最优化小论文

22120307 陈景龙
北京交通大学

日期：2023 年 1 月 15 日

摘　　要

寻求 NP-Hard 问题的较优算法是一重要内容，其中，与独立集相关的问题较为常见。本文将从独立集的性质入手，介绍基于多种思想的几类独立集算法。对于一般图，本文在搜索算法的基础上，提出了动态规划的优化算法，同时分析并测试了几种算法在随机数据下的运行效率；对于特殊图，本文介绍了两种针对特殊图的算法思想，提出了求解“k-仙人图”的独立集问题的算法思想及其扩展应用。

关键词： 优化问题，独立集算法，NP-Hard

目录

1 前言 **3**

2 定义及约定 **3**

3 一般图的独立集问题 **3**

3.1 基于极大独立集搜索的独立集算法 4

3.1.1 朴素的搜索算法 4

3.1.2 极大独立集与 Bron-Kerbosch 算法 4

3.1.3 极大独立集的个数 5

3.1.4 应用 7

3.2 基于动态规划的独立集算法 8

3.2.1 算法 8

3.2.2 效率优化 10

3.2.3 与搜索算法的联系 11

3.2.4 测试与对比 11

4 特殊图的独立集问题 **12**

4.1 基于图匹配思想的最大独立集算法 12

4.1.1 二分图的最大独立集 12

4.1.2 无爪图的最大独立集 12

4.2 基于图上阶段划分思想的最大独立集算法 13

4.2.1 分层图上的动态规划 13

4.2.2 “k-仙人图”上的动态规划 13

5 总结 **16**

1 前言

不少与独立集有关的问题（最大独立集、最大带权独立集、独立集计数等问题）都是图论中经典的 NP-Hard 问题，在信息学竞赛中广泛出现，然而解决独立集问题的算法效率通常不高。由此，本人对此类问题进行了更深入的研究，希望能用更加高效的方法解决此类问题。

本文的研究分为两部分，第一部分介绍了两种求解一般图的独立集问题的算法：基于极大独立集搜索的独立集算法和基于动态规划的独立集算法，第二部分介绍了两类特殊图的独立集算法，分别基于图匹配思想和图上的阶段划分思想。

2 定义及约定

独立集问题有多种形式。为了方便描述，以下给出一些定义。

定义 2.1. 对于无向图 $G = (V, E)$ 和点 $u, v \in V$，若 $(u, v) \in E$，则称 u, v 相邻 (adjacent)；定义点 $v \in V$ 的领域 (neighborhood) 为 V 中与 v 相邻的结点集合，记为 $N(v)$；另外，$N_G(v)$ 表示 v 在图 G 中的领域。

定义 2.2. 点 v 的度 (degree) $\deg(v)$ 定义为 $N(v)$ 的大小，即 $\deg(v) = |N(v)|$；另外，$\deg_G(v)$ 表示 v 在图 G 中的度。

定义 2.3. 无向图 $G = (V, E)$ 的一个独立集 (independent set) 定义为 V 的一个子集，满足子集中的结点两两不相邻。形式化地，I 是 G 的一个独立集，当且仅当 $I \subseteq V$ 且 $\forall u, v \in I$，$(u, v) \notin E$。

定义 2.4. 无向图 $G = (V, E)$ 的一个最大独立集 (maximum independent set) 是指 G 中所含结点数 $|I|$ 最多的独立集 I。

定义 2.5. 无向图 $G = (V, E)$ 的独立数 (independence number)[1]定义为 G 的最大独立集 I 所含的结点数 $|I|$，记为 $\alpha(G)$。

定义 2.6. 无向图 $G = (V, E)$ 在 $S \subseteq V$ 上的导出子图 (induced subgraph) 定义为以 S 为点集，两端点都在 S 内的边为边集构成的图，记为 $G[S]$。

本文中的所有问题均以最大独立集问题为例，即对于给定的无向图 $G = (V, E)$，找出 G 的一个最大独立集 I。

为了方便起见，约定 n 为图 G 的阶数（即顶点数），m 为图 G 的边数，即 $n = |V|$，$m = |E|$。

3 一般图的独立集问题

目前，解决一般图的大多数独立集相关的问题都不存在多项式时间的算法，只能用复杂度较优的指数级算法。

[1]导出子图有点导出子图和边导出子图两种，本文中均指前者。

事实上，已有不少理论复杂度十分优秀的求图的最大独立集的算法，能够快速计算出上百阶的无向图的最大独立集[2]，但这些算法实现往往过于复杂，难以应用。笔者选择了一些相对高效又较易于实现的算法进行了研究。

3.1 基于极大独立集搜索的独立集算法

3.1.1 朴素的搜索算法

最朴素的搜索算法非常简单：用深度优先搜素枚举 V 的子集 $I \subseteq V$，即按一定顺序枚举每个点以 $v \in V$ 是否属于 I，一旦存在 $(u,v) \in E$ 使得 $u,v \in I$，就回溯。输出枚举的所有独立集 I 中，$|I|$ 最大的一个。该算法的复杂度为 $O(2^n m)$，效率太低。

针对最大独立集这一问题，可以进行一些剪枝。例如：

1. 若 $\deg(v) = 0$，则不存在与 v 关联的边，故总可以令 $v \in I$。
2. 若 $\deg(v) = 1$，考虑唯一的与 v 关联的结点 u，若 $u \notin I$，则总可以令 $v \in I$；否则，从 I 中删去 u 并加入 v，I 的大小不变。因此总可以令 $v \in I$。
3. 搜索时记录当前搜到的独立集的大小的最大值 a，记 P 为 $V - I$ 中不与 I 中结点相邻的点集，当 $|I| + |P| \leq a$ 时可进行最优性剪枝。

然而加入这些剪枝之后，复杂度仍然很高，难以接受。

3.1.2 极大独立集与 Bron-Kerbosch 算法

朴素的搜索算法效率太低，有没什么好的方法米优化呢？我们提出极大独立集的概念：

定义 3.1. 无向图 $G = (V, E)$ 的一个极大独立集 (maximal independent set) 是指 G 的一个独立集 I，满足对于任意的结点 $v \in V - I$，点集 $I + \{v\}$ 不是独立集。

