柔性計算法與應用 2021Fall

以基因演算法實作一維裝箱問題
Genetic Algorithm for 1D Bin Packing Problem

資管所 R09725060 黃瀚陞

壹、 背景與動機

裝箱問題(Bin Packing)是經典的作業研究問題，不論是在運輸、製造、包裝等場域都有可能會遇到裝箱問題。裝箱問題的應用與延伸也十分廣泛，除了傳統一維的裝箱問題是考量一個因素（像是重量、長度等）外，還有二維的裝箱問題，即是同時考量兩個因素（像是長度與寬度），如皮革的切割或是停車位的劃分。甚至再更困難一點，還有三維的裝箱問題，例如運輸業的貨櫃要怎麼裝載等等。過去在修作業研究時就對於裝箱問題印象深刻，並且知道裝箱問題已經被證明是 NP-Hard 的問題，當時也在作業研究的課堂上學到了一些 Heuristic 的演算法來解裝箱問題。

這學期在柔性計算法這門課上學到了不少 Metaheuristic 的演算法可以用來去解那些數學上的困難問題，得到一個雖然未必最佳但是足夠好的解，因此我希望將這學期學到的這些演算法應用在裝箱問題上面。而在眾多的演算法之中，基因演算法(Genetic Algorithm , GA)擁有良好的求解能力，而且收斂速度快並且演算法的概念易懂且彈性高，因此成為了我此次報告所選擇使用的演算法工具。

不過，因為基因演算法是十分著名的演算法，過去也有許多學者提出不同的編碼或是演化方式來進行裝箱問題的求解。在這篇報告當中，我是參考了 Falkenauer, E.和 Delchambre, A.(1992)以及 Falkenauer, E.(1996)所提出的方式來實作。這兩篇研究首次使用了 group-based 的方式來進行基因演算法的編碼，許多後期在解 grouping 問題的論文都有引用這兩篇研究，相信是非常經典而且有公信力的論文。

綜合以上，此次報告我希望能夠以本學期在課堂上學習到的知識，以基因演算法來實作一維裝箱問題的求解工具。除了求解之外，也盡可能設計友善的介面讓有需要的人可以直接利用我實作的工具來求解裝箱問題。

貳、 研究問題與研究方法

一、研究問題

本次報告要研究的是一維的裝箱問題。一維裝箱問題的定義是：給定一個有限的數字集合 O（物體的尺寸）、常數 B（箱子的大小）以及常數 N（箱子的個數），是否可以將所有的物體都裝進 N 個箱子裡面。也就是說，是否可以在 O 集合裡面找到一個分割方式，可以將 O 分成小於等於 N 個子集，而這些子集中的物體總和不超過 B。

在 Falkenauer, E.和 Delchambre, A.(1992)的研究當中，將裝箱問題化約成一個最佳化問題，並且定義了此問題的 cost function，是為：

$$f_{BPP} = \frac{\sum_{i=1\sim N}(fill_i/C)^k}{N}$$

其中，$fill_i$代表的是在第 i 個箱子當中物體的總尺寸，k 則是一個可以根據情況所調整的參數，作者是建議將 k 設定為 2。使用這個 cost function 就可以同時考慮 N 的大小以及物體的填裝狀況。舉例來說，若是在相同 N 的情況之下，我們會希望每個箱子都被填的越集中在某些箱子越好，如此一來便會有一些箱子剩下來的空間較大，未來也有更多的彈性可以填充其他物體。

在這篇報告當中，我使用了 Falkenauer, E.和 Delchambre, A.所提出的 cost function 作為評估一個裝箱分配的好壞，達到 N 盡可能最小、填裝程度盡可能集中的目標。

二、研究方法

此篇報告使用基因演算法來實作一維的裝箱問題，以下針對基因演算法的各步驟進行說明。

1. Encoding

使用 group-based 的方式來進行編碼。在此種方式編碼下，每一個 gene 代表的是一個箱子(bin)。而每一個 chromosome 則是會被分成兩個部分，object-part 代表的是相對應的物體是被放在哪一個 bin 當中，group-part 代表的 bin 則是會被用來進行基因演算法的執行與操作。舉例來說：ABDBCEB:BECDA 這個 chromosome 代表的是總共用了 ABCDE 五個 bins，其中第一個物體被放在 A 中，第二個物體被放在 B 中，第三個物體被放在 D 中。

