BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC MỎ - ĐỊA CHẤT

BÁO CÁO NGHIÊN CỨU KHOA HỌC

Đề tài: Thuật toán Kiến

Phần 2 : Cài đặt thuật toán kiến và

chạy thử nghiệm trên một số thành phố hữu hạn

Giảng viên thực hiện: ThS. Dương Chí Thiện

Hà Nội, 06/2023

MỤC LỤC

CHƯƠNG I: TỐI ƯU TỔ HỢP

1. Bài toán tối ưu tổ hợp tổng quát.

Trong thực tế và khi xây dựng các hệ thông tin, ta thường gặp các bài toán tối ưu tổ hợp (TƯTH). Trong đó phải tìm các giá trị cho các biến rời rạc để làm cựctrị hàm mục tiêu nào đó. Đa số các bài toán này thuộc lớp NP-khó. Trừ các bài toán cỡ nhỏ có thể tìm lời giải bằng cách tìm kiếm vét cạn, còn lại thì thường không thể tìm được lời giải tối ưu.

Đối với các bài toán cỡ lớn không có phương pháp giải đúng, đến nay người ta vẫn dùng các cách tiếp cận sau:

1) Tìm kiếm heuristic để tìm lời giải đủ tốt;
2) Tìm kiếm cục bộ để tìm lời giải tối ưu địa phương;
3) Tìm lời giải gần đúng nhờ các thuật toán mô phỏng tự nhiên như: mô phỏng luyện kim, giải thuật di truyền, tối ưu bầy đàn,…

Hai cách tiếp cận đầu thường cho lời giải nhanh nhưng không thể cải thiện thêm lời giải tìm được, nên cách tiếp cận thứ ba đang được sử dụng rộng rãi cho các bài toán cỡ lớn.

Trong các phương pháp mô phỏng tự nhiên, tối ưu đàn kiến (*Ant Colony Optimization - ACO*) là cách tiếp cận metaheuristic tương đối mới, được giới thiệu bơei Dorigo năm 1991 đang được nghiên cứu và ứng dụng rộng rãi cho các bài toán TƯTH khó.

Các thuật toán ACO sử dụng kết hợp thông tin kinh nghiệm (he uristic) và học tăng cường qua các vết mùi của các con kiến nhân tạo để giải

các bài toán TƯTH bằng cách đưa về bài toán tìm đường đi tối ưu trên đồ thị cấu trúc tương ứng của bài toán. Phương pháp này được áp dụng rộng rãi để giải nhiều bài toán khó và hiệu quả nổi trội của chúng so với các phương pháp mô phỏng tự nhiên khác đã được chứng tỏ bằng thực nghiệm.

Khi áp dụng các thuật toán tối ưu đàn kiến thông dụng như ACS và MMAS, người ta phải tìm một lời giải đủ tốt, trên cơ sở đó xác định các tham số cho cận trên và cận dưới của vết mùi. Điều này gây nhiều khó khăn khi áp dụng thuật toán cho các bài toán mới. Ngoài ra, lượng mùi cập nhật cho mỗi thành phần trong đồ thị tỷ lệ với giá trị hàm mục tiêu của lời giải chứa nó liệu có phản ánh đúng thông tin học tăng cường hay không cũng còn phải thảo luận.

Việc nghiên cứu sâu hơn về các thuật toán ACO và ứng dụng của nó đang được nhiều người quan tâm. Từ năm 1998 đến nay, cứ 2 năm thì có một hội nghị quốc tế về phương pháp này tổ chức Brussels.

2. Bài toán người chào hàng (TSP).

Bài toán người chào hàng (Traveling Salesman Problem - TSP) là bài toán TƯTH điển hình, được nghiên cứu nhiều và được xem là bài toán chuẩn để đánh giá hiệu quả các lược đồ giải bài toán TƯTH mới (xem [30,31]).

Bài toán được phát biểu như sau:

Có một tập gồm n thành phố (hoặc điểm tiêu thụ) $C = \{ci\}$ n, độ dài đường đi trực tiếp từ ci đến cj là di,j . Một người chào hàng muốn tìm một hành trình ngắn nhất từ nơi ở, đi qua mỗi thành phố đúng một lần để giới thiệu sản phẩm cho khách hàng, sau đó trở về thành phố xuất phát.

Như vậy, bài toán này chính là bài toán tìm chu trình Hamilton có độ dài ngắn nhất trên đồ thị đầy đủ có trọng số $G = (,)$, trong đó là tập đỉnh với

nhãn là các thành phố trong C, là các cạnh nối các thành phố tương ứng, độ dài các cạnh chính là độ dài đường đi giữa các thành phố. Trong trường hợp này, tập S sẽ là các chu trình Hamilton trên G, f là độ dài của chu trình, Ω là ràng buộc đòi hỏi chu trình là chu trình Hamilton (qua tất cả các đỉnh, mỗi đỉnh đúng một lần), C là tập thành phố được x.

(trùng với), C_0 trùng với C, tập X là vectơ độ dài n: $x = (x_1, \dots, x_n)$ với $x_i \in C$ i ≤

n, còn X^* là các vectơ trong đó x_i khác x_j đối với mọi cặp (i, j).

Do đó, lời giải tối ưu của bài toán TSP là một hoán vị π của tập đỉnh

$\{c_1, c, \dots, c_n\}$ sao cho hàm độ dài $f(\pi)$ là nhỏ nhất, trong đó $f(\pi)$ được tính theo (1.1):

$f(\pi) = \sum_{i=1}^{n-1} d(\pi(i), \pi(i+1)) + d(\pi(n), \pi(1))$ (1.1) ở đây $d(u, v)$ là khoảng cách từ u đến v.

Bài toán TSP được xem là bài toán chuẩn để kiểm định hiệu quả của các phương pháp giải bài toán TƯTH mới với thư viện dữ liệu chuẩn TSPLIB (Reinelt, 1991) tại địa chỉ [77] (Dữ liệu trong nó sẽ được sử dụng trong luận án này).

Bài toán này có nhiều ứng dụng thực tiễn, chẳng hạn như: khoan các lỗ trên bảng mạch in (Reinelt, 1994) hay định vị các thiết bị X-quang (Bland & Shallcross, 1989)… [31].

3. Cách tiếp cận.

Trên đây cho thấy các bài toán TƯTH có thể đưa về bài toán tìm kiếm trên đồ thị. Các bài toán này có thể giải đúng hoặc gần đúng. Với những bài

toán cỡ nhỏ hoặc có dạng đặc biệt người ta có thể tìm lời giải tối ưu nhờ tìm kiếm vét cạn hoặc bằng một thuật toán với thời gian đa thức, được xây dựng dựa trên các phân tích toán học. Nhiều bài toán trong số đó là NP-khó, nên với các bài toán cỡ lớn, người ta phải tìm lời giải gần đúng. Các thuật toán giải gần đúng các bài toán TƯTH khó thường dựa trên 3 kỹ thuật cơ bản: heuristic cấu trúc (construction heuristic), tìm kiếm cục bộ (local search) và Metaheuristic.

3.1. Heuristic cấu trúc.

Khi không thể tìm được lời giải tối ưu của bài toán, trong thực hành người tatìm lời giải gần đúng. Một kỹ thuật hay được dùng là heuristic cấu trúc, trong đó lời giải của bài toán TƯTH được xây dựng theo cách mở rộng tuần tự. Từ thành phần khởitạo trong tập C_0 ở mục 1.1, từng bước mở rộng không quay lui, bằng cách thêm vàocác thành phần mới theo phương thức ngẫu nhiên hay tất định dựa trên các quy tắc heuristic đã chọn. Các quy tắc heuristic này thường được xây dựng dựa trên các kếtquả phân tích toán học hoặc kinh nghiệm. Phương pháp heuristic cấu trúc tham ăn sau đây cho ta hình dung được cách tiếp cận này (Hình 1.1).

```
Procedure Heuristic cấu trúc tham ăn;
Begin
        sp ← chọn thành phần u0 trong C0;
        while (chưa xây dựng xong lời giải) do
                c ← GreedyComponent(sp);
                sp ← sp^ c;
        end-while
        s ← sp;
        Đưa ra lời giải s;
End;
```

Hình 1.1: Phương pháp heuristic cấu trúc tham ăn

Trong đó GreedyComponent(s_p) có nghĩa là chọn thành phần bổ sung vào $s_p$theo quy tắc heuristic đã có. Ký hiệu s_p ^ c là kết quả phép toán thêm thành phần $c$vào s_p.

Dễ dàng hình dung phương pháp này khi áp dụng thuật toán cho bài toán TSP với đồ thị đầy đủ và sử dụng quy tắc heuristic *láng giềng gần nhất để chọn đỉnh thêm vào* (tức là chọn đỉnh gần nhất chưa đi qua để thêm vào hành trình). Các thuật toán này có ưu điểm là tốn ít thời gian chạy nhưng nhược điểm chính là không cải tiến lời giải được.

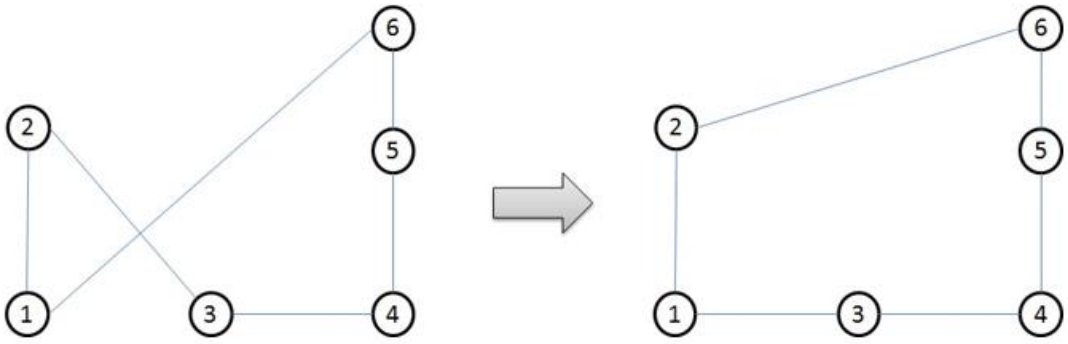
3.2. Tìm kiếm cục bộ (local search).

