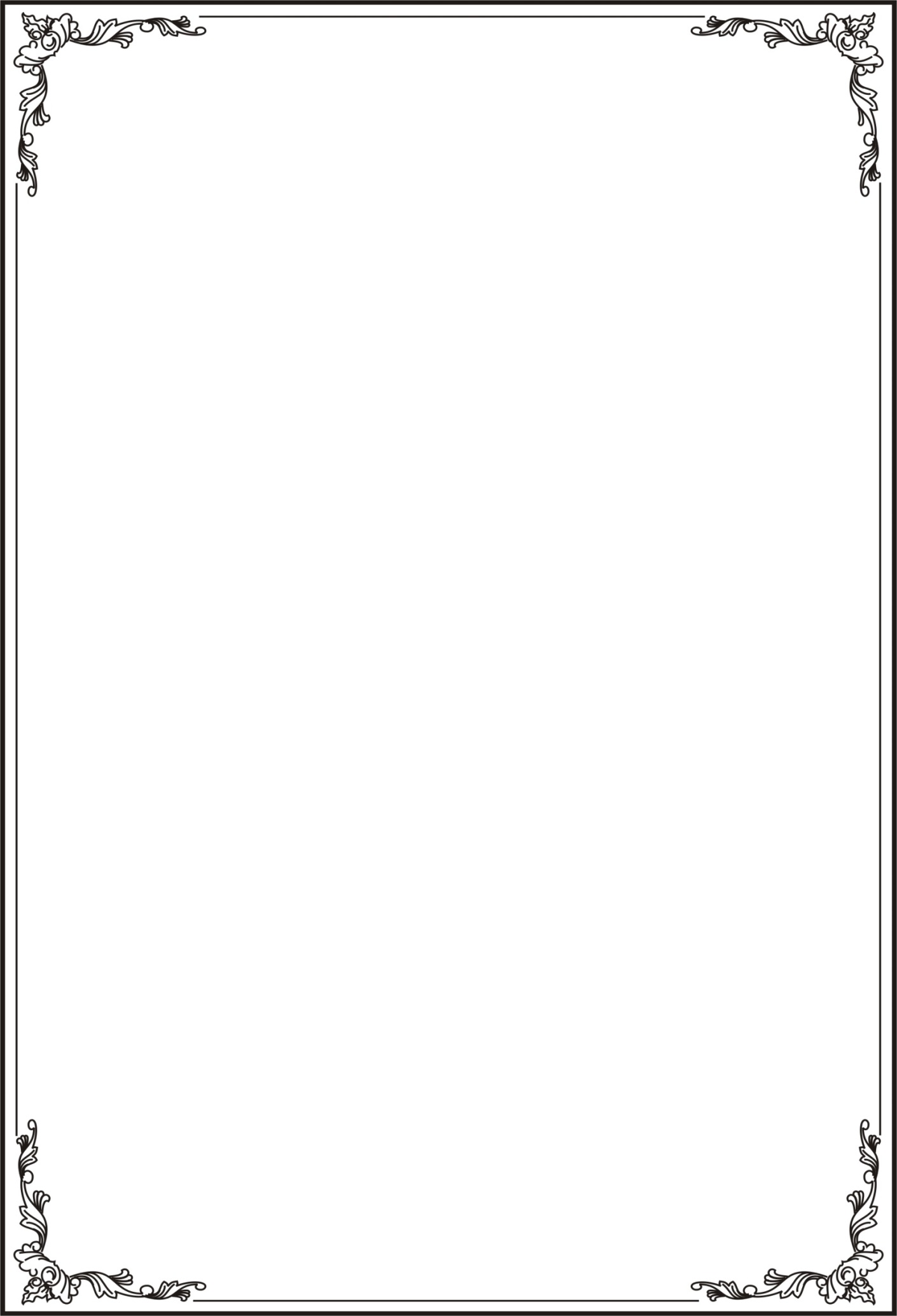
**BỘ GIAO THÔNG VẬN TẢI**



2025

**TRƯỜNG ĐẠI HỌC GIAO THÔNG VẬN TẢI TP.HCM**

**KHOA CÔNG NGHỆ THÔNG TIN**

**A blue and red text on a white background

Description automatically generated----------------**

**BÁO CÁO MÔN PHÂN TÍCH THIẾT KẾ GIẢI THUẬT**

**Tên đề tài:** **Phân tích và so sánh các thuật toán Heuristic cho Bài toán Định tuyến Phương tiện (VRP)**

|  |  |
| --- | --- |
| **Giảng viên hướng dẫn:** | ***Thầy Nguyễn Văn Huy*** |
| **Sinh viên thực hiện:** | ***Nguyễn Hoàng Nam***  ***Triệu Hồng Phúc***  ***Nguyễn Lương Minh Trí***  ***Ngô Minh Khang*** |
| **Mssv:** | ***052205010829***  ***091205000064***  ***075205013722***  ***086205011340*** |
| **Lớp:** | ***HT2301CLCB*** |

**LỜI CẢM ƠN**

Đầu tiên, nhóm em xin gửi lời cảm ơn đến Trường Đại họcGiao Thông Vận Tải TP.HCM đã đưa bộ môn Phân Tích Thiết Kế Giải Thuật vào chương trình giảng dạy để chúng em có cơ hội tiếp thu kiến thức quý giá. Đặc biệt, em xin gửi lời cảm ơn chân thành nhất đến thầy Nguyễn Văn Huy đã truyền đạt cho chúng em kiến thức bằng cả tất cả tâm huyết. Thời gian học bộ môn của thầy là khoảng thời gian tuyệt vời vì em không chỉ được học lý thuyết mà còn nắm bắt được những kinh nghiệm thực tế hữu ích. Đây sẽ là hành trang để em có thể vững bước trên con đường đã lựa chọn ban đầu. Bộ môn Phân Tích Thiết Kế Giải Thuật không chỉ bổ ích mà còn có tính thực tế cao.

Tuy nhiên, do vốn kiến thức còn nhiều hạn chế và khả năng tiếp thu thực tế còn nhiều bỡ ngỡ. Mặc dù em đã cố gắng hết sức nhưng chắc chắn bài tiểu luận khó có thể tránh khỏi những thiếu sót và nhiều chỗ còn chưa chính xác, kính mong thầy xem xét và góp ý để bài tiểu luận của em được hoàn thiện hơn.

Em xin chân thành cảm ơn!

**MỤC LỤC**

[I. TỔNG QUAN VỀ BÀI TOÁN ĐỊNH TUYẾN XE 10](#_Toc196172599)

[1. Giới thiệu bài toán 10](#_Toc196172600)

[2. Các khái niệm được sử dụng trong bài toán bao gồm 11](#_Toc196172601)

[3. Các biến thể của bài toán VRP 12](#_Toc196172602)

[4. Các hướng tiếp cận 13](#_Toc196172603)

[II. BÀI TOÁN ĐỊNH TUYẾN XE CÓ DUNG LƯỢNG GIỚI HẠN(CVRP) 16](#_Toc196172604)

[1. Giới thiệu bài toán 16](#_Toc196172605)

[2. Xây dựng mô hình toán học: 17](#_Toc196172606)

[3. Phân loại các phương pháp giải bài toán CVRP 19](#_Toc196172607)

[III. CHI TIẾT CÁC THUẬT TOÁN 20](#_Toc196172608)

[A. THUẬT TOÁN SWEEP 20](#_Toc196172609)

[1. Giới Thiệu Về Thuật Toán 20](#_Toc196172610)

[**1.1 Đặt vấn đề** 20](#_Toc196172611)

[**1.2 Giới thiệu về thuật toán Sweep** 21](#_Toc196172612)

[**1.3 Ý Tưởng thuật toán** 21](#_Toc196172613)

[2. Các Bước Triển Khai Chi Tiết của thuật toán ví dụ 22](#_Toc196172614)

[3. Pseudocode 26](#_Toc196172615)

[4. Code python 27](#_Toc196172616)

[5. Ví dụ minh hoạt thuật toán 31](#_Toc196172617)

[6.Phân tích hiệu suất và tham số ( có thể lấy dữ liệu từ chính ví dụ ) 32](#_Toc196172618)

[7. Ưu Và Nhược Điểm 33](#_Toc196172619)

[8. Ứng Dụng 34](#_Toc196172620)

[9. Kết Luận 35](#_Toc196172621)

[B) THUẬT TOÁN CLARKE AND WRIGHT. 35](#_Toc196172622)

[1. Giới thiệu về thuật toán 35](#_Toc196172623)

[**1.1** **Giới thiệu về thuật toán Clarke and Wright.** 36](#_Toc196172625)

[**1.2 Ý Tưởng thuật toán Clarke and Wright.** 36](#_Toc196172626)

[2. Các Bước Triển Khai Chi Tiết của thuật toán 38](#_Toc196172627)

[3. Pseucode 41](#_Toc196172628)

[4. Code python 42](#_Toc196172629)

[5. Minh họa bài toán: 46](#_Toc196172630)

[6. Phân tích hiệu suất và tham số. 48](#_Toc196172631)

[7. Ưu Và Nhược Điểm 50](#_Toc196172632)

[8. Ứng Dụng 51](#_Toc196172633)

[9. Tóm Tắt & Kết Luận 51](#_Toc196172634)

[C)THUẬT TOÁN TABU SEARCH 52](#_Toc196172635)

[1. Giới Thiệu Về Thuật Toán 52](#_Toc196172636)

[**1.1 Đặt vấn đề** 52](#_Toc196172637)

[**1.2 Giới thiệu về thuật toán Tìm Kiếm Tabu (Tabu Search)** 52](#_Toc196172638)

[**1.3 Ý Tưởng thuật toán** 52](#_Toc196172639)

[2. Các Bước Triển Khai Chi Tiết của thuật toán (theo code cung cấp) 53](#_Toc196172640)

[3. Pseudocode (Mô tả logic code) 55](#_Toc196172641)

[4. Code python (Giải thích code cung cấp) 56](#_Toc196172642)

[5. Ví dụ minh họa thuật toán (Step-by-step với tối ưu hóa) 59](#_Toc196172643)

[6. Phân tích hiệu suất và tham số 63](#_Toc196172644)

[7. Ưu Và Nhược Điểm 64](#_Toc196172645)

[8. Ứng Dụng 65](#_Toc196172646)

[9. Kết Luận 65](#_Toc196172647)

[D) THUẬT TOÁN DI TRUYỀN (GENITIC ALGORITHM) 66](#_Toc196172648)

[1. Giới Thiệu Về Thuật Toán 66](#_Toc196172649)

[**1.1 Giới thiệu về Thuật Toán Di Truyền (Genetic Algorithm - GA)** 66](#_Toc196172651)

[**1.2 Ý Tưởng thuật toán** 66](#_Toc196172652)

[2. Các Bước Triển Khai Chi Tiết của Thuật Toán (ví dụ) 67](#_Toc196172653)

[3. Pseudocode 67](#_Toc196172654)

[4. Code Python (Ví dụ đơn giản hóa) 67](#_Toc196172655)

[5. Ví Dụ / Minh Họa (Đơn Giản Hóa) 70](#_Toc196172656)

[6. Phân tích hiệu suất và tham số 72](#_Toc196172659)

[7. Ưu Và Nhược Điểm 73](#_Toc196172663)

[8. Ứng Dụng 74](#_Toc196172664)

[9. Tóm Tắt & Kết Luận 74](#_Toc196172665)

[IV. SO SÁNH VÀ ĐÁNH GIÁ KẾT QUẢ 75](#_Toc196172668)

[V. TỔNG KẾT 77](#_Toc196172669)

**NHẬN XÉT CỦA GIÁO VIÊN**

TPHCM, ngày … tháng … năm 2025

Ký và ghi họ tên

1. **TỔNG QUAN VỀ BÀI TOÁN ĐỊNH TUYẾN XE**

**1. Giới thiệu bài toán**

Bài toán định tuyến xe (Vehicle Routing Problem – VRP)là một bài toán tối ưu tổ hợp kinh điển, tập trung vào việc xác định các tuyến đường tối ưu cho một đội xe nhằm phục vụ một tập hợp khách hàng có vị trí và nhu cầu cụ thể. Mục tiêu chính là tối thiểu hóa tổng chi phí vận hành, quãng đường di chuyển hoặc thời gian phục vụ, đồng thời tuân thủ các ràng buộc về năng lực vận chuyển và yêu cầu dịch vụ của khách hàng.​

VRP được xem là một trong những bài toán quan trọng trong lĩnh vực quản lý chuỗi cung ứng và logistics, bởi nó ảnh hưởng trực tiếp đến hiệu quả phân phối hàng hóa và chi phí vận hành. Việc giải quyết hiệu quả bài toán này có thể giúp doanh nghiệp giảm thiểu chi phí, nâng cao chất lượng dịch vụ và tối ưu hóa hoạt động vận tải.​

VRP và các biến thể của nó thuộc lớp các bài toán NP-khó, nghĩa là không có thuật toán giải chính xác nào có thể giải quyết chúng trong thời gian đa thức đối với mọi trường hợp. Do đó, trong thực tế, người ta thường sử dụng các phương pháp gần đúng như thuật toán tham lam, tìm kiếm cục bộ, hoặc các metaheuristic như thuật toán di truyền, tìm kiếm Tabu, và mô phỏng tôi luyện để tìm kiếm giải pháp gần tối ưu trong thời gian hợp lý.​

Với tầm quan trọng và tính ứng dụng rộng rãi trong thực tế, VRP đã và đang là chủ đề nghiên cứu sôi nổi trong hơn 50 năm qua, thu hút sự quan tâm của nhiều nhà khoa học và chuyên gia trong lĩnh vực tối ưu hóa, logistics và quản lý chuỗi cung ứng.​

## **2. Các khái niệm được sử dụng trong bài toán bao gồm**

**Xe Vận Chuyển (Vehicle)**

Xe là phương tiện thực hiện việc giao hàng, mỗi xe có thể có các đặc điểm riêng biệt như:

* Sức chứa (Capacity): Khối lượng hàng hóa tối đa mà xe có thể chở.
* Chi phí vận hành:
  + Chi phí cố định: Chi phí phát sinh khi xe bắt đầu hành trình, không phụ thuộc vào quãng đường di chuyển.
  + Chi phí biến đổi: Chi phí thay đổi theo quãng đường hoặc thời gian di chuyển.
* Giới hạn hành trình: Khoảng cách hoặc thời gian tối đa mà xe có thể di chuyển trong một hành trình.
* Loại hàng hóa phù hợp: Một số xe chỉ phù hợp với các loại hàng hóa nhất định (ví dụ: hàng đông lạnh, hàng dễ vỡ).

**Kho Hàng (Depot)**

Kho hàng là điểm xuất phát và kết thúc của các xe, nơi lưu trữ hàng hóa trước khi được phân phối. Trong một số bài toán, có thể có nhiều kho hàng với vị trí và khả năng phục vụ khác nhau.

**Khách Hàng (Customer)**

Mỗi khách hàng yêu cầu một lượng hàng hóa cụ thể và có thể có các ràng buộc như:

* Nhu cầu (Demand): Lượng hàng hóa cần được giao.
* Thời gian phục vụ (Time Window): Khoảng thời gian mà khách hàng có thể nhận hàng.
* Thời gian phục vụ (Service Time): Thời gian cần thiết để hoàn thành việc giao hàng tại điểm đó.

**Tuyến Đường (Route)**

Tuyến đường là chuỗi các điểm mà một xe sẽ ghé qua để giao hàng, bắt đầu và kết thúc tại kho hàng. Mỗi tuyến đường cần đảm bảo:

* Tổng nhu cầu của khách hàng trên tuyến không vượt quá sức chứa của xe.
* Tuân thủ các ràng buộc về thời gian và loại hàng hóa.
* Tối ưu hóa mục tiêu đề ra (ví dụ: tổng quãng đường, chi phí, thời gian).

**Mục Tiêu Định Tuyến**

Tùy thuộc vào yêu cầu cụ thể, bài toán VRP có thể hướng đến các mục tiêu như:

* Tối thiểu hóa tổng quãng đường hoặc chi phí vận chuyển.
* Tối thiểu hóa số lượng xe sử dụng.
* Tối đa hóa mức độ phục vụ khách hàng (ví dụ: giao hàng đúng thời gian).

## **3. Các biến thể của bài toán VRP**

Bài toán VRP có rất nhiều biến thể dựa trên các yêu cầu vận chuyển cụ thể của các bài toán thực tế. Trong phần này, tôi sẽ trình bày một số biến thể tồn tại trong thực tế của bài toán VRP và được phân chia theo từng đặc điểm cụ thể như đặc điểm về đội xe, về yêu cầu vận chuyển hay về vấn đề lợi nhuận, các ràng buộc nội tuyến và liên tuyền, ...

**CVRP (Capacitated VRP):** Đây là một trong những biến thể cơ bản và phổ biến nhất. Mỗi xe có một sức chứa nhất định và yêu cầu đặt ra là phải tìm tuyến đường cho các xe sao cho tổng khối lượng hàng hóa trên mỗi chuyến đi không vượt quá sức chứa xe.

**MDVRP (Multi-Depot VRP):** Trong biến thể này, thay vì chỉ có một kho hàng, bài toán có nhiều kho khác nhau. Mỗi xe chỉ được gắn với một kho cụ thể gọi là kho hàng chủ (home depot). Đây là mô hình phù hợp với các doanh nghiệp có nhiều trung tâm phân phối.

**VRPTW (VRP with Time Windows):** Trong biến thể này, mỗi khách hàng có một khoảng thời gian cho phép nhận hàng (time window). Xe có thể đến sớm hơn nhưng phải chờ đến đúng thời điểm mới được giao hàng, và không được đến trễ hơn thời điểm cho phép. Biến thể này áp dụng cho các hệ thống giao hàng theo lịch hẹn hoặc có giới hạn về giờ hoạt động.

**VRPPD (VRP with Pickup and Delivery):** Xe phải thực hiện cả hai nhiệm vụ: lấy hàng từ một số điểm (pickup) và giao hàng đến các điểm khác (delivery). Điều quan trọng là phải đảm bảo thứ tự ưu tiên: phải đến điểm lấy hàng trước điểm giao hàng tương ứng.

**VRPB (VRP with Backhauls):** Gần giống VRPPD, nhưng ở đây xe phải hoàn thành toàn bộ việc giao hàng (linehaul) trước, sau đó mới được quay lại nhận hàng từ các điểm backhaul để mang về kho. Mô hình này thường dùng trong logistics hai chiều.

**VRPM (VRP with Multiple Trips):** Cho phép mỗi xe được thực hiện nhiều hơn một chuyến đi trong một ngày, tức là xe có thể quay về kho để nạp thêm hàng rồi tiếp tục giao hàng. Điều kiện là tổng thời gian thực hiện không vượt quá giới hạn cho phép.

**SDVRP (Site-Dependent VRP):** Một số khách hàng chỉ cho phép một số loại xe nhất định phục vụ, do hạn chế về không gian hoặc quy định (ví dụ: hẻm nhỏ, cấm xe tải). Bài toán phải đảm bảo chỉ các xe phù hợp mới được phân tuyến đến những khách hàng này.

## **4. Các hướng tiếp cận**

Bài toán định tuyến xe (Vehicle Routing Problem - VRP) là một trong những bài toán tối ưu hóa tổ hợp có độ phức tạp rất cao, thuộc lớp NP-khó. Vì vậy, việc tìm ra lời giải tối ưu trong thời gian hợp lý là một thách thức lớn, đặc biệt khi bài toán có quy mô lớn hoặc nhiều ràng buộc thực tế. Các hướng tiếp cận giải bài toán này được chia thành hai nhóm chính: phương pháp chính xác và phương pháp gần đúng.

* **Phương pháp chính xác (Exact Methods)**

Nhóm phương pháp này hướng đến việc tìm ra lời giải tối ưu tuyệt đối cho bài toán. Tuy nhiên, do giới hạn về thời gian tính toán và tài nguyên xử lý, chúng thường chỉ khả thi với các bài toán VRP có quy mô nhỏ hoặc ít ràng buộc. Các kỹ thuật chính xác phổ biến bao gồm:

Thuật toán nhánh và cận (Branch and Bound): Tìm kiếm giải pháp tối ưu bằng cách chia nhỏ không gian tìm kiếm và loại bỏ các giải pháp không khả thi.

Quy hoạch động (Dynamic Programming): Chia bài toán thành các bài toán con và lưu trữ kết quả để tối ưu hóa thời gian tính toán.

Quy hoạch tuyến tính nguyên (Integer Linear Programming): Giải quyết bài toán tối ưu với các biến quyết định phải là số nguyên.

Thuật toán nhánh – cắt (Branch and Cut): Kết hợp nhánh và cận với kỹ thuật cắt để loại bỏ các giải pháp không khả thi trong quy hoạch tuyến tính nguyên.

Mặc dù đảm bảo được lời giải tối ưu, nhưng các phương pháp này hiếm khi được áp dụng trong các trường hợp thực tế lớn do chi phí tính toán quá cao.

* **Phương pháp gần đúng (Approximate Methods)**

Để giải quyết các bài toán VRP có quy mô lớn hoặc phức tạp hơn, các phương pháp gần đúng được sử dụng nhằm tìm ra lời giải gần tối ưu trong thời gian chấp nhận được. Nhóm này bao gồm các giải thuật heuristic cổ điển, tìm kiếm cục bộ (local search) và metaheuristic hiện đại.

**a) Thuật toán Heuristic**

Heuristic là những chiến lược dựa trên kinh nghiệm nhằm nhanh chóng xây dựng lời giải khả thi. Mặc dù không đảm bảo tối ưu tuyệt đối, các thuật toán này rất hiệu quả trong việc tạo ra lời giải tốt với chi phí thấp.

