人工智能：智能规划

饶洋辉
计算机学院
中山大学
raoyangh@mail.sysu.edu.cn
http://cse.sysu.edu.cn/node/2471

课件来源：中山大学刘咏梅教授等

规划

我们已经学习了知识表示与推理、搜索、强化学习的相关内容。

智能体并非被动地解决问题或推理，而是主动做出行动。

我们希望它们能智能地行动：

- 采取有目的的动作；
- 预测动作的预期效果；
- 组织动作来实现复杂的目标。

封闭世界假设（CWA）

用于表示世界状态的知识库是一系列取值为真的基原子（类似于数据库）。

封闭世界假设：

- 知识库中提到的常量就是论域的所有对象，知识库中没有提到的常量都不是论域中的对象；
- 如果一个基原子表达式不在已知的事实列表中，那么它的取值必定为假。

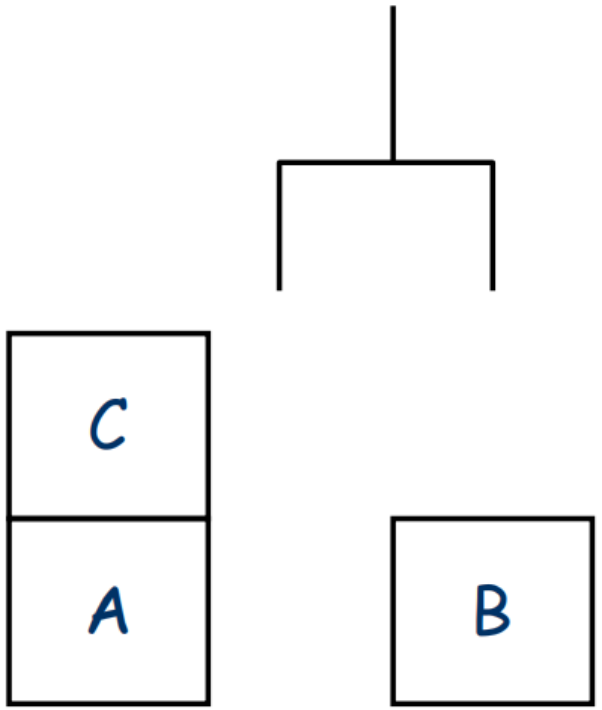
由此给定了初始状态的全部信息。

封闭世界知识库下的查询

CWA将知识库当作数据库，即查询得到的才为真。这样的知识库我们称为封闭世界知识库（CW-KB）。

有了CW-KB，我们可以给出任意复杂的一阶逻辑形式表达式的真值。这类似于数据库中的查询操作。

CW-KB示例



KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

Arbitrary queries:

1. clear(c) $\wedge$ clear(b)?
2. $\neg$on(b,c)?
3. on(a,c) $\vee$ on(b,c)?
4. $\exists$X.on(X,c)? (D = {a,b,c})
5. $\forall$X.ontable(X)
   $\rightarrow$ X = a $\vee$ X = b?

STRIPS表示

STRIPS（Stanford Research Institute Problem Solver）是Fikes和Nilsson于1971年开发的自动规划器。

这个名字后来被用来指代此规划划器的输入语言规范。

STRIPS下，动作被建模为修改世界的方式。

由于世界表示为CW-KB，因此STRIPS动作对应于CW-KB的更新操作。

一个动作会产生一个新的知识库，来描述新的世界，即执行动作之后产生的世界。

STRIPS动作

STRIPS用三个列表表示一个动作：

- 动作的前提条件列表
- 动作的增加效果列表
- 动作的删除效果列表

这些列表中包含变量，因此我们可以用一个表达式表达一类动作。

每次对动作中变量的实例化都会产生一个特定的动作对象。

STRIPS动作示例

pickup(X):

