任务规划

Task Planning

理论、方法与应用

编著者

（待填写）

2026 年 1 月 29 日

内部资料，仅供教学使用

前言

任务规划（Task Planning）是人工智能领域的核心研究方向之一，它研究如何让智能系统自主地制定行动序列以实现特定目标。从早期的 STRIPS 系统到现代的大语言模型驱动规划，任务规划理论与技术经历了半个多世纪的发展，已广泛应用于机器人、物流、交通、军事、游戏等众多领域。

本书旨在为本科生和研究生提供一本系统、全面的任务规划教材。全书共分四篇十六章：

- **第一篇：基础篇**（第 1–6 章）：介绍任务规划的基本概念、状态空间搜索、经典规划理论、启发式方法、时态规划和层次任务网络规划，构建扎实的理论基础。
- **第二篇：方法篇**（第 7–10 章）：深入讨论约束满足与规划、不确定性规划、多智能体规划和运动规划等进阶方法。
- **第三篇：应用篇**（第 11–13 章）：通过交通运输规划、通信与编码传输规划、军事任务规划三个应用领域，展示任务规划技术的实际应用。
- **第四篇：前沿篇**（第 14–16 章，标注★）：介绍基于学习的规划方法、大语言模型与任务规划、规划系统的可解释性与安全性等前沿研究方向，适合研究生深入学习。

本书特色：

1. **理论与实践并重**：既有严谨的数学定义和算法分析，又有丰富的应用示例。
2. **示例驱动**：每章包含多个来自交通、通信、军事等领域的详细示例。
3. **分层设计**：基础内容面向本科生，前沿内容（带★标记）面向研究生。
4. **配套资源**：提供 PDDL 参考手册、规划器使用指南和编程实验指导。

编者

2026 年

目录

第一部分

基础篇

第一章　绪论

1.1　任务规划的基本概念

任务规划（Task Planning）是人工智能领域的核心研究方向之一，它研究如何让智能系统自主地制定行动序列以实现特定目标。简单来说，任务规划回答的是” 做什么” 和” 怎么做” 的问题。

定义 1.1 (任务规划问题)**.** 一个任务规划问题可以形式化地定义为一个四元组 $\mathcal{P} = \langle S, A, I, G \rangle$，其中：

- S 是状态空间，表示系统所有可能状态的集合；
- A 是动作集合，表示智能体可执行的所有动作；
- $I \in S$ 是初始状态；
- $G \subseteq S$ 是目标状态集合。

规划的任务是找到一个动作序列 $\pi = \langle a_1, a_2, \ldots, a_n \rangle$，使得从初始状态 I 出发，依次执行这些动作后，系统能够到达某个目标状态 $g \in G$。

任务规划与人类的日常决策密切相关。当我们计划一次旅行、安排一天的工作、或者思考如何完成一个复杂项目时，我们都在进行某种形式的规划。人工智能领域的任务规划研究旨在将这种能力赋予计算机系统。

1.2　任务规划的发展历程

任务规划的研究始于 20 世纪 60 年代末，至今已经历了半个多世纪的发展。

1.2.1　早期探索（1969–1980）

1969 年，斯坦福研究院（Stanford Research Institute）的研究人员开发了 **STRIPS**（Stanford Research Institute Problem Solver）系统，这是第一个具有重要影响力的自动规划系统。STRIPS 引入了用前提条件和效果来描述动作的方法，这一表示方法至今仍是规划领域的基础。

同一时期，**积木世界**（Blocks World）成为规划研究的标准测试问题。在这个问题中，机器人需要通过移动积木来实现特定的堆叠配置。

1.2.2　经典规划时期（1980–1995）

20 世纪 80 年代至 90 年代中期，研究者们发展了多种规划方法：

- **偏序规划**（Partial-Order Planning）：允许规划中的动作保持部分有序，提高了规划的灵活性。
- **层次任务网络规划**（HTN Planning）：通过任务分解的方式，将复杂任务分解为简单子任务。
- **规划图方法**：Blum 和 Furst 于 1995 年提出的 GraphPlan 算法，通过构建规划图来加速规划过程。

1.2.3　现代规划时期（1995–2015）

这一时期的重要进展包括：

- **PDDL 语言**：1998 年，为了统一规划问题的描述，研究者们制定了**规划领域定义语言**（PDDL），并成为国际规划竞赛的标准语言。
- **启发式搜索规划**：FF 规划器和 Fast Downward 系统的出现，使得规划器能够高效处理大规模问题。
- **SAT 规划**：将规划问题转换为布尔可满足性问题，利用 SAT 求解器的强大能力。

1.2.4　智能时代（2015–至今）

近年来，深度学习和大语言模型的发展为任务规划带来了新的机遇：

- **强化学习与规划**：AlphaGo 等系统展示了学习与规划结合的强大能力。
- **神经符号规划**：将神经网络的学习能力与符号规划的推理能力相结合。
- **大语言模型规划**：探索利用 GPT、Claude 等大语言模型进行任务规划。

1.3　任务规划的应用领域

任务规划技术已广泛应用于多个领域，本节通过三个典型示例来说明。

示例 1.1 (交通物流调度问题)**.** 某快递公司需要为 5 辆配送车规划当日的配送路线。已知：

- 配送中心位于城市中心，坐标为 $(0, 0)$；
- 共有 50 个配送点，每个点有一定的货物需求量；
- 每辆车的载重限制为 2 吨，行驶里程限制为 200 公里；
- 部分配送点有时间窗要求（如” 上午 9 点前送达”）。

这是一个典型的**带时间窗的车辆路径问题**（VRPTW）。规划系统需要确定：

1. 每辆车访问哪些配送点（任务分配）；
2. 每辆车的访问顺序（路径规划）；
3. 何时到达每个配送点（调度）。

目标是在满足所有约束的前提下，最小化总行驶距离或总配送时间。

示例 1.2 (机器人任务执行)**.** 考虑一个家庭服务机器人，用户发出指令：” 把客厅桌上的杯子放到厨房水槽里。”

机器人需要规划以下动作序列：

1. 导航到客厅；
2. 定位桌子和杯子；
3. 移动到桌子前的合适位置；
4. 伸出机械臂抓取杯子；
5. 导航到厨房（同时保持杯子稳定）；
6. 定位水槽；
7. 将杯子放入水槽。

这涉及到**任务与运动规划**（TAMP）的结合：高层的任务规划决定” 做什么”，底层的运动规划决定” 怎么移动”。

示例 1.3 (军事作战任务分配)**.** 在一次无人机侦察任务中，指挥中心需要为 10 架无人机分配侦察任务。已知：

- 有 20 个需要侦察的目标区域；
- 每架无人机的续航时间为 4 小时；

- 不同目标区域的优先级不同；
- 存在禁飞区和威胁区域；
- 部分目标需要多架无人机协同侦察。

规划系统需要考虑：

1. 目标分配：哪些无人机负责哪些目标；
2. 航迹规划：每架无人机的飞行路线；
3. 时间协调：多机协同侦察的时间安排；
4. 应急处理：遇到突发情况时的重规划。

这是**多智能体任务规划**与**路径规划**的综合问题。

1.4　任务规划与相关学科的关系

任务规划是一个跨学科的研究领域，与多个学科密切相关：

- **人工智能**：任务规划是 AI 的核心研究方向之一，与知识表示、推理、搜索等紧密相关。
- **运筹学**：许多规划问题可以建模为优化问题，运筹学提供了丰富的求解方法。
- **控制理论**：在机器人规划中，控制理论为运动执行提供了理论基础。
- **计算机科学**：算法设计、复杂性理论为规划提供了分析工具。
- **认知科学**：人类规划行为的研究为人工规划系统提供了启发。

图1.1展示了任务规划与相关学科的关系。

1.5　本书结构与学习指南

本书共分四篇十六章，内容安排如下：

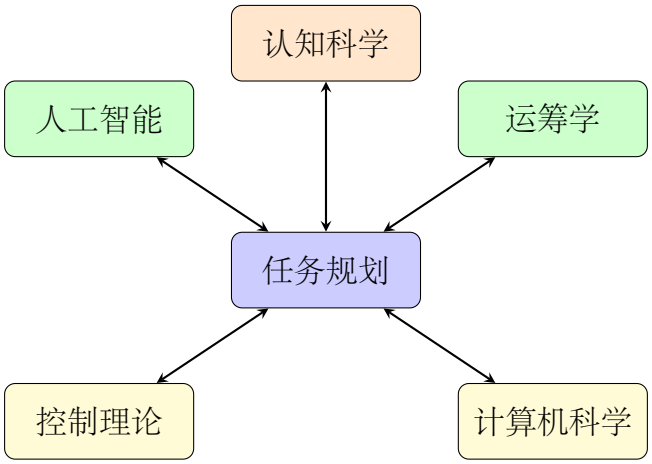


图 1.1: 任务规划与相关学科的关系

1.5.1 内容结构

第一篇：基础篇（第 1–6 章） 建立任务规划的理论基础，包括状态空间搜索、经典规划理论、启发式方法、时态规划和层次任务网络规划。这部分内容是后续学习的基础，建议按顺序学习。

第二篇：方法篇（第 7–10 章） 介绍进阶的规划方法，包括约束满足、不确定性规划、多智能体规划和运动规划。这些章节相对独立，可以根据兴趣选择性学习。

第三篇：应用篇（第 11–13 章） 通过三个应用领域展示任务规划技术的实际应用。建议在学完基础篇后，选择感兴趣的应用领域深入学习。

第四篇：前沿篇（第 14–16 章） 介绍最新的研究进展，适合研究生深入学习。这些章节标有★符号。

1.5.2 学习建议

1. **理论与实践结合**：在学习理论知识的同时，动手完成附录 D 中的编程实验。
2. **循序渐进**：先掌握基础篇的内容，再学习进阶内容。
3. **关注示例**：每章的示例都经过精心设计，建议仔细研读。
4. **完成习题**：每章末尾的习题有助于检验学习效果。
5. **查阅参考文献**：对感兴趣的主题，可以通过参考文献进一步深入学习。

本章小结

本章介绍了任务规划的基本概念、发展历程、应用领域以及与相关学科的关系。任务规划研究如何让智能系统自主制定行动序列以实现目标，是人工智能领域的核心研究方向。从 1969 年的 STRIPS 系统到现代的大语言模型规划，任务规划技术不断发展，应用领域日益广泛。

习题

1. 试用自己的语言解释什么是任务规划，并举出日常生活中的两个任务规划例子。
2. 对于示例1.1中的快递配送问题，讨论以下问题：
   (a) 如果取消时间窗约束，问题会变得更简单还是更复杂？为什么？
   (b) 如果允许配送点之间的货物中转，会对问题产生什么影响？
3. 查阅资料，了解 STRIPS 系统的历史，回答：
   (a) STRIPS 的名称是什么的缩写？
   (b) STRIPS 最初用于什么应用场景？
   (c) STRIPS 表示方法的核心思想是什么？
4. 思考题：你认为任务规划与机器学习有什么区别和联系？
5. 编程练习：用你熟悉的编程语言，实现一个简单的”积木世界”问题的状态表示，包括：
   (a) 定义状态的数据结构；
   (b) 实现检查某个积木是否在另一个积木上方的函数；
   (c) 实现打印当前状态的函数。

第二章　状态空间与搜索基础

状态空间搜索是任务规划的理论基础。本章将介绍状态空间的形式化表示方法，以及各种搜索算法。

2.1　状态空间表示

2.1.1　状态的形式化定义

定义 2.1 (状态)**.** 状态（State）是对系统在某一时刻所处情况的完整描述。在规划问题中，状态通常用一组**状态变量**（State Variables）或**命题**（Propositions）来表示。

以积木世界为例，假设有三个积木 A、B、C 和一个桌面 Table。我们可以用以下谓词来描述状态：

- $\mathrm{On}(x, y)$：积木 x 在 y 上面
- $\mathrm{OnTable}(x)$：积木 x 在桌面上
- $\mathrm{Clear}(x)$：积木 x 上面没有其他积木
- $\mathrm{Holding}(x)$：机械手正在抓取积木 x
- ArmEmpty：机械手是空的

一个具体的状态可以表示为这些谓词的集合：

$$s_0 = \{\mathrm{OnTable}(A), \mathrm{On}(B, A), \mathrm{Clear}(B), \mathrm{OnTable}(C), \mathrm{Clear}(C), \mathrm{ArmEmpty}\} \tag{2.1}$$

2.1.2　动作与状态转移

定义 2.2 (动作)**.** 动作（Action）是智能体可以执行的操作，它将系统从一个状态转移到另一个状态。一个动作 a 可以用三元组 $\langle \mathrm{Pre}(a), \mathrm{Add}(a), \mathrm{Del}(a) \rangle$ 来描述：

- $\mathrm{Pre}(a)$：前提条件，动作执行前必须满足的条件
- $\mathrm{Add}(a)$：添加列表，动作执行后新增的事实
- $\mathrm{Del}(a)$：删除列表，动作执行后删除的事实

