ĐẠI HỌC BÁCH KHOA HÀ NỘI
TRƯỜNG CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

———————— * ————————

BÁO CÁO MÔN HỌC
TỐI ƯU LẬP KẾ HOẠCH

ĐỀ TÀI: BIN PACKING WITH LOWER AND UPPER BOUND CAPACITY CONSTRAINT

Sinh viên thực hiện: **Nguyễn Long Nhật - 20215440**
Nguyễn Bá Minh Đạt - 20215342
Nguyễn Đức Nghĩa - 20204674

Giảng viên hướng dẫn: TS. Bùi Quốc Trung

HÀ NỘI 06 - 2024

I. Giới thiệu bài toán

Bài toán xoay quanh các đơn hàng, mỗi đơn hàng có chi phí để vận chuyển và giá trị cùng nhiều phương tiện. Mỗi phương tiện chỉ có thể vận chuyển được tối đa một số lượng nhất định đơn hàng, quyết định bởi cận trên tải trọng của phương tiện và tổng chi phí các đơn hàng được chọn. Hơn nữa, mỗi phương tiện chỉ được xuất phát khi đã chở được một số tối thiểu đơn hàng, được quyết định tương tự với cận dưới của tải trọng. Ta muốn sắp xếp các đơn hàng vào các phương tiện sao cho tối đa hóa giá trị các đơn hàng được vận chuyển.
Ta có thể miêu tả cụ thể bài toán như sau:
+ Input bao gồm
- Số đơn hàng N
- Số phương tiện K
- Trọng lượng và giá trị của mỗi đơn hàng i: c_i và p_i
- Cận trên và cận dưới tải trọng của phương tiện j: u_j và l_j

+ Output: Một cách sắp xếp các đơn hàng vào K phương tiện, thỏa mãn cận trên và cận dưới tải trọng mỗi phương tiện sao cho tổng giá trị của chúng là lớn nhất.

II. Các phương pháp giải bài toán

1. Solver MIP và CP của thư viện Or-tools

Mô hình hoá bài toán

Biến:
X[i, j] = 0 nếu xe i không chứa kiện hàng j $\quad \forall\, i \in [1, K],\ j \in [1, N]$
X[i, j] = 1 nếu xe i chứa kiện hàng j $\quad \forall\, i \in [1, K],\ j \in [1, N]$

Ràng buộc:

$$l_i \leq \sum_{j=1}^{N} X[i,j] * c_j \leq u_i \quad \forall\, i \in [1, K]$$

$$\sum_{j=1}^{N} X[i,j] \leq 1 \quad \forall\, i \in [1, K]$$

Hàm mục tiêu:

$$Max: \sum_{i=1}^{K} \sum_{j=1}^{N} X[i,j] * p_i$$

Áp dụng mô hình hoá ở trên, sử dụng OR-tools của Google để thêm các biến, ràng buộc và hàm mục tiêu để chạy ra kết quả. MIP và CP chỉ khác nhau ở phần khai model và thêm ràng buộc (chi tiết ở phần code demo đi kèm)

Kết quả so sánh cho thấy CP và MIP (mô tả kỹ ở phần thực nghiệm) đều cho ra kết quả như nhau tuy nhiên CP có thời gian chạy là nhanh hơn so với MIP

