程式設計 (112-1) 作業六 – 書面報告

B12705055 張佑丞

October 2023

1 題目架構

根據題目敘述，我們需要做的是找到一組解決方案 x（將 x_{ij} 輛車型為 i 的車分配至場站 j），使得在考慮競食效應下能夠擁有最大化利潤，也就是將以下式子的值最大化

$$\sum_{i\in I}\sum_{j\in J} 24R_i(U_{ij} - \sum_{k\in K} Q^{(k)} \sum_{j'\in N_j^{(k)}} \sum_{i'\in I} x_{i'j'} + Q^{(0)})x_{ij} \tag{1}$$

考慮到題目限制 n 及 m 並不會太大，我們可以試著在每回合遍歷過整個解陣列，逐一檢查並找到在哪個場站投了哪一輛車能讓當前目標式值最大化，然後將其加一後再進行下一輪，直到無論投了哪輛車在哪個站場都無法讓目標式值更大為止。將其寫成 Pseudo Code 後即為:

```
while(true)
    Set when x[l][v] add 1 can optimaize the profit
    Initialize [l,v] = [-1,-1]
    for i from 0 to n-1
        for j from 0 to m-1
            Set x[i][j] add 1
            if Value of x[i][j] is the greatest
                Set [l, v] = [i, j]
            Set x[i][j] deduct 1
    if l is -1
        break the loop
```

而要求出當下的目標值也很簡單，依照上述目標式(1)，可以透過以下 Pseudo Code 實現，其中我們定義 $N_j^{(k)}$ 為與站場 j 距離為 k 的所有站場所形成的集合

```
getVal (to compute the current value of the given solution)

Set N[j][k] = {l}, where Distance between j and l is k
Initialize the return value Ret to 0
for i from 0 to n-1
    for j from 0 to m-1
        Ret add ...
        for k from 0 to 3
            for l in N[j][k]
                for t from 0 to n-1
                    Ret deduct ...
            Ret add ...
Return Ret
```

如此一來解法的基本雛型便已大功告成。接著我們分析其時間複雜度來判斷這個程式碼的運行效率如何，由於每次取得當下目標值都需要遍歷過整個大小為 $N \times M$ 的解陣列 x，雙層迴圈內又需要執行複雜度 $O(NM)$ 的計算，最終 `getVal` 這個函數的複雜度即為兩者相乘也就是 $O(N^2M^2)$ 搭配上前頁所提及的算法，總複雜度即為 $O(N^3M^3)$。

2 優化計算目標值函數

考慮到題目限制 $1 \leq n \leq 20, 1 \leq m \leq 500$，$O(N^3M^3)$ 的計算量大約為 10^{12}，我們能發現這並非一個足夠有效率的解法，我們能夠思考是否有方法能減低其他無謂的運算。而事實上是有的，回想起這個演算法的關鍵「在哪個場站投了哪『一』輛車 ...」，我們可以很好地運用這個特性，將思考從「每投一次就計算一次目標值」變成「在這裡投了一輛車對整體目標值的改變」。依據上述目標式(1)，每新增一輛車在 x_{lv}，在式中會影響的地方即為 x_{lv} 加一，以及其他「與 x_{ij} 有關的 $x_{i'j'}$」[1]加一，而變化後增加或減少的量即為

$$24R_l(U_{lv} - \sum_{k \in K} Q^{(k)} \sum_{j' \in N_v^{(k)}} \sum_{i' \in I} x_{i'j'} + Q^{(0)}) + (-24 \sum_{k \in K} Q^{(k)} \sum_{j' \in N_v^{(k)}} \sum_{i' \in I} R_i x_{i'j'}) \tag{2}$$

而這個式子看似複雜，實則不難理解，簡單討論 x_{ij} 是否等於 x_{lv} 即可得出。由此式，我們接著定義兩個參數 `SumOfVal` 與 `SumOfStation`，其中 $\texttt{SumOfVal}^{(j)} = \sum_{i \in I} R_i \times x[i][j]$，而 $\texttt{SumOfStation}^{(j)} = \sum_{i \in I} x_{ij}$。我們可以發現，只要好好維護好這兩個參數的值，式(2)便