Kỹ thuật tìm kiếm cục bộ hay còn gọi là tìm kiếm địa phương, thực hiện bằng cách bắt đầu từ một phương án chấp nhận được, lặp lại bước cải tiến lời giải nhờ các thay đổi cục bộ. Để thực hiện kỹ thuật này, ta cần xác định được cấu trúc lân cận của mỗi phương án (lời giải) đang xét, tức là những phương án chấp nhận được, gần với nó nhất, nhờ thay đổi một số thành phần. Cách thường dùng là lân cận k-thay đổi, tức là lân cận bao gồm các phương án chấp nhận được khác với phương án đang xét nhờ thay đổi nhiều nhất k thành phần.

Ví dụ. Lân cận 2-thay đổi của một lời giải s trong bài toán TSP bao gồm tất cả các lời giải s có thể nhận được từ s bằng cách đổi hai cạnh. Hình 1.2 chỉ ra một ví dụ một lời giải nhận được bằng cách thay hai cạnh (2,3), (1,6) bằng hai cạnh (1,3), (2,6).

Việc cải tiến trong các bước lặp thường chọn theo phương pháp leo đồi dựa theo hai chiến lược: Chiến lược tốt nhất và chiến lược tốt hơn. Với chiến lược tốt nhất, người ta thực hiện chọn lời giải tốt nhất trong lân cận để

làm lời giải cải tiến. Tuy nhiên, khi bài toán cỡ lớn có thể không tìm được lời giải tốt nhất do bị hạn chế về thời gian. Còn với chiến lược tốt hơn, ta chọn phương án đầu tiên trong lân cận, cải thiện được hàm mục tiêu. Nhược điểm của tìm kiếm cục bộ là thường chỉ cho cực trị địa phương.



Hình 1.2: Lời giải nhận được nhờ thay 2 cạnh (2,3), (1,6) bằng (1,3), (2,6)

Các kỹ thuật trên thường được kết hợp, tạo thành các hệ lai trong các phương pháp mô phỏng tự nhiên dựa trên quần thể, chẳng hạn như thuật toán di truyền (GA) hoặc tối ưu đàn kiến (ACO).

3.3. Metaheuristic.

Phương pháp metaheuristic là một phương pháp heuristic tổng quát được thiết kế, định hướng cho các thuật toán cụ thể (bao gồm cả heuristic cấu trúc và tìm kiếm cục bộ). Như vậy, một metaheuristic là một lược đồ thuật toán tổng quát ứng dụng cho các bài toán tối ưu khác nhau, với một chút sửa đổi cho phù hợp với từng bài toán.

Memetic là một mô hình theo phương pháp metaheuristic. Trong các thuật toán được thiết kế theo memetic, người ta tạo ra nhiều thế hệ quần thể

lời giải chấp nhận được. Trong mỗi quần thể của thế hệ tương ứng, ta chỉ chọn ra một số lời giải (chẳng hạn lời giải tốt nhất) để thực hiện tìm kiếm cục bộ nhằm cải thiện chất lượng. Quá trình tiến hóa này cho ta tìm được lời giải tốt nhất có thể. Hình 1.3 mô tả một thuật toán memetic sử dụng tính toán tiến hóa (*Evolutionary Computing - EC*):

```
Proedure Thuật toán memetic-EC;
Begin
    Initialize: Tạo ra quần thể đầu tiên;
    while điều kiện dừng chưa thỏa mãn do
        Đánh giá các cá thể trong quần thể;
        Thực hiện tiến hóa quần thể nhờ các toán tử cho trước;
        Chọn tập con fi_il để cải tiến nhờ thủ tục tìm kiếm cục bộ;
        for mỗi cá thể trong fi_il do
            Thực hiện tìm kiếm cục bộ;
        end-for
        Chọn phần tử tốt nhất;
    end-while;
    Đưa ra lời giải tốt nhất;
End;
```

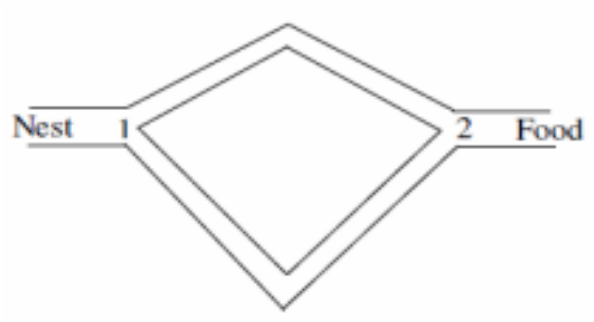
Hình 1.3: Thuật toán memetic sử dụng EC

Trong ứng dụng thực tế, các thuật toán ACO thường được kết hợp với tìm kiếm cục bộ theo mô hình memetic này.

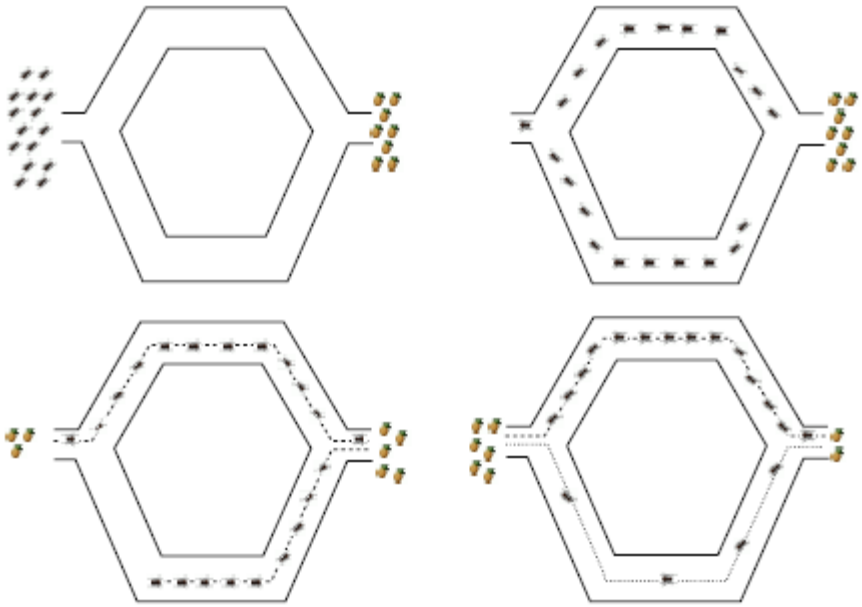
CHƯƠNG II: XÂY DỰNG THUẬT TOÁN ĐÀN KIẾN

Loài kiến là loài sâu bọ có tính chất xã hội, chúng sống thành từng đàn, bởi vậy có sự tác động lẫn nhau, chúng thạo tìm kiếm thức ăn và hoàn thành những nhiệm vụ từ kiến chỉ huy. Một điều thú vị trong tìm kiếm thức ăn của vài con kiến đặc biệt là khả năng của chúng để tìm kiếm đường đi ngắn nhất giữa tổ kiến và nguồn thức ăn. Trên thực tế, điều dễ nhận thấy có trong suy nghĩ nhưng nhiều con kiến hầu hết không nhận ra vì chúng không dùng thị giác để tìm kiếm những đầu mối thức ăn.

Tất cả mọi con kiến hầu như là mù, chúng chỉ có thể tương tác với nhau và với môi trường bằng cách sử dụng pheromone: đi đến đâu chúng xịt pheromone ra đến đấy. Mỗi một con kiến tại mỗi vị trí quyết định hướng đi tiếp theo dựa vào nồng độ pheromone của các hướng. Tại vị trí mà nồng độ pheromone xung quanh đều bằng nhau hoặc không có pheromone thì chúng sẽ quyết định hướng đi một cách ngẫu nhiên. Cứ như vậy thì các con kiến cứ đi theo bước chân của nhau và tạo thành một đường đi (path). Ta xét trường hợp tổ kiến ở vị trí 1 và nguồn thức ăn ở vị trí 2 như hình vẽ



Giả sử tại thời điểm ban đầu có 2 con kiến ra đi tìm thức ăn. Vì ban đầu chưa có pheromone nên chúng chọn 2 hướng đi khác nhau một cách ngẫu nhiên. Một hướng có đường đi đến nguồn thức ăn dài hơn hướng kia. Trong giai đoạn đầu các con kiến đi sau sẽ cảm nhận thấy nồng độ pheromone của cả 2 hướng là như nhau nên cũng chọn đi theo một trong 2 hướng một cách ngẫu nhiên. Tuy nhiên đường đi ngắn hơn làm cho khoảng thời gian di chuyển từ tổ đến nguồn thức ăn rồi quay trở lại của mỗi con kiến theo con đường đó cũng ngắn hơn và do đó mật độ di chuyển qua lại của đàn kiến tại mỗi vị trí của con đường ngắn sẽ cao hơn con đường dài. Do mật độ qua lại lớn hơn dẫn đến kết quả là nồng độ pheromone trên con đường ngắn càng ngày càng cao hơn con đường dài. Kết quả cuối cùng là đàn kiến ngày càng từ bỏ con đường dài và đi theo con dường ngắn. Đến một lúc nào đó sẽ không còn con kiến nào đi theo con đường dài nữa mà tất cả đều đi theo con đường ngắn.



Thuật toán dựa trên hoạt động của đàn kiến có một số biến thể. Dạng đơn giản nhất gọi là AS (Ant System). Thuật toán này chỉ dùng để giải quyết bài toán tìm đường. Ở mức cao hơn là thuật toán ACO (Ant Colony Optimization).

1. Từ những con kiến trong tự nhiên tới thuật toán ACO.

Thuật toán ACO lấy ý tưởng từ việc kiếm thức ăn của đàn kiến ngoài thực tế để giải quyết các bài toán tối ưu tổ hợp. Chúng dựa trên cơ sở một đàn kiến nhân tạo, chúng được tính toán tìm kiếm thức ăn nhờ mùi lạ nhân tạo.

Cấu trúc cơ bản của thuật toán ACO: trong mỗi thuật toán, tất cả kiến đi xây dựng cách giải quyết bài toán bằng cách xây dựng một đồ thị. Mỗi cạnh của đồ thị miêu tả các bước kiến có thể đi được kết hợp từ hai loại thông tin hướng dẫn kiến di chuyển:

Thông tin kinh nghiệm (heuristic information): giới hạn kinh nghiệm ưu tiên di chuyển từ nút r tới s…của cạnh a_{rs}. Nó được đánh dấu bởi η_{rs}.

Thông tin này không được thay đổi bởi kiến trong suốt quá trình chạy thuật toán.

Thông tin mùi lạ nhân tạo (artificial pheromone trail information), nó giới hạn "nghiên cứu sự thèm muốn" của chuyển động là kiến nhân tạo và bắt chước mùi lạ thực tế của đàn kiến tự nhiên. Thông tin này bị thay đổi trong suốt quá trình thuật toán chạy phụ thuộc vào cách giải quyết được tìm thấy bởi những con kiến. Nó được đánh dấu bởi τ_{rs}.

