1. Giới thiệu

Bài báo giới thiệu một cách triển khai thuật toán Hybrid Genetic Search (HGS) cho bài toán Vehicle Routing Problem with Time Windows (VRPTW) (VRPTW). Dựa trên phiên bản HGS-CVRP, nhóm tác giả đã mở rộng để xử lý các ràng buộc thời gian, bổ sung thêm một số operator, đồng thời điều chỉnh các tham số và tối ưu hiệu năng thực thi.

Đây không phải là một công trình được công bố trên tạp chí học thuật chính quy, mà là một bản tổng hợp và cải tiến từ các nghiên cứu trước đó. Các thuật toán được triển khai và thử nghiệm trên bộ dữ liệu VRPTW của cuộc thi DIMACS lần thứ 12, và bộ giải này đạt hạng nhất trong giai đoạn đầu của hạng mục VRPTW.

1.1. Bài toán

Cho $G = (V, E)$ là một đồ thị đầy đủ có hướng.

Đỉnh $v_0 \in V$ thể hiện điểm tập kết, nơi tập hợp m xe với sức chứa Q. Còn lại các đỉnh $v_i \in V^{CST}$ với $V^{CST} = V \setminus \{v_0\}$ với $i \in [1, n]$ thể hiện các khách hàng cần được phục vụ. Mỗi khách hàng có nhu cầu không âm q_i, thời gian cần để phục vụ τ_i, và khoảng thời gian mà xe được phép phục vụ là $[e_i, l_i]$

Cạnh $(i, j) \in E$ thể hiện một đường đi trực tiếp từ v_i đến v_j với khoảng cách c_{ij} và thời lượng cần để di chuyển δ_{ij}.

Một hành trình khả thi r được định nghĩa là một chu trình trong G xuất phát và kết thúc tại v_0, sao cho tổng nhu cầu của khách hàng trong r nhỏ hơn hoặc bằng Q.

Bài toán VRPTW hướng đến việc xây dựng m chuyến đi cho các xe để thăm các đỉnh đúng một lần trong khoảng thời gian cho phép, và cực tiểu hoá tổng khoảng cách đã di chuyển.

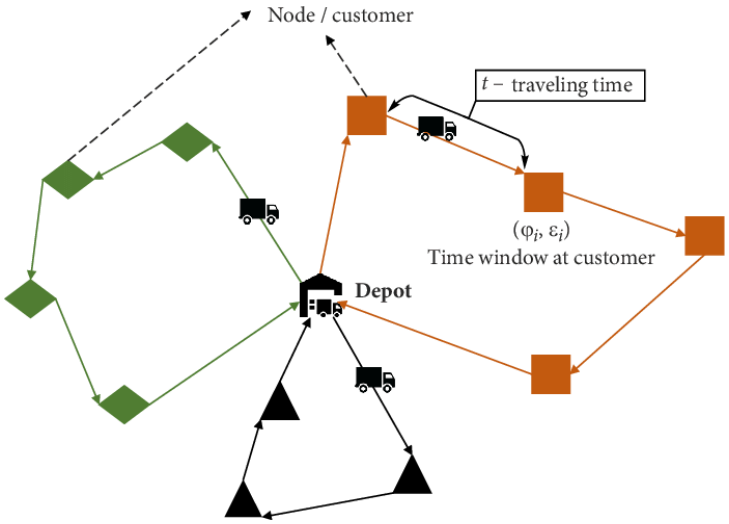


Figure 1: An example about VRPTW (Wang et al. 2020)

1.2. Các nghiên cứu liên quan

Table 1. VRPs of previous studies (source: current study)