通常情况下，一个图的极大独立集个数比独立集个数少得多。例如 50 个结点构成的链有 32,951,280,099 个独立集，却只有 1,221,537 个极大独立集。另外，不少有关独立集的组合优化问题都可以只考虑极大独立集，最大独立集问题就是这样一个例子：

定理 3.1. *每个最大独立集都是极大独立集。*

证明. 设 I 是一个最大独立集。对于任意的 $v \in V - I$，假如 $I + \{v\}$ 是独立集，因为 $|I + \{v\}| = |I| + 1 > |I|$，所以 $I + \{v\}$ 是一个比 I 更大的独立集，也就是说，I 不是最大独立集，与假设矛盾。所以 I 一定是一个极大独立集。 □

因此，如果能找出 G 的所有极大独立集，就能找出 G 的最大独立集。

求极大独立集的算法有很多，其中 Bron-Kerbosch 算法是实现较为简洁的一种。下面介绍求极大独立集的 Bron-Kerbosch 算法。

Bron-Kerbosch 算法可以对任意的无向图 G 求出其所有的极大独立集，其基本思想仍然是搜索优化。伪代码如下[3]：

[2]Robson 提出了一种复杂度约 $O(1.1888^n)$ 的最大独立集算法

[3]Bron-Kerbosch 算法有多种实现方式，本文介绍其中的一种。

Algorithm 1 BronKerbosch(R, P, X)

```
if P = X = ∅ then
    print R
end if
选择结点 u ∈ P ∪ X，使得 |P ∩ ({u} ∪ N(u))| 最小
for ∀v ∈ P ∩ ({u} ∪ N(u)) do
    BronKerbosch(P ∪ {v}, P − ({v} ∪ N(v)), X − ({v} ∪ N(v)))
    P ← P − {v}
    X ← X ∪ {v}
end for
```

调用 BronKerbosch(R, P, X) 函数，将输出 G 的所有包含 R 中的所有结点、P 中的任意多个结点且不包含 X 中的结点的所有极大独立集。调用 BronKerbosch$(\varnothing, V, \varnothing)$ 即可得到 G 的所有极大独立集。

实现时，集合可以用压位的方法存储，即用一个大小为 $\lceil \frac{n}{64} \rceil$ 的 64 位整数数组 A 存一个大小为 n 的集合 A'，$x \in A'$ 当且仅当 $A[\lceil \frac{x}{64} \rceil]AND2^{x \mod 64} > 0$（$A$ 数组下标从 0 开始）。因为独立集问题中，图的阶数 n 不会很大，所以压位的数组大小可以近似认为是一个常数。

上述算法的最关键之处在于 Pivoting。算法过程中，有一步是选择结点 $u \in P \cup X$，使得 $|P \cap (\{u\} \cup N(u))|$ 最小，u 称为 Pivot 结点。之后枚举 $\{u\} \cup N(u)$ 中，属于独立集 R 的第一个结点 v。这就是 Pivoting 的过程。

如果直按搜索极大独立集的话，效率是很低的，因为会搜到很多不是极大的独立集。例如当 G 为 n 阶零图[4]时，显然 V 是 G 的唯一的极大独立集，然而朴素的搜索枚举了某个结点不属于极大独立集时，尽管不可能搜出极大独立集，但算法还会继续搜索下去，浪费了大量时间。Pivoting 的正确性基于以下定理：

定理 3.2. *对于无向图 $G = (V, E)$ 和 $v \in V$，G 的任意极大独立集 I 满足 $I \cap (\{v\} \cup N(v)) \neq \varnothing$。*

证明. 证明假设存在极大独立集 I，满足 $I \cap (\{v\} \cup N(v)) = \varnothing$，则对于任意 $u \in I$，$u \notin \{v\} \cup N(v)$，即 $u \neq v$ 且 $(u, v) \neq E$。

因此 $I \cup \{v\}$ 也是一个独立集，且 $I \subsetneq I \cup \{v\}$，这说明 I 不是极大独立集，矛盾。 □

这样，我们就证明了该定理的正确性。尽管这样仍然会搜到一些不是极大的独立集，但这样的集合显然少了很多。

3.1.3 极大独立集的个数

之前我们只是感性地认识了极大独立集比较少，这里将给出 Bron-Kerbosch 算法的递归次数上界：

定理 3.3. *Bron-Kerbosch 算法的递归调用次数为 $O(3^{\frac{n}{3}})$。*

[4]零图定义为没有边的图，即 $G = (V, E)$ 为零图当且仅当 $E = \varnothing$。

由此可以得到推论：

定理 3.4. *n 阶无向图的极大独立集个数为 $O(3^{\frac{n}{3}})$。*

这个上界是很容易达到的，构造 $\lfloor \frac{n}{3} \rfloor$ 个相互独立的三元环即可。但在图随机生成的情况下，这个上界是很不满的。为了说明这一点，笔者对随机图的极大独立集个数进行了研究。

从边数为 m 的 n 阶简单无向图中随机生成一个图 $G = (V, E)$，记 G 的极大独立集个数 x，即

$$x = \sum_{S \subseteq V} \mathbb{I}(S \text{ is a maximal independent set })$$

其中 $\mathbb{I}(\cdot)$ 是示性函数。

考虑计算 x 的期望值 $\mathbb{E}(x)$。S 是 G 的极大独立集的条件为：

- S 是独立集，即对于任意的 $u, v \in S$，$(u, v) \notin E$；
- 对于任意的 $v \in V - S$，$V + \{v\}$ 不是独立集，即 $V - S$ 中的每个点至少与 S 中的一个点相邻。

用容斥原理，枚举 k 个 $V - S$ 中的点不与 S 中的点相邻。记 $i = |S|$，则剩下 $n - i - k$ 个点可以和 S 中的点连边，以及 $V - S$ 中任意两点（一共 $\frac{1}{2}(n-i)(n-i-1)$ 对点）可以连边。则满足 S 是极大独立集的图 G 个数为

$$\sum_{k=0}^{n-i} (-1)^k \binom{n-i}{k} \binom{(n-i-k)i + \frac{(n-i)(n-i-1)}{2}}{m}.$$

