BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

BÁO CÁO TỔNG KẾT
ĐỀ TÀI NGHIÊN CỨU KHOA HỌC CỦA SINH VIÊN

NGHIÊN CỨU GIẢI THUẬT TIẾN HÓA ĐA NHIỆM VỤ TRONG GIẢI QUYẾT BÀI TOÁN TỐI ƯU

Sinh viên thực hiện:

1. Lê Văn Nghiêm	Lớp: DHCNTT04 – K13
2. Hoàng Thị Sao Mai	Lớp: DHCNTT06 – K15
3. Tạ Văn Toàn	Lớp: DHCNTT04 – K13
4. Nguyễn Quang Linh	Lớp: DHCNTT06 – K15
5. Nguyễn Hoàng Minh	Lớp: DHCNTT06 – K15

Khoa: Công nghệ thông tin

Người hướng dẫn: **TS. Nguyễn Thị Mỹ Bình**

Hà Nội, 05/2022

BỘ CÔNG THƯƠNG
TRƯỜNG ĐẠI HỌC CÔNG NGHIỆP HÀ NỘI

BÁO CÁO TỔNG KẾT
ĐỀ TÀI NGHIÊN CỨU KHOA HỌC CỦA SINH VIÊN

NGHIÊN CỨU GIẢI THUẬT TIẾN HÓA ĐA NHIỆM VỤ TRONG GIẢI QUYẾT BÀI TOÁN TỐI ƯU

Sinh viên thực hiện:

	1. Lê Văn Nghiêm	Giới tính:	Nam
	2. Hoàng Thị Sao Mai		Nữ
	3. Tạ Văn Toàn		Nam
	4. Nguyễn Quang Linh		Nam
	5. Nguyễn Hoàng Minh		Nam
Lớp:	DHCNTT04 – K13	Năm thứ:	4/4
	DHCNTT06 – K15		2/4
	DHCNTT04 – K13		4/4
	DHCNTT06 – K15		2/4
	DHCNTT06 – K15		2/4

Dân tộc: Kinh

Khoa: Công nghệ thông tin

Người hướng dẫn: **TS. Nguyễn Thị Mỹ Bình**

Hà Nội, 05/2022

LỜI CẢM ƠN

Đầu tiên, chúng em xin gửi lời cảm ơn chân thành và sâu sắc tới giáo viên hướng dẫn là **TS. Nguyễn Thị Mỹ Bình** đã tận tình chỉ bảo, hướng dẫn và đồng thời mang đến cho chúng em những tài liệu, kiến thức quý giá để giúp chúng em có thể tìm hiểu, trau dồi, nâng cao kiến thức và hoàn thành đề tài nghiên cứu này.

Chúng em xin gửi lời cảm ơn đến Trường Đại học Công nghiệp Hà Nội cũng như khoa Công nghệ thông tin đã tạo điều kiện cho chúng em được tham gia làm nghiên cứu khoa học. Chúng em đã có điều kiện tìm hiểu với các nội dung mới lạ đòi hỏi nhiều kiến thức cao hơn. Bên cạnh việc tạo ra cơ hội cho chúng em tiếp xúc các vấn đề mới thì Nhà trường cùng khoa cũng tạo điều kiện cho chúng em được nghiên cứu tại phòng Lab Optimization, Modeling and Simulation. Phòng Lab với điều kiện vật chất hiện đại, tiện nghi đã giúp chúng em có một tinh thần thoải mái khi tiếp xúc và nghiên cứu một đề tài mới và khó.

Chúng em xin chân thành cảm ơn!

MỤC LỤC

DANH MỤC THUẬT NGỮ VÀ CÁC TỪ VIẾT TẮT VÀ KÝ HIỆU

Các từ viết tắt, ký hiệu

STT	Các từ viết tắt, ký hiệu	Dịch nghĩa
1	TSP	Traveling Salesman Problem
2	GA	Genetic Algorithm
3	MFO	Multifactorial Optimization
4	MFEA	Multifactorial Evolution Algorithm
5	EA	Evolution Algorithm
6	SOO	Single-Objective Optimization
7	MOO	Multi-Objective Optimization

Các thuật ngữ

STT	Các thuật ngữ	Dịch nghĩa
1	Crossover	Lai ghép
2	Mutation	Đột biến
3	Selection	Chọn lọc
4	Task	Tác vụ
5	Offspring	Thế hệ con cái

DANH MỤC BẢNG BIỂU, BIỂU ĐỒ

DANH MỤC HÌNH ẢNH

MỞ ĐẦU

Lý do chọn đề tài

Bài toán người du lịch là một trong số các bài toán được ứng dụng rộng rãi trong nhiều lĩnh vực thực tế của đời sống như tối ưu hệ thống giao thông, tối ưu mạng cảm biến, …Bài toán người du lịch TSP là một bài toán thuộc lớp NP-Khó đã được nghiên cứu và sử dụng rất nhiều phương pháp khác nhau để giải quyết như phương pháp xấp xỉ, các thuật toán heuristic để giải quyết.

Những phương pháp được lựa chọn này đều xử lý một bài toán trong một lần thực hiện. Tuy nhiên, hiện nay, khoa học máy tính phát triển và yêu cầu về việc nhiều bài toán tối ưu cần được giải quyết cùng một lúc. Vì vậy, nhận thấy vấn đề cấp thiết và có vai trò quan trọng em đã lựa chọn đề tài "*Nghiên cứu giải thuật tiến hóa đa nhiệm vụ trong giải quyết bài toán tối ưu*" nhằm nghiên cứu, tìm hiểu về giải thuật tiến hóa đa nhiệm vụ - một cách phổ biến để giải quyết vấn đề thực hiện nhiều bài toán tối ưu cùng một lúc.

Mục tiêu của đề tài

Nghiên cứu, áp dụng và đánh giá giải thuật tiến hóa đa nhiệm vụ trong giải quyết bài toán tối ưu, cụ thể xét bài toán Người du lịch.

Đề tài tập trung cụ thể vào:

- Tìm hiểu về bài toán người du lịch TSP.
- Nghiên cứu tổng quan về giải thuật tiến hóa, giải thuật di truyền.
- Nghiên cứu giải thuật tiến hóa đa nhiệm vụ MFEA.
- Áp dụng giải thuật tiến hóa đa nhiệm vụ MFEA cho bài toán tìm đường đi TSP.
- Nghiên cứu ngôn ngữ Java, cài đặt và đánh giá hiệu quả của thuật toán tiến hóa đa nhiệm vụ MFEA trong giải quyết bài toán Người du lịch TSP.

Đối tượng nghiên cứu

- Nghiên cứu các thuật toán của lý thuyết độ thị giải bài toán TSP đã công bố ở trong và ngoài nước.
- Nghiên cứu các thuật giải di truyền, giải thuật tiến hóa đa nhiệm vụ MFEA.
- Nghiên cứu áp dụng giải thuật tiến hóa đa nhiệm vụ MFEA cho bài toán TSP.
- Nghiên cứu tài liệu về ngôn ngữ Java để cài đặt các thuật toán thử nghiệm.

- Cài đặt và đánh giá hiệu quả của thuật toán tiến hóa đa nhiệm vụ cho bài toán TSP.

