TRƯỜNG ĐẠI HỌC BÁCH KHOA HÀ NỘI

Viện Công nghệ thông tin và Truyền thông

Tài liệu đặc tả yêu cầu phần mềm

(Software Requirement Specification – SRS)

Ứng dụng để chứng minh trực quan các thuật toán tìm kiếm siêu mô phỏng

Môn: Software Engineering

Nhóm 6

Nguyễn Thị Thu Thảo

Nguyễn Trọng Khang

Nguyễn Tài Huy

*Hà Nội, ngày 9 tháng 05 năm* *2021*

Mục lục

Mục lục 1

1 Giới thiệu 3

1.1 Mục đích 3

1.2 Phạm vi 3

1.3 Từ điển thuật ngữ 3

1.4 Tài liệu tham khảo 4

2 Mô tả tổng quan 5

2.1 Các tác nhân 5

2.2 Biểu đồ use case tổng quan 5

2.3 Quy trình nghiệp vụ 7

3 Đặc tả các chức năng 12

3.1 Đặc tả use case UC001 “Tạo đồ thị” 12

4 Các yêu cầu khác 19

4.1 Chức năng (Functionality) 19

4.2 Tính dễ dùng (Usability) 19

4.3 Các yêu cầu khác 19

# Giới thiệu

## Mục đích

Tài liệu này đưa ra để xây dựng một hệ thống trực quan để minh họa thuật toán tìm kiếm siêu mô phỏng trong việc giải bài toán người bán hàng. Tài liệu bao gồm mục đích, tính năng, các giao diện, ràng buộc của hệ thống cần thực hiện để phản ứng với các kích thích bên ngoài.

## Phạm vi

Phạm vi của tài liệu này là áp dụng thuật toán tìm kiếm siêu mô phỏng trong việc giải bài toán người bán hàng. Bài toán người bán hàng (Travelling salesman problem-TSP) đặt ra để trả lời cho câu hỏi:”Cho 1 danh sách các thành phố và khoảng cách giữa chúng, vậy đâu là đường đi ngắn nhất qua tất cả các thành phố với điều kiện chỉ đi qua mỗi thành phố 1 lần và quay lại thành phố đầu tiên khi kết thúc?”. TSP có thể được thể hiện dưới dạng 1 sơ đồ có trọng số vô hướng, với mỗi thành phố là các đỉnh, khoảng cách giữa chúng là khối lượng của các cạnh. Khi này, bài toán trở thành tìm khối lượng nhỏ nhất khi bắt đầu và kết thúc ở 1 đỉnh và đi qua mỗi đỉnh đúng 1 lần.

Cụ thể ở ứng dụng này, người dùng có thể tự tạo 1 sơ đồ với số lượng đỉnh và khối lượng cạnh tự chọn. Từ đó, họ có thể tự chọn 1 trong 3 phương án (simulated annealing, tabu search, artificial bee colony) để thực hiện giải bài toán.

## Từ điển thuật ngữ

* Bài toán người bán hàng (Travelling salesman problem-TSP): là 1 bài toán NP-khó thuộc thể loại tối ưu rời rạc hay tổ hợp được nghiên cứu trong vận trù học hoặc lý thuyết khoa học máy tính. Bài toán được phát biểu như sau: Cho trước 1 danh sách các thành phố và khoảng cách giữa chúng, tìm chu trình ngắn nhất thăm mỗi thành phố đúng 1 lần. Bài toán này được nêu ra bởi nhà toán học người Ai-len William Rowan Hamilton và nhà toán học người Anh Thomas Kirkman. Mục tiêu chính của bài toán là để tìm con đường ngắn nhất qua 1 tập N đỉnh mà trong đó mỗi đỉnh chỉ được đi qua 1 lần. Ở đây, bài toán có thể được phát biểu là 1 mạng lưới với Ni là số đỉnh, n1 là điểm bắt đầu và thời lượng di chuyển hay chi phí với các cặp đỉnh có thứ tự. TSP có rất nhiều ứng dụng như định tuyến xe, giải trình tự công việc, phân tích dữ liệu tổ hợp, v.v.
* Metaheuristic: trong khoa học máy tính và tối ưu hóa toán học, metaheiristic là 1 quy trình cấp cao hoặc heuristic được thiết kế để tìm, tạo hoặc chọn 1 heiristic (thuật toán tìm kiếm từng phần) có thể cung cấp 1 giải pháp đủ tốt cho một vấn đề tối ưu hóa, đặc biệt là với thông tin hoặc năng lực tính toán hạn chế. Metaheuristics lấy mẫu một tập hợp con các dung dịch lớn để có thể liệt kê hoàn toàn hoặc khám phá theo cách khác. Metaheuristics có thể đưa ra tương đối ít giả định về vấn đề tối ưu hóa đang được giải quyết và do đó có thể sử dụng được cho nhiều vấn đề khác nhau.

