

**ĐẠI HỌC QUỐC GIA HÀ NỘI  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

**TRẦN LAN PHƯƠNG**

**MỘT THUẬT TOÁN TỐI ƯU ĐÀN KIẾN  
GIẢI BÀI TOÁN ĐIỀU PHỐI XE**

**LUẬN VĂN THẠC SĨ**

**Ngành: Kỹ thuật phần mềm**

**Hà Nội, năm 2019**

**ĐẠI HỌC QUỐC GIA HÀ NỘI  
TRƯỜNG ĐẠI HỌC CÔNG NGHỆ**

**TRẦN LAN PHƯƠNG**

**MỘT THUẬT TOÁN TỐI ƯU ĐÀN KIẾN  
GIẢI BÀI TOÁN ĐIỀU PHỐI XE**

Ngành : Kỹ thuật phần mềm

Chuyên ngành : Kỹ thuật phần mềm

Mã số : 8480103.01

**LUẬN VĂN THẠC SĨ**

**Ngành: Kỹ thuật phần mềm**

**Người hướng dẫn khoa học: PGS. TS Hoàng Xuân Huân**

**Hà Nội, năm 2019**

## **LỜI CAM ĐOAN**

Tôi xin cam đoan rằng luận văn này của tự cá nhân tôi tìm hiểu, nghiên cứu dưới sự hướng dẫn giúp đỡ của PGS.TS Hoàng Xuân Huấn. Trong toàn bộ nội dung của luận văn, những điều đã được trình bày hoặc là của chính cá nhân tôi hoặc là được tổng hợp từ nhiều nguồn tài liệu. Các tài liệu tham khảo được trích dẫn và chú thích đầy đủ. Các số liệu được trích dẫn trong luận văn này là trung thực. Kết quả nghiên cứu này không trùng với bất cứ công trình nào đã được công bố trước đây.

Tôi xin hoàn toàn chịu trách nhiệm với lời cam đoan của mình.

**TÁC GIẢ LUẬN VĂN**

**Trần Lan Phương**

## LỜI CẢM ƠN

Luận văn “*Một thuật toán tối ưu đàn kiến giải bài toán điều phối xe*” được hoàn thành với sự giúp đỡ tận tình của các thầy giáo, cô giáo trường Đại học công nghệ - Đại học Quốc gia Hà Nội, các đồng nghiệp, gia đình và sự nỗ lực của bản thân trong suốt quá trình học tập và thực hiện luận văn.

Trước tiên, em xin chân thành cảm ơn tới Ban giám hiệu nhà trường, phòng Đào tạo Đại học và Sau đại học, khoa Công nghệ thông tin và các thầy giáo, cô giáo trong trường đã tận tình truyền đạt kiến thức, giúp đỡ em trong suốt quá trình học tập chương trình cao học tại trường.

Đặc biệt em xin bày tỏ lòng biết ơn sâu sắc tới thầy giáo PGS.TS Hoàng Xuân Huấn, Trường Đại học Công nghệ - Đại học Quốc gia Hà Nội đã tận tình chỉ dẫn, giúp đỡ và cung cấp cho em những kiến thức, tài liệu cần thiết để hoàn thành luận văn này.

Cuối cùng, em xin gửi lời cảm ơn tới gia đình, bạn bè đồng nghiệp và người thân đã tin tưởng, giúp đỡ, động viên, khích lệ em trong suốt quá trình làm luận văn tốt nghiệp. Do thời gian và kiến thức có hạn chắc chắn luận văn cũng không thể tránh khỏi những thiếu sót, hạn chế. Kính mong nhận được sự chỉ bảo và góp ý của quý Thầy, Cô.

*Em xin chân thành cảm ơn!*

*Hà Nội, tháng 06 năm 2019*

Học viên

**Trần Lan Phương**

# MỤC LỤC

MỞ ĐẦU.....	1
CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN ĐỊNH TUYẾN XE.....	3
1.1.    Phát biểu bài toán định tuyến xe .....	3
1.2.    Các biến thể quan trọng của bài toán định tuyến xe.....	5
1.2.1.  Dựa vào cấu trúc đường đi .....	5
1.2.2.  Dựa vào yêu cầu vận chuyển.....	5
1.2.3.  Dựa vào ràng buộc nội tuyến .....	6
1.2.3.1.  Ràng buộc về lượng hàng hóa.....	6
1.2.3.2.  Ràng buộc về độ dài lộ trình .....	7
1.2.3.3.  Ràng buộc về việc tái sử dụng xe.....	7
1.2.3.4.  Ràng buộc về thời gian cho mỗi lộ trình .....	7
1.2.4.  Dựa vào đặc điểm đội xe.....	8
1.2.5.  Dựa vào ràng buộc liên tuyến.....	9
1.2.6.  Dựa vào hàm mục tiêu .....	10
1.3.    Các hướng tiếp cận và ứng dụng của bài toán định tuyến xe .....	10
1.4.    Kết luận chương .....	12
CHƯƠNG 2: THUẬT TOÁN TỐI ƯU ĐÀN KIẾN .....	13
2.1.    Giới thiệu về thuật toán.....	13
2.2.    Từ kiến tự nhiên đến kiến nhân tạo .....	13
2.2.1.  Đàn kiến tự nhiên.....	14
2.2.2.  Đàn kiến nhân tạo .....	15
2.3.    Trình bày giải thuật.....	15
2.3.1.  Đồ thị cấu trúc .....	16
2.3.2.  Trình bày về thuật toán ACO cơ bản.....	18
2.3.3.  Quy tắc cập nhật vết mùi.....	19
2.3.3.1.  Thuật toán AS .....	19
2.3.3.2.  Thuật toán ACS.....	22
2.3.3.3.  Thuật toán Max-Min .....	23
2.3.3.4.  Thuật toán Min- Max tron.....	25
2.4.    Một số vấn đề liên quan khi áp dụng ACO.....	26

2.4.1. Đặc tính hội tụ .....	26
2.4.2. Thực hiện song song .....	26
2.4.3. ACO kết hợp với tìm kiếm cục bộ.....	27
2.4.4. Thông tin heuristic .....	27
2.4.5. Số lượng kiến.....	28
2.4.6. Tham số bay hơi .....	28
2.5. Kết luận chương. ....	28
<b>CHƯƠNG 3: ỨNG DỤNG THUẬT TOÁN TỐI ƯU ĐÀN KIẾN GIẢI BÀI TOÁN</b>	
<b>MPDPTW .....</b>	<b>29</b>
3.1. Phát biểu bài toán .....	29
3.1.1. Giới thiệu bài toán .....	29
3.1.2. Xây dựng mô hình toán học .....	30
3.2. Một số phương pháp giải quyết bài toán MPDPTW .....	32
3.3. Thuật toán ACO giải quyết bài toán MPDPTW .....	33
3.3.1. Đồ thị cấu trúc .....	33
3.3.2. Xây dựng giải pháp.....	34
3.3.3. Quy tắc cập nhật mùi .....	36
3.3.4. Thủ tục tìm kiếm cục bộ .....	36
3.4. Kết luận chương .....	38
<b>CHƯƠNG 4: CÀI ĐẶT VÀ ĐÁNH GIÁ THỰC NGHIỆM .....</b>	<b>39</b>
4.1. Cài đặt chương trình .....	39
4.2. Mô tả dữ liệu thực nghiệm.....	41
4.3. Hiệu năng lời giải mô hình toán học .....	42
<b>KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN .....</b>	<b>50</b>
<b>TÀI LIỆU THAM KHẢO .....</b>	<b>51</b>
<b>PHỤ LỤC.....</b>	<b>54</b>

## DANH MỤC THUẬT NGỮ VIẾT TẮT

STT	Từ viết tắt	Từ đầy đủ	Ý nghĩa
1	ACO	Ant Colony Optimization	Tối ưu đàn kiến
2	ACS	Ant Colony System	Hệ đàn kiến
3	AS	Ant System	Hệ kiến
4	CG	Column Generation	Kỹ thuật sinh cột
5	CVRP	Capacitated Vehicle Routing Problem	Bài toán định tuyến xe hạn chế về sức chứa
6	DVRP	Distance Vehicle Routing Problem	Bài toán định tuyến xe hạn chế về khoảng cách
7	GA	Genetic Algorithm	Giải thuật di truyền
8	MMAS	Max – Min Ant System	Hệ kiến Max – Min
9	MPDPTW	Multi – Pickup and Delivery Problem with TimeWindows	Bài toán định tuyến xe đa điểm đón và giao hàng với thời gian cửa sổ
10	PDP	Pickup and Delivery Problem	Bài toán giao và nhận hàng
11	SA	Simulated Annealing	Giải thuật luyện kim
12	SMMAS	Smooth – Max Min Ant System	Hệ kiến Max – Min trơn
13	SOP	Sequential Ordering Problem	Bài toán đặt hàng tuần tự
14	TSP	Travelling Salesman Problem	Bài toán Người bán hàng
15	TU' TH	Tối ưu hóa tổ hợp	Tối ưu hóa tổ hợp
16	VRP	Vehicle Routing Problem	Bài toán định tuyến xe

# DANH MỤC HÌNH VẼ

Hình 1. 1 Ví dụ cho bài toán Người bán hàng– TSP.....	3
Hình 1. 2 Mô phỏng bài toán VRP. ....	4
Hình 1. 3 Mô phỏng bài toán CVRP.....	6
Hình 2. 1 Ví dụ về hoạt động của đàn kiến trong thực tế.....	14
Hình 2. 2 Ví dụ về hoạt động của đàn kiến nhân tạo. ....	15
Hình 2. 3 Đồ thị cấu trúc tổng quát cho bài toán cực trị hàm $f(x_1, \dots, x_n)$ .....	17
Hình 2. 4 Đặc tả thuật toán ACO.....	19
Hình 3. 1 Minh họa các yêu cầu R1 và R6 thuộc tuyến đường của xe k. ....	30
Hình 3. 2 Đồ thị cấu trúc cho bài toán MPDPTW.....	33
Hình 3. 3 Mô tả thuật toán T1.....	36
Hình 3. 4 Mô tả thuật toán T2.....	37
Hình 3. 5 Mô tả quá trình tìm kiếm cục bộ.....	37
Hình 4. 1 Các bộ dữ liệu thử nghiệm.....	42
Hình 4. 2 Mô tả dữ liệu đầu vào.....	43
Hình 4. 3 Mô tả kết quả chương trình. ....	44
Hình 4. 4 Mở chương trình .....	54
Hình 4. 5 Chạy chương trình .....	55



# DANH MỤC BẢNG BIỂU

<i>Bảng 1. 1 Các biến thể của bài toán VRP phân chia theo đặc điểm đội xe. ....</i>	<i>8</i>
<i>Bảng 4. 1 Kết quả thực nghiệm của ACO sử dụng SMMAS. ....</i>	<i>45</i>
<i>Bảng 4. 2 Bảng so sánh ACO và CPLEX. ....</i>	<i>46</i>
<i>Bảng 4. 3 So sánh kết quả tốt nhất của SMMAS và MMAS có cùng vòng lặp. ....</i>	<i>48</i>
<i>Bảng 4. 4 So sánh kết quả tốt nhất của SMMAS và MMAS có cùng thời gian chạy. ....</i>	<i>49</i>

# MỞ ĐẦU

## 1. Lý do chọn đề tài

Trong những năm gần đây, công nghệ thông tin giữ vai trò quan trọng trong hầu hết các lĩnh vực của đời sống, xã hội. Đặc biệt, mỗi ngày có một lượng lớn chi phí được sử dụng cho quá trình vận chuyển hàng hóa, du lịch trong các bài toán vận tải. Việc đưa ra lịch trình di chuyển của phương tiện vận tải một cách hợp lý giúp giảm chi phí được sử dụng cho nhiên liệu, thiết bị, phí bảo trì xe và tiền lương của lái xe là vô cùng quan trọng. Do đó, việc nghiên cứu các cách thức để đưa ra kế hoạch định tuyến tối ưu bằng chương trình máy tính rất có giá trị.

Bài toán định tuyến xe (*Vehicle Routing Problem - VRP*) là bài toán đã được nghiên cứu trong suốt hơn 50 năm qua của Vận Trù Học với nhiều ứng dụng trong thực tế, đặc biệt là trong lĩnh vực giao thông vận tải và hậu cần,... Bài toán có liên quan đến việc thiết lập hành trình cho các phương tiện từ kho đi giao hàng tới các thành phố và quay trở lại kho ban đầu mà không vượt quá năng lực hạn chế của mỗi xe với tổng chi phí tối thiểu. Bài toán này đã được chứng minh thuộc lớp các bài toán NP-khó[8], có rất nhiều giải thuật khác nhau đã được đề xuất để tìm lời giải tối ưu cho bài toán này như: thuật toán di truyền, kỹ thuật nhánh cận, thuật toán sinh cột, ... Tuy nhiên các giải thuật trên đều tốn nhiều chi phí về thời gian hoặc không gian lớn. Thuật toán tối ưu hóa đàn kiến được đề xuất bởi Dorigo từ năm 1991 đến nay đã có nhiều cải tiến về quy tắc cập nhật vết mùi làm cho thuật toán trở nên hiệu quả hơn.

Do vậy, tôi đã lựa chọn đề tài “*Một thuật toán tối ưu đàn kiến giải quyết bài toán điều phối xe*” để nghiên cứu.

## 2. Lịch sử vấn đề nghiên cứu

Bài toán định tuyến xe (*Vehicle Routing Problem-VRP*) được nghiên cứu trên nhiều ràng buộc khác nhau và có rất nhiều giải thuật được đề xuất cho bài toán này. Trong đó bài toán định tuyến xe đa điểm đón và giao hàng với thời gian cửa sổ (*Multi Pickup and Delivery Problem with TimeWindows-MPDPTW*) là một biến thể của bài toán VRP đã được nghiên cứu trong nhiều ứng dụng thực tế như giao hàng thực phẩm, xe đưa đón học sinh tới trường,... Bài toán thuộc lớp các bài toán NP-Khó được đề xuất bởi Nacache và các cộng sự vào năm 2018 [4] trên tạp chí Vận Trù Học và họ giải quyết bài toán một cách chính xác nhờ áp dụng kỹ thuật nhánh cận và phát triển một kỹ thuật tìm kiếm heuristic là A hybrid Adaptive Large Neighborhood Search (ALNS) để cải thiện lời giải. Tuy nhiên các giải thuật trên đều tốn chi phí về thời gian hoặc không gian lớn, không khả thi để đưa ra lời giải tối ưu cho bài toán định tuyến có kích thước lớn.

### 3. Mục đích nghiên cứu

Mục tiêu của luận văn này là đưa ra một giải pháp để giải quyết bài toán với kích thước lớn và dễ dàng cho việc cài đặt thực nghiệm. Cụ thể, chúng tôi áp dụng một thuật toán tối ưu đàn kiến (ACO) với quy tắc cập nhật mùi Max-Min tron (SMMAS) có tìm kiếm địa phương để đưa ra lời giải cho bài toán định tuyến xe đa điểm đón và giao hàng với thời gian cửa sổ (MPDPTW)

### 4. Đối tượng và phạm vi nghiên cứu

- Các lý thuyết cơ bản về bài toán định tuyến xe.
- Bài toán định tuyến xe đa điểm đón và giao hàng với thời gian cửa sổ (MPDPTW)
- Ngôn ngữ lập trình Java

### 5. Phương pháp nghiên cứu

- Tìm hiểu bài toán định tuyến xe và các biến thể quan trọng của bài toán này.
- Tìm hiểu thuật toán tối ưu đàn kiến.
- Xây dựng mô hình tối ưu (mô hình toán học) cho bài toán định tuyến xe đa điểm đón và giao hàng với thời gian cửa sổ (MPDPTW).
- Sử dụng thuật toán tối ưu đàn kiến tìm lời giải tối ưu cho mô hình đã xây dựng.
- Sử dụng ngôn ngữ lập trình Java để kiểm nghiệm và đánh giá.

### 6. Cấu trúc luận văn

Nội dung của luận văn được trình bày trong 4 chương:

- Chương I: giới thiệu về bài toán định tuyến xe, các biến thể của bài toán cũng như các cách tiếp cận để giải quyết bài toán này.
- Chương II: trình bày thuật toán tối ưu đàn kiến.
- Chương III: giới thiệu bài toán định tuyến xe đa điểm đón và giao hàng với thời gian cửa sổ (MPDPTW): phát biểu bài toán, trình bày các phương pháp đã sử dụng để giải quyết bài toán này. Sau đó, chúng tôi xây dựng một thuật toán tối ưu đàn kiến (ACO) để giải quyết bài toán này.
- Chương IV: diễn tả cách thức cài đặt, bộ dữ liệu thực nghiệm và sau đó là đánh giá thực nghiệm cho mô hình được xây dựng trong chương III.
- Kết luận
- Tài liệu tham khảo.

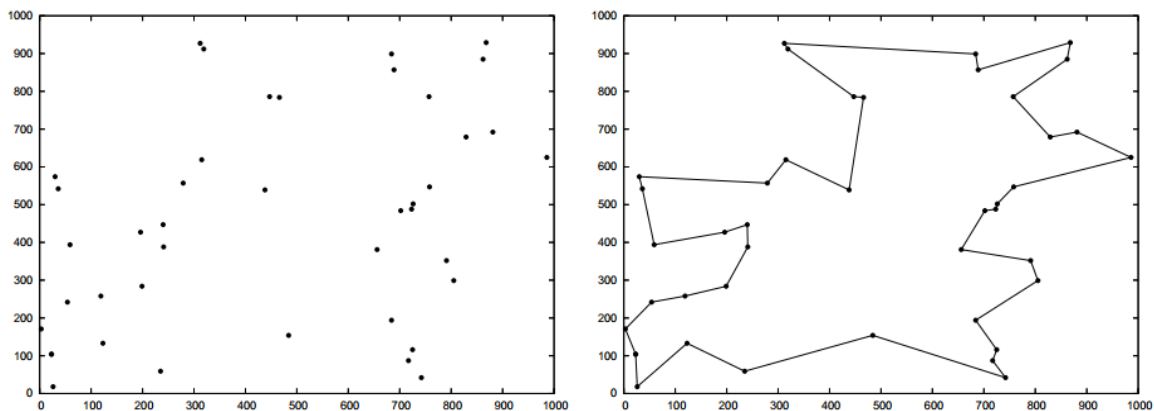
# CHƯƠNG 1: GIỚI THIỆU BÀI TOÁN ĐỊNH TUYẾN XE

Ở chương này, chúng tôi đưa ra cái nhìn tổng quan về bài toán định tuyến xe thông qua các dạng biến thể của bài toán. Trình bày một số cách tiếp cận để giải quyết bài toán này và ứng dụng của nó trong thực tế.

## 1.1. Phát biểu bài toán định tuyến xe

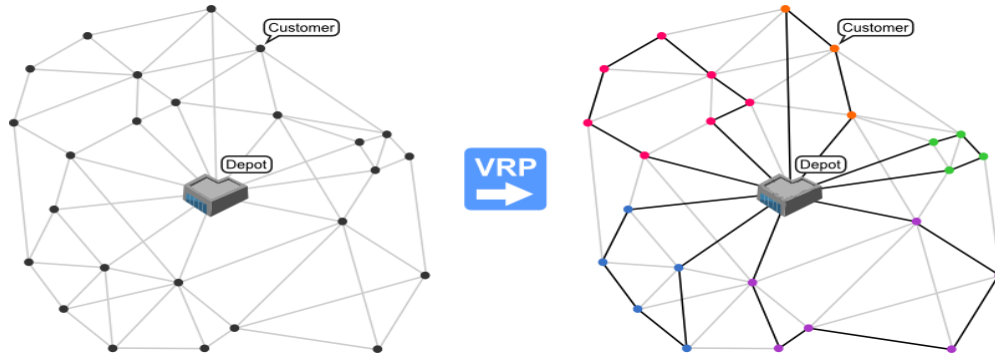
Bài toán Người bán hàng (*Travelling Salesman Problem* - gọi tắt là *TSP*) chính là trường hợp đơn giản nhất của bài toán định tuyến xe (*Vehicle Routing Problem* - *VRP*) với một xe giao hàng duy nhất - người bán hàng. Xuất phát từ một thành phố, người bán hàng sẽ đi giao hàng tại tất cả các thành phố và quay trở về thành phố ban đầu. Nhiệm vụ của bài toán là phải tìm một lộ trình tối ưu nhất (ví dụ như tổng độ dài quãng đường dịch chuyển là nhỏ nhất) để người bán hàng đi giao hàng cho tất cả thành phố theo dự định, mỗi thành phố được ghé thăm duy nhất một lần. Hình 1.1 minh họa cho bài toán TSP với dữ liệu đầu vào được cho ở hình bên trái và hình bên phải chính là một giải pháp tối ưu được đưa ra cho bài toán.

Bài toán TSP có thể được mô hình hóa bằng một đồ thị, trong đó các đỉnh của đồ thị tương ứng với các thành phố, các cạnh tương ứng với đường đi giữa các thành phố, khoảng cách giữa các thành phố là trọng số tương ứng của các cạnh nối chúng. Lời giải tối ưu của bài toán TSP là một đường đi ngắn nhất nối tất cả các điểm trên đồ thị hay còn gọi là một chu trình Hamilton ngắn nhất.



Hình 1. 1 Ví dụ cho bài toán Người bán hàng– TSP.

Trên cơ sở mở rộng bài toán TSP, bài toán VRP cơ bản bao gồm một tập các xe được tập kết tại kho hàng và mỗi khách hàng có các yêu cầu vận chuyển khác nhau. Vấn đề đặt ra là phải tìm cách định tuyến cho tập các xe phục vụ được tất cả khách hàng với chi phí vận chuyển là nhỏ nhất.



Hình 1. 2 Mô phỏng bài toán VRP.