Phạm vi nghiên cứu

Bài toán Người du lịch TSP là một bài toán phổ biến khi tìm hiểu về dạng bài toán tối ưu. Do vậy, hiện nay có nhiều phương pháp giải bài toán TSP như sử dụng quy hoạch tuyến tính, thuật toán vét cạn, kỹ thuật nhánh và cận, thuật toán tối ưu hóa đàn kiến và các thuật toán heuristic. Tuy nhiên, trong đề tài này, em chỉ đi sâu tập trung nghiên cứu hướng tiếp cận bằng giải thuật tiến hóa đa nhiệm vụ MFEA cho bài toán Người du lịch TSP và đánh giá hiệu quả của thuật toán.

Nội dung nghiên cứu

- Nghiên cứu tổng quan các nghiên cứu giải bài toán TSP và các hướng tiếp cận giải quyết bài toán này.
- Nghiên cứu tổng quan về giải thuật tiến hóa, di truyền.
- Nghiên cứu thuật toán tiến hóa đa nhiệm vụ MFEA.
- Nghiên cứu áp dụng thuật toán tiến hóa đa nhiệm vụ MFEA cho bài toán TSP.
- Nghiên cứu ngôn ngữ Java để cài đặt thuật toán.
- Lập trình và đánh giá hiệu quả của thuật toán tiến hóa đa nhiệm vụ cho bài toán TSP.
- Viết báo cáo nghiên cứu.

Cấu trúc của bài báo cáo

Cấu trúc các Chương tiếp theo gồm:

- *Chương 2* trình bày các cơ sở lý thuyết về bài toán tối ưu, chi tiết nội dung bài toán Người du lịch gồm: phát biểu bài toán, lịch sử bài toán, phân tích độ phức tạp, lý do sử dụng giải thuật tiến hóa đa nhiệm vụ MFEA để giải quyết bài toán. Đồng thời chương này sẽ trình bày về giải thuật tiến hóa, giải thuật di truyền, giải thuật tiến hóa đa nhiệm vụ MFEA gồm: lịch sử giải thuật, phát biểu giải thuật, các khái niệm và cấu trúc thuật toán MFEA.
- *Chương 3* trình bày về áp dụng giải thuật tiến hóa đa nhiệm vụ để giải bài toán người du lịch, kết quả chạy chương trình với các bộ cơ sở dữ liệu khác nhau.
- Cuối cùng em sẽ đưa ra kết luận và đề cập hướng phát triển của đề tài.

CHƯƠNG 1. CƠ SỞ LÝ THUYẾT

1.1 Bài toán tối ưu

1.1.1 Tổng quan về bài toán tối ưu

1.1.1.1 Lịch sử bài toán tối ưu

Vào thế kỷ XVIII, một hướng nghiên cứu bài toán cực trị hàm mục tiêu xuất hiện là phiếm hàm tích phân và được gọi là Phép tính biến phân.

Những năm 30-40 của thế kỷ XX xuất hiện Lý thuyết Quy hoạch tuyến tính.

Những năm 50- thế kỷ XX xuất hiện Quy hoạch lồi.

Từ những những năm 70 của thế kỷ XX hình thành nhiều hướng nghiên cứu khác nhau như Tối ưu không lồi, tối ưu phi tuyến, tối ưu rời rạc, tối ưu tổ hợp và tối ưu đa mục tiêu.

Từ những năm 50-60 của thế kỷ XX xuất hiện Lý thuyết điều khiển được và điều khiển tối ưu.

1.1.1.2 Định nghĩa

Trong thực tế có nhiều tình huống với nhiều phương án giải quyết khác nhau khiến chúng ta phải lựa chọn để tìm ra phương án tốt nhất. Quan điểm "tốt nhất" này phụ thuộc vào từng tình huống và từng mục đích, hoặc tối đa (maximize) hoặc tối thiểu (minimize) một tiêu chí nào đó. Ví dụ như khi có nhiều cách để đi từ nhà đến nơi làm việc, có người chọn cách đi ngắn nhất, có người lại chọn cách đi ít tắc đường nhất hay gặp ít đèn tín hiệu giao thông nhất.

Việc tìm phương án tốt nhất trong số những phương án có thể như trên chính là việc giải quyết bài toán tối ưu (optimization problem) trong lĩnh vực toán và công nghệ thông tin. Nhìn chung, các bài toán tối ưu luôn có hai đặc điểm sau:

- Số lời giải nhiều hơn một. Tập các lời giải này được gọi là không gian lời giải hay không gian tìm kiếm.
- Mọi lời giải của một bài toán đều được định lượng độ "tốt" bởi cùng một hàm tính theo tiêu chí của bài toán đó.

Và để giải quyết những bài toán này (cũng như những bài toán khác), trước hết cần phải phát biểu chúng theo mô hình toán học (problem formulation). Trong đó liệt kê dữ liệu đầu vào, yêu cầu đầu ra, mục tiêu cụ thể bằng công thức toán và các ràng buộc (nếu có) một cách rõ ràng và logic. Từ đó, lời giải sẽ được biểu diễn

thành một hay nhiều biến quyết định (decision variable) sao cho ứng với mỗi giá trị khác nhau sẽ thu được lời giải không giống nhau. Dựa vào miền giá trị (rời rạc hay liên tục) của các biến này, bài toán tối ưu được chia thành hai loại:

- Bài toán tối ưu rời rạc, còn gọi là bài toán tối ưu tổ hợp (combinatorial optimization problem).
- Bài toán tối ưu liên tục (continuous optimization problem)

1.1.1.3 Phân loại bài toán tối ưu

a) Bài toán tối ưu tổ hợp

Bài toán tối ưu 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. Bất kỳ một bài toán tối ưu tổ hợp nào cũng có thể được định nghĩa hình thức bởi một bộ 4:

$$I = (U, P, f, extr)$$

Trong đó:

- U: không gian lời giải
- P: các ràng buộc
- f: hàm mục tiêu, là một ánh xạ U thuộc tập số thực $\boldsymbol{R}$
- $extr$: cực trị, thường là cực đại hoặc cực tiểu

Ở đây, không gian lời giải U là hữu hạn, rời rạc. Các lời giải trong không gian U nếu thỏa mãn các ràng buộc P được gọi là lời giải khả thi hay lời giải chấp nhận được.

Một số bài toán tiêu biểu thuộc lớp bài toán tối ưu tổ hợp này là:

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

b) Bài toán tối ưu liên tục

Khi các biến biểu diễn lời giải nhận giá trị liên tục trong không gian số thực thì bài toán tối ưu trở thành tối ưu liên tục. Mô hình chuẩn của bài toán tối ưu này như sau:

$$minimize\ maximize / f(x) \text{ sao cho:}$$
$$g_i(x) \leq 0, i = 1, \ldots, m$$
$$h_i(x) = 0, i = 1, \ldots, p$$

Trong đó:

- $f(x)$: $R^n \to R$ là hàm mục tiêu cần được tối thiểu (minimize) hoặc tối đa (maximize) với giá trị của biến x
- $g_i(x) \leq 0$: là các ràng buộc bất đẳng thức
- $h_i(x) = 0$: là các ràng buộc đẳng thức
- $m, p \in N$

Trường hợp $m = p = 0$ thì ta gọi đó là bài toán tối ưu (liên tục) không ràng buộc.