Các heuristic phổ biến cho bài toán VRP bao gồm:

* **Thuật toán Sweep**  
  Sweep là một trong những heuristic đơn giản và trực quan. Bằng cách sắp xếp khách hàng theo góc cực từ kho, thuật toán quét qua để tạo các cụm tuyến phù hợp với ràng buộc về sức chứa xe, sau đó sắp xếp thứ tự các điểm trong tuyến.
* **Thuật toán Clarke and Wright (Savings Algorithm)**  
  Đây là một heuristic nổi tiếng, dựa trên việc kết hợp các chuyến đi đơn lẻ thành các tuyến hiệu quả hơn bằng cách tính toán mức tiết kiệm chi phí (savings) khi gộp hai điểm khách hàng vào cùng một tuyến.
* **Thuật toán insertion (chèn khách hàng vào tuyến)**  
  Xây dựng lời giải từng bước bằng cách chọn vị trí chèn khách hàng tối ưu vào tuyến hiện có, đảm bảo hiệu quả chi phí và ràng buộc.
* **Chiến lược gom nhóm trước, lập tuyến sau (Cluster-first, Route-second)**  
  Phân nhóm các khách hàng thành từng cụm dựa trên vị trí địa lý, sau đó tối ưu tuyến cho từng cụm. Sweep chính là một đại diện tiêu biểu của chiến lược này.

**b) Tìm kiếm cục bộ (Local Search)**

Xuất phát từ một lời giải khả thi ban đầu (thường do heuristic tạo ra), local search cải tiến lời giải bằng cách thực hiện các thay đổi nhỏ có lợi (như hoán đổi điểm đến, chèn điểm mới, đảo ngược đoạn đường). Đây là nền tảng để kết hợp với các thuật toán metaheuristic nâng cao.

**c) Thuật toán Metaheuristic**

Metaheuristic là nhóm giải thuật hiện đại, kết hợp giữa tư duy heuristic với các chiến lược thông minh để khai phá không gian lời giải rộng hơn, tránh rơi vào cực trị cục bộ. Đây là nhóm phương pháp được sử dụng phổ biến nhất hiện nay trong các nghiên cứu và ứng dụng VRP thực tế.

Các metaheuristic tiêu biểu bao gồm:

* **Thuật toán Tabu Search**  
  Tabu Search sử dụng kỹ thuật bộ nhớ để tránh quay lại các trạng thái đã duyệt, nhờ đó vượt qua cực trị cục bộ và tiếp tục khám phá các lời giải tốt hơn trong không gian tìm kiếm.
* **Thuật toán Di truyền (Genetic Algorithm)**  
  Genetic Algorithm mô phỏng quá trình tiến hóa tự nhiên, sử dụng cơ chế lai ghép, đột biến và chọn lọc để tạo ra thế hệ lời giải mới ngày càng tốt hơn. Đây là thuật toán đặc biệt phù hợp với bài toán VRP nhờ khả năng mở rộng và đa dạng hóa lời giải.

1. **BÀI TOÁN ĐỊNH TUYẾN XE CÓ DUNG LƯỢNG GIỚI HẠN(CVRP)**
   1. **Giới thiệu bài toán**

Bài toán Định tuyến xe có dung lượng giới hạn (Capacitated Vehicle Routing Problem – CVRP) là một trong những bài toán cơ bản và quan trọng nhất trong lớp các bài toán định tuyến phương tiện (Vehicle Routing Problem – VRP). Bài toán lần đầu tiên được đề xuất bởi Dantzig và Ramser vào năm 1959, với bối cảnh thực tế là tối ưu hóa việc vận chuyển xăng đến các trạm xăng.

Bài toánCVRP phát biểu như sau: Cho một kho hàng và một tập các khách hàng với nhu cầu cụ thể, sử dụng một đội xe có giới hạn về sức chứa để giao hàng sao cho:

* Mỗi khách hàng đều được phục vụ đúng nhu cầu,
* Mỗi xe không vượt quá dung lượng tải cho phép,
* Mỗi tuyến đường bắt đầu và kết thúc tại kho,
* Và tổng quãng đường hoặc chi phí vận chuyển là nhỏ nhất.

CVRP có ý nghĩa quan trọng trong thực tiễn, đặc biệt trong các lĩnh vực vận tải, logistics, và phân phối hàng hóa, nơi mà chi phí vận chuyển chiếm một phần lớn trong tổng chi phí sản phẩm. Việc tối ưu hoá lộ trình giao hàng không chỉ giúp tiết kiệm chi phí mà còn nâng cao hiệu quả hoạt động và chất lượng dịch vụ.

Do CVRP là bài toán NP-hard, việc tìm ra lời giải tối ưu cho các bài toán có quy mô lớn là rất khó khăn. Vì vậy, các phương pháp giải thường được chia thành hai nhóm chính:

* Phương pháp chính xác (Exact methods): Như Quy hoạch động, Nhánh và cận, Nhánh – cắt,... đảm bảo tìm ra lời giải tối ưu nhưng tốn nhiều thời gian tính toán và thường chỉ áp dụng được cho các bài toán nhỏ (dưới 50 khách hàng).
* Phương pháp gần đúng và meta-heuristic: Như giải thuật Sweep, Clarke and Wright, 2-opt, các thuật toán tiến hóa,... giúp tìm ra lời giải tốt trong thời gian ngắn và thường được dùng cho các bài toán lớn (trên 100 khách hàng).

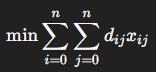
Với tầm quan trọng cả về mặt lý thuyết và ứng dụng thực tế, CVRP đã và đang là đối tượng nghiên cứu rộng rãi, với nhiều thuật toán và cải tiến được phát triển nhằm giải quyết bài toán hiệu quả hơn trong môi trường thực tiễn ngày càng đa dạng và phức tạp.

* 1. **Xây dựng mô hình toán học**Bài toán Định tuyến xe có dung lượng giới hạn (CVRP) có thể được mô hình hóa dưới dạng bài toán tối ưu tổ hợp. Mục tiêu là xác định các tuyến đường cho một đội xe bắt đầu và kết thúc tại kho hàng sao cho:
* Mỗi khách hàng được phục vụ đúng một lần bởi một xe duy nhất.
* Tổng nhu cầu của các khách hàng trên mỗi tuyến không vượt quá dung lượng chứa của xe.
* Tổng chi phí hoặc tổng khoảng cách di chuyển của tất cả các xe là nhỏ nhất.

**Ký hiệu:**

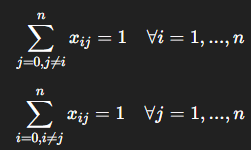
* G =(V,E) đồ thị có hướng hoặc vô hướng biểu diễn mạng lưới giao thông.
* V = {0,1,...,n: tập các đỉnh, trong đó đỉnh 0 là kho hàng, các đỉnh còn lại là khách hàng.
* dij ​: khoảng cách hoặc chi phí từ đỉnh i đến j.
* qi​: nhu cầu của khách hàng i.
* Q: dung lượng tối đa của mỗi xe.
* K: số lượng xe sẵn có.
* xij ∈ {0,1}: biến quyết định, bằng 1 nếu xe đi từ đỉnh i đến j, ngược lại bằng 0.
* ui: biến phụ để loại trừ chu trình con (subtour elimination).

**Hàm mục tiêu:**

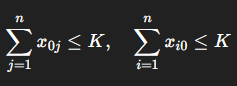


**Các ràng buộc:**

1. **Ràng buộc phục vụ mỗi khách hàng đúng một lần:**



1. **Ràng buộc khởi đầu và kết thúc tại kho:**



1. **Ràng buộc loại trừ chu trình con:**



1. **Giới hạn tải xe:**



1. **Ràng buộc nhị phân:**



1. **Phân loại các phương pháp giải bài toán CVRP**

**Phương pháp chính xác (Exact methods)**

Các phương pháp chính xác đảm bảo tìm ra lời giải tối ưu nhưng thường chỉ áp dụng được với bài toán có quy mô nhỏ do độ phức tạp tính toán cao.

* **Quy hoạch động (Dynamic Programming)**  
  Áp dụng cho các bài toán dạng chuỗi nhỏ. Dễ gặp giới hạn về bộ nhớ và thời gian khi mở rộng quy mô.
* **Nhánh và cận (Branch and Bound)**  
  Xây dựng cây tìm kiếm và cắt tỉa những nhánh không thể dẫn đến lời giải tốt hơn. Tốn nhiều thời gian khi số khách hàng lớn.
* **Nhánh – cắt (Branch and Cut)**  
  Kết hợp giữa phương pháp nhánh – cận và các bất đẳng thức cắt (cutting planes). Hiệu quả với các bài toán tuyến tính nguyên lớn hơn, thường là phương pháp mạnh nhất trong nhóm exact.

**Phương pháp gần đúng (Heuristics)**

Các phương pháp này tìm lời giải gần tối ưu với thời gian tính toán nhanh hơn, phù hợp với các bài toán thực tế lớn.

* **Sweep Algorithm**  
  Gom nhóm các khách hàng theo góc quay quanh kho và tạo tuyến. Đơn giản và hiệu quả cho phân bố địa lý tròn.
* **Clarke and Wright Saving Algorithm**  
  Dựa trên lợi ích tiết kiệm khi kết hợp hai tuyến riêng biệt thành một tuyến. Phổ biến vì tốc độ và tính đơn giản.
* **2-opt, 3-opt**  
  Cải tiến lời giải bằng cách hoán đổi các đoạn của tuyến đường để giảm tổng chi phí.

**Phương pháp meta-heuristic**

Giải pháp dựa trên nguyên lý tìm kiếm thông minh, mô phỏng các quá trình tự nhiên hoặc xã hội, thích hợp cho bài toán lớn và phức tạp.

* **Genetic Algorithm (GA)**  
  Mô phỏng quá trình tiến hóa tự nhiên với các bước: chọn lọc, lai ghép, đột biến. Tốt trong tìm kiếm không gian lớn.
* **Simulated Annealing (SA)**  
  Mô phỏng quá trình tôi luyện kim loại, cho phép chấp nhận lời giải xấu trong giai đoạn đầu để tránh kẹt cực tiểu cục bộ.
* **Tabu Search**  
  Dùng danh sách cấm (tabu list) để tránh lặp lại những giải pháp cũ, mở rộng tìm kiếm.

1. **CHI TIẾT CÁC THUẬT TOÁN**

## **A. THUẬT TOÁN SWEEP**

### **1. Giới Thiệu Về Thuật Toán**

#### **1.1 Đặt vấn đề**

Trong lĩnh vực vận tải và logistics, một trong những bài toán quan trọng và phổ biến nhất là làm sao để tối ưu hóa quá trình phân phối hàng hóa từ một kho trung tâm đến nhiều điểm giao hàng khác nhau. Việc lập kế hoạch vận chuyển hợp lý không chỉ giúp giảm chi phí vận hành, tiết kiệm thời gian và nhiên liệu, mà còn nâng cao chất lượng dịch vụ và mức độ hài lòng của khách hàng.

Trong đó, Bài toán "định tuyến phương tiện có ràng buộc về tải trọngđược xem là một mô hình điển hình để giải quyết vấn đề này. Bài toán yêu cầu xác định các tuyến đường di chuyển cho một đội xe có sức chứa giới hạn, sao cho mỗi khách hàng được phục vụ đúng nhu cầu, và tổng quãng đường di chuyển là ngắn nhất. Tuy nhiên, CVRP là một bài toán NP-hard trong toán học tổ hợp, nghĩa là khi số lượng khách hàng tăng lên, số phương án khả thi sẽ tăng theo cấp số nhân. Điều này khiến việc tìm lời giải tối ưu bằng phương pháp vét cạn (brute-force) gần như không khả thi về mặt tính toán.

Vì vậy, trong thực tế, người ta thường sử dụng các thuật toán heuristic – là các phương pháp tìm lời giải gần tối ưu trong khoảng thời gian hợp lý. Những thuật toán này không đảm bảo tìm ra lời giải tốt nhất tuyệt đối, nhưng có thể đưa ra lời giải đủ tốt để áp dụng vào thực tiễn.

Một trong những thuật toán heuristic nổi bật và dễ triển khai nhất là thuật toán Sweep. Đây là thuật toán xây dựng lời giải dựa trên nguyên tắc quét các điểm khách hàng theo thứ tự góc tọa độ cực, nhằm phân nhóm khách hàng thành các tuyến sao cho phù hợp với tải trọng của xe. Với ưu điểm về tốc độ xử lý nhanh, cấu trúc đơn giản và khả năng mở rộng, thuật toán Sweep đã và đang được ứng dụng rộng rãi trong các hệ thống phân phối thực tế của nhiều doanh nghiệp vận tải và thương mại điện tử.

#### **1.2 Giới thiệu về thuật toán Sweep**

Thuật toán Sweep được giới thiệu lần đầu bởi Wren vào năm 1971 và sau đó được phát triển thêm bởi Wren và Holliday vào năm 1972 để giải quyết bài toán CVRP với một hoặc nhiều kho trung tâm và các điểm giao hàng nằm trong mặt phẳng Euclide . Thuật ngữ "Sweep" được đặt ra bởi Gillett và Miller vào năm 1974 khi họ áp dụng phương pháp này cho các bài toán định tuyến phương tiện.​Quá trình thực hiện gồm các bước chính:

1. Chuyển đổi tọa độ của tất cả các khách hàng sang hệ tọa độ cực, lấy kho làm gốc tọa độ.
2. Sắp xếp các khách hàng theo thứ tự tăng dần của góc cực (theo chiều kim đồng hồ hoặc ngược chiều kim đồng hồ).
3. Quét từ khách hàng đầu tiên, lần lượt thêm vào tuyến cho đến khi tổng nhu cầu vượt quá tải trọng xe.
4. Lặp lại quá trình để lập tuyến tiếp theo.
5. Tối ưu thứ tự đi trong từng tuyến bằng các thuật toán TSP (như Nearest Neighbor).

Thuật toán Sweep đặc biệt hiệu quả khi các điểm giao hàng phân bố đều hoặc tương đối đồng đều xung quanh kho trung tâm. Với cấu trúc đơn giản, dễ hiểu, dễ cài đặt và tốc độ xử lý nhanh, thuật toán Sweep rất phù hợp với các ứng dụng thực tế cần giải nhanh trong các hệ thống hỗ trợ quyết định, phần mềm quản lý logistics hoặc hệ thống phân phối hàng hóa.

#### **1.3 Ý Tưởng thuật toán**

Ý tưởng cốt lõi của thuật toán Sweep là tận dụng đặc điểm vị trí địa lý của các điểm giao hàng để phân tách bài toán lớn CVRP thành các bài toán con nhỏ hơn và dễ dàng giải quyết. Thay vì tìm cách tối ưu hóa tuyến đường cho tất cả các khách hàng cùng lúc — một bài toán phức tạp và mất rất nhiều thời gian tính toán — thuật toán Sweep đi theo hướng phân nhóm khách hàng trước, rồi mới xây dựng các tuyến đường cho từng nhóm.

* **Chuyển đổi tọa độ và sắp xếp theo góc cực**

Bước đầu tiên trong thuật toán Sweep là chuyển đổi các tọa độ của khách hàng từ hệ Descartes (tọa độ x, y) sang hệ tọa độ cực (góc và khoảng cách từ kho trung tâm). Mỗi điểm khách hàng lúc này được coi như một vector nối từ kho đến vị trí của khách hàng đó.

Khi biểu diễn các điểm khách hàng trong hệ tọa độ cực, thuật toán tiến hành sắp xếp các khách hàng theo góc cực từ 0 đến 360 độ (theo chiều kim đồng hồ hoặc ngược chiều kim đồng hồ). Việc sắp xếp này là rất quan trọng, vì nó sẽ xác định thứ tự để quét các khách hàng xung quanh kho.

* **Quá trình quét và phân nhóm**

Sau khi đã sắp xếp các khách hàng theo góc, thuật toán bắt đầu quét qua các điểm khách hàng theo thứ tự góc tăng dần. Trong quá trình quét, mỗi khách hàng được thêm vào tuyến xe hiện tại cho đến khi tổng nhu cầu của nhóm vượt quá tải trọng của xe. Khi một nhóm đã đầy, thuật toán sẽ khởi tạo một tuyến mới và tiếp tục quét các khách hàng tiếp theo. Quá trình này sẽ được lặp lại cho đến khi tất cả các khách hàng được phân nhóm thành các tuyến hoàn chỉnh.

* **Tối ưu hóa tuyến đường**

Sau khi các nhóm khách hàng được phân chia, thuật toán sẽ sử dụng các thuật toán giải quyết bài toán TSP (Traveling Salesman Problem) như Nearest Neighbor hoặc các phương pháp tối ưu hóa khác để sắp xếp thứ tự các khách hàng trong từng tuyến. Mục tiêu là giảm thiểu tổng quãng đường di chuyển cho mỗi tuyến xe, giúp tăng hiệu quả phân phối.

* **Tính trực quan và ứng dụng thực tiễn**

Thuật toán Sweep có một lợi thế lớn là rất trực quan và dễ triển khai. Nó không yêu cầu quá nhiều tính toán phức tạp mà chỉ đơn giản là quét các khách hàng theo góc và phân nhóm chúng vào các tuyến. Điều này làm cho thuật toán rất phù hợp với các tình huống thực tế trong vận tải và phân phối hàng hóa.

Thuật toán này đặc biệt hiệu quả khi các điểm giao hàng rải rác đồng đều xung quanh kho trung tâm, vì khi đó, việc quét theo góc sẽ tạo ra các tuyến đường hợp lý mà không gặp phải sự phân bố khách hàng không đều, điều này có thể làm thuật toán kém hiệu quả.

### **2. Các Bước Triển Khai Chi Tiết của thuật toán ví dụ**

**Bước 1: Tiền Xử Lý và Chuyển Đổi Tọa Độ**

* **Đầu vào:**
  + **Depot:** Vị trí của kho trung tâm, với tọa độ (x0,y0)
  + **Khách hàng:** Danh sách các khách hàng với tọa độ (xi,yi) và nhu cầu tương ứng di​ (với i = 1,2,…,n).
  + **Giới hạn tải trọng:** Sức chứa của mỗi xe vận chuyển, ký hiệu là Q.
* **Phân tích:**
  + Bước này đảm bảo rằng dữ liệu về các khách hàng, bao gồm tọa độ và nhu cầu, được chuẩn bị hợp lệ.
  + Chuyển đổi từ hệ tọa độ Descartes sang hệ tọa độ cực giúp đơn giản hóa việc phân nhóm dựa trên vị trí góc của khách hàng so với depot.
* **Thực hiện:**
  + Với mỗi khách hàng, sử dụng hàm atan2 để tính góc cực θi so với kho trung tâm:

θi = atan2 (yi−y0, xi−x0)

Đồng thời, tính khoảng cách từ depot nếu cần (để có thể sử dụng cho giai đoạn tối ưu tuyến sau này).

**Bước 2: Sắp Xếp Các Khách Hàng Theo Góc Cực**

* **Ý tưởng:**
  + Sắp xếp các khách hàng theo thứ tự tăng dần của θi từ 0 đến 2π (hoặc từ 0° đến 360°).
* **Thực hiện:**
  + Tạo danh sách các khách hàng đã gắn giá trị θi
  + Sắp xếp danh sách này theo thứ tự tăng dần của góc.
* **Kết quả:**
  + Có được một danh sách khách hàng được sắp xếp, phản ánh “trật tự quét” xung quanh depot.

**Bước 3: Quét và Phân Nhóm Khách Hàng (Tạo Tuyến)**

* **Quy tắc phân nhóm dựa trên tải trọng:**
  + Khởi tạo một tuyến mới, với tải trọng hiện tại load=0 và danh sách khách hàng rỗng.
  + Duyệt qua các khách hàng theo thứ tự đã sắp xếp:
    - **Nếu:** load + di ≤ Q

⇒ Thêm khách hàng i vào tuyến hiện tại và cập nhật

load = load + di

* + - **Nếu:** load + di > Q   
      ⇒ Kết thúc tuyến hiện tại, lưu lại tuyến vào danh sách kết quả, sau đó khởi tạo một tuyến mới bắt đầu với khách hàng i và đặt load = di
* **Xử lý trường hợp đặc biệt:**
  + Nếu nhu cầu của một khách hàng riêng lẻ vượt quá Q, cần có chiến lược xử lý riêng (ví dụ: có thể chia nhỏ nhu cầu hoặc đánh dấu trường hợp không khả thi).
* **Kết quả:**
  + Thu được tập hợp các tuyến, mỗi tuyến chứa danh sách các khách hàng với tổng nhu cầu không vượt quá tải trọng Q.

**Bước 4: Tối Ưu Hóa Thứ Tự Đi Trong Từng Tuyến (Ứng Dụng TSP)**

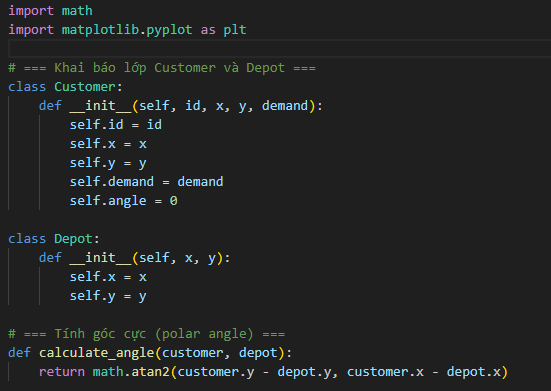
* **Mục tiêu:**
  + Sau khi phân nhóm thành các tuyến, mỗi tuyến cần được tối ưu hóa để rút ngắn quãng đường di chuyển tổng thể.
* **Phương pháp:**
  + Với mỗi tuyến đã có, áp dụng thuật toán giải bài toán người bán hàng (TSP) để sắp xếp thứ tự ghé thăm các khách hàng.
  + Các heuristic phổ biến như:
    - **Nearest Neighbor:** Lựa chọn khách hàng gần nhất tiếp theo.
    - **2-opt hoặc 3-opt:** Cải tiến tuyến đường đã có bằng cách hoán đổi vị trí các điểm để giảm tổng khoảng cách.
* **Kết quả:**
  + Mỗi tuyến được tối ưu về mặt đường đi, giúp giảm thiểu quãng đường và thời gian di chuyển thực tế.