- Pre: {handempty, clear(X), ontable(X)}
- Adds: {holding(X)}
- Dels: {handempty, clear(X), ontable(X)}

pickup(X)是一个STRIPS运算符
而pickup(a)则为一个动作实例

STRIPS动作的操作

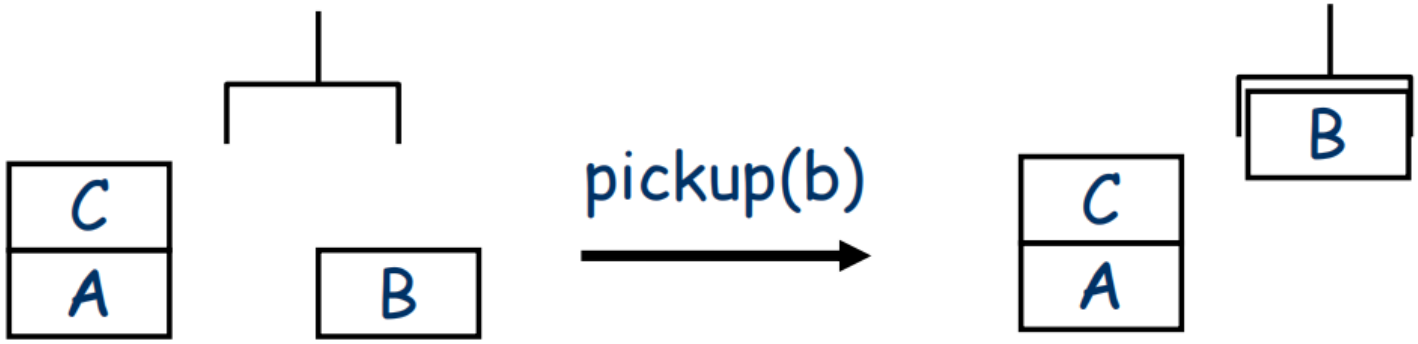
一个给定的STRIPS动作能应用到一个状态（CW-KB）需要满足：

- KB中它的前提条件列表都为真（这相当于成员测试，因为我们只有基原子表达式）

若STRIPS动作被应用到一个状态上，则新状态按如下规则产生：

- 从KB中去掉该动作的删除效果列表项
- 将动作增加列表中的所有项加到KB中

STRIPS动作的操作示例



pre = {handmpty,
clear(b),
ontable(b)}

add = {holding(b)}

del = {handmpty,
clear(b),
ontable(b)}

KB = {handempty
clear(c), clear(b),
on(c,a),
ontable(a), ontable(b)}

KB = { holding(b),
clear(c),
on(c,a),
ontable(a)}

积木世界的STRIPS运算符

pickup(X) (from the table)
Pre: {clear(X), ontable(X),handempty}
Add: {holding(X)}
Dels: {clear(X), ontable(X),handempty}

putdown(X) (on the table)
Pre: {holding(X)}
Add: {clear(X), ontable(X), handempty}
Del: {holding(X)}

积木世界的STRIPS运算符

unstack(X,Y) (pickup from a stack of blocks)
Pre: {clear(X), on(X,Y), handempty}
Add: {holding(X), clear(Y)}
Del: {clear(X), on(X,Y), handempty}

stack(X,Y) (putdown on a block)
Pre: {holding(X),clear(Y)}
Add: {on(X,Y), handempty, clear(X)}
Del: {holding(X),clear(Y)}

八数码规划问题

常量：

九个位置用P1-P9表示；

8块数码用T1-T8表示，空格则用B表示。

P1	P2	P3
P4	P5	P6
P7	P8	P9

八数码规划问题

谓词：

adjacent(X,Y)：X, Y 位置相邻

at(X,Y)：数码X在Y位置上

P1	P2	P3
P4	P5	P6
P7	P8	P9

八数码规划问题

运算符：

slide(T,X,Y)