Hình 1.2 mô phỏng một bài toán VRP trong đó hình bên trái thể hiện các xe được tập kết tại Depot, mỗi khách hàng được biểu diễn bởi một điểm chấm đen với yêu cầu vận chuyển của họ. Các cạnh nối giữa các điểm diễn tả đường đi giữa chúng. Hình bên phải diễn tả lời giải cho bài toán, trong đó các chu trình nối bởi các đường nét đậm diễn tả lộ trình của các tuyến xe phục vụ các khách hàng.

Trong một bài toán định tuyến xe cơ bản, các xe sẽ xuất phát từ các kho hàng, đi giao hàng hoặc nhận hàng từ khách hàng và quay trở về điểm xuất phát. Các khái niệm được sử dụng trong bài toán bao gồm:

- ✓ Xe (*Vehicle*): phương tiện được dùng để vận chuyển hàng hóa. Trong thực tế hầu như các xe là không đồng nhất, chúng được phân loại dựa vào các đặc điểm như sức chứa của xe (tức tải trọng hàng hóa tối đa xe có thể đáp ứng), loại hàng hóa mà xe có thể vận chuyển (hàng hóa đông lạnh, hàng hóa khô...), chi phí vận chuyển (có hai loại chi phí thông dụng: chi phí cố định – chi phí cần thiết ban đầu để xe có thể khởi hành, chi phí này không phụ thuộc vào quãng đường mà xe phải đi; chi phí động – chi phí tiêu tốn trên từng đơn vị quãng đường mà xe phải đi) ...
- ✓ Kho hàng (*Depot*): là nơi cất trữ hàng hóa hay cũng có thể là địa điểm xuất phát/quay về của các xe. Trong một số bài toán, hàng hóa cần giao có thể được cất trữ trong một vài kho hàng.
- ✓ Khách hàng (*Customer*): có thể đón hàng do xe giao tới hoặc chuyển hàng lên xe để vận chuyển về kho hoặc cả hai. Mỗi khách hàng yêu cầu một lượng hàng hóa nhất định, có thể đưa ra một số yêu cầu khác về thời gian cho phép xe đến giao hàng, thời gian cho phép bốc dỡ hàng...

- ✓ Lộ trình (*Route*): mỗi hành trình bắt đầu đi từ điểm xuất phát rồi quay trở về điểm ban đầu (kho hàng) của một xe được coi là một lộ trình.

Bài toán định tuyến xe được xem là một trong những bài toán phức tạp và kinh điển nhất của Vận trù học. Có thể phát biểu bài toán VRP cơ bản một cách đơn giản như sau: Có một tập hợp  $M$  xe giống nhau cùng xuất phát tại một kho hàng đi làm nhiệm vụ giao hàng cho  $N$  khách hàng, mỗi khách hàng đòi hỏi cung cấp một lượng hàng nhất định. Yêu cầu đặt ra của bài toán là tìm đường đi ngắn nhất cho  $M$  xe đáp ứng được tất cả các đòi hỏi của khách hàng.

## 1.2. Các biến thể quan trọng của bài toán định tuyến xe

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, chúng tôi sẽ trình bày các 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,...

### 1.2.1. Dựa vào cấu trúc đường đi

- Bài toán VRP có các khách hàng được biểu diễn bởi các cạnh (*Arc Routing Problem – ARP*) là một bài toán đặc biệt, thay vì các khách hàng được biểu diễn bằng các điểm trong đồ thị như trong các bài toán VRP thông thường thì sẽ được biểu diễn bằng các cạnh, tương ứng với các đoạn đường đi trong thực tế. Chẳng hạn như bài toán người phát báo, tìm đường đi cho xe phun nước ra đường cho hạ nhiệt độ đường nhựa,...

- Bài toán VRP có khoảng cách bất đối xứng (*Asymmetric VRP- AVR*): trong bài toán này đồ thị biểu diễn đường đi là một đồ thị có hướng.

### 1.2.2. Dựa vào yêu cầu vận chuyển

- Bài toán VRP với yêu cầu giao hàng trước (*VRP with Backhauls - VRPB*): Trong bài toán này, ngầm định rằng các lộ trình giao hàng/ nhận hàng đều bắt đầu/ kết thúc tại kho hàng. Các xe phải thực hiện các yêu cầu giao hàng xong rồi mới đến các yêu cầu nhận hàng để đảm bảo lượng hàng cần chứa vượt quá sức chứa của xe.

- Bài toán kết hợp cả yêu cầu giao và nhận hàng (*Mixed VRPB*): Tức là việc giao và nhận hàng có thể xen kẽ nhau hoặc không. Ràng buộc ở đây là tổng lượng hàng nhận từ khách và lượng hàng có trên xe phải không vượt quá sức chứa tối đa của xe.

- Bài toán giao và nhận hàng đồng thời (*VRP with Simultaneous Pickup and Delivery – VRPSD*): Trong khi khách hàng ở bài toán *VRPB* và *Mixed VRPB* chỉ yêu cầu giao hoặc nhận hàng thì với bài toán này, một khách hàng có 2 yêu cầu vận chuyển - lấy hàng từ kho giao cho khách và nhận hàng từ khách chuyển về kho. Hơn nữa, việc giao và nhận hàng cho một khách hàng phải được thực hiện bởi cùng một xe cụ thể và trong duy nhất một

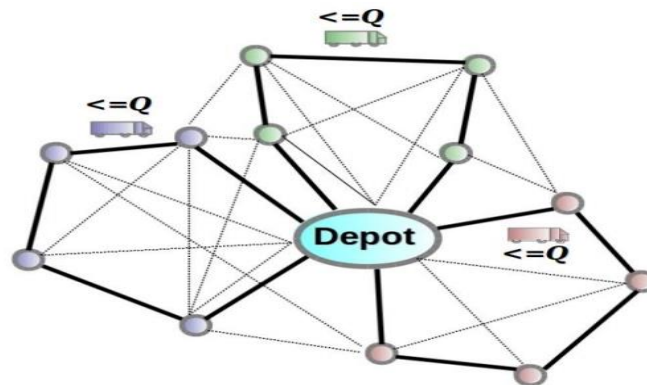
lượt, tức là xe chỉ về kho khi đã thực hiện yêu cầu giao – nhận của khách hàng. Ứng dụng thực tế của bài toán này là việc giao đồ uống đóng chai cho các cửa hàng ăn và thu nhận vỏ chai về kho để tái sử dụng; taxi chở khách từ sân bay về khách sạn hoặc ngược lại... Ràng buộc sức chứa của bài toán này là lượng hàng cần giao cũng như lượng hàng nhận từ khách không vượt quá sức chứa của xe.

### 1.2.3. Dựa vào ràng buộc nội tuyến

Yếu tố giúp phân loại các dạng biến thể của bài toán VRP là các ràng buộc nội tuyến trong bài toán, tức các ràng buộc cần có của một lộ trình khả thi. Trong phần này, chúng tôi trình bày các ràng buộc về: lượng hàng hóa, độ dài lộ trình, việc tái sử dụng xe trong các lộ trình trong ngày, vấn đề sắp xếp thời gian biểu và việc kết hợp các ràng buộc này trong các bài toán thực tế. Các ràng buộc này được phân vào nhóm các ràng buộc nội tuyến, tức các ràng buộc cho một lộ trình độc lập, không ảnh hưởng đến các lộ trình khác.

#### 1.2.3.1. Ràng buộc về lượng hàng hóa

Bài toán VRP với ràng buộc sức chứa (*Capacitated Vehicle Routing Problem - CVRP*): Bài toán bao gồm một tập xe và một tập yêu cầu của khách hàng, mỗi xe có một sức chứa là khác nhau, mục tiêu của bài toán này là phải tìm đường đi cho tất cả các xe phục vụ được hết các yêu cầu của khách hàng sao cho tổng lượng hàng hóa mà xe phải chở trong một lộ trình không được vượt quá sức chứa của xe.



Hình 1. 3 Mô phỏng bài toán CVRP.

Khi sức chứa của xe được mô tả cụ thể hơn bởi trọng lượng, dung tích, khoảng cách giữa các hàng hóa trên xe với nhau ... ta có bài toán CVRP 2 chiều (*CVRP with 2-dimensional Loading constraints, 2L-CVRP*) và CVRP 3 chiều (*CVRP with 3-dimensional Loading constraints, 3L-CVRP*).

Với bài toán CVRP 2 chiều, các hàng hóa có dạng hình chữ nhật sẽ được xếp vào ngăn, vào vị trí có hình chữ nhật phù hợp. Hàng hóa giao cho mỗi nhóm khách hàng được

thực hiện bởi một loại xe nhất định. Việc xếp hàng lên xe còn cần quan tâm đến tính xoay được hay không của hàng hóa.

Bài toán CVRP 3 chiều mở rộng thêm một số ràng buộc cụ thể để đảm bảo tính ổn định của hàng hóa chứa trên xe, đặc biệt quan trọng đối với hàng hóa dễ vỡ, giúp cho việc tháo dỡ hàng hóa nhanh chóng hơn.

Khi sức chứa của xe không được mô tả cụ thể bởi trọng lượng hay dung tích, cụ thể là trong bài toán *VRP with Compartments (VRPC)*, các xe được chia thành các ngăn và hàng hóa không được để lẫn giữa các ngăn (ngăn nhiệt độ thường - ngăn đông lạnh, ngăn đồ ăn – ngăn đồ dùng khác...).

#### 1.2.3.2. Ràng buộc về độ dài lộ trình

Bài toán VRP với ràng buộc quãng đường đi tối đa của mỗi xe (*Distance- Constrained VRP - DVRP*): trong bài toán ứng với mỗi loại xe có một tham số để giới hạn độ dài quãng đường tối đa mà xe được phép đi. Mục tiêu của bài toán này là định tuyến cho tập các xe đi làm nhiệm vụ sao cho tổng quãng đường mà mỗi xe phải đi không được vượt quá tham số giới hạn đã đặt ra.

#### 1.2.3.3. Ràng buộc về việc tái sử dụng xe

Hầu hết các biến thể của bài toán VRP đều ngầm định rằng mỗi xe chỉ chạy một lộ trình thuộc tập lộ trình định ra trong ngày. Tuy nhiên, trong những trường hợp trọng tải  $Q$  của xe nhỏ hoặc các ràng buộc khác làm cho lượng khách hàng được phục vụ trên mỗi lộ trình là nhỏ, số lượng xe sẵn dùng là hạn chế, khi đó, cần tái sử dụng xe, tức là mỗi xe cần chạy nhiều hơn một lộ trình. Nghĩa là một chiếc xe có thể bắt đầu từ kho hàng đi giao hàng tại các địa điểm rồi quay trở về điểm xuất phát ban đầu và tiếp tục lấy hàng đi giao cho đến khi vượt quá tải trọng hay vi phạm thời gian giao hàng cho phép của mỗi xe.

#### 1.2.3.4. Ràng buộc về thời gian cho mỗi lộ trình

Bài toán VRP với ràng buộc thời gian cửa sổ (*VRP with Time Windows - VRPTW*): trong bài toán này, mỗi khách hàng sẽ chỉ cho phép xe đến giao hoặc nhận hàng trong khoảng thời gian nhất định. Ví dụ như khách hàng  $i$  chỉ cho phép xe đến giao hàng trong khoảng thời gian biểu diễn bởi đoạn  $[a_i, b_i]$ , nghĩa là nếu xe đến giao hàng cho khách hàng thứ  $i$  vào trước thời điểm  $a_i$ , xe sẽ phải đứng chờ cho đến thời điểm  $a_i$  mới được giao hàng, đồng thời việc giao hàng của xe cho khách hàng thứ  $i$  phải được kết thúc trước thời điểm  $b_i$ . Nếu ràng buộc thời gian đối với khách hàng thứ  $i$   $[a_i - b_i]$  được thay thế bởi một tập các khoảng thời gian cho phép  $([a_i^1, b_i^1], ..., [a_i^p, b_i^p])$  thì bài toán VRPTW trở thành bài toán *VRP with Multiple- time Windows*.



#### 1.2.4. Dựa vào đặc điểm đội xe

- Bài toán VRP với nhiều kho hàng (*Multiple Depot VRP - MDVRP*): mỗi xe được gán với một kho hàng nhất định, là nơi bắt đầu và kết thúc mỗi lộ trình của xe. Trong thực tế thì các kho có sức chứa nhất định, một tập con các xe được phân chia tập hợp tại một kho hàng cụ thể, tức là điểm bắt đầu và kết thúc lộ trình của tập các xe là tại kho hàng này.
- Lớp bài toán VRP với tập các xe hỗn hợp (*Heterogeneous or mixed Fleet VRP*): trường hợp này bao gồm tập các loại xe khác nhau về sức chứa, tốc độ, sự tiêu hao nhiên liệu trong quá trình sử dụng...

Để phân loại các bài toán con trong lớp bài toán này, cần dựa vào 3 yếu tố:

- Số lượng xe sẵn có (fleet size): có giới hạn hay không.
- Chi phí cố định (fixed costs): chi phí được cấp ban đầu cho việc vận chuyển.
- Chi phí lộ trình (routing costs): chi phí trong quá trình vận chuyển.

Cách phân loại bài toán VRP dựa vào 3 yếu tố trên được đưa ra bởi Baldacci, Battarra và Vigo [9] như sau:

**Bảng 1. 1 Các biến thể của bài toán VRP phân chia theo đặc điểm đội xe.**

<i>Tên viết tắt</i>	<i>Tên bài toán</i>	<i>Số lượng</i>	<i>Chi phí cố định</i>	<i>Chi phí lộ trình</i>
<i>HVRPFD</i>	Heterogeneous VRP with Fixed Costs and Vehicle- Dependent Routing Costs	giới hạn	được xem xét	phụ thuộc
<i>HVRPD</i>	Heterogeneous VRP with Vehicle- Dependent Routing Costs	giới hạn	không được xem xét	phụ thuộc
<i>FSMFD</i>	Fleet Size and Mix VRP with Fixed Costs and Vehicle- Dependent Routing Costs	không giới hạn	được xem xét	phụ thuộc
<i>FSMD</i>	Fleet Size and Mix VRP with VehicleDependent Routing Costs	không giới hạn	không được xem xét	phụ thuộc
<i>FSMF</i>	Fleet Size and Mix VRP with Fixed Costs	không giới hạn	được xem xét	không phụ thuộc

### 1.2.5. Dựa vào ràng buộc liên tuyến

Các bài toán VRP trong thực tế thường bao gồm một số ràng buộc liên như sau:

- Ràng buộc cân bằng: ràng buộc này quan tâm đến sự công bằng, cân bằng khối lượng công việc giữa các xe. Các yếu tố: thời gian, số điểm dừng, độ dài tuyến đường hay lượng hàng hóa của mỗi lộ trình đều được xem xét. Trong các bài toán thực tế cần xem xét ràng buộc cân bằng trong mối quan hệ với các ràng buộc khác để đạt lợi nhuận cao nhất.
- Ràng buộc tài nguyên của hệ thống: ràng buộc được đề cập là các ràng buộc tài nguyên chung của hệ thống. Ví dụ, diện tích mỗi kho hàng chỉ đủ phân bổ cho một số lượng xe nhất định; việc xử lý hàng hóa ở kho chỉ xử lý được với một tốc độ nhất định và chỉ làm việc trong một khoảng thời gian quy định trước, hàng hóa về sau thời gian đó phải đợi xử lý trong ngày hôm sau...
- Vấn đề đồng bộ hóa: *VRP with Multiple Synchronization constraints - VRPMS*. Trong lớp bài toán này, có một số yếu tố cần quan tâm sau:
  - Đồng bộ hóa các yêu cầu: việc định tuyến các xe phải phục vụ hết các yêu cầu của khách hàng. Với các bài toán VRP đơn giản thì đây có thể chỉ là vấn đề phân chia tập yêu cầu của khách hàng thành các tập con và định tuyến các xe tương ứng với mỗi tập con đó. Tuy nhiên, với các bài toán VRP phức tạp hơn (như các bài toán VRP chia nhỏ đơn hàng (*SDVRP*), phục vụ theo chu kỳ (*PVRP*)...), việc đồng bộ hóa các lộ trình cũng là vấn đề đáng quan tâm.
  - Đồng bộ quá trình thực hiện: Các xe khác nhau thì thực hiện những công việc khác nhau ở những địa điểm giống nhau hoặc khác nhau. Một số trường hợp thực tế yêu cầu cụ thể công việc của xe X phải thực hiện trước công việc của xe Y. Ví dụ, hai xe X và Y có nhiệm vụ lắp đặt thiết bị mạng cho hai địa điểm X' và Y', trong đó địa điểm X' là nơi cung cấp dịch vụ mạng cho địa điểm Y'. Khi đó, công việc của xe X nhất thiết phải thực hiện trước công việc của xe Y.
  - Đồng bộ hóa hoạt động: Trong một số trường hợp đặc biệt, hai hoặc nhiều xe có thể cùng chạy trên một tuyến đường. Ví dụ, xe dọn tuyết vào mùa đông gồm hai xe cùng chạy trên một tuyến đường: một xe xới tuyết lên và một xe dọn sạch tuyết.
  - Đồng bộ hóa tải trọng: là việc đồng bộ lượng hàng giao, nhận và chuyển giữa các địa điểm và giữa các xe trong quá trình tương tác lẫn nhau.
  - Đồng bộ hóa tài nguyên hệ thống: ràng buộc về sức chứa của mỗi xe và ràng buộc tài nguyên chung của hệ thống.

### 1.2.6. Dựa vào hàm mục tiêu

Các biến thể của bài toán VRP kể trên đều được xét đến trong trường hợp hàm mục tiêu đơn thuần là tối thiểu hóa chi phí vận chuyển. Tuy nhiên, có rất nhiều mục tiêu tối ưu hóa cần được quan tâm trong các bài toán thực tế.

Trong một số trường hợp, các mục tiêu cần tối ưu có thể mâu thuẫn nhau. Ví dụ, trong bài toán định tuyến xe quan tâm đến mức độ hài lòng của khách hàng, mục tiêu tối ưu mức độ hài lòng của khách hàng có thể mâu thuẫn với mục tiêu tối thiểu hóa thời gian vận chuyển, tối thiểu hóa số lượng xe sử dụng. Khi đó, cần dựa vào thứ tự ưu tiên của các mục tiêu để đưa ra phương án định tuyến tốt nhất. Thông thường, mục tiêu tối thiểu hóa số xe sử dụng sẽ được xét đến trước, với số lượng xe được gán cố định, các mục tiêu khác mới được xét đến để xây dựng các phương án tối ưu hóa.

### 1.3. Các hướng tiếp cận và ứng dụng của bài toán định tuyến xe

Bài toán định tuyến xe và các biến thể của nó được coi là bài toán tối ưu hóa tổ hợp có độ phức tạp tính toán cao và được phân loại thuộc lớp NP- khó[8]. Các hướng tiếp cận cho bài toán này được chia làm 2 loại chính [29]: các phương pháp chính xác và các phương pháp gần đúng.

Các phương pháp chính xác thường được áp dụng để đưa ra lời giải tối ưu cho bài toán. Các phương pháp này chủ yếu áp dụng để giải quyết các bài toán VRP có kích thước nhỏ và số ràng buộc ít do bị hạn chế về thời gian tìm kiếm lời giải. Phần lớn các thuật toán chính xác giải bài toán VRP được phát triển từ các thuật toán chính xác giải bài toán TSP, bao gồm: thuật toán nhánh cận (*branch and bound*)[32], thuật toán sinh cột (GA) [33], ...

Các phương pháp gần đúng gồm các thuật giải cho chất lượng lời giải gần với lời giải tối ưu như nhóm các giải thuật heuristic cổ điển, nhóm các giải thuật tìm kiếm cục bộ và nhóm các giải thuật metaheuristic. Các thuật toán tiêu biểu của phương pháp này được đề xuất trong [30][31].

- Các giải thuật heuristic cổ điển được phát triển vào những năm 1960-1990. Đến nay, các giải thuật này thường được kết hợp với metaheuristic với nhiệm vụ sinh lời giải khởi tạo hoặc cải thiện chất lượng lời giải sẵn có.
  - Nhóm các giải thuật khởi tạo
    - Các giải thuật *Savings*: khởi tạo  $m$  lộ trình tương ứng với  $m$  khách hàng. Sau đó, thực hiện ghép  $m$  các lộ trình này lại với nhau cho đến khi không thể ghép được nữa (do vi phạm giới hạn sức chứa của xe), việc chọn các lộ trình để ghép lại với nhau dựa trên một hàm *saving*.