由于边数为 m 的 n 阶简单图共有 $\binom{\frac{n(n-1)}{2}}{m}$ 个，故有

$$\begin{aligned}
E(x) &= \sum_{S \subseteq V} P(\mathbb{I}(S \text{ is a maximal independent set })) \\
&= \sum_{i=0}^{n} \sum_{S \subseteq V, |S|=i} \binom{\frac{n(n-1)}{2}}{m}^{-1} \sum_{k=0}^{n-i} (-1)^k \binom{n-i}{k} \binom{(n-i-k)i + \frac{(n-i)(n-i-1)}{2}}{m} \\
&= \binom{\frac{n(n-1)}{2}}{m}^{-1} \sum_{i=0}^{n} \binom{n}{i} \sum_{k=0}^{n-i} (-1)^k \binom{n-i}{k} \binom{(n-i-k)i + \frac{(n-i)(n-i-1)}{2}}{m}
\end{aligned}$$

下面给出了一些计算结果（四舍五入）：

$E(x)$	$m = n$	$m = \lfloor \sqrt{3}n \rfloor$	$m = 2n$	$m = 3n$	$m = \frac{n^2}{4}$
$n = 20$	84	101	99	81	49
$n = 30$	706	933	909	691	157
$n = 40$	5.95×10^3	8.67×10^3	8.40×10^3	5.88×10^3	403
$n = 50$	5.02×10^4	8.07×10^4	7.76×10^4	5.01×10^4	891
$n = 60$	4.23×10^5	7.51×10^5	7.18×10^5	4.27×10^5	1779
$n = 70$	3.57×10^6	6.99×10^6	6.64×10^6	3.63×10^6	3291
$n = 80$	3.01×10^7	6.51×10^7	6.14×10^7	3.10×10^7	5730
$n = 90$	2.54×10^8	6.06×10^8	5.68×10^8	2.64×10^8	9506
$n = 100$	2.15×10^9	5.64×10^9	5.25×10^9	2.25×10^9	15154

可见随机情况下，极大独立集的个数远少于 $3^{\frac{n}{3}}$。另外，当 m 接近 $\sqrt{3}n$ 时，独立集个数 x 的期望值 $\mathbb{E}(x)$ 最大，而过于稠密的图，独立集个数相当少。

根据该结论，还可以进一步得出，$x \geq k \cdot \mathbb{E}(x)$ 的概率不超过 $\frac{1}{k}$，所以在大多数情况下随机图的极大独立集个数不会比期望值大太多。

值得注意的是，Bron-Kerbosch 算法复杂度不和极大独立集个数直接相关，所以用极大独立集个数的期望值分析 Bron-Kerbosch 算法的期望运行时间并不准确；事实上，存在复杂度和极大独立集个数直按相关的极大独立集搜索算法。

3.1.4 应用

介绍了极大独立集的性质及算法之后，我们来看看它有哪些应用。

例 3.1 (图的 3-染色问题)**.** 给定 n 阶简单无向图 $G = (V, E)$，用三种颜色对 V 中的结点进行染色，使得每条边 $(u, v) \in E$ 的两端点 u, v 颜色不同。满足 $n \leq 40$。

图的 3-染色问题也是著名的 NP-Hard 问题。朴素的算法是每次枚举一个与已确定颜色的结点相邻的结点颜色，需要枚举 $O(2^n)$ 中情况，无法通过 $n = 40$ 的数据。如何才能更加高效地求解?

先给出一个定理:

定理 3.5. *无向图 $G = (V, E)$ 能够 3-染色的充要条件是 G 存在一个极大独立集 I，使得图 $G - I$ 是二分图*[5]。

证明. (充分性) 设 I 为 G 的一个极大独立集，且 $G - I$ 为二分图，根据二分图的性质，存在点集 $X \subseteq V - I$，记 $Y = V - I - X$，使得对任意 $u, v \in X$ 或 $u, v \in Y$，有 $(u, v) \notin E$。

因为 I 是 G 的独立集，所以 $\forall u, v \in I$，有 $(u, v) \notin E$。

因此将 I 中的点染色为 1，X 中的点染色为 2，Y 中的点染色为 3 是一种合法方案。

(必要性) 设 $G = (V, E)$ 能够 3-染色，记 X, Y, Z 分别为染颜色 1, 2, 3 的点集。

由定义，对任意 $(u, v) \in E$，结点 u, v 不属于这三个集合中的同一个集合，因此 X, Y, Z 都是独立集。

如果 X 不是极大独立集，则存在以 $v \in V - X$，使 $X + \{v\}$ 是独立集。将 v 加入点集 X，同时，若 $v \in Y$，则将 v 从 Y 中删去；否则 $v \in Z$，将 v 从 Z 中删去。重复此过程直至 X 是极大独立集为止。

显然此时 Y, Z 仍然是 G 的独立集，即对于 $u, v \in Y$ 或 $u, v \in Z$，有 $(u, v) \notin E$，故 $G[Y \cup Z]$ 是二分图。问题得证。 □

判断图是否为二分图以及将其进行染色可以在 $O(m)$ 的时间内解决。因此只需用枚举图 G 的所有极大独立集 I，然后判断图 $G - I$ 是否为二分图:

1. 若对所有的极大独立集 I，图 $G - I$ 都不是二分图，则图 G 不能 3-染色。

[5]无向图 $G = (V, E)$ 是二分图 (bipartite graph) 定义为可以将 V 划分为两个集合 S 和 $V - S$，使得每条边的两个端点不在同一个集合内，即 $\forall (u, v) \in E$，$u \in S, v \in V - S$ 或 $u \in V - S, v \in S$。

2. 若存在一个极大独立集 I，使得图 $G-I$ 是二分图，则图 G 能 3-染色：将 I 中的结点用颜色 I 染色，$G-I$ 用颜色 2 和 3 进行二分图染色即可。