## Tài liệu tham khảo

* <https://en.wikipedia.org/wiki/Metaheuristic>
* <https://en.wikipedia.org/wiki/Travelling_salesman_problem>
* <http://www.ijarse.com/images/fullpdf/1519813763_NMCOE4098IJARSE.pdf>
* <https://www.researchgate.net/publication/331585233_Tabu_Search_Method_for_Solving_the_Traveling_salesman_Problem?fbclid=IwAR3iL_ELK4oupFUiml29CrlepiovGtUcXqK3VlQ9emJX-2aCNYNEYVztACg>
* <https://www.youtube.com/watch?v=KP0O06C9jc8>
* <https://www.hindawi.com/journals/cin/2016/1712630/>

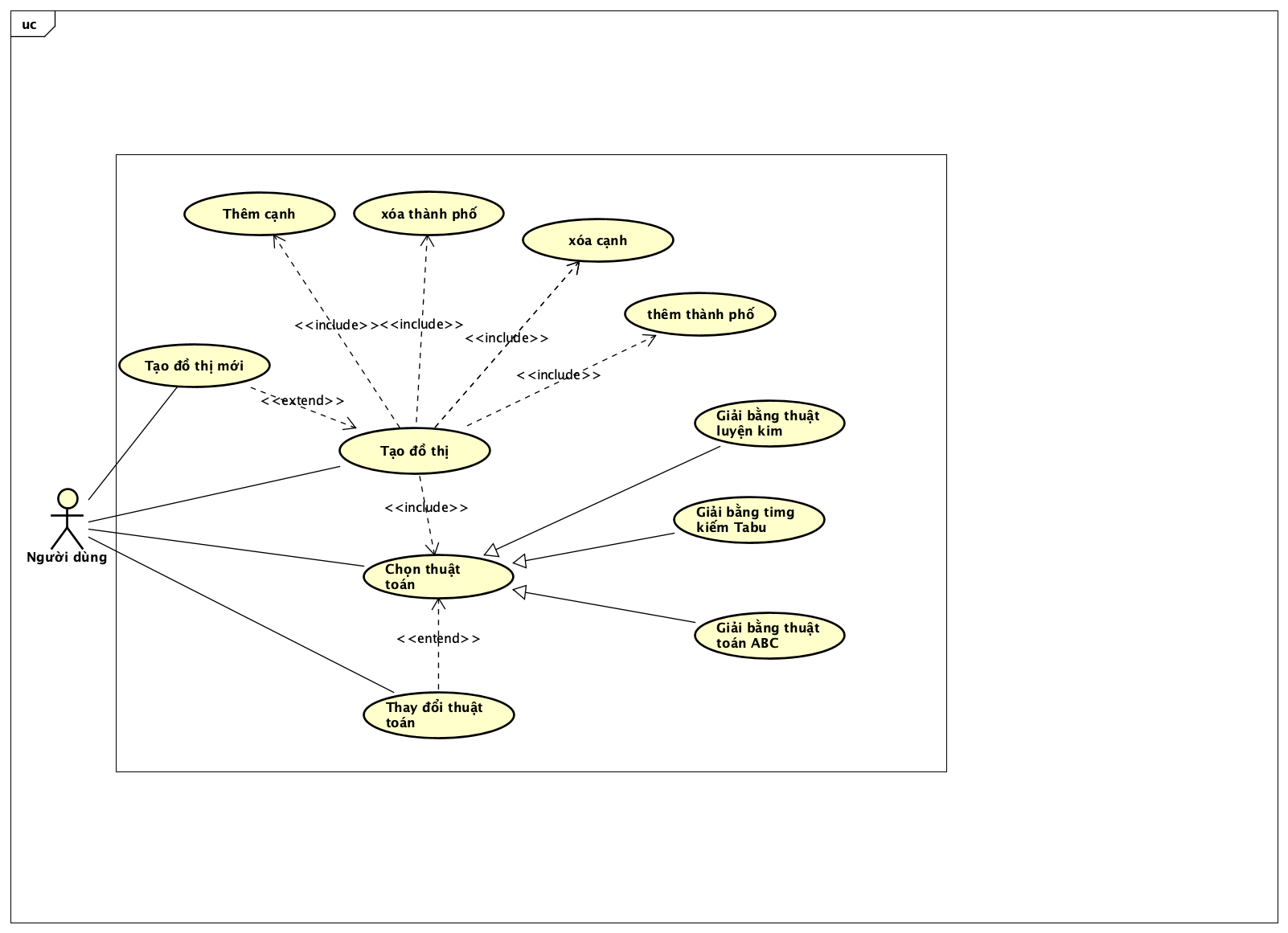
# Mô tả tổng quan

## Các tác nhân

Phần mềm có 1 tác nhân là Người dùng.

