# Introduction

- Incremental search method 增量搜尋方法(ex: DynamicSWSF-FP)

會重複使用先前搜索的信息，以比從頭解決每個搜索任務更快地找到一系列相似搜索任務的解決方案。

- Heuristic search methods 啟發式搜索方法(ex:A\*)

使用目標距離的近似形式的啟發式知識來集中搜索，比無信息的搜索方法更快地解決搜索問題。評估每個搜索節點的啟發式估計，以引導搜索過程。啟發式是一種評估函數，它提供了有關搜索空間中每個節點的期望性質或可能性的估計。

1. LPA\*

- 歸納DynamicSWSF-FP和A\* ，使用兩種不同的技術來減少其計劃時間

- 會隨著時間動態更新路徑(不一定要有新的障礙物)

2. Focused Dynamic A\*（D\*）

- 基於A\*，但是只更新受到影響的局部路徑(只有在路徑受影響才更新)

- 會從終點反向往起點進行搜索

3. D\* Lite

- 在D\* 的基礎上使用LPA\* 的規劃方式來修改局部路徑

# Lifelong Planning A\*

- A\* : 一次性搜索

- Lifelong Planning A\* : 在圖的邊緣成本改變時更新路徑

LPA\*是A\*的增量版本，適用於已知圖上的有限圖搜索問題，其邊緣成本隨時間增加或減少（也可用於建模添加或刪除的邊緣或頂點）。

S表示圖的有限頂點集。

Succ(s)表示頂點s的後繼集。

Pred(s)表示頂點s的前驅集。

c(s, s’)表示從頂點s移動到頂點s’的成本。

LPA始終確定從給定的前驅點start到給定的目標點goal的最短路徑，了解圖的拓撲和當前邊緣成本。

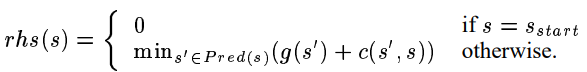
我們使用g\*(s)表示頂點s的起始距離，即從start到S的最短路徑的長度。

像A一樣，LPA使用啟發式h(s, s')來近似頂點s的目標距離。

啟發式需要是非負的並且一致，即遵守三角不等式h(goal, goal)=0和對於所有Succ(s)中的頂點s和s’（其中s不等於goal），滿足h(s, goal)<c(s, s’)+h(s’, goal)。

## The Variables

LPA維護每個頂點s的起始距離g\*(s)的估計，這些值直接對應於A搜索的g值。LPA將它們從一個搜索傳遞到另一個搜索。LPA還維護了一種起始距離的第二種估計。rhs值是基於g值的一步先行值，因此可能比g值更有信息。它們始終滿足以下關係（不變性1）

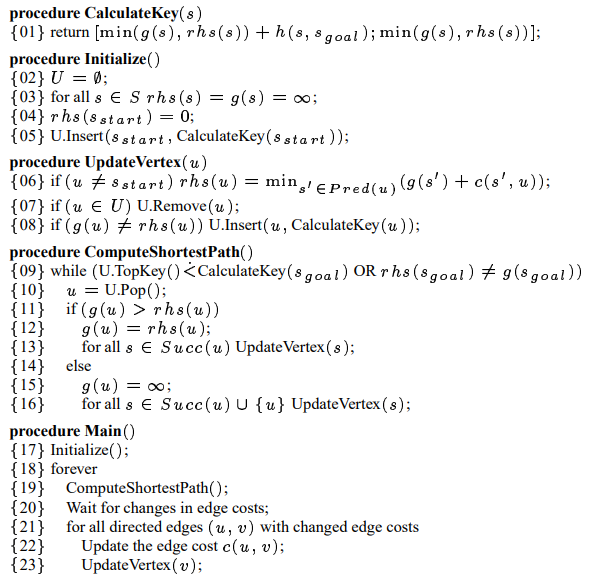


一個頂點如果其g值等於其rhs值，則稱為本地一致（locally consistent），否則稱為本地不一致（locally inconsistent）。

如果所有頂點都是本地一致的，那麼所有頂點的g值將等於它們各自的起始距離。在這種情況下，可以追踪回從起點到任何頂點u的最短路徑，方法是始終從當前頂點s （從u開始），轉換到任何最小化g(s')+c(s,s')的前驅頂點s' （可以任意解決平局），直到到達起點。 （這與圖1不同，在圖1中，使用目標距離而不是起始距離來確定最短路徑，可以沿著最短路徑從起點到終點移動，方法是始終從當前頂點s（從起點開始）移動到任何最小化c(s,s')+g(s')的後繼頂點s'，直到到達終點。）然而，當一些邊的成本發生變化後，LPA不會使所有頂點都變得本地一致。相反，它使用啟發式來集中搜索，僅更新計算最短路徑所需的相關g值。為此，LPA維護一個優先順序佇列。