2. Crossover

crossover 的操作會進行在染色體的 group-part 上。首先會先針對父染色體與母染色體決定兩個切點，接著將父（母）染色體兩切點中的 gene 插入母（父）染色體的第一個切點位置，並且刪除父（母）染色體中含有新加入物體的 bin。

舉例來說，若父染色體與其切割點為 A|BCD|EF，母染色體則是 ab|cd|，若將母染色體中位於兩切點中的 gene 插入父染色體中的第一個切點位置，則可以得到 AcdBCDEF，若 CEF 三個 Bin 中包含了 cd 中的物體，則刪除 CEF，經過上述步驟，可以得到的子染色體 AcdBD。

此外，刪除了 CEF 三個 Bin 很有可能會造成某些物體一起連帶被刪除，因此還需要再透過 First Fit 的演算法來將遺漏的物體重新放回染色體當中。

3. Mutation

本篇報告實作方法的 mutation 則是相對簡單，只需要在要進行 mutation 的染色體中隨機刪除幾個 gene，並且再利用 First Fit 演算法重新將被刪除的物體隨機放回染色體中（因為是使用 First Fit 演算法，所以會跟原本的位置不一樣，故可完成 mutation）。

4. Inversion

比較特別的是，Falkenauer, E.和 Delchambre, A.有在方法中加入了 inversion 的操作。使用 inversion 是希望可以把填裝比較滿的 bin 擺放在相近的位置。對於裝箱問題來說，交換 bin 的位置並不會影響目標函式值，但如此一來，下一次 crossover 時，這些填裝比較滿的 bin 就比較不容易被拆散，藉此可以讓最終表現更好。

參、 系統實作

一、資料結構

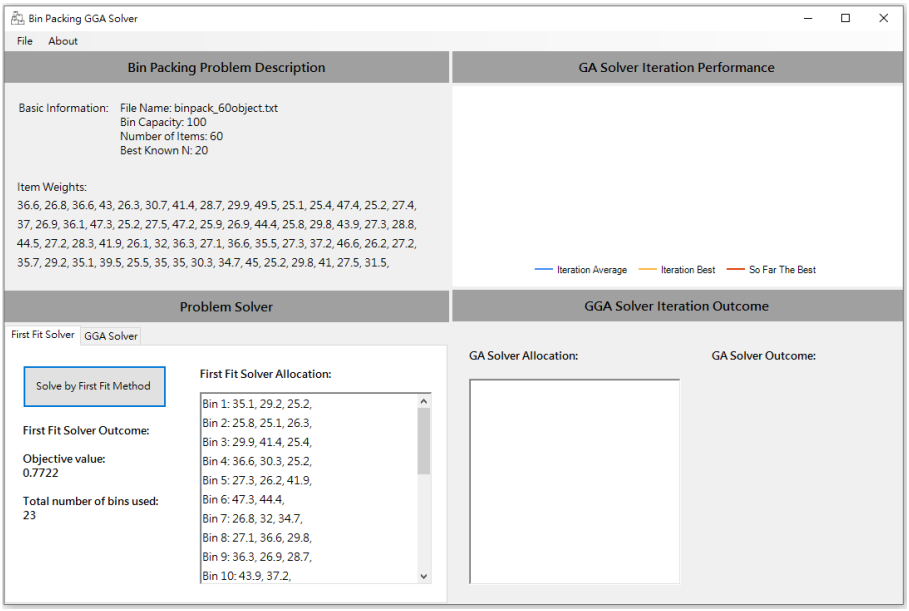
由於是使用 group-based 的方式來對染色體進行編碼，染色體中每一個 gene 代表的就是一個 bin，所以沒辦法像過去一樣使用一個值來代表一個 gene，因此我利用一個 array 來代表一個 gene(bin)，而每個染色體又包含了許多 gene，因此一個染色體就是用一個 array of array 的方式來代表。最後，每一個 iteration 必須要產生一群群體，因此就會是以 array of array of array 來代表。程式當中 double[ ][ ][ ] chromosomes 即是代表每個 iteration 的群體。其餘會使用到的變數皆是如上所述的邏輯，如 soFarTheBestSolution 就會是一個 array of array，objectives 則會是一個 array，以此類推。

