arXiv:2603.19429v1 [cs.AI] 19 Mar 2026

当实例化与不实例化都不理想时——将规划编码为可满足性的部分基化编码

Jo˜ao Filipe[1],**Gregor Behnke**[1]

[1]阿姆斯特丹大学、逻辑、语言与计算研究所、荷兰 j.sa@uva.nl，g.behnke@uva.nl

摘要

经典规划问题通常使用提升的一阶表示来定义，这种表示具有紧凑性和通用性。尽管大多数规划器会将这些表示实例化以简化推理，但这可能导致规模呈指数级增长。近年来，一些方法转而直接在提升层级上运行，以避免完全实例化。

我们通过引入三种 SAT 编码，在完全提升与完全基化的规划之间探索折中方案：保持动作处于提升状态，同时对谓词进行部分实例化。不同于以往随计划长度呈二次增长的 SAT 编码，我们的方法呈线性增长，从而能够在更长的计划上实现更好的性能。在实证评估中，我们最优的编码在难以实例化的领域上的长度最优规划方面，优于最先进水平。

引言

自动规划旨在寻找一个动作序列，将初始状态转换为满足给定目标的状态。过去几十年的大多数研究都集中于求解能够完全实例化的规划问题，而提升式规划任务——即使用无函数符号的一阶语言描述动作的规划——自 Corrêa 等人（2020）的开创性工作以来取得了显著进展。借助提升式规划，人们得以成功处理规模远超以往的规划问题。

当前的提升式规划方法主要分为两类：基于搜索的方法和基于向逻辑的转换的方法。本文聚焦于后者。

Shaik 和 van de Pol（2022）提出了将提升式规划转换为量化布尔公式（QBF）的方法，其中通过对可能作为谓词论元的对象进行迭代，使用全称量化来编码提升。同时，H¨oller 和 Behnke（2022）提出了一种转换为布尔可满足性（SAT）的编码方法——称为 LiSAT——该方法目前是计划长度最优规划的最先进方法，对于单位代价问题而言也就是最优规划。与实例化的 SAT 规划器类似，它将计划视为一系列时间步，并生成 SAT 公式 ϕ_ℓ；当且仅当存在长度不超过 ℓ 的计划时，这些公式才是可满足的。为了生成最优计划，它会不断迭代 ℓ，直到找到计划为止。LiSAT 使用变量 a^t 和 $(v = o)^t$ 来表示：动作 a 在时间 t 执行，并且该动作的论元变量 v分别指代对象 o。为了编码动作的前置条件得到满足以及效果得以应用，LiSAT 使用了因果链接的概念（Penberthy 和 Weld，1992），将以某个事实为前置条件的动作与使该事实为真的动作连接起来。因此，ϕ_ℓ 的规模随 ℓ 呈二次增长，这并不理想。

我们探讨基于 SAT 的规划中完全实例化与完全提升之间的折中方案。为此，我们提出三种编码，它们只要求对规划问题进行部分实例化。在这三种编码中，动作均采用完全提升式编码。对于状态描述，即问题中的谓词，我们研究了完全实例化方法，以及另外两种用提升表示当前状态的方法；后两种方法会显式编码谓词及其论元对象。对于后两种方法，我们采用提升互斥组（Helmert，2009），从而能够紧凑地提升式编码问题中的部分谓词。在此情形下，我们提出了两种对象编码方式：独热编码和二进制编码。

尽管完全实例化方法无法与其他基于可满足性的方法竞争，但我们的新型部分基化编码在九个基准领域中的五个领域优于 LiSAT。

预备知识

提升式规划 我们将提升式 STRIPS 规划问题（Fikes 和 Nilsson，1971）形式化为一个元组：$\Pi = \langle \mathcal{O}, \mathcal{P}, \mathcal{A}, \mathcal{I}, \mathcal{G} \rangle$。其中，$\mathcal{O}$ 是对象集合，$\mathcal{P}$ 是谓词符号集合，$\mathcal{A}$ 是动作符号集合，$\mathcal{I}$ 是初始状态，$\mathcal{G}$ 是目标条件。每个对象 $o \in \mathcal{O}$ 都有一个类型 t_o。某类型的所有对象集合记为 $\mathcal{O}^t$。类似地，对于变量 v，我们用 $\mathcal{O}^v$ 表示属于 v类型的所有对象。通常，类型通过类型层次结构定义，其中一种类型 t_1 是另一种类型 t_2的子类型。我们通过将 $\mathcal{O}^{t_1}$ 的所有对象添加到 $\mathcal{O}^{t_2}$中来扁平化这一层次结构。我们假设类型层次结构是一棵树（基准实例中始终如此）。每个谓词符号 $p^n \in \mathcal{P}$都有一个元数 n和一个带类型变量的元组：$p^n(v_1^{t_1}, \ldots, v_n^{t_n})$，其中 $v_i^{t_i}$表示变量 v_i 属于类型 t_{i_o}。我们将谓词与具体选定的论元（可以是变量或对象）$p^n(a_1, \ldots, a_n)$合称为一个原子。当所有变量都替换为相应类型的对象后

本文为一篇论文的扩展版本，版权归 © 2026 美国人工智能促进协会（www.aaai.org）所有。保留所有权利。扩展内容采用 CC BY-SA 许可。

，该谓词就被视为已接地，此时称为事实。与谓词符号类似，每个动作符号 $a^n \in \mathcal{A}$都有一个元数 n和一个带类型变量的元组；当所有变量都替换为相应类型的对象后，该动作符号就被视为已接地。为描述动作的语义，我们定义三个函数： $prec$、 add和 del。它们分别将动作映射到其前置条件、添加效果和删除效果，这些条件和效果都是以动作的论元作为变量的原子集合。最后， $\mathcal{I}$ 和 $\mathcal{G}$ 是事实集合，分别表示初始状态和目标条件。

定义 1。（适用性）。基化动作 $a(o_1, \ldots, o_n)$在状态 s 中适用，当且仅当 $prec(a(o_1, \ldots, o_n)) \subseteq s$。

定义 2。（状态转移）。给定状态 s 和一个适用的基化动作 a，转移函数 $\tau(s, a(o_1, \ldots, o_n)) = (s \backslash del(a(o_1, \ldots, o_n))) \cup add(a(o_1, \ldots, o_n))$ 给出结果状态。

定义 3。（解）。解是一个基化动作序列 $\pi = (a_0(o_1^1, \ldots, o_{n_1}^1), \ldots, a_m(o_1^m, \ldots, o_{n_m}^m))$，满足 a_i 在 s_i中适用，$s_{i+1} = \tau(s_i, a_i(o_1^i, \ldots, o_{n_i}^i))$，$s_0 = \mathcal{I}$和 $\mathcal{G} \subseteq s_{m+1}$

定义 4。（最优长度）。若解 $\pi = (a_1, \ldots, a_n)$ 的长度不大于任何其他解 $\pi' = (b_1, \ldots, b_m)$，即满足 $n \leq m$，则称其为长度最优解。

提升互斥组 在规划问题中，并非所有语法上有效的状态都能从 $\mathcal{I}$到达。相反，某些不变量对所有通过执行动作可达的状态都成立。互斥组就是这类不变量之一：它们表明，在一组事实中，任何可达状态下至多有一个事实为真。例如，一个包裹不可能同时位于两个不同的位置。因此，诸如 $at(package1, lisbon)$ 和 $at(package1, amsterdam)$ 这样的事实构成一个互斥组。

在实践中，将互斥组定义、计算和使用为事实集合存在问题，因为互斥组通常数量众多且结构复杂。取而代之，可以将范围限制在提升互斥组（LMG；Helmert 2009）。提升互斥组使用变量描述，因此其每个基化实例化都是一个互斥组。无需对所有事实和动作进行基化实例化，即可获得 LMG（Helmert 2009；Fišer 2020），但这些过程通常无法找到所有 LMG，只能找到其中一个（在实践中足够用的）子集。在我们的实验中，我们采用了 Fišer（2020）提出的基于提升事实变更互斥组的发现方法，该方法能够快速生成大量 LMG。

现在，我们通过定义可能成为 LMG 的候选结构，来定义 LMG 的语法。

定义5。 提升互斥组（*LMG*）候选项 $\mathcal{L}$ 是一个三元组 $\langle$*fix, cnt* $P\rangle$，其中：*(1) fix* 是固定变量的集合；*(2) cnt* 是计数变量的集合，且 *fix* $\cap$*cnt* $= \emptyset$；*(3)* P 是一个原子集合，其变量来自 *fix* $\cup$*cnt*。

从概念上说，固定变量 fix的每种实例化都应产生一个互斥组。该互斥组中的事实通过计算计数变量 cnt的所有基实例化得到。

例如，我们可能有 $\mathcal{L} = \langle \{?p\}, \{?l\}, \{at(?p, ?l)\} \rangle$。每个……的实例化将固定变量 $?p$ 与对象 x 结合起来，会得到一个互斥组，其元素是形式为 $at(x, y)$ 的事实——其中 y 是变量 $?l$ 的一个可能对象。

