

排程問題之資料複雜度

資財系 08 黃大祐

資財系 08 劉書維

壹、緒論

在資訊領域資料結構與演算法中，有詳細講述空間複雜度與時間複雜度，這些都是撰寫程式時須注意的基本概念，也有整理出各搜尋法的時間複雜度，那進入我們研究主題前先快速複習一些基本觀念

一、空間複雜度：

程式執行時所需的記憶體空間，包含固定空間（Fixed Space）與變動空間（Variable Space）。

（一）固定空間（C）：

並非主要考量，包括簡單變數（int, float）常數、固定大小的結構變數（陣列等）

（二）變動空間（SP）：

包括傳值呼叫(Call By Value)的結構型參數、遞迴所需的堆疊空間

所以空間函式 $S(P)$ 的表示法： $S(P) = C + SP$

二、時間複雜度：

程式執行完所需的計算時間，時間函式的表示法 $T(n)$ 可以用下列常見方式表示。

（一）大寫的英文字母 O 函數：代表演算法執行步驟數目上限。

（二）大寫的希臘字母 Ω 函數：代表演算法執行步驟數目下限。

（三）大寫的希臘字母 Θ 函數：代表同時滿足上限與下限。

• 常見搜尋法的時間複雜度：

	Time Complexity			Space Complexity	Stable/Unstable
	Best	Worst	Avg.		
Insert Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Stable
Select Sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	Unstable
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	Stable
Quick Sort	$O(n \log n)$	$O(n^2)$	$O(n \log n)$	$O(\log n) \sim O(n)$	Unstable
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	Stable
Heap Sort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	Unstable
Radix Sort	$O(d \times (n+r))$			$O(r \times n)$	Stable

但是，隨著資訊科技的進步，大數據（Big data）時代來臨，資料也隨之越來越

多，因此外國論文也因此提出了新的概念：資料複雜度，開始討論一組資料的困難程度，那一組資料的好壞是否真的對於執行時間有顯著的差別呢？我們就以一些簡單常見的搜尋法來解釋。

三、資料複雜度

為了驗證資料的好壞對於整體效能的影響，本研究選擇以經典排序法做為實驗，並使用 Python 程式語言及 Pycharm、Spyder 兩種不同編譯器當作實驗環境，用以將實驗成果更客觀，不會因為編譯器而受限，在此之間我們將排序依照內部及外部、穩定與不穩定、是否為原地置換做簡單的介紹。

（一）內部與外部排序：

1. 內部排序（Internal Sort）：

資料筆數少，可以全部放到記憶體中排序，一般的演算法皆為內部排序。

2. 外部排序（External Sort）：

資料筆數大，導致無法放入記憶體中排序，需透過其它儲存裝置輔助，外部排序通常會分次載入部份的資料到記憶體，用內部排序演算法排序後再回存到記憶體或是合併結果。

（二）穩定與不穩定：

1. 穩定（Stable）：

相同鍵值的資料，排序後順序和排序前一樣。

2. 不穩定（Unstable）：

相同鍵值的資料，排序後順序不一定和排序前一樣。

（三）原地置換（In-Place）：

使用資料原來的資料結構(陣列)進行排序，不需使用暫存的輔助資料結構。

（四）經典排序法：

1. 氣泡排序法（Bubble Sort）：

（1）方法：

- a. 先對連在一起未排序的資料兩兩比對掃描。
- b. 兩兩比對時，會將未排序的最大值，藉由交換（swap）移到未排序資料中的右邊。
- c. 直到全部資料最右邊是排序好的最大值，接續排第二大值。

（2）時間複雜度：

a. 最好狀況： $O(n)$

當資料的順序由小到大，第一次執行後不需進行任何交換。

b. 最差狀況： $O(n^2)$

當資料的順序由大到小，每回合執行 $n-1$ 、 $n-2$ 、...、 1 次
 $(n-1) + (n-2) + \dots + 1 = n(n-1)/2 \Rightarrow O(n^2)$

c. 平均狀況： $O(n^2)$

第 n 筆資料，平均比較 $(n-1)/2$ 次， $(n^2-n)/2 \Rightarrow O(n^2)$

(3) 空間複雜度： $\theta(1)$

(4) 穩定性：Stable

(5) 結論：

Inverse data : 1000、999....1

Normal data : 1、2....1000

In-place:

Inverse data's time: 15.14480831916461

Normal data's time: 0.0018248485239791279

Multiplier: 8299.213945791498

(Pycharm)

2. 選擇排序法 (Selection Sort)：

(1) 方法：

- 從第一筆資料，由左往右掃描未排序資料，在從右邊未排序資料中找出最大值(或最小值)。
- 將最大值(或最小值)加入已排序的資料中。
- 重複以上行為，直至排序完成。

(2) 時間複雜度：

- 最好狀況： $O(n^2)$
- 最差狀況： $O(n^2)$
- 平均狀況： $O(n^2)$

(3) 空間複雜度： $\theta(1)$

(4) 穩定性：Stable

(5) 結論：

如果我們在資料上稍做更動，可以發現資料由大到小排會讓排序法效能變差。

Inverse data : 1000、999....1

Normal data : 1、2....1000

In-place:

Inverse data's time: 5.158402364562789

Normal data's time: 4.958648849378733

Multiplier: 1.040283859827881

Extra Space:

Inverse data's time: 1.8611929517726367

Normal data's time: 1.0545978215848208

Multiplier: 1.7648367118525683

(Spyder)

```
In-place:
Inverse data's time: 4.292346548287501
Normal data's time: 3.184321783510967
Multiplier: 1.3479625616085975
Extra Space:
Inverse data's time: 1.7863624572814398
Normal data's time: 0.9782400123242407
Multiplier: 1.8260983345356603
```

Process finished with exit code 0 (Pycharm)

3. 插入排序法 (Insertion Sort) :

(1) 方法 :

- a. 將資料分成已排序、未排序兩部份。
- b. 依序由未排序中的第一筆(正在處理的值)，插入到已排序中的適當位置。
- c. 比較時，若遇到的值比正處理的值大或相等，則將值往右移。

(2) 時間複雜度 :

- a. 最好狀況： $O(1)$
當資料的順序恰好為由小到大時，每回合只需比較 1 次。
- b. 最差狀況： $O(n^2)$
當資料的順序恰好為由大到小時，第 i 回合需比 i 次。
- c. 平均狀況： $O(n^2)$
第 n 筆資料，平均比較 $n/2$ 次。

(3) 空間複雜度： $\theta(1)$

(4) 穩定性：Stable

(5) 結論：

Inverse data : 1000、999...1

Normal data : 1、2...1000

```
In-place:
Inverse data's time: 11.05562257616657
Normal data's time: 0.002595868736079865
Multiplier: 4258.92974575523
```

(Spyder)

```
In-place:
Inverse data's time: 8.312839374439056
Normal data's time: 0.001947184737877805
Multiplier: 4269.158037618476

Process finished with exit code 0 (Pycharm)
```

由上述三種不同排序法，可以發現資料的好壞對於整體程式效能的影響，比想像中還要大，像是在氣泡排序法時，若為較差的資料，會比好的資料慢上 8000 多倍，現在的資料數及資料大小還算很小，如果在更大的資料數，這個差距肯定是非常可觀的。

貳、簡介

一、為何選擇這題目？

隨著資訊科技的進步和網路的發達、電腦運算能力增強，以及資料蒐集與儲存技術持續改進的影響，加速了資料的取得與累積，而這也就是現在市場面臨的一個大方向，大數據(Big Data)。我們可以從龐大的資料中，發掘先前未知且具潛在有用的資訊模型，進而轉化為有價值的資訊或知識，協助決策者做出適當的決策。

然而，要從這麼多的資料中，找出有效的資料並不容易，因此我們需要考量資料複雜度，是否會影響到我們模型的建模，所以我們將資料複雜度設定為我們研究的主軸，再輔以合併工廠經常需要用到的工作排程，讓我們能學以致用，將這些理論一一實現在產業之中。

