Bài toán . Hệ thống lấy hàng động truck và drone hoạt động song song và ràng buộc time window

Input: Hệ thống lấy hàng gồm: 1 tập K truck đồng nhất, D drone đồng nhất thực cùng xuất phát từ Depot 0 phục vụ tập C. Trong đó, tập khách hàng C được chia làm 2 phần

- C1 - bắt buộc phải phục vụ xe tải do khối lượng quá lớn hoặc nằm ngoài khả năng

bay của drone

- C2- có thể phục vụ bởi cả drone hoặc truck

Constraints:

- Xe tải có tải trọng tối đa M_T
- Drone có tải trọng tốt đa M_D và bán kính bay tối đa là L_D
- Khách:
  - Thời gian đợi tối đa của một khách hàng là L_w (thời gian đợi = thời điểm gói hàng về đến kho hàng – thời điểm gói hàng được lấy xong)
  - Khách hàng i xuất hiện một cách dynamic (không biết trước) vào thời điểm r_i và yêu cầu phục vụ trong khoảng thời gian [e_i, l_i]
  - Mỗi gói hàng của khách hàng i có tải trọng yêu cầu: d_i
- Hành trình: Xe tải/drone được phép thực hiện nhiều hành trình, mỗi hành trình có thể phục vụ nhiều khách hàng sao cho không vi phạm các ràng buộc ở bên trên
- Thời gian sạc/thay pin drone, thời gian phục vụ khách hàng coi như = 0

Output:

Hành trình của các Truck, Drone

Objectives:

- Objective 1: tối thiểu hoá thời gian hoàn thành nhiệm vụ
- Objective 2: tối đa hoá số khách được phục vụ

Hướng tiếp cận: Sử dụng Genetic Programming. Thuật toán sẽ chia nhỏ thời gian của hệ thống thành các time slot a1, a2,..., an. Thời điểm ai sẽ chạy thuật toán để lập lịch cho xe vận hành vào thời điểm a(i+1)

Tham khảo: Bài báo bên dưới

Thuật toán sẽ xây dựng 2 luật routing rule và sequencing rule.

Routing rule xác định k xe có độ ưu tiên lớn nhất để thực hiện request req

Tại thời điểm a(i+1) phương tiện k sẽ có reqQueue các khách hàng chờ được phục vụ

Các yếu tố (nút lá của cây GP) ảnh hưởng tới routing rule là:

- Số lượng request trong reqQueue chia cho tổng số request của bài toán
- Chênh lệch giữa capacity của vehicle với tổng demand từ các request trong reqQueue chia cho tổng demand của cả bài toán

- Thời gian di chuyển từ median của các request trong reqQueue tới req đang xét chia cho thời gian close của bài toán
- Thời gian di chuyển từ vị trí hiện tại của xe tới req đang xét chia cho thời gian close của bài toán
- Ở demand của req đang xét chia cho tổng số demand của cả bài toán
- Kiểm tra xem vehicle là drone hay truck

```
def routing_rule(opt, vehicle, p, req):
    if opt == 0:
        return len(vehicle.reqQueue) / float(len(p.requests))
    elif opt == 1:
        return (vehicle.capacity - vehicle.sum_of_req_demand()) /
p.sum_of_req_demand()
    elif opt == 2:
        return vehicle.moving_time(vehicle.median_of_req_loc(),
req.location) / p.depot.time_window[1]
    elif opt == 3:
        return -vehicle.moving_time_to(req.location) /
p.depot.time_window[1]
    elif opt == 4:
        return req.demand / p.sum_of_req_demand()
    elif opt == 5:
        return 1.0 if vehicle.type == "DRONE" else 0.0
    else:
        return None

```

Sequencing rule: sequencing rule thì sẽ tính giá trị chỉ số cho vehicle với từng request trong reqQueue của vehicle đó để tìm ra request có chỉ số min để ưu tiên thực hiện.
Các yếu tố ảnh hưởng tới Sequencing rule

- Thời gian di chuyển từ vị trí hiện tại của xe tới req đang xét chia cho thời gian close của bài toán
- Chênh lệch giữa thời gian hiện tại của bài toán và thời gian request được đánh giá để xử lý chia cho thời gian close của bài toán
- TimeUnit Close là khoảng thời gian giữa lúc request đóng và thời gian vehicle hoàn thành request hiện tại đang làm.