2. Thuật toán nhánh cận áp dụng cận về tính "khả thi"

Ý tưởng của thuật toán nhánh cận là sử dụng một hàm đặc biệt để đánh giá độ tốt của cây hiện tại. Từ ý tưởng đó, ta có thể thêm một số điều kiện bắt buộc dừng quá trình tìm kiếm trong cây tìm kiếm lời giải hiện tại.
Đầu tiên ta chú ý đến ràng buộc về tải trọng của xe. Ta biết rằng trong một lời giải hợp lệ, tất cả ràng buộc về tải trọng đều phải được thỏa mãn.
Ta cố gắng tìm cách sắp xếp hợp lệ các đơn hàng vào **từng xe một** theo thứ tự đã định sẵn bằng cách duyệt danh sách các đơn hàng. Tức là ta giải bài toán xếp hàng vào xe 1 với tập đơn hàng có sẵn, với mỗi lời giải ta bỏ bớt các đơn hàng đã xếp khỏi tập đơn và tiếp tục giải với xe 2. Ta đánh giá lời giải khi đã xếp hết k xe.
Bằng việc sắp xếp danh sách đơn hàng theo thứ tự c_i tăng dần, ta biết chắc chắn không tồn tại khả năng sắp xếp thỏa mãn các đơn hàng hiện tại vào phương tiện k nếu ta duyệt phải đơn hàng i mà:

$$accum + c_i > u_j,$$

$$với\ accum\ là\ tổng\ trọng\ lượng\ của\ các\ đơn\ hàng\ trong\ xe\ k\ hiện\ tại$$

Ta xét các xe theo thứ tự giảm dần của độ rộng khoảng cận dưới – cận trên, sau đó đến thứ tự giảm dần giá trị của cận dưới.
Sau khi hoàn thành gán đơn hàng cho một xe bất kỳ, nếu tổng trọng lượng các đơn hàng còn lại nhỏ hơn tổng cận dưới tải trọng của các xe chưa gán thì hiển nhiên nhánh lời giải hiện tại không thể đi tiếp được nữa.

$$remain < \sum_{i=1}^{k} (1 - \tau_i) l_i$$

$$với\ \tau_i = 1\ nghĩa\ là\ đã\ gán\ các\ đơn\ hàng\ thành\ công\ cho\ xe\ i; 0\ nếu\ ngược\ lại$$

Ngoài ra, một điều kiện hiển nhiên để dừng toàn bộ hàm tìm kiếm là khi đã xếp được tất cả đơn hàng lên xe:

$$best = \sum_{i=1}^{n} p_i$$

$$với\ best\ là\ giá\ trị\ hàm\ mục\ tiêu\ của\ lời\ giải\ tốt\ nhất\ tính\ đến\ hiện\ tại$$

3. Thuật toán nhánh cận dựa trên giá trị lớn nhất có thể thêm được

Thuật toán áp dụng ý tưởng về việc lấy sắp xếp các c_i chưa được xếp theo thứ tự tăng dần, tương tự với các p_i còn lại. Sau đó ta sẽ duyệt các xe sao cho lượng tải trọng có thể thêm cho mỗi xe là ít nhất và lượng giá trị được thêm là cao nhất. Từ đó ta sẽ thu được lượng giá trị tốt nhất có thể của những trường hợp còn lại. Nếu lượng giá trị này nhỏ hơn bài toán hiện tại, ta có thể cắt nhánh tại đây.

4. Greedy (Thuật toán tham lam 1)

Ý tưởng thuật toán:
Ưu tiên xếp từng đơn hàng vào phương tiện với các tiêu chí:

- Đối với mỗi đơn hàng i, ưu tiên xếp đơn hàng vào phương tiện với các chỉ số của tiêu chí giảm dần và thứ tự ưu tiên các tiêu chí như sau:
  - Hiệu quả (efficiency) = v_i / c_i
  - Giá trị v_i
  - Chi phí c_i
- Đối với mỗi phương tiện j, ưu tiên chọn phương tiện với các chỉ số của tiêu chí tăng dần và thứ tự ưu tiên các tiêu chí như sau:
  - Cận dưới của tải trọng l_j
  - Cận trên của tải trọng u_j
  - Sự chênh lệch (difference) = u_j - l_j

Quy trình thực hiện của thuật toán:

Sắp xếp và phân chia đơn hàng:

- Sắp xếp các đơn hàng dựa trên tiêu chí ưu tiên
- Sắp xếp các phương tiện dựa trên tiêu chí ưu tiên

