ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

LUẬN VĂN THẠC SĨ

Phương pháp tìm kiếm lân cận rộng thích ứng
cho bài toán định tuyến xe

Đại học Quốc gia Hà Nội
Trường Đại học Khoa học Tự nhiên
Khoa Toán-Cơ-Tin học

Giảng viên hướng dẫn: TS. Hoàng Nam Dũng

Học viên: Nguyễn Mạnh Linh

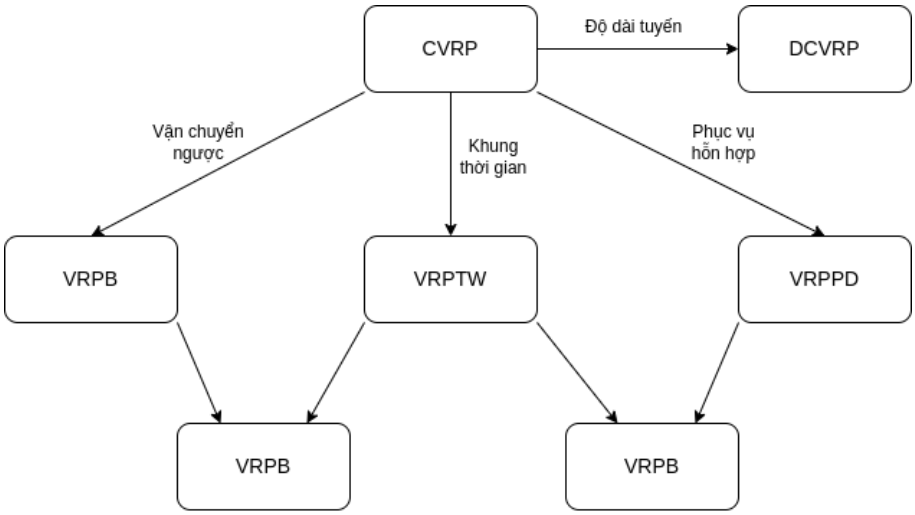
Mục lục

1 Mở đầu

2 Định nghĩa và một số kí hiệu

Trong chương này, chúng ta sẽ đưa ra định nghĩa chính tắc cho lớp các bài toán định tuyến xe. Các định nghĩa này được xây dựng theo ngôn ngữ của lý thuyết đồ thị được đưa ra bởi Toth (2002) [5].

Các bài toán được mô tả thuộc lớp VRP (*vehicle routing problem*) bao gồm *định tuyến xe với ràng buộc tải trọng* - CVRP (*capacited VRP*), *định tuyến xe với ràng buộc khung thời gian* - VRPTW (*VRP with time windows*), *định tuyến xe với lấy và giao hàng* - VRPPD (*VRP with pickup and delivery*).



Hình 2.1: Các bài toán, biến thể của VRP

2.1 Định nghĩa bài toán

2.1.1 CVRP

Trước hết ta xem xét mô hình cho bài toán *nguyên bản*: *bài toán định tuyến xe với ràng buộc tải trọng*. Một cách tự nhiên, tại sao không phải là VRP (không ràng buộc)? Bạn sẽ thấy rằng nếu không có bất kì ràng buộc nào thì một xe có thể phục vụ tất cả các yêu cầu và bài toán VRP sẽ suy biến về TSP (*travelling salesman problem*). Ít nhất ràng buộc về tải trọng là thực tế và giữ cho mỗi xe chỉ phục vụ được một số yêu cầu nhất định (trong trường hợp số yêu cầu không quá nhỏ cũng như tải trọng của xe là quá lớn).

Gọi $G = (V, A)$ là một độ thị đầy đủ với $V = \{0, ..., n\}$ là tập nút và A là tập các cung. Các nút $i = 1, ..., n$ đại diện cho các yêu cầu hay khách hàng cần phục vụ, nút 0 là kho hàng. Đôi khi, kho hàng cũng được biểu diễn bằng nút $n + 1$.

Một số không âm được gọi là chi phí c_{ij} đại diện cho mỗi cung $(i, j) \in A$. Nói cách khác c_{ij} là chi phí cần bỏ ra để di chuyển từ nút i tới nút j. Trong bài toán này và hầu hết các bài toán định tuyến ta không định nghĩa cạnh (i, i) nên có thể gán $c_{ii} = \infty$ với $i \in V$.

Nếu đồ thị là có hướng thì ma trận chi phí c là bất đối xứng, khi đó ta có bài toán CVRP bất đối xứng ACVRP (*asymetric CVRP*). Ngược lại nếu $c_{ij} = c_{ji}$ với mọi $(i, j) \in A$ ta có bài toán CVRP đối xứng SCVRP (*symetric CVRP*) và các cung của A được thay thế bằng tập cách cạnh vô hướng E. Với một cạnh $e \in E$, ta định nghĩa $\alpha(e)$ và $\beta(e)$ là nút bắt đầu và kết thúc của cạnh.

Đồ thị G phải là đồ thị kết nối mạnh và nhìn chung ta giả thiết đồ thị G là đầy đủ. Với một nút i, gọi $\Delta^+(i)$ là tập ra của i (*forward star*), được định nghĩa là tập các nút j mà cung $(i, j) \in A$, nói cách khác đây là tập các nút có thể tiếp cận trực tiếp từ nút i. Tương tự như vậy, $\Delta^- i$ là tập vào của i (*backward star*), được định nghĩa là tập các nút j mà cung $(j, i) \in A$ hay là tập các nút tiếp cận trực tiếp tới nút i. Với một tập nút con $S \subseteq V$, gọi $\delta(S)$ là tập các cạnh $e \in E$ chỉ có một hoặc cả hai đầu mút thuộc S. Để thuận tiện, khi xét một nút $i \in V$, ta viết $\delta(i)$ thay cho $\delta(\{i\})$.

Trong hầu hết các bài toán thực tế, ma trận chi phí thỏa mãn bất đẳng thức tam giác

$$c_{ij} + c_{jk} \geq c_{ik} \quad \forall i, j, k \in V \tag{2.1}$$

Nói cách khác việc đi trực tiếp từ nút i tới nút j luôn tốn ít chi phí hơn là đi gián tiếp. Với nhiều thuật toán, bất đẳng thức tam giác là điều kiện cần, điều này có thể được đảm bảo bằng cách thêm một đại lượng dương lớn (hợp lý) vào chi phí của mỗi cung. Ta chú ý thêm rằng nếu chi phí của mỗi cung thuộc đồ thị bằng với chi phí của đường đi ngắn nhất giữa hai đầu mút của cung thì mà trận chi phí thỏa mãn bất đẳng thức tam giác.

Trong nhiều trường hợp, tập các nút nằm trên một mặt phẳng, vị trí của chúng được cho bởi tọa độ và chi phí c_{ij} của mỗi cung $(i, j) \in A$ là khoảng cách Euclide giữa hai điểm ứng với nút i và j. Khi đó, ma trận chi phí là đối xứng và thỏa mãn bất đẳng thức tam giác. Bài toán này được gọi là *Euclidian* SCVRP.

Mỗi khách hàng i có một nhu cầu (về tải trọng) là d_i và nhu cầu của kho $d_0 = 0$. Với một tập nút $S \subseteq V$, ta kí hiệu $d(S) = \sum_{i \in S} d_i$ là tổng nhu cầu của tập.

Một tập hợp K đại diện cho các xe, mỗi xe có tải trọng C và sẵn sàng ở kho. Ta giả thiết $d_i \leq C$ với mỗi $i = 1, ..., n$. Giả thiết này là cần thiết để mỗi khách hàng đều được phục vụ. Mỗi xe phục vụ nhiều nhất một tuyến và ta giả thiết K không nhỏ hơn K_{min} với K_{min} là số xe ít nhất cần để phục vụ toàn bộ khách hàng.

Với một tập $S \subseteq V \setminus \{0\}$, ta gọi $r(S)$ là số xe ít nhất để phục vụ toàn bộ khách hàng thuộc tập S. Chú ý rằng $r(V \setminus \{0\}) = K_{min}$.

CVRP yêu cầu tìm một tập chính xác K các chu trình đơn (mỗi chu trình ứng với một tuyến đường) với tổng chi phí của tất cả các cung thuộc các chu trình này là nhỏ nhất. Lời giải phải thỏa mãn các ràng buộc sau:

(i) Mỗi chu trình đều đi qua nút ứng với kho hàng

(ii) Mỗi nút ứng với một khách hàng được đi qua bởi đúng một chu trình

(iii) Tổng nhu cầu của các khách hàng trong mỗi chu trình không được vượt quá tải trọng của xe.

2.1.2 CVRPTW

Bài toán định tuyến xe với ràng buộc thời gian - VRPTW (*VRP with time windows*) là một mở rộng của CVRP. Trong đó ngoài ràng buộc về tải trọng cho mỗi xe, mỗi khách hàng i bị ràng buộc bởi một khoảng thời gian $[a_i, b_i]$ được gọi là khung thời gian hay cửa sổ thơi gian (*time window*). Thời gian phục vụ khách hàng i là s_i. Thời gian di chuyển từ nút i tới nút j là t_{ij} với mỗi cung $(i, j) \in A$ hay t_e với $e \in E$. Ngoài ra nếu xe đến nút i sớm thì phải chờ đến thời gian a_i mới được phục vụ. Nếu xe đến nút i muộn hơn thì khách hàng sẽ không được phục vụ.

