人工智能基础

何　清
中国科学院大学
中国科学院计算技术研究所
中国科学院智能信息处理重点实验室
机器学习与数据挖掘课题组
heqing@ict.ac.cn

中国科学院
CHINESE ACADEMY OF SCIENCES

CH10 Classical Planning

经典的规划问题

ICT 中国科学院计算技术研究所
Institute of Computing Technology,Chinese Academy of Sciences

内容提要

- 规划问题定义（PDDL）为一个搜索问题
- 前向搜索，后向搜索，启发式搜索
- 从规划图获得启发式及提取规则

规划

- 规划是智能体对动作进行推理的一种主要形式，它很大程度上体现了智能体的智能性。同时，规划也是描述智能体行为的主要方式
- 规划是为了建立一个控制算法，使智能智能体能够为实现目标，对动作过程进行综合

经典规划问题

- 经典的规划理论认为规划要解决的问题
- 规划的输入
  - 用某种形式语言描述的初始世界状态
  - 用某种形式语言描述的智能体目标
  - 用某种形式语言描述的智能体可能采用的动作，通常也叫做领域知识
- 输出
  - 可以在某个满足初始状态描述的世界中执行并达到智能体目标的一个动作序列

经典规划的假设

- 原子执行时间
- 确定性结果
- 智能体全知
- 智能体唯一能动性

经典规划系统

- 人们对经典规划算法进行了大量的研究，提出了许多有效的算法，如STRIPS，TWEAK，UCPOP和SNLP等
- 新的进展主要在GraphPlan和SATPLAN方面

Making plans

Initial condition in KB:

$At(Agent, [1,1], S_0)$
$At(Gold, [1,2], S_0)$

Query: $Ask(KB, \exists\, s \;\; Holding(Gold, s))$

i.e., in what situation will I be holding the gold?

Answer: $\{s/Result(Grab, Result(Forward, S_0))\}$

i.e., go forward and then grab the gold

This assumes that the agent is interested in plans starting at S_0 and that S_0 is the only situation described in the KB

Making plans: A better way

Represent plans as action sequences $[a_1, a_2, \ldots, a_n]$

$PlanResult(p, s)$ is the result of executing p in s

Then the query $Ask(KB, \exists p \;\; Holding(Gold, PlanResult(p, S_0)))$
has the solution $\{p/[Forward, Grab]\}$

Definition of $PlanResult$ in terms of $Result$:

$$\forall s \;\; PlanResult([\,], s) = s$$
$$\forall a, p, s \;\; PlanResult([a|p], s) = PlanResult(p, Result(a, s))$$

Planning systems are special-purpose reasoners designed to do this type of inference more efficiently than a general-purpose reasoner

3.谓词逻辑表示的经典例子

机器人移盒子(1/5)

例2.4 机器人移盒子

解：分别定义描述状态和动作的谓词

描述状态的谓词：

TABLE(x)： x是桌子

EMPTY(y)： y手中是空的

AT(y, z)： y在z处

HOLDS(y, w)： y拿着w

ON(w, x)： w在x桌面上

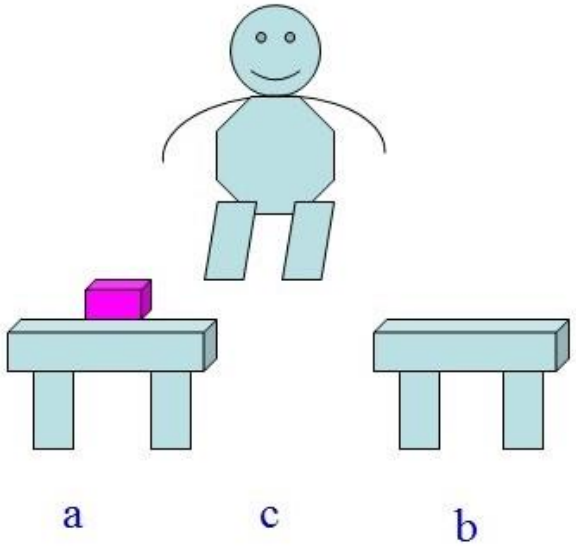
变元的个体域：

x的个体域是{a, b}

y的个体域是{robot}

z的个体域是{a, b, c}

w的个体域是{box}



3.谓词逻辑表示的经典例子

机器人移盒子(2/5)

问题的初始状态：

AT(robot, c)
EMPTY(robot)
ON(box, a)
TABLE(a)
TABLE(b)

问题的目标状态：

AT(robot, c)
EMPTY(robot)
ON(box, b)
TABLE(a)
TABLE(b)

机器人行动的目标是把问题的初始状态转换为目标状态，而要实现问题状态的转换需要完成一系列的操作。

描述操作的谓词

条件部分：用来说明执行该操作必须具备的先决条件，用谓词公式来表示。

动作部分：给出了该操作对问题状态的改变情况，通过在执行该操作前的问题状态中删去和增加相应的谓词来实现。

这些操作包括：

Goto(x, y)：从x处走到y处。

Pickup(x)：在x处拿起盒子。

Setdown(y)：在x处放下盒子。

8.3 USING FIRST-ORDER LOGIC

3.谓词逻辑表示的经典例子
机器人移盒子(3/5)

各操作的条件和动作：

Goto(x，y)

条件：AT(robot，x)

动作：删除表：AT(robot，x)

添加表：AT(robot，y)

Pickup(x)

条件：ON(box，x)，TABLE(x)，AT(robot，x)，EMPTY(robot)

动作：删除表：EMPTY(robot)，ON(box，x)

添加表：HOLDS(robot，box)

Setdown(x)

条件：AT(robot，x)，TABLE(x)，HOLDS(robot，box)

动作：删除表：HOLDS(robot，box)