示例 2.1 (积木世界的动作定义)**. 拾取动作** PickUp(x)：从桌面上拾取积木 x

$$\text{Pre} : \{\text{OnTable}(x), \text{Clear}(x), \text{ArmEmpty}\} \tag{2.2}$$

$$\text{Add} : \{\text{Holding}(x)\} \tag{2.3}$$

$$\text{Del} : \{\text{OnTable}(x), \text{ArmEmpty}\} \tag{2.4}$$

放下动作 PutDown(x)：将手中的积木 x 放到桌面上

$$\text{Pre} : \{\text{Holding}(x)\} \tag{2.5}$$

$$\text{Add} : \{\text{OnTable}(x), \text{Clear}(x), \text{ArmEmpty}\} \tag{2.6}$$

$$\text{Del} : \{\text{Holding}(x)\} \tag{2.7}$$

堆叠动作 Stack(x, y)：将手中的积木 x 放到积木 y 上

$$\text{Pre} : \{\text{Holding}(x), \text{Clear}(y)\} \tag{2.8}$$

$$\text{Add} : \{\text{On}(x, y), \text{Clear}(x), \text{ArmEmpty}\} \tag{2.9}$$

$$\text{Del} : \{\text{Holding}(x), \text{Clear}(y)\} \tag{2.10}$$

拆卸动作 Unstack(x, y)：从积木 y 上取下积木 x

$$\text{Pre} : \{\text{On}(x, y), \text{Clear}(x), \text{ArmEmpty}\} \tag{2.11}$$

$$\text{Add} : \{\text{Holding}(x), \text{Clear}(y)\} \tag{2.12}$$

$$\text{Del} : \{\text{On}(x, y), \text{ArmEmpty}\} \tag{2.13}$$

定义 2.3 (状态转移函数)**.** 给定状态 s 和动作 a，如果 $\text{Pre}(a) \subseteq s$，则 a 在 s 中是**可应用的**（Applicable）。状态转移函数 γ 定义为：

$$\gamma(s, a) = (s \setminus \text{Del}(a)) \cup \text{Add}(a) \tag{2.14}$$

2.1.3　目标状态与规划问题

定义 2.4 (规划问题)**.** 一个**经典规划问题**可以定义为三元组 $\mathcal{P} = \langle \mathcal{D}, s_0, g \rangle$，其中：

- $\mathcal{D}$ 是规划领域，包括状态变量和动作定义
- s_0 是初始状态
- g 是目标条件（一组必须满足的命题）

定义 2.5 (解（规划）)**.** 规划问题 $\mathcal{P}$ 的解是一个动作序列 $\pi = \langle a_1, a_2, \ldots, a_n \rangle$，使得：

1. a_1 在 s_0 中可应用
2. 对于 $i = 1, \ldots, n-1$，a_{i+1} 在 $\gamma(\cdots\gamma(\gamma(s_0, a_1), a_2)\cdots, a_i)$ 中可应用
3. $g \subseteq \gamma(\cdots\gamma(\gamma(s_0, a_1), a_2)\cdots, a_n)$

2.2 盲目搜索策略

盲目搜索（Uninformed Search）也称为无信息搜索，是指在搜索过程中不使用问题特定知识的搜索方法。

2.2.1 广度优先搜索

广度优先搜索（Breadth-First Search，BFS）按层次顺序扩展节点，先扩展所有深度为 d 的节点，再扩展深度为 $d+1$ 的节点。

输入: 规划问题 $\mathcal{P} = \langle \mathcal{D}, s_0, g \rangle$
输出: 解（动作序列）或失败

frontier $\leftarrow$ 队列，初始包含 $(s_0, \langle\rangle)$;
explored $\leftarrow \emptyset$;
while *frontier* 非空 **do**
  $(s, \pi) \leftarrow$ frontier.dequeue();
  if $g \subseteq s$ **then**
    返回 π
  end
  explored $\leftarrow$ explored $\cup \{s\}$;
  foreach 动作 a 在 s 中可应用 **do**
    $s' \leftarrow \gamma(s, a)$;
    if $s' \notin$ *explored* 且 $s' \notin$ *frontier* **then**
      frontier.enqueue($(s', \pi \cdot \langle a \rangle)$);
    end
  end
end
返回 失败

算法 1: 广度优先搜索

性质分析:

- **完备性**：如果解存在，BFS 一定能找到
- **最优性**：BFS 找到的解是步数最少的（假设所有动作代价相同）
- **时间复杂度**：$O(b^d)$，其中 b 是分支因子，d 是解的深度
- **空间复杂度**：$O(b^d)$

2.2.2　深度优先搜索

深度优先搜索（Depth-First Search，DFS）总是扩展搜索树中最深的节点。

输入: 规划问题 $\mathcal{P} = \langle \mathcal{D}, s_0, g \rangle$

输出: 解（动作序列）或失败

frontier $\leftarrow$ 栈，初始包含 $(s_0, \langle\rangle)$;

explored $\leftarrow \emptyset$;

while *frontier* 非空 **do**

  $(s, \pi) \leftarrow$ frontier.pop();

  if $g \subseteq s$ **then**

    返回 π

  end

  explored $\leftarrow$ explored $\cup \{s\}$;

  foreach 动作 a 在 s 中可应用 **do**

    $s' \leftarrow \gamma(s, a)$;

    if $s' \notin$ *explored* **then**

      frontier.push($(s', \pi \cdot \langle a \rangle)$);

    end

  end

end

返回 失败

算法 2: 深度优先搜索

性质分析:

- **完备性**：在有限状态空间中完备（需要避免重复状态）
- **最优性**：不保证最优
- **时间复杂度**：$O(b^m)$，其中 m 是状态空间的最大深度
- **空间复杂度**：$O(bm)$

2.2.3　迭代加深搜索

迭代加深搜索（Iterative Deepening Search，IDS）结合了 BFS 的完备性和 DFS 的空间效率。

输入: 规划问题 $\mathcal{P}$
输出: 解或失败
for $depth \leftarrow 0$ **到** ∞ **do**
  result $\leftarrow$ DepthLimitedSearch($\mathcal{P}$, depth);
  if *result* $\neq$ *cutoff* **then**
    返回 *result*
  end
end

算法 3: 迭代加深搜索

示例 2.2 (八数码问题求解)**.** 八数码问题（8-Puzzle）是一个经典的搜索问题。在 3×3 的棋盘上有 8 个编号为 1–8 的滑块和一个空格，目标是通过移动滑块将棋盘从初始状态变换到目标状态。

初始状态:

2	8	3
1	6	4
7		5

目标状态:

1	2	3
8		4
7	6	5

状态表示：用 9 元组 $(p_1, p_2, \ldots, p_9)$ 表示，其中 p_i 是位置 i 上的滑块编号（0 表示空格）。

动作：上、下、左、右（移动空格）。

搜索结果比较:

算法	扩展节点数	解的长度
BFS	约 46,000	23
DFS	可能很大	不确定
IDS	约 47,000	23

2.3 启发式搜索

启发式搜索（Heuristic Search）利用问题特定的知识来引导搜索方向，从而提高搜索效率。

2.3.1　启发函数设计

定义 2.6 (启发函数)**.** 启发函数 $h : S \to \mathbb{R}_{\geq 0}$ 估计从状态 s 到达目标状态的代价。如果 $h(s) = 0$ 当且仅当 s 满足目标条件，则 h 是**目标感知的**（Goal-Aware）。

定义 2.7 (可采纳启发式)**.** 如果对于所有状态 s，$h(s) \leq h^*(s)$，其中 $h^*(s)$ 是从 s 到目标的真实最优代价，则 h 是**可采纳的**（Admissible）。

定义 2.8 (一致启发式)**.** 如果对于所有状态 s 和动作 a，$h(s) \leq c(s,a) + h(\gamma(s,a))$，其中 $c(s,a)$ 是执行 a 的代价，则 h 是**一致的**（Consistent）。

2.3.2　A* 算法

A* 算法是最著名的启发式搜索算法，它使用评估函数 $f(s) = g(s) + h(s)$ 来选择下一个扩展的节点，其中 $g(s)$ 是从初始状态到 s 的实际代价。

输入: 规划问题 $\mathcal{P}$，启发函数 h
输出: 最优解或失败
open $\leftarrow$ 优先队列，按 f 值排序，初始包含 $(s_0, \langle\rangle, 0)$;
closed $\leftarrow \emptyset$;
while *open* 非空 **do**
  $(s, \pi, g) \leftarrow$ open.extractMin();
  if $g \subseteq s$ **then**
    返回 π
  end
  if $s \in$ *closed* **then**
    continue;
  end
  closed $\leftarrow$ closed $\cup \{s\}$;
  foreach 动作 a 在 s 中可应用 **do**
    $s' \leftarrow \gamma(s, a)$;
    $g' \leftarrow g + c(s, a)$;
    if $s' \notin$ *closed* **then**
      open.insert($(s', \pi \cdot \langle a \rangle, g')$);
    end
  end
end
返回 失败

算法 4: A* 算法

定理 2.1. 如果启发函数 h 是可采纳的，则 A* 算法返回的解是最优的。

定理 2.2. 如果启发函数 h 是一致的，则 A* 算法在扩展每个节点时，已经找到了到达该节点的最优路径。

2.3.3 IDA* 算法

IDA*（Iterative Deepening A*）是 A* 的迭代加深版本，用深度限制代替优先队列，节省了空间。

示例 2.3 (路径规划中的 A* 应用)**.** 考虑在一个 10×10 的网格地图中，从起点 $(1, 1)$ 到达终点 $(10, 10)$，地图中有障碍物。

状态：机器人当前位置 (x, y)

动作：上、下、左、右移动（代价为 1）或对角线移动（代价为 $\sqrt{2}$）

启发函数：常用的启发函数包括：

- **曼哈顿距离**：$h_1(s) = |x - x_g| + |y - y_g|$（仅考虑四方向移动时可采纳）
- **欧几里得距离**：$h_2(s) = \sqrt{(x - x_g)^2 + (y - y_g)^2}$（总是可采纳）
- **切比雪夫距离**：$h_3(s) = \max(|x - x_g|, |y - y_g|)$（考虑对角移动时可采纳）

图2.1展示了使用不同启发函数时 A* 算法的搜索过程。

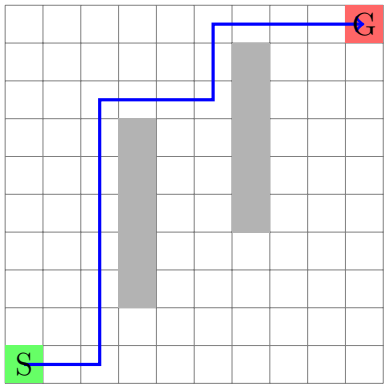


图 2.1: 网格地图中的 A* 路径规划示例

表 2.1: 搜索算法性能比较

算法	完备性	最优性	时间复杂度	空间复杂度
BFS	是	是 *	$O(b^d)$	$O(b^d)$
DFS	否 **	否	$O(b^m)$	$O(bm)$
IDS	是	是 *	$O(b^d)$	$O(bd)$
A*	是	是 ***	$O(b^d)$	$O(b^d)$
IDA*	是	是 ***	$O(b^d)$	$O(bd)$

* 假设所有动作代价相同；** 在无限空间中不完备；*** 要求启发函数可采纳

2.4　搜索算法的性能分析

2.4.1　完备性与最优性

2.4.2　时间复杂度与空间复杂度

规划问题的计算复杂性是一个重要的理论问题。

定理 2.3. 经典规划问题（判定版本）是 PSPACE 完全的。

这意味着规划问题在最坏情况下非常困难。然而，实际问题往往具有特殊结构，可以用启发式方法高效求解。

本章小结

本章介绍了状态空间搜索的基本概念和算法。状态空间由状态、动作和转移函数组成。盲目搜索算法（BFS、DFS、IDS）不使用问题特定知识，而启发式搜索（A*、IDA*）利用启发函数引导搜索。选择合适的搜索算法和启发函数对规划系统的效率至关重要。

习题

1. 对于示例2.2中的八数码问题，设计两种不同的启发函数，并分析其可采纳性。

2. 证明：如果启发函数 h 是一致的，则 h 必定是可采纳的。

3. 实现 BFS 和 A* 算法，求解 15 数码问题，比较两者的性能。

4. 考虑一个机器人在 $n \times n$ 网格中导航的问题，分析以下情况下 A* 算法的时间复杂度：

(a) 使用 $h(s) = 0$（退化为 Dijkstra 算法）

(b) 使用完美启发函数 $h(s) = h^*(s)$

5. 设计一个规划问题，使得 DFS 比 BFS 效率更高，并解释原因。

第三章　经典规划理论

本章介绍经典规划的核心理论，包括 STRIPS 表示语言、PDDL 规划领域定义语言，以及基本的规划算法。

3.1　STRIPS 表示语言

STRIPS（Stanford Research Institute Problem Solver）是 1969 年由 Fikes 和 Nilsson 提出的规划系统，其表示方法至今仍是规划领域的基础。

3.1.1　前提条件与效果

STRIPS 使用**前提条件**（Preconditions）和**效果**（Effects）来描述动作：

定义 3.1 (STRIPS 动作)**.** 一个 STRIPS 动作 a 由三部分组成：

- **前提条件** $\mathrm{Pre}(a)$：动作执行前必须满足的条件集合
- **添加效果** $\mathrm{Add}(a)$：动作执行后变为真的事实集合
- **删除效果** $\mathrm{Del}(a)$：动作执行后变为假的事实集合

3.1.2　动作模式定义

在实际应用中，我们通常定义**动作模式**（Action Schema），即带参数的动作模板。

示例 3.1 (积木世界问题)**.** 积木世界是规划领域的经典测试问题。假设有若干积木和一个机械手，目标是通过移动积木实现特定的堆叠配置。

谓词定义：

- $\mathrm{on}(x, y)$：积木 x 在积木 y 上面
- $\mathrm{ontable}(x)$：积木 x 在桌面上
- $\mathrm{clear}(x)$：积木 x 上面没有其他积木
- $\mathrm{holding}(x)$：机械手正在抓取积木 x
- arm-empty：机械手是空的

动作模式：

动作：pick-up(x) **前提**：ontable(x) $\wedge$ clear(x) $\wedge$ arm-empty **效果**：holding(x) $\wedge \neg$ontable(x) $\wedge \neg$arm-empty

