DAG 压缩图基本问题的复杂性

FlorianChudigiewitsch 德国吕贝克大学理论计算机科学研究所

Till Tantau 德国吕贝克大学理论计算机科学研究所**Felix Winkler**德国吕贝克大学

arXiv:2603.22063v1 [cs.DS] 23 Mar 2026

摘要

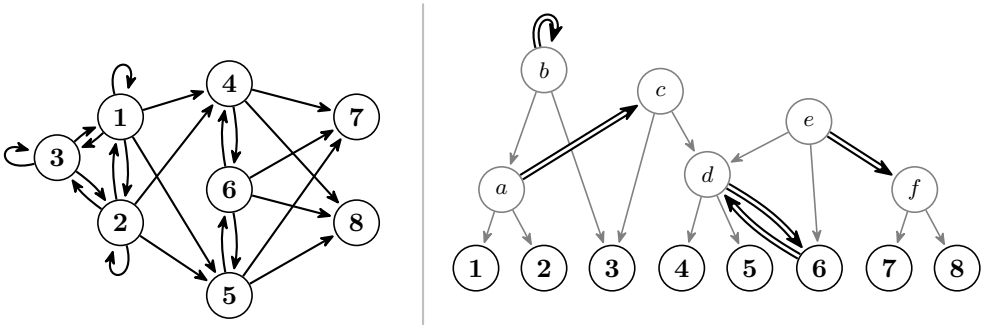
图（通常是稠密图）的 *DAG* 压缩是一种简单的数据结构，用于存储顶点簇之间的连接方式；这些簇通过有向无环图（DAG）中可达汇点的集合间接描述。它们推广了树压缩，在树压缩中，簇构成树状层次结构。我们首次证明，DAG 压缩能够实现比树压缩更高效的压缩。我们研究DAG 压缩，是因为若干简单的标准算法（例如图上的广度优先搜索）可以直接在压缩图上而非原图上运行，而且关键在于，其运行时间取决于压缩图（通常较小）的规模。我们进一步扩展了可采用这种方式实现的算法列表，证明计算最小生成树的 Kruskal 算法可以改造为直接在DAG 压缩上运行。在另一方面，我们回答了先前研究提出的核心开放问题：为给定图计算最小规模DAG压缩有多难？这是 NP 困难的；即使在动态场景下也是如此。在动态场景中，当输入图中添加或删除一条边时，我们必须以最优方式更新 DAG 压缩。

2012年 ACM 学科分类计算理论 → 图算法分析；计算理论 → 数据结构设计与分析

关键词和短语 图压缩、图算法、复杂性

1 引言

数据压缩是处理和存储海量数据不可或缺的工具，也已成为理论计算机科学中的重要研究主题 [2, 3, 6, 7, 9, 14, 16]。我们关注稠密图的压缩——在现代应用中，稠密图很容易包含数十亿条边——并希望能够直接在压缩图上运行基本算法，而无需将其解压缩。近期提出了一种特别简单的有向图压缩方法 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ [1]，即 DAG 压缩。它们是三元组 $D = (V, A, E)$，其满足以下条件（我们用粗体表示原图，并用常规字体表示压缩图）：首先，(V, A) 是一个有向无环图（DAG），其汇点恰好是 $\boldsymbol{V} \subseteq V$ 中的顶点。每个顶点 $v \in V$ 表示一个簇 $\boldsymbol{C}_D(v) \subseteq \boldsymbol{V}$，定义为在 (V, A) 中从 v 可达的汇点集合（当 D 的含义可从上下文中明确时，我们会省略下标）。其次，$E \subseteq V \times V$ 是压缩边的集合，使得 $\boldsymbol{E} = \bigcup_{(u,v)\in E} \boldsymbol{C}(u) \times \boldsymbol{C}(v)$，这意味着每条压缩边 (u, v) 都编码了 $\boldsymbol{G}$ 中从 $\boldsymbol{C}(u)$ 内的顶点到 $\boldsymbol{C}(v)$ 内顶点的所有可能边。图 1 展示了一个示例。



待压缩的 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$　　DAG 压缩 $D = (V, A, E)$ 的 $\boldsymbol{G}$

图 1 待压缩的图 $\boldsymbol{G}$，其顶点集为 $\boldsymbol{V} = \{1, \ldots, 8\}$，以及对 $\boldsymbol{G}$ 的 DAG 压缩。该压缩向 $\boldsymbol{V}$ 添加了新的簇顶点，得到 $V = \boldsymbol{V} \cup \{a, \ldots, f\}$。簇 DAG 的弧集为 A，如灰色所示：例如，弧 $(a, 1)$ 和 (b, a) 属于 A。对于每个$v \in V$，集合 $\boldsymbol{C}(v)$ 是从 v 出发在 C 中可达的汇点集合，因此 $\boldsymbol{C}(a) = \{1, 2\}$，且 $\boldsymbol{C}(b) = \{1, 2, 3\}$，而 $\boldsymbol{C}(3) = \{3\}$。$E$ 中的压缩边用双线表示。压缩边 $(a, c) \in E$ 表示在 $\boldsymbol{G}$ 中，$\boldsymbol{C}(a) = \{1, 2\}$ 中的每个顶点都与 $\boldsymbol{C}(c) = \{3, 4, 5\}$ 中的每个顶点之间存在边；压缩循环 $(b, b) \in E$ 则意味着 $\boldsymbol{C}(b) = \{1, 2, 3\}$ 是 $\boldsymbol{G}$ 中的一个团。总计，$\boldsymbol{E} = \bigcup_{(u,v)\in E} \boldsymbol{C}(u) \times \boldsymbol{C}(v)$。

DAG 是否是描述图压缩中所用簇的恰当数据结构，并非一目了然。为什么不使用树呢？树或许更自然，更重要的是，可以高效编码——这在数据压缩领域是一项理想的特性。然而，DAG 压缩最终被证明是更灵活的数据结构。例如，给定两个可能重叠的图 $\boldsymbol{G}_1$ 和 $\boldsymbol{G}_2$ 的不相交 DAG 压缩 D_1 和 D_2，D_1 和 D_2 的并集就是 $\boldsymbol{G}_1$ 和 $\boldsymbol{G}_2$ 的并集的 DAG 压缩。相比之下，树压缩似乎不具备如此简便的组合性质，尽管当时尚无正式论证证明 DAG 比树更适合图压缩。

我们研究DAG 压缩而非树压缩的第二个原因是，“从算法角度看，这些压缩是免费获得的”，也就是说，已知可直接在压缩上运行的算法不仅适用于树形聚类，也适用于任意DAG

压缩。具体而言，Bannach 等人 [1] 证明，标准图算法深度优先搜索、广度优先搜索、强连通分量、拓扑排序，以及*Dijkstra* 算法都可以直接应用于DAG压缩图。至关重要的是，该实现的运行时间为 $O(s)$（对于 Dijkstra 算法则为 $O(s\log\ s)$），其中s是压缩的（较小）规模，而非原图的（较大）规模。Bannach 等人还表明，许多稠密图（例如孪生宽度有界的图）都存在DAG 压缩，其规模为 s，与待压缩图的顶点数呈线性关系，而非与通常呈二次方关系的边数呈线性关系。

先前工作中尚未解决的核心问题是：寻找给定图的（规模）最优压缩有多困难，也就是以下问题的复杂性：

▶**问题 1.1** (最小 DAG 压缩)。输入：一个有向图 $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$，$k\in\mathbb{N}$。问题：$\boldsymbol{G}$ 是否存在一个 DAG 压缩 $D=(V,A,E)$，使得 $|E|+|A|\leq k$?

我们的贡献。在本文中，我们首先扩展了可直接处理 DAG压缩图的算法清单，加入了另一项基础算法：我们证明，计算最小生成树（最小生成树）的 Kruskal 算法可以实现为适用于 DAG 压缩的算法。形式化地，我们有（α 为反阿克曼函数）：

▶ **定理1.2**。输入加权连通无向图 $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E},\boldsymbol{w})$的加权 DAG 压缩 $D=(V,A,E,w)$ 后，我们可以在 $O\big((|A|+|E|)\cdot\alpha(|\boldsymbol{V}|)+t_{\text{sort}}(|E|)\big)$ 时间内计算出 $\boldsymbol{G}$ 的一个最小生成树，其中 t_{sort} 是对权重数组进行排序所需的时间。

如果我们对权重一无所知，t_{sort} 是 $\Theta(|E|\log\ |E|)$，但如果已知权重可以例如通过桶排序处理，那么 t_{sort} 可能是线性的，因此可以忽略。

与 Bannach 等人的算法不同，[1], 这些算法“只是”标准算法，但应用于由DAG 压缩生成的特殊“切换图”，且相对于未压缩的图具有“距离保持”性质；我们的 Kruskal 算法则需要一些专门的算法改进。尤其是，这是首个可用于加权 DAG 压缩的算法。在这里，输入图的一条边可以由多条压缩边表示，而原图中一条边的权重 $\boldsymbol{e}$ 定义为表示压缩中 $\boldsymbol{e}$ 的任意压缩边 e 的最小权重。

我们要探讨的第二个问题是：DAG 压缩是否确实优于概念上更简单的树压缩。树压缩还有一项额外优势：我们可以仅用 $O(|\boldsymbol{V}|)$ 比特轻松存储整个树结构。我们证明答案是肯定的：考虑车图$:\{1,\dots,g\}\times\{1,\dots,g\}\ \big\{\big((r_1,c_1),(r_2,c_2)\big)\ |r_1=r_2\vee c_1=c_2\big\}$，之所以称为车图，是因为它对应于国际象棋中车的可能走法。我们可以轻松地对该图进行DAG压缩，使得 $|A|=2g^2$ 和 $|E|=g^2$。不过，通过一个非平凡的论证可以证明如下结论：

▶ **定理 1.3.** 设 $T=(V,A,E)$ 是车图 $\boldsymbol{R}_{g\times g}$的一种树压缩。那么，压缩边的数量 $|E|$ 至少为 $g^3/32-g^2$。

因此，具有 n 个顶点、$n=g^2$ 为 $\boldsymbol{R}_{g\times g}$ 的车图，其树压缩需要 $\Omega(n^{3/2})$ 条压缩边。由于原始边的数量为 $|\boldsymbol{E}|=2g^3=2n^{3/2}$，这意味着使用树形聚类无法实现显著压缩。相比之下，DAG 压缩对于 $s=|A|+|E|$ 总共只需要 $O(s)$ 条边。总而言之，使用 DAG 压缩车图，相比使用树压缩，效果可提高 $\sqrt{n}$ 倍。

我们要探讨并回答的第三个问题，也是核心问题，是计算最优 DAG 压缩有多困难。我们证明：

▶ **定理 1.4**。 最小 DAG 压缩是NP完全的 。

证明通过对集合覆盖问题进行非平凡归约得出。有趣的是，对于 DAG 压缩，“搜索不能归约为判定”，这意味着，即使我们获得了最小 DAG 压缩预言机，也不清楚这如何帮助我们计算最优DAG 压缩：通常的策略是不断向不断扩展的部分解中添加边，并查询预言机，以判断是否仍能维持最优解；但这种策略可能行不通，因为后续的图可能比中间图具有小得多的压缩规模。这引出了若干非平凡问题：当图中添加或删除一条边时，更新给定的最优 DAG压缩有多困难？[1] 我们证明以下两个问题都是 NP难的：

▶**问题 1.5** (min-dag-compression-add)。
输入： 一个图 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$，一个$DAG$压缩 $D = (V, A, E)$，它是 $\boldsymbol{G}$ 的压缩，且已知其大小最小为 $|A| + |E|$，另给定一条新边 $\boldsymbol{e} \in \boldsymbol{V} \times \boldsymbol{V}$ 和一个整数 $k \in \mathbb{N}$。问题： 是否存在一个DAG压缩 $D' = (V, A', E')$，它是$(\boldsymbol{V}, \boldsymbol{E} \cup \{\boldsymbol{e}\})$的压缩，且满足 $|A'| + |E'| \leq k$?

▶**问题 1.6** (min-dag-compression-delete)。
输入： 一个图 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$，一个$DAG$压缩 $D = (V, A, E)$，它是 $\boldsymbol{G}$ 的压缩，并保证其 e 最小规模为 $|A| + |E|$，一条边 $\boldsymbol{e} \in \boldsymbol{E}$，以及一个整数 $k \in \mathbb{N}$。
问题： 是否存在一个DAG压缩 $D' = (V, A', E')$，它是$(\boldsymbol{V}, \boldsymbol{E} \setminus \{\boldsymbol{e}\})$的压缩，且满足 $|A'| + |E'| \leq k$?