不僅如此，現今的科技領域當紅的還有工業 4.0 這塊，有些無人工廠已經將排程的演算法運用在分流的部份，我們希望此研究可以幫助工廠在排程時可以避免遇到資料複雜度太高，影響作業流程進行的狀況發生，我們將這些可能因素以程式實驗，讓更多人了解到怎樣的資料樣本會對模型產生較大的影響，以此做為警惕，並希望能將這些因素一一剷除，帶給工廠更高的效能。

參、主題

一、排程(Flow shop)：

(一)定義：

就像車子進場維修一樣每台汽車都要經過清洗、烤漆、鈑金等步驟且順序不可改變，每一個步驟就是一台機器 (machine)，車輛不只一

台，也就是有許多的工作（job），且每台車在不同步驟中的所需時間不同，那在這過程中哪輛車子先開始維修，才可最快完成工作，就是這個 Flow shop 所強調的主題。

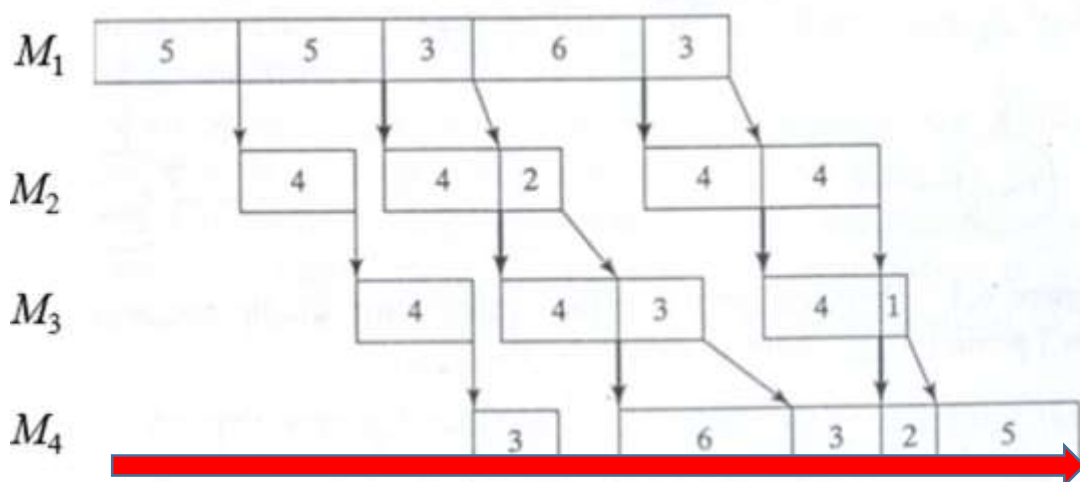
(二) 圖例說明：（工作按照順序，單位：分鐘）

jobs	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

j_k ：表示第 k 個工作

M_i ：表示第 i 台機器

p_{i,j_k} ：表示第 k 個工作在第 i 台機器所需的時間



此圖所需總時間為 34 分鐘（紅色箭頭線長度）

(三) 時間複雜度：

1. 兩台機器時：

限制式較簡單在 1954 年 Johnson's algorithm 中被證明是 $O(n \log n)$ 。

2. 三台機器時：

其複雜度大量提高，此時是一個 NP-hard problem，但是在隨機資料時程是處理速度依然很快速，但是在 3 partition theory 所產生的資料卻處理非常慢，這也就是我們所強調的資料複雜度的

應用。

二、3 partition theory：

(一) 他是一個 NP-hard 問題：

如果我們講「這個問題很難」，這句話可能有兩種不同的意義：

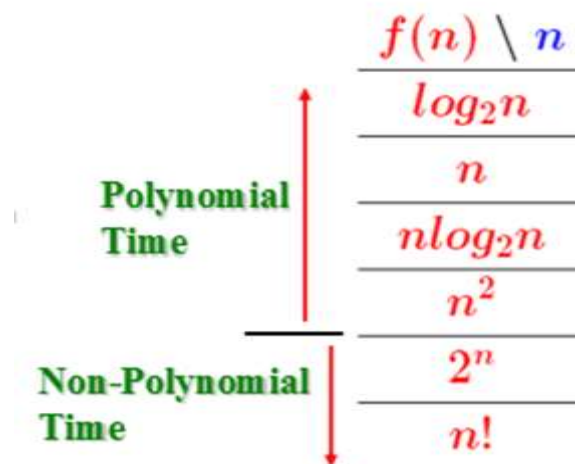
[意義一]