[1]參考式(1)中的 $\sum_{j' \in N_j^{(k)}} \sum_{i' \in I} x_{i'j'}$

可以化簡至非常簡單，如下所示。

$$24R_l(U_{lv} - \sum_{k\in K} Q^{(k)} \sum_{j'\in N_v^{(k)}} \texttt{SumOfStation}^{(j')} + Q^{(0)}) - 24\sum_{k\in K} Q^{(k)} \sum_{j'\in N_v^{(k)}} \texttt{SumOfVal}^{(j')}$$

而要維護這兩個參數也十分簡單，由於每次搜尋後僅增加一輛車，故只要在每次搜尋最後對 `SumOfVal[j]` 和 `SumOfStation[j]` 分別加上 $R[i]$ 與 1 即可，同樣我們可以將此查詢目標值的函數寫成 Pseudo Code，如下

```
FastVal (to compute the curret value in an optimized way)

Set return value Ret to current value before operation
for k from 0 to 3
    for l in S[j][k]
        Ret Deduct SumOfStation[l]...

for k from 0 to 3
    for l in S[j][k]
        Ret Deduct SumOfVal[l]...

Return Ret
```

由此代碼可見，這樣做的時間複雜度能夠大大降低，僅須 $O(M)$ 即可完成一次目標值的查詢，搭配第一頁所提到的 `TA-algorithm` 的算法，我們可以得出總複雜度即為 $O(NM^2)$，計算量大約落在 10^7，可謂是足夠有效率的算法了！而這也確實能在 PDOGS 上順利執行，並得到 15790656 分。

3 將 TA-algorithm 再優化

我們回到 TA-algorithm 的初衷「逐一檢查哪個組合加一後能使目標值最大化」，我們可以思考一個問題「一定要每次都走最大值的路線才能使最終目標值最大化嗎?」，在演算法上我們稱這樣的想法為貪心法 (Greedy)，而貪心法的正確性是需要透過嚴謹的數學證明才能保證的。在這裡我們可以直接嘗試「是否不走當下最大值也能達到甚至超越原本的最大值?」我們不難想像若是暴力解一定無法在指定時限內完成，此時我們需要做一個合理的假設「只有當前目標值與當前最大值在指定誤差範圍內，我們才都納入考慮」，換句話說，若是當前目標值與最大值相差甚遠，未來能「追上」的機率也十分低，故在此我們暫不考慮。接著在實作過程，與上次不同，我採用遞迴 (Recursive) 搭配剪枝 (Pruning) 的技巧

進行，演算法 Pseudo Code 如下

```
solve with given current value before operation

Find max possible value and set it as MX
if MX > Global MX
    Set Global MX = MX
    Save the current solution
for i from 0 to n-1
    for j from 0 to m-1
        if(MX - current value ≤ DiffAcceptable)
            Set x[i][j] add 1
            solve() with current value
            Set x[i][j] deduct 1
```

不過這樣做會有兩項缺點，第一：遞迴時限不易估計，不容易判斷何時能夠使用遞迴，我的解決辦法是只在前二十項小測資執行，只要當前運行時間以超出指定範圍，便跳回原本的算法執行。第二：`DiffAcceptable` 參數[2]需要多次嘗試來找到能使最終目標值達到最大化的參數值。不過就最終結果而言確實是合理且使最大目標值能夠提升的算法，最後我也在 PDOGS 得到了 15815091 的分數，雖然與 `TA-algorithm`[3]的成績相差不大，但卻足以體現這個算法確實有將最大值改善的效果。

4 心得

其實一開始看到這題我是完全沒有任何想法的，不過好險在題目最後一頁有提供 `TA-algorithm` 的算法，讓我有個目標可以朝向，接著開始寫後就是長達三天的陣痛期，完全不知道自己到底錯在哪，這裡我想特別感謝 TA 廷旭！雖然沒有明確點出我的 bug 在哪，但也正面地肯定我的想法，接著就是一連串的 debug，最後竟然是 $D[i][j]$ 沒考慮到距離超過 1000 的 case，而讓那些 case 的距離都和初始化一樣為 0，然後一丟就成功拿到 75 分了 XDD，另外也感謝助教在我和他分享我後面想到的算法時跟我科普了許多關於「meta-heuristic algorithm（元啟發式演算法）」的知識，實在是讓我獲益非常多！未來有機會我也會試著實作看看那些算法的！最後再次謝謝 TA 廷旭和孔老師，這個禮拜我寫程式寫得很充實！

[2]這裡的誤差不侷限於相減的誤差，也可以使用比例上的誤差等

[3]事實上 `TA-algorithm` 已經是一個很不錯的演算法了，能提昇的空間也較有限