生成所有给定可图度序列的带标记无三角形图的高效算法*

Kai Wang计算机科学系 佐治亚南方大学美国佐治亚州斯泰茨伯勒
kwang@georgiasouthern.edu

arXiv:2601.15943v1 [math.CO] 22 Jan 2026

摘要—我们扩展了此前的算法。该算法可生成所有具有给定可图度序列的带标记图；扩展后的算法可生成所有具有给定可图度序列的带标记无三角形图。该算法采用多种剪枝技术，无须先生成输入序列的所有标记实现，再逐一检验每个标记实现是否为无三角形图。只需增加一项简单测试，判断生成的每个无三角形实现是否为二部图，即可进一步将其扩展为生成所有具有给定可图度序列的带标记二部图。与原算法一样，所有输出图均按字典序生成。这些算法也很容易并行化。

索引词—**可图度序列、标记实现、无三角形图、二部图**

I. 引言

有限简单图 $G = (V,\ E)$ 若不含三个构成团的顶点 K_3，则称为无三角形图。一般而言，若顶点集 V 不含构成团 K_r的r子集，则称其为 K_r-自由图。给定一个阶为 n的图 G，判定其是否为无三角形图至多需要 $O(n^3)$ 时间。然而，判定其是否包含一个团K_r是 NP 完全问题，其中 r 是输入参数，并满足$3 \le r \le n$ [10] G 。

一个相关的判定问题 $P(\boldsymbol{d},\ r)$ 如下：给定一个带有 $d_n \ge 1$ 的可图度序列 $\boldsymbol{d} = \{d_1 \ge d_2 \ge \cdots \ge d_n\}$，以及满足 $3 \le r \le n$ 的参数 r，是否存在 $\boldsymbol{d}$ 的一个无 K_r 简单图实现? 在特殊情形 $r = 3$ 下，该问题是判断 $\boldsymbol{d}$ 是否有无三角形实现。尽管有人猜想这一一般判定问题 $P(\boldsymbol{d},\ r)$ 是难解的 [2],，但我们尚未发现该问题是 NP 完全的严格证明。事实上，我们猜想该问题不属于 NP，因为形式语言$L = \{\langle G, r\rangle |$ 图 G 不含大小为 $r\}$ 的团（它是 NP 完全形式语言 $L = \{\langle G, r\rangle |$ 图 G 含有大小为 $r\}$ 的团的补语言）是 coNP 完全的（利用 NP 完全语言的补语言是 coNP 完全这一事实），而且我们猜想每个 coNP 完全语言都不属于 NP[5]。然而，特殊情形问题 $P(\boldsymbol{d},\ 3)$ 属于 NP，因为判断一个给定图是否无三角形至多需要 $O(n^3)$ 时间，因此 $\boldsymbol{d}$ 的一个无三角形实现（如果存在）可以作为任何输入 $\boldsymbol{d}$ 属于形式语言 $P(\boldsymbol{d},\ 3) = \{\langle \boldsymbol{d}\rangle |$的证书：可图序列 $\boldsymbol{d}$ 有一个无 K_3 实现。$\}$。不过，要确定性地对这个特殊情形的判定问题给出“是”的答案，可能并不需要这样的证书。

本研究获得了佐治亚南方大学工程与 Computing 学院科研种子基金的资助。计算实验得到了佐治亚南方大学 Talon 集群的支持。

还可以考虑以下与判定问题 $P(\boldsymbol{d}, 3)$ 相关的组合生成问题:

(A) 生成 $\boldsymbol{d}$ 的某个带标号的无三角形实现。(B) 生成 $\boldsymbol{d}$ 的所有带标号的无三角形实现 $G = (V, E)$。将 $\boldsymbol{d}$ 的所有带标号的无三角形实现的集合记为 $\mathcal{S}(\boldsymbol{d})$。(C) 均匀随机生成 $\boldsymbol{d}$ 的一个带标号的无三角形实现。

这些问题似乎比 [21] 中相应的组合生成问题更复杂；在那些问题中，我们要寻找 $\boldsymbol{d}$ 的实现，而不施加任何额外的结构限制。本文将主要讨论问题 (B)。

无三角形图的一个特殊子类是二部图。存在色数任意高的无三角形图 [15], 并且判断给定的阶为 n 、色数为 $\le r$ 的无三角形图 G 是否具有这一性质是 NP 完全问题，其中 r 是与满足 $3 \le r \le n$ [11] 的图 G 一同给出的输入参数。众所周知，判断一个图（无论是否无三角形）是否为二部图很容易，因此我们也可以考虑上述三个组合生成问题的变体，即生成 $\boldsymbol{d}$ 的带标号二部图实现。

本文其余部分安排如下。第 II 节回顾我们的基本生成算法，该算法用于系统地生成 [21] 中给定可图序列的所有标记实现。第 III 节和第 IV 节分别介绍我们用于生成给定可图序列的所有无三角形实现和二部实现的简易算法。第 V 节对我们的算法进行粗略的复杂度分析。在