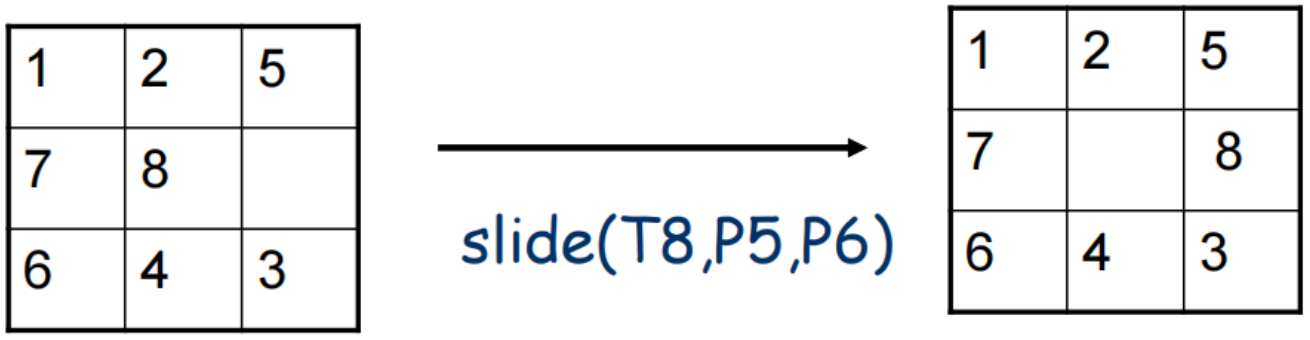
Pre: {at(T,X), at(B,Y), adjacent(X,Y)}

Add: {at(B,X), at(T,Y)}

Del: {at(T,X), at(B,Y)}

at(T1,P1), at(T5,P3),
at(T8,P5), at(B,P6), …,

at(T1,P1), at(T5,P3),
at(B,P5), at(T8,P6), …,



用搜索来求解规划问题

给定:

- 一个CW-KB，代表初始状态
- 一组将状态映射到新状态的STRIPS等运算符
- 一个目标状态（由公式指定）

这种规划问题就是确定一个动作序列，当它应用于初始CW-KB时，会产生一个新的满足目标的CW-KB。

这就是所谓的经典规划任务。

用搜索来求解规划问题

这是一个搜索问题，其中我们的状态表示是CW-KB。

初始的CW-KB即是初始状态。

这些动作是将状态（CW-KB）映射到新状态（新的CW-KB）。

我们可以检查任何状态是否满足目标（CW-KB）。

通常，目标是一系列基原子事实的合取，需要检查它们是否都包含在CW-KB中。

存在的问题与解决方案

问题：搜索树通常相当大。

- 随机重新配置9个块需要数千CPU秒数

但STRIPS等动作表示暗含了下述特点：

- 每个动作只影响一小部分事实

规划算法旨在利用这种表示的如下优势：动作显式改变的事实具有“局部性”。

我们将关注一种方法：使用启发式搜索的松弛启发式规划。

其中，启发式规则是领域无关的。

松弛问题

我们假设:

- 各动作的前提条件均为真值表达式
- 目标也由真值表达式组成

回顾八数码问题的动作约束:

- 只能将数码在相邻位置移动
- 数码只能被移动到空格上

在启发式搜索（即A搜索）的基础上，我们可以选择一个约束进行松弛，比如忽略动作的删除效果。通过松弛问题（此时启发式函数满足可采纳性$h(n) \leq h^*(n)$，相应的方法为A*搜索），我们可以得到规划解的一个启发式估计。

松弛化—积木世界STRIPS运算符

- pickup(X)
  Pre: {handempty, ontable(X), clear(X)}
  Add: {holding(X)}
  ~~Del: {handempty, ontable(X), clear(X)}~~
- putdown(X)
  Pre: {holding(X)}
  Add: {handempty, ontable(X), clear(X)}
  ~~Del: {holding(X)}~~
- unstack(X,Y)
  Pre: {handempty, clear(X), on(X,Y)}
  Add: {holding(X), clear(Y)}
  ~~Del: {handempty, clear(X), on(X,Y)}~~
- stack(X,Y)
  Pre: {holding(X),clear(Y)}
  Add: {handempty, clear(X), on(X,Y)}
  ~~Del: {holding(X),clear(Y)}~~

启发式估计计算（一般了解）

由此，我们了解到，最优的松弛规划解的大小可以作为原问题的可采纳的启发式函数值。

但是，计算最优松弛规划解是一个NP难问题。

那么，要如何计算这种启发式函数值呢?