**Bước 5: Tổng Hợp Kết Quả và Đánh Giá**

* **Tổng hợp:**
  + Gộp các tuyến đã tối ưu lại để tạo thành bảng kế hoạch vận tải hoàn chỉnh.
  + Tính tổng quãng đường di chuyển của tất cả các tuyến và so sánh với các phương pháp khác (nếu cần).
* **Đánh giá:**
  + Xác định số xe cần sử dụng.
  + Kiểm tra tổng nhu cầu phục vụ có thỏa mãn ràng buộc về tải trọng không.
  + Đánh giá hiệu quả của thuật toán theo thời gian tính toán và chất lượng lời giải (gần tối ưu, tối ưu được hay không).
* **Phản hồi và cải tiến:**
  + Nếu lời giải không đạt yêu cầu, có thể xem xét áp dụng các kỹ thuật cải tiến cục bộ (như 2-opt, 3-opt) trên toàn bộ kết quả.

### **3. Pseudocode**

### **4. Code python**



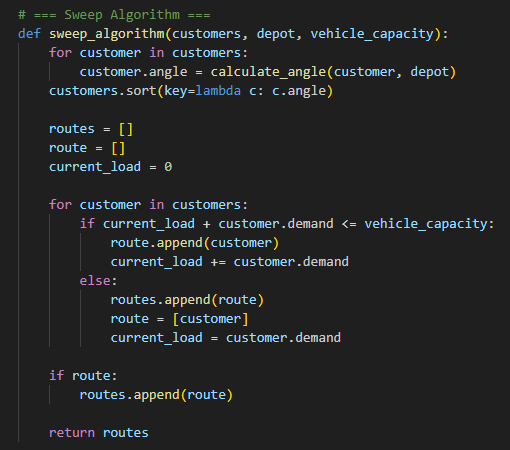
**import math**: Nhập thư viện math để sử dụng các hàm toán học, cụ thể là atan2 để tính góc.   
**import matplotlib.pyplot as plt**: Nhập thư viện vẽ đồ thị matplotlib.pyplot

**class Customer:**Định nghĩa lớp Customer để biểu diễn thông tin của một khách hàng:

* \_\_init\_\_: Phương thức khởi tạo, nhận id, tọa độ x, y, nhu cầu demand và gán chúng cho thuộc tính của đối tượng Customer. Thuộc tính angle được khởi tạo bằng 0.

**class Depot:** Định nghĩa lớp Depot để biểu diễn thông tin của một kho hàng:

**def calculate\_angle(customer, depot):** Định nghĩa hàm calculate\_angle nhận vào một đối tượng Customer và một đối tượng Depot.



**def sweep\_algorithm**

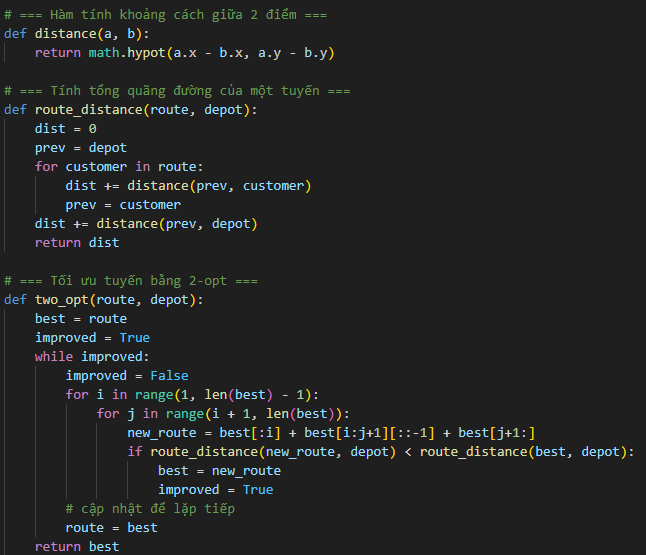
Hàm này nhận vào ba tham số:

* customers: Một danh sách các đối tượng Customer.
* depot: Một đối tượng Depot (điểm xuất phát và kết thúc của các xe).
* vehicle\_capacity: Sức chứa tối đa của mỗi xe.

**customer.angle = calculate\_angle(customer, depot)**: Tính toán góc cực của mỗi khách hàng so với depot

**customers.sort(key=lambda c: c.angle)**: Sắp xếp danh sách customers theo thứ tự tăng dần của thuộc tính angle.

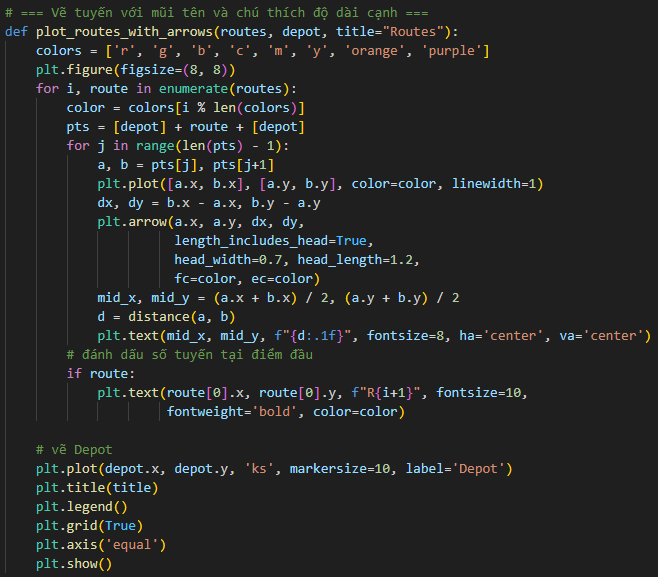
**if current\_load + customer.demand <= vehicle\_capacity:** Kiểm tra xem việc thêm khách hàng hiện tại vào tuyến đường hiện tại có vượt quá sức chứa của xe hay không.

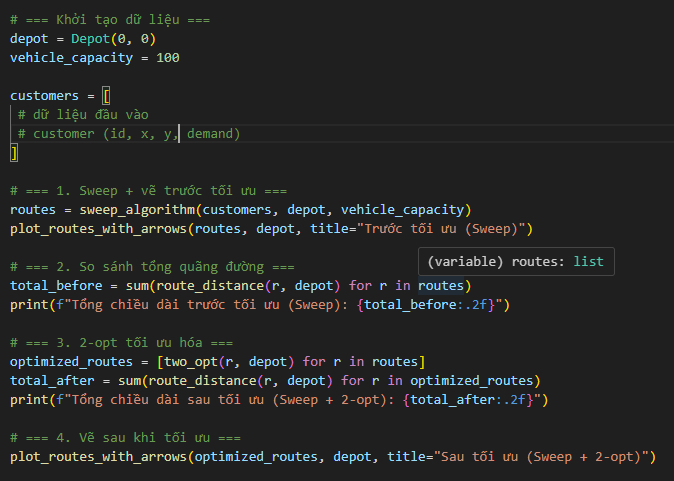


**dist += distance(prev, customer)**: Tính khoảng cách từ điểm trước đó (prev) đến khách hàng hiện tại và cộng vào tổng quãng đường.

**dist += distance(prev, depot)**: Sau khi ghé thăm tất cả khách hàng, tính khoảng cách từ khách hàng cuối cùng (prev) trở về depot và cộng vào tổng quãng đường.

**new\_route = best[:i] + best[i:j+1][::-1] + best[j+1:]**: Tạo một tuyến đường mới bằng cách đảo ngược thứ tự các khách hàng giữa vị trí i và j trong tuyến đường hiện tại (best). Đây là thao tác "2-opt" - loại bỏ hai cạnh và thay thế bằng hai cạnh mới.





### **5. Ví dụ minh hoạt thuật toán**

**CVRP : 1 Depot + 4 Khách hàng**

* **Depot: Tọa độ (0, 0), Capacity mỗi xe: 30**
* **Khách hàng:**

| **ID** | **Tọa độ (X, Y)** | **Nhu cầu** |
| --- | --- | --- |
| **1** | **(4, 1)** | **10** |
| **2** | **(6, 2)** | **15** |
| **3** | **(-2, 3)** | **10** |
| **4** | **(-4, 1)** | **10** |

**Bước 1: Tính góc từ Depot đến từng khách**

**Tính góc (theta) bằng atan2(y, x) (đơn vị: độ):**

| **ID** | **(X,Y)** | **Nhu cầu** | **Góc (°)** |
| --- | --- | --- | --- |
| **1** | **(4,1)** | **10** | **~14°** |
| **2** | **(6,2)** | **15** | **~18°** |
| **3** | **(-2,3)** | **10** | **~123°** |
| **4** | **(-4,1)** | **10** | **~165°** |

**Bước 2: Sắp xếp theo góc tăng dần**

Kết quả thứ tự quét: [1 → 2 → 3 → 4]

**Bước 3: Gom khách vào các tuyến**

**Xe 1**:

* Ghé khách 1 (10) → khách 2 (15) → Tổng: 25 ≤ 30
* Tuyến 1: Depot → 1 → 2 → Depot

**Xe 2**:

* Ghé khách 3 (10) → khách 4 (10) → Tổng: 20 ≤ 30
* Tuyến 2: Depot → 3 → 4 → Depot

1. **Phân tích hiệu suất và tham số ( có thể lấy dữ liệu từ chính ví dụ )**

**Thời gian chạy (Time Complexity)**

* Sweep algorithm có thời gian xử lý rất nhanh, với độ phức tạp chính:

| **Mô tả** | **Độ phức tạp** |
| --- | --- |
| Tính góc đến từng khách hàng | O(n) |
| Sắp xếp khách theo góc | O(n log n) |
| Gom nhóm theo sức chứa | O(n) |

* Tổng thời gian chạy: O(n log n) cực kì nhanh

**Độ chính xác so với lời giải tối ưu**

* Sweep thường không cho lời giải tối ưu, nhưng:

| **Tình huống** | **Hiệu quả** |
| --- | --- |
| Khách hàng phân bố đều quanh depot (tròn/quạt) | Rất tốt |
| Khách hàng phân bố cụm hoặc lệch | Có thể kém tối ưu |
| Không có tối ưu cục bộ (2-opt, 3-opt...) | Dễ bị đường zig-zag |

**Tham số ảnh hưởng đến kết quả**

| **Tham số** | **Ảnh hưởng** |
| --- | --- |
| **Vị trí depot** | Trung tâm → góc phân bố hợp lý; lệch → góc sai lệch → nhóm không tối ưu |
| **Tọa độ** | Càng bất đối xứng → kết quả càng kém |
| **Vehicle capacity** | Nếu nhỏ → nhiều nhóm hơn → cần sắp nhóm thông minh hơn |
| **Thứ tự quét (góc)** | Có thể sweep từ 0° hoặc từ 90°, 180° để cải thiện nhóm |
| **Tối ưu sau Sweep** | Dùng 2-opt hoặc metaheuristic sẽ tăng độ chính xác đáng kể |

### **7. Ưu Và Nhược Điểm**

**Ưu Điểm**

* **Đơn giản và trực quan:**  
  Thuật toán Sweep dựa trên ý tưởng chuyển tọa độ và quét theo góc, giúp người dùng dễ hình dung quá trình phân nhóm khách hàng theo vị trí địa lý quanh kho trung tâm.
* **Tốc độ xử lý nhanh:**  
  Vì không cần phải duyệt qua tất cả các khả năng kết hợp, thuật toán hoạt động hiệu quả với độ phức tạp tính toán thấp, phù hợp với các tình huống cần phản ứng nhanh trong thực tiễn.
* **Giảm số lượng xe sử dụng:**  
  Qua việc phân nhóm khách hàng theo thứ tự quét và kiểm tra tổng nhu cầu so với tải trọng, thuật toán giúp tạo ra các tuyến hợp lý, tối ưu hóa số lượng xe cần thiết cho việc phân phối.
* **Khả năng áp dụng rộng rãi:**  
  Đặc biệt hiệu quả khi các điểm giao hàng phân bố đồng đều quanh depot, thuật toán Sweep có thể dùng như một giải pháp khởi đầu hoặc kết hợp với các thuật toán tối ưu cục bộ để cải thiện thêm kết quả.

**Nhược Điểm**

* **Phụ thuộc vào phân bố khách hàng:**  
  Nếu các điểm giao hàng có sự phân bố không đều (ví dụ, tập trung quá đông hay quá rời rạc ở một vài góc), thuật toán có thể tạo ra những tuyến không tối ưu.
* **Không đảm bảo tối ưu tuyệt đối:**  
  Do là một phương pháp heuristic, thuật toán Sweep chỉ cho ra lời giải gần tối ưu. Trong một số trường hợp phức tạp, kết quả có thể bị lệch so với lời giải tối ưu tuyệt đối.
* **Hạn chế trong điều kiện địa hình phức tạp:**  
  Nếu điều kiện thực tế có nhiều yếu tố phức tạp (rào cản giao thông, địa hình gồ ghề), việc dựa vào khoảng cách Euclid và góc để phân nhóm có thể không phản ánh đầy đủ các yêu cầu thực tế.
* **Kết hợp với TSP cần tối ưu thêm:**  
  Sau khi phân nhóm, việc tối ưu thứ tự các khách hàng trong từng tuyến bằng các thuật toán TSP (như Nearest Neighbor hay 2-opt) lại là một quá trình phụ thuộc và có thể tốn thời gian nếu không được chọn lựa kỹ

### **8. Ứng Dụng**

Thuật toán Sweep được sử dụng rộng rãi trong các lĩnh vực liên quan đến vận tải và logistics. Một số ứng dụng tiêu biểu bao gồm:

* **Quản lý và lập kế hoạch vận tải:**  
  Tích hợp vào hệ thống phần mềm quản lý logistics để xác định các tuyến đường giao hàng hợp lý từ kho trung tâm đến các điểm giao hàng.
* **Phân phối hàng hóa cho các doanh nghiệp thương mại điện tử:**  
  Giúp tối ưu hóa lộ trình giao hàng trong hệ thống phân phối của các nền tảng thương mại điện tử, đặc biệt khi có nhiều điểm giao hàng được đặt quanh khu vực đô thị.
* **Ứng dụng trong ngành giao nhận và vận chuyển:**  
  Sử dụng trong các công ty giao hàng nhanh, dịch vụ chuyển phát, nơi yêu cầu lên tuyến đường di chuyển một cách tối ưu và giảm chi phí nhiên liệu.
* **Hỗ trợ ra quyết định trong logistic:**  
  Dùng như một công cụ ban đầu để phân tích và đề xuất các giải pháp phân phối, sau đó có thể kết hợp với các thuật toán tối ưu hóa khác trong việc nâng cao hiệu quả hệ thống.

### **9. Kết Luận**

Thuật toán Sweep thể hiện rõ vai trò của các giải pháp heuristic trong việc giải quyết các bài toán tối ưu tổ hợp phức tạp như CVRP. Nhờ vào tính trực quan và cấu trúc phân nhóm đơn giản, thuật toán giúp chia nhỏ bài toán lớn thành các bài toán con dễ xử lý hơn. Điều này góp phần giảm đáng kể độ phức tạp tính toán và cho phép hệ thống đưa ra quyết định nhanh chóng, đặc biệt phù hợp trong bối cảnh các hệ thống logistics hiện đại đòi hỏi phản ứng linh hoạt và kịp thời.

Tuy vậy, Sweep không đảm bảo tìm ra lời giải tối ưu toàn cục. Kết quả của thuật toán phụ thuộc vào vị trí không gian của các điểm giao hàng và cách sắp xếp thứ tự ghé thăm trong từng nhóm. Chính vì thế, thuật toán thường được sử dụng như bước khởi tạo lời giải ban đầu, sau đó kết hợp với các kỹ thuật tối ưu cục bộ như 2-opt, 3-opt hoặc các phương pháp metaheuristic để cải thiện chất lượng lời giải.

Dù còn một số hạn chế, thuật toán Sweep vẫn giữ vai trò quan trọng nhờ sự cân bằng giữa tính đơn giản, tốc độ và khả năng mở rộng. Trong thực tế, nó được áp dụng rộng rãi trong các doanh nghiệp vận tải, giao nhận, và hệ thống quản lý phân phối hàng hóa. Sweep không phải là “lời giải cuối cùng” cho bài toán CVRP, nhưng là một công cụ hữu ích để nhanh chóng tạo ra các phương án khả thi, từ đó làm nền tảng cho những giải pháp tối ưu hơn trong các hệ thống vận hành phức tạp.

### **B) THUẬT TOÁN CLARKE AND WRIGHT.**

### **1. Giới thiệu về thuật toán**

* 1. ***Giới thiệu về thuật toán Clarke and Wright.***

Thuật toán **Clarke and Wright Savings Algorithm**, được đề xuất bởi **G. Clarke** và **J.W. Wright** vào năm 1964, là một phương pháp heuristic dựa trên khái niệm "tiết kiệm" (savings) để xây dựng các tuyến đường cho CVRP. Thuật toán này đặc biệt hiệu quả trong việc tạo ra các tuyến đường khả thi với chi phí thấp mà không cần khám phá toàn bộ không gian lời giải.Thuật toán bắt đầu bằng việc giả định mỗi khách hàng được phục vụ bởi một xe riêng (tuyến độc lập từ depot đến khách hàng và quay về). Sau đó, nó hợp nhất các tuyến dựa trên giá trị tiết kiệm chi phí (savings) khi kết nối nhiều khách hàng vào cùng một tuyến, đồng thời đảm bảo không vi phạm ràng buộc sức chứa của xe. Đây được xem là một giải pháp kinh điển vì:

* **Hiệu quả:** Nó thường tìm ra các lộ trình tốt, giúp giảm đáng kể chi phí vận hành so với các cách làm đơn giản.
* **Nhanh chóng:** Thuật toán này tính toán tương đối nhanh, phù hợp cho các bài toán có quy mô vừa phải.
* **Dễ hiểu:** Logic đằng sau nó khá trực quan, dựa trên một nguyên tắc cốt lõi là "tiết kiệm".
* **Nền tảng:** Nó là một *thuật toán heuristic xây dựng* (từng bước tạo ra giải pháp) và có tính *tham lam* (luôn ưu tiên lựa chọn mang lại lợi ích lớn nhất ngay tại thời điểm quyết định).

#### **1.2 Ý Tưởng thuật toán Clarke and Wright.**

**1.2.1** Thuật toán **Clarke and Wright Savings Algorithm** hoạt động dựa trên việc tối ưu hóa chi phí di chuyển bằng cách hợp nhất các tuyến đường riêng lẻ thành các tuyến chung, trong khi đảm bảo xe có thể phục vụ nhiều khách hàng nhất có thể mà không vượt quá sức chứa Q. Ý tưởng cốt lõi là:

* **Khởi tạo (Tốn kém nhất):** Ban đầu, thuật toán giả định tình huống tệ nhất: mỗi khách hàng i được phục vụ bởi một chuyến xe riêng biệt đi từ Kho đến i rồi quay về Kho (Lộ trình: 0→i→0).
* Tính **giá trị tiết kiệm** (savings) khi hợp nhất hai khách hàng i và j vào một tuyến chung (depot → i → j → depot) thay vì hai tuyến riêng.
* Ưu tiên hợp nhất các cặp khách hàng có giá trị tiết kiệm lớn, đồng thời kiểm tra ràng buộc sức chứa để đảm bảo tuyến mới không vượt quá Q.
* Tiếp tục hợp nhất cho đến khi không thể thêm khách hàng vào tuyến nào nữa, đảm bảo tối ưu hóa số lượng khách hàng trên mỗi tuyến và giảm thiểu tổng quãng đường.

**1.2.2 Khởi tạo:**

* -Mỗi khách hàng i (i = 1, 2, ..., n) được gán một tuyến riêng: depot (0) → i → depot.
* -Tính ma trận khoảng cách Euclidean giữa depot và các khách hàng, cũng như giữa các khách hàng:
* Ghi nhận nhu cầu di của mỗi khách hàng i và sức chứa Q của xe

**1.2.3 Tính giá trị tiết kiệm:**

* Với mỗi cặp khách hàng (i, j), tính giá trị tiết kiệm khi hợp nhất hai tuyến chứa i và j:

S(i,j) = c(0,i) +c (0,j) − c(i,j)

* Trong đó:
* c(0,i): Khoảng cách từ depot đến khách hàng i.
* c(0,j)c(0,j)c(0,j): Khoảng cách từ depot đến khách hàng j.
* c(i,j)c(i,j)c(i,j): Khoảng cách từ khách hàng i đến khách hàng j.
* s(i,j)s(i,j)s(i,j): Giá trị tiết kiệm nếu nối i và j vào một tuyến.
* Sắp xếp danh sách các giá trị s(i,j)s(i,j)s(i,j) theo thứ tự giảm dần.

**1.2.4 Xây dựng tuyến:**

* Bắt đầu với một tuyến rỗng hoặc tuyến đầu tiên.
* Duyệt qua danh sách tiết kiệm từ lớn đến nhỏ. Với mỗi cặp (i, j):
* Kiểm tra xem i và j có thể được hợp nhất vào tuyến hiện tại hay không:
* i và j phải nằm ở đầu hoặc cuối của tuyến hiện tại (tức là gần depot trong trình tự tuyến).
* Tổng nhu cầu của tuyến sau khi thêm i và j không vượt quá sức chứa Q:

với dk​ là nhu cầu của các khách hàng trong tuyến.

* Việc hợp nhất không tạo vòng (cycle) trong tuyến.
* Nếu hợp nhất khả thi, cập nhật tuyến bằng cách thêm i và j, đồng thời cập nhật tổng nhu cầu và quãng đường của tuyến.
* Nếu tuyến hiện tại không thể thêm khách hàng nữa (do ràng buộc sức chứa hoặc không còn hợp nhất khả thi), bắt đầu một tuyến mới và lặp lại quá trình.

**1.2.5 Kết thúc:**

* Lặp lại bước hợp nhất cho đến khi tất cả khách hàng được gán vào các tuyến hoặc không còn cặp (i, j) nào khả thi.
* Kết quả là tập hợp các tuyến, trong đó mỗi tuyến chứa số lượng khách hàng tối đa có thể (trong giới hạn sức chứa Q) và tổng quãng đường được giảm thiểu.

**1.2.6 Đặc điểm bài toán:**

-Tối ưu số lượng khách hàng: Thuật toán này ưu tiên mở rộng một tuyến để chứa nhiều khách hàng nhất có thể trước khi tạo tuyến mới, phù hợp với yêu cầu "xe đáp ứng nhu cầu nhiều khách hàng nhất".