Giới thiệu các bước ảnh hưởng từ những con kiến thật vào ACO. Có hai vấn đề cần chú ý:

- Chúng trừu tượng hoá vài mô hình thức ăn của kiến ngoài thực tế để tìm ra đường đi tìm kiếm thức ăn ngắn nhất.

- Chúng bao gồm vài đặc điểm không giống với tự nhiên nhưng lại cho phép thuật toán phát triển chứa đựng cách giải quyết tốt tới bài toán bị cản (ví dụ: sử dụng thông tin kinh nghiệm để hướng dẫn chuyển động của kiến).

Cách thức hoạt động cơ bản của một thuật toán ACO như sau: m kiến nhân tạo di chuyển, đồng thời và không đồng bộ, qua các trạng thái liền kề của bài toán. Sự di chuyển này theo một tập quy tắc làm cơ sở từ những vùng thông tin có sẵn ở các thành phần (các nút). Vùng thông tin này bao gồm thông tin kinh nghiệm và thông tin mùi lạ để hướng dẫn tìm kiếm. Qua sự di chuyển trên đồ thị kiến xây dựng được cách giải quyết. Những con kiến sẽ giải phóng mùi lạ ở mỗi lần chúng đi qua một cạnh (kết nối) trong khi xây dựng cách giải quyết (cập nhật từng bước mùi lạ trực tuyến). Mỗi lần những con kiến sinh ra cách giải quyết, nó được đánh giá và nó có thể tạo luồng mùi lạ là hoạt động của chất lượng của cách giải quyết của kiến (cập nhật lại

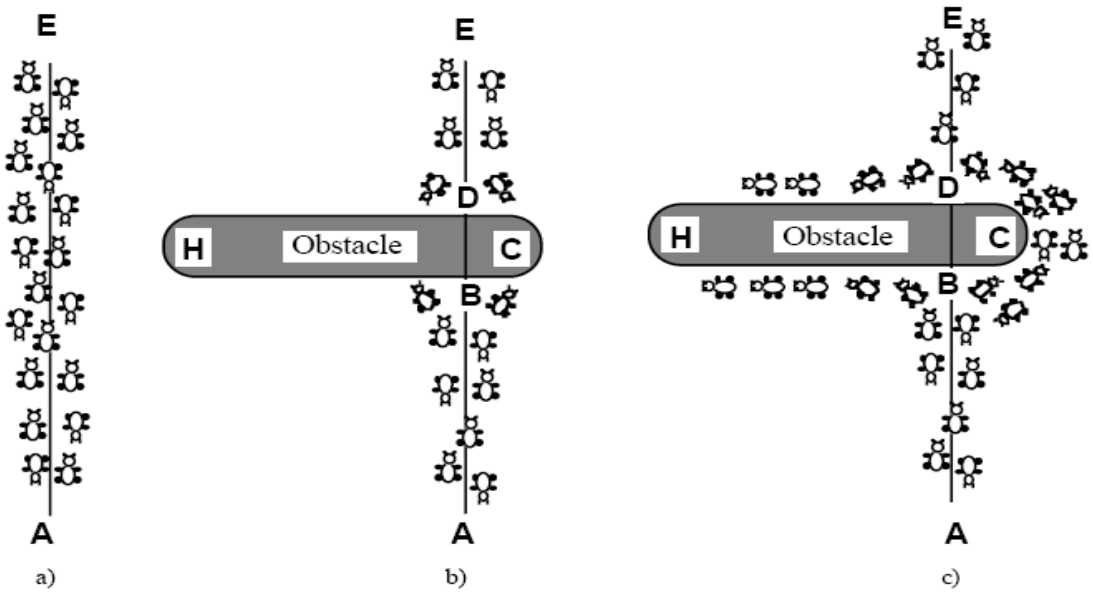
mùi lạ trực tuyến). Thông tin này sẽ hướng dẫn tìm kiếm cho những con kiến đi sau.

Hơn thế nữa, cách thức sinh hoạt động của thuật toán ACO bao gồm thêm hai thủ tục, sự bay hơi mùi lạ (*pheromone trail evaporation*) và hoạt động lạ (*daemon actions*). Sự bay hơi của mùi lạ được khởi sự từ môi trường và nó được sử dụng như là một kĩ thuật để tránh tìm kiếm bị dừng lại và cho phép kiến khảo sát vùng không gian mới. Daemon actions là những hoạt động tối ưu như một bản sao tự nhiên để thực hiện những nhiệm vụ từ một mục tiêu xa tới vùng của kiến.

2. Giới thiệu về thuật toán

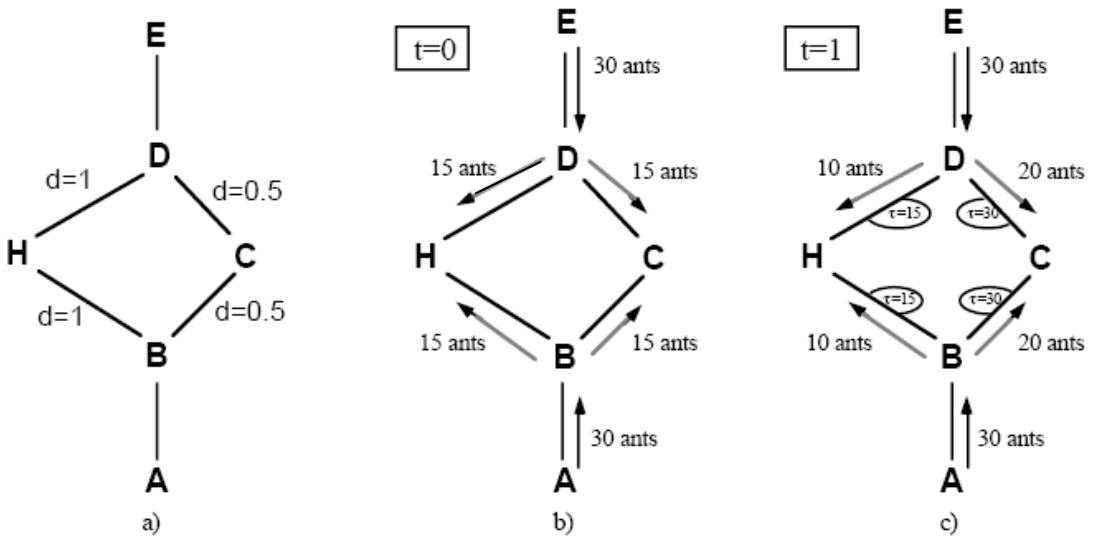
Các thuật toán kiến là các thuật toán dựa vào sự quan sát các bầy kiến thực. Kiến là loại cá thể sống bầy đàn. Chúng giao tiếp với nhau thông qua mùi mà chúng để lại trên hành trình mà chúng đi qua. Mỗi kiến khi đi qua một đoạn đường sẽ để lại trên đoạn đó một chất mà chúng ta gọi là mùi. Số lượng mùi sẽ tăng lên khi có nhiều kiến cùng đi qua. Các con kiến khác sẽ tìm đường dựa vào mật độ mùi trên đường, mật độ mùi càng lớn thì chúng càng có xu hướng chọn. Dựa vào hành vi tìm kiếm này mà đàn kiến tìm được đường đi ngắn nhất từ tổ đến nguồn thức ăn và sau đó quay trở tổ của mình.

Sau đây là ví dụ về luồng đi của đàn kiến thực tế.



Hình 1

a. Kiến đi theo đường thẳng giữa A và E
b. Khi có chướng ngại vật kiến sẽ chọn hướng đi, có hai hướng với khả năng kiến sẽ chọn là như nhau.
c. Trên đường ngắn hơn thì nhiều mùi (pheromone) hơn



Hình 2

Xem hình 2a là giải thích rõ tình huống trong hình 1b.

Giả sử khoảng cách DH=BH=DB qua C và =1, C là điểm nằm giữa B và D(hình 2a). Bây giờ chúng ta xem xét điều gì xảy ra tại những khoảng thời gian rời rạc: t=0, 1, 2… Giả định rằng 30 con kiến mới đi từ A đến B, 30 con từ E đến D, mỗi kiến di chuyển với tốc độ một đơn vị thời gian và khi di chuyển kiến để tại thời điểm t một vệt pheromone với nồng độ là 1. Để đơn giản chúng ta xét lượng pheromone bay hơi hoàn toàn và liên tục trong khoảng thời gian (t+1, t+2).

Tại thời điểm t=0, thì không có vệt mùi nào trên cạnh và có 30 kiến ở B, 30 ở D. Việc lựa chọn đường đi của chúng ta ngẫu nhiên do đó, trung bình từ mỗi nút có 15 con kiến sẽ đi đến H và 15 con sẽ đi đến C (hình 2b)

Tại thời điểm t=1, 30 con kiến mới đi từ A đến B, lúc này nó sẽ chọn hướng đến C hoặc hướng đến H. Tại hướng đến H có vệt mùi 15 do 15 con kiến đi từ B đến H, tại hướng đến C có vệt mùi 30 do 15 kiến đi từ B đến D và 15 con đi từ D đến B thông qua C (hình 2c). Do đó khả năng kiến hướng đến chọn đường đến C, do đó số kiến mong muốn đi đến C sẽ gấp đôi số kiến đi đến H (20 con đến C và 10 con đến H). Tương tự như vậy cho 30 con kiến mới đi từ D đến B.

Quá trình sẽ liên tục cho đến khi tất cả kiến sẽ chọn đường đi ngắn nhất.

Trên đây chúng ta mô tả hành vi tìm kiếm của bầy kiến thực.Sau đây , chúng ta sẽ tìm hiểu sâu hơn về các thuật toán kiến.

Thuật toán tối ưu bầy kiến (ACO) nghiên cứu các hệ thống nhân tạo dựa vào hành vi tìm kiếm của bầy kiến thực và được sử dụng để giải quyết các vấn đề về tối ưu rời rạc.Thuật toán bầy kiến siêu tìm kiếm(ACO meta_heuristic) lần đầu tiên được Dorigo, Di Caro và Gambardella đề xuất vào năm 1999.

Metaheuristic là một tập các khái niệm về thuật toán được sử dụng để xác định các phương thức tìm kiếm thích hợp cho một tập các vấn đề khác nhau. Hay nói cách khác, một siêu tìm kiếm (meta-heuristic) có thể coi là một phương thức tìm kiếm đa năng.

