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 MỘT SỐ LỚP BÀI TOÁN ĐỊNH TUYẾN PHƯƠNG TIỆN

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

Từ xưa tới nay, giao vận luôn là một trong những ngành đóng vai trò quan trọng trong nền kinh tế. Giao vận và quản lý chuỗi cung ứng đóng vai trò là một cầu nối giữa các đơn vị sản xuất và người tiêu dùng. Nó cũng là một trong những ngành có ảnh hưởng lớn đến sự phát triển của một quốc gia.

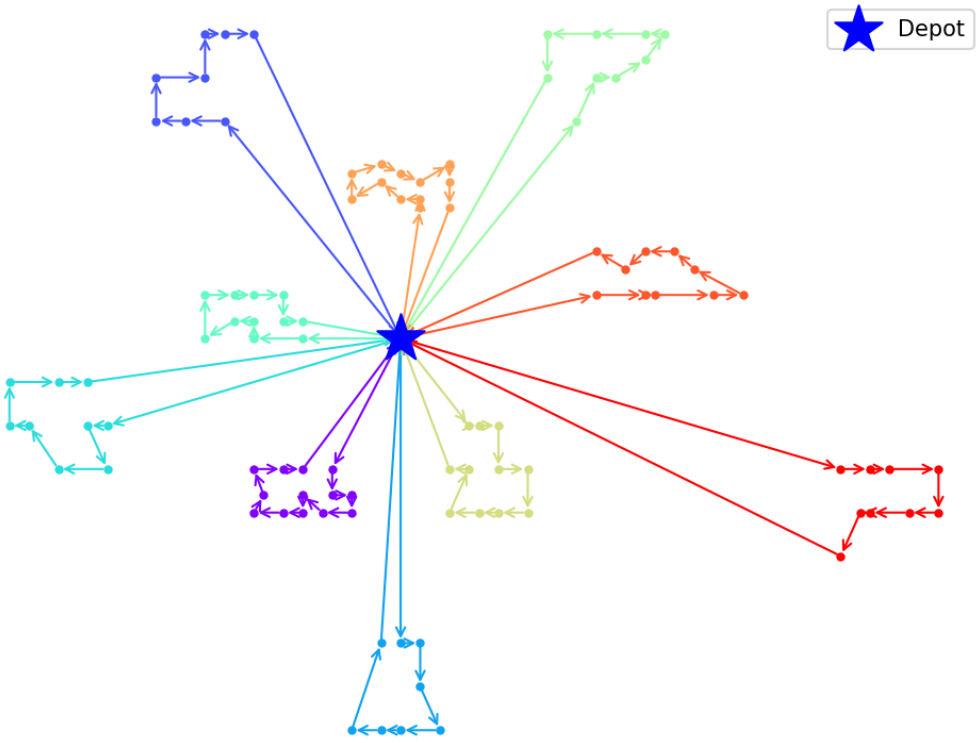
Từ những thập niên 90 của thế kỉ trước, thời kì bùng nổ của Internet đã thúc đẩy một hình thức bán hàng hoàn toàn mới, đó là bán hàng trực tuyến. Hàng loạt các sàn thương mại điện tử lớn ra đời, có thể kể đến như Amazon (Mỹ - 1994), Alibaba (Trung Quốc - 1999), Rakuten (Nhật Bản - 1997). Ngày nay các sàn thương mại điện tử này trở thành những công ty hàng đầu thế giới không chỉ ở lĩnh vực bán hàng mà là cả công nghệ. Việc phát triển vũ bão của các sàn thương mại điện tử dẫn đến số lượng hàng hóa được tiêu thụ trên toàn cầu tăng lên một cách đáng kinh ngạc so với bán hàng truyền thống. Logistic và quản lý chuỗi cung ứng là một trong những xương sống của thương mại điện tử cùng với *nền tảng công nghệ*, *nền tảng thanh toán* hay *chăm sóc khách hàng*... Để quản lý, giao vận số lượng đơn hàng lớn như vậy tới tay khách hàng, cách thức làm việc trong ngành logistic cũng phải thay đổi rất nhiều, áp dụng những công nghệ hiện đại hơn cách làm truyền thống.

Một trong những bài toán quan trọng nhất của logistic là bài toán vận tải hay định tuyến xe (hay người giao hàng). Với lượng xe có tải trọng hữu hạn, chúng ta cần định tuyến các xe này để giao hàng cho khách hàng ở các địa điểm khác nhau từ kho. Mục tiêu là tối ưu tổng quãng đường di chuyển của tất cả các xe để giảm chi phí vận hành. Lớp bài toán như vậy được gọi là *Vehicle Routing Problem - VRP*. Phiên bản đơn giản của VRP là TSP (*Travelling Salesman Problem*) hay bài toán người đưa hàng. Trong TSP, chúng ta có một người đưa hàng cần đi qua tất cả các địa điểm để giao hàng và quay trở về điểm xuất phát. TSP hay VRP là nhóm bài toán NP-hard, tức là không thể giải được trong thời gian đa thức. Tuy nhiên, với sự phát triển của các thuật toán hiện đại, chúng ta có thể tìm được nghiệm "gần" tối ưu các bài toán NP-hard với chất lượng tương đối cao trong thời gian hợp lý.

VRP lần đầu tiên được giới thiệu bởi Dantzig, George B và Ramser, John H (1959) [10], rất sớm trước thời kì bùng nổ Internet những năm 90! Tuy nhiên, cho tới nay, VRP vẫn thu hút được sự quan tâm lớn từ cộng đồng những nhà toán học hay khoa học máy tính bởi tính ứng dụng cao của nó. Với nhu cầu giao vận cùng lượng khách hàng cần phục vụ ngày càng tăng của các doanh nghiệp, việc tối ưu hóa chi phí vận hành là một vấn đề cấp thiết. Ngoài ra, việc tìm ra lời giải "gần" tối ưu cần diễn ra

nhanh chóng để tránh người giao hàng, hay xe phải chờ lâu để có được lộ trình cần thiết trước khi thực hiện giao hàng.

Trong luận văn này, tác giả nghiên cứu mô hình toán học của VRP cùng các biến thể của nó và các thuật toán giải quyết bài toán này. Tác giả sử dụng thuật toán ALNS (*Adaptive Large Neighborhood Search - Ropke, Stefan và Pisinger, David (2006)* [28]). ALNS được phát triển dựa trên LNS - *Large Neighbourhood Search - Shaw (1997, 1998)* [30], [31]. Hiệu năng và chất lượng nghiệm của ALNS được đánh giá trên các tập dữ liệu thực tế với số lượng yêu cầu (khách hàng) từ nhỏ tới rất lớn. Tác giả đề xuất một thuật toán hủy mới cho ALNS gọi là *Bad Route Removal* giúp giảm số xe sử dụng một cách nhanh chóng. Thêm vào đó, tác giả cũng đề xuất một hiệu chỉnh cho việc tự động điều chỉnh trọng số lựa chọn thuật toán trong ALNS giúp tăng hiệu năng và tiết kiệm tài nguyên so với ALNS gốc một cách đáng kể, thuật toán được đặt tên là B-ALNS (*Boosted - Adaptive Large Neighborhood Search*).

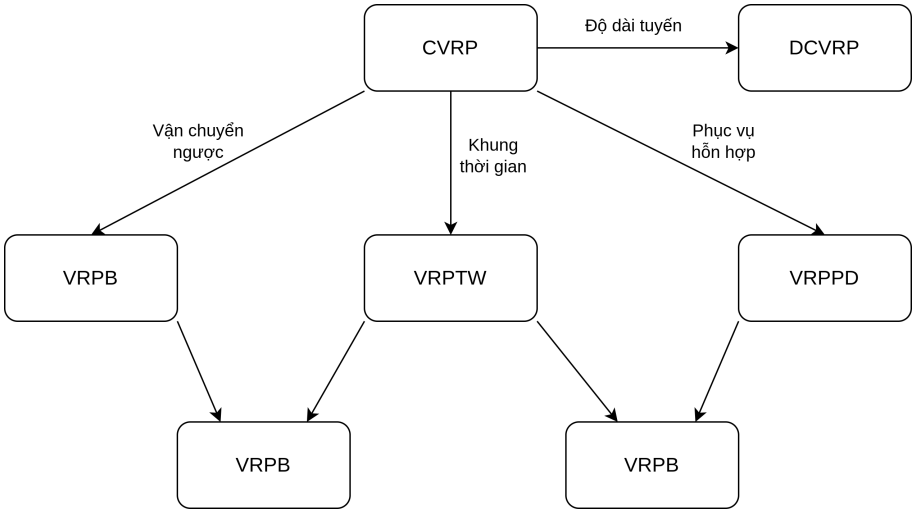


Hình 1.1: VRP với 10 xe phục vụ 100 khách hàng (cấu hình Solomon C101)

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) [32].

