NP 難題演算法系列 · 第 09 篇

# 旅行推銷員問題

TSP — Held–Karp DP & Nearest-Neighbor + 2-opt

最著名的 NP-Hard 問題：精確解的極限在哪、啟發式又能多接近最優

## Contents



## 1 問題是什麼

**定義 1.1** (TSP)**.** 給定 $n$ 個城市與兩兩距離 $d(i,j)$，求一條**經過每城恰一次並回到起點**的最短迴路。判定版是 NP-Complete；**強** NP-Hard（距離數值再小也難——對比背包的弱 NP-Hard）。一般距離下連常數倍近似都是 NP-Hard；**度量 TSP**（滿足三角不等式）才有 1.5 倍近似（Christofides）。

**白話比喻**

外送員今天有 30 個地址要跑。路線的好壞直接是油錢和時間：$30! \approx 2.7 \times 10^{32}$ 種順序，比全宇宙的星星還多。但外送平台每天在幾秒內排出幾千條「夠好」的路線——本篇就是要講清楚**精確**和**夠好**這兩條路各自怎麼走、走到哪會撞牆。

# 2 精確解：Held–Karp 位元 DP

## 2.1 狀態設計（上一篇的直接升級）

Hamiltonian 判存在用布林，這裡問最短，把布林換成距離：

$$dp[S][v] = \text{從城0 出發、恰好經過集合}S\text{、目前在}v\text{ 的最短長度}$$

**初始**：$dp[\{0\}][0] = 0$。　**轉移**：

$$dp[S \cup \{u\}][u] = \min\big(dp[S \cup \{u\}][u],\ dp[S][v] + d(v,u)\big)$$

**答案**：$\min_{v \neq 0}\ dp[\text{全集}][v] + d(v,0)$。時間 $O(2^n n^2)$、空間 $O(2^n n)$。

## 2.2 規模牆

| $n$ | 轉移數 $2^n n^2$ | DP 表記憶體（double） |
|---|---|---|
| 15 | $7.4 \times 10^6$ | 4 MB |
| 20 | $4.2 \times 10^8$ | 168 MB |
| 25 | $2.1 \times 10^{10}$ | 6.7 GB |
| 30 | $9.7 \times 10^{11}$ | 258 GB |

$n \approx 22$ 是家用電腦的實際極限。但相比 $n!$ 枚舉（$n = 20$ 時 $2.4 \times 10^{18}$）已是天壤之別，而且 $O(2^n \,\text{poly})$ 至今仍是一般 TSP 的最佳精確複雜度——60 年沒人打破。

# 3 啟發式一：最近鄰（Nearest Neighbor）

從起點開始，每步走向**最近的未訪城市**，最後回起點。$O(n^2)$、三行寫完。品質：平均比最優差 10–25%，最壞可達 $\Theta(\log n)$ 倍——它會把「順路該撿的城市」留到最後變成昂貴的長跳。

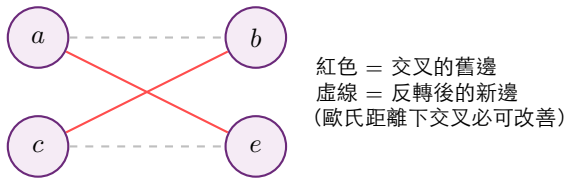
**實務技巧**

廉價補救 1：**多起點**——每個城市當一次起點跑 $n$ 遍取最好，$O(n^3)$ 常見改善 5% 左右（見執行結果 Example 3）。

# 4 啟發式二：2-opt 局部搜尋

## 4.1 核心動作：拆兩邊、反轉重接

若 $d(a,c) + d(b,e) < d(a,b) + d(c,e)$，把 tour 中的 $(a,b)$ 與 $(c,e)$ 拆掉、中段反轉——直觀上是**解開一個交叉**：




反覆掃描所有 $(i,j)$ 對直到無法改善（局部最優）。每輪 $O(n^2)$，通常幾輪收斂。NN 建構 + 2-opt 改善是「兩分鐘寫完、效果驚人」的組合拳：實測小型歐氏實例常直接命中最優（見執行結果）。

**筆記**

再往上是 3-opt、Lin–Kernighan（LKH）——LKH 在十萬城市級實例上能到最優 1% 以內，是目前實務最強的 TSP 啟發式。思想同源：不斷做「拆 $k$ 條邊重接」的局部改善。