- Các giải thuật *Insertion*: Lối giải được xây dựng bằng cách lần lượt chèn mỗi khách hàng vào một lộ trình, các lộ trình có thể được xây dựng đồng thời hoặc tuần tự. Chèn khách hàng vào lộ trình sao cho tổng quãng đường mà xe phải đi thêm là nhỏ nhất (áp dụng giải thuật tham lam).
- Các giải thuật gom nhóm khách hàng trước, tìm đường đi sau (*cluster-first, route-second*): Ban đầu, chia tập khách hàng thành các tập con, mỗi tập con này tương ứng với một lộ trình, sau đó xác định đường đi cụ thể cho từng lộ trình.
- Nhóm các giải thuật cải tiến chất lượng lời giải sẵn có (*improvement heuristics*): Các thuật giải thuộc nhóm này chỉnh sửa lời giải hiện tại thông qua hai loại phép *dịch chuyển* đó là phép *dịch chuyển* chỉ tác động trên một lộ trình duy nhất (*intra-route moves*) và phép *dịch chuyển* cùng lúc tác động trên nhiều lộ (*inter-route move*).
- Nhóm giải thuật tìm kiếm cục bộ (*Local Search*): bắt nguồn từ một phương án chấp nhận được sẽ thực hiện lặp lại các bước cải tiến lời giải dựa vào thay đổi cục bộ. Kỹ thuật này đòi hỏi phải xác định được cấu trúc lân cận của mỗi lời giải đang xét, nghĩa là chọn được tập các phương án chấp nhận được gần nhất với nó nhờ thay đổi thứ tự một số thành phần. Trong tìm kiếm cục bộ có hai yếu tố quan trọng cần quan tâm là tính tăng cường và tính đa dạng của quá trình tìm kiếm lời giải. Tính tăng cường là khả năng tìm kiếm quanh những vùng không gian được dự đoán là sẽ chứa các lời giải tối ưu, tính đa dạng là khả năng tìm kiếm tới những vùng không gian chứa các lời giải tránh việc tập trung tìm kiếm quanh không gian chứa điểm tối ưu cục bộ.
- Nhóm các giải thuật metaheuristic được đề xuất từ năm 1990 và được xem là nhóm các hướng tiếp cận có triển vọng nhất hiện nay và được đông đảo các nhà nghiên cứu quan tâm bởi vì với các bài toán có không gian tìm kiếm lớn thì các giải thuật này cho lời giải tương đối tốt trong khoảng thời gian chấp nhận được mà các giải thuật chính xác là không khả thi. Tuy nhiên, các giải thuật này đòi hỏi phải lựa chọn được các tham số phù hợp, cách thông dụng nhất ngày nay là dựa vào kinh nghiệm. Nhóm các giải thuật metaheuristic bao gồm:
  - Giải thuật luyện kim (*Simulated Annealing*)[34]: mô phỏng quá trình luyện kim thông thường, trong đó tinh thể kim loại được nung nóng rồi sau đó được làm nguội rất chậm cho tới khi nó đạt được cấu hình tinh thể cứng nhất (ở trạng thái năng lượng nhỏ nhất). Quá trình làm nguội đủ chậm cho kết quả cuối cùng sẽ là kim loại với cấu trúc rất tốt.
  - Giải thuật dựa vào miền láng giềng: Variable Neighborhood Search (VNS)[35], Large Neighborhood Search và Adaptive Large Neighborhood Search [37].

- Nhóm các giải thuật dựa trên quần thể: Giải thuật di truyền (*Genetic Algorithm*)[36], giải thuật tối ưu hóa đàn kiến (*Ant Colony Algorithm*)[11][13].

Trong thực tế, các bài toán VRP thường rất phức tạp, bởi có sự kết hợp của nhiều ràng buộc khác nhau. Chúng tôi sẽ đưa ra một mô hình bài toán VRP được ứng dụng để giải quyết các bài toán trong thực tế là bài toán giao hàng lạnh trong thành phố của công ty Cổ phần sữa Việt Nam Vinamilk trên cơ sở TP.Hồ Chí Minh[3]. Bài toán này xuất phát từ yêu cầu trong thực tế và có nhiều ràng buộc đặc trưng như ràng buộc về thời gian, đội xe không đồng nhất, loại hàng hóa là đa dạng,... Mỗi ngày các xe vận chuyển hàng hóa của công ty đều khởi hành từ một trạm xuất phát duy nhất là Xí Nghiệp Kho Vận Vinamilk – TP.HCM và đi giao hết số lượng hàng đã được phân rồi quay trở lại trạm xuất phát ban đầu. Trong quá trình giao hàng các xe sẽ phải đến các kho hàng lấy hàng đem đi giao cho khách. Công ty Vinamilk TP.HCM có 3 kho hàng: Kho Trường Thọ và kho Thống Nhất– Quận Thủ Đức (2 kho này có vị trí rất gần nhau nên có thể xem như 1 kho duy nhất) và kho thứ hai là kho Sài Gòn– Quận 12. Thời gian mở cửa của các kho hàng này trong khoảng từ 6:30 đến 15:00. Tại cùng một thời điểm, chỉ có tối đa 5 xe tập kết tại kho. Các loại mặt hàng lạnh của Công ty Vinamilk TP.HCM bao gồm hai nhóm: nhóm kem và nhóm sữa chua. Đối với các xe chở hàng lạnh yêu cầu phải có máy lạnh và nhóm kem yêu cầu nhiệt độ phải thấp hơn nhóm sữa chua. Do vậy, xe chở kem có thể chở được các mặt hàng thuộc nhóm sữa chua, nhưng ngược lại xe chở nhóm sữa chua thì không chở được nhóm kem. Đội xe chở hàng lạnh của Vinamilk TP.HCM gồm 25 chiếc, với sức chứa và khả năng chở khác nhau. Trong đó, có 11 xe có thể chở được cả kem và sữa chua còn các xe khác chỉ có thể chở sữa chua. Tại cùng một thời điểm, mỗi xe chỉ được chở một nhóm mặt hàng. Ở đây có thể xem chi phí vận chuyển của các xe là như nhau (mặc dù sức chứa mỗi xe là khác nhau) ... Có thể thấy rằng, bài toán giao hàng của công ty Cổ phần sữa Việt Nam Vinamilk TP.HCM là một bài toán VRP rất phức tạp, là kết hợp của nhiều ràng buộc khác nhau trong các bài toán VRP cơ bản.

#### **1.4. Kết luận chương**

Trong chương này chúng tôi đã giới thiệu về bài toán định tuyến xe, các biến thể quan trọng của bài toán và một số cách tiếp cận để giải bài toán này. Đây là cơ sở để chúng tôi nghiên cứu và xây dựng lời giải tối ưu cho bài toán “Đa điểm đón và giao hàng với thời gian cửa sổ”, một biến thể của bài toán định tuyến xe.

## CHƯƠNG 2: THUẬT TOÁN TỐI ƯU ĐÀN KIẾN

Chúng tôi đã trình bày về bài toán định tuyến xe, các biến thể quan trọng và một số cách tiếp cận để giải bài toán trong chương 1. Trong chương 2 chúng tôi sẽ tìm hiểu thuật toán tối ưu đàn kiến để giải bài toán này.

### 2.1. Giới thiệu về thuật toán

Bài toán tối ưu hóa tổ hợp (TU' TH) là bài toán hấp dẫn và thú vị, các bài toán này có thể đưa về bài toán tìm kiếm trên đồ thị và cho lời giải đúng hoặc gần đúng. Nhiều bài toán TU' TH thuộc lớp bài toán NP-khó và không giải được trong thời gian đa thức. Thay vì giải quyết một cách chính xác thì người ta thường giải bài toán này bằng phương pháp gần đúng và lời giải thu được gần với lời giải tối ưu và thời gian chạy khá nhanh. Các thuật toán gần đúng này thường được gọi là các thuật toán heuristic và được áp dụng để giải các bài toán cụ thể trong thực tế. Metaheuristic là mở rộng của thuật toán heuristic tổng quát được thiết kế để giải quyết một lớp các bài toán rộng lớn. Trong đó, phương pháp tối ưu hóa đàn kiến (*Ant Colony Optimization – ACO*) là một phương pháp metaheuristic dựa trên ý tưởng mô phỏng hành vi của đàn kiến trong tự nhiên thông qua cách tìm đường đi của chúng từ tổ tới nguồn thức ăn dựa vào mật độ mùi (*Pheromone*) mà các con kiến để lại trên đường đi.

Thuật toán tối ưu đàn kiến (ACO) được giới thiệu bởi Dorigo và lần đầu tiên được ứng dụng giải bài toán phân loại các trạm làm việc vào năm 1991. Thuật toán ACO có đặc điểm là kết hợp giữa các thông tin cấu trúc của lời giải tốt trong tương lai với các thông tin của lời giải tốt đã tìm trước đó.

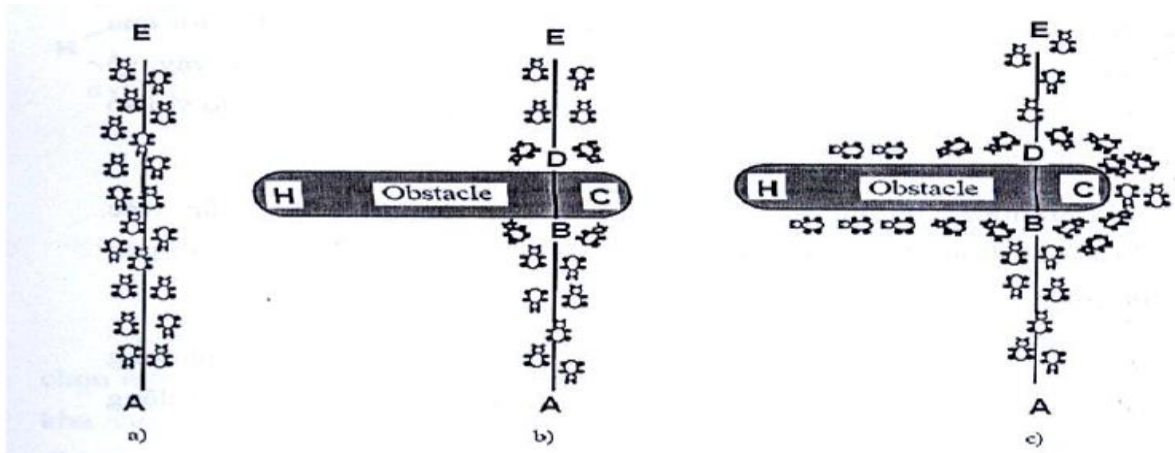
Hiệu quả của thuật toán ACO được thể hiện khi so sánh với các thuật toán nổi tiếng như thuật toán di truyền (GA), Tabu Search, Local Search,... Người ta đã ứng dụng thành công các thuật toán tối ưu đàn kiến trong một số bài toán tối ưu tổ hợp thường gặp như: bài toán người bán hàng, bài toán tô màu trên đồ thị, bài toán lập lịch,...

### 2.2. Từ kiến tự nhiên đến kiến nhân tạo

Trong quá trình tìm kiếm đường đi, các kiến thực hiện trao đổi thông tin gián tiếp thông qua phương thức tự tổ chức. Mặc dù đơn giản nhưng phương thức này tạo điều kiện cho các kiến có thể thực hiện được các công việc phức tạp vượt quá khả năng của mỗi kiến, đặc điểm nổi bật đó là khả năng tìm đường đi ngắn nhất từ tổ kiến tới nguồn thức ăn mặc dù kiến không thể đo được độ dài đường đi. Vậy trước tiên ta xét xem các kiến tìm đường đi bằng cách nào mà có thể giải quyết được các vấn đề tối ưu hóa.

### 2.2.1. Đàn kiến tự nhiên

Trong quá trình tìm kiếm đường đi, các con kiến sẽ để lại một chất hóa học trên đường gọi là vết mùi (*pheromone*) để đánh dấu đường đã đi. Các kiến sẽ cảm nhận vết mùi để tìm được đường đi từ tổ tới nguồn thức ăn mà các con kiến khác đã khám phá bằng cách chọn ngẫu nhiên có định hướng thông qua nồng độ vết mùi. Các kiến chịu ảnh hưởng từ các vết mùi của các con kiến khác để lại chính là ý tưởng dùng để thiết kế thuật toán ACO.



Hình 2. 1 Ví dụ về hoạt động của đàn kiến trong thực tế.

Hình 2.1 biểu diễn hành trình của đàn kiến trong thực tế. Trong (hình a) đàn kiến thực hiện di chuyển từ tổ E đến nơi chứa thức ăn A và ngược lại. Khi gặp cản trở xuất hiện trên đường tại vị trí B hoặc D (hình b), đàn kiến phải quyết định rẽ trái hoặc rẽ phải. Cách lựa chọn đường đi có ảnh hưởng từ mùi hương thu hút trên tuyến đường do các con kiến khác để lại.

Nếu mùi hương thu hút từ bên phía phải lớn hơn phía trái sẽ tạo cho các kiến một động lực mạnh mẽ hơn và theo đó xác suất rẽ phải sẽ lớn hơn rẽ trái. Ban đầu, con kiến đầu đàn đi đến điểm B (hoặc D) sẽ có xác suất rẽ phải như rẽ trái (do chưa có mùi hương thu hút trước đó trên hai tuyến đường này). Do tuyến đường B – C – D ngắn hơn tuyến đường B – H – D nên con kiến đầu đàn đi theo B – C – D sẽ đến điểm D trước con kiến đầu đàn đi theo B – H – D (hình c).

Kết quả cho thấy con kiến đi từ E đến D sẽ cảm nhận thấy mùi hương trên tuyến đường D – C – B khiến cho các con kiến khác quyết định đi theo tuyến đường D – C – B – A và theo thời gian tuyến đường D – C – B sẽ được nhiều kiến lựa chọn hơn so với tuyến đường B – H – D.

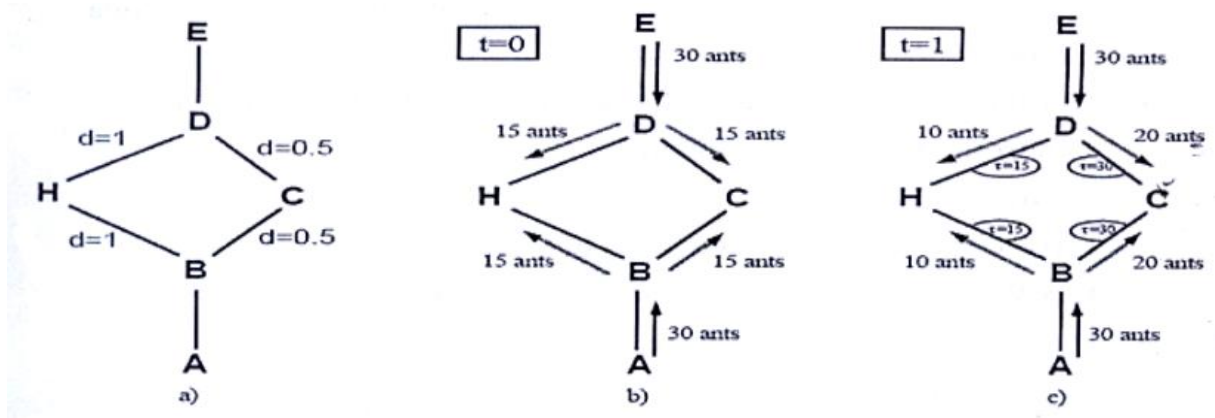
Có thể thấy rằng mùi hương thu hút ở tuyến đường ngắn hơn lớn hơn so với tuyến đường dài sẽ thúc đẩy các kiến chọn tuyến đường ngắn hơn để đi là nhiều hơn.

### 2.2.2. Đàn kiến nhân tạo

Từ thực nghiệm ở trên ta có thể thấy rằng đàn kiến tự nhiên có thể tìm đường đi ngắn nhất giữa hai vị trí nhờ sử dụng quy luật di chuyển theo xác suất của thông tin địa phương. Vết mùi để lại trên đường đi của kiến cho ta liên tưởng đến cách học tăng cường trong các bài toán chọn tác động tối ưu, mô phỏng bài toán tìm đường đi ngắn nhất giữa hai điểm trên đồ thị.

Trong các bài toán ứng dụng thực tế, từ mỗi đỉnh có thể có nhiều cạnh nên nhiều con kiến tự nhiên sẽ bị đi luân quẩn và kém hiệu quả nên người ta thường dùng đàn kiến nhân tạo. Mỗi kiến nhân tạo có nhiều khả năng hơn kiến tự nhiên. Dưới đây sẽ trình bày một vài đặc điểm của kiến nhân tạo:

- Kiến nhân tạo có bộ nhớ riêng/ ký ức nhất định nên có khả năng ghi nhớ các đỉnh đã thăm trong hành trình và tính toán được độ dài đường đi nó chọn.
- Kiến không hoàn toàn mù, chúng có một vài tri thức nhất định nên có thể quan sát, đánh giá các thay đổi của môi trường.
- Kiến nhân tạo được sống trong môi trường có miền thời gian là rời rạc.



Hình 2. 2 Ví dụ về hoạt động của đàn kiến nhân tạo.

Ý tưởng về đàn kiến nhân tạo được nêu ra là nếu tại một điểm bất kỳ, một con kiến nhân tạo sẽ thực hiện chọn điểm tiếp theo để đi từ tập các điểm thuộc các đường đi khác nhau và điểm được chọn là điểm có thể dẫn đến đường đi là ngắn nhất.

### 2.3. Trình bày giải thuật

Khi áp dụng thuật toán ACO vào các bài toán thực tế, có bốn yếu tố quan trọng quyết định hiệu quả của thuật toán:

- 1) *Xây dựng đồ thị cấu trúc*: Phụ thuộc đặc điểm của mỗi bài toán cụ thể.



- 2) *Xây dựng lời giải tuần tự*: Phụ thuộc đặc điểm của mỗi bài toán cụ thể.
- 3) *Xác định thông tin heuristic*: có thể có hoặc không, thông tin heuristic là thông tin kinh nghiệm giúp tăng hiệu quả của thuật toán.
- 4) *Chọn quy tắc cập nhật mùi*: thể hiện chiến lược học của thuật toán. Dùng để phân biệt các thuật toán ACO.

### 2.3.1. Đồ thị cấu trúc

*Xét bài toán tối ưu tổ hợp tổng quát*

Mỗi bài toán tối ưu tổ hợp tổng quát được biểu diễn dưới dạng bài toán cực tiểu hóa với một bộ ba  $(S, f, \Omega)$ , trong đó  $S$  là tập hữu hạn các trạng thái (lời giải tiềm năng hay phương án),  $f$  là hàm mục tiêu xác định trên  $S$  còn  $\Omega$  là tập các ràng buộc để xác định  $S$  qua các thành phần của tập hữu hạn  $C$  và các liên kết của tập này. Mỗi phương án  $s \in S$  thỏa mãn các ràng buộc  $\Omega$  gọi là phương án chấp nhận được. Mục tiêu của chúng là tìm ra phương án  $s^*$  tối ưu hóa toàn cục đối với hàm mục tiêu  $f$ , nói cách khác chính là tìm phương án  $s^*$  sao cho  $f(s^*) \leq f(s)$  với mọi  $s \in S$ . Đối với bài toán này ta có 3 cách giải quyết đó là: vét cạn, kỹ thuật ăn tham hoặc phương pháp tối ưu trong lĩnh vực NP-khó.

Đặc tính của  $C, S, \Omega$  như sau:

- 1) Ký hiệu  $X$  là tập các vector trong  $C$  độ dài không quá  $h$ :  $X = \{ \langle u_0, \dots, u_k \rangle : u_i \in C, \forall i \leq k \leq h \}$ . Khi đó, mỗi phương án  $s$  trong  $S$  được xác định bởi ít nhất một vector trong  $X$  như ở điểm 2.
- 2) Tồn tại tập con  $X^*$  của  $X$  và ánh xạ  $\varphi$  từ  $X^*$  lên  $S$  sao cho  $\varphi^{-1}(s)$  không rỗng với  $\forall s \in S$ , trong đó tập  $X^*$  có thể được xây dựng từ tập con  $C_0$  nào đó của  $C$  nhờ mở rộng tuần tự như ở điểm 3 dưới đây.
- 3) Từ  $C_0$  ta mở rộng tuần tự thành  $X^*$  theo thủ tục tuần tự sau:

i) Gọi  $x_0 = \langle u_0 \rangle$  là mở rộng được với  $\forall u_0 \in C_0$ .

ii) Giả sử  $x_k = \langle u_0, \dots, u_k \rangle$  là mở rộng được từ các ràng buộc  $\Omega$  và chưa thuộc vào  $X^*$ . Từ tập ràng buộc  $\Omega$ , xác định tập con  $J(x_k)$  của  $C$ , sao cho  $\forall u_{k+1} \in J(x_k)$  thì  $x_{k+1} = \langle u_0, \dots, u_k, u_{k+1} \rangle$  là mở rộng được hoặc  $x_k \in X^*$  khi  $J(x_k)$  là rỗng.

iii)  $\forall u_0 \in C_0$  thủ tục mở rộng nêu trên cho phép ta xây dựng được mọi phần tử của  $X^*$ .

*Xây dựng đồ thị cấu trúc*

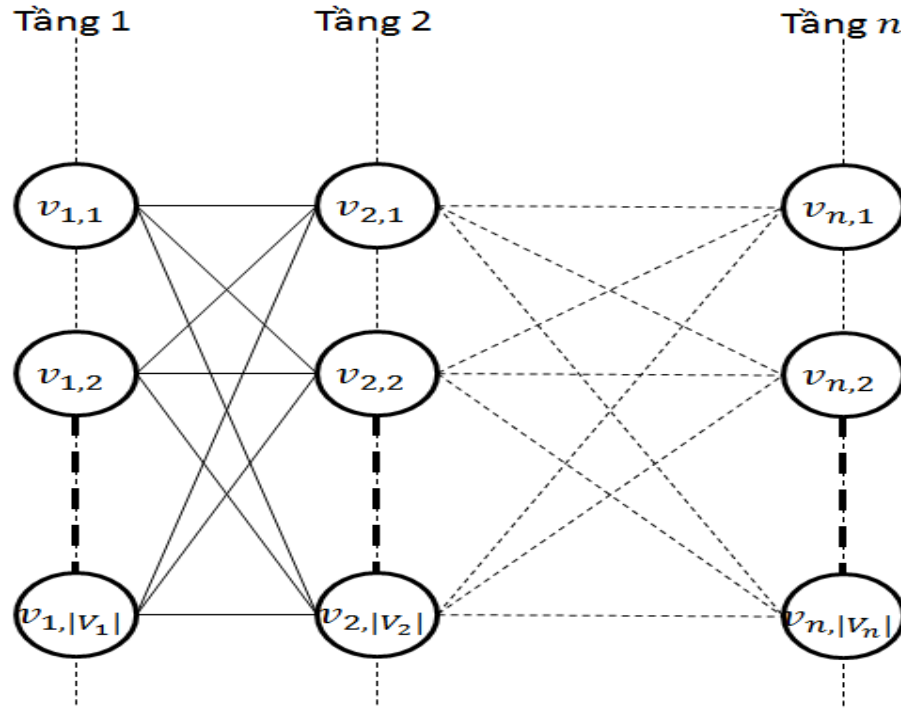
Mỗi bài toán TUTH được xem như một bài toán tìm kiếm vector độ dài không quá  $h$  trên đồ thị đầy đủ có các đỉnh được gán nhãn trong tập  $C$ . Để tìm các lời giải chấp nhận được, ta xây dựng đồ thị đầy đủ với tập đỉnh  $V$  sao cho mỗi đỉnh của nó tương ứng với mỗi

thành phần của  $C$ . Các lời giải chấp nhận được sẽ là các vector được xác định theo thủ tục tuần tự hay bước ngẫu nhiên.

