HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Nguyễn Dương Kiên

NGHIÊN CỨU VỀ THUẬT TOÁN TIẾN HÓA ĐA NHÂN TỐ GIẢI QUYẾT BÀI TOÁN TỐI ƯU

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

Hà Nội - 2020

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

Nguyễn Dương Kiên

NGHIÊN CỨU VỀ THUẬT TOÁN TIẾN HÓA ĐA NHÂN TỐ GIẢI QUYẾT BÀI TOÁN TỐI ƯU

Chuyên ngành: Hệ thống thông tin

Mã số: 8.48.01.04

LUẬN VĂN THẠC SĨ KỸ THUẬT

(Theo định hướng ứng dụng)

NGƯỜI HƯỚNG DẪN KHOA HỌC: TS. Trần Quý Nam

Hà Nội - 2020

LỜI CẢM ƠN

Luận văn được hoàn thành dưới sự hướng dẫn và chỉ bảo nhiệt tình của TS. Trần Quý Nam giảng viên học viện bưu chính viễn thông. Trong quá trình làm việc, em không chỉ tích lũy được nhiều kiến thức và kinh nghiệm quý báu mà còn được học hỏi ở Thầy một tinh thần làm việc khoa học, đầy tinh thần trách nhiệm. Qua đây, em xin bày tỏ lòng biết ơn chân thành và sâu sắc tới Thầy.

Nhân dịp này, em cũng xin bày tỏ lòng biết ơn tới toàn thể các thầy cô giáo trong học viện Công nghệ Bưu chính Viễn thông những người đã dạy bảo em tận tình trong suốt quá trình học tập và tạo điều kiện về cơ sở vật chất để em có thể hoàn thành tốt luận văn này.

Cuối cùng, em xin được gửi lời cảm ơn chân thành tới gia đình, bạn bè đã cổ vũ, động viên và tạo điều kiện cho em trong quá trình học tập và thực hiện luận văn.

Dù đã cố gắng hết sức cùng với sự tận tâm của thầy giáo hướng dẫn tuy nhiên do trình độ còn hạn chế nên khó tránh khỏi những thiếu sót, em rất mong nhận được sự đóng góp ý kiến của thầy cô và các bạn.

Hà Nội, ngày tháng năm 2020

Học viên cao học

Nguyễn Dương Kiên

MỤC LỤC

DANH SÁCH BẢNG

DANH SÁCH HÌNH VẼ

LỜI NÓI ĐẦU

Trong những năm vừa qua, các thuật toán tiến hóa được áp dụng để giải quyết nhiều bài toán tối ưu trong khoa học máy tính và trong thực tế. Tuy nhiên, việc thiết kế các thuật toán tiến hóa mới chỉ tập trung vào việc giải quyết có hiệu quả một bài toán tối ưu tại một thời điểm, chưa có thuật toán tiến hóa giải quyết đồng thời các bài toán tối ưu hóa chỉ sử dụng duy nhất một quần thể. Do vậy, luận văn sẽ tìm hiểu một mô hình tiến hóa mới trong tính toán tiến hóa: mô hình tiến hóa đa nhân tố (Multifactorial Optimization) cho phép giải đồng thời nhiều bài toán tối ưu mà chỉ dựa trên một quần thể tiến hóa duy nhất.

Thuật toán tiến hóa (Evolutionary Algorithms - EAs) dựa theo học thuyết Darwin nói chung được hình thành trên quan niệm cho rằng, quá trình tiến hóa là quá trình hoàn hảo nhất vì tự nó đã mang tính tối ưu [1]. Tính tối ưu được thể hiện ở chỗ, cá thể sau được sinh ra bao giờ cũng tốt hơn, hoàn hảo hơn cá thể cha-mẹ, chúng có khả năng thích nghi với sự thay đổi của môi trường cao hơn cá thể cha-mẹ. Thuật toán tiến hóa được áp dụng trong các bài toán tối ưu. Bài toán tối ưu là bài toán tìm giá trị cực đại hoặc cực tiểu của một hàm hoặc một quá trình nào đó. Cơ chế này được sử dụng trong nhiều lĩnh vực như vật lý, hóa học, kinh tế,... Trong thuật toán tiến hóa, một nhóm các cá thể (giải pháp của bài toán) sẽ được khởi tạo ngẫu nhiên. Trong mỗi thế hệ, những cá thể tốt, thích nghi với môi trường (bài toán) sẽ được giữ lại. Quá trình tiếp tục cho đến khi gặp điều kiện dừng của bài toán. Có nhiều thuật toán tiến hóa khác nhau như: thuật toán di truyền (Genetic Algorithm – GA), thuật toán tối ưu hóa bầy đàn (Particle Swarm Optimization – PSO), thuật toán đàn kiến (Ant Colony Optimization - ACO),....Trong đó, thuật toán di truyền được xây dựng dựa trên quy luật tiến hóa sinh học hay phát triển tự nhiên của một quần thể sống. Các cá thể trải qua một quá trình phát triển và sinh sản để tạo ra những cá thể mới cho thế hệ kế tiếp. Trong quá trình tăng trưởng và phát triển những cá thể xấu tức là những cá thể không thích nghi được với môi trường sẽ bị đào thải, ngược lại, những cá thể tốt sẽ được giữ lại (đây chính là quá trình chọn lọc) và được lai ghép (quá trình lai ghép) để tạo ra những cá thể mới cho thế hệ sau. Những cá thể

mới được sinh ra mang những tính trạng của cá thể cha-mẹ (còn gọi là hiện tượng di truyền). Thuật toán tối ưu bầy đàn được xây dựng dựa vào quá trình mô phỏng sinh học của đàn chim. Để hiểu rõ về thuật toán, hãy xem một ví dụ về quá trình tìm kiếm thức ăn của một đàn chim. Tại thời điểm tìm kiếm cả đàn bay theo một hướng nào đó, có thể là ngẫu nhiên. Tuy nhiên, sau một thời gian tìm kiếm một số cá thể trong đàn bắt đầu tìm ra được nơi có chứa thức ăn. Tùy vào số lượng thức ăn vừa tìm được mà các cá thể gửi tín hiệu đến các cá thể đang tìm kiếm ở vùng lân cận. Tín hiệu này được lan truyền trên toàn quần thể. Dựa vào thông tin nhận được, mỗi cá thể sẽ điều chỉnh hướng bay và vận tốc bay theo hướng về nơi có nhiều thức ăn nhất. Cơ chế truyền tin như vậy thường được xem là một kiểu hình của trí tuệ bầy đàn. Cơ chế này giúp đàn chim tìm ra nơi có nhiều thức ăn nhất trên không gian tìm kiếm [2]. Các thuật toán tiến hóa trên chỉ dừng lại ở việc giải quyết một bài toán tối ưu tại một thời điểm. Tuy nhiên, hầu hết các ứng dụng trong thực tế đều yêu cầu phải giải quyết nhiều bài toán tối ưu cùng lúc, ví dụ như ứng dụng trong tính toán đám mây. Do đó, luận văn sẽ tìm hiểu một mô hình tiến hóa mới: mô hình tiến hóa đa nhân tố (Multifactorial Optimization - MFO). Mô hình tiến hóa đa nhân tố là mô hình tiến hóa tổng hợp để giải quyết đồng thời nhiều bài toán tối ưu. Mỗi bài toán tối ưu được coi như một nhân tố ảnh hưởng đến quá trình tiến hóa. Ưu điểm của phương pháp này là chúng ta có thể chuyển vật liệu di truyền từ các bài toán tối ưu đơn giản đến các bài toán tối ưu phức tạp. Điều này có thể đẩy nhanh quá trình tối ưu hóa, giảm thời gian thực hiện.

Cấu trúc luận văn được tổ chức như sau

Chương 1: Luận văn sẽ trình bày tổng quan về bài toán tối ưu hóa và các phương pháp để giải quyết một bài toán tối ưu hóa.

Chương 2: Luận văn sẽ trình bày về mô hình tiến hóa đa nhân tố và giải thuật tiến hóa đa nhân tố để giải quyết bài toán tối ưu hóa.

Chương 3: Áp dụng thuật toán tiến hóa đa nhân tố để giải các bài toán tối ưu đơn mục tiêu

Chương 1 TỔNG QUAN

1.1 Bài toán tối ưu

Giải thuật di truyền (Di truyền - Genetic Algorithm (GA)) Tối ưu hóa là cơ chế tìm giá trị cực tiểu hoặc cực đại của một hàm hoặc một quá trình nào đó. Cơ chế này được sử dụng trong nhiều lĩnh vực như vật lý, hóa học, kinh tế...để đạt được mục đích là tối đa hóa hiệu quả, sản xuất hoặc các thước đo khác. Tối ưu hóa liên quan đến hai khái niệm cực tiểu và cực đại của một hàm f nào đó. Đây là hai bài toán đối lập nhau, trong đó, tìm cực tiểu của hàm f tương đương với tìm cực đại của hàm $-f$.

Về mặt toán học, bài toán tìm cực đại được định nghĩa như sau:

$$f : R^n \rightarrow R, \tag{1.1}$$

$$Tìm\ X^* \in R^n\ để\ f(X^*) > f(X),\ \forall X \in R^n.$$

Miền R^n được gọi là *không gian tìm kiếm*. Mỗi phần tử thuộc R^n được gọi là một *giải pháp* trong không gian tìm kiếm, X^* được gọi là *giải pháp tối ưu*. Hàm f được gọi là *hàm mục tiêu*, hàm này xác định trong không gian n chiều và nhận giá trị thực.

Bài toán tối ưu hóa được chia làm hai loại chính đó là Tối ưu rời rạc hay còn gọi là tối ưu tổ hợp (TƯTH) và Tối ưu liên tục. Trong chương này tác giả sẽ chỉ tập trung vào tối ưu hóa tổ hợp

- Dựa vào số lượng mục tiêu: đơn mục tiêu, đa mục tiêu
- Dựa vào ràng buộc: có ràng buộc, không có ràng buộc
- Dựa vào miền giá trị của biến: tối ưu liên tục hay còn gọi là tối ưu tổ hợp (TƯTH), tối ưu rời rạc

Trong chương này tác giả sẽ chỉ tập trung vào tối ưu hóa tổ hợp

1.1.1 Tối ưu hóa tổ hợp

Một cách tổng quát, mỗi bài toán TƯTH có thể phát biểu như sau: Cho một bộ ba (S,f,Ω), trong đó S là tập hữu hạn trạng thái (lời giải tiềm năng hay phương án), f là hàm mục tiêu xác định trên S, còn Ω là tập các ràng buộc. Mỗi phương án s

$\in S$ thỏa mãn các ràng buộc ω gọi là phương án (hay lời giải) chấp nhận được. Mục đích của ta là tìm phương án chấp nhận được $s*$ tối ưu hóa toàn cục hàm mục tiêu f. Chẳng hạn với bài toán cực tiểu thì $f(s*) \leq f(s)$ với mọi phương án chấp nhận được s. Hay có thể tóm gọn lại: bài toán được gọi là tối ưu tổ hợp khi các biến quyết định nhận giá trị trong một tập rời rạc, được giới hạn bởi một số ràng buộc. Chúng ta có một số bài toán tiểu biểu cho lớp bài toán này là [4]:

- Bài toán người du lịch (Traveling Salesman Problem)
- Cây khung nhỏ nhất (Miminum Spanning Tree)
- Bài toán phân công (Assignment Problem)
- Bài toán cái túi (Knapsack Problem)

Và để minh họa cho phần lý thuyết tổng quát, tác giả sẽ tập trung vào hai bài toán là bài toán phân công và bài toán cây khung nhỏ nhất.

Bài toán phân công

Bài toán có thể phát biểu như sau:

Giả sử có n người $a_1,a_2,...,a_n$ cần làm các công việc $J_1,J_2,...,J_n$ và ta cũng có một bảng kích thước n x n thể hiện mức độ hiệu quả trong công việc. Câu hỏi đặt ra là ta phải phân công các công việc cho n người này như thế nào để đảm bảo mỗi người được nhận làm một công việc, mỗi công việc được giải quyết bởi một người và tổng hiệu quả của các công việc là lớn nhất? (Để đơn giản hóa vấn đề, các chỉ số đánh giá năng suất được xét là các chỉ số nguyên)

Bằng phương pháp lập luận tương tự như trong ví dụ dẫn nhập bằng các phép tính, ta sẽ thu được một bảng A kích thước n x n thể hiện mức độ không phù hợp sao cho tất cả các phần tử trên bảng là các số không âm và có ít nhất một số 0 trên mỗi hàng và mỗi cột.

Nếu như ta có thể tìm được n số 0 – độc lập (hai số 0 bất kì không nằm trên cùng một hàng hay cùng một cột) thì bài toán của ta được giải quyết. Trong trường hợp ngược lại ta cần đến một kết quả cơ cấu sau phụ trợ cho quá trình phân tính bài toán:

Định lý Konig-Egervary [6] [7].

Bài toán cây khung ngắn nhất

Cho $G = (X,E)$ là một đồ thị liên thông và $T = (X,F)$ là một đồ thị bộ phận của G. Nếu T là cây thì T được gọi là một cây khung của G Cây khung còn có thể được gọi bằng các tên khác như cây bao trùm, cây phủ hoặc là cây tối đại. Sử dụng thuật toán Prim ta có thể giải bài toán như sau:

Đầu vào:

Đồ thị liên thông $G = (X,E)$,X gồm N đỉnh

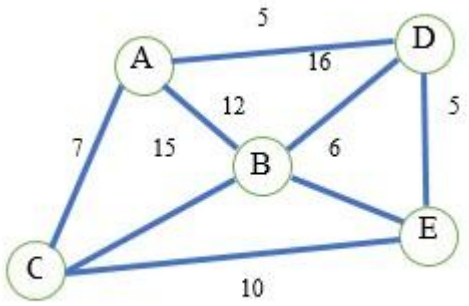
Đầu ra:

Cây khung $T = (V,U)$ của G

Ràng buộc:

1. Chọn tùy ý 1 đỉnh $v \in X$ và khởi tạo $V := \{v\}$; $U := \varphi$;
2. Chọn cạnh e có trọng lượng nhỏ nhất trong các cạnh (w,v) mà $w \in X/V$ và $v \in V$.
3. $V := V \cup \{w\}$; $U := U \cup \{e\}$
4. Nếu U đủ $N - 1$ cạnh thì dừng, ngược lại lặp từ thao tác số 2.

Ví dụ: Tìm cây khung của đồ thị sau



Hình 1-1: Ví dụ cây khung của đồ thị

Bài toán người du lịch

Bài toán người du lịch được phát biểu như sau: "Có *n* thành phố (đánh số từ 1 đến *n*). Một người du lịch, xuất phát từ thành thành phố *s*, muốn đi thăm tất cả các thành phố khác, mỗi thành phố đúng một lần, rồi lại quay về nơi xuất phát. Giả thiết biết chi phí đi từ thành phố i đến thành phố *j* là *c*(*i*,*j*), $1 \le i,j \le n$. Hãy tìm một hành trình cho người du lịch sao cho chi phí của hành trình này là nhỏ nhất".