本文中，我们经常使用部分实例化的 LMG，并将其称为部分提升互斥组（PLMG）。在这类互斥组中，固定变量已经完成实例化。

定义6。 部分提升互斥组（*PLMG*）候选项 $\mathcal{M}$ 是一个二元组 $\langle cnt, P \rangle$，其中： cnt 是计数变量的集合， P 是一个原子集合，其变量来自 cnt。我们用 $\mathfrak{C}(\mathcal{M})$ 表示 *PLMG* 的计数变量，用 $\mathfrak{L}(\mathcal{M})$表示其原子。

在 LMG 和 PLMG 中，所有变量 v 都与类型 $\mathcal{O}^v$相关联。给定一个 LMG 以及固定变量 I 的一个实例： $fix \to O$，将固定变量 v 的所有出现替换为 $I(v)$，即可得到一个 PLMG。通过针对计数变量 cnt计算其原子的所有基化实例化，一个 PLMG 蕴含一个单独的互斥组 $\mathfrak{M}$ 。我们将这组基化实例化事实写作 $F(\mathcal{M})$。

定义7。 *PLMG* 候选 $\mathcal{M}$ 是 *PLMG*，当且仅当 $F(\mathcal{M})$是一个互斥组，也就是说：对于通过执行动作可达的每个状态，至多一个事实 $f \in F(\mathcal{M})$ 为真。*LMG* 候选是 *LMG*，当且仅当其固定变量的每个实例化都会产生一个 *PLMG*。

互斥组只要求在每个可达状态中，组内至多一个事实为真。在许多情况下，组内还会有至少一个 事实为真。我们将此类互斥组、LMG 和 PLMG 称为恰好一个互斥组、LMG 和 PLMG。我们采用的基于提升事实变更互斥组的 LMG推断机制（Fišer 2020），会确定每个 LMG 是恰好一个 LMG 还是至多一个 LMG。这一机制自然也适用于生成的 PLMG。

示例 *1*。考虑运输领域。该领域的一个 *LMG* 是：$\langle$*{?p}, {?l, ?v}, {at(?p,?l) in(?p,?v)}*$\rangle$。由此，我们可以生成与包裹数量相同的 *PLMG*。对于每个包裹 p，我们随后得到一个 *PLMG*组 $\mathcal{M}$ *with*$\mathfrak{C}(\mathcal{M}) = \{?l, ?v\}$ *and* $\mathfrak{L}(\mathcal{M}) = \{at(p, ?l)\ in(p, ?v)\}$。

布尔可满足性 布尔可满足性（SAT）是指确定合取范式（CNF）中的命题公式是否存在一种变量真值赋值，使该公式为真。CNF 公式由子句的合取（$\wedge$）构成，每个子句都是文字的析取（$\vee$）。文字是一个命题变量（简称变量），x 或其否定 $\neg x$。SAT 求解器随手可得，它们可以确定公式是否可满足；若可满足，则会给出一个满足该公式的真值赋值。为简化记法，我们不会以 CNF 形式呈现 SAT 公式，但所有公式都可以轻松转换为 CNF。

相关工作

自 Kautz 和 Selman（1992）的研究以来，基于 SAT 的规划一直是一项重要技术。大多数方法遵循一个共同思路：生成 SAT公式 ϕ_ℓ，使其可满足当且仅当存在长度为 $\leq \ell$ 的计划。随后将这些公式交给 SAT 求解器

或者采用增量方式（用于最优规划），或者按照某种调度（例如，满意化规划中的（Rintanen 2012, 2011））进行求解。找到一个可满足的 ϕ_i 公式后，即可提取计划。此后开发了多个 SAT 规划系统，包括 SAT-PLAN04（Kautz、Selman 和 Hoffmann 2006）、Madagascar（Rintanen 2012）、Aquaplanning [1]，和 SASE（Huang、Chen 和 Zhang 2012），它们都依赖完全实例化的表示。虽然实例化简化了推理，但在复杂且规模庞大的领域中会带来可扩展性问题。

为缓解这一问题，研究者提出了无需完全实例化的替代方法。早期方法包括由 Kautz、McAllester 和 Selman（1996）以及 Ernst、Millstein 和 Weld（1997）提出的算子拆分。Kautz、McAllester 和 Selman 的简单拆分为每个提升动作分配一组独立的参数变量，而 Ernst、Millstein 和 Weld 则提出了一种重载版本，让所有动作共享同一组参数变量。随后，Robinson 等人（2009）通过对论元分组，将这一思路扩展为支持动作并行。不过，正如 Kautz、McAllester 和 Selman（1996）所主张的，这些方法都对谓词和状态进行了完全实例化。

LiSAT（H¨oller 和 Behnke，2022）受计划空间规划启发，采用完全提升且无状态的编码，从而避免了这一局限。它不跟踪状态随时间的变化，而是确保每个动作的前置条件均由先前的动作（或初始状态）满足，且不会被中间执行的动作所破坏。这样便无需对事实进行实例化。

值得注意的是，LiSAT 的编码会导致公式大小随计划长度 L 呈二次增长，因为每个前提条件都必须与任意更早步骤中可能实现它的动作相关联。这限制了 LiSAT 在需要长计划的问题中的可扩展性，尽管它在长度最优设置下性能出色。LiSAT 也无法利用问题固有的结构，例如 LMG——而这可能会显著增加推理开销。

动作的提升式表示

Espasa 等人（2021）提出了一种处理动作论元的新方法，LiSAT 也采用了这一方法，即一种新的算子拆分方式——统一参数。这种方法与 Ernst、Millstein 和 Weld（1997年）的重载拆分类似，都允许不同动作共享参数变量。不过，它并非简单地按论元索引进行共享，而是考虑论元的类型，只共享类型相同的论元，无论它们在动作的论元列表中处于什么位置。这对于不同类型所包含的对象数量差异很大的领域尤其有用。对于每种类型，计算所有动作中的论元总数，并取最大值作为该类型所需的占位符数量。这一过程消除了不同动作间论元顺序的差异，因为每个占位符都唯一对应一种类型。这样一来，公式便不再依赖各个动作中特定的论元顺序。

示例 2. 考虑运输领域中的动作 $drive(?v-vehicle, ?l1-location, ?l2-location)$, $drop(?v-vehicle, ?l-location, ?p-package)$ 和 $pickup(?v-vehicle, ?l-location, ?p-package)$。

[1]https://github.com/domschrei/aquaplanning

驾驶	?v	?l	?l	
drop	?v	?l		?p
拾取	?v	?l		?p
UA	车辆	位置	位置	包裹

表1：统一参数

此示例涉及三种不同的类型：车辆 *(v)*、位置 *(l)* 和包裹 *(p)*。令 $num_{type}(action)$ 表示某个动作所采用的特定类型论元的数量。各动作中每种类型出现的总次数总结如下：

- $num_v(drive) = num_v(drop) = num_v(pickup) = 1$
- $num_l(drive) = 2$, $num_l(drop) = num_l(pickup) = 1$
- $num_p(drive) = 0$, $num_p(drop) = num_p(pickup) = 1$

最终得到的统一参数包含四个占位符：一个用于车辆，两个用于位置，一个用于包裹。表*1*更清楚地展示了所有动作的论元与最终统一参数之间的对应关系。

由于我们将基于 LiSAT 的概念（包括其统一参数）构建自己的编码，因此简要回顾 LiSAT 公式中与我们的工作相关且会予以保留的部分。为了对动作及其论元进行编码，H¨oller 和 Behnke（2022）引入了两类变量：

- a^t：表示动作 a 是否在时间步 t 执行。
- $(v = o)^t$：表示对象 o 在时间步 t 是否被统一参数 v 选中。

我们用 $\mathcal{A}^t$ 表示时间步 t 的所有 a^t 变量的集合，并用 UA_v^t 表示时间 t 统一参数 v 的所有 $(v = o)^t$ 变量的集合。我们用 U 表示所有统一参数的集合。最后，令 $args$ 为一个函数，它将动作映射到该动作所包含的统一参数集合。我们需要确保每个时间步 t 至多执行一个动作，且其论元已正确选定。为此，我们使用 $atMostOne$ 宏；给定一组决策变量，该宏会生成确保至多一个变量为真的子句，从而得到以下约束：$atMostOne(\mathcal{A}^t)$、$atMostOne(\text{UA}_v^t)$ 和 $a^t \implies \bigvee_{x \in \text{统一参数}_v^t} x$。可满足性约束的编码方式取决于变量数量。对于不超过 256 个变量的情况，使用成对二次编码；否则使用二进制计数器编码（Sinz 2005）。我们保留 H¨oller 和 Behnke（2022）所述的所有静态前提条件相关约束，为简洁起见不再赘述。

我们还保留约束 $a^t \Longrightarrow \bigvee_{\forall b \in \mathcal{A}} b^{t-1}$，通过强制计划开头包含动作的时间步连续排列，减少公式中的对称性。

