Tên: Trương Văn Khải

MSSV: 21520274

Lớp: CS106.N21 (Môn: Trí tuệ nhân tạo)

GVHD: Dr. Lương Ngọc Hoàng

BÁO CÁO BÀI TẬP 2:

SOLVING KNAPSACK PROBLEM VỚI GOOGLE OR TOOLS

Yêu cầu:

1) Solving knapsack with OR-Tools: https://developers.google.com/optimization/bin/knapsack

2) Knapsack test instances: https://github.com/likr/kplib

Có tổng cộng 13 nhóm test cases (00-12) cho bài toán knapsack trong link [2]. Trong mỗi nhóm, chúng ta cần chọn ra ít nhất 5 test cases (càng nhiều test cases càng tốt) có kích thước khác nhau (ví dụ 50 items, 100 items, 200 items, 500 items, 1000 items,...), và giải các test cases này bằng OR Tools như sau:

1. Chọn một mốc chi phí tính toán phù hợp với máy tính của mỗi bạn (ví dụ tối đa 3 phút cho mỗi lần chạy).

2. Thiết lập thực nghiệm sao cho OR Tools sẽ dừng khi mà thời gian tính toán cho mỗi lần chạy đã sử dụng hết. (Tham khảo set_time_limit tại: https://developers.google.com/optimization/reference/algorithms/knapsack_solver/KnapsackSolver)

3. Lưu lại kết quả của mỗi lần chạy mỗi test case. Lời giải tìm ra có phải là lời giải tối ưu của test case đó hay không?

4. Lập bảng thống kê: Tên của mỗi test case, giá trị của lời giải, tổng trọng lượng các items trong lời giải, lời giải tìm ra có phải là tối ưu hay không?

5. Dựa vào kết quả thống kê, kết luận trong 13 nhóm test cases, nhóm nào là dễ và nhóm nào là khó?

```
Knapsack_ORTool/
├── __pycache__/
├── Dataset/
│   └── kplib-master/
│       ├── 00Uncorrelated/
│       ├── 01WeaklyCorrelated/
│       ├── 02StronglyCorrelated/
│       ├── 03InverseStronglyCorrelated/
│       ├── 04AlmostStronglyCorrelated/
│       ├── 05SubsetSum/
│       ├── 06UncorrelatedWithSimilarWeights/
│       ├── 07SpannerUncorrelated/
│       ├── 08SpannerWeaklyCorrelated/
│       ├── 09SpannerStronglyCorrelated/
│       ├── 10MultipleStronglyCorrelated/
│       ├── 11ProfitCeiling/
│       └── 12Circle/
├── Deap_knapsack.py                    :xây dựng code cho thuật toán Tiến hóa (Genetic Algorithm).
├── GoogleORTools.py                    :xây dựng code sử dụng công cụ Google Or Tool.
├── knapsack.py                         :xây dựng lớp Knapsack01Problem và các hàm để hỗ trợ code Genetic Algorithm.
├── main.py                             :xây dựng hàm thực thi và xuất tệp dữ liệu kết quả dưới dạng .csv.
├── Result.csv                          :tệp dữ liệu kết quả được lưu dưới dạng .csv.
├── testcasePrepocessing.py             :xây dựng hàm trả về một danh sách các đường dẫn của các tệp testcase cần kiểm tra.
├── [21520274]Knapsack-Assignment.docx  :tệp báo cáo cho bài Knapsack-Assignment lưu dưới dạng docx.
└── [21520274]Knapsack-Assignment.pdf   :tệp báo cáo cho bài Knapsack-Assignment lưu dưới dạng pdf.
```

Fig1: Sơ đồ tổ chức thư mục

1. TỔNG QUAN

1.1. KNAPSACK PROBLEMS LÀ GÌ ?

- Trong Knapsack problems, người chơi sẽ được giao cho 1 bộ những vật thể với giá trị và thể tích được biết trước. Nếu mà tổng lượng thể tích của vật thể lớn hơn thể tích của balo thì balo sẽ không thể nào chứa hết các vật thể đó. Do đó, người chơi sẽ phải bỏ lại một số vật thể lại. Bài toán ba lô là bài toán yêu cầu người chơi phải bỏ những vật thể vào balo sao cho tối ưu nhất được sức chứa của cái ba lô.