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) [32]:

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

[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) [22]. 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 [6]

3.1.2 Quy hoạch động

Eilon, Watson-Gandy và Christofides (1971) [6] 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) [23] và Laporte, Nobert, Desrochers (1985) [24] và mở rộng công thức TSP

cổ điển của Dantzig, Fulkerson, Johnson (1954) [9]. 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) [16], 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) [3] dựa trên mô hình TSP của Finke, Claus, Gunn (1984) [14]. 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) [4]. 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) [7], 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) [19]), tối ưu tuyến đường hợp nhất toàn cục thông qua việc sử dụng thuật toán phù hợp (Altinkemer, Gavish (1991) [1] và Wark, Holt (1994) [33]), tăng tốc tính toán (Paessens (1988) [26])...

3.2.2 Phân cụm trước, định tuyến sau

Heuristic phân cụm trước, định tuyến sau của Fisher, Jaikumar (1981) [15] 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) [2]

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) [18]), *simulated annealing* (Kirkpatrick, Gelatt, and Vecchi (1983) [21]), *deterministic annealing* (Dueck, Scheurer (1990) [13]; Dueck (1993) [12]), *variable neighbourhood search* (Mladenović, Hansen (1997) [25]), *large neighbourhood search*(Shaw (1998) [31]) và *adaptive large neighbourhood search* (Ropke, Pisinger (2006) [28]). 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) [8] và hiện nay được biết đến như là phương pháp tìm kiếm dựa trên thuộc tính (Derigs, Kaiser (2007) [11]).

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) [13]), $x_{t+1} := x$ 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) [12]), 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) [31]. *Adaptive large neighbourhood search - ALNS* (Ropke, Pisinger (2006) [28]) 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) [20]) 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) [17]; Schumann, Retzko (1995) [29]. 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) [27]

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) [31] 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) [30]; Bent, Van Hentenryck (2003) [5]) 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) [30]).

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) [28] 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 Thêm nhiễu khi 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:

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

với α (có thể âm hoặc dương) và γ (dương) là các tham số điều khiển, β là một số ngẫu nhiên trong khoảng $(0, 1)$. Trong phần thực nghiệm, tác giả sẽ chỉ ra sự vượt

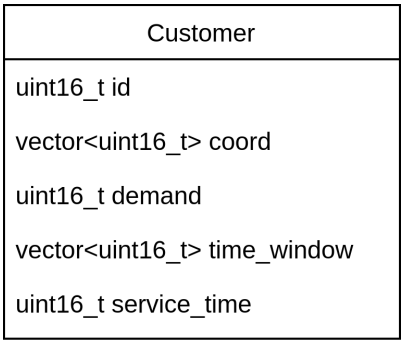
trội về hiệu năng khi thêm nhiễu vào việc điều chỉnh trọng số như trên. Chính vì vậy, thuật toán được đặt tên là B-ALNS (*Boosted - Adaptive Large Neighborhood Search*).

5 Ứng dụng ALNS vào VRPTW

Tiếp theo chúng ta sẽ xây dựng chương trình để giải VRPTW với ALNS. Mã giả của thuật toán đã được trình bày ở các phần trước. Ngoài ra, để tránh sự dài dòng không cần thiết, tác giả chỉ đưa ra những thành phần chính cùng với một số lưu ý quan trọng khi cài đặt. Chương trình được xây dựng theo tiêu chuẩn lập trình hướng đối tượng.

5.1 Các đối tượng chính

Trước hết chúng ta xây dựng các đối tượng chính. Các đối tượng được xây dựng với kiểu dữ liệu của ngôn ngữ `C++`, ta hoàn toàn có thể triển khai tương tự với các ngôn ngữ lập trình khác. Các kiểu số nguyên được sử dụng là `uint16_t` với giá trị lớn nhất là 65535 (đủ lớn để lưu số lượng khách hàng thực tế cũng như nhu cầu hay tải trọng của khách hàng...) để tiết kiệm bộ nhớ. Các kiểu số thực được sử dụng là `float`. Đôi khi, trong một số framework (như Google Ortool chẳng hạn), thay vì sử dụng số thực để lưu khoảng cách hay thời gian di chuyển giữa các yêu cầu hoặc giá trị hàm mục tiêu, số nguyên được sử dụng để tiết kiệm bộ nhớ cũng như tính toán nhanh hơn. Nếu cần nhiều hơn độ chính xác, ta có thể nhân tất cả với một hệ số (1000 chẳng hạn để đạt độ chính xác 3 chữ số sau dấu phảy) và chia lại khi nhận kết quả cuối cùng. Các đối tượng được liệt kê dưới đây.



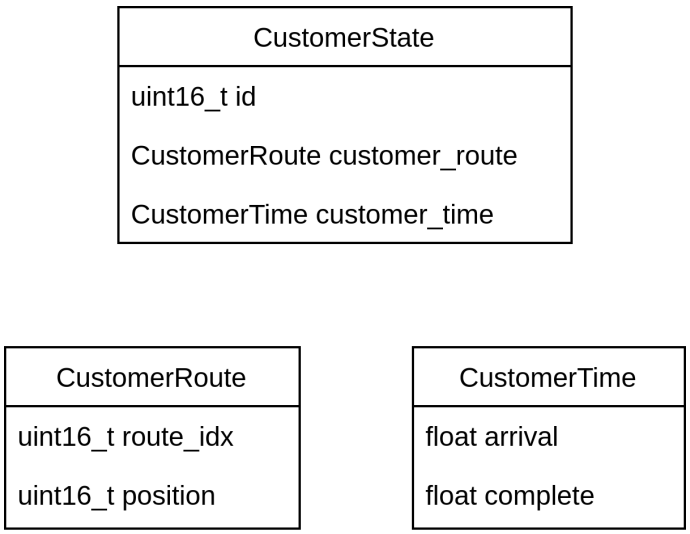
Hình 5.1: Đối tượng thuộc tính của khách hàng

`Customer`: đối tượng lưu các thuộc tính của một yêu cầu. Đối với VRPTW, mỗi yêu cầu là một khách hàng (đơn).

- `id`: id của khách hàng.
- `coord`: mảng lưu toạ độ của khách hàng (vĩ độ, kinh độ). Đôi khi chúng ta không cần quan tâm tới tọa độ của khách hàng mà quan tâm tới khoảng

cách hoặc thời gian di chuyển giữa các khách hàng, nên thuộc tính này có thể không cần thiết đối với một số cấu hình trong thực tế.

- `demand` : nhu cầu (về tải) của khách hàng.
- `time_window` : mảng lưu khung thời gian để phục vụ khách hàng. Mảng gồm 2 phần tử lần lượt là thời điểm sớm nhất và muộn nhất để phục vụ khách hàng.
- `service_time` : thời gian phục vụ khách hàng.



Hình 5.2: Đối tượng trạng thái của khách hàng

`CustomerRoute` : đối tượng lưu trạng thái về tuyến đường.