ACO là một meta-heuristic, trong đó một tập các con kiến nhân tạo phối hợp tìm kiếm các giải pháp tốt cho các vấn đề về tối ưu rời rạc. Sự phối hợp là yếu tố cối lõi của các thuật toán ACO. Các con kiến nhân tạo liên lạc với nhau thông qua trung gian mà ta thường gọi là mùi.

Các thuật toán ACO được sử dụng để giải quyết các vấn đề về tối ưu tổ hợp tĩnh và động. Các vấn đề tĩnh là các vấn đề mà ở đó các đặc tính của vấn đề là không thay đổi trong suốt quá trình giải quyết vấn đề. Còn các vấn đề động thì ngược lại là một hàm các tham số mà giá trị của nó là động hay thay đổi trong quá trình giải quyết vấn đề, ví dụ bài toán người đưa thư là một vấn đề dynamic problem

Hệ thống ACO lần đầu tiên được Marco Dorigo giới thiệu trong luận văn của mình vào năm 1992, và được gọi là Hệ thống kiến (Ant System, hay AS). AS là kết quả của việc nghiên cứu trên hướng tiếp cận trí tuệ máy tính nhằm tối ưu tổ hợp mà Dorigo được hướng dẫn ở Politecnico di milano với sự hợp tác của Alberto Colorni và Vittorio Maniezzo. AS ban đầu được áp dụng cho bài toán người du lịch (TSP) và QAP

Cũng vào năm 1992, tại hội nghị sự sống nhân tạo lần đầu tiên ở châu Âu , Dorigo và các cộng sự đã công bố bài: sự tối ưu được phân bố bởi đàn kiến.

Tiếp theo tại hội nghị quốc tế thứ hai về giải quyết các vấn đề song song trong tự nhiên ở Hà Lan (1992), ông và các cộng sự đã công bố bài: nghiên cứu về các đặc tính của một giải thuật kiến.

Kể từ năm 1995 Dorigo, Gambardella và Stützle đã phát triển các sơ đồ AS khác nhau. Dorigo và Gambardella đã đề xuất Hệ thống bầy kiến (Ant

Colony System, hay ACS) trong khi Stützle and Hoos đề xuất MAX-MIN Ant System (MMAS). Tất cả đều áp dụng cho bài toán người du lịch đối xứng hay không đối xứng và cho kết quả mỹ mãn. Dorigo, Gambardella and Stützle cũng đề xuất những phiên bản lai của ACO với tìm kiếm địa phương.

Vào năm 1995, L.M. Gambardella và M. Dorigo đã đề xuất hệ thống Ant-Q, là một cách tiếp cận học tăng cường cho cho bài toán TSP.Và nó được áp dụng trong Học Máy.

Tiếp đó, vào năm 1996, trong bài báo công nghệ của mình tại Bruxelles M. Dorigo và L.M. Gambardella đã công bố hệ thống Ant Conoly System. Đây là hệ thống đề cập đến cách học phối hợp áp dụng cho bài toán TSP .

Cũng trong năm 1996 này, T. Stützle và H. H. Hoos đã đề xuất hệt thống Max-Min Ant System . Đây là một hệ thống cải tiến hệ thống AntSystem ban đầu và được đánh giá là hệ thống tính toán trong tương lai.

Sau đó, vào năm 1997, G. Di Caro và M. Dorigo đã đề xuất hệ thống AntNet. Đây là cách tiếp cận về định hướng sự thích nghi. Và phiên bản cuối cùng của hệ thống AntNet về điều khiển mạng truyền thông đã được công bố vào năm 1998.

Cũng trong năm 1997, hệ thống Rank-based Ant System, một hệ thống cải tiến hệ thống kiến ban đầu về nghiên cứu hệ thống tính toán đã được đề xuất bởi B. Bullnheimer, R. F. Hartl và C. Strauss. Phiên bản cuối cùng của hệ thống này được công bố vào năm 1999.

Vào năm 2001, C. Blum, A. Roli, và M. Dorigo đã cho công bố về hệ thống kiến mới là Hyper Cube – ACO. Phiên bản mở rộng tiếp đó đã được công bố vào năm 2004.

Hầu hết các nghiên cứu gần đây về ACO tập trung vào việc phát triển các thuật toán biến thể để làm tăng hiệu năng tính toán của thuật toán Ant System ban đầu.

Trên đây là sơ lược chung về các thuật toán kiến, mục tiếp theo sẽ mô tả về sơ đồ chung của thuật toán kiến.

3. Sơ đồ chung thuật toán đàn kiến

```
Procedure ACO
    Initial();
    While (!ĐK dừng) do
        ConstructSolutions();
        LocalSearch();  /*Tuỳ ý, có thể có hoặc không
        UpdateTrails();
    End;
End;
```

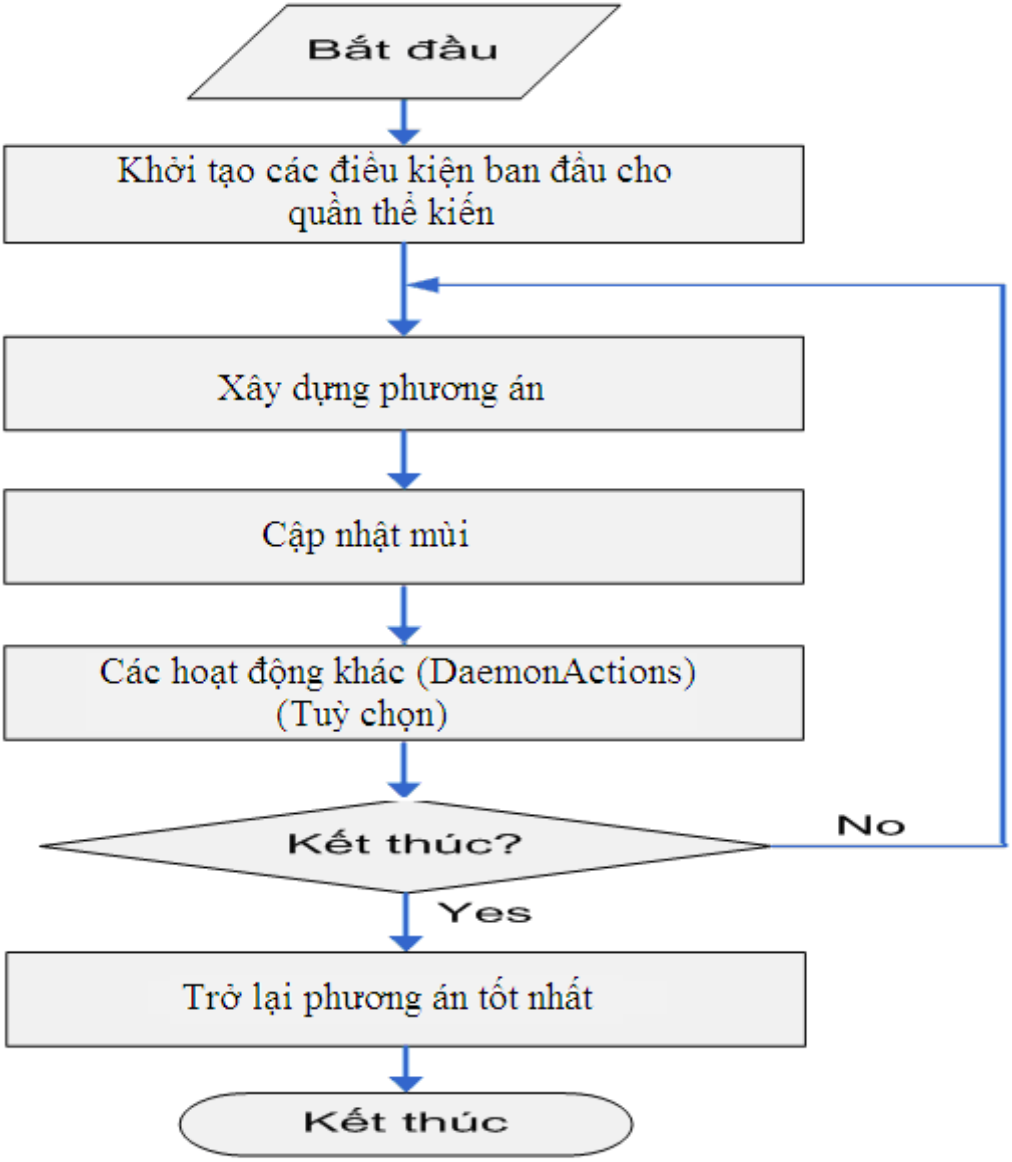
trong đó:

ĐK dừng (tức là điều kiện dừng) là điều kiện đạt được khi thuật toán ở trạng thái kết thúc. Với bài toán người đưa thư thì ĐK dừng là điều kiện đạt được khi số vòng lặp của thuật toán = số vòng lặp lớn nhất do người dùng tự định nghĩa hoặc là tất cả đàn kiến đều đi theo một đường (tức là đường đi ngắn nhất).

ConstrucSolutions() là hàm xây dựng một giải pháp có thể theo phương pháp siêu tìm kiếm(meta-heuristic), với bài toán người đưa thư thì đó là hàm xây dựng chu trình cho mỗi kiến .

UpdateTrails() là hàm cập nhật mùi cho hành trình mà kiến đã đi qua.

LocalSearch() là hàm tìm kiếm địa phương, giúp tìm ra tối ưu cục bộ.



Hình 3. Sơ đồ chung của thuật toán bầy kiến

4. Các bước giải quyết bài toán đàn kiến

Từ thuật toán trên ta có thể rút ra các bước giải quyết một bài toán ứng dụng với thuật toán đàn kiến:

Bước 1: Thể hiện bài toán trong khung của tập các thành phần và sự chuyển đổi hoặc bởi một đồ thị được đánh dấu bởi kiến đề xây dựng cách giải quyết.

Bước 2: Định nghĩa thích hợp cho mùi lạ τ_{rs} là một xu hướng quyết định. Đó là một bước chủ yếu trong việc hình thanhg thuật toán ACO và xác định rõ mùi lạ không là một nhiệm vụ tầm thường và nó tính toán yêu cầu bên trong của bài toán sau đáp án.