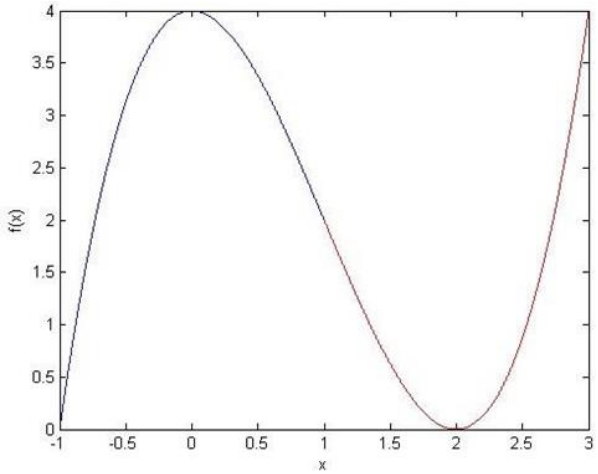
Tìm cực trị (cực đại hoặc cực tiểu) của hàm số là một ví dụ điển hình cho lớp bài toán này.

Chẳng hạn: minimize $f(x) = x^3 - 3x^2 + 4$

sao cho: $1 \leq x \leq 3, x \in R$

Đầu vào của bài toán chính là đoạn số thực $[1, 3]$ còn đầu ra là một giá trị thuộc đoạn đó sao cho hàm mục tiêu $f(x)$ có giá trị nhỏ nhất.

Rõ ràng, có rất nhiều giá trị x nằm trong đoạn $[1, 3]$ nhưng chỉ có duy nhất một giá trị $x = 2$ (xem Hình 1.1) làm hàm $f(x)$ tối thiểu. Khi đó ta nói $x = 2$ là lời giải tối ưu còn $f^* = f(x^*) = 0$ là giá trị tối ưu của bài toán.



Hình 1.1 Hình minh họa đồ thị hàm số $f(x) = x^3 - 3x^2 + 4$

Sau khi bài toán đã được mô hình hóa, vấn đề tiếp theo được đặt ra là giải quyết chúng như thế nào?

1.1.2 Bài toán người du lịch

1.1.2.1 Lịch sử bài toán

Bài toán người du lịch (Traveling Salesman Problem – TSP) là một bài toán NP-Hard thuộc thể loại tối ưu tổ hợp. Nội dung bài toán có thể hiểu khái quát như sau: Cho trước một danh sách các thành phố và khoảng cách giữa chúng, tìm chu trình ngắn nhất đi qua tất cả các thành phố đúng một lần.

Bài toán được nêu ra lần đầu vào năm 1930 và là một trong số những bài toán được nghiên cứu sâu nhất trong tối ưu hóa. Nó thường được dùng làm thước đo cho nhiều phương pháp tối ưu hóa.

Nguồn gốc của bài toán người du lịch vẫn chưa được biết rõ. Một cuốn sổ tay dành cho người du lịch xuất bản năm 1832 có đề cập đến bài toán này và có ví dụ cho chu trình trong nước Đức và Thụy Sĩ, nhưng không chứa bất kỳ nội dung toán học nào.

Bài toán người du lịch được định nghĩa trong thế kỉ 19 bởi nhà toán học Ireland William Rowan Hamilton và nhà toán học Anh Thomas Kirkman. Trò chơi Icosa của Hamilton là một trò chơi giải trí dựa trên việc tìm kiếm chu trình Hamilton. Trường hợp tổng quát của TSP có thể được nghiên cứu lần đầu tiên bởi các nhà toán học ở Vienna và Harvard trong những năm 1930, đặc biệt là Karl Menger, người đã định nghĩa bài toán, xem xét thuật toán hiển nhiên nhất cho bài toán, và phát hiện ra thuật toán láng giềng gần nhất là không tối ưu.

Hassler Whitney ở đại học Princeton đưa ra tên bài toán người du lịch ngay sau đó.

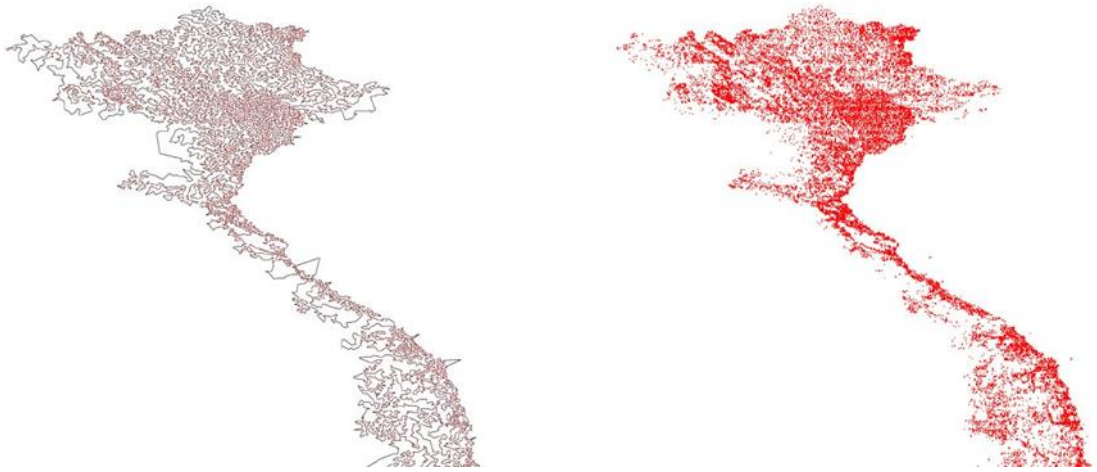
Trong những năm 1950 và 1960, bài toán trở nên phổ biến trong giới nghiên cứu khoa học ở châu Âu và Mỹ. George Dantzig, Delbert Ray Fulkerson và Selmer.

M. Johnson ở công ty RAND tại Santa Monica đã có đóng góp quan trọng cho bài toán này, biểu diễn bài toán dưới dạng quy hoạch nguyên và đưa ra phương pháp mặt phẳng cắt để tìm ra lời giải. Với phương pháp mới này, họ đã giải được tối ưu một trường hợp có 49 thành phố bằng cách xây dựng một chu trình và chứng minh rằng không có chu trình nào ngắn hơn. Trong những thập niên tiếp theo, bài toán được nghiên cứu bởi nhiều nhà nghiên cứu trong các lĩnh vực toán học, khoa học máy tính, hóa học, vật lý, và các ngành khác. Năm 1972, Richard M. Karp chứng minh rằng bài toán chu trình Hamilton là NP-đầy đủ, kéo theo bài toán TSP cũng là NP-đầy đủ. Đây là một lý giải toán học cho sự khó khăn trong việc tìm kiếm chu trình ngắn nhất.

Một bước tiến lớn được thực hiện cuối thập niên 1970 và 1980 khi Grötschel, Padberg, Rinaldi và cộng sự đã giải được những trường hợp lên tới 2392 thành phố, sử dụng phương pháp mặt phẳng cắt và nhánh cận.