相关工作。 图压缩技术的研究历史悠久且成果丰硕，至今仍备受关注 [2, 3, 6, 7, 9, 14, 16]。本文研究的DAG 压缩数据结构最近在 [1] 中被提出。已有许多图压缩方法，其中一种与DAG 压缩尤为相近的方法在 [15] 中被提出。它们使用类似的“超节点”和“超边”，其中连接两个超节点的超边表示这两个超节点所含顶点之间的所有边。不过，该方法将顶点划分为超节点；与之相反，DAG 压缩允许对顶点进行嵌套聚类。 [13]中的方法与 [15], 相似，但加入了额外的边修正，以还原原图。另一种类似的方法是用于稠密图的可视化工具，称为幂图 [11], ，它使用模块通过一条边显示两个模块之间的所有边。模块可以嵌套，但如果两个模块之间存在重叠，其中一个必须完全包含在另一个之中；也就是说，按我们的术语，这种工具对应于树压缩。Dwyer 等人 [10]证明，寻找仅含单个模块的最小幂图是NP-难问题，并指出这种困难很可能也适用于更一般的情形。

DAG压缩最初受到有界孪生宽度图的启发。孪生宽度是 Bonnet 等人 [4] 引入的参数：这类图总是存在规模为线性的树压缩。事实上， [5], 引入的孪生模型，是树压缩的一种特殊情形。Bannach 等人 [1]证明，存在一些图（例如前面提到的车图）具有规模为线性的 DAG 压缩，但孪生宽度并非有界。这一结果表明，并且 [1], 中也提出了猜想，但并不能证明：车图不存在规模为线性的树压缩。

[1] 当然，如果我们能够高效地判定最小 DAG 压缩，那么 P = NP，而且构造最优DAG 压缩也会变得容易，因为此时问题“能否将某个部分DAG 压缩扩展为大小为 k DAG 的 $\boldsymbol{G}$? ”也属于 P。不过，这并未回答单次更新的难度问题。

关于寻找规模最小的DAG 压缩的复杂性，Bannach 等人 [1] 已经证明，给定输入 G 和一个数值 k，判定 G 是否存在具有 $|E| \leq k$ 的DAG 压缩（V, A, E）是NP难的。换言之，寻找压缩边数量最少的压缩是困难的——但正如 [1]，中已经指出的，这“并非真正的问题”，因为对（V, A, E）的任何编码都必须同时编码 A，所以以牺牲 $|A|$ 为代价来最小化$|E|$，只有理论意义。当然，直观上看，单独最小化 $|E|$ 很难，而最小化$|A| + |E|$ 却突然变得容易，似乎不太可能（尽管并非不可能）；本文的结果表明，要证明这种直觉正确，复杂性出乎意料地高。

本文结构。 在第2节介绍必要的术语后，我们将分别用一节讨论此前的三个定理：在第3节，我们通过展示如何利用 DAG 压缩更高效地求出最小生成树，证明定理 1.2。在第4节，我们通过证明车图的任意树压缩大小的下界，说明 DAG 压缩可以得到严格小于树压缩的压缩结果，从而证明定理 1.3。最后，我们在第5节讨论计算和更新最优 DAG 压缩的难处理性，并证明定理 1.4。移至附录的证明在正文中以“▼”替代。

2 背景

本文中，（有向）图是一个有序对 $G = (V, E)$，由顶点集合 V和边关系 $E \subseteq V \times V$组成。加权图是一个三元组 $G = (V, E, w)$，其中 $w\colon E \to \mathbb{N}$。图 $G = (V, E)$中的孪生点是指一对顶点 t_1, t_2，它们的入邻域和出邻域相同，也就是说，$\{v \mid (v, t_1) \in E\} = \{v \mid (v, t_2) \in E\}$ 且 $\{v \mid (t_1, v) \in E\} = \{v \mid (t_2, v) \in E\}$。图中从 v_1 到 v_k 、长度为 k 的游走，是由顶点组成的序列 $(v_1, \ldots, v_k)$，满足对所有 $i \in \{1, \ldots, k-1\}$，$(v_i, v_{i+1}) \in E$ 成立；路径是所有顶点互不相同的游走；环是所有顶点互不相同（首尾顶点必须相同）的游走。若图中没有环，则称其为无环图；按照惯例，我们将无环图称为 DAG（有向无环图）。树是一种 DAG，其中存在唯一的根 $r \in V$，使得对于所有顶点 $v \in V$，从 r 到 v都存在唯一一条路径。森林是若干棵顶点互不相交的树的并。DAG 中的汇点是出度为 0 的顶点 $v \in V$，也就是说，不存在顶点 u 使得 $(v, u) \in E$。树的汇点也称为叶节点。

回顾引言中对DAG 压缩的定义：我们从一个“普通、通常较为稠密”的图 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ 开始，为了更好地区分它与压缩中使用的顶点和边，我们用粗体表示该图。我们使用一个 DAG 压缩 [1],对其进行压缩，该压缩是一个三元组 $D = (V, A, E)$，由以下部分组成：

1. 一个簇 $DAG\,(V, A)$，其汇点集合恰好为 $\boldsymbol{V}$。集合 A 就是簇DAG的边关系，但为了在表述上与 $\boldsymbol{E}$ 中的普通边区分开来，我们将其中的边 A 称为弧。我们为每个簇顶点 $v \in V$ 关联一个簇 $\boldsymbol{C}_D(v) \subseteq \boldsymbol{V}$（或者在 D 的含义明确时，简称为 $\boldsymbol{C}(v)$），定义为$\boldsymbol{C}(v) := \{u \in V \mid u$ 可从 v到达$\}$。注意，对所有$\boldsymbol{v} \in \boldsymbol{V} \subseteq V$，$\boldsymbol{C}(\boldsymbol{v}) = \{\boldsymbol{v}\}$ 均成立，因此每个原始顶点的单元素集合都可作为一个簇。
2. 由 $E \subseteq V \times V$ 条压缩边 组成的集合，其中包含 $\boldsymbol{E} = \bigcup_{(u,\ v) \in E} \boldsymbol{C}(u) \times \boldsymbol{C}(v)$。每条连接两个簇顶点的压缩边 $(u,\ v) \in E$ 表示：在 $\boldsymbol{G}$ 中，簇 $\boldsymbol{C}(u)$中的所有顶点都与簇 $\boldsymbol{C}(v)$中的所有顶点相连。如果这两个簇不相交，那么 $\boldsymbol{C}(u) \times \boldsymbol{C}(v)$ 是一个双团 （完全二部图）。如果这两个簇有重叠，我们称 $\boldsymbol{C}(u) \times \boldsymbol{C}(v)$ 为积（由 u的簇和 v的簇构成）。

6 关于 DAG 压缩图基本问题的复杂性

我们将大小 定义为 DAG 压缩 $D = (V,\ A,\ E)$ 的大小，即 $|A| + |E|$。（我们不将 $|V|$ 计入大小的原因是， $V \setminus \boldsymbol{V}$ 不可能包含汇点，因此，所有“编码簇DAG 的成本都已通过计入 $|A|$而得到合理补偿”。）

上述定义很容易推广到无向图。无向图是形如 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ 的二元组，其中 $\boldsymbol{V}$ 是顶点集合， $\boldsymbol{E} \subseteq \{\{\boldsymbol{u}, \boldsymbol{v}\} \mid \boldsymbol{u}, \boldsymbol{v} \in \boldsymbol{V}\}$ 是无向边集合。无向树是一种无向图，其中从根顶点到每个其他顶点都存在唯一一条路径。对于两个集合 A 和 B，令 $A \otimes B$ 表示集合 $\{\{a, b\} \mid a \in A, b \in B\}$。那么，无向 DAG 压缩就是一个三元组 $D = (V, A, E)$，由一个（仍然有向的）簇 DAG (V, A)（其汇点集合恰好为 $\boldsymbol{V}$）以及无向压缩边集合 $E \subseteq V \otimes V$ 构成，其中 $\boldsymbol{E} = \bigcup_{\{u,v\} \in E} \boldsymbol{C}(u) \otimes \boldsymbol{C}(v)$；簇和簇顶点的定义与有向情形相同。

在最小生成树的背景下，输入是带权无向图，它们是三元组 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{w})$，其中 $\boldsymbol{w}$: $\boldsymbol{E} \to \mathbb{N}$。 $\boldsymbol{G}$ 的（无向）加权 DAG 压缩是一个四元组 $D = (V, A, E, w)$，满足 (V, A, E) 是 $(\boldsymbol{V}, \boldsymbol{E})$ 的 DAG 压缩，并且对于每个 $\{\boldsymbol{x}, \boldsymbol{y}\} \in \boldsymbol{E}$，都有 $\boldsymbol{w}(\{\boldsymbol{x}, \boldsymbol{y}\}) = \min_{\{u,v\} \in E, \boldsymbol{x} \in C(u), \boldsymbol{y} \in C(v)} w(\{u, v\})$。

在常数时间内查找汇点。对于我们的一些算法，快速获取簇顶点 $v \in V$ 的“ $\boldsymbol{C}(v)$ 中某个任意元素”会很有用；也就是说，获取一个从 v 可达的汇点，该汇点位于 $(V,\ A)$ 中。原则上，这很容易实现；不过，要在 $O(1)$ 时间内执行此操作，需要进行一些初始预处理：

▶ **引理 2.1 (▼)**。 给定一个DAG压缩 $D = (V, A, E)$ 以及 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$，我们可以在 $O(|A| + |V|)$ 时间内计算出一个函数 $\boldsymbol{c}$: $V \to \boldsymbol{V}$，使得对所有 $v \in V$， $\boldsymbol{c}(v) \in \boldsymbol{C}(v)$ 。

3 在 DAG 压缩图上计算最小生成树

假设给定一个图 $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ 的DAG 压缩 $D = (V, A, E)$，并希望解决一个标准问题，例如计算 $\boldsymbol{G}$ 的强连通分量。当然，我们可以先解压缩 D，再对 $\boldsymbol{G}$ 执行深度优先搜索（DFS），但这样就违背了压缩图的初衷。相反，我们需要以DFS 的方式实现DFS，使其能够直接在 D 上运行，而无需计算 $\boldsymbol{G}$ ——并且最好让运行时间与 D 的大小呈线性关系，而不是与 $\boldsymbol{G}$ 的边数呈线性关系。正如 Bannach 等人 [1]，所示，包括 DFS 在内的多种算法都可以这样实现。

接下来，我们在这份清单中再加入一种标准算法，即一种用于计算最小生成树的 Kruskal 算法变体，它可以直接在加权连通无向图$\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E}, \boldsymbol{w})$的加权无向 DAG 压缩 $D = (V, A, E, w)$上运行。回顾一下， $\boldsymbol{G}$ 的生成树 $\boldsymbol{T} = (\boldsymbol{V}, \boldsymbol{E}')$是一棵包含$\boldsymbol{E}' \subseteq \boldsymbol{E}$的无向树。 $\boldsymbol{T}$的权重 为 $\sum_{\boldsymbol{e} \in \boldsymbol{E}'} \boldsymbol{w}(\boldsymbol{e})$。最小生成树 (MST)是权重最小的生成树。显然，只有连通图才有生成树。对于非连通图， $\boldsymbol{G}$ 的（最小）生成森林 是一个无向森林，其中的树分别是 $\boldsymbol{G}$各个连通分量的最小生成树。Kruskal 算法 [12] （用于未压缩图）见算法 1；关于并查集数据结构的更多背景知识，请参见 B 节。主循环中始终保持的不变量是：

▶ **不变式 3.1**。 （$\boldsymbol{V}$、 $\boldsymbol{M}$）是（$\boldsymbol{V}$、$\{\boldsymbol{e}_1$、.....、 $\boldsymbol{e}_i\}$）的最小生成森林。

在 DAG压缩图上运行 Kruskal 算法。我们为 DAG 压缩图设计的 Kruskal 算法思路很简单：遍历压缩边，而不是普通边，并以某种方式高效处理由压缩边表示的边，见算法 2。

算法 1 Kruskal 算法，用于计算连通无向图 (V, M) 的最小生成树，其中图为 (V, E)，各条边依次为 (u_1, v_1) 至 (u_m, v_m)。全局变量M 存储当前已计算出的解的部分，而全局变量 P 是并查集数据结构，当前保存着 V 的一个划分，使得 (V, M) 的连通分量恰好是 P 中的各个集合。