這個問題也許目前已經有一些還不錯的近似解法，只是想進一步找出真正最佳的方法是件困難的事。意思指的是對人而言很困難。

[意義二]

這個問題本身就難以找出解決方法。意思指的是對計算機而言很難。

所以難的問題又可分為下面三種，先介紹何謂多項式時間 (Polynomial Time)：



1. NP (Non-deterministic Polynomial)：

這類的問題只要給個解答，可以在多項式複雜度內很快地驗證出這個解答是否正確。但在未給解答的情況下，無法在多項式複雜度內解決。

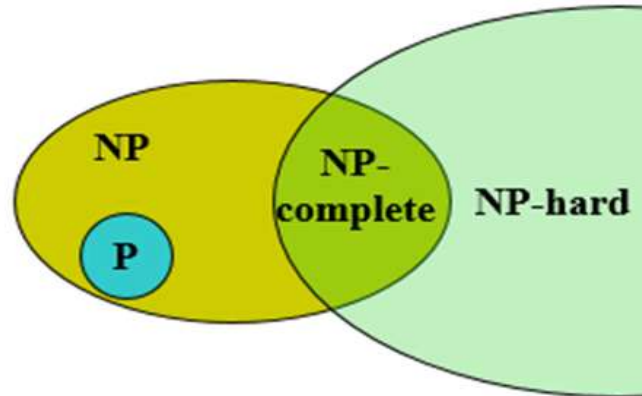
2. NP-hard：

此類問題至今仍未找到一個多項式複雜度的決定性演算法，且一般相信沒有多項式複雜度的決定性演算法存在。

3. NP-Complete：

屬於 NP 且屬於 NP-hard 的問題。

• 一般而言，理論學家相信上述問題之集合圖示如下：



(二) 實驗資料的規則：

- $p_{10} = 0, p_{20} = b, p_{30} = 2b,$
- $p_{1j} = 2b, p_{2j} = b, p_{3j} = 2b, j = 1, \dots, t-1,$
- $p_{1t} = 2b, p_{2t} = b, p_{3t} = 0,$
- $p_{1,t+j} = 0, p_{2,t+j} = a_j, p_{3,t+j} = 0, j = 1, \dots, 3t.$

所以，我們先假設 $t=2, b=30$ 分鐘，代表我現在工作數有 9 個 ($4t+1$)，每一個工作時間規則如下矩陣表示：

$$\begin{bmatrix} 0 & 60 & 60 & 0 & 0 & 0 & 0 & 0 & 0 \\ 30 & 30 & 30 & a_1 & a_2 & a_3 & a_4 & a_5 & a_6 \\ 60 & 60 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

其中的 $a_1 \sim a_3$ 和 $a_4 \sim a_6$ 都是相加為 30 分鐘的 partition 資料。

而此組特殊資料和相同工作量為 9 的隨機資料跑最佳化時所需時間較多，在資料量更大的情況下，電腦所需的處理時間差異會越來越明顯。

三、DEMO：

(一) 介紹限制式與變數：

j ：代表工作

k ：代表工作在第幾個位置

i ：代表機器編號

$p_{i,j}$ ：代表工作 j 在 i 機器中的處理時間

m ：代表機器個數

n ：代表工作個數

$x_{j,k}$ ：是一個二元變數，假如其值是 1 代表工作 j 應放入順序 k

$C_{i,k}$ ：代表在機器 i 中完成第 k 個順序的工作地所需時間

所以我們的目標式如下：（兩個機器的情況下）

$$\min \sum_{k=1}^n C_{2,k}$$

其限制式如下：（兩個機器的情況下）

$$\sum_{j=1}^n x_{j,k} = 1, \quad 1 \leq k \leq n;$$

$$\sum_{k=1}^n x_{j,k} = 1, \quad 1 \leq j \leq n;$$

$$C_{1,1} = \sum_{j=1}^n p_{1,j} x_{j,1};$$

$$C_{1,k+1} = C_{1,k} + \sum_{j=1}^n p_{1,j} x_{j,k+1}, \quad 1 \leq k \leq n-1;$$