## Biểu đồ use case tổng quan

Khi mới bắt đầu chương trình người dùng sẽ phải tạo đồ thị bằng các phương thức như thêm cạnh, xóa cạnh, thêm thành phố, xóa thành phố. Sau kkhi tạo xong đồ thị người dùng có thể chọn thuật toán để giải quyết vẫn đề TSP . Cuối cùng , người dùng có thể chạy chương trình bằng cách tự dộng hoặc bằng cơ( từng bước một ) .

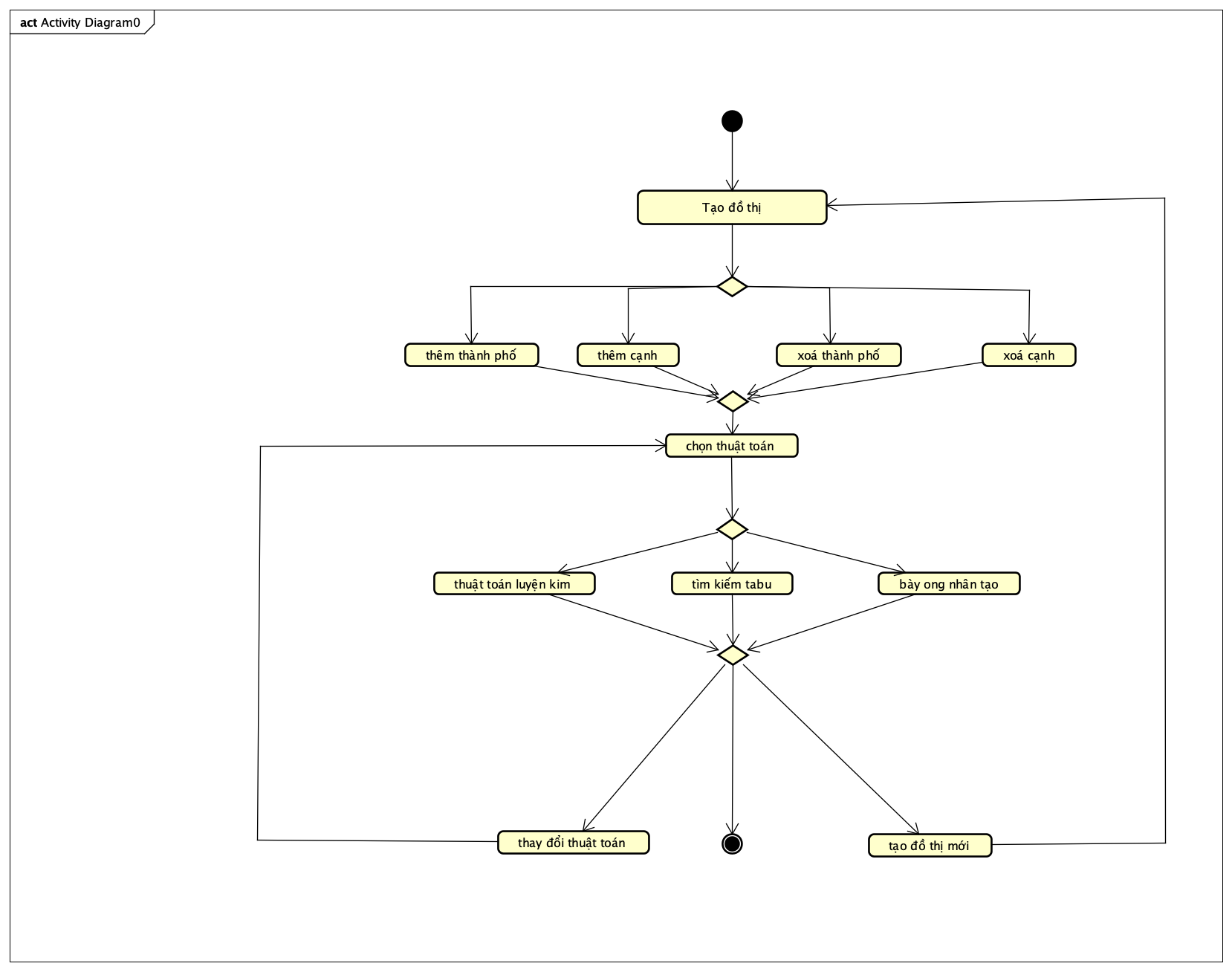


## Quy trình nghiệp vụ

Khi chương trình bắt đầu người dùng cần phải tạo đồ thị . Người dùng cần thêm thành phố, cạnh và các thong số của dồ thị.