1 算法 kruskal($V, E = \{\{u_1, v_1\} \ldots, \{u_m, v_m\}\}$ w) 2 按权重对 E 排序，使得对于 $i < j$，有 $w(\{u_i, v_i\}) \leq w(\{u_j, v_j\})$3 $M \leftarrow \emptyset$4 $P \leftarrow$ initialize-union-find(V) 5 对每个 $i \in \{1, \ldots, m\}$ 执行 6 调用 add-edge(u_i, v_i) 7 返回 M8 9 算法 add-edge(u, v) 10 如果 find(u) $\neq$find(v)，则 11 $M \leftarrow M \cup \{\{u, v\}\}$ 12 P.unite(u, v)

算法 2 将 Kruskal 算法调整为适用于 DAG 压缩图。注意，P 是 V 的一个划分（而不是 V的划分）。

1 算法 ***kruskal-dag-compressed***$(V, A, E = \{\{u_1, v_1\} \ldots, \{u_m, v_m\}\}$ w***)*** ***2*** 按权重对 E 排序，使得对于 $i < j$，有 $w(\{u_i, v_i\}) \leq w(\{u_j, v_j\})$3 $M \leftarrow \emptyset$4 $P \leftarrow$ ***initialize-union-find(V)*** // V是 (V, A) 的汇点 ***5*** 对每个 $i \in \{1, \ldots, m\}$ 执行 ***6*** 调用 ***add-edges-represented-by***(u_i, v_i) // 在下文的算法***3***中实现 ***7*** 返回 M

不变式 3.1 的对应表述——现在我们需要证明它在算法 2 的主循环期间始终成立——应为 M 是 $\big(V, (C(u_1) \otimes C(v_1)) \cup \cdots \cup (C(u_i) \otimes C(v_i))\big)$的最小生成森林。不过，我们将证明以下稍有不同的不变式成立，这会让我们的证明更灵活一些：

▶ **不变式 3.2**。(V, M) 是$\big(V, E'\big)$的最小生成森林，其中 E' 是一个集合，满足 $E \supseteq E' \supseteq (C(u_1) \otimes C(v_1)) \cup \cdots \cup (C(u_i) \otimes C(v_i))$。

换句话说，我们的不变式仅仅指出，我们始终存储某个边集合的最小生成森林；该集合至少包含前i 轮处理过的所有未压缩边，至多包含所有未压缩边。因此，最终我们得到整个图（V，E）的最小生成森林，从而得到一棵最小生成树。

实现 add-edges-represented-by 显而易见的问题在于，每条压缩边 (u, v) 都对应着一个由原始边构成的完整集合 $C(u) \otimes C(v)$（最多有 $\binom{n}{2} + n$ 条），而我们未必能自由选择将哪些边添加到 最小生成树 中，因为 $C(u)$ 和 $C(v)$ 的某些部分可能已经属于更大的集合 $U \in P$ ——也可能不是。更复杂的是，$C(u)$ 和 $C(v)$ 可能相交，也可能只与其中某些 $U \in P$ 部分相交。事实上，甚至仅仅针对每条边 $\{u, v\} \in E$ 计算 $C(u)$ 和 $C(v)$ 都会耗费太多时间。为了解决这些问题，我们采用一个简单的定义：

▶ **定义3.3. </style}顶点 是干净** 的，当且仅当 $C(v) \subseteq U$ 对于某个 $U \in P$成立。

换句话说，干净顶点的簇必须完全包含在划分 P 中的某一个集合内。假设我们能够轻松地确保一个顶点变得干净，那么实现 add-edges-represented-by 就很简单，见算法3。

算法3 处理 $C(u) \otimes C(v)$ 中的所有边。回顾一下，根据引理 2.1，我们可以在 $O(1)$ 时间内计算出 $C(v)$ 中的任意顶点 $c(v)$。

```
1 算法 add-edges-represented-by(u, v)2 调用
make-clean(u, c(v))3 调用 make-clean(v, c(u))4
调用 add-edge(c(u) c(v))
```

暂且假设两次 make-clean 调用确保在调用 add-edge 时，u 和 v 都是已清理的。接下来这个引理将证明该调用是正确的：

▶ **引理3.4** (▼)。假设不变式 *3.2* 对 $i-1$ 成立，并且我们对已清理顶点 u_i 和 v_i 执行 ADD-EDGE$(c(u_i)\ c(v_i))$ 。那么此后该不变式仍对 i 成立。

请注意，调用 unite 不会导致任何已清理顶点失去这一状态。

当然，并非所有顶点在开始时都是 clean：事实上，在 kruskal-dag-compressed 开始时，我们只将 $v.clean$ 初始化为 $true$，将 $v \in V$ 初始化为 $unknown$，并将所有 $v \in V \setminus V$ 初始化为未知状态。幸运的是，在处理压缩边 $\{u,v\} \in E$时，可以通过一种简单的递归方式使顶点变为 clean；具体实现见算法 4，示例见第 23 页的图7。

算法 4 在添加压缩边时确保顶点为 clean 的递归过程。

```
1 算法 make-clean(v, r) // 前置条件： r ∈ V和 C
2 (v) ⊗ {r} ⊆ E如果 v.clean ≠true，则对每个
3 w ∈ V满足 (v, w) ∈ A 执行调用
4 make-clean(w, r) 调用 add-edge(c(w) r)v.
5 clean ←true
6
7
```

▶ **引理3.5** (▼)。假设不变式 *3.2* 对 $i-1$ 成立，并执行 MAKE-CLEAN$(v_i, c(u_i))$。那么之后 v_i 将变为 *clean*，并且不变式对 $i-1$仍然成立。

总而言之，我们得到了引言中的定理1.2：

▷ 断言（定理1.2）。输入一个加权DAG 压缩 $D = (V, A, E, w)$，它表示一个加权连通无向图 $G = (V, E, w)$，我们可以在$O\big((|A|+|E|)\cdot \alpha(|V|) + t_{\text{sort}}(|E|)\big)$时间内计算其最小生成树 G，其中 t_{sort} 是对权重数组进行排序所需的时间。

证明。我们运行算法 2，其正确性由引理 3.4 和引理 3.5 得出。接下来需要说明运行时间符合预期：首先，根据引理 2.1，c 的预计算需要$O(|A|+|E|+|V|)$时间；由于图是连通的，这为 $O(|A|+|E|)$。其次，注意在算法的整个运行过程中，无论是直接调用算法 4，还是通过递归调用，在任何一次调用中，A 中的每条边都至多处理一次，从而调用一次添加边。同样，每条边 $\{u_i, v_i\} \in E$ 也只处理一次，并调用一次添加边。由于添加边 需要 $\alpha(|V|)$摊还时间，因此得到所声称的运行时间。◀

4 树压缩规模的下界

树压缩是 DAG 压缩 (V, A, E)，其中 (V, A) 是一棵树。除了概念上更简单之外，它们还支持更高效的编码（n 个顶点的树可以轻松地用 $O(n)$ 位编码），因此引出了一个问题：DAG 压缩是否比树压缩更有优势。早期研究 [1] 曾猜想情况可能如此（直觉也强烈支持这一点），但一直未能找到证明。我们将在下文证明定理 1.3，从而解决这个问题。定理内容如下：设 $T = (V, A, E)$ 是车图 $\boldsymbol{R}_{g\times g}$ 的一种树压缩。那么，压缩边的数量 $|E|$ 至少为 $g^3/32 - g^2$。回顾一下，n 个顶点的车图 $\boldsymbol{R}_{g\times g}$ 具有一个大小为 $g \times g$ 的网格，其顶点为 $g = \sqrt{n}$；当且仅当两个顶点位于同一行或同一列时，它们之间才有边。容易构造一个大小为 $|A| + |E| = 2g^2 + 2g$ 的 DAG 压缩 (V, A, E) 来表示 $\boldsymbol{R}_{g\times g}$：具体而言，为每个 $i \in \{1, \ldots, g\}$ 引入一个行簇顶点 r_i，并引入一个列簇顶点 c_i；在 A 中，将每个 r_i 与网格中第 i 行的所有顶点相连，将每个 c_i 与第 c_i 列的所有顶点相连，并对所有 $i \in \{1, \ldots, g\}$，将压缩环 (r_i, r_i) 和 (c_i, c_i) 加入 E，以表示行团和列团。总之，车图存在大小为 $O(n)$ 的 DAG 压缩，而定理 1.3 表明，任何树压缩的大小都至少为 $|A| + |E| \in \Omega(n^{3/2})$。

证明需要一个引理：

▶ **引理 4.1 (▼)**。对于 $\boldsymbol{R}_{g\times g} = (\boldsymbol{V}, \boldsymbol{E})$ 设 $\boldsymbol{X}, \boldsymbol{X}' \subseteq \boldsymbol{V}$ 为满足 $|\boldsymbol{X}| \geq 3$、$|\boldsymbol{X}'| \geq 3$ 和 $\boldsymbol{X} \times \boldsymbol{X}' \subseteq \boldsymbol{E}$ 的集合。那么，$\boldsymbol{X} \cup \boldsymbol{X}'$ 中的所有顶点都位于同一行或同一列。

定理 1.3 的证明。 设 $T = (V, A, E)$ 是车图 $\boldsymbol{R}_{g\times g}$ 的一个树压缩，且 $|E|$ 是最小的。令 $\boldsymbol{V}$ 表示 $\boldsymbol{R}_{g\times g}$ 的顶点集，也就是该 $g \times g$ 网格。我们可以假设簇树 $T = (V, A)$ 是一棵二叉树，即每个簇顶点 $v \in V$ 恰好有两个子节点，或者是叶节点（此时是 $\boldsymbol{V}$ 的一个元素），因为我们可以轻易地调整 A 使其满足此条件，同时不改变 E。

若簇顶点 $v \in V$ 是大 $|\boldsymbol{C}(v)| \geq 3$，则称其为大顶点。若 $\boldsymbol{C}(v)$ 中的所有顶点都位于同一行，则大顶点称为水平；若它们都位于同一列，则称为垂直。既非水平也非垂直的大顶点，若存在一对 $(\boldsymbol{r}_0, \boldsymbol{c}_0)$，称为 v 的交叉中心，使得所有 $(\boldsymbol{r}, \boldsymbol{c}) \in \boldsymbol{C}(v)$ 都具有 $\boldsymbol{r} = \boldsymbol{r}_0$ 或 $\boldsymbol{c} = \boldsymbol{c}_0$，则称为交叉顶点。注意，当树中 v 是 u 的父节点，且 v 和 u 都是交叉顶点时，v 和 u 具有相同的交叉中心。图 2 展示了一个示例。

我们可以直接得到以下几个观察结果：

1. 如果 $(u, v) \in E$ 是连接两个大顶点的压缩边，那么 $\boldsymbol{C}(u) \cup \boldsymbol{C}(v)$ 中的所有顶点必定都位于同一行或同一列（这是引理 4.1 的直接推论）。

2. 如果 $(u, v) \in E$ 连接一个交叉顶点 u 和一个顶点 v，那么 v 必定是 u 的交叉中心（因为 u 既非水平也非垂直，它包含来自不同行和不同列的顶点，因此只能连接到交叉中心）。此外，我们可以假设 u 的父节点 p 不是交叉顶点（因为在 E 中用 (p, v) 替换 (u, v)，只会得到一个可能更小的树压缩；回想一下，u 和 p 必须具有相同的交叉中心）。

3. 不可能存在 $(u, v) \in E$ 其中 u 很大，但甚至不是交叉顶点。

现在我们来计算 $\boldsymbol{E}$ 中有多少条边可以由 E 中的压缩边覆盖。我们首先考虑连接交叉顶点 u 及其交叉中心 v 的边 $(u, v) \in E$：这样的压缩边在 $\boldsymbol{E}$ 中至多可以表示 $|\boldsymbol{C}(u)|$ 条边。关键在于，正如我们所论证的，u 将是一个极大交叉（并且不可能在朝向根的更高处还有交叉：一旦失去交叉的性质，就无法重新获得），集合 $\boldsymbol{C}(u)$ 两两