动作：put-down(x) **前提**：holding(x) **效果**：ontable(x) $\wedge$ clear(x) $\wedge$ arm-empty $\wedge \neg$holding(x)

动作：stack(x, y) **前提**：holding(x) $\wedge$ clear(y) **效果**：on(x, y) $\wedge$ clear(x) $\wedge$ arm-empty $\wedge \neg$holding(x) $\wedge \neg$clear(y)

动作：unstack(x, y) **前提**：on(x, y) $\wedge$ clear(x) $\wedge$ arm-empty **效果**：holding(x) $\wedge$ clear(y) $\wedge \neg$on(x, y) $\wedge \neg$arm-empty

3.2 PDDL 规划领域定义语言

PDDL（Planning Domain Definition Language）是 1998 年为国际规划竞赛制定的标准规划语言。

3.2.1 PDDL 语法结构

PDDL 将规划问题分为两个文件：

- **领域文件**（Domain File）：定义谓词和动作模式
- **问题文件**（Problem File）：定义具体实例的初始状态和目标

3.2.2 领域文件与问题文件

示例 3.2 (物流运输领域建模)**.** 考虑一个简化的物流问题：有若干城市、卡车和包裹，目标是将包裹运送到指定目的地。

领域文件（logistics-domain.pddl）：

```
(define (domain logistics)
  (:requirements :strips :typing)

```

```
  (:types
    city location thing - object
    package vehicle - thing
    truck airplane - vehicle
  )

  (:predicates
    (in-city ?loc - location ?city - city)
    (at ?obj - thing ?loc - location)
    (in ?pkg - package ?veh - vehicle)
  )

  (:action load-truck
    :parameters (?pkg - package ?truck - truck ?loc - location)
    :precondition (and
      (at ?truck ?loc)
      (at ?pkg ?loc)
    )
    :effect (and
      (not (at ?pkg ?loc))
      (in ?pkg ?truck)
    )
  )

  (:action unload-truck
    :parameters (?pkg - package ?truck - truck ?loc - location)
    :precondition (and
      (at ?truck ?loc)
      (in ?pkg ?truck)
    )
    :effect (and
      (not (in ?pkg ?truck))
      (at ?pkg ?loc)
    )
  )

  (:action drive-truck
    :parameters (?truck - truck ?from ?to - location ?city - city)
```

```
    :precondition (and
      (at ?truck ?from)
      (in-city ?from ?city)
      (in-city ?to ?city)
    )
    :effect (and
      (not (at ?truck ?from))
      (at ?truck ?to)
    )
  )
)
```

问题文件（logistics-problem.pddl）：

```
(define (problem logistics-1)
  (:domain logistics)

  (:objects
    city1 city2 - city
    loc1-1 loc1-2 loc2-1 loc2-2 - location
    truck1 truck2 - truck
    pkg1 pkg2 - package
  )

  (:init
    (in-city loc1-1 city1) (in-city loc1-2 city1)
    (in-city loc2-1 city2) (in-city loc2-2 city2)
    (at truck1 loc1-1)
    (at truck2 loc2-1)
    (at pkg1 loc1-2)
    (at pkg2 loc2-1)
  )

  (:goal (and
    (at pkg1 loc2-2)
    (at pkg2 loc1-1)
  ))
)
```

3.2.3 PDDL 扩展特性

PDDL 经历了多个版本的演进，引入了许多扩展特性：

表 3.1: PDDL 版本演进

版本	年份	主要特性
PDDL 1.2	1998	STRIPS、类型、量词
PDDL 2.1	2002	数值 fluents、持续性动作
PDDL 2.2	2004	派生谓词、时态初始文字
PDDL 3.0	2006	轨迹约束、偏好
PDDL 3.1	2008	对象 fluents

3.3 前向状态空间规划

前向规划（Forward Planning）也称为**前进规划**（Progression），从初始状态出发，逐步应用动作直到达到目标。

3.3.1 基本算法

前向规划的基本思想是将规划问题转化为图搜索问题，其中：

- 节点是状态
- 边是动作
- 初始节点是 s_0
- 目标节点是满足 g 的状态

3.3.2 可达性分析

定义 3.2 (可达状态)**.** 从初始状态 s_0 出发，通过执行一系列动作可以到达的状态称为**可达状态**。所有可达状态的集合称为**可达状态空间**。

3.4 后向状态空间规划

后向规划（Backward Planning）也称为**回归规划**（Regression），从目标条件出发，逆向推导需要满足的前提条件。

3.4.1 回归搜索

定义 3.3 (回归)**.** 给定目标条件 g 和动作 a，如果 a 可能实现 g 中的某些子目标，则 g 关于 a 的**回归**定义为：

$$\text{regress}(g, a) = (g \setminus \text{Add}(a)) \cup \text{Pre}(a) \tag{3.1}$$

前提是 $\text{Add}(a) \cap g \neq \emptyset$ 且 $\text{Del}(a) \cap g = \emptyset$。

3.4.2 相关性分析

在后向搜索中，我们只考虑与目标**相关**的动作，即那些能够实现至少一个目标子条件的动作。

3.5 规划图与 GraphPlan 算法

GraphPlan 是 1995 年由 Blum 和 Furst 提出的规划算法，它通过构建规划图来加速规划过程。

3.5.1 规划图构造

定义 3.4 (规划图)**.** 规划图是一个有向分层图，包含交替的**命题层**和**动作层**：

- 命题层包含在该时刻可能为真的命题
- 动作层包含在该时刻可能执行的动作
- 层之间通过前提条件和效果连接

3.5.2 互斥关系

定义 3.5 (互斥)**.** 两个动作（或两个命题）是**互斥的**（Mutex），如果它们不能在同一规划步骤中同时出现。动作互斥的条件包括：

1. **不一致效果**：一个动作的效果否定另一个的效果
2. **干扰**：一个动作的效果否定另一个的前提
3. **竞争需求**：两个动作的前提条件互斥

3.5.3　解的提取

在规划图构造完成后，使用后向搜索提取解。

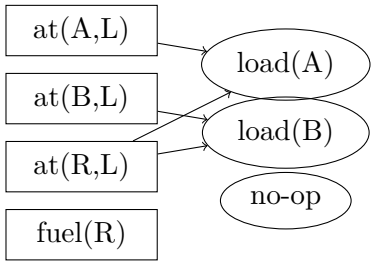
示例 3.3 (火箭运输问题)**.** 考虑一个简化的火箭运输问题：

- 有两个地点：伦敦（London）和巴黎（Paris）
- 有一枚火箭 R1，初始在伦敦
- 有两件货物 A 和 B，初始都在伦敦
- 目标：将 A 和 B 都运送到巴黎

动作：

- load(c, r, l)：在地点 l 将货物 c 装载到火箭 r
- unload(c, r, l)：在地点 l 将货物 c 从火箭 r 卸载
- move$(r, from, to)$：火箭 r 从 $from$ 飞到 to

规划图（部分）：



解：

1. load(A, R1, London)
2. load(B, R1, London)
3. move(R1, London, Paris)
4. unload(A, R1, Paris)
5. unload(B, R1, Paris)

本章小结

本章介绍了经典规划的核心理论。STRIPS 是最早也是最基础的规划表示方法，通过前提条件和效果描述动作。PDDL 是现代规划的标准语言，支持类型、数值、时态等扩展特性。前向规划和后向规划是两种基本的规划策略，GraphPlan 通过构建规划图和分析互斥关系来加速规划过程。

习题

1. 用 STRIPS 格式定义一个”河内塔”（Tower of Hanoi）问题的动作。
2. 编写 PDDL 领域文件，建模一个简单的”机器人取物”问题：机器人需要在房间之间移动并拾取物品。
3. 对于示例3.3中的火箭运输问题，手工构造完整的规划图（直到所有目标第一次同时出现且不互斥）。
4. 分析前向规划和后向规划各自的优缺点，讨论在什么情况下应该选择哪种方法。
5. 实现一个简单的 GraphPlan 算法，求解积木世界问题。

第四章　启发式规划方法

启发式方法是现代规划器成功的关键。本章介绍规划中常用的启发式函数设计方法。

4.1　松弛启发式

松弛（Relaxation）是设计启发式的一种通用方法，通过简化原问题来获得代价的下界估计。

4.1.1　删除松弛

定义 4.1 (删除松弛)**. 删除松弛**（Delete Relaxation）是指忽略所有动作的删除效果。在删除松弛下，一旦某个命题变为真，它将永远保持为真。

4.1.2　h^{add} 与 h^{max} 启发式

定义 4.2 (h^{max} 启发式)**.** h^{max} 启发式估计达到目标集合中" 最难" 达到的子目标的代价：

$$h^{max}(s) = \max_{p \in g} h_p^{max}(s) \tag{4.1}$$

其中 $h_p^{max}(s)$ 是从状态 s 达到命题 p 的估计代价。

定义 4.3 (h^{add} 启发式)**.** h^{add} 启发式假设各子目标独立，将达到各子目标的代价相加：

$$h^{add}(s) = \sum_{p \in g} h_p^{add}(s) \tag{4.2}$$

h^{max} 是可采纳的但信息量较少；h^{add} 信息量丰富但通常高估真实代价。

4.1.3　h^{FF} 启发式

FF 启发式由 Hoffmann 和 Nebel 提出，是最成功的规划启发式之一。它基于松弛规划图，计算从当前状态到目标的松弛计划长度。

输入: 当前状态 s，目标 g
输出: 启发式值 $h^{FF}(s)$
构建松弛规划图直到 g 中所有命题出现;
if *g 中存在命题从未出现* **then**
  返回 ∞
end
从最后一层开始，使用贪心方法提取松弛计划;
返回 *松弛计划中的动作数量*

算法 5: FF 启发式计算

4.2 抽象启发式

抽象（Abstraction）通过将多个状态映射到同一抽象状态来简化问题。

4.2.1 模式数据库

定义 4.4 (模式数据库)**. 模式数据库**（Pattern Database，PDB）预先计算抽象空间中所有状态到目标的精确代价，然后在搜索时作为启发式查表使用。

4.2.2 合并与收缩

合并与收缩（Merge-and-Shrink）是一种系统化构建抽象的方法，通过合并变量和收缩状态空间来控制抽象的大小。

4.3 地标启发式

地标（Landmark）是在任何解中都必须在某个时刻为真的命题或必须执行的动作。

4.3.1 地标识别

识别地标的方法包括:

- 基于松弛规划图的方法
- 基于回归的方法
- 基于 SAT/CSP 的方法

4.3.2　地标计数启发式

定义 4.5 (地标计数启发式)**.** 地标计数启发式 h^{LM} 估计从当前状态到目标还需要达成的地标数量：

$$h^{LM}(s) = |\{l \in L : l \text{ 在} s \text{ 中未被满足且尚未达成}\}| \tag{4.3}$$

4.4　现代规划器架构

4.4.1　Fast Downward 系统

Fast Downward 是最成功的现代规划系统之一，由 Helmert 于 2006 年提出。其主要特点包括：

- 将 PDDL 转换为多值状态变量表示（SAS+）
- 支持多种启发式函数
- 支持多种搜索算法
- 模块化设计，易于扩展

4.4.2　LAMA 规划器

LAMA（Landmarks, Actions, and Multi-heuristics Anytime）规划器结合了地标启发式和 FF 启发式，在国际规划竞赛中表现优异。

示例 4.1 (国际规划竞赛问题求解)**.** 国际规划竞赛（IPC）是规划领域的重要评测平台。以下是 LAMA 在 IPC 2011 物流领域问题上的表现：

问题规模	Fast Downward	LAMA	最优规划器
小（<100 状态）	0.1s	0.1s	0.5s
中（<10000 状态）	2.3s	1.8s	超时
大（>100000 状态）	45s	32s	超时

这说明启发式搜索规划器在大规模问题上具有显著优势。

本章小结

本章介绍了规划中的启发式方法。松弛启发式通过简化问题获得代价估计，其中 h^{FF} 是最成功的启发式之一。抽象启发式通过状态聚合来简化问题。地标启发式利用问题的结构信息。现代规划器如 Fast Downward 和 LAMA 结合了多种技术，在实际问题中表现出色。

习题

1. 对于积木世界问题，计算 h^{add} 和 h^{max} 启发式的值，并比较它们与真实代价的关系。

2. 证明 h^{max} 是可采纳的启发式。

3. 为八数码问题设计一个模式数据库，讨论如何选择模式以获得信息丰富的启发式。

4. 在一个物流问题中，识别可能的地标，并解释为什么它们必须在任何解中出现。

5. 下载并安装 Fast Downward，使用不同的启发式配置求解 IPC 基准问题，比较性能差异。

第五章　时态规划与调度

本章介绍考虑时间约束的规划问题，包括时态规划和调度问题。

5.1　时态逻辑基础

5.1.1　时态算子

时态逻辑引入了描述时间的算子：

- $\Box\phi$（总是）：ϕ 在所有未来时刻都为真
- $\Diamond\phi$（最终）：ϕ 在某个未来时刻为真
- $\bigcirc\phi$（下一刻）：ϕ 在下一时刻为真
- $\phi \mathcal{U} \psi$（直到）：ϕ 保持为真直到 ψ 变为真

5.1.2　时态约束

定义 5.1 (时态约束)**.** 时态约束规定动作之间的时间关系，常见形式包括：

- 先后约束：动作 a 必须在动作 b 之前完成
- 同时约束：动作 a 和 b 必须同时执行
- 时间窗约束：动作必须在指定时间范围内执行
- 持续时间约束：动作执行需要一定的时间

