NP 難題演算法系列 · 第 12 篇

工作排程問題

Job Scheduling — Graham's List Scheduling & LPT

n 個工作分給 m 台機器、最晚完工時間最短：近似演算法理論的誕生地（Graham 1966）

Contents

1 問題是什麼 **1**

1.1 兩個下界 1

2 演算法一：List Scheduling（Graham 1966） **1**

2.1 規則 1

2.2 弱點 2

3 演算法二：LPT（Longest Processing Time first） **2**

3.1 規則 2

3.2 手算範例：緊例（$m = 2$） 2

4 精確解：回溯 + 剪枝 **2**

5 完整 C++ 程式 **2**

6 執行結果與解讀 **5**

7 實務應用 **6**

8 練習題 **6**

9 小結 **6**

1 問題是什麼

定義 1.1 ($P \,\|\, C_{\max}$)**.** n 個工作，第 j 個需時 p_j；m 台**相同**機器，每台一次做一個工作、不可中斷。目標：分配工作使 **makespan**（最晚完工的機器的完工時間）$C_{\max}$ 最小。$m \geq 2$ 即 NP-Hard：$m = 2$ 就是 Partition（兩堆越平均 makespan 越小）。

白話比喻

廚房有 4 個爐子、14 道菜，每道菜火候時間不同。客人只在乎**最後一道菜什麼時候出**。怎麼分爐子？——這就是 makespan 排程，資料中心把批次任務丟給伺服器叢集時解的正是同一題。

1.1 兩個下界

任何排法都不可能低於：

$$C_{\max} \;\geq\; \max\Bigg(\underbrace{\tfrac{1}{m}\sum_j p_j}_{\text{平均負載}},\;\; \underbrace{\max_j p_j}_{\text{最長單一工作}}\Bigg)$$

2 演算法一：List Scheduling（Graham 1966）

2.1 規則

照**任意順序**把工作一個個丟給**目前最空**的機器。$O(n \log m)$、線上可用（工作到了就分派，不用預知未來）。

定理 2.1 (Graham 界)**.** $LS \leq \left(2 - \frac{1}{m}\right) OPT$。

Proof. 設最後完工的機器上、最後放進去的工作是 j，它在時刻 s 開始。s 之前**所有**機器都忙（否則 j 會被丟去更空的機器），故 $s \leq \frac{1}{m}\sum_{i \neq j} p_i$。於是

$$C_{\max} = s + p_j \;\leq\; \frac{\sum_i p_i}{m} + \Big(1 - \frac{1}{m}\Big)p_j \;\leq\; OPT + \Big(1 - \frac{1}{m}\Big)OPT.$$

兩項分別用了上面的兩個下界。**這是史上第一個帶證明的近似比**，近似演算法這個領域由此開張。 □

2.2 弱點

到貨順序可以很壞：一堆小工作先來把機器墊得參差不齊，最後來一個大工作就慘了。

3 演算法二：LPT（Longest Processing Time first）

3.1 規則

先把工作**由長到短排序**，再跑 List Scheduling。直覺同 FFD：大石頭先放，沙子後撒。

定理 3.1 (Graham 1969)**.** $LPT \leq \left(\frac{4}{3} - \frac{1}{3m}\right) OPT$，且此界緊。

3.2 手算範例：緊例（$m = 2$）

工作 $\{3, 3, 2, 2, 2\}$（總和 12，$OPT = 6$：如 $\{3,3\}$ 和 $\{2,2,2\}$）。LPT 過程：

步驟	分派	M0 負載	M1 負載
3	→ M0	3	0
3	→ M1	3	3
2	→ M0（同載取編號小）	5	3
2	→ M1	5	5
2	→ M0	**7**	5

$LPT = 7$、$OPT = 6$，比率 7/6 恰等於 $\frac{4}{3} - \frac{1}{3 \cdot 2}$——理論界在五個小數字上就被打滿。