Study	Objective				Problem description						Mathematical model and algorithm
					network configuration (N – nodes)	Constraint			Data		
	cost	time	no of vehicle	dis-tance		time window	vehicle capacity	kind of products	experi-ence	reality	
Qi, Hu (2020)	✓		✓		13 N				✓		MIP, heuristic
Kantawong, Pravesjit (2020)	✓			✓	100 N	✓			✓		MIP, artificial bee colony
Londoño *et al.* (2021)	✓			✓	51 N				✓		MILP, local search
Aggarwal, Kumar (2019)	✓		✓		60 N	✓			✓		MIP
Pérez-Rodríguez, Hernández-Aguirre (2019)				✓	6 N	✓				✓	an estimation of distribution
Tasar *et al.* (2019)	✓		✓		100 N				✓		MIP, heuristic
Zhu, Hu (2019)	✓				200 N		✓		✓		MILP, response surface method
Ruiz *et al.* (2019)	✓				14 N		✓			✓	MIP, biased random – key genetic
Zhang *et al.* (2017)	✓				27 N	✓			✓		tabu search, the artificial bee colony
Birim (2016)	✓				10 N		✓		✓		MILP, simulated annealing
Afifi *et al.* (2016)		✓			30 N	✓			✓		MIP, simulated annealing
Spliet, Desaulniers (2015)	✓				60 N	✓	✓		✓		MIP, exact branch – price – cut algorithm
Current study	✓		✓		39 N	✓	✓	✓		✓	MIP, clustering algorithm

2. Thuật toán

2.1. Hàm mục tiêu

Xét một hành trình r, xuất phát từ v_0 ($\sigma_0^r = 0$), thăm n_r khách hàng $\left(\sigma_1^r, ..., \sigma_{n_r}^r\right)$, sau đó quay trở lại điểm xuất phát $\sigma_{n_r+1}^r = 0$. Gọi $\boldsymbol{t}^r = \left(t_0^r, ..., t_{n_r+1}^r\right)$ là thời điểm ta phục vụ điểm thứ i. Đường đi từ đỉnh σ_i^r đến σ_{i+1}^r sẽ gây ra độ trễ (time warp, nghĩa là thời gian vượt quá khả năng chờ của khách hàng) là $\mathrm{tw}_{i,i+1} = \max\left(t_i^r + \tau_{\sigma_i^r} + \delta_{\sigma_i^r \sigma_{i+1}^r} - t_{i+1}^r, 0\right)$.

Các đại lượng sau đây đặc trưng cho tuyến đường r:

- Trọng tải $q(r) = \sum_{i=1}^{n_r} q_{\sigma_i^r}$
- Tổng khoảng cách $c(r) = \sum_{i=0}^{n_r} c_{\sigma_i^r \sigma_{i+1}^r}$
- Tổng độ trễ $\mathrm{tw}(r) = \sum_{i=0}^{n_r} \mathrm{tw}_{i,i+1}$

Ta cần tìm các hành trình sao cho tổng chi phí của các hành trình là tối thiểu, với chi phí của mỗi hành trình được tính bằng công thức:

$$\varphi(r) = c_r + \omega^Q \max(0, q(r) - Q) + \omega^{\mathrm{TW}}\ \mathrm{tw}(r)$$

Trong đó ω là các hệ số phạt tương ứng cho độ trễ và trọng tải thừa

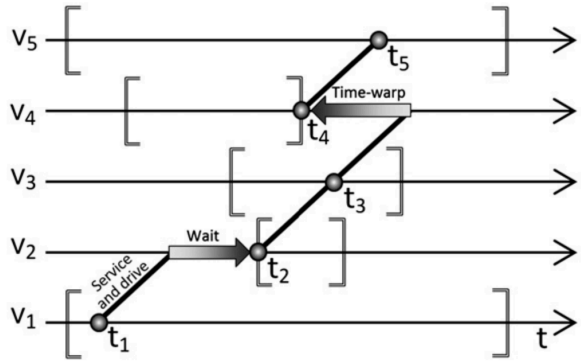
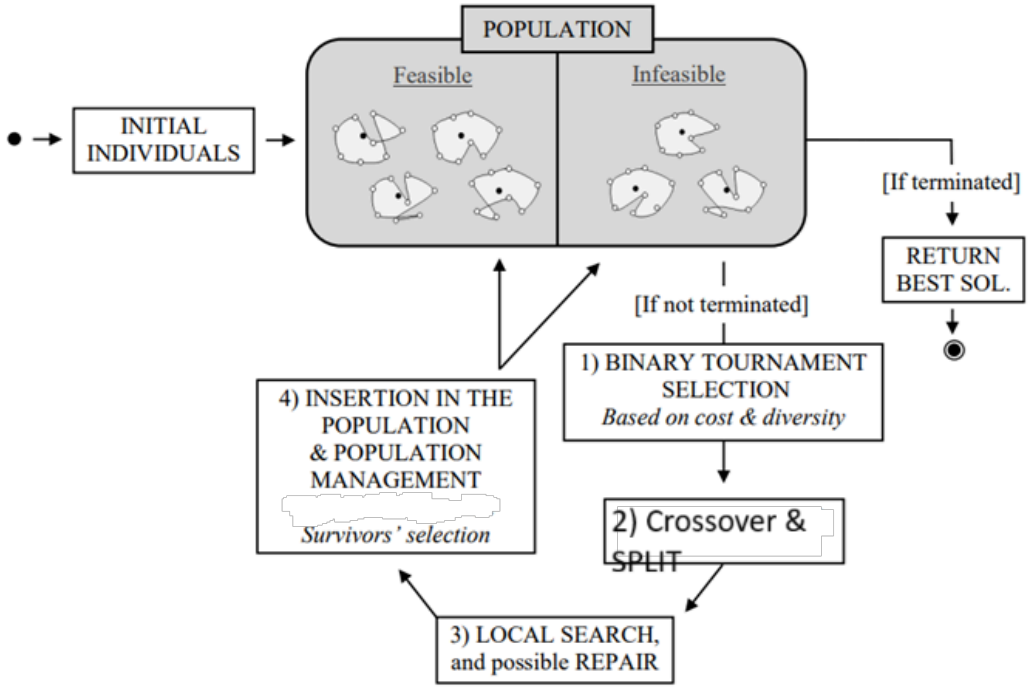


