NP 難題演算法系列 · 第 06 篇

頂點覆蓋問題

Vertex Cover — Exact FPT Branching & 2-Approximation

用最少的守衛看住所有走廊：FPT 分支與「保證不超過兩倍」的近似法的最佳教室

Contents

1 問題是什麼

白話比喻

博物館的走廊構成一張圖，守衛只能站在**交叉口**（點），站定後能看住所有**通到這個交叉口的走廊**（鄰接邊）。最少派幾個守衛，讓每條走廊都有人看？

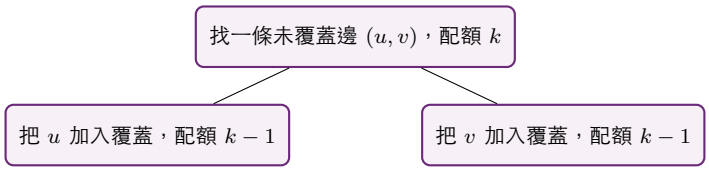
定義 1.1 (Vertex Cover)**.** 圖 $G = (V, E)$ 的**頂點覆蓋**是集合 $C \subseteq V$ 使每條邊至少一端在 C 中。目標：最小化 $|C|$。判定版（$|C| \le k$？）是 NP-Complete（Karp 21 之一，由 3-SAT 經 Clique 歸約）。

性質 1.1 (三位一體)**.** C 是覆蓋 $\iff$ $V \setminus C$ 是獨立集。因此 $\min VC = n - \max IS$。（詳見本系列第 7 篇。）

2 演算法一：精確解——FPT 分支限界

2.1 一條邊的二選一

任何一條未覆蓋的邊 (u, v)，答案**必定**含 u 或含 v（否則這條邊沒人管）。於是：



每分支配額減 1，樹深最多 k，所以：

$$T = O\big(2^k \cdot \mathrm{poly}(n)\big)$$

2.2 FPT：把指數關在參數裡

這叫**固定參數可解（Fixed-Parameter Tractable）**：指數只依賴**參數** k（覆蓋大小），不依賴 n。$n = 10^5$、$k = 20$ 完全可行——這正是「NP-Hard 不等於絕望」的第三條出路（前兩條是近似與特例）。本篇程式 Example 4 用 $n = 100$ 的圖示範：搜尋深度由 k 決定，跟 n 無關。

求**最小**覆蓋：k 從 0 遞增，第一個成功的 k 即答案。

實務技巧

更快的 FPT 技巧（進階）：度數 $\geq k+1$ 的點必在覆蓋中（否則它的 $k+1$ 條邊要 $k+1$ 個不同鄰居來蓋，超額）——先做這種**核化（kernelization）**可把圖縮到 $O(k^2)$ 條邊再分支。現代紀錄約 $O(1.2^k)$。

3 演算法二：2-近似——極大匹配

3.1 演算法（簡單得不像話）

1. 掃描每條邊 (u, v)：若兩端都還沒被選，**把 u、v 一起加入覆蓋**。
2. 結束。所選的邊構成一組**極大匹配** M，覆蓋大小 $= 2|M|$。

定理 3.1 (2-近似保證)**.** $|C_{\mathrm{alg}}| \leq 2 \cdot OPT$。

Proof. 匹配 M 中的邊兩兩不共點，任何覆蓋（包括最優的）都必須**每條匹配邊至少選一端**，故 $OPT \geq |M|$。而 $|C_{\mathrm{alg}}| = 2|M| \leq 2 \cdot OPT$。 □

常見誤解

反直覺警告：「每次貪心挑**度數最大**的點」看起來更聰明，卻**沒有常數近似保證**（最壞 $\Theta(\log n)$ 倍）；而「笨笨地兩端都拿」反而有 2 倍保證。近似演算法的世界裡，**可證明的笨**勝過**無保證的聰明**。順帶一提：假設 Unique Games Conjecture 成立，2 倍就是多項式時間能做到的極限——這個笨方法竟是最優近似。

3.2 手算範例：路徑 P_6

邊：$0-1,\ 1-2,\ 2-3,\ 3-4,\ 4-5$。

- 近似法：取 $(0,1) \Rightarrow$ 選 $\{0,1\}$；$(1,2)$ 已蓋；取 $(2,3) \Rightarrow$ 加 $\{2,3\}$；取 $(4,5) \Rightarrow$ 加 $\{4,5\}$。共 **6** 個點。
- 最優解：兩個點不夠——$\{1,3\}$ 蓋不住 $4-5$、$\{1,4\}$ 蓋不住 $2-3$，任兩點都會漏邊。三個點可行，例如 $\{0,2,4\}$ 或 $\{1,3,4\}$，故 $OPT = \mathbf{3}$。
- 比率恰為 $6/3 = 2$——2-近似的上界在路徑圖上**被打滿**。