命题逻辑中的部分实例化

我们提出了三种基于 SAT 的编码，通过采用部分实例化，消除了 LiSAT（H¨oller 和 Behnke 2022）中存在的二次扩展。LiSAT 公式大小呈二次增长的主要原因在于需要对因果链接进行编码，而这又是由于没有显式跟踪状态

所导致的。相反，我们将显式编码状态，从而不再需要对因果链接进行编码，并确保我们的公式呈线性增长。我们的公式总体形式为 $F = F_{\mathcal{I}} \wedge F_{\mathcal{G}} \wedge \bigwedge_{t=0}^{\ell} \tau(t,\ t+1)$，其中 $\tau(t, t+1)$ 对从时间步 t 到时间步 $t+1$的状态转移进行编码。在本文其余部分介绍的编码中，均指定了 $\tau(t,\ t+1)$。由于其大小不依赖于时间步的数量，$|\tau(t,\ t+1)|$ 相对于 ℓ 是一个常数，并且仅随 $|\Pi|$扩展。因此，F 随 ℓ线性扩展。

我们总体采用的方法是始终对动作保持完全提升，同时对状态进行完全或部分实例化（具体取决于编码方式）。这种区分是合理的，因为谓词的元数通常不超过其所属动作的元数——这意味着谓词以及状态描述的实例化规模小于动作。将动作保留在提升层级，可以保留 LiSAT 对动作的紧凑表示。我们的三种编码在状态表示的实例化程度上有所不同：一种对谓词进行完全实例化，另外两种则使用 PLMG 对其进行部分实例化，但它们对谓词的编码方式各不相同。

使用完全实例化事实进行编码

我们首先介绍一种简单的方法，作为基线。在这种方法中，我们对所有谓词进行完全实例化，并将它们用作状态表示。事实上，这种编码与 Kautz、McAllester 和 Selman，以及 Kautz 和 Selman（1996）提出的方法类似，后者由 Ernst、Millstein 和 Weld（1997）实现并自动化。两种编码的主要区别在于：（1）我们使用统一参数，而非简单拆分或重载拆分；（2）我们采用 LiSAT 对类型标注和类型约束的处理方式；（3）我们采用 LiSAT 专门针对静态前提条件的编码。除了用于编码选定动作的变量 a^t 和 $(v = o)^t$ 之外，我们还引入以下变量来描述状态：

- f^t：表示事实 f 在时间步 t是否为真。
- $c^{f,\ a\ ,\ p\ (v_1,\ ...,\ v_n)}_t$：表示动作 a 通过其效果 $p(v_1,\ ...,\ v_n)$，在时间步 t使事实 f为真，其中 f是 $p(v_1,\ ...,\ v_n)$ 的实例化。

对于这种编码，我们将所有谓词完全实例化，以获得包含所有事实的集合 $\mathcal{F}$。进一步假设，我们对长度为 L的计划进行编码，并且存在一个函数 $achievers$，它接收一个事实作为论元，并返回与该事实对应的动作及其效果中的谓词（添加或删除）。

首先，我们分别通过 $\mathcal{F}_I = \bigwedge_{f\in\mathcal{I}} f^0 \wedge \bigwedge_{f\notin\mathcal{I}} \neg f^0$和 $\mathcal{F}_G = \bigwedge_{f\in\mathcal{G}} f^L$在时间步 0 和 L断言初始状态和目标状态。约束 (1) 强制所选动作的前置条件成立。每个时间步都会生成带有参数 t 的公式。注意，在本例中，$p(o_1,\ ...,\ o_n)$ 是一个特定事实。

$$\forall a\in\mathcal{A}\forall p(v_1,...,v_n)\in prec(a)\forall o_1\in\mathcal{O}^{v_1},...,o_n\in\mathcal{O}^{v_n}:$$
$$a^t \wedge \bigwedge_{i=1}^{n}(v_i = o_i)^t \implies p(o_1,...,o_n)^t \tag{1}$$

约束（2）确保动作的效果在下一个时间点生效。否定（¬）存在于删除效果中，但不存在于添加效果中。约束（3）引入了原因变量，用于记录事实变为真的原因（无¬）或变为假的原因（¬）——具体取决于该效果是在添加还是删除该事实。

$$\forall a\in\mathcal{A}\forall(\neg)p(v_1,...,v_n)\in \mathit{eff}(a)\forall o_1\in\mathcal{O}^{v_1},...,o_n\in\mathcal{O}^{v_n}:$$
$$a^t \wedge \bigwedge_{i=1}^{n}(v_i = o_i)^t \implies (\neg)p(o_1,...,o_n)^{t+1} \tag{2}$$

$$c^{(\neg)p(o_1,...,o_n),t}_{a,p(v_1,...,v_n)} \implies a^t \wedge \bigwedge_{i=1}^{n}(v_i = o_i)^t \tag{3}$$

最后，约束 (4) 和 (5) 指定，事实真值的变化是由动作的效果引起的。

$$\forall f\in\mathcal{F}: f^t \wedge \neg f^{t+1} \implies \bigvee_{a,p(v_1,...,v_n)\in achievers(\neg f)} c^{\neg f,t+1}_{a,p(v_1,...,v_n)} \tag{4}$$

$$\forall f\in\mathcal{F}: \neg f^t \wedge f^{t+1} \implies \bigvee_{a,p(v_1,...,v_n)\in achievers(f)} c^{f,t+1}_{a,p(v_1,...,v_n)} \tag{5}$$

例 3. 考虑运输领域中的一个简单规划问题，其中只有 *3* 个对象：*v*-车辆、*p*-包裹、*l*-位置。在初始状态中，v 位于 l，而 p 在 v 中。在目标状态中，p 位于 l。这个问题长度为 *1* 的编码将包含以下子句（完整编码会包含更多子句，但为简洁起见，我们只列出相关子句）：

- *Precondition check:*
  - $drop^1 \wedge (v_1 = v)^1 \wedge (v_2 = l)^1 \implies at(v,l)^0$
  - $drop^1 \wedge (v_1 = v)^1 \wedge (v_4 = p)^1 \implies in(p,v)^0$
- *Effects:*
  - $drop^1 \wedge (v_1 = v)^1 \wedge (v_4 = p)^1 \implies \neg in(p,v)^1$
  - $drop^1 \wedge (v_2 = l)^1 \wedge (v_4 = p)^1 \implies at(p,l)^1$
- *Cause Variables:*
  - $c^{at(p,l),1}_{drop,at(v_2,v_4)} \implies drop^1 \wedge (v_2 = l)^1 \wedge (v_4 = p)^1$
  - $c^{\neg in(p,v),1}_{drop,at(v_1,v_4)} \implies drop^1 \wedge (v_1 = v)^1 \wedge (v_4 = p)^1$
- *Frame Axioms:*
  - $in(p,v)^0 \wedge \neg in(p,v)^1 \implies c^{\neg in(p,v),1}_{drop,at(v_1,v_4)}$
  - $\neg at(p,l)^0 \wedge at(p,l)^1 \implies c^{at(p,l),1}_{drop,at(v_2,v_4)}$

谓词剪枝 作为适用于我们所有编码的一项重要优化，我们会从所有事实 $\mathcal{F}$的列表中移除事实 f，因为对于任何计划而言，它们的真值都无关紧要。直观地说，只有当某个事实出现在目标中或作为前提条件时，它才可能影响计划。既不出现在这两处的事实，不会影响任何动作执行的成功或失败，也不会影响目标的满足，因此可以安全地丢弃。在基于实例的规划中，通常会剪枝这类无关事实（例如，参见 Muise、McIlraith 和 Beck（2012，第 8 定义）关于基于实例的概率规划的讨论）。由于采用了提升式方法，我们无法逐一分析每个地面事实。因此，我们会移除所有未出现在任何前提条件中的谓词。这可能会过度删除一些未出现在任何前提条件中、但仅出现在目标中的事实。作为例外，目标事实会作为独立事实加入编码过程，以确保正确检查目标条件。我们将此过程称为谓词剪枝。

使用部分接地事实进行编码

本节介绍本研究的创新贡献。我们提出了一类新的可满足性编码，利用部分提升互斥组（PLMG）对状态描述进行部分实例化，同时保持动作编码完全提升式。这种方法利用规划问题的内在结构来提高编码的可扩展性。与以往将提升式规划编码为可满足性问题的方法不同，后者的公式大小会随计划长度呈二次增长，而我们的编码的公式大小则呈线性增长，从而解决了现有方法的一项关键局限。

在实践中，完全接地编码会导致公式规模大得难以接受。例如，对象数量过多（超过 10^3）或谓词元数过高（大于 2），都可能导致接地时产生大量事实。应对这一问题的一种方法是利用规划问题固有的内部结构。为此，我们利用 PLMG；它们使我们能够只对谓词进行“部分”接地，从而得到更紧凑的表示。

设 $\mathcal{M}$ 为一个 PLMG。由此可知，在每个状态中，其文字的所有基元实例（即常规事实）中至多只有一个为真。我们不再生成所有这些文字，而是使用 PLMG 的计数变量来隐式表示当前为真的事实。为此，我们需要（1）确定该事实是由 $\mathfrak{L}(\mathcal{M})$ 中的哪个原子接地而来，以及（2）其中计数变量的取值。这样可以实现更紧凑的表示。