Thường thì ma trận chi phí và ma trận thời gian di chuyển là như nhau, hơn nữa các xe được giả thiết đều xuất phát từ kho tại thời điểm 0. Ràng buộc thời gian dẫn tới mỗi tuyến đường là có hướng (có thứ tự đi đến các nút) ngay cả khi ma trận chi phí là đối xứng. Chính vì thế, VRPTW thường được mô tả như một bài toán bất đối xứng.

VRPTW yêu cầu tìm một tập chính xác K chu trình đơn với tổng chi phí là nhỏ nhất, thỏa mãn các ràng buộc sau đây:

(i) Mỗi chu trình đều đi qua nút ứng với kho hàng

(ii) Mỗi nút ứng với một khách hàng được đi qua bởi đúng một chu trình

(iii) Tổng nhu cầu của các khách hàng trong mỗi chu trình không được vượt quá tải trọng của xe

(iv) Với mỗi khách hàng i, thời gian bắt đầu phục vụ phải nằm trong khung thời gian $[a_i, b_i]$ và xe ngừng phục vụ sau khoảng thời gian s_i

VRPTW là bài toán NP-khó, nó là trường hợp tổng quát của CVRP. Nếu ta đặt $a_i = 0$ và $b_i = \infty$ với $i \in V \setminus \{0\}$ thì VRPTW suy biến về CVRP. Ngoài ra ta cũng thu được biến thể TSP với ràng buộc thời gian (TSPTW) nếu $C \geq d(V)$ và $K = 1$.

2.1.3 VRPPD

Một biến thể khác nữa của CVRP là bài toán định tuyến xe với lấy và giao hàng (*VRP with pickup and delivery - VRPPD*). Trong đó, mỗi khách hàng i có thêm hai đại lượng đặc trưng nữa là d_i và p_i lần lượt là nhu cầu lấy và giao tại khách hàng i. Đôi khi chỉ một đại lượng $d_i = d_i - p_i$ được sử dụng cho mỗi khách hàng i để chỉ lượng nhu cầu chênh lệch giữa việc lấy và giao hàng (có thể là số âm). Với mỗi khách hàng i, gọi O_i là nút đại diện cho việc giao hàng và D_i là nút đại diện cho điểm lấy hàng.

Giả thiết rằng, tại mỗi điểm khách hàng, điểm giao được phục vụ trước điểm lấy. Do đó, tải hiện tại của một xe trứ khi tới điểm đã cho là tải ban đầu trừ đi tổng nhu cầu đã giao cộng với tổng nhu cầu đã lấy.

VRPPD yếu cầu tìm chính xác một tập K các chu trình đơn với tổng chi phí là nhỏ nhất, thỏa mãn các ràng buộc sau đây:

(i) Mỗi chu trình đều đi qua nút ứng với kho hàng

(ii) Mỗi nút ứng với một khách hàng được đi qua bởi đúng một chu trình

(iii) Tải hiện tại của xe trong suốt quá trình phục vụ không âm và không được vượt quá tải trọng của xe

(iv) Với mỗi khách hàng i, khách hàng O_i khác với kho phải được phục vụ trong cùng một tuyến và trước khách hàng i

(v) Với mỗi khách hàng i, khách hàng D_i khác với kho phải được phụ vụ trong cùng một tuyến và sau khách hàng i.

VRPPD là trường hợp tổng quát của CVRP. Nếu ta đặt $O_i = D_i = 0$ và $p_i = 0$ cho mọi $i \in V$ thì VRPPD suy biến về CVRP. Hơn nữa nếu đặt $K = 1$ thì ta thu được TSP với lấy và giao hàng (*TSP with pickup and delivery - TSPPD*).

2.2 Mô hình toán học

Chương này trình bày biểu diễn toán học cho bài toán VRPTW. Trong luận văn này, tác giả tập trung giải quyết VRPTW, từ đó ta cũng có thể giản ước về CVRP

cũng như tổng quát với VRPPD (*VRP with pickup and delevery*) hoặc PDPTW (*pickup and delivery with time window*). Như đã trình bày ở chương trước VRPTW là một mở rộng của CVRP với ràng buộc khung thời gian. Trong đó mỗi khách hàng i được ràng buộc bởi một khung thời gian $[a_i, b_i]$. Xe không được đến i tại thời điểm $t_i > b_i$, ngoài ra nếu đến sớm hơn thởi điểm a_i hay $t_i < a_i$ thì xe cần phải chờ tới thời điểm a_i để phục vụ khách hàng. Thời gian phục vụ của khách hàng i là s_i.

VRPTW là bài toán NP-khó, việc tìm lời giải hay nghiệm tối ưu (chính xác) gần như là bất khả thi. Để dễ hình dung, xét bài toán VRP, với số lượng khách hàng $n = 100$, và chỉ một xe, số lượng lời giải là $n! \approx 10^{158}$. Nếu ta có số CPU ước tính bằng toàn bộ số nguyên tử trong vũ trụ $n_{CPU} \approx 10^{80}$, thời gian nhỏ nhất là thời gian Plank $t_p \approx 5.39 \times 10^{-44}$. Để kiểm tra toàn bộ lời giải có phải nghiệm tối ưu ta cần thời gian $T \approx 10^{158} \times 5.39 \times 10^{-44}/10^{80} \approx 5.39 \times 10^{34}$. Để so sánh, tuổi của vũ trụ được ước tính khoảng 4.33×10^{17}. Nghĩa là ta sẽ mất thời gian lớn gấp cỡ *một trăm triệu tỉ* lần tuổi của cả vũ trụ! [1]

Như đã trình bày ở chương trước, VRPTW được định nghĩa trên đồ thị $G = (V, A)$, kho hàng được biểu diễn bởi nút 0 và $n + 1$. Một tuyến thỏa mãn là một đường đi trên đồ thị G bắt đầu từ 0 và kết thúc ở $n + 1$. Nếu kho hàng được biểu diễn chỉ bởi nút 0 thì tuyến thỏa mãn là một đơn chu trình trên đồ thị G chứ nút 0. Khung thời gian của nút 0 và $n + 1$ là $[a_0, b_0] = [a_{n+1}, b_{n+1}] = [E, L]$, trong đó E và L lần lượt là thời gian sớm nhất rời kho và thời gian muộn nhất trở về kho. Ngoài ra, thời gian phục vụ và nhu cầu của kho đều được đặt bằng 0, hay $s_0 = s_{n+1} = 0$ và $d_0 = d_{n+1} = 0$. Lời giải chấp nhận được chỉ tồn tại nếu $a_0 = E \leq \min_{i \in V \setminus \{0\}}\{b_i - t_{0i}\}$ và $b_{n+1} = L \geq \min_{i \in V \setminus \{0\}}\{a_i + s_i + t_{0i}\}$. Chú ý rằng, cung $(i, j) \in A$ có thể được bỏ đi nếu không thỏa mãn ràng buộc thời gian $a_i + s_i + t_{ij} > b_j$ hoặc vi phạm ràng buộc về tải trọng $d_i + d_j > C$. Cuối cùng nếu mục tiêu chính là giảm thiểu số lượng xe thì cung $(0, n + 1)$ với $c_{0,n+1} = t_{0,n+1} = 0$ phải được thêm vào A.

Tiếp theo, chúng ta trình bày một mô hình toán cho VRPTW với hai biến: biến x_{ijk} (*flow variable*) với $(i, j) \in A, k \in K$ nhận giá trị 1 nếu xe k đi trực tiếp từ nút i tới nút j và 0 nếu ngược lại. Biến w_{ik} với $i \in V, k \in K$ là thời gian bắt đầu phục vụ khách hàng i bởi xe k. VRPTW được mô hình một cách chính tắc như sau theo Toth (2002) [5]:

$$\min \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{2.2}$$

[1] Slides của Thibaut Vidal (SOICT, Nha Trang 2017)

Với ràng buộc:

$$\sum_{k\in K}\sum_{j\in\Delta^+(i)} x_{ijk} = 1 \qquad \forall i \in N, \quad (2.3)$$

$$\sum_{j\in\Delta^+(0)} x_{0jk} = 1 \qquad \forall k \in K, \quad (2.4)$$

$$\sum_{i\in\Delta^-(j)} x_{ijk} - \sum_{i\in\Delta^+(j)} x_{jik} = 0 \qquad \forall k \in K, j \in N, \quad (2.5)$$

$$\sum_{i\in\Delta^-(n+1)} x_{i,n+1,k} = 1 \qquad \forall k \in K, \quad (2.6)$$

$$x_{ijk}(w_{ik} + s_i + t_{ij} - w_{jk}) \leq 0 \qquad \forall k \in K, (i,j) \in A, \quad (2.7)$$

$$a_i \sum_{j\in\Delta^+(i)} x_{ijk} \leq w_{ik} \leq b_i \sum_{j\in\Delta^+(i)} x_{ijk} \qquad \forall k \in K, i \in N, \quad (2.8)$$

$$E \leq w_{ik} \leq L \qquad \forall k \in K, i \in \{0, n+1\}, \quad (2.9)$$

$$\sum_{i\in N} d_i \sum_{j\in\Delta^+(i)} x_{ijk} \leq C \qquad \forall k \in K, \quad (2.10)$$

$$x_{ijk} \geq 0 \qquad \forall k \in K, (i,j) \in A, \quad (2.11)$$

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, (i,j) \in A. \quad (2.12)$$

Hàm mục tiêu trong phương trình (2.2) biểu diễn tổng chi phí của tất cả các tuyến đường. Tập $N = V \setminus \{0\}$ biểu diễn cho tập khách hàng.