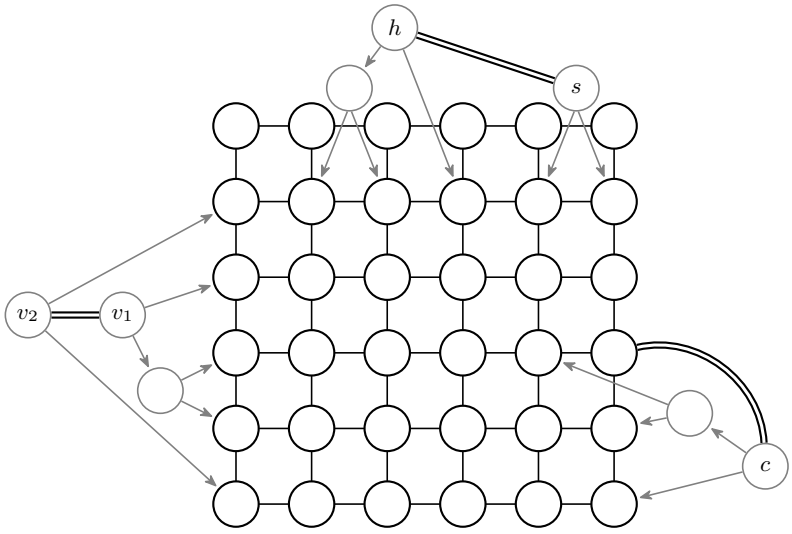


图 2 车图部分树压缩的示例，其中包含 $g = 6$。为便于阅读，行团和列团仅用车图中 $\boldsymbol{E}$ 直接邻居之间的边表示。顶点 v_1 和 v_2 是垂直的，顶点 h 是水平的，但 s 不是水平的，因为它不是大顶点。交叉顶点 c通过一条压缩边连接到它的交叉中心。

不相交。也就是说，它们的大小之和至多为 $|\boldsymbol{V}| = g^2$。总而言之，涉及交叉顶点的压缩边 $(u,\ v)$ 至多可以表示 $\boldsymbol{E}$中的 g^2 条边。

接下来，对于给定的行 $\boldsymbol{r} \in \{1, \ldots, g\}$，考虑该行中所有满足 $\boldsymbol{c} \in \{1, \ldots, g\}$ 的顶点 $(\boldsymbol{r}, \boldsymbol{c})$，并找出作为 $(\boldsymbol{r}, \boldsymbol{c})$ 祖先的第一个大顶点 v。直观地说，如果 v 是水平顶点，这有助于压缩该行中的边，因此我们来统计¯不满足这一条件的顶点 v 的数量，并将此数量记为 $h_{\boldsymbol{r}}$。同理，将 $\bar{v}_{\boldsymbol{c}}$ 定义为列 $\boldsymbol{c}$ 中第一个大祖先不是垂直顶点的顶点数量。由于顶点 $(\boldsymbol{r}, \boldsymbol{c})$ 的第一个大祖先不可能同时既是水平顶点¯又是垂直顶点，因此每个顶点 $\boldsymbol{V}$ 都会计入某个 $h_{\boldsymbol{r}}$ 或 $\bar{v}_{\boldsymbol{c}}$。特别地，$\sum_{\boldsymbol{r} \in \{1,\ldots,g\}} \bar{h}_{\boldsymbol{r}} + \sum_{\boldsymbol{c} \in \{1,\ldots,g\}} \bar{v}_{\boldsymbol{c}} \geq g^2$。这意味着至少有一个和为 $g^2/2$，不妨设为第一个（另一种情况对称）。

现在我们有 $\sum_{\boldsymbol{r} \in \{1,\ldots,g\}} \bar{h}_{\boldsymbol{r}} \geq g^2/2$。这意味着，$g$ 个数的和至少为 $g^2/2$，其中每个数都不超过 g。只有至少一半的数不小于 $g/2$时，这才有可能。因此，$g/2$ 行中至少有一行满足 $h_{\boldsymbol{r}} \geq g/2$。现在考虑这样一行：至少有 $g/2$ 个顶点 $\boldsymbol{v}$没有以水平顶点作为其第一个大祖先。考虑它们之间 $\boldsymbol{E}$ 中的至少 $g^2/4$ 条边，以及这些边如何用压缩边表示：它们不能表示为两个大顶点之间的压缩边，因为这两个顶点必须是水平顶点，但 $\boldsymbol{v}$ 的祖先中没有水平顶点。它们也不能表示为包含一条非交叉大边的压缩顶点。剩下的只有 V 中非大顶点之间的压缩边，以及交叉顶点与其交叉中心之间的压缩边。

总之，我们得出需要使用非大型簇顶点之间以及交叉顶点与其交叉中心之间的压缩边，覆盖至少 $(g^2/4) \cdot (g/2) = g^3/8$ 条边，位于 $\boldsymbol{E}$ 中。我们已经论证，交叉顶点与其交叉中心之间的压缩边最多只能贡献 条边。现在，由于两个小簇顶点之间的一条压缩边最多只能贡献 *4* 条边，因此至少需要 $(g^3/8)/4 - g^2 = g^3/32 - g^2$ 条压缩边，位于 E。◀

根据上述结果，一个包含 n 个顶点的图，其 DAG 压缩大小可以达到 $O(n)$，而

最佳树压缩的大小为 $\Theta(n^{3/2})$，这意味着压缩效果提升了 $\sqrt{n}$ 倍。我们还能做得更好吗？由于一个图最多可以有 $O(n^2)$ 条边，理论上的最大提升倍数为 n。我们认为，通过将车图的论证推广到更高维的车图，可以任意接近这一倍数。例如，设 $\boldsymbol{R}_{g\times g\times g}$ 为一个图，其顶点构成一个三维网格，且当且仅当 $\boldsymbol{x}=\boldsymbol{x}'$ 或 $\boldsymbol{y}=\boldsymbol{y}'$ 或 $\boldsymbol{z}=\boldsymbol{z}'$ 成立时，$(\boldsymbol{x},\boldsymbol{y},\boldsymbol{z})$ 与 $(\boldsymbol{x}',\boldsymbol{y}',\boldsymbol{z}')$ 之间有一条边。同样，很容易找到该图的一个大小为 $O(g^3)=O(n)$ 的 DAG 压缩，但我们认为，可以调整关于$\boldsymbol{R}_{g\times g}$ 的论证，从而证明 $\boldsymbol{R}_{g\times g\times g}$ 的任何树压缩大小都为 $\Omega(g^5)=n^{5/3}$。然而，这一论证似乎并不直接，因此我们提出如下猜想：

▶**猜想 4.2.** 任意对 d 维车图 $\boldsymbol{R}_{g^d}$ 进行的树压缩，其大小均为$\Omega(n^{(2d-1)/d})$。

5 计算和更新 DAG 压缩的困难性

先前的工作和早期结果都表明，DAG 压缩为在大型稠密图上高效运行算法提供了一种途径。显然，首先需要获得DAG 压缩。对于某些应用而言，如果图是通过算法生成的，这可能很容易做到；但一般情况下，我们会得到一个图 $\boldsymbol{G}$ 作为输入，并需要计算尽可能小的DAG 压缩(V,A,E)。不幸的是，接下来我们将证明定理 1.4，该定理指出最小 DAG 压缩是NP-完全的。

证明通过从集合覆盖问题进行归约得到。给定一个集合 $T=\{S_1,\dots,S_n\}$，回顾一下，集合 V的覆盖是一个子集 $X\subseteq T$，满足 $V\subseteq\bigcup X$。

▶**问题 5.1**（集合覆盖）。
输入：一个全集 U，一个集合族 $T=\{S_1,\dots,S_n\}$，其中各元素是 U 的子集，$k\in\mathbb{N}$。问题：是否存在一个覆盖 $X\subseteq T$，使其覆盖 U，且大小为 $|X|\le k$?

接下来，我们将首先证明若干关于最优DAG 压缩性质的一般性引理；我们认为这些引理本身也很有趣，因为它们让我们进一步了解DAG 压缩的能力与局限。之后将给出定理 1.4 的证明。本节末尾，我们将介绍该定理在动态场景下的一个变体，这表明，不仅“从头开始”计算最优DAG 压缩很困难，即使在不改变簇DAG的情况下更新一个已有的压缩也很困难。

最优 DAG 压缩的性质。 下文证明的引理均指出，对于所有图 $\boldsymbol{G}$（可能满足某些限制），都存在 一个具有某些性质的最优 DAG压缩。证明总是先考虑一个违反所述性质的最优 DAG 压缩，然后论证我们可以通过稍微修改压缩来“修复”这一违反，从而得到一个仍然最优的新DAG 压缩。通过可能重复这一处理，我们可以得到一个满足该性质的最优 DAG压缩。对于第一个性质，回想一下，图 $G=(V,E)$中的一个孪生点 是指一对顶点 t_1,t_2，它们的入邻域和出邻域相同；图 3 左侧给出了一个例子。

▶ **引理 5.2**（孪生点仍为孪生点 ▼）。 每个图 $\boldsymbol{G}$ 都有一种最优的 DAG 压缩（V,A,E），使得 $\boldsymbol{G}$ 中的所有孪生点在（V,A）和（V,E）中也都是孪生点。

第二个性质涉及“有向二分图”的压缩所构成的簇 DAG 。这类图的顶点集可以划分为两侧 $\boldsymbol{V}_1$ 和 $\boldsymbol{V}_2$，并且边只从 $\boldsymbol{V}_1$ 指向 $\boldsymbol{V}_2$，也就是说，$\boldsymbol{E}\subseteq\boldsymbol{V}_1\times\boldsymbol{V}_2$。示例见图 4。

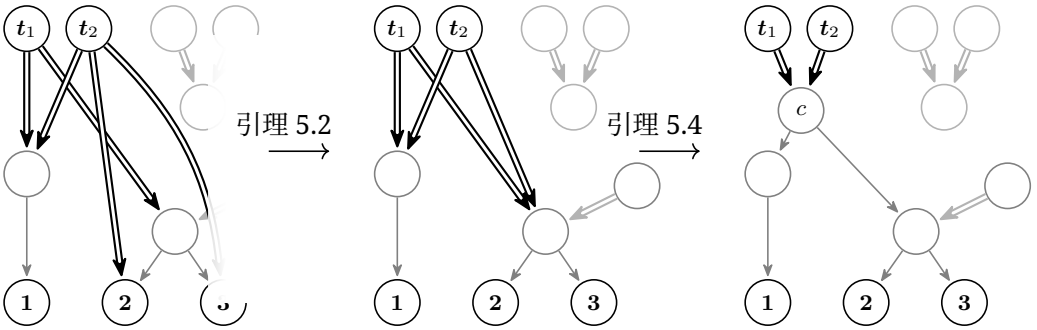


图 3 保持最优 DAG 压缩（V， A， E）的变换步骤。第一步中，我们将（$\boldsymbol{V}$， $\boldsymbol{E}$）中的孪生点 $\boldsymbol{t}_1$ 和 $\boldsymbol{t}_2$，通过将总度数较小的孪生点镜像到总度数较大的孪生点，使其也分别成为（V， A）和（V， E）中的孪生点。第二步中，我们添加一个新的簇顶点 c 并重新连接边，以减少与这些孪生点关联的压缩边数量。

▶ **引理 5.3** (簇仅位于一侧 ▼)。 每个有向二分图 $\boldsymbol{G} = (\boldsymbol{V}_1 \cup \boldsymbol{V}_2, \boldsymbol{E})$都存在一个最优 DAG 压缩 (V, A, E)，使得对于所有 $v \in V \setminus \boldsymbol{V}_1$，都有 $\boldsymbol{C}(v) \subseteq \boldsymbol{V}_2$。

注意，DAG 引理 5.3 中的压缩具有这样的性质：所有压缩边都从 $\boldsymbol{V}_1$ 中的一个顶点指向 $V \setminus \boldsymbol{V}_1$中的一个簇顶点。

第三个引理是前两个引理的一个有趣结合，例子见图 3 右侧：

▶ **引理 5.4** (孪生点只有一条压缩边 ▼)。 每个有向二分图$\boldsymbol{G} = (\boldsymbol{V}_1 \cup \boldsymbol{V}_2, \boldsymbol{E})$ 都存在一个最优的 DAG 压缩(V, A, E)，使得所有孪生点 $\boldsymbol{t}_1, \boldsymbol{t}_2 \in \boldsymbol{V}_1$至多有一条关联的压缩边。

归约。 现在，我们已基本准备好介绍从集合覆盖到最小 DAG 压缩的归约，并证明定理 1.4。还需要最后一个定义和引理：将 T 个 U 的子集组成的子集族编码为图的标准方法，是构造二分关联图，其中一侧是 U，另一侧为每个$S \in T$ 设置一个顶点 a_S，并从每个 a_S 向 S的所有元素连边。对我们而言，使用关联图的“孪生”版本会很有用，其中 a_S 额外拥有一个孪生点 b_S:

▶ **定义 5.5**。 设 T 是 U的一个子集族。 T的孪生关联图$\boldsymbol{T} = (\boldsymbol{V}_1 \cup \boldsymbol{V}_2, \boldsymbol{E})$ 是有向二分图，其顶点集为 $\boldsymbol{V}_1 = \{\boldsymbol{a}_S \mid S \in T\} \cup \{\boldsymbol{b}_S \mid S \in T\}, \boldsymbol{V}_2 = U$，其边集为 $\boldsymbol{E} = \{(\boldsymbol{a}_S, \boldsymbol{s}) \mid \boldsymbol{s} \in S \in T\} \cup \{(\boldsymbol{b}_S, \boldsymbol{s}) \mid \boldsymbol{s} \in S \in T\}$。