实际上，Fišer（2020）的LMG推断会生成多于必要数量的LMG，而且这些LMG之间存在显著重叠。我们采用 Helmert（2009）的贪心PLMG选择方法，该方法多年来一直是标准做法。非贪心替代方案通常对性能影响甚微，甚至会产生负面影响。首先，如果某个LMG的实例化包含谓词 p 的所有事实，我们就选取由它实例化得到的所有PLMG，并将 p 从考虑范围中移除。我们重复这一过程，直到无法再找到符合条件的LMG。然后，我们将剩余的谓词（通常只占全部谓词的一小部分，也可能一个都没有）和LMG完全实例化为PLMG。我们采用贪心方式选择PLMG，首先选择能够覆盖最多尚未覆盖事实的PLMG，直到剩余PLMG均无法覆盖新事实。这样，我们就得到一组用于编码的PLMG，即 $\mathfrak{P}$。经过这一过程后，仍可能存在未被任何 PLMG覆盖的事实 $\mathfrak{U}$。为简单起见，我们假设每个PLMG中不存在两个具有相同谓词的文字，也就是说，可以根据谓词识别它们；Fišer（2020）生成的所有LMG组都满足这一条件。

对于该编码，我们仅为 $\mathfrak{U}$ 中未覆盖事实保留变量 f^t 和原因变量。对于这些事实，我们也保留此前所述的编码。我们移除与所有其他事实相关的事实，并改用 PLMG 进行编码。为此，我们引入新变量：

- $(c^{\mathcal{M}} = o)^t$：表示 PLMG $\mathcal{M}$ 的计数变量 c 在时间 t是否具有值 o。
- $(p^{\mathcal{M}})^t$：表示 PLMG $\mathcal{M}$的谓词 p 的文字在时间 t为真。
- $(v_j \in D(c^{\mathcal{M}}))^t$：表示在时间 t，分配给统一参数 v_j 的对象是否与 PLMG $\mathcal{M}$ 的计数变量 c 属于相同类型。
- $(c^{\mathcal{M}} \equiv v_j)^t$：表示在时间 t，PLMG $\mathcal{M}$的计数变量 c 是否与统一参数 v_j 具有相同的值。
- $(c^{\mathcal{M}} \leftarrow a_{p(v_1,\ldots,v_n)})^t$：表示动作 a 通过其效果 $p(v_1,\ldots,v_n)$ 设置了计数变量 $c^{\mathcal{M}}$ 的值。
- $(p^{\mathcal{M}} \leftarrow a_{p(v_1,\ldots,v_n)})^t$：表示动作 a 通过其效果 $p(v_1,\ldots,v_n)$将 PLMG $\mathcal{M}$ 的谓词设置为 p。
- $(c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t$：表示 PLMG $\mathcal{M}$ 的计数变量 c 在时间 $t-1$ 和 t 的值不同。

为方便记号，我们定义函数 $\mathbf{O}$，它以 PLMG 的一个文字作为论元。对于该文字在第 j 个参数位置上的每个对象 o，函数都会返回一对值 (o, j)。我们引入 $\mathbf{V}$，使其对于该文字在第 j 个参数位置上的每个变量 v，都返回一对值 (v, j)。

首先，我们针对每个 PLMG 断言：每个计数变量恰有一个取值，且恰有一个文字被选中。此约束仅适用于恰有一个 PLMG。对于并非恰有一个的 PLMG，我们会向其中添加一个新的文字“none”；当该 PLMG 中其他取值均不为真时，该文字为真。这样，我们也可以强制这些 PLMG 中恰有一个文字为真。

$$\forall \mathcal{M} \in \mathfrak{P}\ \forall c \in \mathfrak{C}(\mathcal{M}):$$
$$atMostOne(\{(c^{\mathcal{M}} = o)^t \mid o \in \mathcal{O}^c\}) \wedge \bigvee_{o \in \mathcal{O}^c} (c^{\mathcal{M}} = o)^t \quad (6)$$

$$\forall \mathcal{M} \in \mathfrak{P}:$$
$$atMostOne(\{(p_i^{\mathcal{M}})^t \mid p(x_1,\ldots,x_m) \in \mathfrak{L}(\mathcal{M})\}) \quad (7)$$
$$\bigvee_{p(x_1,\ldots,x_m) \in \mathfrak{L}(\mathcal{M})} (p_i^{\mathcal{M}})^t \quad (8)$$

第二步，我们需要正确地断言初始状态。为此，我们遍历初始状态和 PLMG 中的所有事实，并检查 PLMG 是否包含匹配文字；如果包含，就断言该文字。这里的匹配是指：如果两者在某个给定的参数位置上都有对象，则这些对象相同；否则，初始状态事实中的对象属于 PLMG 文字变量的类型。我们也会为目标生成对称的子句；为简洁起见，此处省略。

$$\forall p(o_1,...,o_m) \in \mathcal{I}\ \forall \mathcal{M} \in \mathfrak{P}\ \forall p(x_1,\ldots,x_m) \in \mathfrak{L}(\mathcal{M}):$$
$$\text{if } \forall (o,j) \in \mathbf{O}(p(x_1,...,x_{m_i})): o = o_j$$
$$\text{and } \forall (c,j) \in \mathbf{V}(p(x_1,...,x_{m_i})): o_j \in \mathcal{O}^c \text{ then}$$
$$(p^{\mathcal{M}})^0 \wedge \bigwedge_{(c,j) \in \mathbf{V}(p(x_1,\ldots,x_m))} (c^{\mathcal{M}} = o_j)^0 \quad (9)$$

第三，我们必须断言动作的前置条件确实得到满足。也就是说，对于每个包含该前置条件的 PLMG，若将其选定论元实例化，则这一实例化必须在前一个时间步的某个 PLMG 中为真。这里，我们不直接访问 PLMG 的计数变量的取值，而是断言中间的 $(c^{\mathcal{M}} \equiv v_j)^t$ 变量为真，表示某个特定计数变量必须等于某个特定统一参数。

$$\forall a \in \mathcal{A} \forall p(v_1,\ldots,v_m) \in prec(a)$$
$$\forall \mathcal{M} \in \mathfrak{P}\ \forall p(x_1,\ldots,x_m) \in \mathfrak{L}(\mathcal{M}):$$
$$\text{if } \forall (o,j) \in \mathbf{O}(p(x_1,...,x_m)): o \in \mathcal{O}^{v_j} \text{ then}$$

$$a^t \wedge \bigwedge_{(o,j)\in \mathbf{O}(p(x_1,\ldots,x_m))} (v_j = o)^t \wedge \bigwedge_{(c,j)\in \mathbf{V}(p(x_1,\ldots,x_m))} (v_j \in D(c))^t \implies (p^{\mathcal{M}})^t \wedge \bigwedge_{(c,j)\in \mathbf{V}(p(x_1,\ldots,x_m))} (c^{\mathcal{M}} \equiv v_j)^t \quad (10)$$

类似地，我们必须断言动作的效果在同一时间步发生，从而将 PLMG 的计数变量设为特定值。请注意，效果可以是正向（添加）或负向（删除）的。后一种情况下，我们需要对蕴含式的右侧取反，以确保 PLMG 在下一个时间步不会取被删除的值。

$$\forall a \in \mathcal{A} \forall (\neg) p(v_1,\ldots,v_n) \in \mathit{eff}(a)$$
$$\forall \mathcal{M} \in \mathfrak{P}\ \forall p(x_1,\ldots,x_m) \in \mathfrak{L}(\mathcal{M}):$$
$$\text{if } \forall (o,j) \in \mathbf{O}(p(x_1,...,x_m)) : o \in \mathcal{O}^{v_j} \text{ then}$$
$$a^t \wedge \bigwedge_{(o,j)\in \mathbf{O}(p(x_1,\ldots,x_m))} (v_j = o)^t \wedge \bigwedge_{(c,j)\in \mathbf{V}(p(x_1,\ldots,x_m))} (v_j \in D(c))^t \implies (\neg) \left[(p^{\mathcal{M}})^{t+1} \wedge \bigwedge_{(c,j)\in \mathbf{V}(p(x_1,\ldots,x_m))} (c^{\mathcal{M}} \equiv v_j)^{t+1} \right] \quad (11)$$

对于 PLMG，可能会执行一个对其产生删除效果、但不产生添加效果的动作。在这种情况下，PLMG 会变为非活动状态，即其所有基事实均为假。在我们的编码中，此时文字 "none" 会变为真——这是由公式 8 强制得出的。接下来引入的框架公理确保 PLMG 中不会选中其他文字。对于通过提升事实变更互斥组生成的 LMG（Fišer 2020），此外还有一点：PLMG 一旦变为非活动状态，就再也不会变为活动状态，即 "none" 文字一旦为真，便会一直为真（Fišer 2020，定理 6）。我们使用公式 $(\text{none}^{\mathcal{M}})^t \implies (\text{none}^{\mathcal{M}})^{t+1}$ 来强制这一点。如果使用通过其他方法生成的 LMG，则不得添加此约束。其余编码自然允许 "none" 再次变为假，并将 PLMG 中正确的基事实设为真。