- Ràng buộc (2.3) đảm bảo rằng mỗi khách hàng chỉ được phục vụ bởi một xe.
- Ràng buộc (2.4) đảm bảo rằng mỗi xe phải xuất phát từ kho hàng.
- Ràng buộc (2.5) đảm bảo rằng trên một tuyến, nếu khách hàng i được phục vụ thì trước và sau đó đều có một khách hàng khác được phục vụ hoặc trước và sau đó là kho hàng. Nói cách khác, khách hàng i phải ở giữa tuyến.
- Ràng buộc (2.6) đảm bảo rằng mỗi xe phải trở về kho hàng.
- Ràng buộc (2.7) đảm bảo về khung thời gian khi xe đi từ khách hàng i tới khách hàng j. Nếu xe k đi từ khách hàng i tới khách hàng j thì thời gian bắt đầu phục vụ khách hàng i cộng với thời gian phục vụ khách hàng i cộng với thời gian di chuyển từ khách hàng i tới khách hàng j phải nhỏ hơn hoặc bằng thời gian bắt đầu phục vụ khách hàng j. Dấu bằng xảy ra khi xe đến j sau thời điểm a_j (khách hàng j được phục vụ luôn), nếu đến sớm hơn thì xe phải chờ để phục vụ khách hàng.

- Ràng buộc (2.8) đảm bảo rằng thời gian bắt đầu phục vụ khách hàng i bởi xe k nằm trong khung thời gian $[a_i, b_i]$.
- Ràng buộc (2.9) đảm bảo rằng thời gian bắt đầu phục vụ khách hàng i bất kì phải nằm trong khoảng thời gian từ sớm nhất xuất phát từ kho và muộn nhất về kho.
- Ràng buộc (2.10) đảm bảo rằng tổng tải của mỗi xe không được vượt quá tải trọng tối đa C.
- Ràng buộc (2.11) và (2.12) đảm bảo điều kiện nhị phân của *flow variable* x_{ijk}.

Ta có thể nhận thấy rằng, ràng buộc (2.8) ép $w_{ik} = 0$ nếu như khách hàng i không được phục vụ bởi xe k. Điều kiện nhị phân trong ràng buộc (2.12) cho phép ràng buộc (2.7) được thay thế bởi

$$w_{ik} + s_i + t_{ij} - w_{jk} \leq (1 - x_{ijk})M_{ij} \quad \forall k \in K, (i,j) \in A, \tag{2.13}$$

với M_{ij} là các hằng số rất lớn. Hơn nữa M_{ij} có thể thay bằng $\max\{b_i + s_i + t_{ij} - a_j, 0\}$ với $(i,j) \in A$ và như vậy ta chỉ cần kiểm tra ràng buộc (2.7) và (2.13) cho các cung $(i,j) \in A$ thỏa mãn $M_{ij} > 0$. Mặt khác, khi $\max\{b_i + s_i + t_{ij} - a_j, 0\} = 0$ các điều kiện này được thỏa mãn với mọi w_{ik}, w_{jk} và x_{ijk}.

Chúng ta không cần đưa ra mô hình cho CVRP nữa, bởi ta có thể bỏ qua các ràng buộc về thời gian ở điều kiện từ (2.7) đến (2.9). Khi đó VRPTW suy biến về CVRP như đã trình bày ở những phần trước đó. Tác giả cũng không đưa ra mô hình cho VRPPD hay PDPTW để tránh sự phức tạp. VRPTW vừa đủ để ta có một mô hình đẹp và thực tế.

3 Một số phương pháp cho VRP

Trong chương này, chúng ta sẽ xem xét một số khái niệm cần thiết và phương pháp để giải (lớp) bài toán định tuyến xe. Trong suốt chặng đường hơn 50 năm của bài toán VRP, có rất nhiều phương pháp được nghiên cứu và thực nghiệm từ các thuật toán giải chính xác đến các thuật toán xấp xỉ. Ba lớp thuật toán được trình bày bao gồm *thuật toán chính xác*, *heuristics cổ điển* và *metaheuristics*. Lớp các thuật toán được trình bày một cách khái quát theo Laporte, Gilbert (2009) [2]. Cuối cùng tác giả đưa ra lựa chọn và đi sâu vào thuật toán tìm kiếm lân cận rộng thích ứng - ALNS (*Adaptive Large Neighborhood Search*) để giải quyết bài toán VRPTW trong luận văn này.

3.1 Thuật toán chính xác

3.1.1 Nhánh và cận

Một trong những thuật toán chính xác được nghiên cứu sớm nhất là *nhánh và cận*, lần đầu xuất hiện trong bài báo *"An Algorithm for the Vehicle Dispatching Problem"* của N. Christofides và S. Eilon năm 1969 [1]

3.1.2 Quy hoạch động

Eilon, Watson-Gandy và Christofides (1971) [1] đưa ra lời giải cho bài toán VRP bằng phương pháp quy hoạch động. Gọi $c(S)$ là chi phí tối ưu của một tuyến ứng với tập nút $S \subseteq V \setminus \{0\}$. Mục tiêu là cực tiểu hóa $\sum_{r=1}^{m} c(S_r)$ trên tất cả các quy hoạch khả dĩ $\{S_1, ..., S_m\}$ của $V \setminus \{0\}$. Gọi $f_k(U)$ là chi phí nhỏ nhất có thể đạt được khi sử dụng k xe cho một tập con U của $V \setminus \{0\}$. Ta có:

$$f_k(U) = \begin{cases} c(U) \quad \text{nếu } k = 1, \\ \min_{U^* \subseteq U \subseteq V \setminus \{0\}} \{f_{k-1}(U \setminus U^*) + c(U^*)\} \quad \text{nếu } k > 1. \end{cases} \tag{3.1}$$

Chi phí tối ưu là $f_m(V \setminus \{0\})$ và các tuyến là các phân hoạch của $V \setminus \{0\}$ theo phương trình (3.1).

3.1.3 Công thức dòng xe và thuật toán

Công thức 2-chỉ số cho bài toán VRP được nghiên cứu đầu tiên bởi Laporte, Nobert (1983) [3] và Laporte, Nobert, Desrochers (1985) [4] và mở rộng công thức TSP

cổ điển của Dantzig, Fulkerson, Johnson (1954) [**dantzig1954solution**]. Gọi x_{ij} là biến 0-1-2 bằng số lần một xe đi qua cung (i, j). Bài toán được mô hình hóa như sau:

$$\text{cực tiểu} \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{3.2}$$

với ràng buộc:

$$\sum_{j=1}^{n} x_{0j} = 2m, \tag{3.3}$$

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \qquad (k \in V \setminus \{0\}), \tag{3.4}$$

$$\sum_{i,j\in S} x_{ij} \leq |S| - \nu(S) \qquad (S \subseteq V \setminus \{0\}), \tag{3.5}$$

$$x_{0j} = 0, 1, 2 \qquad (j \in V \setminus \{0\}), \tag{3.6}$$

$$x_{ij} = 0, 1 \qquad (i, j \in V \setminus \{0\}), \tag{3.7}$$

trong đó $\nu(S)$ là cận dưới của số lượng xe cần thiết để phục vụ tập S.

3.1.4 Công thức dòng hàng và thuật toán

Trong công thức dòng hàng, biến y_{ij} (hoặc y_{ijk}) định nghĩa tải (lượng hàng) của xe mang theo trên cung (i, j). Ví dụ được trình bày bởi Gavish, Graves (1979) [**gavish1978travelling**], tuy nhiên các tác giả không đưa ra kết quả tính toán. Các ví dụ gần đây hơn được nghiên cứu bởi Baldacci, Hadjiconstantinou, Mingozzi (2004) [**baldacci2004exact**] dựa trên mô hình TSP của Finke, Claus, Gunn (1984) [**finke1984two**]. Công thức cho một đồ thị mở rộng $\bar{G} = (\bar{V}, \bar{E})$, với $\bar{V} = V \cup \{(i, n+1) : i \in V\}$. Một tuyến được định nghĩa là một đường đi có hướng từ 0 đến $n+1$. Biến nhị phân x_{ij} bằng 1 khi và chỉ khi cạnh (i, j) được chọn vào tuyến. Biến y_{ij} định nghĩa tải của xe trên cung (i, j) và $y_{ji} = Q - y_{ij}$ biểu diễn xe rỗng trên cung (j, i) mỗi khi $x_{ij} = 1$. Công thức dòng hàng được mô hình hóa như sau:

$$\text{cực tiểu} \sum_{(i,j)\in E} c_{ij}x_{ij} \tag{3.8}$$

với ràng buộc:

$$\sum_{j\in\bar{V}}(y_{ji}-y_{ij})=2q_i \qquad (i\in V\setminus\{0\}), \quad (3.9)$$

$$\sum_{j\in V\setminus\{0\}} y_{0j}=\sum_{i\in V\setminus\{0\}} q_i, \quad (3.10)$$

$$\sum_{j\in V\setminus\{0\}} y_{j0}=mQ-\sum_{i\in V\setminus\{0\}} q_i, \quad (3.11)$$

$$\sum_{j\in V\setminus\{0\}} y_{n+1,j}=mQ, \quad (3.12)$$

$$y_{ij}+y_{ji}=Qx_{ij} \qquad ((i,j)\in\bar{E}), \quad (3.13)$$

$$\sum_{i<k} x_{ik}+\sum_{j>k} x_{kj}=2 \qquad (k\in V\setminus\{0\}), \quad (3.14)$$

$$y_{ij}\geq 0, y_{ji}\geq 0 \qquad ((i,j)\in\bar{E}), \quad (3.15)$$

$$x_{ij}=0,1 \qquad ((i,j)\in\bar{E}). \quad (3.16)$$

Bài toán này được giải bằng *branch-and-cut* với các bất đẳng thức VRP được biểu diễn theo các biến x_{ij}