Mỗi hành trình của người du lịch được biểu diễn bằng một hoán vị $X = (x_1, x_2, \dots, x_n)$ của $\{1, 2, \dots, n\}$ với $x_1 = s$ (hoán vị này biểu diễn hành trình $x_1 \to x_2 \to \cdots \to x_{n-1} \to x_n \to x_1$). Chi phí của hành trình X được tính bằng công thức $c(x_1, x_2) + c(x_2, x_3) + \cdots + c(x_n, x_1)$. Như thế, mô hình toán học của bài toán người du lịch là: Tìm $X \in D: f(X) \to min$ rong đó D là tập các hoán vị $X = (x_1, x_2, \dots, x_n)$ của $\{1, 2, \dots, n\}$ có $x_1 = s$ (cho trước) và $f(X) = c(x_1, x_2) + c(x_2, x_3) + \cdots + c(x_n, x_1)$.

Tên gọi bài toán người du lịch mang tính chất tượng trưng, nó dùng để gọi chung cho các bài toán có mô hình toán học như trên mặc dù phát biểu có nội dung khác, chẳng hạn bài toán tìm chu trình sản xuất cho một nhà máy hóa chất sao cho chi phí xúc rửa các thiết bị (như bể chứa, ống dẫn, ...), mỗi khi chuyển từ loại hóa chất này sang loại hóa chất khác của chu trình, là ít nhất.

Bài toán cái túi

Bài toán cái túi được phát biểu như sau: "Có *n* đồ vật (đánh số từ 1 đến *n*). Với mỗi đồ vật *i*, ta biết p_i,v_i lần lượt là các trọng lượng và giá trị của vật đó (i = 1,2,...,*n*). Giả thiết có một cái túi, sức chứa không quá *w* đơn vị trọng lượng. Hãy tìm một phương án chọn đồ vật bỏ vào túi để có thể mang đi được sao cho tổng giá trị các vật được mang là lớn nhất". Một phương án chọn đồ vật là một tập con của tập 1,2,...,*n*, vì thế có thể biểu diễn mỗi phương án như thế như một dãy nhị phân *X* = (x_1,x_2,...,x_n), trong đó x_i = 1 khi và chỉ khi vật *i* được chọn (i = 1,2,...,*n*). Tổng trọng lượng của các vật được mang theo phương án này là $p_1x_1 + p_2x_2 + \dots + p_nx_n$. Điều kiện các vật được chọn mang đi được là điều kiện tổng này không vượt quá *w* (sức chứa của cái túi). Tổng giá trị các vật được mang theo phương án *X* là $v_1x_1 + v_2x_2 + \dots + v_nx_n$. Từ đó ta nhận được mô hình toán học của bài toán cái túi như sau:

Tìm $X \in D : f(X) \rightarrow max$ trong đó D là tập hợp các dãy nhị phân $X = (x_1,x_2,...,x_n)$ thỏa mãn bất đẳng thức $p_1x_1 + p_2x_2 + ... + p_nx_n \rightarrow wvf(X) = v_1x_1 + v_2x_2 + ... + v_nx_n$. Bài toán cái túi có nội dung giống như bài toán của người leo núi trước khi thám hiểm: chọn những vật đem theo sao cho sức anh ta mang được với tổng giá trị sử dụng trong chuyến leo núi là lớn nhất, vì thế bài toán này còn có tên gọi khác là bài toán của người leo núi. Bài toán người du lịch là thí dụ cho những bài toán tối ưu với mục tiêu là chi phí, còn bài toán cái túi là thí dụ cho những bài toán tối ưu với mục tiêu là hiệu quả.

Sự bùng nổ tổ hợp

Tuy nhiên, để áp dụng những phương thức khác nhau để giải các bài toán tối ưu phía trên. Có thể sẽ phải đối mặt với nhiều vấn đề tiềm ẩn khác phát sinh khi dữ liệu có số lượng lớn chiều.

Nếu ta có nhiều đặc trưng hơn so với *observations* sẽ dẫn tới nguy cơ mô hình bị vượt quá khả năng – Điều này sẽ gây ra hiệu suất cực kỳ tệ cho quá trình sample.

Hay nói cách khác là số lỗi sẽ tăng lên cùng với sự gia tăng số lượng đặc trưng. Và điều này được gọi là lời nguyền về đa chiều (*The curse of dimenstionality*) hay nói cách khác là sự bùng nổ tổ hợp. Nó dẫn đến thực tế là các thuật toán khó thiết kế hơn trong các không gian nhiều chiều và thường có thời gian chạy theo cấp số nhân của chiều. Về mặt lý thuất, số lượng kích thước cao hơn cho phép lưu trữ nhiều thông tin hơn, nhưng thực tế nó hiếm khi giúp được gì cho chúng ta do khả năng nhiễu và dư thừa cao trong dữ liệu của thế giới thực.

Thu thập một số lượng lớn dữ liệu có thể dẫn đến về nhiều chiều , trong đó các chiều phức tạp nhưng với ít thông tin hơn và không có giá trị đáng kể có thể sẽ thu phải do một lượng lớn dữ liệu.

1.1.2 Giải bải toán tối ưu

Để giải bài toán tối ưu, việc cần làm chính là tìm kiếm phương án làm cho hàm mục tiêu đạt giá trị nhỏ nhất (hoặc lớn nhất). Với các bài toán khó cỡ nhỏ,

người ta có thể tìm lời giải tối ưu nhờ tìm kiếm vét cạn. Tuy nhiên với các bài toán cỡ lỡn thì đến nay chưa thể có thuật toán tìm lời giải đúng với thời gian đa thức nên chỉ có thể tìm lời giải gần đúng hay đủ tốt

Theo cách tiếp cận truyền thống hay là tiếp cận cứng, các thuật toán gần đúng phải được chứng minh tính hội tụ hoặc ước lượng được tỷ lệ tối ưu. Với việc đòi hỏi khắt khe về toán học như vậy làm hạn chế số lượng các thuật toán công bố, không đáp ứng được nhu cầu ngày càng phong phú và đa dạng trong nghiên cứu và ứng dụng.

Có những phương pháp đưa ra được lời giải chính xác nhưng chi phí tính toán lớn, nhưng cũng có những phương pháp lại chỉ đưa ra được lời giải gần đúng, tương đối tuy thế thời gian để tính toán bài toán lại nhỏ hơn rất nhiều.

Chúng ta có thể phân thành hai phương pháp chính đó là : phương pháp giải chính xác và phương pháp giải gần đúng

Giải thuật chính xác

- Vét cạn

Vét cạn, duyệt, quay lui... là một số tên gọi tuy không đồng nghĩa nhưng cùng chỉ một phương pháp rất đơn giản trong tin học và toán học: tìm nghiệm của một bài toán bằng cách xem xét tất cả các phương án có thể. Đối với con người thì phương pháp này thường là không khả thi vì số phương án cần kiểm tra quá lớn. Tuy nhiên đối với máy tính, nhờ tốc độ xử lý nhanh, máy tính có thể giải rất nhiều bài toán bằng phương pháp này. Nhưng về cơ bản phương pháp này tốn rất nhiều thời gian và khó thực hiện, ngay cả trên những máy tính hiện đại nhất vì sự xuất hiện của bùng nổ tổ hợp

Ví dụ về bài toán liệt kê tập hoán vị của n phần tử, có $n!$ hoán ví. Nếu n nhỏ, có thể vét cạn các phương án của nó. $n = 10$, có $10!$. Tuy nhiên, nếu n lớn, $n = 15!$ ta cần phải duyệt qua 1307674368000 phương án. Giả sử máy tính có tốc độ tính toán là 1 tỉ phép tính một giây và để liệt kê hết một hoán vị thì cần phải thực hiện 100 phép tính. Như vậy để duyệt qua toàn bộ không gian lời giải cần phải mất 130767

giây ≈ 36,3 tiếng. Vì vậy cần những phương pháp tối ưu hơn, phù hợp hơn để giảm bớt không gian tìm kiếm thì mới mang lại tính khả thi để áp dụng trong thực tế khi phải đối mặt với những bài toán có kích thước lớn. Khi đó có một vấn đề được đặt ra là cần tận dụng các thông tin đã tìm được trong quá trình liệt kê lời giải để loại bỏ các phương án chắc chắn không phải là tối ưu. Và thuật toán để giúp chúng ta làm điều đó có tên là: thuật toán nhánh cận. Tác giả sẽ đề cập đến ngay sau đây. Ngoài ra tuy có nhược điểm lớn như vậy về vấn đề thời gian tính toán, tuy nhiên phương pháp này vẫn có những ưu điểm nhất định của mình. Ưu điểm lớn nhất là luôn đảm bảo tìm ra nghiệm chính xác. Một số ưu điểm khác đó là tốn rất ít bộ nhớ và cài đặt đơn giản (đối với tin học)

- Thuật toán nhánh cận

Như đã đề cập ở trên, nhánh – cận là một thuật toán cải tiến dựa trên vét cạn. Mục đích là để xây dựng những phương án khả thi, trong quá trình liệt kê, những phương án này sẽ dựa vào thông tin tìm được để loại bỏ sớm những phương án chắc chắn không phải tối ưu. Nhờ vậy không gian tìm kiếm cũng được thu hẹp lại mà vẫn đảm bảo kết quả chính xác và thời gian giải cũng giảm xuống. Thuật toán này lần đầu được giới thiệu vào năm 1960 bởi Land A.H và Doig A.G trong [8] để giải bài toán quy hoạch nguyên. Cho tới nay, phương pháp này vẫn được áp dụng rộng rãi để giải các bài toán tối ưu khó giải quyết. Trong thuật toán này, chúng ta sẽ từng bước xây dựng các phương án cho bài toán với tất cả các khả năng có thể xảy ra, trong đó mỗi nhánh của phương án đang được xây dựng bởi thuật toán sẽ chấm dứt khi biết được tổng trọng số của phương án này vượt quá giá trị cận dưới (giá trị hàm mục tiêu của phương án đã được xác định trước đó tính đến thời điểm hiện tại là tốt nhất).

Phương pháp này có thể mô hình hóa nghiệm thành một vector $X = (x_1,x_2,...,x_n)$, mỗi thành phần $x_i (i = 1,2,...,n)$ được chọn ra từ tập S_i. Mỗi nghiệm X của bài toán được xác định độ tốt bằng một hàm $f(x)$ và mục tiêu là tìm ra nghiệm X có giá trị $f(x)$ là nhỏ nhất (hoặc lớn nhất) tuỳ theo ngữ cảnh.

Tư tưởng chính của nhánh cận như sau: Giả sử ta đã xây dung được k thành phần từ x_1 đến x_k, giờ ta chuẩn bị mở rộng thành phần thứ x_k +1 từ x_k. Nhưng khi đánh giá ta lại thấy tất cả các nghiệm mở rộng từ x_k không có nghiệm nào có giá trị tốt hơn giá trị tối ưu ta đã biết tại thời điểm đó, vậy thì ta không cần mở rộng nữa, như vậy ta đã cắt bỏ đi một nhánh, giảm được số nghiệm phải tìm rất nhiều.

Điều khó nhất ở đây là phải đánh giá được các nghiệm mở rộng, nếu đánh giá được tốt, thuật toán nhánh cận sẽ chạy nhanh hơn rất nhiều so với vét cạn.

- Quy hoạch động

Phương pháp quy hoạch động dùng để giải bài toán tối ưu có bản chất đệ quy, tức là việc tìm phương án tối ưu cho bài toán đó có thể đưa về tìm phương án tối ưu của một số hữu hạn các bài toán con.

Đối với một số bài toán đệ quy, nguyên lý chia để trị (divide and conquer) thường đóng vai trò chủ đạo trong việc thiết kế thuật toán. Để giải quyết một bài toán lớn, ta chia nó thành nhiều bài toán con cùng dạng với nó để có thể giải quyết độc lập.

Điều đó càng thể hiện rõ trong phương án quy hoạch động: Khi không biết phải giải quyết những bài toán con nào, ta sẽ đi giải quyết toàn bộ các bài toán con và lưu trữ những lời giải hay đáp số của chúng với mục đích sử dụng lại theo một sự phối hợp nào đó để giải quyết những bài toán tổng quát hơn.

Và đó chính là điểm khác nhau cơ bản giữa Quy hoạch động và phép phân giải đệ quy, đây cũng chính là nội dung của phương pháp quy hoạch động:

Phép phân giải đệ quy bắt đầu từ bài toán lớn phân ra thành nhiều bài toán con và đi giải từng bài toán con đó. Việc giải từng bài toán con lại đưa về phép phân ra nhiều bài toán nhỏ hơn và lại đi giải các bài toán nhỏ hơn đó bất kể nó đã được giải hay chưa – hay còn gọi là phương pháp theo cấu trúc từ trên xuống (top-down)

Quy hoạch động bắt đầu từ việc giải tất cả các bài toán nhỏ nhất (bài toán cơ sở) để từ đó từng bước giải quyết những bài toán lớn hơn, cho tới khi giải được bài

toán lớn nhất (bài toán ban đầu) – hay còn gọi là phương pháp đi từ dưới lên (bottom-up)

Bài toán giải theo phương pháp quy hoạch động gọi là bài toán quy hoạch động. Công thức phối hợp nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là công thức truy hồi của quy hoạch động.

Tập các bài toán có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là cơ sở quy hoạch động. Không gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là bảng phương án của quy hoạch động.

Để một bài toán có thể áp dụng phương pháp quy hoạch động, ta cần phải xét xem phương pháp đó có thỏa mãn những yêu cầu dưới đây không:

- Bài toán lớn phải phân rã được thành nhiều bài toán con, có sự phối hợp lời giải của các bài toán con đó để cho ra lời giải của bài toán lớn.
- Vì quy hoạch động là đi giải tất cả các bài toán con, nên nếu không đủ không gian bộ nhớ vật lý lưu trữ lời giải (bộ nhớ, đĩa, ...) để phối hợp chúng thì phương pháp này không thể thực hiện được.
- Quá trình từ bài toán cơ sở tìm ra lời giải bài toán ban đầu phải qua hữu hạn bước.

Các bước để chuẩn bị cho một chương trình/ bài toán sử dụng quy hoạch động:

- Giải tất cả các bài toán cơ sở (thông thường rất dễ), lưu các lời giải vào bảng phương án.
- Dùng công thức truy hồi phối hợp những lời giải của các bài toán nhỏ đã lưu trong bảng phương án để tìm lời giải của các bài toán lớn hơn rồi lưu chúng vào bảng phương án. Cho tới khi bài toán ban đầu tìm được lời giải. – Dựa vào bảng phương án, truy vết tìm ra nghiệm tối ưu.

Cho tới nay, vẫn chưa có một định lý nào cho biết một cách chính xác những bài toán nào có thể giải quyết bằng phương pháp quy hoạch động là

hiệu quả nhất. Tuy nhiên ta có thể xem xét bài toán ấy có thể giải được bằng phương pháp này hay không bằng cách trả lời các câu hỏi sau:

1. "Một nghiệm tối ưu của bài toán lớn có phải là sự phối hợp các nghiệm tối ưu của các bài toán con hay không?"
2. "Liệu có thể nào lưu trữ được nghiệm các bài toàn con dưới một hình thức nào đó để phối hợp tìm được nghiệm bài toán lớn?"