Fig2: Knapsack problems

2. MỤC TIÊU BÀI BÁO CÁO

- Phân tích các thuật toán dành cho bài toán "Knapsack Problems" và đề xuất sử dụng hai hướng giải quyết đó là: Sử dụng Google OR Tool và Evolution Algorithm từ DEAP.
- Chạy thực nghiệm tool Google OR Tool sử dụng kĩ thuật nhánh và cận Brand and Bounds.
- Chạy thuật nghiệm thuật toán Evolution Algorithm từ thư viện DEAP để cố gắng đạt được kết quả tối ưu.
- Phân tích kết quả số liệu thu thập được từ hai cách chạy trên và đưa ra được bản kết quả.

3. THIẾT LẬP BÀI TOÁN

3.1. Trích xuất dữ liệu:

- Do có rất nhiều TestCase để kiểm tra, em không thể nào chạy qua hết tất cả các trường hợp nên em sẽ chọn chiến lược là: Lấy 5 bộ n = [50, 100, 200, 500, 1000]. Với mỗi bộ thì em sẽ chỉ lấy 1 file duy nhất để kiểm tra đó là file s000.kp trong thư mục R01000.
- Cách trích xuất dữ liệu được em xây dựng trong file **testcaseProcessing.py** như sau:

testcasePrepocessing.py > ...

```python
import os

def Get_All_TestCases():
    # n=50,100,200,500,1000
    # R=1000
    #Choose these testcase
    temp_name1=['n00050', 'n00100', 'n00200', 'n00500', 'n01000']
    temp_name2=['R01000']
    path='.\\Dataset\\kplib-master\\'
    temp_dir=os.listdir(r'.\Dataset\\kplib-master')
    temp_dir.sort()
    name_dir=[]
    for i in range(0,13):
        name_dir.append(temp_dir[i])
    All_TestCase=[]
    for name in name_dir:
        for name1 in temp_name1:
            for name2 in temp_name2:
                temp_name3 = os.listdir(path+name+'\\'+name1+'\\'+name2)
                All_TestCase.append(name+'\\'+name1+'\\'+name2+'\\'+temp_name3[0]) #chose testcase s000.kp
    return All_TestCase
```

Fig3: testProcessing Code

- Trong đó **Hàm Get_All_TestCases()** trả về một danh sách các chuỗi biểu diễn tên (đường dẫn) của các tệp Testcase cần kiểm tra.

3.2. Google OR Tool

- Để giải quyết bài toán này thì em sử dụng dụng thuật toán Branch and Bounds (nhánh và cận) được tích hợp trong công cụ Google Or Tool. Code được em xây dựng cho thuật toán này được lưu ở file **GoogleORTools.py.**
- Để xác định được thuật toán đó có tối ưu hay không thì chúng ta dựa vào thời gian chạy của thuật toán trên test case đó. Nếu thời gian chạy trên test case đó lớn hơn time_limit thì thuật

toán đó không tối ưu. Trường hợp ngược lại thì nó tối ưu. Trong bài này, em chọn **time_limit = 120.0 giây**.

- Em có nhận thấy được có những Testcase là $\Sigma\, weight[i] \leq capacites$. Vì vậy, có sử dụng vài phép biến đổi để tối ưu thuật toán hơn bằng cách:
  $\boldsymbol{capacites = \min(\Sigma\, weight[i], capacites)}$
- Thuật toán Brand and Bounds được tích hợp trong Google Or Tool sẽ giải quyết bài toán với độ phức tạp là O (n x $\Sigma\, weight[i]$) .

