第5章

规　　划

前几章介绍的搜索技术已被应用于比简单谜题更实用的领域。规划的应用范围非常广泛。这里的目标是优化工程流程中的行动序列：设计高效的装配线、优化机器人的行为或开发有用的人工智能软件。

这一领域的内容非常丰富，如果对其所有主要方面进行全面的阐述，需要一整本书的篇幅。由于篇幅有限，我们将把注意力缩小到确定性规划上，即每个行动的结果不受偶然性或对手干扰的影响。本章的核心内容是 STRIPS，这是一种自 20 世纪 70 年代以来就一直存在的传奇方法。它的精髓可以在简单的积木世界中解释，但这种方法很容易推广到更广泛的环境中。

为了让读者相信规划应用是非常有用的，本章接下来将介绍几个模型应用，从流行的旅行推销员问题到背包问题，再到工作车间调度问题，不一而足。

5.1　玩具积木

人工智能规划所面临的问题很容易通过一个领域来说明，这个领域的简单性和清晰性使其在本科生和研究生阶段的教师中都很受欢迎，它就是积木世界。一个典型的任务是找到一连串动作，将一组积木按照预定的方式排列起来。[①]

5.1.1　移动积木

考虑一个只有 3 个方块的场景，如图 5.1 所示。任务是将左边的初始状态转换为右边的最终状态。每次只能移动一个方块，而且不用关心各列之间的距离。

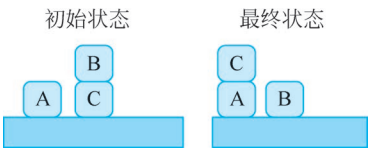

图 5.1　找到一个动作序列，通过每次移动一个方块，将初始状态转换为最终状态

从表面上看，这似乎是微不足道的，读者无疑会立刻明白解题思路：把 B 放在桌子上，然后把 C 放在 A 上。例如，我们知道，在初始状态下，把 A 从桌子上移

① 作者在阅读 Ginsberg(1993 年)的优秀教科书(即使现已过时)时首次熟悉了这个领域。

开，放在 B 的上面是愚蠢的。我们希望机器能够采取同样的观点。

这里要处理的是一个经典的搜索问题，其特征包括初始状态、最终状态、中间状态和搜索操作符。让我们来看看在计算机程序中实现这一问题的一种常用方法。

5.1.2 描述符

首先，我们需要一组描述符来描述每个给定状态。描述符应提供所有必要的信息，以决定在此状态下是否可以执行某个操作，以及该操作的后果是什么。回想一下这个程序的主要约束条件：每次只能移动一个方块。这意味着，只有当一个方块的顶部没有任何东西时，它才可以被移动。因此，我们的第一个描述符将告诉智能体区块 x 的顶部是否空闲：

$$\text{clear}(x) \tag{5.1}$$

对于每个方块，智能体还需要知道它当前的位置是在桌子上还是在另一个方块上。这就需要使用下面的描述符来说明 x 位于 y 上：

$$\text{loc}(x,y) \tag{5.2}$$

在这个描述符中，与前一个描述符一样，变量 x 可以被实例化[①]到任何一个方块：第二个变量 y 可以实例化到任何位置，可以是一个方块 A、B、C 或表。此外，我们要求 y 的实例化值与 x 的不同。

5.1.3 状态描述示例

观察图 5.1 中的初始状态，发现它可以用以下描述符来描述：

$$S_I = \{\text{loc(A,table)},\text{loc(C,table)},\text{loc(B,C)},\text{clear(A)},\text{clear(B)}\}$$

从初始状态开始，我们希望找到一连串操作，将 S_I 转换为最终状态 S_F 的操作序列：

$$S_F = \{\text{loc(A,table)},\text{loc(B,table)},\text{loc(C,A)},\text{clear(B)},\text{clear(C)}\}$$

5.1.4 注释

描述符的具体选择由程序员负责。在这里的具体案例中，创建描述符很容易。一般有两个描述符就足够了：clear(x)和 loc(x,y)。在更现实的环境中，可能存在多种选择，这在很大程度上取决于程序员的经验和技能。但有些限制必须遵守。描述符必须能够明确地描述任何可能的状态，并有助于确定具体操作的合法性。

注意，有些描述符可能是多余的。因此，在图 5.1 的简单场景中，我们可以假定任何未给出位置的方块位于桌子上（而不是另一个方块上）。此外，描述符 clear(table)也是多余的，因为它告诉我们有东西可以放在桌子上。如果表格足够大，情况就总是如此。