▶ **引理 5.6** (▼)。 设 T 是 U 的一个子集族。对于一个不是任何 $S \in T$ 的子集的集合 $R \subseteq U$，令 $T\prime = T \cup \{R\}$。令 $k_=$ 为满足 $\bigcup X_= = R$ 的集合 $X_= \subseteq T$ 的最小大小（若不存在这样的集合，则令 $k_= = \infty$），并令 $k_\supseteq$ 为满足 $\bigcup X_\supseteq \supseteq R$ 的集合 $X_\supseteq \subseteq T$ 的最小大小。令 s 和 s' 分别为 T 和 T' 的孪生关联图的最优 DAG 压缩大小。则 $s + k_\supseteq + 2 \leq s' \leq s + k_= + 2$。

定理 1.4 的证明。 我们只需通过将集合覆盖归约到最小 DAG 压缩，证明其 NP-困难性。设 T 是 U 的一个输入集合族，且 k 是一个数值。我们可以假设一些平凡情况已得到处理，即 $\bigcup T = U$（也就是说， U 确实可以被覆盖），并且 $U \notin T$。我们还假设 $U = \{1, \ldots, n\}$ 成立。

关键思路是通过将 T替换为 $\{\{s\} \mid s \in T\} \cup \bigcup_{i=1}^{n} \{S \cap \{1, \ldots, i\} \mid S \in T\} \setminus \{\emptyset\}$，为其施加一些结构。注意，我们实质上向 T添加了所有单元素集合以及

所有“ $S \in T$的所有非空初始段”，其中初始段是相对于全集 U的顺序而言的。例如， $T = \{\{1, 4, 5, 6\}, \{2, 3, 5, 7\}\}$ 会被替换为$\{\{1\} \ldots, \{7\}, \{1, 4\}, \{2, 3\}, \{1, 4, 5\}, \{2, 3, 5\}, \{1, 4, 5, 6\}, \{2, 3, 5, 7\}\}$。注意，对于在 T中用尽可能少的集合覆盖 U 而言，这些新增集合毫无帮助，因此原始 T 属于集合覆盖，当且仅当新的实例也属于集合覆盖。对于所得实例 T，我们将元素编号为 $T = \{S_1, \ldots, S_m\}$，其中 S_i 按标准顺序排列；也就是说，前 S_1到 S_n 个集合的大小为 1，接着是所有大小为 2 的 S_i ，然后是所有大小为 3 的集合，依此类推。

令 $T_i := \{S_1, \ldots, S_i\}$ 为仅包含前 i 个集合的集合族，令 $\boldsymbol{T}_i$ 为 T_i 的孪生关联图，并令 s_i 为 $\boldsymbol{T}_i$ 的最优 DAG 压缩的大小。显然， $s_n = 2n$ （对于每个 $\{j\}$， $\in T_n$ 中的 $\boldsymbol{a}_{\{j\}}$ 和 $\boldsymbol{b}_{\{j\}}$ 都可以且必须通过一条压缩边连接到 $j \in U$）。由引理 5.6 可知，对于所有 $i \in \{n+1, \ldots, m\}$，都有 $s_{i-1} + X_{\supseteq} + 2 \leq s_i \leq s_{i-1} + X_{=} + 2$。然而，对于这些 i，我们有 $|X_{\supseteq}| = |X_m| = 2$，因为任意一个 S_i 都恰好可以由 T_{i-1} 中的两个集合覆盖，即缺少最高编号 $j \in S_i$ 的 S_i 初始段以及单元素集合 $S_j = \{j\}$，但无法由 T_{i-1} 中的任何单个集合覆盖。总而言之，对于 $i \in \{n+1, \ldots, m\}$，有 $s_i = s_{i-1} + 4$，因此 $s_m = 2n + 4$ （ $m - n - 1$） $= 4m - 2n - 4$。

现在，这个归约要判断 $T \cup \{U\}$ 的孪生关联图是否存在大小为 $4m - 2n - 2 + k = s_m + k + 2$ 的最优DAG 压缩。我们可以应用引理 5.6，并注意到由于 $R = U$ 是整个全集，任何 $X_{\supseteq}$ 也都是一个 $X_{=}$。特别地，当且仅当存在大小为 $|X| \leq k$、覆盖 U 的集合覆盖 $X \subseteq T$ 时，才存在这样的最优 DAG 压缩。◀

更新最优压缩的困难性。 回顾引言可知，研究如何更新已有的最优DAG 压缩，以反映原图中添加或删除一条边，也具有独立的意义；这一问题同样是NP-完全的：

▶**定理 5.7 (▼)。** MIN-DAG-COMPRESSION-ADD 是NP-完全的。

▶**定理 5.8 (▼)。** MIN-DAG-COMPRESSION-DEL 是NP-完全的。

6 结论

本文研究了DAG 压缩中基本算法的复杂性。我们证明，计算最小规模的DAG 压缩是NP-完全的，而更新DAG 压缩以处理底层DAG 中新增或删除的一条边，甚至也属于NP-困难问题。事实上，仔细考察定理 5.7 和 5.8 的证明可知，即使我们进一步要求更新过程中簇DAG 保持不变，该问题仍然是NP-完全的。另一方面，我们展示了如何以近乎线性的时间（相对于 DAG 压缩的规模，而非原图的规模）构造最小生成树。我们还发现，与树压缩相比，DAG 压缩所得的压缩结果严格更小。

寻找最小 DAG 压缩在一般情况下难以处理，这引出了更多有趣的问题：鉴于 Bannach 等人 [1] 的结果——当给定相应的收缩序列时，可以轻松地为孪生宽度有界的图构造最小 DAG 压缩——我们可以进一步探究，哪些结构性质（例如图中关于团、二部团以及具有公共邻域的簇的信息）可用于构造最小 DAG压缩。

我们在论文中证明了，DAG 压缩严格优于树压缩。为了更好地理解这两种压缩类型的能力差异，进一步研究哪些图族的最小 DAG 压缩严格小于最小树压缩，将会很有意义。

14 DAG 压缩图基本问题的复杂性

另一个有趣的方向是进一步扩展使用DAG 压缩的算法工具箱，从而为图提供更多更快的基本算法。一个具体的例子是在 DAG 压缩上计算最大流。

参考文献

1 Max Bannach、Florian Andreas Marwitz 和 Till Tantau。通过 DAG 压缩实现更快的图算法。载于 Olaf Beyersdorff、Mamadou Moustapha Kanté、Orna Kupferman 和 Daniel Lokshtanov 编，第41届计算机科学理论方面国际研讨会（STACS 2024），2024年3月12日至14日，法国克莱蒙费朗，LIPIcs 第289卷，第8:1–8:18页。达格施图尔城堡—莱布尼茨信息学中心，2024年。URL：https://doi.org/10.4230/LIPIcs.STACS.2024.8, doi:10.4230/LIPICS.STACS.2024.8。2 Daniel K. Blandford、Guy E. Blelloch 和 Ian A. Kash。可分图的紧凑表示。载于《第十四届 ACM-SIAM 离散算法年度研讨会论文集》，2003年1月12日至14日，美国马里兰州巴尔的摩，第679–688页。ACM/SIAM，2003年。URL： http://dl.acm.org/citation.cfm?id=644108.644219。3 Paolo Boldi 和 Sebastiano Vigna。WebGraph 框架 I：压缩技术。载于 Stuart I. Feldman、Mike Uretsky、Marc Najork 和 Craig E. Wills 编，《第13届万维网国际会议论文集》，WWW 2004，2004年5月17日至20日，美国纽约州纽约市，第595–602页。ACM，2004年。doi:10.1145/988672.988752。4 Édouard Bonnet、Eun Jung Kim、Stéphan Thomassé 和 Rémi Watrigant。孪生宽度 I：可处理的一阶逻辑模型检验。《ACM学报》，69(1):3:1–3:46，2022年。 doi:10.1145/3486655。5 Édouard Bonnet、Jaroslav Nešetřil、Patrice O. de Mendez、Sebastian Siebertz 和 Stephan Thomassé。孪生宽度与排列。载于《2023年欧洲组合学、图论与应用会议论文集》（EUROCOMB 2023）。马萨里克大学，捷克共和国布尔诺，2023年。doi:10.5817/CZ.MUNI.EUROCOMB23-022。6 Giorgos Bouritsas、Andreas Loukas、Nikolaos Karalias 和 Michael M. Bronstein。划分与编码：学习如何压缩图。载于 Marc'Aurelio Ranzato、Alina Beygelzimer、Yann N. Dauphin、Percy Liang 和 Jennifer Wortman Vaughan 编，《神经信息处理系统进展第34卷：神经信息处理系统年度会议 2021》（NeurIPS 2021），2021年12月6日至14日，线上举行，第18603–18619页，2021年。URL：https://proceedings.neurips.cc/paper/2021/hash/9a4d6e8685bd057e4f68930bd7c8ecc0-Abstract.html。7 Flavio Chierichetti、Ravi Kumar、Silvio Lattanzi、Michael Mitzenmacher、Alessandro Panconesi 和 Prabhakar Raghavan。关于社交网络压缩。载于 John F. Elder IV、Françoise Fogelman-Soulié、Peter A. Flach 和 Mohammed Javeed Zaki 编，《第15届ACM SIGKDD知识发现与数据挖掘国际会议论文集》，法国巴黎，2009年6月28日至7月1日，第219–228页。ACM，2009年。 doi:10.1145/1557019.1557049。8 Thomas H. Cormen、Charles E. Leiserson、Ronald L. Rivest 和 Clifford Stein。《算法导论》第4版。麻省理工学院出版社，2009年。URL： http://mitpress.mit.edu/books/introduction-algorithms。9 Laxman Dhulipala、Igor Kabiljo、Brian Karrer、Giuseppe Ottaviano、Sergey Pupyrev 和 Alon Shalita。通过递归图二分压缩图和索引。载于 Balaji Krishnapuram、Mohak Shah、Alexander J. Smola、Charu C. Aggarwal、Dou Shen 和 Rajeev Rastogi 编，《第22届ACM SIGKDD知识发现与数据挖掘国际会议论文集》，美国加利福尼亚州旧金山，2016年8月13日至17日，第1535–1544页。ACM，2016年。doi:10.1145/2939672.2939862。10 Tim Dwyer、Christopher Mears、Kerri Morgan、Todd Niven、Kim Marriott 和 Mark Wallace。改进稠密图的最优和近似幂图压缩，以实现更清晰的可视化。载于 Issei Fujishiro、Ulrik Brandes、Hans Hagen 和 Shigeo Takahashi 编，IEEE太平洋可视化研讨会（PacificVis 2014），日本横滨，2014年3月4日至7日，第

105–112页。IEEE计算机学会，2014年。URL： https://doi.org/10.1109/PacificVis.2014.46, doi:10.1109/PACIFICVIS.2014.46。11 Tim Dwyer、Nathalie Henry Riche、Kim Marriott 和 Christopher Mears。用于稠密有向图可视化的边压缩技术。IEEE可视化与计算机图形学汇刊，19(12):2596–2605，2013年。 doi:10.1109/TVCG.2013.151。12 Joseph Kruskal。关于最短生成子树与旅行商问题。《美国数学学会会刊》，7:48–50，1956年。13 Saket Navlakha、Rajeev Rastogi 和 Nisheeth Shrivastava。具有有界误差的图摘要。载于 Jason Tsong-Li Wang 编，《ACM SIGMOD数据管理国际会议论文集》，SIGMOD 2008，加拿大不列颠哥伦比亚省温哥华，2008年6月10日至12日，第419–432页。ACM，2008年。 doi:10.1145/1376616.1376661。14 Julian Shun、Laxman Dhulipala 和 Guy E. Blelloch。更小、更快：使用 ligra 对压缩图进行并行处理+。载于 Ali Bilgin、Michael W. Marcellin、Joan Serra-Sagristà 和 James A. Storer 编，2015年数据压缩会议（DCC 2015），美国犹他州雪鸟镇，2015年4月7日至9日，第403–412页。IEEE，2015年。 doi:10.1109/DCC.2015.8。15 Hannu Toivonen、Fang Zhou、Aleksi Hartikainen 和 Atte Hinkka。加权图压缩。载于 Chid Apté、Joydeep Ghosh 和 Padhraic Smyth 编，《第17届ACM SIGKDD知识发现与数据挖掘国际会议论文集》，美国加利福尼亚州圣迭戈，2011年8月21日至24日，第965–973页。ACM，2011年。 doi:10.1145/2020408.2020566。16 Luca Versari。《大型图压缩技术：理论与实践》。博士论文，比萨大学，意大利，2021年。可见： https://www.lucaversari.it/phd/main.pdf。