3.3. Genetic Algorithm

- Em sử dụng mã nguồn của giải thuật Tiến Hóa (Evolution Algorithm) từ framework DEAP với một vài thay đổi phù hợp với mục đích chạy thực nghiệm của mình.
- Em chọn các hằng số tiến hóa như sau:

  P_CROSSOVER = 0.9
  P_MUTATION = 0.1
  MAX_GENERATIONS = 100
  HALL_OF_FAME_SIZE = 1
- Bài này em sử dụng thuật toán Tiến Hóa để tối đa hóa giá trị của Knapsack Problem. Thuật toán này sẽ được thực thi cho đến khi số thế hệ đã đạt đến giới hạn tối đa được chỉ định trong biến MAX_GENERATIONS.
- Để xác định được thuật toán đó có tối ưu hay không thì chúng ta dựa vào thời gian chạy của thuật toán trên testcase đó. Nếu thời gian chạy trên test case đó lớn hơn time_limit thì thuật toán đó không tối ưu. Trường hợp ngược lại thì nó tối ưu. Trong bài này, em chọn **time_limit = 120.0 giây**.

Đầu ra kết quả của cả hai thuật toán **Genetic Algorithm** và **Google Or Tool** đều được lưu trong file **Result.csv**. File **Result.csv** được chương trình xuất bảng dữ liệu tệp **.csv** và được xây dựng trong tệp main.py.

```
def main():
    columns_name = ['Case','Total_Value_Deap','Total_Weight_Deap','Is_TimeLimit_Deap','Total_Value_GGoT','Total_Weight_GGoT','Is_TimeLimit_GGoT']
    deap_knapsack = dks.solve_deap()
    ggortool = got.solve_ggorTool()
    #Results are lists of information
    num_case=len(ggortool)
    data=[]
    #Concanating corresponding lists
    for row in range(num_case):
        data.append(deap_knapsack[row] + ggortool[row])
    df=pd.DataFrame(data,columns = columns_name)
    #Result to csv
    df.to_csv('Result.csv')
```

Fig4: Code main.py

"**columns_name**" là một list chứa tên các cột được sử dụng trong bảng kết quả **Result.csv.** Các thành phần trong "**columns_name**" có ý nghĩa như sau:

- 'case': Tên testcase (đường dẫn dữ liệu).
- 'Total_Value_Deap': Tổng giá trị các món đồ đã chọn sử dụng thuật toán GA từ DEAP.
- 'Total_Weight_Deap': Tổng trọng lượng các món đồ đã chọn sử dụng thuật toán GA từ DEAP.
- 'Is_TimeLimit_Deap: Thời gian chạy thuật toán GA từ DEAP có quá timelimt hay không.
- 'Total_Value_GGoT': Tổng giá trị các món đồ đã chọn sử dụng công cụ Google Or Tool.
- 'Total_Weight_ GGoT': Tổng trọng lượng các món đồ đã chọn sử dụng công cụ Google Or Tool.
- 'Is_TimeLimit_GGoT': Thời gian sử dụng công cụ Google Or Tool có quá timelimt hay không.

4. SO SÁNH HIỆU SUẤT GIỮA GENETIC ALGORITHM VÀ GOOGLE OR TOOL

4.1. Tổng quát

- Em cho cả hai thuật toán đều chạy trên Laptop của em vì vậy nó có cùng nền tảng và cùng bộ nhớ máy tính để thực thi, tránh sự bất công.
- Thuật toán di truyền (Genetic Algorithm) và OR-Tools đều là các phương pháp tối ưu hóa sử dụng trong các bài toán tổ hợp, bao gồm bài toán cái túi (Knapsack Problem).
- Cấu hình máy tính em sử dụng để chạy các thuật toán:

```
System Model: Katana GF66 11UC
        BIOS: E1582IMS.304
   Processor: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz (16 CPUs), ~2.3GHz
      Memory: 8192MB RAM
   Page file: 13543MB used, 5006MB available
DirectX Version: DirectX 12
```

4.2. Chi tiết

- Việc xử lí các Testcase đối với 2 giải thuật trên, em có thể chia input thành các dạng dữ liệu như sau:
  - Dữ liệu **dễ** xử lí.
  - Dữ liệu **khó** xử lí.