Sau khi đã tạo xong đồ thị, người dùng chọn thuật toán sử dụng cho chương trình:Thuật toán luyện kim, tìm kiếm Tabu, thuật toán bầy ong nhân tạo.

Sau khi đã chắc chắn về thuật toán bạn có thể chạy thuật toán bằng 2 cách : Tự dộng hoặc cơ. Nếu bạn cho chương trình chạy tự động bạn cần nhập tốc độ chạy của chương trình . Nếu có vấn đề gì bạ có thể kết thúc chạy thuật toán sớm . Sau khi chạy thuật toán , bạn có thể chọn lại thuật toán , tạo đồ thị mới nếu cần hoặc là kết thúc chương trình.



# Đặc tả các chức năng

Chi tiết về các use case được đưa ra trong phần 2 được đặc tả trong các phần dưới đây.

## Đặc tả use case UC001 “Tạo đồ thị”

|  |  |  |  |
| --- | --- | --- | --- |
| Mã Use case | UC001 | Tên Use case | Tạo đồ thị |
| Tác nhân | Người dùng | | |
| Tiền điều kiện | Không | | |
| Luồng sự kiện chính  (Thành công) | |  |  |  | | --- | --- | --- | | STT | Thực hiện bởi | Hành động | |  | Khách | Chọn 1 trong 4 chức nặng (thêm thành phố, xoá thành phố, thêm cạnh, xoá cạnh) | |  | Khách | Thêm, Xoá thánh phố bằng cách ấn chuột trái lên màn hình hiển thị  Thêm, Xoá cạnh bằng cách kéo từ thành phố này đến thành phố khác | |  | Hệ thống | Hiện thị cạnh, thành phố được thêm lên màn hình hoặc xoá cạnh, thành phố bị xoá ở màn hình | |  | Khách | Chọn tự tạo cạnh cho các thành phố | |  | Hệ thống | Tạo các cạnh tự động để bất kì 2 thành phố đều có đường đi | |  | Hệ thống | Hiển thị bảng trong đó có các thông số đường đi từ thành phồ i đến thành phố j | | | |
| Luồng sự kiện thay thế | |  |  |  | | --- | --- | --- | | STT | Thực hiện bởi | Hành động | | 3.a | Hệ thống | Hế thống chỉ xoá cạnh, thành phố khi thành phố, cạnh đó tồn tại | | | |
| Hậu điều kiện | Không | | |

## Đặc tả use case UC002 “Chọn thuật toán”

|  |  |  |  |
| --- | --- | --- | --- |
| Mã Use case | UC002 | Tên Use case | Chọn thuật toán |
| Tác nhân | Người dùng | | |
| Tiền điều kiện | Không | | |
| Luồng sự kiện chính  (Thành công) | |  |  |  | | --- | --- | --- | | STT | Thực hiện bởi | Hành động | |  | Người dùng | Chọn 1 trong 3 thuật toán | |  | Hệ thống | Đọc thuật toán được chọn | | | |
| Luồng sự kiện thay thế | |  |  |  | | --- | --- | --- | | STT | Thực hiện bởi | Hành động | | | |
| Hậu điều kiện | Không | | |

## Đặc tả use case UC003 “Chạy chương trình”

|  |  |  |  |
| --- | --- | --- | --- |
| Mã Use case | UC002 | Tên Use case | Tạo menu |
| Tác nhân | Người dùng | | |
| Tiền điều kiện | Người dùng nhập xong thông tin đồ thị, chọn thuật toán cần chạy | | |
| Luồng sự kiện chính  (Thành công) | |  |  |  | | --- | --- | --- | | STT | Thực hiện bởi | Hành động | |  | Người dùng | Chọn 1 trong 2 cách chạy chương trình | |  | Hệ thống | Vísualize thuật toán | | | |
| Luồng sự kiện thay thế | |  |  |  | | --- | --- | --- | | STT | Thực hiện bởi | Hành động | | | |
| Hậu điều kiện | Không | | |

# Các yêu cầu khác

<Đưa ra các yêu cầu khác nếu có, bao gồm các yêu cầu phi chức năng như hiệu năng, độ tin cậy, tính dễ dùng, tính dễ bảo trì; hoặc các yêu cầu về mặt kỹ thuật như về CSDL, công nghệ sử dụng…>

## Chức năng (Functionality)

* Trong các chuỗi sự kiện của các use case, tất cả các bước có thao tác với CSDL, nếu có lỗi trong quá trình kết nối hoặc thao tác, cần có thông báo lỗi tương ứng để tác nhân biết là lỗi liên quan đến CSDL chứ không liên quan tới lỗi của người dùng
* Các use case do Quản trị viên và Người dùng sử dụng thì Khách cần đăng nhập với vai trò tương ứng
* Định dạng hiển thị chung như sau:
  + Số căn phải
  + Chữ căn trái
  + Font: Arial 14, màu đen
  + Nền trắng

## Tính dễ dùng (Usability)

Các chức năng cần được thiết kế sao cho dễ thao tác. Cần có hướng dẫn cụ thể lỗi sai của người dùng để người dùng biết định vị lỗi, biết lỗi gì và biết cách sửa lỗi.