除了第貳段所提到的 cost function 會被用來做為評估一個解好壞的依據，每一個解會使用到的 bin 個數也是我所在意的。因此在計算 cost function 中的 N 時也有使用 int[ ] binused 來記錄每個染色體會使用幾個 bin。另外，也用了 double[ ][ ] binAccumulation 來記錄每個染色體中的每個 bin 共容納多少容量，避免過多的重複計算。

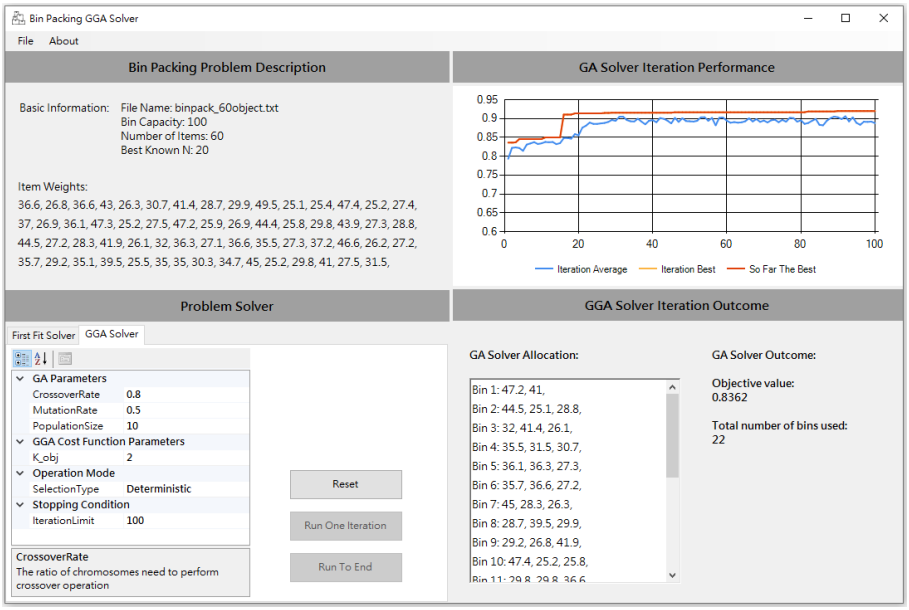
二、演算法步驟

演算法的步驟上，如同前段的研究方法說明，將資料讀入之後會先利用 First Fit 演算法來進行初代群體的產生。接著使用者所設定的 crossoverRate 以及 mutationRate 進行 crossover 和 mutation。其中，進行 mutation 時我有設定被刪除的 gene 數量不可超過染色體中 gene 總數的一半，避免過多的 gene 突變。接著，在決定完優秀的染色體進入下一個 iteration 之前，會先把獲選的優秀染色體進行 invert。由於參考的研究當中並沒有詳細說明要如何進行 invert，因此我的做法先挑出一個染色體中 bin fill 最高的 bin，再從第二到第四高的 bin 中隨機挑選出一個，將選出的兩個 gene 中間的 gene 顛倒過來。例如 BECDA 中是 B 和 D 被選中，最後顛倒的結果就會是 CEBDA。如此一來，便完成了一個 iteration，最後重複以上步驟，直到終止條件滿足。

三、介面設計與系統操作



介面的設計上，使用者可以先使用上方的選單，點選 File，匯入 item 的資料。使用者可以先在左下角中使用 First Fit Solver 得到 First Fit 演算法的計算結果。計算結果除了會顯示需要用到的 Bin 數量、目標函式的值，還會將每個 Bin 應該要怎麼分配也一同呈現。

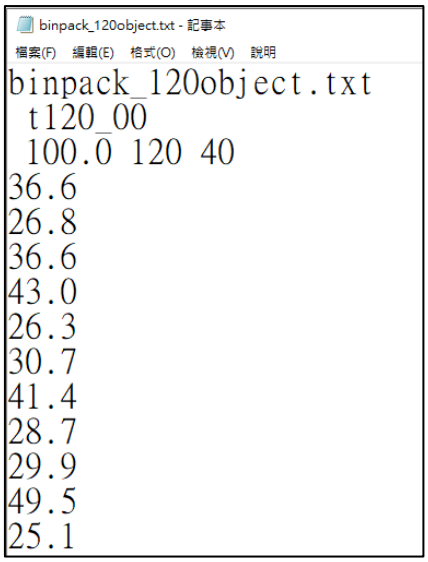


使用者亦可使用 GGA Solver，也就是本報告中的基因演算法求解裝箱問題。使用者可以調整 crossoverRate、mutationRate、PopulationSize、IterationLimit、K_obj（cost function 參數）、SelectionType(Deterministic 或 stochastic)。求解過程中每個 iteration 都會被記錄，iteration 的表現會被繪製成右上的線圖呈現。最後的結果則是呈現於右下方。

