NP 難題演算法系列 · 第 07 篇

獨立集與最大團

Independent Set / Clique — Two Sides of One Coin

「彼此都不認識的最大群體」與「彼此都認識的最大群體」——在補圖裡它們是同一個問題

Contents

1 問題是什麼

定義 1.1 (Independent Set)**.** $S \subseteq V$ 是**獨立集**若 S 內任兩點**不相鄰**。目標：最大化 $|S|$（記 $\alpha(G)$）。

定義 1.2 (Clique)**.** $K \subseteq V$ 是**團**若 K 內任兩點**都相鄰**。目標：最大化 $|K|$（記 $\omega(G)$）。

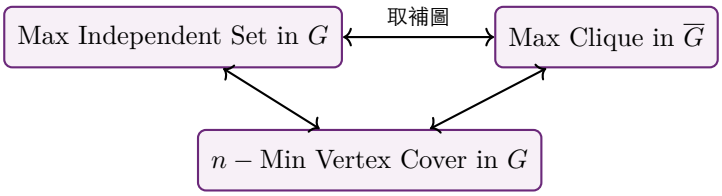
白話比喻

一場宴會的社交圖：邊 = 兩人認識。「找最多人的一桌，讓大家**互相都認識**」是 Clique；「找最多人的一桌，讓大家**互相都不認識**（逼他們社交）」是 Independent Set。同一批客人，翻轉「認識/不認識」的定義，兩題互換——這就是補圖。

1.1 三位一體

定理 1.1 (等價三角)**.** 對任意圖 G 與其補圖 $\overline{G}$（邊集取反）：

$$\alpha(G) \;=\; \omega(\overline{G}) \;=\; n - \min \mathrm{VertexCover}(G)$$



三個問題可互相歸約，NP-Complete 身分一起確立（Karp 21）。但注意微妙差異：Vertex Cover 有 2-近似與 FPT 演算法，Independent Set / Clique 卻**兩者皆無**——除非 P=NP，Max Clique 連 $n^{1-\varepsilon}$ 倍的近似都做不到（Håstad 1999）。「n 減去好近似的東西」不會自動是好近似，這是近似理論最重要的陷阱之一。

2 演算法一：最大獨立集的分支剪枝

2.1 分支規則

挑剩餘圖中**度數最大**的點 v：

- 分支 A：**不選** v ——從圖中刪 v。
- 分支 B：**選** v ——刪 v 與其全部鄰居（他們不能再選），計數 $+1$。

若剩餘點全是孤立點，全收。用 bitmask 表示「剩餘點集合」讓刪點變一個 AND 運算。最壞仍指數，但在 $n = 20$、43 條邊的隨機圖上只需 339 個節點（暴力枚舉要 $2^{20} \approx 10^6$ 個子集）。

3 演算法二：最大團的 Bron–Kerbosch

3.1 三個集合的舞蹈

維護 (R, P, X)：R = 目前團、P = 還能加入的候選（與 R 全相鄰）、X = 處理過不准再用（防止重複列舉同一個極大團）：

```
expand(R, P, X):
    if P and X both empty:  report R as maximal clique
    choose pivot u in P|X maximizing |N(u) & P|
    for each v in P \ N(u):                 // 只展開 pivot 蓋不住的
        expand(R+{v}, P & N(v), X & N(v))
        move v from P to X
```

Pivot 的直覺：若 $v \in N(u)$，那麼含 v 的極大團也會在展開 u 的分支裡被找到，跳過它不漏解卻大砍分支。理論上限 $O(3^{n/3})$——恰等於一張圖最多可能有的極大團數（Moon–Moser 圖），所以這是**輸出最優**的枚舉法。

3.2 手算範例：C_5

環 $0-1-2-3-4-0$：任三點必有兩點不相鄰，故 $\omega = 2$（任一條邊）。獨立集 $\alpha = 2$（如 $\{0,2\}$）；$\{0,2,4\}$ 不行因為 $4-0$ 是邊。驗證三位一體：$\min VC = 5 - 2 = 3$（如 $\{1,3,4\}$）。C_5 是最小的「$\alpha + \omega$ 都嚴格小於界」的自補圖，也是完美圖理論的著名反例。