3.1.5 Công thức phân hoạch tập hợp và thuật toán

Công thức phân hoạch tập hợp đơn giản của VRP lần đầu được nghiên cứu bởi Balinski, Quandt (1964) [**balinski1964integer**]. Gọi r là một tuyến, a_{ir} là hệ số nhị phân có giá trị bằng 1 khi và chỉ khi nút $i\in V\setminus\{0\}$ thuộc tuyến r, gọi c^* là chi phí tối ưu của tuyến r và gọi y_k la biến nhị phân bằng 1 khi và chỉ khi tuyến r được dùng trong lời giải tối ưu. Bài toán được mô hình hóa như sau:

$$\text{cực tiểu } \sum_r c_r^* y_r \quad (3.17)$$

với ràng buộc:

$$\sum_r a_{ir}=1 \qquad (i\in V\setminus\{0\}), \quad (3.18)$$

$$\sum_r y_r=m, \quad (3.19)$$

$$y_r=0,1 \qquad (\text{mọi } r). \quad (3.20)$$

Nói một cách chặt chẽ thì ràng buộc (3.19) không phải một phần của công thức phân hoạch tập hợp chuẩn, tuy nhiên nó được sử dụng bởi hầu hết các nhà nghiên cứu trong trường hợp VRP.

3.2 Heuristics cổ điển

Nhìn chung thì các thuật toán giải chính xác khó đảm bảo hiệu năng trong thực tế khi mà các tập dữ liệu ngày các lớn và các doanh nghiệp cần phục vụ khách hàng một cách nhanh chóng và tiết kiệm. Thực tế người ta cần tìm ra một (số) lời giải chập nhận được đủ tốt trong một khoảng thời gian "hợp lý". Từ những năm 1964 cho đến 1990, rất nhiều heuristics được nghiên cứu. Một số ít là đưa ra thuật toán hoàn toàn mới còn hầu hết là cải tiến thuật toán đã có.

3.2.1 Thuật toán tiết kiệm

Thuật toán tiết kiệm được đưa ra bởi Clark, Wright (1964) [**clarke1964scheduling**], mô tả và cài đặt khá đơn giản nhưng vẫn đưa ra được nghiệm tốt. Chính vì thế, thuật toán này được sử dụng rất rộng rãi. Thuật toán bắt đầu với nghiệm ban đầu với n tuyến $(0, i, 0)$ với $i \in V \setminus \{0\}$. Tại mỗi vòng lặp thuật toán nối tuyến kết thúc với i với một tuyến khác bắt đầu với j cực đại hóa đại lượng *tiết kiệm* $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ và lới giải mới thỏa mãn các ràng buộc. Quá trình kết thúc khi không thể nối các tuyến vào nữa.

Một số cải tiến được đề xuất, ví dụ như nhân c_{ij} với một trọng số dương λ (Golden, Magnanti, Nguyen (1977) [**golden1977implementing**]), tối ưu tuyến đường hợp nhất toàn cục thông qua việc sử dụng thuật toán phù hợp (Altinkemer, Gavish (1991) [**altinkemer1991parallel**] và Wark, Holt (1994) [**wark1994repeated**]), tăng tốc tính toán (Paessens (1988) [])...

3.2.2 Phân cụm trước, định tuyến sau

Heuristic phân cụm trước, định tuyến sau của Fisher, Jaikumar (1981) [] trước hết đặt m tâm và phân cụm sao cho tổng khoảng cách từ các nút đến tâm của nó là nhỏ nhất và thỏa mãn về ràng buộc tải trọng. Sau đó trên mỗi cụm, tuyến đường được thiết lập bằng cách giải bài toán TSP. Một vài chiến thuật để khởi tạo cũng như lựa chọn tâm cụm được trình bày trong Baker, Sheasby (1999) []

3.2.3 Heuristics cải tiến

3.3 Metaheuristics

Metaheuristics có thể được phân loại thành tìm kiếm lân cận, tìm kiếm phổ biến và cơ chế học. Hầu hết các thuật toán metaheuristics cho VRP đều dựa trên tìm kiếm lân cận và là các phương pháp cải tiến. Các thuật toán tốt nhất khá mạnh mẽ ngay cả khi nghiệm khởi tạo có chất lượng thấp. Một xu hướng chung là thay vì sử dụng chỉ một thuật toán, người ta thường kết hợp nhiều thuật toán lại với nhau. Các thuật toán như vậy được gọi là thuật toán lai.

Tiếp theo tác giả trình bày ý tưởng chính của một số lớp thuật toán metaheurictics.

3.3.1 Tìm kiếm cục bộ

Về cơ bản, tìm kiếm lân cận cố gắng "khám phá" không gian nghiệm bằng cách "di chuyển" quanh nghiệm hiện tại tới một nghiệm khác trong vùng lần cận của nó. Một số phương pháp có thể kể đến như *tabu search* (Glover (1986) [**glover1986future**]), *simulated annealing* (Kirkpatrick, Gelatt, and Vecchi (1983) [**kirkpatrick1983optimization**]), *deterministic annealing* (Dueck, Scheurer (1990) [**dueck1990threshold**]; Dueck (1993) [**dueck1993new**]), *variable neighbourhood search* (Mladenović, Hansen (1997) [**mladenovic1997variable**]), *large neighbourhood search*(Shaw (1998) [**shaw1998using**]) và *adaptive large neighbourhood search* (Ropke, Pisinger (2006) [**ropke2006adaptive**]). Các thành phần chính của tìm kiếm lân cận là các quy tắc để xác định vùng lân cận của một nghiệm và cơ chế để khám phá vùng lân cận đó.

Trong *tabu search* không gian nghiệm được khám phá bằng cách di chuyển từ nghiệm hiện tại đến nghiệm tốt nhất trong một tập con của lân cận của nghiệm đó. Để tránh việc lặp lại các nghiệm, các nghiệm được gán một thuộc tính gắn với nghiệm hiện tại để không được chọn trong một số lần lặp tiếp theo. Một nghiệm trở thành nghiệm tốt nhất trong số các nghiệm đã biết có thuộc tính gắn với thuộc tính hiện tại. Nguyên lý này được trình bày đầu tiên bởi Cordeau, Gendreau, Laporte (1997) [**cordeau1997tabu**] và hiện nay được biết đến như là phương pháp tìm kiếm dựa trên thuộc tính (Derigs, Kaiser (2007) [**derigs2007applying**]).

Trong *simulated annealing* một gnhiệm x được chọn ngẫu nhiên trong lân cận $N(x_t)$ của nghiệm hiện tại x_t tại vòng lặp t. Nếu hàm mục tiêu f cực tiểu, ta gán $x_{t+1} := x$ với $f(x_{t+1}) \leq f(x_t)$. Ngược lại gán $x_{t+1} := x$ với một xác suất p_t và gán $x_{t+1} := x_t$ với xác suất $1 - p_t$. Trong đó, p_t là một hàm giảm theo t và $f(x) - f(x_t)$.

Trong *deterministic annealing*, nghiệm x cũng được chọn ngẫu nhiên trong lân cận $N(x_t)$. Trong thuật toán *threshold-accepting* (Dueck, Scheurer (1990) [**dueck1990threshold**]), $x_{t+1} := x$ khi $f(x) < f(x_t)+\theta_1$, với θ_1 là một trọng số dương; ngược lại gán $x_{t+1} := x_t$. Trong *record-to-record travel* (Dueck (1993) [**dueck1993new**]), với nghiệm tốt nhất hiện tại x^*, gán $x_{t+1} := x$ nếu $f(x) \leq \theta_2 f(x^*)$, với θ_2 là một trọng số dương; ngược lại gán $x_{t+1} := x_t$.; ngược lại gán $x_{t+1} = x_t$.

Trong *Variable neighbourhood search* (Mladenović, Hansen (1997) []), tác giả xem xét một dánh sách được sắp xếp của các lân cận. Thuật toán bắt đầu với một lân cận và chuyển qua lân cận tiếp theo cho đến khi đạt tới nghiệm tối ưu cục bộ. Việc tìm kiếm được khởi tạo lại khi một nghiệm tốt hơn được tìm thấy hoặc tất cả các lân cận đã dược xét qua. *Very large-scale neighbourhood search - LNS* bỏ đi và tạo lại một (vài) phần của nghiệm hiện tại để tìm kiếm nghiệm tốt hơn. Nguyên lí này giống như cơ chế hủy và tạo lại được trình bày bởi Shaw (1998) [**shaw1998using**]. *Adaptive large neighbourhood search - ALNS* (Ropke, Pisinger (2006) [**ropke2006adaptive**]) được biết đến như là một phiên bản mạnh mẽ hơn của *large neighbourhood search*, trong đó các thuật toán hủy hay tạo lại được lựa chọn một cách linh hoạt và thích ứng với trạng thái hiện tại của hệ. LNS và ALNS là cảm hứng chính cho luận văn này. Trong các phần tiếp theo tác giả sẽ trình bày chi tiết về ALNS.