$$C_{2,k} \geq C_{1,k} + \sum_{j=1}^n p_{2,j} x_{j,k}, \quad 1 \leq k \leq n;$$

$$C_{2,k+1} \geq C_{2,k} + \sum_{j=1}^n p_{2,j} x_{j,k+1}, \quad 1 \leq k \leq n-1.$$

兩台機器會有六個限制式，三台機器會有八個限制式，以此類推。也因此就是時間加總的限制式，每一台機器一次只能有一個工作，且工作都要按照機器順序進行，這些限制式可以藉由調整工作順序將我們的所需時間做小化。

（二）使用工具：Gurobi

Gurobi 是由美國 Gurobi 公司開發的新一代大規模數學規劃最佳化器，在 Decision Tree for Optimization Software 網站舉行的第三方最佳化器評估中，展示出更快的速度和精度，成為最佳化領域的新翹楚。Gurobi 特點包括以下：

1. 採用最新優化技術，充分利用多核處理器優勢。
2. 任何版本都支持並行計算，並且計算結果確定而非隨機。
3. 提供了方便輕巧的接口，支持 C++, Java, Python, .Net 開發，內存消耗少。
4. 支持多種平台，包括 Windows, Linux, Mac OS X。
5. 支持 AMPL, GAMS, AIMMS, Tomlab 和 Windows Solver Foundation 建模環境。

6. 單一版本，開發版本也就是發布版本，程序轉移便捷。

7. Gurobi 為學校教師和學生提供了免費版本。

(三) 程式實作：

1. 三台機器的實作：

以工作數目有 13 筆當示範，為了評估我們的研究成果是否符合預期，我們也引用隨機資料為本研究之參照組。

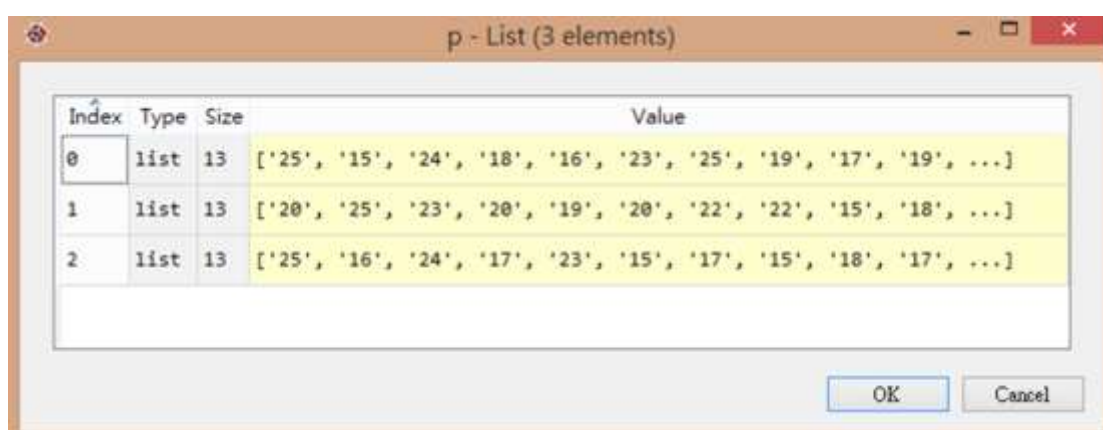
(1) 先創造一個隨機資料和一個 3 partition 資料當作輸入：

• 隨機資料：

```
In [1]: runfile('C:/Users/USER/.spyder-py3/random data.py', wdir='C:/Users/USER/.spyder-py3')
```

