NP 難題演算法系列 · 第 01 篇

3-SAT 布林可滿足性

Boolean Satisfiability — Backtracking & DPLL

史上第一個被證明 NP-Complete 的問題：給一堆「三選一」的條件，找出讓全部條件成立的真假指派

Contents

1 問題是什麼

白話比喻

想像你在排一場婚宴座位。每位親戚都提出條件：「A 桌要有我、或別讓二叔坐 A 桌、或把我排到 C 桌」——每個條件都是**三個要求中至少滿足一個**。你要找出一種安排讓**所有**條件同時成立。這就是 3-SAT：每個子句給你三個機會，你只要讓每個子句至少中一個。

定義 1.1 (CNF 與 3-SAT)**.** 布林變數 $x_1, \ldots, x_n$ 的**文字（literal）**是 x_i 或其否定 $\neg x_i$。**子句（clause）**是若干文字的 OR，例如 $(x_1 \vee \neg x_2 \vee x_3)$。**CNF 公式**是若干子句的 AND。**3-SAT** 問：每個子句恰有 3 個文字的 CNF 公式，是否存在一組真假指派使公式為真？

範例 1.1. 公式 $\varphi = (x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3)$。取 $x_1 = $ 真, $x_2 = $ 假, $x_3 = $ 假：四個子句依序由 x_1、$\neg x_2$、x_1、$\neg x_3$ 滿足，全部成立，故 φ 可滿足。

1.1 為什麼 3-SAT 是「難題之王」

- **Cook–Levin 定理（1971）**：SAT 是第一個被證明的 NP-Complete 問題；3-SAT 是它的標準化版本。任何 NP 問題都能在多項式時間內「翻譯」成 3-SAT。
- 之後 Karp 用歸約把 3-SAT 的難度傳染給 21 個問題（Clique、Vertex Cover、Hamiltonian ……），本系列後面的每一篇，難度的源頭都是 3-SAT。
- **2 與 3 的分水嶺**：2-SAT（每子句兩個文字）可用強連通分量在多項式時間解決；加到 3 個文字就 NP-Complete。難度的跳變常常就發生在一個參數 $+1$。

常見誤解

「NP-Complete」不代表「每個實例都算不動」。實務 SAT 求解器天天在解上百萬變數的工業實例——難的是**最壞情況**，不是**每種情況**。

2 演算法一：暴力枚舉

n 個變數共有 2^n 組指派，逐一檢查每組是否滿足全部 m 個子句：

$$T(n, m) = O(2^n \cdot m \cdot 3)$$

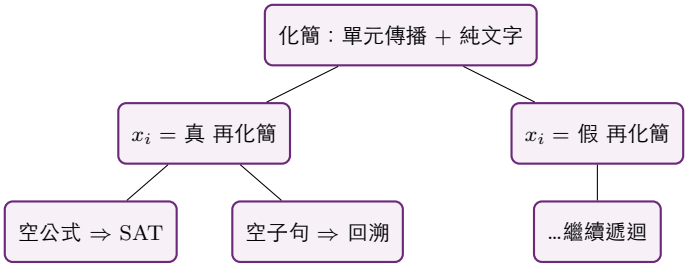
$n = 20$ 時約 10^6 組還行；$n = 50$ 時 10^{15} 組就永遠跑不完。暴力法的價值是**當標準答案**：用它驗證聰明演算法沒寫錯（本篇程式的 Example 3 正是這麼做）。

3 演算法二：DPLL

DPLL（Davis–Putnam–Logemann–Loveland, 1962）是所有現代 SAT 求解器的骨架。它仍是回溯搜尋，但加了三個關鍵推理規則，讓搜尋樹大幅縮小。