第四步，我们创建框架公理，确保状态不会随时间推移而改变，除非这种改变是由某个动作的效果引起的。对于使用 PLMG 表示状态，这意味着只有当某个动作将计数变量设为特定值（即对象）时，该变量的值才能改变。对此有两种可能的编码方式：一种是编码规定，变量要取得某个值，某个动作必须先将其设为该值；另一种是编码规定，如果变量的值发生了变化（无论变成什么值），某个动作必须先设置过该变量。第一种编码相当繁琐且规模较大，因为它需要为计数变量可能取的每个值都创建子句。此外，效果的编码已经使用了类型为 $(c^{\mathcal{M}} \equiv v_j)^t$ 的变量；若此变量为真，则表示变量 c 已被设为某个特定值，即统一参数 v_j 的值。我们首先对能够改变变量值的原因进行编码，分别用 $(c^{\mathcal{M}} \leftarrow a_{p(v_1,\ldots,v_n)})^t$ 和 $(p^{\mathcal{M}} \leftarrow a_{p(v_1,\ldots,v_n)})^t$ 变量表示，并将它们称为原因变量。

$$\forall a \in \mathcal{A} \forall p(v_1,\ldots,v_n) \in \text{效应}(a)$$
$$\forall \mathcal{M} \in \mathfrak{P}\ \forall p(x_1,\ldots,x_m) \in \mathfrak{L}(\mathcal{M}):$$
$$\text{如果 } \forall (o,j) \in \mathbf{O}(p(x_1,\ldots,x_m)): o \in \mathcal{O}^{v_j} \text{ 则}$$
$$a^t \wedge \bigwedge_{(o,j)\in \mathbf{O}(p(x_1,\ldots,x_m))} (v_j = o)^t \wedge \bigwedge_{(c,j)\in \mathbf{V}(p(x_1,\ldots,x_m))} (v_j \in D(c))^t \implies (p^{\mathcal{M}} \leftarrow a_{p(v_1,\ldots,v_n)})^{t+1} \wedge \bigwedge_{(c,j)\in \mathbf{V}(p(x_1,\ldots,x_m))} (c^{\mathcal{M}} \leftarrow a_{p(v_1,\ldots,v_n)})^{t+1} \quad (12)$$
$$(p^{\mathcal{M}} \leftarrow a_{p(v_1,\ldots,v_n)})^{t+1} \vee (c^{\mathcal{M}} \leftarrow a_{p(v_1,\ldots,v_n)})^{t+1} \implies a^t \wedge \bigwedge_{(o,j)\in \mathbf{O}(p(x_1,\ldots,x_m))} (v_j = o)^t \wedge \bigwedge_{(c,j)\in \mathbf{V}(p(x_1,\ldots,x_m))} (v_j \in D(c))^t \quad (13)$$

我们必须将计数变量被设置的原因与该变量值的变化联系起来。令 $\mathbf{C}(c^{\mathcal{M}},\ t)$ 表示计数变量 c 的所有原因变量的集合。除变为"none"之外，任何文字的变化都必须有原因。

$$\forall \mathcal{M} \in \mathfrak{P}$$
$$\forall c \in \mathfrak{C}(\mathcal{M}) : (c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t \implies \bigvee_{x\in \mathbb{C}(c^{\mathcal{M}},t)} x \quad (14)$$
$$\forall p(x_1,...,x_m) \in \mathfrak{L}(\mathcal{M}) : \neg(p^{\mathcal{M}})^t \wedge (p^{\mathcal{M}})^{t+1} \implies \bigvee_{x\in \mathbb{C}(p^{\mathcal{M}},t)} x \quad (15)$$

作为第五步也是最后一步，我们需要断言在整个编码过程中使用的中间辅助变量 $(v_j \in D(c^{\mathcal{M}}))^t$、$(c^{\mathcal{M}} \equiv v_j)^t$ 和 $(c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t$ 的语义；它们分别表示统一参数的值属于计数变量的类型、计数变量与统一参数具有相同的值，以及计数变量的值发生了变化。

$$\forall \mathcal{M} \in \mathfrak{P}\ \forall c \in \mathfrak{C}(\mathcal{M})\ \forall v \in \mathbb{U}:$$
$$\forall o \in \mathcal{O}^v \setminus \mathcal{O}^c : (v = o)^t \implies \neg(v \in D(c^{\mathcal{M}}))^t \quad (16)$$
$$\forall o \in \mathcal{O}^v \cap \mathcal{O}^c : \quad (v = o)^t \implies (v \in D(c^{\mathcal{M}}))^t \quad (17)$$
$$\wedge (c^{\mathcal{M}} = o)^t \wedge (v = o)^t \implies (c^{\mathcal{M}} \equiv v)^t \quad (18)$$
$$\wedge (c^{\mathcal{M}} \equiv v)^t \implies \left[(c^{\mathcal{M}} = o)^t \Leftrightarrow (v = o)^t\right] \quad (19)$$
$$\forall o \in \mathcal{O}^c \setminus \mathcal{O}^v : (c^{\mathcal{M}} = o)^t \implies \neg(c^{\mathcal{M}} \equiv v)^t \quad (20)$$
$$\forall \mathcal{M} \in \mathfrak{P}\ \forall c \in \mathfrak{C}(\mathcal{M}) \forall o \in \mathcal{O}^c :$$
$$(c^{\mathcal{M}} = o)^{t-1} \wedge \neg(c^{\mathcal{M}} = o)^t \implies (c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t \quad (21)$$

在公式21中，只需编码"下降沿"，因为始终恰好有一个 $(c^{\mathcal{M}} = o)^t$ 为真。

示例4。考虑与上一个示例中相同的规划问题。假设存在以下 *PLMG*：$\mathcal{M}$：$\langle \{p\}, \{?\ v、?\ l\}, \{in\ (p、?\ v)\ \ at\ (p、?\ l)\ \} \rangle$ 以及 m：$\langle \{v\}, \{?\ l\}, \{at\ (v、?\ l)\ \} \rangle$。

- 前提条件检查：
  - $drop^1 \wedge (v_4 = p)^1 \wedge (v_1 \in D(v_cnt))^1 \implies (in^{\mathcal{M}})^0 \wedge (v_cnt == v_1)^0$
  - $drop^1 \wedge (v_1 = v)^1 \wedge (v_2 \in D(l_cnt))^1 \implies (at^m)^0 \wedge (l_cnt == v_2)^0$
- 效果：逻辑与前置条件相同，只是蕴含式右侧采用了不同的时间步。

	Fully		Partially		Binary		LiSAT	PWL A*		CPDDL A*	MpC		FD A* lmc		Binary	LiSAT	PWL alt-bfws1	CPDDL gbfs	MpC		FD LAMA
	-	PP	-	PP	-	PP	-	h^1lmc	lmc	逆			-	AS	PP	-	hff	hff	逆		-	AS
块 (40)	21	21	15	15	**40**	**40**	**40**	0	2	34	4	4	12	12	**40**	**40**	22	3	4	0	12	12
子节点 (144)	48	48	**96**	**96**	**96**	**96**	83	2	1	6	1	0	8	6	**144**	**144**	104	47	66	66	110	138
GED (312)	67	67	64	64	64	64	**72**	32	0	36	56	62	40	50	77	110	286	88	183	77	**312**	**312**
Logistics（40）	26	26	**40**	**40**	**40**	**40**	32	5	0	31	0	0	16	13	**40**	**40**	**40**	13	0	0	24	16
操作系统（56）	54	54	54	54	54	54	**56**	44	24	32	0	0	18	44	51	**55**	50	31	0	0	18	46
管道 (50)	35	35	**36**	**36**	32	34	21	9	9	15	7	6	10	10	43	25	**48**	20	10	9	18	37
Rover (40)	5	5	**16**	**16**	12	12	4	2	0	3	0	0	5	5	12	4	**40**	**40**	0	0	12	12
Visitall (180)	29	34	57	110	56	**115**	107	73	30	62	18	28	71	68	**178**	176	147	121	18	56	90	90
Labyrinth（20）	8	8	8	8	8	8	**12**	1	0	1	0	2	1	8	8	12	5	0	3	1	**15**	3
Cov.（882）	293	298	386	439	402	**463**	427	168	66	220			86	102	181	216		593	606	7	**42**	
得分 (9)	4.07	4.10	5.05	5.34	5.49	**5.86**	5.32	1.72	0.83	3.12	0.53		0.68	1.97	2.72			6.71	6.51	**7.11**		