- 从状态S开始构建分层结构来实现目标状态。
- 计算松弛规划中需要执行的动作数。
- 将其作为S到目标状态的启发式函数值。

可达性分析（一般了解）

- 从初始状态S_0开始
- 在状态层和动作层之间交替
- 查找前提条件包含在S_0中的所有动作
- 这些动作组成第一个动作层A_0
- 下一状态层$S_1 = S_0 \cup A_0$中所有动作的增加效果
- 重复执行此过程

可达性分析（一般了解）

一般来说：

- A_i为不包含在A_{i-1}中且前提条件包含在S_i中的动作集合
- $S_i = S_{i-1} \cup A_i$中所有动作的增加效果

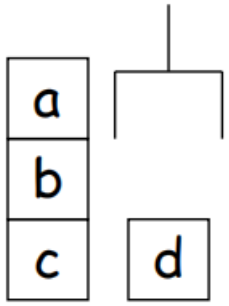
直观来说：

- A_i层的动作可以在规划中的第 i 步来执行
- S_i层的事实是可能被长度为 i 的规划实现的

迭代至：

- 目标状态在S_i 中被包含
- 或者S_i保持不变（稳定点）

示例（一般了解）

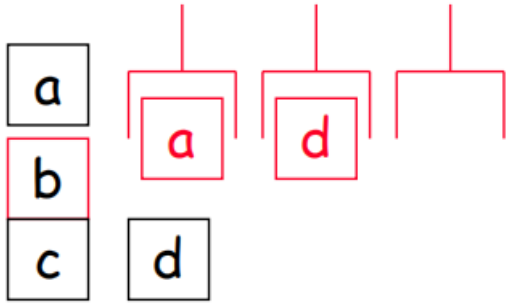


on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty

S_0

unstack(a,b)
pickup(d)

A_0

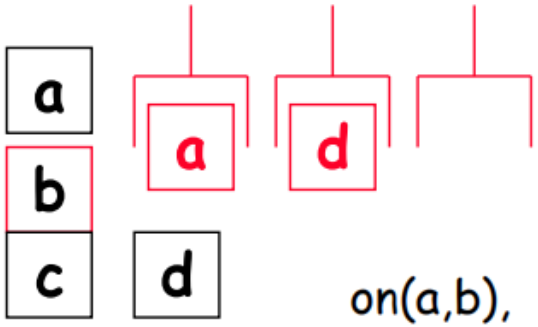


on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
handempty,
clear(d),
holding(a),
clear(b),
holding(d)

S_1

this is not a state as some of these facts cannot be true at the same time!

示例（一般了解）



on(a,b),
on(b,c),
ontable(c),
ontable(d),
clear(a),
clear(d),
handempty,
holding(a),
clear(b),
holding(d)

S_1

unstack(a,b)
pickup(d)