Ta có thể thấy yếu tố bộ nhớ thật sự trở thành vấn đề khi mà kích thước bài toán tăng lên mà không có biện pháp lưu lời giải con một cách thích hợp. Đây chính là một điểm yếu của phương pháp này. Tuy nhiên vẫn có thể giải quyết cải tiến bằng cách giảm thiểu bộ nhớ qua xóa kết quả các bài toán con mà chúng ta chắc chắn không dùng đến ở trên.

Trên đây là những đặc điểm chính về phương pháp quy hoạch động. Qua những thuật toán trên, chúng ta có thể thấy phương pháp giải chính xác đều giúp đưa ra được kết quả cuối cùng là tối ưu. Tuy nhiên sẽ đi kèm với đó là thời gian, đôi khi sẽ lên tới hàm mũ. Do đó ảnh hưởng đến tính thực tế của các phương pháp này, gây khó khăn cho việc áp dụng vào các bài toán có kích thước lớn. Trong thực tế, đôi khi ta phải chấp nhận những lời giải gần tối ưu (tức là chưa phải tối ưu nhất) để đổi lấy thời gian tính toán. Vì vậy điều này đã dẫn đến sự ra đời của các phương pháp giải gần đúng sẽ được đề cập ở chương tiếp theo.

Giải thuật xấp xỉ (phương pháp giải gần đúng)

Giải thuật xấp xỉ được đưa ra nhằm khắc phục các hạn của giải thuật chính xác khi có sự bùng nổ tổ hợp, nghĩa là không gian tìm kiếm quá lớn mà nguyên nhân là do kích thước dữ liệu đầu vào tăng lên. Mục đích của loại thuật toán này không phải để tìm ra lời giải tối ưu chắc chắn mà để tìm ra lời giải gần tối ưu nhất nhưng trong thời gian chấp nhận được.

Đã có nhiều giải thuật gần đúng được đưa ra trong những năm gần đây. Và đã có nhiều đóng góp để mang lại những thành tựu to lớn.

Và chúng ta có thể kể ra các cách tiếp cận sau:

1. Tìm kiếm Heuristic, trong đó dựa trên phân tích toán học, người ta đưa ra các quy tắc định hướng tìm kiếm một lời giải đủ tốt.
2. Sử dụng các kỹ thuật 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 phòng tự nhiên [9, 10, 11] như mô phỏng luyện kim (Simulated Annealing - SA), giải thuật di truyền (Genetic Algorithm - GA), tối ưu bầy đàn (*Partical Swarm Optimization - PSO*), thuật toán tiến hóa (*Evolutionary Algorithms-EA*), ... Những phương pháp này là cách tiếp cận Meta-heuristic, được giới thiệu bởi Dorigo năm 1991 [12, 13, 14] đang được nghiên cứu và ứng dụng rộng rãi cho các bài toán TƯTH khó [14]

   Sau đây tác giả sẽ đi sâu vào hai cách chính đó là Heuristic và Meta-heuristic

   Heuristic

   - Phương pháp tham lam (*Gready*)

Gọi là thuật toán tham lam nhưng thực chất tham lam không được gọi là thuật toán, mà nó là một kỹ thuật, một phương pháp để ta tiến hành giải một bài toán lập trình. Vậy thì thuật toán tham lam là gì?

Thuật toán tham lam là một thuật toán giải quyết một bài toán theo kiểu Heuristic để tìm kiếm lựa chọn tối ưu ở mỗi bước với hy vọng tìm đượcc tối ưu toàn bộ. Hay nói cách khác, sự lựa chọn tốt nhất ở mỗi bước sẽ dẫn tới lời giải tối ưu nhất.

Vậy thì chọn lựa tối ưu hóa bằng cách nào?

Giả sử bạn có một hàm cần để tối ưu hóa (cực đại hóa hoặc cực tiểu hóa). Một thuật toán tham lam sẽ thực hiện các lựa chọn tham lam ở mỗi bước để đảm bảo rằng hàm đã cho là tối ưu. Thuật toán tham lam chỉ có một lần tính toán lời giải tối ưu với mục đích nó không bao giờ trở lại và đảo ngược quyết định.

Chẳng hạn áp dụng thuật toán tham lam với bài toán hành trình của người bán hàng, ta có giải thuật sau: "Ở mỗi bước hãy đi đến thành phố gần thành phố hiện tại nhất". Nói chung, giải thuật tham lam có năm thành phần:

- Một tập hợp các ứng viên (candidate), để từ đó tạo ra lời giải.
- Một hàm lựa chọn, để theo đó lựa chọn ứng viên tốt nhất để bổ sung vào lời giải.
- Một hàm khả thi (feasibility), dùng để quyết định nếu một ứng viên có thể được dùng để xây dựng lời giải.
- Một hàm mục tiêu, ấn định giá trị của lời giải hoặc một lời giải chưa hoàn chỉnh.
- Một hàm đánh giá, chỉ ra khi nào ta tìm ra một lời giải hoàn chỉnh.

Thuật toán tham lam có một vài sự thuận lợi và không thuận lợi:

- Khá dễ để tiến hành một thuật toán tham lam cho một bài toán.
- Phân tích thời gian chạy của thuật toán tham lam sẽ dễ dàng hơn kỹ thuật khác (như chia để trị). Với kỹ thuật chia để trị, không rõ ràng liệu kỹ thuật này là nhanh hay chậm. Lý do là ở mỗi mức của đệ quy kích thước nhỏ hơn và số lượng của bài toán con lớn hơn.
- Khó khăn của tham lam là bạn rất vất vả để hiểu chính xác vấn đề. Thậm chí với giải thuật chính xác rồi, cũng rất khó khăn để chứng minh tại sao nó lại đúng. Chứng minh một giải thuật tham lam sẽ có cảm giác như một nghệ thuật hơn là một môn khoa học, vì nó đòi hỏi rất nhiều sức sáng tạo.

Có một số thuật toán dựa trên tư tưởng của phương pháp tham lam thật sự tìm được phương án tối ưu (chẳng hạn thuật toán Kruscal tìm cây khung cực tiểu), còn lại đa số các thuật toán dựa trên phương pháp tham lam thường là thuật toán gần đúng, chỉ cho một lời giải xấp xỉ lời giải tối ưu.

Meta-heuristic

- Thuật toán tiến hóa (Evolutionary Algorithms- EA)

Các thuật toán tiến hóa [1,2,3] là các kỹ thuật tối ưu Meta-Heuristics dựa trên nguyên lý của Darwin về sự lựa chọn tự nhiên. Thuật toán bắt đầu với nhóm các cá thể (gọi là quần thể) trả qua các thao tác (lai ghép, đột biến) tương tụ như quá trình

sinh sản trong tự nhiên để tạo ra thế hệ các con cháu. Tiếp theo đó, các tương tác trên chính các cá thể đó bảo tồn tính di truyền làm cho một số cá thể phù hợp tốt hơn với "môi trường" và loại bỏ các cá thể xấu đối với "môi trường". Từ "môi trường" ở đây được sử dụng như một phép ẩn dụ cho ngữ cảnh của hàm mục tiêu đang được tối ưu hóa.

Có rất nhiều thuật toán tiến hóa đã được ra đời dựa trên nguyên lý này và đã được áp dụng thành công để giải các bài toán tối ưu kháhc nhau trong khoa học và kỹ thuật. Nhưng phần lớn trong cách thiết kế của các thuật toán tiến hóa đều chưa tập trung vào giải một bài toán tối ưu tại mỗi thời điểm dựa trên một quần thể mà chưa có sự quan tâm đến việc giải quyết nhiều bài toán tối ưu khác nhau đồng thời trên cùng một quần thể.

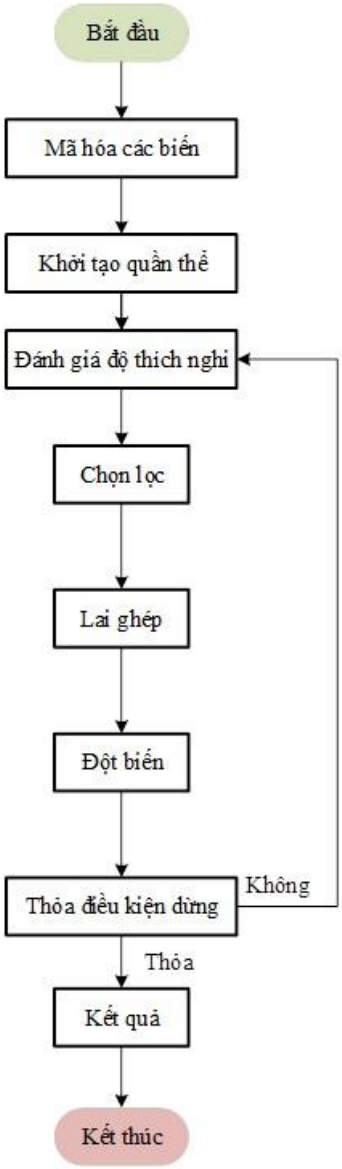
Và trong mảng giải thuật tiến hóa, đã có một số lớp giải thuật được phát triển và áp dụng phổ biến có thể kể đến sau đây:

- Giải thuật di truyền (*Genetic Algorithms -GA*) – Mô hình mô phỏng qui luật đấu tranh sinh tồn của tự nhiên.
- Lập trình di truyền (*Genetic Programming - GP*) – Phương pháp máy học nhằm tối ưu quần thể các chương trình máy tính để thực hiện một nhiệm vụ tính toán cho trước.
- Lập trình tiến hóa (*Evolution Programming*) – Mô hình mô phỏng các hành vi thích ứng trong quá trình tiến hóa (tiến hóa kiểu hình).
- Tiến hóa vi phân (*Differential Evolution*) – Có nhiều đặc điểm tương đồng với thuật toán di truyền (*GA*) nhưng có các bước lai ghép và đột biến có mô tả rõ ràng hơn bằng các công thức toán.

- Giải thuật di truyền (*Genetic Algorithms -GA*)

Trong số các giải thuật kể trên, thì giải thuật di truyền là một trong những mô hình tính toán phổ biến và thành công nhất trong lĩnh vực tính toán thông minh. Cùng với các kỹ thuật tính toán thông minh khác như tính toán mờ (*fuzzy computing*), mạng Nơ-ron (*neural networks*), hệ đa tác tử (*multiagent systems*), trí

tuệ bầy đàn (*swarm intelligence*), giải thuật di truyền ngày càng phát triển, được áp dụng rộng rãi trong các lĩnh vực của cuộc sống. Tác giả sẽ giới thiệu sơ lược về GA và những bước cơ bản trong GA là gì. Giải thuật di truyền là một kỹ thuật dựa trên cách mô phỏng sự tiến hóa của con người hay của sinh vật nói chung (dựa trên thuyết tiến hóa muôn loài của Darwin – hay nói cách khác là cùng hệ tư tưởng với thuật toán tiến hóa) trong điều kiện quy định sẵn của môi trường. Ý tưởng của giải thuật di truyền để giải một bài toán tối ưu là tìm một tập hợp của những giải pháp, sau đó cho "tiến triển" theo hướng chọn lọc để tìm những giải pháp tốt dần hơn. Mục tiêu của giải thuật di truyền là đưa ra lời giải "tốt" có thể là tối ưu hay xấp xỉ tối ưu. Cũng như các thuật toán tiến hóa nói chung, hình thành dựa trên quan niệm cho rằng quá tình tiến hóa tự nhiên là quá trình hoàn hảo nhất, hợp lý nhất và tự nó đã mang tính tối ưu. Quá tình tiến hóa tối ưu ở chỗ, thế hệ sau bao giờ cũng tốt hơn (phát triển hơn, hoàn thiện – như đã đề cập ở trên). Các cá thể mới sinh ra trong quá trình tiến hóa nhờ sự lai ghép ở thế hệ cha mẹ. Một cá thể mới có thể mang những đặc tính của cha mẹ (di truyền), cũng có thể mang những đặc tính hoàn toàn mới (đột biến). Di truyền và đột biến là hai cơ chế có vai trò quan trọng như nhau trong tiến trình tiến hóa, dù rằng đột biến xảy ra với xác suất nhỏ hơn nhiều so với hiện tượng di truyền. Các thuật toán tiến hóa tuy có những điểm khác biệt nhưng đều mô phỏng bốn quá trình cơ bản: khởi tạo, lai ghép, đột biến, sinh sản và chọn lọc tự nhiên.



Hình 1-2: Sơ đồ khối cấu trúc thuật toán di truyền

Sơ đồ thuật toán GA tổng quát có thể được biểu diễn như sau:

Thuật toán: Giải thuật di truyền	
1	Khởi tạo biến đếm t = 0
2	Khởi tạo quần thể C(0) với n cá thể
3	**while** chưa gặp điều kiện dừng **do**
4	Với mỗi cá thể $x_i(t)$, tính độ thích nghi $f(x_i(t))$ với $i = 1..n$
5	Thực hiện lai ghép và đột biến
6	Lựa chọn ra quần thể mới $C(t+1)$
7	$t = t + 1$
8	**end while**
9	**return** Cá thể tốt nhất trong quần thể

- Kỹ Thuật mã hóa

Mã hóa trong giải thuật di truyền là biểu diễn các nhiễm sắc thể chứa thông tin cho lời giải. Một số cách mã hóa được sử dụng là: mã hóa nhị phân – Binary coding, mã hóa k mức – K-nary coding, mã hóa theo số thực – Real-number coding. Quá trình mã hóa có thể biểu diễn các đầu vào thành các dãy nhiễm sắc thể theo mảng một chiều hoặc nhiều chiều.

Việc lựa chọn phương thức mã hóa tùy thuộc vào bài toán giải quyết. Thông thường hay dùng mã hóa nhị phân. Ví dụ dưới đây mô tả cách mã hóa các số thực thành các bit nhị phân:

VD: Cần mã hóa biến $z \in [x,y] \subseteq \Re$ bằng một tập các bit nhị phân $\{a1,...,aL\} \in \{0,1\}L$

Ánh xạ $\Gamma : \{0,1\}^L \rightarrow [x,y]$ sẽ được xác định như sau:

$$\Gamma(a_1, ..., a_L) = x + \frac{y-x}{2^L - 1} \cdot \left(\sum_{j-0}^{L-1} a_{L-j} - 2^j \right) \in [x, y] \qquad (1.2)$$

Như vậy. Theo cách mã hóa trên thì chỉ 2^L giá trị đầu ra được xác định. L phụ thuộc vào độ chính xác của lời giải (chính xác đến bao nhiêu chữ số thập phân), độ chính xác càng cao thì nhiễm sắc thể có độ dài càng lớn và sự tiến hóa càng chậm

- Khởi tạo quần thể

Khi chọn được cách mã hóa phù hợp, người ta tiến hành mã hóa các biến đầu vào thành các cá thể (nhiễm sắc thể), tập hợp các nhiễm sắc thể này sẽ tạo thành một quần thể. Cách dễ dàng nhất để tạo ra quần thể đó là gán các giá trị ngẫu nhiên cho biến cần tối ưu sao cho vẫn thảo mãn các ràng buộc. Việc khởi tạo ngẫu nhiên cho biến cần tối ưu sao cho vẫn thỏa mãn các ràng buộc. Việc khởi tạo ngẫu nhiên đảm bảo ràng quần thể có thể bao phủ được nhiều không gian tìm kiếm nhất có thể. Nếu không gian tìm kiếm không được bao phủ bởi quần thể ban đầu, rất có thể những phần đó sẽ bị lãng quên bởi quá trình tìm kiếm.