3.3.2 Tìm kiếm quần thể

Tìm kiếm quần thể hoạt động với một quần thể các nghiệm. Thuật toán di truyền (Holland (1975) [**holland1975adaptation**]) là ví dụ tốt nhất cho mô hình này. Tại mỗi vòng lặp, một vài nghiệm cha dược trích xuất từ quần thể hiện tại và kết hợp để tạo ra các nghiệm con. Nghiệm con sau đó được thay bằng những phần tệ nhất trong quần thể nếu điều này cải thiện nghiệm tốt nhất hiện tại. Về cơ bản, thuật toán áp dụng đa dạng hóa các cơ chế, gọi là đột biến cho thế hệ nghiệm con trước khi xem xét việc đưa chúng vào quần thể.

3.3.3 Cơ chế học

Cơ chế học vay mượn ý tưởng từ trí tuệ nhân tạo với mạng thần kinh (neural networks). Thuật toán học hỏi kinh nghiệm và điều chỉnh các trọng số qua các vòng lặp. Ứng dụng với VRP được trình bày bởi Ghaziri (1991) [**ghaziri1991solving**]; Schumann, Retzko (1995) [**schumann1995self**]. Thuật toán tối ưu đàn kiến cũng là một dạng khác của cơ chế học. Nó bắt chước hành vi của đàn kiến trong việc tìm thức ăn và để lại vết trên đường đi. Theo thời gian, vết được để lại nhiều hơn trên

đường đi ngắn nhất và qua đó, kiến đi theo con đường này. Ứng dụng đầy đủ được trình bày bởi Reimann, Doerner, Hartl (2004) [**reimann2004d**]

4 Phương pháp tìm kiếm lân cận

Như đã trình bày trong chương trước, hầu hết các thuật toán hiện đại đều dựa trên tìm kiếm lân cận và thuộc lớp cải tiến. Thuật toán không cố gắng để đưa ra ngay một nghiệm tối ưu từ khi tất cả các yêu cầu chưa được phục vụ. Thay vào đó, ta xuất phát từ một nghiệm chấp nhận được (thường được khởi tạo nhanh như thuật toán tham lam chẳng hạn), sau đó qua mỗi vòng lặp, nghiệm được cải thiện dần cho đến khi điều kiện dừng được thỏa mãn. Tại mỗi vòng lặp, thuật toán cố gắng khám phá không gian nghiệm bằng cách xét các lân cận của nghiệm hiện tại để tìm một nghiệm tối ưu cục bộ.

Chiến thuật này cần ta định nghĩa rõ ràng khái niệm "lân cận", "tối ưu cục bộ", "tiêu chí chấp nhận nghiệm" và "điều kiện dừng". Lân cận được hiểu là một nghiệm chấp nhận được (theo nghĩa thỏa mãn các ràng buộc) với một thay đổi không quá lớn từ nghiệm hiện tại. Ví dụ, một vài yêu cầu từ tuyến này được trao đổi với tuyến khác hoặc được chuyển hẳn sang một tuyến khác hay là tạo một tuyến mới. Nghiệm tối ưu cục bộ là nghiệm tốt nhất ta có thể tìm được trong lân cận như vậy. Tuy nhiên, nếu ta tiếp tục tìm kiếm từ nghiệm tối ưu cục bộ thì đôi khi thuật toán bị "bẫy" tại nghiệm đó. Cụ thể hơn, thuật toán trải qua nhiều vòng lặp mà chỉ xét các nghiệm lân cận của một tối ưu cục bộ hoặc độ cải thiện là rất chậm. Chính vì vậy có nhiều tiêu chí chấp nhận hay tiêu chí lựa chọn nghiệm để thoát khỏi bẫy này. Ví dụ như *tabu search*, *simulated annealing*, *threshold-accepting* hay *record-to-record travel* đã được trình bày ở chương trước. Nhìn chung, thay vì tiếp tục tìm kiếm từ một tối ưu cục bộ, chúng ta đưa vào hàm mục tiêu một hệ số phạt (nhỏ) nào đó để có thể tìm kiếm từ một nghiệm tệ hơn nghiệm tối ưu cục bộ một chút. Điều kiện dừng thường được áp dụng là số vòng lặp tối đa hoặc thời gian tối đa.

4.1 Tìm kiếm lân cận rộng

Phương pháp tìm kiếm lân cận rộng (Large neighbourhood search - LNS) được trình bày bởi Shaw (1998) [**shaw1998using**] thuộc lớp các thuật toán tìm kiếm lân cận. LNS dựa trên việc liên tục bỏ đi yêu cầu và tối ưu lại nghiệm. Nghĩa là một số yêu cầu được bỏ đi khỏi tuyến (theo một tiêu chí nào đó) và được thêm lại vào các tuyến (khác) với mục đích làm giảm hàm mục tiêu.

Thuật toán giả định rằng lời giải ban đầu s đã có, ví dụ được tạo bằng một heuristic đơn giản. Tham số thứ 2 q xác định phạm vi tìm kiếm.

Algorithm 1 LNS Heuristic

Require: $s \in solutions, q \in \mathbb{N}$

1: solution $s_{best} = s$;
2: **repeat**
3: $s' = s$;
4: remove q requests from s';
5: reinsert removed requests into s';
6: **if** $f(s') < f(s)$ **then**
7: $s_{best} = s'$;
8: **if** $accept(s', s)$ **then**
9: $s = s'$;
10: **until** stop-criterion met
11: **return** s_{best};

Dòng 4 và 5 của thuật toán là phần thú vị của heuristic. Ở dòng 4, một số yêu cầu được loại bỏ khỏi phương án hiện tại *s'*, các yêu cầu lại được thêm vào ở dòng 5. Hiệu năng cũng như sự mạnh mẽ của heuristic phụ thuộc vào sự lựa chọn chiến thuật bỏ và thêm lại các yêu cầu. Trong các bài trước đó về LNS cho VRPTW và PDPTW (Shaw (1997) [**shaw1997new**]; Bent, Van Hentenryck (2003) [**bent2003two**]) các phương pháp *gần tối ưu* được sử dụng để thêm lại các yêu cầu. Mặc dù các cách thêm lại yêu cầu heuristic thường có chất lượng kém, nhưng chất lượng của LNS heurustic lại rất tốt, bởi vì các bước xấu được tạo ra bởi heuristic thêm lại yêu cầu dẫn đến sự đa dạng hóa hiệu quả của quá trình tìm kiếm.

Phần còn lại của thuật toán cập nhật phương án tốt nhất (hiện tại) và tìm kiếm phương án mới (tốt hơn). Một tiêu chí chấp nhận đơn giản là chấp nhận tất cả các phương án cải tiến. Tiêu chí này đã được sử dụng trong các triển khai LNS trước đó (Shaw (1997) [**shaw1997new**]).

Dòng 10 kiểm tra điều kiện dừng đã đạt được hay chưa.

Tham số $q \in \{0, ..., n\}$ xác định kích cỡ tập lân cận. Nếu $q = 0$ thì có nghĩa là không có bước tìm kiếm nào hết vì không có yêu cầu nào được bỏ đi. Mặt khác nếu $q = n$, thì bài toán được giải luôn qua mỗi vòng lặp. Nói chung, q càng lớn thì càng dễ di chuyển quanh không gian nghiệm, tuy nhiên khi q lớn dần lên thì bước thêm lại yêu cầu sẽ chậm hơn.

Ngoài ra thay vì xem xét quá trình LNS như là một chuỗi hành động xoá và thêm lại, chúng ta có thể coi quá trình này là chuỗi hành động sửa lỗi và tối ưu. Cách nhìn này giúp chúng ta áp dụng chiến thuật này không chỉ cho bài toán VRP mà còn có thể áp dụng cho các bài toán tối ưu tổ hợp khác nữa. Chính vì tính chất mạnh mẽ này, tác giả đã lựa chọn LNS làm nền tảng cho phương pháp tìm kiếm lân cận trong luận văn này.

4.1.1 Thuật toán hủy

Phương pháp xóa tệ nhất

Cho 1 yêu cầu i được phục vụ bởi vài xe trong tập nghiệm s, chi phí của yêu cầu *cost* được định nghĩa như sau: $cost(i, s) = f(s) - f_{-i}(s)$ với $f_{-i}(s)$ là chi phí của nghiệm mà không có yêu cầu i (yêu cầu được xóa mà không chuyển đến hàng chờ). Chiến thuật ở đây là ta xóa đi những yêu cầu có chi phí cao và cố gắng thêm lại vào các tuyến với chi phí ít hơn.

Tuy nhiên chúng ta không xóa đi chính xác các yêu cầu có chi phí cao nhất mà thay vào đó chúng ta chọn ngẫu nhiên 1 yêu cầu có chi phí cao. Điều này được thực hiện để tránh việc xóa các yêu cầu có chi phí cao nhất liên tục và thuật toán bị bẫy trong một nghiệm tối ưu cục bộ.

Algorithm 2 Worst Removal

Require: $s \in solutions, q \in \mathbb{N}, p \in \mathbb{R}_+$

1: **while** $q > 0$ **do**
2: Array: L = All planned requests i, sorted by descending *cost(i,s)*;
3: choose a random number y in the interval $[0, 1)$;
4: request: $r = L\left[y^p |L|\right]$;
5: remove r from solution s;
6: $q = q - 1$;

Phương pháp xóa ngẫu nhiên

Thuật toán xóa ngẫu nhiên đơn giản chọn ngẫu nhiên q yêu cầu và loại bỏ chúng khỏi nghiệm hiện tại. Kỹ thuật này có thể coi là 1 trường hợp đặc biệt của phương pháp xóa Shaw với $p = 1$.