添加表：EMPTY(robot)，ON(box，x)

各操作的执行方法：

机器人每执行一操作前，都要检查该操作的先决条件是否可以满足。如果满足，就执行相应的操作；否则再检查下一个操作。

8.3 USING FIRST-ORDER LOGIC

3.谓词逻辑表示的经典例子

机器人移盒子(4/5)

这个机器人行动规划问题的求解过程如下：

状态1(初始状态)

AT(robot, c)
EMPTY(robot)
ON(box, a)
TABLE(a)
TABLE(b)

开始 ==========>

状态2

AT(robot, a)
EMPTY(robot)
ON(box, a)
TABLE(a)
TABLE(b)

Goto(c, a) ==========>

状态3

AT(robot, a)
HOLDS(robot, box)
TABLE(a)
TABLE(b)

Pickup(a) ==========>

3.谓词逻辑表示的经典例子

机器人移盒子(5/5)

状态4

AT(robot, b)

Goto(a, b) HOLDS(robot, box)

==========> TABLE(a)

TABLE(b)

状态5

AT(robot, b)

Setdown(b) EMPTY(robot)

==========> ON(box, b)

TABLE(a)

TABLE(b)

状态6(目标状态)

AT(robot, c)

Goto(b, c) EMPTY(robot)

==========> ON(box, b)

TABLE(a)

TABLE(b)

8.3 USING FIRST-ORDER LOGIC

3.谓词逻辑表示的经典例子
猴子摘香蕉(1/4)

例2.5猴子摘香蕉问题

解：先定义谓词

描述状态的谓词：

AT(x, y)：x在y处

ONBOX：猴子在箱子上

HB：猴子得到香蕉

个体域：

x： {monkey, box, banana}

Y： {a, b, c}

问题的初始状态

AT(monkey, a)

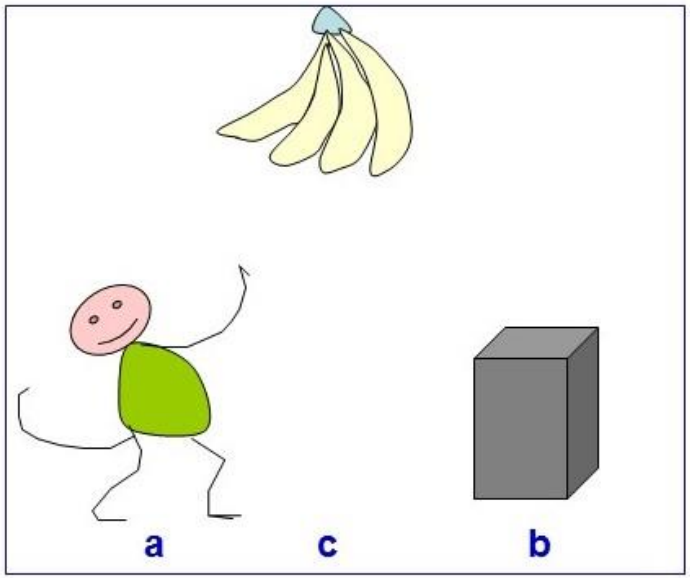
AT(box, b)

¬ ONBOX， ¬ HB

问题的目标状态

AT(monkey, c)， AT(box, c)

ONBOX， HB



3.谓词逻辑表示的经典例子

猴子摘香蕉(2/4)

描述操作的谓词：

Goto(u, v)：猴子从u处走到v处

Pushbox(v, w)：猴子推着箱子从v处移到w处

Climbbox：猴子爬上箱子

Grasp：猴子摘取香蕉

各操作的条件和动作：

Goto(u, v)

条件：¬ONBOX， AT(monkey, u)，

动作：删除表：AT(monkey, u)　　添加表：AT(monkey, v)

Pushbox(v, w)

条件：¬ONBOX， AT(monkey, v)， AT(box, v)

动作：删除表：AT(monkey, v)， AT(box, v)

添加表：AT(monkey, w)， AT(box, w)

Climbbox

条件：¬ONBOX， AT(monkey, w)， AT(box, w)

动作：删除表：¬ONBOX　　添加表：ONBOX

Grasp

条件：ONBOX， AT(box, c)

动作：删除表：¬HB　　添加表：HB

8.3 USING FIRST-ORDER LOGIC

3.谓词逻辑表示的经典例子
猴子摘香蕉(3/4)

猴子摘香蕉问题的求解过程如下：

状态1（初始状态）

AT(monkey, a)

开始 AT(box, b)

==========> ¬ ONBOX

¬ HB

状态2

AT(monkey, b)

Goto(a, b) AT(box, b)

==========> ¬ ONBOX

¬ HB

状态3

AT(monkey, c)

Pushbox(b, c) AT(box, c)

==========> ¬ ONBOX

¬ HB

3.谓词逻辑表示的经典例子

猴子摘香蕉(4/4)

状态4

AT(monkey, c)

AT(box, c)

Climbbox ==========> ONBOX

¬ HB

状态5(目标状态)

AT(monkey, c)

AT(box, c)

Grasp ==========> ONBOX

HB

为什么需要规划

- 上述例子中的一系列动作是如何做出的，这就需要规划

规划问题定义(PDDL)

- Planning Domain Definition Language（简称PDDL，规划区域定义语言）
- **规划**:设计一个动作规划以达到目标
  - ch03的基于搜索的问题求解Agent
  - ch07的逻辑Agent是一种规划Agent
- 用PDDL能够描述ch03和ch07无法描述的规划问题
- 规划研究者选择了**要素化**（factored representation）
- 用一组变量表示世界的一个状态的表示方法。可使用PDDL语言