Kích thước của quần thể ban đầu cũng ảnh hưởng đến độ phức tạp tính toán và khả năng thăm dò vùng khả dụng. Số lượng các thể lớn làm tăng tính đa dạng, qua đó cải thiện kahr năng thăm dò của các cá thể. Tuy nhiên càng nhiều cá thể thì độ phức tạp tính toán càng tăng lên sau mỗi thế hệ. Mặc kahsc, một quần thể có kích thước nhỏ sẽ chỉ đại diện được một phần nhỏ trong không gian tìm kiếm. Mặc dù độ phức tạp về tính toán cho mỗi thế hệ là thấp, nhưng GA có thể phải sử dụng nhiều thế hệ hơn cho việc tìm kiếm. Với trường hợp quần thể với kích thước nhỏ, rất có thể phải tăng tỉ lệ đột biến để tăng vùng khám phá

Bên cạnh sử dụng sự ngẫu nhiên, sự khởi tạo có thể sử dụng từ:

+ Một quần thể cũ được lưu lại từ trước.

+ Một tập các lời giải cung cấp bởi các chuyên gia.

+ Một tập các lời giải cung cấp bởi các thuật toán tìm kiếm.

- Chọn lọc

Ở mỗi thế hệ, dựa trên giá trị của hàm mục tiêu, các cá thể có độ thích nghi tốt sẽ được chọn lọc để tạo thành quần thể ở thế hệ mới và được chuẩn bị cho việc thực hiện các phép toán lai ghép và đột biến sau này. Mục đích của phép chọn lọc là tập trung sự tìm kiếm trên miền "hứa hẹn". Phép này bắt nguyền từ học thuyết của Darwin về "Sự sống sót của các cá thể thích nghi nhất". Có hai chiến lược chọn lọc quan trọng là:

+ Lựa chọn quần thể mới: Quần thể mới được lựa chọn để đưa vào lần tiến hóa tiếp theo. Có nhiều cách lựa chọn, ví dụ như chỉ lựa chọn từ con cái hoặc lựa chọn từ cả cha mẹ và con. Toán tử lựa chọn phải đảm bảo rằng các cá thể tốt có cơ hội sống sót tốt hơn những cá thể khác trong quần thể.
+ Lựa chọn cha mẹ: Con cái được tạo ra bằng cách lai ghép và đột biến. Trong quá trình lai ghép, các cá thể có độ thích nghi cao nên có nhiều cơ hội sinh sản hơn để đảm bảo rằng có con cái có chứa chất liệu di truyền tốt.

Một số phép chọn lọc thường được sử dụng bao gồm:

+ Roulette wheel Selection - Chọn lọc ngẫu nhiên theo bánh xe Roulette;
+ Fitness Proportionate Selection- Chọn lọc theo tỷ lệ thích nghi;
+ Linear Ranking Selection- Chọn lọc theo thứ hạng tuyến tính;
+ Local Tournament Selection- Chọn lọc theo cạnh tranh cục bộ.

- Lai ghép

Trong giải thuật di truyền, số lượng các thể trong quần thể ở mỗi thế hệ là không đổi. Phép chọn lọc đã chọn ra một số cá thể có độ thích nghi cao và loại bỏ đi một số cá thể thích nghi thấp. Sự thiếu hụt của số lượng quần thể khi mất đi các cá thể thích nghi thấp sẽ được bổ xung bằng việc lấy các cá thể có độ thích nghi cao là thế hệ cha mẹ, tạo ra các thế hệ con cái bằng phép lai ghép và đột biến trên các cá thể thích nghi cao này. Kết quả là thế hệ mới được hình thành giữ nguyên về số

lượng bao gồm các cá thể thích nghi cao và con cái của chúng qua các phép lai ghép và đột biến.

Phép lai ghép là tạo ra các nhiễm sắc thể con cái (*offspring*) từ các nhiễm sắc thể cha mẹ (*parent*) được lựa chọn.

- Đột biến

Thuật toán tiến hóa được đề cập ở mục này chính là đối tượng nghiên cứu chính của tác giả trong bài viết này. Trong chương tiếp theo tác giả sẽ đi sâu hơn và phân tích chi tiết về thuật toán này cùng những yếu tố xung quanh đối tượng.

- Thuật toán bầy đàn (Partical Swarm Optimization - PSO)

PSO được giới thiệu đầu tiên vào năm 1995 bởi James Kennedy và C. Eberhart [2]. Thuật toán có nhiều ứng dụng quan trọng trong tất cả các lĩnh vực mà ở đó đòi hỏi giải quyết các bài toán tối ưu hóa. Thuật toán PSO được xây dựng dựa vào quá trình mô phỏng sinh học của đàn chim. Để hiểu rõ về thuật toán PSO, hãy xem một ví dụ đơn giản về quá trình tìm kiếm thức ăn của một đàn chim. Không gian tìm kiếm thức ăn lúc này là toàn bộ không gian ba chiều mà chúng ta đang sinh sống. Tại thời điểm bắt đầu tìm kiếm cả đàn bay theo một hướng nào đó, có thể là ngẫu nhiên. Tuy nhiên, sau một thời gian tìm kiếm một số cá thể trong đàn bắt đầu tìm được ra nơi có chứa thức ăn. Tùy theo số lượng thức ăn vừa tìm kiếm mà các cá thể gửi tín hiệu đến các các thể đang tìm kiếm ở vùng lân cận. Tín hiệu này lan truyền trên toàn quần thể. Dựa vào thông tin nhận được, mỗi cá thể sẽ điều chỉnh hướng bay và vận tốc bay theo hướng về nơi có nhiều thức ăn nhất. Cơ chế truyền tin như vậy thường được xem là một kiểu hình của trí tuệ bầy đàn. Cơ chế này giúp cả đàn chim tìm ra nơi có nhiều thức ăn nhất trên không gian tìm kiếm.

Ta bắt đầu xem xét sự liên hệ giữa bài toán tìm thức ăn với bài toán tìm cực tiểu (cực đại) theo cách như sau. Giả sử rằng số lượng thức ăn tại một vị trí tỉ lệ nghịch với giá trị của hàm *f* tại vị trí đó. Có nghĩa là ở một vị trí mà giá trị của hàm *f* càng nhỏ (càng lớn) thì số lượng thức ăn càng lớn (càng nhỏ). Việc tìm vùng chứa

thức ăn nhiều nhất tương tự như việc tìm ra vùng chứa điểm cực tiểu của hàm *f* trên không gian tìm kiếm .

PSO được khởi tạo bằng một nhóm cá thể (nghiệm) ngẫu nhiên $p_1,p_2,...,p_{Npop}$, và sau đó, tìm nghiệm tối ưu bằng cách cập nhật các thế hệ. Trong mỗi thế hệ, mỗi cá thể p_i được cập nhật theo hai giá trị tốt nhất. Giá trị thứ nhất là nghiệm tốt nhất ở thời điểm hiện tại, gọi là $pbest_i$. Một nghiệm tối ưu khác mà các cá thể này bám theo là nghiệm tối ưu toàn cục *gbest*, đây là nghiệm tốt nhất mà cá cá thể lân cận cá thể này đạt được cho tới thời điểm hiện tại. Nói cách khác, mỗi cá thể trong quần thể cập nhật vị trí theo vị trí tốt nhất của nó và vị trí tôt nhất của các thể trong quần thể tính đến thời điểm hiện tại. Đối với mỗi cá thể p_i chúng ta có:

- $X_i^t[]$ là vị trí của cá thể *i* ở thế hệ thứ *t*,
- $V_i^t[]$ là vận tốc của cá thể *i* ở thế hệ thứ *t*
- $pbest^t{}_i[]$ là vị trí tốt nhất của cá thể *i* ở thế hệ thứ *t*,

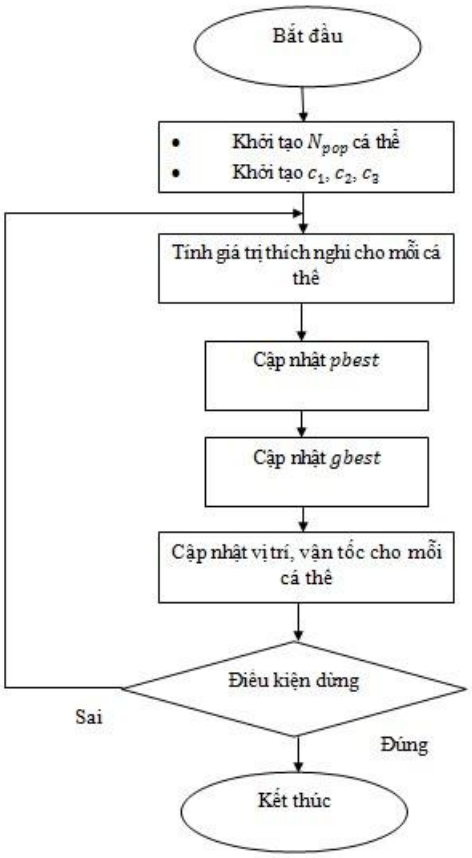
$$pbest_i^t[] = \begin{cases} pbest_i^{t-1}[], & nếu\ f(X_i^t[]) \le f(pbest_i^{t-1}[]) \\ X_i^t[], & nếu\ f(X_i^t[]) > f(pbest_i^{t-1}[]). \end{cases} \quad (1.3)$$

- *gbest*(*t*) là cá thể tốt nhất của tất cả các cá thể ở thế hệ thứ *t*

$$gbest^t[] \in \{pbest^t{}_1[],...,pbest^t{}_{Npop}[]\} | f(gbest^t[]) = max\{f(pbest^t{}_1[]),...,f(pbest^t{}_{Npop}[])\}. \quad (1.4)$$

Thuật toán PSO bao gồm ba bước chính được lặp đi lặp lại cho đến khi gặp điều kiện dừng [2].

1. Bước 1: Đánh giá độ thích nghi của các cá thể,
2. Bước 2: Cập nhật các thể tốt nhất ở thời điểm hiện tại, và các cá thể tốt nhất qua các thế hệ,
3. Bước 3: Cập nhật vận tốc và vị trí của mỗi cá thể.



Hình 1-3: Sơ đồ khối thuật toán PSO

Hai bước đầu tiên của thuật toán khá đơn giản. Ở bước thứ nhất, việc đánh giá độ thích nghi của các cá thể được tính thông qua một hàm lượng giá gọi là hàm *fitness*, hàm này thông thường chính là hàm mục tiêu của mỗi bài toán. Ở bước thứ hai, các cá thể tốt nhất ở thời điểm hiện tại và các cá thể tốt nhất qua các thế hệ được cập nhật bằng cách so sánh hàm lượng giá mới được đánh giá theo các cá thể hiện tại và các cá thể tốt nhất trước đây. Bước cập nhật vận tốc và vị trí là bước quan trọng nhất, chịu trách nhiệm về khả năng tối ưu hóa của thuật toán PSO. Vận tốc của mỗi cá thể được cập nhật bằng cách sử dụng phương trình sau:

$$V_i^{t+1}[] = w * V_i^t[] + c_1 * r_1 * (pbest^t{}_i[] - X_i^t[]) + c_2 * r_2 * (gbest^t[] - X_i^t[]). \quad (1.5)$$

Ba thành phần của phương trình cập nhật vận tốc đóng các vai trò khác nhau trong thuật toán PSO.

- Thành phần đầu tiên $w * V_i^t[]$ là *thành phần quán tính* giữ cho các cá thể chuyển động theo cùng một hướng. w được gọi là trọng số quán tính thường nhận các giá trị

nằm giữa 0.8 và 1.2, nó có vai trò làm tăng hoặc giảm quán tính theo hướng ban đầu của nó. Nói chung, các giá trị nhỏ hơn của trọng số quán tính làm tăng tốc độ hội tụ của cả bầy đàn để tối ưu, các giá trị lớn hơn của trọng số quán tính có vai trò mở rộng không gian tìm kiếm.

- Thành phần thứ hai$c_1 * r_1 * (pbest_i^t[] - X_i^t[])$ được gọi là *thành phần nhận thức* hoạt động như bộ nhớ của cá thể, nó có xu hướng quay trở lại khu vực của không gian tìm kiếm mà nó đạt được giá trị của hàm lượng giá cao. Hệ số nhận thức c_1 thường là 2 và ảnh hưởng đến các cá thể khi tiến đến cá thể tốt nhất $pbest^t_i$.
- Thành phần thứ ba $c_2 * r_2 * (gbest^t[] - X_i^t[])$, được gọi là *thành phần xã hội*, làm cho các cá thể di chuyển đến khu vực tốt nhất của cả bầy đã tìm thấy. Hệ số xã hội c_2 thường là 2, và ảnh hưởng đến các cá thể khi tiến đến cá thể tốt nhất *gbest* trong lịch sử của cả bầy.
- Ngoài ra, các giá trị ngẫu nhiên r_1, r_2 trong *thành phần nhận thức* và *thành phần xã hội* có những tác động ngẫu nhiên trong quá trình cập nhật vận tốc.

Để cho các cá thể không di chuyển quá xa khỏi không gian tìm kiếm, thuật toán PSO thường sử dụng một kỹ thuật được gọi là kỹ thuật vận tốc kẹp để hạn chế tốc độ tối đa của mỗi cá thể. Đối với một không gian tìm kiếm được giới hạn bởi phạm vi $[-x_{max}, x_{max}]$, vận tốc kẹp giới hạn trong khoảng $[-v_{max}, v_{max}]$, trong đó $v_{max} = k \times x_{max}$. Giá trị k là tham số được định nghĩa trước, nó thường nằm trong khoảng $0.1 \leq k \leq 1.0$.