在LPA中，優先順序佇列始終包含完全本地不一致的頂點（不變性2）。這些是LPA潛在需要更新的頂點，以使它們變得本地一致。優先順序佇列中頂點的優先順序始終與其關鍵字相同（不變性3），其關鍵字是一個具有兩個分量的向量：k(s) = [k1(s); k2(s)]，其中k1(s) = min(g(s), rhs(s)) + h(s, goal)和k2(s) = min(g(s), rhs(s))。

這裡，關鍵字的第一個分量k1(s)直接對應於A中使用的f值：f(s) = g(s) + h(s, goal)，因為LPA的g值和rhs值對應於A的g值，而LPA的h值對應於A的h值。關鍵字的第二個分量k2(s)對應於A\*的g值。關鍵字根據字典排序進行比較



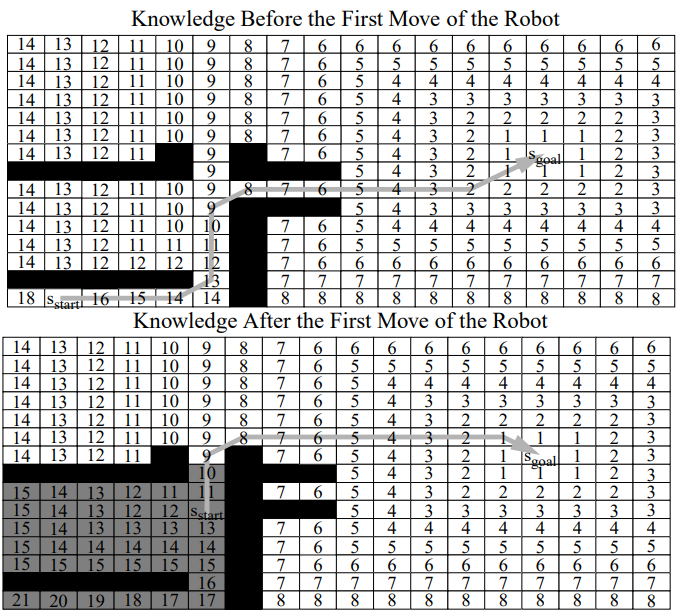
## The Algorithm

# D\* Lite

在未知地形的情況下，機器人需要觀察附近八個點來計算到目標的最短路徑。

如果在開始移動後，發現了有新的障礙物，就需要重新計算當前位置與目標位置的新距離，但是其他大部分的單元格通常都不會在最短路徑上。

D\* 會在有障礙物時對整張圖進行更改，而LPA\* 只會在邊緣成本發生變化時更改最短路徑。D\* Lite主要利用LPA\* 用來識別與新的最短路徑有關的單元格，並進行更新。



>識別灰色部分，並僅更新這些部分

## Search Direction

因為LPA\*是從終點往起點搜尋，所以在D\* Lite套用LPA\*時要把兩點交換(g值是目標距離的估計)。通過交換起始頂點和目標頂點並反轉所有的邊。在ComputeShortestPath()返回後，始終從當前頂點s移動到任何最小化c(s,s')+g(s')的後繼點s'，直到到達目標。

## Heap Reordering

Main()需要沿著由CalculatePath()計算的路徑移動機器人。每次機器人移動後並發現邊緣成本發生變化時，Main()都可以重新計算優先順序佇列中頂點的優先級。除非重新計算優先級，否則它們不滿足不變性3，因為它們基於相對於機器人的舊頂點計算的啟發式。

然而，由於優先順序佇列通常包含大量頂點，反復重新排序優先順序佇列可能是昂貴的。因此，D\* Lite使用一種源自D\*的方法，避免重新排序優先順序佇列，即這些優先順序是LPA用於相應頂點的優先順序的下限。

啟發式h(s，s’)現在需要是非負的，並滿足h(s，s’)<c(s，s’)和h(s，s’’) < h(s，s’)+h(s’，s’’)對於所有頂點s，s’，s’’，其中c(s，s’)表示從頂點s到頂點s’的最短路徑的成本。此要求不是限制性的，因為如果啟發式是通過放鬆搜索問題來得到的，則這兩個性質幾乎總是成立，並且適用於本文中使用的啟發式。機器人從頂點s移動到檢測到邊緣成本變化的某個頂點s’後，優先順序的第一個元素最多可能減少h(s，s’)。 （第二個分量不依賴於啟發式，因此保持不變。）因此，為了保持下限，D Lite需要從優先順序佇列中所有頂點的優先順序的第一個元素中減去h(s，s’)。但是，由於優先順序佇列中所有頂點的h(s，s’)都相同，如果不進行減法，優先順序佇列中頂點的順序不會改變。然後，當計算新的優先順序時，與優先順序佇列中的優先順序相比，它們的第一個分量相對於h(s，s’)太小。因此，每次邊緣成本發生變化時，都必須將h(s，s’)添加到它們的第一個分量中。如果機器人再次移動，然後再次檢測到成本變化，則這些常數需要相加。