-Tối ưu quãng đường: Việc sắp xếp tiết kiệm theo thứ tự giảm dần đảm bảo các hợp nhất có lợi nhất (giảm quãng đường nhiều nhất) được thực hiện trước.

-Xử lý tọa độ (x, y): Khoảng cách Euclidean được sử dụng để tính chi phí, phù hợp với dữ liệu tọa độ trong đề bài.

### **2. Các Bước Triển Khai Chi Tiết của thuật toán ví dụ: 1) Input đầu vào:**

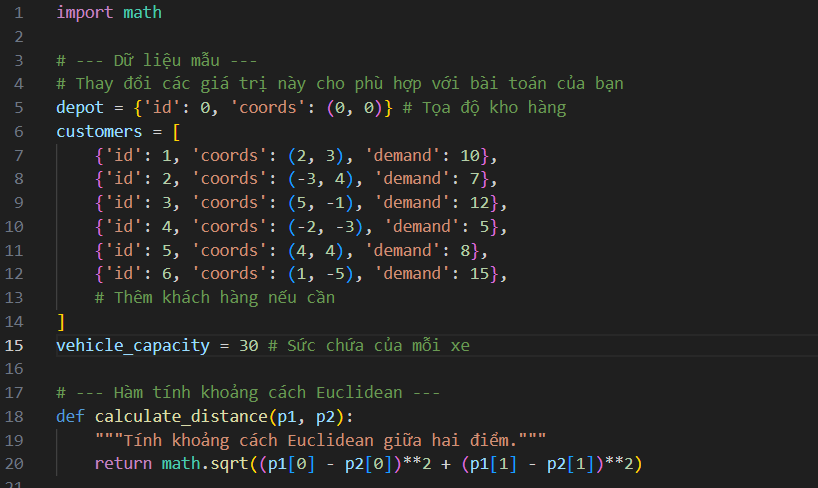
* Tọa độ kho (Depot): D = (0 , 0)
* Danh sách khách hàng: C={1,2,..,n}
* Trong đó:
* Tọa độ khách hàng: Pi = (xi , yi).
* Nhu cầu (Demand): qi,
* Sức chứa của xe: Q .Giả định tất cả các xe có cùng sức chứa.
* **Số lượng xe:** Không cần định trước cho C&W cơ bản, thuật toán sẽ tự xác định số xe cần thiết.

**2) Các bước triển khai trực tiếp:**

* ***Bước 1:***
* Tính khoảng cách Euclid giữa tất cả các cặp điểm (Kho và Khách hàng, Khách hàng và Khách hàng).
* Khoảng cách giữa hai điểm A=(xA​,yA​) và B=(xB​,yB​) là:
* Lưu trữ các khoảng cách này vào một ma trận (hoặc cấu trúc dữ liệu tương tự) để dễ dàng truy xuất:
* d0i​: Khoảng cách từ Kho (0) đến Khách hàng i.
* dij​: Khoảng cách từ Khách hàng i đến Khách hàng j.
* ***Bước 2:***
* Với mọi cặp khách hàng **phân biệt** (i,j) (i khác j, i,j ∈ C):
* Tính giá trị tiết kiệm khi nối trực tiếp i và j thay vì cả hai cùng quay về Kho: sij ​= d0i ​+ d0j ​− dij​
* Lưu các giá trị tiết kiệm này vào một danh sách, thường kèm theo cặp khách hàng tương ứng. Ví dụ: List\_Savings = [(s\_12, 1, 2), (s\_13, 1, 3), ..., (s\_n-1,n, n-1, n)].
* ***Bước 3:*** Khởi tạo các tuyến đường ban đầu
* Tạo ra n tuyến đường ban đầu, mỗi tuyến phục vụ đúng một khách hàng: Tuyến Ri​=[0,i,0] (Đi từ Kho, đến khách hàng i, quay về Kho).
* Theo dõi tải trọng (load) hiện tại của mỗi tuyến: Load(Ri​)=qi​.
* Lưu trữ thông tin: Mỗi khách hàng i hiện đang thuộc tuyến Ri​.
* ***Bước 4:*** Sắp xếp Danh sách Tiết kiệm
* Sắp xếp List\_Savings theo thứ tự **giảm dần** của giá trị tiết kiệm sij​.
* ***Bước 5:*** Vòng lặp Hợp nhất Tuyến đường.
* Duyệt qua danh sách List\_Savings đã sắp xếp từ trên xuống dưới (từ tiết kiệm cao nhất). Với mỗi phần tử (s\_ij, i, j):
* **Xác định Tuyến:** Tìm tuyến đường hiện tại chứa khách hàng i (gọi là Ri​) và tuyến đường hiện tại chứa khách hàng j (gọi là Rj​).
* **Kiểm tra Khác Tuyến:** Ri​ và Rj​ có phải là hai tuyến khác nhau không? Nếu chúng giống nhau (tức là i và j đã cùng trên một tuyến), bỏ qua bước này và xét cặp tiết kiệm tiếp theo.
* Kiểm tra Khả năng Nối (Điều kiện Quan trọng):
* Khách hàng i có phải là điểm **kề Kho** trong tuyến Ri​ không? (Tức là i là điểm ngay sau Kho hoặc ngay trước Kho trong chuỗi điểm của tuyến Ri​).
* Khách hàng j có phải là điểm **kề Kho** trong tuyến Rj​ không?
* **Điều kiện nối cơ bản:** Chỉ thực hiện nối nếu i là điểm *cuối cùng* trước khi về Kho trong Ri​ (vd: [0,...,k,i,0]) VÀ j là điểm *đầu tiên* sau khi rời Kho trong Rj​ (vd: [0,j,l,...,0]).
* **Kiểm tra Sức chứa:** Tính tổng nhu cầu của tuyến mới nếu hợp nhất: Total\_Load=Load(Ri​)+Load(Rj​). Tổng tải này có **nhỏ hơn hoặc bằng** sức chứa xe Q không (Total\_Load≤Q)?
* Thực hiện Hợp nhất (Nếu tất cả điều kiện trên đều Đúng)
* **Ghép chuỗi:** Tạo tuyến mới Rnew​ bằng cách ghép phần đầu của Ri​ (bỏ điểm 0 cuối) với phần sau của Rj​ (bỏ điểm 0 đầu). Ví dụ: Nếu Ri​=[0,...,k,i,0] và Rj​=[0,j,l,...,0], thì Rnew​=[0,...,k,i,j,l,...,0].
* **Cập nhật Tải trọng:** Load(Rnew​) = Total\_Load.
* **Cập nhật Danh sách Tuyến:** Xóa Ri​ và Rj​ khỏi danh sách các tuyến đang hoạt động. Thêm Rnew​ vào.
* **Cập nhật Thông tin Khách hàng:** Ghi nhận rằng tất cả khách hàng trước đây thuộc Ri​ hoặc Rj​ bây giờ đều thuộc Rnew​. Lặp lại quá trình trên cho đến khi duyệt hết danh sách List\_Savings.
* ***Bước 6:*** Hoàn thiện và xuất kết quả

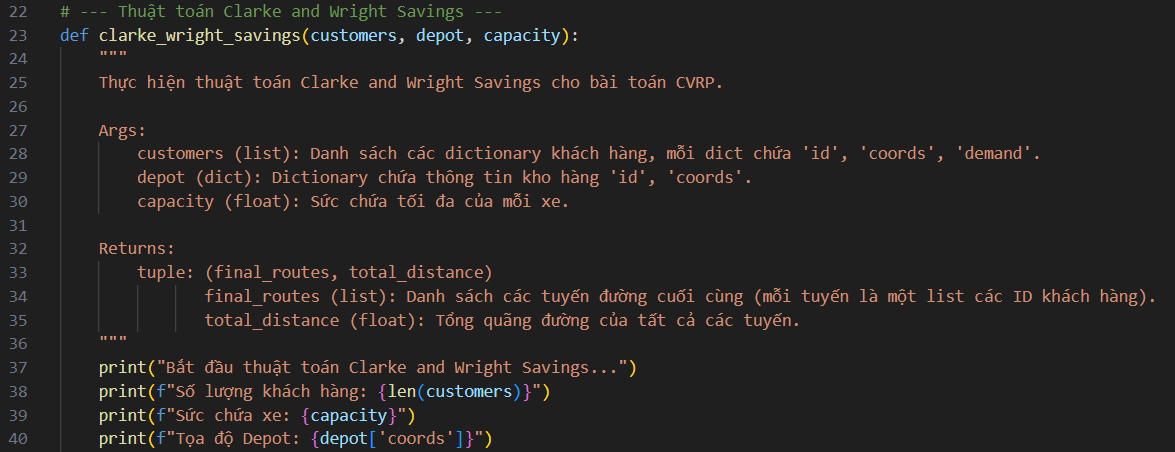
### **3. Pseucode**

### **4. Code python**



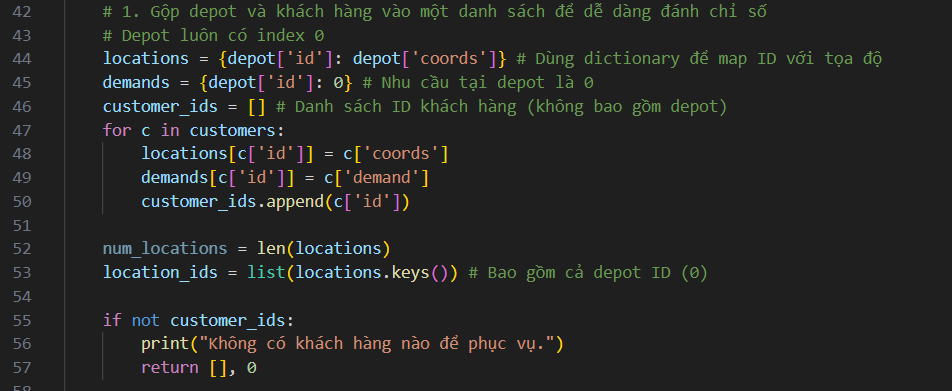
Hình 1 Thiết lập dữ liệu đầu vào

-Đoạn code này thiết lập các dữ liệu đầu vào cần thiết cho một bài toán tối ưu hóa tuyến đường vận chuyển (Vehicle Routing Problem - VRP), bao gồm vị trí kho, danh sách khách hàng (vị trí và nhu cầu), sức chứa của xe. Nó cũng định nghĩa một hàm tiện ích quan trọng để tính khoảng cách giữa hai điểm bất kỳ dựa trên tọa độ của chúng. Đây là những thành phần cơ bản để bắt đầu xây dựng một giải thuật VRP như Clarke and Wright Savings.



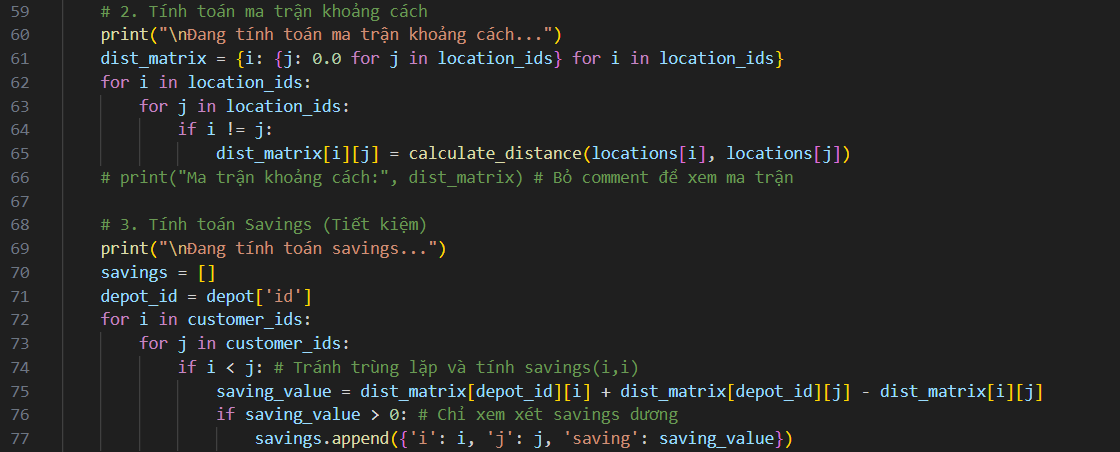
Hình 2 Định nghĩa hàm chính clarke\_wright\_savings

-Đoạn code này định nghĩa hàm chính clarke\_wright\_savings để giải bài toán CVRP bằng thuật toán Clarke and Wright. Nó có phần mô tả (docstring) rất rõ ràng về chức năng, đầu vào và đầu ra. Phần thân hàm bắt đầu bằng việc in ra các thông tin cơ bản của bài toán như số khách hàng, sức chứa xe và vị trí kho hàng để người dùng dễ theo dõi quá trình chạy



Hình 3 Tạo các cấu trúc dữ liệu quan trọng

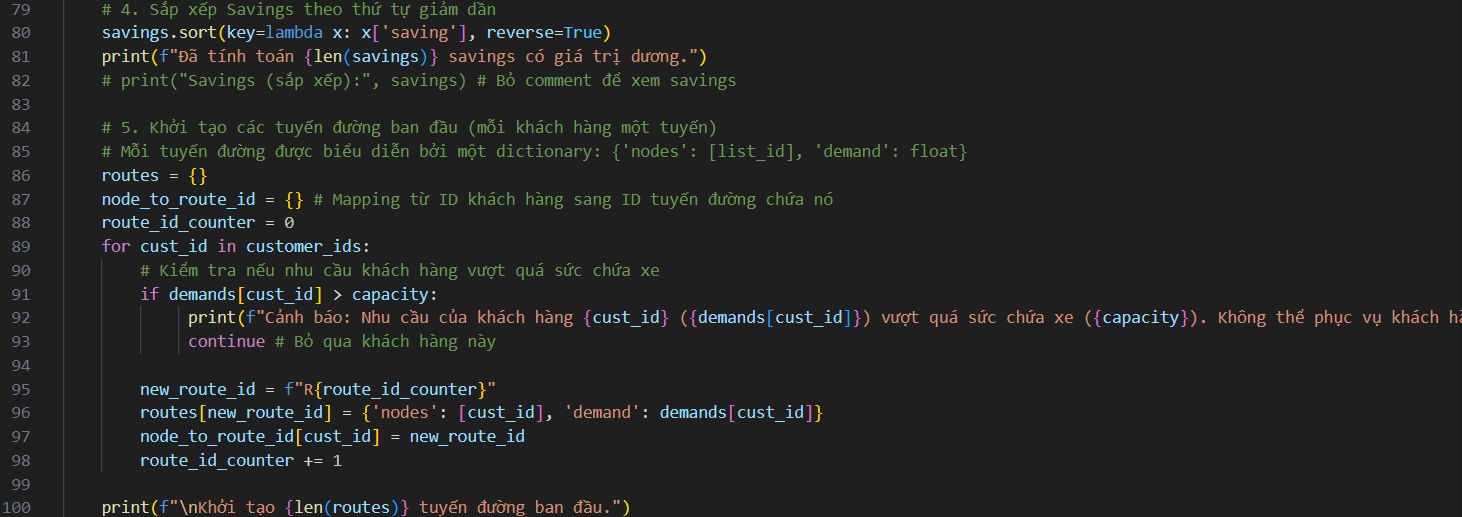
-Đoạn 1 tạo các cấu trúc dữ liệu (locations, demands, customer\_ids, location\_ids) để lưu trữ và truy cập thông tin vị trí, nhu cầu một cách thống nhất và hiệu quả cho cả depot và khách hàng. Và xử lý trường hợp đặc biệt khi không có khách hàng nào cần phục vụ, đảm bảo thuật toán không bị lỗi và trả về kết quả hợp lệ.



Hình 4 Tạo ma trận và tính toán Savings

-Đoạn #2 tạo ra một bảng (ma trận) lưu trữ khoảng cách giữa mọi cặp địa điểm có trong location\_ids.

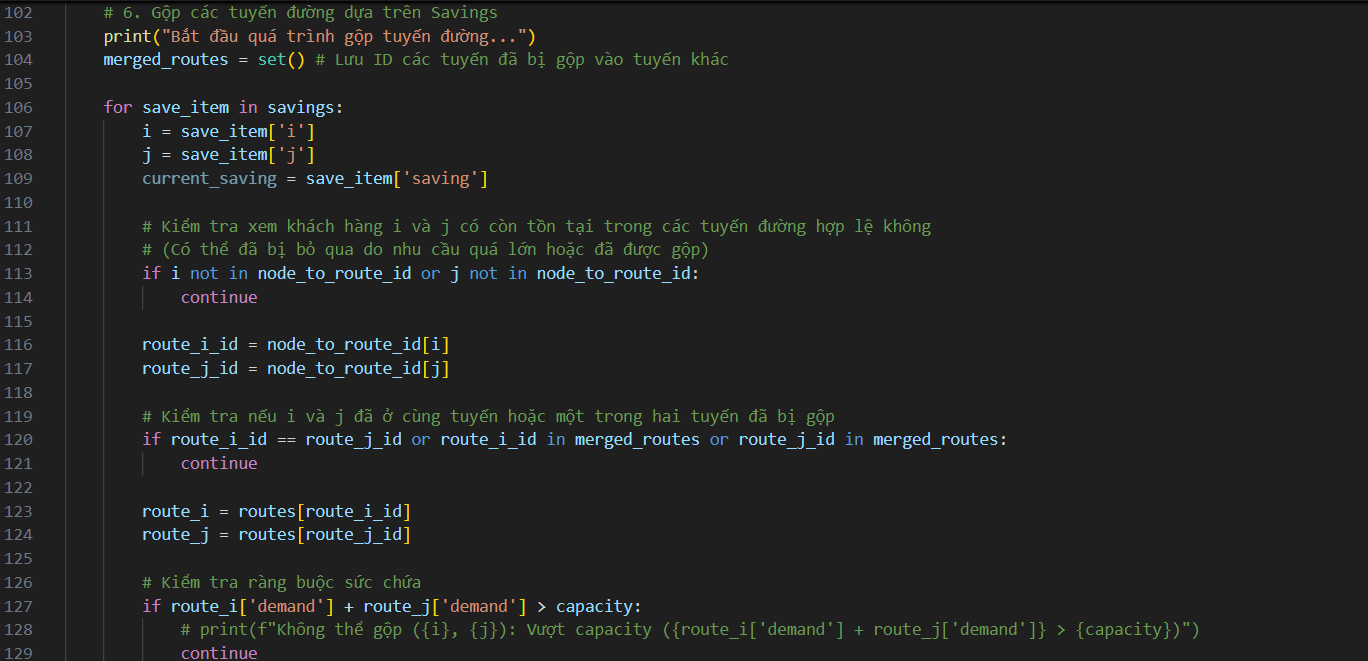
-Đoạn #3 tính toán mức độ "tiết kiệm" chi phí (thường là quãng đường) khi phục vụ hai khách hàng i và j liên tiếp trên cùng một tuyến đường thay vì đi hai tuyến riêng biệt từ kho. Kết quả được lưu lại để sử dụng trong các bước tiếp theo của thuật toán tối ưu hóa lộ trình.



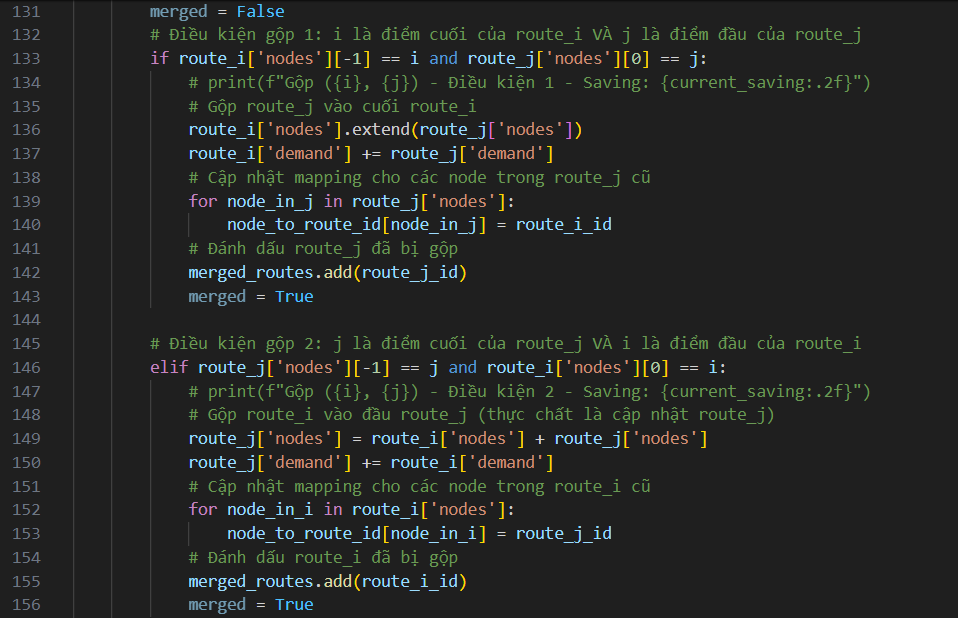
Hình 5 Sắp xếp Savings và khởi tạo các tuyến đường ban đầu

-Đoạn 4 sắp xếp các cặp khách hàng tiềm năng dựa trên mức độ tiết kiệm quãng đường khi ghép chúng lại, theo thứ tự từ tiết kiệm nhiều nhất đến ít nhất.

-Đoạn 5 thiết lập trạng thái ban đầu cho thuật toán Savings: mỗi khách hàng (nếu có thể phục vụ được dựa trên sức chứa) được gán vào một tuyến đường riêng biệt đi từ kho đến khách hàng và quay về.