表2：长度最优（左）和满足即可（右）规划的覆盖结果。每组的最大值以粗体显示。

- 框架公理：
  - $drop^1 \wedge\ v_4 = p\ ^1 \wedge\ v_2 \in D\ l\ cnt\ \ ^1 \implies\ at^{\mathcal{M}} \leftarrow ( )$ $drop\ ^1 \wedge\ l\ cnt \leftarrow drop\ ^1$ $(( )) ( ) ( )$– $(at^{\mathcal{M}} \leftarrow drop)^1 \vee (l\ cnt \leftarrow drop)$ $^1 \implies drop^1 \wedge (v_4 = p)^1 \wedge (v_1 \in D(cnt))^1$
- 连接因果变量：
  - (差异_ l $cnt)^1 \implies (l\ cnt \leftarrow drop)^1$– ¬($at^{\mathcal{M}})^0 \wedge (at^{\mathcal{M}})^1 \implies (at^{\mathcal{M}} \leftarrow drop)^1$
- 断言中间变量：
  - $(v_2 = l)^1 \implies (v_2 \in D(l_cnt))^1$
  - $(l_cnt == v_2)^1 \wedge (l_cnt = l)^1 \implies (v_2 = l)^1$
  - $(l_cnt == v_2)^1 \wedge (v_2 = l)^1 \implies (l_cnt = l)^1$
  - $(l_cnt = l)^1 \wedge (v_2 = l)^1 \implies (l_cnt == v_2)^1$
  - $(l_cnt = l)^1 \wedge \neg(l_cnt = l)^1 \implies (diff_l_cnt)^1$

尽管像这样的简单示例无法证明这一点，但约束 *(16)* 和 *(20)*，以及变量 $(v_j \in D(c^{\mathcal{M}}))^t$，对于以下情况的编码至关重要：*PLMG* 中计数变量的类型是相应统一参数的子类型，从而确保不会强制推出不可能的蕴含式。

对象的二进制编码

在采用部分接地事实的编码中，变量 $(c^{\mathcal{M}} = o)^t$ 所包含的信息密度相当低，因为可接受对象 o 对应的变量中，最多只有一个为真。因此，这种编码需要数量呈线性增长的变量——这可能会造成阻碍，尤其是在包含许多对象的规划问题中。给定一组对象O，可以通过位表示，仅用 $\lceil \log_2(|O|) \rceil$ 个变量来编码 $(c^{\mathcal{M}} = o)^t$ 个变量所包含的信息。这提高了我们生成的SAT公式中赋值的信息密度，同时也减少了公式的整体大小（以子句数量计）。据我们所知，Ernst、Millstein 和 Weld（1997年）首次提出了这一观察。他们仅将其应用于接地动作集合。

为简化公式生成，我们首先假设所有变量都可以取任何对象作为其值。此外，我们假设对象集合 $\mathcal{O}$ 是一个有序集合，且对象的编号为 o_0 至 $o_{|\mathcal{O}|-1}$。要表示一个特定对象，需要 $B = \lceil \log_2(|\mathcal{O}|) \rceil$ 个变量。我们引入两种新的变量类型来替代$(v = o)^t$和$(c^{\mathcal{M}} = o)^t$：

- v_b^t：表示分配给统一参数 v 的对象索引的第 b 位在时间 t 为真。
- $(c^{\mathcal{M}})_b^t$：表示分配给变量 $c^{\mathcal{M}}$ 的对象索引的第 b位在时间 t为真。

为便于记号表示，我们定义函数 $\beta(x,\ b)$：当 x 的第 b 位为 1 时，该函数为真，否则为假。同样，我们写作 $(\neg)^p$，其中 p 是一个真值，用来表示当 p 为真时应当出现否定，而当 p 为假时则不应出现否定。我们用 $\mathcal{B}$ 表示集合 $\{1, \ldots, B\}$。

我们首先将表示统一参数的变量（$(v = o)\ ^t$ 的值与 v_i^t 变量相连接。

$$\forall v \forall o_i \in \mathcal{O} \forall b \in \mathcal{B} : (v = o_i)^t \implies (\neg)^{\neg\beta(i,b)} v_b^t \quad (22)$$

完全移除统一参数的变量（$(v_j = o)\ ^t$ 的独热表示，并删除公式22中的子句，是可行的。这样做带来的缩减效果远不如替换（$(c^{\mathcal{M}} = o)\ ^t$ 变量明显：统一参数仅占所有编码变量的一小部分。另一方面，使用纯二进制表示会使动作层级上的统一参数编码约束（静态前提条件、不等式）变得更加复杂，因此我们在此保留独热编码。

其次，我们必须针对每个变量 $c^{\mathcal{M}}$ 进行编码，确保由 $(c^{\mathcal{M}})_i^t$ 变量中的位掩码所标识的对象属于该变量的类型。将此类型记为 $t(c)$。预处理后，我们知道类型为 $t(c)$的对象构成了所有对象序列的一个子序列；也就是说，存在上界$u_{t(c)}$ 和下界 $\ell_{t(c)}$，使得类型 $t(c)$恰好包含对象 $\{o_i \mid \ell_{t(c)} \le i \le u_{t(c)}\}$。为实现这些限制，我们添加以下子句。请注意，位 0 是该表示中的最低有效位。

$$\forall \mathcal{M} \in \mathfrak{P} \forall c \in \mathfrak{C}(\mathcal{M}) \forall b \in \mathcal{B}$$
$$\text{if } \beta(\ell_{t(c)}, b) : \bigwedge_{b' \in \{b+1, \ldots B\}} (\neg)^{\neg\beta(\ell_{t(c)}, b')} (c^{\mathcal{M}})_{b'}^t \implies (c^{\mathcal{M}})_b^t \quad (23)$$

$$\text{if } \neg\beta(u_{t(c)},b): \bigwedge_{b'\in\{b+1,\dots B\}} (\neg)^{\neg\beta(u_{t(c)},b')}(c^{\mathcal{M}})^t_{b'} \Longrightarrow \neg(c^{\mathcal{M}})^t_b \quad (24)$$

第三，我们必须替换定义 $(c^{\mathcal{M}} \equiv v_j)^t$语义的子句。我们引入一种新的变量类型来表示逐位相等：

- $(c^{\mathcal{M}} \equiv v_j)^t_b$：表示所分配给对象在时间 t分配给 $c^{\mathcal{M}}$ 和 v 的索引相同。如果 $c^{\mathcal{M}}$ 和 v 的第 b位相等，则$(c^{\mathcal{M}} \equiv v_j)^t_b$ 为真；如果所有$(c^{\mathcal{M}} \equiv v_j)^t_b$ 均为真，则 $c^{\mathcal{M}}$ 和 v 相等。

$$\forall \mathcal{M} \in \mathfrak{P} \forall c \in \mathfrak{C}(\mathcal{M}) \forall v_j \in \mathbb{U}\ \ \forall b \in \mathcal{B}:$$

$$v^t_b \wedge (c^{\mathcal{M}})^t_b \implies (c^{\mathcal{M}} \equiv v_j)^t_b \quad (25)$$

$$\neg v^t_b \wedge \neg(c^{\mathcal{M}})^t_b \Rightarrow (c^{\mathcal{M}} \equiv v_j)^t_b \quad (26)$$

$$\left(\bigwedge_{b\in\mathcal{B}} (c^{\mathcal{M}} \equiv v_j)^t_b\right) \Rightarrow (c^{\mathcal{M}} \equiv v_j)^t \quad (27)$$

反过来，如果$(c^{\mathcal{M}} \equiv v_j)^t$ 为真，则 $c^{\mathcal{M}}$ 和 v 的所有位都必须相等。

$$(c^{\mathcal{M}})^t_b \wedge (c^{\mathcal{M}} \equiv v_j)^t \implies v^t_b \quad (28)$$

$$v^t_b \wedge (c^{\mathcal{M}} \equiv v_j)^t \implies (c^{\mathcal{M}})^t_b \quad (29)$$

最后，我们必须替换定义$(c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t$ 变量语义的子句，以编码$c^{\mathcal{M}}$ 的值从时间 t 到时间 $t+1$发生了变化这一事实。我们需要为每一位引入一个辅助变量：

- $(c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t_i$：表示变量 $c^{\mathcal{M}}$ 的值在时间 t 和 $t+1$之间的第 i位不同。

接着，我们添加以下子句 $\forall \mathcal{M} \in \mathfrak{P} \forall c \in \mathfrak{C}(\mathcal{M})$：

$$(c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t \implies \bigvee_{b\in[1,\dots,B]} (c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t_b \quad (30)$$

$$\forall b \in \mathcal{B}: (c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t_b \implies (c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t \quad (31)$$

$$\forall b \in \mathcal{B}: (c^{\mathcal{M}})^t_b \wedge \neg(c^{\mathcal{M}})^{t+1}_b \implies (c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t_b \quad (32)$$

$$\forall b \in \mathcal{B}: \neg(c^{\mathcal{M}})^t_b \wedge (c^{\mathcal{M}})^{t+1}_b \implies (c^{\mathcal{M}} \not\equiv c^{\mathcal{M}})^t_b \quad (33)$$