5.2　PDDL 2.1 时态扩展

5.2.1　持续性动作

定义 5.2 (持续性动作)**. 持续性动作**（Durative Action）有明确的开始时刻、结束时刻和持续时间。其定义包括：

- `:duration`：动作的持续时间

- `:condition`：开始条件、结束条件和持续条件
- `:effect`：开始效果和结束效果

```
(:durative-action drive
  :parameters (?truck - truck ?from ?to - location)
  :duration (= ?duration (travel-time ?from ?to))
  :condition (and
    (at start (at ?truck ?from))
    (over all (road ?from ?to))
  )
  :effect (and
    (at start (not (at ?truck ?from)))
    (at end (at ?truck ?to))
  )
)
```

5.2.2 并发执行

时态规划允许多个动作并发执行，但需要满足以下条件：

1. 不存在资源冲突
2. 不违反因果依赖
3. 满足时间约束

5.3 时态规划算法

5.3.1 SAPA 规划器

SAPA 是一个前向搜索时态规划器，使用时态启发式引导搜索。

5.3.2 TFD 规划器

TFD（Temporal Fast Downward）扩展了 Fast Downward 以支持时态规划。

5.4　调度问题

5.4.1　作业车间调度

定义 5.3 (作业车间调度)**. 作业车间调度**（Job Shop Scheduling）问题定义如下：

- 有 n 个作业（jobs）和 m 台机器（machines）
- 每个作业包含若干工序，工序之间有先后约束
- 每个工序需要在特定机器上处理特定时间
- 目标是最小化总完工时间（makespan）

5.4.2　资源约束项目调度

定义 5.4 (RCPSP)**. 资源约束项目调度问题**（Resource-Constrained Project Scheduling Problem，RCPSP）是一类考虑有限资源约束的调度问题。

示例 5.1 (生产线调度优化)**.** 某制造企业有 3 条生产线和 10 个生产订单。每个订单需要经过多道工序，部分工序可以在不同生产线上执行。约束条件包括：

- 每条生产线同一时刻只能处理一个工序
- 同一订单的工序之间有先后关系
- 部分订单有交货期限制

目标是最小化总延迟时间或最大化产能利用率。

示例 5.2 (航班调度问题)**.** 机场需要为当日 100 架次航班分配停机位和登机口。约束条件包括：

- 每个停机位同一时刻只能停放一架飞机
- 登机口与停机位之间需要满足一定的对应关系
- 不同机型对停机位有不同要求
- 需要预留足够的周转时间

目标是最小化乘客步行距离或最大化机场运营效率。

本章小结

本章介绍了时态规划与调度。时态规划扩展了经典规划以处理时间约束和持续性动作。PDDL 2.1 引入了持续性动作和数值 fluent。调度问题是规划的重要应用领域，包括作业车间调度和资源约束项目调度。

习题

1. 用 PDDL 2.1 编写一个简单的时态规划领域，包含至少两个持续性动作。
2. 对于示例5.1中的生产调度问题，画出一个可行的甘特图。
3. 比较规划问题和调度问题的异同点。
4. 分析时态规划相比经典规划的计算复杂性。
5. 设计一个启发式函数，用于评估时态规划中的部分计划质量。

第六章　层次任务网络规划

层次任务网络（Hierarchical Task Network，HTN）规划是一种通过任务分解来求解规划问题的方法。

6.1　HTN 基本概念

6.1.1　任务与方法

定义 6.1 (任务)**.** 在 HTN 规划中，**任务**分为两类：

- **原子任务**（Primitive Task）：可以直接执行的动作
- **复合任务**（Compound Task）：需要进一步分解的抽象任务

定义 6.2 (方法)**. 方法**（Method）定义了如何将复合任务分解为子任务序列。一个方法 m 包含：

- 任务头：要分解的复合任务
- 前提条件：方法适用的条件
- 任务网络：分解后的子任务及其约束

6.1.2　任务分解

HTN 规划的核心思想是递归分解：从初始任务网络开始，不断选择复合任务并应用方法进行分解，直到所有任务都是原子任务。

6.2　SHOP 规划系统

SHOP（Simple Hierarchical Ordered Planner）是一个广泛使用的 HTN 规划系统。

6.2.1 SHOP 算法

输入: 当前状态 s，任务列表 T
输出: 规划 π 或失败
if T 为空 **then**
  返回 空规划 $\langle\rangle$
end
$t \leftarrow T$ 的第一个任务;
if t 是原子任务 **then**
  if t 的前提条件在 s 中满足 **then**
    $s' \leftarrow$ 执行 t 后的状态;
    $\pi \leftarrow \mathrm{SHOP}(s', T \setminus \{t\})$;
    if $\pi \neq$ 失败 **then**
      返回 $\langle t \rangle \cdot \pi$
    end
  end
else
  foreach 方法 m 可以分解 t **do**
    if m 的前提条件在 s 中满足 **then**
      $T' \leftarrow$ 用 m 的子任务替换 T 中的 t;
      $\pi \leftarrow \mathrm{SHOP}(s, T')$;
      if $\pi \neq$ 失败 **then**
        返回 π
      end
    end
  end
end
返回 失败

算法 6: SHOP 算法

6.2.2 SHOP2 扩展

SHOP2 是 SHOP 的扩展版本，支持：

- 偏序任务网络
- 分支和循环
- 数值计算

- 外部函数调用

6.3　HDDL 表示语言

HDDL（Hierarchical Domain Definition Language）是 HTN 规划的标准表示语言。

6.3.1　全序与偏序 HTN

- **全序 HTN**：子任务之间有严格的顺序约束
- **偏序 HTN**：子任务之间只有部分顺序约束

6.3.2　HDDL 语法

```
(:method deliver-package
  :parameters (?p - package ?from ?to - location)
  :task (deliver ?p ?to)
  :precondition (at ?p ?from)
  :ordered-subtasks (and
    (pick-up ?p ?from)
    (transport ?p ?from ?to)
    (put-down ?p ?to)
  )
)
```

6.4　HTN 应用案例

示例 6.1 (军事作战任务分解). 考虑一个空中打击任务的层次分解：

顶层任务：执行空中打击任务

分解方法：

1. 任务规划阶段
   - 情报收集与分析
   - 目标确认
   - 航线规划
2. 任务准备阶段

- 装备检查
- 燃料加注
- 弹药装载

3. 任务执行阶段

- 起飞
- 航渡
- 目标攻击
- 返航

4. 任务评估阶段

- 战果评估
- 任务总结

示例 6.2 (智能家居任务规划)**.** 智能家居系统收到用户指令：” 准备晚餐派对”

任务分解：

1. 环境准备

- 调节室温至 22 度
- 调暗灯光
- 播放背景音乐

2. 餐饮准备

- 预热烤箱
- 准备食材（通知用户）
- 设置定时器

3. 安全检查

- 检查门锁状态
- 启动访客模式

示例 6.3 (游戏 AI 行为规划)**.** 在策略游戏中，AI 需要规划” 建造军事基地”：

方法 1（资源充足时）：

1. 选择建造地点

2. 派遣工人

3. 建造兵营

4. 建造防御塔

5. 训练士兵

方法 2（资源不足时）：

1. 采集资源

2. 递归调用” 建造军事基地”

本章小结

HTN 规划通过任务分解的方式求解规划问题，更符合人类的规划思维方式。SHOP 系列是最著名的 HTN 规划系统，HDDL 是 HTN 规划的标准表示语言。HTN 规划在军事、智能家居、游戏 AI 等领域有广泛应用。

习题

1. 设计一个 HTN 领域，对” 组织一次旅行” 任务进行分解。

2. 比较 HTN 规划和经典规划的优缺点。

3. 用 HDDL 编写示例6.2中的智能家居任务规划领域。

4. 分析 HTN 规划的完备性：在什么条件下 HTN 规划能够找到解？

5. 设计一个 HTN 规划问题，使得全序和偏序分解产生不同的结果。

第二部分

方法篇

第七章　约束满足与规划

本章介绍约束满足问题及其在规划中的应用。

7.1　约束满足问题

7.1.1　CSP 形式化

定义 7.1 (约束满足问题)**. 约束满足问题**（Constraint Satisfaction Problem，CSP）定义为三元组 $\langle X, D, C \rangle$：

- $X = \{x_1, x_2, \ldots, x_n\}$ 是变量集合
- $D = \{D_1, D_2, \ldots, D_n\}$ 是值域集合，D_i 是 x_i 的可能取值
- $C = \{c_1, c_2, \ldots, c_m\}$ 是约束集合

7.1.2　约束传播

约束传播（Constraint Propagation）通过推理减少变量的值域。常用技术包括：

- **节点一致性**：确保每个变量的值域满足一元约束
- **弧一致性**：确保每对变量满足二元约束
- **路径一致性**：确保任意三个变量的组合一致

7.1.3 回溯搜索

输入: CSP 问题 $\langle X, D, C\rangle$，部分赋值 σ
输出: 完整解或失败
if σ 是完整赋值 **then**
  返回 σ
end
选择一个未赋值变量 x;
foreach $v \in D_x$ 按某种顺序 **do**
  if $\sigma \cup \{x = v\}$ 满足所有约束 **then**
    result $\leftarrow$ Backtrack($\sigma \cup \{x = v\}$);
    if *result* $\neq$ 失败 **then**
      返回 *result*
    end
  end
end
返回 失败

算法 7: CSP 回溯搜索

7.2 规划问题的 SAT 编码

7.2.1 命题逻辑基础

命题逻辑的基本概念：

- **命题变量**：取值为真或假的变量
- **文字**：命题变量或其否定
- **子句**：文字的析取
- **CNF**：子句的合取

7.2.2 规划到 SAT 的转换

将规划问题编码为 SAT 的关键思想是引入时间步：

- p_t：命题 p 在时刻 t 为真
- a_t：动作 a 在时刻 t 执行

编码规则：

1. 初始状态：$\bigwedge_{p\in s_0} p_0 \wedge \bigwedge_{p\notin s_0} \neg p_0$
2. 目标状态：$\bigwedge_{p\in g} p_T$
3. 动作前提：$a_t \rightarrow \bigwedge_{p\in \mathrm{Pre}(a)} p_t$
4. 动作效果：$a_t \rightarrow \bigwedge_{p\in \mathrm{Add}(a)} p_{t+1}$
5. 帧公理：$(p_t \wedge \neg p_{t+1}) \rightarrow \bigvee_{a:p\in \mathrm{Del}(a)} a_t$

7.2.3　SAT 求解器应用

示例 7.1 (布尔可满足性规划). 考虑一个简单的积木问题，初始状态 A 在桌上，B 在 A 上；目标是 B 在桌上，A 在 B 上。

SAT 编码（假设最多 2 步）：

命题变量：

- $\mathrm{on}(A,B)_t$, $\mathrm{on}(B,A)_t$：积木位置
- $\mathrm{ontable}(A)_t$, $\mathrm{ontable}(B)_t$：在桌上
- $\mathrm{unstack}(B,A)_t$, $\mathrm{stack}(A,B)_t$, ...：动作

初始状态子句：

$$\mathrm{ontable}(A)_0 \tag{7.1}$$
$$\mathrm{on}(B,A)_0 \tag{7.2}$$
$$\neg\mathrm{on}(A,B)_0 \tag{7.3}$$
$$\neg\mathrm{ontable}(B)_0 \tag{7.4}$$

目标子句：

$$\mathrm{ontable}(B)_2 \tag{7.5}$$
$$\mathrm{on}(A,B)_2 \tag{7.6}$$

SAT 求解器返回满足赋值，从中提取动作序列。

7.3 规划问题的 SMT 编码

7.3.1 SMT 理论

SMT（Satisfiability Modulo Theories）扩展了 SAT，支持更丰富的理论：

- 线性算术
- 数组理论
- 位向量
- 非线性算术

7.3.2 时态规划的 SMT 方法

SMT 特别适合时态规划，因为可以直接处理连续时间变量和数值约束。

本章小结

本章介绍了约束满足与规划的关系。CSP 是一种强大的建模语言，规划问题可以编码为 SAT 或 SMT 问题。现代 SAT/SMT 求解器的强大能力使得这种方法在实践中非常有效。

习题

1. 将八皇后问题建模为 CSP，并用回溯搜索求解。
2. 为示例7.1中的积木问题写出完整的 SAT 编码。
3. 比较 SAT 规划和启发式搜索规划的优缺点。
4. 设计一个需要数值约束的规划问题，说明为什么 SMT 比 SAT 更适合。
5. 实现一个简单的 SAT 规划器，测试其在小规模问题上的性能。

第八章　不确定性规划

本章讨论在不确定环境下的规划问题。

8.1　非确定性规划

8.1.1　条件规划

定义 8.1 (条件规划)**. 条件规划**（Contingent Planning）生成的计划包含条件分支，根据执行过程中观测到的信息选择不同的动作。

条件计划可以表示为树结构：

- 内部节点是观测或条件判断
- 叶节点是动作序列

8.1.2　一致性规划

定义 8.2 (一致性规划)**. 一致性规划**（Conformant Planning）在没有任何观测能力的情况下进行规划。计划必须在所有可能的初始状态下都能达到目标。

8.2　部分可观测规划

8.2.1　信念状态

定义 8.3 (信念状态)**.** 当智能体不能完全观测环境状态时，它维护一个**信念状态**（Belief State），表示对当前可能状态的概率分布或集合。

8.2.2　信念空间搜索

在信念空间中搜索，节点是信念状态，边是动作和观测的组合。

示例 8.1 (传感器受限的机器人导航)**.** 一个机器人在 4×4 网格中导航，但传感器只能检测相邻格子是否有障碍物，不能确定自己的精确位置。

信念状态：可能位置的集合

动作：上、下、左、右移动

观测：相邻四个方向的障碍物情况

机器人需要在不确定自己位置的情况下，规划到达目标区域的策略。通过执行动作和获取观测，逐渐缩小信念状态，最终确定位置并到达目标。

8.3 概率规划

8.3.1 马尔可夫决策过程（MDP）

定义 8.4 (MDP)**. 马尔可夫决策过程**定义为五元组 $\langle S, A, T, R, \gamma \rangle$：

- S：状态空间
- A：动作空间
- $T : S \times A \times S \to [0, 1]$：转移概率函数
- $R : S \times A \to \mathbb{R}$：奖励函数
- $\gamma \in [0, 1)$：折扣因子