本题中，由于 G 是简单图，$m=O(n^2)$，所以该算法时间复杂度为 $O(3^{\frac{n}{3}}n^2)$。

例 3.2 (小 Q 运动季测试点 10)**.** 给定一个 n 元一次同余方程组

$$\begin{cases} a_{0,0}x_0+a_{0,1}x_1+\ldots+a_{0,n-1}x_{n-1}\equiv c_0 \pmod{b_0} \\ a_{1,0}x_0+a_{1,1}x_1+\ldots+a_{1,n-1}x_{n-1}\equiv c_1 \pmod{b_1} \\ \ldots \\ a_{m-1,0}x_0+a_{m-1,1}x_1+\ldots+a_{m-1,n-1}x_{n-1}\equiv c_{m-1} \pmod{b_{m-1}} \end{cases}$$

求一组解 $(x_1,x_2,\ldots,x_n)$ 满足尽量多的方程。

本例中仅讨论测试点 10。该测试点中，通过建立图论模型，将每个方程看成一个点，相互冲突的方程间连一条边，可以转化为点数 $n=90$，边数 $m=223$ 的无向图的最大独立集问题。由于具体转化过程超出了本文的范围，故略去。

用 Bron-Kerbosch 算法搜出所有极大独立集，输出其中最大的一个即可。这样做的效率如何呢?

笔者将朴素搜索算法 Simple-Search 和基于极大独立集的搜索算法 Maximal-Search 进行比较，两个算法仅使用了最基本的剪枝：将剩余的点全部加入 I 都不大于当前搜到的点集，得到的点集大小都不超过当前搜到的最大的独立集，则剪枝。由于仅仅测试的是 Maximal- Search 是否有比 Simple-Search 更优的运行效率，这里并没有加入更多依赖问题性质的剪枝。

对于 Simple-Search，笔者的程序经过运行若干小时，仍然只能得到大小为了 33 的独立集，并且程序未能结束。然而对于 Maximal-Search，笔者的程序仅用不到 1min 就得到了一组大小为 34 的独立集，仅用 3min 就证明，图的独立数确实为 34。可见用极大独立集进行搜索确实能大幅提高运行效率。

通过加入更多的剪枝优化，可以进一步缩短算法运行时间。

3.2 基于动态规划的独立集算法

动态规划是一种高效、灵活的处理问题的方法。在独立集问题中，动态规划不仅能求解最优化类问题（如最大独立集、最大权独立集），还能求解计数类问题（如独立集计数）。下面仍以最大独立集问题为例，但为了体现动态规划的通用性，接下来的讨论将不加入最优性剪枝等仅针对最优化问题的剪枝。

3.2.1 算法

如果要使用动态规划求解独立集问题，就需要将问题化为规模更小的子问题。对于独立集，我们有以下两个定理:

定理 3.6. *对于无向图 $G=(V,E)$ 和 $V'\subseteq V$，则对于任意 $I\subseteq V'$，I 是 G 的独立集当且仅当 I 是 $G[V']$ 的独立集。*

证明. (充分性) 当 I 是 $G[V']$ 的独立集时，对于任意 $(u,v)\in E$，若 $u,v\in V'$，显然 u,v 不同时属于 I；若 u,v 有一个不属于 V'，不妨设 $u\notin V'$，那么 $u\notin I$。因此 I 是 G 的独立集。

(必要性) 当 I 是 G 的独立集时，由于 $G[V']$ 的每条边都属于 G，故 $G[V']$ 的每条边至少有一个端点不属于 I。因此 I 是 $G[V']$ 的独立集。 □

定理 3.7. *对于无向图 $G=(V,E)$ 和 $v\in V$，若 $I\subseteq V$ 且 $v\in I$，则 I 是 G 的独立集当且仅当 $I-\{v\}$ 是 $G[V-\{v\}-N(v)]$ 的独立集。*

证明. 记 $T=V-\{v\}-N(v)$。

(充分性) 当 I 是 G 的独立集时，$N(v)\cap I=\varnothing$，所以 $I-\{v\}\subseteq T$，因为 I 在 G 中任意两点不相邻，且 $I-\{v\}\subseteq I$，$G[T]\subseteq G$，所以 $I-\{v\}$ 是 $G[T]$ 的独立集；

(必要性) 当 $I-\{v\}$ 是 $G[T]$ 的独立集时，因为 $I-\{v\}\subseteq V-\{v\}-N(v)$，所以 $N(v)\cap I=\varnothing$，又因为 G 比 $G[T]$ 多的边均与 v 或 $N(v)$ 中的结点相邻，所以 I 是 G 的独立集。 □

根据以上两个定理，我们可以用状态压缩的动态规划 (DP) 对于任意的无向图 $G=(V,E)$ 求出 G 的独立数 $\alpha(G)$。

对点集 $S\subseteq V$，定义 $f(S)$ 为 S 在 G 上的导出子图的独立数，即 $f(S)=\alpha(G[S])$，显然 $f(\varnothing)=0$。

考虑 $S\neq\varnothing$ 的情况。任取以 $v\in S$，考虑一个点集 $I\subseteq S$。若 $v\notin I$，则 I 是 $G[S]$ 的独立集当且仅当 I 是 $G[S-\{v\}]$ 的独立集：若 $v\in I$，则 I 是 $G[S]$ 的独立集当且仅当 $I-\{v\}$ 是 $G[S-\{v\}-N(v)]$ 的独立集。由此可得：

$$f(S)=\begin{cases}0, & S=\varnothing\\ \max\{f(S-\{v\}),f(S-\{v\}-N(v))+1\},\forall v\in S, & S\neq\varnothing\end{cases}\tag{1}$$

实现时，将图中结点编号为 $0,1,\ldots,n-1$，结点 v 可以选取 S 中编号最大的点。同样可以使用压位技巧来存集合 S。另外，计算 f 可以使用记忆化搜索，状态可以用 Hash Table 来存储。