Trong thập niên 1990, Applegate, Bixby, Chvátal, và Cook phát triển một chương trình mang tên Concorde giải được nhiều trường hợp có độ lớn kỷ lục hiện nay. Gerhard Reinelt xuất bản một bộ dữ liệu các trường hợp có độ khó khác nhau mang tên TSPLIB năm 1991, và nó đã được sử dụng bởi nhiều nhóm nghiên cứu để so sánh kết quả. Năm 2005, Cook và cộng sự đã giải được một trường hợp có 33810 thành phố, xuất phát từ một bài toán thiết kế vi mạch. Đây là trường hợp lớn nhất đã được

giải trong TSPLIB. Trong nhiều trường hợp khác với hàng triệu thành phố, người ta đã tìm được lời giải với sai số không quá 1% so với lời giải tối ưu.

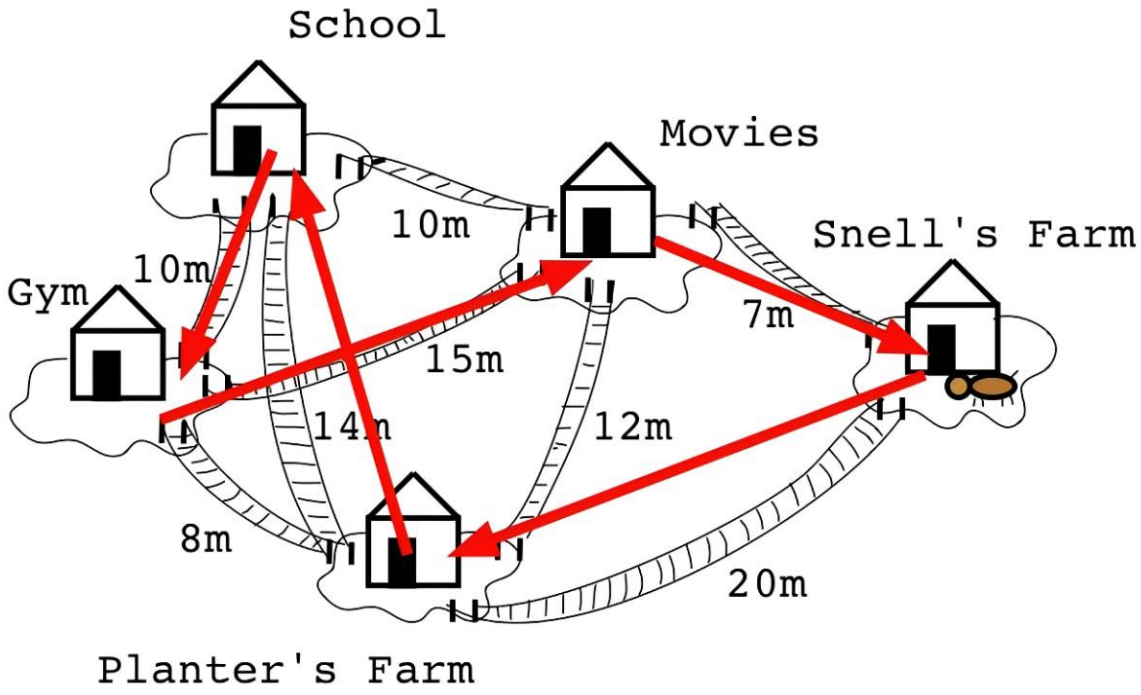


Hình 1.2: 22,775 thành phố ở Việt Nam, trích xuất dữ liệu từ Cơ quan Bản đồ và Hình ảnh Quốc gia về tên các đối tượng địa lý

1.1.2.2 Phát biểu bài toán

Cho đồ thị đầy đủ n đỉnh vô hướng, có trọng số $G = (V, E)$. Tìm chu trình $v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_n \rightarrow v_1$ với $v_i \in V, i = 1, \dots n$ sao cho tổng trọng số hành trình trên các cạnh $(v_i, v_i + 1)$ và (v_n, v_1) là nhỏ nhất.

Một chu trình như vậy còn gọi là chu trình Hamilton, nó đi qua tất cả các đỉnh của đồ thị đúng một lần. Đồ thị đầy đủ, luôn tồn tại chu trình Hamilton.



Hình 1.3: Minh họa bài toán người du lịch

1.1.2.3 Độ phức tạp của bài toán

Bài toán TSP thuộc lớp bài toán NP-Hard (lớp các bài toán không có giải thuật trong thời gian đa thức).

Việc thực hiện liệt kê tất cả các chu trình là điều gần như không thể với số đỉnh lớn (đồ thị n đỉnh phải duyệt $n!$ chu trình). Số chu trình phải duyệt tăng rất nhanh khi số đỉnh n càng lớn. Ngay với 1 đồ thị 100 đỉnh, việc duyệt toàn bộ cũng là điều rất khó thực hiện.

Phiên bản quyết định của bài toán người du lịch là NP-đầy đủ. Ngay cả khi khoảng cách giữa các thành phố là khoảng cách Euclide, bài toán vẫn là NP-khó. Với n thành phố thì có: $1/2 \times (n - 1)!$ đường đi.

Trong trường hợp tổng quát, bài toán người du lịch là NPO-đầy đủ. Khi các khoảng cách thỏa mãn bất đẳng thức tam giác và đối xứng, bài toán là APX-đầy đủ và thuật toán Christofides có thể tìm lời giải xấp xỉ không quá 1,5 lần lời giải tối ưu. Khi các khoảng cách là bất đối xứng nhưng thỏa mãn bất đẳng thức tam giác, thuật toán tốt nhất hiện nay đạt tỉ lệ xấp xỉ $O(\log n / \log\log n)$

Việc thực hiện liệt kê hết tất cả các chu trình là điều gần như không thể với số đỉnh lớn (đồ thị n đỉnh phải duyệt $n!$ chu trình). Số chu trình phải duyệt tăng rất nhanh khi số đỉnh n càng lớn. Ngay với 1 đồ thị 100 đỉnh, việc duyệt toàn bộ cũng là điều rất khó thực hiện.

1.1.2.4 Ứng dụng của bài toán

Bài toán TSP đã có nhiều ứng dụng trong lập kế hoạch, hậu cần, cũng như thiết kế vi mạch, …

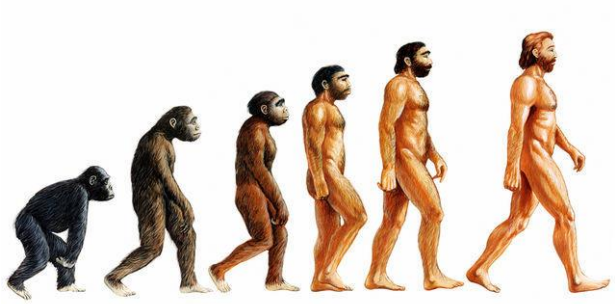
1.2 Thuật toán tiến hóa đa nhiệm

1.2.1 Thuật toán tiến hóa

1.2.1.1 Tiến hóa sinh học

Trong sinh học, tiến hóa là sự thay đổi đặc tính di truyền của một quần thể sinh học qua những thế hệ nối tiếp nhau. Những đặc tính này là sự biểu hiện của các gen được truyền từ cha mẹ sang con cái thông qua quá trình sinh sản. Nguyên nhân dẫn đến sự khác biệt của những đặc tính đó trong quần thể là do kết quả của đột biến, tái tổ hợp di truyền và nguồn gốc các biến dị di truyền khác. Hiện tượng tiến hóa xảy ra khi các tác nhân tiến hóa như chọn lọc tự nhiên (bao gồm cả chọn lọc giới tính) và trôi dạt

di truyền tác động lên sự đa dạng của những đặc tính này, dẫn đến kết quả là vài đặc tính sẽ trở nên phổ biến hoặc hiếm gặp hơn ở trong quần thể.



