Planning Problems

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

Contents

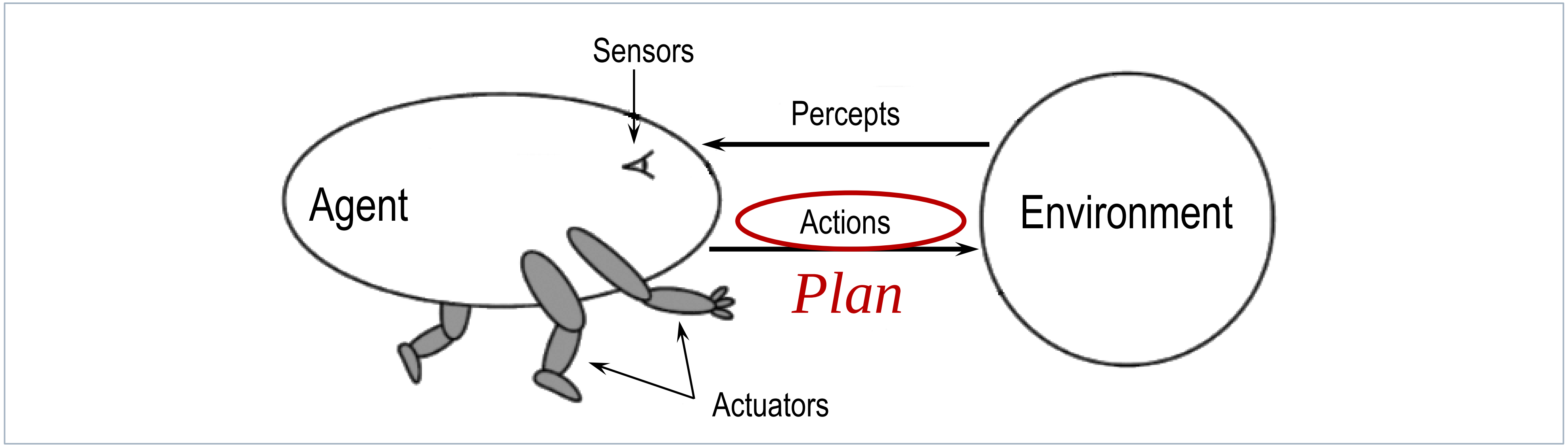
What is Planning 什么是规划

- We have defined AI as the study of rational action. Action is a critical part for an intelligent agent.

  我们已经把人工智能定义为理性动作的研究。动作是智能体的一个关键部分。

- Planning means devising a plan of action to achieve one's goals.

  规划意味着制定一套行动计划来达到既定的目标。



What are Planning Problems 什么是规划问题

- A longer definition 较长的定义

  given the descriptions for a problem in the real world:

  给定现实世界中一个问题的描述：

  - the initial states, the desired goals, and the possible actions,

    初始状态、预期目标、和可能的动作，

  planning is to find a plan that is generating a sequence of actions that leads from any of the initial states to one of the goal states.

  规划是找到一个计划：它产生从任何初始状态到达一个目标状态的一系列动作。

- A shorter definition 较短的定义

  devising a plan of action to achieve one's goals.

  制定一个达到既定目标的行动计划。

What is Classical Planning 什么是经典规划

Classical planning has following features:

经典规划具有如下特征：

fully observable	完全可观测
a unique known initial state	唯一已知初始状态
static environments	静态环境
deterministic actions	确定性的动作
can be taken only one at a time	每次仅一个动作
a single agent	单一智能体

Simplest planning known as Classical Planning

简单规划被称为经典规划

Planning Difficulties 规划的难度

Properties 特性	Questions 问题
actions 动作	• deterministic or nondeterministic? 确定性还是不确定性 • have a duration? 有一段持续时间 • can take concurrently or only one at a time? 可并发执行还是串行
state variables 状态变量	• discrete or continuous? 离散还是连续
initial states 初始状态	• finite or arbitrarily many? 有限还是任意多
objective 目标	• to reach a designated goal state? 要达到指定的目标状态 • to maximize a reward function? 要最大化回报函数
agents 智能体	• only one or several? 仅一个还是多个 • cooperative or selfish? 合作还是单干

Problem-solving Agent vs. Planning Agent 问题求解智能体与规划智能体

	Problem-solving agent 问题求解智能体	Planning agent 规划智能体
State (Initial / Goal) 状态（初始/目标）	Atomic representation 原子表示	Factored representation 因子表示 -- collection of variables 变量的集合
Action 动作	Instantiated actions 实例化动作	Actions schemas 动作模式 -- use Planning Domain Definition Language (PDDL) 使用规划领域定义语言PDDL
Heuristic 启发法	Domain-specific heuristics 领域特定启发法	Domain-independent heuristics 领域无关启发法

About PDDL 关于PDDL

- PDDL (Planning Domain Definition Language) is an attempt to standardize AI planning languages. First developed in 1998.

  PDDL（规划领域定义语言）是对AI规划语言标准化的一种尝试。于1998年首次开发。

- The latest version is PDDL 3.1 (2011), its BNF syntax definition can be found from the IPC-2014 homepage:

  最新版是PDDL 3.1 (2011)，其BNF语法定义可以从IPC-2014主页找到：

  https://helios.hud.ac.uk/scommv/IPC-14/software.html

The PDDL used in this course 本课程使用的PDDL

- It select a simple version, and alter its syntax to be consistent with the rest of the course.

  选择了最简单的版本，并且修改了其语法，以便与课程的其它部分保持一致。

Three Components to Define a Planning Task 定义规划任务的三个要素

- **State** 状态
  - represented as a conjunction of fluents (fluents: a relation that varies from one to next).
    表示为变数的合取（fluents：从一个到另一个变化的关系）。
    e.g., $At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$.
- **Actions** 动作
  - described by a set of action schemas, implicitly define the functions.
    用一组动作模式描述，隐式定义函数。
    e.g., $\textsc{Action}(s)$, $\textsc{Result}(s, a)$.
- **Goal** 目标
  - represented as a conjunction of literals (literals: an elementary proposition).
    表示为文字的合取（literals：一个基本的命题）。
    e.g., $At(p, SFO) \wedge Plane(p)$.

Example 1: Air cargo transport 航空货物运输

- **Problem:** 问题

  To load cargo, then fly, and unload it. 装货、然后飞行、再卸货。

  - from *SFO* (San Francisco Airport) to *JFK* (New York John Fitzgerald Kennedy Airport). 从SFO（旧金山机场）到JFK（纽约约翰·菲茨杰拉德·肯尼迪机场）。

- **Actions:** 动作
  - *Load*(.)
  - *Unload*(.)
  - *Fly*(.)

- **Predicates:** 谓词
  - *In*(c, p) -- cargo c is inside plane p, 货物c在飞机p内，
  - *At*(x, a) -- object x (either plane or cargo) is at airport a. 物体x（飞机或货物）在机场a。

Example 1: Air cargo transport 航空货物运输

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \land Cargo(C_1) \land$
$\quad Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \land Airport(JFK) \land Airport(SFO))$

$Goal(At(C_1, JFK) \land At(C_2, SFO))$

$Action(Load(c, p, a),$
$\quad$ PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $\neg At(c, a) \land In(c, p))$

$Action(Unload(c, p, a),$
$\quad$ PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
$\quad$ EFFECT: $At(c, a) \land \neg In(c, p))$

$Action(Fly(p, from, to),$
$\quad$ PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
$\quad$ EFFECT: $\neg At(p, from) \land At(p, to))$

A PDDL description for the air cargo transportation planning problem
针对航空货物运输规划问题的PDDL描述

Example 1: Air cargo transport 航空货物运输

Solution 解答

$$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK), Load(C_2, P_2, JFK), Fly(P_2, JFK , SFO), Unload(C_2, P_2, SFO)]$$

Spurious action 谬误动作

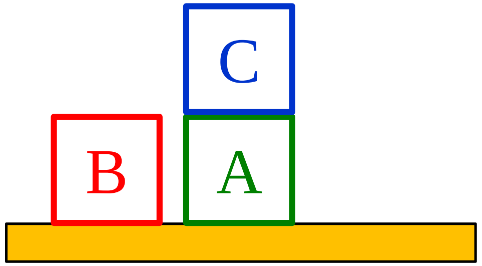
$$Fly(P_1, JFK, JFK)$$

Contradictory effect 矛盾作用

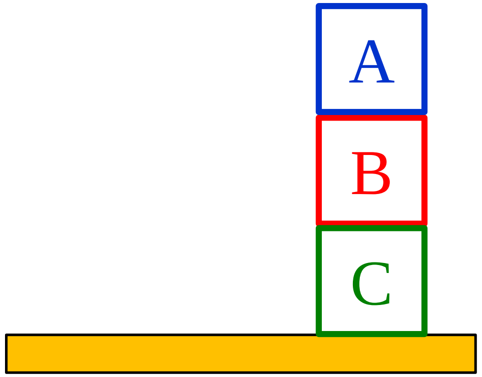
$$At(P_1, JFK) \wedge \neg At(P_1, JFK)$$