## 5 手算範例：4 城市

距離矩陣（對稱）：$d_{01}=10, d_{02}=15, d_{03}=20, d_{12}=35, d_{13}=25, d_{23}=30$。
所有迴路只有 $(4-1)!/2=3$ 條本質不同：

$$0\to1\to2\to3\to0:\ 10+35+30+20=95;\quad 0\to1\to3\to2\to0:\ 10+25+30+15=\mathbf{80};$$

$$0\to2\to1\to3\to0:\ 15+35+25+20=95.$$

最優 80。NN 從 0 出發：最近是 1（10），1 的最近未訪是 3（25），再 2（30），回 0（15）——恰好也是 80。NN 不總是這麼幸運，Example 2 就沒中。

## 6 完整 C++ 程式

包含：Held–Karp（含 tour 還原）、NN、多起點 NN、2-opt、隨機歐氏實例產生器、與最優的 gap 統計、規模牆實測。編譯：

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

```cpp
// tsp.cpp
// ------------------------------------------------------------------
// TSP（旅行推銷員問題）：
//   1. Held-Karp 位元 DP：精確解 O(2^n · n^2)，含路徑還原
//   2. 最近鄰啟發式（Nearest Neighbor）：O(n^2) 快速建構
//   3. 2-opt 局部搜尋：反覆拆兩邊重接，改善 NN 的解
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o tsp tsp.cpp
// ------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <algorithm>
#include <cmath>
#include <random>
#include <chrono>
#include <iomanip>

using namespace std;

using Matrix = vector<vector<double>>;

// ---------- 1. Held-Karp ----------
// dp[S][v] = 從城市 0 出發、恰好經過集合 S、目前在 v 的最短距離
pair<double, vector<int>> heldKarp(const Matrix& d) {
    int n = (int)d.size();
    const double INF = 1e18;
    vector<vector<double>> dp(1 << n, vector<double>(n, INF));
    vector<vector<int>> par(1 << n, vector<int>(n, -1));
    dp[1][0] = 0;                                   // 從 0 出發
    for (int S = 1; S < (1 << n); ++S) {
        if (!(S & 1)) continue;                     // 必含起點 0
        for (int v = 0; v < n; ++v) {
```

```
            if (dp[S][v] >= INF) continue;
            for (int u = 0; u < n; ++u) {
                if (S & (1 << u)) continue;
                int S2 = S | (1 << u);
                double nd = dp[S][v] + d[v][u];
                if (nd < dp[S2][u]) {
                    dp[S2][u] = nd;
                    par[S2][u] = v;
                }
            }
        }
    }
    int full = (1 << n) - 1;
    double best = INF;
    int last = -1;
    for (int v = 1; v < n; ++v)                          // 回到 0 收尾
        if (dp[full][v] + d[v][0] < best) {
            best = dp[full][v] + d[v][0];
            last = v;
        }
    vector<int> tour;                                    // 還原
    int S = full, v = last;
    while (v != -1) {
        tour.push_back(v);
        int p = par[S][v];
        S &= ~(1 << v);
        v = p;
    }
    reverse(tour.begin(), tour.end());                   // 0 ... last
    tour.push_back(0);                                   // 閉合
    return {best, tour};
}

// ---------- 2. 最近鄰啟發式 ----------
pair<double, vector<int>> nearestNeighbor(const Matrix& d, int start = 0)
    {
    int n = (int)d.size();
    vector<bool> used(n, false);
    vector<int> tour = {start};
    used[start] = true;
    double len = 0;
    for (int step = 1; step < n; ++step) {
        int cur = tour.back(), next = -1;
        for (int v = 0; v < n; ++v)
            if (!used[v] && (next == -1 || d[cur][v] < d[cur][next]))
                next = v;
        used[next] = true;
        len += d[cur][next];
        tour.push_back(next);
    }
    len += d[tour.back()][start];
    tour.push_back(start);
    return {len, tour};
}

// ---------- 3. 2-opt 局部搜尋 ----------
// 拆掉 (t[i],t[i+1]) 和 (t[j],t[j+1])，把中段反轉重接；有改善就做
pair<double, vector<int>> twoOpt(const Matrix& d, vector<int> tour) {
    int n = (int)tour.size() - 1;                        // 尾巴是回到起點
```

```
    bool improved = true;
    while (improved) {
        improved = false;
        for (int i = 0; i < n - 1; ++i)
            for (int j = i + 1; j < n; ++j) {
                int a = tour[i], b = tour[i + 1];
                int c = tour[j], e = tour[j + 1];
                double delta = d[a][c] + d[b][e] - d[a][b] - d[c][e];
                if (delta < -1e-10) {
                    reverse(tour.begin() + i + 1, tour.begin() + j + 1);
                    improved = true;
                }
            }
    }
    double len = 0;
    for (size_t i = 0; i + 1 < tour.size(); ++i)
        len += d[tour[i]][tour[i + 1]];