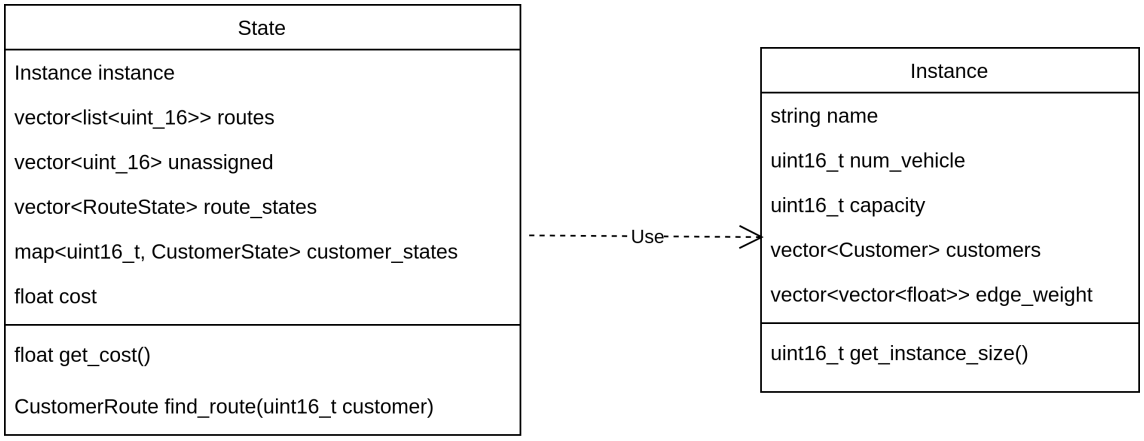
- `route_idx` : id của tuyến chứa khách hàng.
- `position` : vị trí của khách hàng trong tuyến.

`CustomerTime` : đối tượng lưu trạng thái về thời gian của yêu cầu.

- `arrival` : thời gian xe đến vị trí của khách hàng.
- `complete` : thời gian xe hoàn thành phục vụ khách hàng.

`CustomerState` : đối tượng lưu trạng thái của khách hàng.

- `id` : id của khách hàng.
- `customer_route` : trạng thái về tuyến đường của khách hàng.
- `customer_time` : trạng thái về thời gian của khách hàng.



Hình 5.3: Đối tượng trạng thái của hệ

`Instance` : đối tượng lưu cấu hình của bài toán.

- `name` : tên cấu hình (ví dụ `C101` trong tập Solomon (1987)).
- `num_vehicle` : số xe tối đa được sử dụng.
- `capacity` : tải trọng của mỗi xe.
- `customers` : mảng các yêu cầu.
- `edge_weight` : ma trận trọng số cạnh. Ma trận này có thể là ma trận khoảng cách hay thời gian di chuyển giữa các yêu cầu.
- `get_instance_size()` : phương thức trả về kích thước của cấu hình (số khách hàng cộng với kho).

`State` : đối tượng lưu trạng thái của bài toán.

- `instance` : tham chiếu tới cấu hình bài toán.
- `routes` : các tuyến đường. Các tuyến đường được tổ chức như một `vector` với mỗi thành phần là một `linked_list` . Cấu trúc dữ liệu `linked_list` được lựa chọn phù hợp với ALNS khi ta có thể bỏ đi hoặc chèn thêm các yêu cầu vào giữa tuyến một cách nhanh chóng. Trong nhiều thuật toán khác, các yêu cầu được trao đổi giữa các tuyến thì `vector` là phù hợp hơn do ta có thể truy cập đến phần từ (yêu cầu) theo index rất nhanh mà không phải duyệt lần lượt từ đầu mảng giống như `linked_list` . Việc sử dụng `linked_list` cho ALNS là cực kì phù hợp, thao bỏ đi hay chèn thêm phần tử vào giữa tuyến nhanh hơn khoảng 250 lần khi so sánh với việc sử dụng cấu trúc dữ liệu `vector` !

- `unassigned` : danh sách các khách hàng chưa được phục vụ bởi bất kì xe nào.
- `customer_states` : trạng thái của các khách hàng.
- `cost` : tổng chi phí của hệ (hay nghiệm) hiện tại.
- `get_cost()` : phương thức trả về cost hiên tại.
- `find_route(customer)` : phương thức trả về trạng thái của một khách hàng với đầu vào là id của khách hàng.

5.2 Triển khai thuật toán

Chương trình được chia làm ba giai đoạn *Tiền xử lý*, *Chương trình chính*, *Đo đạc*.

- *Tiền xử lý* : trong giai đoạn này, các cấu hình trong các tập dữ liệu, được đọc vào và tính toán ma trận khoảng cách giữa các yêu cầu và lưu trữ vào các files. Các files có định dạng json và có schema như đối tượng `Instance` đã trình được trình bày trong mục 5.1. Các files này sẽ được đọc vào trong giai đoạn *Chương trình chính* để giảm thiểu thời gian tính toán. Để đơn giản, phần tiền xử lý sử dụng ngôn ngữ lập trình `python` .
- *Chương trình chính*: Đây là chương trình triển khai ALNS, được viết bằng ngôn ngữ `C++` . Các độ đo của chương trình được ghi ra các file logs để sử dụng cho giai đoạn *đo đạc*.
- *Đo đạc*: Trong giai đoạn này, các file logs được đọc vào và các độ đo được tính toán và phân tích.

Một trong những điểm mạnh của ALNS là triển khai được song song. Chúng ta có thể chạy nhiều luồng cùng một lúc để tăng tốc độ tính toán. Điều này đặc biệt hữu ích khi cấu hình cần giải có nhiều yêu cầu. Khi tìm kiếm nghiệm trong lân cận của nghiệm hiện tại, ta khởi tạo nhiều luồng cùng tìm kiếm. Điều này không những tăng tốc chương trình mà còn giúp cho thuật toán khó bị bẫy trong nghiệm tối ưu cục bộ hơn khi triển khai đơn luồng do sự đa dạng của việc tìm kiếm (bằng nhiều luồng). Ngoài ra có những cách triển khai đa luồng khác. Ví dụ, trong thuật toán "hủy" và "sửa", ta hoàn toàn có thể bỏ đi các yêu cầu hoặc thêm lại các yêu cầu một cách song song từ các tuyến đường khác nhau.

Trong thực tế, tác giả chọn cách triển khai đầu tiên vì đơn giản và dễ hiểu. Chúng ta chỉ cần khởi tạo các luồng một lần mà không cần phải cấp lại tài nguyên. Với cách làm thứ 2, ta liên tục phải tạo các luồng mới và cấp lại tài nguyên cho chúng,

điều này có thể làm giảm hiệu năng của chương trình vì thực tế thì việc khởi tạo luồng là đắt đỏ. Tác giả cũng thử nghiệm cách làm thứ 2 với thư viện `OpenMP` (là một thư viện hỗ trợ lập trình bất đồng bộ và song song nổi tiếng dành cho `C++`) tuy nhiên hiệu năng không đạt tốt như cách triển khai đa luồng đầu tiên.

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).

(a) C-class (b) R-class (c) RC-class

Hình 6.1: Lớp các cấu hình

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

6.1.1 Số lượng yêu cầu liệu nhỏ

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.

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.

Tập R1 và R2 có các yêu cầu được tạo hoàn toàn ngẫu nhiên thế nên cũng có nhiều nghiệm chấp nhận được và thuật toán cũng khó bị bẫy ở một nghiệm tối ưu cục bộ.

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

Tuy nhiên, thuật toán cũng mất nhiều thời gian để tìm nghiệm tối ưu hơn do có quá nhiều nghiệm thỏa mãn các ràng buộc. Riêng với tập R2, ALNS đã tìm ra nghiệm với số xe ít hơn nghiệm tốt nhất đã biết mà tổng khoảng cách chỉ chênh lệch nhỏ.

Tập RC1 và RC2 chứa các yêu cầu ở các vị trí được phân cụm một cách tương đối (lai giữa tập R và tập C) được đánh giá là tập khó, do các vị trí của yêu cầu đủ ngẫu nhiên để bài toán có nhiều nghiệm chấp nhận được nhưng lại dễ bị bẫy ở nghiệm tối ưu cục bộ khi mà xe đã phụ vụ các yêu cầu ở trong một cụm chỉ cần ta bỏ đi một (vài) yêu cầu "quan trọng" trong cụm là đã nhận được một nghiệm tệ hơn trước khá nhiều. Mặc dù vậy ALNS vẫn tỏ ra hiệu quả khi tìm được nghiệm với chênh lệch trung bình 0.36% và 0.74% cho RC1 và RC2. Ngoài ra, ALNS cũng tìm được nghiệm với số xe ít hơn nghiệm tốt nhất đã biết trong một số cấu hình của tập RC2.