第 VI 节展示一些实验结果。第 VII 节总结全文，并提出一些未来的研究方向。

II. 基本生成算法回顾

我们用于生成给定输入 $\boldsymbol{d}$ 的所有带标号无三角形实现的穷举生成算法，其整体框架与 [21] 中所述的算法类似，并且采用回溯算法：算法遍历并构建的搜索树 $\mathcal{T}(\boldsymbol{d})$ 中的每个节点，都表示输入序列 $\boldsymbol{d}$的一个部分或完整的带标号无三角形实现。每个完整实现 G 具有相同的顶点集 $V = \{v_1, v_2, \cdots, v_n\}$，满足：（i） $d_G(v_i) = d_i$对于 $i = 1, \cdots, n$；以及（ii） G不含三角形。所有这些带标号无三角形实现的全序，是根据顶点标号的字典序 $v_1 < v_2 < \cdots < v_n$推导出的带标号实现的字典序（该全序的定义见 [21]）。我们的算法将按照这一顺序生成所有输出的无三角形图。我们仍将使用“节点”一词指搜索树 $\mathcal{T}(\boldsymbol{d})$中的一个点，并使用“顶点”一词指算法将要生成的 $\boldsymbol{d}$ 的带标号无三角形实现 G 中的一个点。在算法执行过程中的任意时刻， G 中顶点 v 的剩余度都是一个动态量，表示为满足 $\boldsymbol{d}$中规定的度要求，v 尚待分配的邻居数。根据剩余度是否为 0，G 中的顶点被称为饱和或未饱和。为了便于比较和描述，我们用 $\mathcal{T}_0(\boldsymbol{d})$表示在 [21] 中为所有带标号实现遍历并构建的通用穷举生成算法的搜索树；而用 $\mathcal{T}(\boldsymbol{d})$表示本文中用于生成无三角形实现的算法所遍历并构建的搜索树。

III. 无三角形实现的生成算法

我们假设输入 $\boldsymbol{d}$ 是一个可图度序列。（有几种等价且高效的判定准则可以判断是否满足这一条件 [17], , 其中 Erd˝os-Gallai 判据 [3] 可能是最广为人知的。Havel-Hakimi 判据 [6], [7] 也以能够完成这一任务而闻名。）然而，我们并不假设它存在无三角形实现。因此，在我们的算法开始构建搜索树 $\mathcal{T}$（$\boldsymbol{d}$）以生成所有无三角形实现之前，它会先进行三项简单检查（我们称这些检查为第 1 阶段操作），以排除 $\boldsymbol{d}$ 不存在无三角形实现的简单情形。

根据 Mantel 定理 [12], , 阶为 n的无三角形简单图最多有 $\lfloor \frac{n^2}{4} \rfloor$条边。因此，如果输入 $\boldsymbol{d}$的度数和 $S = \sum_{i=1}^{n} d_i$满足 $S > \lfloor \frac{n^2}{2} \rfloor$（请回想，简单图的度数和等于其边数的两倍），那么我们的算法便可立即报告不存在无三角形实现并停止。

下一条用于证明输入 $\boldsymbol{d}$ 没有无三角形实现的规则，采用了 [4] 中引入的任意可图度序列的剩余量概念。剩余量 $R(\boldsymbol{d})$ 是可图度序列 $\boldsymbol{d} = \{d_1 \geq d_2 \geq \cdots \geq d_n\}$ 在迭代执行 Havel-Hakimi 约化过程 [6], [7]的最后一步中得到的零的个数。如 [4],所示，可图度序列 $\boldsymbol{d}$ 的剩余量 $R(\boldsymbol{d})$ 是其每个实现的独立数的下界。若用 $\boldsymbol{d}$ 表示 $\boldsymbol{d}$的补可图度序列（即 $\underline{\boldsymbol{d}} = \{\overline{n-1-d_n} \geq n-1-d_{n-1} \geq \cdots \geq n-1-d_1\}$），它是 $\boldsymbol{d}$的任意实现的补图的度序列，那么，当算法发现 $R(\boldsymbol{d}) \geq 3$时，就可以立即报告不存在无三角形实现并停止。这是因为在这种情况下， $\boldsymbol{d}$ 的每个实现的团数都会 ≥ 3（也就是说，$\overline{\text{}}$包含一个三角形）。值得注意的是，对于长度为 n 的任意输入 $\boldsymbol{d}$， $R(\boldsymbol{d})$ 都可以在 $O(n^2)$ 时间内轻松计算出来。对于我们的算法，最有趣的输入 $\boldsymbol{d}$ 满足 $R(\boldsymbol{d}) = 2$ ($R(\boldsymbol{d}) = 1$ 当且仅当 $\boldsymbol{d}$ 是由 n 个零构成的度序列；在假设输入 $\boldsymbol{d}$ 满足 $d_n \geq 1$ 时，这种情况不可能发生）。此外，$R(\boldsymbol{d})$ 不一定是 $\boldsymbol{d}$ 所有实现中最小独立数的紧下界。事实上，这两个量之间的差可以任意大 [9]。因此，当我们的算法发现 $R(\boldsymbol{d}) = 2$ 时，就会继续进入阶段 2（见下文），并有可能最终报告不存在无三角形实现。