Phương pháp xóa Shaw

Phương pháp xóa này được phát triển bởi Shaw (1997, 1998) []. Cách trình bày trong phần này đã được chỉnh sửa lại để phù hợp với VRPTW. Ý tưởng chung là xóa bỏ các yêu cầu có "liên quan", vì chúng ta hy vọng sẽ dễ dàng thêm lại các yêu cầu tương tự với nhau để tạo ra các nghiệm mới có thể tốt hơn. Nếu chúng ta chọn xóa bỏ các yêu cầu rất khác nhau, thì sau đó, việc thêm các yêu cầu mới sẽ không nhận lại được điều gì do các yêu cầu này có thể chỉ được thêm vào tại vị trí ban đầu của chúng hoặc ở các vị trí tệ. Mức độ liên quan giữa 2 yêu cầu i và j được định nghĩa dựa trên chỉ số độ liên quan $R(i, j)$. Chỉ số này càng thấp thì 2 yêu cầu càng "giống" nhau.

Chỉ số độ tương đồng được sử dụng bao gồm phụ thuộc vào ba điều kiện: khoảng cách, thời gian, khối lượng. Các điều kiện này được đánh trọng số và ký hiệu lần lượt là φ, χ và ψ. Chỉ số độ tương đồng được tính như sau:

$$R(i, j) = \varphi d_{ij} + \chi|t_i - t_j| + \psi|l_i - l_j| \tag{4.1}$$

d_{ij} là khoảng cách từ i tới j, t_i là thời gian khi đến địa điểm i, l_i là tải của xe tại i.

Mức độ liên quan được sử dụng để xóa các yêu cầu được mô tả trong Algorithm 3. Ban đầu, một yêu cầu được chọn ngẫu nhiên. Trong các vòng lặp tiếp theo, thuật toán sẽ thực hiện xóa các yêu cầu "giống" với các yêu cầu đã được xóa. Tham số $p \geqslant 1$ biểu diễn cho sự ngẫu nhiên trong cách lựa chọn yêu cầu (p càng thấp thì độ ngẫu nhiên càng cao).

Algorithm 3 Shaw Removal

Require: $s \in solutions, q \in \mathbb{N}, p \in \mathbb{R}_+$

1: request: r = a randomly selected request from s;
2: set of requests: $\mathbb{D} = \{r\}$;
3: **while** $|\mathbb{D}| < q$ **do**
4: r = a randomly selected request from $\mathbb{D}$;
5: Array: L = an array containing all request from s not in $\mathbb{D}$;
6: sort L such that $i < j \Rightarrow R(r, L[i]) < R(r, L[j])$;
7: choose a random number y from the interval [0, 1);
8: $\mathbb{D} = \mathbb{D} \cup L[y^p|L|]$
9: remove the requests in $\mathbb{D}$ from s;

Tinh thần của thuật toán Shaw là cố gắng bỏ đi top các yêu cầu có độ đo liên quan $R(i, j)$ nhỏ. Top các yêu cầu này được kiểm soát bằng tham số p. Tuy nhiên việc

sắp xếp các yêu cầu (mảng L) theo thứ tự tăng dần của $R(r, L[i])$ là phép toán tốn chi phí. Trong thực tế cài đặt chúng ta có thể sử dụng cấu trúc dữ liệu *min heap* để lấy ra top các yêu cầu có độ đo liên quan nhỏ nhất.

Hủy tuyến

Thuật toán hủy tuyến được tác giả đề xuất trong luận văn này. Tinh thần chung của thuật toán là chúng ta cố gắng bỏ đi toàn bộ yêu cầu trên một hoặc một vài tuyến đường nào đó. Việc bỏ đi cả tuyến cũng tương đương với việc làm giảm số xe cần để phục vụ khách hàng. Mục tiêu của CVRPTW là giảm tổng khoảng cách di chuyển và nghiệm thu được phải thỏa mãn các ràng buộc về tải trọng, số xe cũng như khung thời gian. Tuy nhiên, trong thực tế vận hành của một doanh nghiệp, giảm bớt được số xe phục vụ là điều rất có ý nghĩa bởi chi phí mua (thuê) xe là đắt đỏ so với việc di chuyển xa hơn một vài phần trăm.

Áp dụng tư tưởng *greedy*, tác giả cho rằng các tuyến nên được bỏ đi là các tuyến có chi phí trung bình trên mỗi khách hàng cao. Chúng ta kì vọng có thể thêm lại các khách hàng đó vào các tuyến khác "thoáng" hơn hay là chi phí trung bình trên mỗi khách hàng thấp hơn.

$$\text{avg\_cost}(r, s) = \frac{\text{cost}(r, s)}{\text{size}(r) - 1} \tag{4.2}$$

với r là tuyến được chọn, $\text{cost}(r, s)$ là chi phí của tuyến r trong nghiệm s, $\text{size}(r)$ là số phần tử của tuyến r. Lưu ý rằng tuyến bắt đầu và kết thúc tại nút 0.

Ngoài ra để tránh việc bỏ đi tuyến "tệ" một cách cứng nhắc, tác giả đưa vào một nhiễu nhỏ.

$$\text{avg\_cost}(r, s) = \frac{\text{cost}(r, s)}{\text{size}(r) - 1} + \lambda p d_{max} \tag{4.3}$$

với d_{max} là khoảng cách lớn nhất giữa hai yêu cầu, p là một số ngẫu nhiên trong khoảng $(-1, 1)$ và λ là một hằng số điều khiển.

Ngoài ra, chúng ta cũng không muốn bỏ đi các tuyến đang phục vụ nhiều yêu cầu vì khi thêm lại thuật toán sẽ mất rất nhiều thời gian. Chính vì thế, tác giả chỉ bỏ đi những tuyến có số yêu cầu ít hơn số yêu cầu trung bình trên mỗi tuyến của nghiệm hiện tại.

Cuối cùng, ta bỏ đi toàn bộ yêu cầu trên tuyến có số yêu cầu ít hơn số yêu cầu trung bình trên tất cả các tuyến và có chi phí trung bình trên mỗi yêu cầu cao nhất.

4.1.2 Thuật toán sửa

4.1.3 Phương pháp tham lam cơ bản

Heuristic tham lam cơ bản (*basic greedy*) là 1 kỹ thuật xây dựng đơn giản. Nó thực hiện tối đa n lần lặp, chèn thêm 1 yêu cầu trong mỗi bước. Với $\Delta f_{i,k}$ biểu diễn cho sự thay đổi hàm mục tiêu bằng cách chèn thêm yêu cầu i vào tuyến đường k tại vị trí mà giá trị tăng thêm của hàm mục tiêu là nhỏ nhất. Nếu không chèn yêu cầu i vào tuyến đường k thì ta đặt $\Delta f_{i,k} = \infty$ và $c_i = \min_{k \in K}\{\Delta f_{i,k}\}$. Nói cách khác, c_i là chi phí khi chèn thêm yêu cầu i vào vị trí tốt nhất của nghiệm hiện tại, gọi là vị trí chi phí nhỏ nhất. Cuối cùng, ta chọn yêu cầu i với: $\min_{i \in U} c_i$ và chèn nó vào vị trí chi phí nhỏ nhất. Quá trình này tiếp tục cho đến khi tất cả các yêu cầu được thêm hoặc không còn yêu cầu nào nữa.

Trong mỗi vòng lặp, thuật toán chỉ thay đổi một tuyến đường (tuyến mà yêu cầu mới được thêm vào), và ta không cần phải tính toán lại chi phí chèn thêm trong tất cả các tuyến đường khác. Điểm này giúp tăng hiệu năng cho thuật toán. Đặc biệt nếu hàm mục tiêu là tổng quãng đường đi chuyển thì chúng ta có thể tính toán rất nhanh chi phí tăng thêm này. Giả sửa ta cần chèn thêm yêu cầu u và giữa hai yêu cầu i và j trong tuyến đường k. Khi đó, chi phí tăng thêm của việc chèn thêm yêu cầu u vào giữa i và j là: $\Delta f_{u,i,j,k} = d_{iu} + d_{uj} - d_{ij}$.

Dễ dàng nhận thấy vấn đề với cách tiếp cận này là nó thường trì hoãn việc đặt các yêu cầu có chi phí cao cho các lần lặp cuối cùng, nơi chúng ta không có nhiều cơ hội cho việc chèn thêm yêu cầu vì nhiều tuyến đường đều đã kín.

4.1.4 Phương pháp tham lam với nhiễu ngẫu nhiên

Để giải quyết một phần hạn chế của phương pháp tham lam cơ bản, ta có thể thêm một chút nhiễu vào hàm mục tiêu. Điều này giúp thuật toán có thể không chọn chèn yêu cầu vào tuyến làm tăng chi phí ít nhất mà có thể là ít thứ hai hoặc thứ ba.

$$\text{noise} = \lambda p d_{\max} \tag{4.4}$$

với $d_{\max}$ là khoảng cách lớn nhất giữa hai yêu cầu, p là một số ngẫu nhiên trong khoảng $(-1, 1)$ và λ là một hằng số điều khiển. Sau đó ta gán

$$\Delta f_{u,i,j,k} := \Delta f_{u,i,j,k} + \text{noise} \tag{4.5}$$

4.1.5 Phương pháp regret heuristic

Regret heuristic đã được sử dụng bởi Potvin và Rousseau (1993) [] cho VRPTW. Phương pháp này cố gắng cải thiện nhược điểm của kỹ thuật tham lam bằng cách kiểm tra lại kết quả sau khi chọn chèn thêm yêu cầu. Đặt $x_{ik} \in \{1, ..., m\}$ là biến biểu diễn tuyến đường cho yêu cầu i có chi phí chèn thêm vào thấp thứ k, điều này có nghĩa là $\Delta f_{i,x_{ik}} \leqslant \Delta f_{i,x_{ik'}}$. Sử dụng ký hiệu này, ta có thể biểu diễn c_i từ phần 4.1.3 như sau: $c_i = \Delta f_{i,x_{i1}}$. Trong phương pháp này, chúng ta có thể định nghĩa 1 giá trị *regret* $c_i^* = \Delta f_{i,x_{i2}} - \Delta f_{i,x_{i1}}$. Nói cách khác, giá trị regret là khoảng cách giữa chi phí của việc chèn thêm yêu cầu vào tuyến đường tốt nhất so với tuyến đường tốt thứ 2. Trong mỗi vòng lặp, thuật toán chọn ra yêu cầu i thỏa mãn điều kiện: $max_{i \in U}\{c_i^*\}$. Với điều kiện này, yêu cầu được đảm bảo thêm vào vị trí có chi phí nhỏ nhất. Điều này khiến cho các ràng buộc bị phá vỡ. Nói cách khác, thuật toán sẽ thực hiện chèn yêu cầu vào phương án nếu không sẽ hối tiếc khi không thực hiện điều này.