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

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

6.1.2 Số lượng yêu cầu lớn và rất lớn

Với số lượng yêu cầu là 200, ALNS vẫn tỏ ra hiệu quả với thời gian timeout 1 phút. ALNS cho chênh lệch trung bình 0.23%, 1.60% và 1.73% lần lượt cho các lớp C, R, RC so với nghiệm tốt nhất đã biết. Ngoài ra, ALNS tỏ ra ổn định khi độ lệch chuẩn giữa các lần chạy là rất nhỏ khi so sánh với giá trị hàm mục tiêu. Số xe sử dụng cũng không có sự khác biệt so với nghiệm tốt nhất đã biết.

Đối với tập 400 yêu cầu, ALNS cho kết quả không tốt như với các tập có số lượng yêu cầu nhỏ với thời gian timeout 1 phút. Độ chênh lệch trung bình lần lượt là 1.99%, 4.35% và 5.78% cho lớp C, R và RC khi so sánh với nghiệm tốt nhất đã biết. Ta có thể thấy rằng, đối với lớp khó như RC, ALNS không còn duy trì được hiệu quả như khi giải bài toán lớp C và R. Tuy nhiên, ALNS vẫn ổn định khi độ lệch chuẩn giữa các lần đo vẫn rất nhỏ so với giá trị hàm mục tiêu.

Dưới đây là các kết quả đo cho các cấu hình 200, 400, 600, 800 và 1000 yêu cầu cho 3 lớp C, R và RC.

Trong đó

- *ins: tên cấu hình*
- *alns best: giá trị hàm mục tiêu tốt nhất đạt được bởi ALNS*
- *alns avg: giá trị hàm mục tiêu trung bình (với 5 lần đo) đạt được bởi ALNS*
- *alns std: độ lệch chuẩn của giá trị hàm mục tiêu (với 5 lần đo) đạt được bởi ALNS*
- *nv: số xe được sử dụng bởi ALNS*
- *bk cost: giá trị hàm mục tiêu tốt nhất đã biết*
- *bk nv: số xe được sử dụng bởi nghiệm tốt nhất đã biết*
- *gap (%): chênh lệch giữa giá trị hàm mục tiêu tốt nhất đạt được bởi ALNS và giá trị hàm mục tiêu tốt nhất đã biết*

Thời gian timeout là 1 phút.

Nếu cho rằng việc chênh lệch chất lượng nghiệm so với nghiệm tốt nhất đã biết không vượt quá 10% là đủ thì ALNS thể hiện tốt đối với các cấu hình từ 600 yêu cầu trở xuống. Ngoài ra, tác giả tiến hành chạy thử nghiệm với thời gian tối đa 1 phút. Con số 1 phút được lựa chọn để phù hợp với yêu cầu thực tế khi mà ta không

thể chờ quá lâu để nhận được kết quả. Một chiến thuật khác là ta có thể lên lịch cho các xe vào ngày hôm trước để có các tuyến đường cho ngày hôm sau chẳng hạn (cho chương trình chạy "qua đêm", hay chạy với thời gian chờ lâu có thể là vài tiếng đồng hồ). Tác giả đã thử nghiệm và nhận thấy rằng, với thời gian chạy lâu hơn nữa ALNS cho kết quả tốt hơn nhưng tốc độ giảm nghiệm là chậm, do càng gần với nghiệm tối ưu thì các tuyến đường khá khó để thay đổi.

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
C1_2_1	2,704.57	2,704.57	0	20	2,698.6	20	0.22
C1_2_2	2,700.65	2,700.65	0	20	2,694.3	20	0.24
C1_2_3	2,682.18	2,711.28	45.2	20	2,675.8	20	0.24
C1_2_4	2,631.89	2,647.68	17.2	19	2,625.6	19	0.24
C1_2_5	2,702.05	2,711.04	17.99	20	2,694.9	20	0.27
C1_2_6	2,701.04	2,701.04	0	20	2,694.9	20	0.23
C1_2_7	2,701.04	2,701.04	0	20	2,694.9	20	0.23
C1_2_8	2,690.27	2,690.27	0	20	2,684	20	0.23
C1_2_9	2,645.47	2,650.42	9.9	19	2,639.6	19	0.22
C1_2_10	2,630.95	2,651.39	24.08	19	2,624.7	19	0.24
avg	2,679.01	2,686.94	11.44		2,672.73		0.23

Bảng 6.6: Kết quả đo với tập HG_C_1_2 200 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
R1_2_1	4,708.02	4,720.47	13.83	23	4,667.2	23	0.87
R1_2_2	4,011.45	4,029.16	12.06	20	3,919.9	20	2.34
R1_2_3	3,416.04	3,444.85	19.13	19	3,373.9	18	1.25
R1_2_4	3,121.85	3,146.43	15.45	19	3,047.6	18	2.44
R1_2_5	4,096.03	4,128.44	17.06	20	4,053.2	20	1.06
R1_2_6	3,588.31	3,626.85	22.15	20	3,559.1	19	0.82
R1_2_7	3,199.57	3,240.56	24.57	18	3,141.9	18	1.84
R1_2_8	2,973.75	3,006.53	18.22	18	2,938.4	18	1.2
R1_2_9	3,784.88	3,816.64	31.13	20	3,734.7	19	1.34
R1_2_10	3,385.86	3,416.78	29.63	19	3,293.1	18	2.82
avg	3,628.58	3,657.67	20.33		3,572.90		1.60

Bảng 6.7: Kết quả đo với tập HG_R_1_2 200 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
RC1_2_1	3536.71	3565.66	20.7	20	3516.9	20	0.56
RC1_2_2	3253.94	3273.73	21.67	19	3221.6	19	1
RC1_2_3	3034.11	3073.05	38.72	19	3001.4	18	1.09
RC1_2_4	2903.03	2944.38	33.08	19	2845.2	18	2.03
RC1_2_5	3388.43	3430.72	48.65	19	3325.6	19	1.89
RC1_2_6	3369.85	3406.8	24	19	3300.7	19	2.1
RC1_2_7	3240.26	3264.27	16.71	19	3177.8	19	1.97
RC1_2_8	3142.02	3176.37	22.88	19	3060	19	2.68
RC1_2_9	3123.98	3136.91	7.07	19	3073.3	19	1.65
RC1_2_10	3060.09	3080.15	19.51	19	2990.5	19	2.33
avg	3,205.24	3,235.20	25.30		3,151.30		1.73

Bảng 6.8: Kết quả đo với tập HG_RC_1_2 200 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
C1_4_1	7,152.06	7,155.8	7.49	40	7,138.8	40	0.19
C1_4_2	7,127.29	7,167.34	78.96	40	7,113.3	40	0.2
C1_4_3	7,160.22	7,279.36	128.8	39	6,929.9	38	3.32
C1_4_4	7,111.32	7,160.06	52.95	37	6,777.7	37	4.92
C1_4_5	7,152.06	7,172.31	31.98	40	7,138.8	40	0.19
C1_4_6	7,153.45	7,157.2	7.49	40	7,140.1	40	0.19
C1_4_7	7,149.43	7,185.08	71.22	40	7,136.2	40	0.19
C1_4_8	7,179.2	7,360.85	98.82	40	7,083	39	1.36
C1_4_9	7,170.61	7,240.77	71.64	38	6,927.8	37	3.5
C1_4_10	7,225.71	7,288.42	36.02	38	6,825.4	37	5.86
avg	7,158.13	7,216.72	58.54		7,021.10		1.99