Example 2: The blocks world 积木世界

- Problem: 问题
  - three blocks sitting on a table, the goal is to get block A on B, and block B on C.
    桌子上放着三块儿积木，目标是使积木A放在B、并且B放在C上。
- Actions: 动作
  - *Move*(.), *MoveToTable*(.)
- Predicates: 谓词
  - *On*(b, x) -- block b is on x (either another block or table)
    积木b在x上（其它积木或桌子）
  - *Clear*(x) -- true when nothing is on x.
    当x上空无一物时为真。



Start State



Goal State

Example 2: The blocks world 积木世界

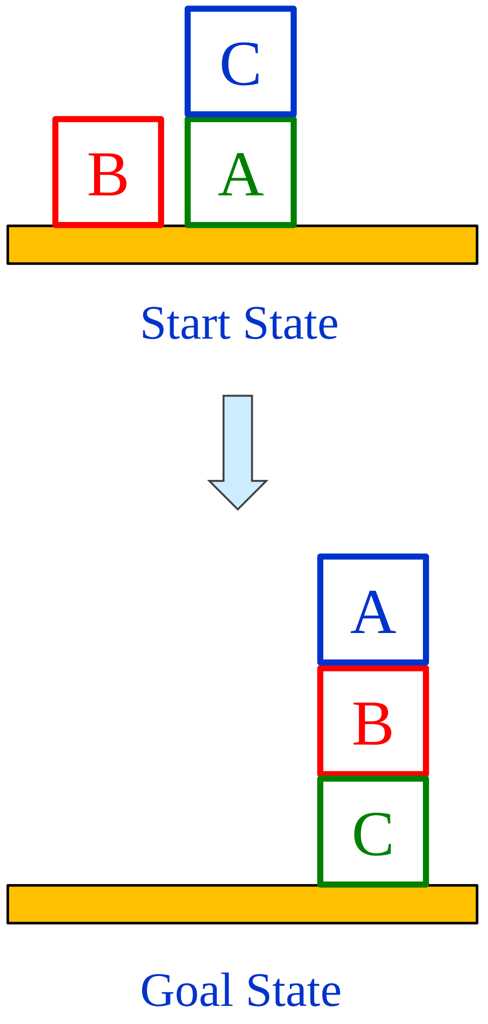
$$Init(On(A, \mathrm{Table}) \land On(B, Table) \land On(C, A)$$
$$\land\ Block(A) \land Block(B) \land Block(C) \land Clear(B) \land Clear(C))$$

$$Goal(On(A, B) \land On(B, C))$$

$$Action(Move(b, x, y),$$
$$\text{PRECOND: } On(b, x) \land Clear(b) \land Clear(y) \land$$
$$Block(b) \land Block(y) \land$$
$$(b \neq x) \land (b \neq y) \land (x \neq y),$$
$$\text{EFFECT: } On(b, y) \land Clear(x) \land \neg On(b, x) \land \neg Clear(y))$$

$$Action(MoveToTable(b, x),$$
$$\text{PRECOND: } On(b, x) \land Clear(b) \land Block(b) \land (b \neq x),$$
$$\text{EFFECT: } On(b, Table) \land Clear(x) \land \neg On(b, x))$$

A PDDL description for the blocks world problem
针对积木世界问题的PDDL描述



Thank you for your attention!

AI

Classic Planning

School of Electronic and Computer Engineering
Peking University

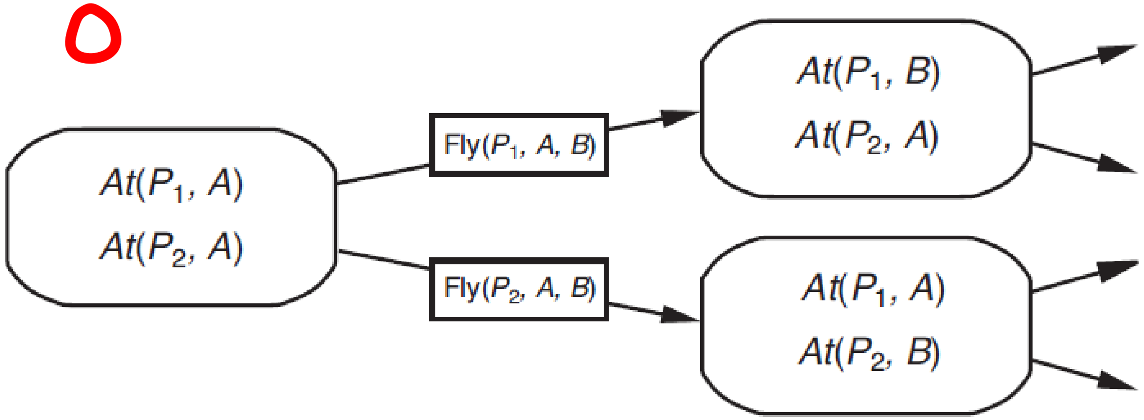
Wang Wenmin

Contents

Two approaches to searching for a plan 搜索计划的两种方式

☐ 1) Forward state-space search 前向状态空间搜索

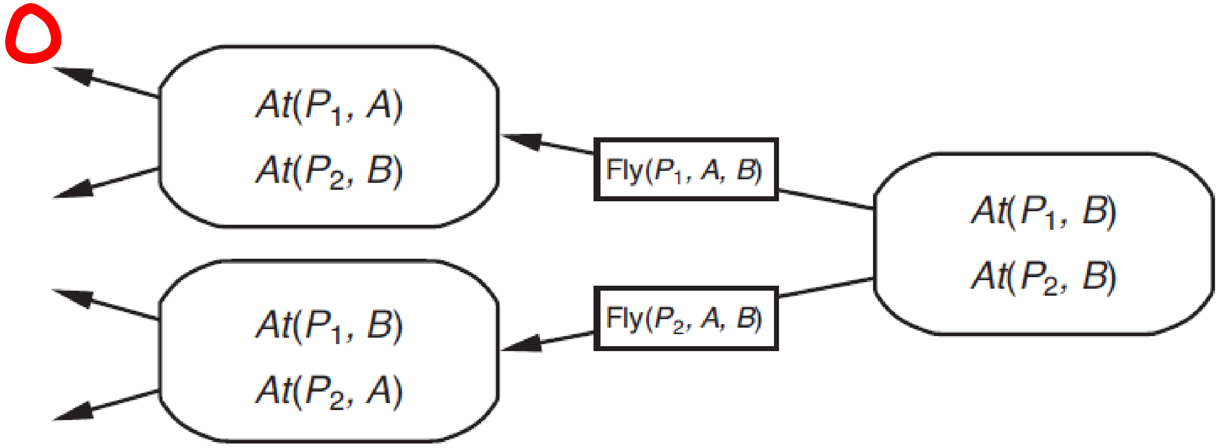
- starting in the initial state,
  从初始状态开始,
- using the problem's actions,
  运用该问题的动作,
- search forward for a member of the goal states.
  朝着一个目标状态向前搜索。



Two approaches to searching for a plan 搜索计划的两种方式

□ 2) Backward relevant-states search 后向状态空间搜索

- starting at the set of states representing the goal,
  从表示该目标的状态集开始,
- using the inverse of the actions,
  运用反向的动作,
- search backward for the initial state.
  朝着初始状态向后搜索。



Heuristics for planning 规划的启发法

- Think of a search problem as a graph 将搜索问题视为一个图
  - where the nodes are states and the edges are actions, to find a path connecting the initial state to a goal state.
    其中节点表示状态、边为动作，寻找一条连接初始状态至某个目标状态的路径。
- Two ways to make this problem easier 该问题简化的两种方式
  - adding edges 增加边
    add more edges to the graph, making it easier to find a path.
    在图上增加更多的边，使之容易找到一条路径。
  - state abstraction 状态抽象
    group multiple nodes together, form an abstraction of the state space that has fewer states, thus is easier to search.
    将多个节点组织在一起，形成具有较少状态的一个状态空间抽象，从而容易搜索。

Two heuristics by adding edges to the graph 图中添加边的两种启发法

1) Ignore-preconditions heuristic 忽略前提启发法

- Drop all preconditions from actions.
  放弃动作中所有的前提条件。
- Every action becomes applicable in every state, and any single goal fluent can be achieved in one step.
  每个动作变成可作用于每个状态，并且任一目标变数可以用一个步骤实现。

Example: 8-puzzle as a planning problem 8数码难题作为规划问题

$$\begin{aligned}
&\mathit{Action}(\mathit{Slide}(t, s_1, s_2),\\
&\quad \text{PRECOND: } On(t, s_1) \land Tile(t) \cancel{\land Blank(s_2) \land Adjacent(s_1, s_2)})\\
&\quad \text{EFFECT: } On(t, s_2) \land Blank(s_1) \land \neg On(t, s_1) \land \neg Blank(s_2))
\end{aligned}$$

Removing the two preconditions, any tile can move in one action to any space, and get the number-of-misplaced-tiles heuristic.