Việc phân tích dữ liệu này là dữ liệu **dễ** hay **khó** thì có thể phụ thuộc vào nhiều yếu tố. Tuy nhiên, phương pháp để xác định rằng loại dữ liệu này là khó hay dễ ở bài báo cáo của em sẽ dựa vào **số lượng testcase trung bình bị limited.** Tức là, nếu nhóm dữ liệu đó có **số lượng testcase bị limited** lớn hơn **giá trị trung bình** thì nhóm dữ liệu đó sẽ được phân nhóm là **dữ liệu khó xử lí**, nếu ngược lại thì sẽ được phân loại thành **dữ liệu dễ xử lí**.

4.3. Bảng dữ liệu kết quả chạy

- Như đã nêu ở phần đầu, bộ dữ liệu được đưa vào để chạy là một bộ dữ liệu tốt. Tuy nhiên, trong bộ dữ liệu có quá nhiều testcase cho nên em không thể chạy hết được. Cho nên, em chọn chiến lược là mỗi mục em chỉ lấy item đầu tiên (**s000.kp**) cho bộ dữ liệu với **n = ['n00050', 'n00100', 'n00200', 'n00500', 'n01000']** của **'R01000'**. Tức là tổng cộng sẽ có **13 * 5 = 65 testcase** cho cả hai thuật toán **Genetic Algorithm** and **Knapsack Using Google OR Tool**.

	Case	Total_Value_Deap	Total_Weight_Deap	Is_TimeLimit_Deap	Total_Value_GGoT	Total_Weight_GGoT	Is_TimeLimit_GGoT
0	00Uncorrelated_n00050_R01000_s000.kp	19632	14743	TRUE	20995	14721	TRUE
1	00Uncorrelated_n00100_R01000_s000.kp	45568	22530	TRUE	46537	22519	TRUE
2	00Uncorrelated_n00200_R01000_s000.kp	82571	50257	TRUE	84317	50302	TRUE
3	00Uncorrelated_n00500_R01000_s000.kp	199735	118628	TRUE	207992	118693	TRUE
4	00Uncorrelated_n01000_R01000_s000.kp	370533	252379	TRUE	400811	252480	TRUE
5	01WeaklyCorrelated_n00050_R01000_s000.kp	15574	14229	TRUE	15768	14232	TRUE
6	01WeaklyCorrelated_n00100_R01000_s000.kp	30504	28988	TRUE	31064	29013	TRUE
7	01WeaklyCorrelated_n00200_R01000_s000.kp	55926	51542	TRUE	56976	51563	TRUE
8	01WeaklyCorrelated_n00500_R01000_s000.kp	137440	127266	TRUE	139258	127276	TRUE
9	01WeaklyCorrelated_n01000_R01000_s000.kp	268355	245937	TRUE	273052	245972	TRUE
10	02StronglyCorrelated_n00050_R01000_s000.kp	17429	14229	TRUE	17539	14239	TRUE
11	02StronglyCorrelated_n00100_R01000_s000.kp	35216	29016	TRUE	35617	29017	TRUE
12	02StronglyCorrelated_n00200_R01000_s000.kp	64961	51561	TRUE	65363	51563	FALSE
13	02StronglyCorrelated_n00500_R01000_s000.kp	160678	127278	TRUE	162178	127278	FALSE
14	02StronglyCorrelated_n01000_R01000_s000.kp	312464	245964	TRUE	316372	245972	FALSE
15	03InverseStronglyCorrelated_n00050_R01000_s000.kp	14752	16652	TRUE	14914	16714	TRUE
16	03InverseStronglyCorrelated_n00100_R01000_s000.kp	30067	33967	TRUE	30468	33968	TRUE
17	03InverseStronglyCorrelated_n00200_R01000_s000.kp	54359	61459	TRUE	54964	61464	TRUE
18	03InverseStronglyCorrelated_n00500_R01000_s000.kp	135124	152024	TRUE	136031	152031	FALSE
19	03InverseStronglyCorrelated_n01000_R01000_s000.kp	260217	295417	TRUE	263977	295477	FALSE
20	04AlmostStronglyCorrelated_n00050_R01000_s000.kp	17326	14221	TRUE	17556	14238	TRUE
21	04AlmostStronglyCorrelated_n00100_R01000_s000.kp	35212	29016	TRUE	35611	29016	TRUE
22	04AlmostStronglyCorrelated_n00200_R01000_s000.kp	64779	51563	TRUE	65385	51563	FALSE
23	04AlmostStronglyCorrelated_n00500_R01000_s000.kp	160565	127278	TRUE	162154	127278	FALSE
24	04AlmostStronglyCorrelated_n01000_R01000_s000.kp	312379	245913	TRUE	316415	245972	TRUE
25	05SubsetSum_n00050_R01000_s000.kp	14239	14239	TRUE	14239	14239	TRUE
26	05SubsetSum_n00100_R01000_s000.kp	29017	29017	TRUE	29017	29017	TRUE
27	05SubsetSum_n00200_R01000_s000.kp	51563	51563	TRUE	51563	51563	TRUE