Fig. 1. Illustration of waiting times and time warps.

2.2. Tổng quan



2.3. Quần thể

Thuật toán duy trì 2 quần thể, bao gồm các lời giải hợp lệ và không hợp lệ. Mỗi lần sản sinh ra một cá thể mới, chúng được chèn vào quần thể phù hợp. Ban đầu, 4μ lời giải được sinh ra dựa trên heuristics kiến trúc, cải tiến bằng local search và chèn lại vào bể tương ứng. Khi có một quần thể đạt $\mu + \lambda$ cá thể, các cá thể sẽ bị loại dần cho đến khi quần thể còn μ lời giải. Việc này có thể thực hiện bằng việc xoá các lời giải trùng nhau. Sau khi lọc trùng, ta xoá các lời giải có độ phù hợp tồi nhất đi.

2.4. Heuristics kiến trúc

Ban đầu, 100 lời giải ngẫu nhiên được sinh ra, sau đó được hiệu chỉnh sử dụng local search. Ngoài ra, để tăng tốc độ hội tụ, tác giả còn cài đặt 3 thuật toán tham lam để sinh lời giải ban đầu khác nhau bao gồm *nearest*, *farthest* và *sweep*, mỗi thuật toán được sử dụng để xây dựng 5% lời giải ban đầu.

2.4.1. *nearest, farthest*

Ta sẽ xây dựng từng hành trình một. Ban đầu, một điểm bất kì gần/xa điểm tập kết nhất được chèn vào. Sau đó, ta lặp lại việc tìm một khách hàng chưa được phục vụ và vị trí để chèn nó vào hành trình hiện tại sao cho khoảng cách tăng lên là ít nhất và việc chèn khách hàng này vào không gây ra bất kì vi phạm nào về time window hoặc capacity, và tiến hành chèn vào. Nếu không thể chèn thêm bất kì khách hàng nào vào hành trình hiện tại, ta bắt đầu một chuyến mới và tiến hành tương tự.

Do việc chèn như vậy là tất định, nên để dựng được nhiều hơn một lời giải, ta có thể xây dựng thêm các lời giải gần khả thi, nghĩa là cho phép tổng độ trễ trong khoảng từ 0 đến 100, và tổng khối lượng thừa trong khoảng từ 0 đến 50.

```
S = {1, 2, 3, ..., n}
solutions <- []

for t = 1 to m:
  customers <- []

  i <- nearest/farthest customers from depot in S
  while S is not empty:
    j <- a customer in S that can be add to `customers` and causes least detour
distance
    if j not exists:
      break
    add j to customers, remove j from S

  add customers to solutions
```