技术性证明

下文给出正文中省略的证明。在每种情况下，我们都会再次陈述定理或引理的结论，以方便读者。

▶ **引理 2.1 的结论。**对于输入的一个 DAG 压缩 $D=(V,A,E)$ 和 $\boldsymbol{G}=(\boldsymbol{V},\boldsymbol{E})$，我们可以在 $O(|A|+|V|)$ 时间内计算一个函数 $\boldsymbol{c}\colon V\to\boldsymbol{V}$，使得对所有 $v\in V$，都有 $\boldsymbol{c}(v)\in\boldsymbol{C}(v)$。

证明。计算 (V,A) 的一个拓扑排序 $<$，使得 $(u,v)\in A$ 蕴含 $u<v$；因此，按照 u，最小的顶点是源点，最大的顶点是汇点。按照 $<$ 的降序遍历 V，并设置 $\boldsymbol{c}(v):=v$（如果 v 是汇点）；否则设置$\boldsymbol{c}(v):=\boldsymbol{c}(u)$，其中 u 是满足 $(v,u)\in A$ 的任意顶点。显然，我们始终有 $\boldsymbol{c}(v)\in\boldsymbol{C}(v)$，因为 $\boldsymbol{c}(v)$ 总是被设置为 $\boldsymbol{V}$ 中的一个顶点，且该顶点在 (V,A) 中可从 v 到达。运行时间由拓扑排序所需时间的标准上界得出。◀

A.1 第3节的证明

▶ **引理3.4的断言。**假设不变式 *3.2* 对 $i-1$ 成立，并且我们对已清理顶点 u_i 和 v_i执行 添加边$(\boldsymbol{c}(u_i)\ \boldsymbol{c}(v_i))$ 。那么之后不变式仍对 i 成立。

证明。调用会检查 $P.\mathsf{find}(\boldsymbol{c}(u_i))=P.\mathsf{find}(\boldsymbol{c}(v_i))$ 是否成立。首先假设成立。那么 $\boldsymbol{c}(u_i)$ 和 $\boldsymbol{c}(v_i)$ 属于划分 $\boldsymbol{U}$ 中的同一集合；并且由于 u_i 和 v_i已清理，$\boldsymbol{C}(u_i)$ 中以及 $\boldsymbol{C}(v_i)$ 中的所有其他顶点也属于该集合。特别地，$\boldsymbol{C}(u_i)\otimes\boldsymbol{C}(v_i)$ 中没有任何边连接尚未属于划分 $\boldsymbol{U}$ 中同一集合的顶点，因此可以全部跳过。其次，假设不成立。那么我们可以安全地将 $\{\boldsymbol{c}(u_i)\ \boldsymbol{c}(v_i)\}\in\boldsymbol{C}(u_i)\otimes\boldsymbol{C}(v_i)$ 添加到 $\boldsymbol{M}$ 并调用 unite，这会将 $P(\boldsymbol{c}(u_i))$ 和 $P(\boldsymbol{c}(v_i))$ unite。再次，由于已清理，在新的划分中，$\boldsymbol{C}(u_i)$ 中的所有内容以及 $\boldsymbol{C}(v_i)$ 中的所有内容最终都会与 $\boldsymbol{c}(u_i)$ 和 $\boldsymbol{c}(v_i)$ 属于同一集合。◀

▶ **引理3.5 的断言。**假设对于 $i-1$，不变式 *3.2* 成立，并且我们执行了 $\textsc{make-clean}(v_i,\boldsymbol{c}(u_i))$。那么之后 v_i 将是干净的，并且对于 $i-1$，该不变式仍然成立。

证明。令 $\boldsymbol{E}'$ 为该不变式中已覆盖的边的集合。证明采用结构归纳法。只有在 v 尚未标记为干净时，我们才需要证明相应结论。考虑 v 的每个子节点 $w\in V$，即 $(v,w)\in A$。如果 w 尚未干净，我们就在第 5 行调用 make-clean$(w,\boldsymbol{r})$；根据归纳假设，这将确保 w 是干净的（并且请注意，递归调用仍满足前置条件）。考虑调用 add-edge$(\boldsymbol{c}(w)\ \boldsymbol{r})$：由于 w 和 $\boldsymbol{r}$ 现在都是干净的（作为汇点的 $\boldsymbol{r}$ 自动是干净的），我们已在引理3.4 中论证过，必要时我们会向 $\boldsymbol{M}$ 正确添加一条边，使 $\boldsymbol{M}$ 成为 $\boldsymbol{E}'\cup(\boldsymbol{C}(w)\otimes\{\boldsymbol{r}\})$ 的生成森林。

关键观察是，在循环结束时，$w\in V$的所有子节点 v 都已清理干净，并且 $\boldsymbol{M}$将是 $\boldsymbol{E}''$ 的生成森林：$=\boldsymbol{E}'\cup\bigcup_{(v,w)\in A}(\boldsymbol{C}(w)\otimes\{\boldsymbol{r}\})$。然而，由于 $\boldsymbol{r}$ 与 $\boldsymbol{E}''$中所有 $\boldsymbol{C}(w)$里的所有顶点都相连，因此所有 $\boldsymbol{C}(w)$都位于 $\boldsymbol{E}''$的同一个连通分量中。根据定义，在生成森林 $\boldsymbol{M}$中，它们也必须位于同一个连通分量中。由于 $\boldsymbol{M}$ 的连通分量恰好就是 P中的集合，我们得出结论：存在唯一的 $\boldsymbol{U}\in P$，使得$\bigcup_{(v,w)\in A}\boldsymbol{C}(w)\subseteq\boldsymbol{U}$。由于 $\boldsymbol{C}(v)=\bigcup_{(v,w)\in A}\boldsymbol{C}(w)$，我们得出结论：$\boldsymbol{C}(v)$完全包含在某个 $\boldsymbol{U}\in P$ 中，因此，在最后一行将 v 标记为已清理是正确的。

此外，在每次迭代中，生成森林都是最小生成森林，因为边按权重升序排列。◀

A.2 第4节的证明

▶ **引理 4.1 的断言**。对于 $R_{g\times g}=(V,E)$，设 X、$X'\subseteq V$为满足 $|X|\geq 3$和$|X'|\geq 3$且 $X\times X'\subseteq E$的集合。那么， $X\cup X'$ 中的所有顶点都位于同一行或同一列。

证明。 假设结论不成立。现在进一步假设，至少， X中的顶点位于同一行 r。那么必定存在一个$(r',c')\in X'$ 满足 $r\neq r'$。但是，由于 X 中的所有顶点都必须位于不同的列（它们位于同一行），(r',c') 将至少与两个同时位于不同的行（即 r）和不同的列中的顶点相连。用类似的论证可知， X 中的顶点也不可能全部位于同一列。

当三个顶点既不全位于同一行，也不全位于同一列时，其中两个必定既位于不同的行，也位于不同的列，也就是说，必定存在 (r_1,c_1),$(r_2,c_2)\in X$满足 $r_1\neq r_2$ 和 $c_1\neq c_2$。现在考虑任意顶点 $(r',c')\in X'$。由于它与 X中的所有顶点相连，因此它必须与 (r_1,c_1) 共同行或同列，并且也必须与 (r_2,c_2) 共同行或同列。只有在 $r'=r_1$ 和 $c'=c_2$ 或 $r'=r_2$ 和 $c'=c_1$ 的情况下才可能如此（其他情况可由 $r_1\neq r_2$ 和 $c_1\neq c_2$排除）。但这意味着 (r',c') 只有两种 可能性，这与 $|X'|>2$ 矛盾。◀

A.3 第5节证明

▶ **引理 5.2 的断言** (孪生点仍为孪生点)。 每个图 G 都有一个最优的 DAG压缩 (V,A,E)，使得 G 中的所有孪生点在 (V,A) 和 (V,E)中也都是孪生点。

证明。 假设 G 中存在孪生点 $\{t_1,t_2\}$，它们在 $(,A)$ 中或在 (V,E) 中不是孪生点。不失一般性，假设 t_1 的总度数较小（即在 (V,A) 和 (V,E) 中，其入邻域和出邻域大小之和在这两个顶点中较小）。然后移除所有与 t_2 相 incident 的弧和压缩边，并将与 t_1 相 incident 的相同弧和压缩边也添加到 t_2上。由于我们选择了总度数最小的孪生点，因此删除的弧和压缩边至少与添加的一样多。因此，所得的DAG 压缩仍然是最优的，并且很容易看出，它仍然表示相同的图 G。我们可以重复这一处理，直到 G 中的所有孪生点在 (V,A) 中以及在 (V,E) 中也都是孪生点。◀

▶ **引理 5.3 的断言**（簇仅位于一侧）。每个有向二分图 $G=(V_1\cup V_2,E)$ 都存在一个最优的DAG压缩（V,A,E），使得对于所有 $v\in V\setminus V_1$，都有$C(v)\subseteq V_2$。

证明。令 $U_1=\{u\in V\mid C(u)\cap V_1\neq\emptyset\}$ 表示包含来自 V_1 的顶点的簇顶点，并对 V_2以同样方式定义 U_2。注意，在任何最优DAG 压缩中， U_1 和 U_2 必定不相交，因为压缩边无法使用同时属于二者的簇顶点 v，这意味着我们可以安全地移除 v，从而减小压缩的规模。

现在，命题仅指出存在一种最优 DAG 压缩，其中包含 $U_1=V_1$。假设目前尚非如此。考虑任意一个在限制于 U_1 的簇 DAG 中作为源点的 $u_1\in U_1\setminus V_1$（例如图 4 中的 $u_1=a$ 或 $u_1=d$）。那么， E 中所有涉及 u_1 的压缩边都具有 $(u_1,v)\in E$ 的形式，其中 $v\in U_2$；而 A 中所有涉及u_1 的簇弧都具有 (u_1,v) 的形式，其中 $v\in U_1$。现在我们交换它们，也就是说，通过移除所有涉及 u_1的压缩边和簇弧，从 E构造出 E'，并从 A 构造出 A'，然后改为添加以下内容：对于每个原来的 $(u_1,v)\in E$，我们向 A'添加 (u_1,v)；对于每个原来的 $(u_1,v)\in A$，我们向 E' 添加 (v,u_1)（注意方向）。

我们声称，该变换会得到原图的一种新的最优 DAG 压缩。显然，大小没有变化。要看出同样的边 $\boldsymbol{E}$ 仍得到表示，考虑原来的压缩边 $(u_1,\ v) \in E$。它们共同表示了这些边（回顾一下，$\boldsymbol{C}(u_1) = \bigcup_{(u_1,\ u)\in A} \boldsymbol{C}(u)$）：

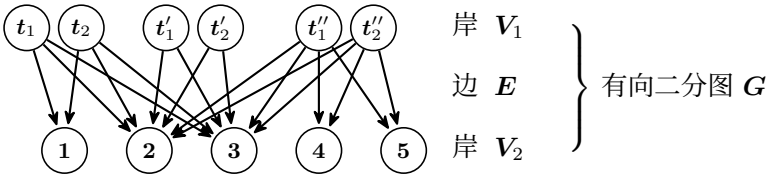
$$\bigcup_{(u_1,v)\in E} (\boldsymbol{C}(u_1) \times \boldsymbol{C}(v)) = \left(\bigcup_{(u_1,u)\in A} \boldsymbol{C}(u) \right) \times \left(\bigcup_{(u_1,v)\in E} \boldsymbol{C}(v) \right).$$

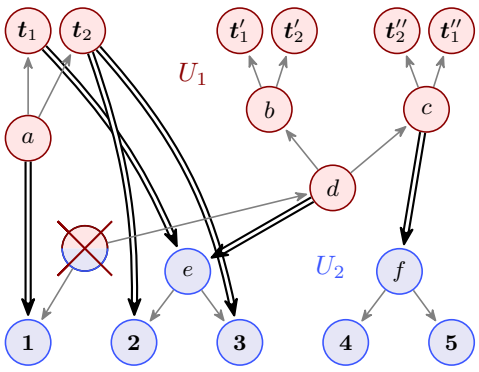
在新的压缩中，新的压缩边共同表示（其中 $\boldsymbol{C}'$· 是新图中的簇函数）

$$\bigcup_{(v,u_1)\in E'} (\boldsymbol{C}'(v) \times \boldsymbol{C}'(u_1)) = \left(\bigcup_{(v,u_1)\in E'} \boldsymbol{C}'(v) \right) \times \left(\bigcup_{(u_1,u)\in A'} \boldsymbol{C}'(u) \right).$$