Bảng 6.9: Kết quả đo với tập HG_C_1_4 400 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
R1_4_1	10,688.08	10,709.24	17.6	41	10,305.8	41	3.71
R1_4_2	9,149.2	9,221.35	56.53	39	8,873.2	37	3.11
R1_4_3	8,132.17	8,226.37	105.28	37	7,781.6	37	4.51
R1_4_4	7,690.06	7,766.42	65.65	36	7,266.2	36	5.83
R1_4_5	9,508.73	9,624.12	68.39	39	9,184.6	37	3.53
R1_4_6	8,621.57	8,753.94	70.05	37	8,340.4	36	3.37
R1_4_7	7,946.67	7,993.13	43.45	37	7,599.8	36	4.56
R1_4_8	7,594.01	7,679.99	46.64	37	7,240.5	36	4.88
R1_4_9	9,104.48	9,182.77	55.57	38	8,673.8	37	4.97
R1_4_10	8,482.36	8,520.08	34.86	37	8,077.8	36	5.01
avg	8,691.74	8,767.74	56.40		8,334.37		4.35

Bảng 6.10: Kết quả đo với tập HG_R_1_4 400 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
RC1_4_1	8,969.66	9,054.11	67.58	39	8,522.9	37	5.24
RC1_4_2	8,360.36	8,422.77	63.37	38	7,878.2	36	6.12
RC1_4_3	7,969.51	8,027.17	35.9	37	7,516.9	37	6.02
RC1_4_4	7,651.64	7,694.7	53.93	37	7,292.9	36	4.92
RC1_4_5	8,635.31	8,707.26	58.6	38	8,152.3	37	5.92
RC1_4_6	8,636.92	8,715.55	67.04	38	8,148	37	6
RC1_4_7	8,449.56	8,527.71	57.43	37	7,932.5	37	6.52
RC1_4_8	8,201.97	8,326.8	67.88	38	7,757.2	36	5.73
RC1_4_9	8,188.35	8,243.74	62.3	37	7,717.7	36	6.1
RC1_4_10	7,979.06	8,080.82	69.84	37	7,581.2	36	5.25
avg	8,304.23	8,380.06	60.39		7,849.98		5.78

Bảng 6.11: Kết quả đo với tập HG_RC_1_4 400 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
C1_6_1	14,095.64	14,095.64	0	60	14,076.6	60	0.14
C1_6_2	14,137.09	14,382.74	134.48	59	13,948.3	58	1.35
C1_6_3	15,123.71	15,277.54	100.57	57	13,756.5	57	9.94
C1_6_4	14,698.3	14,952.4	172.99	56	13,538.6	56	8.57
C1_6_5	14,085.72	14,207.91	183.08	60	14,066.8	60	0.13
C1_6_6	14,118.06	14,405.1	248.54	60	14,070.9	60	0.34
C1_6_7	14,614.07	14,905.21	276.97	61	14,066.8	60	3.89
C1_6_8	15,014.29	15,470.25	373.38	61	13,991.2	58	7.31
C1_6_9	15,213.84	15,362.78	225.23	59	13,664.5	56	11.34
C1_6_10	15,168.81	15,611.42	311.86	58	13,617.5	56	11.39
avg	14,626.95	14,867.10	202.71		13,879.77		5.44

Bảng 6.12: Kết quả đo với tập HG_C_1_6 600 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
R1_6_1	22,534.32	22,810.62	165.87	63	21,274.2	58	5.92
R1_6_2	19,751.55	19,960.87	115.13	57	18,519.8	56	6.65
R1_6_3	18,268.88	18,398.55	87.36	55	16,874.9	54	8.26
R1_6_4	16,863.99	16,997.55	99.47	55	15,720.8	54	7.27
R1_6_5	20,818.89	20,944.12	85.35	58	19,294.9	55	7.9
R1_6_6	19,168.72	19,293.47	88.32	56	17,763.7	54	7.91
R1_6_7	17,959.3	181,00.5	129.91	56	16,496.2	54	8.87
R1_6_8	16,650.4	167,75.82	92.25	55	15,584.3	54	6.84
R1_6_9	19,882.68	20,099.04	183	57	18,474.1	55	7.62
R1_6_10	18,899.08	18,967.51	48.97	56	17,583.7	54	7.48
avg	19,079.78	19,234.81	109.56		17,758.66		7.47

Bảng 6.13: Kết quả đo với tập HG_R_1_6 600 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
RC1_6_1	18048.91	18228.51	134.64	57	16944.2	56	6.52
RC1_6_2	17003.23	17059.88	50.84	56	15890.6	55	7
RC1_6_3	16476.72	16561.34	116.14	57	15181.3	55	8.53
RC1_6_4	15956.21	16102.1	106.26	56	14753.2	55	8.15
RC1_6_5	17819.13	17948.16	93.89	57	16536.3	55	7.76
RC1_6_6	17718.49	17855.99	97.8	57	16473.3	55	7.56
RC1_6_7	17360.14	17516.52	154.79	56	16055.3	55	8.13
RC1_6_8	17257.21	17338.39	60.79	56	15891.8	55	8.59
RC1_6_9	17278.83	17423.02	122.84	56	15803.5	55	9.34
RC1_6_10	16938.62	17186.96	190.67	56	15651.3	55	8.22
avg	17,185.75	17,322.09	112.87		15,918.08		7.98

Bảng 6.14: Kết quả đo với tập HG_RC_1_6 600 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
C1_8_1	25,550.61	25,834.96	332.72	81	25,156.9	80	1.57
C1_8_2	26,698.48	26,923.41	247.14	80	24,974.1	78	6.9
C1_8_3	26,912.55	27,224.07	248.94	75	24,156.1	73	11.41
C1_8_4	26,295.21	27,023.75	487.76	73	23,797.3	72	10.5
C1_8_5	26,172.88	27,155.03	747.1	82	25,138.6	80	4.11
C1_8_6	27,929.18	28,304.47	313.55	85	25,133.3	80	11.12
C1_8_7	28,011.92	28,502.49	470.66	84	25,127.3	80	11.48
C1_8_8	28,368.69	28,716.28	208.72	83	24,809.7	76	14.35
C1_8_9	27,663.78	28,370.25	486.05	78	24,200.4	74	14.31
C1_8_10	27,710.71	28,278.32	392.63	76	24,026.7	73	15.33
avg	27,131.40	27,633.31	393.53		24,652.04		10.11

Bảng 6.15: Kết quả đo với tập HG_C_1_8 800 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
R1_8_1	39,957.46	40,399.61	312.62	85	36,345	80	9.94
R1_8_2	35,708.47	36,107.65	215.79	76	32,277.6	72	10.63
R1_8_3	32,405.25	32,804.12	284	74	29,301.2	72	10.59
R1_8_4	30,538.49	30,847.3	183.56	73	27,734.7	72	10.11
R1_8_5	37,304.47	37,473.07	182.18	76	33,494	72	11.38
R1_8_6	34,447.73	34,694.32	166.39	74	30,872.4	72	11.58
R1_8_7	32,102.03	32,443.81	191.64	73	28,789	72	11.51
R1_8_8	30,172.06	30,459.2	311.32	73	27,609.4	72	9.28
R1_8_9	35,753.09	36,114.12	242.75	76	32,257.3	72	10.84
R1_8_10	34,338.56	34,614.73	211.16	75	30,918.3	72	11.06
avg	34,272.76	34,595.79	230.14		30,959.89		10.69

Bảng 6.16: Kết quả đo với tập HG_R_1_8 800 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
RC1_8_1	32,845.93	32,945.3	75.88	77	29,952.8	74	9.66
RC1_8_2	31,055.24	31,295.32	209.05	75	28,290.1	74	9.77
RC1_8_3	30,059.19	30,435.63	196.98	74	27,447.7	73	9.51
RC1_8_4	28,711.26	28,997.17	192.21	74	26,557.2	73	8.11
RC1_8_5	32,178.34	32,350.29	122.35	75	29,219.9	74	10.12
RC1_8_6	32,314.8	32,447.26	110.15	75	29,148.7	73	10.86
RC1_8_7	31,509.08	31,795.07	286.75	75	28,734	73	9.66
RC1_8_8	31,359.46	31,618.27	199.77	74	28,390	73	10.46
RC1_8_9	31,387.83	31,709.16	234.78	74	28,331.6	73	10.79
RC1_8_10	31,031.65	31,323.19	285.86	74	28,168.5	73	10.16
avg	31,245.28	31,491.67	191.38		28,424.05		9.91