去掉两个前提条件后，任何棋子可以用一个动作移动到任意空间，从而得到错放棋子个数的启发法。

Two heuristics by adding edges to the graph 图中添加边的两种启发法

- 2) Ignore-delete-lists heuristic 忽略删除表启发法
  - Remove the delete lists from all actions,
    从所有动作中移除删除表，
    i.e., removing all negative literals from effects.
    即，从作用中删除所有的否定文字。
  - That makes it possible to make monotonic progress towards goal:
    这样就使其可以朝向目标单调进展：
    no action will ever undo progress made by another action.
    任何动作都不会取消另一个动作的进展。

What is a planning graph 什么是规划图

- A directed graph organized into *levels*: 组成层次的有向图：
  - first, a level S_0 for initial state, consisting of nodes representing each fluent;
    首先，初始状态的层次 S_0，包含 表示每个变数的节点；
  - then, a level A_0 consisting of nodes for each action may be applicable in S_0;
    然后，层次 A_0，包含可能适用于 S_0 的每个动作的节点；
  - then, alternating levels S_i followed by A_i;
    然后，交替进入层次 S_i，接着是 A_i；
  - until we reach a termination condition.
    直到到达一个结束条件。
- Work only for propositional planning problems 仅适用于命题规划问题
  - ones with no variables.
    无变量项。

Example 1: Have cake and eat cake too 有蛋糕和吃蛋糕

Init(*Have*(*Cake*))

Goal(*Have*(*Cake*) $\wedge$ *Eaten*(*Cake*))

Action(*Eat*(*Cake*)

PRECOND: *Have*(*Cake*)

EFFECT: ¬ *Have*(*Cake*) $\wedge$ *Eaten*(*Cake*))

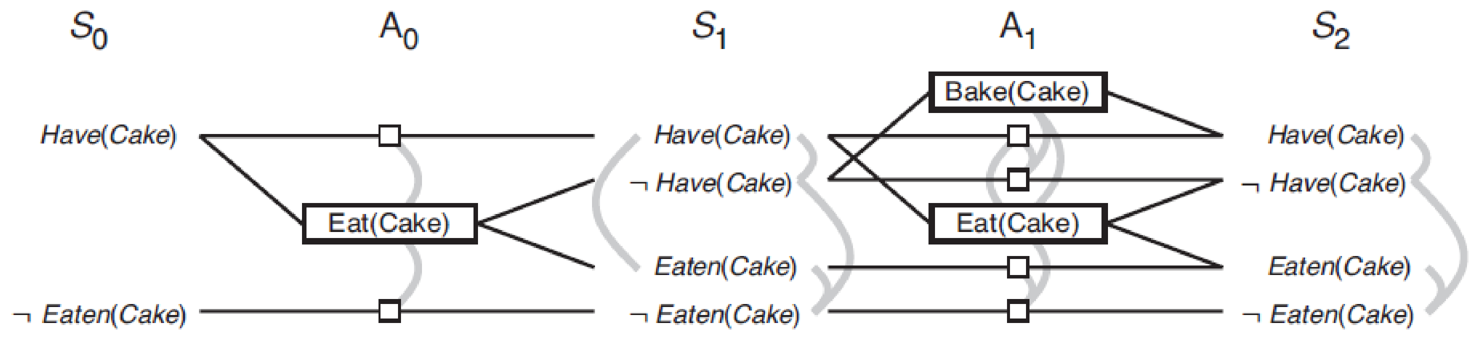
Action(*Bake*(*Cake*)

PRECOND: ¬ *Have*(*Cake*)

EFFECT: *Have*(*Cake*))

The "have cake and eat cake too" problem.

"有蛋糕和吃蛋糕"问题



The "have cake and eat cake too" planning graph.

"有蛋糕和吃蛋糕"规划图

GRAPH-PLAN algorithm GRAPH-PLAN算法

function GRAPH-PLAN(*problem*) **returns** solution or failure
  graph ← INITIAL-PLAN-GRAPH (*problem*)
  goals ← CONJUNCTS(*problem*.GOAL)
  nogoods ←an empty hash table
  for *tl* = 0 **to** ∞ **do**
    if *goals* all non-mutex in S_t of *graph* **then**
      solution ← EXTRACT-SOLUTION(*graph*, *goals*, NUMLEVELS(*graph*), *nogoods*)
      if *solution* ≠ *failure* **then return** *solution*
    if *graph* and *nogoods* have both leveled off **then return** *failure*
    graph ← EXPAND-GRAPH(*graph*, *problem*)

It calls EXPAND-GRAPH to add a level, until either a solution is found by EXTRACT-SOLUTION, or no solution is possible.

调用EXPAND-GRAPH来增加一层，直到通过调用EXTRACT-SOLUTION找到一个解，或者没有可能存在的解。

Example 2: Spare tire problem 备用轮胎问题

$Init(Tire(Flat) \wedge Tire(Spare) \wedge At(Flat, Axle) \wedge At(\mathrm{Spare, Trunk}))$

$Goal(At(Spare, Axle))$

$Action(Remove(obj, loc),$
  PRECOND: $At(obj, loc)$
  EFFECT: $\neg At(obj, loc) \wedge \mathrm{At}(obj\ , Ground))$

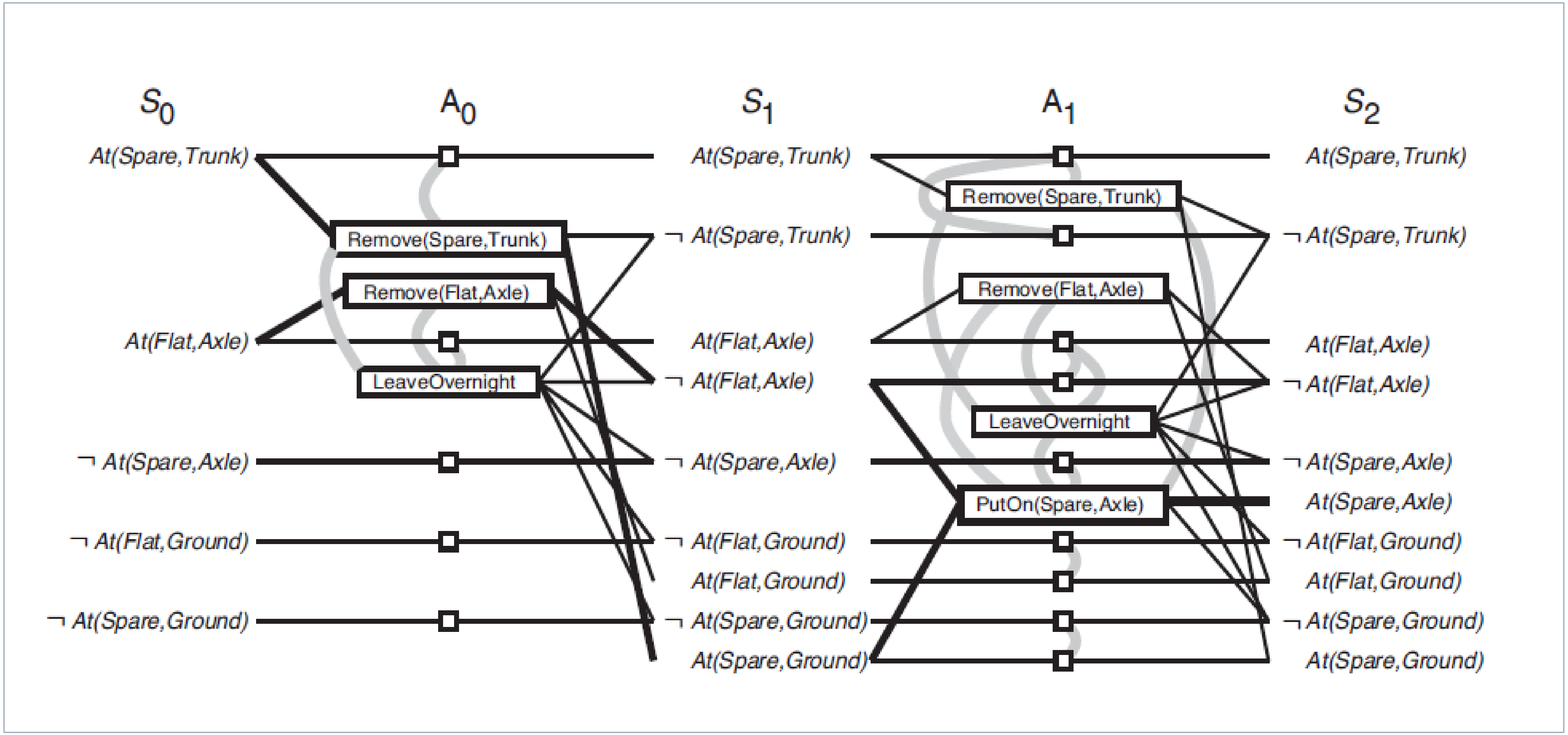
$Action(PutOn(t, Axle),$
  PRECOND: $Tire(t) \wedge At(t, Ground) \wedge \neg At(Flat, Axle)$
  EFFECT: $\neg At(t, Ground) \wedge At(t, Axle))$

$Action(LeaveOvernight,$
  PRECOND:
  EFFECT: $\neg At(Spare, Ground) \wedge \neg At(Spare, Axle) \wedge \neg At(Spare, Trunk) \wedge$
  $\neg At(Flat, Ground) \wedge \neg At(Flat, Axle) \wedge \neg At(Flat, Trunk))$

The initial state has a flat tire on the axle and a good spare tire in the trunk, and the goal is to have the spare tire properly mounted onto the car's axle.