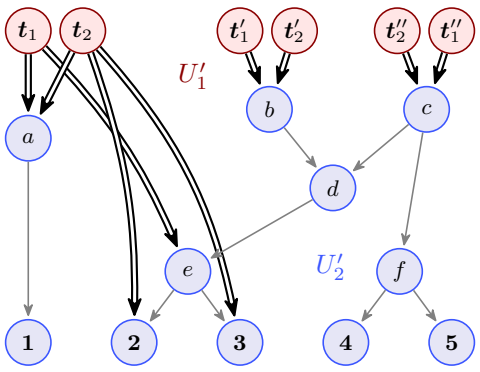
比较最后两个值，并注意到对于所有 $x \neq u_1$，$\boldsymbol{C}(x) = \boldsymbol{C}'(x)$，我们可以看出它们相同。

通过根据需要反复进行处理，我们得到一个具有 $U_1 = \boldsymbol{V}_1$ 的压缩。





DAG 压缩与 $\boldsymbol{V}$ 中的非平凡簇



DAG压缩源于引理 5.3 的变换

图 4 引理 5.3 中变换的示例，用于确保有向图 $\boldsymbol{G} = (\boldsymbol{V}_1 \cup \boldsymbol{V}_2, \boldsymbol{E})$ 的最优 DAG 压缩仅具有“岸 $\boldsymbol{V}_2$ 中的簇”，这意味着对于所有 $v \in V \setminus \boldsymbol{V}_1$，$\boldsymbol{C}(v) \subseteq \boldsymbol{V}_2$ 均成立。回顾一下，U_1 和 U_2 分别定义为其簇与 $\boldsymbol{V}_1$ 和 $\boldsymbol{V}_2$ 相交的簇顶点集合。注意，$U_1 \cap U_2$ 中的顶点没有关联的压缩边，可以将其移除。引理 5.3 中的“切换”示例发生在 a：压缩边 $(a, 1) \in E$ 被弧 $(a, 1) \in A'$替代，而两条弧 $(a, \boldsymbol{t}_1) \in A$ 和 $(a, \boldsymbol{t}_2) \in A$ 则被反方向上的压缩边 $(\boldsymbol{t}_1, a) \in E'$ 和 $(\boldsymbol{t}_2, a) \in E'$ 替代。其他切换首先发生在 d，随后发生在 b和 c，从而得到右侧所示的压缩。

▶**引理 5.4 的断言** (孪生点只有一条压缩边)。 每个有向二分图 $\boldsymbol{G} = (\boldsymbol{V}_1 \cup \boldsymbol{V}_2, \boldsymbol{E})$ 都存在一个最优的 DAG 压缩(V, A, E)，使得所有孪生点$\boldsymbol{t}_1, \boldsymbol{t}_2 \in \boldsymbol{V}_1$ 至多关联一条压缩边。

证明。设给定图 $\boldsymbol{G}$ 的一个最优 DAG 压缩 (V, A, E)。根据引理 5.2，我们可以假设 $\boldsymbol{G}$ 的所有孪生点在 (V, A) 和 (V, E) 中也都是孪生点。根据引理 5.3，我们可以假设所有 $v \in V \setminus \boldsymbol{V}_1$ 都具有 $\boldsymbol{C}(v) \subseteq \boldsymbol{V}_2$；并且注意，第二个引理中的变换不会破坏“孪生点仍为孪生点”这一性质。现在假设存在孪生点 $\boldsymbol{t}_1, \boldsymbol{t}_2 \in \boldsymbol{V}_1$，使得 $(\boldsymbol{t}_1, u) \in \boldsymbol{E}$，并且对于某个 $u \neq v$，还有 $(\boldsymbol{t}_1, v) \in \boldsymbol{E}$。由于 $\boldsymbol{t}_1$ 和 $\boldsymbol{t}_2$ 在 (V, A) 中也互为孪生点，因此还有 $(\boldsymbol{t}_2, u)$、$(\boldsymbol{t}_2, v) \in \boldsymbol{E}$。删除这四条压缩边，在 V 中添加一个新的簇顶点 c，在 A 中添加两条簇弧 (c, u) 和 (c, v)，并在 E 中添加两条压缩边 $(\boldsymbol{t}_1, c)$ 和 $(\boldsymbol{t}_2, c)$。显然，我们现在仍然得到图 $\boldsymbol{G}$ 的一个最优 DAG 压缩，但 $\boldsymbol{t}_1$ 和 $\boldsymbol{t}_2$ 在 E 中的度都减少了 1。只要可能，就重复此变换，最终得到一个满足该断言的图。◀

▶**引理 5.6 的断言**。设 T 是 U 的一个子集族。设 $T\prime = T \cup \{R\}$，其中集合 $R \subseteq U$ 不是任何 $S \in T$ 的子集。设 $k_=$ 是满足 $\bigcup X_= = R$ 的集合 $X_= \subseteq T$ 的最小大小（若不存在这样的集合，则令 $k_= = \infty$），并设 $k_\supseteq$ 是满足 $\bigcup X_\supseteq \supseteq R$ 的集合 $X_\supseteq \subseteq T$ 的最小大小。设 s 和 s' 分别为 T 和 T' 的孪生关联图的最优 DAG 压缩的大小。则 $s + k_\supseteq + 2 \leq s' \leq s + k_= + 2$。

证明。考虑 T 的孪生关联图 $\boldsymbol{T} = (\boldsymbol{V}_1 \cup \boldsymbol{V}_2, \boldsymbol{E})$ 的一个最优 DAG 压缩 (V, A, E)。根据引理 5.4，我们可以假设 $\boldsymbol{T}$ 的孪生点，特别是 $\boldsymbol{V}_1$ 中的所有顶点，在 E 中都只有一条关联的压缩边。这意味着每条压缩边 $(\boldsymbol{a}_S, c_S) \in E$ 必须给出 $\boldsymbol{a}_S \times S \subseteq \boldsymbol{E}$ 中的所有元素，因此 $\boldsymbol{C}(c_S) = S$。换言之，对于每个 $S \in T$，都存在一个簇顶点 $c_S \in V$，使得 $\boldsymbol{C}(c_S) = S$。

为证明 $s' \leq s + k_= + 2$ 成立，按如下方式从 (V, A, E) 构造 $(V' A', E')$：令 $V' = V \cup \{\boldsymbol{a}_R, \boldsymbol{b}_R, c_R\}$，令 $E' = E \cup (\{\boldsymbol{a}_R, \boldsymbol{b}_R\} \times \{c_R\})$，并令 $A' = A \cup \{(c_R, c_S) \mid S \in X_=\}$。显然，$|A'| + |E'| = |A| + k_= + |E| + 2$，且 $(V' A', E')$ 是 T' 的孪生关联图的 DAG 压缩。

为证明 $s' \geq s + k_\supseteq + 2$，从 T' 的孪生关联图的一个最优 DAG 压缩 $(V' A', E')$ 出发，并像之前一样，令其满足如下性质：对于每个 $S \in T'$，都存在一个簇顶点 $c_S \in V'$，使得 $\boldsymbol{C}'(c_S) = S$。特别地，必定存在一个顶点 c_R，使得 $\boldsymbol{C}'(c_R) = R$，且有压缩边 $(\boldsymbol{a}_R, c_R)$ 和 $(\boldsymbol{b}_R, c_R)$。考虑弧 $(c_R, v) \in A'$：我们必须有 $\boldsymbol{C}'(v) \subseteq S \in T$，否则，我们可以通过将 c_R 直接连接到 v 的子节点并移除 v，得到更优的压缩。特别地，满足 $(c_r, v) \in A'$ 的 v 数量至少为 $|X_\supseteq|$，因为 c_R 的子节点必须覆盖 R，所以每个 $\boldsymbol{C}'(c_R)$ 都包含在某个 $S \in T$ 中，这意味着此类弧至少有 $|X_\supseteq|$ 条（也可能更多）。这意味着，通过移除 $\boldsymbol{a}_R$ 和 $\boldsymbol{b}_R$ 及其各自关联的两条压缩边，并移除所有弧 $(c_R, v) \in A'$，我们可以构造一个大小至多为 $s' - k_\supseteq - 2$ 的 T 的孪生关联图的 DAG 压缩 (V, A, E)。◀

▶**定理 5.7 的证明**。 MIN-DAG-COMPRESSION-ADD 是NP完全问题。

证明。 我们采用与定理 1.4 的证明相同的思路，在那里我们将集合覆盖归约为最小 DAG 压缩。与该证明一样，我们从 U 的子集集合 T 开始，但现在令 $U = \{2, \ldots, n+1\}$。将 T 替换为 $T\prime = \{\{1\} \cup S \mid S \in T\}$，也就是说，将集合族中的所有集合都加上 1，并将这些集合称为受感染。显然，这不会改变最优集合覆盖的大小，但现在包含$\{1\} \cup U$。对于这个新集合 $T\prime$，我们继续

按照定理 1.4 的证明添加集合 T'，使得对于大小递增的集合 S_i，$\hat{T}' = \{S_1, \ldots, S_m\}$，任意集合的所有初始段都已包含在 $\hat{T}'$ 中。关键的是，注意到 $\hat{T}'$ 中的所有集合都受感染，也就是说，都包含 1。最后，回顾定理 1.4 的证明，$\hat{T}'$ 的双生关联图 $\hat{\boldsymbol{T}}' = (\boldsymbol{V}_1 \cup \boldsymbol{V}_2, \boldsymbol{E})$ 存在一个大小为 $4m - 2(n+1) - 4$ 的最优 DAG 压缩 (V, A, E)，其中 $n+1 = |U \cup \{1\}|$，并注意到我们可以轻松计算出这个最优DAG压缩。

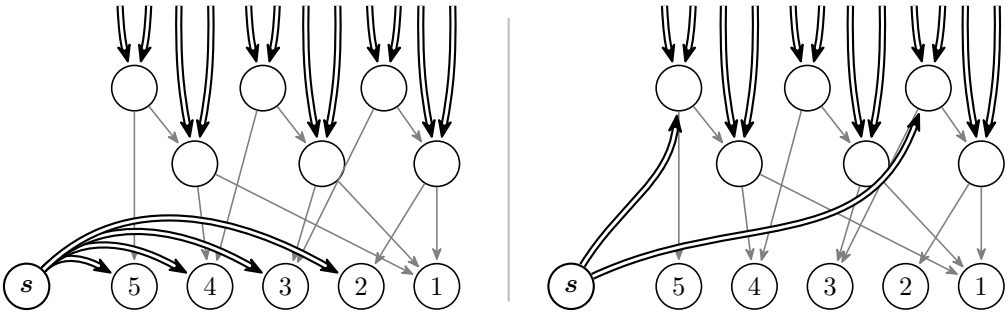


图 5 定理 5.7 中的构造：左侧是 $\hat{T}'$ 的孪生关联图的一种最优 DAG 压缩，其中 T 中的所有集合都包含特殊元素 1，并且从特殊顶点 s 向 $\{2, \ldots, n+1\}$ 中的所有元素连边（但不与 1 连边）。由于 s 没有边连到 1，因此压缩时不能使用任何“受感染”的簇顶点 c（即 $s \in \boldsymbol{C}(c)$），这意味着图示的压缩是最优的。向 $\boldsymbol{E}$ 添加单条边 $(s, 1)$ 会彻底改变情况：现在，只需将 s 与最小集合覆盖中的簇顶点相连，就能压缩 $\{s\} \times \{1, \ldots, n+1\}$。

此处我们偏离前面的证明：通过向 $\boldsymbol{V}_1$ 添加一个新顶点 s，从 $\hat{\boldsymbol{T}}'$ 构造图 $\boldsymbol{G} = (\boldsymbol{V}'_1 \cup \boldsymbol{V}_2, \boldsymbol{E}')$，使得 $\boldsymbol{V}'_1 = \boldsymbol{V}_1 \cup \{s\}$；并将边 (s, u) 添加到 $\boldsymbol{E}$，其中所有 $u \in \{2, \ldots, n+1\} = U$ 均适用，因此 $\boldsymbol{E}' = \boldsymbol{E} \cup \{(s, u) \mid u \in U\}$（但请注意，$(s, 1) \notin \boldsymbol{E}'$）。显然，$D = (V \cup \{s\}\ A, E \cup \{(s, u) \mid u \in U\})$ 是 $\boldsymbol{G}$ 的一种大小为 $4m - 2(n+1) - 4 + n$ 的 DAG 压缩。