3.1 三個核心規則

1. **單元傳播（Unit Propagation）**：若某子句只剩一個未定文字，該文字**被迫**為真。例如 (x_3) 只剩 x_3，那 x_3 必須是真——沒有分支的必要。連鎖反應常常一路推倒一整排骨牌。
2. **純文字消去（Pure Literal Elimination）**：若變數 x_i 在剩餘公式中**只以一種極性**出現（全是 x_i 或全是 $\neg x_i$），直接把它設為讓那些子句滿足的值，穩賺不賠。
3. **分支（Splitting）**：推不動了才挑一個變數 x_i，先試 $x_i = $ 真，失敗再試 $x_i = $ 假。挑「出現次數最多」的變數是簡單有效的啟發式。



3.2 終止條件

- 公式化簡後**沒有子句了** ⇒ 全部滿足，回報 SAT（目前的指派就是解）。
- 出現**空子句**（子句的文字全為假）⇒ 這條路矛盾，回溯。

最壞情況仍是 $O(2^n)$——3-SAT 是 NP-Complete，這無法避免——但實務上單元傳播會砍掉絕大多數分支。本篇程式在 $n = 20$、$m = 80$ 的隨機實例上只拜訪了 8 個節點，而暴力法要看 10^6 組。

3.3 手算走一遍

用 $\varphi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3) \wedge (x_2)$（示範用 2–3 文字混合）：

1. 子句 (x_2) 是單元子句 ⇒ 強制 $x_2 =$ 真。
2. 化簡：$(x_1 \vee x_2)$ 已滿足消失；$(\neg x_2 \vee \neg x_3)$ 剩 $(\neg x_3)$——又是單元子句 ⇒ $x_3 =$ 假。
3. 化簡：$(\neg x_1 \vee x_3)$ 剩 $(\neg x_1)$ ⇒ $x_1 =$ 假。
4. 公式清空 ⇒ SAT，解為 $x_1 =$ 假、$x_2 =$ 真、$x_3 =$ 假。**全程零分支**。

4 相變現象：隨機 3-SAT 什麼時候最難

隨機生成 n 個變數、m 個子句的 3-SAT，實驗發現以 $m/n \approx 4.27$ 為界：

- $m/n \ll 4.27$：約束少，幾乎必定 SAT，而且很好找。
- $m/n \gg 4.27$：約束太多，幾乎必定 UNSAT，矛盾很快就撞到。
- $m/n \approx 4.27$：SAT 與 UNSAT 各半，搜尋樹最深，**最難的實例都住在這裡**。