将规划问题定义为一个搜索问题

- **状态：流（fluent）**（状态变量的同义词）**的合取**,这些流是基元的（无变量），无函数的原子
- 例如,Poor∧Unknown可能代表倒霉的Agent的状态
  在一个包裹传递问题中的状态可以是At(Truck 1 , Melbourne)∧At(Truck 2 ,Sydney)
- 在一个状态中，以下流是不允许的：At(x,y)（有变量）、￢Poor（）、At(Father(Fred),Sydney)（使用了函数符号）
- **动作模式**：动作可用动作模式来描述,动作模式隐式描述了ACTIONS(s)和RESULT(s, a)

例子：直升机动作模式

- 例如直升机从一个地点飞行到另一个地点的动作模式

  Action(Fly(p, from, to),

  PRECOND: At(p, from) ∧Plane(p) ∧Airport(from)∧Airport(to)

  EFFECT: ¬At(p,from) ∧At(p,to))
- 这个模式有动作名、模式中用到的所有变量列表、前提(precondition)、以及效果(effect)
- PDDL根据什么发生了变化来描述一个动作的结果;不提及保持不变的东西
- **动作的前提和效果都是文字(原子语句或原子语句的否定)的合取**。前提定义了动作能被执行的状态,效果定义了执行这个动作的结果

例子：直升机动作模式

- 通过为所有的变量代入值而得到基元动作:

  Action(Fly(P 1 , SFO, JFK),

  PRECOND: At(P 1 , SFO) ∧Plane(P 1) ∧Airport(SFO)

  ∧Airport(JFK)

  EFFECT: ¬At(P 1 , SFO) ∧At(P 1 , JFK))

- ACTIONS(s):一个动作a能在状态s被执行,如果s蕴涵了a的前提
- 蕴涵也可用集合语义来表达:s ⊨q当且仅当q中的正文字都在s中,且q中的负文字都不在s中。
- 用形式符号,可以说a∈ACTIONS(s)) ⇔ s ⊨ PRECOND(a)
- 如果状态s满足前提,我们称在状态s动作a是**适用的**(applicable)

例子：直升机动作模式

- RESULT(s,a)
- 在状态s执行动作a的结果定义为状态s’,它由一组流(fluent)表示
- 这组流由s开始,去掉在动作效果中以负文字出现的流(称之为删除列表delete list或DEL(a)),并增加在动作效果中以正文字出现的流(称之为增加列表add list或ADD(a)): $s' = RESULT(s,a) = (s - DEL(a)) \cup ADD(a)$

例子：直升机动作模式

- 初始状态是基元原子的合取
- 对于所有状态,使用封闭世界假设,这意味着任何没有提到的原子都是假的
- 目标就像前件:是可以含有变量的文字(正文字或负文字)的合取,像At(p,SFO)∧Plane 变量是存在量化的
- 当我们能找到一个动作序列,使得在蕴涵了目标的状态s结束,问题就得到了解决
- 例如,状态Plane(Plane 1)∧At(Plane 1 ,SFO)蕴涵了目标At(p, SFO)∧Plane(p)

STRIPS表示法

- 世界状态：一个世界状态（或简称为状态）是一组谓词的集合
- 目标：表示世界状态的谓词的合取
- 动作：动作用前提条件和动作效果来表示
  - 前提条件：动作执行前必须满足的状态
  - 动作效果：动作执行后可以保证为真的状态

STRIPS表示法-规划

- 动作模板：表示一组可能的动作，即包含自由变量的动作
- 动作步骤：表示一个具体的动作，由动作模板中的变量添加约束得到
- 规划：规划用动作步骤和动作步骤上的约束来表示，约束包括：
  - 动作步骤间的顺序约束，根据顺序约束可以分为全序规划和偏序规划
  - 动作步骤中变量或常量之间的等价、不等价、大于、小于等约束

经典规划举例

初始状态　　目标状态

规划表示

- 初始状态：

  On(table, A), On(A, B)

- 目标状态：

  On(B, A), On(table, B)

- 基本知识：

- 动作模板：

  Move(?Obj, ?dest)

  precondition： Clear(?Obj)

  effect： delete： On(?Obj, ?src)

  add： On(?Obj, ?dest)