控制问题

如果你在回答下列任何问题时遇到困难，那么请返回阅读前文的相应部分。

- 什么是描述符？请举例说明。
- 工程师在为特定应用领域设计描述符时应遵守哪些规则？

① 这里的实例化是指用常量替换变量。

5.2 可用操作

现在我们已经知道如何描述各个状态，下一个问题就是如何描述执行每个状态下可用的搜索操作符。

5.2.1 玩具场景的操作

在图 5.1 所示的场景中，搜索操作符将一个方块从当前位置移除并转移到另一个位置。可以用以下公式来表示这一操作。

$$a = \text{move}(x, y, z) \tag{5.3}$$

其解释是："将方块 x 从其当前位于 y 上的位置移除，并将其置于 z 上"。在这里，变量 x 可以被实例化为 A、B 或 C，而 y 和 z 可以被实例化为表格或 A、B、C 中的一个。

注意，动作 a 是通用的，因为它必须首先实例化为一个具体的版本才能适用。通用操作所代表的版本数量等于其实例的数量。

5.2.2 前提条件列表

在执行一个动作之前，智能体必须确定该动作在给定状态下是否合法，是否可以执行。例如，如果 A 在桌子上，那么就不能从 B 的上面移走它；这就排除了 move(A,B,C)这个动作，只有当状态描述包含 loc(A,B)时，这个动作才是合法的。我们意识到，要使任何具体动作在特定状态下合法，该状态必须满足特定于该动作的某些前提条件。这些前提条件的形式是一组必须包含在状态描述中的描述符。

这意味着在定义一个动作时，工程师必须定义该动作的前提条件列表。因此，move(x, y,z)必须满足 3 个前提条件。第一，只有当 x 存在于 y 的顶端时，才能将其从 y 的顶端移除。第二，x 的顶部必须是空的，因为智能体一次只能移动一个方块。第三，目的地的顶部也必须是空的，因为如果已经有其他东西在那里，智能体就不能把 x 放在 z 的顶部。在 5.1 节介绍的描述符的帮助下，这 3 个前提条件可以用下面的公式表示：

$$P(a) = \{\text{loc}(x, y), \text{clear}(x), \text{clear}(z)\} \tag{5.4}$$

要使动作 a 在描述符集 S 所描述的状态中合法，必须满足以下条件：

$$P(a) \subseteq S \tag{5.5}$$

5.2.3 添加列表

执行操作后，系统的状态会发生变化。5.1 节建议用描述符列表来描述任何状态。执行操作的结果是修改该列表。有些项目被添加，有些项目被删除。

具体来说，操作 $a = \text{move}(x, y, z)$将 x 从 y 的顶部移除并放到 z 上，会在列表中添加两个描述符；其中一个指定 x 现在位于 z 上，另一个指定 y 的顶部现在是空的。这反映在下面的添加列表中：

$$A(a) = \{\text{loc}(x, z), \text{clear}(y)\} \tag{5.6}$$

5.2.4 删除列表

除了向状态描述中添加一些描述符外，该操作还导致从状态描述中删除一些描述符。

执行 $a=\text{move}(x,y,z)$操作后，方块 x 将不再位于 y 上，z 的顶部也不再是空的，因为 x 已被放置在那里。这反映在下面的删除列表中：

$$D(a)=\{\text{loc}(x,y),\text{clear}(z)\} \tag{5.7}$$

5.2.5 定义 move(x，y，z)

总体来说，任何操作都由上述 3 个列表明确定义：前提条件列表 $P(a)$、添加列表 $A(a)$ 和删除列表 $D(a)$。表 5.1 总结了 move(x，y，z)的这些列表。请注意，所有参数 x、y 和 z 都是变量。这使得定义具有通用性。在实际执行的动作中，变量必须替换为常量。

表 5.1 通用操作的定义 move(x，y，z)

$a=\text{move}(x,y,z)$
$P(a)=\{\text{loc}(x,y),\text{clear}(x),\text{clear}(z)\}$
$A(a)=\{\text{loc}(x,z),\text{clear}(y)\}$
$D(a)=\{\text{loc}(x,y),\text{clear}(z)\}$

5.2.6 通用操作的实例化

表 5.1 中的定义是通用的，因为它采用了包含变量的一般形式。为了能将 move(x，y，z)应用到具体的状态，程序必须将变量 x、y 和 z 实例化为可用的常量 A、B、C 或表。

变量与常量的替换必须在定义泛型动作的 3 个列表中的每一个列表中一致进行。为了便于说明，下面给出一个实例化的例子：

$$b=\text{move}(\text{A},\text{B},\text{C})：$$
$$P(b)=\{\text{loc}(\text{A},\text{B}),\text{clear}(\text{A}),\text{clear}(\text{C})\}$$
$$A(b)=\{\text{loc}(\text{A},\text{C}),\text{clear}(\text{B})\}$$
$$D(b)=\{\text{loc}(\text{A},\text{B}),\text{clear}(\text{C})\}$$