Bảng 6.17: Kết quả đo với tập HG_RC_1_8 800 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
C1_10_1	44,214.75	45,678.8	1,113.93	104	42,444.8	100	4.17
C1_10_2	45,238.08	46,051.56	589.11	101	41,337.8	94	9.44
C1_10_3	45,272.64	46,070.45	529.64	94	40,060	91	13.01
C1_10_4	45,747.3	46,000.23	159.43	92	39,434.1	90	16.01
C1_10_5	48,159.25	48,680.45	472.71	109	42,434.8	100	13.49
C1_10_6	48,562.93	49,888.23	868.44	109	42,437	100	14.44
C1_10_7	48,053.69	49,347.02	809.02	106	42,420.4	100	13.28
C1_10_8	49,702.64	50,228.59	575.78	106	41,648	95	19.34
C1_10_9	47,972.91	48,869.52	537.98	100	40,288.4	90	19.07
C1_10_10	47,131.75	48,231.19	670.04	97	39,816.8	90	18.37
avg	47,005.59	47,904.60	632.61		41,232.21		14.06

Bảng 6.18: Kết quả đo với tập HG_C_1_10 1000 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
R1_10_1	60,581.8	60,995.25	273.58	103	53,026.1	95	14.25
R1_10_2	56,255.34	56,723.58	472.21	95	48,261.6	91	16.56
R1_10_3	52,795.73	53,081.74	225.01	93	44,673.3	91	18.18
R1_10_4	48,386.45	49,079.46	466.74	92	42,440.7	91	14.01
R1_10_5	56,925.27	57,977.91	811.43	95	50,406.7	91	12.93
R1_10_6	54,800.49	55,053.29	259.44	93	46,928.2	91	16.78
R1_10_7	51,441.07	52,184.01	459.41	93	43,997.4	91	16.92
R1_10_8	48,616.89	48,852.4	220.02	92	42,279.3	91	14.99
R1_10_9	56,403.15	57,143.03	413.89	93	49,162.8	91	14.73
R1_10_10	55,043.99	55,274.53	166.75	94	47,364.6	91	16.21
avg	54,125.02	54,636.52	376.85		46,854.07		15.56

Bảng 6.19: Kết quả đo với tập HG_R_1_10 1000 yêu cầu

ins	alns best	alns avg	alns std	nv	bk cost	bk nv	gap (%)
RC1_10_1	51,798.33	52,030.75	185.35	96	45,790.7	90	13.12
RC1_10_2	49,501.03	49,828.93	285.98	92	43,678.3	90	13.33
RC1_10_3	47,870	48,389.62	452.14	91	42,121.9	90	13.65
RC1_10_4	46,208.46	46,561.7	228.08	91	41,357.4	90	11.73
RC1_10_5	51,256.07	51,556.27	196.57	94	45,028.1	90	13.83
RC1_10_6	51,328.39	51,602.46	368.44	92	44,898.2	90	14.32
RC1_10_7	50,404.57	50,890.2	394.46	92	44,409	90	13.5
RC1_10_8	49,827.15	50,261.65	263.19	92	43,916.5	90	13.46
RC1_10_9	49,170.31	49,728.92	447.75	92	43,858	90	12.11
RC1_10_10	48,966.51	49,653.21	389.99	91	43,533.7	90	12.48
avg	49,633.08	50,050.37	321.19		43859.18		13.15

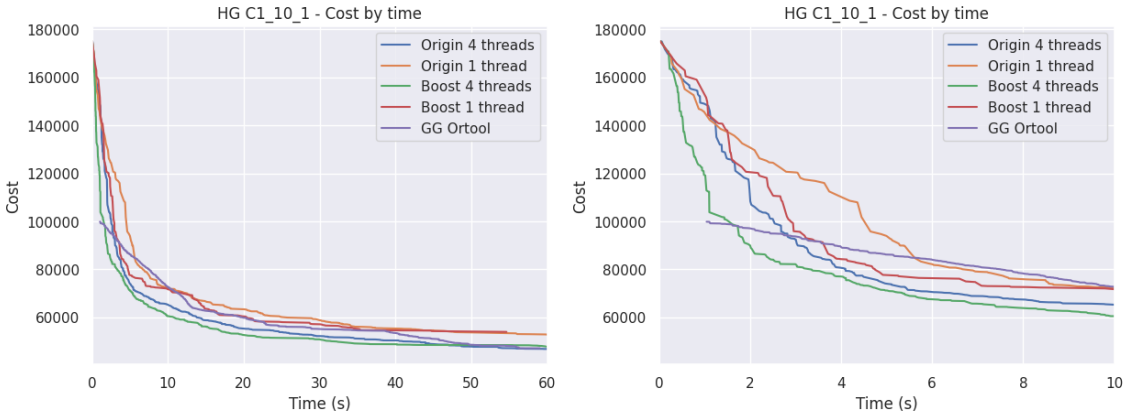
Bảng 6.20: Kết quả đo với tập HG_RC_1_10 1000 yêu cầu

6.2 Hiệu năng thuật toán

Trong phần này chúng ta sẽ xem xét hiệu năng của ALNS. Để thấy rõ sự khác biệt, các đồ thị được trình bày dưới đây là kết quả cho cấu hình 1000 yêu cầu. Ngoài ra, để tránh dài dòng, các đồ thị được thể hiện cho 3 cấu hình C1_10_1, R1_10_1, RC1_10_1. Hiệu năng của các thuật toán đối với các cấu hình khác là tương tự. Chúng ta sẽ so sánh hiệu năng của ALNS nguyên bản, B-ALNS và một framework rất nổi tiếng trong cộng đồng là `Google OR-Tools`. Hiệu năng được so sánh cho cả phiên bản đơn luồng và đa luồng của ALNS.

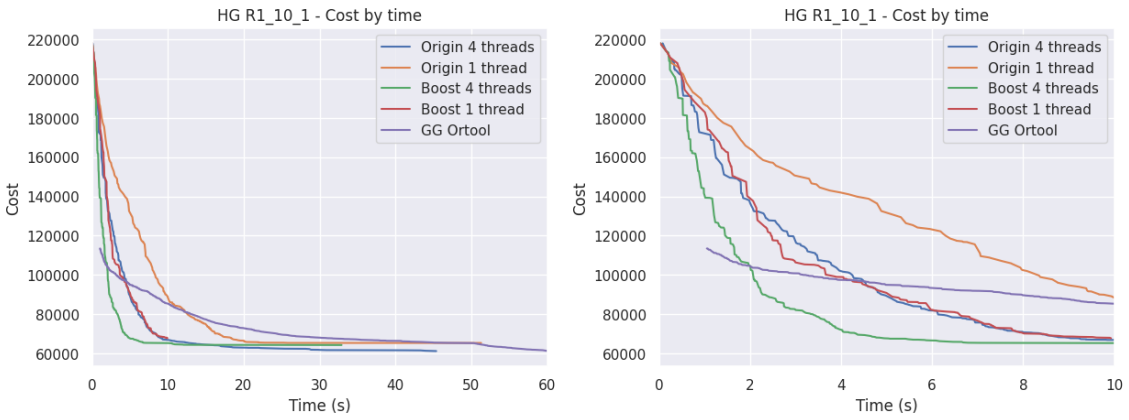
6.2.1 Giá trị hàm mục tiêu

Trước hết, giá trị hàm mục tiêu theo thời gian được xem xét. Các đồ thị dưới đây cho thấy giá trị hàm mục tiêu theo thời gian khi chạy thuật toán với thời gian chạy 60 giây và được phóng đại để nhìn rõ hơn sự khác biệt trong 10 giây đầu tiên. Đồ thị được chia làm 3 phần tương ứng với 3 cấu hình khác nhau.