4 完整 C++ 程式

包含：k-覆蓋分支限界（FPT）、遞增求最小覆蓋、極大匹配 2-近似、最大度貪心對照組、覆蓋驗證器。編譯：

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

```cpp
// vertex_cover.cpp
// ------------------------------------------------------------------
// Vertex Cover（頂點覆蓋）：
//   1. 精確解：k-分支限界（FPT，O(2^k · poly)）+ 求最小覆蓋
//   2. 2-近似：極大匹配的兩端點（保證 <= 2·OPT）
//   3. 度數啟發式（貪婪挑最大度）——無保證的對照組
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o vertex_cover
    vertex_cover.cpp
// ------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <algorithm>
#include <set>

using namespace std;

using Edges = vector<pair<int,int>>;

// ---------- 1. 精確：分支限界 ----------
// 觀察：任一條未覆蓋邊 (u,v)，答案必含 u 或 v -> 兩分支，深度 <= k
class VertexCoverExact {
public:
    long long nodes = 0;

    // 是否存在大小 <= k 的覆蓋；cover 傳回實際選點
    bool solveK(const Edges& edges, int n, int k, vector<int>& cover) {
        nodes = 0;
        vector<bool> chosen(n, false);
        bool ok = dfs(edges, chosen, k);
        cover.clear();
        for (int i = 0; i < n; ++i)
            if (chosen[i]) cover.push_back(i);
        return ok;
    }

    // 最小覆蓋：k 從 0 遞增
    vector<int> minimumCover(const Edges& edges, int n) {
```

```
        vector<int> cover;
        for (int k = 0; k <= n; ++k)
            if (solveK(edges, n, k, cover)) return cover;
        return cover;
    }
private:
    bool dfs(const Edges& edges, vector<bool>& chosen, int k) {
        ++nodes;
        // 找第一條未覆蓋的邊
        const pair<int,int>* uncovered = nullptr;
        for (const auto& e : edges)
            if (!chosen[e.first] && !chosen[e.second]) { uncovered = &e;
                break; }
        if (!uncovered) return true;                        // 全覆蓋
        if (k == 0) return false;                           // 配額用盡
        auto [u, v] = *uncovered;
        chosen[u] = true;                                   // 分支 1：選 u
        if (dfs(edges, chosen, k - 1)) return true;
        chosen[u] = false;
        chosen[v] = true;                                   // 分支 2：選 v
        if (dfs(edges, chosen, k - 1)) return true;
        chosen[v] = false;
        return false;
    }
};

// ---------- 2. 2-近似：極大匹配 ----------
vector<int> vertexCover2Approx(const Edges& edges, int n) {
    vector<bool> in(n, false);
    vector<int> cover;
    for (auto [u, v] : edges) {
        if (in[u] || in[v]) continue;
        in[u] = in[v] = true;                               // 兩端都拿
        cover.push_back(u);
        cover.push_back(v);
    }
    return cover;
}

// ---------- 3. 貪婪最大度（教學對照：無近似保證！） ----------
vector<int> vertexCoverGreedyDegree(Edges edges, int n) {
    vector<int> cover;
    while (true) {
        vector<int> deg(n, 0);
        for (auto [u, v] : edges) { ++deg[u]; ++deg[v]; }
        int best = max_element(deg.begin(), deg.end()) - deg.begin();
        if (deg[best] == 0) break;
        cover.push_back(best);
        Edges rest;
        for (auto& e : edges)
            if (e.first != best && e.second != best) rest.push_back(e);
        edges = std::move(rest);
    }
    return cover;
}

static bool verifyCover(const Edges& edges, const vector<int>& cover) {
    set<int> s(cover.begin(), cover.end());
    for (auto [u, v] : edges)
```

```
        if (!s.count(u) && !s.count(v)) return false;
    return true;
}

static void show(const string& tag, const vector<int>& cover, const
    Edges& e) {
    cout << tag << " size=" << cover.size() << "  {";
    for (size_t i = 0; i < cover.size(); ++i)
        cout << cover[i] << (i + 1 < cover.size() ? "," : "");
    cout << "}  valid=" << (verifyCover(e, cover) ? "yes" : "NO!") <<
       "\n";
}

int main() {
    // ---- 範例 1：星星圖（貪婪==最優，近似=2x） ----
    cout << "=== Example 1: star graph K1,5 ===\n";
    Edges star = {{0,1},{0,2},{0,3},{0,4},{0,5}};
    VertexCoverExact ex;
    show("exact  ", ex.minimumCover(star, 6), star);
    show("2-apx  ", vertexCover2Approx(star, 6), star);
    show("greedy ", vertexCoverGreedyDegree(star, 6), star);
    cout << "\n";

    // ---- 範例 2：路徑 P6（近似的表現） ----
    cout << "=== Example 2: path P6 ===\n";
    Edges path = {{0,1},{1,2},{2,3},{3,4},{4,5}};
    show("exact  ", ex.minimumCover(path, 6), path);
    show("2-apx  ", vertexCover2Approx(path, 6), path);
    cout << "\n";

    // ---- 範例 3：Petersen 圖（OPT=6） ----
    cout << "=== Example 3: Petersen graph ===\n";
    Edges pet = {
        {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}};
    auto opt = ex.minimumCover(pet, 10);
    show("exact  ", opt, pet);
    cout << "  (branching nodes = " << ex.nodes << ")\n";
    show("2-apx  ", vertexCover2Approx(pet, 10), pet);
    show("greedy ", vertexCoverGreedyDegree(pet, 10), pet);
    cout << "\n";

    // ---- 範例 4：FPT 的意義——n 大但 k 小 ----
    cout << "=== Example 4: FPT — big n, small k ===\n";
    // 100 個頂點的稀疏圖：一條長鏈 + 幾條額外邊
    Edges big;
    for (int i = 0; i < 20; ++i) big.push_back({i, i + 1});
    big.push_back({30, 40});
    big.push_back({50, 60});
    big.push_back({70, 80});
    vector<int> cov;
    for (int k = 1; k <= 14; ++k) {
        if (ex.solveK(big, 100, k, cov)) {
            cout << "n=100, |E|=" << big.size()
                 << " -> minimum cover k=" << k
                 << " (nodes=" << ex.nodes << ")\n";
            break;
        }
```

```
    }
    cout << "Search depth bounded by k, not n -> FPT: O(2^k * poly(n))\n";
    return 0;
}
```