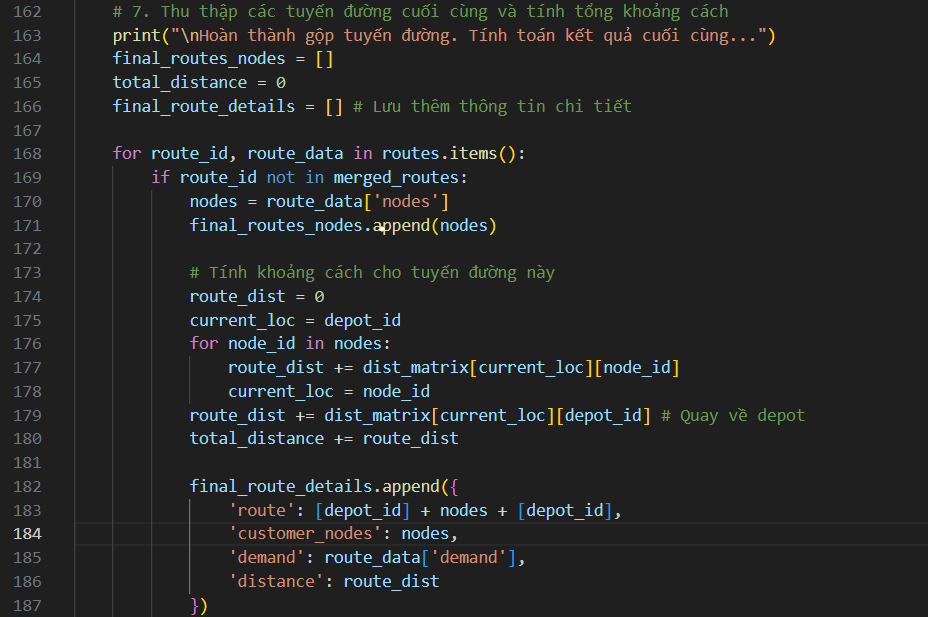


Hình 6 Vòng lặp hợp nhất tuyến đường



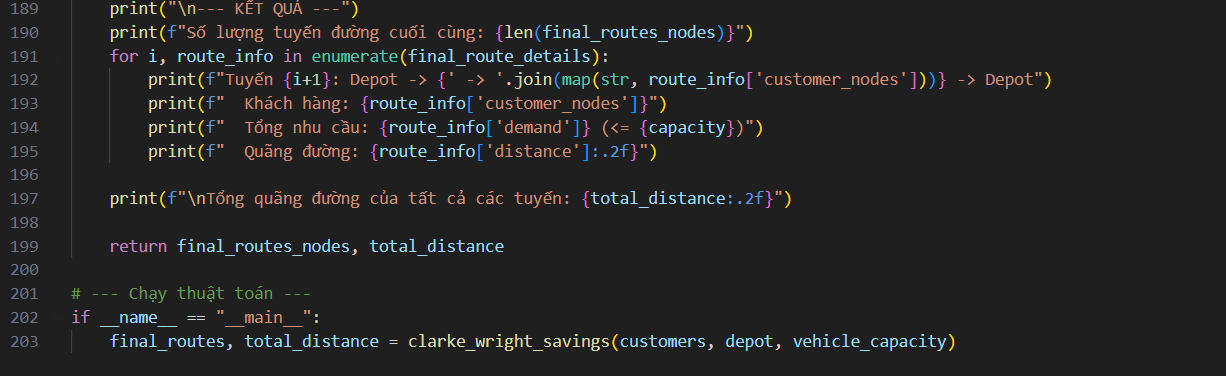
Hình 7 Vòng lặp hợp nhất tuyến đường

- Đoạn 6 này cố gắng gộp hai tuyến đường route\_i và route\_j nếu chúng thỏa mãn một trong hai điều kiện về điểm đầu và điểm cuối (cho phép nối tiếp theo một trong hai hướng). Sau khi gộp, thông tin về các nút, tổng nhu cầu và mapping nút-tuyến đường sẽ được cập nhật, và tuyến đường đã bị gộp sẽ được đánh dấu để tránh gộp lại.



Hình 8 Tập hợp các tuyến đường cuối cùng

-Đoạn 7 thực hiện bước cuối cùng trong quá trình xử lý các tuyến đường: tập hợp các tuyến đường cuối cùng (những tuyến đường chưa bị gộp) và tính toán tổng khoảng cách di chuyển.



Hình 9 In ra kết quả cuối cùng

-Đoạn code này là phần cuối cùng, dùng để in ra kết quả cuối cùng của thuật toán tối ưu hóa tuyến đường và trả về các kết quả này.

### **5. Minh họa bài toán:**

* **Ví dụ: Ta có:**
* **1 kho (Depot)** tại tọa độ (0, 0).
* **3 khách hàng (Customers)** với tọa độ và nhu cầu như sau:
* Khách hàng 1: (2, 0), nhu cầu = 2
* Khách hàng 2: (0, 2), nhu cầu = 3
* Khách hàng 3: (2, 2), nhu cầu = 1
* **1 xe** với sức chứa (Capacity) = 5.

Mục tiêu bài toán này là tìm lộ trình tối ưu cho chiếc xe để phục vụ tất cả khách hàng sao cho tổng quãng đường di chuyển là ngắn nhất và không vượt quá sức chứa của xe.

* **Giải chi tiết:**
* Bước 1: Tính khoảng cách giữa tất cả các cặp địa điểm (khách hàng và kho). Chúng ta sử dụng công thức khoảng cách Euclidean:
* Kho đến khách hàng 1 (dD1) = 2
* Kho đến khách hàng 2 (dD2) = 2
* Kho đến khách hàng 3 (dD3) =
* Khách hàng 1 đến khách hàng 2 (d12)=
* Khách hàng 1 đến khách hàng 3 (d13) = 2
* Khách hàng 2 đến khách hàng 3 (d23) = 2
* Bước 2: Tính Savings cho từng cặp khách hàng theo công thức:

Sij​ = dD i​+ dDj​ − dij​

* S12 ​= dD1 ​+ dD2 ​− d12 ​= 2 + 2 − 2.83 = 1.17
* S13​= dD1 ​+ dD3 ​− d13 ​= 2 + 2.83 – 2 = 2.83
* S23 ​= dD2​ + dD3​ − d23 ​= 2 + 2.83 – 2 = 2.83
* Bước 3: Sắp xếp các giá trị "Savings" theo thứ tự giảm dần.
  + 1. S13​=2.83
    2. S23​=2.83
    3. S12​=1.17
* Bước 4: Lặp lại cho đến khi tất cả khách hàng được phục vụ hoặc không thể gộp thêm.

Bắt đầu với giá trị Savings lớn nhất và kiểm tra xem việc gộp hai tuyến đường có hợp lệ không:

* **Xét cặp 1-3 (S13​=2.83):**
* Tuyến đường hiện tại: D-C1-D và D-C3-D
* Tổng nhu cầu của C1 và C3: 2 + 1 = 3 (≤ 5, không vượt quá sức chứa)
* Hai khách hàng C1 và C3 chưa thuộc cùng một tuyến đường.
* **Gộp tuyến đường:** D-C1-C3-D
* Khách hàng đã phục vụ: {1, 3}
* Xét cặp 2-3 (S23​=2.83):
* Tuyến đường hiện tại: D-C1-C3-D và D-C2-D
* Khách hàng C3 đã thuộc tuyến đường D-C1-C3-D. Chúng ta chỉ có thể gộp nếu C2 có thể được thêm vào một trong hai đầu của tuyến đường hiện tại mà không vi phạm ràng buộc sức chứa.
* Thử thêm C2 vào cuối: D-C1-C3-C2-D (Nhu cầu: 2 + 1 + 3 = 6 > 5, vượt quá sức chứa).
* Thử thêm C2 vào đầu: D-C2-C1-C3-D (Nhu cầu: 3 + 2 + 1 = 6 > 5, vượt quá sức chứa).
* Do đó, không thể gộp C2 vào tuyến đường hiện tại.
* Xét cặp 1-2 (S12​=1.17):
* Tuyến đường hiện tại: D-C1-C3-D và D-C2-D
* Hai khách hàng C1 và C2 chưa thuộc cùng một tuyến đường.
* Thử gộp: D-C1-C2-D (Nhu cầu: 2 + 3 = 5 ≤ 5)
* Tuyến đường hiện tại sau bước trước là D-C1-C3-D. Chúng ta cần xem xét việc kết hợp C2 với tuyến đường này.
* Thử thêm C2 vào cuối: D-C1-C3-C2-D (Nhu cầu = 6 > 5).
* Thử thêm C2 vào đầu: D-C2-C1-C3-D (Nhu cầu = 6 > 5).
* Do đó, không thể gộp C1 và C2 trực tiếp vào cùng một tuyến đường trong bối cảnh tuyến đường D-C1-C3-D đã hình thành.
* Kết quả:Nhận thấy rằng với sức chứa là 5, chúng ta không thể phục vụ tất cả khách hàng bằng một xe duy nhất. Vì vậy chúng ta sẽ chia làm 2 tuyến đường với 2 xe khác nhau. Tuyến đầu tiên sẽ là ( Kho 🡪 khách 1 🡪 khách 3 🡪 Kho). Tuyến 2 sẽ là (Kho 🡪 Khách 2 🡪 Kho).

### **6. Phân tích hiệu suất và tham số.**

* **Thời gian chạy:**
* **Tính toán ma trận khoảng cách:** Nếu có n khách hàng và 1 kho, có tổng cộng n+1 địa điểm. Việc tính toán khoảng cách giữa tất cả các cặp địa điểm mất O(n2) thời gian.
* **Tính toán ma trận "savings":** Tương tự, việc tính toán giá trị "savings" cho tất cả các cặp khách hàng cũng mất O(n2) thời gian.
* **Sắp xếp các giá trị "savings":** Việc sắp xếp O(n2) giá trị "savings" thường mất O(n2logn) thời gian sử dụng các thuật toán sắp xếp hiệu quả (ví dụ: mergesort, quicksort).
* **Gộp các tuyến đường:** Quá trình lặp để gộp các tuyến đường có thể khác nhau tùy thuộc vào cách triển khai. Trong trường hợp xấu nhất, có thể có O(n2) lần thử gộp, và mỗi lần kiểm tra tính hợp lệ (ví dụ: kiểm tra sức chứa, đảm bảo không tạo thành các chu trình con không kết nối với kho) có thể mất O(n) thời gian (để duyệt qua các khách hàng trên tuyến đường). Tuy nhiên, trong thực tế, số lần gộp thường ít hơn nhiều. Một cách tiếp cận hiệu quả hơn có thể sử dụng các cấu trúc dữ liệu để theo dõi các tuyến đường hiện tại và việc kiểm tra tính hợp lệ có thể được thực hiện nhanh hơn.

**Ước tính độ phức tạp tổng thể:** Dựa trên các bước trên, độ phức tạp thời gian của giải thuật Clarke and Wright Savings thường dao động trong khoảng O(n2logn) đến O(n3) tùy thuộc vào cách triển khai chi tiết của giai đoạn gộp và kiểm tra ràng buộc. Nhìn chung, đây là một độ phức tạp chấp nhận được cho số lượng khách hàng vừa phải (vài trăm).

* **Độ chính xác so với lời giải tối ưu:**
* **Tính chất Heuristic:** Điều quan trọng cần nhấn mạnh là Clarke and Wright Savings là một thuật toán heuristic. Điều này có nghĩa là nó không đảm bảo tìm được **lời giải tối ưu toàn cục** cho bài toán CVRP.
* **Sai lệch so với tối ưu:** Độ lệch giữa giải pháp tìm được bởi Clarke and Wright Savings và lời giải tối ưu có thể khác nhau tùy thuộc vào đặc điểm cụ thể của từng bài toán (ví dụ: phân bố khách hàng, sức chứa xe). Trong nhiều trường hợp thực tế, thuật toán có thể cho ra các giải pháp **gần tối ưu** hoặc **chấp nhận được**, đặc biệt khi được kết hợp với các kỹ thuật cải thiện hậu kỳ (post-optimization techniques) như 2-opt, 3-opt.
* **Khó đánh giá độ chính xác tuyệt đối:** Để đánh giá độ chính xác tuyệt đối, chúng ta cần biết lời giải tối ưu. Việc tìm lời giải tối ưu cho các bài toán CVRP lớn là một vấn đề NP-khó, và thường chỉ có thể thực hiện được cho các bài toán có kích thước rất nhỏ bằng các thuật toán chính xác (ví dụ: branch and cut).
* **So sánh với các thuật toán khác:** So với các thuật toán heuristic khác cho CVRP (ví dụ: thuật toán kiến, thuật toán di truyền, thuật toán tabu search), Clarke and Wright Savings thường đơn giản hơn và có thời gian chạy nhanh hơn, nhưng có thể không đạt được độ chính xác cao bằng các thuật toán phức tạp hơn này trên các bài toán lớn và khó.
* **Tham số ảnh hưởng đến kết quả:**
* **Tọa độ khách hàng và kho:** Vị trí địa lý quyết định ma trận khoảng cách và giá trị "savings", từ đó ảnh hưởng trực tiếp đến quyết định gộp tuyến đường. Các cụm khách hàng gần nhau có xu hướng được gộp lại, giảm quãng đường di chuyển.
* **Nhu cầu của khách hàng:** Nhu cầu của từng khách hàng giới hạn số lượng khách hàng có thể được phục vụ trên một tuyến đường duy nhất do ràng buộc về sức chứa. Sự phân bố nhu cầu (đồng đều hay không) sẽ ảnh hưởng đến cách các tuyến đường được hình thành.
* **Sức chứa của xe:** Sức chứa là một ràng buộc cứng. Xe có sức chứa lớn hơn có khả năng phục vụ nhiều khách hàng hơn trên một tuyến đường, có thể dẫn đến ít tuyến đường hơn và tổng quãng đường ngắn hơn (nếu việc gộp là hợp lý về mặt địa lý).
* **Hàm tính khoảng cách:** Việc sử dụng các hàm tính khoảng cách khác nhau (ví dụ: Euclidean vs. Manhattan) sẽ tạo ra các ma trận khoảng cách và "savings" khác nhau, do đó có thể dẫn đến các giải pháp khác nhau.
* **Quy tắc phá vỡ hòa (Tie-breaking rule):** Khi có nhiều cặp khách hàng có cùng giá trị "savings", quy tắc chọn cặp nào để xem xét trước có thể ảnh hưởng đến kết quả cuối cùng. Một số quy tắc có thể ưu tiên các cặp có khoảng cách giữa các khách hàng nhỏ hơn hoặc các khách hàng gần kho hơn.
* **Thứ tự ban đầu của các cặp "savings":** Mặc dù chúng ta sắp xếp theo thứ tự giảm dần, thứ tự của các cặp có cùng giá trị "savings" có thể ảnh hưởng đến quá trình gộp.
* **Các ràng buộc phụ (nếu có):** Nếu bài toán có thêm các ràng buộc như cửa sổ thời gian, việc xem xét và tuân thủ các ràng buộc này trong quá trình gộp sẽ ảnh hưởng đến tính khả thi và chất lượng của giải pháp.

### **7. Ưu Và Nhược Điểm**

* **Ưu điểm:**
* **Tính đơn giản và dễ cài đặt:** Thuật toán Clarke and Wright Savings tương đối dễ hiểu và triển khai so với các thuật toán tối ưu hóa phức tạp hơn cho CVRP.
* **Hiệu quả tính toán cho số lượng khách hàng vừa phải:** Với số lượng khách hàng không quá lớn (thường dưới vài trăm), thuật toán có thể cho kết quả tốt trong thời gian chấp nhận được. Độ phức tạp chính nằm ở việc tính toán ma trận khoảng cách và ma trận "savings" (O(n2)), và sau đó là quá trình lặp để gộp các tuyến đường (tùy thuộc vào cách triển khai, có thể dao động nhưng thường không quá cao).
* **Tìm được giải pháp chấp nhận được (heuristic):** Thuật toán thường cho ra các giải pháp tốt, giảm đáng kể tổng quãng đường di chuyển và số lượng tuyến đường so với việc phục vụ từng khách hàng riêng lẻ.
* **Linh hoạt trong việc kết hợp các ràng buộc phụ:** Mặc dù cơ bản, thuật toán có thể được mở rộng để xem xét các ràng buộc phụ như cửa sổ thời gian (time windows) hoặc các loại xe khác nhau, nhưng việc này có thể làm tăng độ phức tạp.
* **Nhược điểm:**
* **Không đảm bảo tối ưu toàn cục:** Vì là một thuật toán heuristic dựa trên việc gộp cục bộ, Clarke and Wright Savings không đảm bảo tìm được giải pháp tối ưu tuyệt đối cho bài toán CVRP. Các quyết định gộp ban đầu dựa trên "savings" lớn nhất có thể bỏ lỡ các kết hợp khác dẫn đến giải pháp tốt hơn về tổng thể.
* **Phụ thuộc vào thứ tự xét các cặp:** Thứ tự xử lý các cặp khách hàng có cùng giá trị "savings" có thể ảnh hưởng đến kết quả cuối cùng. Việc không có một quy tắc phá vỡ hòa (tie-breaking rule) nhất quán có thể dẫn đến các giải pháp khác nhau.
* **Khó khăn với cấu trúc dữ liệu phức tạp:** Hiệu suất có thể giảm nếu việc quản lý và kiểm tra các ràng buộc (đặc biệt là sức chứa và các ràng buộc phụ) không được thực hiện hiệu quả.
* **Có thể bị "mắc kẹt" trong các tối ưu cục bộ:** Thuật toán có thể đạt đến một trạng thái mà không thể gộp thêm tuyến đường nào mà không vi phạm ràng buộc, nhưng giải pháp này có thể không phải là tốt nhất toàn cục.

### **8. Ứng Dụng**

Trong thực tế đời sống, việc áp dụng thuật toán Clarke and Wright Savings Algorithm (hoặc các biến thể và thuật toán phức tạp hơn dựa trên nền tảng này) mang lại nhiều lợi ích thiết thực:

* **Giảm chi phí vận chuyển:** Tiết kiệm nhiên liệu, chi phí bảo trì xe và nhân công lái xe do quãng đường di chuyển ngắn hơn và số lượng xe cần thiết có thể ít hơn.
* **Tăng hiệu quả hoạt động:** Phục vụ được nhiều khách hàng hơn trong cùng một khoảng thời gian, tăng năng suất của đội xe và nhân viên.
* **Cải thiện chất lượng dịch vụ:** Đảm bảo giao hàng đúng hẹn hơn do tối ưu hóa lộ trình và giảm thiểu thời gian di chuyển không cần thiết.
* **Giảm tác động môi trường:** Lượng khí thải từ xe giảm do tổng quãng đường di chuyển ít hơn, góp phần vào sự phát triển bền vững.

### **9. Tóm Tắt & Kết Luận**

* **Tóm Lược:** Thuật toán Clarke and Wright Savings Algorithm là một phương pháp heuristic cổ điển và hiệu quả để giải bài toán Định tuyến Xe có Ràng buộc về Tải trọng (CVRP). Ý tưởng cốt lõi của thuật toán là bắt đầu với các tuyến đường riêng lẻ cho từng khách hàng, sau đó iteratively (lặp đi lặp lại) hợp nhất các tuyến đường dựa trên "mức tiết kiệm" quãng đường đạt được khi kết hợp hai khách hàng vào cùng một tuyến. Quá trình hợp nhất này được thực hiện theo thứ tự giảm dần của mức tiết kiệm, đồng thời đảm bảo không vượt quá giới hạn tải trọng của xe. Mục tiêu chính là tối đa hóa số lượng khách hàng mà một xe có thể phục vụ trong một hành trình đồng thời giảm thiểu tổng quãng đường di chuyển của đội xe.
* **Ý Nghĩa:** Thuật toán Clarke and Wright Savings Algorithm nổi bật như một heuristic nhanh chóng và trực quan để tìm ra các giải pháp ban đầu tốt cho bài toán CVRP. Điểm mạnh chính của nó nằm ở sự đơn giản trong logic và hiệu quả tính toán, đặc biệt hữu ích khi cần có một giải pháp chấp nhận được trong thời gian ngắn cho các bài toán có quy mô vừa và nhỏ. Tuy nhiên, do bản chất là một heuristic greedy (tham lam), thuật toán này không đảm bảo tìm được nghiệm tối ưu toàn cục và có thể bị mắc kẹt trong các cực tiểu cục bộ, đặc biệt đối với các bài toán có không gian tìm kiếm phức tạp hoặc các ràng buộc chặt chẽ.

Trong bối cảnh các kỹ thuật tối ưu hóa, Clarke and Wright Savings Algorithm thường được xem là một phương pháp xây dựng nghiệm (construction heuristic). Nó cung cấp một nền tảng tốt để so sánh với các thuật toán phức tạp hơn như metaheuristics (ví dụ: Simulated Annealing, Genetic Algorithm, Tabu Search), vốn có khả năng khám phá không gian tìm kiếm rộng hơn và có thể tìm ra các giải pháp tốt hơn, đặc biệt cho các bài toán CVRP lớn và khó. Mặc dù vậy, sự đơn giản và tốc độ của Clarke and Wright vẫn khiến nó trở thành một lựa chọn hữu ích cho việc khởi tạo giải pháp ban đầu hoặc cho các ứng dụng thực tế không đòi hỏi nghiệm tối ưu tuyệt đối.

## **C)THUẬT TOÁN TABU SEARCH**

### **1. Giới Thiệu Về Thuật Toán**

#### **1.1 Đặt vấn đề**

Trong lĩnh vực vận tải và logistics, việc tối ưu hóa quá trình phân phối hàng hóa từ một kho trung tâm đến nhiều điểm giao hàng là bài toán cốt lõi. Lập kế hoạch vận chuyển hiệu quả giúp giảm chi phí, tiết kiệm thời gian, nhiên liệu và nâng cao sự hài lòng của khách hàng. Bài toán "định tuyến phương tiện có ràng buộc về tải trọng" (Capacitated Vehicle Routing Problem - CVRP) là mô hình điển hình, yêu cầu xác định các tuyến đường cho đội xe có sức chứa giới hạn để phục vụ khách hàng với tổng quãng đường ngắn nhất. Tuy nhiên, CVRP là bài toán NP-hard, việc tìm lời giải tối ưu tuyệt đối bằng cách thử mọi phương án (vét cạn) là không khả thi khi số lượng khách hàng lớn. Do đó, các thuật toán heuristic, như Tìm kiếm Tabu, được sử dụng để tìm lời giải đủ tốt trong thời gian hợp lý.