    return {len, tour};
}

static Matrix randomEuclidean(int n, mt19937& rng,
                              vector<pair<double,double>>* ptsOut =
                                 nullptr) {
    uniform_real_distribution<double> coord(0, 100);
    vector<pair<double,double>> pts;
    for (int i = 0; i < n; ++i) pts.push_back({coord(rng), coord(rng)});
    Matrix d(n, vector<double>(n, 0));
    for (int i = 0; i < n; ++i)
        for (int j = 0; j < n; ++j)
            d[i][j] = hypot(pts[i].first - pts[j].first,
                            pts[i].second - pts[j].second);
    if (ptsOut) *ptsOut = pts;
    return d;
}

static void printTour(const vector<int>& t) {
    for (size_t i = 0; i < t.size(); ++i)
        cout << t[i] << (i + 1 < t.size() ? "-" : "\n");
}

int main() {
    cout << fixed << setprecision(2);

    // ---- 範例 1：手算小例（4 城市） ----
    cout << "=== Example 1: 4-city worked example ===\n";
    Matrix m4 = {{0, 10, 15, 20},
                 {10, 0, 35, 25},
                 {15, 35, 0, 30},
                 {20, 25, 30, 0}};
    auto [opt4, tour4] = heldKarp(m4);
    cout << "Held-Karp optimal = " << opt4 << " (expect 80)  tour: ";
    printTour(tour4);
    auto [nn4, nnTour4] = nearestNeighbor(m4);
    cout << "Nearest neighbor  = " << nn4 << "  tour: ";
    printTour(nnTour4);
    cout << "\n";

    // ---- 範例 2：NN 可以多糟 + 2-opt 補救 ----
    cout << "=== Example 2: NN vs NN+2opt vs optimal (n=12, Euclidean)
```

```
        ===\n";
148     mt19937 rng(20260724);
149     Matrix d12 = randomEuclidean(12, rng);
150     auto t0 = chrono::steady_clock::now();
151     auto [opt, optTour] = heldKarp(d12);
152     auto t1 = chrono::steady_clock::now();
153     auto [nnLen, nnTour] = nearestNeighbor(d12);
154     auto [toLen, toTour] = twoOpt(d12, nnTour);
155     auto t2 = chrono::steady_clock::now();
156     cout << "optimal (Held-Karp): " << opt << "  ("
157          << chrono::duration<double, milli>(t1 - t0).count() << " ms)\n";
158     cout << "nearest neighbor   : " << nnLen
159          << "  (+" << 100 * (nnLen / opt - 1) << "% above optimal)\n";
160     cout << "NN + 2-opt         : " << toLen
161          << "  (+" << 100 * (toLen / opt - 1) << "% above optimal, "
162          << chrono::duration<double, milli>(t2 - t1).count() << " ms)\n";
163     cout << "optimal tour: ";
164     printTour(optTour);
165     cout << "2-opt tour  : ";
166     printTour(toTour);
167     (void)optTour;
168     cout << "\n";
169
170     // ---- 範例 3：多起點 NN（廉價的改善技巧） ----
171     cout << "=== Example 3: multi-start nearest neighbor ===\n";
172     double bestNN = 1e18;
173     int bestStart = 0;
174     for (int s = 0; s < 12; ++s) {
175         auto [len, t] = nearestNeighbor(d12, s);
176         if (len < bestNN) { bestNN = len; bestStart = s; }
177         (void)t;
178     }
179     cout << "best of 12 starts: " << bestNN << " (start=" << bestStart
180          << ", single-start was " << nnLen << ")\n\n";
181
182     // ---- 範例 4：規模牆——Held-Karp 的極限 ----
183     cout << "=== Example 4: the scaling wall ===\n";
184     for (int n : {13, 16, 19}) {
185         Matrix d = randomEuclidean(n, rng);
186         auto s0 = chrono::steady_clock::now();
187         auto [o, ot] = heldKarp(d);
188         auto s1 = chrono::steady_clock::now();
189         auto [nl, nt] = nearestNeighbor(d);
190         auto [tl, tt] = twoOpt(d, nt);
191         auto s2 = chrono::steady_clock::now();
192         (void)ot; (void)tt;
193         cout << "n=" << n
194              << " | HK " << chrono::duration<double, milli>(s1 -
                    s0).count()
195              << " ms (opt=" << o << ")"
196              << " | NN+2opt " << chrono::duration<double, milli>(s2 -
                    s1).count()
197              << " ms (len=" << tl << ", gap +"
198              << 100 * (tl / o - 1) << "%)\n";
199         (void)nl;
200     }
201     cout << "Held-Karp memory: 2^n * n doubles -> n=25 needs ~6.7 GB.
        Game over.\n";
202     return 0;
```

```
}
```