Bước 3: Định nghĩa thích hợp kinh nghiệm cho mỗi quyết định để một con kiến có thể xây dựng cách giải quyết, ví dụ: định nghĩa thông tin kinh nghiệm η_{rs} kết hợp mỗi thành phần hoặc trạng thái chuyển đổi. Thông tin kinh nghiệm chủ yếu cho việc tìm kiếm tốt nếu thuật toán tìm kiếm vùng không có sẵn hoặc không thể ứng dụng.

Bước 4: Nếu thực hiện được, tạo ra một vùng thuật toán tìm kiếm hiệu quả cho bài toán sau đáp án bởi vì kết quả của nhiều ứng dụng ACO cho bài toán tổ hợp tối ưu NP-hard thể hiện qua sự tìm kiếm tốt nhất đạt được khi ACO có vùng lạc quan.

Bước 5: Lựa chọn một thuật toán ACO và ứng dụng nó vào những bài toán cần giải quyết.

Bước 6: Các tham số phù hợp của thuật toán ACO. Một điểm bắt đầu tốt cho tham số phù hợp là sử dụng cài đặt tham số để tìm kiếm tốt khi ứng dụng thuật toán ACO vào bài toán đơn giản hoặc các bài toán khác nhau. Một vấn đề khác chi phối thời gian trong nhiệm phù hợp là để sử dụng thủ tục động cho tham số phù hợp.

Nó nên xoá các bước tiếp có thể chỉ đưa ra một hướng dẫn sử dụng thuật toán ACO. Thêm nữa, việc sử dụng là sự kết hợp các quá trình ở đó với

vài bài toán sâu hơn và hoạt động của thuật toán, vài lựa chọn ban đầu cần phải sửa lại. Cuối cùng, chúng ta muốn trên thực tế, điều quan trọng nhất của các bước là đầu tiên phải khớp bởi vì lựa chọn tồi ở trạng thái này tính không thể tính với một tham số gốc phù hợp tốt.

5. Các sơ đồ thuật toán khác phát triển trên mô hình ACO

Nhiều thuật toán đã được đưa ra dựa trên mô hình thuật toán **metaheuristic ACO**. Trong các mô hình đưa ra để giải quyết các bài toán tổ hợp tối ưu **NP-khó** sau đây xin trình bày chi tiết về 5 mô hình. Các mô hình này là phát triển dựa trên mô hình thuật toán ACO cụ thể trình bày ở phần trên. Theo các nghiên cứu cho thấy khi sử dụng thuật toán bầy kiến nói chung các thông tin pheromone và heuristic có thể áp dụng cho các nút hoặc cạnh nối. Trong các thuật toán đưa ra sau đây thì thông tin pheromone và heuristic chỉ gắn với các cạnh mà thôi.

5.1. Thuật toán Ant System (AS)

Được phát triển bởi Dorigo, Maniezzo và Colorni năm 1991, là thuật toán ACO đầu tiên. Ban đầu có 3 biến thể khác nhau là: AS-Density, AS-Quantity và AS-Cycle khác nhau bởi cách thức cập nhật thông tin Pheromone.

Trong đó:

- AS-Density: Thì đàn kiến sẽ đặt thêm pheromone trong quá trình xây dựng lời giải (online step-by-step pheromone update), lượng pheromone để cập nhật là một hằng số.
- AS-Quantity: Thì đàn kiến sẽ đặt thêm pheromone trong quá trình xây dựng lời giải (online step-by-step pheromone update), lượng pheromone để cập nhật là phụ thuộc vào độ mong muốn (thông tin heuristic) với đoạn đường đi qua η_{ij}.

➢ AS-Cycle: Thông tin pheromone sẽ được cập nhật khi lời giải đã hoàn thành (online delayed pheromone update). Đây là mô hình cho kết quả tốt nhất và được coi như là thuật toán AS.

❖ ***Quy tắc di chuyển của kiến***

Trong thuật toán AS, kiến xây dựng một đường đi bắt đầu tại một đỉnh được chọn ngẫu nhiên.

Tại đỉnh i, một con kiến k sẽ chọn đỉnh j chưa được đi qua trong tập láng giềng của i theo công thức sau:

$$p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l \in N_i^k}(\tau_{il})^{\alpha}(\eta_{il})^{\beta}}, j \in N_i^k \qquad (2.1)$$

Trong đó:

p_{ij}^{k} : xác suất con kiến k lựa chọn cạnh (i,j)
τ_{ij} : nồng độ vết mùi trên cạnh (i,j)
α : hệ số điều chỉnh ảnh hưởng của τ_{ij}
η_{ij} : thông tin heuristic giúp đánh giá chính xác

Sự lựa chọn của con khi quyết định đi từ đỉnh i qua đỉnh j và được tính theo công thức:

$$\eta_{ij} = \frac{1}{d_{ij}} \qquad (2.2)$$

d_{ij} : khoảng cách giữa đỉnh i và đỉnh j
β : hệ số điều chỉnh ảnh hưởng của η_{ij}
N_i^k : tập các đỉnh láng giềng của i mà con kiến k chưa đi qua

❖ ***Quy tắc cập nhật thông tin mùi***

Trong quá trình di chuyển tìm đường đi của đàn kiến, chúng thực hiện cập nhật thông tin mùi trên những đoạn đường mà chúng đã đi qua. Gắn với mỗi cạnh (i,j) nồng độ vết mùi τ_{ij} và thông số heuristic η_{ij} trên cạnh đó.

Ban đầu nồng độ mùi trên mỗi cạnh (i,j) được khởi tạo một hằng số c, hoặc được xác định theo công thức:

$$\tau_{ij} = \tau_0 = \frac{m}{C^{nn}}, \forall(i,j) \qquad (2.3)$$

Việc cập nhập pherpmone được tiến hành như sau:

➢ Đầu tiên tất cả pheromone trên các cung sẽ được giảm đi bởi một lượng:

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

Với *p* trong khoảng (0,1) là tốc độ bay hơi của pheromone.

➢ Tiếp theo mỗi con kiến trong đàn sẽ đặt thêm một lượng thông tin pheromone trên những cung mà chúng đã đi qua trong hành trình của chúng.

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

Trong đó: deta τ_{ij} là lượng pheromone mà con kiến k đặt lên cạnh mà nó đã đi qua và được tính như sau:

$$\Delta\tau_{ij}^{k} = \begin{cases} 1/C^{k}, & \text{nếu kiến k qua cung (i,j)} \\ 0, & \text{ngược lại} \end{cases} \quad (2.6)$$

Với: C^k là độ dài đường đi của con kiến thứ k sau khi hoàn thành đường đi, tức là bằng tổng các cung thuộc đường đi mà kiến đã đi qua.

5.2. Thuật toán Ant Colony System (ACS)

Phát triển từ thuật toán AS

❖ ***Quy tắc di chuyển của kiến***

Trong thuật toán ACS, con kiến k đang ở đỉnh i, việc kiến chọn đỉnh j để di chuyển đến được xác định bằng quy luật như sau:

- Cho q_o là một hằng số cho trước ($0<q_0<1$)
- Chọn ngẫu nhiên một giá trị q trong khoảng [0,1]
- Nếu $q<q_0$ kiến k chọn điểm j di chuyển tiếp theo dựa trên giá trị lớn nhất của thông tin mùi và thông tin heuristic có trên cạnh tương ứng với công thức:

$$j = \arg_{l \in N_i^k} \max\left(\tau_{il}(\eta_{il})^{\beta}\right) \quad (2.7)$$

- Nếu $q>q_0$ kiến k sẽ chọn đỉnh j chưa được đi qua trong tập láng giềng của I theo một quy luật phân bổ xác xuất được xác định theo công thức sau:

$$p_{ij}^{k} = \frac{(\tau_{ij})^{\alpha}(\eta_{ij})^{\beta}}{\sum_{l \in N_i^k}(\tau_{il})^{\alpha}(\eta_{il})^{\beta}}, j \in N_i^k$$

❖ ***Quy tắc cập nhật thông tin mùi***

Cập nhật thông tin mùi toàn cục:

Một con kiến có đường đi tốt nhất sau mỗi lần lặp thì được phép cập nhật thông tin pheromone. Việc cập nhật được thực hiện theo công thức sau:

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

Cập nhật thông tin mùi cục bộ:

Công thức sau:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\tau_0 \qquad (2.9)$$

Với *p*: là tham số bay hơi năm trong khoảng (0,1)

$\tau_0 = 1/(nC_{nm})$

n: là số dỉnh hay là số thành phố

C_{nm}: chiều dài hành trình cho bởi phương pháp tìm kiếm gần nhát

5.3. Thuật toán Max–Min Ant System (MMAS)

Được phát triển bởi Stutzle và Hooss vào năm 1996, được mở rộng lên từ hệ thống AS. Luật di chuyển của kiến được thực hiện tương tự như trong thuật toán ACS

❖ ***Quy tắc cập nhật thông tin mùi***

Thuật toán MMAS thực hiện việc cập nhật thông tin mùi khi toàn bộ kiến trong đàn hoàn thành lời giải và lượng thông tin mùi chỉ cập nhật trên các cạnh thuộc lời giải tối ưu nhất. Ban đầu cũng thực hiện bay hơi thông

tin mùi trên các cạnh thuộc lơi giải tối ưu với một lượng được xác định tại công thức (2.4).

Lượng pheromone trên một cạnh được xác định như sau:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best}$$

với $$\Delta\tau_{ij}^{best} = \begin{cases} 1/C_{best} & \text{nếu kiến qua cạnh (i,j)} \\ 0 & \text{ngược lại} \end{cases}$$

C_{best} là độ dài đường đi ngắn nhất của con kiến thứ k sau khi cả đàn hoàn thành đường đi.

❖ ***Khởi tạo và khởi tạo lại thông tin mùi***

Thuật toán MMAS đã thêm vào giá trị cận trên và giá trị cận dưới cho thông tin pheromone gọi là τ_{min} và τ_{max}

Sau mỗi lần cập nhật giá trị thông tin mùi τ_{ij} nếu $\tau_{ij} < \tau_{min}$ thì sẽ gán $\tau_{ij} = \tau_{min}$ và nếu $\tau_{ij} > \tau_{max}$ thì gán $\tau_{ij} = \tau_{max}$

Giá trị cận trên τ_{max} thường được thiết lập với công thức sau:

$$\tau_{max} = {}^{1}/_{\rho C_{best}}$$

Giá trị cận dưới τ_{min} được xác định bằng công thức:

$$\tau_{min} = \tau_{max}/2n.$$

5.4. Thuật toán Rank-Based Ant System (RBAS)

Đây cũng là một thuật toán được mở rộng phát triển từ hệ thống AS đưa ra bởi Bullnheimer, Hartl và Strauss vào năm 1997. Thuật toán này đưa vào ý tưởng xếp hạng cho các lời giải khi thực hiện cập nhật pheromone. Cụ thể như sau:

- Đầu tiên, m con kiến được xếp hạng theo thứ tự giảm dần dựa theo chất lượng lời giải mà nó thu được. Ví dụ: (S_1, S_2, … S_{m-1}, S_m) trong đó S_1 là phương án tốt nhất.