希望得到的规划结果

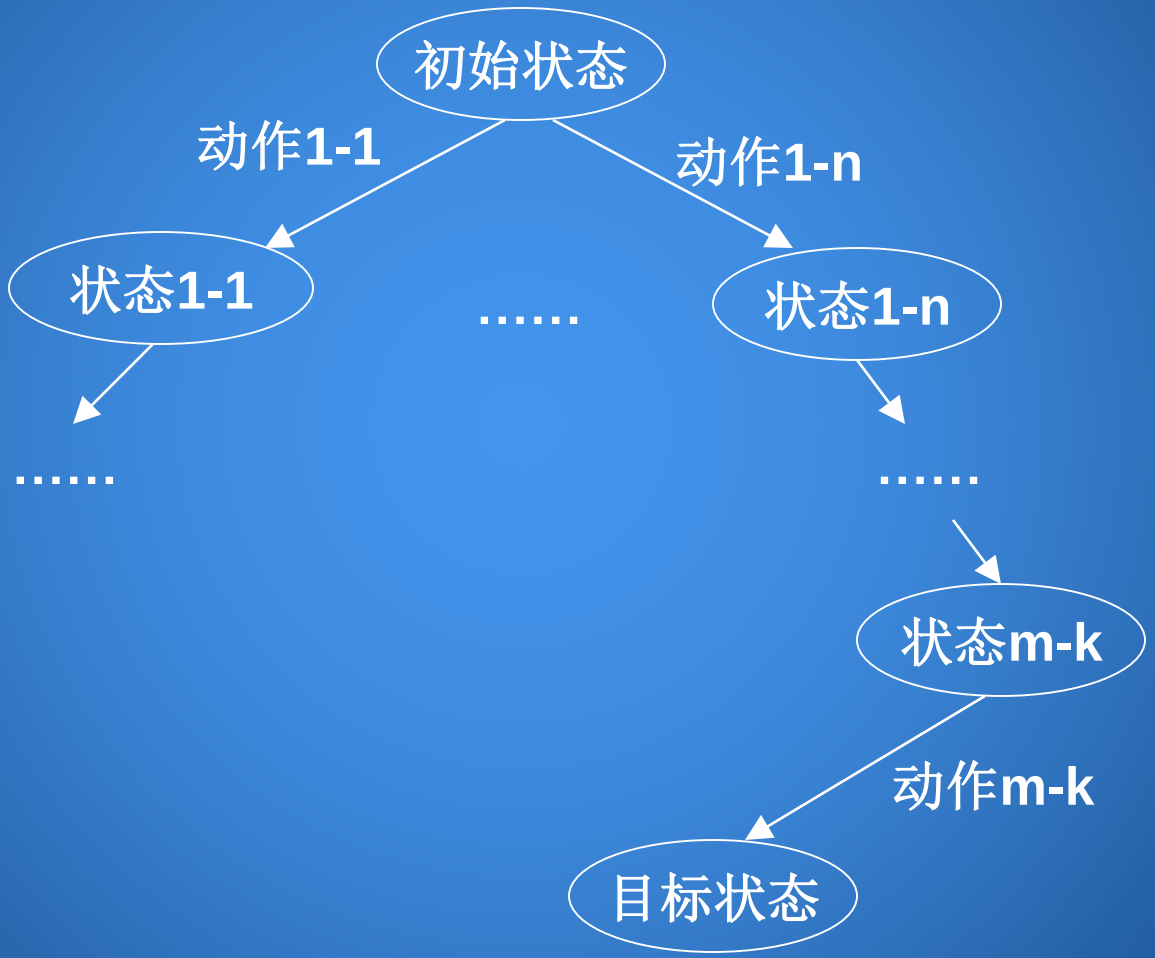
希望得到的结果：
第一步：Move(B, table)
第二步：Move(A, B)

实际的表示为：
Step1.Action=Move, Step1.Obj=B, Step1.dest=table,
Step2.Action=Move, Step2.Obj=A, Step2.dest=B,
Step1 < Step2

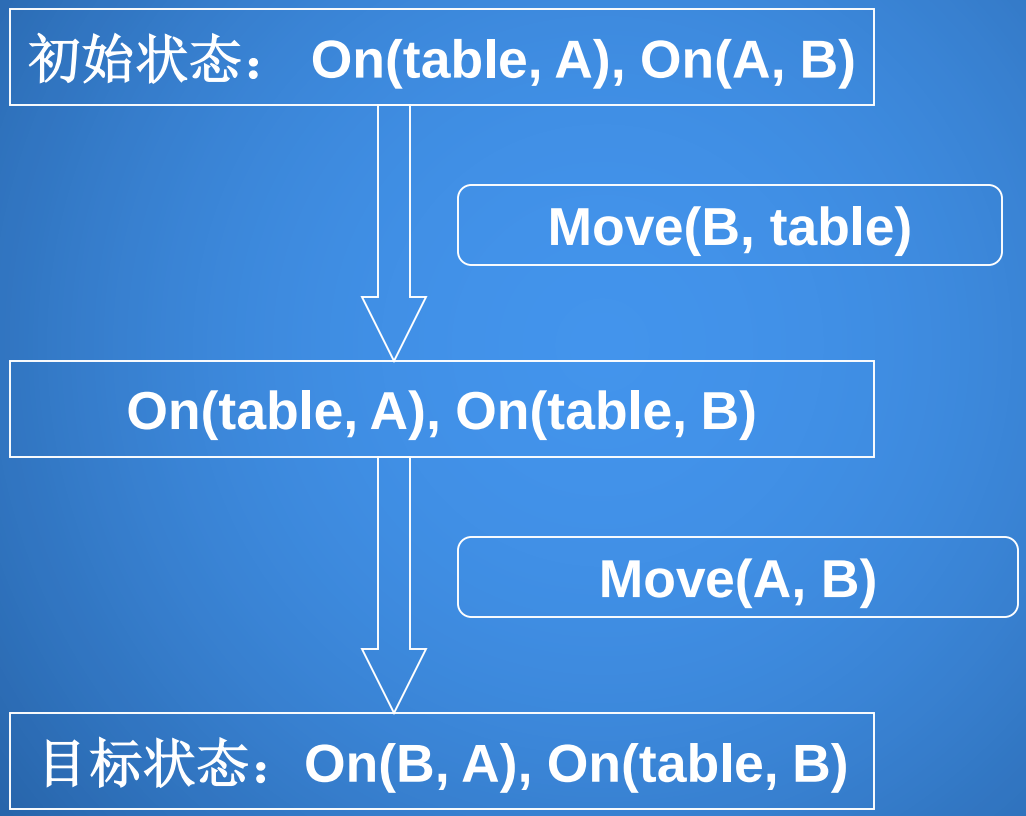
规划算法

- 规划算法实质上是一种搜索算法
- 搜索一个动作步骤序列，使得经过这个动作步骤的作用后，世界状态由初始状态变化为目标状态
- 算法应包含以下功能：
  - 选择动作：选择当前状态下能够执行的动作
  - 变量绑定（匹配）：根据当前状态，对动作中的变量进行约束
  - 回溯：发现死锁状态后，可以回退到以前的某个状态

规划搜索



规划结果



经典规划的复杂性

- 本节考虑规划的理论上的复杂性，并区分两个决策问题
- PlanSAT：询问是否存在解决一个规划问题的某个规划
  限界（bounded）
- PlanSAT：询问是否存在用于找到最优规划的、长度小于k的一个解
- 限界（bounded）PlanSAT是NP-完全的，而PlanSAT属于P
- 换言之，最优规划通常困难，但次优规划有时是容易的