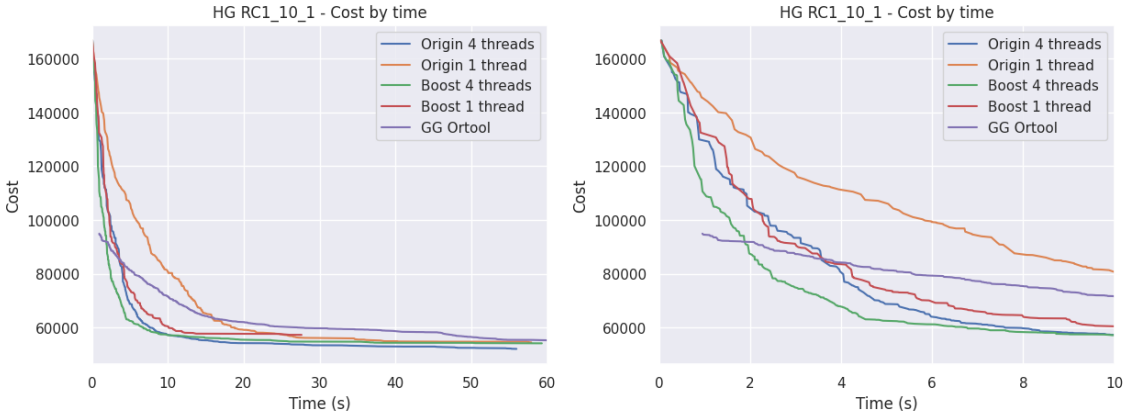
(a) 60s (b) 10s

Hình 6.2: Giá trị hàm mục tiêu theo thời gian, cấu hình C1_10_1



(a) 60s (b) 10s

Hình 6.3: Giá trị hàm mục tiêu theo thời gian, cấu hình R1_10_1



(a) 60s (b) 10s

Hình 6.4: Giá trị hàm mục tiêu theo thời gian, cấu hình RC1_10_1

Với thời gian chạy lâu, các thuật toán đều chững lại bởi khi đó các tuyến đường đã khá chật chội, việc "sửa chữa" là khó khăn hơn giai đoạn đầu rất nhiều. Nói cách khác, thuật toán bị bẫy trong nghiệm tối ưu cục bộ. Khi chạy thuật toán với thời gian lâu hơn (timeout 10 phút), tác giả nhận thấy hàm mục tiêu không giảm đáng kể nữa. Chính vì thế, thời gian chạy 60 giây được lựa chọn để vừa phù hợp với thực tế và tránh việc chạy quá lâu.

Ta thấy một xu hướng rõ ràng, trong giai đoạn đầu, B-ALNS tăng tốc hiệu năng của ALNS một cách đáng kể. Với thời gian chạy dưới 10 giây, B-ALNS đơn luồng cho hiệu năng gần như tương đương với ALNS với 4 luồng. Nghĩa là, B-ALNS tiết kiệm khoảng 75% tài nguyên CPU để đạt kết quả tương đương với ALNS trong thời gian chạy dưới 10 giây. Chưa kể, memory cũng được tiết kiệm một cách đáng kể khi

B-ALNS sử dụng 1 luồng thay vì 4 luồng. Khi so sánh với ALNS đơn luồng, B-ALNS đơn luồng cho hiệu năng vượt trội. Điều này cho thấy, B-ALNS có thể được sử dụng để giải quyết các bài toán lớn với tài nguyên hạn chế.

Khi nâng số luồng của B-ALNS lên bằng với số luồng của ALNS, ta thấy rằng, hàm mục tiêu giảm nhanh hơn trong giai đoạn dầu. Đặc biệt trong các cấu hình mà địa điểm của yêu cầu có độ ngẫu nhiên cao (lớp R và RC), B-ALNS cho hiệu năng rất tốt.

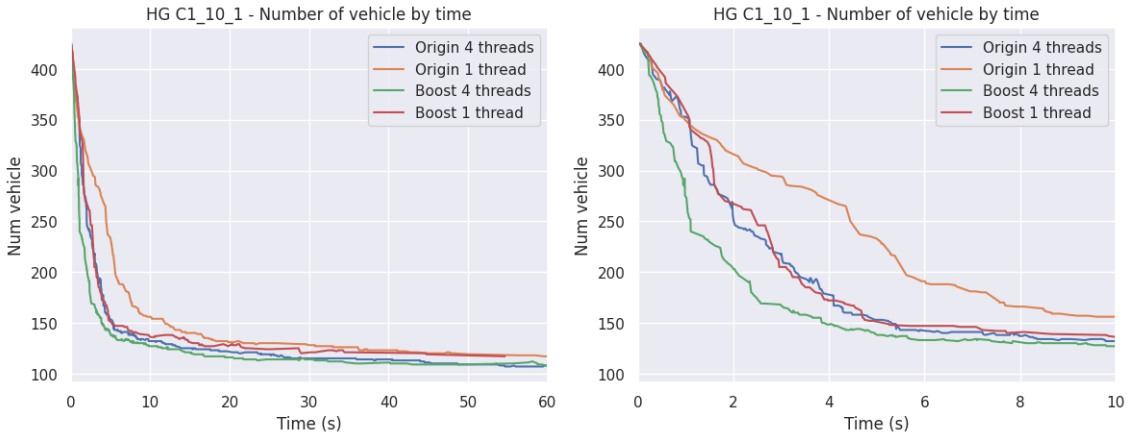
Khi so sánh với `Google OR-Tools`, ta thấy rằng, `Google OR-Tools` cho hiệu năng tốt trong thời gian chạy từ 2 đến 4 giây đầy tiên. Tuy nhiên với thời gian chạy dưới 1 giây, `Google OR-Tools` không thể cho ra kết quả. Với thời gian chạy lâu hơn, `Google OR-Tools` cho hiệu năng không tốt bằng ALNS hay B-ALNS.

Như vậy đối với các nghiệp vụ yêu cầu một kết quả tốt trong thời gian ngắn, B-ALNS là lựa chọn tốt nhất. Khi tiến hành đo đạc với thời gian chạy dài (timeout lớn hơn 1 phút), tác giả nhận thấy rằng, ALNS là tốt nhất trong các thuật toán được đề cập ở đây. Các kết quả được chỉ ra trong phần trước là giá trị hàm mục tiêu khi sử dụng ALNS nguyên bản.

6.2.2 Số xe

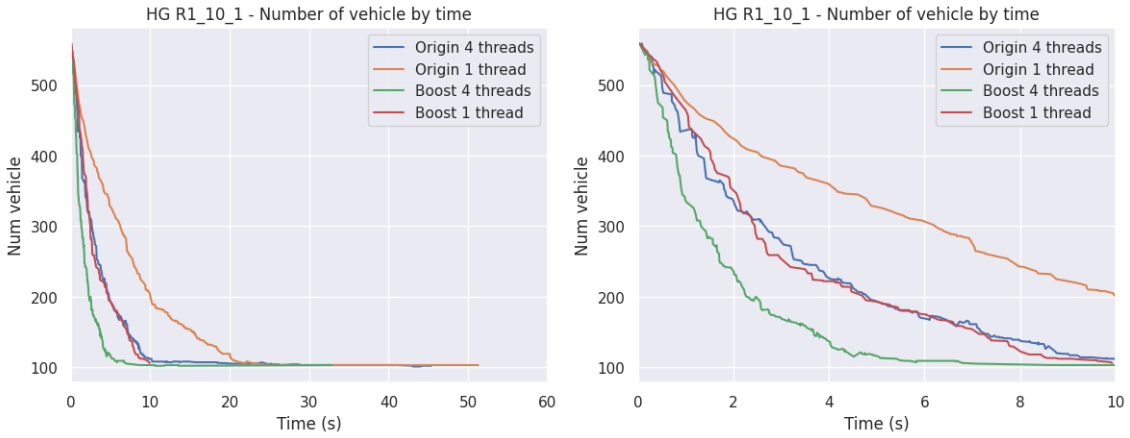
Chúng ta cũng nhận thấy một xu hướng tương tự như khi so sánh hiệu năng của các thuật toán khi sử dụng độ đo là giá trị hàm mục tiêu. B-ALNS cho hiệu năng vượt trội trong giai đoạn đầu. B-ALNS đơn luồng cho hiệu năng tương đương với ALNS với 4 luồng. Với cấu hình lớp R, B-ALNS tiết kiệm được khoảng 30% số xe so với ALNS trong thời gian chạy từ 2 tới 4 giây! Việc tiết kiệm hàng trăm xe có ý nghĩa rất lớn trong thực tế. Thường thì chi phí cho một xe (thuê, hoặc mua, nhiên liệu, chi phí cho tài xế, ...) là rất lớn. Việc tiết kiệm số xe sẽ giúp giảm chi phí vận hành của doanh nghiệp một cách đáng kể.