我们的算法采用的最后一条简单规则，用于证明输入 $\boldsymbol{d}$ 不存在无三角形实现。该规则使用了 Murphy [14], 中引入的 Murphy 界概念，此处记作 $\beta(\boldsymbol{d})$；它也是 $\boldsymbol{d}$的任意实现的独立数下界。因此，当算法发现 $\beta(\boldsymbol{d}) \geq 3$时，也可以立即报告不存在无三角形实现并停止。对于长度为 n 的任意给定输入 $\boldsymbol{d}$， Murphy 界 $\beta(\boldsymbol{d})$ 可在 $O(n)$ 时间内轻松计算。与剩余量的情况类似，大多数有趣的输入 $\boldsymbol{d}$ 都满足 $\beta(\boldsymbol{d}) \equiv 2$ ($\beta(\boldsymbol{d}) \equiv 1$， 当且仅当 $\boldsymbol{d}$ 是由 n 个零构成的度序列时。此外， $\beta(\boldsymbol{d})$不一定是 $\boldsymbol{d}$所有实现中最小独立数的紧下界；因此，当算法发现 $\beta(\boldsymbol{d}) = 2$（在发现 $R(\boldsymbol{d}) = 2$之后）时，仍需继续进入阶段 2。注意，存在 $\boldsymbol{d}$使得$\beta(\boldsymbol{d}) \geq 3$ 且 $R(\boldsymbol{d}) = 2$；另外也存在 $\boldsymbol{d}$ 使得$R(\boldsymbol{d}) \geq 3$ 且 $\overline{\beta}(\boldsymbol{d}) = 2$。

上述三条简单规则实际上也用于判断给定输入 $\boldsymbol{d}$ 是否存在二部实现[20]。如果输入 $\boldsymbol{d}$ 在阶段 1 通过上述三条规则的检验，那么我们的算法将进入阶段 2，构建搜索树 $\mathcal{T}(\boldsymbol{d})$，如 [21] 所示，以构造所有带标号的无三角形实现。我们强调，输入 $\boldsymbol{d}$ 有可能通过阶段 1 的三条规则检验，但仍不存在无三角形实现。这意味着算法可能需要耗费大量工作来构建和遍历搜索树 $\mathcal{T}(\boldsymbol{d})$，然后在阶段 2 结束时报告未找到无三角形实现。我们注意到，问题 $P(\boldsymbol{d}, 3)$的判定版本的复杂度尚不明确，而且我们怀疑该判定问题有可能无法在多项式时间内求解。因此，对于不存在无三角形实现的输入 $\boldsymbol{d}$ ，我们当前的生成算法并不总能避免进入阶段 2。

在第二阶段，算法从搜索树 $\mathcal{T}(\boldsymbol{d})$的根节点开始，该节点表示一个包含一组孤立顶点 V的标号图 $\{v_1, v_2, \cdots, v_n\}$。一般而言，对于每个$1 \leq k \leq n$，在搜索树 $\mathcal{T}(\boldsymbol{d})$深度 k 处的搜索树节点表示正在构造的部分标号图 G ；其中，顶点 v_1、v_2、$\cdots$、v_k已分别达到饱和状态（即已分别分配 d_1、d_2、$\cdots$、d_k个邻点），且迄今为止该部分实现 G中不含三角形；剩余顶点 v_{k+1}、v_{k+2}、$\cdots$、v_n则可能适当地添加更多邻点，从而生成一个完整的带标号无三角形实现 $\boldsymbol{d}$ ，前提是存在一个带标号的无三角形实现 $\boldsymbol{d}$ ，其中将当前部分实现 G 及顶点 v_1、v_2、$\cdots$、v_k当前的邻点分配作为子群包含在内。

与 [21], 中一样，我们的算法将以深度优先方式遍历 $\mathcal{T}(\boldsymbol{d})$的搜索树节点，并且在遍历整个搜索树的过程中，只维护一份（部分或完整的） G 的 $\boldsymbol{d}$带标号的无三角形实现；从一个搜索树节点转到下一个节点时，算法会动态更新这份标号图 G 。遍历深度为 k的搜索树节点时，算法会根据标号图 G中的顶点 v_k是否已饱和而采取不同的处理方式。

如果 v_k 尚未饱和，那么算法将为 v_k构建一个候选邻居集合 C_k ，其中 C_k 是 $U_k = \{v_{k+1}, \cdots, v_n\}$ 中那些在正在构造的带标号无三角形图 G的副本中尚未饱和的顶点所组成集合的一个剪枝子集。我们无需考虑将 $L_k = \{v_1, v_2, \cdots, v_{k-1}\}$ 中的任何顶点纳入 C_k，因为在搜索树深度 k 处， L_k 中的所有顶点都必定已经饱和。构造集合 C_k时将应用以下剪枝:

(1) 如果未饱和顶点 $y \in U_k$ 与 v_k 的任一邻点 $x \in L_k$ 相邻，那么它就不会被纳入 C_k。这是因为我们要避免在由 $\{v_k, x, y\}$ 构成的 G 中出现三角形。

如果不应用上述剪枝，由于假设输入 $\boldsymbol{d}$是一个可图度序列，我们总有$|C_k| \geq r_k$；其中， r_k 是当前搜索树节点处的 G 中尚未为 v_k 添加任何邻居之前的剩余度。当应用剪枝时，可能会出现$|C_k| < r_k$ ，从而使 v_k 无法以无三角形的方式达到饱和，因此会剪去以当前搜索树节点为根、并会出现在 $\mathcal{T}_0$（$\boldsymbol{d}$）中的这部分搜索树。如果剪枝后的邻居候选集 C_k 对于 v_k 至少包含 r_k 个元素，那么我们的算法将使用子集枚举算法，从 C_k 中选出字典序最小的 r_k 个邻居，使得在 G中为 v_k 添加这组r_k 个邻居后，仍有可能通过适当为 $\{v_{k+1}, \cdots, v_n\}$添加其他邻居，完成 $\boldsymbol{d}$ 的带标号无三角形实现 G 。与 [21],一样，算法还需要使用若干种等价判据中的任一种，验证为 v_k 从 C_k 中添加可能的 r_k邻居子集后， n 个顶点的剩余度序列 $\boldsymbol{r}$仍然可图，以便作出判断给定的整数序列是否为可图序列[17]。然而，在对从 C_k 中为 v_k选取 r_k个邻居子集后得到的每个剩余度序列执行可图性检查之前（或之后），算法会执行下方所示的另外三项检查，以确定从 C_k 中为 v_k 选取 r_k个邻居子集的当前选择，是否会导致无法完成 $\boldsymbol{d}$的带标号无三角形实现 G。