初始状态是车轴上有一个瘪的轮胎并且后备箱里有一个好的备胎，而目标是将这个备胎正确地装在车轴上。

Example 2: Planning graph for spare tire problem 备用轮胎问题的规划图



Other Approaches of Classical Planning 其它经典规划方法

- Four other influential approaches:
  其它四种有影响力的方法：
  - 1) planning as Boolean satisfiability,
    化作布尔可满足性的规划
  - 2) planning as first-order logical deduction,
    化作一阶逻辑推理的规划
  - 3) planning as constraint satisfaction,
    化作约束满足的规划
  - 4) planning as plan refinement.
    化作规划精进的规划

1) Planning as Boolean satisfiability 化作布尔可满足性的规划

- Boolean Satisfiability (SAT) 布尔可满足性 (SAT)

  It is the problem of determining if there exists an interpretation that satisfies a given Boolean formula.

  这是确定是否存在满足给定布尔表达式的解释的问题。

  - Satisfiable formula 可满足表达式

    if the variables of a given Boolean formula can be consistently replaced by the values TRUE or FALSE which make the formula evaluates to TRUE.

    如果给定布尔表达式的变量可一直被TRUE和FALSE值替换，使得表达式的结果为TRUE。

  - Unsatisfiable formula 不可满足表达式

    if no such assignment exists, the function expressed by the formula is identically FALSE for all possible variable assignments.

    如果没有这样的赋值存在，即对所有可能的变量赋值，该布尔表达式的结果始终FALSE。

Example: Planning as Boolean satisfiability 化作布尔可满足性的规划

- **Satisfiable formula** 可满足表达式

  the formula "*a* AND NOT *b*" is satisfiable, because one can find values

  表达式 "*a* AND NOT *b*" 是可满足的，因为人们可以找到值

$$a = \text{TRUE}, \quad \text{and} \quad b = \text{FALSE}$$

  which make "*a* AND NOT *b*" to be TRUE.

  使得表达式 "*a* AND NOT *b*"为TRUE。

- **Unsatisfiable formula** 不可满足表达式

  the formula "*a* AND NOT *a*" is unsatisfiable.

  表达式 "*a* AND NOT *b*" 是不可满足的。

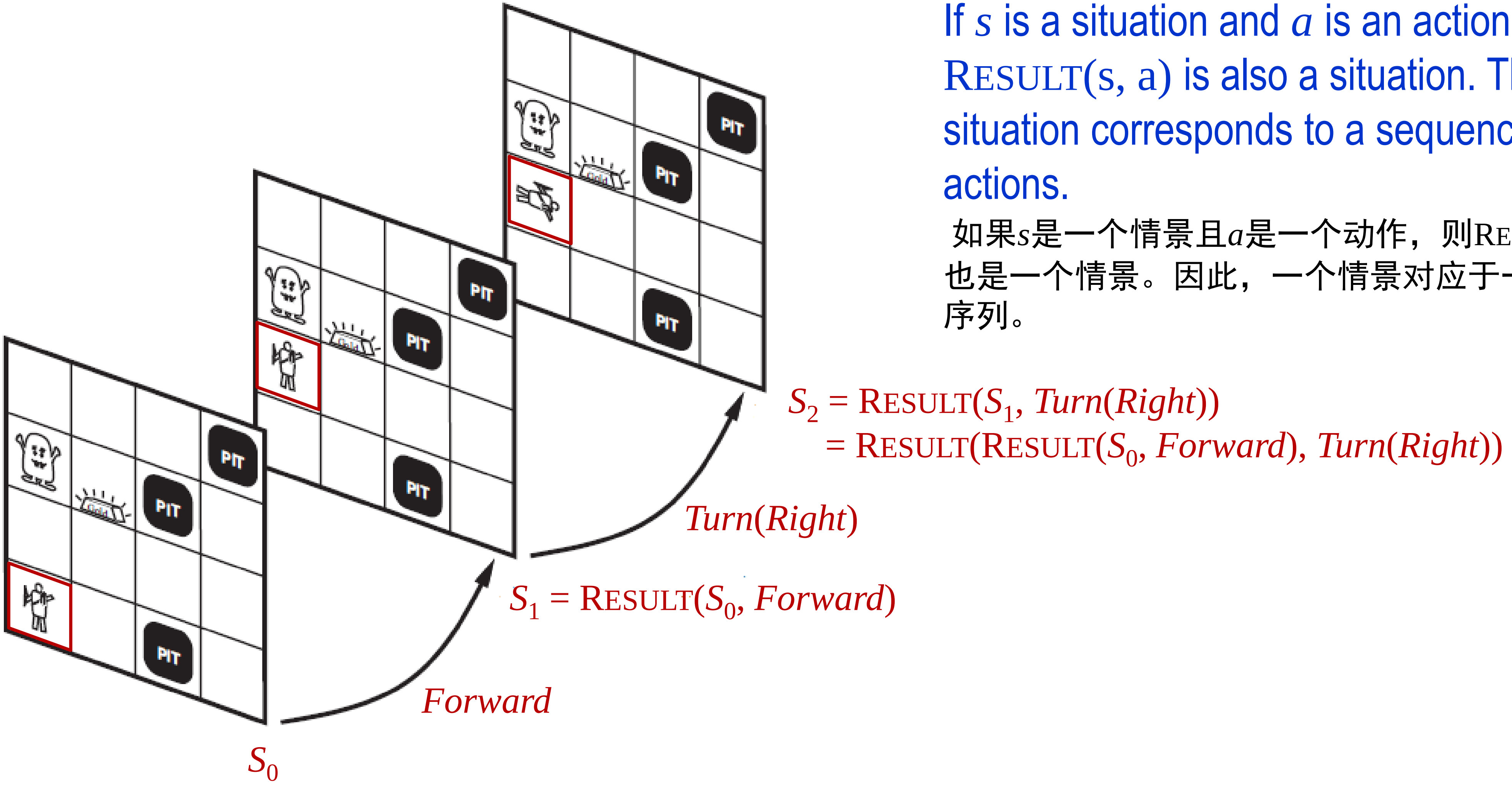
2) Planning as first-order logical deduction 化作一阶逻辑推理的规划

- PDDL is difficult to express some planning problems:
  PDDL难以表达某些规划问题：
  - e.g. can't express the goal: "move all the cargo from A to B regardless of how many pieces of cargo there are".
    例如无法表示如下目标，“把所有的货物从A移到B，不管有多少件货物”。
- Propositional logic also has limitations for some planning problems:
  命题逻辑对某些规划问题也有局限性：
  - e.g. no way to say: "the agent would be facing south at time 2 if it executed a right turn at time 1; otherwise it would be facing east."
    例如无法表达：“智能体若在时间1执行了一个右转则将在时间2时朝南；否则将朝东。
- First-order logic lets us get around those limitations.
  一阶逻辑则让我们摆脱这些局限性。

Situation calculus in first-order logic 一阶逻辑中的情景演算

- It is a logic formalism designed for representing and reasoning about dynamical domains. Its main elements are actions, fluents and situations.
  是设计用于动态域的表示和推理的一种逻辑形式论。其主要元素是动作、变数和情景。
- Situation calculus in first-order logic: 一阶逻辑中的情景演算：
  - Initial state is called a *situation*. A solution is a situation that satisfies the goal.
    初始状态称为一个情景。一个解是满足目标的动作序列。
  - A function or relation that can vary from one situation to the next is a *fluent*.
    可将一个情景转变到下一个的函数或关系是变数。
  - Each *action*'s preconditions are described with a *possibility axiom.*
    每个动作的前提用一个可能性公理来描述。
  - Each fluent is described with a *successor-state axiom.*
    每个变数用一个后记状态公理来描述。
  - Need *unique action axioms* so that the agent can deduce that.
    需要唯一动作公理以便智能体能够对其进行推理。

Situations as actions in Wumpus world 魔兽世界中情景为动作



If s is a situation and a is an action, then RESULT(s, a) is also a situation. Thus, a situation corresponds to a sequence of actions.