Với thời gian chạy lâu hơn, đương nhiên chúng ta rất khó để giảm được số xe nữa, vì hầu hết các tuyến đường đến lúc này đã chật chội hơn đáng kể so với giai đoạn đầu.



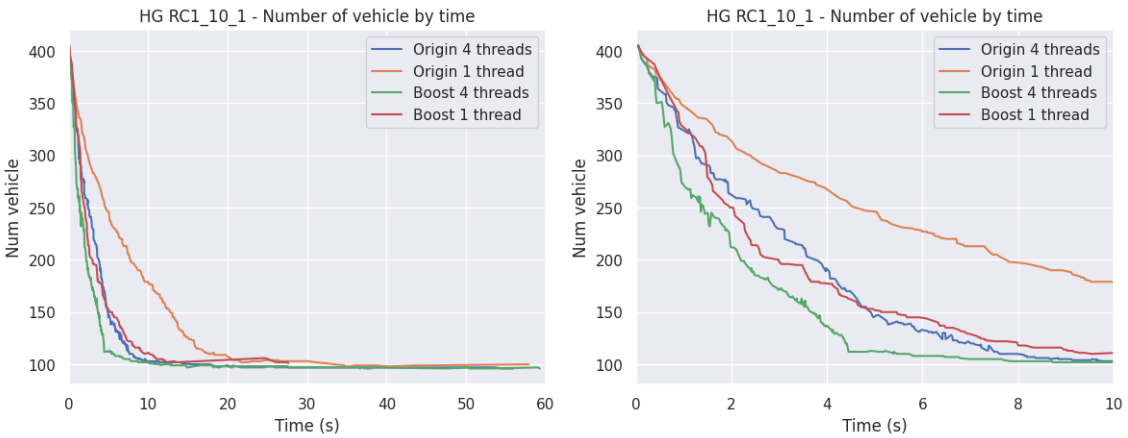
(a) 60s (b) 10s

Hình 6.5: Số xe sử dụng theo thời gian, cấu hình C1_10_1



(a) 60s (b) 10s

Hình 6.6: Số xe sử dụng theo thời gian, cấu hình R1_10_1



(a) 60s (b) 10s

Hình 6.7: Số xe sử dụng theo thời gian, cấu hình RC1_10_1

7 Kết luận

Tài liệu tham khảo

[1] Kemal Altinkemer and Bezalel Gavish. "Parallel savings based heuristics for the delivery problem". In: *Operations research* 39.3 (1991), pp. 456–469.

[2] Barrie M Baker and Janice Sheasby. "Extensions to the generalised assignment heuristic for vehicle routing". In: *European Journal of Operational Research* 119.1 (1999), pp. 147–157.

[3] Roberto Baldacci, Eleni Hadjiconstantinou, and Aristide Mingozzi. "An exact algorithm for the capacitated vehicle routing problem based on a two-commodity network flow formulation". In: *Operations research* 52.5 (2004), pp. 723–738.

[4] Michel L Balinski and Richard E Quandt. "On an integer program for a delivery problem". In: *Operations research* 12.2 (1964), pp. 300–304.

[5] Russell Bent and Pascal Van Hentenryck. "A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows". In: *International Conference on Principles and Practice of Constraint Programming*. Springer. 2003, pp. 123–137.

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

[7] Geoff Clarke and John W Wright. "Scheduling of vehicles from a central depot to a number of delivery points". In: *Operations research* 12.4 (1964), pp. 568–581.

[8] Jean-François Cordeau, Michel Gendreau, and Gilbert Laporte. "A tabu search heuristic for periodic and multi-depot vehicle routing problems". In: *Networks: An International Journal* 30.2 (1997), pp. 105–119.

[9] George Dantzig, Ray Fulkerson, and Selmer Johnson. "Solution of a large-scale traveling-salesman problem". In: *Journal of the operations research society of America* 2.4 (1954), pp. 393–410.

[10] George B Dantzig and John H Ramser. “The truck dispatching problem”. In: *Management science* 6.1 (1959), pp. 80–91.

[11] Ulrich Derigs and R Kaiser. “Applying the attribute based hill climber heuristic to the vehicle routing problem”. In: *European Journal of Operational Research* 177.2 (2007), pp. 719–732.

[12] Gunter Dueck. “New optimization heuristics: The great deluge algorithm and the record-to-record travel”. In: *Journal of Computational physics* 104.1 (1993), pp. 86–92.

[13] Gunter Dueck and Tobias Scheuer. “Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing”. In: *Journal of computational physics* 90.1 (1990), pp. 161–175.

[14] Gerd Finke. “A two-commodity network flow approach to the traveling salesman problem”. In: *Congresses Numeration* 41 (1984), pp. 167–178.

[15] Marshall L Fisher and Ramchandran Jaikumar. “A generalized assignment heuristic for vehicle routing”. In: *Networks* 11.2 (1981), pp. 109–124.

[16] Bezalel Gavish and Stephen C Graves. “The travelling salesman problem and related problems”. In: (1978).

[17] HEL Ghaziri. “Solving routing problems by a self-organizing map”. In: *Artificial neural network* (1991), pp. 829–834.

[18] Fred Glover. “Future paths for integer programming and links to artificial intelligence”. In: *Computers & operations research* 13.5 (1986), pp. 533–549.

[19] Bruce L Golden, Thomas L Magnanti, and Hien Q Nguyen. “Implementing vehicle routing algorithms”. In: *Networks* 7.2 (1977), pp. 113–148.

[20] JH Holland. “Adaptation in neural and artificial system”. In: *Ann Arbor, Univeristy of Michigan Press* (1975).

[21] Scott Kirkpatrick, C Daniel Gelatt Jr, and Mario P Vecchi. “Optimization by simulated annealing”. In: *science* 220.4598 (1983), pp. 671–680.

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

[23] 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.

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

[25] Nenad Mladenović and Pierre Hansen. “Variable neighborhood search”. In: *Computers & operations research* 24.11 (1997), pp. 1097–1100.

[26] Heinrich Paessens. “The savings algorithm for the vehicle routing problem”. In: *European Journal of Operational Research* 34.3 (1988), pp. 336–344.

[27] Marc Reimann, Karl Doerner, and Richard F Hartl. “D-ants: Savings based ants divide and conquer the vehicle routing problem”. In: *Computers & Operations Research* 31.4 (2004), pp. 563–591.

[28] Stefan Ropke and David Pisinger. “An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows”. In: *Transportation science* 40.4 (2006), pp. 455–472.

[29] M Schumann and R Retzko. “Self-organizing maps for vehicle routing problems minimizing an explicit cost function”. In: *Proceedings of the International Conference on Artificial Neural Networks*. Paris. 1995, pp. 401–406.

[30] Paul Shaw. “A new local search algorithm providing high quality solutions to vehicle routing problems”. In: *APES Group, Dept of Computer Science, University of Strathclyde, Glasgow, Scotland, UK* 46 (1997).

[31] Paul Shaw. “Using constraint programming and local search methods to solve vehicle routing problems”. In: *International conference on principles and practice of constraint programming*. Springer. 1998, pp. 417–431.

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

[33] Peter Wark and John Holt. “A repeated matching heuristic for the vehicle routeing problem”. In: *Journal of the Operational Research Society* 45.10 (1994), pp. 1156–1167.