(i) 令 σ_1 为所选 r_k 子集中各邻点在与 v_k 投影邻接后、来自 C_k 的剩余度之和，其中这些邻点属于 v_k 。令 σ_2 为剩余度为正、但不在当前所选 r_k 子集中的那些顶点的剩余度之和，这些顶点来自 C_k。请注意，用于计算 σ_2 的这些顶点可能在 C_k 中，也可能不在其中，因为为避免在 G 中出现三角形，某些剩余度为正的顶点可能尚未被加入 C_k （参见我们在构造 C_k 时采用的前述剪枝）。如果 $\sigma_1 > \sigma_2$，那么当前所选的 C_k 的 r_k 子集无法使 v_k 饱和；也就是说，它会导致无法完成 $\boldsymbol{d}$ 的带标号无三角形实现 G 。这是因为来自 C_k 的这些 r_k 顶点都会成为 v_k 的邻居，而为了避免在 G 中出现三角形，在当前搜索树节点的任何后代搜索树节点中，我们都不能让其中任意两个顶点相邻。这意味着，为这些 r_k 顶点各自添加的其他邻点必须来自那些剩余度仍为正、但位于当前 r_k 邻点子集之外的顶点；这些邻点来自 C_k ，且属于 v_k。因此，如果 $\sigma_2 < \sigma_1$，那么就无法以无三角形的方式使当前 v_k 的 r_k 邻点子集中的所有顶点都饱和，同时又在后代搜索树节点中保留当前为 v_k 所选的邻点。

(ii) 计算 σ_1 时，我们会跟踪从 C_k 中选出的、作为 v_k 的邻居的 r_k 个顶点中， m 的最大剩余度；这些顶点均已在当前搜索树节点上与 v_k 相邻。计算 σ_2 时，我们会跟踪这样的顶点数量 c：它们仍具有正剩余度，但不属于当前为 v_k 选取的 C_k 的 r_k-子集（回想一下，σ_2 是这些 c 个顶点的剩余度之和：见上文）。如果 $m > c$，那么当前为 v_k 选取的 C_k 的 r_k-子集也无法饱和。如上所述，在当前搜索树节点的任何后代搜索树节点中，这些为 v_k 从 C_k 选出的 r_k 个邻点的每个后续邻居，都必须来自当前 C_k 的 r_k-子集之外且剩余度为正的顶点集合，以避免在 G 中形成三角形。如果 $c < m$，那么 C_k 的 r_k-子集中剩余度最大的顶点 m（回想一下， m 是在该顶点与 v_k 相邻之后计算的）在任何后代搜索树节点中都没有足够数量的邻居。

(iii) 如果 $m = c$while $\sigma_1 \neq \sigma_2$，那么当前选定的 r_k 子集 C_k 仍无法饱和 v_k。这是因为所有剩余度为正的 c 顶点

在所选 r_k 子集 C_k 之外，在它们 全部 与所选 r_k 子集 C_k 中剩余度最大的顶点相邻之后，彼此之间都不能再添加边（同样是为了避免在 G 中形成三角形）。因此，后代搜索树节点中这些 c 顶点的任何后续邻居都必须来自当前选定的 r_k 子集 C_k。因此，为了满足可行性，必须有 $\sigma_1 - m = \sigma_2 - c$ （即 $\sigma_1 = \sigma_2$）。

对于枚举出的 C_k 的每个 r_k-子集，上述三项可行性检查 (i) $\sigma_1 \leq \sigma_2$ 、(ii) $m \leq c$ 和 (iii) $m = c \Rightarrow \sigma_1 = \sigma_2$ ，既可以在检查由从 C_k 中为 v_k 添加 r_k-子集邻居后得到的剩余度序列 $\boldsymbol{r}$ 的可图性之前进行，也可以在之后进行。如果三项可行性检查未能全部通过，则可以跳过对 $\boldsymbol{r}$ 的图性检验。如果 $\boldsymbol{r}$ 的图性检验未能通过，则可以跳过全部三项可行性检查。一旦算法找到从 C_k 中为 v_k 选取的字典序最小的 r_k 邻居子集，并且该子集通过全部三项可行性检查以及对 $\boldsymbol{r}$ 的图性检验，算法就会相应地更新正在构造的标号图副本 G，然后向下遍历到搜索树 $\mathcal{T}(\boldsymbol{d})$ 的下一深度 $k+1$。如 [21]，所述，为了找到一个可能可行的、从 C_k 中为 v_k 选取的 r_k 邻居子集（此处 $K = r_k$，$M = |C_k|$ ），我们需要一种基本的组合算法，按字典序生成含有 M 个元素的集合的所有 K-子集。我们注意到，通过所有这些检验并找到一个可能可行的、从 C_k 中为 v_k 选取的 r_k 邻居子集，并不能保证以当前搜索树节点为根的搜索树部分 $\mathcal{T}(\boldsymbol{d})$ 中存在 $\boldsymbol{d}$ 的带标号无三角形实现。