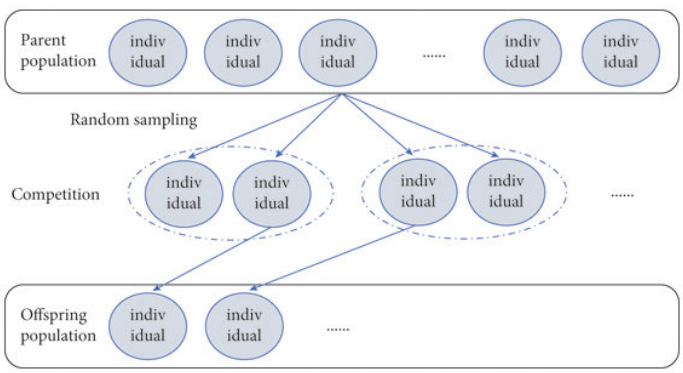
2.4.2. *sweep*

- Sắp xếp các khách hàng theo góc giữa nó và điểm xuất phát.
- Thêm các khách hàng cho đến khi quá trọng tải.
- Với mỗi chuyến đi:
  - Sắp xếp các khách hàng có time window ngắn ($< 50\%$) tăng dần theo l_i
  - Chèn lại các khách hàng có time window dài sao cho khoảng cách tăng là ít nhất.

Lưu ý thuật toán có thể cho ra các cấu hình vi phạm ràng buộc về thời gian. Để có thể xây dựng thêm lời giải, ta chạy lại thuật toán với trọng tải giảm ngẫu nhiên một lượng từ 0 đến 40%.

2.5. Lai tạo

2.5.1. Lựa chọn cha mẹ



Để chọn một lời giải, thuật toán thực hiện "binary tournament" để lựa chọn, cụ thể như sau:

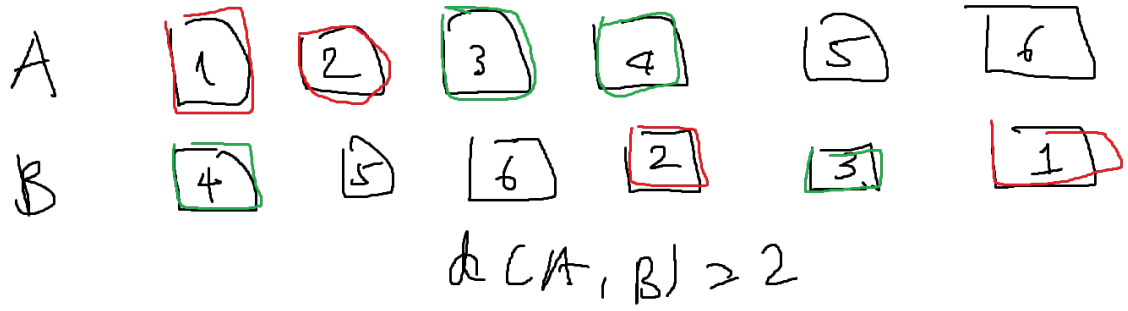
- Chọn ngẫu nhiên 2 lời giải trong tập các lời giải hiện tại với xác suất như nhau
- Trong 2 lời giải đó, chọn lời giải có độ phù hợp tốt hơn

Độ phù hợp của một lời giải được tính bằng công thức:

$f_P(S) = f_P^{\varphi}(S) + \left(1 - \frac{n^{\text{ELITE}}}{|P|}\right) f_P^{\text{DIV}}(S)$

Trong đó:

- Hạng của một lời giải được tính bằng $\frac{\text{chỉ số của lời giải sau khi sắp xếp theo chất lượng}}{\text{tổng số lời giải}}$
- $f_P^{\varphi}(S)$ là "hạng" của lời giải S, sắp xếp theo chất lượng lời giải.
- $f_P^{\text{DIV}}(S)$ là "hạng" của lời giải S, khi xét khả năng mở rộng (diversity contribution)
  - Có nhiều cách tính độ tốt khi xét đến khả năng mở rộng, nhưng theo thực nghiệm thì tốt nhất là sử dụng "broken pairs distance"
  - Broken pairs distance của một cặp A và B được tính bằng số cặp (i, j) kề nhau trong A nhưng **không** kề nhau trong B (tính cả depot).



  - Khả năng mở rộng của lời giải S, được tính bằng trung bình các broken pair distance từ S đến n^{CLOSEST} lời giải có BPS đến S là nhỏ nhất.