Phân chia đơn hàng vào phương tiện:
Thuật toán sẽ thử tất cả các tổ hợp ưu tiên đơn hàng và phương tiện theo thứ tự các tiêu chí ưu tiên để tìm ra cách phân chia hợp lý sao cho thoả mãn các ràng buộc và tối đa hoá được tổng giá trị đơn hàng

- Đầu tiên, phân chia các đơn hàng để đảm bảo các phương tiện thỏa mãn điều kiện dung lượng thấp nhất.
- Sau đó, kiểm tra lại nếu có phương tiện nào chưa thỏa mãn điều kiện tải trọng thấp nhất, thuật toán sẽ đưa ra thông báo lỗi đồng nghĩa với không có tổ hợp ghép cặp ưu tiên đơn hàng và phương tiện nào thoả mãn
- Sau khi kiểm tra, nếu có tổ hợp ghép cặp ưu tiên đơn hàng và phương tiện thoả mãn các ràng buộc thì cuối cùng, phân chia các đơn hàng còn lại vào các phương tiện sao cho không vượt quá dung lượng cao nhất.

Tính toán kết quả:

- Sau khi phân chia xong, tính toán tổng giá trị của các đơn hàng đã phục vụ và số lượng đơn hàng được phục vụ.

Nhận xét:
- Thuật toán chạy khá nhanh và kết quả cũng khá tốt, tuy nhiên chiến lược tham lam này có thể sẽ không tìm ra lời giải với 1 số test case do không thoả mãn điều kiện ràng buộc cận dưới (nghĩa là khi xếp đơn hàng vào thêm thì sẽ vượt quá cận trên, mà không xếp thêm đơn hàng vào thì chưa đạt đủ cận dưới)
- Hướng cải tiến: Cải tiến hoặc tìm hiểu chiến lược tham lam khác tổng quát hơn

5. Thuật toán tham lam 2

Thuật toán tham lam thứ hai có ý tưởng rất đơn giản đó chính là ta sẽ sắp xếp các đơn hàng theo giá trị giảm dần về giá trị (value), nếu có value bằng nhau thì ta sẽ ưu tiên đơn hàng có chi phí (cost) nhỏ hơn trước.

Chiến lược sắp xếp đơn hàng vào phương tiện:

- Duyệt qua các đơn hàng theo thứ tự đã sắp xếp và xếp lên các phương tiện lần lượt cho tới khi phương tiện không thể chứa thêm đơn hàng nào nữa (đạt tối đa tải trọng – upper bound) thì sẽ chuyển qua xếp các đơn hàng lên các phương tiện khác

- Sau khi đã xếp hết các đơn hàng, duyệt qua các phương tiện để kiểm tra về điều kiện ràng buộc về cập dưới – cận trên của phương tiện, nếu có phương tiện nào thoả mãn về điều kiện này thì ta thêm vào không gian lời giải bài toán. Ngược lại nếu tồn tại phương tiện nào không thoả mãn ràng buộc tải trọng nằm trong đoạn từ lower bound – upper bound này thì ta sẽ không thêm vào không gian lời giải bài toán và đương nhiên ta cũng sẽ bỏ qua các đơn hàng được xếp lên các phương tiện này.

→ Có thể thấy với cách tham lam này khá đơn giản và thoả mãn được nhiều test case (hầu hết), tuy nhiên với chiến lược tham lam này thì chỉ có thể tìm ra được lời giải chấp nhận được, giá trị hàm mục tiêu chưa được tối ưu.