- Pheromone chỉ được đặt thêm trên các cung của б -1 con kiến có lời giải tốt nhất. Lượng pheromone cũng phụ thuộc trực tiếp vào thứ hạng sắp xếp của con kiến.
- Các đoạn đường đi của lời giải tốt nhất được nhận thêm một lượng pheromone phụ thuộc vào chất lượng lời giải.

Các công thức như sau:

$$\tau_{rs} \leftarrow \tau_{rs} + \sigma.\Delta\tau_{rs}^{gb} + \Delta\tau_{rs}^{rank}.$$

Trong đó $\Delta\tau_{rs}^{gb} = \begin{cases} f(C(S_{global-best})), & a_{rs} \in S_{global-best} \\ 0, & \text{trái l¹ i.} \end{cases}$

Và

$$\Delta\tau_{rs}^{rank} = \begin{cases} \sum_{\mu=1}^{\sigma-1}(\sigma-\mu).f\left(C(S_{\mu}^{'})\right), & a_{rs} \in S_{\mu}^{'} \\ 0, & \text{trái l¹ i.} \end{cases}$$

Tóm tắt thủ tục cập nhật pheromone của thuật toán này:

```
1  Procedure daemon_actions
2      for each S_k do local_search (S_k)   {optional}
3      rank (S_1,...,S_m) in decreasing order of solution
       quality into (S'_1,...,S'_m)
4      if (best (S'_1, S_global-best))
5          S_global-best = S'_1
6      end if
7      for μ=1 to σ−1 do
8          for each edge a_rs ∈ S'_μ do
9              τ_rs = τ_rs + (σ−μ).f(C(S'_μ))
10         end for
11     end for
12     for each edge a_rs ∈ S_global-best do
13             τ_rs = τ_rs + σ.f(C(S_global-best))
```

14 **end for**
15 **end Procedure**

5.5. Thuật toán Best-Worst Ant System(BWAS)

Thuật toán được đưa ra bởi Cordon vào năm 1999. Thuật toán này bao gồm một thuật toán mở rộng khác của AS là MMAS (về luật di chuyển và việc bay hơi của pheromone). Bên cạnh đó trong thuật toán này còn quan tâm tới của việc tối ưu cục bộ một cách hệ thống để nâng cao chất lượng lời giải của con kiến. Trong thuật toán BWAS có 3 **daemon action** thêm vào gồm có:

- Đầu tiên, áp dụng luật có tên *best-worst pheromone update* để tăng cường pheromone trên các đoạn đường đi qua bởi lời giải tốt nhất toàn cục (global best solution). Thêm vào đó luật này sẽ phạt những cạnh của lời giải tồi nhất trong lần lặp $S_{current\text{-}worst}$.
- Áp dụng *Pheromone trail mutation* để đi theo các hướng khác nhau trong quá trình tìm kiếm.
- BWAS có cơ chế khởi tạo lại thông tin pheromone khi thuật toán bị đình trệ, bằng cách thiết lập pheromone trail cho tất cả các thành phần bằng τ_0 .

Mô hình thủ tục **Daemon action** của thuật toán BWAS như sau:

1 **Procedure** daemon_actions
2 **for each** S_k **do** local_search (S_k)
3 $S_{current-best}$ = best_solution (S_k)
4 **if** (best ($S_{current-best}, S_{global-best}$))
5 $S_{global-best} = S_{current-best}$
6 **end if**
7 **for** each edge $a_{rs} \in S_{global-best}$ **do**
8 $\tau_{rs} = \tau_{rs} + p.f\left(C(S_{global-best})\right)$
9 $sum = sum + \tau_{rs}$

10 **end for**

11 $\tau_{threshold} = \frac{sum}{|S_{globad-best}|}$

12 $S_{current-worst} = worst\_solution \quad (S_k)$

13 **for** each edge $a_{rs} \in S_{current-worst}$ and $a_{rs} \notin S_{global-best}$ **do**

14 $\tau_{rs} = (1-p).\tau_{rs}$

15 **end for**

16 $mut = mut(it, \tau_{threshold})$

17 **for** each nút / component $r \in \{1,...,l\}$ **do**

18 z = generate_random_value_in_[0,1]

19 **if** $(z <= P_m)$

20 s = generate_random_value_in_[1,…, 1]

21 a = generate_random_value_in_[0,1]

22 if $(a = 0) \quad \tau_{rs} = \tau_{rs} + mut$

23 else $\tau_{rs} = \tau_{rs} - mut$

24 **end if**

25 **end for**

26 **if** (stagnation_condition)

27 **for** each a_{rs} **do** $\tau_{rs} = \tau_0$

28 **end if**

29 **end Procedure**

Mục này chỉ đưa ra 5 mô hình thuật toán ACO phát triển từ hệ thống Ant System. Nhưng đó chỉ là một số các dạng tiêu biểu của thuật toán ACO, còn tồn tại rất nhiều các biến thể khác. Và trong đồ án sẽ áp dụng thuật toán theo mô hình hệ thống MMAS để giải bài toán CPMP. Mô hình thuật toán MMAS là một trong các thuật toán hiệu quả nhất của các thuật toán bầy kiến.

6. Thuật toán đàn kiến song song

Từ sơ đồ giải thuật ta nhận thấy các cá thể kiến trong giải thuật là rất độc lập với nhau và vì vậy ý tưởng song song đơn giản và hiệu quả nhất là phân chia kiến ra các bộ xử lý khác nhau , các bộ xử lý mạnh có thể nhận nhiều

kiến, các bộ xử lý yếu hơn sẽ nhận ít kiến hơn. Việc phân chia như vậy sẽ làm tăng hiệu suất của giải thuật,tuy nhiên khi tới bước cập nhập ma trận mùi các bộ xử lý cần phải trao đổi dữ liệu với nhau, tùy vào thông tin được trao đổi và mô hình các bộ xử lý mà ta có các kiểu thuật toán song song khác nhau và các tham số khác nhau cho giải thuật.

All-to-all topology:Các cụm kiến gửi thông tin tới tất cả các cụm kiến khác

(Directed or undirected) ring topology: trong mô hình directed ring colony cụm kiến (z +1 mod ***p)*** + 1 là hàng xóm của cụm ***i*** cho tất cả các cụm kiến và trong mô hình undirected ring colony cụm kiến ***(i*** - 1 mod ***p)*** + 1 là hàng xóm của cụm kiến
i cho tất cả các cụm kiến.

Hypercube topology: Mô hình này yêu cầu có ***p*** = ***2^k*** cụm kiến và mỗi cụm kiến I là hàng xóm với cụm kiến j nếu và chỉ nếu kiểu biểu diễn nhị phân của i và j chỉ khác nhau 1 bit. Vì vậy mỗi cụm kiến chỉ có k hàng xóm.

Random topology:Trong mô hình này các hàng xóm của mỗi cụm kiến được đinh nghĩa một cách ngẫu nhiên trong mỗi bước trao đổi thông tin .Có nhiều phương thức xác định hàng xóm ngẫu nhiên trong trường hợp này

Cũng có thể phân biệt các giải thuật với nhau bằng loại thông tin gửi nhận qua mỗi bước lặp.

Lời giải: Trong chiến thuật này các lời giải tố đã tìm ra sẽ được gửi đi tới các cụm kiến khác .có nhiều kiểu lời giải có thể được gửi đi

Kiến: Lời giải của một con kiến từ lần lặp này được gửi tới cụm kiến khác, thông thường đây là lời giải của con kiến tốt nhất

Lời giải toàn cục tốt nhất. Lời giải tốt nhất của các cụm kiến được gửi đi cho tất cả các cụm kiến

Lời giải của hàng xóm tốt nhất . Lời giải tốt nhất của các cụm kiến được gửi tới các hàng xóm

Lời giải cục bộ tốt nhất. Lời giải cục bộ tốt nhất được gửi đi tới các hàng xóm

Vector mùi . Thay vì gửi lời giải thì vector mùi sẽ được gửi sau mỗi bước lặp.

CHƯƠNG III: MỘT SỐ ỨNG DỤNG VỀ THUẬT TOÁN

1. Ứng dụng thuật toán ACO

Thuật toán ACO được ứng dụng cho một số lượng lớn bài toán tối ưu tổ hợp. Những ứng dụng hiện nay của ACO chia thành hai lớp ứng dụng: lớp bài toán tối ưu tổ hợp NP-hard cho công nghệ cũ thường ít thức ăn. Đặc tính thành công nhất của ứng dụng ACO tới những bài toán mà ở đó kiến kết hợp với vùng tìm kiếm để có cách giải quyết tốt. Lớp ứng dụng thứ hai là bài toán tìm đường đi ngắn nhất, ở đó khoảng cách bài toán giải quyết thay đổi ở thời gian thực thi bài toán. Những thay đổi này có thể ảnh hưởng không đổi của bài toán như đã có sẵn, Nếu ảnh hưởng bị lẫn lộn, đặc tính được coi như chi phí cạnh, thay đổi theo thời gian. Trong trường hợp này, thuật toán mô phỏng

theo bài toán động. Lớp ứng dụng thứ hai của ACO để kết nối đường thông tin.

Thay cho việc đưa chi tiết các ứng dụng khác nhau của ACO, chúng ta mô tả ngắn lịch sử phát triển của ứng dụng, tổng quan mở rộng trong ứng dụng của ACO. Bài toán tổ hợp đầu tiên được giải quyết bởi thuật toán ACO là bài toán người du lịch (TSP) bởi vì bài toán này được biết nhiều nhất của NP-hard, tìm đường đi ngắn nhất, để dễ dàng mô phỏng sự sinh hoạt của loài kiến để giải quyết nó. Từ khi ứng dụng đầu tiên của AS trong luận án của Dorigo'sPhD năm 1991, nó trở thành một đánh giá chung của vài đóng góp tốt hơn trong mô hình thực thi ACO hơn AS.