28	05SubsetSum_n00500_R01000_s000.kp	127278	127278	TRUE	127278	127278	TRUE
29	05SubsetSum_n01000_R01000_s000.kp	245972	245972	TRUE	245972	245972	TRUE
30	06UncorrelatedWithSimilarWeights_n00050_R01000_s000.kp	19220	2401484	TRUE	19676	2401482	TRUE
31	06UncorrelatedWithSimilarWeights_n00100_R01000_s000.kp	39216	4902313	TRUE	39791	4902253	TRUE
32	06UncorrelatedWithSimilarWeights_n00200_R01000_s000.kp	73565	9904482	TRUE	75678	9904900	TRUE
33	06UncorrelatedWithSimilarWeights_n00500_R01000_s000.kp	182941	24711869	TRUE	189769	24712055	FALSE
34	06UncorrelatedWithSimilarWeights_n01000_R01000_s000.kp	347123	49525408	TRUE	371246	49525319	TRUE
35	07SpannerUncorrelated_n00050_R01000_s000.kp	13472	4569	TRUE	13472	4569	TRUE
36	07SpannerUncorrelated_n00100_R01000_s000.kp	23616	8750	TRUE	24228	8748	FALSE
37	07SpannerUncorrelated_n00200_R01000_s000.kp	47552	17264	TRUE	47836	17274	FALSE
38	07SpannerUncorrelated_n00500_R01000_s000.kp	113384	42887	TRUE	114616	42898	FALSE
39	07SpannerUncorrelated_n01000_R01000_s000.kp	225332	84649	TRUE	228624	84656	FALSE
40	08SpannerWeaklyCorrelated_n00050_R01000_s000.kp	10354	11452	TRUE	10354	11452	TRUE
41	08SpannerWeaklyCorrelated_n00100_R01000_s000.kp	20080	20816	TRUE	20550	20824	TRUE
42	08SpannerWeaklyCorrelated_n00200_R01000_s000.kp	39524	41128	TRUE	40575	41116	TRUE
43	08SpannerWeaklyCorrelated_n00500_R01000_s000.kp	97946	100048	TRUE	98713	100076	FALSE
44	08SpannerWeaklyCorrelated_n01000_R01000_s000.kp	191536	198632	TRUE	196050	198664	FALSE
45	09SpannerStronglyCorrelated_n00050_R01000_s000.kp	28344	11544	TRUE	28440	11540	FALSE
46	09SpannerStronglyCorrelated_n00100_R01000_s000.kp	51152	20952	TRUE	51656	20956	TRUE
47	09SpannerStronglyCorrelated_n00200_R01000_s000.kp	100888	41288	TRUE	101888	41288	FALSE
48	09SpannerStronglyCorrelated_n00500_R01000_s000.kp	244296	99996	TRUE	245128	99928	FALSE
49	09SpannerStronglyCorrelated_n01000_R01000_s000.kp	485200	198800	TRUE	488672	198772	FALSE
50	10MultipleStronglyCorrelated_n00050_R01000_s000.kp	20837	14237	TRUE	21338	14238	TRUE
51	10MultipleStronglyCorrelated_n00100_R01000_s000.kp	42814	29014	TRUE	43316	29016	TRUE
52	10MultipleStronglyCorrelated_n00200_R01000_s000.kp	80349	51549	TRUE	81658	51558	TRUE
53	10MultipleStronglyCorrelated_n00500_R01000_s000.kp	200543	127243	TRUE	203778	127278	FALSE
54	10MultipleStronglyCorrelated_n01000_R01000_s000.kp	390056	245956	TRUE	399170	245970	FALSE
55	11ProfitCeiling_n00050_R01000_s000.kp	14220	14239	TRUE	14229	14238	TRUE
56	11ProfitCeiling_n00100_R01000_s000.kp	28980	29017	TRUE	29001	29015	FALSE
57	11ProfitCeiling_n00200_R01000_s000.kp	51504	51563	TRUE	51540	51562	FALSE
58	11ProfitCeiling_n00500_R01000_s000.kp	127164	127277	TRUE	127239	127277	FALSE
59	11ProfitCeiling_n01000_R01000_s000.kp	245769	245971	TRUE	245877	245972	FALSE
60	12Circle_n00050_R01000_s000.kp	300026	14239	TRUE	300031	14239	TRUE
61	12Circle_n00100_R01000_s000.kp	611404	29017	TRUE	611418	29017	TRUE
62	12Circle_n00200_R01000_s000.kp	1086463	51563	TRUE	1086483	51563	FALSE
63	12Circle_n00500_R01000_s000.kp	2681819	127278	TRUE	2681868	127278	FALSE
64	12Circle_n01000_R01000_s000.kp	5182784	245972	TRUE	5182856	245972	TRUE