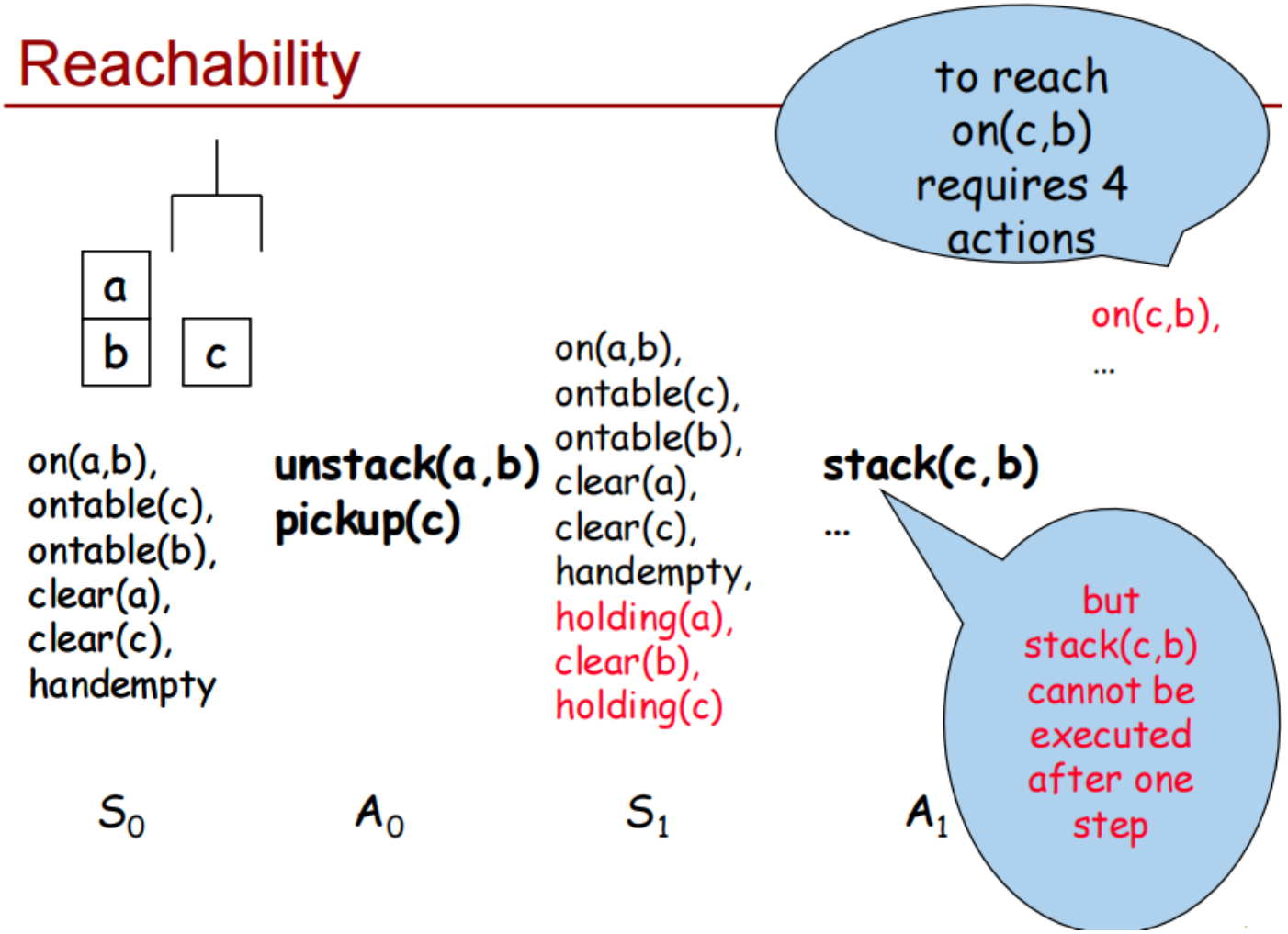
} from A_0

putdown(a),
putdown(d),
stack(a,b),
stack(a,a),
stack(d,a),
stack(d,b),
stack(d,d),
unstack(b,c)
...

} Impossible, but we don't know because we ignore dels.

A_1

示例（一般了解）


Reachability
to reach on(c,b) requires 4 actions
a
b
c
on(a,b), ontable(c), ontable(b), clear(a), clear(c), handempty
unstack(a,b)
pickup(c)
on(a,b), ontable(c), ontable(b), clear(a), clear(c), handempty, holding(a), clear(b), holding(c)
stack(c,b)
...
on(c,b),
...
but stack(c,b) cannot be executed after one step
S_0
A_0
S_1
A_1

动作计数（一般了解）

- 假设目标G包含在状态层中
- 我们想计算一个好的松弛计划
- 其思想是为每个 i 选择A_i的最小子集

动作计数（一般了解）

CountActions(G,S_K):
/* Here G is contained in S_K, and we compute the number of actions contained in a relaxed plan achieving G. */

If $K = 0$ return 0

Split G into $G_P = G \cap S_{K-1}$ and $G_N = G - G_P$

- G_P contains the previously achieved (in S_{K-1}), and
- G_N contains the just achieved parts of G (only in S_K).