Sau khi tính toán vận tốc cho mỗi cá thể, mỗi vị trí của các cá thể được cập nhật bằng cách áp dụng công thức:

$$X_{it+1}[] = X_{it}[] + V_{it+1}[]. \quad (1.6)$$

Thuật toán PSO là một quá trình ngẫu nhiên, nên chúng ta không thể đảm bảo chắc chắn nó sẽ dừng sau hữu hạn bước. Vì vậy, để đảm bảo thuật toán PSO sẽ kết thúc, người dùng thường phải định nghĩa điều kiện dừng cho thuật toán. Một vài trường hợp dừng thông thường như sau:

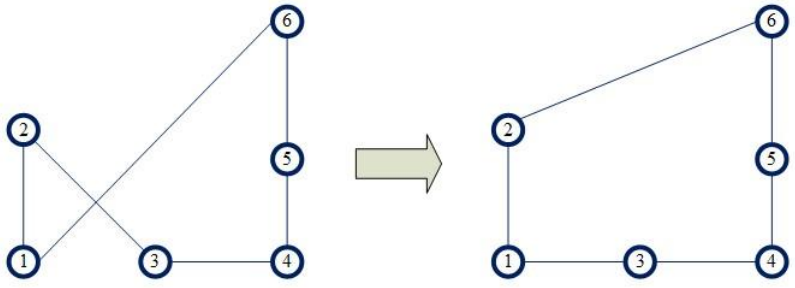
1. Kết thúc theo kết quả: Một khi đạt đến mức giá trị yêu cầu thì kết thúc kết quả thực hiện.
2. Kết thúc dựa vào số lần lặp: Thuật toán dừng khi kết thúc một số hữu hạn các lần lặp.
3. Kết thúc theo thời gian: không quan tâm đến số lần lặp và kết quả như thế nào, giải thuật sẽ kết thúc sau một thời gian quy định nào đó.
4. Tổ hợp: dùng nhiều phương pháp khác nhau để kết thúc giải thuật: chạy theo số lần lặp, tiếp đến đánh giá cho kết quả....

• Thuật toán tìm kiếm cục bộ (Local search)[39, 40]

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 thưởng dùng là lân cần 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 Người du lịch bao gồm tất cả các lời giải s^1 có thể nhận được từ *s* bằng cách đổi hai cạnh. Hình 1.4 chỉ ra một ví dụ một lời giải nhận được bằng cách thay cạnh (2,3), (1,6) bằng hai cạnh (1,3), (2,6).

Bên cạnh những ưu điểm của giải thuật xấp xỉ, tìm kiếm cục bộ có nhược điểm là thường chỉ cho cực trị địa phương.

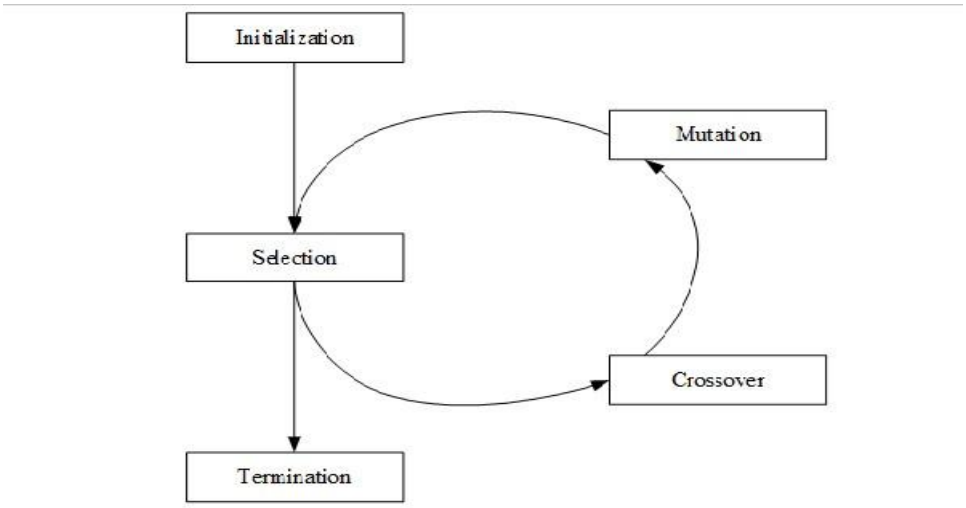


Hình 1-4: Lời giải nhận được nhờ thay 2 cạnh (2,3), (1,6) bằng (1,3), (2,6)

1.2 Thuật toán tiến hóa

Như đã được đề cập ở chương trên, Các thuật toán tiến hóa (EA) là một cách tiếp cận Meta-heuristic nhằm giải quyết các vấn đề không thể dễ dàng giải quyết trong thời gian đa thức (Polunomial Time), chẳng hạn như các vấn đề NP-khó cổ điển, và những vấn đề khác phải mất rất nhiều thời gian để có thể xử lý triệt để, hay nói cách khác là ra được kết quả tối ưu. Khi được sử dụng riêng, chứng thường được áp dụng cho các vấn đề tổ hợp; tuy nhiên, các thuật toán di truyền thường được sử dụng song song với các phương pháp khác, hoạt động như một cách nhanh chóng để tìm một nơi bắt đầu tối ưu để một thuật toán khác hoạt động. Nói cách khác EA sẽ giúp tạo tiền đề để sử dụng thuật toán thứ 2, thứ 3 tìm ra kết quả mong muốn với độ chính xác cao hơn.

Tiền đề của một thuật toán tiến hóa khá là đơn giản, chúng ta có thể dễ dàng tóm tắt lại dựa theo quá trình chọn lọc tự nhiên. Một EA chứa bốn bước tổng thể: khởi tạo, chọn lọc, vận hành di truyền và chấm dứt. Các bước này mỗi bước tương ứng, đại khái, cho một khía cạnh cụ thể của chọn lọc tự nhiên và cung cấp các cách dễ dàng để mô đun hóa các triển khai của thể loại thuật toán này. Nói một cách đơn giản, trong một EA, các thành viên phù hợp sẽ tồn tại và sinh sôi nảy nở, trong khi các thành viên khác không còn phù hợp sẽ chết đi và không đóng góp vào nhóm gen của các thế hệ tiếp theo nữa, giống như chọn lọc tự nhiên.



Hình 1-5: Mô tả các bước tổng thể của các EA

EA đã được sử dụng rộng rãi để giải quyết các vấn đề tối ưu hóa tương ứng trong nhiều lĩnh vực, ví dụ: tối ưu hóa mục tiêu đơn – Single-objective Optimization [Guo và Yang, 2015], tối ưu hóa đa mục tiêu – Multiobjective Optimization [Coello, 2006; Fonseca và Fleming, 1995] và như vậy.

Các thuật toán tiến hóa hầu hết đều được thiết kế để tập trung vào việc giải một bài toán tối ưu tại mỗi thời điểm dựa trên một quần thể mà chưa có sự quan tâm đến việc giải quyết nhiều bài toán tối ưu khác nhau đồng thời trên cùng một quần thể. Và do đó dẫn đến sự ra đời của một dạng hoàn toàn mới của các thuật toán tiến hóa, đó là đa tác vụ tiến hóa (evolutionary multitasking), cho phép giải đồng thời nhiều bài toán tối ưu khác nhau trên một quần thể duy nhất và được gọi là tối ưu hóa đa nhân tố (multifactorial optimization- MFO) [29].

Chương 2 TIẾN HÓA ĐA NHÂN TỐ

2.1 Các khái niệm liên quan

Trong phần này, tác giả sẽ giải thích các khái niệm của multifactorial optimization (MFO) và mô tả hiệu suất tương đối của một cá nhân trong một dân số được đánh giá trong một môi trường đa tác vụ.

Trước khi tiếp tục, điều quan trọng là phải xem xét đầu tiên của mối quan hệ giữa các nhiệm vụ cấu thành trong MFO. Nó là giá trị để đề cập đến khái niệm về đa tác vụ tiến hóa không áp đặt bất kỳ hạn chế nghiêm ngặt về mối quan hệ giữa các tác vụ. Cụ thể, các tác vụ có thể là khác biệt. hoặc là các thành phần phụ thuộc lẫn nhau của một vấn đề nhiều thành phần lớn. Tuy nhiên, trong bài viết này, tác giả sẽ tập trung vào trường hợp khi không có kiến thức về bất kỳ sự phụ thuộc giữa các tác vụ. Nói cách khác, chỉ xem xét đa nhiệm trên các vấn đề tối ưu hóa mà thường được xem độc lập (như các tác vụ khác biệt) . Vì vậy, mục đích của MFO không phải là để tìm ra sự cân bằng tối ưu giữa các hàm mục tiêu cấu thành. Thay vào đó, mục tiêu là tối ưu hóa toàn bộ tác vụ một cách đầy đủ và đồng thời, với sự trợ giúp của sự tìm kiếm dựa trên quần thể.

Xem xét tình huống với K tác vụ tối ưu hóa được thực hiện đồng thời. Không mất tính tổng quát, giả sử rằng tất cả các tác vụ đều là bài toán tối thiểu hóa (minimization problem). Tác vụ thứ j^{th} ký hiệu là T_j, được xem là có không gian tìm kiếm X_j với hàm mục tiêu là $f_i : X_j \rightarrow R$. Thêm nữa, mỗi một tác vụ có thể bị ràng buộc bởi một số các phương trình và bất phương trình điều kiện phải được thỏa mãn đối với một giải pháp được xem là phù hợp.

Trong bối cảnh như vậy, người ta định nghĩa MFO là một chiến lược đa nhiệm tiến hóa được xây dựng trên sự song song hóa hoàn toàn của việc tìm kiếm dựa vào quần thể với mục đích tìm $\{x_1,x_2,...,x_{K-1}1,x_K\} = argmin\{f_1(x),f_2(x),...,f_{K-1}(x),f_K(x)\}$, trong đó x_j là một giải pháp phù hợp trong X_j.

Do các nguyên tắc cơ bản trong thiết kế của một thuật toán tiến hóa đều dựa trên nguyên tắc chọn lọc tự nhiên của Darwin, trước tiên cần phải định lượng "thể lực" (fitness) của một cá thể trong một môi trường đa tác vụ. Dựa theo đó, ta sẽ xác định một tập hợp các thuộc tính của mỗi p_i riêng lẻ, trong đó $i \in \{1,2,...,|P|\}$, trong một quần thể P. Hãy nhớ rằng vì MFEA được cung cấp một không gian tìm kiếm thống nhất *Y* bao gồm X_1,X_2,...,vX_k, mỗi cá thể có thể được giải mã thành một đại diện giải pháp riêng cho mỗi tác vụ đối với từng nhiệm vụ *K*.Dạng giải mã của cá thể pi có thể được viết như là$\{x_1^i, x_2^i, \ldots, x_K^i\}$, trong đó$x_1^i \in X_1, x_2^i \in X_2, \ldots$ và $x^i_K \in X_K$.

Định nghĩa 1 (*Factorial Cost* – giá của của một cá thể đối với từng tác vụ):

Factorial cost Ψ^i_j của cá thể p_i đối với tác vụ T_j là $\Psi^i_j = \lambda.\delta_j^i + f_j^i$; trong đó λ là một hằng số phạt lớn, f_j^i và δ_j^i là giá trị mục tiêu và tổng số sự vi phạm ràng buộc tương ứng của p_i đối với tác vụ T_j. Do vậy, nếu pi phù hợp đối với tác vụ T_j (sự vi phạm ràng buộc bằng 0), chúng ta có $\Psi^i_j = f_j^i$.

Định nghĩa 2 (*Factorial rank* – Xếp hạng của một cá thể đối với từng tác vụ):

Factorial rank r_j^i của cá thể pi trên tác vụ T_j là chỉ số (index) cuả p_i trong quần thể *P* được sắp sếp theo thứ tự tăng dần của Ψ^i_j.

Trong khi gán Factorial rank, mỗi khi $\Psi_j^a = \Psi_j^b$ đối với một cặp cá thể p_a và p_b, thứ tự trước sau được lựa chọn ngẫu nhiên. Tuy nhiên, hiệu năng của hai cá thể là tương đương đối với tác vụ thứ j^{th}, vì vậy gán nhãn chung như là $j-counterparts$ (bản sao hay giống nhau đối với tác vụ thứ j^{th}).

Định nghĩa 3 *(Scalar Fitness* – Đo độ thích nghi):

Danh sách các xếp hạng của một cá thể pi trên tất cả các tác vụ (*K* tác vụ) $\{r_1^i, r_2^i, \ldots, r_K^i\}$ có thể được biến đổi sang một hình thức đơn giản hơn là giá trị Scalar Fitness ϕ_i dựa trên thứ tự xếp hạng tốt nhất của p_i trên tất cả các tác vụ, nghĩa là $\varphi_i = 1/\min_{j\in\{1,2,\ldots,K\}}\{r_j^i\}$.

Định nghĩa 4 (*Skill Factor* - Tác vụ đầy đủ or tác vụ tốt nhất (nhân tố kỹ năng):

Skill factor τ_i của cá thể p_i là tác vụ trong số tất cả các tác vụ của MFO mà cá thể p_i là hiệu quả nhất, nghĩa là $\tau_i = argmin\{r_j^i\}$ trong đó $j \in \{1,2,...,K\}$.

Khi sự thích hợp của các cá thể được qui theo định nghĩa 3, so sánh hiệu năng có thể được thực hiện một cách dễ dàng. Ví dụ, p_a được xem là trội hơn p_b trong ngữ cảnh đa nhân tố nếu $\phi_a > \phi_b$. Nếu hai cá thể có cùng skill factor $\tau_a = \tau_b = j$ (phù hợp với tác vụ thứ $j\{th\}$ nhất), gán nhãn chúng là strong counterparts (giống nhau mạnh).

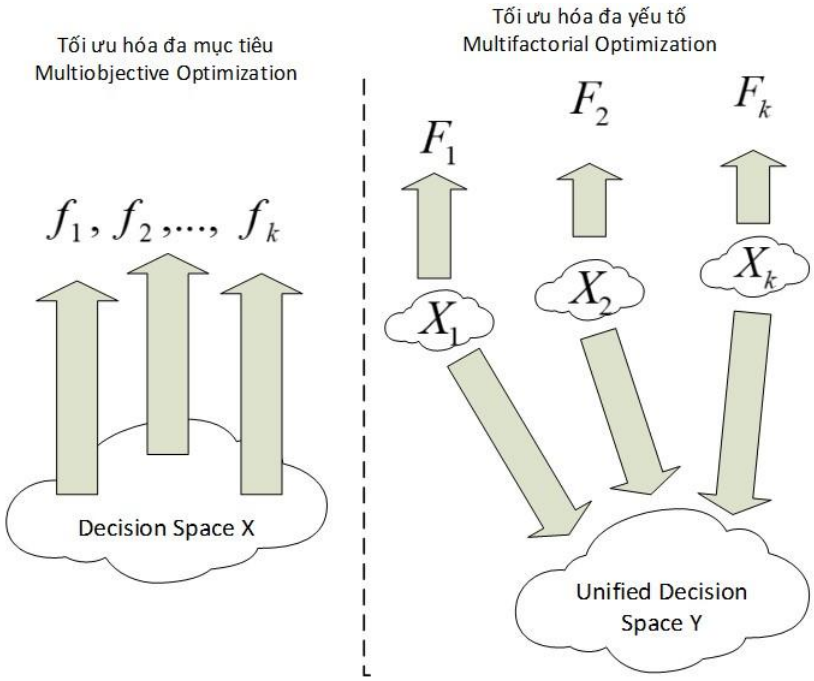
Điều quan trọng cần lưu ý là các thủ tục được mô tả từ trước đến nay để so sánh các cá thể không phải là tuyệt đối. Vì factorial rank của một cá thể (và rõ ràng scalar finess và skill factor của nó) phụ thuộc vào hiệu suất của mọi cá thể khác trong quần thể, sự so sánh là phụ thuộc vào quần thể thực tế. Tuy nhiên, thủ tục đảm bảo rằng nếu một cá thể p^* ánh xạ tới sự tối ưu hóa toàn cục của một tác vụ bất kỳ thì $\phi^* \geq \phi_i$ với mọi $i \in \{1,2,...,K\}$.