#### **1.2 Giới thiệu về thuật toán Tìm Kiếm Tabu (Tabu Search)**

Tìm kiếm Tabu (Tabu Search - TS) là một thuật toán metaheuristic được đề xuất bởi Fred Glover vào những năm 1980 để giải quyết các bài toán tối ưu hóa tổ hợp. Khác với các thuật toán tìm kiếm cục bộ đơn giản thường bị mắc kẹt tại các điểm cực tiểu địa phương, TS sử dụng một "bộ nhớ" (danh sách Tabu) để ghi lại các giải pháp hoặc các thành phần của giải pháp đã được khám phá gần đây. Điều này giúp thuật toán tránh quay lại các trạng thái đã xét và khuyến khích khám phá các vùng mới trong không gian lời giải, kể cả khi phải tạm thời chấp nhận một lời giải kém hơn.

#### **1.3 Ý Tưởng thuật toán**

Ý tưởng cốt lõi của Tìm kiếm Tabu là mô phỏng trí nhớ của con người trong quá trình giải quyết vấn đề: tránh lặp lại những hành động đã thực hiện gần đây và sử dụng thông tin quá khứ để đưa ra quyết định tốt hơn trong tương lai.

* **Tìm kiếm dựa trên lân cận:** Thuật toán bắt đầu từ một lời giải ban đầu và lặp đi lặp lại việc di chuyển đến một lời giải "lân cận". Lời giải lân cận được tạo ra bằng cách áp dụng một "phép biến đổi" (move) nhỏ lên lời giải hiện tại.
* **Bộ nhớ ngắn hạn (Danh sách Tabu):** TS duy trì một danh sách các thuộc tính của các phép biến đổi hoặc các lời giải đã thực hiện gần đây. Các phép biến đổi tạo ra trạng thái có thuộc tính trong danh sách này bị "cấm" (tabu) trong một số vòng lặp nhất định (gọi là nhiệm kỳ tabu - tabu tenure).
* **Cơ chế chấp nhận nước đi:** Ở mỗi bước, TS thường chọn nước đi tốt nhất trong số các nước đi lân cận *không bị cấm*. Điều quan trọng là TS có thể chấp nhận cả những nước đi làm xấu đi hàm mục tiêu (tăng tổng khoảng cách) nếu chúng không bị cấm, giúp nó thoát khỏi các điểm tối ưu cục bộ.
* **Bộ nhớ dài hạn (Tùy chọn):** Các phiên bản nâng cao của TS có thể sử dụng bộ nhớ dài hạn để ghi nhận tần suất hoặc chất lượng của các thành phần giải pháp, nhằm tăng cường (intensify) tìm kiếm ở các vùng hứa hẹn hoặc đa dạng hóa (diversify) tìm kiếm sang các vùng chưa được khám phá kỹ.
* **Tiêu chuẩn Ngưỡng vọng (Aspiration Criteria):** Cho phép một nước đi bị cấm được thực hiện nếu nó dẫn đến một kết quả đủ tốt (ví dụ, tốt hơn lời giải tốt nhất từng tìm thấy).

Trong bối cảnh code được cung cấp, TS được áp dụng bằng cách khám phá không gian các *hoán vị* của khách hàng, sử dụng phép hoán vị (swap) làm phép biến đổi lân cận và danh sách tabu lưu trữ các *hoán vị đầy đủ* đã duyệt gần đây.

### **2. Các Bước Triển Khai Chi Tiết của thuật toán (theo code cung cấp)**

Thuật toán Tabu Search được triển khai trong code Python hoạt động qua các bước chính sau:

* **Bước 1: Khởi tạo**
  + Tạo một lời giải ban đầu (initial\_solution): Là một hoán vị ngẫu nhiên của tất cả các ID khách hàng (từ 1 đến n).
  + Tính chi phí ban đầu: Sử dụng hàm split\_routes để chia hoán vị khách hàng thành các tuyến đường dựa trên sức chứa xe (100). Sau đó, tính tổng khoảng cách của tất cả các tuyến này bằng hàm total\_distance.
  + Lưu lời giải ban đầu này là best\_solution và chi phí ban đầu là best\_distance.
  + Đặt current\_solution bằng best\_solution.
  + Khởi tạo tabu\_list là một danh sách rỗng.
* **Bước 2: Vòng lặp chính (Thực hiện iterations lần)**
  + **Tạo lân cận:** Từ current\_solution, tạo ra tất cả các lời giải lân cận có thể bằng cách hoán vị (swap) hai khách hàng bất kỳ trong chuỗi hoán vị (generate\_neighbors).
  + **Lọc Tabu:** Loại bỏ khỏi danh sách lân cận những lời giải (hoán vị) nào đang có mặt trong tabu\_list. *Lưu ý: Cách triển khai này lưu trữ toàn bộ hoán vị trong tabu list.*
  + **Kiểm tra lân cận hợp lệ:** Nếu không còn lân cận nào sau khi lọc tabu, bỏ qua vòng lặp hiện tại (continue).
  + **Đánh giá lân cận:** Với mỗi lân cận hợp lệ còn lại:
    - Sử dụng split\_routes để tạo các tuyến đường thực tế từ hoán vị lân cận.
    - Tính tổng khoảng cách của các tuyến đường này (sum([total\_distance(route) for route in split\_routes(x)])).
  + **Chọn lân cận tốt nhất:** Sắp xếp các lân cận hợp lệ theo tổng khoảng cách tăng dần và chọn lân cận có khoảng cách nhỏ nhất (best\_neighbor).
  + **Cập nhật lời giải tốt nhất (Best Solution):** So sánh khoảng cách của best\_neighbor (best\_neighbor\_distance) với best\_distance. Nếu best\_neighbor\_distance nhỏ hơn, cập nhật best\_solution thành best\_neighbor và best\_distance thành best\_neighbor\_distance. *Lưu ý: Không có tiêu chuẩn ngưỡng vọng, nên nếu nước đi tốt nhất toàn cục bị tabu, nó sẽ không được chọn lại cho đến khi ra khỏi danh sách tabu.*
  + **Cập nhật lời giải hiện tại:** Cập nhật current\_solution thành best\_neighbor (luôn di chuyển đến lân cận không tabu tốt nhất).
  + **Cập nhật danh sách Tabu:** Thêm current\_solution (là best\_neighbor vừa chọn) vào cuối tabu\_list. Nếu kích thước tabu\_list vượt quá tabu\_size, xóa phần tử đầu tiên (cũ nhất) ra khỏi danh sách.
* **Bước 3: Trả về kết quả**
  + Sau khi hoàn thành số vòng lặp (iterations), trả về best\_solution (hoán vị tốt nhất tìm được) và best\_distance (tổng khoảng cách tương ứng). Hàm split\_routes sau đó được gọi bên ngoài để hiển thị các tuyến đường cụ thể từ best\_solution.

### **3. Pseudocode (Mô tả logic code)**

FUNCTION TabuSearch(locations, demands, capacity, iterations, tabu\_size):  
 // Khởi tạo  
 current\_solution = GenerateRandomPermutation(customers)  
 best\_solution = current\_solution  
 routes = SplitSolutionIntoRoutes(current\_solution, capacity, demands)  
 best\_distance = CalculateTotalDistance(routes, locations)  
 tabu\_list = EmptyList() // Danh sách lưu các hoán vị bị cấm  
  
 FOR i FROM 1 TO iterations:  
 // Tạo và đánh giá lân cận  
 all\_neighbors = GenerateSwapNeighbors(current\_solution)  
 valid\_neighbors = []  
 neighbor\_evaluations = {} // Lưu {neighbor\_permutation: total\_distance}  
  
 FOR neighbor IN all\_neighbors:  
 IF neighbor IS NOT IN tabu\_list:  
 Add neighbor TO valid\_neighbors  
 neighbor\_routes = SplitSolutionIntoRoutes(neighbor, capacity, demands)  
 neighbor\_distance = CalculateTotalDistance(neighbor\_routes, locations)  
 neighbor\_evaluations[neighbor] = neighbor\_distance  
 END IF  
 END FOR  
  
 // Nếu không có lân cận hợp lệ  
 IF valid\_neighbors IS EMPTY:  
 CONTINUE // Bỏ qua vòng lặp này  
 END IF  
  
 // Chọn lân cận tốt nhất không bị tabu  
 best\_neighbor\_in\_iteration = FindNeighborWithMinDistance(valid\_neighbors, neighbor\_evaluations)  
 best\_neighbor\_distance\_in\_iteration = neighbor\_evaluations[best\_neighbor\_in\_iteration]  
  
 // Cập nhật lời giải tốt nhất toàn cục  
 IF best\_neighbor\_distance\_in\_iteration < best\_distance:  
 best\_solution = best\_neighbor\_in\_iteration  
 best\_distance = best\_neighbor\_distance\_in\_iteration  
 END IF  
  
 // Di chuyển đến lân cận tốt nhất  
 current\_solution = best\_neighbor\_in\_iteration  
  
 // Cập nhật danh sách Tabu  
 Add current\_solution TO tabu\_list  
 IF Length(tabu\_list) > tabu\_size:  
 RemoveOldestElement(tabu\_list)  
 END IF  
 END FOR  
  
 RETURN best\_solution, best\_distance  
END FUNCTION  
  
// Các hàm phụ trợ:  
FUNCTION GenerateRandomPermutation(customers)  
FUNCTION SplitSolutionIntoRoutes(permutation, capacity, demands) // Trả về list các routes [[0,...0], [0,...0]]  
FUNCTION CalculateTotalDistance(routes, locations) // Tính tổng khoảng cách các routes  
FUNCTION GenerateSwapNeighbors(permutation) // Tạo các hoán vị bằng cách swap 2 phần tử  
FUNCTION FindNeighborWithMinDistance(neighbors, evaluations)

### **4. Code python (Giải thích code cung cấp)**

import math  
import random  
import copy  
  
# Khai báo input  
locations = [  
 (0, 0), (10, 5), (12, 8), (5, 10), (15, 12), (7, 4), (9, 15), (4, 6), (8, 13), (14, 5),  
 (11, 10), (3, 8), (6, 14), (13, 9), (2, 5), (9, 6), (7, 12), (10, 2), (5, 3), (6, 7), (13, 7)  
]  
demands = [  
 0, 10, 15, 20, 25, 30, 10, 15, 20, 30,  
 10, 15, 10, 25, 20, 15, 20, 10, 15, 20, 20  
]  
  
# Tính khoảng cách Euclid  
def euclidean(p1, p2):  
 return round(math.hypot(p1[0] - p2[0], p1[1] - p2[1]), 2)  
  
def total\_distance(route):  
 dist = 0  
 for i in range(len(route) - 1):  
 dist += euclidean(locations[route[i]], locations[route[i+1]])  
 return round(dist, 2)  
  
def split\_routes(solution):  
 routes, current\_route, load = [], [0], 0  
 for customer in solution:  
 if load + demands[customer] > 100:  
 current\_route.append(0)  
 routes.append(current\_route)  
 current\_route, load = [0, customer], demands[customer]  
 else:  
 current\_route.append(customer)  
 load += demands[customer]  
 current\_route.append(0)  
 routes.append(current\_route)  
 return routes  
  
def initial\_solution():  
 customers = list(range(1, len(locations)))  
 random.shuffle(customers)  
 return customers  
  
def generate\_neighbors(solution):  
 neighbors = []  
 for i in range(len(solution)):  
 for j in range(i+1, len(solution)):  
 neighbor = solution[:]  
 neighbor[i], neighbor[j] = neighbor[j], neighbor[i]  
 neighbors.append(neighbor)  
 return neighbors  
  
def tabu\_search(iterations=500, tabu\_size=30):  
 best\_solution = initial\_solution()  
 best\_distance = sum([total\_distance(route) for route in split\_routes(best\_solution)])  
 current\_solution = best\_solution[:]  
 tabu\_list = []  
  
 for \_ in range(iterations):  
 neighbors = generate\_neighbors(current\_solution)  
 neighbors = [n for n in neighbors if n not in tabu\_list]  
  
 if not neighbors:  
 continue  
  
 neighbors.sort(key=lambda x: sum([total\_distance(route) for route in split\_routes(x)]))  
 best\_neighbor = neighbors[0]  
 best\_neighbor\_distance = sum([total\_distance(route) for route in split\_routes(best\_neighbor)])  
  
 if best\_neighbor\_distance < best\_distance:  
 best\_solution = best\_neighbor[:]  
 best\_distance = best\_neighbor\_distance  
  
 current\_solution = best\_neighbor[:]  
 tabu\_list.append(current\_solution)  
 if len(tabu\_list) > tabu\_size:  
 tabu\_list.pop(0)  
  
 return best\_solution, best\_distance  
  
# Chạy Tabu Search  
best\_sol, best\_dist = tabu\_search()  
best\_routes = split\_routes(best\_sol)  
  
print("Tổng quãng đường tối ưu:", best\_dist)  
print("\nCác tuyến đường tối ưu:")  
for i, route in enumerate(best\_routes):  
 print(f"\nTuyến {i+1}:")  
 for point in route:  
 print(f"{point}")

 **Khai báo input:**

* locations: Danh sách các tuple (x, y) chứa tọa độ của depot (index 0) và các khách hàng.
* demands: Danh sách nhu cầu tương ứng với locations.
* VEHICLE\_CAPACITY (mặc định trong split\_routes là 100): Sức chứa xe.

 **Hàm tính toán cơ bản:**

* euclidean(p1, p2): Tính khoảng cách Euclid giữa hai điểm (tuple).
* total\_distance(route): Tính tổng khoảng cách của một tuyến đơn lẻ (đã bao gồm depot ở đầu và cuối).

 **Hàm xử lý lời giải:**

* split\_routes(solution): Chuyển đổi một hoán vị khách hàng (solution) thành danh sách các tuyến đường (routes) dựa trên VEHICLE\_CAPACITY. Hàm này tự động thêm depot (0) vào đầu/cuối mỗi tuyến con.
* initial\_solution(): Tạo một hoán vị ngẫu nhiên ban đầu của các ID khách hàng.
* generate\_neighbors(solution): Tạo danh sách các hoán vị lân cận bằng cách áp dụng phép toán hoán vị (swap) cho tất cả các cặp khách hàng trong solution.

 **Hàm chính tabu\_search(iterations, tabu\_size):**

* Thực hiện logic Tabu Search như mô tả ở Mục 2 và 3.
* Khởi tạo lời giải và tính chi phí ban đầu.
* Lặp qua số iterations xác định.
* Trong mỗi vòng lặp: tạo lân cận, lọc tabu (dựa trên hoán vị đầy đủ), đánh giá lân cận (tính tổng khoảng cách sau khi split\_routes), chọn lân cận không tabu tốt nhất, cập nhật lời giải hiện tại và lời giải tốt nhất, cập nhật danh sách tabu (FIFO).

 **Chạy thuật toán:**

* Gọi tabu\_search() để tìm hoán vị tốt nhất (best\_sol) và tổng khoảng cách (best\_dist).
* Gọi split\_routes(best\_sol) để lấy danh sách các tuyến đường cụ thể từ hoán vị tốt nhất.
* In kết quả.

### **5. Ví dụ minh họa thuật toán (Step-by-step với tối ưu hóa)**

Để giúp người đọc dễ hình dung quá trình tối ưu của thuật toán Tabu Search thông qua phép toán Swap, chúng ta sẽ xem xét một ví dụ đơn giản với:

* **Depot (0):** Tọa độ (0, 0)
* **Sức chứa xe (Capacity):** 100 (Theo code)
* **Khách hàng (Customers):**
  + ID 5: (7, 4), Demand = 30
  + ID 14: (2, 5), Demand = 20
  + ID 18: (5, 3), Demand = 15
* **Tham số Tabu Search:** iterations (đủ lớn), tabu\_size = 2 (giả định)

**Tính toán trước khoảng cách Euclid giữa các điểm:**

* d(0, 5) = eu((0,0), (7,4)) = sqrt(7² + 4²) ≈ 8.06
* d(0, 14) = eu((0,0), (2,5)) = sqrt(2² + 5²) ≈ 5.39
* d(0, 18) = eu((0,0), (5,3)) = sqrt(5² + 3²) ≈ 5.83
* d(5, 14) = eu((7,4), (2,5)) = sqrt((7-2)² + (4-5)²) = sqrt(5² + (-1)²) ≈ 5.10
* d(5, 18) = eu((7,4), (5,3)) = sqrt((7-5)² + (4-3)²) = sqrt(2² + 1²) ≈ 2.24
* d(14, 18) = eu((2,5), (5,3)) = sqrt((2-5)² + (5-3)²) = sqrt((-3)² + 2²) ≈ 3.61

**Iteration 0: Khởi tạo**

1. **initial\_solution():** Giả sử hàm trả về hoán vị ngẫu nhiên current\_solution = [5, 18, 14].
2. **split\_routes([5, 18, 14]):**
   * Tổng demand = 30 (KH 5) + 15 (KH 18) + 20 (KH 14) = 65.
   * Vì 65 <= 100 (Capacity), tất cả khách hàng nằm trên một tuyến.
   * routes = [[0, 5, 18, 14, 0]]
3. **Tính best\_distance (chi phí ban đầu):**
   * total\_distance([0, 5, 18, 14, 0]) = d(0,5) + d(5,18) + d(18,14) + d(14,0)
   * = 8.06 + 2.24 + 3.61 + 5.39 = **19.30**
4. **Cập nhật:**
   * best\_solution = [5, 18, 14]
   * best\_distance = 19.30
   * current\_solution = [5, 18, 14]
   * tabu\_list = []

**Iteration 1:**

1. **current\_solution = [5, 18, 14]**
2. **generate\_neighbors([5, 18, 14]):**
   * Swap(0,1): n1 = [18, 5, 14]
   * Swap(0,2): n2 = [14, 18, 5]
   * Swap(1,2): n3 = [5, 14, 18]
3. **Lọc Tabu:** tabu\_list rỗng. Tất cả n1, n2, n3 đều hợp lệ.
4. **Đánh giá lân cận (Tất cả đều tạo 1 tuyến duy nhất do tổng demand <= 100):**
   * **Đánh giá n1 = [18, 5, 14]:**
     + Tuyến: [0, 18, 5, 14, 0]
     + Khoảng cách = d(0,18) + d(18,5) + d(5,14) + d(14,0)
     + = 5.83 + 2.24 + 5.10 + 5.39 = **18.56**
   * **Đánh giá n2 = [14, 18, 5]:**
     + Tuyến: [0, 14, 18, 5, 0]
     + Khoảng cách = d(0,14) + d(14,18) + d(18,5) + d(5,0)
     + = 5.39 + 3.61 + 2.24 + 8.06 = **19.30**
   * **Đánh giá n3 = [5, 14, 18]:**
     + Tuyến: [0, 5, 14, 18, 0]
     + Khoảng cách = d(0,5) + d(5,14) + d(14,18) + d(18,0)
     + = 8.06 + 5.10 + 3.61 + 5.83 = **22.60**
5. **Chọn lân cận tốt nhất:** So sánh các khoảng cách (18.56, 19.30, 22.60). Lân cận n1 = [18, 5, 14] có khoảng cách nhỏ nhất (18.56).
   * best\_neighbor = [18, 5, 14]
   * best\_neighbor\_distance = 18.56
6. **Cập nhật best\_solution:**
   * So sánh best\_neighbor\_distance (18.56) với best\_distance (19.30).
   * Vì 18.56 < 19.30, đã tìm thấy lời giải tốt hơn!
   * Cập nhật: best\_solution = [18, 5, 14]
   * Cập nhật: best\_distance = 18.56
7. **Cập nhật current\_solution:** Di chuyển đến lân cận tốt nhất vừa chọn.
   * current\_solution = [18, 5, 14]
8. **Cập nhật tabu\_list:** Thêm hoán vị vừa chọn vào danh sách.
   * tabu\_list = [[18, 5, 14]] (Kích thước hiện tại là 1, nhỏ hơn tabu\_size=2)

**Iteration 2:**

1. **current\_solution = [18, 5, 14]**
2. **generate\_neighbors([18, 5, 14]):**
   * Swap(0,1): n1\_new = [5, 18, 14] (Khoảng cách = 19.30)
   * Swap(0,2): n2\_new = [14, 5, 18]
   * Swap(1,2): n3\_new = [18, 14, 5]
3. **Lọc Tabu:** tabu\_list = [[18, 5, 14]]. Kiểm tra các lân cận:
   * n1\_new: Hợp lệ.
   * n2\_new: Hợp lệ.
   * n3\_new: Hợp lệ.
   * (Nếu một lân cận trùng với [18, 5, 14] thì nó sẽ bị loại).
4. **Đánh giá lân cận hợp lệ:**
   * **Đánh giá n1\_new = [5, 18, 14]:** Khoảng cách = **19.30** (đã tính ở Iteration 0).
   * **Đánh giá n2\_new = [14, 5, 18]:**
     + Tuyến: [0, 14, 5, 18, 0]
     + Khoảng cách = d(0,14) + d(14,5) + d(5,18) + d(18,0)
     + = 5.39 + 5.10 + 2.24 + 5.83 = **18.56**
   * **Đánh giá n3\_new = [18, 14, 5]:**
     + Tuyến: [0, 18, 14, 5, 0]
     + Khoảng cách = d(0,18) + d(18,14) + d(14,5) + d(5,0)
     + = 5.83 + 3.61 + 5.10 + 8.06 = **22.60**
5. **Chọn lân cận tốt nhất:** So sánh các khoảng cách hợp lệ (19.30, 18.56, 22.60). Lân cận n2\_new = [14, 5, 18] có khoảng cách nhỏ nhất (18.56).
   * best\_neighbor = [14, 5, 18]
   * best\_neighbor\_distance = 18.56
6. **Cập nhật best\_solution:**
   * So sánh best\_neighbor\_distance (18.56) với best\_distance (18.56).
   * Vì 18.56 không nhỏ hơn 18.56, không cập nhật best\_solution. Nó vẫn là [18, 5, 14].
7. **Cập nhật current\_solution:** Di chuyển đến lân cận tốt nhất vừa chọn.
   * current\_solution = [14, 5, 18]
8. **Cập nhật tabu\_list:** Thêm hoán vị vừa chọn vào danh sách.
   * tabu\_list = [[18, 5, 14], [14, 5, 18]] (Kích thước hiện tại là 2, bằng tabu\_size=2).