Thông thường, trong các bài toán thuộc lớp NP-khó, người ta đưa ra các phương pháp heuristic để tìm lời giải đủ tốt cho bài toán. Sau đó, các thuật toán ACO sẽ kết hợp thông tin heuristic này với phương pháp học tăng cường dựa vào mô phỏng hành vi của đàn kiến, để tìm được lời giải tốt hơn.

Ta xét đồ thị  $G = (V, E, H, \tau)$  là đồ thị cấu trúc của bài toán tối ưu tổ hợp, trong đó  $V$  là tập đỉnh,  $E$  là tập các cạnh,  $H$  là vector các trọng số heuristic của cạnh và  $\tau$  là vector biểu thị các thông tin học tăng cường  $\tau_{i,j}$ . Từ các cạnh ta xây dựng tập  $X^*$  nhờ mở rộng tập  $C_0$  theo thủ tục tuần tự. Nếu không có thông tin heuristics thì ta xem  $H$  có các thành phần như nhau và bằng 1.

Trường hợp tổng quát,  $G$  là đồ thị đầy đủ. Tuy nhiên, tùy theo ràng buộc của từng bài toán cụ thể, các cạnh có thể loại bớt để giảm miền tìm kiếm lời giải theo thủ tục mở rộng tuần tự. Chẳng hạn, với bài toán tìm cực trị của hàm giải tích  $f(x_1, \dots, x_n)$ , với  $x_i$  thuộc tập giá trị hữu hạn  $V_i$ , đồ thị cấu trúc gồm  $n$  tầng, tầng  $i$  chứa các đỉnh thuộc tập  $V_i$ , còn tập cạnh  $E$  chỉ gồm các cạnh nối các đỉnh thuộc tầng  $i$  với các đỉnh thuộc tầng  $i + 1$  ( $i = 1, 2, \dots, n - 1$ ) như hình 2.3. Khi đó tập  $C_0$  là tập  $V_1$ , mỗi mở rộng tuần tự của lời giải sẽ được xây dựng từ một đỉnh thuộc tập này.



Hình 2. 3 Đồ thị cấu trúc tổng quát cho bài toán cực trị hàm  $f(x_1, \dots, x_n)$ .

### 2.3.2. Trình bày về thuật toán ACO cơ bản

Khi xây dựng lời giải trên đồ thị cấu trúc ta sử dụng  $m$  con kiến. Điều kiện để quá trình tìm kiếm lời giải trên đồ thị kết thúc là dựa vào số bước lặp hoặc giới hạn thời gian chạy cho trước.

#### *Xây dựng lời giải*

Lời giải trên đồ thị cấu trúc  $G = (V, E, H, \tau)$  được xây dựng như sau: Ban đầu, khởi tạo  $m$  con kiến, tại mỗi lần lặp con kiến sẽ chọn ngẫu nhiên một đỉnh để làm khởi tạo ban đầu  $x_0 = \langle u_0 \rangle$  với  $u_0 \in C_0$ . Sau đó các con kiến thực hiện xây dựng lời giải theo thủ tục bước ngẫu nhiên. Dựa vào lời giải đã tìm được đàn kiến sẽ thực hiện cập nhật mùi theo cách học tăng cường.

Từ đỉnh  $u_0$  ta tiến hành mở rộng các đỉnh cho đến khi thuộc vào  $X^*$ , nghĩa quá trình mở rộng tiếp tục cho tới khi tìm được lời giải chấp nhận được. Giả sử con kiến đang ở đỉnh  $i = u_k$  ( $x_k = \langle u_0, \dots, u_k \rangle$ ) và có một đỉnh  $j = u_{k+1}$  ( $u_{k+1} \in J(x_k)$ ) được lựa chọn ngay sau đỉnh  $i$  dựa vào xác suất  $P(j)$  như sau:

$$P(j) = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in J(x_k)} [\tau_{il}]^\alpha [\eta_{il}]^\beta} & j \in J(x_k) \\ 0 & j \notin J(x_k) \end{cases} \quad (2.1)$$

Trong đó :

$\tau_{ij}, \eta_{ij}$ : Giá trị thông tin vết mùi và thông tin heuristic.

$\alpha, \beta$ : Hai tham số quyết định sự ảnh hưởng tương quan giữa thông tin mùi và thông tin heuristic. Nếu  $\alpha = 0$  không có học tăng cường. Nếu  $\beta = 0$  thì không có thông tin heuristic mà chỉ có thông tin học tăng cường biểu thị qua vết mùi được sử dụng.

$l$ : Đỉnh lân cận của đỉnh  $i$  mà kiến có thể đi đến.

#### *Cập nhật mùi*

Dựa trên lời giải tìm được, đàn kiến sẽ thực hiện cập nhật mùi theo cách học tăng cường. Vết mùi để lại trên mỗi cạnh của lời giải tìm được sẽ được điều chỉnh tăng hoặc giảm tùy theo đánh giá mức độ ưu tiên tìm kiếm về sau. Lượng mùi theo mỗi quy tắc cập nhật mùi khác nhau cho ta các thuật toán khác nhau. Nhưng đa số chúng đều có dạng:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \Delta(i, j) \quad (2.2)$$

Trong đó:  $\rho$ : hằng số bay hơi thuộc khoảng (0,1).

$\Delta(i, j)$ : mật độ mùi do kiến để lại

Các bước thực hiện của thuật toán ACO được đặc tả như trong hình 2.4.

**Procedure** Thuật toán ACO;

**Begin**

    Khởi tạo tham số, ma trận mùi, khởi tạo  $m$  con kiến;

**Repeat**

        Mỗi con kiến xây dựng lời giải;

        Cập nhật mùi;

        Cập nhật lời giải tốt nhất;

**Until** (điều kiện kết thúc);

    Đưa ra lời giải tốt nhất;

**End** ;

*Hình 2. 4 Đặc tả thuật toán ACO.*

*Nhận xét chung về các thuật toán ACO*

Các thuật toán ACO có ưu điểm như sau:

1) Nhờ có các thông tin heuristic nên việc tìm kiếm ngẫu nhiên giúp ta có được lời giải tốt hơn và có thể tìm được lời giải tối ưu.

2) Việc học tăng cường thông qua thông tin về nồng độ vết mùi giúp thu hẹp không gian tìm kiếm nhưng vẫn không loại bỏ các lời giải tốt nhờ vậy chất lượng thuật toán được nâng cao.

### 2.3.3. Quy tắc cập nhật vết mùi

Quy tắc cập nhật vết mùi thể hiện chiến lược học của thuật toán, với mỗi quy tắc cập nhật mùi khác nhau cho ta các thuật toán ACO khác nhau. Dưới đây, chúng tôi sẽ giới thiệu một vài quy tắc cập nhật mùi theo trình tự thời gian xuất hiện.

#### 2.3.3.1. Thuật toán AS

Đây là thuật toán ACO đầu tiên được Dorigo đề xuất năm 1991. Ban đầu thuật toán AS có ba phiên bản đề xuất là ant – density, ant – quantity và ant – cycle. Trong đó phiên bản ant-density và ant-quantity, kiến sẽ thực hiện cập nhật vết mùi trực tiếp lên cạnh vừa đi, còn trong phiên bản ant-cycle con kiến cập nhật vết mùi khi tất cả các kiến đã xây dựng xong hành trình và lượng mùi được cập nhật của mỗi kiến dựa vào độ dài hành trình mà kiến tìm được. Thuật toán ant – cycle được xem là hiệu quả hơn so với hai thuật toán ant –

density và ant – quantity nên khi nhắc tới thuật toán AS người ta thường quan tâm đến phiên bản ant – cycle.

Trong thuật toán AS có 2 yếu tố cần quan tâm đó là xây dựng lời giải và cập nhật mùi.

Vết mùi khởi tạo cho tất cả các cạnh:  $\tau_{ij} = \tau_0 = \frac{m}{C^{nn}}$  trong đó  $m$  là số lượng kiến,  $C^{nn}$  độ dài lời giải tìm được của thuật toán heuristic. Nếu khởi tạo vết mùi  $\tau_0$  quá thấp thì việc tìm kiếm có khuynh hướng nhanh chóng hội tụ quanh những hành trình đầu tiên tìm được, dẫn đến việc tìm kiếm hướng vào vùng này và chất lượng lời giải kém, ngược lại nếu khởi tạo vết mùi quá cao thì có thể phải mất nhiều vòng lặp để bay hơi mùi trên các cạnh tồi và thêm mùi cho các cạnh tốt để hướng việc tìm kiếm vào vùng không gian có chất lượng tốt.

#### *Xây dựng lời giải*

Trong AS,  $m$  con kiến đồng thời xây dựng lời giải. Ban đầu các con kiến được đặt ngẫu nhiên tại các thành phố. Tại mỗi bước, kiến sẽ lựa chọn đỉnh đến tiếp theo dựa vào xác suất, gọi là ngẫu nhiên theo tỉ lệ (*random proportional*). Cụ thể, kiến  $k$  đang ở đỉnh  $i$  sẽ lựa chọn đỉnh  $j$  theo xác suất:

$$p_{ij}^k = \begin{cases} \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, & \text{nếu } j \in N_i^k \\ 0 & \text{nếu } j \notin N_i^k \end{cases} \quad (2.3)$$

Trong đó  $\eta_{ij} = \frac{1}{d_{ij}}$  là giá trị heuristic, hai tham số  $\alpha, \beta$  quyết định đến sự ảnh hưởng tương quan giữa thông tin mùi và thông tin heuristic,  $N_i^k$  là các đỉnh lân cận của đỉnh  $i$  mà kiến  $k$  có thể đi đến (là tập các đỉnh mà kiến  $k$  chưa ghé thăm, xác suất lựa chọn các đỉnh không thuộc  $N_i^k$  bằng 0). Bằng sự lựa chọn theo xác suất ngẫu nhiên này, cạnh  $(i, j)$  sẽ có xác suất lựa chọn cao phụ thuộc vào giá trị thông tin vết mùi  $\tau_{ij}$  và thông tin heuristic  $\eta_{ij}$ . Trong đó hai tham số  $\alpha, \beta$  có vai trò như sau: nếu  $\alpha = 0$  thì thành phố gần nhất sẽ được ưu tiên lựa chọn, khi đó thuật toán tương đương với thuật toán chọn ngẫu nhiên theo nghịch đảo độ dài cạnh mà không có học tăng cường. Nếu  $\beta = 0$  thì không có thông tin heuristic mà chỉ có thông tin học tăng cường được biểu thị qua vết mùi được sử dụng. Với  $\alpha > 1$  thì các kiến sẽ có xu hướng lựa chọn đi theo cùng một hành trình dẫn đến thuật toán nhanh chóng bị tắc nghẽn và lời giải tìm được hội tụ về lời giải tối ưu địa phương.

Để cài đặt thuật toán này, mỗi con kiến  $k$  sẽ có một bộ nhớ  $M^k$  chứa thông tin các thành phố mà con kiến đã đi qua. Thông tin trong bộ nhớ dùng để xác định các thành phố

lân cận phù hợp  $N_i^k$  và giúp cho các kiến tính được độ dài hành trình  $T^k$  cũng như dùng để xác định các cạnh được cập nhật vết mùi.

Để xây dựng lời giải có hai cách thực hiện đó là xây dựng lời giải song song và xây dựng tuần tự. Trong cách xây dựng song song, tại mỗi bước tất cả các con kiến sẽ di chuyển sang đỉnh tiếp theo, còn cách xây dựng tuần tự là lần lượt từng kiến xây dựng lời giải (con kiến này xây dựng xong mới đến con kiến tiếp theo). Lưu ý rằng trong AS, hai cách xây dựng này là như nhau vì không ảnh hưởng gì đến thuật toán nhưng điều này không đúng với thuật toán ACS.

#### *Cập nhật mùi*

Vết mùi được cập nhật ngay sau khi các kiến xây dựng xong hành trình. Ban đầu, tất cả các cạnh sẽ bị mất đi một lượng mùi do bay hơi, sau đó các cạnh có con kiến đi qua sẽ được thêm một lượng mùi. Việc bay hơi mùi trên các cạnh được thực hiện như sau:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} \quad (2.4)$$

Trong đó  $0 < \rho \leq 1$  là hệ số bay hơi. Tham số  $\rho$  được sử dụng để tránh sự tích tụ vết mùi quá nhiều trên một cạnh và giúp cho con kiến “quên” đi các quyết định sai lầm. Trong thực tế, nếu một cạnh không được con kiến lựa chọn thì vết mùi nhanh chóng bị giảm theo cấp số nhân. Sau khi bay hơi, tất cả các con kiến sẽ để lại vết mùi mà nó đi qua:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^m \Delta\tau_{ij}^k \quad (2.5)$$

Trong đó  $\Delta\tau_{ij}^k$  là lượng mùi do con kiến  $k$  cập nhật trên cạnh mà nó đi qua. Giá trị này được tính bằng:

$$\Delta\tau_{ij}^k = \begin{cases} \frac{1}{C^k} & \text{nếu cạnh } (i, j) \text{ thuộc } T^k \\ 0 & \text{ngược lại} \end{cases} \quad (2.6)$$

Trong đó:  $C^k$  là độ dài hành trình  $T^k$  do con kiến  $k$  xây dựng, giá trị này được tính bằng tổng độ dài các cạnh thuộc hành trình. Theo công thức (2.6), các cạnh thuộc hành trình tốt hơn sẽ được cập nhật nhiều hơn. Vì vậy, cạnh nào càng có nhiều kiến sử dụng và là cạnh thuộc hành trình ngắn sẽ càng được cập nhật vết mùi nhiều hơn và được các con kiến lựa chọn nhiều hơn trong các vòng lặp sau.

### 2.3.3.2. Thuật toán ACS

Thuật toán ACS được đề xuất bởi Dorigo & Gambardella vào năm 1997 [9], ASC khác với AS ở ba điểm chính sau:

- Thứ nhất, ACS sử dụng quy tắc lựa chọn dựa trên thông tin tích lũy nhiều hơn AS dẫn đến việc khai thác kinh nghiệm tìm kiếm của ACS mạnh hơn AS.
- Thứ hai, trong ACS chỉ các cạnh thuộc lời giải tốt đến thời điểm hiện tại (G-best) mới bị bay hơi mùi và để lại mùi.
- Thứ ba, mỗi lần con kiến đi qua cạnh  $(i, j)$  thì vết mùi sẽ bị giảm trên cạnh  $(i, j)$  để tăng cường thăm dò đường đi mới.

#### *Xây dựng lời giải*

Trong ACS, khi con kiến  $k$  đang đứng ở đỉnh  $i$  nó lựa chọn di chuyển đến đỉnh  $j$  theo qui tắc sau:

$$j = \begin{cases} \operatorname{argmax}_{l \in N_i^k} \{\tau_{il} [\eta_{il}]^\beta\}, & \text{nếu } q \leq q_0 \\ J, & \text{ngược lại} \end{cases} \quad (2.7)$$

trong đó  $q$  là một biến ngẫu nhiên phân bố đều trong  $[0,1]$ ,  $q_0$  ( $0 \leq q_0 \leq 1$ ) là một tham số cho trước và  $J$  là một biến ngẫu nhiên lựa chọn theo phân bố xác suất như trong (2.3) với  $\alpha = 1$ . Nghĩa là, với xác suất  $q_0$  con kiến lựa chọn khả năng tốt nhất có thể dựa trên kết hợp của thông tin học từ vết mùi và thông tin, với xác suất  $(1 - q_0)$  con kiến thực hiện khám phá trên các cạnh. Điều chỉnh tham số  $q_0$  cho phép thay đổi mức độ khai thác và lựa chọn tập trung tìm kiếm quanh lời giải G- best (Global- best) hoặc khám phá các hành trình khác.

#### *Cập nhật mùi toàn cục*

Trong thuật toán ACS sau mỗi bước lặp chỉ có con kiến tìm được lời giải tốt nhất (Global- best) được phép để lại vết mùi. Cụ thể, việc cập nhật mùi trên các cạnh  $(i, j)$  thuộc Global- best trong ACS được thực hiện như sau:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta\tau_{ij}^{best} \quad (2.8)$$

trong đó  $\Delta\tau_{ij}^{best} = \frac{1}{C^{G-best}}$ ,  $C^{G-best}$  là độ dài lời giải tốt nhất,  $T^{G-best}$  là tập các cạnh thuộc lời giải tốt nhất. Trong thuật toán ACS vết mùi chỉ được cập nhật ở các cạnh thuộc  $T^{G-best}$ , bao gồm cả bay hơi và để lại mùi chứ không phải cập nhật cho tất cả các cạnh như trong thuật toán AS.

#### *Cập nhật mùi cục bộ*

Không chỉ cập nhật mùi toàn cục, thuật toán ACS còn sử dụng cập nhật mùi cục bộ. Việc cập nhật mùi cục bộ được thực hiện khi cạnh  $(i, j)$  có con kiến đi qua theo công thức:

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi\tau_0 \quad (2.9)$$

trong đó  $\xi (0 < \xi < 1)$  và  $\tau_0$  là hai tham số. Giá trị  $\tau_0$  là giá trị vết mùi ban đầu dùng để khởi tạo mùi cho tất cả các cạnh. Theo thực nghiệm, giá trị tốt cho  $\xi$  bằng 0.1, giá trị  $\tau_0$  là  $\frac{1}{nC^{nn}}$ , trong đó  $n$  là số thành phố,  $C^{nn}$  là độ dài hành trình theo thuật toán heuristic ăn tham. Hiệu quả của thuật toán cập nhật mùi cục bộ là mỗi khi kiến đi qua cạnh  $(i, j)$  thì vết mùi trên cạnh này bị giảm làm cho các kiến có xu hướng ít lựa chọn lại cạnh này. Hay nói cách khác, việc cập nhật mùi cục bộ giúp tăng cường khám phá các cạnh mà kiến chưa sử dụng. Trong thực tế, hiệu quả của cách cập nhật vết mùi này là làm cho thuật toán không bị tắc nghẽn, nghĩa là các con kiến không bị tập trung vào cùng một tuyến đường giống như AS.

Cần chú ý rằng, với thuật toán AS thì việc các con kiến xây dựng hành trình song song hay tuần tự là không ảnh hưởng gì, nhưng trong thuật toán ACS thì lại có ảnh hưởng vì thuật toán ACS có sử dụng cập nhật mùi cục bộ.

Thuật toán ACS là thuật toán ACO đầu tiên sử dụng danh sách ứng cử viên để hạn chế số lượng lựa chọn trong quá trình xây dựng lời giải. Danh sách ứng cử viên được lựa chọn dựa vào một số tiêu chí heuristic. Ví dụ như trong bài toán TSP, danh sách ứng cử viên cho mỗi thành phố  $i$  là các thành phố  $j$  gần nhất với  $i$ . Danh sách ứng cử viên có thể được xây dựng trước khi thực hiện tìm kiếm và sẽ được giữ cố định trong suốt quá trình tìm kiếm. Khi con kiến đang ở đỉnh  $i$  nó sẽ lựa chọn đỉnh  $j$  bước đến tiếp theo trong số các ứng cử viên chưa được thăm, trong trường hợp tất cả các điểm trong danh sách ứng cử viên đều được thăm thì sẽ chọn một điểm chưa được thăm ngoài danh sách. Trong bài toán TSP, kết quả thực nghiệm cho thấy hiệu quả của việc sử dụng danh sách ứng cử viên giúp tăng chất lượng lời giải và làm giảm độ phức tạp.

### 2.3.3.3. Thuật toán Max-Min

Thuật toán Max-Min được ký hiệu là MMAS do Stutzle và Hoos đề xuất năm 2000 [9] với bốn điểm thay đổi so với AS như sau:

- Thứ nhất, để tăng cường khai thác lời giải tốt nhất tìm được: chỉ con kiến có lời giải tốt nhất tìm được trong bước lặp hiện tại (*Iteration - best*) hoặc lời giải tốt nhất cho đến thời điểm hiện tại (*Global-best*) được cập nhật mùi. Điều này có thể dẫn đến tắc nghẽn vì tất cả các kiến sẽ đi cùng một hành trình do lượng mùi ở các cạnh thuộc hành trình tốt được cập nhật quá nhiều mà hành trình này không phải là hành trình tối ưu.



- Thứ hai, MMAS có miền giới hạn cho vết mùi thuộc  $[\tau_{min}, \tau_{max}]$ .
- Thứ ba, ban đầu vết mùi trên các cạnh được khởi tạo bằng  $\tau_{max}$  và hệ số bay hơi là nhỏ giúp tăng cường khám phá trong giai đoạn đầu.
- Cuối cùng, khi tắc nghẽn hoặc không tìm được lời giải tốt hơn sau một số bước lặp thì vết mùi sẽ được khởi tạo lại.