状态空间搜索规划算法

- 前向状态空间搜索
- 规划问题的描述定义了一个搜索问题
- 启发式搜索算法(第3章)或局部搜索算法(第4章)可求解规划问题
- 前向搜索从初始状态开始搜索状态空间,寻找一个目标
  s'=RESULT(s,a) = (s –DEL(a))∪ADD(a)
- 前向搜索：从初始状态出发，使用问题的动作，向前搜索目标状态

状态空间搜索规划算法

- **后向搜索**：从目标的状态集出发，使用动作的逆，向后搜索初始状态
  - 前向搜索容易探索到无关动作
  - 规划问题常有大的状态空间
- 对前向搜索来说，显然，没有精确的启发式，即使相对小的问题实例也是无望的
- 但是规划的许多现实应用还是能自动导出非常强的独立于领域的启发知识，这使得前向搜索

后向相关状态搜索

- 从目标开始，向后应用动作，直到找到达到初始状态的步骤序列
- 在每一步考虑一组相关状态，而不是单个状态
- 从目标开始，对一组状态进行描述的文字之合取,可喜的是，PDDL表示的设计使得后退动作很容易

冲突

一个规划正确的必要条件是，在规划的每个实现中，每个动作的每个前提条件在动作执行之前都满足。造成规划不正确的原因是一个动作会破坏其他动作间的因果链，即冲突

因果链

因果链描述一个动作为另一个动作建立一个前提条件。

一个规划中，动作 E 和动作 U 之间存在因果链，当且仅当

1、$(E \prec U) \in OO$，即 E 和 U 之间有顺序约束；

2、$\exists e_E \in E_E$ 使得 $(e_E \approx p_U,)$ 其中 $p_U \in P_U$，即E的一个效果为U的一个前提；

3、$\forall T$，如果 $(E \prec T),(T \prec U)$ 并且 $\forall e_T \in E_T$，那么 $\neg(e_T \approx p_U)$ 并且 $\neg(e_T \approx \neg p_U)$，即发生在E和U之间的所有动作都不影响 p_U。

冲突

冲突描述的是一个因果链被其它规划中的某些动作破坏。

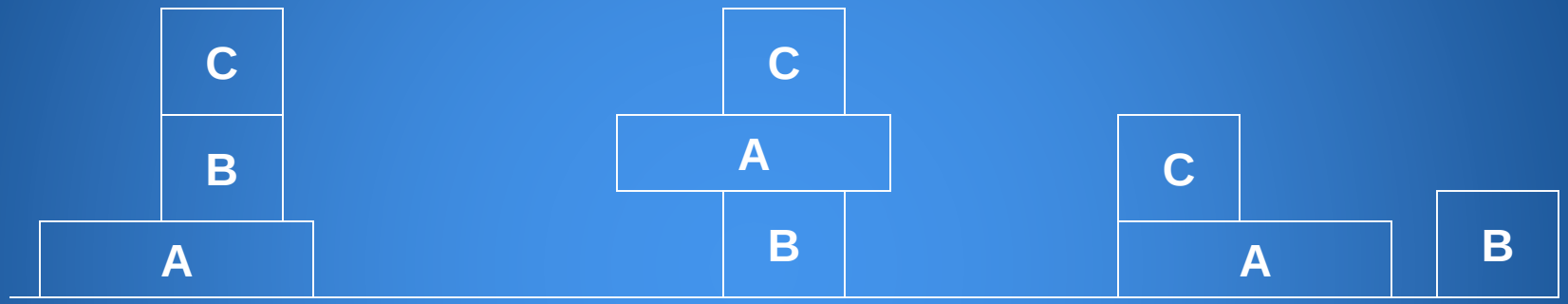
设存在因果链$\langle E, p_U, U\rangle$和另一个动作C，如果

1、$\neg(C \prec E)$ 并且$\neg(U \prec C)$ ，即C可能在E和U之间执行；

2、$\exists e_C \in E_C$使得$\neg(e_C \neq \neg p_U)$，即C的效果之一 e_C可能破坏p_U

那么称C为的$\langle E, p_U, U\rangle$一个威胁。当C在E和U之间被执行时，就发生冲突。

冲突举例



初始状态　　目标状态　　冲突状态

冲突描述

规划

Step1=Move(?obj,?dest), Step1.obj=C, Step1.dest≠B;

Step2=Move(?obj,?dest), Step2.obj=B, Step2.dest=Table;

Step3=Move(?obj,?dest), Step3.obj=A, Step3.dest=B;

Step4=Move(?obj,?dest), Step4.obj=C, Step4.dest=A;

Step1<Step2, Step2<Step3, Step3<Step4

当按照下面的约束执行时，发生冲突

Step1.dest=A

Step3无法执行

冲突消解方法

冲突威胁到规划的正确性，消解冲突的方法就是在规划之间添加新的约束，破坏冲突产生的条件。Chapman阐明了一个充分必要的冲突消解集合，其中包括“升级”、“降级”、“分离”和“引入修复”四种约束。

设动作的C一个效果 e_C 威胁因果链 $\langle E, p_U, U\rangle$ ，那么下列任何一个约束对于消解它都是充分的：$U \prec C$

（1）升级：加入顺序约束 $C \prec E$ ，

（2）降级：加入顺序约束 $p_U \neq e_C$

（3）分离：加入新的变量约束使得 ，

（4）引入修复：选择某个规划中已有动作或者新动作W，使得 $\exists e_W \in E_W$，$C \prec W \prec U$ 而且 $(e_W \approx p_U)$ 或者 $(e_W \approx \neg e_C)$