## Các yêu cầu khác

* Hiệu năng
* Tính đáng tin cậy
* Tính dễ bảo trì
* Tính khả chuyển.

**\*TÌM HIỂU CÁC THUẬT TOÁN:**

1. **SIMULATED ANNEALING** (thuật toán mô phỏng luyện kim)

Simulated annealing (SA) là 1 thuật toán dựa trên các heuristic về xác suất, dúng chó 1 lớp các vấn đề để tìm 1 số điều kiện tốt nhất trong 1 không gian tìm kiếm lớn. Nó thường được dùng khi không gian tìm kiếm là hữu hạn. Đối với 1 số vấn đề cố định, SA có thể cho hiệu quả tốt hơn cả thuật toán vét cạn, khi mà đơn thuần chỉ cần 1 kết quả chấp nhận được. Thuật toán được nêu ra độc lập bởi Scott Kirkpartrick, C. Daniel Gelatt, Mario P. Vecchi vào năm 1983 và bởi Vlado Cerny năm 1985.

Ý tưởng thuật toán:

Tên gọi và ý tưởng của thuật toán bắt nguồn từ kỹ thuật luyện kim trong ngành luyện kim, 1 kỹ thuật sử dụng khéo léo việc nung nóng rồi làm nguội vật liệu kim loại làm tăng kích thước và giảm những chỗ khuyết lõm của chúng. Nhiệt độ làm làm cho các nguyên tử trong vật liệu trở nên mất liên kết và rời khỏi vị trí ban đầu của chúng. Khi nhiệt độ cao, các nguyên tử chuyển động hỗn độn với tốc độ và tần suất lớn, và khi nhiệt độ giảm dần, các nguyên tử cũng vì thế mà chuyển động ít dần đi. Tương tự như quá trình vật lý này, từng bước của thuật toán SA cố gắng thay thế giải pháp hiện tại bằng 1 giải pháp ngẫu nhiên, các giải pháp mới sau đó có khả năng được chấp nhận dựa trên 1 xác suất, quyết định bởi sự khác biệt của giá trị 1 hàm tương ứng và 1 tham số toàn cục T (temperature) giảm dần suốt quá trình thức hiện thuật toán. Sự phụ thuộc này cũng như việc lựa chọn giữa giải pháp trước đó và giải pháp hiện tại hầu như là ngẫu nhiên với T lớn, tuy nhiên sẽ cho kết quả chọn lựa tốt hơn nếu T giảm dần về 0.

Tổng quát:

Trong thuật toán SA, mỗi điểm s của không gian tìm kiếm là tương tự với 1 trạng thái của 1 số hệ thống vật lý. Hàm E(s) tượng trưng cho nội năng của hệ thống trong trạng thái đó.

Tại mỗi bước, quy tắc heuristic sẽ xem xét và đưa ra 1 số trạng thái kề s’ của s, và xác suất quyết định giữa thay đổi hệ thống sang s’ hay vẫn giữ nguyên s. Những xác suất này đưa hệ thống tới trạng thái có năng lượng thấp hơn. Bước này sẽ được lặp đi lặp lại cho đến khi hệ thống đạt được 1 trạng thái đủ tốt hoặc cho đến khi ko thể tiếp tục.

Trạng thái kề: là trạng thái mới của vấn đề được sinh ra sau khi biến đổi trạng thái đã biết theo 1 số cách cụ thể.