Định nghĩa 5 (*Multifactorial optimality* - Sự tối ưu đa nhân tố): Một cá thể p^*, với một danh sách các giá trị mục tiêu $\{f_1^*, f_2^*, ..., f_K^*\}$, được xem là tối ưu hóa trong ngữ cảnh đa nhân tố nếu và chỉ nếu $\exists j \in \{1,2,...,K\}$sao cho $f_j^* \leq f_j(x_j)$ đối với mọi x_j phù hợp trong X_j.

Tối ưu hóa đa yếu tố (MFO) vs Tối ưu hóa đa mục tiêu (MOO):

Vì tối ưu hóa đa mục tiêu và MFO đều liên quan đến việc tối ưu hóa một tập hợp các hàm mục tiêu, nên có thể thấy sự tương đồng về khái niệm tồn tại giữa chúng. Tuy nhiên, cần nhấn mạnh một sự khác biệt cơ bản giữa hai mô hình này. Trong khi Tiến hóa đa tác vụ nhằm thúc đẩy sự tương đồng ẩn tàng của quá trình tìm kiếm dựa trên dân số để khai thác sự bổ sung tìm ẩn giữa các tác vụ riêng biệt, Tối ưu hóa đa mục tiêu giải quyết xung đột giữa các mục tiêu cạnh tranh của cùng một nhiệm vụ. Qua minh họa tóm tắt ở hình 2.1. Nó cho thấy rằng sự tồn tại đồng thời của nhiều không gian tìm kiếm không đồng nhất trong trường hợp đa tác vụ. Mặc khác, để tối ưu hóa đa mục tiêu, thường tồn tại một không gian tìm kiếm cho

một tác vụ nhất định, với tất cả các hàm mục tiêu tùy thuộc vào các biến có trong không gian đó. Lưu ý: trên thực tế, một môi trường đa nhiệm có thể bao gồm một tác vụ tối ưu hóa đa mục tiêu như một trong nhiều tác vụ đồng thời khác.



Hình 2-1:Nổi bật sự khác biệt giữa tối ưu hóa đa mục tiêu và đa yếu tố

2.2 Giải thuật tiến hóa đa nhân tố

Trong tiểu mục này, tác giả sẽ trình bày về Thuật toán tiến hóa đa nhân tố (MFEA), một bộ khung đa tác vụ có hiệu quả dựa trên các mô hình văn hóa sinh học của kế thừa đa yếu tố [34]. Vì các hoạt động của phương pháp này dựa trên việc truyền các yếu tố sinh học cũng như các khối văn hóa từ phía bố mẹ sang con cái, MFEA được coi là thuộc về lĩnh vực tính toán ghi nhớ – một lĩnh vực mà gần đây đang nổi lên như một mô hình tính toán thành công tổng hợp các nguyên tắc chọn lọc tự nhiên của Darwin với khái niệm Dawkins về meme như là một đơn vị cơ bản của tiến hóa văn hóa.

2.3 Khởi tạo quần thể

Trong phần này, tác giả sẽ đi vào phân tích bộ mã giả của MFEA. Như đã chỉ ra ở mục 2.3.1, MFEA bắt đầu bằng cách tạo ngẫu nhiên một quần thể gồm n cá thể trong một không gian tìm kiếm đơn nhất *Y*. Ngoài ra, mỗi cá thể trong tập hợp quần

thể đều được chỉ định trước một yếu tố kỹ năng cụ thể (Xem định nghĩa 4) theo cách đảm bảo cho mọi tác vụ đều có số lượng đại diện hợp lý. Nhấn mạnh rằng yếu tố kỹ năng của một cá thể (nghĩa là, tác vụ mà cá thể đó được liên kết) được xem như là một đại diện tính toán của đặc điểm văn hóa được chỉ định trước đó. Ý nghĩa của bước này là đảm bảo rằng một cá thể chỉ được đánh giá liên quan đến một tác vụ duy nhất (nghĩa là chỉ có nhân tố kỹ năng của nó) trong số tất cả các tác vụ khác trong môi trường đa tác vụ. Làm như vậy được coi là thiết thực vì việc đánh giá toàn bộ từng cá thể cho mọi tác vụ thường sẽ đòi hỏi tính toán, đặc biệt là khi K (số lượng tác vụ trong môi trường đa tác vụ) trên nên lớn hơn. Phần còn lại của MFEA tiến hành tương tự như bất kỳ quy trình thực thi tiến hóa tiêu chuẩn nào. Trên thực tế, phải dề cập ở đây chính là các cơ chế di truyền cơ bản có thể được mượn từ bất kỳ thuật toán dựa trên quần thể nào có sẵn trong tài liệu, ghi nhớ tính chất và yêu cầu của vấn đề đa tác vụ trong tầm tay. Sự sai lệch đáng kể duy nhất so với cách tiếp cận truyền thống xảy ra trong điều kiện đánh giá con cái, tính đến các đặc điểm văn hóa thông qua các yếu tố kỹ năng.

Thuật toán 1: Câu trúc cơ bản của MFEA

1 Khởi tạo quần thể ban đầu và lưu trữ trong *current-pop* (*P*)

2 Đánh giá mọi cá thể đối với mọi tác vụ tối ưu trong môi trường đa tác vụ

3 Tính toán skill factor τ của mỗi cá thể

4 **While** (điều kiện dừng chưa thỏa mãn) **do**

- i) Áp dụng các thao tác di truyền tới *current-pop* (*P*) để sinh ra *offspring-pop* (*C*) [*Xem thuật toán 2*]
- ii) Đánh giá các cá thể trong *offspring-pop* **chỉ đối với các tác vụ được lựa chọn** [*Xem thuật toán 3*]
- iii) Gộp *current-pop* (*P*) và*offspring-pop* (*C*) tạo thành *intermediate-pop* (*P*∪*C*)
- iv) Cập nhật *scalar fitness* và *skill factor* cho mọi cá thể trong *intermediate-pop* (*P*∪*C*)
- v) Lựa chọn các cá thể tốt nhất từ *intermediate-pop* (*P*∪*C*) để hình thành *current-pop* (*P*) kế tiếp.

5 **End while**

Hình 2-2: Mô tả các bước tổng thể của các EA(thuật toán 1)

Giả sử rằng K tác vụ tối ưu được thực hiện đồng thời, số chiều của tác vụ thứ j^{th} là D_j. Theo đó, chúng ta định nghĩa không gian tìm kiếm hợp nhất với số chiều là

$D_{multitask} = max_j\{D_j\}$ với $j \in \{1,2,...,K\}$. Vectơ này được gọi là *chromosome* của cá thể. Trong bước khởi tạo quần thể, mọi cá thể đều là một vectơ của $D_{multitask}$ các biến ngẫu nhiên (có giá trị trong khoảng [0,1]). Về cơ bản, chiều thứ i^{th} của không gian tìm kiếm hợp nhất được biểu diễn bởi một khóa ngẫu nhiên y_i, và khoảng giá trị cố định (từ 0 đến 1) biểu diễn ràng buộc của không gian hợp nhất. Với các biểu diễn như vậy, đối với tác vụ T_j trong cá thể p_i, chúng ta sẽ sử dụng Dj khóa ngẫu nhiên đầu tiên của *chromosome* tương ứng với p_i. Động lực đằng sau việc sử dụng một kỹ thuật mã hóa như vậy, thay vì chỉ đơn giản là ghép các biến của từng nhiệm vụ tối ưu hóa để tạo thành một *chromosome* khổng lồ của $D_1 + D_2 + ... + D_K$ là:

a. Từ quan điểm thực tế là nó sẽ giúp tránh các thách thức liên quan tới lời nguyền của số chiều (*The curse of dimensionality*) khi một số tác vụ với không gian tìm kiếm đa chiều sẽ được giải quyết đồng thời.

b. Trên cơ sở lý thuyết, nó được coi là một phương tiện hiệu quả để tiếp cận sức mạnh của tìm kiếm dựa trên quần thể. Như *schemata* (hay khối xây dựng di truyền) [32] tương ứng với các tác vụ tối ưu hóa khác nhau được chứa trong một nhóm vật liệu di truyền thống nhất, chúng được EA xử lý song song. Quan trọng nhất,là điều này sẽ khuyến khích việc phát hiện và chuyển giao vật liệu di truyền hữu ích từ nhiệm vụ này sang nhiệm vụ khác một cách hiệu quả. Hơn nữa, vì một cá nhân trong quần thể có thể thừa hưởng các khối xây dựng di truyền tương ứng với nhiều tác vụ tối ưu hóa, sự tương đồng với di truyền da yếu tô trở nên có ý nghĩa hơn.

2.4 Kỹ thuật di truyền

Tương tự như các thuật toán EA cổ điển, MFEA cũng sử dụng hai toán tử di truyền đó là *lai ghép (crossover)* và *đột biến (mutation)*. Điểm khác biệt chính của MFEA là hai cá thể cha mẹ được lựa chọn ngẫu nhiên cho phép toán lai ghép phải đáp ứng một số điều kiện. Nguyên tắc tiếp theo là việc lai ghép ngẫu nhiên chỉ ra rằng các cá thể thích kết hợp với cá thể thuộc cùng một kiểu "văn hoá": Trong MFEA, *skill factor* (τ) được xem như là một đại diện tính toán của sự thiên vị văn hoá *(culture bias)* của một cá thể. Do vậy, hai cá thể cha mẹ được lựa chọn ngẫu nhiên chỉ được thực hiện lai ghép nếu chúng có cùng *skill factor*. Ngược lại, nếu

skill factor là khác nhau, sự lai ghép chỉ xảy ra theo một xác suất lai ghép ngẫu nhiên được qui định *(rmp)* hoặc phép đột biến được sử dụng. Các bước tạo ra các cá thể con được mô tả trong thuật toán 2.

Thuật toán 2: Các thao tác di truyền

Xét hai cá thể cha mẹ p_a và p_b được lựa chọn ngẫu nhiên trong *current-pop* (P)

Sinh một số ngẫu nhiên *rand* trong khoảng [0,1]

If ($\tau_a = \tau_b$) hoặc (*rand* <*rmp*) **then**

i) Thực hiện lai ghép hai cá thể p_a và p_b sinh ra hai cá thể con c_a và c_b

else

i) Đột biến p_a sinh ra cá thể con c_a

ii) Đột biến p_b sinh ra cá thể con c_b

End if

Hình 2-3: Mô tả các bước tổng thể của các EA (thuật toán 2)

Sự xuất hiện của giao phối chọn lọc trong thế giới tự nhiên được sử dụng trong các mô hình thừa kế đa yếu tố nhằm để giải thích các đặc điểm phả hệ kéo dài qua nhiều thế hệ [25]. Ở đây trong bài viết này, Tham số *rmp* được sử dụng để cân bằng khai thác và tìm kiếm không gian tìm kiếm. Giá trị của *rmp* gần với 0 hàm ý rằng chỉ các cá thể có "văn hoá" giống nhau mới được lai ghép, trong khi giá trị gần 1 cho phép lai ghép hoàn toàn ngẫu nhiên. Trong trường hợp trước, lai ghép nội văn hóa – *intra-cultural mating* (nghĩa là giữa các ứng cử viên mẹ có cùng *skill factor*) và các biến thể di truyền nhỏ do đột biến (xảy ra khi các ứng viên cha mẹ có các yếu tố kỹ năng khác nhau), tạo điều kiện thuận lợi cho việc quét các vùng giới hạn của không gian tìm kiếm. Kết quả là luôn có xu hướng các giải pháp bị mắc lại trong tối ưu cục bộ. Ngược lại, sự lai ghép giao văn hóa – *Cross-cultural mating* xu hướng tăng lên khả năng xảy ra dưới giá trị lớn hơn của *rmp* (gần tới 1) cho khém khám phá toàn bộ không gian tìm kiếm, từ đó tạo điều kiện thoát khỏi tối ưu cục bộ. Hơn nữa, lai ghép độc nhất giữa các cá thể thuộc một cùng nền văn hóa có thể dẫn đến mất đi yếu tố di truyền tốt và đa dạng có sẵn từ các nền văn hóa khác. Do đó *rmp*

phải được chọn để đảm bảo cân bằng tốt giữa quét kỹ các vùng nhỏ trong không gian tìm kiếm và khám phá toàn bộ không gian.

Cuối cùng, trong khi chọn các toán tử chéo và đột biến, phải nhớ rằng các khoá ngẫu nhiên được trình bày, trước đó luôn được hiểu là các biến liên tục, ngay cả khi vấn đề tối ưu hóa cơ bản là rời rạc. Điều này khuyến khích việc sử dụng các toán tử di truyền được mã hóa hiện có hoặc thiết kế các toán tử mới để điều hướng cải thiện cảnh quan tổng hợp liên quan tới MFO.

2.5 Đánh giá có chọn lọc

Trong khi đánh giá một cá thể đơn lẻ cho một tác vụ T_j, bước đầu tiên là giải mã các khóa ngẫu nhiên của nó thành một đầu vào có ý nghĩa cho tác vụ đó. Nói cách khác, biểu diễn khóa ngẫu nhiên đóng vai trò là không gian tìm kiếm thống nhất, từ đó có thể suy ra các biểu diễn giải pháp cụ thể cho vấn đề. Đối với trường hợp tối ưu hóa liên tục, điều này được hoàn thành một cách đơn giản. Ví dụ, hãy xem xét biến thứ i(xi) của vấn đề thực tế được giới hạn trong phạm vị $[L_i,U_i]$. Nếu khóa ngẫu nhiên tương ứng của một cá nhận có giá trị y_i thì ánh xạ của nó ở trong không gian tìm kiếm của vấn đề thực tế được đưa ra bởi$x_i = L_i + (U_i - L_i).y_i$. Ngược lại, đối với trường hợp tối ưu hóa rời rạc, sơ đồ giải mã nhiễm sắc thể thường phụ thuộc vào vấn đề.