Hình 1.4 Minh họa tiến hóa trong tự nhiên

Tiến hóa trong tự nhiên là quá trình tuyệt vời khi tối ưu hóa liên tục về gen về các đặc tính tốt từ các thế hệ trước biểu hiện ở các thế hệ sau. Vì vậy đã có nhiều nhà nghiên cứu tìm tòi nghiên cứu ra các giải thuật giải các bài toán tối ưu dựa trên phương pháp tiến hóa tự nhiên này.

1.2.1.2 Giải thuật tiến hóa

Một giải thuật tiến hóa (Evolutionary Algorithm - EA) là một thuật toán tối ưu hóa heuristic sử dụng các kỹ thuật bắt nguồn từ các cơ chế tiến hóa hữu cơ (Organic Evolution) chẳng hạn như biến dị, tái tổ hợp và chọn lọc tự nhiên để tìm một cấu hình tối ưu cho một hệ thống với các ràng buộc cụ thể.

Các thuật toán tiến hóa EA (Back, Hammel, & Schwefel, 1997; Coello, 2006; & Fonseca & Fleming, 2007) là các kỹ thuật tối ưu Heuristics dựa trên nguyên lý của Darwin về sự lựa chọn tự nhiên.

Thuật toán EA bao gồm rất nhiều mô hình khác nhau với ý tưởng là sử dụng các tác động trên một quần thể để biến đổi quần thể, nâng cao khả năng thích nghi của quần thể.

```
1 Khởi tạo cá thể
2 Đánh giá độ thích nghi
3 while điều kiện dừng chưa thỏa do
4 |   Lai ghép và Đột biến
5 |   Đánh giá độ thích nghi
6 |   Chọn lọc cá thể
7 end
```

Hình 1.5: Cấu trúc tổng quát thuật toán tiến hóa

Các thuật toán EA đã được phát triển theo nhiều hướng khác nhau như Tiến hóa vi phân (thuật ngữ gốc: Differential Evolution - DE), Tối ưu bầy đàn (thuật ngữ gốc: Particle Swarm Optimization - PSO), Tối ưu đàn kiến (thuật ngữ gốc: Ant Colony Optimization - ACO), ... với cùng mục tiêu là giải quyết bài toán tối ưu hóa liên tục. Qua lý thuyết và thực tiễn, EA đã được minh chứng là một bộ tối ưu hóa thể hiện được tính khoa học và tính hiệu quả của chúng.

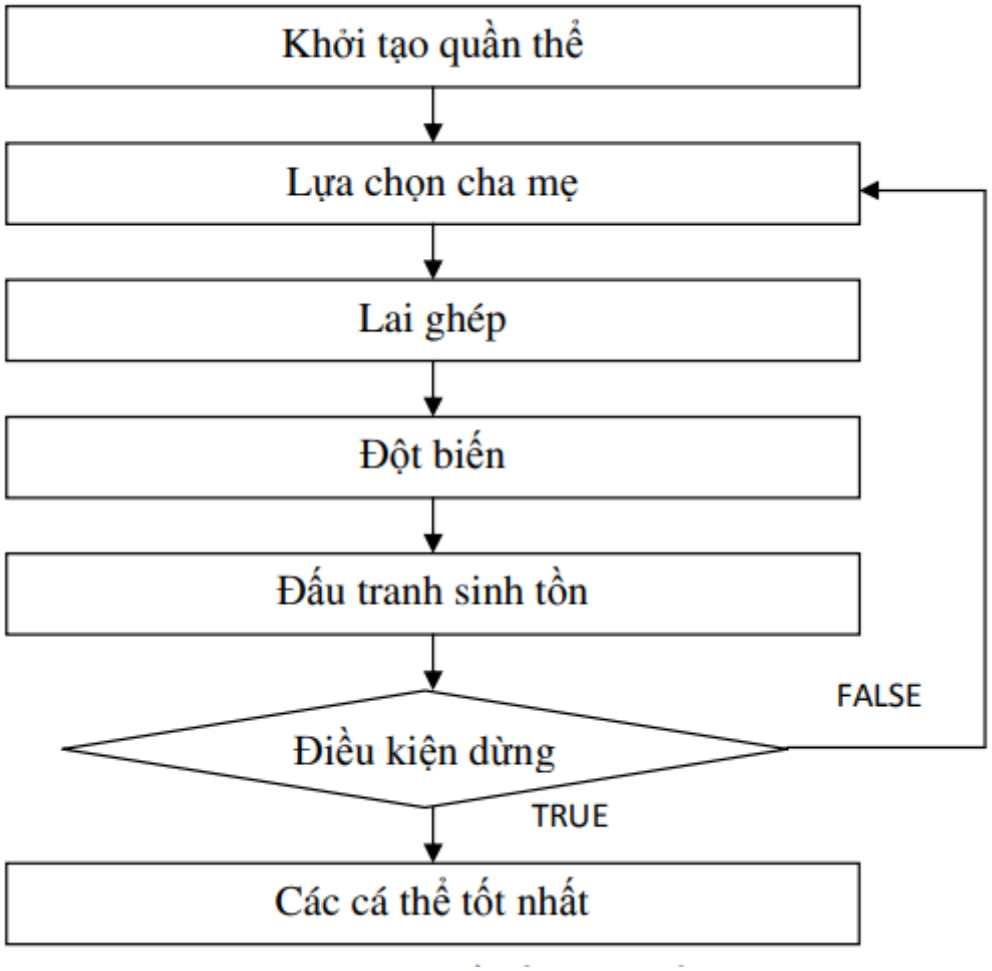
Các thuật toán EA thường khác nhau chủ yếu ở cách tổ hợp lời giải mới qua các bước lai ghép (cross over) và đột biến (mutation).

Trong vài thập kỷ qua, các thuật toán tiến hóa đã được áp dụng thành công để giải các bài toán tối ưu khác nhau trong khoa học.

1.2.1.3 Thuật toán di truyền (GA – Genetic Algorithm)

Giải thuật di truyền cũng dựa trên quan niệm cho rằng quá trình tiến hóa tự nhiên là hoàn hảo nhất, hợp lý nhất và bản thân nó đã thể hiện tính tối ưu.

Giải thuật di truyền bao gồm bốn bước chính: Mã hóa lời giải, khởi tạo quần thể, sử dụng các phép toán di truyền và đánh giá độ thích nghi. Thuật toán được thực hiện qua nhiều thế hệ thì lời giải đưa ra càng được tối ưu.



Hình 1.6: Sơ đồ thuật toán di truyền

a) Khởi tạo quần thể

Quần thể trong thuật toán GA là tập hợp những cá thể - lời giải hợp lệ hoặc khả thi đối với bài toán đang được xem xét. Việc khởi tạo quần thể cùng với việc chọn lọc cá thể cho thế hệ tiếp theo đóng vai trò quan trọng trong việc đảm bảo các cá thể trong không gian tìm kiếm cũng như đầu vào của các toán tử tiến hóa là hợp lệ.