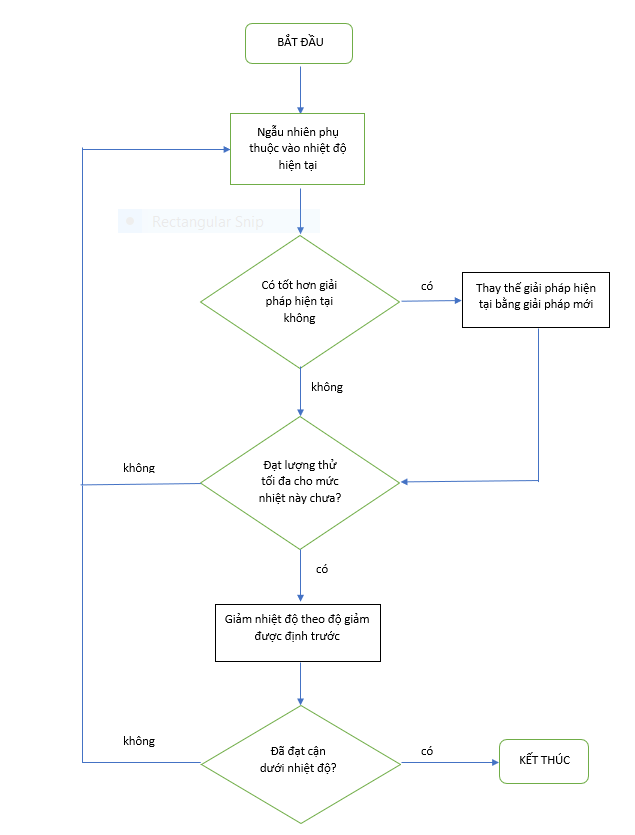
Xác suất chấp nhận:

Xác suất của việc chuyển từ s sang s’ được định nghĩa bởi 1 hàm xác suất chấp nhận: P(e,e’,T) với hàm phụ thuộc vào các nguồn năng lượng e=E(s) và e’=E(s’) của 2 trạng thái và 1 tham số toàn cục T là nhiệt độ. Những trạng thái có năng lượng thấp hơn sẽ tốt hơn những trạng thái có năng lượng cao hơn. Hàm xác suât P phải cho kết quả dương khi e’>e.

Khi xây dựng hàm P, T dần tiến tới 0, P dần tới 0 nếu e’>e và luôn dương. Khi T càng nhỏ, hệ thống sẽ ưu tiên đi về nơi có năng lượng thấp, tránh di chuyển ngược lại. Khi T=0 thuật toán trở về thuật toán tham lam (greedy algorithm). Ngoài ra hàm P thường được xây dựng sao cho tỉ lệ thuận với độ chênh lệch năng lượng giữa trạng thái tiếp theo và trạng thái hiện tại, nói cách khác khi e’ – e càng giảm thì P càng giảm, vì vậy xác suất chuyển từ trạng thái hiện tại tới trạng thái có độ chênh lệch năng lượng thấp thì khó thấp hơn so với độ trạng thái có độ chênh lệch năng lượng cao.

Quá trình thực hiện:

Khởi tạo 1 giá trị T lớn và giảm dần sau mỗi vòng lặp. Kết thúc khi T=0.

****

Áp dụng vào bài toán TSP:

1 số tham số:

* Trạng thái: 1 trạng thái được định nghĩa là 1 hoán vị các thành phố thể hiện cho 1 cách để đi qua tất cả các thành phố.
* Không gian trạng thái: tập hợp tất cả các trạng thái có thể có của các thành phố (tập hợp các cách đi có thể).
* Hàm năng lượng E(): tổng chi phí của 1 cách đi
* Trạng thái kề
* Hàm P: P=EXP[(Current\_cost-New\_cost)/Temperature] (EXP là hàm lũy thừa của E)
* Nhiệt độ T

Thuật toán áp dụng vào bài toán:

Initialization(Current\_solution, Temperature)

Calculation of the Current\_cost //Tính chi phí trạng thái hiện tại

LOOP

New\_state //khởi tạo trạng thái kế

Calculation of the new\_Cost //Tính chi phí trạng thái mới

//Nếu trạng thái mới có chi phí thấp hơn

IF Δ(Current\_cost – new\_Cost) >=0 THEN

Current\_state=New\_state

//Ngược lại

ELSE

//Nếu rơi vào xác suất chấp nhận

IF EXP[(Current\_cost-New\_cost)/Temperature) > Random (0,1)

THEN

//Chấp nhận trạng thái mới

Current\_state=New\_state

ELSE

//Bỏ qua

//Giảm nhiệt độ

EXIT When STOP\_CRITERION

END LOOP

1. **TABU SEARCH**

Step1:Init

-Đưa vào số lượng thành phố (n), và khoảng cách giữa các thành thố.

-Đưa vào điều kiện dừng(Dừng sau bao nhiêu bước lặp)

-Xét số t = sqrt(n).(số lượng vòng lặp để giữ tabu list)

-khởi tạo tabu list là rỗng [0 0 0 0 0].

Step2: Sử dụng tham lam để timg lời giải tốt nhất kí hiệu là x\* với hàm z\*, cùng thời điểm đó xét lời giải hiện tại là x như nhau

X :curent solution

X\*=x :best solution

Z\*=z(x) :z\* là tổng khoảng cách của lời giải x

Step3: Vòng lặp

While(stop crierion is not satisfied)

Do{

Begin:

* Xác định các hàng xóm qua tạo nhiều lời giải bằng việc di chuyển 2 thành phố. Việc di chuyển TSP : đổi chỗ 2 thành phố ngẫu nhiên từ lời giải x

}

Step3.1:Chọn ra 1 bước chuyển tốt nhất từ N(x) qua việc chọn lời giải nhỏ nhất z(x) từ thay đổi x thành x’ và hàm thành z(x’) sau đố thêm vào tabu list

Kiểm tra x’ có phải Tabu không?