注意，在计算机程序中实现实例化是很容易的。只需将每个变量替换为移动参数列表中相应位置的常量即可。在这个例子中，x 被替换为 A，y 被替换为 B，z 被替换为 C。

5.2.7 有多少个实例

从理论上讲，一个通用操作(如 move(x，y，z))代表了一整套实例。因此，对于可以替代 x 的 3 个方块(即 A、B、C)，就有 3 个 y 的实例(即剩下的两个方块加上表格)，以及两个由 z 代表的目的地(即剩下的最后一个方块加上表格)。这意味着实例总数为 $3\times3\times2=18$。尽管如此，也不能忘记，并非所有这些操作都处于给定的合法状态。

5.2.8 执行操作

在求解过程中，智能体必须决定在给定状态下可以采取哪些可用的行动。我们的积木世界只有一个通用动作，即 move(x，y，z)，我们已经看到它有 18 个实例。智能体会剔除那些当前状态不满足其前提条件的实例。如果 S 是表征给定状态的指令集，那么只有当 $P(a)\subseteq S$ 时，实例化的动作 a 才能被执行。

执行该操作意味着从 S 中删除 $D(a)$中列出的所有描述符，并将 $A(a)$中列出的所有描述符添加到 S 中。

5.2.9 示例

让我们来看看操作 $a=\text{move}(x,y,z)$，表 5.1 列出了以下前提条件：

$$P(a)=\{\text{loc}(x,y),\text{clear}(x),\text{clear}(z)\}$$

假设智能体面对的状态描述如下：

$$S=\{\text{loc}(\text{A},\text{table}),\text{loc}(\text{B},\text{table}),\text{loc}(\text{C},\text{A}),\text{clear}(\text{B}),\text{clear}(\text{C})\}$$

考虑两个实例：$a_1=\text{move}(\text{A},\text{B},\text{C})$和 $a_2=\text{move}(\text{C},\text{A},\text{B})$。

下面是它们的前提条件：

$$P(a_1)=\{\text{loc}(\text{A},\text{B}),\text{clear}(\text{A}),\text{clear}(\text{C})\}$$

$$P(a_2)=\{\text{loc}(\text{C},\text{A}),\text{clear}(\text{C}),\text{clear}(\text{B})\}$$

根据 $P(a)\subseteq S$ 的要求检查这些前提条件，可以很容易完成验证。这意味着 a_2 在 S 中是合法的，而 a_1 不合法。

控制问题

如果你在回答下列任何问题时遇到困难，那么请返回阅读前文的相应部分。

- 请解释 $P(a)$、$A(a)$和 $D(a)$这 3 个列表在什么意义上定义了动作 a。
- 什么是动作的实例化？计算机程序如何根据动作的通用定义创建实例？
- 计算机程序如何确定具体实例在给定状态下是合法的？$A(a)$和 $D(a)$这两个列表是如何用来修改状态描述的？

5.3 使用 STRIPS 进行规划

让我们来看看以 STRIPS(斯坦福研究所问题解决程序的缩写)著称的传奇人工智能程序。在此介绍它的原因不仅仅在于它的历史价值。相反，该程序展示了一种有趣而有用的搜索替代方案。

5.3.1 目标集

STRIPS 通过一组需要满足的目标来描述最终状态。例如，在如图 5.1 所示的场景中，STRIPS 将通过以下目标来定义最终状态：

$$Z=\{\text{loc}(\text{C},\text{A}),\text{loc}(\text{A},\text{table}),\text{loc}(\text{B},\text{table})\}$$

只有当所有目标都得到满足，即当前状态描述包含了所有这些描述符时，智能体才达到最终状态。如果用 S 表示描述一个状态的列表，那么如果 $Z\subseteq S$，智能体就会认为这个状态是最终状态。注意，目标列表不包含那些可以被认为是多余的描述符。

5.3.2 一般理念

STRIPS 是倒着进行的。它从最终状态(目标集)开始，然后试图回到初始状态，并记录在此过程中所采取的行动，原理如下：

首先，智能体必须考虑最终状态的要求在初始状态就已经满足的可能性。如果我们用 Z 表示目标列表，用 S_1 表示对初始状态的描述，那么 $Z\subseteq S_1$ 表示无须进一步分析。

在相反的情况下，即 $Z\nsubseteq S_1$，智能体试图找出可能导致最终状态的最后一个行动——一

个可能添加了 Z 中最后一个或多个缺失目标的行动。在上一段例子的特定情况下，最后一个行动可能添加了以下一个或多个目标：loc(A,C)、loc(A,table)或 loc(B,table)。

一旦找到了这样一个动作，STRIPS 就会问在什么状态下可能会执行最后一个动作，即搜索中的倒数第二个状态。倒数第二个状态的特征是目标集被修改了；这个目标集不包含最后一个行动所添加的目标。确定了这一点后，智能体(再次)询问这些目标是否已经存在于初始状态 S_1 中。如果是，则程序成功结束。如果没有，则递归重复同样的过程。