  $\Delta(S) = \frac{1}{n^{\text{CLOSEST}}} \sum_{S_2} \delta(S, S_2)$

- Hệ số $\left(1 - \frac{n^{\text{ELITE}}}{|P|}\right)$ được sử dụng để đảm bảo ta vẫn giữ lại được n^{ELITE} lời giải chất lượng tốt nhất trong suốt quá trình tìm kiếm

2.5.2. Thao tác

Toán tử tương giao chéo (Crossover operator). Xem lời giải như một hoán vị độ dài n. Chọn ngẫu nhiên một đoạn con $[l, r]$ của cha, và đưa đoạn này vào lời giải của con. Sau đó, xuất phát từ $r + 1$, ta lần lượt đưa các nút chưa được sử dụng theo thứ tự xuất hiện trong lời giải mẹ vào lời giải con.

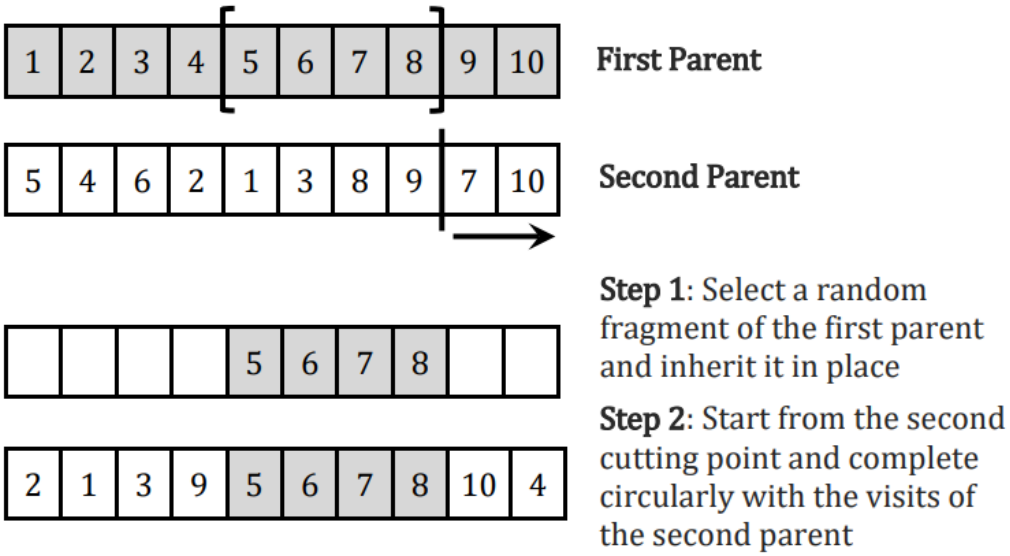


Figure 2: OX Crossover

Toán tử tương giao chéo này được thực hiện mà không xét đến các ràng buộc về khối lượng, thời gian hay về depot. Vì vậy, từ hoán vị đã tương giao, ta sử dụng thuật toán SPLIT để chia hoán vị đã tương giao về các xe khác nhau sao cho thoả mãn ràng buộc về trọng lượng.

(*SPLIT là một thuật toán giúp các lớp thuật toán di truyền có thể cho ra kết quả tốt hơn thuật toán tabu search truyền thống, khi nó giúp cho việc lai ghép 2 lời giải một cách dễ dàng hơn bằng việc cho phép phân chia một hoán vị ra các hành trình con. Từ đó, ta có thể sử dụng các phương pháp lai ghép truyền thống giữa các hoán vị cho bài toán này.*)

Thuật toán SPLIT có thể hiểu đơn giản như một thuật toán quy hoạch động:

Algorithm 1: Class Split Algorithm: Fleet limited to m vehicles

```
1  for k ← 1 to m do
2      for t ← 1 to n do
3          f[k][t] ← ∞;
4  f[0][0] ← 0
5  for k ← 1 to m do
6      for i ← 0 to n do
7          j ← i + 1;
8          while j ≤ n and canAdd(j) do
9              if f[k][j] > f[k − 1][i] + cost(i + 1, j) then
10                 f[k][j] ← f[k − 1][i] + cost(i + 1, j);
11                 pred[k][j] ← i
12         j ← j + 1;
```