**(Quá trình tiếp tục...)**

### **6. Phân tích hiệu suất và tham số**

* **Thời gian chạy (Time Complexity):**
  + Khởi tạo: Tạo hoán vị O(n), split\_routes O(n), total\_distance O(n) -> Tổng O(n).
  + Mỗi vòng lặp Tabu Search:
    - generate\_neighbors (Swap): Tạo O(n^2) lân cận.
    - Lọc Tabu: O(n^2 \* n \* tabu\_size) nếu kiểm tra bằng in trên list, hoặc O(n^2) nếu dùng set cho tabu\_list (nhưng code dùng list). *Đây là điểm yếu về hiệu năng.*
    - Đánh giá mỗi lân cận: split\_routes O(n), total\_distance O(n) -> Tổng O(n).
    - Tổng đánh giá: O(n^2 \* n) = O(n^3).
    - Sắp xếp lân cận: O(N log N) với N = O(n^2) -> O(n^2 log n^2) = O(n^2 log n).
  + Tổng cộng cho k vòng lặp: Khoảng O(k \* n^3) hoặc tệ hơn nếu lọc tabu không hiệu quả. Độ phức tạp khá cao do cách biểu diễn và đánh giá lân cận.
* **Độ chính xác so với lời giải tối ưu:**
  + Là thuật toán heuristic, không đảm bảo tìm ra lời giải tối ưu toàn cục.
  + Chất lượng lời giải phụ thuộc nhiều vào lời giải ban đầu, cấu trúc lân cận (chỉ có Swap là hạn chế), kích thước danh sách tabu và số vòng lặp.
  + Việc lưu toàn bộ lời giải vào tabu list có thể ngăn cản việc khám phá hiệu quả.

* **Tham số ảnh hưởng đến kết quả:**
  + iterations: Số vòng lặp. Càng lớn càng có khả năng tìm lời giải tốt hơn nhưng tốn thời gian. Cần đủ lớn để hội tụ.
  + tabu\_size: Kích thước danh sách tabu.
    - Quá nhỏ: Dễ bị quay lại các lời giải cũ, mắc kẹt trong chu trình ngắn.
    - Quá lớn: Hạn chế quá nhiều nước đi, làm chậm quá trình hội tụ hoặc bỏ lỡ các vùng tốt. Cần cân bằng và thử nghiệm.
  + Cách biểu diễn lời giải và split\_routes: Ảnh hưởng lớn đến chi phí tính toán và chất lượng lời giải.
  + Phép toán lân cận (generate\_neighbors): Chỉ dùng Swap có thể không đủ mạnh để khám phá hiệu quả. Các phép toán khác như Relocate, 2-opt thường hiệu quả hơn cho CVRP.

### **7. Ưu Và Nhược Điểm**

**Ưu Điểm:**

* **Khái niệm cơ bản dễ hiểu:** Ý tưởng về tìm kiếm lân cận và sử dụng bộ nhớ (tabu list) để tránh lặp lại là tương đối trực quan.
* **Có khả năng thoát cực tiểu địa phương:** Nhờ cơ chế tabu và việc chấp nhận các nước đi không cải thiện, thuật toán có thể vượt qua các điểm tối ưu cục bộ tốt hơn so với tìm kiếm cục bộ đơn giản.
* **Tương đối dễ cài đặt (với logic code này):** Sử dụng hoán vị và phép swap làm cho việc tạo lân cận đơn giản.

**Nhược Điểm:**

* **Biểu diễn lời giải không hiệu quả:** Biểu diễn bằng một hoán vị duy nhất và phải dùng split\_routes liên tục để đánh giá là tốn kém về mặt tính toán và có thể không phải là cách tự nhiên nhất để xử lý các tuyến đường trong CVRP.
* **Lân cận hạn chế:** Chỉ sử dụng phép Swap có thể không đủ mạnh để khám phá hiệu quả không gian lời giải CVRP. Thiếu các phép toán như Relocate (di chuyển khách hàng giữa các tuyến hoặc trong cùng tuyến), 2-opt (đảo đoạn trong tuyến).
* **Chiến lược Tabu kém hiệu quả:** Lưu trữ *toàn bộ* hoán vị trong danh sách tabu rất tốn bộ nhớ và làm cho việc kiểm tra tabu chậm (O(n \* tabu\_size) cho mỗi lân cận nếu dùng list). Thông thường, chỉ lưu *thuộc tính* của nước đi (ví dụ: cặp khách hàng vừa swap, hoặc khách hàng vừa di chuyển) sẽ hiệu quả hơn nhiều.
* **Thiếu Tiêu chuẩn Ngưỡng vọng:** Code không có Aspiration Criteria, có thể bỏ lỡ cơ hội cải thiện lời giải tốt nhất nếu nước đi đó tình cờ bị tabu.
* **Độ nhạy với tham số:** Hiệu quả thuật toán phụ thuộc lớn vào việc lựa chọn iterations và tabu\_size.
* **Không đảm bảo tối ưu:** Là heuristic, chỉ tìm lời giải xấp xỉ.

### **8. Ứng Dụng**

Các giải pháp cho bài toán CVRP, bao gồm cả những giải pháp tìm được bằng Tabu Search, có ứng dụng rộng rãi trong thực tế:

* **Quản lý và lập kế hoạch vận tải:** Tối ưu hóa lộ trình cho đội xe giao hàng, xe chở rác, xe buýt trường học, dịch vụ sửa chữa tận nơi,....
* **Logistics và chuỗi cung ứng:** Giảm chi phí vận chuyển, cải thiện thời gian giao hàng, tăng hiệu quả sử dụng phương tiện.
* **Thương mại điện tử:** Lập kế hoạch giao hàng cho các đơn hàng online một cách hiệu quả, đặc biệt trong các khu vực đô thị đông đúc.
* **Dịch vụ giao nhận:** Tối ưu hóa tuyến đường cho các công ty chuyển phát nhanh, giao đồ ăn,....

### **9. Kết Luận**

Thuật toán Tìm kiếm Tabu là một phương pháp metaheuristic mạnh mẽ để giải quyết các bài toán tối ưu tổ hợp phức tạp như CVRP. Nó cải tiến dựa trên tìm kiếm cục bộ bằng cách tích hợp cơ chế bộ nhớ (danh sách tabu) để hướng dẫn tìm kiếm ra khỏi các cực tiểu địa phương và khám phá không gian lời giải rộng hơn.

Bản triển khai được phân tích trong báo cáo này sử dụng một cách tiếp cận dựa trên hoán vị khách hàng, phép toán lân cận là Swap và danh sách tabu lưu trữ các hoán vị đầy đủ. Mặc dù cách tiếp cận này có thể tạo ra lời giải khả thi cho CVRP, nó cũng bộc lộ một số hạn chế về hiệu quả tính toán (do split\_routes và chiến lược tabu) và khả năng khám phá (do lân cận chỉ dùng Swap).

Để cải thiện hiệu quả của việc áp dụng Tabu Search cho CVRP, các hướng phát triển tiềm năng bao gồm:

* Sử dụng biểu diễn lời giải dựa trên danh sách các tuyến đường trực tiếp.
* Bổ sung các phép toán lân cận mạnh mẽ hơn (Relocate, 2-opt, Cross-exchange).
* Áp dụng chiến lược tabu hiệu quả hơn (lưu thuộc tính nước đi thay vì toàn bộ lời giải).
* Tích hợp Tiêu chuẩn Ngưỡng vọng.
* Kết hợp các kỹ thuật đa dạng hóa và tăng cường.

Tóm lại, Tìm kiếm Tabu vẫn là một công cụ giá trị trong hộp công cụ giải quyết CVRP. Bản cài đặt cụ thể này là một điểm khởi đầu, nhưng việc tinh chỉnh chiến lược biểu diễn, lân cận và tabu là rất quan trọng để đạt được hiệu suất tốt nhất trong các ứng dụng thực tế.

## **D) THUẬT TOÁN DI TRUYỀN (GENITIC ALGORITHM)**

### 1. **Giới Thiệu Về Thuật Toán**

#### **1.1 Giới thiệu về Thuật Toán Di Truyền (Genetic Algorithm - GA)**

Thuật toán Di truyền (GA) là một thuật toán tìm kiếm và tối ưu hóa dựa trên cơ chế chọn lọc tự nhiên và di truyền học. GA thuộc nhóm các thuật toán tiến hóa (Evolutionary Algorithms) và thường được sử dụng để giải quyết các bài toán tối ưu hóa phức tạp, đặc biệt là các bài toán có không gian tìm kiếm lớn và không liên tục.

#### **1.2 Ý Tưởng thuật toán**

Ý tưởng cốt lõi của thuật toán di truyền là mô phỏng quá trình tiến hóa của sinh vật. Thuật toán bắt đầu với một quần thể (population) gồm các cá thể (individuals), mỗi cá thể đại diện cho một giải pháp tiềm năng cho bài toán. Sau đó, thông qua các phép toán di truyền như chọn lọc (selection), lai ghép (crossover), và đột biến (mutation), quần thể này sẽ dần dần tiến hóa qua các thế hệ (generations) để tìm ra các giải pháp tốt hơn.

**Giải thích nguyên tắc chính đằng sau thuật toán một cách đơn giản:**

GA hoạt động tương tự như cách các loài sinh vật tiến hóa để thích nghi tốt hơn với môi trường của chúng. Các cá thể “khỏe mạnh” (giải pháp tốt) có khả năng cao hơn được chọn để “sinh sản” (lai ghép) và tạo ra các cá thể con. Các cá thể con này có thể thừa hưởng các đặc điểm tốt từ cha mẹ của chúng. Đồng thời, quá trình đột biến giúp giới thiệu sự đa dạng vào quần thể, tránh việc thuật toán bị mắc kẹt trong các cực trị cục bộ. Qua nhiều thế hệ, quần thể sẽ dần dần chứa các cá thể có chất lượng cao hơn, đại diện cho các giải pháp tốt cho bài toán.

**Áp dụng cho CVRP:**

Trong bài toán CVRP, mỗi cá thể trong quần thể GA sẽ đại diện cho một cách định tuyến các xe để phục vụ tất cả các khách hàng. “Sức khỏe” của một cá thể (hàm mục tiêu) thường được đánh giá dựa trên tổng quãng đường di chuyển và việc tuân thủ các ràng buộc (ví dụ: không vượt quá tải trọng xe). Thuật toán sẽ tiến hành các bước chọn lọc, lai ghép và đột biến trên các tuyến đường này để tạo ra các tuyến đường mới, hy vọng sẽ có tổng quãng đường ngắn hơn và vẫn đảm bảo các ràng buộc về tải trọng.

### **2. Các Bước Triển Khai Chi Tiết của Thuật Toán (ví dụ)**

(Phần này sẽ được minh họa chi tiết hơn ở mục 5 với ví dụ cụ thể)

### **3. Pseudocode**

1. Khởi\_tạo\_Quần\_thể(Kích thước Quần thể):

- Tạo một quần thể các nhiễm sắc thể ngẫu nhiên (các giải pháp tiềm năng).

- Mỗi nhiễm sắc thể biểu diễn một chuỗi các lần ghé thăm khách hàng.

- Một biểu diễn phổ biến là một hoán vị các khách hàng, với các ký hiệu kho hàng được chèn vào để xác định các tuyến đường.

- Đảm bảo các giải pháp ban đầu là khả thi (tuân thủ các ràng buộc về sức chứa của xe). Nếu không, áp dụng cơ chế sửa chữa hoặc phạt nặng trong hàm thích nghi.

- Trả về quần thể ban đầu.

2. Đánh\_giá\_Quần\_thể(Quần thể):

- Đối với mỗi nhiễm sắc thể trong Quần thể:

- Giải mã nhiễm sắc thể thành một tập hợp các tuyến đường của xe.

- Tính toán độ thích nghi của nhiễm sắc thể.

- Độ thích nghi thường dựa trên tổng chi phí (ví dụ: tổng khoảng cách) của tất cả các tuyến đường.

- Áp dụng hình phạt cho độ thích nghi nếu các ràng buộc về sức chứa bị vi phạm. Hình phạt càng cao khi mức độ vi phạm càng lớn.

- Lưu trữ giá trị độ thích nghi cho mỗi nhiễm sắc thể.

3. Lặp lại cho i từ 1 đến Số\_Lượng\_Thế\_hệ:

- Chọn\_Cha\_mẹ(Quần thể):

- Chọn một tập hợp các nhiễm sắc thể cha mẹ từ Quần thể hiện tại dựa trên độ thích nghi của chúng.

- Các phương pháp chọn phổ biến: Chọn Bánh xe Roulette, Chọn Giải đấu, Chọn Xếp hạng. Các cá thể có độ thích nghi cao hơn có cơ hội được chọn cao hơn.

- Tạo\_Quần\_thể\_mới(Cha mẹ):

- Khởi tạo một danh sách rỗng cho quần thể mới.

- Trong khi kích thước quần thể mới nhỏ hơn Kích thước Quần thể:

- Chọn hai cha mẹ từ tập hợp Cha mẹ đã chọn.

- Thực hiện Lai ghép (với xác suất Tỷ lệ Lai ghép):

- Áp dụng một toán tử lai ghép cho hai cha mẹ để tạo ra một hoặc hai nhiễm sắc thể con.

- Các toán tử lai ghép VRP phổ biến: Lai ghép Thứ tự (OX), Lai ghép Ánh xạ Một phần (PMX), Lai ghép Chu trình (CX). Các toán tử này cần xử lý tính chất hoán vị và có thể là các ký hiệu kho hàng.

- Thêm các nhiễm sắc thể con vào danh sách quần thể mới.

- Nếu không xảy ra lai ghép (với xác suất 1 - Tỷ lệ Lai ghép):

- Sao chép trực tiếp các cha mẹ vào danh sách quần thể mới.

- Đột\_biến\_Quần\_thể(Quần thể mới):

- Đối với mỗi nhiễm sắc thể trong Quần thể mới (trừ có thể là những nhiễm sắc thể tốt nhất):

- Thực hiện Đột biến (với xác suất Tỷ lệ Đột biến):

- Áp dụng một toán tử đột biến cho nhiễm sắc thể.

- Các toán tử đột biến VRP phổ biến: Đột biến Hoán đổi, Đột biến Chèn, Đột biến Đảo ngược. Các toán tử này sửa đổi chuỗi các lần ghé thăm khách hàng.

- Đảm bảo đột biến không tạo ra các nhiễm sắc thể không hợp lệ hoặc áp dụng cơ chế sửa chữa.

- Đánh\_giá\_Quần\_thể(Quần thể mới):

- Tính toán độ thích nghi của mỗi nhiễm sắc thể trong Quần thể mới.

- Thay\_thế\_Quần\_thể(Quần thể hiện tại, Quần thể mới):

- Thay thế Quần thể hiện tại bằng Quần thể mới.

- Chủ nghĩa tinh hoa (Elitism): Tùy chọn, giữ lại (các) nhiễm sắc thể tốt nhất từ Quần thể hiện tại sang Quần thể mới trước khi thay thế để đảm bảo giải pháp tốt nhất được tìm thấy cho đến nay không bị mất đi.

4. Điều\_kiện\_Kết\_thúc:

- Vòng lặp kết thúc sau một số thế hệ cố định (như được định nghĩa bởi Số\_Lượng\_Thế\_hệ).

- Các điều kiện kết thúc khác có thể bao gồm: không có sự cải thiện về độ thích nghi tốt nhất trong một số thế hệ nhất định, hoặc đạt được giá trị độ thích nghi mục tiêu.

5. Trả về nhiễm sắc thể có độ thích nghi tốt nhất được tìm thấy trong suốt các thế hệ.

### **4. Code Python (Ví dụ đơn giản hóa)**

import random  
  
class Customer:  
 def \_\_init\_\_(self, id, demand):  
 self.id = id  
 self.demand = demand  
  
class Vehicle:  
 def \_\_init\_\_(self, capacity):  
 self.capacity = capacity  
  
def create\_initial\_population(num\_individuals, customers, num\_vehicles):  
 # Đây là một cách khởi tạo rất đơn giản, có thể không đảm bảo tính khả thi  
 population = []  
 for \_ in range(num\_individuals):  
 random.shuffle(customers)  
 routes = [[] for \_ in range(num\_vehicles)]  
 customer\_index = 0  
 for i in range(len(customers)):  
 routes[i % num\_vehicles].append(customers[i].id)  
 population.append(routes)  
 return population  
  
def calculate\_fitness(  
 individual\_routes, customers\_data, vehicle\_capacity, distance\_matrix  
):  
 total\_distance = 0  
 feasible = True  
 for route in individual\_routes:  
 current\_demand = 0  
 if route:  
 total\_distance += distance\_matrix[0][  
 route[0]  
 ] # Kho đến khách hàng đầu tiên  
 for i in range(len(route) - 1):  
 total\_distance += distance\_matrix[route[i]][route[i + 1]]  
 current\_demand += customers\_data[  
 route[i] - 1  
 ].demand # Giả sử ID bắt đầu từ 1  
 total\_distance += distance\_matrix[route[-1]][  
 0  
 ] # Khách hàng cuối cùng về kho  
 current\_demand += customers\_data[route[-1] - 1].demand  
 if current\_demand > vehicle\_capacity:  
 feasible = False  
 if not feasible:  
 return float("inf") # Trả về vô cùng nếu không khả thi  
 return total\_distance  
  
  
def selection(population, fitnesses, num\_parents):  
 # Ví dụ: Tournament selection  
 parents = []  
 for \_ in range(num\_parents):  
 tournament = random.sample(  
 range(len(population)), 3  
 ) # Chọn ngẫu nhiên 3 cá thể  
 winner\_index = min(tournament, key=lambda i: fitnesses[i])  
 parents.append(population[winner\_index])  
 return parents  
  
  
def crossover(parent1, parent2):  
 # Đây là một ví dụ crossover đơn giản, cần cải tiến cho CVRP  
 if not parent1 or not parent2:  
 return parent1, parent2  
 child1 = []  
 child2 = []  
 # Thực hiện crossover dựa trên thứ tự khách hàng  
 # Cần một phương pháp crossover phù hợp hơn cho bài toán định tuyến  
 return parent1, parent2  
  
  
def mutate(individual, mutation\_rate):  
 # Đây là một ví dụ đột biến đơn giản, cần cải tiến cho CVRP  
 if random.random() < mutation\_rate:  
 # Ví dụ: Đổi chỗ hai khách hàng trong một tuyến đường ngẫu nhiên  
 route\_index = random.randint(0, len(individual) - 1)  
 if len(individual[route\_index]) > 1:  
 i, j = random.sample(range(len(individual[route\_index])), 2)  
 individual[route\_index][i], individual[route\_index][j] = (  
 individual[route\_index][j],  
 individual[route\_index][i],  
 )  
 return individual  
  
  
# --- Ví dụ sử dụng ---  
num\_customers = 4  
customers\_data = [Customer(1, 2), Customer(2, 3), Customer(3, 4), Customer(4, 2)]  
num\_vehicles = 2  
vehicle\_capacity = 6  
distance\_matrix = [  
 [0, 10, 12, 16, 8],  
 [10, 0, 5, 9, 11],  
 [12, 5, 0, 7, 13],  
 [16, 9, 7, 0, 10],  
 [8, 11, 13, 10, 0],  
] # Kho là nút 0, khách hàng 1 đến 4  
  
population\_size = 50  
num\_generations = 100  
mutation\_rate = 0.1  
num\_parents = 20  
  
population = create\_initial\_population(population\_size, customers\_data[:], num\_vehicles)  
  
for generation in range(num\_generations):  
 fitnesses = [  
 calculate\_fitness(ind, customers\_data, vehicle\_capacity, distance\_matrix)  
 for ind in population  
 ]  
 best\_fitness = min(fitnesses)  
 best\_individual = population[fitnesses.index(best\_fitness)]  
 print(  
 f"Thế hệ {generation}: Fitness tốt nhất = {best\_fitness}, Tuyến đường = {best\_individual}"  
 )  
  
 parents = selection(population, fitnesses, num\_parents)  
 new\_population = parents[:]  
  
 while len(new\_population) < population\_size:  
 parent1 = random.choice(parents)  
 parent2 = random.choice(parents)  
 child1, child2 = crossover(parent1, parent2)  
 new\_population.append(mutate(child1[:], mutation\_rate))  
 if len(new\_population) < population\_size:  
 new\_population.append(mutate(child2[:], mutation\_rate))  
  
 population = new\_population  
  
fitnesses = [  
 calculate\_fitness(ind, customers\_data, vehicle\_capacity, distance\_matrix)  
 for ind in population  
]  
best\_fitness = min(fitnesses)  
best\_individual = population[fitnesses.index(best\_fitness)]  
print(  
 f"\nKết quả cuối cùng: Fitness tốt nhất = {best\_fitness}, Tuyến đường = {best\_individual}"  
)

**Lưu ý:** Đoạn code Python trên chỉ là một ví dụ rất đơn giản và chưa hoàn thiện cho bài toán CVRP. Các phép toán chọn lọc, lai ghép và đột biến cần được thiết kế phù hợp hơn để giải quyết bài toán này hiệu quả. Ví dụ, việc biểu diễn nghiệm (chromosome) và các phép toán di truyền cần đảm bảo tính hợp lệ của các tuyến đường (mỗi khách hàng được phục vụ một lần, không vượt quá tải trọng).

### **5. Ví Dụ / Minh Họa (Đơn Giản Hóa)**

### Kịch Bản:

Một công ty có một kho hàng (Depot) và 4 khách hàng (Customer 1, 2, 3, 4) với nhu cầu lần lượt là 2, 3, 4 và 2 đơn vị hàng hóa. Công ty có 2 xe tải, mỗi xe có sức chứa tối đa là 6 đơn vị hàng hóa. Ma trận khoảng cách giữa kho và các khách hàng (và giữa các khách hàng) được cho như sau:

0 (Kho) 1 2 3 4  
0 (Kho) 0 10 12 16 8  
1 10 0 5 9 11  
2 12 5 0 7 13  
3 16 9 7 0 10  
4 8 11 13 10 0

Mục tiêu là tìm ra hai tuyến đường cho hai xe tải sao cho tổng quãng đường di chuyển là nhỏ nhất và không có xe nào chở quá 6 đơn vị hàng hóa.