輸入j:13

請輸入k(3 machines):3



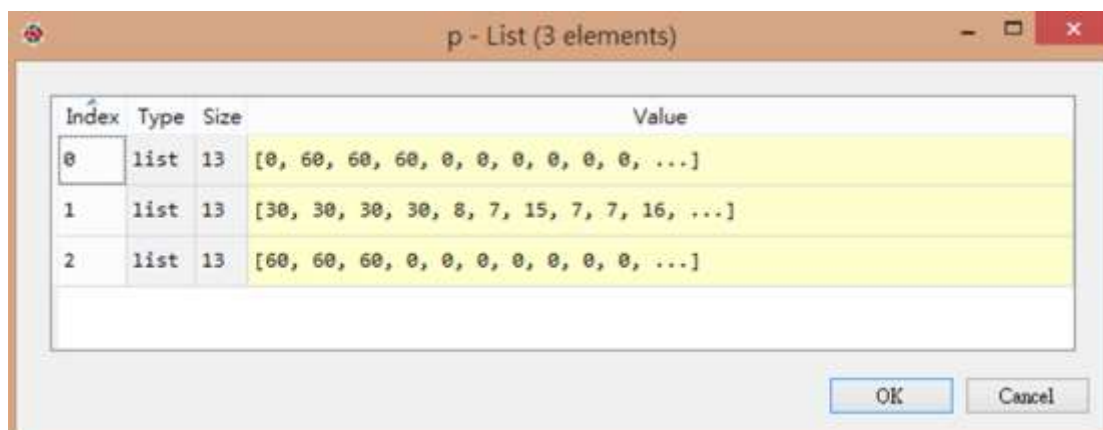
Index	Type	Size	Value
0	list	13	['25', '15', '24', '18', '16', '23', '25', '19', '17', '19', ...]
1	list	13	['20', '25', '23', '20', '19', '20', '22', '22', '15', '18', ...]
2	list	13	['25', '16', '24', '17', '23', '15', '17', '15', '18', '17', ...]

• 3partition 資料：

```
In [3]: runfile('C:/Users/USER/.spyder-py3/3 partition data.py', wdir='C:/Users/USER/.spyder-py3')
```

輸入j(4t+1):13

請輸入k(3 machines):3



Index	Type	Size	Value
0	list	13	[0, 60, 60, 60, 0, 0, 0, 0, 0, 0, ...]
1	list	13	[30, 30, 30, 30, 8, 7, 15, 7, 7, 16, ...]
2	list	13	[60, 60, 60, 0, 0, 0, 0, 0, 0, 0, ...]

(2) 就可以開始計算所需最小時間與電腦所需的處理時間

• 隨機資料的結果：

```
Obj:294
Total: 0.13602304458618164
```

- 3partition 資料的結果：

```
Obj:210
Total: 1.3615334033966064
```

由結果可以得知同樣都是將時間最小化，3partition 資料在電腦中所需的處理時間明顯大很多，這也就是我們資料複雜度的呈現：不只是資料的筆數，資料的結構也可以影響電腦的處理速度非常顯著。