Thuật toán 3: Di truyền văn hóa theo chiều dọc thông qua sự bắt chước có chọn lọc

Một cá thể 'c' hoặc có cá thể cha mẹ p_a và p_b hoặc một cha mẹ (p_a hay p_b)

1 **If** ('c' có hai cha mẹ) **then**

i) Sinh một số ngẫu nhiên *rand* giữa 0 và 1

ii) **If** (*rand* <0.5) **then**

'c' bắt chước p_a: cá thể con được đánh giá chỉ cho tác vụ có thứ tự là τ_a

iii) **else**

'c' bắt chước p_b: cá thể con được đánh giá chỉ cho tác vụ có thứ tự là τ_b

iv) **end if**

2 **Else**

i) 'c' bắt chước theo cá thể cha mẹ (đột biến): cá thể con được đánh giá chỉ cho tác vụ có thứ tự τ của cá thể cha mẹ đột biến tạo ra nó

3 **end if**

4 Factorial cost của cá thể c trên các tác vụ còn lại được đặt là ∞ (một số rất lớn)

Hình 2-4: Mô tả các bước tổng thể của EA (thuật toán 3)

Rõ ràng, chi phí tính toán cho việc đánh giá từng cá thể cho mọi tác vụ đang giải quyết thường quá đắt và do đó không được mong muốn. Để làm cho MFO tính toán thực tế, MFEA phải được thiết kế hiệu quả. Điều này đạt được ở đây bằng cách giảm tổng số các đánh giá càng nhiều càng tốt. Một quan sát quan trọng mà chúng tôi đưa ra là một cá thể được tạo ra trong MFEA có thể sẽ không đạt được hiệu năng cao cho tất cả các tác vụ. Do đó, một cá thể chỉ được đánh giá cho các tác vụ được chọn mà nó rất có thể thực hiện tốt. Để kết hợp tính năng này vào MFEA một cách đơn giản, thuật toán mượn ý tưởng của việc lan truyền "trào lưu văn hóa"[32]. Trong thừa kế đa nhân tố, lan truyền "trào lưu văn hoá" từ cha mẹ đến con cái (theo chiều dọc – từ trên xuống dưới) là một kiểu *(mode)* của sự kế thừa song song với sự thừa hưởng sinh học theo cách thức mà kiểu hình của một con cái bị ảnh hưởng trực tiếp bởi kiểu hình của cha mẹ [30].

Sự tương tự tính toán của hiện tượng nói trên được thực hiện trong MFEA bằng cách cho phép con cái bắt chước nhân tố kỹ năng (đặc điểm hay trào lưu văn hoá) của bất kỳ người nào là cha mẹ. Tính năng này, được gọi là *bắt chước có chọn lọc (selective imitation)*, đã đạt bằng cách làm theo các bước trong Thuật toán 3 đó

là: thay vì đánh giá cá thể con cho mọi tác vụ, nó chỉ được đánh giá cho một tác vụ. Điều đáng chú ý là việc kết hợp các hiệu hứng văn hóa theo cách thức quy định làmgiamr đáng kể tổng số đánh giá chức năng cần thiết. Trong thực tế, đối với bài toán K-factorial, các đánh giá chức năng bị giảm gần như theo hệ số K so với trường hợp sẽ xảy ra nếu một cá thể được đánh giá cho tất cả các tác vụ.

2.6 Sự lựa chọn

Như được thể hiện trong Thuật toán 1, MFEA tuân theo chiến lược tinh hoa, đảm bảo rằng những cá thể tốt nhất sẽ tồn tại qua các thế hệ. Để xác định các cá nhân tốt nhất, MFEA cũng lựa chọn các cá thể tốt hơn cho thế hệ kế tiếp, tức là các cá thể có giá trị ϕ (*scalar fitness*) lớn hơn.

Tóm tắt các tính năng nổi bật của MFEA

Các EA tiêu chuẩn thường tạo ra lượng lớn quần thể của lời giải đề cử, tất cả gần như không đủ sức cho tác vụ trong tầm tay. Ngược lại trong một môi trường đa nhiệm, khả năng cao một cá thể được tạo ngẫu nhiên hoặc biến đổi có khả năng đủ sức hoàn thành ít nhất một tác vụ. Cơ chế của MFEA xây dựng dựa trên quan sát bằng cách phân chia quần thể thành các nhóm kỹ năng khác nhau, mỗi nhóm xuất sắc trong một tác vụ khác nhau. Thú vị hơn và quan trong hơn là, có thể có vật liệu di truyền được tạo ra trong một nhóm hóa ra cũng hữ ích cho một tác vụ khác. Do đó, trong các tình huống như vậy, phạm vi chuyển gen qua các tác vụ có thể có khả năng dẫn đến việc không khó phát hiện ra tối ưu hóa toàn cục. Trong MFEA, việc chuyển giao vật liệu di truyền được tạo điều kiện bằng cách cho phép các nhóm kỹ năng khác nhau giao tiếp với nhau một cách có kiểm soát, thông qua thỉnh thoảng trao đổi nhiễm sắc thể. Điều này đạt được hoàn toàn nhờ hai thành phần của thuật toán hoạt động trong mô hình, đó là:

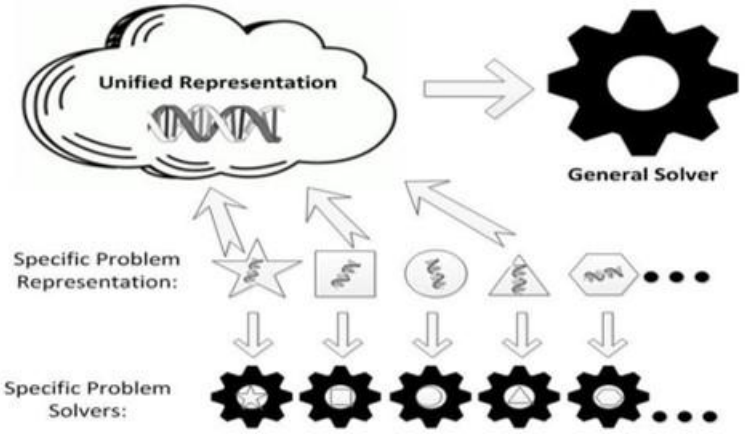
a) rmp, cho phép các cá thể có các yếu tố kỹ năng riêng biệt kết hợp với một số xác suất, từ đó tạo ra một môi trường đa văn hóa cho con cháu được nuôi dưỡng.

b) Thực tế là cá thể con được tạo ra sau đó có thể chọn ngẫu nhiên một yếu tố ngẫu nhiên của cha mẹ để bắt chước (Xem thuật toán 3). Mặc dù giao tiếp quá mức có thể gây gián đoạn cho việc tìm kiếm tập trung, việc cấm giao tiếp cũng là

điều không mong muốn vì nó sẽ hạn chế sự khám phá và sức mạnh của sự song song ẩn tàng tạo ra bởi toàn bộ quần thể.

Chương 3 ÁP DỤNG THUẬT TOÁN TIẾN HÓA ĐA NHÂN TỐ ĐỂ GIẢI CÁC BÀI TOÁN TỐI ƯU ĐƠN MỤC TIÊU

Như mục tiêu ban đầu, đề tài sẽ không quá đi chi tiết vào thực nghiệm của việc áp dụng thuật toán trên một bài toán cụ thể nào, thay vào đó, sẽ cố gắng giải thích chi tiết các khái niệm, các bước thực hiện của thuật toán. Thêm vào đó, vấn đề quan trọng nhất cho người muốn thiết kế các thuật toán EA là làm sao biểu diễn mỗi giải pháp của bài toán thực tế thành một cá thể trong thuật toán EA và ngược lại. Vì vậy, phần này tập trung vào việc giới thiệu cách biểu diễn (mã hóa) các giải pháp của các bài toán cụ thể vào cùng "không gian tìm kiếm hợp nhất" và ngược lại (giải mã). Ý tưởng mã hóa và giải mã của thuật toán MFEA có thể minh họa như hình 5. Theo đó, trong với EA truyền thống, mỗi tác vụ có một sự biểu diễn cụ thể (Specific Problem Representation), từ đó tìm ra các lời giải cụ thể (Specific Problem Solvers). Khi chuyển sang MFEA, tất cả các sự biểu diễn của các tác vụ được được chuyển thành một biểu diễn hợp nhất (Unified Representation), và từ đó tìm ra lời giải tổng quát (General Solver).



Hình 3-1: Sự biểu diễn của các tác vụ trên các không gian tìm kiếm khác nhau được chuyển về không gian tìm kiếm hợp nhất

Thông thường, một vấn đề tổ hợp có thể được xác đinh bởi trạng thái hữu hạn $S = \{1,2,3,...,n\}$, dẫn đến một không gian tìm kiếm riêng biệt bao gồm một tập hợp các lời giải khả khi $X \subseteq 2^S$, trong đó mỗi lời giải ánh xạ đến một giá trị hàm

thực, như $f: 2^S \rightarrow \mathbb{R}$. Mô tả có vẻ đơn giản này có thể dẫn đến một loạt các phát biểu đa dạng. Để làm rõ sự mã hóa và giải mã các giải pháp của các bài toán tối ưu khác nhau vào cùng một không gian tìm kiếm hợp nhất, chúng tôi sẽ sử dụng hai bài toán tối ưu là *Knapsack Problem* (KP) và *Quadratic Assignment Problem* (QAP) [31].

3.1 Bài toán Knapsack và bài toán Quadratic Assignment Problem

3.1.1 Bài toán Knapsack

Đầu vào:

- n: số lượng đồ vật
- W: trọng lượng tối đa của ba lô
- c = {c1,c2,...,cn}: véc tơ lưu trọng lượng của các vật
- V = {v1,v2,...,vn}: véc tơ lưu giá trị của các vật

Đầu ra:

- Danh sách các vật được chọn cho vào ba lô Mục tiêu:
- Tổng trọng lượng thu được của ba lô là lớn nhất

Ràng buộc

- Tổng trọng lượng của các vật được chọn không vượt quá trọng lượng của ba lô

3.1.2 Bài toán Quadratic Assignment

Bài toán QAP [31] được phát biểu như sau: QAP có thể phát biểu như sau: Đưa ra *n* phương tiện và n vị trí với khoảng cách giữa các vị trí là *d*(*i*,*j*) và yêu cầu luồng (flow) từ vị trí i đến vị trí j là *f*(*i*,*j*).

Đầu vào:

- n: số lượng vị trí và số lượng cơ sở định đặt tại các vị trí
- d(n×n): ma trận khoảng cách dij khoảng cách giữa vị trí i và vị trí j.
- f(n×n): là luồng yêu cầu giữa cơ sở i và vị trí j

Đầu ra:

- Cách bố trí các cơ sở vào các vị trí. Hay nói cách khác đầu ra là một hoán vị của tập n phần tử $(\varphi_1,\varphi_2,...,\varphi_n)$. Trong đó φ_i cho biết cơ sở nào được đặt tại vị trí i.

Mục tiêu:

- Tổng chi phí là rõ nhất, nghĩa là $\sum_{i=1}^{n}\sum_{j=1}^{n} f(i,j).d(\varphi(i),\varphi(j)) \rightarrow min$.

3.2 Áp dụng thuật toán tiến hóa đa nhân tố để giải đồng thời hai bài toán Knapsack và bài toán Quadratic Assignment Problem

Trong các thuật toán EA truyền thống cho bài toán KP, mỗi một cá thể được biểu diễn bằng một vectơ $x = \{x_1,x_2,...x_n\}$ của các biến nhị phân, trong đó $x_i = 1$ chỉ ra rằng đồ vật thứ ith được chọn để cho vào ba lô, ngược lại, không cho đồ vật thứ ith vào ba lô. Một cách đơn giản để suy luận các biến nhị phân từ các khóa ngẫu nhiên là $x_i = 1$ nếu khóa ngẫu nhiên $y_i \geq 0.5$, ngược lại $x_i = 0$.

Tuy nhiên, kỹ thuật như vậy thường dẫn đến hiệu suất kém cho MFEA. Thay vào đó đề tài sẽ xem xét từng cá thể, được xác định bởi một vector của bộ khóa ngẫu nhiên $(y_1,y_2,...,y_n)$, để biểu diễn một tập hợp n gen có thể di chuyển tự do trên một đoạn của dòng thực, ví dụ: giữa 0 và 1. Mỗi gen tương ứng với một đồ vật độc nhất. Tính tự do di chuyển dành cho các gen ngụ ý rằng vị trí của chúng có thể được điều khiển thông qua các toán tử được mã hóa thực, như mong muốn đối với MFEA.

Các ngẫu khóa có thể được giải mã cho bài toán QAP theo cách đơn giản, như đã được đề xuất trong tài liệu [35]. Lưu ý rằng ngẫu khóa thứ i của một cá thể được xem như một nhãn được gắn thẻ cho phương tiện thứ i. Để gán phương tiện cho các vị trí, các khóa ngẫu nhiễn được sắp xếp đơn giản theo thứ tự tăng dần. Sau đó các phương tiện được chỉ định cho các vị trí theo đúng vị trí của chúng trong bảng sắp xếp. Ví dụ: Xem xét 3 vị trí, được gắn nhãn {1,2,3} và một bộ 3 phương tiện đại diện bởi các ngẫu khóa: (0.7,0.1,0.3). Theo kỹ thuật ấn định mẫu, phương tiện 2 được ấn định với vị trí 1, phương tiện 3 ấn định cho vị trí 2, và phương tiện 1

cho vị trí 3. Liên quan đến tìm kiếm địa phương, trao đổi 2 chiều heuristic đã được sử dụng để cái thiện tác vụ cả các đối tượng.

Trong các thuật toán EA truyền thống cho bài toán QAP, mỗi một cá thể được biểu diễn bằng một vectơ $x = \{x_1,x_2,...x_n\}$ trong đó $x_i \in \{1,2,...,n\}$ và $x_i 6= x_j vii 6= j$; nghĩa là phương tiện thứ i^{th} được ấn định đến vị trí thứ x_i. Một cách đơn giản để suy luận các biến xi từ các khóa ngẫu nhiên $y = \{y_1,y_2,...y_n\}$ đó là xi là số thứ tự (index) của y_i trong dãy $y_1,y_2,...y_n$ được sắp xếp theo thứ tự tăng dần.

Ví dụ số

Để tóm tắt các quy trình giải mã tên miền chéo được mô tả ở đây, ta sẽ tập trung vào một ví dụ minh họa trong đó có ba nhiệm vụ tối ưu hóa riêng biệt được thực hiện đồng thời Giả sử $n = 9$ trong bài toán KP và $n = 6$ trong bài toán QAP.