Mô hình lời giải mới

Mô hình lời giải mới ,không sử dụng các biến nhị phân, được xây dựng như sau:
Mỗi lời giải là một danh sách asssignments nhiều phần tử cấu trúc Assignment đặc biệt:
Struct Assignment:

- int orderID
- int vehicleID

Trong đó orderID đại diện cho đơn hàng và vehicleID đại diện cho xe hàng mà đơn hàng đó được xếp vào. Quy ước vehicleID = 0 nghĩa là đơn hàng không được xếp vào bất cứ xe nào.
Vậy assignments sẽ là một danh sách gồm n phần tử Assignment với n là số đơn hàng nhập vào. Thật ra đây chính là biến thể của mô hình biến sau:

$$X = \{x_1, x_2, \dots, x_n\}$$

$$x_i = j \; nghĩa \; là \; đơn \; hàng \; i \; được \; xếp \; vào \; xe \; j$$

Vậy muốn hiểu được ý nghĩa của lời giải này, hay "giải mã" được nó, ta phải "đọc" các phần tử trong danh sách và xếp đơn hàng vào xe tương ứng. Ta quy ước chiều đọc là chiều từ trái qua phải của danh sách.

6. Ý tưởng của hàm sửa chữa xuất phát từ kết quả sau khi sắp xếp các đơn hàng vào thuật toán tham lam:

+ Tồn tại một số xe có tổng chi phí các đơn hàng vượt quá cận dưới của tải trọng: xe loại 1
+ Tồn tại một số xe có tổng chi phí các đơn hàng thấp hơn cận dưới của tải trọng: xe loại 2
Do đó chúng em đề xuất sử dụng một hàm sửa chữa để "luân chuyển" phần đơn hàng dư ra từ các xe loại 1 sang xe loại 2. Sau khi hoàn thành giai đoạn sắp xếp bằng thuật toán tham lam tương tự thuật toán tham lam thứ 1, ta định nghĩa với mỗi phương tiện:

$o_j = w_j - l_j$ *với* w_j *là tổng chi phí của các đơn hàng trong xe* j

$s_j = u_j - w_j$

Vậy rõ ràng phương tiện loại 1 sẽ có $o_{loại\ 1} > 0$ và ngược lại với phương tiện loại 2.

Function Repair():

For 2nd type Vehicles v2 do:

Calculate s_{v2}

For 1st type Vehicles v1 do :

Calculate o_{v1}

For Order o in v1 that has cost < o_{v1} and do:

If o.cost < Min(o_{v1},s_{v2})

Assign(o,v1)

Remove(o,v2)

ChangeAssignment(o,v1,v2)

$o_{v1} -= o.cost$

$s_{v2} -=$ o.cost

End If

End for

End for

End for

Trong đó hàm Assign() và Remove() có tác dụng thử gán thêm một đơn hàng vào một phương tiện và loại bỏ một đơn hàng khỏi một phương tiện. Hàm ChangeAssignment(o,v1,v2) có tác dụng thay đổi vehicleID của assignment tương ứng với đơn hàng o từ v2 sang v1.

7. Thuật toán tham lam thứ 3

Thuật toán tham lam thứ ba chính là thuật toán tham lam thứ nhất có sự thay đổi trong tiêu chí ưu tiên và áp dụng thêm hàm sửa chữa theo thứ tự tham lam. Cụ thể:

+ Tiêu chí sắp xếp các đơn hàng i là dựa theo thứ tự giảm dần của các p_i, nếu p_i bằng nhau thì theo thứ tự tăng dần của c_i

+ Tiêu chí sắp xếp các xe j là theo thứ tự tăng dần của giá trị $u_j - l_j$

Trong hàm Repair(): các phương tiện loại 1 sẽ được xét theo thứ tự giảm dần của o_j và thứ tự giảm dần của số đơn hàng trong phương tiện nếu o_j bằng nhau.

Các phương tiện loại 2 sẽ được xét theo thứ tự giảm dần của s_j; nếu s_j bằng nhau thì xét theo thứ tự giảm dần của $-o_j$

Ta có thể hiểu giai đoạn một của thuật toán như việc vừa dựng vừa đọc một danh sách assignments với các Assignment được theo tiêu chuẩn của đơn hàng.