5.3.3 具体实例

让我们回到 5.1 节中的积木世界。对于这个特定配置，图 5.2 显示了达到最终状态前的情况。右边是最终状态。在最终状态的左边是 3 个可能的倒数第二个状态，它们允许执行 move(x,y,z)的实例，以添加最后一个缺失的目标。

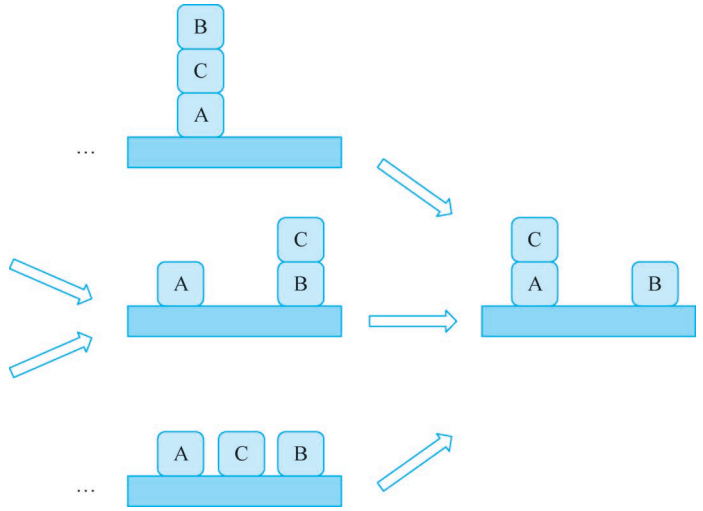

图 5.2 STRIPS 从目标开始向后退，总是试图找出可能在状态描述中增加某些目标的动作

下面是一种可能性。在最上面的配置(由 3 个立方体组成的立柱)中，move(B,C,table)将通过把方块 B 从立柱中移出并放到桌面上来达到最终状态，从而满足目标 loc(B,table)。图 5.2 中显示了另外两个倒数第二个状态，同样，它们也可以允许 move(x,y,z)的适当实例化，从而达到最终状态。

这些倒数第二个状态都与初始状态不同，因此智能体将继续倒推，试图找出哪些动作可能导致倒数第二个状态，以及它们之前的状态可能是什么样子。这个过程不断递归，当达到初始状态时就会停止。

5.3.4 如何确定行动

STRIPS 利用 5.2 节中介绍的列表 $A(a)$和 $D(a)$来识别可能完成最终状态的动作。具体来说，任何将最后一个缺失项添加到目标列表中的行动都必须满足以下两个条件：

$$A(a) \cap Z = \varnothing \tag{5.8}$$

$$D(a) \cap Z = \varnothing \tag{5.9}$$

第一个条件是确保所选的操作至少增加了一个 Z 中列出的目标。第二个条件是确保

这个操作不会删除任何之前可能已经满足的目标。STRIPS 会查看所有可能的可用操作实例，然后选择满足这两个条件的操作。

5.3.5 倒数第二个状态是什么样的

假设智能体已经确定了一个满足 5.3.4 节所述两个条件的行动。那么，这可能就是最终状态序列中的最后一个行动。但是，最后一个动作所对应的状态是什么样子呢？

我们知道，这个动作只能在满足这个动作的前提条件 $P(a)$ 的状态下执行。例如，如果最后的操作是 move(B,C,table)，那么前一个状态的描述必须包含 loc(B,C)和 clear(B)(见图 5.2)。另外，描述不必包含操作 a 增加的描述符，即 $A(a)$ 指定的描述符。这些被添加的描述符中的一些可能也是定义最终状态的目标。

让我们用一组目标来描述倒数第二个状态，其方式与最终状态的定义类似。将前面的考虑形式化，我们会发现倒数第二个状态应该满足如下目标：

$$Z' = \{Z \backslash A(a)\} \cup P(a) \tag{5.10}$$

符号“\”代表“集合减去”。例如，{A,D,F}{A,E}={D,F}：这个运算从第一个集合中删除了第二个集合中的所有元素；在这个特殊的例子中，被删除的元素是 A。

式(5.10)告诉我们，新的目标集 Z' 是通过从 Z 中删除所有将由操作 a 交付的描述符，然后添加那些作为 a 的先决条件的描述符而得到的。

5.3.6 STRIPS 的伪代码

表 5.2 提供了简化版 STRIPS 的伪代码。读者可以回顾前面段落中描述的步骤。开始时，算法确定的操作列表 L_A 是空的。每次递归都会在列表的开头添加一个行动，以实现最终状态的某些目标。一旦后向搜索到达初始状态，行动列表 L_A 中的行动就会正确排序。