如果s是一个情景且a是一个动作，则RESULT(s, a)也是一个情景。因此，一个情景对应于一个动作序列。

3) Planning as constraint satisfaction 化作约束满足的规划

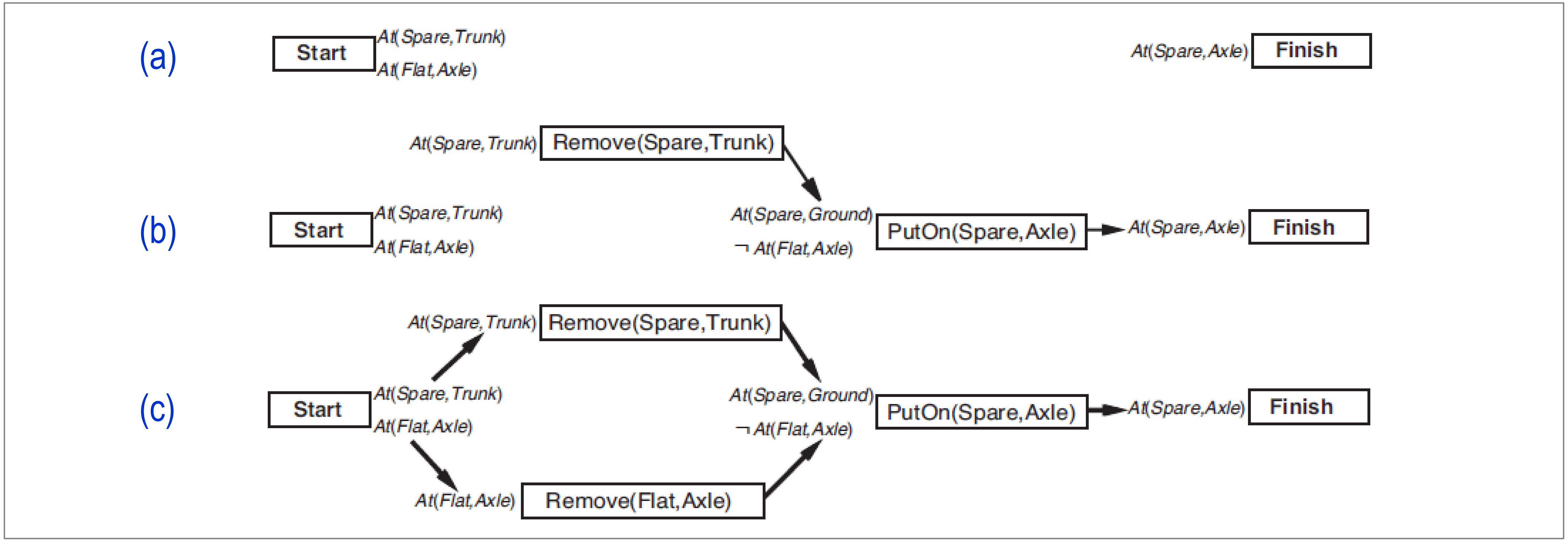
- We have seen 我们已经知道
  - Constraint satisfaction has a lot in common with Boolean satisfiability.
    约束满足与布尔可满足性有许多共性，
  - CSP (constraint satisfaction problem) techniques are effective for scheduling problems.
    CSP（约束满足问题）技术对调度问题很有效。
- So we can 因此我们可以
  - encode a *bounded planning problem* as a CSP, i.e., the problem of finding a plan of length k;
    将有界规划问题进行编码为CSP，例如，寻找一个长度为k的规划的问题；
  - also encode a planning graph into a CSP.
    还可以将规划图编码为CSP。

4) Planning as plan refinement 化作规划精进的规划

- Totally ordered plan 全序规划
  - The totally ordered plan is constructed by all the approaches we have seen so far, consisting of a strictly linear sequence of actions.
    全序规划是由迄今为止我们学到的所有方法所构建的，由严格的线性动作序列组成。
  - This representation ignores the fact that many sub-problems are independent.
    这种表示忽视了许多子问题是独立的这个事实。
- Partially ordered plan 偏序规划
  - An alternative is to represent plans as *partially ordered* structures.
    替代方式是将规划表示为偏序结构。
  - This representation is a set of actions and a set of constraints of the form $Before(a_i, a_j)$, saying that one action occurs before another.
    这种表示是一组动作和一组形式为$Before(a_i, a_j)$的约束，表示一个动作在另一个之前发生。

Example: spare tire problem 备用轮胎问题

Boxes represent actions, arrows indicate orders. 方框表示动作，箭头指出顺序



(a) the tire problem expressed as an empty plan. 将轮胎问题表示为一个空的规划
(b) an incomplete partially ordered plan for the tire problem. 轮胎问题的一个不完全偏序规划
(c) a complete partially-ordered solution. 一个完整的偏序解决方案

Thank you for your attention!

AI

Planning and Scheduling

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

Contents

Planning and Scheduling 规划与调度

- The previous chapter introduced the most basic concepts, representations, and algorithms for planning.
  上一章我们介绍了规划最基本的概念、表示、以及算法。
- The planning and scheduling in the real world are more complex, e.g.,
  现实世界中的规划和调度更为复杂，例如
  - spacecraft, factories, and military campaigns.
    航天器、工厂、以及军事行动。
- They should extend both 它们需要扩展
  - the representation language, and
    表示语言，以及
  - the way the planner interacts with the environment.
    规划者与外部环境交互的方式。

Classical Planning and Its Limitation 经典规划及其局限性

- Classical planning can represent:
  经典规划可以表示：
  - *what to do,*
    做什么
  - in *what order*.
    按什么顺序
- Classical planning cannot represent:
  经典规划无法表示：
  - *how long* an action takes,
    动作持续多长时间
  - *when* it occurs.
    什么时候发生

Plan First and Schedule Later 先规划后调度

- Divide problem into planning phase and scheduling phase.
  将问题分为规划阶段和调度阶段
  - **Planning phase** 规划阶段
    - select actions with some ordering constraints,
      选择具有某种有序约束的动作，
    - to meet the goals of the problem.
      去满足问题的目标。
  - **Scheduling phase** 调度阶段
    - add temporal information to the plan,
      在规划中增加时间信息，
    - to meet resource and deadline constraints.
      去满足资源和期限的约束。

Representing Temporal and Resource Constraints 表征时间和资源约束

- A scheduling problem, consists of a set of jobs, each of which consists a collection of actions with ordering constraints.
  调度问题包含一系列作业，每个作业包含一组具有顺序约束的动作。
- Each action has a duration and a set of resource constraints.
  每个动作有一段持续时间和一组资源约束。
- Each resource constraint specifies: type, number, consumable or reusable.
  每个资源约束指定：类型、数量、可消费或可重用。
- Actions can produce resources, including manufacturing, growing, and resupply.
  动作可以产生资源，包括制造、增产、以及供给动作。
- A solution must specify the start times for each action, and must satisfy all the temporal ordering constraints and resource constraints.
  解决方案需要对每个动作指定起始时间，并且要满足所有的时间顺序约束和资源约束。

Example: A job-shop scheduling 车间作业调度

$A \prec B$ ---- action A must precede B
动作A必须领先于B

$Jobs(\{AddEngine1 \prec AddWheels1 \prec Inspect1\},$
$\{AddEngine2 \prec AddWheels2 \prec Inspect2\})$

$Resources(EngineHoists(1), WheelStations(1), Inspectors(2), LugNuts(500))$

$Action(AddEngine1, \text{DURATION}: 30, \text{USE}: EngineHoists(1))$

$Action(AddEngine2, \text{DURATION}: 60, \text{USE}: EngineHoists(1))$

$Action(AddWheels1, \text{DURATION}: 30,$
$\text{CONSUME}: LugNuts(20), \text{USE}: WheelStations(1))$

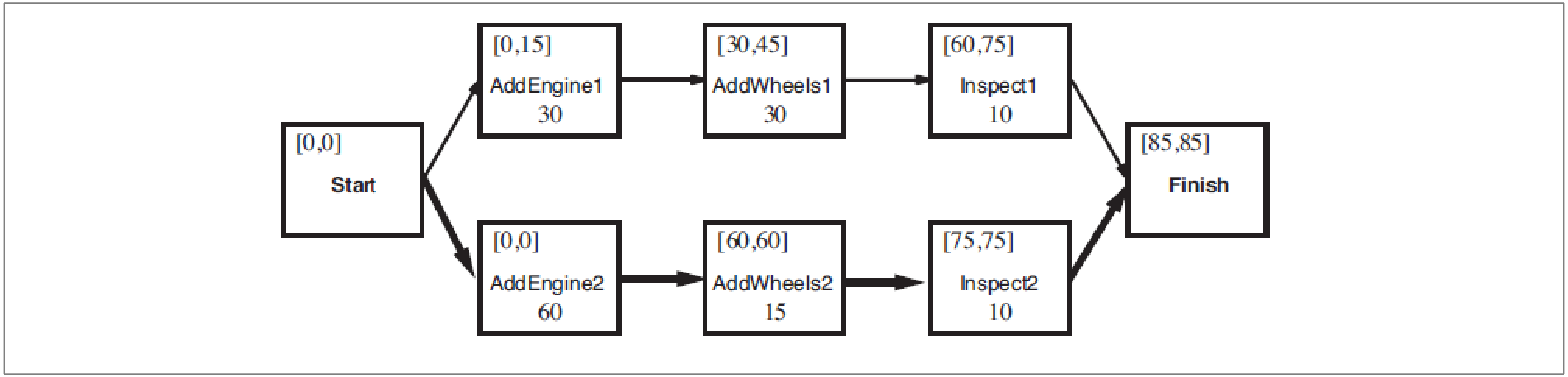
$Action(AddWheels2, \text{DURATION}: 15,$
$\text{CONSUME}: LugNuts(20), \text{USE}: WheelStations(1))$