该算法不仅能求出独立数，还能求出一个最大独立集，见以下伪代码（f 函数为依照 (1) 式定义的记忆化搜索函数）：

如果直接实现，复杂度为 $O(2^{\frac{n}{2}})$（设 Hash Table 的的单次操作时间为 $O(1)$，因为在前 $\frac{n}{2}$ 层递归中，每层递归最多 2 个分支，而递归超过 $\frac{n}{2}$ 层之后，S 中只包含编号前 $\frac{n}{2}$ 的结点，从而总复杂度为 $O(2^{\frac{n}{2}})$。

对于任意的 n，都能构造出使得该算法复杂度为 $\Theta(2^{\frac{n}{2}})$ 的图 G，方法是：将结点 0 和 $\left\lfloor\frac{n}{2}\right\rfloor$ 连边，结点 1 和 $\left\lfloor\frac{n}{2}\right\rfloor+1$ 连边，$\cdots\cdots$，结点 $\left\lfloor\frac{n}{2}\right\rfloor-1$ 和 $2\left\lfloor\frac{n}{2}\right\rfloor-1$ 连边。这样递归的前 $\left\lfloor\frac{n}{2}\right\rfloor$ 层每层都有 2 个分支，且所有的分支都不同。

例 3.3 (团的计数)**.** 给定无向简单图 $G=(V,E)$，求 G 有多少个团。一个团定义为一个点集 $S\subseteq V$，满足 S 中任意两点都有边相连。$n\leq 50$。

记 $\bar{G}$ 为 G 的补图，不难发现，S 是 G 的团当且仅当 S 是 $\bar{G}$ 的独立集。

这是因为，如果 S 是 G 的团，那么 S 中的点在 G 中两两相邻，故在 $\bar{G}$ 中两两不相邻，即 S 是 $\bar{G}$ 的独立集。反之，如果 S 不是 G 的团，即存在两点 u,v 在 G 中不相邻，则 u,v 在 $\bar{G}$ 中相邻，也就是说，S 不是 $\bar{G}$ 的独立集。

Algorithm 2 Subset-Dynamic-Programming

```
S = V
I = ∅
while S ≠ ∅ do
    令 v 为 S 中编号最大的点
    if f(S − {v}) > f(S − {v} − N(v))) + 1 then
        S ← S − {v}
    else
        S ← S − {v} − N(v)
        I ← I + {v}
    end if
end while
return I = 0
```

因此，问题转化为独立集计数问题——求 G 的独立集个数。显然这类计数问题无法用搜索优化的策略，不过，使用上述的 Subset-Dynamic- Programming 算法即可在 $O(2^{\frac{n}{2}})$ 时间内解决问题。

3.2.2 效率优化

经过实际测试，Subset-Dynamic-Programming 算法运行效率不如优化的搜索算法。为什么?考虑最大独立集问题，该算法的最坏复杂度为 $O(2^{\frac{n}{2}})$，但搜索剪枝时可以直接处理度为 1 的结点，从而只需要 $O(n)$ 的时间。是否可以类比优化搜索算法，用一些“剪枝”来优化上述 DP 呢?

答案是肯定的。不过笔者并不打算重复之前的优化——既然用了基于 DP 的算法，就应当研究更加通用的优化。例如在求最大权独立集时，直接处理度为 1 的结点是错误的。经过笔者研究，以下优化可以大大提高 DP 的效率：

1. 在状态转移方程中，结点 v 不取 S 中编号最大的点，而取 $G[S]$（注意是导出子图，而不是原图）中度数最大的点；
2. 当图 $G[S]$ 不连通时，记每个连通块的点集分别为 $S_1, S_2, \ldots, S_k$，由于每个连通块是独立的，可以转化为规模更小的子问题解决：
$$f(S) = \sum_{i=1}^{k} f(S_i)$$
3. 当图 $G[S]$ 不含环时，可以改用树形 DP 求解。

优化后的 DP 算法记作 Optimized-Subset-Dynamic-Programming。笔者无法分析该算法在最坏情况下的复杂度是多少，但通过对随机图的测试，Optimized-Subset-Dynamic-Programming 比之前的 Subset-Dynamic-Prograuming 快得多，其中优化 1、2 效果明显，尤其对于较稀疏的图。这是因为较稀疏的图在不断删去度数大的点时，导出子图 $G[S]$ 很容易不连通。

例 3.4 (小 Q 运动季测试点 10)**.** 题意见例题3.2

上文已经提到了用 Maximal-Search 求得该问题的最优解所需的时间。现在我们尝试使用基

于动态规划的独立集算法 Subset-Dynamic-Programming，遗憾的是，测试表明，这个算法运行效率并不高，并且由于状态数过多，空间消耗都无法接受。

我们再试一试 Optimized-Subset-Dynamic-Programming。令人惊讶的是，这个算法的运行效率非常高——经过笔者测试，该算法只用了 1s 就得到了最优解 (大小为 34 的独立集)！并且这个算法运行过程中没有使用依赖任何问题的特殊性的优化（如最优性剪枝）。可见在稀疏图上，Optimized-Subset-Dynamic-Programming 的确是一个优秀的算法。

3.2.3 与搜索算法的联系

事实上，如果把这个算法的记忆化去掉（即不用 Hash Table 存储 f 值），就是一个带优化的搜索算法。这样的搜索算法（记为 Optimized-Search）仍然比朴素的搜索算法 Simple- Search 快，但慢于 Optimized-Subset-Dynamic-Programming。

Optimized-Subset-Dynamic-Programming 结合了搜素和动态规划的优化思想，不仅有比较强的通用性，实现难度也很小。

然而 Optimized-Subset-Dynamic-Programming 有一定的缺陷：空间复杂度比较大。而搜索算法 Optimized-Search 的空间是多项式级别的，支持运行较长时间。因此可以只记忆化较小的 S 的 $f(S)$ 值，剩余部分采用搜索的方法，这样就能在较低的空间需求下解决问题了。