Theo trình tự, hai ứng dụng tiếp theo bài toán nhiệm vụ của phương trình bậc hai (QAP) và bài toán lập lịch cho cửa hàng năm 1994. Giữa các ứng dụng tiếp theo là đường thông tin ứng dụng đầu tiên, bắt đầu năm 1996 với công việc của Schoonderwoerd và công việc AntNet bởi Di Caro và Dorigo. Năm 1997, sau một năm công bố của nhà báo đầu tiên cho ACO năm 1996, số ứng dụng ACO bắt đầu tăng nhanh. Ứng dụng sớm nhất từ 1997 bao gồm bao toán đường xe cũ (vehicle routing), trình tự chuỗi, lập lịch trình cho cửa hàng và bài toán đồ hoạ. Sau đó, nhiều tác giả khác nhau đã sử dụng ACO metaheuristics để giải quyết số lượng lớn bài toán tối ưu tổ hợp như chuỗi chung ngắn nhất, tổng quát nhiệm vụ, tập che phủ, nhiều túi và bài toán hợp lý…Thú vị nhất của người đọc là thấy được tổng kết ứng dụng có sẵn năm 2000. Từ phần ứng dụng trước đó, ACO gần như được sử dụng cho mục đích kĩ thuật, cụ thể để thiết kế nghiên cứu thuật toán cho đọc giả biểu diễn cấu trúc như tập quy tắc lôgic classical, quy tắc logic fuzzy và thông tin Bayesian, đưa ra những kết quả hứa hẹn.

Ngày nay, ACO gần kết quả trạng thái cho vài bài toán để nó áp dụng cho: QAP, trình tự chuỗi, đường đi, lập lịch và thông tin gói điều khiển…Kết quả tính toán sẵn có cho bài toán khác thường rất tốt và đóng tất cả trạng thái lại là điều đáng chú ý, bởi vì nhiều bài toán sẵn sàng bị tấn công mạnh mẽ bởi nỗ lực tìm kiếm. Bên cạnh đó, ACO metaheuristics được áp dụng để giải quyết bài toán mới của thế giới với kết quả đầy hứa hẹn.

2. Ví dụ minh họa

❖ Bài toán người du lịch

• *Phát biểu bài toán*

Một người du lịch đi thăm n thành phố T_1…T_n. Xuất phát từ thành phố nào đó, người du lịch muốn đi thăm tất cả các thnàh phố còn lại, mỗi thành phố đi đúng một lần rồi trở lại thành phố xuất phát. Gọi Cij là chi phí từ T_i đến T_j. Tìm hành trình với tổng chi phí nhỏ nhất.

• *Tư tưởng giải quyết bài toán người du lịch bằng thuật toán kiến*

- Kiến đi sau sử dụng vết mùi lạ (lớp Trail) và kinh nghiệm (lớp Heuristics) của kiến đi trước để tìm kiếm con đường đi cho mình, sau đó con đường tốt hơn sẽ được cập nhật lại.

- Cứ như vậy sẽ cho ta kết quả con đường đi ngắn nhất cần tìm.

• *Thuật toán đàn kiến giải bài toán người du lịch*

Áp dụng thuật toán kiến vào bài toán người du lịch với các thông số của kiến như sau:

m: số lượng kiến

τ_{ij}: nồng độ vết mùi trên cạnh (i,j)

p : là tham số bay hơi nằm trong khoảng (0,1)

p_{ij}^k : xác suất con kiến k lựa chọn cạnh (i,j)

α: hệ số điều chỉnh ảnh hưởng của τ_{ij}

η_{ij} : thông tin heuristic giúp đánh giá chính xác sự lựa chọn của kiến đi từ đỉnh i tới đỉnh j

β : hệ số điều chỉnh ảnh hưởng η_{ij}

N_i^k : tập các đỉnh láng giềng của i mà con kiến k chưa đi qua

$\Delta\tau_{ij}^k$ là lượng pheromone mà con kiến k đặt lên cạnh mà nó đã đi qua

NC: là số bước lặp của thuật toán

S_k: đường đi của kiến k (tập các đỉnh mà kiến k đi qua)

q : là giá trị ngẫu nhiên trong khoảng [0,1]

a_k^i : nhận giá trị True, False tương ứng với con kiến k đã thăm hoặc chưa đi qua đỉnh i

❖ *Các bước xây dựng thuật toán như sau:*

- Đầu vào:

Đồ thị G=(V,E), với tập đỉnh V và tập cạnh E, ma trận trọng số D=d[i,j] với i, j∈ *V*

Số lượng kiến m.

- Đầu ra: đường đi S với khoảng cách ngắn nhất C_s
- Các bước:

Bước 1: Khởi tạo

Khởi tạo các tham số NC, β,α,ρ, số lượng kiến m và số đỉnh n

Khởi tạo độ dài đường đi ngắn nhất C_{best} là hằng số

Tình độ dài đường đi ngắn nhất C_{nn}

Tính giá trị $\tau_{max}=1/\rho C_{nn}$ và $\tau_{min}=\tau_{max}/2$

Khởi tạo giá trị q_0 với ($0 \leq q_0 \leq 1$)

Khởi gắn điều kiện kết thức stop :=1

Thiết lập ma trận pheromone trên tất cả các cạnh

For i:=1 to n do

For j:=1 to n do $\tau_{ij}=\tau_{max}$

Bước 2: Xây dựng đường đi của kiến

Trường hợp hoàn thành xong tất cả các bước lặp: stop >NC thì xuất ra đường đi ngắn nhất và kết thúc

2.1 thiết lập các đỉnh khi kiến chưa đi qua nhận giá trị false

For k:=1 to m do

For i:=1 to n do a_k^i:=false

Thiết lập đường đi của kiến S_k =∅

2.2 Chọn ngẫu nhiên vị trí xuất phát của kiến

For k:=1 to m do

For i:=1 to n do

R←random{1..n}

Bổ sung r vào đường đi S_k:={r}, a_k^r:=true;

Gán độ dài đường đi C_k:=0

2.3 Xác định đỉnh đến tiếp theo của kiến k

- Chọn ngẫu nhiên một giá trị q: q←random{0..1}

- Nếu $q \leq q_0$ kiến k chọn điểm u di chuyển tiếp theo với

$$u = \arg_{l \in N_r^k} \max\left(\tau_{rl}(\eta_{rl})^{\beta}\right)$$

- Nếu $q > q_0$ kiến k sẽ chọn đỉnh u chưa được đi qua trong tập láng giềng của r theo công thức sau:

$$p_{ru}^k = \frac{(\tau_{ru})^{\alpha}(\eta_{ru})^{\beta}}{\sum_{l \in N_r^k}(\tau_{rl})^{\alpha}(\eta_{rl})^{\beta}}, u \in N_r^k$$

- Chọn đỉnh u là đỉnh tiếp theo, bổ sung đỉnh u vào S_k

  S_k:={r,u}
- Tăng độ dài đường đi C_k:=C_k+d_{ru}
- Gán a_k^u:=true

Bước 3: Xác định đường đi ngắn nhất

Ta có C_k là độ dài đường đi của con kiến k với k=[1..m] thu được từ bước 2.

Nếu $C_k < C_{best}$ thì hiệu chỉnh C_{best}:=C_k

Bước 4: Cập nhật thông tin mùi

Tại bước này, chỉ cập nhật thông tin mùi trên đường đi của con kiến k có giá trị C_k nhỏ nhất thu được từ bước 3, tức là giá trị C_{best}

4.1 Bay hơi thông tin mùi trên các cạnh

For i:=1 to n do

For j:=i to n do τ_{ij}:=(1-ρ) τ_{ij}

4.2 Thêm thông tin mùi trên các cạnh thuộc đường đi tốt nhất $S_{k,best}$

Tính giá trị pheromone $\Delta\tau = \frac{1}{Cbest}$

Nếu cạnh (i,j)$\in S_k^{best}$ thì $\tau_{ij} := \tau_{ij} + \Delta\tau$

Kiểm tra thông tin pheromone với cận trên và cận dưới

Nếu $\tau_{ij} < \tau_{min}$ thì $\tau_{ij} = \tau_{min}$

Nếu $\tau_{ij} > \tau_{max}$ thì $\tau_{ij} = \tau_{max}$

Tăng giá trị stop:=stop+1.