4 完整 C++ 程式

包含：bitmask 暴力枚舉（$n \le 20$ 基準）、度數分支剪枝、Bron–Kerbosch with pivot、補圖轉換、三位一體驗證與社群偵測情境。編譯：

```
g++ -std=c++17 -O2 -Wall -Wextra -o independent_set_clique
    independent_set_clique.cpp
```

```cpp
// independent_set_clique.cpp
// ------------------------------------------------------------------
// Independent Set 與 Clique：一體兩面的 NP-Complete 問題
//   1. 最大獨立集：位元枚舉（n<=20 教學版）+ 分支剪枝
//   2. 最大團：Bron－Kerbosch（含 pivot）
//   3. 三位一體驗證：IS(G) = Clique(補圖 G') = n - VC(G)
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o independent_set_clique
    independent_set_clique.cpp
// ------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;

// 鄰接以 bitmask 儲存：adj[v] 的第 u 位 = 1 表示 u,v 相鄰（n <= 30）
struct Graph {
    int n;
    vector<unsigned> adj;
    Graph(int n_) : n(n_), adj(n_, 0) {}
    void addEdge(int u, int v) {
        adj[u] |= 1u << v;
        adj[v] |= 1u << u;
    }
    Graph complement() const {
        Graph g(n);
        unsigned full = (n == 32) ? ~0u : ((1u << n) - 1);
        for (int v = 0; v < n; ++v)
            g.adj[v] = full & ~adj[v] & ~(1u << v);   // 去掉自環
        return g;
    }
};

// ---------- 1a. 最大獨立集：枚舉所有子集（O(2^n)，n 小用） ----------
pair<int, unsigned> maxISBrute(const Graph& g) {
    int best = 0;
    unsigned bestSet = 0;
    for (unsigned s = 0; s < (1u << g.n); ++s) {
        bool ok = true;
        for (int v = 0; v < g.n && ok; ++v)
            if ((s >> v) & 1)
                if (g.adj[v] & s) ok = false;          // v
                    的鄰居也在集合裡
        if (ok && __builtin_popcount(s) > best) {
            best = __builtin_popcount(s);
            bestSet = s;
        }
    }
    return {best, bestSet};
}
```

```cpp

// ---------- 1b. 最大獨立集：分支剪枝 ----------
// 分支規則：挑一個度數最大的點 v -> 不選 v（在剩餘圖刪 v）或選 v（刪 v 和鄰居）
class MaxISBranch {
    const Graph& g;
public:
    long long nodes = 0;
    MaxISBranch(const Graph& g_) : g(g_) {}

    pair<int, unsigned> solve() {
        nodes = 0;
        unsigned full = (g.n == 32) ? ~0u : ((1u << g.n) - 1);
        return dfs(full);
    }
private:
    pair<int, unsigned> dfs(unsigned remain) {
        ++nodes;
        if (!remain) return {0, 0};
        // 找 remain 中度數最大的點
        int best = -1, bestDeg = -1;
        for (int v = 0; v < g.n; ++v)
            if ((remain >> v) & 1) {
                int d = __builtin_popcount(g.adj[v] & remain);
                if (d > bestDeg) { bestDeg = d; best = v; }
            }
        if (bestDeg == 0) {                              // 剩下的全是孤立點
            return {__builtin_popcount(remain), remain};
        }
        // 分支 1：不選 best
        auto [c1, s1] = dfs(remain & ~(1u << best));
        // 分支 2：選 best（刪掉 best 和其鄰居）
        auto [c2, s2] = dfs(remain & ~(1u << best) & ~g.adj[best]);
        ++c2;
        s2 |= 1u << best;
        return c1 >= c2 ? make_pair(c1, s1) : make_pair(c2, s2);
    }
};

// ---------- 2. 最大團：Bron–Kerbosch with pivot ----------
class BronKerbosch {
    const Graph& g;
public:
    int best = 0;
    unsigned bestClique = 0;
    long long calls = 0;
    BronKerbosch(const Graph& g_) : g(g_) {}

    void run() {
        best = 0; bestClique = 0; calls = 0;
        unsigned full = (g.n == 32) ? ~0u : ((1u << g.n) - 1);
        expand(0, full, 0);
    }
private:
    // R = 目前的團，P = 還能加入的候選，X = 已處理過（避免重複）
    void expand(unsigned R, unsigned P, unsigned X) {
        ++calls;
        if (!P && !X) {                                  // R 是極大團
            if (__builtin_popcount(R) > best) {
```

```
                best = __builtin_popcount(R);
                bestClique = R;
            }
            return;