表 5.2 简化版 STRIPS 的伪代码

输入：列表，Z，在最终状态下要满足的目标列表。
初始状态的描述，S_I。
可用的操作及其实例化。
空的行动清单，L_A。
strips(S_I,Z,L_A)。
1. 若 $Z \subseteq S_I$，输出成功，返回 L_A。
2. 找出满足下列条件的每个实例 a：
$A(a) \cap Z \neq \varnothing$，$D(a) \cap Z = \varnothing$
若没有 a 满足上述条件，则以失败告终。
3. 对于满足条件的每个实例：
i. 用一个新目标列表来描述之前一个状态：
$Z' = \{Z \backslash A(a)\} \cup P(a)$
ii. 将 a 放在 L_A 的开头，递归调用 strips(S_I,Z',L_A)。

表 5.2 中伪代码的第 4 行针对的是找不到正确的行动序列，搜索不得不以失败告终的情况。

控制问题

如果你在回答下列任何问题时遇到困难，那么请返回阅读前文的相应部分。

- 解释 STRIPS 的一般程序。目标列表是什么？我们说 STRIPS 是逆向搜索是什么意思？
- 要实现某些最终状态目标的行动必须满足哪些条件？
- 关于倒数第二个状态，也就是执行最终操作前的状态，我们能说些什么呢？
- 这个状态必须包含哪些描述符？
- STRIPS 程序何时停止？

5.4 数值举例

初读 STRIPS 算法时，读者通常会感到困惑。让我们通过一个详细的数值举例来澄清一些不太明显的细节。

5.4.1 应该考虑哪些操作

让我们回到如图 5.2 所示的最终状态。这里要实现的目标归纳如下[①]：

$$Z=\{\text{loc(C,A)},\text{loc(A,table)},\text{loc(B,table)}\} \tag{5.11}$$

假设唯一可用的操作是表 5.1 中定义的 move(x,y,z)。开始时，智能体创建该操作的所有实例，这也意味着要为每个实例指定集合 $P(a_i)$、$A(a_i)$和 $D(a_i)$。下一步将确定满足式(5.8)和式(5.9)的实例。

读者可以很容易地验证，对于式(5.11)中的目标，以下 3 个实例化都满足这些条件。看一下图 5.2 读者就会相信，这些确实是在达到最终状态之前可能执行的最后动作。

1. a_1=move(B,C,table)
2. a_2=move(C,B,A)
3. a_3=move(C,table,A)

5.4.2 检查列表

下面是检查第一个操作 a_1=move(B,C,table)是否符合条件的细节，以作说明。表 5.1 列出了该操作的通用版本的添加列表和删除列表。要进行实例化，程序只需将变量 x、y 和 z 分别替换为动作 a_1 中的常量 B、C 和 table。需要注意的是，要确保按照正确的参数顺序进行替换。这就产生了下面一对列表：

$$A(a_1)=\{\text{loc(B,table)},\text{clear(C)}\}$$

$$D(a_1)=\{\text{loc(B,C)},\text{clear(table)}\}$$

同样，如果表格足够大，那么可以在 $D(a_1)$中省略 clear(table)。此外，桌子永远不会被移动，这意味着在任何移动命令中都不会调查它的清除顶部信息。

我们可以很容易地验证，$A(a_1)\cap Z=\{\text{loc(B,桌子)}\}$。这告诉我们，动作 a_1 应用于某个尚未知晓的倒数第二个状态时，能够交付 loc(B,table)，即 Z 中的一个项目、$D(a_i)\cap Z=\varnothing$，保证了 a_1 不会从 Z 中删除任何可能已经是倒数第二个状态的目标。

总之，a_1 已被证实是“未知”序列中能够将初始状态转换为最终状态的最后一个行动的

① 关于清除顶部的信息是多余的(它只在前置条件集中重要)。这也是 Z 忽略 clear(C)等描述符的原因。

理想候选项。

作为一个简单的练习，建议读者重复同样的步骤来验证 a_2 和 a_3 是否也满足最终动作的条件。

5.4.3 注意事项

人类很容易就能发现这些动作。由于我们的洞察力和对问题的理解，我们会立即“看到”图片中的正确操作，我们的分析是在潜意识中进行的。

计算机缺乏这种洞察力。它们只能运行指示它们执行必要算法的程序。在这里处理的具体案例中，STRIPS 会研究每个动作 a_i 是否满足 $A(a_i) \cap Z = \varnothing$ 和 $D(a_i) \cap Z = \varnothing$ 的要求。