我们声称该压缩是最优的。为此，任取给定的 $\boldsymbol{G}$ 的一个最优 DAG 压缩 (V_0, A_0, E_0)。根据引理 5.2、5.3 和 5.4，对于每个满足 $|S| \geq 2$ 的 $S \in \hat{T}'$，都存在一个满足 $\boldsymbol{C}(c_S) = S$ 的簇顶点 $c_S \in V_0$。这些 c_S 中的任何一个都不可能存在压缩边 (s, c_S)，因为所有这些 S 都已受感染；这意味着我们只能有指向其他簇顶点 c 的 $(s, c) \in \boldsymbol{E}_0$。特别是，如果我们删除所有这些 $m - (n+1)$ 个 c_S，以及与它们相连的弧和压缩边，那么仍必须保留表示 $\{s\} \times U$ 的压缩边和弧。每个 c_S 都有两条入压缩边和至少两条出弧，这意味着我们会删除 $4m - 4n - 4$ 条边，剩下 $2n + n$ 条边。接下来，如果我们删除所有对应于单元素 $S = \{j\}$（其中 $j \in \{2, \ldots, n+1\}$）的 $\boldsymbol{a}_S$ 和 $\boldsymbol{b}_S$，以及它们的压缩边，就会剩下 n 条边。然而，要表示 $\{s\} \times U$，我们至少需要 $|U| = n$ 条压缩边和／或弧，这表明我们的压缩是最优的。

最后，为了完成证明，我们输出 $\boldsymbol{G}$、$\boldsymbol{G}$ 的最优压缩 D、新边 $(s, 1)$，以及数目 $k + 4m - 2(n+1) - 4$。关键观察是，新边 $(s, 1)$ 意味着我们现在必须压缩 $\{s\} \times (\{1\} \cup U)$。现在很容易看出（使用与之前相同的论证），最佳做法就是通过压缩边将 s 连接到 $U \cup \{1\}$ 的最小基数集合覆盖 $X \subseteq T$ 中的各个簇。◀

▶**定理 5.8 的断言**。MIN-DAG-COMPRESSION-DEL 是 NP完全问题。

证明。 我们再次从集合覆盖问题归约。设 $(T,\ k)$ 是集合覆盖的一个实例，其中 T 是 $U = \{2、\ldots\ldots、n+1\}$ 的一个子集族。首先，令 $T' = T \cup \{U \cup \{1\}\}$，并再次向 T' 中添加集合，使得 $T' = \{S_1、\ldots\ldots、S_m\}$ 由 S_i 中大小递增的集合构成，并且任意 S_i 的所有初始段都已属于 T'。回顾一下，如果 $1 \in S$，则称集合 $S \in T'$ 为受感染。

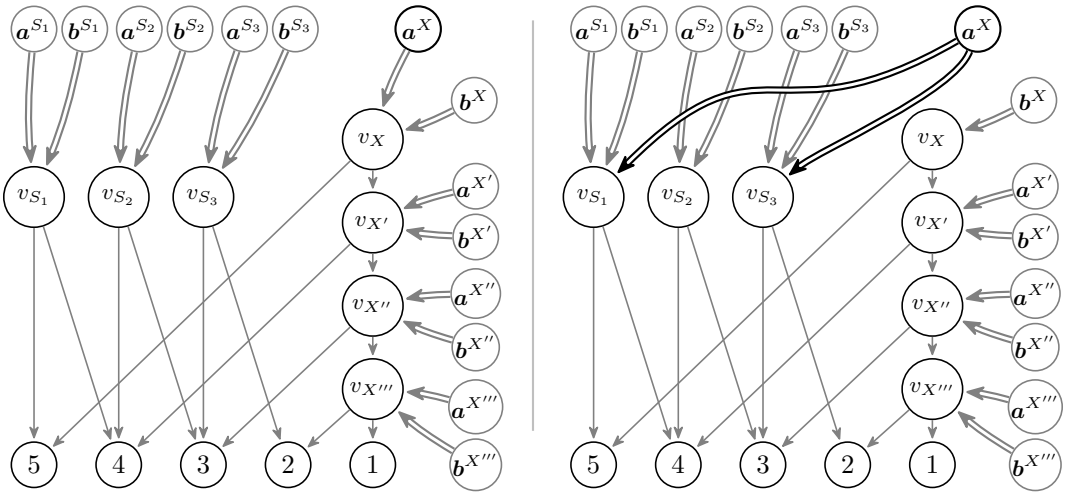


图6 定理 5.8 中的构造：左侧是 $\hat{T}'$ 的孪生关联图的一个最优 DAG 压缩。删除边 $(a^X,\ 1)$ 后，我们可能无法再在最优 DAG 压缩中使用受感染顶点，这意味着新的最优压缩是通过将 a^X 连接到最小集合覆盖得到的。

接下来，我们按照定理 1.4 的证明中所述，构造 $\widehat{T}'$ 的孪生关联图 $\hat{T}'$，并得到一个大小为 $4m - 2(n+1) - 4$ 的最小 DAG 压缩 $D = (V,\ A,\ E)$。

我们选择 $e = (a_{U\cup\{1\}},\ 1)$ 作为待删除的边。此外，我们令 $k' = k + 4m - 2(n+1) - 4$，而针对 min-dag-compression-del 的归约输出为 $(D,\ e,\ k')$。

我们证明该归约的正确性。首先，设 $(T,\ k)$ 是集合覆盖问题的一个实例。因此，存在一个大小至多为 k、覆盖 U 的集合覆盖 T^*。于是，DAG 压缩 $D' = (V,\ A,\ E')$（其中 E' 由 E 删除压缩边 $(a_{U\cup\{1\}},\ v_{U\cup\{1\}})$，并添加压缩边 $\{a_{U\cup\{1\}}\} \times \{v_S \mid C(v_S) \in T^*\}$ 得到）是 $(V,\ E \setminus \{e\})$ 的一个 DAG 压缩 T'，且 $|E'| \leq 2 \cdot |T| + k$。

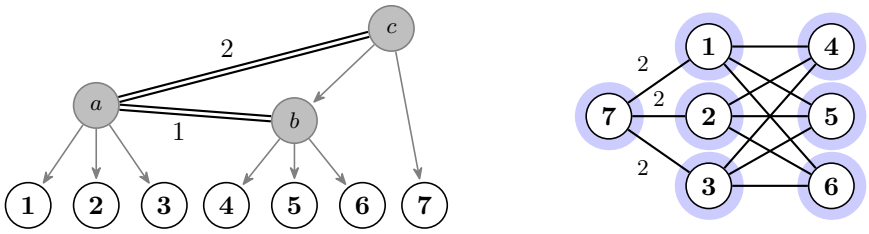
其次，设 (D, e, k') 是 min-dag-compression-del 问题的一个实例，$\hat{T}'$ 是其所表示的图。接着，设 $D' = (V, A, E')$ 是删除了 e 的 T' 的 DAG 压缩，且至多包含 k' 条压缩边。注意，在 E' 中，$a_{U\cup\{1\}}$ 至多有 k 条出压缩边；其余 $2 \cdot |T| - 1$ 个孪生点中的每一个都必须在 E' 中至少有一条出压缩边。此外，$a_{U\cup\{1\}}$ 在 E' 中不能有一条指向受感染顶点的压缩边，因为这会表示边 e。因此，对于每个满足 $(a_{U\cup\{1\}}, v) \in E'$ 的顶点 v，存在一个集合 $S \in T$，使得 $C(v) \subseteq S$，而这些至多 k 个集合 $S \in T$ 的全体构成了 U 的一个集合覆盖。◀

B Kruskal 算法的补充算法和图示

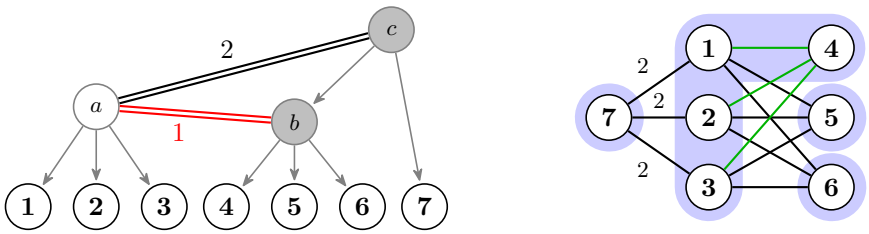
一种并查集数据结构 P用于跟踪 V 的一个划分$\{U_1, \ldots, U_k\}$；对于 $v \in V$，我们记 $P(v)$ 为包含 v 的集合 $U_i \subseteq V$。稍微滥用一下记号，我们也用 $U \in P$ 表示 U 是划分中的一个集合。该数据结构

22 DAG 压缩图基本问题的复杂性

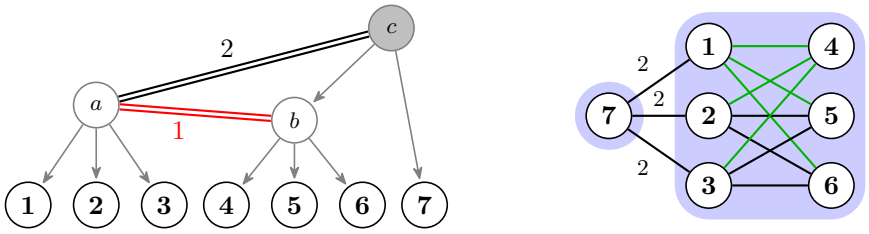
初始化为由单元素集合构成的划分，即对所有 $\boldsymbol{v} \in \boldsymbol{V}$， $P(\boldsymbol{v}) = \{\boldsymbol{v}\}$。它支持两种操作：首先， P.find($\boldsymbol{v}$) 返回某个内部值，并保证 P.find($\boldsymbol{u}$) $= P$.find($\boldsymbol{v}$) 当且仅当 $P(\boldsymbol{u}) = P(\boldsymbol{v})$。其次， P.unite($\boldsymbol{u}_1, \boldsymbol{u}_2$) 将划分更改为 P'，使得 $P'(\boldsymbol{u}_1) = P'(\boldsymbol{u}_2) = P(\boldsymbol{u}_1) \cup P(\boldsymbol{u}_2)$，且对于 $\boldsymbol{v} \notin P(\boldsymbol{u}_1) \cup P(\boldsymbol{u}_2)$，有 $P'(\boldsymbol{v}) = P(\boldsymbol{v})$；换言之， $P(\boldsymbol{u}_1)$ 和 $P(\boldsymbol{u}_2)$ 在 P' 中合并。众所周知， [8, 第 21 章4] 指出，并查集数据结构可以通过这样一种方式实现：一系列 k 次 find 和 unite 操作的耗时为 $k \cdot \alpha(|\boldsymbol{V}|)$，其中 α 是反阿克曼函数。



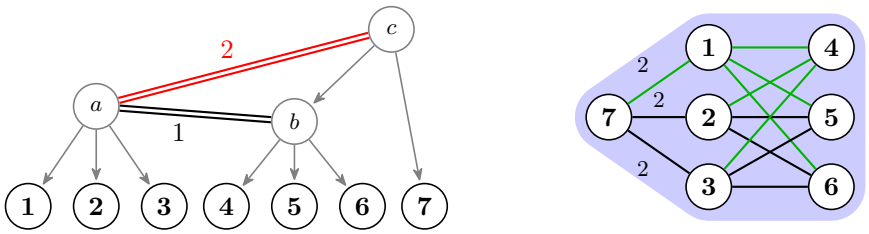
DAG 压缩 $D = (V,\ A,\ E)$ 的　　图 $G = (V,\ E)$，划分 P



处理 $(a,\ b) \in E$：首先，清理 a，将每个子节点连接到 $c(b) = 4$



处理 $(a,\ b) \in E$：其次，清理 b，将每个子节点连接到 $c(a) = 1$



处理 $(a,\ c) \in E$：清理 c，连接子节点 b 和 **7**，如有需要，将其连接到 $c(a) = 1$

图7 在图 G 的 DAG 压缩 D 上计算最小生成树（绿色）的示例。G 中没有权重标签的边，其权重为 1。逐步构建的划分 P 以蓝色显示。V 中已清理的顶点以白色显示；初始时只有汇点是白色的。要处理像 $\{a, b\}$ 这样的压缩边，我们首先通过将其子簇连接到代表顶点 c（b），来清理 a；在此示例中，该顶点是 $4 \in C$（b）。然后我们清理 b。接下来，算法会尝试添加边 $\{c$（a） c（b）$\} = \{1, 4\}$，但并查集数据结构 P 会告知算法这些顶点已属于同一集合，因此不会添加边。处理 $\{a, b\}$ 意味着清理 c；这不会在 c（a）与已清理子节点 b 所属的簇之间添加边，但会在 c（a）与 7 之间添加一条边。最后，边 $\{c$（a） c（c）$\} = \{1, 4\}$ 再次被跳过。