8.3.2 部分可观测 MDP（POMDP）

定义 8.5 (POMDP)**. POMDP** 在 MDP 基础上增加：

- Ω：观测空间
- $O : S \times A \times \Omega \to [0, 1]$：观测概率函数

8.3.3 值迭代与策略迭代

输入: MDP $\langle S, A, T, R, \gamma \rangle$
输出: 最优值函数 V^*
初始化 $V(s) = 0$ 对所有 $s \in S$;
repeat
  foreach $s \in S$ **do**
    $V(s) \leftarrow \max_{a \in A} \left[R(s, a) + \gamma \sum_{s' \in S} T(s, a, s') V(s') \right]$;
  end
until 收敛;
返回 V

算法 8: 值迭代算法

示例 8.2 (无人机任务规划中的不确定性处理)**.** 无人机侦察任务中存在多种不确定性：

- **天气不确定性**：风速、能见度可能变化
- **目标不确定性**：目标可能移动或隐藏
- **传感器不确定性**：侦察可能失败
- **威胁不确定性**：敌方防空系统位置不确定

使用 POMDP 建模：

- 状态包括：无人机位置、剩余燃料、目标状态、威胁状态
- 动作包括：飞向某区域、执行侦察、返航
- 观测包括：是否发现目标、是否被探测
- 奖励设计：发现目标获得正奖励，被击落获得大负奖励

本章小结

本章介绍了不确定性规划的主要方法。条件规划和一致性规划处理非确定性。信念状态用于表示部分可观测环境。MDP 和 POMDP 提供了处理概率不确定性的数学框架。

习题

1. 设计一个需要条件规划的问题场景，画出条件计划树。
2. 对于示例8.1，设计一个缩小信念状态的动作序列。
3. 用 MDP 建模一个简单的路径规划问题，其中动作有时会失败。
4. 比较 MDP 和 POMDP 的计算复杂性差异。
5. 实现值迭代算法，求解一个 5×5 网格世界 MDP。

第九章　多智能体规划

本章讨论多个智能体协同完成任务的规划问题。

9.1　多智能体系统基础

9.1.1　智能体架构

定义 9.1 (智能体)**. 智能体**（Agent）是一个能够感知环境并采取行动以实现目标的自主实体。智能体架构包括：

- 感知模块：获取环境信息
- 决策模块：选择行动
- 执行模块：执行选定的行动
- 通信模块：与其他智能体交互

9.1.2　协调与通信

多智能体系统的核心挑战是协调：

- **集中式协调**：存在中央协调器
- **分布式协调**：智能体自主协商
- **隐式协调**：通过环境间接交互

9.2　分布式规划

9.2.1　任务分配

定义 9.2 (任务分配问题)**.** 给定 n 个智能体和 m 个任务，**任务分配**问题是找到一个分配方案，使得总效用最大化或总代价最小化。

常用算法：

- 拍卖算法
- 匈牙利算法
- 合同网协议

9.2.2 计划合并

当每个智能体独立生成计划后，需要合并这些计划以避免冲突。

9.3 多智能体路径规划（MAPF）

9.3.1 MAPF 问题定义

定义 9.3 (MAPF)**. 多智能体路径规划**（Multi-Agent Path Finding）问题：给定 n 个智能体，每个智能体有起点和终点，在图上找到无冲突的路径使所有智能体到达各自目标。

冲突类型：

- **顶点冲突**：两个智能体同时占用同一顶点
- **边冲突**：两个智能体同时使用同一条边（相向移动）

9.3.2 CBS 算法

定义 9.4 (冲突基搜索)**. CBS**（Conflict-Based Search）是一种两层搜索算法：

- 高层：搜索冲突树，识别和解决冲突
- 低层：为每个智能体规划满足约束的路径

输入: MAPF 问题
输出: 无冲突路径集合或失败
为每个智能体计算最短路径;
root ← 创建根节点，包含所有初始路径;
OPEN ← {root};
while *OPEN* 非空 **do**
  N ← OPEN 中代价最小的节点;
  验证 N 中的路径是否有冲突;
  if 无冲突 **then**
    返回 N 中的路径
  end
  选择第一个冲突 (a_i, a_j, v, t);
  foreach 智能体 $a \in \{a_i, a_j\}$ **do**
    创建子节点 N'，添加约束"a 在时刻 t 不能在 v";
    为 a 重新规划满足新约束的路径;
    if 路径存在 **then**
      将 N' 加入 OPEN;
    end
  end
end
返回 失败

算法 9: CBS 算法

9.3.3 优先级规划

优先级规划按某种顺序依次为智能体规划路径，后规划的智能体需要避让先规划的智能体。

示例 9.1 (仓储机器人协同调度)**.** 某自动化仓库有 50 台 AGV（自动引导车）负责货物搬运。系统需要:

- 分配搬运任务给 AGV
- 规划无冲突的移动路径
- 处理实时订单和突发情况

挑战:

- 狭窄通道可能造成死锁

- 需要实时重规划
- 充电站调度

解决方案:

1. 使用拍卖机制分配任务
2. 使用 CBS 或其变体规划路径
3. 实现滚动窗口重规划

示例 9.2 (无人机蜂群编队规划)**.** 20 架无人机需要从分散位置汇聚形成特定编队，然后保持编队飞向目标。

阶段 1：汇聚规划

- 每架无人机分配编队中的目标位置
- 规划无碰撞的汇聚路径
- 同步到达时间

阶段 2：编队飞行

- 领航无人机规划主航线
- 其他无人机保持相对位置
- 处理编队变换指令

阶段 3：障碍规避

- 检测障碍物
- 编队收缩或分裂
- 通过后恢复编队

9.4　博弈论与规划

9.4.1　纳什均衡

当智能体有各自的目标时，规划问题变成博弈。

定义 9.5 (纳什均衡)**. 纳什均衡**是一个策略组合，其中没有智能体能通过单方面改变策略获得更高收益。

9.4.2 对抗规划

在存在对手的情况下进行规划，需要考虑对手的可能反应。

本章小结

本章介绍了多智能体规划的主要问题和方法。任务分配和计划合并是分布式规划的核心问题。MAPF 是多智能体路径规划的标准形式化，CBS 是求解 MAPF 的有效算法。博弈论为存在竞争关系的多智能体规划提供了理论基础。

习题

1. 设计一个 3 智能体的 MAPF 问题实例，手工用 CBS 算法求解。

2. 比较集中式和分布式多智能体规划的优缺点。

3. 对于示例9.1，讨论如何处理死锁情况。

4. 实现一个简单的优先级规划算法，并分析其完备性。

5. 将示例9.2建模为 MAPF 问题，讨论可能的简化假设。

第十章　运动规划基础

本章介绍机器人运动规划的基本概念和方法。

10.1　构型空间

10.1.1　机器人构型表示

定义 10.1 (构型). 机器人的**构型**（Configuration）是完全描述机器人位置和姿态所需的最小参数集合。构型的维度称为**自由度**（DOF）。

例如：

- 平面移动机器人：3 DOF (x, y, θ)
- 六轴机械臂：6 DOF $(\theta_1, \theta_2, \ldots, \theta_6)$
- 人形机器人：通常 > 20 DOF

10.1.2　障碍物映射

定义 10.2 (构型空间障碍物). 工作空间中的障碍物在构型空间中形成 **C 空间障碍物**（C-obstacle），机器人的任何构型如果导致与障碍物碰撞，则属于 C 空间障碍物。

10.2　采样规划方法

10.2.1　PRM（概率路线图）

定义 10.3 (PRM). **概率路线图**（Probabilistic Roadmap）方法分两个阶段：

1. **学习阶段**：随机采样构型，连接近邻形成路线图
2. **查询阶段**：将起点和终点连接到路线图，搜索路径

10.2.2 RRT（快速探索随机树）

定义 10.4 (RRT)**. 快速探索随机树**（Rapidly-exploring Random Tree）通过增量式构建搜索树来探索构型空间。

输入: 起点 q_{init}，终点 q_{goal}
输出: 从起点到终点的路径
$T \leftarrow$ 初始化树，包含 q_{init};
for $k = 1$ **到** K **do**
  $q_{\text{rand}} \leftarrow$ 随机采样（偶尔采样 q_{goal}）;
  $q_{\text{near}} \leftarrow$ 树中距离 q_{rand} 最近的节点;
  $q_{\text{new}} \leftarrow$ 从 q_{near} 向 q_{rand} 方向扩展一步;
  if 路径 (q_{near}, q_{new}) 无碰撞 **then**
    将 q_{new} 加入树;
    if q_{new} 接近 q_{goal} **then**
      返回 构造路径
    end
  end
end
返回 失败

算法 10: 基本 RRT 算法

10.2.3 RRT* 与渐进最优性

定义 10.5 (RRT*)**. RRT*** 是 RRT 的改进版本，通过重新布线（rewiring）操作保证渐进最优性。

示例 10.1 (机械臂运动规划)**.** 一个 6 自由度工业机械臂需要将工件从 A 点移动到 B 点，工作空间中有障碍物。

构型空间：$\mathbb{R}^6$，每个维度对应一个关节角度

约束:

- 关节角度限制
- 避免碰撞（与障碍物、与自身）
- 奇异点避免

使用 RRT 规划:

1. 在 6 维构型空间中随机采样

2. 使用正向运动学计算末端位置

3. 使用碰撞检测验证路径

4. 路径平滑处理

10.3　任务与运动规划集成（TAMP）

10.3.1　符号-几何接口

TAMP（Task and Motion Planning）的挑战在于连接符号层面的任务规划和几何层面的运动规划。

关键问题：

- 符号动作的几何可行性检验
- 几何约束的符号化表示
- 规划层次之间的信息传递

10.3.2　PDDLStream

定义 10.6 (PDDLStream)**. PDDLStream** 扩展了 PDDL，引入了流（Stream）来生成连续参数（如位姿、轨迹）。

示例 10.2 (服务机器人抓取规划)**.** 服务机器人需要从桌上抓取一个杯子并放入橱柜。

任务层面（符号规划）：

1. 移动到桌子附近

2. 抓取杯子

3. 移动到橱柜附近

4. 打开橱柜门

5. 放入杯子

6. 关闭橱柜门

运动层面（几何规划）：

- 导航路径规划

- 抓取位姿采样
- 机械臂运动规划
- 门把手操作轨迹

集成挑战:

- 某些抓取位姿可能因碰撞不可行
- 放置位置影响后续操作
- 需要同时考虑导航和操作

本章小结

本章介绍了运动规划的基本概念和方法。构型空间是运动规划的数学基础。PRM 和 RRT 是两类主要的采样规划方法。TAMP 集成了任务规划和运动规划，是机器人规划的前沿方向。

习题

1. 对于一个在 10×10 网格中的移动机器人，比较 A* 和 RRT 的性能。
2. 证明 RRT* 具有渐进最优性。
3. 为示例10.1设计碰撞检测算法。
4. 讨论 TAMP 中符号规划和运动规划如何交互。
5. 实现基本的 RRT 算法，在 2D 空间中规划路径。

第三部分

应用篇

第十一章　交通运输规划

本章介绍任务规划在交通运输领域的应用。

11.1　车辆路径问题（VRP）

11.1.1　基本 VRP 模型

定义 11.1 (车辆路径问题)**. 车辆路径问题**（Vehicle Routing Problem，VRP）：给定一个配送中心和若干客户点，确定车辆的配送路线，使得所有客户的需求得到满足，同时优化某个目标（如总行驶距离最短）。

VRP 可以形式化为：

$$\min \quad \sum_{i,j,k} c_{ij} x_{ijk} \tag{11.1}$$

$$\text{s.t.} \quad \sum_{k} \sum_{j} x_{ijk} = 1, \quad \forall i\text{（每个客户恰好被访问一次）} \tag{11.2}$$

$$\sum_{i} d_i \sum_{j} x_{ijk} \le Q_k, \quad \forall k\text{（车辆容量约束）} \tag{11.3}$$

$$x_{ijk} \in \{0, 1\} \tag{11.4}$$

11.1.2　带时间窗的 VRP

定义 11.2 (VRPTW)**. 带时间窗的 VRP**（VRP with Time Windows）要求车辆在客户指定的时间范围内到达并提供服务。

11.1.3　带容量约束的 VRP

CVRP（Capacitated VRP）考虑车辆的载重限制。

11.2　求解方法

11.2.1　精确算法

- 分支定界法

- 分支切割法
- 列生成法

精确算法能够保证找到最优解，但对于大规模问题计算时间过长。

11.2.2　启发式与元启发式

- **构造启发式**：最近邻、节约算法、插入法
- **改进启发式**：2-opt、3-opt、Or-opt
- **元启发式**：遗传算法、模拟退火、禁忌搜索、蚁群优化