Hiện nay, có nhiều phương pháp khởi tạo quần thể cho một thuật toán GA. Việc lựa chọn phương pháp khởi tạo nào sẽ tùy thuộc vào bài toán đang được giải và cách biểu diễn (mã hóa) lời giải.

b) Toán tử lai ghép

Phép lai ghép có cách thực hiện tương tự quá trình lai ghép trong tự nhiên. Các cá thể con tạo ra từ việc cho hai cá thể cha mẹ lai ghép với nhau sẽ kế thừa những đặc điểm từ cả cha và mẹ. Và cách lựa chọn phần gen nào và ghép như thế nào thì sẽ phụ thuộc vào từng mục đích sử dụng, từng bài toán.

c) Toán tử đột biến

Sử dụng toán tử đột biến cũng mang ý nghĩa tương tự như quá trình đột biến trong tiến hóa tự nhiên. Mặc dù việc lai ghép cho phép các cá thể con được thừa hưởng một phần gen của cá thể cha, một phần gen cá thể mẹ, nhưng nó không thực sự tạo ra sự đa dạng và nguồn gen mới cho quần thể. Đột biến thì có thể. Toán tử đột biến sẽ làm thay đổi ngẫu nhiên một phần của cá thể.

d) Chọn lọc cá thể

Một số phương pháp để chọn lọc cá thể cha mẹ tiến hành lai ghép:

1. *Chọn lọc dựa theo giá trị thích nghi (fitness based selection):* Chọn lọc cá thể cha mẹ dựa theo giá trị thích nghi thường gồm các phương pháp sau: lựa chọn xén (truncation selection), lựa chọn theo vòng quay roulette (roulette wheel selection), lựa chọn theo kiểu rải (stochastic universal sampling), …
2. *Chọn lọc ngẫu nhiên (random selection):* Các cá thể cha mẹ được chọn lọc hoàn toàn ngẫu nhiên theo số lượng hoặc tỉ lệ xác định sẵn. Trong một số bài toán, có những cá thể cha mẹ không phải là tốt nhất trong quần thể nhưng khi lai ghép với nhau lại có thể cho ra những cá thể con rất tốt.
3. *Chọn lọc dựa theo thứ hạng trong quần thể (Rank selection):* Đây là một trong những cách làm phổ biến trong thuật toán tiến hóa. Trong cách chọn này, số nguyên $k > 0$ được xác định làm tham số cho phép chọn lọc cạnh tranh. Ở mỗi lượt chọn, có một số lượng cá thể được chọn ngẫu nhiên từ quần thể đang xét. Dựa vào độ thích nghi, cá thể tốt nhất trong số cá thể được chọn ngẫu nhiên sẽ được chọn làm cá thể cha mẹ.
4. *Chọn lọc cạnh tranh (tournament selection):* Đây là một trong những cách làm phổ biến trong thuật toán tiến hóa. Trong cách chọn này, số nguyên $k > 0$ được xác định làm tham số cho phép chọn lọc cạnh tranh. Ở mỗi lượt chọn, có một số lượng cá thể được chọn ngẫu nhiên từ quần thể đang xét. Dựa vào độ thích nghi, cá thể tốt nhất trong số cá thể được chọn ngẫu nhiên sẽ được chọn làm cá thể cha mẹ.

e) Điều kiện dừng của thuật toán (termination condition)

Tùy từng bài toán sẽ chọn lựa điều kiện dừng khác nhau. Mục đích của việc chọn lựa điều kiện dừng để có thể tìm ra lời giải gần nhất với lời giải đúng và thỏa mãn tiêu chí đề ra (thời gian, chi phí, tính đúng đắn, …). Để tìm được thời điểm đó, thông thường các điều kiện sau có thể được chọn làm điều kiện dừng: khi chất lượng lời giải không tiếp tục cải thiện, sau một số thế hệ nhất định, khi hàm mục tiêu hoặc độ thích nghi của cá thể đạt ngưỡng nhất định, …

1.2.2 Thuật toán tiến hóa đa nhiệm

Các thuật toán tiến hóa đã áp dụng thành công để giải các vấn đề tối ưu trong khoa học kỹ thuật. Các vấn đề này thường được phân vào hai nhóm:

1) Tối ưu hóa đơn mục tiêu SOO (single-objective optimization), trong đó mỗi điểm trong không gian tìm kiếm của bài toán được ánh xạ thành một giá trị mục tiêu vô hướng.
2) Tối ưu hóa đa mục tiêu MOO (multi-objective optimization), trong đó mỗi một điểm trong không gian tìm kiếm của bài toán được ánh xạ thành một vectơ (các giá trị) mục tiêu.

Hầu hết các thuật toán đó đều được xây dựng để giải quyết một bài toán tối ưu tại một thời điểm dựa trên một quần thể. Tuy nhiên khi khoa học máy tính ngày càng phát triển này phát sinh nhiều vấn đề phức tạp và yêu cầu cần giải nhiều bài toán tối ưu cùng một lúc thì những thuật toán trên chưa đáp ứng được điều này. Do vậy, Gupta, Ong, & Feng vào năm 2017 đã đưa ra 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 nhiệm MFO (multifactorial optimization)

1.2.2.1 Tối ưu hóa đa nhiệm MFO (multifactorial optimization)

Giống như các thuật toán tiến hóa cổ điển dựa trên sinh học tiến hóa, MFO cũng được thiết kế dựa trên mô hình của sự kế thừa đa nhiệm, trong đó các đặc điểm phát triển của thế hệ con cái (offspring) bị ảnh hưởng bởi sự tương tác của các yếu tố di truyền và văn hoá (Cloninger, Rice, & Reich, 1979; Tayarani, & Bennett, 2013). Nói một cách khác, di truyền và văn hóa không thể xem xét một cách độc lập.

Ví dụ, trong khi những gì một cá thể học được có thể phụ thuộc vào kiểu gen của nó, sự lựa chọn tác động lên hệ gen có thể được tạo ra hoặc thay đổi bởi sự lan rộng của một đặc điểm văn hoá nào đó.

Những đặc điểm văn hoá này thường bắt nguồn từ việc học tập xã hội hoặc từ cha mẹ (parents) trải qua một số tập quán và trở thành sở thích nhất định cho con cái của họ. Sự tương đương tính toán của việc thừa kế đa nhiệm vụ, cho mục đích của đa tác vụ tiến hóa hiệu quả, được thiết lập bằng cách xem xét mỗi tác vụ (task) tối ưu đóng góp một "môi trường" khác nhau trong đó các cá thể con cháu được tiến hóa. Do đó, MFO dẫn tới sự tồn tại của nhiều biểu tượng văn hóa (culture bias hay memes), là một trào lưu văn hóa hoặc một ý tưởng lan truyền rộng rãi và nhanh chóng trong xã hội.

Trong lịch sử, memes giống như một "vấn đề văn hóa" lan truyền từ người này sang người khác thông qua việc truyền miệng, bắt chước, … hay "tin tưởng quá" thành truyền thuyết, truyện ngụ ngôn,…) mà mỗi cái tương ứng với một tác vụ tối ưu. Vì vậy, mặc dù cấu trúc cơ bản của MFO sẽ tương tự như của một EA cổ điển, nó được tăng cường với các tính năng được vay mượn từ các mô hình đa thừa kế.