2. 多組資料的處理與統計意義：

為了不使電腦處理時間因為極端狀況而有偏頗，我們也可以一次製造 10 組的 3 partition 資料，並找出其平均與最大最小值，以做出更適當分析結果與比較。

- 輸入：

```
In [3]: runfile('F:/106下/專題/multiple 3 partition data.py', wdir='F:/106下/專題')

輸入j(4t+1):13

請輸入k(3 machines):3

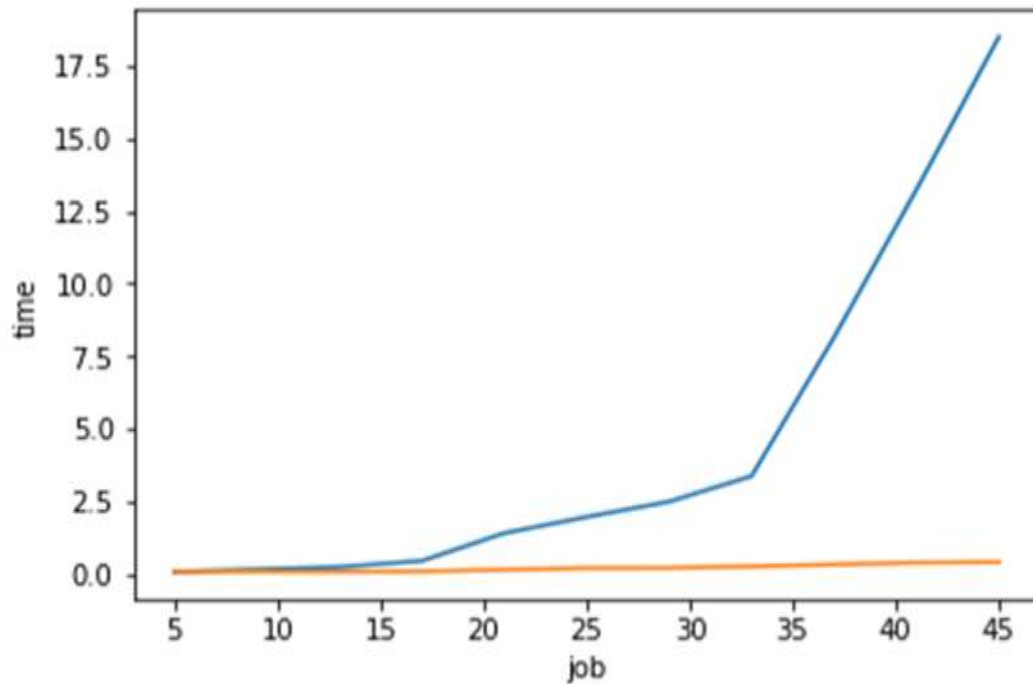
請輸入組數:10
```

- 結果：

```
Obj:210
Total time: [0.1604294776916504, 0.19222664833068848, 0.16909480094909668,
0.2603280544281006, 0.2808830738067627, 0.33405590057373047,
0.23135876655578613, 0.21308493614196777, 0.27077674865722656,
0.3927171230316162]
Max time: 0.3927171230316162
Min time: 0.1604294776916504
Average time: 0.2504955530166626
```

從輸出的結果，我們將跑的次數分別寫入矩陣做儲存，並得到最大、最小和平均時間，而藉由 Gurobi 設定的限制式、目標函數及資料大小(ex:相加 a1~a3 和為 0)，我們得到目標解為 210。

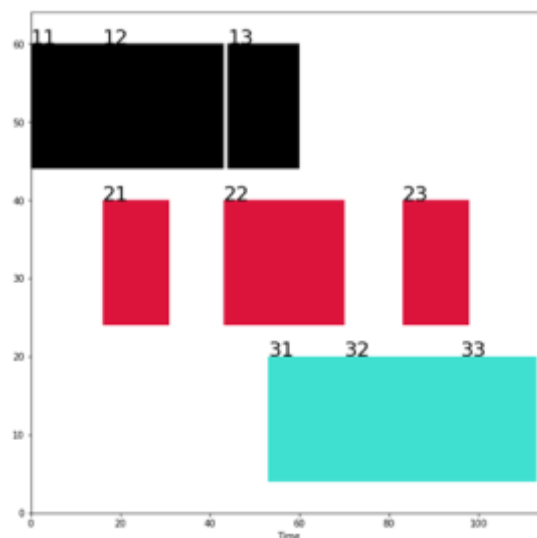
也因此我們將隨機資料和 3 partition 資料在相同工作數下電腦所需處理時間的折線圖如下：（橘線是隨機資料，藍線是 3 partition 資料）



結果我們得到處理時間差異真的非常巨大，更遑論資料有一萬筆時的差異了。

3. 以甘特圖視覺化呈現：

結合 `matplotlib` package，將實驗的工作分配和時間自動畫成甘特圖，由此一來可以方便使用者立即了解資料結果，並能驗證此次實驗是否成功，下圖即為利用 `python` 自動產生之圖表。