如果符合要求，则可将该操作标记为候选操作；如果不符合要求，则忽略该操作。与人类不同的是，计算机是机械地完成这一切的，没有任何意识或意图。

5.4.4 描述前一个状态

动作 a_1 已被确定为可能导致最终状态的操作之一。然而，该操作只能在满足操作前提条件 $P(a_1)$ 的状态下执行。在 STRIPS 中，这个前一个状态（倒数第二个状态）的特征是一个目标列表。

回顾一下，$A(a_1) = \{\text{clear(C)}, \text{loc(B, table)}\}$，$P(a_1) = \{\text{clear(B)}, \text{loc(B, C)}\}$，$Z = \{\text{loc(C, A)}, \text{loc(A, table)}, \text{loc(B, table)}\}$。式(5.10)指定了定义最终状态的目标。可以看到，$Z \backslash A(a_1)$ 从 Z 中删除了描述符 loc(B, table)，我们很容易建立倒数第二个状态的目标列表：

$$\begin{aligned} Z' &= \{Z \backslash A(a)\} \cup P(a) \\ &= \{\text{loc(C, A)}, \text{loc(A, table)}\} \cup \{\text{clear(B)}, \text{loc(B, C)}\} \\ &= \{\text{loc(C, A)}, \text{loc(A, table)}, \text{loc(B, C)}, \text{clear(B)}\} \end{aligned}$$

注意，Z' 描述的是图 5.2 中 3 个可能的倒数第二个状态中最顶端的状态：3 个方块组成的列。

5.4.5 迭代过程

前面几段说明了如何创建由 Z' 中所列目标定义的倒数第二个状态。下一步，智能体将检查这些目标是否已经在初始状态中得到满足。如果是，程序就成功停止；如果不是，就递归地重复同样的程序，Z' 现在成为下一个“最终状态”。当然，程序员还必须考虑从初始状态永远无法达到最终状态的可能性。因此必须制定相应的停止标准。

控制问题

如果你在回答下列任何问题时遇到困难，那么请返回阅读前文的相应部分。

- 计算机的规划方法与人类的规划方法有何不同？
- 动手模拟 STRIPS 解决图 5.1 中的问题。作为一个简单的练习，不仅要对动作 a_1 进行模拟，还要对后面两个动作中的至少一个进行模拟。

5.5 人工智能规划的高级应用

在玩具积木上展示 STRIPS 程序的精髓，是因为这个场景非常简单易懂。然而，它的简单性可能回避了该方法的一些现实挑战。此外，琐碎的场景有时会让读者提出一个合理的问题："这一切有什么用？"为了提供一个更广阔的图景，本节将介绍并简要讨论一些更高级的规划应用。

由于这些应用具有某些共同特点，因此可以将它们分成几组，每组由专家知道如何处理的模型来代表。在这些模型中，最受欢迎的是旅行推销员、背包问题和工作车间调度。当然，一章的篇幅不足以详细介绍它们的解决方案——尽管读者现在应该已经能够使用搜索技术来解决它们。此外，最近的经验表明，第 8 章将介绍的群体智能领域的技术能更有效地解决这个问题。因此，在这里我们将仅限于简要概述每个模型应用的性质。

5.5.1 旅行推销员问题

其原理如图 5.3 所示。向推销员提供了一组城市和任意两个城市之间的距离表。推销员的任务是找出访问所有城市的顺序，同时尽量减少所走的总路程。

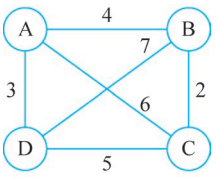

图 5.3 旅行推销员问题。节点是城市，边是连接，整数是距离。推销员希望找到穿越所有城市的最短路径。如果有几十个或更多的城市，计算难度就会很大

从图 5.3 中可以看出，如果推销员按照 ABCDA 的顺序走，总路程为 4＋2＋5＋3＝14。如果顺序是 ACDBA，则总路程为 6＋5＋7＋4＝22。那么其他路线呢？如果只有 4 个城市，那么列举所有城市是可行的。但如果有很多城市，那么排列组合的数量就会增加，超出所有实用的界限；机械的数字运算注定要失败。

在人工智能以及一些更传统的领域中，已经研究出了许多应对这一挑战的方法，其中一些非常有效，另一些则不那么有效。在现阶段，读者不难提出一种基于盲搜索的解决方案。

5.5.2 包裹投递和数据包路由

旅行推销员提供了一个通用模型，用于表示一家将许多包裹送往许多目的地的公司的需求。对于每次单独的旅行，司机都会收到一份地址列表，需要在尽可能短的时间内用最少的燃料到达目的地。每次行程都不同，计算机必须为每次行程找到最佳路径，而且必须能够在短时间内完成。在突然发生变化的情况下（例如，当新地址出现或某个现有地址被删除时），就需要立即更新。