11.2.3　混合智能方法

结合精确方法和启发式的优点，如：

- 大规模邻域搜索（LNS）
- 自适应大规模邻域搜索（ALNS）
- 混合遗传算法

示例 11.1 (快递配送路径优化)**.** 某快递公司早班需要配送 200 个包裹到城区各处。

已知条件：

- 配送中心位置
- 200 个配送点的位置和包裹重量
- 10 辆配送车，每辆载重 500kg
- 部分配送点有时间要求

优化目标：最小化总行驶距离

求解过程：

1. 使用聚类算法初步划分区域
2. 为每个区域用节约算法构造初始路线
3. 使用 ALNS 进行改进
4. 检验时间窗约束，调整路线

优化结果：相比人工规划，总行驶距离减少 15%，配送时间缩短 20%。

示例 11.2 (公交线路规划)**.** 某城市需要设计新的公交线网。

输入数据：

- 城市道路网络
- 居民出行 OD 矩阵
- 现有公交站点
- 车辆配置

规划内容：

1. 线路走向设计
2. 站点设置
3. 发车频率确定
4. 车辆调度计划

优化目标：最大化乘客覆盖率，最小化总运营成本。

11.3 动态路径规划

11.3.1 实时重规划

在实际运营中，经常需要根据实时情况调整路线：

- 新订单插入
- 交通拥堵
- 车辆故障
- 客户取消

11.3.2　交通流预测

利用历史数据和实时数据预测交通流，提前调整路线。

示例 11.3 (网约车调度系统)**.** 网约车平台需要实时匹配乘客和司机。

核心问题：

- 订单分配：哪个司机接哪个乘客
- 路径规划：如何到达乘客位置和目的地
- 拼车匹配：多个乘客如何共乘
- 供需调度：如何引导空闲车辆

算法特点：

- 毫秒级响应要求
- 考虑未来需求预测
- 平衡效率和公平性

11.4　多式联运规划

示例 11.4 (集装箱多式联运优化)**.** 某物流公司需要将货物从上海港运送到成都仓库。

可选方式：

- 全程公路运输（快但贵）
- 铁路 + 公路（经济但慢）
- 水运 + 铁路 + 公路（最经济但最慢）

决策因素：

- 货物时效性要求
- 各环节成本
- 转运时间和成本
- 运力可用性

优化模型：多目标规划，平衡成本、时间和可靠性。

本章小结

本章介绍了任务规划在交通运输领域的应用。VRP 是物流配送的核心问题，有多种变体和求解方法。动态路径规划处理实时变化的情况。多式联运规划优化不同运输方式的组合。

习题

1. 用节约算法为一个 10 客户的 VRP 问题构造初始解。

2. 实现 2-opt 改进算法，改进 TSP 问题的初始解。

3. 对于示例11.1，讨论如何处理配送过程中的新订单。

4. 比较遗传算法和禁忌搜索在 VRP 问题上的性能。

5. 设计一个简单的网约车匹配算法，考虑等待时间和绕路距离。

第十二章　通信与编码传输规划

本章介绍任务规划在通信网络领域的应用。

12.1　通信网络资源调度

12.1.1　信道分配

定义 12.1 (信道分配问题). 在无线通信网络中，**信道分配**问题是将有限的频谱资源分配给各用户或基站，以最大化系统容量并最小化干扰。

信道分配可以建模为图着色问题：

- 节点表示通信链路
- 边表示干扰关系
- 颜色表示信道

12.1.2　功率控制

功率控制问题是确定每个发射机的发射功率，以满足通信质量要求并最小化总功耗或干扰。

12.2　无线传感器网络任务规划

12.2.1　数据采集调度

定义 12.2 (数据采集调度). 在无线传感器网络中，**数据采集调度**确定每个传感器节点何时进行数据采集、处理和传输，以平衡数据时效性和能量消耗。

12.2.2　能量感知路由

由于传感器节点通常由电池供电，路由协议需要考虑能量因素：

- 最小能量路由

- 能量均衡路由
- 基于剩余能量的路由

示例 12.1 (物联网数据汇聚规划)**.** 某智慧农业系统部署了 1000 个土壤传感器，需要定期向网关汇报数据。

约束条件：

- 每个传感器电池容量有限
- 传输距离影响能耗
- 数据需要在规定时间内到达
- 部分节点可以进行数据聚合

规划内容：

1. 确定数据汇聚树结构
2. 调度每个节点的醒睡周期
3. 确定数据聚合点
4. 优化传输时隙分配

优化目标：最大化网络寿命，保证数据时效性。

12.3 卫星通信任务规划

12.3.1 卫星过境调度

定义 12.3 (卫星过境调度)**.** 地面站需要与多颗卫星通信，但视窗时间有限。**卫星过境调度**问题是确定地面站与哪颗卫星在何时通信。

12.3.2 星地链路规划

示例 12.2 (遥感卫星成像任务规划)**.** 某遥感卫星星座需要完成 100 个地面目标的成像任务。

约束条件：

- 每颗卫星的轨道决定了可观测时间窗

- 卫星姿态调整需要时间
- 星上存储容量有限
- 数据需要在一定时间内下传

规划决策：

1. 任务分配：哪颗卫星观测哪个目标
2. 观测调度：何时进行观测
3. 数据下传：何时通过哪个地面站下传

优化目标：最大化完成的任务数，优先满足高优先级任务。
规划算法：

- 基于约束满足的方法
- 启发式搜索
- 遗传算法

12.4　数据中心任务调度

示例 12.3 (云计算资源编排)**.** 云计算平台需要调度大量虚拟机和容器。
任务类型：

- 批处理任务：可延迟执行
- 交互式任务：需要快速响应
- 流式任务：持续执行

资源约束：

- CPU、内存、存储、网络带宽
- 物理机的容量限制
- 数据局部性要求
- 服务质量（QoS）保证

调度目标：

- 最小化任务完成时间
- 最大化资源利用率
- 最小化能耗
- 保证服务等级协议（SLA）

调度策略：

1. 基于优先级的调度
2. 基于公平性的调度
3. 基于预测的调度
4. 基于强化学习的调度

本章小结

本章介绍了任务规划在通信领域的应用。信道分配和功率控制是无线网络的基本优化问题。无线传感器网络需要考虑能量约束。卫星通信和数据中心调度是大规模资源分配问题。

习题

1. 将一个简单的信道分配问题建模为图着色问题并求解。
2. 设计一个能量感知的路由算法，分析其对网络寿命的影响。
3. 对于示例12.2，用整数规划建模并求解一个小规模实例。
4. 比较不同云计算调度策略的优缺点。
5. 设计一个物联网数据采集调度算法，考虑能量和时延约束。

第十三章　军事任务规划

本章介绍任务规划在军事领域的应用。

13.1　军事规划概述

13.1.1　作战规划层次

军事规划通常分为多个层次：

- **战略层**：国家级战略目标和资源分配
- **战役层**：战区级作战计划
- **战术层**：具体作战行动规划
- **技术层**：武器系统和平台控制

13.1.2　C4ISR 系统

定义 13.1 (C4ISR)**. C4ISR** 代表指挥（Command）、控制（Control）、通信（Communication）、计算机（Computer）、情报（Intelligence）、监视（Surveillance）和侦察（Reconnaissance）。

任务规划系统是 C4ISR 的核心组件之一。

13.2　兵力部署规划

13.2.1　力量配置优化

定义 13.2 (兵力部署问题)**.** 给定作战目标和可用兵力，确定各作战单元的部署位置和任务分配，以最大化作战效能。

13.2.2　后勤保障规划

后勤保障规划包括：

- 物资补给调度
- 运输路线规划
- 维修保障安排
- 医疗后送计划

示例 13.1 (两栖登陆作战兵力部署). 某两栖作战需要在指定海岸登陆并建立滩头阵地。

可用兵力:

- 3 个海军陆战队营
- 2 个装甲连
- 空中支援力量
- 舰炮火力支援

规划内容:

1. 登陆点选择
2. 各波次兵力编组
3. 火力支援计划
4. 后续梯队跟进时机

约束条件:

- 滩头地形限制
- 敌方防御部署
- 潮汐和天气窗口
- 登陆舰艇容量

13.3 无人系统任务规划

13.3.1 无人机航迹规划

定义 13.3 (航迹规划). 为无人机确定从起点到目标再返回的飞行路径，需要考虑威胁规避、燃料约束、任务时限等因素。

航迹规划方法：

- Voronoi 图方法
- A* 及其变体
- RRT 方法
- 势场法

13.3.2　无人车任务分配

无人地面车辆（UGV）任务规划包括：

- 巡逻路线规划
- 侦察任务分配
- 物资运输调度

13.3.3　有人-无人协同

MUM-T（Manned-Unmanned Teaming）是现代军事的重要作战概念：

- 有人平台指挥控制
- 无人平台执行危险任务
- 协同态势感知
- 任务动态分配

示例 13.2 (无人机侦察监视任务)**.** 使用 4 架侦察无人机对某区域进行持续监视。

任务要求：

- 监视区域面积：100×100 km^2
- 监视周期：24 小时
- 重点目标需要持续监视
- 发现可疑目标需要详细侦察

规划内容：

1. 巡逻航线设计

2. 换班和加油调度
3. 应急任务响应
4. 通信中继安排

优化目标：最大化区域覆盖率，最小化目标发现延迟。

示例 13.3 (无人机蜂群协同打击)**.** 无人机蜂群对多个目标实施协同打击。

任务场景：

- 30 架察打一体无人机
- 10 个已知目标
- 未知的防空威胁
- 通信可能被干扰

规划层次：

1. **任务分配**：确定每架无人机的目标
2. **航迹规划**：规划进入和撤离航线
3. **协同时序**：同步攻击时间
4. **应急处置**：处理损失和新发现目标

分布式规划：

- 中央规划初始方案
- 各无人机本地调整
- 基于共识的协调
- 自主重规划能力

13.4 电子战任务规划

示例 13.4 (电子干扰任务规划)**.** 电子战飞机需要对敌方雷达网络实施干扰。

情报输入：

- 敌方雷达位置和类型

- 雷达工作参数
- 我方干扰设备能力

规划内容：

1. 干扰阵位选择
2. 干扰样式和时序
3. 功率和频率分配
4. 协同配合计划

效能评估：

- 雷达探测概率降低程度
- 掩护目标的暴露风险
- 干扰资源消耗

本章小结

本章介绍了任务规划在军事领域的应用。军事规划涵盖多个层次，从战略到技术层面。无人系统任务规划是当前的研究热点，包括航迹规划、任务分配和有人-无人协同。电子战规划是信息化战争的重要组成部分。

习题

1. 设计一个简单的无人机航迹规划算法，考虑禁飞区和威胁区。
2. 对于示例13.1，分析规划中需要考虑的不确定性因素。
3. 比较集中式和分布式无人机蜂群规划的优缺点。
4. 用 HTN 方法建模一个军事任务分解过程。
5. 讨论人工智能在军事任务规划中的伦理问题。

第四部分

前沿篇

第十四章　基于学习的规划方法 ★

本章介绍机器学习与规划的结合，属于研究生拓展内容。

14.1　强化学习与规划

14.1.1　模型规划 RL

定义 14.1 (基于模型的强化学习)**. 基于模型的强化学习**（Model-Based RL）通过学习环境动力学模型，然后在模型上进行规划来做出决策。

基本框架：

1. 与环境交互收集数据
2. 学习转移模型 $\hat{T}(s'|s,a)$ 和奖励模型 $\hat{R}(s,a)$
3. 在学习的模型上规划
4. 执行规划得到的动作

14.1.2　AlphaGo 与蒙特卡洛树搜索

AlphaGo 结合了深度学习和蒙特卡洛树搜索（MCTS）：

- **策略网络**：估计动作概率分布
- **价值网络**：估计局面价值
- **MCTS**：使用网络引导搜索

输入: 根节点状态 s
输出: 最佳动作
for $i = 1$ **到** N **do**
  node ← root;
  // 选择阶段
  while *node* **完全展开且非叶节点 do**
    node ← UCB1 选择的子节点;
  end
  // 展开阶段
  if *node* **未完全展开 then**
    node ← 展开一个未尝试的动作;
  end
  // 模拟阶段
  v ← 随机模拟到终局;
  // 回传阶段
  回传 v 更新路径上所有节点的统计;
end
返回 *访问次数最多的动作*

算法 11: MCTS 基本算法

14.2 模仿学习

14.2.1 行为克隆

定义 14.2 (行为克隆)**. 行为克隆**（Behavior Cloning）直接从专家演示中学习策略，将状态-动作对作为监督学习问题。

问题：分布漂移（Distribution Shift）——训练时的状态分布与执行时不同。

14.2.2 逆强化学习

定义 14.3 (逆强化学习)**. 逆强化学习**（Inverse Reinforcement Learning，IRL）从专家演示中推断奖励函数，然后用标准 RL 方法学习策略。

14.3　神经网络规划器

14.3.1　神经符号规划

神经符号规划结合神经网络的学习能力和符号规划的推理能力：

- 神经网络处理感知和特征提取
- 符号系统处理推理和规划
- 接口层连接两个组件

14.3.2　图神经网络在规划中的应用

图神经网络（GNN）可以处理规划问题的结构化表示：

- 状态表示为图结构
- GNN 学习状态特征
- 用于启发式估计或策略学习

14.4　迁移学习与元学习

14.4.1　领域迁移

在源领域学习的规划知识可以迁移到目标领域：

- 启发式函数迁移
- 动作模式迁移
- 领域知识迁移

14.4.2　少样本规划

元学习使规划器能够快速适应新领域：

- 学习如何学习规划
- 从少量样本中提取领域特征
- 快速调整规划策略

本章小结

本章介绍了基于学习的规划方法。强化学习与规划的结合是重要的研究方向，AlphaGo 是成功案例。模仿学习从专家演示中学习。神经符号规划结合了深度学习和符号推理的优势。迁移学习和元学习帮助规划器泛化到新问题。