$Action(Inspect_i, \text{DURATION}: 10, \text{USE}: Inspectors(1))$

A job-shop scheduling for assembling two cars
组装两辆汽车的车间作业调度

Solving Scheduling Problems 求解调度问题

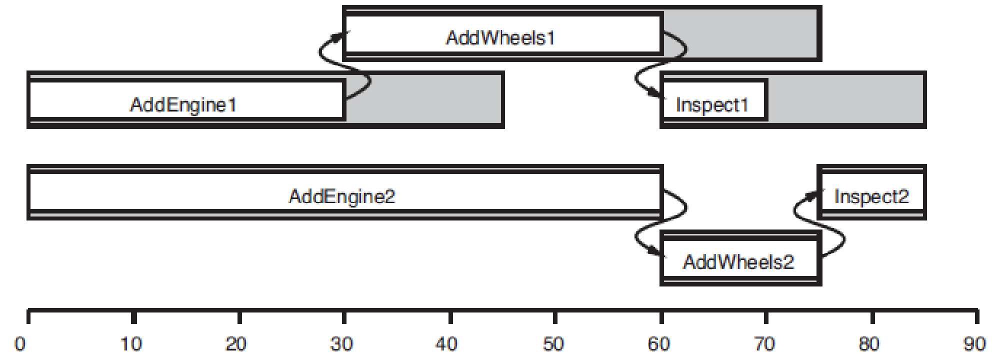
- To minimize plan duration, must find the earliest start times for all the actions consistent with the ordering constraints.
  要使规划持续时间最短，必须找到与排序约束一致的所有动作的最早开始时间。
- To view these ordering constraints as a directed graph.
  将这些排序约束视为一个有向图。



A directed graph of temporal constraints for job-shop scheduling problem
一个车间调度问题的时间约束有向图

Solving Scheduling Problems 求解调度问题

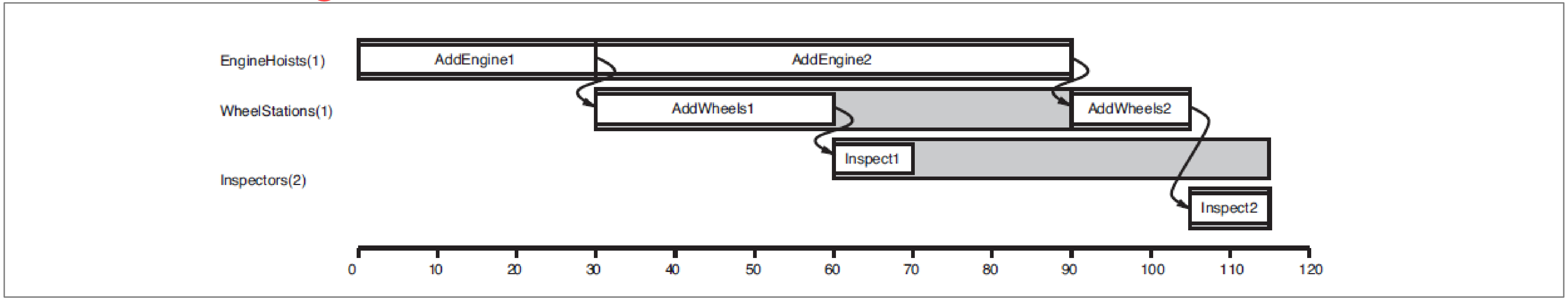
- Can apply the critical path method (CPM) to this graph to determine the possible start and end times of each action.
  可以将关键路径法 (CPM) 用于该图，来确定每个动作可能的开始与结束时间。
- A path through a graph representing a partial-order plan is a linearly ordered sequence of actions.
  一个表示偏序计划的图的路径是一个线性排序的动作序列。



A timeline of temporal constraints for job-shop scheduling problem
一个车间作业调度问题的时间约束的时间表

Solving Scheduling Problems 求解调度问题

- If we introduce resource constraints, the resulting constraints on start and end times become more complicated.
  如果我们引入资源约束，所导致的开始和结束时间的约束变得更加复杂。



A timeline of resource constraints for job-shop scheduling problem
一个车间作业调度问题的资源约束的时间表

- The left-hand margin lists the three reusable resources, and actions are shown aligned horizontally with the resources they use.
  左边列出了三个可重用资源，并且，动作与它们所使用的资源水平对齐显示。

Thank you for your attention!

AI

Real-World Planning

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

Contents

Classical planning vs. Hierarchical planning 经典规划和分层规划

Classical planning 经典规划

- feature: a fixed set of actions.
  特征：一组固定的动作
- problem: a state-of-the-art algorithms can generate solutions containing thousands of actions.
  问题：最新式的算法可以生成包含数千个动作的解。

Hierarchical planning 分层规划

- feature: decompose high-level, abstract tasks into low-level, concrete tasks.
  特征：将高层、抽象的任务分解为低层、具象的任务
- benefit: at each level of the hierarchy, a computational task is reduced to a small number of activities, so the computational cost is small.
  益处：在层次结构的每一级，计算任务被缩减为少量活动，因此计算成本也减少。

Primitive action and High-level action 基本动作和高层动作

- Primitive action 基本动作
  - Means the actions in classical planning, with standard precondition - effect schemas.
    指的是经典规划中的动作，具有经典的前提-效用模式。
  - Has no refinements.
    没有提炼过程。
- High-level action (HLA) 高级动作 (HLA)
  - Key additional concept for hierarchical task networks (HTN) planning.
    层次任务网络 (HTN) 规划中的重要概念。
  - Each HLA has one or more possible refinements, each of which may be an HLA, or a primitive action.
    每个HLA有一个或多个可能的提炼，每个动作可以是一个HLA、或一个基本动作。

Example: Refinement 提炼

- The action is “Go to San Francisco airport”, represented formally as:
  该动作是“去旧金山机场”，形式化表示为：

$$Go(Home, SFO).$$

- May have two possible refinements: 1) drive a car to get to the airport, or 2) take a taxi to get to the airport.
  可以有两种可能的提炼：1）开车去机场，或 2）打车去机场。

Refinement(*Go*(*Home*, *SFO*),
  STEPS: [*Drive*(*Home*, *SFOLongTermParking*), *Shuttle*(*SFOLongTermParking*, *SFO*)])
Refinement(*Go*(*Home*, *SFO*),
  STEPS: [*Taxi*(*Home*, *SFO*)])

What is multi-agent planning 什么是多智能体规划

- So far, we have assumed that only one agent is doing the planning.
  迄今为止，我们假设仅有一个智能体在做计划。
- When there are multiple agents in the environment, each agent faces a multi-agent planning problem in which it tries to achieve its own goals with the help or hindrance of others.
  当环境中有多个智能体时，每个面临多智能体规划问题，试图通过其他智能体的帮助或阻碍达到自己的目标。
- This planning involves coordinating resources and activities of multiple agents.
  这种多智能体规划涉及多个智能体之间协调资源和活动。
- The topic also involves how agents can do this in real time while executing plans (distributed continual planning).
  该主题也涉及到多个智能体在执行计划（分布式连续规划）时如何能够实时动作。

Single-agent vs. Multi-agent problem 单智能体与多智能体问题

Single-agent problem 单智能体问题

- **Multi-effector** 多效用器

  an agent with multiple effectors that can operate concurrently, e.g., a human who can type and speak at the same time.

  一个智能体有多个可以并发运行的效用器。例如，一个人可以同时一边打字一边说话。

- **Multi-body** 多躯体

  effectors are physically decoupled into detached units, but act as a single body, e.g., a fleet of delivery robots in a factory.

  效应器物理分解为独立的单元，但是作为一个躯体动作。例如，工厂里的传送机器人机群。

Multi-agent problem 多智能体问题

- multiple agents coordinate the resources and actions.

  多智能体之间协调资源与动作。

Characteristics of multi-agent 多智能体的特性

- Autonomy: 自主性
  the agents are at least partially independent, self-aware, autonomous.
  这些智能体至少是部分独立、自我意识的、自主的。
- Local views: 局部视野
  no agent has a full global view of the system, or the system is too complex for an agent to make practical use of such knowledge.
  没有智能体对系统具有全局视野，或者系统太复杂，一个智能体无法实际使用这些知识。
- Decentralization: 分散化
  no designated controlling agent, for each agent may need to include communicative actions with other bodies.
  不指定控制智能体，每个智能体可能需要包含与其它躯体进行沟通的动作。
  - e.g., multiple reconnaissance robots. 例如：多机器人侦查。

Issues in Multi-agent Planning 多机器人规划中的问题

- The clearest case of a multi-agent problem is when the agents have different goals.
  多智能体问题最明显的案例是这些智能体具有不同目标时。
- The issues in multi-agent planning can be divided roughly into two sets:
  多智能体规划中的问题可以大致分为两类：
  - 1) involving issues of representing and planning for multiple simultaneous actions.
    多同步动作的表示与规划所涉及的问题。
    - these occur in all settings from multi-effector to multi-agent planning.
      这些问题从多效应器到多智能体规划的所有状况下都会发生。
  - 2) involving issues of cooperation, coordination, and competition arising in true multi-agent settings.
    真正的多智能体环境中所发生的合作、协调和竞争的问题。

1) Planning with multiple simultaneous actions 具有多同步动作的规划

- **Actor** 行动者

  a generic term to cover effectors, bodies, and agents.