Fig5: Bảng thực thi 65 testcase đã chọn cho cả 2 thuật toán

- Các ô có giá trị FALSE được tô màu vàng là những testcase đó mà solver đã thực thi quá thời gian timelimt là 120.0 giây để giải. **Nhưng việc có tối ưu chưa thi chưa thể xác định được**.

Total Testcase	65		
Runtime Limit Testcase using GE	65	Runtime Limit Testcase using GG Or Tool	39
Optimal Testcase using GE	0	Optimal Testcase using GG Or Tool	26

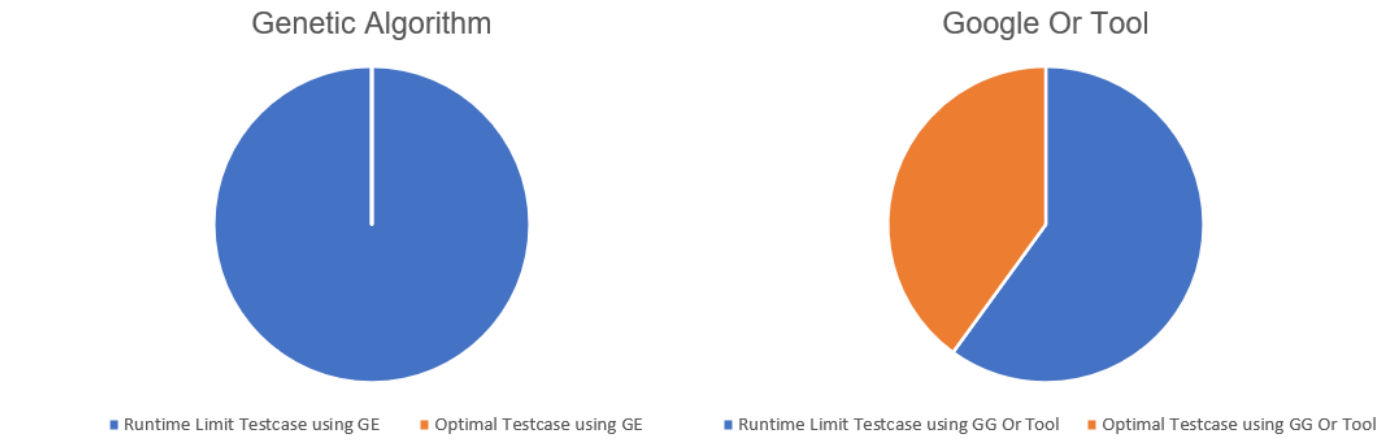

Fig6: Biểu đồ thể hiện tỉ lệ số testcase mà solver cho lời giải tối ưu và giải quá timelimit

4.4. Nhận xét

- Vì **Thuật toán Di Truyền** (**Genetic Algorithm**) cho biểu hiện quá tốt cho các testcase em đã chọn (cả 65 testcase đều cho ra kết quả là **True**, tức là kết quả tối ưu) nên ta có thể kết luận sơ bộ là **Thuật toán Di Truyền GE** đã cho ra kết quả vượt trội hơn **Google Or Tool về thời gian chạy**.
- Để phân loại các thuật toán khó và dễ, ta chỉ có thể dựa trên phương pháp sử dụng **Google Or Tool** để phân loại (Vì **Genetic Algorithm** đã cho kết quả quá tốt):
    - Cho từng nhóm dữ liệu (**00Uncorrelated** đến **12Circle**): Số testcase cho ra kết quả là **False** lần lượt là: **runtime_test** = [0, 0, 3, 2, 2, 0, 1, 4, 2, 4, 2, 4, 2]