4 精確解：回溯 + 剪枝

小實例可回溯枚舉「每個工作放哪台」，三個剪枝讓它實用：(1) 工作先由大到小排（大工作的選擇最受限，先定）；(2) **對稱剪枝**——負載相同的機器只試一台（機器可交換）；(3) 目前最大負載 ≥ 已知最佳即砍，且用 **LPT 當初始上界**。本篇程式以此為 100 個隨機實例提供標準答案。

5 完整 C++ 程式

包含：List Scheduling、LPT、回溯精確解（LPT 上界 + 對稱剪枝）、下界計算、LPT 緊例、與 Partition 的關係展示、100 實例統計。編譯：

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

```cpp
// job_scheduling.cpp
// ------------------------------------------------------------------
// Job Scheduling（等同機台排程，目標：最小化 makespan）
// P || Cmax 是 NP-Hard（m>=2 時，因為 Partition 可歸約過來）
//   1. List Scheduling（Graham 1966）：任意順序丟給最空的機器
//      保證 <= (2 - 1/m)· OPT
//   2. LPT（Longest Processing Time first）：先排序再 List
//      保證 <= (4/3 - 1/(3m))· OPT
//   3. 精確解：回溯 + 剪枝（小實例）
//
// Compile: g++ -std=c++17 -O2 -Wall -Wextra -o job_scheduling
    job_scheduling.cpp
// ------------------------------------------------------------------
#include <iostream>
#include <vector>
#include <algorithm>
#include <numeric>
#include <functional>

using namespace std;

struct Schedule {
    vector<vector<int>> machineJobs;   // 每台機器分到的工時清單
    int makespan = 0;
};

// ---------- 1. List Scheduling ----------
Schedule listScheduling(const vector<int>& jobs, int m) {
    Schedule s;
    s.machineJobs.assign(m, {});
    vector<int> load(m, 0);
    for (int p : jobs) {
        int target = min_element(load.begin(), load.end()) - load.begin();
        s.machineJobs[target].push_back(p);
        load[target] += p;
    }
    s.makespan = *max_element(load.begin(), load.end());
    return s;
}

// ---------- 2. LPT ----------
Schedule lptScheduling(vector<int> jobs, int m) {
    sort(jobs.begin(), jobs.end(), greater<int>());
    return listScheduling(jobs, m);
```

```cpp
}

// ---------- 3. 精確解：回溯 ----------
class ExactScheduler {
    vector<int> jobs;
    int m, best;
public:
    long long nodes = 0;
    int solve(vector<int> js, int m_) {
        jobs = std::move(js);
        sort(jobs.begin(), jobs.end(), greater<int>());   // 大工作先放
        m = m_;
        best = lptScheduling(jobs, m).makespan;           // LPT 當初始上界
        nodes = 0;
        vector<int> load(m, 0);
        dfs(0, load);
        return best;
    }
private:
    void dfs(int idx, vector<int>& load) {
        ++nodes;
        int cur = *max_element(load.begin(), load.end());
        if (cur >= best) return;                          // 剪枝
        if (idx == (int)jobs.size()) {
            best = cur;
            return;
        }
        for (int k = 0; k < m; ++k) {
            // 對稱剪枝：負載相同的機器只試一台
            bool dup = false;
            for (int k2 = 0; k2 < k; ++k2)
                if (load[k2] == load[k]) { dup = true; break; }
            if (dup) continue;
            load[k] += jobs[idx];
            dfs(idx + 1, load);
            load[k] -= jobs[idx];
        }
    }
};

static int theoreticalLB(const vector<int>& jobs, int m) {
    int total = accumulate(jobs.begin(), jobs.end(), 0);
    int lb = (total + m - 1) / m;                         // 平均負載
    lb = max(lb, *max_element(jobs.begin(), jobs.end()));  // 最大單工
    return lb;
}

static void show(const string& tag, const Schedule& s) {