3.2.4 测试与对比

笔者在研究出上述动态规划算法及优化之后，将该算法（及优化后的算法）和之前的基于搜索的算法进行了实现，并且用随机图测试了这些算法的期望运行效率。下表的第一行中，n, m 代表 n 阶 m 边随机图，最后一列 $90, 223$ 代表 WC2013 《小 Q 运动季》的测试点 10 对应的图。表格内的时间代表算法在对应的图上的期望运行时间估计值，“-”表示运行时间过长，未测出。

算法	$40, 60$	$50, 85$	$60, 120$	$90, 223$
Simple-Search	2s	-	-	-
Maximal-Search	< 0.01s	< 0.1s	1s	-
Subset-Dynamic-Programming	< 0.01s	< 0.1s	1s	-
Optimized-Subset-Dynamic-Programming-1	< 0.01s	< 0.01s	< 0.01s	1s
Optimized-Subset-Dynamic-Programming-2	< 0.01s	< 0.01s	< 0.01s	1s

Maximal-Search 和 Subset-Dynamic-Programming 分别采用了“搜索剪枝”和“记忆化”的优化，其效果比较接近，Optimized-Subset-Dynamic-Programming 则结合了两者的优势，效率严格高于这两个算法。值得一提的是，Optimized-Subset-Dynaric-Programming-1 和 Optimized- Subset-Dynamic-Programming-2 的区别在于后者加入了优化 3（转为树形 DP），尽管优化 3 看起来很高效——把指数级的问题用线性时间解决，但经过测试，两者运行效率几乎无差别。

4 特殊图的独立集问题

上文介绍了解决一般图的独立集的基本思想（搜索优化以及动态规划），这些方法复杂度均为指数级。本节中，我们将进一步探讨特殊图的独立集问题——当图本身具有一定特殊性质时，能否用多项式复杂度解决同样的问题?

4.1 基于图匹配思想的最大独立集算法

4.1.1 二分图的最大独立集

二分图的最大独立集是一个经典问题。我们有以下定理：

定理 4.1. *对于 n 阶二分图 G，$\alpha(G) = n - v(G)$，其中 $\alpha(G)$，$v(G)$ 分别为图 G 的独立数和匹配数。*

该定理的证明可以在很多材料中找到，故证明略。

用匈牙利算法或网络流求出二分图 $G = (X, Y, E)$ 的一个匹配数以 $v(G)$，即可得到 G 的独立数 $\alpha(G)$。另外，用网络流建图后求最小割可以得到一个最大独立集 I。

这个算法只能解决最优化类的独立集问题，不能解决更加复杂的问题（妇计数或带有其它限制等）。

4.1.2 无爪图的最大独立集

二分图的最大独立集给了我们一个思路求图的最大匹配时，可以通过找增广路不断增加匹配大小，那么求其它图的最大独立集能否也采用增广的方式? 遗憾的是，在任意图上，两个独立集的对称差[6]的导出子图不一定是若干条路径或环，所以并不能用找增广路的方法求最大独立集。

不过，在一种特殊的图上，这种算法是可行的。

定义 4.1. 无爪图 (claw-fiee sraph) 定义为所有导出子图都不是 $K_{1,3}$ 的无向图。其中 $K_{1,3}$ 称为爪 (claw)，即两部分别含有 1 个点和 3 个点的完全二分图。

无爪图的最大独立集可以用类似一般图匹配的算法来求解。该算法依赖于以下定理：

定理 4.2. *设 I_1, I_2 为无爪图 G 的两个独立集，则 $G[I_1 \Delta I_2]$ 的每一个连通块都是一条简单路径或简单环。*

证明. 设 $v \in I_1$，则 $N(v) \cap I_1 = \varnothing$，在 $G[I_1 \Delta I_2]$ 中，v 的度数为 $|N(v) \cap I_2|$。

假设存在三个不同的点 $v_1, v_2, v_3 \in N(v) \cap I_2$，因为 I_2 是独立集，所以 v_1, v_2, v_3 两两不相邻，因此 $G[\{v, v_1, v_2, v_3\}]$ 是一个爪，矛盾。

因此 $|N(v) \cap I_2| \leq 2$，即 $G[I_1 \Delta I_2]$ 的所有点度数均不超过 2，定理得证。 □

[6]集合 A, B 的对称差 $A \Delta B = \{x \mid \mathbb{I}(x \in A) \neq \mathbb{I}(x \in B)\}$

注意到如果 $|I_1| < |I_2|$，那么 $G[I_1 \Delta I_2]$ 必然存在一个连通块 C，满足连通块中属于 I_2 的结点比属于 I_1 的结点多。由于 C 中属于 I_1, I_2 的结点交替出现，当 C 为简单环时，C 中属于 I_1, I_2 的结点一样多，而当 C 为简单路径时，C 中属于 I_1, I_2 的结点个数相差 1，故必然存在一条路径满足属于 I_2 的点数比属于 I_1 的点数多 1。我们把 C 称为 I_1 的增广路 (augment path)。

这样，就可以类比一般图最大匹配的算法，用增广路算法求无爪图 $G = (V, E)$ 的最大独立集：初始时令 $I = \varnothing$，每次从一个点出发找一条增广路，然后将增广路上的点状态取反，即：原来不属于独立集的点加入独立集，原来属于独立集的点从独立集中删去。该算法的实现类似 Edmonds 的带花树算法。

4.2 基于图上阶段划分思想的最大独立集算法

4.2.1 分层图上的动态规划

对于图 $G = (V, E)$，将点集 V 划分为 k 个不相交的集合 $V_1, V_2, \cdots, V_k$，使得对任意 $u \in V_i, v \in V_j$，若 $|i - j| > 1$，则 $(u, v) \notin E$，则称集合序列 $< V_1, V_2, \cdots, V_l >$ 是 G 的一个分层。如果每个 V_i 中的结点都不多，那么可以按 $V_1, V_2, \cdots, V_k$ 顺序进行决策，在每个阶段只需状压一个层的选取情况即可，效率远高于一般图中的对整个图状压 DP。