  一个涵盖效用器、躯体和智能体的通用术语。

- **Multi-actor** 多行动者

  a generic term to treat multi-effector, multi-body, and multi-agent.

  一个涉猎多效用器、多躯体、以及多智能体的通用术语。

- **Multiple simultaneous actions** 多同步动作

  for multi-actor, to work out how to define:

  对于多行动者，要解决如何定义：

  - transition models, correct plans, and efficient planning algorithms.

    迁移模型、正确的规划、以及有效的规划算法。

Example: Doubles tennis problem 双打网球问题

$Actors(A, B)$

$Init(At(A, LeftBaseline) \land At(B, RightNet) \land$
$\quad Approaching(Ball, RightBaseline)) \land Partner(A, B) \land Partner(B, A)$

$Goal(Returned(Ball) \land (At(a, RightNet) \lor At(a, LeftNet))$

$Action(Hit(actor, Ball),$
$\quad$ PRECOND: $Approaching(Ball, loc) \land At(actor, loc)$
$\quad$ EFFECT: $Returned(Ball))$

$Action(Go(actor, to),$
$\quad$ PRECOND: $At(actor, loc) \land to \neq loc,$
$\quad$ EFFECT: $At(actor, to) \land \neg At(actor, loc))$

- Two actors *A* and *B* are playing together. 两个行动者*A*和*B*一起打球。
- They can be in one of four locations: 他们可以位于四个位置中的一个：
  LeftBaseline, *RightBaseline*, *LeftNet*, and *RightNet*.
- The ball can be returned only if a player is in the right place. 只有当球手位于正确的地方时才可以回球。
- Each action must include the actor as an argument. 每个动作必须包含该行动者作为参数。

2) Planning with multiple agents 具有多智能体的规划

Cooperation and coordination are the feature of multiple agents planning.

合作与协调是多智能体规划的特征。

- **Convention** 协定

  A convention is any constraint on the selection of joint plans.

  It is an option to adopt a convention before engaging in joint activity.

  协定是选择联合计划时的约束。在参与联合行动之前，通过一项协定是一个选项。

- **Communication** 通信

  Agents use it to achieve common knowledge of a feasible joint plan.

  智能体用它来获得可行的联合计划的共同知识。

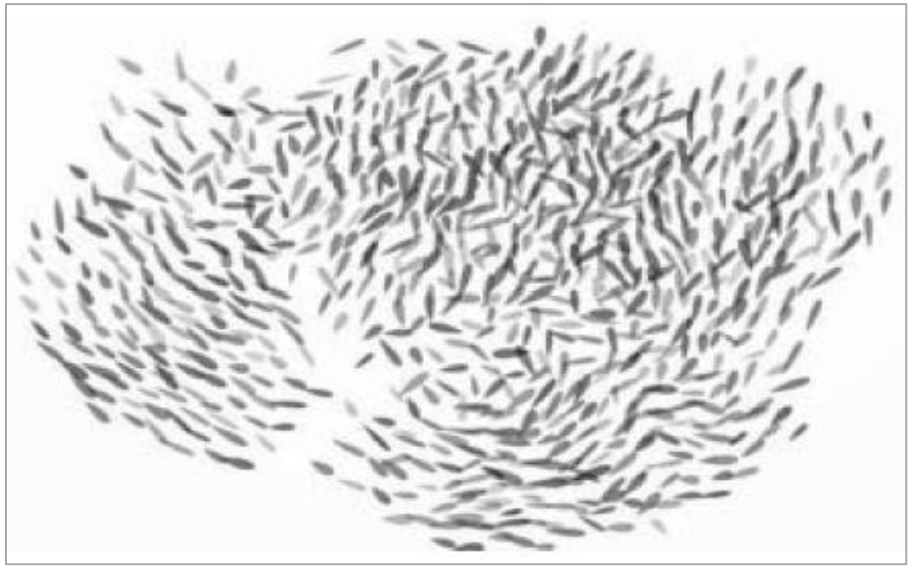
- **Plan recognition** 规划认可

  It is the approach to coordination works to determine a joint plan unambiguously.

  是进行协调工作的方法，用来明确地决定一个联合计划。

Example: Cooperative behavior in flock of birds 鸟群中的合作行为

(a)



(b)

(a) An actual flock of birds. 一个实际的鸟群。

(b) A simulated flock of birds using Reynold's boids model. 用Reynold的boids模型模拟的鸟群。

Particle Swarm Optimization 粒子群优化

Reynold's Boids Model 雷诺的Boids模型

- Boids is an program, developed by Craig Reynolds in 1986.
  Boids是一个程序，由克雷格·雷诺于1986年研发。
- Boids simulates the flocking behavior of birds. The rules in Boids are as follows:
  Boids仿真鸟群的群体行为。Boids中的规则如下：

Rule 规则	Score 成绩	Behavior 行为
Cohesion 聚集	a positive one 正值	getting closer to the average position of the neighbors 接近相邻鸟的平均位置
Separation 分离	a negative one 负值	getting too close to any one neighbor 过于接近任一个相邻的鸟
Alignment 对齐	a positive one 正值	getting closer to the average heading of the neighbors 接近相邻鸟的平均航向

Thank you for your attention!

AI

Decision-theoretic Planning

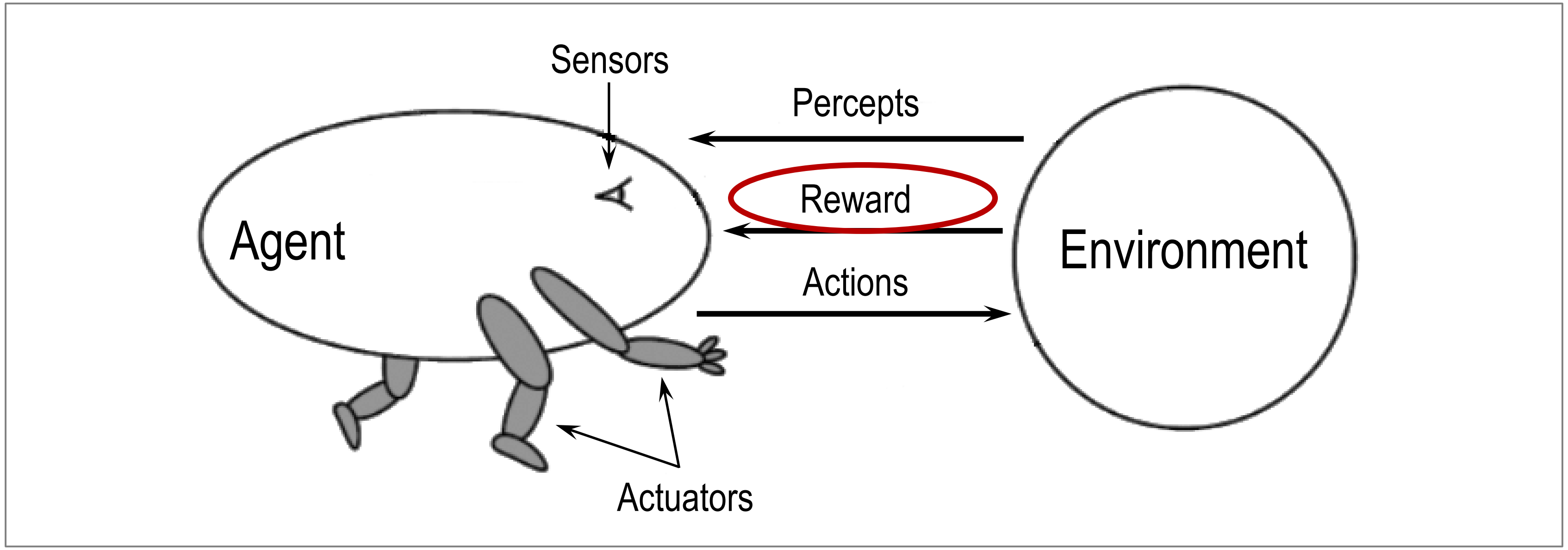
AI

School of Electronic and Computer Engineering
Peking University

Wang Wenmin

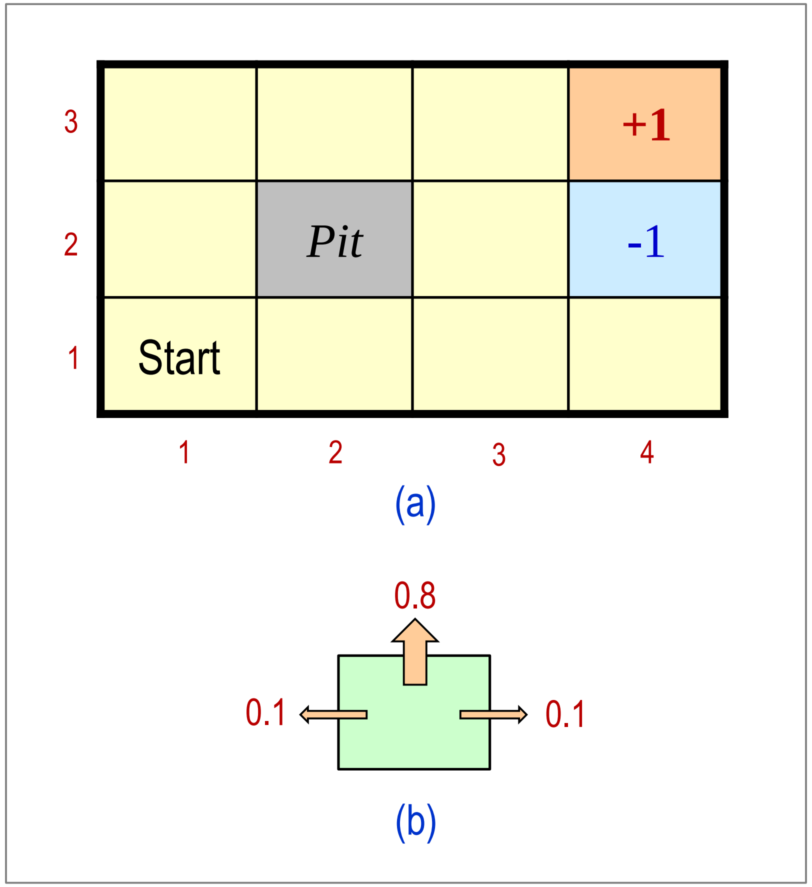
What is Decision-theoretic Planning 什么是决策理论规划