本篇程式的 Example 4 直接重現這條曲線（$n = 24$、每點 40 次抽樣）。

5 完整 C++ 程式

程式包含：暴力枚舉、完整 DPLL（單元傳播、純文字消去、最多出現分支啟發）、隨機 3-SAT 產生器、相變實驗。編譯：

```
g++ -std=c++17 -O2 -Wall -Wextra -o sat3_dpll sat3_dpll.cpp
```

```cpp
// sat3_dpll.cpp
// ------------------------------------------------------------------
// 3-SAT：暴力枚舉 vs. DPLL（單元傳播 + 純文字消去 + 回溯）
//
// 文字編碼：變數 v（1 起算）的正文字 = +v，負文字 = -v。
// 子句 = 文字的 vector；公式 = 子句的 vector（CNF）。
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o sat3_dpll sat3_dpll.cpp
// ------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <algorithm>
#include <random>
#include <chrono>
```

```cpp
#include <cmath>

using namespace std;

using Clause  = vector<int>;
using Formula = vector<Clause>;

// ---------- 共用：檢驗一組完整賦值是否滿足公式 ----------
bool satisfies(const Formula& f, const vector<int>& assign) {
    for (const auto& cl : f) {
        bool ok = false;
        for (int lit : cl) {
            int v = abs(lit);
            if ((lit > 0 && assign[v] == 1) || (lit < 0 && assign[v] == -1)) {
                ok = true; break;
            }
        }
        if (!ok) return false;
    }
    return true;
}

// ---------- 方法一：暴力枚舉 O(2^n · m) ----------
bool bruteForceSAT(const Formula& f, int numVars, vector<int>& model) {
    for (long long mask = 0; mask < (1LL << numVars); ++mask) {
        vector<int> assign(numVars + 1);
        for (int v = 1; v <= numVars; ++v)
            assign[v] = (mask >> (v - 1)) & 1 ? 1 : -1;
        if (satisfies(f, assign)) { model = assign; return true; }
    }
    return false;
}

// ---------- 方法二：DPLL ----------
class DPLL {
public:
    long long nodes = 0;                    // 統計：拜訪的搜尋節點數
    vector<int> model;

    bool solve(int numVars, const Formula& f) {
        nodes = 0;
        return dfs(f, vector<int>(numVars + 1, 0));
    }
private:
    // 化簡公式：刪掉已滿足子句、刪掉為假的文字
    // 回傳 false 表示出現空子句（矛盾）
    static bool simplify(const Formula& in, const vector<int>& assign,
                         Formula& out) {
        out.clear();
        for (const auto& cl : in) {
            Clause reduced;
            bool sat = false;
            for (int lit : cl) {
                int v = abs(lit), want = lit > 0 ? 1 : -1;
                if (assign[v] == want) { sat = true; break; }
                if (assign[v] == 0) reduced.push_back(lit);
            }
            if (sat) continue;
```

```
            if (reduced.empty()) return false;   // 空子句
            out.push_back(reduced);
        }
        return true;
    }

    bool dfs(Formula f, vector<int> assign) {
        ++nodes;
        // 1. 單元傳播：只剩一個文字的子句 -> 強制賦值
        bool changed = true;
        while (changed) {
            changed = false;
            Formula g;
            if (!simplify(f, assign, g)) return false;
            f = std::move(g);
            if (f.empty()) { model = assign; return true; }
            for (const auto& cl : f)
                if (cl.size() == 1) {
                    int lit = cl[0];
                    assign[abs(lit)] = lit > 0 ? 1 : -1;
                    changed = true;
                    break;
                }
        }
        // 2. 純文字消去：某變數只以單一極性出現 -> 直接滿足它
        {
            vector<int> polarity(assign.size(), 0);  // 0未見 1正 -1負 2混合
            for (const auto& cl : f)
                for (int lit : cl) {
                    int v = abs(lit), p = lit > 0 ? 1 : -1;
                    if (polarity[v] == 0) polarity[v] = p;
                    else if (polarity[v] != p) polarity[v] = 2;
                }
            for (int v = 1; v < (int)assign.size(); ++v)
                if (assign[v] == 0 && (polarity[v] == 1 || polarity[v] == -1)) {
                    assign[v] = polarity[v];
                    return dfs(f, assign);           // 化簡後重來
                }
        }
        // 3. 分支：挑出現次數最多的變數（簡單啟發）
        {
            vector<int> cnt(assign.size(), 0);
            for (const auto& cl : f)
                for (int lit : cl)
                    if (assign[abs(lit)] == 0) ++cnt[abs(lit)];