### Cập nhật mùi

Sau khi các con kiến xây dựng xong lời giải, vết mùi sẽ được cập nhật bằng thủ tục bay hơi giống như AS (công thức 2.3), sau đó thêm một lượng mùi cho tất cả các cạnh thuộc lời giải tốt như sau:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best} \quad (2.10)$$

trong đó  $\Delta\tau_{ij}^{best} = \frac{1}{c_{G-best}}$  khi thêm mùi ở Global – best (G-best) hoặc  $\Delta\tau_{ij}^{best} = \frac{1}{c_{I-best}}$  khi thêm mùi ở Iteration – best (I-best) và vết mùi được giới hạn trong khoảng  $[\tau_{min}, \tau_{max}]$  như sau:

$$\tau_{i,j} = \begin{cases} \tau_{max} & \text{nếu } \tau_{i,j} > \tau_{max} \\ \tau_{i,j} & \text{nếu } \tau_{i,j} \in [\tau_{min}, \tau_{max}] \\ \tau_{min} & \text{nếu } \tau_{i,j} < \tau_{min} \end{cases} \quad \tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best} \quad (2.11)$$

Trong thuật toán MMAS, lời giải I-best và G-best được dùng thay phiên nhau. Nếu luôn cập nhật bằng G-best thì việc tìm kiếm sẽ sớm định hướng quanh  $T^{G-best}$  là tập các cạnh thuộc lời giải tốt nhất, còn khi cập nhật bằng I-best thì số lượng cạnh được cập nhật mùi nhiều dẫn đến việc tìm kiếm giảm tính định hướng hơn.

### Giới hạn vết mùi

Trong MMAS, có sử dụng giới hạn cận trên  $\tau_{max}$  và cận dưới  $\tau_{min}$  của vết mùi trên tất cả các cạnh để tránh tình trạng tắc nghẽn. Hơn nữa, việc giới hạn vết mùi này có ảnh hưởng đến giới hạn xác suất  $p_{ij}$  trong đoạn  $[p_{min}, p_{max}]$  để chọn đỉnh  $j$  tiếp theo khi kiến đang ở đỉnh  $i$ , với  $0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1$ . Chỉ khi  $|N_i^k| = 1$  thì  $p_{min} = p_{max} = 1$ .

Thuật toán MMAS đặt lại cận trên  $\tau_{max}$  bằng  $\frac{1}{\rho c_{G-best}}$  và cận dưới  $\tau_{min} = \frac{\tau_{max}}{a}$  trong đó  $a$  là một tham số khi tìm được lời giải tốt hơn. Kết quả thực nghiệm chỉ ra rằng: để tránh tắc nghẽn cận dưới  $\tau_{min}$  đóng vai trò quan trọng hơn  $\tau_{max}$  nhưng  $\tau_{max}$  lại hữu ích trong việc thiết đặt giá trị vết mùi khi khởi tạo lại.

### *Khởi trị và khởi tạo lại vết mùi*

Ban đầu, vết mùi trên tất cả các cạnh được đặt bằng ước lượng cận trên của vết mùi  $\tau_{max}$ . Kết hợp với tham số bay hơi nhỏ sẽ làm chậm sự khác biệt vết mùi của các cạnh, do vậy giai đoạn đầu của MMAS chỉ mang tính khám phá.

Để tăng cường khả năng khai thác lời giải tốt nhất, MMAS thực hiện khởi tạo lại vết mùi khi gặp tình trạng tắc nghẽn hoặc sau một số bước lặp mà không tìm được lời giải tốt hơn.

Thuật toán MMAS được các nhà nghiên cứu quan tâm nhiều nhất trong các thuật toán ACO và nó có rất nhiều mở rộng. Một trong các cải tiến là khi khởi tạo lại vết mùi, việc cập nhật mùi dựa trên lời giải tốt nhất tìm được tính từ khi khởi tạo lại vết mùi thay vì cố định trong G-best. Ngoài ra, một cải tiến khác đó là sử dụng luật di chuyển theo kiểu ACS.

#### **2.3.3.4. Thuật toán Min- Max trơn**

Thuật toán Max-Min trơn (Smoothed Max-Min Ant System) kí hiệu là SMMAS được Đỗ Đức Đông và Hoàng Xuân Huân đề xuất năm 2012 [2].

SMMAS đơn giản, dễ cài đặt và nó chính là cải tiến của thuật toán MMAS. Hiệu quả nổi trội của nó được kiểm định bằng thực nghiệm thông qua các bài toán chuẩn như: Người bán hàng, lập lịch sản xuất, quy hoạch toàn phương nhị phân không ràng buộc.

Trong SMMAS, thay vì giảm vết mùi ở các cạnh không thuộc lời giải tốt quá nhanh như trong quy tắc MMAS thì SMMAS thực hiện cập nhật vết mùi toàn cục cho tất cả các cạnh như sau:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta_{ij} \quad (2.12)$$

Với:

$$\Delta_{ij} = \begin{cases} \rho\tau_{min} & \text{nếu } (i, j) \notin w(t) \\ \rho\tau_{max} & \text{nếu } (i, j) \in w(t) \end{cases} \quad (2.13)$$

Ký hiệu  $w(t)$  là lời giải tốt nhất tại lần lặp thứ  $t$  mà các con kiến tìm được.

Khi cài đặt chọn  $\tau_0 = \tau_{max}$ .

## 2.4. Một số vấn đề liên quan khi áp dụng ACO

### 2.4.1. Đặc tính hội tụ

Gutjahr [10] là một trong những người đầu tiên khởi đầu việc nghiên cứu đặc tính hội tụ của thuật toán MMAS nhưng không sử dụng thông tin heuristic. Ký hiệu  $P(t)$  là xác suất tìm được lời giải của thuật toán MMAS trong vòng  $t$  phép lặp,  $w(t)$  là lời giải tốt nhất tại bước lặp  $t$ . Nhờ sử dụng mô hình Markov không thuần nhất, Gutjahr đã chứng minh rằng với xác suất bằng 1 ta có :

$$\lim_{t \rightarrow \infty} w(t) = w^*, \lim_{t \rightarrow \infty} P(t) = 1 \quad (2.14)$$

$$\lim_{t \rightarrow \infty} \tau_{i,j} = \tau_{max} \text{ với } \forall(i, j) \text{ thuộc lời giải tối ưu tìm được.} \quad (2.15)$$

Mô hình này của Gutjahr không áp dụng được cho thuật toán ACS. Đối với trường hợp thuật toán MMAS không sử dụng thông tin heuristic, Stützle và Dorigo đã chứng minh rằng:

$$\forall \varepsilon > 0, \text{ tồn tại } t \text{ đủ lớn thì } P(t) > 1 - \varepsilon \quad (2.16)$$

do đó:

$$\lim_{t \rightarrow \infty} P(t) = 1. \quad (2.17)$$

Các tác giả cũng chứng minh rằng kết quả này cũng đúng với thuật toán ACS. Giả thiết rằng đã tìm được lời giải tối ưu sau hữu hạn bước, Stützle và Dorigo suy luận rằng vết mùi của các cạnh thuộc lời giải tối ưu sẽ hội tụ đến  $\tau_{max}$ , còn vết mùi trên các cạnh không thuộc lời giải tối ưu tìm được này sẽ hội tụ về  $\tau_{min}$  hoặc  $\tau_0$ .

Plegrini và Elloro [11] đã chỉ ra rằng sau một thời gian chạy thì đa số vết mùi trên cạnh là bé và chỉ có số ít cạnh có vết mùi là lớn nổi trội.

### 2.4.2. Thực hiện song song

Đặc tính tự nhiên của các thuật toán ACO cho phép chúng có thể thực hiện song song theo dữ liệu hoặc theo quần thể [12]. Trong thực tế, có nhiều mô hình song song đã được áp dụng cho các thuật toán dựa trên quần thể và tương thích với ACO. Phần lớn các chiến lược song song trực tiếp có thể chia thành chiến lược mịn (fine-grained) và chiến lược thô (coarse-grained). Đặc tính của chiến lược mịn là rất ít bộ xử lý được chỉ định để xử lý đơn và việc trao đổi thông tin giữa các bộ xử lý là thường xuyên. Ngược lại, với chiến lược thô thì một lượng lớn gần như tất cả bộ xử lý được chỉ định để xử lý đơn và sự trao đổi thông tin giữa các bộ xử lý là rất ít.

Mô hình song song fine-grained đã được hai nhà nghiên cứu Bolondi & Bondanza áp dụng trong thuật toán AS để giải bài toán TSP trên máy CM-2 kết nối thông qua cách tiếp cận gán một bộ xử lý cho mỗi con kiến. Kết quả thực nghiệm cho thấy rằng việc trao đổi thông tin lớn có thể là trở ngại đối với cách tiếp cận này vì phần lớn thời gian dùng để liên lạc nhằm cập nhật vết mùi.

Đối với mô hình song song coarse-grained, nhiều nhà nghiên cứu (Bolondi & Bondanza; Bullnheimer và cộng sự) cho thấy rằng mô hình này mang đến nhiều hứa hẹn hơn cho ACO. Với trường hợp này,  $p$  đàn kiến chạy song song trên  $p$  bộ vi xử lý.

Stutzle đã thực hiện chạy độc lập song song của nhiều thuật toán ACO. Các kết quả thực nghiệm cho thấy rằng phương pháp này là rất hiệu quả và là cách dễ nhất để thực hiện song song thuật toán ngẫu nhiên.

#### **2.4.3. ACO kết hợp với tìm kiếm cục bộ**

Nhiều tài liệu chỉ ra rằng đối với các phương pháp metaheuristic, việc kết hợp xây dựng lời giải với kỹ thuật tìm kiếm cục bộ là một cách tiếp cận đầy hứa hẹn để thu được lời giải tốt có chất lượng cao.

Mô hình ACO có thể bao gồm cả kỹ thuật tìm kiếm cục bộ. Áp dụng kỹ thuật tìm kiếm cục bộ để có thể thu được lời giải mới tốt hơn lời giải mà trước đó mà kiến đã xây dựng. Việc cập nhật mùi được thực hiện trên tất cả các cạnh thuộc lời giải tối ưu địa phương. Trên thực tế, từ các thực nghiệm cho thấy rằng việc kết hợp tìm kiếm cục bộ cải tiến được lời giải là khá cao bởi vì cách xây dựng lời giải của ACO sử dụng lân cận khác với tìm kiếm cục bộ. Sự kết hợp xây dựng lời giải với tìm kiếm cục bộ sẽ là một cách tiếp cận đầy triển vọng.

#### **2.4.4. Thông tin heuristic**

Chúng ta biết rằng khi thuật toán ACO mà không sử dụng tìm kiếm cục bộ, thông tin heuristic là điều rất cần thiết để có được lời giải tốt. Trong thực tế, ở giai đoạn đầu vết mùi được khởi tạo là như nhau nên khi đó vết mùi không thể giúp kiến nhân tạo tìm đường đi dẫn tới các lời giải tốt vì chưa có sự khác nhau nhiều. Do vậy, thông tin heuristic dùng để khắc phục điều này, nó giúp kiến nhân tạo có thể xây dựng được hành trình tốt ngay từ giai đoạn đầu. Một số trường hợp, nhờ kết hợp tìm kiếm cục bộ nên kiến nhân tạo vẫn có thể tìm được lời giải tốt ngay trong giai đoạn đầu mà không cần sử dụng thông tin heuristic nào cả, mặc dù quá trình tìm kiếm có chậm hơn. Vì vậy, thông tin heuristic có thể không còn quá cần thiết.

#### 2.4.5. Số lượng kiến

Như đã trình bày ở trên, nếu không sử dụng tìm kiếm cục bộ và thông tin heuristic ít (hoặc không có thông tin heuristic) thì trong giai đoạn đầu vết mùi không thể giúp kiến nhân tạo tìm được đường đi dẫn tới các lời giải tốt. Còn trong trường hợp sử dụng số lượng kiến ít thì sẽ không tìm được lời giải tốt trong giai đoạn đầu dẫn đến việc cập nhật vết mùi được cập nhật dựa vào các lời giải không tốt sẽ khiến các kiến thực hiện tìm kiếm xung quanh lời giải không tốt và do đó thuật toán sẽ kém hiệu quả. Để khắc phục phần nào nhược điểm này người ta thực hiện tăng số lượng kiến giúp tăng khả năng tìm được lời giải tốt trong mỗi vòng lặp. Với trường hợp có sử dụng tìm kiếm cục bộ hoặc có thông tin heuristic mạnh việc sử dụng nhiều kiến là có thể là lãng phí. Theo kinh nghiệm của các tác giả khi làm thực nghiệm, nếu có sử dụng tìm kiếm cục bộ hoặc có thông tin heuristic mạnh thì số lượng kiến thường đặt từ 10 đến 30 con kiến, ngược lại trong trường hợp bài toán có kích thước lớn thì số kiến sẽ được đặt nhiều hơn.

#### 2.4.6. Tham số bay hơi

Tại mỗi vòng lặp, kiến có thể xây dựng được lời giải tốt (có sử dụng tìm kiếm cục bộ hoặc thông tin heuristic mạnh) thì tham số bay hơi sẽ được xác lập có giá trị lớn để giúp cho con kiến quên đi những lời giải đã xây dựng và tập chung tìm kiếm quanh lời giải tốt mới được xây dựng. Ngược lại, nếu tại mỗi vòng lặp khả năng con kiến tìm được lời giải tốt không cao thì tham số bay hơi nên được xác lập có giá trị nhỏ.

### 2.5. Kết luận chương.

ACO là một phương pháp metaheuristic đang được sử dụng rộng rãi để giải các bài toán TỰTH phức tạp và hiệu quả nổi trội của chúng đã được chứng minh bằng thực nghiệm. Phương pháp dựa trên mô phỏng cách tìm đường đi của con kiến thực trong tự nhiên. Trong đó, lời giải chấp nhận được của bài toán được các con kiến xây dựng nhờ thủ tục bước ngẫu nhiên trên đồ thị cấu trúc. Việc tìm kiếm đỉnh mới mà kiến sẽ bước đến tiếp theo dựa trên sự kết hợp thông tin heuristic và thông tin học tăng cường biểu thị bởi vết mùi.

Khi áp dụng phương pháp này, có bốn yếu tố quan trọng:

- 1) Xây dựng đồ thị cấu trúc
- 2) Xây dựng lời giải tuần tự
- 3) Xác định thông tin heuristic
- 4) Chọn quy tắc cập nhật mùi

Trong đó, quy tắc cập nhật vết mùi là yếu tố được đề xuất, nghiên cứu cải tiến nhiều nhất và nó cũng được dùng để phân biệt giữa các thuật toán ACO còn các yếu tố còn lại phụ thuộc vào từng bài toán cụ thể.

## CHƯƠNG 3: ỨNG DỤNG THUẬT TOÁN TỐI ƯU ĐÀN KIẾN GIẢI BÀI TOÁN MPDPTW

### 3.1. Phát biểu bài toán

#### 3.1.1. Giới thiệu bài toán

Bài toán MPDPTW chứa  $n$  yêu cầu và  $m$  phương tiện (xe).  $P = \{1, \dots, p\}$  là tập các nút (điểm) đón và  $D = \{p+1, \dots, p+n\}$  là tập các nút (điểm) giao hàng trong đó  $|D| = n$  và  $p \geq n$ .  $R = \{r_1, \dots, r_n\}$  là tập các yêu cầu được định tuyến. Mỗi yêu cầu  $r \in R$  diễn tả một tập các nút đón  $P_r \subseteq P$  và một nút giao hàng  $d_r \in D$ . Mỗi yêu cầu luôn chứa ít nhất một nút đón.  $N = P \cup D$  là tập các điểm khách hàng.  $r_{(i)}$  là yêu cầu tại nút  $i \in N$ .  $K = \{1, \dots, m\}$  là tập phương tiện khả dụng.

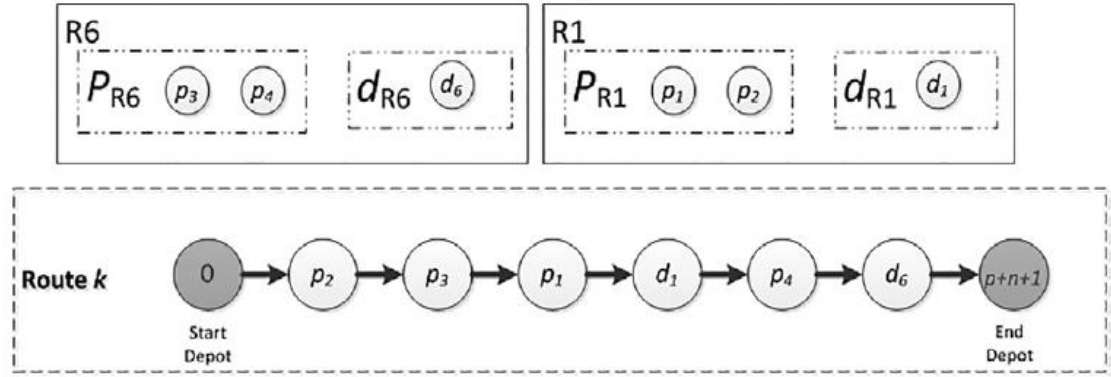
Bài toán MPDPTW được mô hình hóa bởi đồ thị  $G = (V, A)$  bao gồm tập các nút  $V = N \cup \{0, p+n+1\}$  trong đó 0 và  $p+n+1$  là kho bắt đầu và kho kết thúc. Mỗi nút  $i \in V$  có thời gian phục vụ là  $s_i$  và thời gian cửa sổ (timewindows) trong khoảng  $[a_i, b_i]$ . Do có ràng buộc về thời gian nên nếu xe đến nút  $i$  trước thời điểm  $a_i$  thì xe sẽ phải đứng chờ cho đến thời điểm  $a_i$  mới được bắt đầu dịch vụ, ngoài ra xe phải đến nút  $i$  trước thời điểm  $b_i$ , sao cho dịch vụ tại nút  $i$  được thực hiện trong khoảng thời gian cho phép của nó.

Tập các cung là  $A = V \times V$  loại trừ các cung thuộc giải pháp không khả thi: bỏ qua các cung  $(i, j)$  nếu:

- $i$  là một nút đón và  $j$  là nút giao của nó nếu  $b_j < a_i + s_i + t_{ij}$ .
- $i$  là một nút giao và  $j$  là một nút đón của nó.
- $i$  là kho bắt đầu và  $j$  là một nút giao.
- $i$  là một nút đón và  $j$  là kho kết thúc.

Mỗi cung  $(i, j) \in A$  có khoảng cách  $d_{ij} \geq 0$  và thời gian di chuyển từ  $i$  đến  $j$  là  $t_{ij} \geq 0$ .  $A^+(i)$  và  $A^-(i)$  là tập các cung đi và đến từ nút  $i \in V$ .

Giải pháp cho bài toán nhằm tối ưu chi phí định tuyến và phân chia tất cả các yêu cầu cho các xe, đảm bảo rằng một yêu cầu chỉ được phục vụ bởi một xe duy nhất.



Hình 3. 1 Minh họa các yêu cầu R1 và R6 thuộc tuyến đường của xe k.

Hình 3.1 là một ví dụ minh họa gồm hai yêu cầu R1 và R6 và một tuyến đường liên quan đến xe k. Trong yêu cầu R6 có hai nút đón là  $p_3$ ,  $p_4$  và nút giao là  $d_6$ . Yêu cầu R1 có hai nút đón là  $p_1$ ,  $p_2$  và nút giao là  $d_1$ . Tuyến đường phục vụ bởi xe k có một số ràng buộc như: nút  $p_1$  không được phục vụ trước nút  $p_2$  trong cùng một yêu cầu, nút giao  $d_1$  được phục vụ khi  $p_1$  và  $p_2$  đã được duyệt qua bởi xe k. Tương tự, với yêu cầu R6, nút  $p_3$  phải được phục vụ trước nút  $p_4$  trong cùng một yêu cầu, nút giao  $d_6$  được phục vụ khi  $p_3$  và  $p_4$  đã được duyệt qua bởi xe k. Do vậy, tuyến đường phục vụ bởi xe k thỏa mãn yêu cầu R1 và R6 là

$$\boxed{0} \xrightarrow{\text{Start Depot}} p_2 \rightarrow p_3 \rightarrow p_1 \rightarrow d_1 \rightarrow p_4 \rightarrow d_6 \rightarrow \boxed{p+n+1} \xrightarrow{\text{End Depot}}$$

### 3.1.2. Xây dựng mô hình toán học

Bài toán MPDPTW được mô tả chi tiết như sau:

Công thức toán học có các biến quyết định:

$$x_{ij}^k = \begin{cases} 1 & \text{nếu cung } (i,j) \text{ được duyệt qua bởi xe } k \\ 0 & \text{nếu ngược lại} \end{cases}$$

$$y_{rk} = \begin{cases} 1 & \text{nếu yêu cầu } r \text{ thỏa mãn bởi xe } k \\ 0 & \text{nếu ngược lại} \end{cases}$$

$S_i$ : thời gian bắt đầu dịch vụ tại nút  $i \in V$

Hàm mục tiêu có thể được biểu diễn như sau:

$$\text{Min} \sum_{k \in K} \sum_{(i,j) \in A} c_{ij} x_{ij}^k \quad (3.1)$$

sao cho

$$\sum_{j \in A^+(i)} x_{ij}^k = y_{r(i)k} \quad k \in K, i \in N \quad (3.2)$$

$$\sum_{j \in A^-(i)} x_{ji}^k = y_{r(i)k} \quad k \in K, i \in N \quad (3.3)$$

$$\sum_{j \in A^+(0)} x_{0j}^k \leq 1 \quad k \in K \quad (3.4)$$

$$\sum_{k \in K} y_{rk} = 1 \quad r \in R \quad (3.5)$$

$$S_j \geq S_i + (s_i + t_{ij} + M) \sum_{k \in K} x_{ij}^k - M \quad (i,j) \in A \quad (3.6)$$

$$a_i \leq S_i \leq b_i \quad i \in V \quad (3.7)$$

$$S_{d_r} \geq S_i + s_i + t_{id_r} \quad i \in P_r, r \in R \quad (3.8)$$

$$x_{ij}^k, y_{rk} \in \{0,1\} \quad (i,j) \in A, r \in R, k \in K \quad (3.9)$$

$$S_i \geq 0 \quad i \in V \quad (3.10)$$

Hàm mục tiêu (3.1) là để tối thiểu hóa tổng chi phí vận chuyển. Ràng buộc (3.2) và (3.3) là ràng buộc bậc kết hợp bởi mỗi nút được thăm bởi xe  $k$ . Đảm bảo rằng tất cả các nút của một yêu cầu được thăm bởi cùng một xe. Ràng buộc (3.4) đảm bảo rằng có nhiều nhất  $K$  xe được sử dụng trong giải pháp (đảm bảo rằng mỗi xe chỉ được sử dụng đúng một lần). Ràng buộc (3.5) mỗi một yêu cầu  $r$  chỉ được phục vụ bởi duy nhất một xe. Ràng buộc (3.6) và (3.7) đảm bảo tính khả thi về lịch trình với hạn chế về thời gian. Thứ tự ưu tiên được thể



hiện thông qua ràng buộc (3.8). Ràng buộc (3.9) và (3.10) là các điều kiện xác định miền giá trị cho biến.  $M$  là một số đủ lớn và nó bằng  $\max\{b_i + s_i + t_{ij} - a_j, 0\}$  cho ràng buộc (3.7).