8. Hàm tìm kiếm cục bộ ngẫu nhiên

Ý tưởng tìm kiếm cục bộ xuất phát từ một lời giải ngẫu nhiên. Lời giải ngẫu nhiên đó tuân theo format của mô hình lời giải đã được đề cập ở mục 3 và được khởi tạo như sau:

+ Với mỗi Assignment thì vehicleID ← random(K); hàm random trả lại một giá trị nguyên ngẫu nhiên trong đoạn [0;K].

Hàm tìm kiếm cục bộ cần thêm một danh sách để liên tục kiểm tra trạng thái của các phương tiện vehicles: mỗi phần tử trong vehicles có cấu trúc như sau:

Struct Vehicle{

int lowB (chính là l_j)

int upB (chính là u_j)

int cur (chính là w_j)

int profit (chính là tổng giá trị của toàn bộ đơn hàng có trong xe tính đến hiện tại)

List<Order> (lưu giữ toàn bộ đơn hàng được gán vào xe)

}

Hàm tìm kiếm cục bộ ngẫu nhiên gồm 3 giai đoạn:

- Giai đoạn 1: Duyệt từ đầu đến cuối chuỗi assignments hiện tại và gán mỗi đơn hàng vào xe tương ứng. Nếu không gán được thì đánh dấu lại
- Giai đoạn 2: Duyệt từ đầu đến cuối chuỗi assignments hiện tại và gán mỗi đơn hàng chưa gán được vào một phương tiện ngẫu nhiên
- Giai đoạn 3: Kích hoạt hàm Repair() ngẫu nhiên để sửa lời giải.

Hàm có thể được miêu tả bằng mã giả như sau:

```
Function RandomLocalSearch(List assignments){
    Shuffle assignments (1)
    //The 1st Assigning round
    For Assignment in assignments do:
        If vehicleID != 0 do:
            If !Assign(orderID,vehicleID) do
                vehicleID = 0
            End if
        End if
    End for
    Shuffle assignments (2)
    Shuffle vehicles (3)
    //The 2nd Assigning Round
    For Assignment in assignments do:
        If vehicleID == 0 do:
            For Vehicle v in vehicles do:
                If Assign(orderID, v.vehicleID) do
                    vehicleID ← v.vehicleID
                    break
                End if
            End For
        End if
    End for
    Classify() (4)
//Hàm classify() được sử dụng để phân chia các phương tiện thành loại 1 và loại 2 và shuffle 2 danh sách đó
    Repair()
}
```

Hàm Assign(orderID, vehicleID) sẽ gán đơn hàng orderID vào phương tiện orderID. Nếu gán được; tức