程式碼 1: vertex_cover.cpp — 頂點覆蓋精確解與 2-近似完整實作

5 執行結果與解讀

```
=== Example 1: star graph K1,5 ===
exact   size=1  {0}              valid=yes
2-apx   size=2  {0,1}            valid=yes
greedy  size=1  {0}              valid=yes

=== Example 2: path P6 ===
exact   size=3  {0,2,4}          valid=yes
2-apx   size=6  {0,1,2,3,4,5}    valid=yes

=== Example 3: Petersen graph ===
exact   size=6  {0,1,3,7,8,9}    valid=yes   (branching nodes = 30)
2-apx   size=8  {0,1,2,3,5,7,9,6} valid=yes
greedy  size=6  {0,2,6,3,5,9}    valid=yes

=== Example 4: FPT --- big n, small k ===
n=100, |E|=23 -> minimum cover k=13 (nodes=16372)
Search depth bounded by k, not n -> FPT: O(2^k * poly(n))
```

筆記

三組對照很有戲：(1) 星星圖上近似法拿 2 個點（最優 1 個）——比率 2 又一次打滿；(2) P_6 上近似法拿了**全部** 6 個點，正好是最優 3 的兩倍——但**永遠不會更糟**；(3) 最大度貪心在這三個例子上都表現漂亮，但別忘了它**沒有保證**，存在讓它偏差 $\log n$ 倍的構造。

6 實務應用

- **網路監控**：在最少的路由器上裝監測器，看住所有鏈路。
- **生物資訊**：蛋白質交互作用網路中挑最小的「關鍵蛋白」集合覆蓋所有交互作用；FPT 演算法在此領域實際落地（k 通常很小）。
- **程式分析**：測試用例選擇——用最少的測試覆蓋所有程式路徑對。
- **衝突消解**：資料庫死鎖圖中最少「犧牲」多少交易能打破所有衝突邊。

7 練習題

1. 手算完全二部圖 $K_{3,4}$ 的最小覆蓋（König 定理：二部圖 $\min VC = \max$ 匹配）。
2. 實作核化規則「度 $\geq k+1$ 的點必選」，實測 Example 4 的節點數變化。
3. 構造一個讓最大度貪心輸出 $\Theta(\log n)$ 倍差解的圖（提示：一側是度數階梯的二部圖）。
4. 把 2-近似的邊掃描順序隨機化，在隨機圖上統計平均比率——實務上通常遠好於 2。
5. 挑戰：加權頂點覆蓋的 2-近似（提示：LP 鬆弛 + 取 $x_v \geq 1/2$ 的點，或 primal-dual）。

8 小結

方法	時間	品質	適用時機
FPT 分支	$O(2^k \operatorname{poly}(n))$	精確	$k \lesssim 30$，n 可以很大
匹配 2-近似	$O(E)$	$\le 2 \cdot OPT$	要保證、要快
最大度貪心	$O(VE)$	無保證	只當經驗法、需驗證
König（二部圖特例）	$O(E\sqrt{V})$	精確	圖是二部圖

延伸閱讀：Cygan et al. *Parameterized Algorithms*（FPT 聖經）；Khot & Regev (2008) 2-近似最優性（UGC 條件下）；König 定理與匹配理論。