Mô hình toán học trên đưa ra lời giải tối ưu cho bài toán MPDPTW. Tuy nhiên, bài toán này được Naccache và các cộng sự trong [4] chứng minh thuộc lớp NP-khó vì vậy rất khó khăn trong việc tìm lời giải chính xác. Do đó chúng tôi lựa chọn thuật toán tối ưu đàn kiến để giải bài toán trong trường hợp kích thước bộ dữ liệu lớn và đưa ra lời giải chấp nhận được.

### 3.2. Một số phương pháp giải quyết bài toán MPDPTW

Trong bài toán định tuyến xe đa điểm đón và giao hàng một yêu cầu phải được đón từ nhiều địa điểm khác nhau và chỉ giao tới một địa điểm nhất định. Các xe có thể thực hiện đón các yêu cầu khác nhau theo bất kỳ trình tự nào, miễn là tất cả các điểm đón của một yêu cầu được thực hiện trước khi giao nó. Bài toán này gần đây đã được giới thiệu trong tài liệu của Naccache và các cộng sự được xuất bản năm 2018[4] trên tạp chí Vận Trù Học và tìm thấy nhiều ứng dụng thực tế, đáng chú ý nhất trong giao thực phẩm: khách hàng gọi cho một công ty và đặt một số món ăn từ các nhà hàng khác nhau. Công ty sau đó phải nhận tất cả các món ăn trước khi giao chúng cho khách hàng về nhà. Rõ ràng, công ty có thể kết hợp các đơn đặt hàng của các khách hàng khác nhau trong cùng một chuyến đi. Một số trong những địa điểm này (có thể là lấy hàng hoặc giao hàng) có thể có thời gian cửa sổ (TWs). Các ví dụ thực tế và các bài toán về giao được tìm thấy nghiên cứu của Coelho và các cộng sự năm 2016 [14].

Công việc duy nhất giải quyết bài toán đa điểm đón và giao hàng với thời gian cửa sổ (MPDPTW) của Naccache cùng cộng sự [4] là họ đã phát triển một thuật toán A Hybrid Adaptive Neighborhood Search (ALNS) với các hoạt động cải tiến và đề xuất một công thức nguyên cho bài toán được sử dụng để giải quyết bài toán thông qua thuật toán nhánh cận. MPDPTW mang nhiều đặc điểm với các bài toán được nghiên cứu trước đây, cụ thể là bài toán đón và giao hàng (*the pickup and delivery problem - PDP*) và bài toán đặt hàng tuần tự (*the sequential ordering problem - SOP*).

Các thuật toán heuristic cho PDP là ALNS [15], tối ưu hóa bầy đàn [17]. Các thuật toán chính xác bao gồm nhánh – cắt (Branch and cut) của Ropke et al. [18], branch – cut – price của Ropke và Cordeau [19] và thuật toán thiết lập dựa vào phân vùng của Baldacci et al. [20]. SOP là một bài toán liên quan đến giải quyết bài toán người giao hàng (TSP) với các ràng buộc ưu tiên, có nghĩa là các nút có một thứ tự áp đặt cho các chuyến thăm của họ [21]. Bài toán này cũng mô hình hóa các ứng dụng thực tế trong sản xuất và vận tải [22]. Một số phương pháp tồn tại để giải quyết bài toán SOP, bao gồm tìm kiếm cục bộ [23],

nhánh - cắt [24], thuật toán tuần tự song song [25], thuật toán di truyền [26] và các công cụ lập trình toán học khác [27].

Phần tiếp theo chúng tôi sẽ trình bày về thuật toán tối ưu đàn kiến và các áp dụng đối với bài toán định tuyến xe có đa điểm đón và giao hàng với thời gian cửa sổ cho bài toán trong trường hợp kích thước dữ liệu đầu vào lớn.

### 3.3. Thuật toán ACO giải quyết bài toán MPDPTW

#### 3.3.1. Đồ thị cấu trúc

Xây dựng đồ thị cấu trúc là bước quan trọng trong việc giải bài toán tối ưu tổ hợp bằng thuật toán ACO.

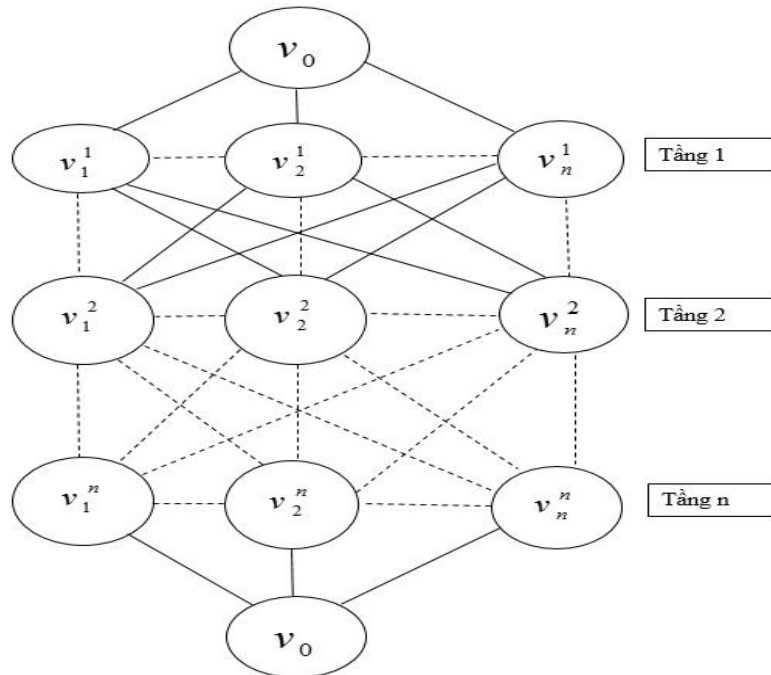
Ta có:

$N = n_1, n_2, \dots, n_n$  là tập các khách hàng (Bao gồm điểm đón và giao hàng)

$M = m_1, m_2, \dots, m_n$  là tập các nhu cầu để di chuyển (Bao gồm dung lượng xe, thời gian cửa sổ, sự kết hợp phải đi đón trước khi giao hàng)

$A$  là tập cạnh nối các điểm thuộc giải pháp khả thi

Đồ thị cấu trúc của bài toán  $G = (N, A)$  được xây dựng như sau:



Hình 3. 2 Đồ thị cấu trúc cho bài toán MPDPTW.

Đồ thị bao gồm n tầng, mỗi tầng  $V_i$  có n điểm như hình 3.2.  $V_0$  là điểm bắt đầu và kết thúc của mỗi lộ trình. Kiến sẽ thực hiện xây dựng lời giải một cách tuần tự. Mỗi kiến xuất

phát từ  $V_0$  đi lần lượt từ tầng 1 xuống tầng  $n$ , tại mỗi tầng sẽ chọn một điểm bất kỳ để đi và lựa chọn dựa trên xác suất được tính dựa theo thông tin heuristic và giá trị vết mùi như trong công thức 3.11. Điểm được chọn sẽ được loại bỏ khỏi tầng tiếp theo của đồ thị. Quá trình này được lặp đi lặp lại cho đến khi kiến đi hết  $n$  tầng của đồ thị. Dưới đây chúng tôi sẽ trình bày cách xây dựng giải pháp cho bài toán này trong phần 3.3.2.

### 3.3.2. Xây dựng giải pháp

Một giải pháp đề xuất cho bài toán này bao gồm một tập các tuyến đường được định tuyến và mỗi tuyến đường được thực bởi một phương tiện. Mỗi con kiến xây dựng các tuyến đường của một giải pháp một cách tuần tự. Kiến tiếp tục chèn các nút vào tuyến đường hiện thời sao cho các nút này không được vi phạm ràng buộc đã đặt ra (dung lượng xe, thời gian hạn chế, sự kết hợp phải đi từ nút đón rồi mới đến nút giao). Nếu giải pháp hiện thời vẫn còn các nút chưa được duyệt qua nhưng chúng không thể được chèn thêm vào tuyến đường đang được xây dựng thì kiến kết thúc tuyến đường hiện tại và bắt đầu tuyến đường mới. Gọi  $\varphi$  là một giải pháp hiện tại chưa hoàn thành và được thực hiện bởi  $r-1$  tuyến đường hoàn thành và một tuyến đường  $r^{th}$  trong xây dựng lời giải. Giả sử  $i$  là nút cuối cùng đã được hoàn thành bởi tuyến đường  $r$ ,  $Q_r$  là trọng tải của xe sau khi đã phục vụ nút  $i$  và  $U$  là tập nút đã phục vụ thỏa mãn nút  $i$  còn nút  $i+1$  chưa được xét đến ( $U$  là tập các ứng cử viên từ  $i$  có thể đi tới). Một nút  $j$  là đủ điều kiện thỏa mãn nếu nó chưa được hoàn thành và thuộc một trong những điều kiện sau:

1.  $0 < j \leq p$  và tồn tại một đường đi thỏa mãn tất cả các nhu cầu trong  $U \cup \{p+j\}$
2.  $p < j \leq (p+n) \wedge j - p \in r$  và tồn tại một đường thỏa mãn tất cả các nhu cầu thuộc  $U \setminus \{j\}$

Có  $\varphi$  là giải pháp chưa hoàn thành,  $i$  là nút cuối cùng thỏa mãn,  $j$  là nút đủ điều kiện và được tìm được từ tuyến đường khả thi. Chứng minh rằng sau khi thực hiện nút  $j$ , tồn tại một tuyến đường khả thi thỏa mãn tất cả các đòi hỏi giao trước đó chưa thỏa mãn trong giải pháp  $\varphi$ . Bài toán là một chu trình Hamilton.

Một nhu cầu  $j$  được thêm vào giải pháp hiện thời  $\varphi$  được chọn một cách ngẫu nhiên sử dụng xác suất:

$$P_{ij}^{\varphi} = \begin{cases} \frac{[\tau_{ij}]^{\alpha} [\eta_{ij}^{\varphi}]^{\beta}}{\sum_{d \in S_i^{\varphi}} [\tau_{id}]^{\alpha} [\eta_{id}^{\varphi}]^{\beta}} & j \in S_i^{\varphi} \\ 0 & j \notin S_i^{\varphi} \end{cases} \quad (3.11)$$

Trong đó:  $P_{ij}^{\varphi}$  biểu diễn xác suất để chọn nút  $j$  để hoàn thành từ nút  $i$  hiện tại.  $\tau_{ij}$  xác định thông tin vết mùi trên cung  $(i,j)$ .  $S_i^{\varphi}$  là tập các nút đủ điều kiện mà chúng tôi thực hiện

sau nút  $i$ . Thông số  $\alpha$  và  $\beta$  điều chỉnh sự quan trọng giữa thông tin vết mùi và thông tin heuristic.  $\eta_{ij}^\varphi$  là thông tin heuristic được sử dụng để hướng dẫn kiến.

$$\eta_{ij}^\varphi = \frac{1}{H_1 + H_2} \quad (3.12)$$

Trong công thức (3.12) chúng tôi sử dụng 2 thông tin heuristic là  $H_1, H_2$ .

Trong đó:

$$H_1 = w_1 * d_{ij} ; w_1 \in (0,1) \quad (3.13)$$

Với  $H_1$  là thông tin heuristic biểu diễn khoảng cách giữa đỉnh  $i$  và  $j$  ( $=d_{ij}$ ).  $w_1$  là một tham số nhận giá trị trong khoảng  $(0,1)$ . Với bài toán này, chúng tôi sử dụng  $w_1 = 0.1$ .

$$H_2 = w_2 * (departureTime - b_i); w_2 \in (0,1) \quad (3.14)$$

$H_2$  là thông tin heuristic biểu diễn sự chênh lệch giữa  $departureTime$  là tổng thời gian phục vụ tính đến thời điểm hiện tại tại nút  $j$  và thời điểm kết thúc tại nút  $i$  hay nói cách khác,  $H_2$  là khoảng thời gian mà kiến phải đợi để có thể phục vụ đỉnh  $j$  sau khi đã đến  $i$ . Với  $w_2$  là tham số tương tự  $w_1$  và chúng tôi sử dụng  $w_2 = 0.6$ .

**Sơ đồ thuật toán của việc xây dựng giải pháp như sau:**

**Procedure** Giải pháp tìm đường đi của bài toán MPDPTW

**Begin**

**Input:**  $U$ : tập nhu cầu thỏa mãn,  $ItMax1$ : số vòng lặp,  $i$ : nhu cầu hiện tại,  $t$ : thời điểm kết thúc nhu cầu  $i$

**While** lần lặp hiện tại  $\leq ItMax1$  và không có tuyến đường khả thi thỏa mãn các nhu cầu trong  $U$  **do**

$U' = U$

Khởi tạo 1 tuyến đường  $r$  bắt đầu từ nút  $i$  tại thời điểm  $t$

**While**  $U' \neq \emptyset$  **do**

Chọn ngẫu nhiên nút  $d \in U'$  thỏa mãn nhu cầu, loại bỏ  $d$  ra khỏi  $U'$  và thêm nó vào  $r$  tại vị trí tốt nhất

**If**  $d$  không được thêm vào  $r$  **then**

quay lại bước  $U'=U$

**End if**

**End while**

**End while**

**End;**

### 3.3.3. Quy tắc cập nhật mùi

Sau khi lời giải của con kiến tốt nhất được tìm thấy chúng tôi áp dụng thủ tục tìm kiếm cục bộ để tăng chất lượng lời giải. Lời giải tốt nhất này được sử dụng để cập nhật vết mùi trên các cạnh theo thuật toán SMMAS (Max-Min tron) được Đỗ Đức Đông và Hoàng Xuân Huân đề xuất năm 2012. [2]

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij} + \rho\Delta_{ij} \quad (3.15)$$

Với:

$$\Delta_{ij} = \begin{cases} \rho\tau_{min} & \text{nếu } (i,j) \notin \text{lời giải tối nhất} \\ \rho\tau_{max} & \text{nếu } (i,j) \in \text{lời giải tối nhất} \end{cases} \quad (3.16)$$

Chọn  $\tau_0 = \tau_{max}$ .

$\tau_{max}, \tau_{min}$ : giới hạn vết mùi trên tất cả các cạnh

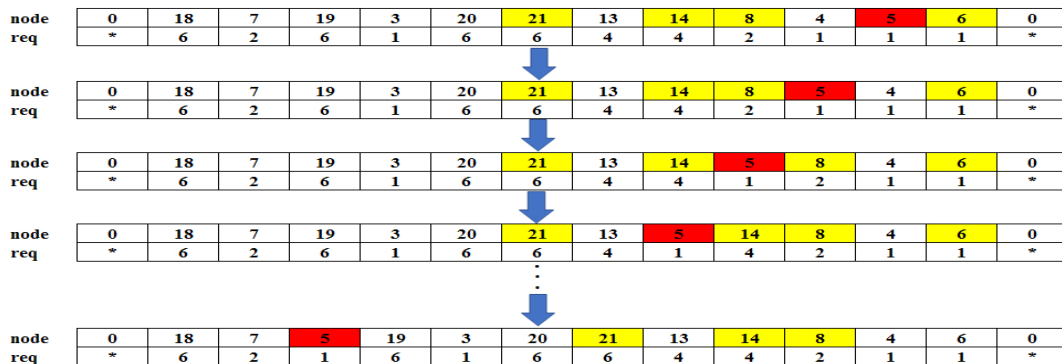
$\rho$ : hệ số bay hơi, là hằng số thuộc khoảng (0,1)

$\Delta_{ij}$ : nồng độ vết mùi do kiến để lại trên cạnh (i,j)

### 3.3.4. Thủ tục tìm kiếm cục bộ

Để tăng cường chất lượng cho lời giải tối ưu ta có thủ tục tìm kiếm cục bộ. Chúng tôi đề xuất hai thuật toán heuristic là T1 và T2 được áp dụng cho bài toán MPDPTW. Trong đó:

T1: được xây dựng dựa trên thuật toán tham lam nhằm giảm khoảng cách di chuyển cho tuyến đường đã định tuyến. T1 được thực hiện bằng cách đổi ngẫu nhiên vị trí của một số nút đón trong cùng một tuyến đường nhằm mục đích tạo ra tuyến đường mới tốt hơn (nếu có). Hình 3.3. là một ví dụ cho giải thuật T1, trong đó đỉnh đón 5 được lần lượt thử chèn vào trước các đỉnh 4, 8, 14, 19.



Hình 3. 3 Mô tả thuật toán T1

T2 được xây dựng tương tự như T1 nhưng thay vì đổi vị trí các nút đón trong một tuyến đường, giải thuật T2 sẽ thay đổi các request thuộc router  $i$  cho router  $j$  bất kỳ trong lời giải tại vị trí tốt nhất sao cho tổng chi phí là tối thiểu. Trong hình 3.4. chúng ta có thể thấy phương án thứ 2 đã được xây dựng bằng cách đổi vị trí request 7 (bao gồm các nút 23, 22, 24) từ router 2 sang router 3. Và phương án 3 được xây dựng bằng cách đổi vị trí request 5 (bao gồm các nút 15, 16, 17) từ router 2 sang router 3.

node	0	18	7	5	19	3	20	21	13	14	8	4	6	0	R1 = 6-4-2-1
req	*	6	2	1	6	1	6	6	4	4	2	1	1	*	
node	0	23	9	22	24	10	11	15	25	16	17	12	0		R2 = 7-5-3
req	*	7	3	7	7	3	3	5	7	5	5	3	*		
node	0	1	2	0											R3 = 0
req	*	0	0	*											

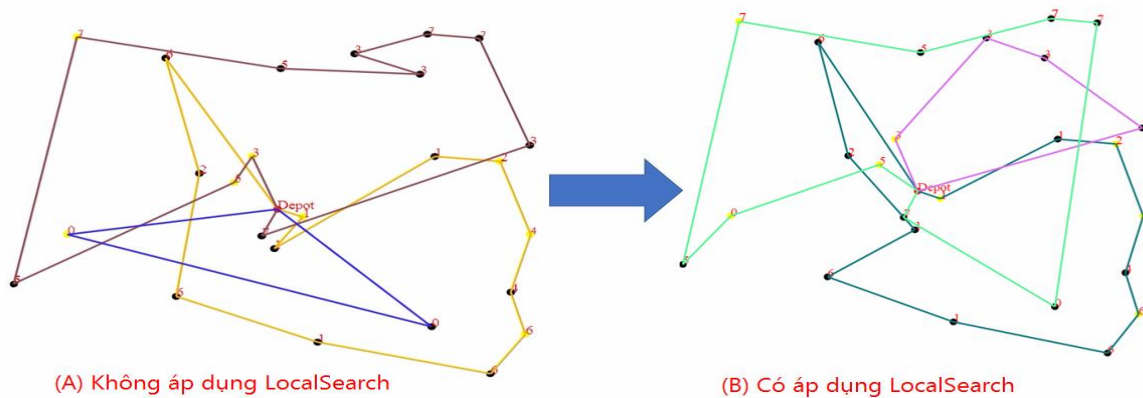
  

node	0	18	7	5	19	3	20	21	13	14	8	4	6	0	R1 = 6-4-2-1
req	*	6	2	1	6	1	6	6	4	4	2	1	1	*	
node	0	9	10	11	15	16	17	12	0						R2 = 5-3
req	*	3	3	3	5	5	5	3	*						
node	0	23	22	24	1	25	2	0							R3 = 7-0
req	*	7	7	7	0	7	0	*							

node	0	18	7	5	19	3	20	21	13	14	8	4	6	0	R1 = 6-4-2-1
req	*	6	2	1	6	1	6	6	4	4	2	1	1	*	
node	0	9	10	11	12	0									R2 = 3
req	*	3	3	3	3	*									
node	0	23	22	24	1	25	15	16	2	17	0				R3 = 7-0-5
req	*	7	7	7	0	7	5	5	0	5	*				

Hình 3. 4 Mô tả thuật toán T2



Hình 3. 5 Mô tả quá trình tìm kiếm cục bộ

### Sơ đồ thuật toán ACO giải bài toán MPDPTW

#### Procedure Thuật toán ACO giải bài toán MPDPTW

##### Begin

Khởi tạo tham số, ma trận vết mùi, khởi tạo m con kiến

**While** Giải pháp tốt nhất trong vòng lặp chưa được cải thiện **do**

**For**  $k = 1$  to  $m$  **do**

        Kiến  $k$  xây dựng một giải pháp thỏa mãn tất cả các nhu cầu

**While** Giải pháp tốt nhất có thể giảm tổng khoảng cách di chuyển **do**

            Áp dụng thuật toán T1 và T2 tương ứng

**End while**

        Cập nhật vết mùi

**End for**

    Cập nhật lời giải

**End while**

Đưa ra lời giải tốt nhất của thuật toán

**End;**

### 3.4. Kết luận chương

Trong chương này chúng tôi đã giới thiệu về bài toán định tuyến xe đa điểm đón và giao với hạn chế thời gian: phát biểu bài toán, xây dựng mô hình toán học cho bài toán và giới thiệu một số phương pháp giải bài toán trước đây. Sau đó, chúng tôi sử dụng thuật toán tối ưu đàn kiến để đưa ra giải pháp tìm lời giải cho bài toán MPDPTW.