- Demand của request chia cho tổng demand của bài toán

- Chênh lệch giữa thời gian hiện tại của bài toán và thời gian mở window của request chia cho thời gian đóng của bài toán

- Thời gian request xuất hiện chia cho thời gian đóng của bài toán

```
def sequencing_rule(opt, vehicle, p, req, cur_time, ready_time):
    if opt == 0:
        return vehicle.moving_time_to(req.location) /
p.depot.time_window[1]
```

```
    elif opt == 1:
        return (cur_time - ready_time) / p.depot.time_window[1]
    elif opt == 2:
        time_until_close = req.time_window[1] - vehicle.busy_until
        if time_until_close <= 0.0001:
            return 1.0
        else:
            return (time_until_close -
vehicle.moving_time_to(req.location)) / time_until_close
    elif opt == 3:
        return req.demand / p.sum_of_req_demand()
    elif opt == 4:
        return (cur_time - req.time_window[0]) /
p.depot.time_window[1]
    elif opt == 5:
        return req.appeared_time / p.depot.time_window[1]

    else:
        return None

```

Bài báo : Policy Evolution for Routing and Scheduling in the Dynamic Parallel Drone-Vehicle Routing Problem with Time Windows

1. Phương pháp luận

1.1 Dispatching policies

Trong bài báo này, DPDTRPTW được phân rã thành hai bài toán con có mối quan hệ với nhau. Bài toán con đầu tiên tập trung vào việc lựa chọn khách hàng và phân công phương tiện, quyết định có chấp nhận từng yêu cầu đến hay không và, nếu được chấp nhận, phân bổ nó cho nguồn lực phù hợp nhất. Bài toán con thứ hai liên quan đến việc sắp xếp dịch vụ, quyết định thứ tự một cách chặt chẽ trong phạm vi thời gian mà các yêu cầu được chấp nhận sẽ được xử lý. Sự phân rã này cho phép hệ thống phản ứng động với các yêu cầu đến trong khi tối ưu hóa cả chất lượng dịch vụ và hiệu quả của hệ thống trong môi trường giao hàng kết hợp. Cụ thể, chúng tôi chuyển hai bài toán con này thành hai quy tắc quyết định tương ứng: một quy tắc định tuyến và một quy tắc sắp xếp. Mỗi quy tắc định tuyến R và quy tắc sắp xếp S là một hàm phụ thuộc vào thời gian của phương tiện $k \in D \cup T$ và khách hàng $i \in C$.

Routing roule: Khi đơn hàng của khách hàng i đến, hệ thống tính giá trị ưu tiên R(k,i) cho mỗi phương tiện $k \in D \cup T$, và gán i cho phương tiện k' mà giá trị này đạt tối đa:

$$k' = \arg\max_{k \in D \cup T} R(k,i).$$

Sau khi k' được xác định, khách hàng i sẽ được thêm vào hàng đợi phương tiện $Q_{k'}$ của phương tiện k'

Sequencing rule: thứ tự Khi một phương tiện k cần quyết định nút nào sẽ ghé thăm tiếp theo, nó đánh giá giá trị ưu tiên S(k,i) cho tất cả khách hàng i hiện đang trong hàng đợi của nó. Nút tiếp theo để ghé thăm i' được chọn là nút tối đa hóa giá trị này, đồng thời vẫn thỏa mãn ràng buộc cửa sổ thời gian của nó:

$$i' = \arg\max_{i \in V \setminus \{0\}} S(k,i).$$

Nếu ràng buộc về sức chứa bị vi phạm, phương tiện k sẽ quay về kho trước khi phục vụ khách hàng đã chọn.

Chiến lược Đánh giá Đối với bài toán động P đã cho, chúng tôi biến đổi nó thành một bài toán tĩnh sT(P) bằng cách chỉ xem xét các khách hàng i có thời gian tạo $t_i < T$ cho một giá trị T nào đó.