        }
        // pivot：從 P|X 挑鄰居最多的 u，只展開 P \ N(u)
        unsigned PX = P | X;
        int pivot = -1, bestCover = -1;
        for (int v = 0; v < g.n; ++v)
            if ((PX >> v) & 1) {
                int c = __builtin_popcount(g.adj[v] & P);
                if (c > bestCover) { bestCover = c; pivot = v; }
            }
        unsigned cand = P & ~g.adj[pivot];
        for (int v = 0; v < g.n; ++v)
            if ((cand >> v) & 1) {
                expand(R | (1u << v), P & g.adj[v], X & g.adj[v]);
                P &= ~(1u << v);
                X |= 1u << v;
            }
    }
};

static void printSet(unsigned s, int n) {
    cout << "{";
    bool first = true;
    for (int v = 0; v < n; ++v)
        if ((s >> v) & 1) {
            cout << (first ? "" : ",") << v;
            first = false;
        }
    cout << "}";
}

int main() {
    // ---- 範例 1：C5 環（IS=2, Clique=2） ----
    cout << "=== Example 1: cycle C5 ===\n";
    Graph c5(5);
    for (int i = 0; i < 5; ++i) c5.addEdge(i, (i + 1) % 5);
    auto [is1, s1] = maxISBrute(c5);
    cout << "max independent set = " << is1 << "  ";
    printSet(s1, 5);
    BronKerbosch bk1(c5);
    bk1.run();
    cout << "\nmax clique = " << bk1.best << "  ";
    printSet(bk1.bestClique, 5);
    cout << "\n\n";

    // ---- 範例 2：三位一體 IS(G) = Clique(G') = n - VC(G) ----
    cout << "=== Example 2: trinity on Petersen graph ===\n";
    Graph pet(10);
    int petE[][2] = {{0,1},{1,2},{2,3},{3,4},{4,0},
                     {5,7},{7,9},{9,6},{6,8},{8,5},
                     {0,5},{1,6},{2,7},{3,8},{4,9}};
    for (auto& e : petE) pet.addEdge(e[0], e[1]);
    auto [isP, setP] = maxISBrute(pet);
    Graph petComp = pet.complement();        //
       注意：要用具名變數，避免懸空參考
    BronKerbosch bkC(petComp);
```

```
    bkC.run();
    cout << "max IS in G            = " << isP << "  ";
    printSet(setP, 10);
    cout << "\nmax clique in G'       = " << bkC.best << "  ";
    printSet(bkC.bestClique, 10);
    cout << "\nn - maxIS = 10 - " << isP << " = " << 10 - isP
         << "  (= minimum vertex cover size)\n\n";

    // ---- 範例 3：分支剪枝 vs 暴力枚舉 ----
    cout << "=== Example 3: branch-and-prune vs brute force (n=20) ===\n";
    Graph g20(20);
    // 決定性的偽隨機圖
    unsigned seed = 123456789;
    auto nextRand = [&seed]() {
        seed = seed * 1103515245 + 12345;
        return (seed >> 16) & 0x7fff;
    };
    int edges = 0;
    for (int u = 0; u < 20; ++u)
        for (int v = u + 1; v < 20; ++v)
            if (nextRand() % 100 < 25) {                // 邊密度 25%
                g20.addEdge(u, v);
                ++edges;
            }
    auto [isB, setB] = maxISBrute(g20);
    MaxISBranch mb(g20);
    auto [isBr, setBr] = mb.solve();
    cout << "n=20, |E|=" << edges << "\n";
    cout << "brute force  : maxIS=" << isB << " (2^20 = 1,048,576
        subsets)\n";
    cout << "branch+prune : maxIS=" << isBr << " (nodes=" << mb.nodes <<
        ")\n";
    cout << "agree = " << (isB == isBr ? "yes" : "NO!") << "\n\n";
    (void)setB; (void)setBr;

    // ---- 範例 4：社群偵測情境——找互相認識的最大群體 ----
    cout << "=== Example 4: find the largest group of mutual friends
        ===\n";
    // 8 人，邊 = 互相認識
    Graph soc(8);
    int socE[][2] = {{0,1},{0,2},{1,2},{2,3},{0,3},{1,3},   // 0,1,2,3
        全連 = K4
                     {3,4},{4,5},{5,6},{6,7},{4,6}};
    for (auto& e : socE) soc.addEdge(e[0], e[1]);
    BronKerbosch bkS(soc);
    bkS.run();
    cout << "max clique size = " << bkS.best << "  members ";
    printSet(bkS.bestClique, 8);
    cout << "  (BK calls=" << bkS.calls << ")\n";
    return 0;
}
```