## CHƯƠNG 4: CÀI ĐẶT VÀ ĐÁNH GIÁ THỰC NGHIỆM

Chương này chúng tôi trình bày về cách thức cài đặt và đánh giá thực nghiệm của giải pháp đã được đưa ra ở trên. Mục 4.1 giới thiệu một số cài đặt chính trong chương trình. Mục 4.2 mô tả dữ liệu thực nghiệm. Cuối cùng, đánh giá hiệu năng của mô hình được đưa ra trong mục 4.3.

### 4.1. Cài đặt chương trình

Chúng tôi sử dụng ngôn ngữ lập trình Java 8 để cài đặt thực nghiệm cho bài toán. Dưới đây là class chính mà chúng tôi đã sử dụng để cài đặt cho bài toán là class Slover.java được kế thừa từ các class là SMMAS.java dùng để xây dựng giải pháp và cập nhật mùi, class LocalSearch.java thực hiện tìm kiếm cục bộ để cải thiện lời giải của con kiến tốt nhất trong mỗi vòng lặp.

#### Class Slover.java

```
public void run() {
    // Khởi tạo tham số
    globalStatistics.startTimer();
    initProblemInstance();
    smmas.setRho(rho);
    smmas.setAlpha(1.0);
    smmas.setBeta(2.0);
    smmas.setnAnts(100);
    smmas.setDepth(problemInstance.noNodes);
    smmas.allocateAnts();
    smmas.allocateStructures();
    smmas.setRandom(new Random(seed));
    smmas.computeNNList();
    smmas.initTry();
    globalStatistics.endTimer("SMMAS Initialization");
```

#### // Khởi tạo LocalSearch

```
this.localSearch = new LocalSearch(problemInstance, new Random(seed));
```



**// Thực hiện thuật toán SMMAS**

```

globalStatistics.startTimer();
smmas.getBestSoFar().feasible = false;
for (int i = 1; i <= maxIterations; i++) {
    IterationStatistic iterationStatistic = new IterationStatistic();
    smmas.setCurrentIteration(i);
    // Xây dựng giải pháp
    iterationStatistic.startTimer();
    smmas.constructSolutions();
    iterationStatistic.endTimer("Construction");
    // Thực hiện LocalSearch lời giải tốt nhất
    executeLocalSearch();
    boolean hasBest = smmas.updateBestSoFar();
    if (hasBest) {
        smmas.setPheromoneBoundsForLS();
    }
    smmas.updateRestartBest();
    iterationStatistic.endTimer("Daemon");
    // Cập nhật thông tin vết mùi
    iterationStatistic.startTimer();
    smmas.evaporation();// tat ca cac canh
    smmas.pheromoneUpdate();
    smmas.checkPheromoneTrailLimits();
    smmas.searchControl(); // TODO: Rever
    iterationStatistic.endTimer("Pheromone");
    // Statistics
    if (i % statisticInterval == 0) {
        iterationStatistic.setIteration(i);
        iterationStatistic.setBestSoFar(smmas.getBestSoFar().totalCost);
        iterationStatistic.setDiversity(smmas.calculateDiversity());
        iterationStatistic.setBranchFactor(smmas.nodeBranching());
    }
}

```

```

        iterationStatistic.setIterationBest(smmas.findBest().totalCost);
        iterationStatistic.setIterationWorst(smmas.findWorst().totalCost);
        iterationStatistic.setFeasible(smmas.getBestSoFar().feasible);

iterationStatistic.setIterationMean(Maths.getMean(smmas.getAntPopulation().stream().
map(Ant::getCost).collect(Collectors.toList())));

iterationStatistic.setIterationSd(Maths.getStd(smmas.getAntPopulation().stream().map(
Ant::getCost).collect(Collectors.toList())));

        iterationStatistic.setPenaltyRate(smmas.getPenaltyRate());
        iterationStatistics.add(iterationStatistic);
        if (showLog) {
            System.out.println(iterationStatistic);
            logInFile(iterationStatistic.toString());
        }
    }
}

// In kết quả
globalStatistics.endTimer("Algorithm");
printFinalRoute();
}

```

## 4.2. Mô tả dữ liệu thực nghiệm

Các bộ dữ liệu chúng tôi sử dụng để kiểm tra được đề xuất bởi Nacache và các cộng sự vào năm 2018 [4] và được mở rộng từ bộ dữ liệu của Li & Lim [2001] với bài toán Pickup and Delivery Problem with TimeWindows - PDPTW lấy từ địa chỉ <https://www.sintef.no/projectweb/top/pdptw/li-lim-benchmark/>.

Mỗi bộ dữ liệu được đặc trưng bởi thời gian cửa sổ (time windows), độ dài tối đa của yêu cầu và số lượng nút. Một bộ dữ liệu được xác định là không có thời gian cửa sổ (Without TWs), thời gian cửa sổ ở mức bình thường (Normal TWs) hoặc thời gian cửa sổ lớn (Large TWs). Đối với mỗi bộ dữ liệu, một yêu cầu có kích thước nhỏ nhất phải bao gồm hai nút trong đó một nút là nút giao. Tùy thuộc vào mỗi bộ dữ liệu, nó có thể bao gồm nhiều nhất 4 nút đón và giao (Yêu cầu ngắn) hoặc 8 nút đón và giao (Yêu cầu dài). Các bộ dữ liệu

kiểm tra của chúng tôi chứa 25, 50, 100 và 400 nút. Chúng tôi sẽ có 24 bộ dữ liệu khác nhau, với mỗi bộ dữ liệu này chúng tôi xây dựng 5 trường hợp khác nhau. Như vậy, chúng tôi sẽ có tổng cộng 120 trường hợp để thử nghiệm.

Hình 4.1 dưới đây sẽ trình bày các bộ dữ liệu được xác định theo các đặc điểm:

Kích thước bộ dữ liệu	Không có thời gian cửa sổ		Thời gian cửa sổ bình thường		Thời gian cửa sổ lớn	
	Yêu cầu ngắn	Yêu cầu dài	Yêu cầu ngắn	Yêu cầu dài	Yêu cầu ngắn	Yêu cầu dài
25	w_4_25	w_8_25	n_4_25	n_8_25	l_4_25	l_8_25
50	w_4_50	w_8_50	n_4_50	n_8_50	l_4_50	l_8_50
100	w_4_100	w_8_100	n_4_100	n_8_100	l_4_100	l_8_100
400	w_4_400	w_8_400	n_4_400	n_8_400	l_4_400	l_8_400

*Hình 4. 1 Các bộ dữ liệu thử nghiệm.*

Ví dụ: Bộ dữ liệu **L\_4\_25\_1.txt** biểu diễn khoảng thời gian cửa sổ lớn, một yêu cầu có tối đa 4 nút đón và giao (thuộc loại yêu cầu ngắn) và bộ dữ liệu gồm có 25 nút.

#### **Định dạng của bộ dữ liệu như sau:**

- Hàng đầu tiên: Số lượng xe (phương tiện); Sức chứa của xe.
- Hàng thứ 2: Biểu diễn kho (điểm bắt đầu và kết thúc của mỗi lộ trình)
- Hàng thứ 3 -> n : Biểu diễn cho mỗi nút.
- Trong mỗi hàng gồm 8 cột được biểu diễn như sau:
  - Cột 1: NodeID
  - Cột 2: Tọa độ X
  - Cột 3: Tọa độ Y
  - Cột 4: Nhu cầu (Demand) (Đối với trường hợp là nút giao thì nhu cầu <0)
  - Cột 5: Thời gian phục vụ sớm nhất (Start TWs)
  - Cột 6: Thời gian phục vụ muộn nhất (End TWs)
  - Cột 7: Trạng thái của nút. (Giá trị: -1 :kho, 0: nút đón, 1: nút giao)
  - Cột 8: Chỉ số của yêu cầu (RequestID).

#### **4.3. Hiệu năng lời giải mô hình toán học**

Chương trình được viết bằng ngôn ngữ Java chạy trên Intel(R) Core(TM) i5-5200U CPU 2.20GHz Ram 4GB, sử dụng hệ điều hành Window 10.

Các tham số cần thiết để chạy thuật toán ACO bao gồm: nAnt, maxIterations, rho, alpha, beta.

Trong đó:

- nAnt: Tổng số kiến

- maxIterations: Số vòng lặp
- rho: Hằng số bay hơi của vết mùi
- alpha, beta: Hai tham số quyết định sự ảnh hưởng tương quan giữa thông tin mùi và thông tin heuristic

**- Kết quả thực nghiệm:**

Chúng tôi tiến hành chạy thực nghiệm trên các bộ dữ liệu của bài toán với các tham số được thiết lập như sau:

Số kiến	100
$\alpha$	1.0
$\beta$	2.0
$\rho$	0.03
Số vòng lặp	2000

**VD:** Hình 4.2 và Hình 4.3 mô tả dữ liệu đầu vào và kết quả đầu ra tương ứng của một trường hợp kiểm tra của chúng tôi với bộ dữ liệu **L\_4\_25**

**Mô tả bộ dữ liệu đầu vào: L\_4\_25\_1.txt**

Số xe = 15	15	200	Sức chứa của xe = 200					
0	250	250	0	0	1925	0	-1	-1
0	396	417	13	520	1130	10	0	0
1	51	286	-13	1169	1779	10	1	0
2	288	439	16	0	593	10	0	1
3	399	175	13	457	1067	10	0	1
4	247	306	33	0	366	10	0	1
5	274	261	-62	1417	1925	10	1	1
6	176	199	28	0	399	10	0	2
7	461	182	-28	339	949	10	1	2
8	489	159	29	1138	1748	10	0	3
9	323	29	7	847	1457	0	0	3
10	385	58	24	115	725	10	0	3
11	226	175	-60	1404	1925	10	1	3
12	471	368	19	138	748	10	0	4
13	490	286	-19	232	842	10	1	4
14	253	50	27	0	546	10	0	5
15	0	356	13	796	1406	10	0	5
16	209	212	-40	1238	1848	10	1	5
17	144	35	26	0	549	10	0	6
18	154	374	36	647	1257	10	0	6
19	452	484	26	988	1598	0	0	6
20	485	428	-88	1200	1810	10	1	6
21	441	7	20	102	712	10	0	7
22	235	288	18	0	350	10	0	7
23	392	1	11	315	925	10	0	7
24	60	5	-49	754	1364	10	1	7
NodeID	Tọa độ x	Tọa độ y	Demand (<0 khi là nút delivery)	StartTW	EndTW	ServiceTime	Pickup = 0 Delivery !=0	Request ID

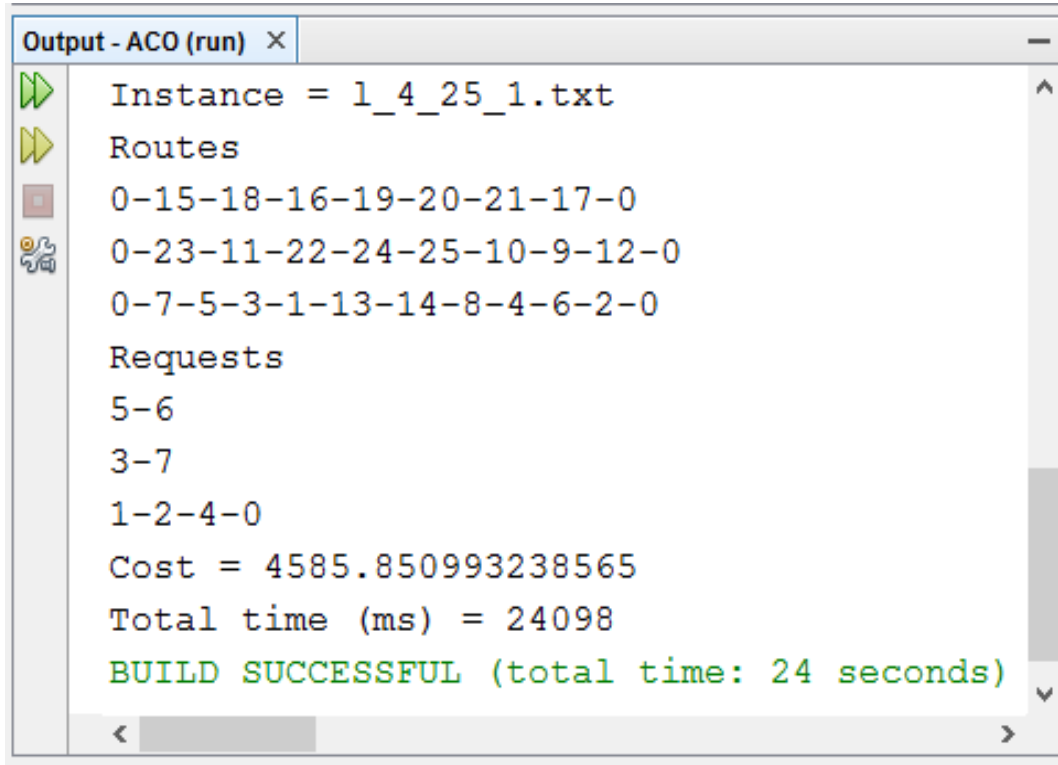
Hình 4. 2 Mô tả dữ liệu đầu vào.

Bộ dữ liệu trong hình 4.2 bao gồm:

- 1 kho, tối đa 15 xe và sức chứa của xe=200
- 25 nodes có số thứ tự từ 0 đến 24
- 8 request có số thứ tự từ 0 đến 7
- 1 request có tối đa 4 nút (tối đa 3 nút đón và chỉ 1 nút giao).

VD: Request 1 gồm các nút : 2,3,4,5 trong đó nút 5 là nút giao còn lại là các nút đón.

**Mô tả dữ liệu đầu ra**



```

Output - ACO (run) x
Instance = 1_4_25_1.txt
Routes
0-15-18-16-19-20-21-17-0
0-23-11-22-24-25-10-9-12-0
0-7-5-3-1-13-14-8-4-6-2-0
Requests
5-6
3-7
1-2-4-0
Cost = 4585.850993238565
Total time (ms) = 24098
BUILD SUCCESSFUL (total time: 24 seconds)
  
```

Hình 4. 3 Mô tả kết quả chương trình.

Kết quả đưa ra lời giải tốt nhất của bộ dữ liệu “L\_4\_25\_1” bao gồm:

- Tập các tuyến đường được định tuyến gồm 3 routers  
*Router 1:* 0-15-18-16-19-20-21-17-0  
*Router 2:* 0-23-11-22-24-25-10-9-12-0  
*Router 3:* 0-7-5-3-1-13-14-8-4-6-20
- Tập các yêu cầu đã được phục vụ tương ứng với 3 tuyến đường đã định tuyến.  
 Ứng với *Router 1* là tập yêu cầu : 5-6  
 Ứng với *Router 2* là tập yêu cầu : 3-7  
 Ứng với *Router 3* là tập yêu cầu : 1-2-4-0
- Tổng chi phí cho toàn bộ giải pháp. Cost = 4585.85
- Tổng thời gian thực hiện chương trình. Time = 24098 (ms)

Bảng 4.1 dưới đây là kết quả chúng tôi thống kê được khi tiến hành chạy thực nghiệm với 24 bộ dữ liệu chuẩn. Do ACO là lớp các thuật toán metaheuristic nên chúng tôi tiến hành chạy chương trình lấy kết quả trung bình sau 10 lần chạy trên mỗi bộ dữ liệu.

**Bảng 4. 1 Kết quả thực nghiệm của ACO sử dụng SMMAS.**

<b>Bộ dữ liệu</b>	<b># Số yêu cầu</b>	<b># Số nút</b>	<b>Kết quả trung bình</b>	<b>Thời gian tốt nhất (ms)</b>
W_4_25	8.6	26	3079.90	16458.40
W_8_25	6.0	26.8	3096.94	15712.20
W_4_50	17.2	50.6	5123.90	54041.80
W_8_50	10.8	51.0	5156.73	44133.00
W_4_100	34.2	101.2	8701.13	272586.20
W_8_100	21.6	103.2	9781.29	208656.40
W_4_400	401.0	133.8	29262.04	3057922.00
W_8_400	402.4	81.6	38393.01	1906448.00
N_4_25	8.6	26.0	4734.82	14739.00
N_8_25	6.0	26.8	4694.74	14682.40
N_4_50	17.2	50.6	8928.97	46676.60
N_8_50	10.8	51.0	8532.98	40307.60
N_4_100	34.2	101.2	15391.03	214317.20
N_8_100	21.6	103.2	16899.39	188463.60
N_4_400	401.0	133.8	49382.61	1983540.20
N_8_400	402.4	81.6	59768.72	1601169.60
L_4_25	8.6	26.0	4117.55	15622.60
L_8_25	6.0	26.8	4424.05	15115.00
L_4_50	17.2	50.6	7244.07	50620.60
L_8_50	10.8	51.0	7500.54	41907.80
L_4_100	34.2	101.2	12240.41	234821.40

L_8_100	21.6	103.2	14618.16	203575.60
L_4_400	401.0	133.8	40630.89	2249541.20
L_8_400	402.4	81.6	50530.35	1723525.60

Nhận xét: Từ kết quả ở bảng 4.1 chúng tôi quan sát thấy rằng, đối với trường hợp bộ dữ liệu có cùng kích thước thì bộ dữ liệu thuộc loại yêu cầu dài (có tối đa 8 nút trong một yêu cầu) sẽ dễ dàng giải quyết hơn bộ dữ liệu thuộc loại yêu cầu ngắn (có tối đa 4 nút trong một yêu cầu) do số yêu cầu cần định tuyến ít hơn nên thời gian chạy nhỏ hơn.

#### **Đánh giá kết quả thực nghiệm:**

Bảng 4.2 so sánh kết quả giữa ACO là lời giải tối ưu của thuật toán được chúng tôi cài đặt và CPLEX [4] là lời giải tối ưu của bài toán được lập trình trên mô hình toán học đã xây dựng ở chương 3 phần 3.1.2.

Sai số được xác định bởi công thức  $Sai\ số\ (\%) = \frac{ACO - CPLEX}{ACO} * 100\%$ . Chúng ta có thể

thấy rằng với tất cả các bộ dữ liệu sai số là tương đối nhỏ, nghĩa là lời giải tối ưu của thuật toán rất gần với lời giải tối ưu của bài toán. Hơn nữa, CPLEX không giải được các trường hợp có 400 nút do nó thiếu bộ nhớ, ACO đã khắc phục được nhược điểm này.

**Bảng 4. 2 Bảng so sánh ACO và CPLEX.**

<b>Bộ dữ liệu</b>	<b>ACO</b>	<b>CPLEX</b>	<b>Sai số (%)</b>
W_4_25	3079.90	3079.90	0.00
W_8_25	3096.94	3047.18	1.61
W_4_50	5123.90	5108.32	0.30
W_8_50	5156.73	4970.50	3.61
W_4_100	8701.13	8432.62	3.09
W_8_100	9781.29	9221.80	5.72
W_4_400	29262.04	-	-
W_8_400	38393.01	-	-
N_4_25	4734.83	4734.83	0.00
N_8_25	4694.74	4646.16	1.03

N_4_50	8928.97	8923.03	0.07
N_8_50	8532.98	8521.67	0.13
N_4_100	15391.03	15217.64	1.13
N_8_100	16899.39	16894.20	0.03
N_4_400	49382.61	-	-
N_8_400	59768.72	-	-
L_4_25	4117.56	4117.56	0.00
L_8_25	4424.05	4424.05	0.00
L_4_50	7244.07	7213.16	0.43
L_8_50	7500.54	7368.10	1.77
L_4_100	12240.41	12060.54	1.47
L_8_100	14618.16	13930.92	4.70
L_4_400	40630.89	-	-
L_8_400	50530.35	-	-

Nhận xét: Dựa vào kết quả thực nghiệm ở bảng 4.2, dễ dàng nhận thấy rằng các bộ dữ liệu thử nghiệm có thời gian cửa sổ ở mức bình thường có độ sai số là không đáng kể, sai số cao nhất chiếm 1.13% ở bộ dữ liệu N\_4\_100. Còn kết quả của các bộ dữ liệu thử nghiệm có thời gian cửa sổ lớn sai số cao nhất là 4.70% ở bộ dữ liệu L\_8\_100. Riêng đối với trường hợp không có thời gian cửa sổ sai số chiếm tỷ lệ cao hơn so với hai trường hợp còn lại, sai số cao nhất chiếm 5.72% ở bộ dữ liệu W\_8\_100.