- **AVERAGE**(**runtime_test**) = 13. Vậy giá trị ngưỡng để để ta có thể xem xét nhóm dữ liệu đó là khó hay dễ sẽ là **ceiling_value** = 2. Nếu $\mathbf{X_i}$ thuộc **runtime_test** > **ceiling_value** thì ta có thể phân nhóm dữ liệu đó sẽ là **dữ liệu khó**, ngược lại ta sẽ phân nhóm dữ liệu đó sẽ là **dữ liệu dễ**.
- Vậy ta có thể kết luận nhóm **dữ liệu khó** sẽ là: **02**StronglyCorrelated, **07**SpannerUncorrelated, **09**SpannerStronglyCorrelated, **11**ProfitCeiling.
- Ta cũng có thể kết luận nhóm **dữ liệu dễ** sẽ là: **00**Uncorrelated, **01**WeaklyCorrelated, **03**InverseStronglyCorrelated, **04**AlmostStronglyCorrelated, **05**SubsetSum, **06**UncorrelatedWithSimilarWeights, **08**SpannerWeaklyCorrelated, **10**MultipleStronglyCorrelated, **12**Circle.

➢ Tuy nhiên, vẫn có rất nhiều ưu điểm và nhược điểm mà cả **Thuật toán Di Truyền** (**Genetic Algorithm**) và **Google Or Tool** cùng nhau giải quyết từng **testcase**. Vì vậy, chúng ta không thể kết luận cuối cùng rằng **phương pháp nào sẽ tốt hơn phương pháp nào**. Mà điều chúng ta cần tìm ra sẽ nên là **Thuật toán Di Truyền** (**Genetic Algorithm**) hay **Google Or Tool sẽ phù hợp hơn để sử dụng cho từng testcase đặc biệt.**