程序必须符合特定领域的特殊性。假设使用的是自行车而不是汽车，地点 X 位于山谷，而 Y 位于山上，那么从 X 到 Y 的距离就会被认为比相反方向的距离要长。此外，从一个城市到达另一个城市所需的时间不一定只取决于距离，道路质量和交通密度也起着作用，后者会随着时间的推移而发生变化。

互联网路由器也面临类似的问题。每次数据包到达后，路由器都要决定将其转发到何处。这需要考虑多种标准，而优化工作远非易事。

5.5.3 救护车路由

另一个有趣的应用是优化救护车的调度方式。每时每刻，许多车辆都会出现在不同的地点，有些有病人，有些没有病人。有病人的救护车可能离急诊室（ER）很近，也可能需要较长时间才能到达。最近的急诊室可能超负荷工作（可能造成行政延误），而稍远一点的急诊室可能有很多空余能力。此外，医院的可用设备也可能不同。

一旦接到呼叫，就必须决定使用哪辆车，以及将病人送往哪个急诊室或医院。首要目标是尽量缩短时间，但也必须考虑上述所有限制因素。从这个意义上说，救护运输版的旅行推销员问题相当复杂，没有简单的解决方案。

5.5.4 背包问题

考虑一组物品，每件物品都有一定的重量和价格。你想把尽可能多的物品装进一个大小有限的背包里。或者，背包只能减轻一定的最大重量，你不可能把所有东西都装进去，因为这样所有物品的总重量就会超过限制。无论哪种情况，都需要做出一些妥协。

人工智能规划的任务就是找出一组物品，使背包中携带物品的总价值最大化，同时又不违反物品总重量或尺寸的限制。

5.5.5 工作车间调度

图 5.4 展示了另一种模型应用的原理。一家制造商需要完成 X、Y、Z 和 W 四项工作，每项工作都需要完成某些任务，这些任务必须按照预先规定的顺序执行。例如，图 5.4 中左上角的工作 X 由 A～E 五项任务组成，流程图告诉我们，A、B 和 C 必须按此具体顺序执行，而其余两项任务 D 和 E 可以独立于 A、B 和 C 执行（与它们并行），但只有在 D 完成后才能开始执行 E。其余 3 个流程图说明了其他 3 项工作所涉及的约束条件。

工作将在 3 台机器 M1、M2 和 M3 上进行，每台机器只能处理部分任务。例如，图 5.4 中显示 M1 可以执行任务 A、B 和 C，但任务 D、E 和 F 必须在其他地方执行：任务 F 在 M2 上执行，任务 E 要么在 M2 上执行，要么在 M3 上执行。每台机器一次只能执行一项任务。调度员要将任务分配给机器，以保证在最短时间内完成任务，同时尽量减少机器的空闲时间。

上述情况称为作业车间调度。在最简单的情况下，每项工作都由一组预定义的任务组成，这些任务可以按照任意顺序执行（没有指定图 5.4 这样的工作流程）。然而，具体的环境和限制条件会使流程图复杂化，以至于问题变得几乎无法解决。即使无法达到绝对的理想状态，工业厂房也至少需要找到一个“相当好”的解决方案。

5.5.6 注意事项

在人工智能发展的早期，许多专家认为，所有这些问题都可以通过经典的规划算法来解决，它们是启发式或盲目搜索技术的理想目标。今天，我们知道了更多。其他方法（其中一些不属于人工智能规划范畴）可以用计算效率高得多的方式来处理这些问题。传统的搜索技术在这里很有用，但未必是最佳选择。