            int best = -1;
            for (int v = 1; v < (int)assign.size(); ++v)
                if (cnt[v] > 0 && (best == -1 || cnt[v] > cnt[best]))
                    best = v;
            if (best == -1) { model = assign; return true; }
            for (int val : {1, -1}) {
                auto a2 = assign;
                a2[best] = val;
                if (dfs(f, a2)) return true;
            }
            return false;
        }
```

```
    }
};

// ---------- 隨機 3-SAT 產生器 ----------
Formula random3SAT(int numVars, int numClauses, mt19937& rng) {
    Formula f;
    uniform_int_distribution<int> var(1, numVars), sign(0, 1);
    for (int i = 0; i < numClauses; ++i) {
        Clause cl;
        while ((int)cl.size() < 3) {
            int v = var(rng);
            bool dup = false;
            for (int lit : cl) if (abs(lit) == v) dup = true;
            if (!dup) cl.push_back(sign(rng) ? v : -v);
        }
        f.push_back(cl);
    }
    return f;
}

static void printModel(const vector<int>& m, int n) {
    for (int v = 1; v <= n; ++v)
        cout << " x" << v << "=" << (m[v] == 1 ? 1 : 0);
}

int main() {
    // ---- 範例 1：可滿足的小公式 ----
    // (x1 v x2 v x3) & (~x1 v ~x2 v x3) & (x1 v ~x2 v ~x3) & (~x1 v x2 v
        ~x3)
    cout << "=== Example 1: hand-crafted 3-SAT (satisfiable) ===\n";
    Formula f1 = {{1,2,3},{-1,-2,3},{1,-2,-3},{-1,2,-3}};
    vector<int> m1;
    bool b1 = bruteForceSAT(f1, 3, m1);
    cout << "brute force: " << (b1 ? "SAT, model:" : "UNSAT");
    if (b1) printModel(m1, 3);
    cout << "\n";
    DPLL d1;
    bool s1 = d1.solve(3, f1);
    cout << "DPLL       : " << (s1 ? "SAT, model:" : "UNSAT");
    if (s1) printModel(d1.model, 3);
    cout << "  (nodes=" << d1.nodes << ")\n\n";

    // ---- 範例 2：不可滿足（8 個子句覆蓋 x1,x2,x3 全部組合） ----
    cout << "=== Example 2: all 8 clauses on 3 vars (unsatisfiable)
        ===\n";
    Formula f2;
    for (int mask = 0; mask < 8; ++mask) {
        Clause cl;
        for (int v = 1; v <= 3; ++v)
            cl.push_back((mask >> (v - 1)) & 1 ? v : -v);
        f2.push_back(cl);
    }
    DPLL d2;
    cout << "DPLL: " << (d2.solve(3, f2) ? "SAT" : "UNSAT")
         << "  (nodes=" << d2.nodes << ")\n\n";

    // ---- 範例 3：隨機 3-SAT，比較暴力與 DPLL 的節點/時間 ----
    cout << "=== Example 3: random 3-SAT, brute force vs DPLL ===\n";
    mt19937 rng(20260724);
```

```cpp
    for (int n : {16, 20}) {
        int m = (int)llround(4.0 * n);          // 子句/變數比 4.0（接近相變 4.27）
        Formula f = random3SAT(n, m, rng);
        auto t0 = chrono::steady_clock::now();
        vector<int> bm;
        bool bb = bruteForceSAT(f, n, bm);
        auto t1 = chrono::steady_clock::now();
        DPLL dd;
        bool ds = dd.solve(n, f);
        auto t2 = chrono::steady_clock::now();
        double bruteMs = chrono::duration<double, milli>(t1 - t0).count();
        double dpllMs  = chrono::duration<double, milli>(t2 - t1).count();
        cout << "n=" << n << " m=" << m
             << " | brute: " << (bb ? "SAT" : "UNSAT")
             << " " << bruteMs << " ms"
             << " | DPLL: " << (ds ? "SAT" : "UNSAT")
             << " " << dpllMs << " ms, nodes=" << dd.nodes;
        cout << " | agree=" << (bb == ds ? "yes" : "NO!") << "\n";
        if (ds && !satisfies(f, dd.model)) cout << "  ERROR: bad model!\n";
    }

    // ---- 範例 4：相變現象觀察（m/n 從 3.0 到 5.5） ----
    cout << "\n=== Example 4: phase transition (fraction SAT over 40 trials) ===\n";
    int n = 24;
    for (double ratio : {3.0, 3.5, 4.0, 4.27, 4.5, 5.0, 5.5}) {
        int satCount = 0, trials = 40;
        long long totalNodes = 0;
        for (int t = 0; t < trials; ++t) {
            Formula f = random3SAT(n, (int)llround(ratio * n), rng);
            DPLL dd;
            if (dd.solve(n, f)) ++satCount;
            totalNodes += dd.nodes;
        }
        cout << "m/n=" << ratio << "  SAT rate=" << satCount << "/" << trials
             << "  avg nodes=" << totalNodes / trials << "\n";
    }
    return 0;
}
```