- Classic planning is to find a plan to achieve its goals with lowest cost.
  经典规划是寻找一个以最小代价到达其目标的计划。
- Decision-theoretic Planning is to find a plan to achieve its goals with maximum expected utility (MEU).
  决策理论规划是寻找一个以最大期望效用 (MEU) 到达其目标的规划。



Example: Grid World 方格世界

- Agent lives in a grid, walls block agent's path. Stochastic movement.
  智能体在格子中，围墙挡住了智能体的去路。随机移动。
- *Transition model*: 转换模型
  - probability 0.8: agent moves up;
    概率0.8：智能体上移；
  - probability 0.1: agent moves right or left;
    概率0.1：智能体左移、右移；
  - no movement: if a wall in the direction;
    不移：若前方是堵墙；
  - reward +1 and –1: two terminal states;
    回报+1和−1：两个终点状态；
  - reward –0.04: other no-terminal states.
    回报−0.04：其它非终点状态。
- Goal: maximize sum of rewards. 目标：回报值最大化。



How to Formulize and Solve 如何形式化与求解

- How to formalize the problems of Decision-theoretic Planning?
  如何对决策理论规划问题进行形式化?
  - Markov Decision Process (MDP)
    马尔科夫决策过程 (MDP)
- How to solve the problems of Markov Decision Process?
  如何对马尔科夫决策过程进行求解?
  - Dynamic Programming
    动态规划

Contents

Markov Decision Process (MDP) 马柯夫决策过程 (MDP)

- It is a *discrete time stochastic control process*, means action outcomes depend only on the current state.

  是一种离散时间随机控制过程，意味着动作结果仅仅依赖于当前状态。

- A Markov Decision Process (MDP) is a 5-tuple (S, A, T, R, γ), where

  一个马柯夫决策过程是一个5元组 (S, A, P, R, γ)，其中

  - a set of states, $s \in S$ 一个状态集，$s \in S$
  - a set actions, $a \in A$ 一个动作集，$a \in A$
  - a transition model, $T(s, a, s')$ 一个迁移模型，$T(s, a, s')$

    Probability that a from s leads to s', i.e., $P(s' | s, a)$

    从s导出s'的概率，即：$P(s' | s, a)$
  - a reward function, $R(s, a, s')$ 一个回报函数，$R(s, a, s')$
  - discount, $\gamma \in [0, 1]$ 衰减，$\gamma \in [0, 1]$

Core Problem 核心问题

- The core problem of classical planning: 经典规划的核心问题
  - agent is in a *deterministic environment,*
    智能体是在一个确定性的环境，
  - solving the problem is to find a plan to achieve its goal.
    求解该问题是找到到一个达其目标的计划。
- The core problem of Markov Decision Process (MDP): 马尔科夫决策过程的核心问题
  - agent is in a *discrete time stochastic environment,*
    智能体处于一个离散时间随机环境，
  - solving the problem is to find a policy to control his process.
    求解该问题是找到一个控制其过程的策略。

Finding policy is the core problem to solve MDPs

Core Problem 核心问题

- Given a MDP(S, A, T, R, γ), a policy is a computable function π that outputs for each state s an action a.

  给定一个MDP(S, A, T, R, γ)，一个策略是一个计算函数π，它对每个状态s生成一个动作a.

  - A *deterministic policy* π is defined as: 一个确定性策略被定义为：

$$\pi : S \rightarrow A$$

  - A *stochastic policy* π can also be defined as: 一个随机策略也可以被定义为：

$$\pi : S \times A \rightarrow [0,1]$$

  where $\pi(s, a) \geq 0$ and $\sum_a \pi(s, a) = 1$

- Goal is to choose a policy π that will maximize some cumulative function of the random rewards.

  目标是选择一个策略π，使随机回报值的一些累积函数最大化。

Utilities and Optimal Policies 效用和优化策略

- In sequential decision problems, preferences are expressed between sequences of states.
  在顺序决策问题中，偏好由状态顺序之间的顺序来表示。
- Usually use an additive utility functions:
  通常采用一个累加效用函数：

$$U([s_0, s_1, s_2, \ldots]) = R(s_0) + R(s_1) + R(s_2) + \ldots = \sum_i R(s_i)$$

- Utility of a *state* (a.k.a. its value) is defined to be:
  一个状态（亦称其值）的效用被定义为：

$U(s_i)$ = expected sum of rewards until termination assuming optimal actions.

假设最佳动作结束之前的预期回报值的总和

- Two optimal policies: Value Iteration and Policy Iteration.
  两个优化策略：值迭代和策略迭代。

1) Value Iteration 值迭代

- Basic idea: 基本思想
  - calculate the utility of each state, and then use the state utilities to select an optimal action in each state.
    计算每个状态的效用，然后使用该状态效用在每个状态中选择一个最佳动作。
  - π function is not used; instead the value of π is calculated within $U(s)$.
    不使用π函数；而π值在$U(s)$中计算。
- Bellman equation for utilities:
  贝尔曼效用等式:

$$U(s) = R(s) + \gamma \max_{\alpha \in A(s)} \sum_{s'} P(s' \mid s, a) U(s')$$

Bellman equation is the basis of value iteration algorithm.

1) Value Iteration 值迭代

function VALUE-ITERATION(mdp, ϵ) **returns** a utility function
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$,
    rewards $R(s)$, discount γ
    ϵ, the maximum error allowed in the utility of any state
  local variables: U, U', vectors of utilities for states in S, initially zero
    δ, the maximum change in the utility of any state in an iteration

  repeat
    $U \leftarrow U'; \delta \leftarrow 0$
    for each state s **in** S **do**
      $$\boxed{U'[s] \leftarrow R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\, U[s']}$$
      if $|U'[s] - U[s]| > \delta$ **then** $\delta \leftarrow |U'[s] - U[s]|$
  until $\delta < \epsilon(1-\gamma)/\gamma$
  return U

The value iteration algorithm for calculating utilities of states.

计算状态效用的值迭代算法

2) Policy Iteration 策略迭代

- Basic idea: alternate the two phases.

  基本思想：交替执行如下两个阶段：

  - Policy evaluation: 策略迭代

    given a policy π_i, calculate utility U_i of each state if π_i were to be executed.

    给定一个策略π_i，如果π_i被执行的话，计算每个状态的效用U_i。

$$U_i(s) = R(s) + \gamma \sum_{s'} P(s' \mid s, \pi_i(s))U_i(s')$$

  - Policy improvement: 策略改善

    calculate a new MEU (maximum expected utility) policy π_{i+1}, using one-step look-ahead based on U_i.

    使用基于U_i的提前看一步法，计算一个新的MEU（最大期待效用）策略π_{i+1}。

$$\pi^*(s) = \gamma \operatorname*{argmax}_{\alpha \in A(s)} \sum_{s'} P(s' \mid s, a)U(s')$$

2) Policy Iteration 策略迭代

function POLICY-ITERATION(mdp) **returns** a policy
  inputs: mdp, an MDP with states S, actions $A(s)$, transition model $P(s' \mid s, a)$
  local variables: U, a vector of utilities for states in S, initially zero
    π, a policy vector indexed by state, initially random

  repeat
    $U \leftarrow$ POLICY-EVALUATION(π, U, mdp)
    $unchanged? \leftarrow$ true
    for each state s **in** S **do**
      if $\boxed{\max_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s'] > \sum_{s'} P(s' \mid s, \pi[s])\ U[s']}$ **then do**
        $\boxed{\pi[s] \leftarrow \operatorname{argmax}_{a \in A(s)} \sum_{s'} P(s' \mid s, a)\ U[s']}$
        $unchanged? \leftarrow$ false
  until $unchanged?$
  return π

The policy iteration algorithm for calculating an optimal policy.

计算最佳策略的值迭代算法

Thank you for your attention!

AI