### Thực Hiện Từng Bước:

**Thế hệ 1 (Khởi tạo):**

Giả sử chúng ta tạo ra một quần thể ban đầu gồm một vài cá thể (cách định tuyến ngẫu nhiên):

* **Cá thể 1:** Xe 1: [1, 2], Xe 2: [3, 4]
* **Cá thể 2:** Xe 1: [1, 3], Xe 2: [2, 4]
* **Cá thể 3:** Xe 1: [4, 1], Xe 2: [2, 3]

**Đánh giá độ thích nghi:**

Chúng ta tính toán tổng quãng đường và kiểm tra ràng buộc tải trọng cho mỗi cá thể.

* **Cá thể 1:**
  + Xe 1: 0 -> 1 -> 2 -> 0 (10 + 5 + 12 = 27), Tổng nhu cầu: 2 + 3 = 5 (<= 6)
  + Xe 2: 0 -> 3 -> 4 -> 0 (16 + 10 + 8 = 34), Tổng nhu cầu: 4 + 2 = 6 (<= 6)
  + Tổng quãng đường: 27 + 34 = 61. **Khả thi.**
* **Cá thể 2:**
  + Xe 1: 0 -> 1 -> 3 -> 0 (10 + 9 + 16 = 35), Tổng nhu cầu: 2 + 4 = 6 (<= 6)
  + Xe 2: 0 -> 2 -> 4 -> 0 (12 + 13 + 8 = 33), Tổng nhu cầu: 3 + 2 = 5 (<= 6)
  + Tổng quãng đường: 35 + 33 = 68. **Khả thi.**
* **Cá thể 3:**
  + Xe 1: 0 -> 4 -> 1 -> 0 (8 + 11 + 10 = 29), Tổng nhu cầu: 2 + 2 = 4 (<= 6)
  + Xe 2: 0 -> 2 -> 3 -> 0 (12 + 7 + 16 = 35), Tổng nhu cầu: 3 + 4 = 7 (> 6). **Không khả thi.**

**Chọn lọc:**

Giả sử Cá thể 1 và Cá thể 2 được chọn làm cha mẹ vì chúng có độ thích nghi tốt hơn (quãng đường ngắn hơn và khả thi). Cá thể 3 bị loại bỏ vì không khả thi.

**Lai ghép:**

Chúng ta áp dụng một phép lai ghép (ví dụ: Order Crossover) lên Cá thể 1 và Cá thể 2 để tạo ra các cá thể con mới. (Việc minh họa phép lai ghép chi tiết ở đây sẽ phức tạp hơn và cần hình ảnh trực quan). Giả sử sau khi lai ghép, chúng ta có các cá thể con:

* **Cá thể con 1:** Xe 1: [1, 3], Xe 2: [2] (chưa hoàn chỉnh, cần xử lý để đảm bảo mỗi khách hàng được phục vụ)
* **Cá thể con 2:** Xe 1: [4], Xe 2: [1, 2, 3] (tương tự, cần xử lý)

**Đột biến:**

Chúng ta áp dụng phép đột biến (ví dụ: swap hai khách hàng trong một tuyến đường) lên các cá thể con. Ví dụ, ở Cá thể con 1, chúng ta có thể đổi chỗ 1 và 3:

* **Cá thể con 1 (sau đột biến):** Xe 1: [3, 1], Xe 2: [2] (vẫn cần xử lý để tạo thành các tuyến đường hợp lệ cho tất cả khách hàng và kiểm tra ràng buộc tải trọng).

**Lặp lại:**

Quá trình chọn lọc, lai ghép và đột biến được lặp lại qua nhiều thế hệ, với hy vọng quần thể sẽ dần dần tiến hóa để tìm ra các giải pháp tốt hơn với tổng quãng đường ngắn hơn và đảm bảo ràng buộc về tải trọng.

**Minh họa Sweep (không áp dụng trực tiếp cho GA):** Thuật toán Sweep thường sử dụng góc để nhóm các khách hàng lại với nhau trước khi định tuyến, không phải là một phần cơ bản của GA.

**Minh họa C&W (không áp dụng trực tiếp cho GA):** Thuật toán Clarke & Wright dựa trên việc tính toán “tiết kiệm” khi kết hợp hai tuyến đường, khác với cơ chế tiến hóa của GA.

**Minh họa GA (tiếp tục ví dụ):**

Sau nhiều thế hệ, quần thể có thể hội tụ về một giải pháp tốt, ví dụ:

* **Giải pháp tốt nhất:** Xe 1: [4, 1], Xe 2: [2, 3] (Tổng quãng đường và tính khả thi sẽ được đánh giá).

**Minh họa Tìm kiếm Tabu (không áp dụng trực tiếp cho GA):** Tìm kiếm Tabu là một metaheuristic khác dựa trên việc cấm các nước đi gần đây để tránh lặp lại, không phải là cơ chế hoạt động của GA.

**Mục tiêu:** Mục tiêu của ví dụ đơn giản này là để hiểu các giai đoạn cơ bản của thuật toán di truyền: khởi tạo, đánh giá, chọn lọc, lai ghép và đột biến. Việc triển khai GA cho CVRP thực tế sẽ phức tạp hơn nhiều, đặc biệt là trong việc thiết kế các phép toán di truyền phù hợp để đảm bảo tính hợp lệ và hiệu quả của các giải pháp.

### **6. Phân tích hiệu suất và tham số**

### **Thời gian chạy (time complexity)**

Thời gian chạy của thuật toán di truyền phụ thuộc vào nhiều yếu tố, bao gồm:

* Kích thước quần thể (N)
* Số thế hệ (G)
* Độ phức tạp của việc đánh giá độ thích nghi (thường phụ thuộc vào số lượng khách hàng và phương pháp tính toán quãng đường)
* Độ phức tạp của các phép toán lai ghép và đột biến

Trong trường hợp tổng quát, thời gian chạy của GA có thể được ước tính là O(G \* N \* (độ phức tạp đánh giá + độ phức tạp lai ghép + độ phức tạp đột biến)). Đối với bài toán CVRP, việc đánh giá độ thích nghi có thể tốn thời gian nếu chúng ta cần kiểm tra tính khả thi của các tuyến đường một cách chi tiết. Các phép toán lai ghép và đột biến cũng cần được thiết kế cẩn thận để tránh tạo ra các giải pháp không hợp lệ và có thể có độ phức tạp khác nhau tùy thuộc vào cách biểu diễn nghiệm.

### **Độ chính xác so với lời giải tối ưu**

Thuật toán di truyền là một metaheuristic, do đó nó không đảm bảo tìm ra lời giải tối ưu cho bài toán CVRP. Tuy nhiên, GA có khả năng khám phá không gian tìm kiếm rộng lớn và thường tìm được các giải pháp gần tối ưu trong thời gian hợp lý, đặc biệt đối với các bài toán có kích thước lớn và độ phức tạp cao.

Để đánh giá độ chính xác của GA, người ta thường so sánh kết quả của nó với các lời giải tối ưu đã biết cho các bài toán benchmark (nếu có). Sự khác biệt giữa giải pháp tìm được bởi GA và giải pháp tối ưu thường được sử dụng để đo lường hiệu suất của thuật toán.

### **Tham số ảnh hưởng lớn tới kết quả**

Một số tham số quan trọng có ảnh hưởng lớn đến hiệu suất của thuật toán di truyền cho CVRP bao gồm:

* **Kích thước quần thể (Population Size):** Quần thể lớn hơn có thể khám phá không gian tìm kiếm rộng hơn nhưng cũng làm tăng thời gian tính toán. Quần thể quá nhỏ có thể dẫn đến việc thuật toán bị mắc kẹt trong các cực trị cục bộ.
* **Số thế hệ (Number of Generations):** Số thế hệ càng lớn thì thuật toán càng có nhiều thời gian để tiến hóa và tìm ra các giải pháp tốt hơn. Tuy nhiên, sau một số thế hệ nhất định, sự cải thiện có thể trở nên không đáng kể.
* **Tỷ lệ lai ghép (Crossover Rate):** Xác suất mà hai cá thể cha mẹ được chọn sẽ thực hiện phép lai ghép. Tỷ lệ quá cao có thể dẫn đến việc mất đi các cá thể tốt, trong khi tỷ lệ quá thấp có thể làm chậm quá trình tìm kiếm.
* **Tỷ lệ đột biến (Mutation Rate):** Xác suất mà một cá thể con sẽ trải qua đột biến. Đột biến giúp duy trì sự đa dạng trong quần thể và tránh việc thuật toán bị mắc kẹt trong các cực trị cục bộ. Tỷ lệ quá cao có thể làm cho thuật toán trở thành một tìm kiếm ngẫu nhiên.
* **Phương pháp chọn lọc (Selection Method):** Các phương pháp chọn lọc khác nhau (ví dụ: roulette wheel, tournament selection) có thể ảnh hưởng đến tốc độ hội tụ và khả năng khám phá không gian tìm kiếm của thuật toán.
* **Phương pháp lai ghép và đột biến (Crossover and Mutation Operators):** Việc lựa chọn các phép toán lai ghép và đột biến phù hợp với đặc thù của bài toán CVRP là rất quan trọng để tạo ra các giải pháp hợp lệ và hiệu quả.

Việc điều chỉnh các tham số này thường được thực hiện thông qua thử nghiệm và kinh nghiệm để đạt được hiệu suất tốt nhất cho một bài toán CVRP cụ thể.

### **7. Ưu Và Nhược Điểm**

**Ưu điểm của thuật toán di truyền cho CVRP:**

* **Khả năng giải quyết các bài toán phức tạp:** GA có thể xử lý các bài toán CVRP có số lượng khách hàng lớn và nhiều ràng buộc phức tạp.
* **Không yêu cầu kiến thức đặc biệt về bài toán:** GA là một thuật toán tìm kiếm tổng quát và không đòi hỏi phải có kiến thức sâu sắc về cấu trúc của không gian giải pháp.
* **Khả năng tìm kiếm toàn cục tốt:** GA có khả năng khám phá không gian tìm kiếm rộng lớn và ít có khả năng bị mắc kẹt trong các cực trị cục bộ so với một số thuật toán heuristic khác.
* **Tính linh hoạt:** GA có thể được tùy chỉnh và kết hợp với các kỹ thuật khác để cải thiện hiệu suất.

**Nhược điểm của thuật toán di truyền cho CVRP:**

* **Không đảm bảo tìm ra nghiệm tối ưu:** GA là một heuristic và không có bằng chứng toán học nào đảm bảo rằng nó sẽ tìm thấy nghiệm tối ưu.
* **Thời gian chạy có thể lớn:** Đối với các bài toán rất lớn, GA có thể mất nhiều thời gian để hội tụ đến một giải pháp tốt.
* **Việc lựa chọn và điều chỉnh tham số có thể khó khăn:** Hiệu suất của GA phụ thuộc nhiều vào việc lựa chọn các tham số phù hợp, và việc tìm ra các giá trị tối ưu cho các tham số này có thể đòi hỏi nhiều thử nghiệm.
* **Việc thiết kế các phép toán di truyền phù hợp cho CVRP là một thách thức:** Cần có các phép toán lai ghép và đột biến đặc biệt để đảm bảo tính hợp lệ của các tuyến đường và duy trì sự đa dạng trong quần thể.

### **8. Ứng Dụng**

Thuật toán di truyền đã được ứng dụng thành công trong nhiều bài toán CVRP thực tế, bao gồm:

* **Lập kế hoạch vận chuyển hàng hóa:** Tìm ra các tuyến đường tối ưu cho xe tải giao hàng để giảm chi phí vận chuyển và thời gian giao hàng.
* **Quản lý đội xe:** Tối ưu hóa việc sử dụng đội xe để phục vụ một số lượng lớn khách hàng với các yêu cầu khác nhau.
* **Logistics và chuỗi cung ứng:** Lập kế hoạch vận chuyển hiệu quả trong toàn bộ chuỗi cung ứng.
* **Các bài toán định tuyến khác:** GA cũng có thể được áp dụng cho các biến thể khác của bài toán định tuyến xe, chẳng hạn như Vehicle Routing Problem with Time Windows (VRPTW) hoặc Multi-Depot Vehicle Routing Problem (MDVRP).

### **9. Tóm Tắt & Kết Luận**

### **Tóm Lược:**

Thuật toán di truyền là một phương pháp metaheuristic mạnh mẽ dựa trên các nguyên tắc của chọn lọc tự nhiên và di truyền học. Trong bài toán CVRP, GA hoạt động bằng cách duy trì một quần thể các giải pháp tiềm năng (các cách định tuyến xe) và tiến hóa quần thể này qua các thế hệ thông qua các phép toán chọn lọc, lai ghép và đột biến để tìm ra các tuyến đường tối ưu với chi phí thấp nhất và tuân thủ các ràng buộc về tải trọng.

### **Ý Nghĩa:**

Thuật toán di truyền là một công cụ hữu ích cho việc giải quyết bài toán CVRP, đặc biệt là đối với các trường hợp phức tạp mà các phương pháp tối ưu hóa chính xác không thể áp dụng được trong thời gian hợp lý. Mặc dù không đảm bảo tìm ra nghiệm tối ưu, GA thường có thể tìm thấy các giải pháp chất lượng cao trong một khoảng thời gian chấp nhận được. Nó là một **metaheuristic mạnh mẽ cho các không gian tìm kiếm phức tạp**, cung cấp một cách tiếp cận linh hoạt và hiệu quả để giải quyết một trong những bài toán tối ưu hóa tổ hợp quan trọng nhất trong lĩnh vực vận tải và logistics.

1. **SO SÁNH VÀ ĐÁNH GIÁ KẾT QUẢ**

**Ví dụ: Cho bảng input**

| **ID** | **X** | **Y** | **Demand** | **Ghi chú** |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 🚚 Depot (điểm bắt đầu & kết thúc) |
| 1 | 10 | 5 | 10 | Khách hàng 1 |
| 2 | 12 | 8 | 15 | Khách hàng 2 |
| 3 | 5 | 10 | 20 | Khách hàng 3 |
| 4 | 15 | 12 | 25 | Khách hàng 4 |
| 5 | 7 | 4 | 30 | Khách hàng 5 |
| 6 | 9 | 15 | 10 | Khách hàng 6 |
| 7 | 4 | 6 | 15 | Khách hàng 7 |
| 8 | 8 | 13 | 20 | Khách hàng 8 |
| 9 | 14 | 5 | 30 | Khách hàng 9 |
| 10 | 11 | 10 | 10 | Khách hàng 10 |
| 11 | 3 | 8 | 15 | Khách hàng 11 |
| 12 | 6 | 14 | 10 | Khách hàng 12 |
| 13 | 13 | 9 | 25 | Khách hàng 13 |
| 14 | 2 | 5 | 20 | Khách hàng 14 |
| 15 | 9 | 6 | 15 | Khách hàng 15 |
| 16 | 7 | 12 | 20 | Khách hàng 16 |
| 17 | 10 | 2 | 10 | Khách hàng 17 |
| 18 | 5 | 3 | 15 | Khách hàng 18 |
| 19 | 6 | 7 | 20 | Khách hàng 19 |
| 20 | 13 | 7 | 20 | Khách hàng 20 |

**📌 Ý nghĩa các cột:**

* **ID**: Số thứ tự khách hàng (Depot luôn là 0)
* **(X, Y)**: Tọa độ trên mặt phẳng 2D
* **Demand**: Lượng hàng cần giao
* **Ghi chú**: Nhận diện vai trò (Depot hay Khách hàng)

**So sánh các thuật toán dựa trên input đã cho:**

**-Thuật toán Tabu Search:** tổng quãng đường 130-140 với 4 xe ( là nhóm Metaheuristic- chuyên về mặt cho ra kết quả gần tối ưu hoặc tối ưu , nhưng tốc độ xử lý chậm , đây là thuật toán xử lý chậm nhất trong 4 thuật toán nhưng cho kết quả tối ưu nhất )

**-Thuật toán Genitic Algorithm:** tổng quãng đường 150-160 với 5 xe ( là nhóm Metaheuristic, đây cũng là dạng thuật toán về cho ra kết quả tối ưu , tốc độ xử lý nhanh hơn Tabu Search ,nhưng kết quả chất lượng không được tối ưu bằng Tabu Search, vì ở input này vẫn còn khá nhỏ nên việc tối ưu của Genitic Algorithm vẫn chưa được thể hiện rõ ràng , tiềm năng xử lý của thuật toán này đối với các input lớn hơn sẽ càng tối ưu hơn)

**-Thuật toán Clarke and Wright Savings**: tổng quãng đường 140.36 với 4 xe (là nhóm Constructive Heuristic- cho ra giải pháp nhanh chóng , nhưng kết quả sẽ không được tối ưu, ở đây tốc độ xử lý của thuật toán này rất cao chỉ thấp hơn thuật toán Sweep , nhưng bù lại kết quả vẫn tối ưu hơn Sweep 1 phần )

**-Thuật toán Sweep:** tổng quãng đường 164,66 với 4 xe ( là nhóm Constructive Heuristic - cho ra giải pháp nhanh nhất , nhưng kết quả cũng k đảm bảo nhất , chỉ nên mang tính tham khảo )

1. **TỔNG KẾT**

Trong bối cảnh thương mại điện tử và dịch vụ giao hàng phát triển mạnh mẽ, bài toán Capacitated Vehicle Routing Problem (CVRP) đóng vai trò then chốt trong việc tối ưu hóa hoạt động phân phối, từ đó giúp doanh nghiệp giảm chi phí vận hành và nâng cao hiệu quả phục vụ khách hàng. Bài toán đặt ra nhiều thách thức do ràng buộc về sức chứa phương tiện và tính chất NP-khó của nó, đòi hỏi những phương pháp giải quyết phù hợp với quy mô và đặc điểm dữ liệu.

Báo cáo đã trình bày tổng quan về các phương pháp giải CVRP, từ thuật toán chính xác cho đến các thuật toán heuristic và meta-heuristic. Trong đó, những thuật toán như Clarke and Wright Savings, Sweeping Algorithm, Genetic Algorithm và Tabu Search đã được triển khai và so sánh hiệu quả thực tế. Kết quả cho thấy mỗi phương pháp đều có ưu và nhược điểm riêng tùy thuộc vào cấu trúc dữ liệu và mục tiêu tối ưu hóa.

Đặc biệt, báo cáo cũng đã chỉ ra những hạn chế của các thuật toán cổ điển trong một số tình huống cụ thể như phân bố theo cụm xuyên tâm, từ đó mở ra hướng nghiên cứu cho các biến thể thuật toán hoặc kết hợp nhiều chiến lược để cải thiện chất lượng lời giải.

**Giá trị của nghiên cứu**

Giải quyết vấn đề thực tế: CVRP phản ánh trực tiếp bài toán vận chuyển hàng hóa có giới hạn sức chứa trong thực tế. Việc áp dụng các giải thuật giúp doanh nghiệp cải thiện hiệu quả hoạt động và tiết kiệm chi phí vận tải.

Tối ưu hóa quy trình vận chuyển: Các thuật toán không chỉ giúp xác định tuyến đường tối ưu mà còn hỗ trợ lập lịch cho phương tiện, góp phần tăng năng suất, giảm thời gian và chi phí vận hành.

Phân tích và đánh giá giải thuật: Việc thực nghiệm và so sánh hiệu quả giữa các phương pháp khác nhau giúp xác định chiến lược giải quyết phù hợp với từng loại dữ liệu và yêu cầu cụ thể.

Phát triển giải pháp mới: Nghiên cứu CVRP tạo tiền đề cho việc phát triển các thuật toán heuristic và metaheuristic mới, có khả năng xử lý tốt hơn các bài toán lớn, đa ràng buộc và phức tạp.

Ứng dụng liên ngành: Kết quả nghiên cứu không chỉ giới hạn trong lĩnh vực logistics mà còn có thể được ứng dụng vào tối ưu mạng viễn thông, quản lý dự án, lập kế hoạch giao thông, định tuyến mạng.

**Hướng nghiên cứu trong tương lai**

Nghiên cứu về bài toán CVRP vẫn còn nhiều tiềm năng để mở rộng và phát triển trong tương lai, với một số định hướng cụ thể như sau:

* So sánh hiệu quả giữa các nhóm giải thuật trên cùng một lớp bài toán: Có thể tiếp tục nghiên cứu và triển khai thêm nhiều giải thuật khác nhau, đặc biệt là các thuật toán metaheuristic như Ant Colony Optimization, Tabu Search, hoặc Particle Swarm Optimization,… nhằm so sánh hiệu quả giữa các phương pháp trên các bộ dữ liệu có kích thước khác nhau. Việc đánh giá cả về chất lượng lời giải lẫn thời gian thực thi sẽ giúp xác định rõ ràng ưu – nhược điểm của từng thuật toán trong từng bối cảnh cụ thể.
* Tập trung vào các bài toán có quy mô lớn: Do tính chất NP-hard của CVRP, các giải thuật chính xác không thể áp dụng cho các bài toán lớn vì chi phí tính toán tăng theo cấp số nhân. Trong khi đó, nhóm thuật toán metaheuristic có khả năng tìm kiếm lời giải gần tối ưu trong thời gian tính toán hợp lý, rất phù hợp cho các bài toán thực tế có hàng trăm hoặc hàng nghìn điểm giao hàng.
* Khảo sát và tăng quy mô bài toán theo hướng tối ưu mạng: Một hướng mở rộng tiềm năng là áp dụng mô hình CVRP cho các bài toán tối ưu hóa mạng viễn thông, nơi các nút mạng đại diện cho trạm phát sóng hoặc trung tâm dữ liệu. Việc tăng quy mô nút mạng không chỉ giúp kiểm nghiệm hiệu năng của các thuật toán mà còn làm rõ giới hạn tính toán và khả năng mở rộng của từng phương pháp trong không gian tìm kiếm lớn và phức tạp.