Tuy nhiên, trong thuật toán HGS-CVRP, thay vì quy hoạch động thuần tuý $O(n^2)$, thuật toán đã sử dụng kĩ thuật stack để giảm độ phức tạp xuống còn $O(n)$

$$D[i] = \sum_{k=1}^{i-1} d_{k,k+1}$$

$$Q[i] = \sum_{k=1}^{i} q_k.$$

$$c(i,j) = d_{0,i+1} + D[j] - D[i+1] + d_{j,0} + \alpha \times \max\{Q[j] - Q[i] - Q, 0\}.$$

$$dominates(i,j) \equiv \begin{cases} p[i] + d_{0,i+1} - D[i+1] + \alpha \times (Q[j] - Q[i]) \leq p[j] + d_{0,j+1} - D[j+1] & \text{if } i < j \\ p[i] + d_{0,i+1} - D[i+1] \leq p[j] + d_{0,j+1} - D[j+1] & \text{if } i > j. \end{cases} \quad (9)$$

Algorithm 2: Linear Split

```
p[0] ← 0
Λ ← (0)
for t = 1 to n do
    p[t] ← p[front] + f(front, t)
    pred[t] ← front
    if t < n then
        if not dominates(back, t) then
            while |Λ| > 0 and dominates(t, back) do
                popBack()
            pushBack(t)
        while |Λ| > 1 and p[front] + f(front, t+1) ≥ p[front2] + f(front2, t+1).
            popFront()
```

Chứng minh tính đúng đắn của thuật toán có thể tham khảo tại paper *Technical Note: Split algorithm in O(n) for the capacitated vehicle routing problem.*

Việc chia hoán vị như thuật toán HGS-CVRP ban đầu có thể gây cả vi phạm về thời gian, khi đó ta sử dụng local search để điều chỉnh với hi vọng toàn bộ vi phạm về thời gian sẽ được giải quyết.

Selective Route Exchange (SREX).

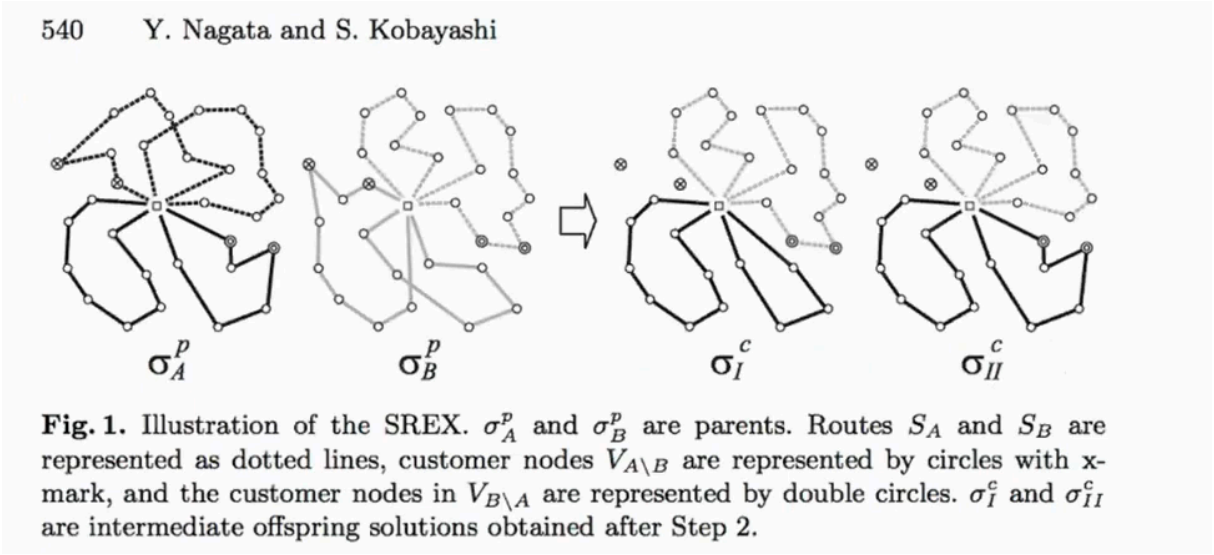


Fig. 1. Illustration of the SREX. σ_A^p and σ_B^p are parents. Routes S_A and S_B are represented as dotted lines, customer nodes $V_{A\setminus B}$ are represented by circles with x-mark, and the customer nodes in $V_{B\setminus A}$ are represented by double circles. σ_I^c and σ_{II}^c are intermediate offspring solutions obtained after Step 2.