我们已经完成了对 v_k 未饱和时生成算法行为的描述。如果 v_k 已经饱和，那么在相同情况下，算法的行为将与 [21] 中的行为类似。设标号图 G 的副本中饱和顶点 v_k 的邻居集合为 $N(v_k) = L(v_k) \cup R(v_k)$（其中，$L(v_k)$是 $L_k = \{v_1, \cdots, v_{k-1}\}$ 中的 v_k 的邻居集合，而 $R(v_k)$是 $U_k = \{v_{k+1}, \cdots, v_n\}$中的 v_k 的邻居集合）；如果算法是第一次访问当前搜索树节点，那么它将直接沿搜索树 $\mathcal{T}(\boldsymbol{d})$ 向下遍历至深度 $k+1$。如果这不是第一次访问当前搜索树节点（即回溯过程中，它从深度为 $k+1$ 的下方搜索树节点访问该节点），那么它将尝试为 v_k 找到一个新的邻居集合 $N'(v_k) = L(v_k) \cup R'(v_k)$：方法是为 v_k 找到下一个有效且按字典序更大的邻居集合 $R'(v_k)$（即按字典序大于 v_k的邻居），并满足$|R'(v_k)| = |R(v_k)|$，从而有可能得到 $\boldsymbol{d}$的一个完整的带标号的无三角形实现 G。为了给 v_k 找到 $R'(v_k)$，我们无需枚举比当前 $R(v_k)$ 按字典序更大的所有 $|R(v_k)|$-子集 $U_k = \{v_{k+1}, \cdots, v_n\}$。相反，我们可以先构造一个针对 v_k 的邻居禁用子集 F_k，其中包含 U_k 中所有与 $L_k = \{v_1, \cdots, v_{k-1}\}$内 v_k 的任一邻居 x 相邻的顶点 y。也就是说，对于某个 $x \in L(v_k)\}$，有 $F_k = \{y \in U_k | yx \in E(G)$；这与我们先前在考虑 v_k 未饱和的情况、构造 C_k 时为避免在 G中出现三角形而采用的剪枝技术类似。形成 F_k 后，我们的算法便枚举所有按字典序大于当前 $R(v_k)$的 $|R(v_k)|$-子集 $U_k - F_k$。这有可能减少为寻找 $R'(v_k)$而枚举的子集数量，并剪去许多原本会出现在 $\mathcal{T}_0(\boldsymbol{d})$中的搜索树节点。对于枚举出的每个 $|R(v_k)|$-子集 $U_k - F_k$，我们仍需按照与 [21]中相同的方式构造剩余度序列 $\boldsymbol{r'}$，并对其进行可图性检验。如果能为 v_k 找到下一个有效且按字典序更大的邻居集合$R'(v_k)$，并且该集合通过了对 $\boldsymbol{r'}$ 进行的可图性检验，那么正在构造的标号图 G副本就会相应更新，随后算法沿搜索树向下遍历至深度 $k+1$。如果未能为 v_k找到这样的集合 $R'(v_k)$，那么算法将在不更改正在构造的标号图 G副本的情况下，回溯到当前搜索树节点的父节点。同样，为了找到 $R'(v_k)$，这里也需要用到基础组合算法，以生成集合中由 M个元素组成的 K-子集（满足 $K = d_k - |L(v_k)| = |R(v_k)|, M = |U_k - F_k|$），且这些子集按字典序大于 $R(v_k)$。还要注意，由于在尝试寻找$R'(v_k)$之前引入了禁用集 F_k，因此在考虑 v_k 未饱和的情况时，我们无需进行上述可行性检查 (i)–(iii) 一类的可行性检查。

有趣的是，我们可以将生成算法构建并遍历的搜索树 $\mathcal{T}(\boldsymbol{d})$ 视为搜索树 $\mathcal{T}_0(\boldsymbol{d})$ 的一棵子树；后者是在尝试构造输入可图序列 $\boldsymbol{d}$ as in [21] 的所有标记实现时构建的。如果输入 $\boldsymbol{d}$ 不存在无三角形实现，那么 $\mathcal{T}_0(\boldsymbol{d})$ 的所有叶节点都会被剪枝，并从 $\mathcal{T}(\boldsymbol{d})$ 中消失；算法在遍历完 $\mathcal{T}(\boldsymbol{d})$ 并于阶段 2 结束时终止后，会报告这一结论。对于大多数存在无三角形实现的输入序列 $\boldsymbol{d}$，我们的算法会保留 $\mathcal{T}_0(\boldsymbol{d})$ 中恰好代表这些无三角形实现的叶节点，同时在搜索树的内部节点处剪掉其他叶节点（它们代表团数为 ≥ 3 的标记实现）。事实上，可以高效计算输入 $\boldsymbol{d}$ 的任一实现的最大团数 $\Omega(\boldsymbol{d})$；相比之下，如 [2] 中所猜想的，计算输入 $\boldsymbol{d}$ 的任一实现的最小团数 $\omega(\boldsymbol{d})$ 是难以处理的。由于剪枝，对于许多输入 $\boldsymbol{d}$，遍历并构建 $\mathcal{T}(\boldsymbol{d})$ 的总运行时间，可能远短于遍历并构建整个 $\mathcal{T}_0(\boldsymbol{d})$ 的时间，正如第 VI 节稍后所展示的。对于某些罕见的输入序列 $\boldsymbol{d}$，其所有实现都是无三角形的。在这种情况下，$\mathcal{T}(\boldsymbol{d})$ 与 $\mathcal{T}_0(\boldsymbol{d})$ 相同（即实际上没有发生剪枝）。显然，如果 $\Omega(\boldsymbol{d}) = 2$，那么 $\mathcal{T}(\boldsymbol{d})$ 与 $\mathcal{T}_0(\boldsymbol{d})$ 完全相同。如果 $\Omega(\boldsymbol{d}) > 2$，那么 $\mathcal{T}(\boldsymbol{d})$ 将是 $\mathcal{T}_0(\boldsymbol{d})$ 的一棵真子树。如果 $\omega(\boldsymbol{d}) = 2$，那么 $\mathcal{T}(\boldsymbol{d})$ 将包含 $\mathcal{T}_0(\boldsymbol{d})$ 的某些叶节点。如果 $\omega(\boldsymbol{d}) > 2$，那么 $\mathcal{T}(\boldsymbol{d})$ 将不包含 $\mathcal{T}_0(\boldsymbol{d})$ 的任何叶节点。