记 $f(i, S)$ 为图 $G[V_1 \cup V_2 \cup \cdots \cup V_i]$ 中包含 S 为子集的最大的独立集，状态转移方程如下：

$$f(i,S) = \begin{cases} -\infty, & S \text{ is not independent,} \\ 0, & i = 0, \\ \max\left\{ f\left(i-1, S'\right) \mid S' \subseteq V_{i-1}, S \cup S' \text{ is independent} \right\} + |S'|, & \text{otherwise} \end{cases}$$

G 的独立数为所有 $G[V_k]$ 的独立集 I 中 $f(k, I)$ 的最大值。该过程同样能求出一个 G 的最大独立集，方法和之前类似，这里不再赘述。

这个算法的时间复杂度为 $O(\sum_{i=1}^{k-1} 2^{|V_i|+|V_{i+1}|})$，不过在大多数情况下，每个 V_i 内的独立集个数并不多，实际的时间效率远高于理论上界。

4.2.2 “k-仙人图”上的动态规划

例 4.1. 给定简单无向图 $G = (V, E)$，保证每条边属于且仅属于一个简单环，求 G 的独立数。$|V| < 50,000$, $|E| \leq 60,000$。

如果 G 不连通，那么求出 G 的每个连通块的独立数并求和即可。下文假设 G 是连通图。每条边最多属于一个简单环的简单连通图称为“仙人掌”，它有什么特殊的性质呢?

我们不妨大胆尝试一下——任选一个 $r \in V$，以 r 为根对图 G 进行深度优先搜索 (depth-frst search，DFS)，得到一个深度优先搜素树（DFS 树）$T = (V, E_T)$。显然 T 是 G 的一个生成树。定义树边为属于 E_T 的边，非树边为不属于 E_T 的边（即属于 $E - E_T$ 的边）。

DFS 树有一个很重要的性质：

定理 4.3. *对任意非树边 $e = (u, v)$，在 T 中或者 u 是 v 的祖先，或者 v 是 u 的祖先。*

证明. 设 $e = (u, v)$ 为非树边，且 u 比 v 先访问到，则访问到 u 时，假如 v 不在以 u 为根的子树内，那么枚举到 u 的出边 e 时，v 未被访问，因此下一步将沿着边 e 访问到 v，从而 $e \in E_T$，与

假设矛盾，从而 u 是 v 的祖先。同理，若 v 比 u 先访问到，则 v 是 u 的祖先。 □

对于非树边 $e=(u,v)$，若树边 $e^{'}$ 在树 T 中从 u 到 v 的简单路径上，则称树边 $e^{'}$ 被非树边 e 覆盖 (cover)。对于仙人掌，我们有：

定理 4.4. *每条树边最多被一条非树边覆盖。*

证明. 假设一条树边 e 被两条非树边 (u_1,v_1)，(u_2,v_2) 覆盖，则 (u_1,v_1) 和 T 中 v_1 到 u_1 的简单路径构成一个简单环 C_1，(u_2,v_2) 和 T 中 v_2 到 u_2 的简单路径构成一个简单环 C_2，而 e 同时属于 C_1 和 C_2，与仙人掌的定义矛盾。因此 e 最多被一条非树边覆盖。 □

这个性质使得我们可以对树 T 进行树形动态规划。初步的想法是：记 $f(i,0)$ 为 T 中以结点 $i\in V$ 为根的子树内最大的独立集大小，$f(i,1)$ 为 i 的父结点属于独立集的情况下，i 子树内最大的独立集大小，然而这样不能保证非树边的两端点不同时属于独立集。

考虑添加一维状态，记 $f(i,j,k)$ 为 i 的父结点属于 $(j=1)$ 或不属于 $(j=0)$ 独立集，覆盖 i 与其父结点 e_i 连边的非树边 $e^{'}=(u_i,v_i)$ 顶端结点 i_i 属于 $(j=1)$ 或不属于 $(j=0)$ 独立集的情况下，i 子树内最大的独立集大小。当 e_i 不属于环时，$k=0$。转移时枚举 i 是否属于独立集即可，注意当 i 是 $e_i^{'}$ 的底端结点 v_i 且 $k=1$ 时，不能选 i。状态转移方程如下（Ch_i，表示 i 的子结点集合）：

- 当 $j=1$ 或 $k=1\wedge i=v_i$ 时：

$$f(i,j,k)=\sum_{c\in \mathrm{Ch}_i} f\left(c,0,\mathbb{I}(e_c'=e_i')\cdot k\right)$$

- 否则

$$f(i,j,k)=\max\left\{\sum_{c\in \mathrm{Ch}_i} f\left(c,0,\mathbb{I}\left(e_c'=e_i'\right)\cdot k\right),1+\sum_{c\in \mathrm{Ch}_i} f\left(c,1,\mathbb{I}(e_c'=e_i')\cdot k+\mathbb{I}\left(u_c=i\right)\right)\right\}$$

用 $O(m)$ 时间预处理所有 u_i,v_i 之后就可以用上述 $O(n)$ 的动态规划求出最大独立集了，时间复杂度 $O(n+m)$。

笔者在研究上述算法之后，思考这种算法能否进行扩展。经过分析，笔者发现，将该算法做一些简单的修改之后，不仅能处理仙人掌，还能处理每条边所属的简单环个数不多的图。

我们把这样的图称为“k-仙人图”，即每条边最多属于 k 个简单环的图，其中 k 的值比较小。

求解 k-仙人图 $G=(V,E)$ 的独立集问题时，同样先取一个点为根对图进行 DFS，得到 DFS 树 T。接下来对每个点 i，记 C_i 为覆盖 i 与其父结点的连边 e_i 的非树边集合，则有一个性质：

定理 4.5. *在 k-仙人图中，对于任意的 $i\in V$，$|C_i|\le k$。*

证明. 对于每个 $(u,v)\in C_i$，(u,v) 和 T 中从 u 到 v 的路径都对应了一个包含 e_i 的简单环，因此必然存在 $|C_i|$ 个包含 e_i 的简单环。