$$cur_{vehicleID} + c_{orderID} \leq u_{vehicleID}$$

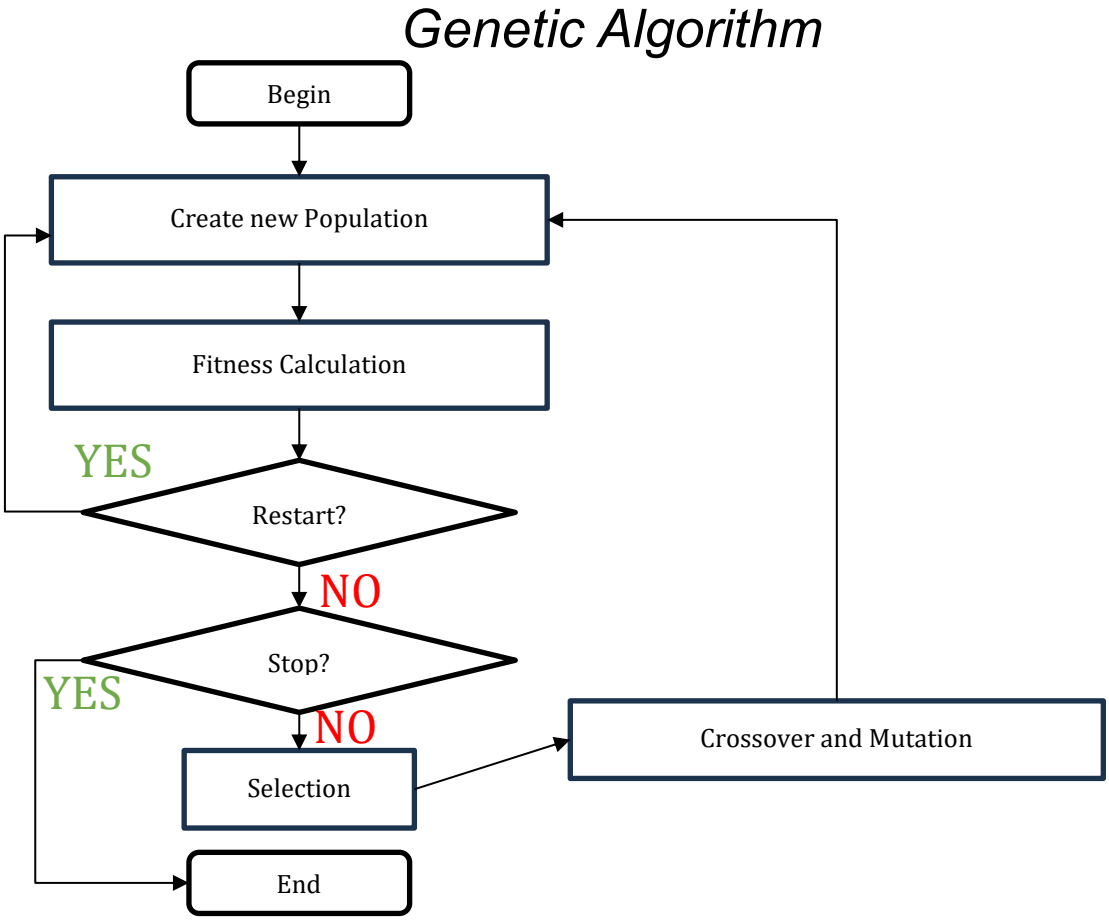
Thì các thông số của xe vehicleID được cập nhật với đơn hàng orderID và hàm return true. Ngược lại hàm trả lại false.

Repair() chính là hàm sửa chữa được đề cập đến ở mục 4.

Tính ngẫu nhiên của hàm được đảm bảo bởi các bước trộn ngẫu nhiên danh sách đơn hàng và xe ở các bước (1),(2),(3) và (4).
Hàm tìm kiếm cục bộ này có tác dụng cải thiện một lời giải (không nhất thiết phải khả thi) hoàn chỉnh với hy vọng tìm được một hàng xóm tốt hơn. 2 giai đoạn đầu tiên cố gắng đưa lời giải về dạng thỏa mãn ràng buộc cận trên tải trọng còn giai đoạn cuối tập trung vào cận dưới tải trọng hơn.

9. Repeated Random Local Search và Genetic Algorithm

Thuật toán di truyền GA có sơ đồ như sau:



Thuật toán bắt đầu bằng việc khởi tạo một quần thể, tức là một tập hợp các cấu trúc đặc biệt trong đó có chứa mô hình lời giải ở phần 2. Mỗi lời giải chính là một danh sách assignments được khởi tạo ngẫu nhiên như phần 6. Sau đó mỗi lời giải được biến đổi bằng hàm tìm kiếm cục bộ ngẫu nhiên và được tính giá trị fitness. Bước khởi tạo này chính là thuật toán Repeated Local Search với mỗi điểm bắt đầu là một lời giải vừa mới khởi tạo.
Vậy ta có thể thực hiện thuật toán Repeated Local Search bằng cách liên tục khởi tạo quần thể lời giải mới.
Ở thuật toán GA mỗi lời giải được đánh giá bằng một hàm fitness, trong trường hợp này hàm fitness chính là tổng các p_i của các đơn hàng đã được gán.
Dựa trên kết quả thực nghiệm thì ở đa số lần kiểm tra thuật toán GA thì ngay sau bước khởi tạo này điểm tối ưu toàn cục của lời giải đã được tìm thấy. Do đó, nếu muốn kiểm tra tính hiệu quả cụ thể của thuật toán GA thì ta cần nhiều test cases khác. Các test cases hiện tại chỉ tập trung vào tính Feasibility của lời giải chứ chưa hướng đến Optimization.