最后我们指出，构建和遍历

搜索树 $\mathcal{T}(\boldsymbol{d})$ 的阶段 2 可以采用与并行化 [21]，中构建和遍历搜索树 $\mathcal{T}_0(\boldsymbol{d})$ 的算法类似的方式实现并行化，尽管该并行算法的负载均衡问题仍然颇具挑战。

IV. 二部实现的扩展生成算法

一个密切相关的组合生成问题是，生成给定可图序列 $\boldsymbol{d}$ 的所有二部实现。由于可以在 $O(n^2)$ 时间内轻松判断任意一个 n 阶简单图是否为二部图（或等价地，是否为二着色），因此，我们只需在第 III 节中先前的生成算法上，添加一项测试，用于判断每个生成的无三角形实现 G 是否为二部图，就可以进一步扩展该算法。这样也就能按字典序生成输入序列 $\boldsymbol{d}$ 的所有带标号的二部图实现。此外，在扩展算法进入阶段 2 以构建并遍历搜索树 $\mathcal{T}(\boldsymbol{d})$ 之前，实际上还可以在 [20] 中采用若干额外的简单高效检查（其中共列出了七条规则）作为阶段 1 操作，排除输入 $\boldsymbol{d}$ 不存在任何二部实现的情形。事实上，这里用于判定输入 $\boldsymbol{d}$ 是否具有潜在二分性的整个判定算法，也可以用于我们的扩展生成算法：在进入阶段 2 之前的阶段 1 中运行该算法；如果阶段 1 判定输入 $\boldsymbol{d}$ 不存在二部实现，则跳过阶段 2。请注意，[20] 的主要目标是解决一个判定问题，而不尝试生成任何二部实现。如果目标是生成输入 $\boldsymbol{d}$ 的所有二部实现，而不考虑任何非二部的无三角形实现，那么与第 III 节所述的无三角形实现生成算法相比，[20] 中的判定算法可以帮助我们的二部实现扩展算法对更多可能的输入 $\boldsymbol{d}$ 跳过阶段 2。我们将 $\boldsymbol{d}$ 的所有带标号二部实现构成的集合记为 $\mathcal{B}(\boldsymbol{d})$。

V. 复杂度分析

我们在第 III 节中仅对用于生成无三角形实现的算法复杂度作非正式讨论。算法的阶段 1 很简单，容易看出其运行时间为$O(n^2)$。阶段 2 较为复杂，很难针对每种具体情况给出严格且准确的运行时间复杂度分析。不过，其复杂度不会高于用于生成所有标记实现的一般算法 [21]，而且对于大多数实现都不是无三角形实现的那些可图序列，由于采用了剪枝，实际复杂度通常会低得多。最优情形下的运行时间复杂度为 $O(n)$，例如对于序列 $((n-1)^1, 1^{n-1})$，它是阶为 n 的星图的度序列。我们预计，要计算所有长度为 n 的无零图序列的平均情形运行时间复杂度，或者计算其中会迫使算法进入阶段 2 的子集的平均情形运行时间复杂度，将会非常困难。空间复杂度很容易看出为 $O(n^2)$。

我们猜想，判定问题 $P(\boldsymbol{d},\ 3)$ 可能是 NP 中既非 NP 完全也不属于 P 的候选问题。也就是说，如果 $P \neq NP$，那么这可能是一个 NP中间问题；对于两个著名问题——图同构和整数分解——也有人提出了同样的猜想 [1]。

VI. 实验评估

我们用 C++ 实现了第 III 节中针对无三角形实现的穷举生成算法，以及第 IV 节中针对二部图实现的扩展生成算法；对于实现输入 $\boldsymbol{d}$ 的标号图 G，我们选用的数据结构是邻接表。