Với một chính sách điều phối $\pi = (R,S)$, chúng tôi tiến hành mô phỏng π trên sT(P) và trích xuất các mục tiêu liên quan f1 và f2. Các giá trị này được chuẩn hóa thành $\bar{f}1$ và $\bar{f}2$. Sau đó, chúng tôi kết hợp hai mục tiêu này bằng một tổng có trọng số:

$$f = \lambda \bar{f}1 + (1-\lambda)\bar{f}2.$$

f sau đó được sử dụng như một ước lượng về mức độ hiệu quả của π khi áp dụng trên P.

Problem augmentation : Vì sT(P) chỉ là một phần nhỏ của vấn đề động P, chúng ta có thể thực hiện tăng cường như sau:

– Đối với mỗi khách hàng i, chúng ta giảm cửa sổ thời gian của i theo một hệ số $sf \in (0,1)$ (hệ số căng thẳng). Cửa sổ thời gian gốc $W = [s_i, e_i]$ sẽ được chuyển thành:

$$W' = [s_i, s_i + (1 - sf)(e_i - s_i)].$$

– Các khách hàng bổ sung sẽ được tạo ra dựa trên các khách hàng có sẵn trong sT(P).

Sau khi tìm được chính sách $\pi*$ hoạt động tốt trên sT(P), chúng ta sẽ sử dụng nó như chiến lược điều phối cho vấn đề động P.

1.2 Genetic programming hyper-heuristic algorithm approach

Để giải quyết các vấn đề phụ này, chúng tôi giới thiệu một thuật toán siêu heuristic lập trình di truyền (GPHH) để xử lý hiệu quả hai tập hợp quy tắc khác nhau. Thuật toán đề xuất được chia thành hai bước: giai đoạn huấn luyện và giai đoạn kiểm tra. Trong giai đoạn huấn luyện, thuật toán tạo ra một tập hợp các quy tắc heuristic thay vì một giải pháp trực tiếp cho vấn đề DPDTRPTW. Những quy tắc này sau đó sẽ được sử dụng làm đầu vào cho giai đoạn kiểm tra, tại đó hiệu quả của chúng được đánh giá.

– **Trainning phase**: Trong giai đoạn này, chúng tôi làm việc với một quần thể, được ký hiệu là P = {p1, p2, ..., p_size}, trong đó mỗi cá thể trong P được biểu diễn dưới dạng đa cây. Giá trị thích nghi của từng cá thể được đánh giá bằng cách áp dụng hai quy tắc trên một bản sao Ttrain. Quần thể tiến hóa thông qua nhiều cơ chế lập trình di truyền trong mỗi thế hệ nhằm tăng cường sự đa dạng và khám phá các giải pháp tiềm năng.

– **Testing phase**: Các quy tắc được rút ra từ giai đoạn huấn luyện sẽ được kiểm tra kỹ lưỡng trên dữ liệu kiểm tra trong giai đoạn này. Quá trình kiểm tra tương tự như một lần đánh giá trong giai đoạn huấn luyện, nhưng diễn ra trong một môi trường mô phỏng lớn hơn. Cần lưu ý rằng các bộ dữ liệu sử dụng cho huấn luyện và kiểm tra là riêng biệt, mặc dù chúng có các đặc điểm tương tự.

GPHH mô phỏng quá trình tiến hóa tự nhiên để nâng cao chất lượng thế hệ con cháu qua nhiều thế hệ kế tiếp. Quá trình này bao gồm bốn bước chính: khởi tạo, đánh giá, lựa chọn và tái tổ hợp. GPHH bắt đầu với một quần thể các cá thể được khởi tạo ngẫu nhiên. Bước đánh giá sẽ đánh giá chất lượng của từng cá thể sử dụng dữ liệu DPDTRPTW. Miễn là tiêu chí dừng chưa được thỏa mãn (tức là số thế hệ hiện tại t nhỏ hơn số thế hệ tối đa G), thuật toán sẽ tiến hành lựa chọn cha mẹ, nơi các cá thể có mức thích nghi cao hơn được chọn để tạo ra thế hệ con mới thông qua các toán tử di truyền như sinh sản, lai ghép và đột biến. Nếu tiêu chí dừng được thỏa mãn, thuật toán sẽ trả về thủ thuật lập lịch tốt nhất $p*$ được tìm thấy trong quá trình này như là lời giải cho bài toán DPDTRPTW.