冲突消解方法举例

规划：

Step1=Move(?obj,?dest), Step1.obj=C, Step1.dest≠B;

Step2=Move(?obj,?dest), Step2.obj=B, Step2.dest=Table;

Step3=Move(?obj,?dest), Step3.obj=A, Step3.dest=B;

Step4=Move(?obj,?dest), Step4.obj=C, Step4.dest=A;

Step1<Step2, Step2<Step3, Step3<Step4

当按照下面的约束执行时，发生冲突：

Step1.dest=A

Step3无法执行。

消解方法：

添加约束Step1.dest≠A

UCPOP算法

- UCPOP是一个用Common Lisp写的偏序规划器
- UCPOP算法不断修改未完成的规划直到所有目标和后续子目标都被满足
- 修改包括向其中加入新动作步骤，向其中加入等价或不等价绑定来约束自由变量，向中加入顺序约束来安排动作步骤的先后顺序

UCPOP算法主要步骤

UCPOP算法的主要步骤：

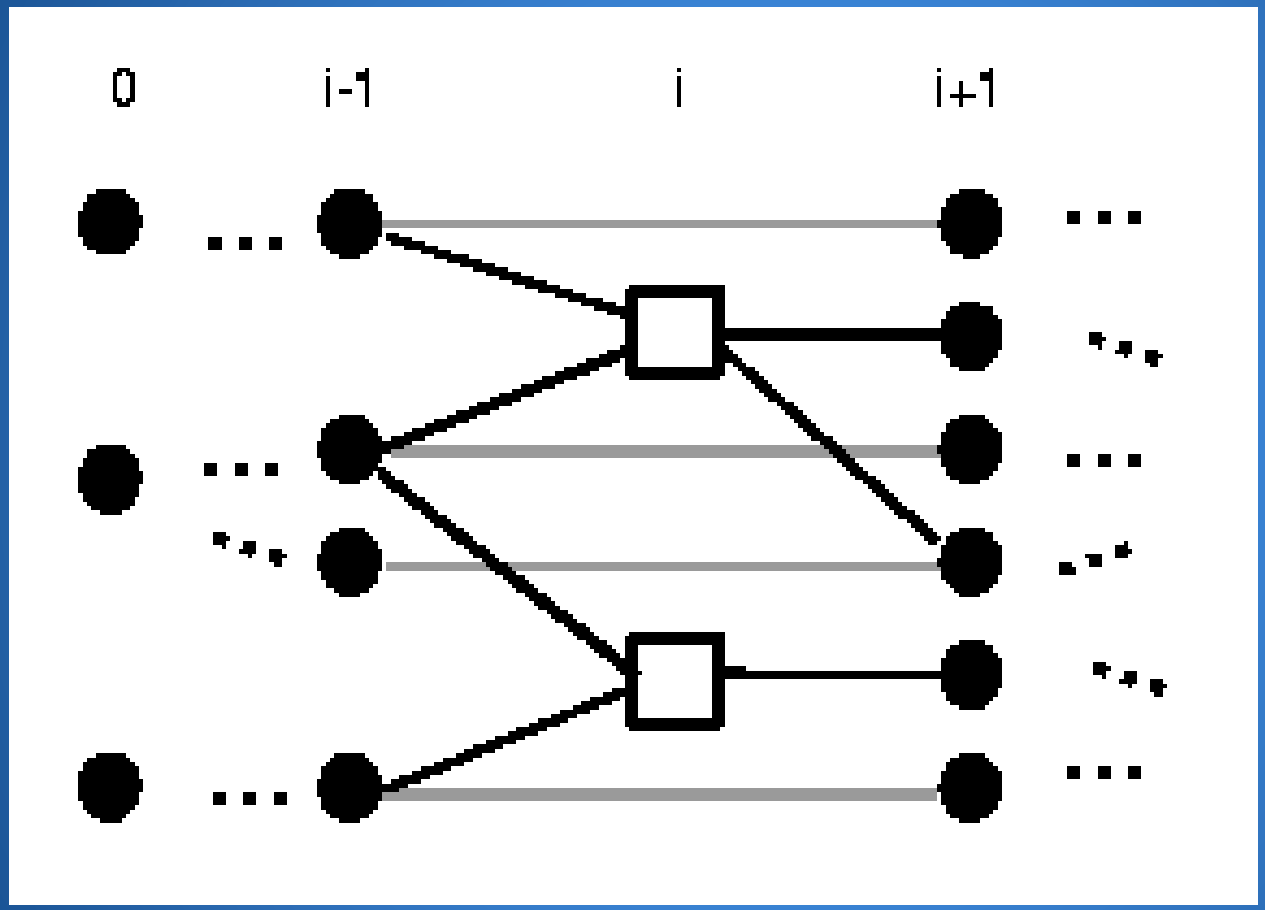
1. 终止条件
2. 选择子目标
3. 选择动作
4. 子目标生成
5. 因果链保护
6. 递归

GraphPlan图规划算法

- 图规划算法包括两个交替进行的过程：
  - 图扩充和方案搜索

    图扩充过程对规划图按时间顺序向前扩充，直到满足规划存在的必要条件（不是充分条件）
  - 然后方案搜索过程在图中进行反向链搜索，寻找解决问题的方案。如果没有找到解决方案，则进一步扩充规划图
- 重复上述循环