    cout << tag << ": makespan = " << s.makespan << "\n";
    for (int k = 0; k < (int)s.machineJobs.size(); ++k) {
        int sum = accumulate(s.machineJobs[k].begin(),
                             s.machineJobs[k].end(), 0);
        cout << "  M" << k << " [";
        for (size_t i = 0; i < s.machineJobs[k].size(); ++i)
            cout << s.machineJobs[k][i]
                 << (i + 1 < s.machineJobs[k].size() ? "," : "");
        cout << "] load " << sum << "\n";
    }
```

```cpp
}

int main() {
    // ---- 範例 1：Graham 的經典壞例（List 被到貨順序坑） ----
    cout << "=== Example 1: arrival order hurts list scheduling ===\n";
    // m=3。順序 {2,2,2,2,2,2,3,3,3} vs 排序後
    vector<int> a = {2, 2, 2, 2, 2, 3, 3, 3, 3};
    int m = 3;
    cout << "jobs {2,2,2,2,2,2,3,3,3}, machines = 3, lower bound = "
         << theoreticalLB(a, m) << "\n";
    show("List (as-is order)", listScheduling(a, m));
    show("LPT  (sorted)     ", lptScheduling(a, m));
    ExactScheduler ex;
    cout << "exact makespan = " << ex.solve(a, m) << "\n\n";

    // ---- 範例 2：LPT 的著名反例（達到 4/3 界） ----
    cout << "=== Example 2: LPT worst case (m=2) ===\n";
    // {3,3,2,2,2}: LPT -> 7, OPT -> 6, 比率 7/6
    vector<int> b = {3, 3, 2, 2, 2};
    cout << "jobs {3,3,2,2,2}, machines = 2\n";
    show("LPT", lptScheduling(b, 2));
    cout << "exact makespan = " << ex.solve(b, 2)
         << "  (LPT/OPT = 7/6, bound 4/3 - 1/6 = 7/6 —— tight!)\n\n";

    // ---- 範例 3：與 Partition 的關係 ----
    cout << "=== Example 3: scheduling on 2 machines IS partition ===\n";
    vector<int> c = {8, 6, 5, 14, 13, 9, 12, 7};   // 總和 74
    cout << "jobs sum = " << accumulate(c.begin(), c.end(), 0)
         << ", machines = 2, perfect split would give makespan 37\n";
    int opt = ex.solve(c, 2);
    cout << "exact makespan = " << opt
         << (opt == 37 ? "  -> a perfect partition exists!" : "") << "\n";
    show("LPT", lptScheduling(c, 2));
    cout << "\n";

    // ---- 範例 4：隨機統計——兩個近似的實際表現 ----
    cout << "=== Example 4: average quality over random instances ===\n";
    unsigned seed = 424242;
    auto nextRand = [&seed]() {
        seed = seed * 1103515245 + 12345;
        return (seed >> 16) & 0x7fff;
    };
    int trials = 100;
    double listSum = 0, lptSum = 0;
    int lptOptimal = 0;
    for (int t = 0; t < trials; ++t) {
        vector<int> jobs;
        for (int i = 0; i < 14; ++i) jobs.push_back(1 + nextRand() % 30);
        int mm = 4;
        int ls = listScheduling(jobs, mm).makespan;
        int lp = lptScheduling(jobs, mm).makespan;
        int op = ex.solve(jobs, mm);
        listSum += (double)ls / op;
        lptSum += (double)lp / op;
        if (lp == op) ++lptOptimal;
    }
    cout << trials << " random instances (14 jobs, p in 1..30, m=4):\n";
    cout << "avg List/OPT = " << listSum / trials
         << "  (guarantee " << 2.0 - 1.0 / 4 << ")\n";
```

```
    cout << "avg LPT /OPT = " << lptSum / trials
         << "  (guarantee " << 4.0 / 3 - 1.0 / 12 << ")\n";
    cout << "LPT hit the optimum " << lptOptimal << "/" << trials << "
       times\n";
    return 0;
}
```