肆、 實驗結果

一、標竿問題與實驗資料

除了系統實作之外，在本篇報告當中還進行了演算法的實驗與比較。由於裝箱問題是經典的作業研究問題，亦有許多標竿問題的資源可以做為實驗的資料。本篇報告自 The Association of the European Operational Research Societies 的網站擷取部分裝箱問題的標竿問題，並且整理成統一格式進行實驗。標竿問題檔案的格式說明如下：



其中，第一行與第二行為檔案名稱與原始檔案的識別碼。第三行由左至右依序是每一個 bin 的容量、總共有多少物體需要被裝箱以及目前所知的最佳 bin 數量。第三行後每一行皆代表一個物體的尺寸。上圖中即是代表每一個 bin 容量為 100，總共有 120 個物體要被裝箱，目前已知最佳解是使用 40 個 bin 可以完成裝箱。

二、實驗結果

本次實驗當中會針對 Heuristic 演算法 First Fit Algorithm (FF)，以及上述所提到的 Grouping Genetic Algorithm (GGA)兩種方法進行比較。使用的實驗資料共三種，分別是物體數量為 20、120 與 250 的資料。GGA 的設定部分，crossoverRate 為 0.8、mutationRate 為 0.5、populationSize 為 10，SelectionType 為 Deterministic，IterationLimit 為 100。FF 和 GGA 兩方法皆會針對每份資料進行 20 次實驗，以下為實驗結果與比較。

1. 以平均結果比較

	FF BinUsed	GGA BinUsed	Optimum BinUsed	FF ObjValue	GGA ObjValue
20 Object	10.900	10.150	10	0.736	0.832
120 Object	45.150	44.050	40	0.795	0.833
250 Object	105.250	103.650	99	0.882	0.907

將 20 次的實驗結果取平均，可以看到 GGA 在 Bin 的使用數量上不論是在那種情況皆優於 FF。此外，隨著物體數量變多，GGA 和 FF 兩者在使用的 Bin 的使用數量上也有差距越來越大的趨勢。而在目標函式值的部分，GGA 亦皆優於 FF。

2. 以最佳情況比較

	FF BinUsed	GGA BinUsed	Optimum BinUsed	FF ObjValue	GGA ObjValue
20 Object	10	10	10	0.855	0.856
120 Object	44	43	40	0.837	0.871
250 Object	104	103	99	0.905	0.917

若是取表現最好的情況來比較，可以看到兩演算法在僅 20 個物體時，在 Bin 的使用數量上皆可達到最佳解。不過在 120 個物體與 250 個物體的情況下，兩者皆無法達最佳解。不過，GGA 仍舊比現比 FF 還來得好。

3. 以最糟情況比較

	FF BinUsed	GGA BinUsed	Optimum BinUsed	FF ObjValue	GGA ObjValue
20 Object	11	11	10	0.713	0.731
120 Object	46	45	40	0.767	0.803
250 Object	107	104	99	0.857	0.891

最後，取最糟情況來比較，可以看出兩者在 20 個物體的情況下，最壞的解都是使用 11 個 Bin。不過在物體數量較多的情況下，FF 所得的解皆比 GGA 所得的解來得差。

伍、 結論與討論

一、結論

在此份報告當中，使用了 Falkenauer, E.和 Delchambre, A.所提出的 group-based 的基因演算法(GGA)時做了一維裝箱問題，並且將其與 heuristic 的 First Fit (FF)演算法進行實驗與比較。

系統實作上，使用 C#進行裝箱問題的求解工具開發，使用者可以直接使用此工具進行裝箱問題的求解，求解方法有 FF 與 GGA 兩種演算法，並且可以針對 GGA 的參數進行特殊設定。

就實驗結果來看，GGA 的表現優於 FF 演算法，不過若是再比較 GGA 與最佳解，可以看出 GGA 在物體數量較多的問題上，仍舊無法達到最佳解，甚至距離最佳解還有點距離。