Phương pháp này có thể mở rộng 1 cách tự nhiên để định nghĩa 1 lớp các phương pháp regret heuristic: phương pháp k-regret heuristic là 1 phương pháp mà mỗi lần thêm yêu cầu vào phương án cần phải thỏa mãn điều kiện:

$$\max_{i \in U}\{\sum_{j=1}^{k}(\Delta f_{i,x_{ij}} - \Delta f_{i,x_{i1}})\} \tag{4.6}$$

Nếu 1 vài yêu cầu không thể được chèn thêm vào ít nhất $m - k + 1$ tuyến đường thì yêu cầu đó sẽ được chèn vào số lượng tuyến đường ít nhất (có thể là 1). Các ràng buộc bị phá vỡ bởi việc lựa chọn yêu cầu với chi phí chèn tốt nhất. Yêu cầu được chèn vào vị trí ít chi phí nhất. Với $k > 2$ thì thuật toán sẽ tính toán chi phí của việc thêm vào 1 yêu cầu qua k tuyến đường tốt nhất và chèn yêu cầu mà khoảng cách chi phí giữa việc thêm nó vào tuyến đường tốt nhất với tuyến đường tốt thứ $k-1$ là lớn nhất. So sánh với 2-regret heuristic, thuật toán với giá trị lớn của k khám phá ra sớm hơn khả năng bị giới hạn khi thêm 1 yêu cầu.

Với việc xét top-k vị trí tốt nhất của mỗi yêu cầu, hiệu năng của *regret-k* sẽ không tốt bằng *basic greedy*. Để dễ hình dung, với n vị trí có thể chèn yêu cầu, trong mỗi bước lặp của thuật toán, với *basic greedy* ta cần duyệt $O(n)$ lần qua các vị trí để tìm ra vị trí chèn tốt nhất; với *regret-k* nếu sử dụng heap ta mất $O(k \times n)$ lần duyệt để lấy ra top-k và $O(k)$ lần duyệt nữa để lấy ra yêu cầu mà khoảng cách Δf trong top-k lớn nhất. Trong thực tế cài đặt, tác giả sử dụng đến $k = 5$ và regret-k chứng tỏ được nó là một phương pháp mạnh và đáng để đánh đổi một chút về mặt hiệu năng.

4.1.6 Tiêu chí chấp nhận nghiệm

Các tiêu chí chấp nhận nghiệm được tóm lược qua bảng 4.1

Phương pháp	Mô tả
Random Walk	Mọi nghiệm s' đều được chấp nhận
Greedy Acceptance	Nghiệm s' được chấp nhận nếu chi phí của nó là nhỏ hơn so với nghiệm hiện tại
Simulated Annealing	Mọi nghiệm cải thiện s' được chấp nhận. Nếu $c(s') > c(s)$ thì s' được chấp nhận với xác suất $\exp\{\frac{c(s)-c(s')}{T}\}$ với T là nhiệt độ. Nhiệt độ T giảm sau mỗi vòng lặp với một hệ số Φ
Threshold Acceptance	Nghiệm s' được chấp nhận nếu $c(s') - c(s) < T$ với T là ngưỡng, ngưỡng này được giảm sau mỗi vòng lặp với hệ số Φ
Great Deluge Algorithm	Nghiệm s' được chấp nhận nếu $c(s') < L$ với một ngưỡng L, ngưỡng này chỉ giảm nếu nghiệm được chấp nhận, và giảm với hệ số Φ

Bảng 4.1: Tiêu chí chấp nhận nghiệm

Trong thực tế, tác giả cài đặt tất cả các tiêu chí chấp nhận trên, tuy nhiên mô phỏng luyện kim (*Simulated Annealing*) cho hiệu năng và chất lượng nghiệm tốt nhất. Nhiệt độ ban đầu được cấu hình trước là T_{start}. Qua mỗi vòng lặp, nhiệt độ được giảm đi $T := T \times \Phi$ với $0 < \Phi < 1$ được gọi là hệ số làm lạnh. Việc lựa chọn T_{start} phụ thuộc vào cấu hình bài toán, do đó, thay vì đặt T_{start} là một tham số cố định, ta sẽ tính toán nó dựa trên cầu hình đầu vào bằng cách sửa dụng nghiệm khởi tạo ban đầu. Chi phí của nghiệm khởi tạo là z (bỏ qua chi phí của các yêu cầu trong hàng chờ). Nhiệt độ ban đầu được đặt sao cho nghiệm tệ hơn $w\%$ được chấp nhận với xác suất 0.5. w lúc này là tham số điều khiển cho nhiệt độ ban đầu.

4.2 Tìm kiếm lân cận rộng thích ứng

Tìm kiếm lân cận rộng thích ứng (*adaptive large neighbourhood search - ALNS*) được giới thiệu bởi Ropke, Pisinger (2006) [**ropke2006adaptive**] là một mở rộng

của LNS. Thay vì chỉ sử dụng một chiến thuật hủy và một chiến thuật thêm lại yêu cầu như LNS, ALNS cho phép lựa chọn nhiều toán tử hủy và thêm lại. Việc này cho phép thuật toán tìm kiếm không gian nghiệm một cách linh hoạt hơn và khó bị bẫy ở một nghiệm tối ưu cục bộ. Điểm thú vị của ALNS là các thuật toán hủy và thêm lại không được chọn một cách ngẫu nhiên mà lựa chọn có trọng số phụ thuộc vào trạng thái (nghiệm hiện tại) của bài toán.

4.2.1 Lựa chọn phương pháp xóa và thêm lại

Để lựa chọn phương pháp xóa và thêm lại, ta gán cho mỗi heuristic một trọng số khác nhau và sử dụng nguyên tắc "bánh xe lựa chọn". Nếu có k heuristic với trọng số $w_i, i \in \{1, ..., k\}$, ta chọn heuristic j với xác suất:

$$p_j = \frac{w_j}{\sum_{i=1}^{k} w_i} \tag{4.7}$$

Lưu ý rằng việc lựa chọn thuật toán xóa và thêm lại là độc lập với nhau. Các trọng số này có thể được thiết lập thủ công và không đổi trong suốt vòng đời của việc tìm kiếm hoặc nó có thể được điều chỉnh tự động để "thích ứng" với trạng thái hiện tại của hệ. Một cách điều chỉnh các tham số này tự động được trình bày ngay sau đây.

4.2.2 Điều chỉnh tham số tự động

Trọng số được điều chỉnh mỗi khi có nghiệm mới được chấp nhận. Ý tưởng chung là theo dõi một điểm số đại diện cho độ hiệu quả của thuật toán trong các vọng lặp gần đây. Điểm số càng cao thì thuật toán được chọn càng hiệu quả. Quá trình tìm kiếm được chia thành nhiều bước, mỗi bước là một số vòng lặp. Điểm của mỗi heuristic được đặt là 0 khi bắt đầu và được tăng thêm $\sigma_1, \sigma_2, \sigma_3$ tùy thuộc vào tình huống Trong mỗi bước, thao tác xóa và thêm lại được cập nhật một lượng như nhau vì ta không chắc sự "cải thiện" nghiệm đến từ việc xóa hay thêm lại yêu cầu. Mỗi khi kết thúc bước, ta tính toán lại trọng số mới sử dụng các điểm sô trên. Gọi ω_{ij} là trọng số của heuristic i trong bước j. Ta đánh các trọng số như nhau tại bước đầu tiên, sau đó khi bước j kết thúc, ta tính lại trọng số cho tất cả các heuristic để sử dụng cho bước $j + 1$ như sau:

$$\omega_{i,j+1} = \omega_{ij}(1 - r) + r\frac{\pi_i}{\theta_i} \tag{4.8}$$

Trong đó π_i là điểm số của heuristic i trong bước j và θ_i là số lần ta cố gắng sử dụng heuristic i trong bước j. Tham số r là tham số điều khiển tốc độ điều chỉnh trọng số. Nếu $r = 0$, nghĩa là chúng ta không sử dụng điểm để điều chỉnh trọng số

Tham số	Mô tả
σ_1	Hành động xóa-chèn cuối cùng dẫn đến một nghiệm mới tốt hơn nghiệm tốt nhất toàn cục
σ_2	Hành động xóa-chèn cuối cùng dẫn đến một nghiệm chưa được chấp nhận trước đó, chi phí tốt hơn chi phí của nghiệm hiện tại
σ_3	Hành động xóa-chèn cuối cùng dẫn đến một nghiệm chưa được chấp nhận trước đó, chi phí của nghiệm mới tệ hơn chi phí của nghiệm hiện tại nhưng thỏa mãn điều kiện chấp nhận nghiệm

Bảng 4.2: Tham số cập nhật trọng số

hay nói cách khác là các trọng số được giữ nguyên từ trong suốt quá trình tìm kiếm. Nếu $r = 1$, nghĩa là ta lấy điểm thu được từ bước gần nhất để quyết định trọng số.