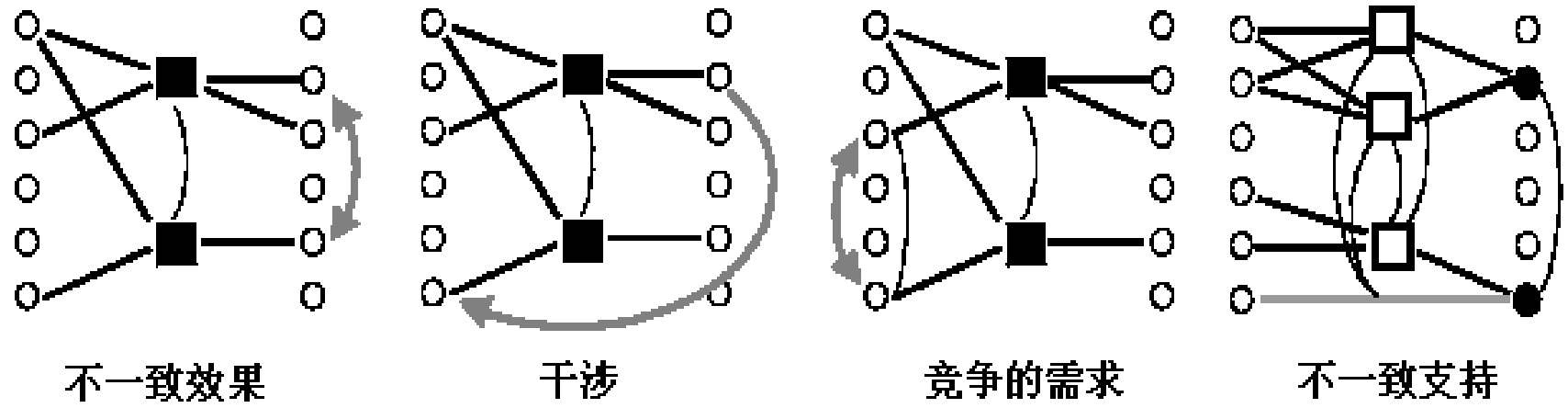
规划图中的节点



规划图中动作间的互斥关系

- 第i层的两个动作实例是互斥的，如果下列条件中至少有一条成立：
  - 不一致效果：一个动作的效果是另一个动作效果的否定
  - 干涉：一个动作删除了另一个动作的前提
  - 竞争的需求：两个动作在i-1层中具有互斥的前提
- 第i层的两个状态是互斥的，如果
  - 一个状态是另一个状态的否定，或者
  - 获得两个状态的方法（i-1层中的动作）是两两互斥的（不一致支持）

互斥关系的图形表示



规划的启发式

- 没有好的启发式函数，无论前向后向都不高效
- 可采纳的启发式。可以通过更容易求解的松弛问题导出
- 将搜索问题想成一个图，节点为状态，边为动作
- 问题是要找一条连接初始状态到目标状态的路径

  有2个方法来松弛这个问题
  - 加入更多的边，使得路径更容易被找到
  - 多个节点组合到一起，将状态空间抽象为具有更少状态的形式
- 忽略前提启发式:省略删除列表启发式

  定义启发式的关键思想是分解

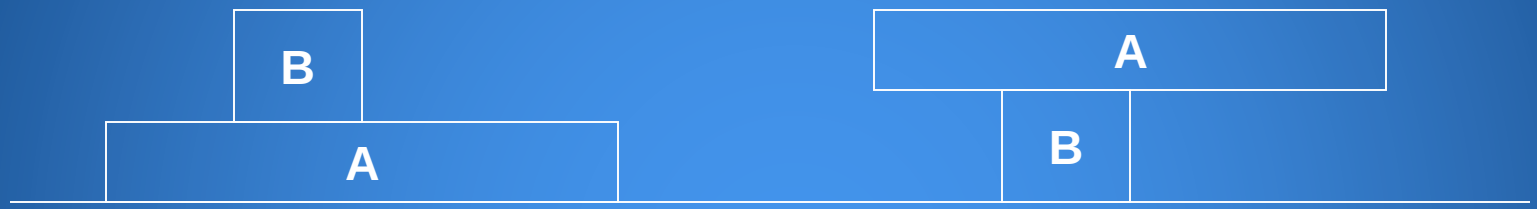
最后一个问题

- 决定哪些动作是后退的候选动作，在前向中我们选择适用的动作——在规划中可能是下一个步骤的那些动作
- 在后向中，我们需要相关的动作——导致当前目标状态的规划中可以作为最后一个步骤的那些动作
- 一个动作要与一个目标相关，则必须明显对目标有贡献
- 尽管后向搜索是分支因子低于前向，然而，后向使用状态集而不是单个状态的事实使得它更加难以想出好的启发知识，这就是当前主流偏向前向的主要原因

多智能体环境下的规划问题

- 多智能体系统规划的特殊之处
  - 智能体可以请求其它智能体协作执行一个联合动作来实现自己的目标
  - 智能体之间的动作冲突必须通过智能体之间的协商来避免

协作动作举例



初始状态　　目标状态


agent1　　agent2

协作动作的表示

动作模板：

Move(?Obj, ?dest)

actor： actor1

precondition： Clear(?Obj) and (Weight(?Obj, ?x) and ?x<=3)

delete： On(?Obj, ?src)

add： On(?Obj, ?dest)

Joint-Move(?Obj, ?dest)

actor： actor1, actor2

precondition： Clear(?Obj) and (Weight(?Obj, ?x) and 3<?x<=6)

delete： On(?Obj, ?src)

add： On(?Obj, ?dest)

物体： Weight(A, 4), Weight(B, 1)

初始状态：

On(table, A), On(A, B)

目标状态：

On(B, A), On(table, B)