程式碼 1: sat3_dpll.cpp — 3-SAT 暴力法與 DPLL 完整實作

6 執行結果與解讀

```
=== Example 1: hand-crafted 3-SAT (satisfiable) ===
brute force: SAT, model: x1=1 x2=0 x3=0
DPLL       : SAT, model: x1=1 x2=1 x3=1  (nodes=3)

=== Example 2: all 8 clauses on 3 vars (unsatisfiable) ===
DPLL: UNSAT  (nodes=7)

=== Example 3: random 3-SAT, brute force vs DPLL ===
n=16 m=64 | brute: SAT 6.23 ms | DPLL: SAT 0.12 ms, nodes=7 | agree=yes
n=20 m=80 | brute: SAT 6.03 ms | DPLL: SAT 0.71 ms, nodes=8 | agree=yes

=== Example 4: phase transition (fraction SAT over 40 trials) ===
```

```
m/n=3      SAT rate=40/40   avg nodes=14
m/n=3.5    SAT rate=38/40   avg nodes=17
m/n=4      SAT rate=37/40   avg nodes=20
m/n=4.27   SAT rate=25/40   avg nodes=21
m/n=4.5    SAT rate=24/40   avg nodes=20
m/n=5      SAT rate=7/40    avg nodes=19
m/n=5.5    SAT rate=5/40    avg nodes=18
```

筆記

三個觀察：(1) Example 1 中兩法都回報 SAT 但給出**不同的解**——SAT 的解可以不唯一，驗證時要驗「是不是解」而非「跟標準答案一不一樣」。(2) Example 2 用 3 變數的全部 8 個子句構造出必然矛盾的公式，DPLL 只花 7 個節點就證明 UNSAT。(3) Example 4 的 SAT 率在 $m/n = 4.27$ 附近從幾乎全 SAT 掉到幾乎全 UNSAT，且平均節點數在該處達到峰值——正是理論預測的相變。

7 實務應用

- **硬體驗證**：晶片電路等價性檢查被編碼成 SAT——「兩個電路存在輸入使輸出不同」可滿足嗎？Intel、AMD 每天用工業級求解器跑這件事。
- **軟體套件相依解析**：apt、conda、cargo 的版本相依解析本質是 SAT（「裝 A 需要 B≥2 或 C；B 與 D 衝突……」）。
- **排程與規劃**：課表、機組人員排班、AI 規劃（STRIPS planning）常被翻譯成 SAT 後丟給求解器。
- **現代求解器**：CDCL（衝突驅動子句學習）= DPLL + 從失敗中學新子句 + 重啟 + 高效資料結構（watched literals）。學會 DPLL，就看懂了 MiniSat、Z3 的心臟。

8 練習題

1. 手算：對 $(x_1 \vee x_2 \vee x_3) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee \neg x_3)$ 跑一遍 DPLL，記錄每次傳播與分支。
2. 把分支啟發從「出現最多」改成「隨機挑」，在 $m/n = 4.27$ 的實例上比較平均節點數。
3. 實作**子句學習**雛形：回溯時記下導致矛盾的指派組合，作為新子句加入公式，觀察節點數變化。
4. 寫一個 2-SAT 的多項式解法（蘊含圖 + SCC），對照本篇程式驗證答案一致。
5. 挑戰：把 N-Queens 編碼成 SAT（每格一個變數），用你的 DPLL 解 $n = 6$。

9 小結

方法	時間複雜度	適用時機
暴力枚舉	$O(2^n m)$	$n \le 25$；當驗證基準
DPLL	最壞 $O(2^n)$，實務遠低	中小型實例、教學、理解求解器
CDCL（工業）	最壞指數，實務百萬變數	真實世界的一切

延伸閱讀：Cook (1971) The Complexity of Theorem-Proving Procedures；Biere et al.,*Handbook of Satisfiability*；MiniSat 原始碼（約 2000 行，出乎意料地好讀）。