1.2.2.2 Các định nghĩa và khái niệm liên quan

Xem xét tình huống trong đó 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 problems). 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 $f_j: X_j \rightarrow R$. 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 tác vụ tiến hóa được xây dựng trên sự song song hóa hoàn toàn của việc tìm kiếm dựa vào quần thể với mục đích tìm $\{x_1, x_2, \dots, x_{K-1}, x_k\} = argmin\{f_1(x), f_2(x), \dots, f_{K-1}(x), f_K(x)\}$, trong đó x_j là một giải pháp phù hợp trong X_j.

Để thiết kế các lời giải cho MFO, cần phải mô hình hóa một kỹ thuật tổng quát cho việc so sánh các cá thể trong một quần thể của một môi trường đa tác vụ. Để làm điều này, đầu tiên phải định nghĩa một tập các thuộc tính cho mọi cá thể p_i, trong đó $i \in \{1, 2, \dots, |P|\}$, trong một quần thể P. Chú ý rằng, các cá thể được mã hóa trong một

không gian tìm kiếm duy nhất chứa đựng $X_1, X_2, \dots, X_K$, và có thể được giải mã thành từng giải pháp cụ thể cho mỗi tác vụ (trong K tác vụ cần tối ưu). Dạng giải mã của cá thể p_i có thể được viết như là $\{x_1^i, x_2^i, \dots, x_K^i\}$, trong đó $x_1^i \in X_1, x_2^i \in X_2, \dots$ và $x_K^i \in X_K$.

Định nghĩa 1 (Factorial Cost: Giá của của một cá thể đối với từng tác vụ): Factorial cost Ψ_j^i của cá thể p_i đối với tác vụ T_j là $\Psi_j^i = \lambda.\delta_j^i + f_j^i$; 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 p_i phù hợp đối với tác vụ T_j (sự vi phạm ràng buộc bằng 0), chúng ta có $\Psi_j^i = f_j^i$.

Định nghĩa 2 (Factorial rank: Xếp hạng của một cá thể đối với từng tác vụ): Factorial rank r_j^i của cá thể p_i trên tác vụ T_j là chỉ số (index) của p_i trong quần thể P được sắp xếp theo thứ tự tăng dần của Ψ_j^i. 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, vì hiệu năng của hai cá thể là tương đương đối với tác vụ thứ j^{th}, vì vậy chúng ta gán nhãn chung như là $j - counterparts$ (bản sao hay giống nhau đối với tác vụ thứ j^{th}).

Định nghĩa 3 (Scalar Fitness: Sự thích hợp): Danh sách các xếp hạng của một cá thể p_i trên tất cả các tác vụ (K tác vụ) $\{r_1^i, r_2^i, \dots, r_K^i\}$ 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 $\varphi_i = 1/\, min\, \{r_j^i\}$ với $j \in \{1, 2, \dots, K\}$.

Định nghĩa 4 (Skill Factor: Tác vụ đầy đủ hoặc 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 $\tau_i = argmin_j\{r_j^i\}$ trong đó $j \in \{1, 2, \dots, K\}$. Một khi sự thích hợp của các cá thể được quy theo Định nghĩa 3, sự 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 nhiệm vụ nếu $\varphi_a > \varphi_b$. Nếu hai cá thể có cùng skill factor $\tau_a = \tau_b = j$ (phù hợp với tác vụ thứ j^{th} nhất), chúng ta 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

fitness 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 $\varphi^* \geq \varphi_i$ với mọi $i \in \{1, 2, \dots, K\}$.

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

1.2.2.3 Thuật toán tối ưu hóa đa nhiệm

Nhiệm vụ của chúng ta khi giải bài toán MFO là cùng lúc tìm bộ nghiệm tối ưu với mỗi thành phần là một lời giải cho bài toán con tương ứng. Gupta, Ong, and Feng đề xuất MFEA nhằm tận dụng ưu thế của tìm kiếm dựa trên quần thể trong việc giải các bài toán tối ưu hóa cùng lúc. Theo hướng tiếp cận này, Gupta, Ong, and Feng cho rằng vật chất di truyền của các bài toán khác nhau có thể có những điểm chung nhất định (ví dụ như không gian nghiệm gần giống nhau) giúp cho việc bài toán này cũng đồng thời hỗ trợ giải quyết bài toán kia hoặc có bài toán có những tính chất nhất định giúp hỗ trợ bài toán kia không bị cục bộ hóa (ví dụ như hàm tối ưu lồi giúp hỗ trợ hàm tối ưu đa cực trị). Rất nhiều các nghiên cứu đã chỉ ra sự hiệu quả của MFEA đối với các bài toán tối ưu hóa đơn mục tiêu và đa mục tiêu

Thuật toán tiến hóa đa nhiệm MFEA (multifactorial evolutionary algorithm) là thuật toán được sử dùng để giải quyết cho vấn đề tối ưu đa nhiệm MFO. Thuật toán MFEA cũng là một thuật toán dựa trên mô hình tiến hóa trong sinh học nhưng có sự khác biệt đó là thay vì chỉ kế thừa đơn lẻ thì sẽ kế thừa đa nhiệm. Vì vậy, cấu trúc cơ bản của thuật toán MFEA sẽ tương tự như thuật toán EA cổ điển, việc thêm vào các đặc trưng văn hóa sẽ biến MFEA thành các lời giải MFO hiệu quả.

Cấu trúc cơ bản của MFEA:

- *Đầu vào*: Quân thể được khởi tạo ngẫu nhiên cho bài toán
- *Đầu ra*: Cá thể tối ưu (lời giải) cho bài toán
- *Chi tiết thuật toán*:

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 của mỗi cá thể
4. While (*điều kiện dừng chưa thỏa mãn*) do:
   - Áp dụng các thao tác di truyền tới current-pop (P) để sinh ra offspring-pop (C)
   - Đánh giá các cá thể trong offspring-pop (C) tạo thành intermediate-pop
   - Cập nhật scalar-fitness và skill factor cho mọi cá thể trong intermediate-pop
   - Lựa chọn các cá thể tốt nhất từ intermediate-pop để hình thành current-pop kế tiếp
5. End while

a) Khởi tạo quần thể

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 $j \in \{1, 2, \dots, K\}$. 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 D_j khóa ngẫu nhiên đầu tiên của chromosome tương ứng với p_i.

b) Toán tử 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 lệch 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 quy định (rmp) hoặc phép đột biến được sử dụng. Các bước tạo ra các cá thể con được mô tả trong thuật toán các thao tác di truyền như sau:

- Đầu vào: Hai cá thể cha mẹ
- Đầu ra: Các cá thể con
- Chi tiết thuật toán:
  1. Xét hai cá thể cha mẹ p_a và p_b được lựa chọn ngẫu nhiên trong current-pop (P)
  2. Sinh ra một số ngẫu nhiên rand trong khoảng $[0,1]$
  3. If ($\tau_a = \tau_b$) hoặc ($rand < rmp$) then

     Thực hiện lai ghép hai cá thể p_a và p_b sinh ra hai cá thể con c_a và c_b
  4. Else

     Đột biến p_a sinh ra cá thể con c_a

     Đột biến p_b sinh ra cá thể con c_b
  5. End If

Sự kế thừa văn hóa của *offspring:*

- Đầu vào: Hai cá thể cha mẹ
- Đầu ra: Cá thể con giống cha hoặc mẹ
- Chi tiết thuật toán: Một cá thể 'c' có thể có cá thể cha mẹ p_a và p_b hoặc chỉ một cha hoặc mẹ (p_a hoặc p_b)
  1. If ('c' có hai cha mẹ p_a và p_b) then

     Sinh một số ngẫu nhiên $rand$ ($0 \leq rand \leq 1$)

     If ($rand < 0.5$) then

     'c' được gắn skill factor là τ_a

     Else

     'c' được gắn skill factor là τ_b

     End If
  2. Else

     'c' được gắn skill factor theo skill factor của cá thể đột biến ra nó

3. End If
4. *Factorial cost* của c trên các tác vụ còn lại được đặt là một số rất lớn

c) Sự lựa chọn

Cũng giống như EA cổ điển, 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.