Find a minimal set of actions A whose add effects cover G_N.

- cannot contain redundant actions,
- but may not be the minimum sized set
  (computing the minimum sized set of actions is the set cover problem and is NP-Hard)

NewG := G_P U preconditions of A.

return CountAction(NewG,S_{K-1}) + size(A)

集合覆盖（一般了解）

给定一组元素{1, 2, …, n}（全集U）和一组集合S，该集合中元素的并集等于全集。

找到S的最小子集，使其元素的并集等于全集，即找到一个最小的集合覆盖。

例如U = {1,2,3,4,5}, S = {{1,2,3},{2,4},{3,4},{4,5}}

最小的集合覆盖为{{1,2,3},{4,5}}

示例（一般了解）

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4], [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

Goal: f_6, f_5, f_1

Actions:

$[f_1]a_1[f_4]$

$[f_2]a_2[f_5]$

$[f_2, f_4, f_5]a_3[f_6$

示例（一般了解）

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$\mathbf{G = \{f_6, f_5, f_1\}}$

$\mathbf{G_N = \{f_6\}}$ **(newly achieved)**

$\mathbf{G_p = \{f_5, f_1\}}$ **(achieved before)**

示例（一般了解）

legend: [pre]act[add]

$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

$S_1 = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$\mathbf{S_2} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$

$\mathbf{G = \{f_6, f_5, f_1\}}$

We split G into G_P and G_N:

CountActs(G, S_2)

$G_P = \{f_5, f_1\}$ //already in S1

$G_N = \{f_6\}$ //New in S2

$A = \{a_3\}$ //adds all in G_N

//the new goal: $G_P \cup$ Pre(A)

$\mathbf{G_1 = \{f_5, f_1, f_2, f_4\}}$

Return

1 + CountActs(G_1, S_1)

示例（一般了解）

Now, we are at level S1

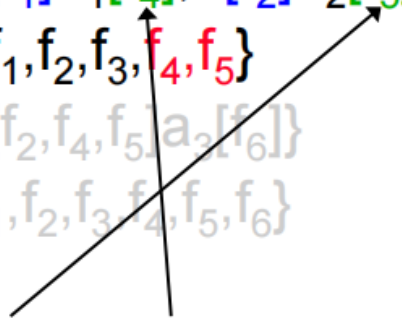
$S_0 = \{f_1, f_2, f_3\}$

$A_0 = \{[f_1]a_1[f_4],\ [f_2]a_2[f_5]\}$

$\mathbf{S_1} = \{f_1, f_2, f_3, f_4, f_5\}$

$A_1 = \{[f_2, f_4, f_5]a_3[f_6]\}$

$S_2 = \{f_1, f_2, f_3, f_4, f_5, f_6\}$



$G_1 = \{f_5, f_1, f_2, f_4\}$

We split G_1 into G_P and G_N:

$\mathbf{G_N} = \{f_5, f_4\}$

$\mathbf{G_P} = \{f_1, f_2\}$

CountActs(G_1,S_1)

$G_P = \{f_1, f_2\}$ //already in S0

$G_N = \{f_4, f_5\}$ //New in S1

$A = \{a_1, a_2\}$ //adds all in G_N

//the new goal: $G_P \cup$ **Pre(A)**

$\mathbf{G_2 = \{f_1, f_2\}}$

Return

2 + CountActs(G_2,S_0)

Next, we are at level S0, simply return 0

So, in total CountActs(G,S2)=1+2+0=3

动作计数（一般了解）

但是，动作计数不会计算最佳松弛计划，因为：

- 选择用于实现G_N的动作集（仅实现G的部分)不一定是最优的。
- 即使我们在每个阶段都选择了真正的最小集A，我们也可能无法获得整个规划的最小动作集合。因为在每个阶段选择的动作集合会影响在下一阶段可以使用的动作。

动作计数（一般了解）

寻找最优松弛规划是一个NP难问题。

所以动作计数不能在不使计算变得更加困难的情况下成为一个可采纳的启发式函数值。

从经验上看，动作计数的改进方法在许多规划任务上表现非常好。