Hai lời giải con khác nhau được tạo ra phụ thuộc vào việc … và ta sẽ chọn lời giải tốt nhất khi xét đến hàm mục tiêu. Vì SREX, nó không dùng đến thuật toán SPLIT kể trên, vì vậy nó phù hợp hơn khi sử dụng để giải bài toán có ràng buộc về thời gian.

Two different offspring solutions are created dependent on from which parents' routes the duplicate nodes are removed and we continue with the best solution (in terms of penalized cost). Since SREX preserves depot visits, it does not use the SPLIT algorithm and is thus more suitable for time windows.

Sửa đổi quần thể. Cuối cùng, ta sinh ra con cháu sử dụng cả hai loại thao tác trên, và tiếp tục đến bước local search với lời giải tốt nhất khi xét 2 thao tác.

2.6. Local search

Yet, these procedures tend inevitably to make for the largest part (90–95%) of the overall computational effort, such that high computational efficiency is required. Three basic aspects are decisive for performance: (1) a suitable choice of neighbourhood, restricted to relevant moves while being large enough to allow some structural solution changes; (2) memory structures to evade redundant move computations; and (3) highly efficient neighbour cost and feasibility evaluations. We introduce new methodologies to address these aspects relatively to the specificities of time-constrained VRPs.

Thủ tục được áp dụng sau khi sinh lời giải ngẫu nhiên, và để hiệu chỉnh lời giải sau khi lai ghép. Các operator được sử dụng:

Swap.

Relocate.

2-opt*. Chọn 2 tuyến đường r và r', mỗi tuyến đường ta chọn một vị trí bất kì. Sau đó đổi chỗ $\left(\sigma_i^r, ..., \sigma_{n_r}^i\right)$ và $\left(\sigma_{i'}^{r'}, ..., \sigma_{n_{r'}}^{i'}\right)$

2-opt. Chọn một tuyến đường r và một đoạn con $[i, j]$ thuộc r, sau đó đảo ngược đoạn $\left(\sigma_i^r, ..., \sigma_j^r\right)$

Swap*.

if the routes involved in a move currently have 0 time-warp, then we first check only if the move will reduce the total distance. If this is not the case, the move can never be improving and we can discard the move, avoiding the expensive TW-data computation.

Để tăng tốc độ cho quá trình local search, các cấu hình láng giềng cũng được cắt tỉa sử dụng các thước đo tương quan. Tập các láng giềng có tương quan trong các bài toán VRP cổ điện liên quan đến một thước đo gần không gian. Tuy nhiên, bài toán VRP có ràng buộc thời gian lại bao gồm một chiều khác, liên quan đến độ gần về thời gian (time proximity), cũng như các vấn đề bổ sung về tính bất đối xứng. Do đó, các quan hệ tương quan trở nên khó định nghĩa hơn.

Tóm lại, việc ta lựa chọn một vị trí i để thực hiện thao tác, thì mình cần phải lựa chọn làm sao để sau khi thực hiện thao tác tại vị trí i, thì chi phí gây ra cho những thằng láng giềng quanh nó (index $i-1$ hoặc $i+1$) là nhỏ nhất. Và chi phí gây ra đó được tính bằng sự đóng góp về mặt thời gian và khoảng cách, được giới thiệu trong công thức sau đây:

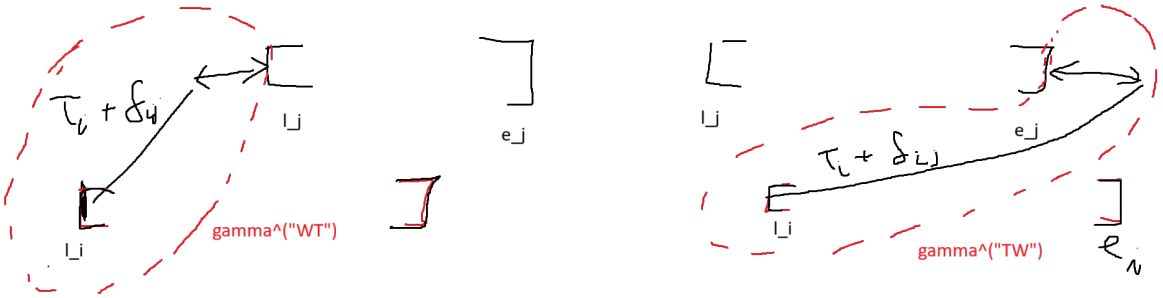
Với mỗi khách hàng i, ta định nghĩa một tập $\Gamma(i)$ gồm Γ khách hàng gần i nhất theo độ đo tương quan dưới đây:

$$\gamma(i,j) = c_{ij} + \gamma^{\mathrm{WT}} \max\left(e_j - \tau_i - \delta_{ij} - l_i, 0\right) + \gamma^{\mathrm{TW}} \max\left(e_i + \tau_i + \delta_{ij} - l_j, 0\right)$$