我們將這個過程儲存在變數K中。因此，每次計算新的優先順序時，都必須將變數K添加到它們的第一個分量中，就像在{0,1}中所做的那樣。然後，在機器人移動之後，優先順序佇列中頂點的順序不會改變，優先順序佇列不需要重新排序。另一方面，這些優先順序始終是LPA\*相應優先順序的下限，優先順序的第一個分量已經增加了K的當前值。我們通過修改ComputeShortestPath()來利用這一特性，如下所示.

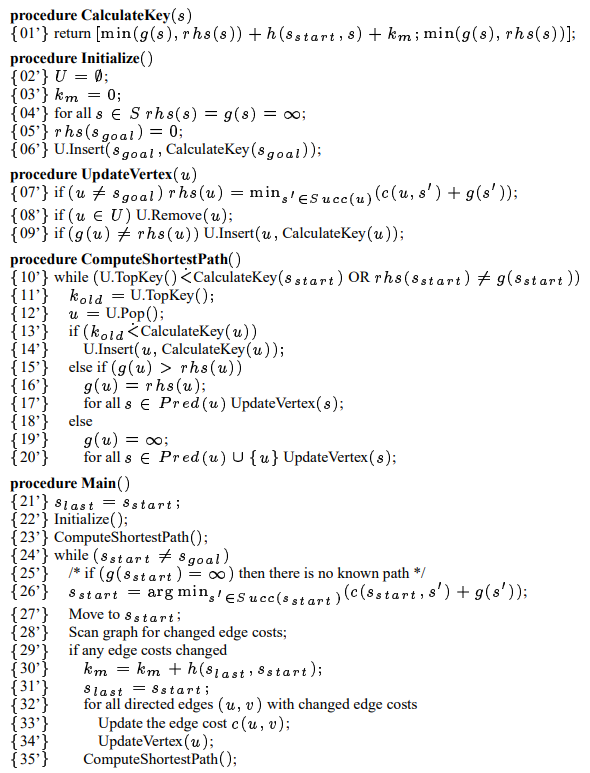
在ComputeShortestPath()從優先順序佇列{12}中刪除了一個具有最小優先順序Kold = U.Topkey()的頂點u之後，它現在使用CalculateKey()來計算它應該具有的優先順序。如果Kold < CalculateKey(u)，那麼它將使用由CalculateKey()計算的優先順序將被刪除的頂點重新插入優先順序佇列{14}。因此，對於優先順序佇列中的所有頂點，它仍然成立：在LPA的優先順序的第一個分量已經增加了Km的當前值之後，它們都是相應優先順序的下限。如果Kold > CalculateKey(u)，那麼它成立Kold = CalculateKey(u)，因為Kold是由CalculateKey()返回的值的下限。在這種情況下，ComputeShortestPath()以與LPA相同的方式展開頂點u。

## Optimizations

1. ComputeShortestPath()的終止條件

它可以更改以使ComputeShortestPath()更有效率。正如所述，ComputeShortestPath()終止的條件是起始頂點在本地一致，並且其關鍵字小於或等於U.TopKey(){10}。但是，ComputeShortestPath()在起始頂點不是本地不一致並且其關鍵字小於或等於U.TopKey()時也可以終止。

要理解為什麼會這樣，假設起始頂點在本地超不一致，其關鍵字小於或等於U.TopKey()。那麼，由於U.TopKey()是任何本地不一致頂點的最小關鍵字，因此其關鍵字必須等於U.TopKey()。因此，ComputeShortestPath()可能會擴展起始頂點，此時它將其g值設置為其rhs值。然後，起始頂點變得本地一致，其關鍵字小於或等於U.TopKey()，因此ComputeShortestPath()終止。此時，起始頂點的g值等於其目標距離。因此，當起始頂點不是本地不一致並且其關鍵字小於或等於U.TopKey(){10}時，ComputeShortestPath()已經可以終止。在這種情況下，起始頂點在ComputeShortestPath()終止後可能仍然保持本地不一致，因此其g值可能不等於其目標距離（但其rhs值是）。這並不是問題，因為g值不用於確定機器人應該如何移動。



## Analytical Results