由于包含 e_i 的简单环个数不超过 k，故 $|C_i|\le k$。 □

接下来，我们利用这个性质设计动态规划。在状态中，需要记录 C_i 中所有边的顶端是否属于独立集，转移方式类似。具体地，记 $f(i,j,S)$ 为 i 的父结点属于 $(j=1)$ 或不属于 $(j=0)$ 独立集，且 C_i 内的边的顶端结点构成的集合 U_i 中属于独立集的结点集合为 S 的情况下，T 中以 i 为根的子树内最大的独立集大小，则状态转移方程如下：

- 当 $j=1$ 或 $S\cup\{i\}$ 不是独立集时：

$$f(i,j,S)=\sum_{c\in \mathrm{Ch}_i} f\left(c,0,S\cap U_i\right)$$

- 否则：

$$f(i,j,S)=\max\left\{\sum_{c\in \mathrm{Ch}_i} f\left(c,0,S\cap U_i\right),1+\sum_{c\in \mathrm{Ch}_i} f\left(c,1,(S\cup\{i\})\cap U_i\right)\right\}$$

和之前的算法类似，当 k 视为常数时，该算法的复杂度为 $O(n)$。

事实上，“k-仙人图”这个条件过于宽松，只要满足覆盖每条树边的非树边数目均不超过 k（甚至只要 $\sum_{v\in V}2^{|C_v|}$ 不大），这个算法的效率都是很高的。

该算法可以拓展到更复杂的问题，如独立集计数。

例 4.2 (子集计数问题测试点 7,8)**.** 给定无向图 $G=(V,E)$，$|V|=n$，$|E|=m$。求有多少个子集 $V'\subseteq V$ 满足 $|V'|=k$ 且 $\forall(u,v)\in E$，$u\notin V'\vee v\notin V'$。由于答案可能很大，只需输出答案对 p 取模的结果。提交答案题。

本例中仅讨论测试点 7,8。这两个测试点满足 G 是连通图数据规模如下表：

测试点编号	$n=$	$m=$	$k=$	$p=$
7	4998	5022	666	1000000009
8	11986	12011	1098	1000000007

问题求的是 G 中大小为 k 的独立集个数。由于 G 是连通图，且 $m-n$ 很小，可以发现 G 可以通过往一个树中加入 $m-n+1$ 条边得到。

如果构出 G 的 DFS 树之后，暴力枚举每条非树边的一端点是否选取，然后用树形动态规划统计独立集个数，可以较快通过测试点 7，但测试点 8 需要运行的时间太长，需要优化。根据本节中提到的思想，每条边所属的简单环的个数不多，因此可以采用上述算法解决。

记

- C_i 为满足 $e\in E'$，u_e 为 i 的祖先（不含 i，v_e 在以 i 为根的子树内的 u_e 集合
- $f(i,j,S,s)$ 为满足 C_i 中属于独立集的结点集合为 S 的前提下，i 的前 j 个子树中，大小为 s 的独立集个数
- $g(i,j,S,s)$ 为满足 C_i 中属于独立集的结点集合为 S 的前提下，i 以及 i 的前 j 个子树中，大小为 s 的独立集个数
- $F(i,S,s)=f(i,t_i,S,s)$，$G(i,S,s)=g(i,t_i,S,s)$，这里 t_i 为 i 的子节点数

对于 $f(i,j,S,s)$，设 i 和 i 的前 $j-1$ 个子树内共有 s_1 个点，第 j 个子树内有 s_2 个点，第 j 个子节点为 c_j，则

$$f(i,j,S,s)=\sum_{x=\max\{s-s_1,0\}}^{\min\{s,s_2\}} f(i,j-1,S,s-x)\cdot G\left(c_j,S\cap C_{c_j},x\right),\quad 0\le s\le s_1+s_2$$

对于 $g(i,j,S,s)$，如果存在边 $e\in E'$ 使得 $u_e\in S$ 且 $v_e=i$，那么 i 不能属于独立集，有

$$g(i,j,S,s)=f(i,j,S,s)$$

否则，存在包含 i 的独立集，这样的独立集个数记为 $g^{'}(i,j,S,s)$，则

$$g^{'}(i,j,S,s)=\sum_{x=\max\{s-s_1,0\}}^{\min\{s-1,s_2\}} g^{'}(i,j-1,S,s-x)\cdot F\left(c_j,(S\cup\{i\})\cap C_{c_j},x\right),\quad 0\le s\le s_1+s_2$$

所以

$$g(i,j,S,s)=g^{'}(i,j,S,s)+f(i,j,S,s),\quad 0\le s\le s_1+s_2$$

边界为 $f(i,0,S,0)=g^{'}(i,0,S,1)=1$，未定义的状态值均为 0。答案就是 $G(r,\varnothing,k)$。

可以只计算满足 $s\le k$ 的状态，复杂度 $O(k\sum_{v\in V}2^{|C_v|})$，其中 $|C_v|$ 通常在 12 以内。整个计算都在模 p 意义下进行即可，内存可以动态分配。

值得注意的是，选取不同的点 $r\in V$ 当根，以及用不同的顺序进行 DFS，运行效率是不同的。可以选择一个根 r 进行 DFS，使得

$$\sum_{v\in V}2^{|C_v|}$$

最小，然后再执行上述算法，以降低时间复杂度。经过实测，测试点 7 的状态数约为 5×10^5，可以在 1s 内通过该测试点；测试点 8 的状态数约为 10^8，可以在 2 min 内通过该测试点。

5 总结

NP-Hard 问题的算法优化方法数不胜数，本文仅仅提到了若干种独立集问题的优化算法，这些方法解决的问题相类似，但思想各有区别——针对普通的最优化问题（如最大独立集），可以用带最优性剪枝的搜素算法减少枚举量；针对计数或有额外约束的问题（如独立集计数），可以用状态压缩动态规划，通过优化状态数来提高运行效率；针对可“增广”的图以及具有明显阶段性的图，又可以用多项式复杂度的算法来高效完成。

同时，本文对几种算法在随机情况下的运行时间进行了分析和比较，让大家对独立集问题求解的效率有更进一步的认识。