Với γ^{WT} là hệ số cho việc chờ đợi khi xuất phát từ i tại thời điểm l_i và đi trực tiếp đến j, và γ^{TW} là hệ số phạt cho time-warp khi đi từ i đến j.

Thước đo tương quan này tương ứng với tổng có trọng số của ba thành phần:

- Khoảng cách
- Thời gian chờ tối thiểu
- Mức phạt tối thiểu



Lúc này, khi thực hiện các thao tác:

- 2-opt: chỉ xét các cặp (i, j) có $\sigma_{j+1} \in \Gamma(\sigma_i)$ hoặc $\sigma_j \in \Gamma(\sigma_{i-1})$

- Swap, replace: ?

Số trạng thái cần duyệt lúc này là $O(\Gamma n)$ thay vì $O(n^2)$

Vì ta cho phép việc tìm kiếm ra các lời giải không hợp lệ, có thể sau khi tìm kiếm cục bộ, lời giải tìm được vẫn không hợp lệ. Khi điều này xảy ra, ta sẽ tiến hành sửa chữa (Repair operation) với xác suất 50%. Nghĩa là ta sẽ thực hiện việc local search với các hệ số phạt tăng lên 10 lần với hi vọng sẽ cho ra một lời giải hợp lệ. Lưu ý, nếu thực hiện lựa chọn sửa chữa, chỉ khi sửa được lời giải ta mới cho lại nó vào quần thể.

2.7. Lựa chọn tham số

Các hằng số cố định: $n^{\text{ELITE}} = 4$, $n^{\text{CLOSEST}} = 5$, $\lambda = 40$.

Ban đầu, ω^Q và ω^{TW} được gán bằng 1. Sau mỗi 100 lần lặp, các tham số được tăng lên 20% nếu như có ít hơn 15% lời giải hợp lệ, và giảm 15% nếu có nhiều hơn 25% lời giải hợp lệ. Bên cạnh đó, nếu không tìm được lời giải nào hợp lệ, các tham số được tăng lên 100%, để tránh việc không tìm được lời giải trong một thời gian quá lâu. Sử dụng cơ chế này tốt hơn dùng penalty lớn từ đầu, do dùng penalty to khiến thuật toán hội tụ quá nhanh về nghiệm cục bộ.

Cách điều chỉnh tham số:

Có hành trình dài (>25 khách)	Có khách hàng với ràng buộc thời gian rộng (>70%)	Hành động
v		$\theta = 15\%, \mu = 25, \Gamma = 40$, tăng μ và Γ lên 5 sau 10000 lần lặp
x	X	$\theta = 100\%, \mu = 25, \Gamma = 40$, tăng μ lên 5 sau 10000 lần lặp
x	v	$\theta = 100\%, \mu = 25, \Gamma = 20$, tăng μ lên 5 sau 20000 lần lặp

Sau 10000 lần lặp mà không có cải tiến nào, ta reset lại quần thể và vẫn giữ nguyên tham số cũ.

2.8. Tổng kết

```
Initialize population with random solutions improved by local search;
while number of iterations without improvement < It_NI and time < T_MAX do
    Select parent solutions P1 and P2;
    Apply the crossover operator on P1 and P2 to generate an offspring C;
    Educate offspring C by local search;
    Insert C into respective subpopulation;
    if C is infeasible then
        With 50% probability, repair C (local search) and insert it into respective
         subpopulation;
    if maximum subpopulation size reached then
        Select survivors;
    Adjust penalty coefficients for infeasibility;
Return best feasible solution;
```

Algorithm 1: HGS-CVRP

3. Reference

- Thi Diem Chau LE, Clustering algorithm for a vehicle routing problem with time windows
- Yuichi Nagata, A penalty-based edge assembly memetic algorithm for the vehicle routing problem with time windows