实验评估

实验在 AMD EPYC 9654 上进行，内存限制为 8GB，时间限制为 1800 秒。所用数据和代码的快照存档于 (Filipe and Behnke 2026)。代码库的最新版本维护在公共代码仓库[2]中。实验结果取自提升式规划的标准基准测试集[3]，并使用 VAL[4]进行了验证。这些基准测试不包含动作代价，因此在我们的评估中，长度最优与代价最优等同。表 2 展示了多个规划器在最优模式（左侧）和满意模式（右侧）下的运行结果。图 1c 展示了最优模式的仙人掌图。我们的编码在启用和不启用谓词剪枝（PP）的情况下运行，并使用 Kissat 可满足性求解器（Biere 等，2024），与 LiSAT 使用的是同一求解器。

[2]https://github.com/galvusdamor/PGSAT [3] https://github.com/abcorrea/htg-domains [4] https://github.com/KCL-Planning/VAL

LiSAT。对于满意模式，我们使用 H¨oller 和 Behnke（2022）提出的迭代算法：依次尝试求解界限为 10、25、50、100 和 200 的公式。每个界限分配的时间预算为$\frac{remaining\ run\ time}{|remaining\ bounds|}$，之后再尝试下一个界限。

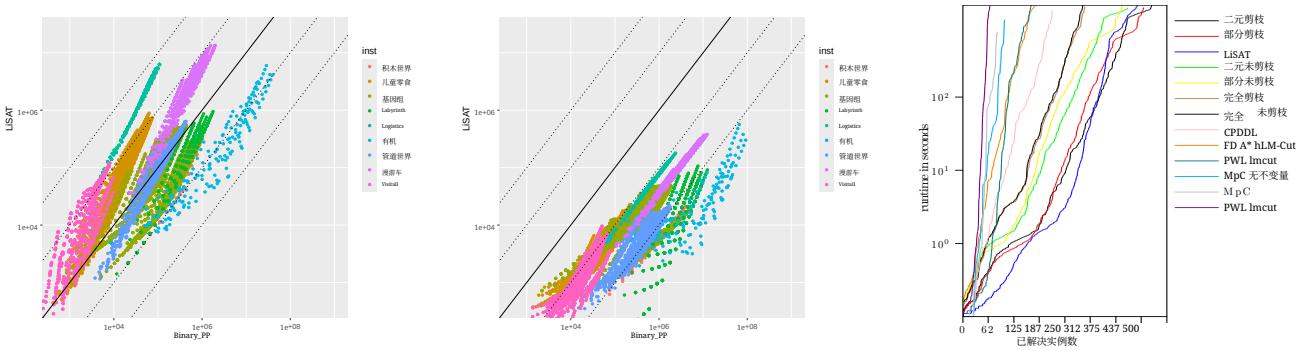
我们将其与最先进的提升式规划器 Pow- erlifted（PWL）和 CPDDL，以及基于具体化表示的规划器 Madagascar（MpC）和 Fast Downward（FD）进行比较。对于 PWL，我们在最优模式下使用 A* 搜索和 h^{max} 启发式，并在满意模式下使用 alt-bfws1 和 h^{ff}。我们还使用了 Wich- lacz 等人（2023）报告的最优配置（PWL A* lmc），该配置提出了 lm-cut（lmc）启发式的提升式版本（Helmert 和 Domshlak，2009）；但即使获得一位作者的技术支持，我们仍无法完全复现他们的结果。尽管如此，我们仍将其列出以供参考。对于 CPDDL，我们采用 Hor˘c´ık 和 Fi˘ser（2023）报告的最佳最优配置和满意配置（盖夫曼图分别减少 95% 和 25%，A* 分别结合 $h^{lm\text{-}cut}$ 和 GBFS 与 h^{ff}）。作为参考，我们还纳入了两个基于具体化表示的规划器：Madagascar（最先进的基于可满足性的具体化规划器，支持和不支持不变量计算两种配置；不使用并行处理，且最优模式下使用算法 S），以及 Fast Downward（Helmert，2006）（支持和不支持动作拆分（Elahi 和 Rintanen，2024）两种配置），后者使用 A* 搜索和 lm-cut 启发式（Helmert 和 Domshlak，2009）（最优模式），以及 LAMA 配置（满意模式）。覆盖率是所有领域中已解决实例的总数，而得分是各领域已解决实例百分比之和。

我们不与 Robinson 等人（2009）提出的方法进行比较，因为该方法并非用于最优规划（它允许动作并行），而且没有公开可用的实现；我们也不与 Shaik 和 van dePol（2022）提出的基于 QBF 的规划器进行比较，因为后者的表现不如 LiSAT。

结果——最优规划在最优设置下，基于搜索的提升方法明显不如基于可满足性的方法：即使是表现最好的方法（CPDDL A* lmc），效果也不如我们表现最差的基于可满足性的方法（Fully，无 PP）。

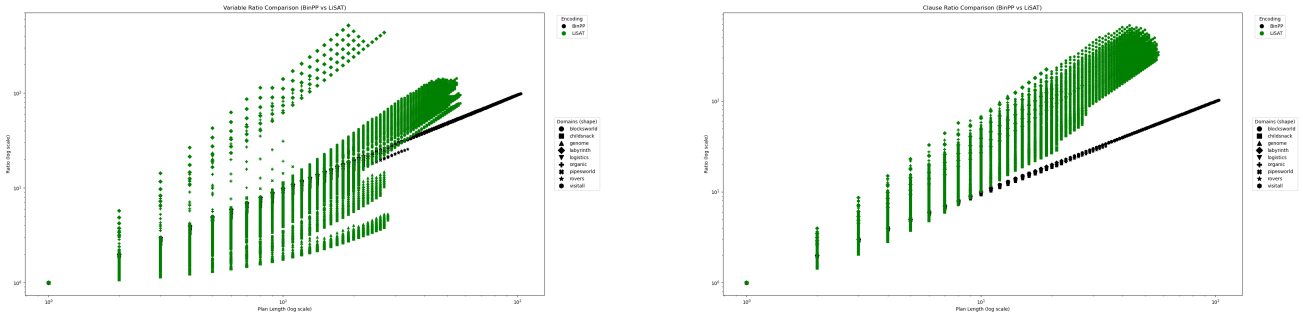
谓词剪枝在 Vis-itall 中的优势最为明显。尽管只剪除了一个谓词（visited），但仅此一项就消除了约一半的基元事实。由于该谓词不属于任何 PLMG，其所有事实都需要进行（代价高昂的）显式编码；而其他所有非静态事实（属于 at-robot 谓词）则构成一个 PLMG。在这里，谓词剪枝大幅减少了未通过 PLMG 紧凑编码、而是采用显式编码的事实数量（只剩下目标）。在 Rovers 中，3 个一元谓词（均属于已通信的 X 数据类型）被剪除。被移除的事实数量并不显著，因此可以解释为何没有提升。对于 Pipesworld（二元），观察到的差异并非由谓词剪枝造成，而是因为其中一种编码未能在时间限制内找到满足赋值。

我们的三种编码得分高于 LiSAT。其中两种的覆盖率也更高。此外，它们仅在三个领域（GED、操作系统、Labyrinth）的性能逊于 LiSAT；但在其中两个领域（GED、操作系统）仍具有竞争力，已解决实例的差距为



（a）变量数量（双对数） (b) 子句数量（双对数） (c) 覆盖率

图1：公式大小、变量数量和覆盖率的比较



(a) 变量比率比较（双对数） (b) 子句比率比较（双对数）

图2：变量/子句数量相对于长度为 1 的公式的比率比较

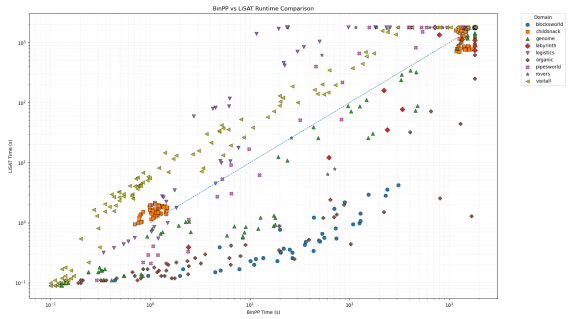


图3：求解时间散点图

该领域实例总数的 $< 5\%$。相比之下，在我们的编码（Binary 谓词剪枝）优于 LiSAT 的五个领域中，有四个领域的已解决实例数量增加了 $\geq 9\%$；其中 Logistics、管道世界和 Rover 的提升尤为显著，差距为 $\geq 20\%$。

结果 – 满意性规划 为简洁起见，这里我们只报告表现最佳的编码（Binary，PP）——就得分而言，它排名第二，仅次于 PWL。尽管与 FD、LiSAT 和 PWL 相比，我们的覆盖率较低，但这主要是由于 GED，其实例数量会使覆盖率产生偏差。重要的是，我们的规划器（以及 LiSAT）在积木世界、儿童零食、Visitall、Labyrinth 和操作系统领域中，优于其他提升式和基于地面的满意性规划器。这表明，基于 SAT 的规划器（我们的规划器和 LiSAT）为当前基于搜索的提升式规划器提供了互补能力，例如可用于组合规划器的一部分。