程式碼 1: job_scheduling.cpp — 列表排程、LPT 與精確解完整實作

6 執行結果與解讀

```
=== Example 2: LPT worst case (m=2) ===
jobs {3,3,2,2,2}, machines = 2
LPT: makespan = 7    M0 [3,2,2] load 7 | M1 [3,2] load 5
exact makespan = 6   (LPT/OPT = 7/6, bound 4/3 - 1/6 = 7/6 -- tight!)

=== Example 3: scheduling on 2 machines IS partition ===
jobs sum = 74, machines = 2, perfect split would give makespan 37
exact makespan = 37  -> a perfect partition exists!
LPT: makespan = 37   M0 [14,9,8,6] load 37 | M1 [13,12,7,5] load 37

=== Example 4: average quality over random instances ===
100 random instances (14 jobs, p in 1..30, m=4):
avg List/OPT = 1.152  (guarantee 1.75)
avg LPT /OPT = 1.028  (guarantee 1.25)
LPT hit the optimum 34/100 times
```

筆記

Example 3 把「$m = 2$ 排程 = Partition」演給你看：總和 74 的八個工作恰好能對半分 (37/37)，LPT 這次也找到了完美分割。Example 4 的統計則是兩代演算法的成績單：排序這一步把平均誤差從 15% 壓到 3%，成本只是一次 $O(n \log n)$——**在丟給貪心之前先排序，是排程界最划算的一行程式碼**。

7 實務應用

- **資料中心**：MapReduce/Spark 把 task 分給 worker、K8s 的 Pod 調度——都是 makespan 的變形（真實版還帶資料親和性、搶佔等約束）。
- **建置系統**：CI 把測試分片到 m 台 runner，最慢的分片決定整條 pipeline 的時間——LPT 是最常用的分片策略。
- **製造業**：注塑機、CNC 的批次生產排程。
- **多核心運算**：OpenMP 的 `schedule(dynamic)` 本質是線上 List Scheduling；`static` 對均勻工作更好——理解本篇就理解了這個選項該怎麼選。
- **理論意義**：這是近似演算法理論的誕生地；後續有 PTAS（Hochbaum–Shmoys 1987）：任給 ε，可 $(1+\varepsilon)$ 近似，時間 $O\big((n/\varepsilon)^{1/\varepsilon^2}\big)$ 級——多項式但 ε 小時天文數字，實務仍用 LPT。

8 練習題

1. 手算：工作 $\{5,4,3,3,2,2,1\}$、$m = 3$，分別跑 List（原順序）與 LPT，對照下界。
2. 構造 $m = 3$ 的 LPT 緊例（提示：$\{5,5,4,4,3,3,3\}$ 附近找，目標比率 $4/3 - 1/9 = 11/9$）。

3. 把精確解的剪枝逐一關掉（不排序／不對稱剪枝／不用 LPT 上界），實測節點數各膨脹多少。
4. 實作 **Multifit**（二分猜 makespan + FFD 裝箱驗證）——排程與裝箱的美妙互換，保證 $1.22\,OPT$。
5. 挑戰：非等速機器（$Q \parallel C_{\max}$，每台有速度 s_i）——修改 LPT 使其分給「完工時刻最早」的機器，實測品質。

9 小結

方法	時間	保證	適用時機
List Scheduling	$O(n \log m)$	$(2 - \frac{1}{m})OPT$	線上、工作陸續到達
LPT	$O(n \log n)$	$(\frac{4}{3} - \frac{1}{3m})OPT$	離線標準解
Multifit	$O(n \log n \log P)$	$1.22\,OPT$	要更緊的保證
回溯精確	指數	精確	$n \lesssim 20$
PTAS	多項式（巨大）	$(1+\varepsilon)OPT$	理論

延伸閱讀：Graham (1966, 1969) 兩篇開山之作；Hochbaum & Shmoys (1987) PTAS；Coffman, Garey & Johnson (1978) Multifit 分析。