式来估计目标距离和优化性能的有信息搜索的例子。我们还区分了树搜索和图搜索算法。