结果——公式大小 图1比较了 Binary PP 和 LiSAT 的子句数量与变量数量。有趣的是，对于相同的计划长度，Binary PP 生成的子句始终多于 LiSAT。即使在最佳情况下（儿童零食），我们的编码生成的子句也仅比 LiSAT 多不到一个数量级，而在大多数情况下，差异为一到两个数量级。这是由于正确编码 PLMG 的语义所需的编码规模更大。另一方面，在大多数实例中，Binary PP 所需的变量更少，有时可少至两个数量级。尽管如此，我们的编码呈线性扩展，而 LiSAT 呈二次扩展的趋势，仍可从 LiSAT 变量和子句数量更快的增长中看出。对于子句数量而言，这一点在 OS（浅蓝色）中尤为明显。我们在附录中提供了更多图表，以更清晰地展示公式的增长情况。

图2展示了双对数图，对比谓词剪枝二进制编码与 LiSAT 中任意长度公式所用变量数量和子句数量相对于长度为 1 的公式的增长情况。对于

为此，对于每个规划问题，我们取计划长度为 1 时编码所用的子句数量（以及变量数量）： N_1。之后，对于任意更大的计划长度，我们将子句数量（以及变量数量）除以 N_1。这样，我们就以计划长度为 1 的公式为基准，对公式的增长进行了归一化。这使我们能够更轻松地比较不同规模（对象数量不同）的实例，以及不同领域（谓词和动作的数量及元数不同）之间的实例。

从子句数量和变量数量两方面来看，我们可以发现，随着计划长度增加，我们的编码呈线性增长，而 LiSAT 则呈预期的二次增长。此外，还可以看出，LiSAT 公式的增长也取决于各领域中的实例：较难的实例增长更大；相比之下，我们的编码在不同领域中的扩展幅度几乎没有差异。

运行时间 图 3 展示了两个规划器求解时间的散点图。有趣的是，LiSAT 与 Binary PP 并无明显优劣之分，表现取决于具体领域（例如，Binary PP 在 Visitall 领域表现较好，而 LiSAT 在积木世界领域表现较好）。这表明两种方法各有所长。

结论

我们提出了三种将部分提升式规划编码为可满足性问题的方法：保持动作完全提升，同时对状态描述进行部分实例化。与以往公式大小随计划长度呈二次增长的编码不同，我们的方法呈线性增长。通过实证，我们证明，所提出的部分实例化编码和二进制编码在长度最优规划方面优于最先进水平的 LiSAT，在满意性规划方面也具有竞争力。此外，我们的新编码与当前最先进的提升式满意规划器形成互补，因为它们能够解决基于搜索的规划器无法处理的问题。

虽然我们的编码针对顺序计划，但未来的工作可以探索整合动作并行。这可能提升可扩展性，并拓宽部分基化编码的适用性，尤其是在满意性规划场景中。编码还可以扩展，以支持负向前提条件和条件效果。

致谢

本出版物是项目“挖掘基于 SAT 的规划中的问题结构”的成果，该项目隶属于开放竞争计划，项目编号为 OCENW.M.22.050，由荷兰研究理事会（NWO）资助。

参考文献

Biere, A.; Faller, T.; Fazekas, K.; Fleury, M.; Froleyks, N.; and Pollitt, F. 2024。CaDiCaL、Gimsatul、IsaSAT 和 Kissat 参加 SAT 竞赛 2024。载于 Heule, M.; Iser, M.; J¨arvisalo, M.; and Suda, M. 编，*SAT 竞赛 2024* 论文集——求解器、基准测试与证明检查器说明，赫尔辛基大学计算机科学系 B 系列报告 B-2024-1 卷，8–10。赫尔辛基大学。

Corrˆea, A. B.; Pommerening, F.; Helmert, M.; and Franc`es, G. 2020。使用查询优化技术进行提升后继生成。载于 Beck, J. C.; Karpas, E.; and Sohrabi, S. 编，第三十届国际自动规划与调度会议论文集（*ICAPS 2020*），80–89。AAAI出版社。Elahi, M.; and Rintanen, J. 2024。通过自动动作模式拆分优化规划领域的优化过程。载于 Dy, J.; and Natarajan, S. 编，第三十八届 *AAAI* 人工智能会议论文集（*AAAI 2024*），20096–20103。AAAI出版社。

Ernst, M. D.；Millstein, T. D.；Weld, D. S.，1997年。《规划问题的自动 SAT 编译》。载于第十五届国际人工智能联合会议论文集，第 2 卷，1169—1176 页。美国加利福尼亚州旧金山：Morgan Kaufmann出版社。

Espasa, J.；Coll, J.；Miguel, I.；Villaret, M.，2021年。《探索 Essence Prime 中的提升规划编码》。载于人工智能研究与发展：加泰罗尼亚人工智能协会第 *23* 届国际会议论文集，第 339 卷，66–75 页。IOS出版社。

Fikes, R. E.; 和 Nilsson, N. J. 1971. STRIPS：将定理证明应用于问题求解的新方法。人工智能, 2: 189–208.

Filipe, J.; 和 Behnke, G. 2026.（数据与代码）当实例化与不实例化都不理想时——将规划编码为可满足性的部分基化编码。https://doi.org/ 10.5281/zenodo.19067374.

Fiˇser, D. 2020. 提升事实交替互斥组与经典规划问题的剪枝基化。载于 Conitzer, V.; 和 Sha, F. 编，第三十四届 *AAAI*人工智能会议论文集（*AAAI 2020*），9835–9842. AAAI出版社.

Helmert, M. 2006. Fast Downward规划系统。人工智能研究期刊，26: 191–246.

Helmert, M. 2009. PDDL规划任务的简洁有限域表示。人工智能，173：503–535。

Helmert, M.; and Domshlak, C. 2009. 地标、关键路径与抽象：究竟有何区别？载于第十九届自动规划与调度国际会议论文集（*ICAPS 2009*），162–169。AAAI出版社。

H¨oller, D.; and Behnke, G. 2022. 用命题逻辑编码提升式经典规划。载于第三十二届国际自动规划与调度会议论文集（*ICAPS 2022*），134–144。AAAI出版社。

Horˇc´ık, R.; and Fiˇser, D. 2023. 提升式规划中的盖夫曼图。载于 第*26*届欧洲人工智能会议论文集（*ECAI2023*），1052–1059。IOS出版社。

Huang, R.; Chen, Y.; and Zhang, W. 2012. SAS$^+$ 规划即为可满足性。《人工智能研究期刊》, 43: 293–328.

Kautz, H.; McAllester, D.; and Selman, B. 1996. 在命题逻辑中编码计划。载于 Aiello, L. C.; Doyle, J.; and Shapiro, S. C. 编，第五届知识表示与推理原理国际会议论文集 *(KR 1996)*, 374–384。Morgan Kaufmann。Kautz, H.; and Selman, B. 1992. 规划即为可满足性。载于 Neumann, B. 编，第十届欧洲人工智能会议论文集 *(ECAI 1992)*, 359–363。约翰·威立父子出版公司。Kautz, H.; and Selman, B. 1996. 突破极限：规划、命题逻辑与随机搜索。载于第十三届全国人工智能会议论文集 *(AAAI 1996)*, 1194–1201。AAAI出版社。Kautz, H.; Selman, B.; and Hoffmann, J. 2006. SatPlan：规划即为可满足性。载于 第五届国际规划竞赛 *(IPC-5)*：规划器摘要。Muise, C. J.; McIlraith, S. A.; and Beck, J. C. 2012. 利用状态相关性改进非确定性规划。载于 McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B. 编，第二十二届自动规划与调度国际会议论文集 *(ICAPS 2012)*, 172–180。AAAI出版社。Penberthy, J. S.; and Weld, D. S. 1992. UCPOP：一种适用于 ADL 的可靠、完备偏序规划器。载于 Nebel, B.; Rich, C.; and Swartout, W. 编，第三届知识表示与推理原理国际会议论文集 *(KR 1992)*, 103–114。Morgan Kaufmann。Rintanen, J. 2011. Madagascar：基于 SAT 的可扩展规划。载于*IPC 2011* 规划器摘要, 61–64。Rintanen, J. 2012. 规划即为可满足性：启发式方法。人工智能, 193: 45–86。Robinson, N.; Gretton, C.; Pham, D. N.; and Sattar, A. 2009. 使用动作的分割表示进行基于 SAT 的并行规划。载于第十九届自动规划与调度国际会议论文集 *(ICAPS 2009)*, 281–288。AAAI出版社。Shaik, I.; and van de Pol, J. 2022. 无需实例化的经典规划 QBF。载于第三十二届自动规划与调度国际会议论文集 *(ICAPS 2022)*, 329–337。AAAI出版社。Sinz, C. 2005. 探索布尔基数约束的最优 CNF 编码。载于 第十一届约束编程原理与实践国际会议论文集 *(CP 2005)*, 第3709卷, 827–831。施普林格。Wichlacz, J.; H¨oller, D.; Fi˘ser, D.; and Hoffmann, J. 2023. 用于提升式最优规划的地标切割启发式。载于第二十六届欧洲人工智能会议论文集 *(ECAI 2023)*, 2623–2630。IOS出版社。