Algorithm 1 Training phase in GPHH

Input:

pop_size: population size;

intermediate_size: offspring population size;

rm: mutation rate;

rc: crossover rate;

G: the number of generation;

Ttrain: training dataset

Output: Best individual $p*$

1: Initialize a population P(0) ;

2: Evaluate every individual in P(0);

3: t ←1

4: while t ≤ G do

5: Offspring population O(t) ← ∅ ;

6: while |O(t)| < intermediate_size do

7: rand ← U(0,1) ;

8: if rand < rc then

9: Select parents pa,pb from P(t−1);

10: o ←crossover(pa,pb) ;

11: else if rand < rc +rm then

12: Select parent p from P(t−1);

13: o ←mutation(p) ;

14: else

15: Select parent p from P(t−1);

16: Reproduce o from p;

17: end if

18: $O(t) \leftarrow O(t) \cup \{o\}$;

19: end while

20: Evaluate O ;

21: $P(t) \leftarrow P(t-1) \cup O(t)$;

22: $P(t) \leftarrow$ ElitismSelection(P(t)) ;

23: $t \leftarrow t+1$;

24: end while

25: $p* \leftarrow$ Best individual in P(t)

26: return $p*$

Individual representation: Mỗi cá thể trong GPHH được biểu diễn dưới dạng cấu trúc chương trình cây, bao gồm các nút nội bộ và các nút cuối. Các nút này xác định cấu trúc và hành vi của chương trình, được điều chỉnh phù hợp với các yêu cầu cụ thể của bài toán tối ưu hóa. Cốt lõi của biểu diễn này là nắm bắt các thuộc tính liên quan đến vấn đề tại các điểm quyết định và mã hóa chúng vào một hàm ưu tiên, xác định các hoạt động hoặc hành động cần được thực hiện.

Bằng cách kết hợp các thuộc tính thông qua các toán tử toán học một cách đệ quy, cây có thể biểu diễn logic quyết định phức tạp, phi tuyến. Điều này cho phép mô hình xem xét một phạm vi thông tin rộng lớn khi xác định mức độ ưu tiên của các hành động, dẫn đến việc ra quyết định chính xác và hiệu quả hơn. Trong số các mô hình được đánh giá, cấu trúc dựa trên cây cho thấy hiệu quả cao nhất nhờ tính gọn gàng, khả năng giải thích và sức mạnh biểu đạt của nó.

Trong bài báo này, chúng tôi sử dụng một biểu diễn cây đôi, trong đó mỗi cá thể được mô hình hóa như một cặp (R, S), với R biểu thị quy tắc định tuyến và S biểu thị quy tắc sắp xếp thứ tự. Cả hai quy tắc đều được mã hóa dưới dạng cây lập trình di truyền (GP). Cụ thể:

– Các nút đầu cuối: các thuộc tính đặc thù của vấn đề được lấy từ môi trường mô phỏng, chẳng hạn như thời gian hiện tại, nhu cầu hiện tại, mức độ sử dụng bộ đệm, hoặc thời gian thực hiện thao tác.

– Các nút nội bộ: các phép toán toán học (ví dụ, +, −, ×, ÷, min, max) được sử dụng để kết hợp các giá trị đầu cuối thành các biểu thức tổng hợp.

Cấu trúc cây kép này cho phép sự tiến hóa độc lập nhưng có phối hợp của các chính sách về sắp xếp thứ tự và điều hướng, giúp GPHH xử lý các nhiệm vụ ra quyết định phức tạp với tính linh hoạt và hiệu suất được cải thiện.

Initialization Chúng tôi áp dụng kỹ thuật khởi tạo nửa-nửa theo kiểu tăng dần để xây dựng quần thể ban đầu. Trong phương pháp này, một nửa cá thể được tạo ra bằng phương pháp đầy đủ, trong khi nửa còn lại được khởi tạo bằng phương pháp một cách tăng trưởng. Sự kết hợp này thúc đẩy sự đa dạng về cấu trúc trong quần thể ban đầu và giúp tránh hội tụ sớm.