Bảng 4.3 so sánh kết quả của các bộ dữ liệu đầu vào kích thước lớn gồm 100 và 400 nút khi chạy cùng vòng lặp giữa thuật toán ACO sử dụng quy tắc cập nhật mùi SMMAS do chúng tôi cài đặt và thuật toán ACO sử dụng quy tắc cập nhật mùi MMAS được lấy từ đường dẫn <https://github.com/schmittjoapedro/aco-vrp-framework/tree/mpdptw/>

**Bảng 4. 3 So sánh kết quả tốt nhất của SMMAS và MMAS có cùng vòng lặp.**

Bộ dữ liệu	SMMAS		MMAS	
	Kết quả trung bình	Thời gian tốt nhất (ms)	Kết quả trung bình	Thời gian tốt nhất (ms)
W_4_100	<b>8701.13</b>	<b>272586.20</b>	8854.75	281967.40
W_8_100	<b>9781.29</b>	<b>208656.40</b>	9848.67	211445.40
N_4_100	<b>15391.03</b>	<b>214317.20</b>	15424.26	216992.20
N_8_100	<b>16899.39</b>	<b>188463.60</b>	16899.61	189715.60
L_4_100	12240.41	<b>234821.40</b>	<b>12212.78</b>	235508.00
L_8_100	14618.16	<b>203575.60</b>	<b>14600.39</b>	203960.60
W_4_400	<b>29262.04</b>	<b>3057922.00</b>	29422.76	3059050.00
W_8_400	<b>38393.01</b>	<b>1906448.00</b>	38528.60	1944719.80
N_4_400	<b>49382.61</b>	<b>1983540.20</b>	49558.12	1997127.80
N_8_400	<b>59768.72</b>	<b>1601169.60</b>	59928.57	1619638.60
L_4_400	40630.89	<b>2249541.20</b>	<b>40455.38</b>	2277968.20
L_8_400	50530.35	<b>1723525.60</b>	<b>50350.62</b>	1738759.00

Nhận xét : Trong bảng 4.3 các kết quả tốt được chúng tôi bôi đậm. Kết quả so sánh cho thấy thuật toán ACO sử dụng SMMAS có thời gian chạy nhanh hơn so với MMAS và chỉ có 4 bộ dữ liệu trên tổng số 12 bộ dữ liệu tức là khoảng 33.3% kết quả cho thấy ACO sử dụng MMAS cho chất lượng lời giải tốt hơn. Như vậy, ta hoàn toàn có thể sử dụng SMMAS để tìm kiếm lời giải.

Bảng 4.4 so sánh kết quả của các bộ dữ liệu đầu vào kích thước lớn gồm 100 và 400 nút khi chạy cùng thời gian giữa thuật toán ACO sử dụng quy tắc cập nhật mùi SMMAS và thuật toán ACO sử dụng quy tắc cập nhật mùi MMAS. Đối với bộ dữ liệu gồm 100 nút chúng tôi cài đặt sau 120s đưa ra kết quả, còn bộ dữ liệu 400 nút đưa ra kết quả sau 600s.

**Bảng 4. 4 So sánh kết quả tốt nhất của SMMAS và MMAS có cùng thời gian chạy.**

Bộ dữ liệu	Kết quả khi sử dụng SMMAS	Kết quả khi sử dụng MMAS
W_4_100	<b>8738.11</b>	8759.58
W_8_100	<b>10123.29</b>	10170.23
N_4_100	<b>15401.96</b>	15430.47
N_8_100	<b>16899.39</b>	16899.61
L_4_100	12240.41	<b>12236.42</b>
L_8_100	<b>14634.35</b>	14638.30
W_4_400	<b>29839.67</b>	30064.50
W_8_400	<b>38669.75</b>	38722.70
N_4_400	<b>49663.37</b>	50115.40
N_8_400	<b>59988.57</b>	60019.07
L_4_400	41170.06	<b>41150.40</b>
L_8_400	<b>50637.63</b>	50723.71

Nhận xét : Bảng 4.4 các kết quả tốt được chúng tôi bôi đậm. Như vậy, chỉ có 2 bộ dữ liệu trên tổng số 12 bộ dữ liệu tức là chiếm khoảng 16.67% kết quả cho thấy MMAS tìm kiếm được lời giải tốt nhanh hơn. Do đó, hoàn toàn có thể sử dụng SMMAS để tìm kiếm lời giải cho bài toán với điều kiện thời gian chạy thấp.

*Kết luận:*

- Thuật toán SMMAS cho lời giải tối ưu gần với lời giải tối ưu của bài toán.
- Khi chạy cùng vòng lặp, thời gian thực hiện thuật toán SMMAS nhanh hơn thuật toán MMAS.
- Khi chạy cùng thời gian, SMMAS cho kết quả tốt hơn MMAS khoảng 83.33% .

## KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Trong nghiên cứu này, chúng tôi đã đưa giải pháp tìm lời giải tối ưu cho bài toán định tuyến xe đa điểm đón và giao hàng với thời gian cửa sổ có kích thước dữ liệu lớn. Cụ thể, chúng tôi áp dụng thuật toán tối ưu đàn kiến cho bài toán có sử dụng công thức cập nhật mùi SMMAS. Kết quả thực nghiệm cho thấy, thuật toán tối ưu đàn kiến sử dụng SMMAS hoạt động hiệu quả và thời gian chạy nhanh hơn và cho chất lượng lời giải tốt hơn so với MMAS, thuật toán đưa ra lời giải cho bài toán trong thời gian chấp nhận được cũng như tỷ lệ sai số giữa lời giải tối ưu của thuật toán và lời giải tối ưu của bài toán là khá nhỏ.

Trong tương lai, đề tài có thể phát triển theo hướng phân tích thêm một số ràng buộc mở rộng cho bài toán như có nhiều hơn một kho chứa các tập xe, ràng buộc thời gian di chuyển trên tuyến đường,... và nghiên cứu các kỹ thuật mới để cải tiến chất lượng cũng như thời gian thực hiện cho bài toán.

## TÀI LIỆU THAM KHẢO

### Tiếng Việt

- [1] Hoàng Xuân Huân, Đỗ Đức Đông (2012), *Giáo trình Tối ưu hóa* – Trường Đại học Công nghệ - Đại học Quốc gia Hà Nội.
- [2] Đỗ Đức Đông (2012), *Phương pháp tối ưu đàn kiến và ứng dụng*, Đại học Công nghệ - Đại học Quốc gia Hà Nội.
- [3] Đặng Thị Thanh Nguyên (2010), *Giải thuật Large Neighborhood Search và Simulated Annealing cho một biến thể thực tế của bài toán Vehicle Routing*, Đại học Khoa học tự nhiên - Đại học Quốc gia TP HCM.

### Tiếng Anh

- [4] Naccache, J.-F. Côté, and L.C. Coelho (2018), *The multi-pickup and delivery problem with time windows*. European Journal of Operational Research, 269(1) pp 353–362.
- [5] Imadeddine Aziez, Jean-François Côté, Leandro C. Coelho (2019), *A Branch-and-Cut Algorithm for the Multi-Pickup and Delivery Problem with Time Windows*, CIRRELT-2019-04.
- [6] Yu Bin, Yang Zhong-Zhen, Yao Baozhen (2009), *An improved ant colony optimization for vehicle routing problem*, European Journal of Operational Research (196), pp 171-176.
- [7] M.Noumbissi Tchoupo, A.Yalaoui, L.Amodeo, F.Yalaoui and F.Lutz (2017), *Ant Colony Optimization Algorithm for Pickup and Delivery Problem with Time Windows*, University of Technology of Troyes, France.
- [8] C. cung hetti, D. Feillet, M. Gendreau, and M.G. Speranza (2010), *Complexity of the VRP and SDVRP*, Transport Res Part C19.
- [9] R. BALDACCI, M. BATTARRA, AND D. VIGO (2008), *Routing a heterogeneous fleet of vehicles*, in *The Vehicle Routing Problem: Latest Advances and New Challenges*, Springer, New York, pp. 3–27.
- [10] M. Dorigo, L. Gambardella (1997), *Ant colony system: A cooperative learning approach to the traveling salesman problem*, IEEE Trans. on evolutionary computation. 1(1), pp. 53-66.
- [11] W. Gutjahr (2002), *"ACO algorithms with guaranteed convergence to the optimal solution"*, Info.Proc. Lett. 83(3), tr. 145-153.
- [12] P. Pellegrini và A. Ellero (2008), *The Small World of Pheromone Trails*, Proc. of the 6th international conference on Ant Colony Optimization and Swarm Intelligence, Brussels, Belgium.

- [13] M. Dorigo, and T. Stützle (2004), *Ant Colony Optimization*, The MIT Press, Cambridge, Massachusetts.
- [14] C. Coelho, J. Renaud, and G. Laporte (2016). Road-based goods transportation: *a survey of real-world logistics applications from 2000 to 2015*. *INFOR: Information Systems and Operational Research*, 54(2), pp 79–96.
- [15] Pisinger and S. Ropke (2007), *A general heuristic for vehicle routing problems*. *Computers & Operations Research*, 34(8), pp 2403–2435.
- [16] Subramanian, L. M. A. Drummond, C. Bentes, L. S. Ochi, and R. Farias (2010), *A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery*. *Computers & Operations Research*, 37(11), pp 1899–1911.
- [17] P. Goksal, I. Karaoglan, and F. Altıparmak (2013). *A hybrid discrete particle swarm optimization for vehicle routing problem with simultaneous pickup and delivery*. *Computers & Industrial Engineering*, 65(1), pp 39–53.
- [18] Ropke, J.-F. Cordeau, and G. Laporte (2007), *Models and branch-and-cut algorithms for pickup and delivery problems with time windows*. *Networks*, 49(4), pp 258–272.
- [19] Ropke and J.-F. Cordeau (2009), *Branch and cut and price for the pickup and delivery problem with time windows*. *Transportation Science*, 43(3), pp 267–286.
- [20] Baldacci, E. Bartolini, and A. Mingozzi (2011), *An exact algorithm for the pickup and delivery problem with time windows*. *Operations Research*, 59(2), pp 414–426.
- [21] F. Escudero (1988), *An inexact algorithm for the sequential ordering problem*. *European Journal of Operational Research*, 37(2), pp 236–249.
- [22] Ezzat, A. M. Abdelbar, and D. C. Wunsch (2014), *An extended EigenAnt colony system applied to the sequential ordering problem*, In 2014 IEEE Symposium on Swarm Intelligence, pp 1-7, Orlando.
- [23] W. P. Savelsbergh (1990), *An efficient implementation of local search algorithms for constrained routing problems*, *European Journal of Operational Research*, 47(1), pp 75–85.
- [24] Ascheuer, M. Jünger, and G. Reinelt (2000), *A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints*, *Computational Optimization and Applications*, 17(1), pp 61–84.
- [25] Guerriero and M. Mancini (2003), *A cooperative parallel rollout algorithm for the sequential ordering problem*, *Parallel Computing*, 29(5) pp 663–677.
- [26] I. Seo and B.R. Moon (2003), *A hybrid genetic algorithm based on complete graph representation for the sequential ordering problem*, In Genetic and Evolutionary Computation Conference, pp 669–680.

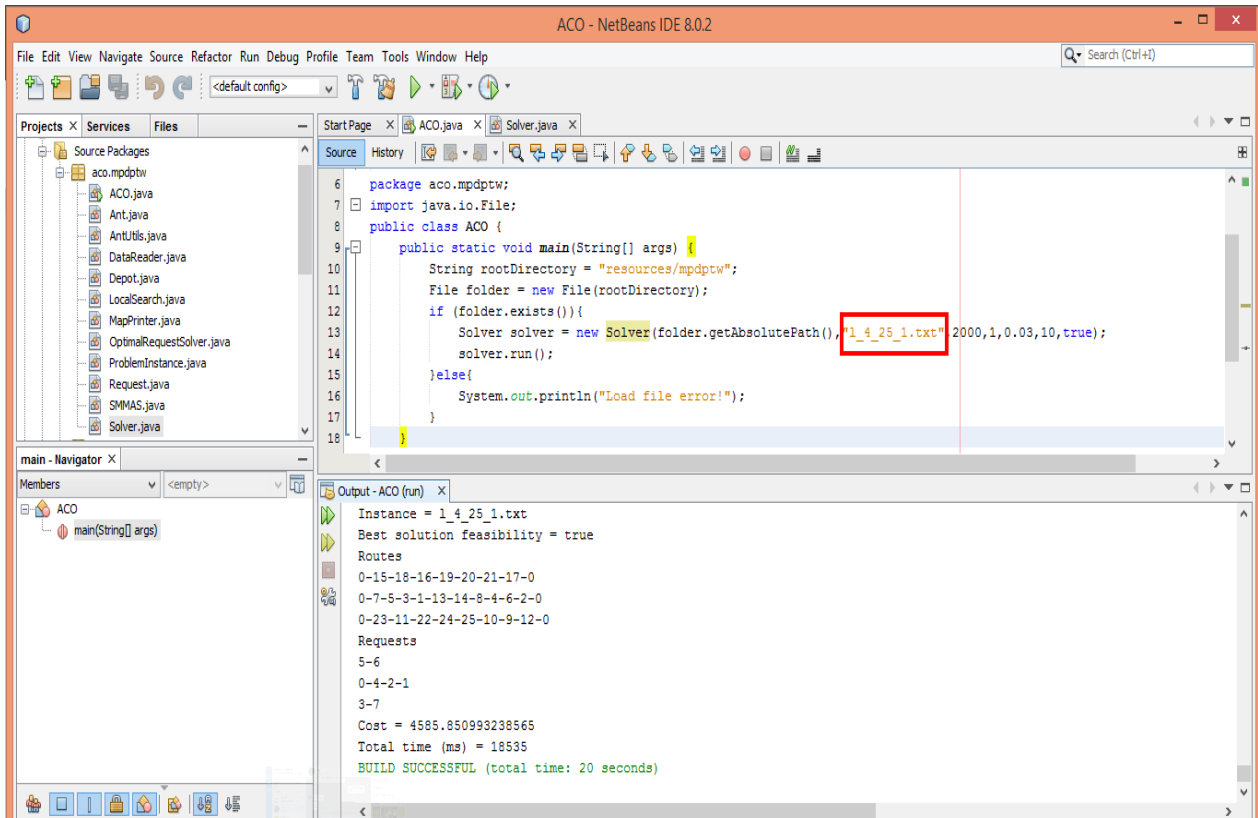
- [27] N. Letchford and J.-J. Salazar-González (2016), *Stronger multi-commodity flow formulations of the (capacitated) sequential ordering problem*. European Journal of Operational Research, 251(1), pp 74–84.
- [28] Alonso-Ayuso, P. Detti, L. F. Escudero, and M. T. Ortuno (2003), *On dual based lower bounds for the sequential ordering problem with precedences and due dates*, Annals of Operations Research, 124(1-4), pp 111–131.
- [29] S. Ropke (2005), *Heuristic and exact algorithms for vehicle routing problems*, Ph.D. Thesis, Computer science department at the University of Copenhagen (DIKU).
- [30] M. Desrochers, J. Lenstra, M. Savelsbergh, and F. Soumis (1988), *Vehicle routing with time windows: optimization and approximation*, In Bruce L. Golden and Arjang A. Assad (editors), *Vehicle Routing: Methods and Studies*, North-Holland, Amsterdam, pp 65-84.
- [31] N. Bansal, A. Blum, S. Chawla, A. Meyerson (2004), *Approximation algorithms for deadline-TSP and vehicle routing with time-windows*, in Proceedings of the thirty-sixth annual ACM symposium on Theory of computing (STOC '04), New York, USA.
- [32] G.Laporte, Y.Nobert (1983), *A branch and bound algorithm for the capacitated vehicle routing problem*, Operations Research Spektrum, pp 77-85.
- [33] F.Liberatore, G.Righini, M.Salani (2010), *A column generation algorithm for the vehicle routing problem with soft time windows*, 4OR quarterly journal of the Belgian, French and Italian Operations Research Societies 9(1), pp 49-82.
- [34] S.C.H.Leung, J.Zheng, D.Zhang, X.Zhou (2010), *Simulated annealing for the vehicle routing problem with two-dimensional loading constraints*, Flexible Services and Manufacturing Journal 22(1), pp 61-82.
- [35] J. Kytojoki, T. Nuortio, O. Braysy and M. Gendreau (2007), *An efficient variable neighborhood search heuristic for very large scale vehicle routing problems*, Computers & Operations Research, vol. 34, pp 2743–2757.
- [36] B.M.Baker, M.A.Ayechew (2003), *A genetic algorithm for the vehicle routing problem*, Computers & Operations Research, pp 787-800.
- [37] S. Ropke and D. Pisinger (2006), *An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows*, Transportation Science, vol. 40, pp 455–472.

## PHỤ LỤC

Phụ lục này trình bày cách chạy chương trình trên và modules cơ bản của thuật toán dưới dạng ngôn ngữ Java 8

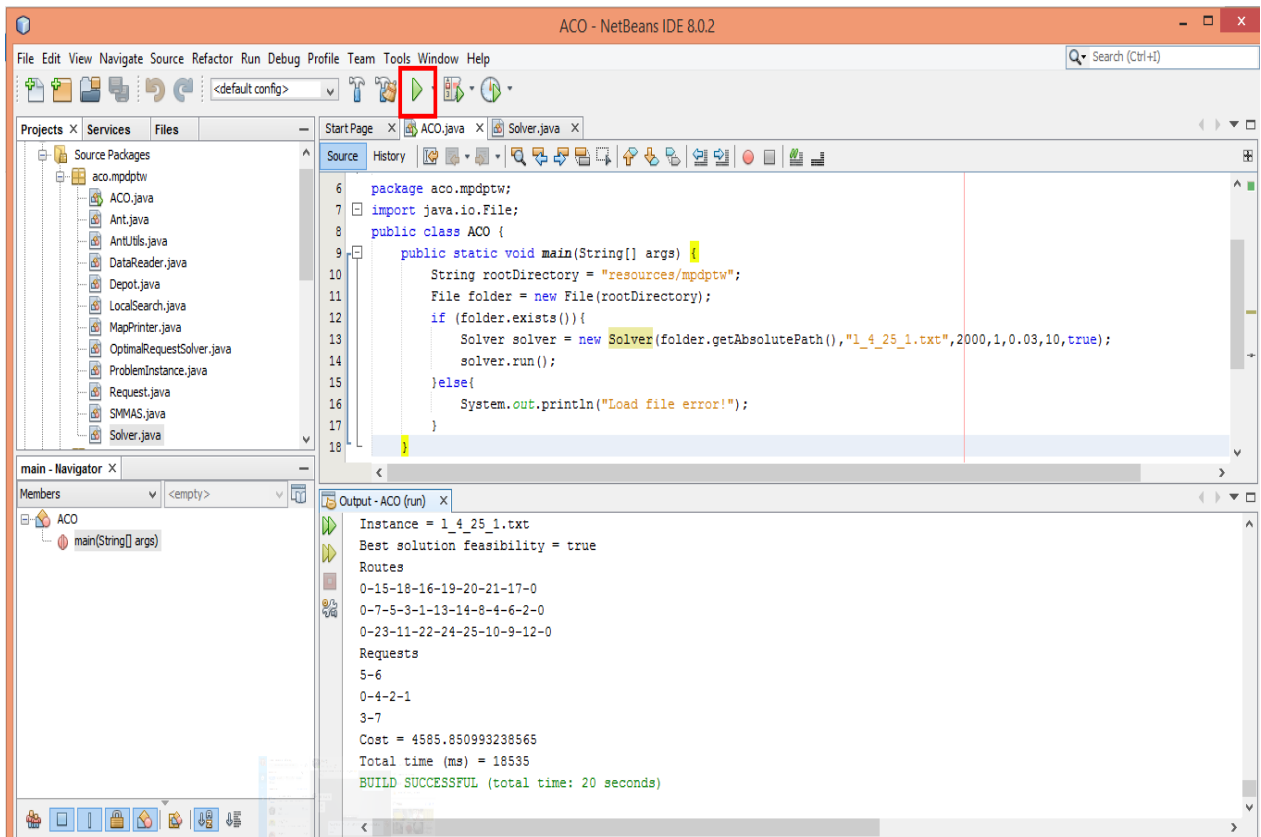
Cách chạy chương trình:

Bước 1: Mở chương trình với IDE. Tại class ACO.java nhập bộ dữ liệu cần chạy vào mục khoanh đỏ như Hình 4.4



Hình 4. 4 Mở chương trình

Bước 2: Bấm nút “Run” hoặc nhấn F6 để chạy chương trình như Hình 4. 5



Hình 4. 5 Chạy chương trình

Kết quả được ghi vào “C:\Temp\mpdptw”



**Module cơ bản:*****Quy tắc cập nhật mùi SMMAS***

```

// Khởi tạo vết mùi
public void initTry() {
    lambda = 0.05;
    restartIteration = 1;
    for (int i = 0; i < bestSoFar.tourLengths.size(); i++) {
        bestSoFar.tourLengths.set(i, Double.MAX_VALUE);
    }
    foundBest = 0;
    trailMax = 1.0 / ((rho) * nnTour());
    trailMin = trailMax / (2.0 * instance.noNodes);
    initPheromoneTrails(trailMax);
}

private void initPheromoneTrails(double initialTrail) {
    for (int i = 0; i < instance.noNodes; i++) {
        for (int j = 0; j < instance.noNodes; j++) {
            if (i != j) {
                pheromoneNodes[i][j] = initialTrail;
            }
        }
    }
}

//Giới hạn vết mùi trong  $[t_{min}, t_{max}]$ 
public void checkPheromoneTrailLimits() {
    for (int i = 0; i < pheromoneNodes.length; i++) {
        for (int j = 0; j < pheromoneNodes.length; j++) {
            if (i != j) {
                if (pheromoneNodes[i][j] < trailMin) {
                    pheromoneNodes[i][j] = trailMin;
                } else if (pheromoneNodes[i][j] > trailMax) {
                    pheromoneNodes[i][j] = trailMax;
                }
            }
        }
    }
}

//Cập nhật vết mùi tại các cạnh không thuộc lời giải tốt nhất
public void evaporation() {
    for (int i = 0; i < pheromoneNodes.length; i++) {
        for (int j = 0; j < pheromoneNodes[i].length; j++) {
            if (i != j) {

```

```

        pheromoneNodes[i][j] = (1.0 - rho) * pheromoneNodes[i][j] + rho
    * trailMin;
    }
    }
}

//Cập nhật vết mùi tại các cạnh thuộc lời giải tốt nhất
public void globalPheromoneDeposit(Ant ant) {
    int from, to;
    double dTau = 1.0 / ant.totalCost;
    for (int i = 0; i < ant.tours.size(); i++) {
        for (int j = 0; j < ant.tours.get(i).size() - 1; j++) {
            from = ant.tours.get(i).get(j);
            to = ant.tours.get(i).get(j + 1);
            pheromoneNodes[from][to] = pheromoneNodes[from][to] + rho *
trailMax;
        }
    }
}

```