程式碼 1: independent_set_clique.cpp — 獨立集與最大團完整實作

5 執行結果與解讀

```
=== Example 1: cycle C5 ===
max independent set = 2  {0,2}
```

```
max clique = 2  {0,1}

=== Example 2: trinity on Petersen graph ===
max IS in G            = 4  {2,4,5,6}
max clique in G'       = 4  {0,2,8,9}
n - maxIS = 10 - 4 = 6  (= minimum vertex cover size)

=== Example 3: branch-and-prune vs brute force (n=20) ===
n=20, |E|=43
brute force  : maxIS=8 (2^20 = 1,048,576 subsets)
branch+prune : maxIS=8 (nodes=339)
agree = yes

=== Example 4: find the largest group of mutual friends ===
max clique size = 4  members {0,1,2,3}  (BK calls=12)
```

筆記

Example 2 用 Petersen 圖同時驗證三位一體：G 的最大獨立集 4、補圖的最大團 4、最小頂點覆蓋 $10-4=6$（與第 6 篇的精確解一致）。注意兩個「4 人組」的成員不同——最優解不唯一，但**大小**必然一致。

常見誤解

工程陷阱（本程式開發時真實踩過）：`BronKerbosch bk(g.complement());` 這行會讓 `bk` 持有**懸空參考**——`complement()` 回傳的暫時物件在分號處就死了，之後的行為是未定義（實測症狀：最大團永遠是 1）。先存進具名變數再傳入。以 `const &` 存放建構參數的類別都有此雷。

6 實務應用

- **社群偵測**：社交網路中的緊密小圈子 = 團；LinkedIn 的「你們可能是同事」背後是稠密子圖挖掘。
- **生物資訊**：蛋白質交互作用網的功能模組、基因共表達群 = 團搜尋；分子對接的相容性圖找最大團是製藥業日常。
- **排程衝突**：任務相容圖的最大獨立集 = 可同時執行的最大任務集。
- **無線通訊**：干擾圖的最大獨立集 = 可同時發送的最大裝置集合。
- **組合拍賣**：標的衝突圖的最大加權獨立集 = 收益最大的得標組合。

7 練習題

1. 手算 Petersen 圖某個大小 4 的獨立集，並驗證其補集是頂點覆蓋。
2. 樹上的最大獨立集有 $O(n)$ 的 DP（每點選/不選）——實作並與本篇程式在隨機樹上對答案。
3. 把 MaxISBranch 加上上界剪枝：剩餘點數 + 已選數 $\le$ 已知最佳就砍。實測節點數。
4. 修改 Bron–Kerbosch 列出**全部**極大團（不只最大），數 Petersen 圖有幾個。
5. 挑戰：加權最大獨立集的分支剪枝（點有權重，最大化總權重）。

8 小結

方法	時間	找什麼	適用時機
bitmask 枚舉	$O(2^n n)$	IS/Clique 精確	$n \le 22$、驗證用
度數分支剪枝	指數、實務小	IS 精確	$n \le 60$ 稀疏圖
Bron–Kerbosch+pivot	$O(3^{n/3})$	所有極大團	中型圖、要列舉
樹/區間圖 DP	$O(n)$	IS 精確	圖有特殊結構

延伸閱讀：Bron & Kerbosch (1973) 原始論文；Tomita et al. (2006) pivot 版最壞情況分析；Håstad (1999) Clique 不可近似性；Robson (1986) $O(1.2^n)$ 最大獨立集。