程式碼 1: tsp.cpp — TSP 精確解與啟發式完整實作

## 7 執行結果與解讀

```
=== Example 1: 4-city worked example ===
Held-Karp optimal = 80.00 (expect 80)  tour: 0-2-3-1-0
Nearest neighbor  = 80.00  tour: 0-1-3-2-0

=== Example 2: NN vs NN+2opt vs optimal (n=12, Euclidean) ===
optimal (Held-Karp): 301.09  (1.32 ms)
nearest neighbor   : 316.19  (+5.01% above optimal)
NN + 2-opt         : 301.09  (+0.00% above optimal, 0.00 ms)

=== Example 3: multi-start nearest neighbor ===
best of 12 starts: 311.10 (start=7, single-start was 316.19)

=== Example 4: the scaling wall ===
n=13 | HK 3.70 ms   (opt=338.65) | NN+2opt 0.01 ms (len=338.65, gap +0.00%)
n=16 | HK 38.10 ms  (opt=335.43) | NN+2opt 0.01 ms (len=335.43, gap +0.00%)
n=19 | HK 407.14 ms (opt=368.26) | NN+2opt 0.06 ms (len=368.26, gap +0.00%)
Held-Karp memory: 2^n * n doubles -> n=25 needs ~6.7 GB. Game over.
```

**筆記**

Example 1 中兩條 tour 方向相反（0−2−3−1−0 與 0−1−3−2−0）——同一條迴路的兩種走向，長度相同。Example 4 的時間欄是「指數牆」的現場直播：$n$ 每 +3，Held–Karp 時間 ×10，而 NN+2-opt 幾乎不動還全中最優。**別被小例騙**：2-opt 在小型歐氏實例上常命中最優，$n$ 上百後 gap 通常會落在 2–5%，這時就需要模擬退火、LKH 或本系列第 13 篇的遺傳演算法。

## 8 實務應用

- **物流路線**：UPS 的 ORION 系統每天為 5.5 萬條路線做 TSP 級優化，號稱一年省 3 億美元油錢。
- **製造**：PCB 鑽孔、雷射切割、3D 列印噴頭路徑——都是「訪問所有點的最短路」。
- **晶片測試**：探針測試順序優化。
- **天文觀測排程**：望遠鏡轉動角度最小化的觀測順序。
- **里程碑**：Concorde 求解器已精確解出 85,900 城的實例（花了 136 CPU 年）——「NP-Hard」與「特定實例解不出」是兩回事的最佳證明。

## 9 練習題

1. 手算 5 城市完全圖（自訂距離）的 Held–Karp 表前兩層（$|S| = 2, 3$ 的所有狀態）。
2. 把 2-opt 改成「first improvement」（找到第一個改善就做）vs 現在的「best sweep」，比較收斂速度與最終品質。
3. 實作 Or-opt（把連續 1–3 個城市搬到別處），疊在 2-opt 之後看還能擠出多少。
4. 對 $n = 100$ 隨機歐氏實例：NN、NN+2opt、多起點 NN+2opt 三者的 gap 統計（以多起點 2-opt 最好值當基準）。

5. 挑戰：實作 MST 下界（最優 tour $\geq$ MST 權重），觀察 2-opt 解與下界的夾擠區間。

## 10 小結

| 方法 | 時間 | 品質 | 適用時機 |
|---|---|---|---|
| Held–Karp | $O(2^n n^2)$ | 精確 | $n \leq 22$ |
| 最近鄰 | $O(n^2)$ | 平均 +10–25% | 要秒出、當初始解 |
| NN + 2-opt | $O(n^2)$/輪 | 平均 +2–5% | 通用首選 |
| Christofides | $O(n^3)$ | $\leq 1.5\times$（度量） | 要理論保證 |
| LKH | 高（啟發式） | +1% 以內 | 大型實例的王者 |

**延伸閱讀**：Applegate et al. *The Traveling Salesman Problem: A Computational Study*（Concorde 團隊）；Lin & Kernighan (1973)；Karlin, Klein & Oveis Gharan (2021) 打破 Christofides 1.5 界的隨機化演算法。