我们首先采用以下策略验证算法设计和实现的正确性。我们使用来自 [16] 的穷举生成算法，生成长度为 n 的所有无零图序列。对于每个这样的图序列 $\boldsymbol{d} = \{d_1 \geq d_2 \geq \cdots \geq d_n\}$，我们使用穷举生成算法的实现，生成并统计标记无三角形图（分别为二部图）实现 $|\mathcal{S}(\boldsymbol{d})|$（分别为 $|\mathcal{B}(\boldsymbol{d})|$）的数量。假设 $\boldsymbol{d}$ 也可以写作 $b_1^{c_1}, b_2^{c_2}, \cdots, b_s^{c_s}$，其中$b_1 > b_2 > \cdots > b_s$是 $\boldsymbol{d}$ 中所有互不相同的度，并且对于 $1 \leq i \leq s$，$\boldsymbol{d}$ 中有 c_i 个 b_i 。那么，不要求顶点标号的字典序与顶点度数的排列顺序一致时，$\boldsymbol{d}$ 的标记无三角形图（分别为二部图）实现数量为 $|\mathcal{S}(\boldsymbol{d})|$ $n!\prod_{i=1}^{s} c_i!$（分别为 $|\mathcal{B}(\boldsymbol{d})|$ $n!\prod_{i=1}^{s} c_i!$）。对长度为 n 的所有无零图序列求和后，我们应得到具有 n 个非孤立顶点的标记无三角形图（分别为二部图）总数。该总数也可以通过 OEIS A345218（记录 n 个顶点上的标记连通无三角形图数量）和 A001832（记录 n 个顶点上的标记连通二部图数量）独立计算（使用欧拉变换 [18] 的一种变体）。我们用这种方法验证了所有 $n \leq 12$ 的实现是否正确。对于 $n \geq 13$，似乎有些图序列 $\boldsymbol{d}$ 所对应的输出标记无三角形图数量 $|\mathcal{S}(\boldsymbol{d})|$ 过大，无法在合理的时间内完成生成。

我们还可以采用另一种策略来验证算法设计和实现是否正确：使用 nauty 中广为人知的工具 **uniqg** [13]。我们可以执行以下验证步骤：(1) 使用我们的实现生成 $\boldsymbol{d}$ 的所有带标号无三角形二部实现；(2) 使用工具 **uniqg** 获取 $\boldsymbol{d}$ 的无标号无三角形二部实现的数量；(3) 对所有长度为 n 的无零图序列 $\boldsymbol{d}$对应的这些数值求和，得到具有 n个非孤立顶点的无标号无三角形二部图的数量。该数量也可以通过 OEIS A024607（记录具有 n个无标号顶点的连通无三角形图的数量）和 A005142（记录具有 n 个无标号顶点的连通二部图的数量）独立计算得出（使用欧拉变换 [18]）。我们还使用这种方法验证了所有 $n \leq 12$的实现是否正确。

表 I 展示了一些特定输入实例 $\boldsymbol{d}$ 的计算结果，以及全部 $|\mathcal{G}(\boldsymbol{d})|$的数量

带标号实现的数量、全部带标号无三角形实现的数量，以及全部带标号二部图实现的数量 $|\mathcal{S}(\boldsymbol{d})|$。其中一些示例输入也出现在 [21]中。对于 $|\mathcal{S}(\boldsymbol{d})|$或 $|\mathcal{B}(\boldsymbol{d})|$，使用我们的实现均可在五秒内计算出这些结果。

表 I一些数值结果 $|\mathcal{S}(\boldsymbol{d})|$ 和 $|\mathcal{B}(\boldsymbol{d})|$

$\boldsymbol{d}$	$\lvert\mathcal{G}(\boldsymbol{d})\rvert$	$\lvert\mathcal{S}(\boldsymbol{d})\rvert$	$\lvert\mathcal{B}(\boldsymbol{d})\rvert$
2,2,2,2,2	12	12	0
3,3,2,2,2	7	1	1
3,3,2,2,1,1	17	6	6
4 3 3 2 2 1 1	65	5	5
4,3,3,3,3,3,3,3	810	0	0
4,4,3,3,3,2,2,1	1931	33	33
3,3,3,3,3,3,3,3	19355	3360	840
4,4,4,4,3,3,3,3	14634	36	36
6 4 4 4 4 4 3 3 2	109055	0	0
4,4,4,4,3,3,3,3,3,3	17262690	203220	27180
6,5,4,4,3,3,3,2,2,2	798279	94	87
3,3,3,3,3,3,3,3,3,3	11180820	1829520	257040

少数可图序列 $\boldsymbol{d}$ 满足 $\Omega(\boldsymbol{d}) = 2$，因此其所有实现都是无三角形的。例如，$(3^1, 1^3)$ 和 (2^5)。大多数可图序列 $\boldsymbol{d}$ 满足 $\Omega(\boldsymbol{d}) > 2$，因此并非其所有实现都是无三角形的。一些可图序列 $\boldsymbol{d}$ 满足 $\omega(\boldsymbol{d}) > 2$，因此它们都不存在无三角形实现。例如，$(4^1, 3^6)$ 和 $(6^1, 4^5, 3^2, 2^1)$。对于后一个序列，我们的算法无需进入阶段 2，就能得出不存在无三角形实现的结论。对于前一个序列，我们的算法在阶段 1 无法得出不存在无三角形实现的结论，因此必须进入阶段 2，遍历整个搜索树后才能报告这一事实。

在那些至少有一个无三角形实现的可图序列 $\boldsymbol{d}$ 中，其无三角形实现中有多少个为二部图，存在三种可能性：(1) 所有无三角形实现都是二部图（例如，可图序列 $(3^2, 2^2, 1^2)$ 和 $(4^4, 3^4)$）；(2) 部分但并非全部无三角形实现是二部图（例如，可图序列 $(4^4, 3^6)$ 和 (3^{10})）；或者 (3) 没有一个无三角形实现是二部图（例如，可图序列 (2^5) 和 $(8^3, 7^1, 4^7, 3^1, 2^1)$，后一个序列未显示在表 I 中）。给定这样的可图序列 $\boldsymbol{d}$（即 $\omega(\boldsymbol{d}) = 2$），我们当然可以使用第四节中的扩展生成算法来判断上述三种情况中的哪一种会发生。然而，如果采用不同的方法，判断三种情况中的哪一种会发生，仍可能颇具挑战性。