由上述可知，GGA 雖然比起 heuristic 的 FF 演算法，有效提升了裝箱問題的表現，但仍舊有不少改善的空間。

二、未來展望

因為此份報告參考的是比較早期提出使用 group-based 的基因演算法求解裝箱問題，後期也有不少研究者提出此方法的改良版，因此未來亦可朝改良的 GGA 演算法進行研究。

另外，本篇報告實作的是一維的裝箱問題，實際上裝箱問題亦有考慮更多因素的二維與三維裝箱問題，這些延伸問題考慮的因素更多，因此也更加困難，更難使用最佳化的方式來求解，如何將 GGA 的演算法擴展到延伸的裝箱問題，或許會有更多不一樣的操作手法。

陸、 參考資料

[1] Falkenauer, E., & Delchambre, A. (1992, May). A genetic algorithm for bin packing and line balancing. In ICRA (pp. 1186-1192).

[2] Falkenauer, E. (1996). A hybrid grouping genetic algorithm for bin packing. Journal of heuristics, 2(1), 5-30.

[3] Hartmanis, Juris. "Computers and intractability: a guide to the theory of np-completeness (michael r. garey and david s. johnson)." Siam Review 24.1 (1982): 90.

[4] The Association of the European Operational Research Societies. (n.d.). Data sets – ESICUP – euro special interest group on cutting and packing. Retrieved January 16, 2022, from https://www.euro-online.org/websites/esicup/data-sets/#1535972134887-953d6528-5b82

附錄、實驗詳細結果

一、First Fit Solver 實驗結果

	20_Obj	20_Bin	120_Obj	120_Bin	250_Obj	250_Bin
1	0.7318	11	0.8047	45	0.8694	106
2	0.8532	10	0.7688	46	0.8862	105
3	0.7221	11	0.8029	45	0.8839	105
4	0.7289	11	0.7981	45	0.8697	106
5	0.7183	11	0.7969	45	0.8836	105
6	0.716	11	0.7705	46	0.9049	104
7	0.7232	11	0.7692	46	0.8839	105
8	0.7286	11	0.8003	45	0.8867	105
9	0.7135	11	0.8326	44	0.8871	105
10	0.7196	11	0.7957	45	0.8867	105
11	0.7251	11	0.7958	45	0.8575	107
12	0.7299	11	0.8016	45	0.9018	104
13	0.8546	10	0.7689	46	0.9008	104
14	0.7299	11	0.8373	44	0.8712	106
15	0.7303	11	0.8312	44	0.8702	106
16	0.7321	11	0.7987	45	0.8846	105
17	0.7162	11	0.7672	46	0.8834	105
18	0.7134	11	0.7962	45	0.8882	105
19	0.7161	11	0.7967	45	0.8855	105
20	0.7213	11	0.7707	46	0.8568	107
Average	0.736205	10.9	0.7952	45.15	0.882105	105.25
Best	0.8546	10	0.8373	44	0.9049	104
Worst	0.7134	11	0.7672	46	0.8568	107

二、GGA Solver 實驗結果

	20_Obj	20_Bin	120_Obj	120_Bin	250_Obj	250_Bin
1	0.8541	10	0.8323	44	0.9171	103
2	0.854	10	0.8334	44	0.9016	104
3	0.855	10	0.8317	44	0.9025	104
4	0.8547	10	0.8307	44	0.901	104
5	0.737	11	0.8329	44	0.9033	104
6	0.8527	10	0.8316	44	0.9018	104
7	0.8542	10	0.8353	44	0.904	104
8	0.7706	10	0.8348	44	0.9169	103
9	0.8552	10	0.8041	45	0.8909	104
10	0.8532	10	0.836	44	0.9007	104
11	0.8543	10	0.8313	44	0.9169	103
12	0.8554	10	0.8047	45	0.9002	104
13	0.854	10	0.8025	45	0.9168	103
14	0.856	10	0.8377	44	0.9013	104
15	0.7344	11	0.8711	43	0.903	104
16	0.8529	10	0.8393	44	0.9169	103
17	0.7305	11	0.833	44	0.9012	104
18	0.8541	10	0.8363	44	0.9007	104
19	0.8534	10	0.833	44	0.9168	103
20	0.8556	10	0.8704	43	0.9174	103
Average	0.832065	10.15	0.833105	44.05	0.90655	103.65
Best	0.856	10	0.8711	43	0.9174	103
Worst	0.7305	11	0.8025	45	0.8909	104