Một test case hướng đến Optimization có thể cần có các đặc điểm sau:
+ Chắc chắn phải có đơn hàng không được xếp vào xe trong mọi trường hợp sắp xếp. Điều này có thể được thể hiện qua tổng $\sum_{i=1}^{N} c_i$ > tổng $\sum_{j=1}^{K} u_j$
+ Giá trị $u_j - l_j$ phải lớn hơn các test cases sẵn có, để tăng số lượng lời giải khả thi.
Có được các test cases như vậy thì các bước tiếp theo sau đây của thuật toán di truyền có thể phát huy tác dụng:
+ Bước Selection: sẽ chọn ra một số lời giải dựa trên hàm fitness từ quần thể lời giải hiện tại để tiến hành lai tạo. Hiện tại bước selection sử dụng hàm BinaryTournament(). Hàm này sẽ ngẫu nhiên chọn ra 2 lời giải để "so đấu" (Tournament) và chọn lời giải có giá trị hàm fitness cao hơn. Lặp lại quy trình đó đến khi đã chọn đủ số lượng lời giải
+ Bước Crossover và Mutation: Là bước lai tạo và thực hiện đột biến để tạo ra quần thể lời giải mới.

- Crossover: Lai tạo 2 lời giải đã có để tạo nên một lời giải mới. Cụ thể là mỗi Assignment trong danh sách assignments của lời giải con có thể nhận giá trị từ Assignment ở vị trí tương ứng trong danh sách của lời giải cha hoặc mẹ.
- Mutation: Bước đột biến ngẫu nhiên hoán đổi giá trị VehicleID của 2 Assignment trong danh sách hiện tại.

Mọi lời giải mới được sinh ra từ bước này đều được sử dụng hàm tìm kiếm cục bộ ngẫu nhiên để tối ưu và tính giá trị fitness.
+ Tiêu chí Restart: khởi tạo lại một quần thể mới khi giá trị fitness cao nhất của tất cả các lời giải tìm được đến hiện tại không có sự cải thiện sau một số vòng lặp tối đa. Khi đó toàn bộ quần thể hiện bị bỏ đi và một quần thể mới được khởi tạo ngẫu nhiên từ ban đầu.
+ Tiêu chí Stop: Dừng thuật toán khi đã đạt đến số vòng lặp tối đa.
Do đặc điểm kết quả của các Test cases hiện tại nên chúng em chỉ kiểm tra hiệu quả của giải thuật Repeated Random Local Search. Các lời giải được sinh ngẫu nhiên và cải thiện bằng hàm Local Search liên tục đến khi tìm được lời giải tối ưu toàn cục. Do tính ngẫu nhiên trong khởi tạo lời giải nên với mỗi test case chúng em kiểm tra 10 lần.

III. Kết quả thực nghiệm và đánh giá

Trong bảng kết quả thử nghiệm dưới đây, kết quả của giải thuật Repeated Local Search là kết quả tốt nhất 10 lần thử nghiệm do sự ngẫu nhiên trong thuật toán, với mỗi lần thực hiện lặp lại tìm kiếm cục bộ 10000 lần. Kết quả được in đỏ nếu lời giải là lời giải tối ưu toàn cục. Những lời giải khác, trừ lời giải sử dụng thư viện or-tools, chỉ được chạy trong tối đa 30s.