– **Full initialization** (Khởi tạo đầy đủ): Tất cả các cây được xây dựng để đạt đến một độ sâu tối đa xác định trước. Mỗi nút trong cây được gán một cách xác định một hàm (đối với các nút nội bộ) hoặc một hằng số đầu cuối (đối với các lá), đảm bảo rằng tất cả các nhánh đều được điền đầy đủ. Mặc dù phương pháp này cho phép khám phá toàn bộ không gian tìm kiếm, nhưng thường dẫn đến các cây lớn và phức tạp, điều này có thể làm tăng chi phí tính toán và giảm hiệu quả của các thao tác di truyền tiếp theo.

– **Grow initialization**(Khởi tạo tăng trưởng): Các cây được tạo với độ sâu biến đổi trong một phạm vi xác định. Tại mỗi nút, một hàm hoặc một hằng số đầu cuối được chọn ngẫu nhiên, dẫn đến các cây có hình dạng và kích thước khác nhau. Phương pháp này góp phần tạo ra một quần thể ban đầu đa dạng hơn bằng cách sản xuất các cá thể với các mức độ phức tạp khác nhau.

Các toán tử di truyền (Genetic Operator) là một thành phần cơ bản của thuật toán tiến hóa, vì chúng chịu trách nhiệm tạo ra các cá thể mới thông qua tái tổ hợp và biến đổi. Trong nghiên cứu này, hai toán tử được sử dụng phổ biến trong GPHH được áp dụng: **sub-tree crossover (giao hoán cây con)** và **sub-tree mutation (đột biến cây con)**.

sub-tree crossover: Toán tử này tạo ra một thế hệ con bằng cách trao đổi các cấu trúc con giữa hai cây cha mẹ. Cụ thể, một cây con ngẫu

nhiên được chọn từ mỗi cha mẹ và hoán đổi với nhau, tạo thành hai cá thể mới. Giả sử depthsub-tree_A và depthsub-tree_B lần lượt biểu thị độ sâu của các cây con được chọn từ Cha mẹ A và B. Gọi depthcut_A và depthcut_B là độ sâu tại đó các cây con này được cắt. Để đảm bảo rằng các cây kết quả không vượt quá độ sâu tối đa cho phép L, các ràng buộc sau phải được thỏa mãn:

$$\text{depthsub-tree\_A} + \text{depthcut\_B} \leq L,$$

$$\text{depthsub-tree\_B} + \text{depthcut\_A} \leq L$$

sub-tree mutation: Giới thiệu sự biến đổi ngẫu nhiên vào quần thể và đóng vai trò quan trọng trong việc duy trì đa dạng di truyền và tránh hội tụ sớm vào cực trị cục bộ. Đối với mỗi cá thể được chọn để đột biến, một nút ngẫu nhiên (điểm đột biến) được chọn trong cây của nó. Cây con lấy gốc từ nút này được thay thế bằng một cây ngẫu nhiên mới được tạo ra. Để duy trì chiều sâu cây tối đa L cho phép, chiều sâu của cây được chèn bị giới hạn ở $L - \text{depthsub}{-}\text{tree}p$, trong đó $\text{depthsub}{-}\text{tree}p$ là chiều sâu của cây tại điểm đột biến. Ràng buộc này đảm bảo rằng cây kết quả không vượt quá chiều sâu tối đa cho phép.

Elitism selection(Lựa chọn ưu tú) Bài báo này sử dụng lựa chọn ưu tú, trong đó các cá thể con được kết hợp với quần thể cha mẹ, và những cá thể có hiệu suất cao nhất được chọn để tiến tới thế hệ tiếp theo. Chiến lược này bảo tồn các giải pháp tốt nhất, đảm bảo các đặc điểm di truyền của chúng đóng góp vào việc lai tạo trong tương lai. Kết quả là, quần thể giữ được các đặc tính chất lượng cao, thúc đẩy hội tụ nhanh hơn về giải pháp tối ưu.