4. 四台機器時的狀況：

原理同 3 `partition theory` 中的應用，即所有工作在第四台機器中的工作時間都是等於零，也就是可以以這樣的方式推展到

各種機器中，其電腦所需的處理時間有和三台機器時的 3 partition theory 相似。

p - List (4 elements)

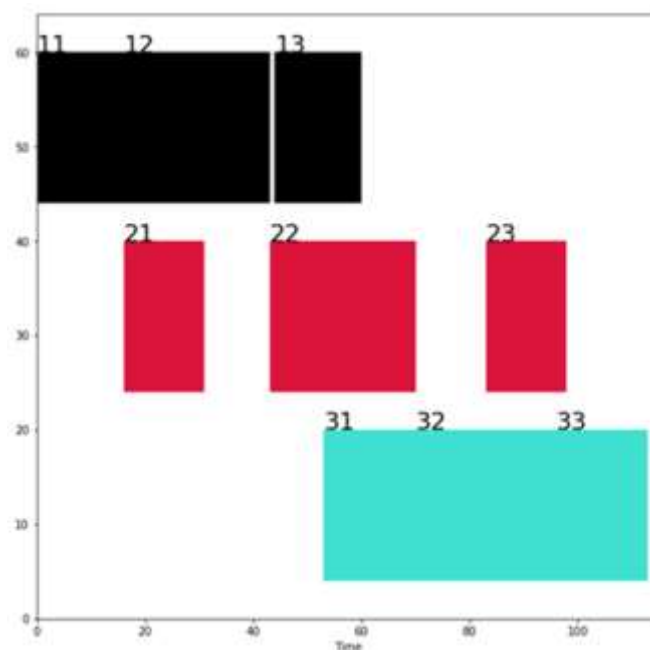
Index	Type	Size	Value
0	list	13	[0, 60, 60, 60, 0, 0, 0, 0, 0, 0, ...]
1	list	13	[30, 30, 30, 30, 7, 9, 14, 9, 7, 14, ...]
2	list	13	[60, 60, 60, 0, 0, 0, 0, 0, 0, 0, ...]
3	list	13	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...]

OK Cancel

5. gurobi 本身設計限制，導致甘特圖錯誤：
由於限制式必須是這樣的形式，才可以找到正確的時間數字，

$$C_{2,k+1} \geq C_{2,k} + \sum_{j=1}^n p_{2,j} x_{j,k+1}, \quad 1 \leq k \leq n-1.$$

但也正因為如此有些特殊結構的工作期完成時間就會因此有所錯誤（以下圖說明）：



此圖中工作 1 在機器 2 中的完成時間是 32（藍箭頭），但卻因為限制式中的大於他的完成時間變成 43（黃箭頭），但是要是將大於去除，又會影響整體最小時間的答案，因此我們也自

已寫了程式可以有效處理此問題，以得到正確的甘特圖，完成更佳的視覺化。

6. 解決方法：

我們撰寫一個調整的程式，將正確的工作順序（gurobi 運算得到），放入我們的程式之中，就可以得到正確的工作完成時間，之後再將開始時間與結束時間，匯入完成甘特圖所需的 excel 檔中，就可以得到正確的甘特圖。

肆、結論及未來研究建議

（一）結論

本研究中，我們得到隨機資料和 3 partition 資料電腦執行處理時間之差異，更遑論資料有多筆資料時的差異了。因此，本研究相信資料複雜度在未來的資料時代將會越來越重要，在大數據時代，數據量龐大已經是家喻戶曉的事情，那接下來下一步要考慮的就是資料的複雜度，這將決定程式進行的效率，也會變成電腦處理速度的探討條件之一。

（二）

經由本研究，未來我們希望能夠加入更多考慮因素，並繼續探討其它 NP – complete 的問題，希望能用程式盡量去逼近接近解，藉而找到近似的答案，證明我們對排程問題之論點，期許能有一天做出對演算法開發領域有貢獻的研究。