	Jury Solution	CP		MIP		Tham lam – 1	Tham lam -2
		Objective	Time(s)	Objective	Time(s)	Objective	Objective
N = 5, K = 2	27	27	0.0472	27	0.0404	27	27
N = 10, K = 2	57	57	0.0404	57	0.0500	57	57
N = 50, K = 3	257	257	0.0440	257	0.0462	257	257
N = 100, K = 5	1065	1139	0.1050	1139	0.0526	1139	1063
N = 200, K = 10	2037	2102	0.3041	2102	1.1309		2049
N = 300, K = 10	2971	3033	0.2399	3033	1.4996	3033	3002
N = 500, K = 50	5109	5227	3.2490	5227	379.8108	5226	5193
N = 800, K = 80	8536	8559	22.0531	8559	3440.9531		8429
N = 900, K = 90	9395	9526	101.3593	9526	1060.7996		9362
N = 1000, K = 100	10368	10380	19.6820	N/A	N/A	10379	10255

	Jury Solution	Tham lam - 3	Repeated Local Search	Feasibility “Branch&Bound”	Branch&Bound
N = 5, K = 2	27	27	27	27	27
N = 10, K = 2	57	57	57	57	57
N = 50, K = 3	257	257	257	257	257
N = 100, K = 5	1065		1139	1139	
N = 200, K = 10	2037	2102	2102		
N = 300, K = 10	2971		3033		
N = 500, K = 50	5109	5227	5227		
N = 800, K = 80	8536		8559		
N = 900, K = 90	9395		9526		
N = 1000, K = 100	10368	10380	10380		

Ký hiệu	Màu
Tìm được lời giải sai	
Không tìm được lời giải do giới hạn thời gian	

Về đặc điểm của các thuật toán, ta nhận thấy:

- Mô hình solver CP và MIP có sự chênh lệch, thể hiện rõ ràng ở những test cases lớn. CP có hiệu quả cao hơn MIP nhiều lần. Điều này có thể là do số lượng những lời giải khả thi so với kích thước không gian lời giải các test cases là rất bé. Do đó sự tập trung vào tính khả thi của CP dẫn đến kết quả này.
- Kết quả của mọi thuật toán tham lam đều rất phụ thuộc vào phân phối của các c_i cũng như khoảng $[l_j, u_j]$, thậm chí phụ thuộc vào cả thứ tự xét các đơn hàng và thứ tự xét các phương tiện. Việc tìm ra thứ tự nào là hợp lý cho 1 test case có lẽ chỉ có thể thông qua thử nghiệm trực tiếp.
- Thuật toán tham lam thứ hai có khả năng tìm được lời giải khả thi rất tốt do có sự tham gia của cả cận dưới l_j và u_j trong thuật toán.
- Ý tưởng sửa chữa lời giải có tiềm năng cao, tuy nhiên nó cần sự ngẫu nhiên và lặp lại nhiều lần. Việc áp dụng cố định ý tưởng vào chỉ một lời giải có thể dẫn đến kết quả không tốt.
- Nói chung, các thuật toán hiện tại chỉ đang tập trung vào **tính khả thi** của lời giải là chính chứ chưa hướng đến **tính tối ưu**. Để hướng đến tính tối ưu ta cần nhiều dữ liệu hơn để thử nghiệm và các dữ liệu đó nên phù hợp với tiêu chuẩn đã đề cập ở phần II.9.
- Thuật toán tham lam có thể thay thế 2 giai đoạn đầu của thuật toán Local Search. Khi đó ở giai đoạn 3 – giai đoạn sử dụng hàm Repair() - ta hoàn toàn có thể xây dựng một xe “ảo” chứa tất cả các đơn chưa được giao mà có:

$$l_{ảo} = 0$$

$$u_{ảo} = \sum_{i=1}^{N} c_i$$

  Khi đó ta coi xe “ảo” là xe loại 1 và gọi hàm Repair() hai lần
  - Lần 1 như ý tưởng gốc
  - Lần 2 coi toàn bộ xe thật là loại 2 và xe ảo là loại 1