由于我们的算法采用了有效的剪枝技术，实现通常能够很快完成，远快于生成 [21] 中所有带标号实现的速度。例如，对于长度为 13 的序列 $(7^1, 5^9, 3^1, 2^1, 1^1)$，算法约 5 秒即可找到全部 27720 个带标号的无三角形实现和全部 17640 个带标号的二部图实现，而生成该序列的所有带标号实现则需要很长时间（> 35 天）。

我们还调整了先前用 C++实现的 James-Riha 算法，在 [8] 中通过纳入 [21] 中提到的 $\boldsymbol{d}$给定可图序列的所有实现所采用的第 III 节和第 IV 节中介绍的剪枝技术，并且只生成无三角形实现或二部图实现。为了验证我们调整后的算法实现是否正确，我们使用工具 **uniqg** 筛除输出图中的同构重复项，并计算一些短可图序列 $\boldsymbol{d}$ 的无标号无三角形实现或二部实现的数量；这些序列 $n \leq 17$ 对应的此类无标号实现数量已预先计算。我们在实验中得到的所有计算结果都与预先计算的值相符。对于输入 $\boldsymbol{d}$，调整后算法输出的图数量通常小于$|\mathcal{S}(\boldsymbol{d})|$ 或 $|\mathcal{B}(\boldsymbol{d})|$，不过通常仍大于$\boldsymbol{d}$的无标号无三角形实现或二部实现的数量。这是因为调整后的算法旨在减少输出中的同构重复项数量，但无法保证将其完全消除。此外，其运行时间通常短于第 III 节和第 IV 节中穷举生成算法的实现，因为这两种算法所完成的任务并不完全相同。

VII. 结论

本文提出了一种算法，能够按字典序生成任意可图序列 $\boldsymbol{d}$ 的所有带标号的无三角形实现。还可以进一步检验生成的图是否为二部图，从而生成所有带标号的二部图实现。该算法可用于解决判定问题 $P(\boldsymbol{d}, 3)$：判断输入的可图序列 $\boldsymbol{d}$ 是否存在无三角形实现。然而，判定 $P(\boldsymbol{d}, 3)$ 的复杂度，以及更一般的问题 $P(\boldsymbol{d}, r)$ 的复杂度，仍然未知。

参考文献

[1] Sanjeev Arora 和 Boaz Barak。《计算复杂性：一种现代方法》。剑桥大学出版社，2009。[2] Douglas Bauer、S. Louis Hakimi、Nathan Kahl、E. Schmeichel。《各种图参数的最佳单调度数界》。*Congressus Numerantium*，192，2008。[3] P. Erd˝os 和 T. Gallai。具有给定顶点度数的图。*Mat. Lapok*，11：264–274，1960。[4] O. Favaron、M. Mah´eo 和 J.-F. Sacl´e。关于图的剩余度。《图论杂志》，15(1)：39–64，1991。[5] Oded Goldreich。《计算复杂性：概念视角》。剑桥大学出版社，2008。[6] S. L. Hakimi。关于将一组整数实现为线性图顶点度数的可实现性 I。工业与应用数学学会杂志，10(3)：496–506，1962。[7] V. Havel。关于有限图存在性的一则说明。*ˇCasopis pro pˇestov´an´ı matematiky*，4：477–480，1955。[8] K. R. James 和 W. Riha。算法28：生成具有给定划分的图的算法。*Computing*，16：153–161，1976。[9] Frank Jelen。具有指定度序列的图的 k-独立性和 k-剩余度（德文）。毕业论文，波恩大学，1996。[10] Richard M. Karp。组合问题之间的可约性。计算机计算的复杂性，85–103，1972。[11] Fr´ed´eric Maffray 和 Myriam Preissmann。无三角形图的 k-可着色问题的 NP 完全性。离散数学，162(1-3)：313–317，1996。[12] W. Mantel。问题28（由 H. Gouwentak、W. Mantel、J. Teixeira de Mattes、F. Schuh 和 W. A. Wythoff 解答）。*WiskundigeOpgaven*，60–61，1907。[13] B. D. McKay 和 A. Piperno。实用图同构，第二部分。符号计算杂志，60：94–112，2014。

[14] O. Murphy。用度数计算的图稳定数下界。离散数学，90(2)：207–211，1991。[15] J. Mycielski。关于图的着色。*Colloq.Math.*，3 162，1955 2 161( )：–。[16] Frank Ruskey、Robert Cohen、Peter Eades 和 Aaron Scott。Alley CATs 寻找理想家园。载于第*25*届东南部组合学、图论与计算会议，102：97–110。Congressus Numerantium，1994。[17] Gerard Sierksma 和 Han Hoogeveen。判定整数序列可图的七项准则。《图论杂志》，15 231，1991 2 223( )：–。[18] N. J. A. Sloane 和 S. Plouffe。《整数序列百科全书》。圣迭戈，佐治亚州：学术出版社，第20–21页，1995。[19] D. L. Wang 和 D. J. Kleitman。具有指定度数的 n-连通图（ $n \geq 2$ 3 3 225 239, ）的 1973 存在性。《网络》，( )：–。[20] Kai Wang。一种检验可图度序列潜在二分性的高效算法。《图论及其应用》，8(1)，2021。[21] Kai Wang。一种生成给定可图度序列的所有带标记图的高效算法。计算科学与计算智能，第2506卷，施普林格，Cham，2024。[22] J. H. Yin。A.R. Rao 对潜在 K_{r+1}-可图序列刻画的一个简短构造性证明。《离散应用数学》，160(3)：352–354，2012。