Việc điều chỉnh trọng số như trên làm tăng xác suất chọn thuật toán xóa (chèn) đã mang lại hiệu quả ở bước trước đó. Về cơ bản, với việc sử dụng chiến thuật điều chỉnh trọng số như trên, ta kì vọng rằng các thuật toán xóa (chèn) đã hiệu quả ở bước trước thì cũng sẽ hiệu quả ở bước tiếp theo.

4.2.3 Cải tiến điều chỉnh tham số tự động

Như đã trình bày, với việc sử dụng chiến thuật lựa chọn tham số tự động như trên, ta kì vọng rằng thuật toán nếu đang hiệu quả thì nó sẽ tiếp tục hiệu quả. Tuy nhiên việc cộng một lượng cố định vào thuật toán đó về lâu dài (khi trải qua nhiều vòng lặp) thì trọng số của nó trở lên lớn dẫn đến xác suất lựa chọn thuật toán này cũng lớn theo. Chúng ta cũng chưa sử dụng yếu tố vòng lặp (hay thời gian chạy). Ý tưởng đơn giản là nếu rất "lâu" rồi ta mới có một thuật toán hiệu quả thì ta cũng nên điều chỉnh trọng số của nó theo "thời gian chờ đó". Giả sử sau m vòng lặp, chúng ta mới lại có một nghiệm được chấp nhận từ lần cuối cùng nghiệm được chấp nhận, trọng số của thuật toán sẽ được điều chỉnh một lượng tỉ lệ với $1 - e^{-\gamma m}$. Hàm exp được sử dụng để chuẩn hóa lượng này trong khoảng $(0, 1)$ khi m lớn hoặc nhỏ. Cuối cùng ta có biểu thức cho trọng số của thuật toán như sau:

$$\omega_{i,j+1} = \omega_{ij}(1 - r) + r\frac{\pi_i}{\theta_i} + \delta(1 - e^{-\gamma m}) \tag{4.9}$$

với δ (có thể âm hoặc dương) và γ (dương) là các tham số điều khiển.

5 Ứng dụng ALNS vào CVRPTW

6 Thực nghiệm và kết quả

Trong phần này, chúng ta sẽ xem xét kết quả thực nghiệm thu được khi áp dụng ALNS cho VRPTW với hai tập dữ liệu của Solomon (1987) và Homberger & Gehring (1999). Ở cả hai tập, các bộ dữ liệu được chia thành 3 loại C, R và RC. Với dữ liệu lớp C, các yêu cầu được phân thành các cụm rõ rệt, lớp R là hoàn toàn ngẫu nhiên và lớp RC là sự kết hợp của hai lớp trên. Tác giả chỉ xem xét các tập từ 100 yêu cầu trở lên (bỏ qua tập Solomon 25, 50 yêu cầu vì nhìn chung số lượng yêu cầu như vậy là quá nhỏ để nhận thấy sự khác biệt khi so sánh ALNS với các thuật toán khác).

Tất cả các thí nghiệm được chạy trên CPU Intel(R) Xeon(R) CPU E5-2680 v4 @ 2.40GHz với Ubuntu 22.04.1 LTS. Mã nguồn được viết bằng ngôn ngữ C++ và được biên dịch bằng GCC 11.4.0 với các tùy chọn tối ưu hóa -O3.

6.1 Chất lượng nghiệm

Trước tiên, chúng ta bắt đầu với tập dữ liệu Solomon (1987). Với các cấu hình loại C, ALNS cho nghiệm cách nghiệm tốt nhất đã biết trung bình 0.20% và 0.42% cho C1 và C2.

ins	cost	nv	bkcost	bknv	gap	ins	cost	nv	bkcost	bknv	gap
c101	828.94	10	827.30	10	0.20	c201	591.56	3	589.10	3	0.42
c102	828.94	10	827.30	10	0.20	c202	591.56	3	589.10	3	0.42
c103	828.06	10	826.30	10	0.21	c203	591.17	3	588.70	3	0.42
c104	824.78	10	822.90	10	0.23	c204	590.60	3	588.10	3	0.42
c105	828.94	10	827.30	10	0.20	c205	588.88	3	586.40	3	0.42
c106	828.94	10	827.30	10	0.20	c206	588.49	3	586.00	3	0.43
c107	828.94	10	827.30	10	0.20	c207	588.29	3	585.80	3	0.42
c108	828.94	10	827.30	10	0.20	c208	588.32	3	585.80	3	0.43
c109	828.94	10	827.30	10	0.20						
avg					0.20						0.42

Bảng 6.1: Kết quả đo với tập Solomon C

ins: cấu hình, cost: chi phí thu được với ALNS, nv: số xe được sử dụng, bkcost: chi phí tốt nhất đã biết, bknv: số xe tốt nhất đã biết, gap (%): khoảng cách so với nghiệm tốt nhất đã biết

Thí nghiệm được thiết lập có thời gian timeout 1 phút, chạy 5 lần và lấy kết quả tốt nhất. Tập C1 và C2 tương đối nhỏ và đã phân cụm nên trong thực tế thuật toán chạy rất nhanh để ra được nghiệm tối ưu và không có sự khác biệt giữa các lần chạy, trên CPU được thí nghiệm, ALNS mất dưới 1 giây để tìm ra nghiệm tối ưu.

instance	alns best	nv	bk cost	bk nv	gap (%)
r101	1,642.88	20	1,637.70	20	0.32
r102	1,472.81	18	1,466.60	18	0.42
r103	1,213.62	15	1,208.70	14	0.41
r104	976.61	11	971.50	11	0.53
r105	1,360.78	15	1,355.30	15	0.40
r106	1,239.37	13	1,234.60	13	0.39
r107	1,073.60	12	1,064.60	11	0.85
r108	944.44	10	932.10	10	1.32
r109	1,152.38	13	1,146.90	13	0.48
r110	1,078.59	12	1,068.00	12	0.99
r111	1,053.50	12	1,048.70	12	0.46
r112	955.68	10	948.60	10	0.75
avg					0.61

Bảng 6.2: Kết quả đo với tập Solomon R1

Tập R1 và R2 có các yêu cầu được tạo hoàn toàn ngẫu nhiên

instance	alns best	nv	bk cost	bk nv	gap (%)
r201	1,152.96	**7**	1,143.20	8	0.32
r202	1,035.32	**7**	1,029.60	8	0.42
r203	880.90	6	870.80	6	0.41
r204	743.91	**4**	731.30	5	0.53
r205	958.81	5	949.80	5	0.40
r206	883.92	5	875.90	5	0.39
r207	806.31	5	794.00	4	0.85
r208	948.57	4	701.00	4	1.77
r209	717.53	5	854.80	5	0.48
r210	909.32	**5**	900.50	6	0.99
r211	1,053.50	5	746.70	4	0.46
avg					1.38

Bảng 6.3: Kết quả đo với tập Solomon R2

instance	alns best	nv	bk cost	bk nv	gap (%)
rc101	1,623.58	16	1,619.80	15	0.23
rc102	1,461.23	14	1,457.40	14	0.26
rc103	1,266.62	11	1,258.00	11	0.69
rc104	1,136.91	10	1,132.30	10	0.41
rc105	1,518.58	16	1,513.70	15	0.32
rc106	1,376.99	13	1,372.70	12	0.31
rc107	1,211.11	12	1,207.80	12	0.27
rc108	1,118.13	11	1,114.20	11	0.35
avg					0.36

Bảng 6.4: Kết quả đo với tập Solomon RC1

instance	alns best	nv	bk cost	bk nv	gap (%)
rc201	1,274.61	**8**	1,261.80	9	1.02
rc202	1,099.54	**6**	1,092.30	8	0.66
rc203	931.16	5	923.70	5	0.81
rc204	788.66	4	783.50	4	0.66
rc205	1,157.66	7	1,154.00	7	0.32
rc206	1,060.50	**6**	1,051.10	7	0.89
rc207	966.08	6	962.90	6	0.33
rc208	785.73	4	776.10	4	1.24
avg					0.74

Bảng 6.5: Kết quả đo với tập Solomon RC2

7 Kết luận

Tài liệu tham khảo

[1] Nicos Christofides and Samuel Eilon. "An algorithm for the vehicle-dispatching problem". In: *Journal of the Operational Research Society* 20 (1969), pp. 309–318.

[2] Gilbert Laporte. "Fifty years of vehicle routing". In: *Transportation science* 43.4 (2009), pp. 408–416.

[3] Gilbert Laporte and Yves Nobert. "A branch and bound algorithm for the capacitated vehicle routing problem". In: *Operations-Research-Spektrum* 5 (1983), pp. 77–85.

[4] Gilbert Laporte, Yves Nobert, and Martin Desrochers. "Optimal routing under capacity and distance restrictions". In: *Operations research* 33.5 (1985), pp. 1050–1073.

[5] Paolo Toth and Daniele Vigo. *The vehicle routing problem*. SIAM, 2002.