习题

1. 解释为什么 AlphaGo 需要结合策略网络和价值网络。

2. 比较行为克隆和逆强化学习的优缺点。

3. 设计一个用 GNN 表示规划问题状态的方案。

4. 讨论神经网络规划器的可解释性问题。

5. 阅读一篇神经符号规划的最新论文，总结其主要贡献。

第十五章　大语言模型与任务规划 ★

本章介绍大语言模型在任务规划中的应用，属于研究生拓展内容。

15.1　LLM 规划能力分析

15.1.1　LLM 的推理能力

大语言模型（如 GPT-4、Claude 等）展示了一定的推理和规划能力：

- 常识推理
- 多步骤问题求解
- 代码生成和调试
- 任务分解

15.1.2　规划基准测试（PlanBench）

定义 15.1 (PlanBench)**. PlanBench** 是评估 LLM 规划能力的基准测试套件，包含基于国际规划竞赛领域的问题。

PlanBench 的测试类别：

- 计划生成（零样本/少样本）
- 计划验证
- 目标识别
- 代价优化

15.1.3　LLM 规划的局限性

研究发现 LLM 在规划任务上存在系统性不足：

1. **长时域规划困难**：步数增加时性能显著下降
2. **计划有效性问题**：生成的计划可能违反约束

3. **缺乏形式化保证**：无法确保解的正确性

4. **结构化推理不足**：对复杂依赖关系的处理能力有限

15.2 LLM 作为规划组件

鉴于 LLM 作为独立规划器的局限性，研究者探索将 LLM 作为规划系统的组件。

15.2.1 LLM+P：自然语言到 PDDL

定义 15.2 (LLM+P)**. LLM+P** 框架使用 LLM 将自然语言问题描述翻译为 PDDL 格式，然后由经典规划器求解。

工作流程：

1. 用户用自然语言描述问题

2. LLM 生成 PDDL 领域和问题文件

3. 经典规划器（如 Fast Downward）求解

4. LLM 将计划翻译回自然语言

15.2.2 LLM 生成启发式函数

LLM 可以为规划问题生成领域特定的启发式函数：

- 输入：PDDL 领域描述
- 输出：Python 实现的启发式函数
- 优势：保持规划器的正确性保证

15.2.3 LLM 作为世界模型

LLM 可以作为世界模型，预测动作的效果：

- 处理开放领域问题
- 利用常识知识
- 但可能产生幻觉

15.3　具身智能规划

15.3.1　SayCan 框架

定义 15.3 (SayCan)**. SayCan** 框架结合 LLM 的语言理解能力和机器人的可供性（affordance）函数：

$$\pi(a|s,i) \propto p_{\text{LLM}}(a|i) \cdot p_{\text{affordance}}(a|s) \tag{15.1}$$

其中 i 是自然语言指令，s 是当前状态。

15.3.2　SayPlan 与 3D 场景图

定义 15.4 (SayPlan)**. SayPlan** 使用 3D 场景图（3DSG）来提供环境的结构化表示，使 LLM 能够在大规模环境中进行可扩展的任务规划。

SayPlan 的关键创新：

- 场景图作为环境表示
- 多层次抽象
- 语义搜索缩小相关范围

15.3.3　Inner Monologue

定义 15.5 (Inner Monologue)**. Inner Monologue** 通过环境反馈形成” 内心独白”，包括：

- 被动场景描述
- 主动场景描述
- 成功检测反馈

这种闭环反馈显著提升了机器人任务执行的成功率。

15.4　多智能体 LLM 系统

15.4.1　LLM-MAS 架构

多智能体 LLM 系统的典型架构：

- 每个智能体由一个 LLM 驱动
- 智能体具有不同的角色和专长
- 通过对话进行协调
- 共同完成复杂任务

15.4.2 智能体协作框架

代表性框架：

- **AutoGen**：微软的多智能体对话框架
- **CAMEL**：角色扮演对话框架
- **MetaGPT**：软件开发多智能体系统

本章小结

本章介绍了大语言模型与任务规划的结合。LLM 展示了一定的规划能力，但存在系统性局限。将 LLM 作为规划系统的组件（如翻译器、启发式生成器）是更有效的方法。具身智能规划框架（SayCan、SayPlan）展示了 LLM 在机器人领域的应用。多智能体 LLM 系统是新兴的研究方向。

习题

1. 分析 LLM 在长时域规划中性能下降的原因。
2. 用 LLM 将一个简单的规划问题翻译为 PDDL，并验证翻译的正确性。
3. 比较 LLM+P 和纯 LLM 规划的优缺点。
4. 讨论 SayCan 中可供性函数的作用。
5. 阅读一篇多智能体 LLM 系统的论文，总结其协调机制。

第十六章　规划系统的可解释性与安全性 ★

本章讨论规划系统的可解释性和安全性问题，属于研究生拓展内容。

16.1　可解释规划

16.1.1　计划解释生成

定义 16.1 (计划解释)**. 计划解释**是向用户说明为什么规划器生成了特定的计划，以及为什么没有选择其他方案。

解释类型：

- **对比解释**：为什么选择动作 A 而不是 B?
- **因果解释**：这个动作导致了什么结果?
- **目标解释**：这个动作如何帮助实现目标?

16.1.2　人机协同规划

人机协同规划中，系统需要：

- 理解人类的意图和偏好
- 解释自己的决策
- 接受人类的反馈和修正
- 保持适当的自主性

定义 16.2 (混合主动规划)**. 混合主动规划**（Mixed-Initiative Planning）中，人类和 AI 系统共同参与规划过程，各自贡献专长。

16.2　规划系统验证

16.2.1　形式化验证方法

定义 16.3 (形式化验证)**. 形式化验证**使用数学方法证明规划系统满足特定性质。

验证目标包括：

- **正确性**：计划能够达到目标
- **完整性**：如果解存在，系统能找到
- **最优性**：找到的解是最优的
- **安全性**：执行过程中不会进入危险状态

16.2.2 模型检测

定义 16.4 (模型检测)**. 模型检测**（Model Checking）自动验证系统模型是否满足给定的时态逻辑性质。

在规划中的应用：

- 验证计划满足安全约束
- 检测死锁和活锁
- 验证时序性质

16.3 安全关键系统规划

16.3.1 安全约束规划

在安全关键系统中，规划必须满足硬性安全约束：

- 机器人必须避免碰撞
- 车辆必须遵守交通规则
- 医疗系统必须避免有害操作

定义 16.5 (安全规划)**. 安全规划**生成的计划保证在执行过程中永远不会进入不安全状态。

方法：

- 将安全约束编码到规划问题中
- 使用屏障函数保证安全
- 在线监控和干预

16.3.2　鲁棒性分析

分析规划系统在以下情况下的表现：

- 模型不准确
- 执行误差
- 环境变化
- 对抗性干扰

16.4　伦理与法规

16.4.1　自主系统伦理

自主规划系统面临的伦理问题：

- **责任归属**：当自主系统造成损害时，谁负责？
- **透明度**：用户是否知道系统如何做决策？
- **公平性**：决策是否存在偏见？
- **自主性边界**：系统何时应该请求人类干预？

16.4.2　军事 AI 治理

军事 AI 规划系统的特殊考虑：

- **人类控制**：致命性决策必须有人类参与
- **国际法遵从**：遵守武装冲突法
- **责任链**：明确指挥和责任关系
- **可预测性**：行为应当可预测和可解释

示例 16.1 (自主驾驶的伦理困境)**.** 自动驾驶车辆面临不可避免的碰撞时，应该如何决策？

场景：车辆前方突然出现行人，左侧是护栏，右侧是其他行人。
伦理问题：

- 是否可以”选择”伤害谁？

- 是否应该保护乘客优先?
- 如何量化不同选择的” 代价”?

技术考虑:

- 规划系统如何表示这类约束?
- 决策过程如何做到可解释?
- 如何满足法规要求?

本章小结

本章讨论了规划系统的可解释性和安全性。可解释规划帮助用户理解和信任系统决策。形式化验证和模型检测为规划系统提供正确性保证。安全关键系统需要特别的规划方法。伦理和法规问题是自主系统部署的重要考虑。

习题

1. 为一个简单的规划问题设计解释生成方法。
2. 讨论人机协同规划中的信任问题。
3. 用模型检测验证一个简单规划系统的安全性质。
4. 分析自动驾驶规划系统的伦理设计原则。
5. 讨论军事 AI 规划系统应该遵循的原则。

附录

附录 A　PDDL 语言参考手册

本附录提供 PDDL（Planning Domain Definition Language）的语法参考。

A.1　PDDL 基本结构

A.1.1　领域文件结构

```
(define (domain <domain-name>)
  (:requirements <requirement-flags>)
  (:types <type-definitions>)
  (:constants <constant-definitions>)
  (:predicates <predicate-definitions>)
  (:functions <function-definitions>)
  (:action <action-definition>)*
)
```

A.1.2　问题文件结构

```
(define (problem <problem-name>)
  (:domain <domain-name>)
  (:objects <object-definitions>)
  (:init <initial-state>)
  (:goal <goal-specification>)
  (:metric <metric-specification>)
)
```

A.2　需求标志

常用需求标志：

- `:strips` – 基本 STRIPS 功能
- `:typing` – 类型系统

- :negative-preconditions – 负前提条件
- :disjunctive-preconditions – 析取前提条件
- :equality – 等式判断
- :existential-preconditions – 存在量词
- :universal-preconditions – 全称量词
- :conditional-effects – 条件效果
- :numeric-fluents – 数值变量
- :durative-actions – 持续性动作
- :duration-inequalities – 持续时间不等式
- :continuous-effects – 连续效果

A.3　类型定义

```
(:types
  location city - object
  truck airplane - vehicle
  package - object
)
```

A.4　谓词定义

```
(:predicates
  (at ?obj - object ?loc - location)
  (in ?pkg - package ?veh - vehicle)
  (connected ?from ?to - location)
)
```

A.5 函数定义

```
(:functions
  (distance ?from ?to - location)
  (fuel ?v - vehicle)
  (total-cost)
)
```

A.6 动作定义

A.6.1 基本动作

```
(:action drive
  :parameters (?t - truck ?from ?to - location)
  :precondition (and
    (at ?t ?from)
    (connected ?from ?to)
  )
  :effect (and
    (not (at ?t ?from))
    (at ?t ?to)
    (increase (total-cost) (distance ?from ?to))
  )
)
```

A.6.2 持续性动作（PDDL 2.1）

```
(:durative-action fly
  :parameters (?a - airplane ?from ?to - city)
  :duration (= ?duration (/ (distance ?from ?to) (speed ?a)))
  :condition (and
    (at start (at ?a ?from))
    (at start (>= (fuel ?a) (fuel-required ?from ?to)))
    (over all (available ?to))
  )
  :effect (and
```

```
    (at start (not (at ?a ?from)))
    (at end (at ?a ?to))
    (at start (decrease (fuel ?a) (fuel-required ?from ?to)))
  )
)
```

A.7　目标规范

A.7.1　简单目标

```
(:goal (and
  (at pkg1 loc-b)
  (at pkg2 loc-c)
))
```

A.7.2　带偏好的目标（PDDL 3.0）

```
(:goal (and
  (at pkg1 loc-b)
  (preference p1 (at pkg2 loc-c))
))

(:metric minimize (+ (total-cost) (* 100 (is-violated p1))))
```

A.8　度量规范

```
(:metric minimize (total-cost))
(:metric maximize (packages-delivered))
(:metric minimize (total-time))
```

A.9　完整示例

A.9.1　物流领域

```
(define (domain logistics)
  (:requirements :strips :typing)

  (:types
    city location thing - object
    package vehicle - thing
    truck airplane - vehicle
    airport - location
  )

  (:predicates
    (in-city ?loc - location ?city - city)
    (at ?obj - thing ?loc - location)
    (in ?pkg - package ?veh - vehicle)
  )

  (:action load-truck
    :parameters (?pkg - package ?truck - truck ?loc - location)
    :precondition (and (at ?truck ?loc) (at ?pkg ?loc))
    :effect (and (not (at ?pkg ?loc)) (in ?pkg ?truck))
  )

  (:action unload-truck
    :parameters (?pkg - package ?truck - truck ?loc - location)
    :precondition (and (at ?truck ?loc) (in ?pkg ?truck))
    :effect (and (not (in ?pkg ?truck)) (at ?pkg ?loc))
  )

  (:action drive-truck
    :parameters (?truck - truck ?from ?to - location ?city - city)
    :precondition (and
      (at ?truck ?from)
      (in-city ?from ?city)
      (in-city ?to ?city)
    )
    :effect (and (not (at ?truck ?from)) (at ?truck ?to))
  )
```

```
38
39    (:action load-airplane
40      :parameters (?pkg - package ?airplane - airplane ?loc - airport)
41      :precondition (and (at ?pkg ?loc) (at ?airplane ?loc))
42      :effect (and (not (at ?pkg ?loc)) (in ?pkg ?airplane))
43    )
44
45    (:action unload-airplane
46      :parameters (?pkg - package ?airplane - airplane ?loc - airport)
47      :precondition (and (in ?pkg ?airplane) (at ?airplane ?loc))
48      :effect (and (not (in ?pkg ?airplane)) (at ?pkg ?loc))
49    )
50
51    (:action fly-airplane
52      :parameters (?airplane - airplane ?from ?to - airport)
53      :precondition (at ?airplane ?from)
54      :effect (and (not (at ?airplane ?from)) (at ?airplane ?to))
55    )
56  )
```

附录 B　常用规划器安装与使用指南

本附录介绍几种常用规划器的安装和使用方法。

B.1　Fast Downward

B.1.1　简介

Fast Downward 是最成功的现代规划系统之一，支持多种启发式和搜索算法。

B.1.2　安装

Linux/macOS：

```
# 克隆仓库
git clone https://github.com/aibasel/downward.git
cd downward

# 编译
./build.py
```