任务分配

规划结果如下：

Step1.Action=Move，Step1.Obj=B, Step1.dest=table,

Step2.Action=Joint-Move，Step2.Obj=A, Step2.dest=B,

Step1 < Step2

然后根据智能体能力库，对动作步骤中的角色进行任务分配。

智能体能力库如下：

agent1：CanDo Move as actor1

CanDo Joint-Move as actor1,actor2

agent2：CanDo Joint-Move as actor1,actor2

需要分配的角色：

Step1(Move).actor1：可选智能体{agent1}

Step2(Joint-Move).actor1：可选智能体{agent1, agent2}

Step2(Joint-Move).actor2：可选智能体{agent1, agent2}

规划间的冲突

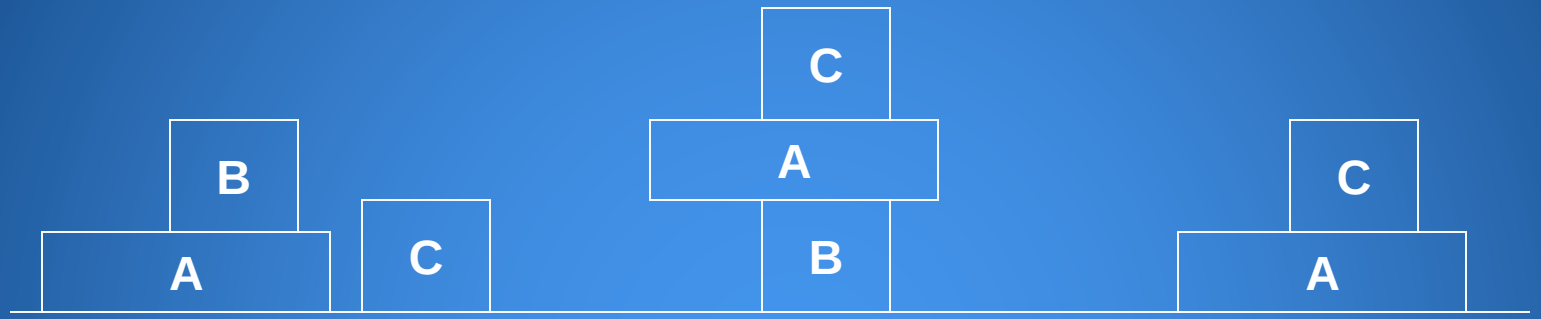
规划间冲突分两种：

- 目标冲突：两个规划的目标冲突。
- 动作冲突：一个规划中的一个动作会破坏另一个规划中的一个动作的前提。

解决方法：

- 目标冲突：放弃一个不太重要的目标。
- 动作冲突：在动作之间添加时序约束。

规划间冲突举例



初始状态　　agent1目标状态　　agent3目标状态

agent1　　agent2　　Agent 3

规划间冲突举例

agent1的目标： On(B,Table), On(A,B), On(C,A)
规划： Move(B, Table)， Move(A, B)， Move(C, A)

agent2的目标： On(C,A)
规划： Move(B, Table)， Move(C, A)

当按照下面的顺序执行时， 发生冲突：
Agent1： Move(B, Table)
Agent2： Move(C, A)
Agent1： Move(A, B)无法执行

规划间冲突消解方法

- 规划间冲突的消解方法也是添加约束，包括顺序约束和变量绑定约束，不过这些约束是在规划之间的约束
- 规划间的约束要靠多个智能体的协调来实现
- 顺序约束：动作执行过程中添加同步动作WaitFor和Inform
- 变量绑定约束：智能体之间通信传递变量约束的更新

冲突消解方法举例

agent1的规划：

Move(B, Table)， Move(A, B)， Move(C, A)

agent2的规划：

Move(B, Table)， Move(C, A)

当按照下面的顺序执行时，发生冲突：

Agent1： Move(B, Table)

Agent2： Move(C, A)

Agent1： Move(A, B)无法执行

消解方法：

升级：添加顺序约束agent1.Move(A, B)<agent2.Move(C, A)

约束一致性

- 通过添加新约束的方法可以消解冲突，但是新约束可能会与原有的约束发生矛盾
- 矛盾分为两种：
  - 顺序约束不一致
  - 变量绑定约束不一致

顺序约束不一致

顺序约束不一致。令R_1 和 R_2为两个顺序约束的集合，R_1 与R_2 在多个智能体的规划中矛盾，当且仅当 $\exists \alpha, \beta \in \sum SO$ 使得：

$(\alpha \prec \beta) \in TR(R_1 \cup \sum OO)$ 并且 $(\beta \prec \alpha) \in TR(R_2 \cup \sum OO)$ 。

例如：agent1.step1<agent2.step1

agent2.step1<agent1.step2

agent1.step2<agent1.step1

变量绑定约束不一致

令R_1和R_2为两个等价和不等价约束的集合。令$Co(R)$为R中所有等价约束，$Nonco(R)$为R中所有不等价约束。

R_1与R_2在多个智能体的规划中矛盾，当且仅当存在变量x和y使得：

$(x \approx y) \in ER(Co(R_1) \cup Co(R_2) \cup \sum Co)$ 而且

$(x \neq y) \in (Nonco(R_1) \cup Nonco(R_2) \cup \sum Nonco)$ 。

例如：agent1.step1.dest=agent2.step1.obj

agent1.step1.dest=A

agent2.step1.obj=B

多智能体规划的过程

- 规划生成动作步骤
- 检测规划间的冲突
- 如果检测到冲突，则消解冲突
- 添加约束消解冲突时，要进行约束一致性判断
- 任务分配

多智能体规划的两种基本方式

- 集中式规划
  - 由中央协调智能体来进行规划，然后将规划结果分配给各智能体执行。
  - 缺点是当任务很多，或者很复杂时，中央协调智能体的负担很重，成为系统的瓶颈
- 分布式规划
  - 每个智能体自行进行规划，冲突的检测和消解通过智能体间的协商解决
  - 缺点是冲突的检测和消解比较困难

新的研究方向

- 动态规划
  - 条件规划
  - 概率规划
- 不完备信息规划

欢迎批评指正！
谢谢！

2019.11.05
heqing@ict.ac.cn