-Nếu không ->3.2

-Ngược lại 3.3

Step3.2:Biểu diễn thay đổi

X=x’,z(x)=z(x’)

Nếu z(x)<z\* -> z\*=z(x),x\*=x

->Step4

Step3.3 Nếu x’ là Aspiration

Nếu đúng -> 3.2

Sai tìm tập hàng xóm

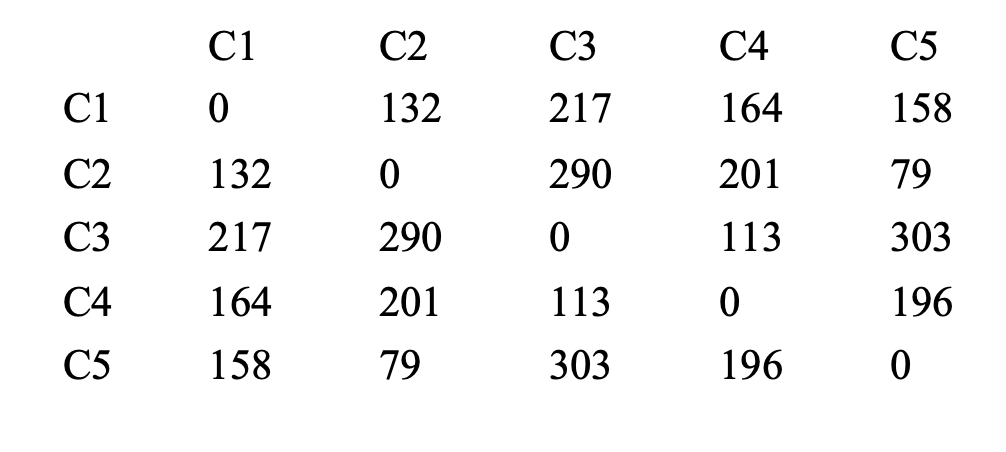
Step4:Ghi lại Tabu từ di chuyển hiện tại vào tabulist giờ tabulist thành [0 0 t 0]