依赖：

- Python 3.6+
- C++20 编译器（GCC 10+ 或 Clang 12+）
- CMake 3.16+

B.1.3　基本使用

```
# 使用A*搜索和LM-cut启发式
./fast-downward.py domain.pddl problem.pddl \
    --search "astar(lmcut())"

# 使用贪婪最佳优先搜索和FF启发式
./fast-downward.py domain.pddl problem.pddl \
```

```
    --search "eager_greedy([ff()])"

# LAMA 配置
./fast-downward.py domain.pddl problem.pddl --alias lama
```

B.1.4　常用配置

配置	说明
`--alias lama`	LAMA 配置，平衡质量和速度
`--alias lama-first`	快速找到第一个解
`--alias seq-opt-lmcut`	最优规划，使用 LM-cut
`--alias seq-sat-lama`	满足性规划，LAMA 风格

B.2　PDDL4J

B.2.1　简介

PDDL4J 是一个 Java 库，提供 PDDL 解析和多种规划算法。

B.2.2　安装

```
# 使用 Maven
<dependency>
    <groupId>fr.uga</groupId>
    <artifactId>pddl4j</artifactId>
    <version>4.0.0</version>
</dependency>
```

B.2.3　命令行使用

```
java -jar pddl4j.jar -o domain.pddl -f problem.pddl
```

B.3 SHOP2

B.3.1 简介

SHOP2 是一个 HTN 规划系统，使用 Lisp 实现。

B.3.2 安装

```
# 下载SHOP2
# 需要Common Lisp环境（如SBCL）

# 在SBCL中加载
(load "shop2.lisp")
```

B.3.3 基本使用

```
;; 定义领域
(defdomain logistics
  (:method (deliver ?pkg ?dest)
    ((at ?pkg ?loc))
    ((transport ?pkg ?loc ?dest)))
  ...
)

;; 求解问题
(find-plans 'logistics-problem :verbose t)
```

B.4 Pyperplan

B.4.1 简介

Pyperplan 是一个教学用途的 Python 规划器，代码简洁易读。

B.4.2 安装

```
pip install pyperplan
```

B.4.3　使用

```
# 命令行
pyperplan domain.pddl problem.pddl

# Python API
from pyperplan import planner
solution = planner.search(domain, problem, "astar", "hadd")
```

B.5　在线规划工具

B.5.1　Planning.Domains

网址：http://planning.domains/
提供在线 PDDL 编辑器和多个规划器。

B.5.2　Web Planner

网址：http://editor.planning.domains/
功能：

- 在线编辑 PDDL
- 语法高亮和检查
- 多规划器支持
- 可视化执行

B.6　调试技巧

B.6.1　常见错误

1. **语法错误**：使用在线编辑器检查
2. **类型不匹配**：检查参数类型定义
3. **无解**：简化问题或检查目标可达性
4. **内存不足**：使用更高效的启发式

B.6.2 性能优化

- 选择合适的启发式函数
- 使用类型系统减少搜索空间
- 添加领域特定的约束
- 考虑问题分解

附录 C　国际规划竞赛（IPC）介绍

本附录介绍国际规划竞赛的背景和主要赛道。

C.1　IPC 历史

国际规划竞赛（International Planning Competition，IPC）是自动规划领域的重要评测平台，始于 1998 年。

表 C.1: IPC 历史

年份	届次	主要创新
1998	IPC-1	引入 PDDL
2000	IPC-2	ADL 特性
2002	IPC-3	时态规划赛道
2004	IPC-4	数值规划
2006	IPC-5	偏好和约束
2008	IPC-6	不确定性赛道
2011	IPC-7	学习赛道
2014	IPC-8	确定性和学习
2018	IPC-9	多个子竞赛
2023	IPC-10	HTN 赛道、学习赛道

C.2　竞赛赛道

C.2.1　经典赛道

特点:

- 完全可观测
- 确定性动作
- 有限状态空间

- 使用 PDDL 描述

评价指标:

- 解的数量
- 解的质量（代价）
- 求解时间
- IPC 得分

C.2.2 最优规划赛道

要求找到代价最小的解，规划器必须证明解的最优性。

C.2.3 满足性规划赛道

只要求找到可行解，不要求最优。

C.2.4 HTN 赛道

使用 HDDL 语言，评估层次任务网络规划器。

C.2.5 学习赛道

规划器可以在相似问题上学习，然后在新问题上测试。

C.3 基准领域

C.3.1 经典领域

Blocksworld 积木世界，最经典的测试领域

Logistics 物流运输，多城市、多车辆

Gripper 机器人搬运球

Satellite 卫星观测调度

Rovers 火星探测器规划

Airport 机场地面交通控制

Pipesworld 管道输油调度

C.3.2　时态领域

Zenotravel 旅行规划，考虑燃料

Depots 仓库物流

DriverLog 司机和卡车调度

C.4　参与 IPC

C.4.1　获取基准问题

```
# 克隆IPC基准仓库
git clone https://github.com/aibasel/downward-benchmarks.git
```

C.4.2　评测工具

- **Lab**：实验框架，用于运行和分析规划实验
- **VAL**：计划验证工具
- **IPC 得分计算**：标准化的评分方法

C.4.3　IPC 得分

对于问题 p 和规划器 P，IPC 得分定义为：

$$\text{Score}(P, p) = \frac{C^*}{C(P, p)} \tag{C.1}$$

其中 C^* 是已知最优解代价，$C(P, p)$ 是 P 找到的解的代价。

总得分是所有问题得分之和。

C.5　优秀规划器

历年 IPC 优秀规划器：

- **Fast Downward**：多次获奖
- **LAMA**：IPC 2011 冠军
- **Scorpion**：最优规划赛道优秀表现

- **BFWS**：最佳优先宽度搜索
- **Ragnarok**：IPC 2023 经典赛道第一

C.6 相关资源

- IPC 官网：https://www.icaps-conference.org/competitions/
- IPC 2023：https://ipc2023.github.io/
- PDDL 参考：https://planning.wiki/
- Downward 基准：https://github.com/aibasel/downward-benchmarks

附录 D　编程实验指导

本附录提供配套的编程实验指导。

D.1　实验 1：状态空间搜索实现

D.1.1　实验目的

1. 理解状态空间搜索的基本原理
2. 实现 BFS、DFS 和 A* 算法
3. 比较不同搜索策略的性能

D.1.2　实验内容

实现八数码问题（8-puzzle）求解器。

任务 1：实现状态表示

```
class PuzzleState:
    def __init__(self, board):
        """board是3x3的列表，0表示空格"""
        self.board = board

    def get_neighbors(self):
        """返回所有可能的后继状态"""
        pass

    def is_goal(self):
        """判断是否为目标状态"""
        pass
```

任务 2：实现 BFS

```
def bfs(initial_state):
    """广度优先搜索"""
    frontier = deque([initial_state])
```

```
    explored = set()
    # 实现搜索逻辑
    pass
```

任务 3：实现 A* 算法

```
def astar(initial_state, heuristic):
    """A*搜索"""
    # 使用优先队列
    # 实现f = g + h的评估
    pass
```

任务 4：实现启发式函数

- 曼哈顿距离
- 错位数
- 线性冲突

D.1.3 实验报告要求

1. 比较 BFS 和 A* 的扩展节点数
2. 分析不同启发式的效果
3. 讨论算法的时间和空间复杂度

D.2 实验 2：PDDL 建模练习

D.2.1 实验目的

1. 掌握 PDDL 语法
2. 学会建模实际问题
3. 使用规划器求解

D.2.2 实验内容

为”机器人仓库”问题编写 PDDL 描述。

问题描述：

- 仓库是 $n \times m$ 的网格
- 有若干机器人和货物
- 机器人可以移动、拾取和放下货物
- 目标是将货物运送到指定位置

任务 1：编写领域文件

- 定义类型：location, robot, package
- 定义谓词：at, holding, empty, adjacent
- 定义动作：move, pick, place

任务 2：编写问题文件

- 定义具体的仓库布局
- 设置初始状态
- 指定目标条件

任务 3：使用规划器求解

```
# 使用Fast Downward
./fast-downward.py warehouse-domain.pddl warehouse-problem.pddl \
    --search "astar(ff())"
```

D.3　实验 3：HTN 规划系统使用

D.3.1　实验目的

1. 理解 HTN 规划的原理
2. 学会使用 SHOP2 或 Pyhop
3. 设计任务分解方法

D.3.2 实验内容

使用 Pyhop 实现”旅行规划”系统。

安装 Pyhop:

```
git clone https://github.com/dananau/pyhop.git
```

任务:

1. 定义原子任务：taxi, walk, fly
2. 定义复合任务：travel
3. 定义分解方法
4. 测试不同场景

```
import pyhop

def travel_by_taxi(state, person, origin, dest):
    """打车方法"""
    if state.cash[person] >= taxi_fare(origin, dest):
        return [('taxi', person, origin, dest)]
    return False

pyhop.declare_methods('travel', travel_by_taxi, travel_by_foot)
```

D.4 实验 4：多智能体路径规划仿真

D.4.1 实验目的

1. 理解 MAPF 问题
2. 实现基本的 MAPF 算法
3. 可视化多智能体路径

D.4.2 实验内容

实现优先级规划和 CBS 算法。

任务 1：实现地图和智能体表示

任务 2：实现优先级规划

```python
def priority_planning(agents, obstacles):
    """按优先级顺序规划"""
    paths = []
    for agent in sorted(agents, key=priority):
        path = astar_with_constraints(agent, paths)
        paths.append(path)
    return paths
```

任务 3：实现冲突检测和 CBS 框架

任务 4：可视化

- 使用 matplotlib 或 pygame
- 动态展示智能体移动
- 标注冲突位置

D.5　实验 5：VRP 求解器开发

D.5.1　实验目的

1. 理解 VRP 问题
2. 实现构造启发式
3. 实现改进启发式

D.5.2　实验内容

为小规模 VRP 问题开发求解器。

任务 1：实现最近邻构造法

```python
def nearest_neighbor(depot, customers, vehicle_capacity):
    """最近邻启发式"""
    routes = []
    unvisited = set(customers)
    while unvisited:
        route = [depot]
        load = 0
        while unvisited:
```

```
            nearest = find_nearest(route[-1], unvisited)
            if load + demand[nearest] <= vehicle_capacity:
                route.append(nearest)
                load += demand[nearest]
                unvisited.remove(nearest)
            else:
                break
        route.append(depot)
        routes.append(route)
    return routes
```

任务 2：实现 2-opt 改进
任务 3：测试和比较

- 在标准测试集上测试
- 比较不同方法的解质量
- 分析计算时间

D.6 提交要求

每个实验需要提交：

1. 源代码（带注释）
2. 实验报告（PDF 格式）
3. 运行结果截图或日志

实验报告应包含：

- 实验目的
- 算法描述
- 实现细节
- 结果分析
- 问题和思考

参考文献

[1] Ghallab M, Nau D, Traverso P. Automated Planning: Theory and Practice[M]. Morgan Kaufmann, 2004.

[2] Russell S, Norvig P. Artificial Intelligence: A Modern Approach (4th Edition)[M]. Pearson, 2020.

[3] Geffner H, Bonet B. A Concise Introduction to Models and Methods for Automated Planning[M]. Morgan & Claypool Publishers, 2013.

[4] 蔡自兴, 刘丽珏, 陈白帆, 等. 人工智能及其应用（第 7 版）[M]. 清华大学出版社, 2024.

[5] 王万良. 人工智能导论（第 5 版）[M]. 高等教育出版社, 2020.

[6] LaValle S M. Planning Algorithms[M]. Cambridge University Press, 2006.

[7] Wooldridge M. An Introduction to MultiAgent Systems (2nd Edition)[M]. John Wiley & Sons, 2009.

[8] Sutton R S, Barto A G. Reinforcement Learning: An Introduction (2nd Edition)[M]. MIT Press, 2018.

[9] Kautz H, Selman B. Planning as satisfiability[C]. ECAI, 1992: 359-363.

[10] Hoffmann J, Nebel B. The FF planning system: Fast plan generation through heuristic search[J]. Journal of Artificial Intelligence Research, 2001, 14: 253-302.

[11] Helmert M. The Fast Downward Planning System[J]. Journal of Artificial Intelligence Research, 2006, 26: 191-246.

[12] Nau D, Au T C, Ilghami O, et al. SHOP2: An HTN planning system[J]. Journal of Artificial Intelligence Research, 2003, 20: 379-404.

[13] Kaelbling L P, Littman M L, Cassandra A R. Planning and acting in partially observable stochastic domains[J]. Artificial Intelligence, 1998, 101(1-2): 99-134.

[14] Stern R, Sturtevant N, Felner A, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks[C]. SOCS, 2019.

[15] Garrett C R, Lozano-Pérez T, Kaelbling L P. PDDLStream: Integrating symbolic planners and blackbox samplers via optimistic adaptive planning[C]. ICAPS, 2020.

术语表

中文术语	英文术语	说明
任务规划	Task Planning	制定行动序列以实现目标的过程
自动规划	Automated Planning	由计算机自动生成规划的技术
状态空间	State Space	所有可能状态的集合
动作	Action	改变系统状态的操作
前提条件	Precondition	动作执行前必须满足的条件
效果	Effect	动作执行后产生的状态变化
启发式	Heuristic	引导搜索的估计函数
STRIPS	STRIPS	Stanford Research Institute Problem Solver
PDDL	PDDL	Planning Domain Definition Language
HTN	HTN	Hierarchical Task Network
时态规划	Temporal Planning	考虑时间约束的规划
调度	Scheduling	为任务分配时间和资源
多智能体系统	Multi-Agent System	多个智能体协作的系统
MDP	MDP	Markov Decision Process
POMDP	POMDP	Partially Observable MDP
运动规划	Motion Planning	规划物理运动轨迹
TAMP	TAMP	Task and Motion Planning
VRP	VRP	Vehicle Routing Problem