❖ **Code bằng C++**

```
#include <cstdlib>
#include <iostream>
#include <string>
#include <sstream>
#include <math.h>
#include <ctime>
using namespace std;

// Constants
#define INVALID -1
#define CITY_AMOUNT 6
#define POPULATION_SIZE 30
#define PHEROMONE_RATE 0.1
#define ALFA 1
#define BETA 2
#define MAX_ITERATIONS 30
#define EVAPORATION_RATE 0.5
#define POSITIVE_CONTS 0.75

struct ant {
        double distance;
        double fitness;
```

```
        int position;
        int route[CITY_AMOUNT];
};

// Variables
ant ants[POPULATION_SIZE];
double pheromone_links[CITY_AMOUNT][CITY_AMOUNT];

int distance_links[CITY_AMOUNT][CITY_AMOUNT] = {
        { INVALID, 7, 4, 3, 11, 1 },
        { 7, INVALID, 2, 8, 10, 8 },
        { 4, 2, INVALID, 9, 9, 3 },
        { 3, 8, 9, INVALID, 5, 4 },
        { 11, 10, 9, 5, INVALID, 3 },
        { 1, 8, 3, 4, 3, INVALID } };

int best_distance = 1000000000;
int worse_distance = 0;
double best_fitness;
double worse_fitness;
int best_route[CITY_AMOUNT];
int worse_route[CITY_AMOUNT];
double average = 0.0;
double variance = 0.0; // phuong sai
double standard_deviation = 0.0; // do lech chuan
int greater_distance = INVALID;

void init_ant(int index);
void seed_initial_pheromone(bool  );
```

```
void get_greater_distance();
bool check_visit(int ant_index, int position);
void build_solution();
void check_best_distance();
void calculate_fitness();
void pheromone_evaporates();
void update_pheromone();
// Auxiliary methods
double calculate_time(clock_t start, clock_t end);
int get_random_number(int from, int to);
void print_route(int ant_index, int route[CITY_AMOUNT], double distance);
void print_pheromone();
string number_to_String(double n);
void print_result();
void calculate_metrics();

int main(int argc, char *argv[]) {
        clock_t time_start = clock();
        // Initializing the random number generator with a temporal seed
        srand(time(0));
        // Initialize the interaction counter
        int iteration = 0;

        // Configuring the initial pheromone designs
        seed_initial_pheromone(false);
        // Taking the longest distance added to 1
        get_greater_distance();

        // Stop condition
```

```
	while (iteration < MAX_ITERATIONS) {
		build_solution();
		check_best_distance();
		calculate_fitness();
		pheromone_evaporates();
		update_pheromone();
		for (int i = 0; i < POPULATION_SIZE; i++) {
			print_route(i, ants[i].route, ants[i].distance);
		}
		iteration++;
	}
	print_pheromone();
	// Printing the final result
	print_result();

	cout << "\nRuntime (ACO): "
			<< calculate_time(time_start, clock()) << " ms" << endl;
	//cin.get();
	return 0;
}

void init_ant(int index) {
	ants[index].distance = 0;
	ants[index].fitness = 0;
	ants[index].position = 0;
	for (int j = 0; j < CITY_AMOUNT; j++) {
		ants[index].route[j] = INVALID;
	}
	// Positioning ant in a random city
```

```
	int random_city = get_random_number(0, (CITY_AMOUNT - 1));
	ants[index].route[0] = random_city;
}

void seed_initial_pheromone(bool random) {
	for (int i = 0; i < CITY_AMOUNT; i++) {
		for (int j = 0; j < CITY_AMOUNT; j++) {
			if (i != j) {
				double random_pheromone = (double) get_random_number(0,
100)
					/ 100.0;
				pheromone_links[i][j] = (random == true) ?
random_pheromone
					: PHEROMONE_RATE;
			} else {
				pheromone_links[i][j] = INVALID;
			}
		}
	}
}

void get_greater_distance() {
	for (int i = 0; i < CITY_AMOUNT; i++) {
		for (int j = 0; j < CITY_AMOUNT; j++) {
			if (distance_links[i][j] > greater_distance) {
				greater_distance = distance_links[i][j];
			}
		}
	}
	greater_distance += 1;
```

```
}

bool check_visit(int ant_index, int position) {
	for (int i = 0; i < CITY_AMOUNT; i++) {
		if (ants[ant_index].route[i] == position) {
			return true;
		}
	}
	return false;
}

void build_solution() {
	// For each ant
	for (int i = 0; i < POPULATION_SIZE; i++) {
		// Initializing the ant
		init_ant(i);

		// As long as it doesn't pass in every city
		while (ants[i].position < (CITY_AMOUNT - 1)) {
			int position = ants[i].position;

			double transition_probability[CITY_AMOUNT];

			double link_rate_sum = 0;
			// Summing up pheromone and heuristic rates
			for (int j = 0; j < CITY_AMOUNT; j++) {
				// If the city already visited, do not enter the analysis
				if (check_visit(i, j) == false) {
					if (pheromone_links[position][j] >= 0
```

```
                                        and distance_links[position][j] >= 0) {
                                    link_rate_sum
                                            +=
pow(pheromone_links[position][j], ALFA)
                                                    * pow(
        greater_distance
                - distance_links[position][j],
        BETA);
                            }
                    }
                }
                // Calculating the probability of transition
                for (int j = 0; j < CITY_AMOUNT; j++) {
                    // If the city already visited, do not enter the analysis
                    if (check_visit(i, j) == false) {
                        if (pheromone_links[position][j] >= 0
                                        and distance_links[position][j] >= 0) {
                            transition_probability[j] = (pow(
                                    pheromone_links[position][j],
ALFA) * pow(
                                    greater_distance -
distance_links[position][j],
                                    BETA)) / link_rate_sum;
                        } else {
                            transition_probability[j] = 0;
                        }
                    } else {
                        transition_probability[j] = 0;
```

```
                        }
                    }

                    // Doing roulette
                    double roulette = (double) get_random_number(0, 100) / 100.0;
                    double minor = 0;
                    double major = 0;

                    // Selecting the next node
                    for (int j = 0; j < CITY_AMOUNT; j++) {
                        // If the city already visited, do not enter the analysis
                        if (check_visit(i, j) == false) {
                            major += transition_probability[j];
                            if (roulette >= minor and roulette <= major) {
                                ants[i].route[position + 1] = j;
                                ants[i].distance += distance_links[position][j];
                                ants[i].position += 1;
                                break;
                            } else {
                                minor = major;
                            }
                        }
                    }
                }
            }
        }

void check_best_distance() {
        for (int i = 0; i < POPULATION_SIZE; i++) {
```

```
                if (ants[i].distance < best_distance) {
                        best_distance = ants[i].distance;
                        for (int j = 0; j < CITY_AMOUNT; j++) {
                                best_route[j] = ants[i].route[j];
                        }
                } else if (ants[i].distance > worse_distance) {
                        worse_distance = ants[i].distance;
                        for (int j = 0; j < CITY_AMOUNT; j++) {
                                worse_route[j] = ants[i].route[j];
                        }
                }
        }
}

void calculate_fitness() {
        for (int i = 0; i < POPULATION_SIZE; i++) {
                double fitness = (double) ants[i].distance / (double) best_distance;
                if (fitness < best_fitness) {
                        best_fitness = fitness;
                } else if (fitness > worse_fitness) {
                        worse_fitness = fitness;
                }
                ants[i].fitness = fitness;
        }
}

void pheromone_evaporates() {
        for (int i = 0; i < CITY_AMOUNT; i++) {
                for (int j = 0; j < CITY_AMOUNT; j++) {
```

```
                        if (pheromone_links[i][j] != INVALID) {
                                pheromone_links[i][j] = (1 - EVAPORATION_RATE)
                                        * pheromone_links[i][j];
                        }
                }
        }
}

void update_pheromone() {
        for (int i = 0; i < POPULATION_SIZE; i++) {
                double pheromone_to_sum = POSITIVE_CONTS / ants[i].fitness;
                for (int j = 0; j < (CITY_AMOUNT - 1); j++) {
                        int city = ants[i].route[j];
                        int next_city = ants[i].route[j + 1];
                        if (pheromone_links[city][next_city] != INVALID) {
                                pheromone_links[city][next_city] += pheromone_to_sum;
                        }
                }
        }
}

// Auxiliary methods
int get_random_number(int from, int to) {
        return (from < to) ? (rand() % to) + from : 0;
}

double calculate_time(clock_t start, clock_t end) {
        return 1000.0 * ((double) (end - start) / (double) CLOCKS_PER_SEC);
}
```

```
void print_route(int ant_index, int route[CITY_AMOUNT], double distance) {
	string temp = "Ant route " + number_to_String(ant_index) + " : ";
	for (unsigned int i = 0; i < CITY_AMOUNT; i++) {
		temp += number_to_String(route[i]);
		if ((i + 1) != CITY_AMOUNT) {
			temp += ", ";
		}
	}
	temp += ". Distance: " + number_to_String(distance) + "\n";
	cout << temp << endl;
}

void print_pheromone() {
	string temp = "Pheromone rates:";
	temp += "\n{";
	for (int i = 0; i < CITY_AMOUNT; i++) {
		temp += "{";
		for (int j = 0; j < CITY_AMOUNT; j++) {
			temp += number_to_String(pheromone_links[i][j]);
			if ((j + 1) != CITY_AMOUNT) {
				temp += ", ";
			}
		}
		if ((i + 1) != CITY_AMOUNT) {
			temp += "}, ";
		} else {
			temp += "}";
		}
```

```
        }
        temp += "}\n";
        cout << temp << endl;
}

void print_result() {
        cout << "Worst result:" << endl;
        cout << "f(x):" << worse_fitness << endl;
        print_route(0, worse_route, worse_distance);

        cout << "Best result:" << endl;
        cout << "f(x):" << best_fitness << endl;
        print_route(0, best_route, best_distance);

        calculate_metrics();
        cout << "Average:" << average << endl;
        cout << "Variance:" << variance << endl;
        cout << "Standard deviation:" << standard_deviation << endl;
}

string number_to_String(double n) {
        stringstream out;
        out << n;
        return out.str();
}

void calculate_metrics() {
        // Calculate the average
        double sum = 0;
```

```
        for (int i = 0; i < POPULATION_SIZE; i++) {
                sum += ants[i].distance;
        }
        average = (double) sum / (double) POPULATION_SIZE;
        // Calculate variance
        sum = 0;
        for (int i = 0; i < POPULATION_SIZE; i++) {
                sum += pow(ants[i].distance - average, 2);
        }
        variance = (double) sum / (double) POPULATION_SIZE;
        // Calculating the standard deviation
        standard_deviation = pow(variance, 0.5);
}
```

❖ **Kết quả chạy chương trình**

Total table static of test			ACO			ACO			ACO			ACO		
			City	Ant	Note	City	Ant	Note	City	Ant	Note	City	Ant	Note
			6	30	Random	6	30	FIX	14	30	Random	14	30	FIX
			Best distance	Worst distance	Total time (ms)	Best distance	Worst distance	Total time (ms)	Best distance	Worst distance	Total time (ms)	Best distance	Worst distance	Total time (ms)
Thien's Cumputer	CPU	Intel Core I5 8th Gen	3	40	2100	3	39	2077	100	276	2479	101	265	2311
	RAM	2 GB												
	HDD	SSD 160Gb												
	Language	C++												
	OS	Window 10												
Hoang's Cumputer	CPU	AMD Ryzen 5 5500U	3	36	566	3	38	549	101	274	547	101	263	570
	RAM	8 GB												
	HDD	SSD 256Gb												
	Language	C++												
	OS	Window 10												
Linh's Cumputer	CPU	Intel(R) Core(TM) i5-6200U	7	39	1146	3	38	1096	123	283	1198	115	267	1196
	RAM	4GB												
	HDD	SSD 128GB												
	Language	C++												
	OS	Window 10												
Phieu's Cumputer	CPU	Intel core i3 gen 3	3	42	404	3	39	323	118	274	536	98	289	10771
	RAM	4096MB												
	HDD	500GB												
	Language	C++												
	OS	Window 7												

MaiChi's Cumputer	CPU	11th Gen Intel(R) Core(TM) i5-1135G7	3	38	284	3	39	271	122	286	431	103	265	380
	RAM	8Gb												
	HDD	256Gb												
	Language	C++												
	OS	Windows 11												
Chi's Cumputer	CPU	AMD Ryzen 5 4600H	3	40	447	3	39	738	109	284	700	115	294	494
	RAM	8 Gb												
	HDD	SSD 1TB												
	Language	C++												
	OS	Window 11												

HaiAnh's Cumputer	CPU	Intel Core i7-12700H	7	39	564	3	39	607	92	297	7806	102	306	7997
	RAM	16GB												
	HDD	SSD 512GB												
	Language	C++												
	OS	Window 11												