Như vậy, $D_{multitask} = max\{9,6\} = 9$. Điều này có nghĩa là, trong không gian hợp nhất, *chromosome* sẽ gồm 9 khóa ngẫu nhiên. Giả sử giá trị của *chromosome* trong không gian hợp nhất như sau:

Biểu diễn của *Chromosome* y = (0.79, 0.31, 0.53, 0.17, 0.60, 0.26, 0.65, 0.69, 0.75)

Stt	1	2	3	4	5	6	7	8	9
y	0.79	0.31	0.53	0.17	0.60	0.26	0.65	0.69	0.75

Dựa vào phân tích ở trên, ta dễ dàng tính được vectơ biểu diễn cá thể tương ứng cho bài toán KP (độ dài 9) và QAP (độ dài 6) như sau:

Cá thể *p* cho bài toán KP: *pKP* = (1,0,1,0,1,0,1,1,1)

Cá thể *p* cho bài toán QAP: *pQAP* = (6,3,4,1,5,2)

Đối với bài toán KP, các ngẫu khóa đã được phát họa thành một hàng, cùng với đó là chỉ số của gen tương ứng. Thuật toán liên kết đơn được sử dụng để phân chia các gen thành hai cụm, với cụm đầu tiên bao gồm các gen 4,6,2 và cụm thứ hai bao gồm các gen 3,5,7,8,9,1. Theo quy trình giải mã được đề xuất cho KP, chúng ta có thể coi các mục 4,6,2 được bao gồm tạm thời trong Knapsac, trong khi tất cả các

mục trong các cụm khác điều bị loại trừ.. Nói các khác, biểu diễn nhiễm sắc thể đặc trưng cho miền của KP hoạt động theo (0 1 0 1 0 1 0 0 0). Do đó, sau đây chúng tôi đề cập đến thủ tục giải mã là phân cụm nhị phân. Cuối cùng đối với QAP, chuỗi các chỉ số gen quy định các phương tiện được gán cho các vị trí. Đôi với y đã cho, đại điện theo miền cụ thể tương đương với (4,6,2,3,5,7,8,9,1). Một tính năng thú vị của sơ đồ biểu diễn khóa ngẫu nhiên, đặc biệt là trong trường hợp các vấn đề giải trình tự như bài toán QAP, nó đảm bảo tính khả thi của con cái trong quá trình hoạt động đột biến và đột biến được mã hóa thực.

3.3 Kết quả mô phỏng

Thuật toán được cài đặt bằng ngôn ngữ lập trình Matlab. Thông số máy tính sử dụng: hệ điều hành Ubuntu, CORE I5, RAM 4GB.

3.3.1 Dữ liệu

- Bài toán QAP: Dữ liệu được khởi tạo ngẫu nhiên 6 bộ dữ liệu với ma trận khoảng cách và ma trận giá trị. Các bộ dữ liệu có kích thước số vị trí lần lượt là 4, 5, 6, 7, 8, 9.
- Bài toán KP: Dữ liệu tạo ngẫu nhiên 6 bộ dữ liệu với véc tơ trọng số và véc tơ giá trị. Các bộ có kích thước là 4, 5, 6, 7, 8, 9

Các bài toán sẽ giải kết hợp:

Bảng 3-1: Kích thước các bài toán sẽ kết hợp giải

Test Bài toán KP	Bài toán QAP	
Test 1	9	4
Test 2	8	5
Test 3	7	6
Test 4	6	7
Test 5	5	8

Test 6	4	9

3.3.2 *Tham số thực nghiệm*

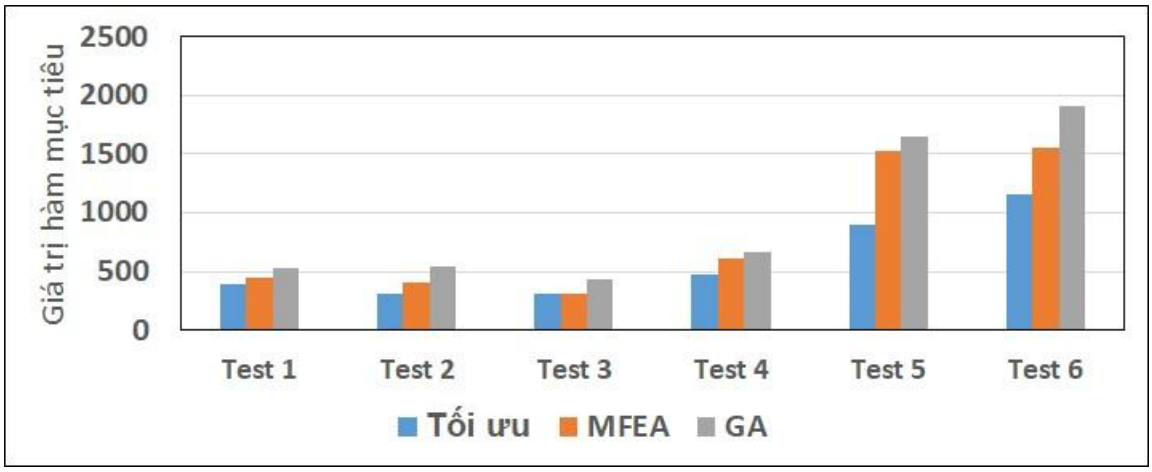
Bảng 3-2: Tham số thực nghiệm

Kích thước quần thể	100
Xác suất lai ghép	0.8
Xác suất đột biến	0.1
Số thế hệ	100

3.3.3 *Kết quả thực nghiệm*

Luận văn sẽ so sánh kết quả đạt được theo các tiêu chí sau:

- Đánh giá kết quả của thuật toán tối ưu đa nhân tố so với giải thuật di truyền trên bài toán QAP và KP



Hình 3-2: So sánh kết quả cảu MFGA với GA và lời giải tối ưu trên bài toán QAP

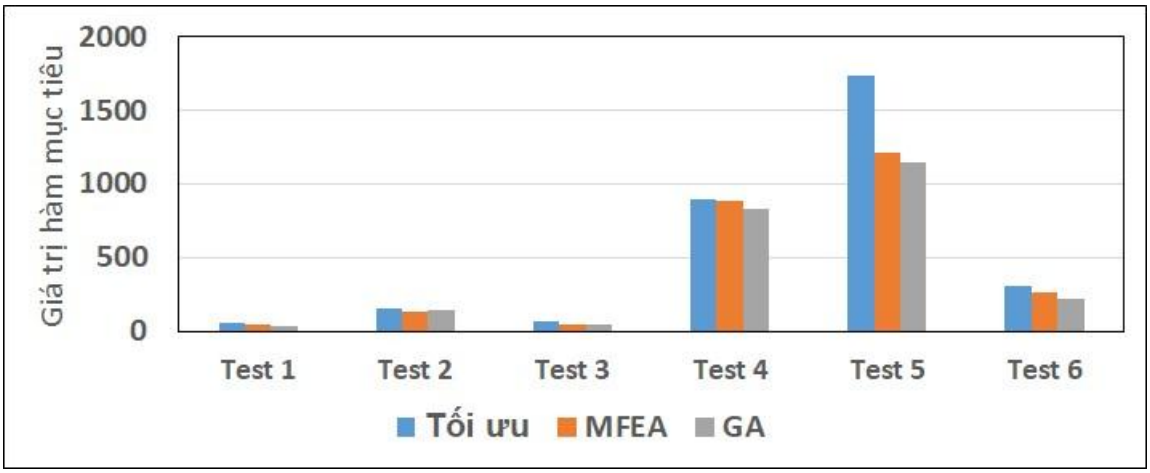
- Đánh giá kết quả của giải thuật tối ưu đa nhân tố và lời giải tối ưu (luận án tìm giải thuật tối ưu sử dụng thuật toán vét cạn) trên bài toán QAP và KP

Hiệu quả của MFEA trên bài toán QAP

Nhìn vào hình 3.2 chúng ta thấy rằng, MFEA cho kết quả tốt hơn GA trên hầu hết các bộ dữ liệu. Kết quả cho thấy rằng, MFEA tốt hơn GA trung bình khoảng 21%. Lời giải của MFGA khá sát với tối ưu. Trên test 3, MFEA cho kết quả bằng với tối ưu

Hiệu quả của MFEA trên bài toán KP

Nhìn vào hình 3.3 chúng ta thấy rằng, MFEA cho kết quả tốt hơn GA trên hầu hết các bộ dữ liệu. Kết quả cho thấy rằng, MFEA tốt hơn GA trung bình khoảng 5%. Lời giải của MFGA khá sát với tối ưu (bằng 92% tối ưu).



Hình 3-3: So sánh kết quả của MFGA với GA và lời giải tối ưu trên bài toán KP

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Kết quả đạt được

Bài toán tối ưu tổ hợp là bài toán khó, đã được nhiều các tác giả nghiên cứu trong thời gian gần đây. Luận văn tập trung vào việc tìm thiểu thuật toán tiến hóa đa nhân tố để giải quyết bài toán tối ưu đơn mục tiêu. Những kết quả đạt được của luận văn như sau:

- Thứ nhất: Luận văn trình bày được những kiến thức tổng quan về bài toán tôi ưu, các cách giải bài toán ối ưu
- Thứ hai: Luận văn đã tìm hiểu sơ đồ chung của thuật toán tiến hóa đa nhân tố để giải bài toán tối ưu hóa đơn mục tiêu.
- Thứ ba: Luận văn tìm hiểu cách áp dụng thuật toán tiến hóa đa nhân tố để giải quyết đồng thời hai bài toán là KP và QAP. Kết quả mô phỏng trên các bộ dữ liệu cho thấy thuật toán hoạt động tốt, cho kết quả sát với tối ưu

Vấn đề tồn đọng và hướng phát triển

- Thực nghiệm trên các bộ dữ liệu lớn hơn để đưa ra được những đánh giá khách quan.
- Tìm hiểu thuật toán tiến hóa đa nhân tố giải quyết các bài toán đơn mục tiêu trong thực tế.
- Tìm hiểu để thuật toán tiến hóa đa nhân tố để giải quyết các bài toán tối ưu hóa đa mục tiêu.

TÀI LIỆU THAM KHẢO

[1] Nguyễn Đức Nghĩa và Nguyễn Tô Thành. Toán rời rạc. Đại học Quốc gia Hà Nội, 2007.

[2] Trương Phước Nhân, Định lý Hall – Định lý Konig, 24/07/2017

[3] Trương Phước Nhân, Định lý Konig – Egervary, 05/08/2017

[4] Nguyễn Gia Như, Lê Đắc Nhường, Lê Vinh Trọng (2013), "A Novel PSO based Algorithm Approach for the cMTS to Improve QoS in Next Generation Networks", Journal of Future Computer and Communication (IJFCC), Singapore, Vol.2(5), 413-417.

[5] Nguyễn Gia Như, Lê Đắc Nhường, Lê Vinh Trọng (2012), " A Novel PSO-Based Algorithm for the Optimal Location of Controllers in Wireless Networks", International Journal of Computer Science and Network Security, VOL.12(8), August 2012, pp.23-27.

[6] Nguyễn Đình Thúc, Lập trình tiến hóa, Nhà xuất bản Giáo Dục, 2009.

[7] Ian Anderson, Ashort Course in Combinatorial Designs, 2012

[8] T. Back, U. Hammel, and H. P. Schwefel, "Evolutionary computation: Comments on the history and current state," IEEE Trans. Evo. Comp., vol. 1, no. 1, pp. 3-17, 1997.

[9] J. C. Bean, "Genetic algorithms and random keys for sequencing and optimization," ORSA J. Comp., vol. 6, no. 2, 1994.

[10] C. Blum (2002), "ACO applied to group shop scheduling: A case study on intensification and diversification", Proc. of ANTS 2002, Third International Workshop on ant algorithms, Vol 2463, pp. 14-27.

[11] C. Blum and M. Dorigo (2004), "The Hyper-Cube Framework for Ant Colony Optimization", IEEE Transactions on Systems, Man, and Cybernetics – Part B, Vol 34 (2), pp. 1161-1172.

[12] James Blondin , *Particle Swarm Optimization: A Tutorial*.

[13] C. A. Coello Coello, “Evolutionary multi-objective optimization: a historical view of the field,” IEEE Comp. Intel. Mag., vol. 1, no. 1, pp. 28-36, Feb. 2006.

[14] R. Dawkins et al, The selfish gene. Oxford university press, 2016.

[15] W. J. Gutjahr (2002), “ACO algorithms with guaranteed convergence to the optimal solution”, Info. Proc. Lett., Vol 83 (3), pp. 145-153.

[16] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. Econometrica: Journal of the Econometric Society, pages 497–520, 1960.

[17] J. Kennedy and R. Eberhart, "Particle swarm optimization," in Neural Networks, 1995.

[18] F. Neumann, D. Sudholt and CarstenWitt (2008), “Rigorous Analyses for the Combination of Ant Colony Optimization and Local Search”, Proceedings of the Sixth International Conference on Ant Colony Optimization and Swarm Intelligence, pp. 132-143.

[19] P. Pellegrini and A. Ellero (2008), “The Small World of Pheromone Trails”, Proc. of the 6th international conference on Ant Colony Optimization and Swarm Intelligence, pp. 387-394.

[20] C. Reeves (1995), Genetic Algorithms and Combinatorial Optimisation: Applications of Modern Heuristic Techniques, In V.J. Rayward-Smith, Alfred Waller Ltd, Henley-on-Thames, UK.

[21] J. Rice, C. R. Cloninger, and T. Reich, “Multifactorial inheritance with cultural transmission and assortative mating. I. Description and basic properties of the unitary models,” Am. J. Hum. Genet. vol. 30, pp. 618-643, 1978.

[22] M. Sampels, K. Socha and M. Manfrin (2002), “A MAX-MIN Ant System for the University Timetabling Problem”, Proc. of the 3rd International Workshop on Ant Algorithms, pp. 1-13.

[23] M. Sampels, K. Socha and M. Manfrin (2003). “Ant Algorithms for the Univerrsity Course Timetabling Problem with Regard to the State-of-the-Art”, Applications of Evolutionary Computing, Proceedings of the EvoWorkshops 2003, 334-345.

[24] A. H. Wright, M. D. Vose, and J. E. Rowe, “Implicit Parallelism,” Lecture Notes in Comp. Sci., vol. 2724, pp. 15051517, 2003.

[25] S. Benedettini, A. Roli, and L. Gaspero (2008), “Two-level ACO for haplotype inference under pure parsimony”, Proc. of the 6th international conference on Ant Colony Optimization and Swarm Intelligence, pp. 179-190.

[26] L. L. Cavalli-Sforza and M. W. Feldman, “Cultural vs Biological inheritance: Phenotypic transmission from parents to children (The effect of parental phenotypes on children’s phenotypes),” Am. J. Hum. Genet, vol. 25, 618-637, 1973.

[27] M. Dorigo (1992), Optimization, learning and natural algorithms, PhD. disserta-tion, Milan Polytechnique, Italy.

[28] M. Dorigo, V. Maniezzo and A. Colorni (1991), The Ant System: An autocatalytic optimizing process, Technical Report 91-016 Revised, Dipartimento di Elettronica, Politecnico di Milano, Milano, Italy.

[29] M. Dorigo, and T. Stu¨tzle (2004), Ant Colony Optimization, The MIT Press, Cambridge, Masachusetts.

[30] M. Dorigo, and T. Stu¨tzle (2004), Ant Colony Optimization, The MIT Press, Cambridge, Masachusetts.

[31] X. Chen, M. H. Lim, and Y.-S. Ong , “Research frontier-memetic computationpast, present & future,” IEEE Computational Intelligence Magazine, vol. 5, no. 2, p. 24, 2010.

[32] X. Chen, Y. S. Ong, M. H. Lim, and K. C. Tan, “A multi-facet survey on memetic computation,” IEEE Trans. Evo. Comp., vol. 15, no. 5, pp. 591-606, Oct. 2011.

[33] M. H. Tayarani-N and A. Prugel-Bennett, “On the landscape of combinatorial optimization problems,” IEEE Trans. Evo. Comp., vol. 18, no. 3, pp. 420-434, 2013.

[34] Proceedings., IEEE International Conference on, 1995