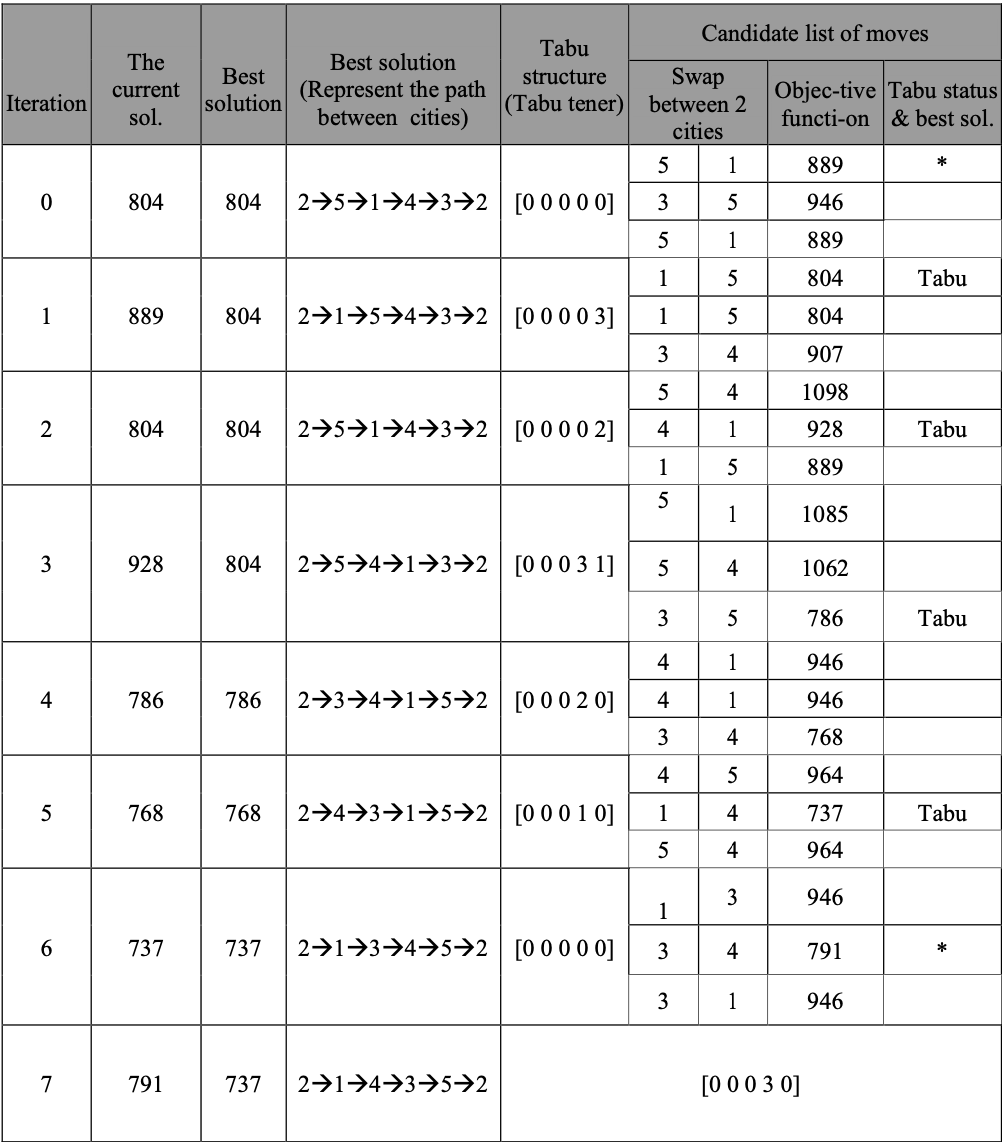
Những vòng lặp sau đó giảm t đi 1

End while

Step5:đưa ra kết quả

Example

****

****

1. **ARTIFICIAL BEE COLONY**

Có n đỉnh(VD: n = 5 như ví dụ của Chip),Chọn điểm xuất phát từ 1

Bước 1: Khởi Tạo quần thể:

* Số lượng ong trong bầy(Tự chọn ) : N(Giả sử N = 6)
* Số chiều của giải pháp D = n-1(Giả sử D = 4 )
* Số lượng bộ đếm tối đa để cải thiện giải pháp L = (N\*D)/2
* Số vọng lặp cảu giả thuật (Tự chọn) I = 50
* Khởi tạo giải pháp ban đầu với số ong thợ là N/2 (= 3 con ong thợ)

+ Một ong thợ sẽ là 1 nghiệm (Chẳng hạn X(0) là đi từ 1->2->3->4->5) thì X(0)=(d(1,2);d(2,3);d(3,4);d(4,5))

+ X(1),X(2) cx tương tự

* Khởi tạo Bộ đếm cảu mỗi X là C= [0,0,0,0]

Bước 2: Xác định giải pháp khởi tạo tốt nhất

* VD: F(0) = Tổng 4 cái d trong X(0)
* Fit(X(0)) =1/(1+F(0));
* Sau đó so sánh 3 cái Fit , X nào có fit lớn nhất thì gắn Xbest = X đó.

Bước 3 : Ong thợ sinh giải pháp mới :

VD : cập nhạt X(0)

* Chọn ngẫu nhiên chỉ số chiều cần cập nhật : k = random(0,1,2,3)(VD k = 3)
* Chọn ngẫu nhiên hàng xóm để cập nhật : j = random(0,1,2)(VD j=1)
* Cập nhật V(0,3) = X(1,3)
* Tính F(V) và fit(V(0))
* Kiểm tra : Nếu fit(V(0))>fit(X(0)) thì Cạp nhật X(0) = V(0) , reset C(0)=0

+ else( Giữ nguyên X(0) và C(0)++ )

Bước 4 : Tính toán xác suất lựa chọn giải pháp dựa trên tất cả các giải pháp của ong thợ:

* P(i)=[fit(i)]/Tổng các fit

Bước 5:Các ong quan sát sinh giải pháp mới :

* Chọn P(i) ngẫu nhiên (i=0,1,2)Giả sử i = 0
* Chọn ngẫu nhiên chỉ số chiều cần cập nhật : k = random(0,1,2,3)(VD k = 2)
* Chọn ngẫu nhiên hàng xóm để cập nhật : j = random(0,1,2)(VD j=0)
* V(0,2) = X(0,2)
* Tính F(V) và fit (V)
* Kiểm tra : Nếu fit(V(0))>fit(X(0)) thì Cạp nhật X(0) = V(0) , reset C(0)=0

+ else( Giữ nguyên X(0) và C(0)++ )

Bước 5:

\*\* Quá trình của ong trinh sát : So sánh C(i) với L

\*\* Nếu C lớn hơn L thì ghi nhận X đó là Xbest của lần này

Sau đó bước lặp cộng thêm 1 và tiếp tục với vong lặp 2

Chương trình dừng lại khi Bước lặp = I.