除了经典搜索，学者还开发了一整套现代技术，即群体智能算法。它们在规划和调度领

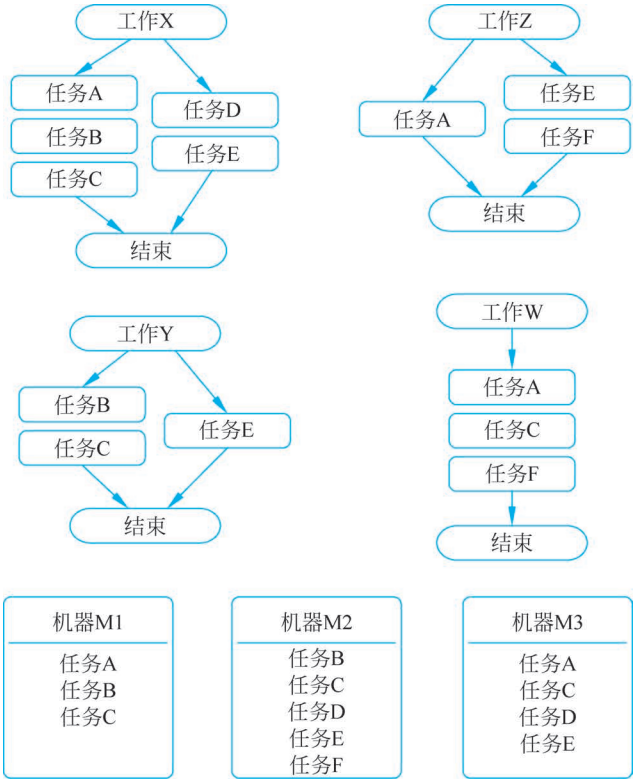

图 5.4 目标是将每项工作所需的任务分配给 3 台机器，以尽量减少总体时间和机器的闲置时间

域的效率非常显著。至少，它们已被证明比 STRIPS 快得多。而且，它们很容易实现。

难怪这项新技术很快就变得如此时髦，以至于传统的搜索技术几乎显得过时了。我们将在第 8 章用整章篇幅来介绍它。

5.5.7 重要评论

回到基于搜索的经典方法，读者现在明白了，如果系统有机会从某种背景知识中获益，那么即使在 STRIPS 中，求解过程也会大大加快。例如，积木世界问题可以依赖以下规则："先放下那些在最终状态下会放在桌面上的积木"，或者"应推迟移动那些在最终状态下位于顶部的积木"。人类解决问题时会下意识地利用许多这样的规则。因此，探索在计算机程序中实现这种推理方式的方法也很有意义。

这种做法属于知识型系统的范畴，具体将从第 9 章开始介绍。

控制问题

如果你在回答下列任何问题时遇到任何困难，那么请返回阅读前文的相应部分。

- 解释推销员问题(TSP)的原理。讨论它的一些变式。
- 举例说明 TSP 在现实世界中的应用，并指出它们偏离 TSP 基本原理的方式。
- 什么是工作车间调度？它的哪些特点使其特别难以实现？这个问题与 TSP 有什么联系，又有什么不同？讨论它的一些变形。

5.6 熟能生巧

为了加深理解，不妨尝试以下的练习、思考题和计算机作业。

- 考虑图 5.5 中的域。假设你想基于 STRIPS 编写一个规划程序，从这些图块中创建各种手工艺品，如柱子、拱门或门。有哪些描述符可以帮助你将问题的状态具体化？如何定义动作？请注意，在这个更一般的设置中，旋转图块或使其位置从水平变为垂直或从垂直变为水平都需要一些操作。

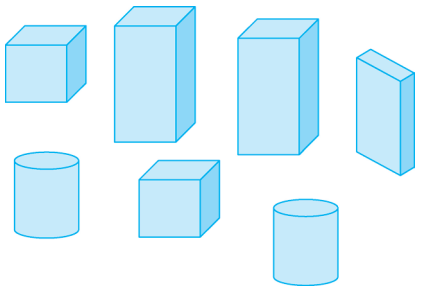

图 5.5 供练习的更高级的积木世界问题

- 旅行推销员和作业车间调度的主要区别在于，后者还需要考虑时间关系。例如，工作任务可能必须按照特定的顺序一个接一个地执行，而且每个任务可能都需要一定的时间才能完成。如何修改 STRIPS 技术，使其能够满足这一更广泛的要求？
- 提出一些可以用 TSP 模拟的现实应用（但不是本章提到的那些）。将如何用 STRIPS 技术解决它们？
- 提出可以用背包问题或工作车间调度来模拟的现实应用。
- 编写一个实现 STRIPS 技术的计算机程序，使其能够处理简单的积木世界。用不同的初始状态和最终状态进行实验，并评估这些实验的计算成本。

5.7 结语

规划是一门成熟的学科，有许多成功的案例。第一个重要里程碑 STRIPS 是由 Fikes 和 Nilsson（1971 年）开发的。这里之所以收录他们的技术，是因为它以非常令人信服的方式证明，人们确实可以通过经典搜索来处理实际工程问题。同时，STRIPS 的经验也让读者进一步认识到，必须为问题的状态找到一个好的表示方法。

规划的主要应用包括旅行推销员、背包问题和工作车间调度等通用任务——本章对所有这些任务都做了简要介绍。有关旅行推销员的更多信息，请参见 Applegateetal；有关背包问题的信息，请参见 MartelloandToth（1990 年）；有关作业车间调度的精彩介绍，请参见 Chakraborty（2009 年）。

这些吸引人的问题不仅在人工智能领域，在计算机科学和应用数学的其他领域都十分有用。它们之所以如此受欢迎，是因为它们可以被证明是许多实际应用的代表。读者将从具体问题映射到通用范式的技能训练中获益匪浅——当然，前提是他们知道如何处理这些范式。

另外，经典的人工智能规划方法已不再像前些年那样强大，第 8 章将讨论的群体智能技术被反复证明更为有效。实际上，这就是本章只介绍了一个著名的代表——STRIPS，而忽略了 20 世纪开发的许多替代技术的原因——群体智能已经在很大程度上取代了它们。