CHƯƠNG 2. NGHIÊN CỨU THUẬT TOÁN TIẾN HÓA ĐA NHIỆM VỤ GIẢI QUYẾT BÀI TOÁN NGƯỜI DU LỊCH

2.1 Giải thuật

2.1.1 Mã hóa bài toán

2.1.1.1 Mã hóa đồ thị đầu vào

Các bộ dữ liệu kiểm thử được lấy tại http://www.tsp.gatech.edu/ (cung cấp các bộ dữ liệu chuẩn trên thực tế)

Đồ thị được mã hóa bằng danh sách mảng các điểm và tọa độ tương ứng của chúng.

```
NAME : a280
COMMENT : drilling problem (Ludwig)
TYPE : TSP
DIMENSION: 280
EDGE_WEIGHT_TYPE : EUC_2D
NODE_COORD_SECTION
  1 288 149
  2 288 129
  3 270 133
  4 256 141
  5 256 157
  6 246 157
  7 236 169
  8 228 169
  9 228 161
 10 220 169
 11 212 169
 12 204 169
 13 196 169
 14 188 169
 15 196 161
 16 188 145
 17 172 145
 18 164 145
 19 156 145
 20 148 145
```

Hình 2.1: Ví dụ về bộ dữ liệu đồ thị chuẩn

Trong đó trọng số cột đầu tiên là số hiệu của đỉnh, số thứ hai là hoành độ, số thứ ba là tung độ. Khoảng cách giữa hai đỉnh $M\ (x_i, y_i)$ và $N\ (x_j, y_j)$ của đồ thị (trọng số cho cạnh) được tính bằng công thức:

$$d_{M,N} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

2.1.1.2 Mã hóa chu trình

Chu trình được mã hóa bằng mảng có thứ tự các số hiệu của đỉnh. Với đồ thị n đỉnh thì mảng có kích thước n phần tử

Mỗi chu trình cần thêm thông tin về chi phí của toàn bộ chu trình đó. Chi phí này được tính bằng tổng độ dài tất cả các cạnh tạo nên chu trình đó.

Mỗi chu trình là một lời giải, trong giải thuật tiến hóa đa nhiệm coi đó như một cá thể. Việc tiến hóa về sau ta dựa trên tập chu trình khởi tạo ban đầu và tìm ra kết quả tốt nhất sau một số thế hệ.

2.1.2 Khởi tạo quần thể

Quần thể ban đầu được khởi tạo bằng cách sinh ngẫu nhiên các chu trình. Kích thước của quần thể sẽ là kích thước của task có kích thước quần thể lớn nhất.

Cài đặt code:

```
private void initializePopulation() {
    for (int i = 0; i < INDIVIDUAL_COUNT; i++) {
        Individual newIndividual = new Individual();
        for (int j = 0; j < unifiedTaskDimension; j++) {
            newIndividual.setChromosome(j, j);
        }
        currentPopulation.setIndividuals(i, newIndividual);
        //Tạo các cá thể
        for (int j = 0; j < INDIVIDUAL_COUNT; j++) {
            randomlyArrange(currentPopulation.getIndividuals().indexOf(newIndividual));
        }
        //Đánh giá từng cá thể với các nhiệm vụ tối ưu hóa
        for (Task task : taskList) {
            calculateFitnessValue(currentPopulation.getIndividuals(i), task);
        }
    }
    //Sắp xếp cá thể theo Fitness từng nhiệm vụ tối ưu.
    for (Task task : taskList) {
        sortByFitness(currentPopulation, task);
    }
    //Tính skill factor.
    for (int i = 0; i < INDIVIDUAL_COUNT; i++) {
        Individual currentIndividual = currentPopulation.getIndividuals(i);
        ArrayList<Integer> ranks = new ArrayList<>(currentIndividual.getFactorialRanks());
        currentIndividual.setSkillFactor(ranks.indexOf(Collections.min(ranks)));
    }
}
```

Hình 2.2: Code khởi tạo quần thể

2.1.3 Lai ghép

Phương thức lai ghép được thực hiện dựa trên 2 cá thể đầu vào:

C1	1	2	5	3	6	9	10	8	4	7

C2	10	2	4	1	5	3	9	6	8	7

Thực hiện lai ghép theo quy tắc:

- Cắt một đoạn (segment) ngẫu nhiên trong cá thể cha mẹ. Ví dụ cắt đoạn từ vị trí 2 đến vị trí 5 của chu trình C1:

Con			5	3	6	9				

- Xét từ đầu đến cuối chu trình C2, nạp dần các điểm chưa có trong con lai theo thứ tự duyệt ta được chu trình mới:

Con	10	2	5	3	6	9	4	1	8	7

Cài đặt code

```
private Individual[] crossOver(int[] soulMates, ArrayList<Individual> parents) {
    int firstCutIndex = new Random().nextInt(unifiedTaskDimension - 2);
    int secondCutIndex = 0;
    boolean done = false;
    //Chọn hai cá thể khác nhau
    while (!done) {
        secondCutIndex = new Random().nextInt(unifiedTaskDimension - 1);
        if (secondCutIndex >= firstCutIndex) {
            done = true;
        }
    }
    //Lấy thông tin từ hai cá thể bố mẹ.
    Individual parentA = parents.get(soulMates[0]);
    Individual parentB = parents.get(soulMates[1]);
    //Lấy đoạn gen từ cá thể bố mẹ
    int[] segmentA = createSegment(firstCutIndex, secondCutIndex, parentA);
    int[] segmentB = createSegment(firstCutIndex, secondCutIndex, parentB);
    Individual[] returnCrossChildren = new Individual[2];
    //Sao chép đoạn gen lấy từ cá thể bố mẹ sang cá thể con
    Integer[] childAChromosome = insertSegment(firstCutIndex, secondCutIndex, segmentA);
    Integer[] childBChromosome = insertSegment(firstCutIndex, secondCutIndex, segmentB);
    //Điền đầy đủ tạo thành cá thể con mới
    cross(firstCutIndex, secondCutIndex, childAChromosome, parentB, parentA);
    cross(firstCutIndex, secondCutIndex, childBChromosome, parentA, parentB);
    returnCrossChildren[0] = new Individual();
    returnCrossChildren[0].setChromosome(new ArrayList<>(Arrays.asList(childAChromosome)));
    returnCrossChildren[0].setParentNumber(2);
    returnCrossChildren[0].setParentASkillFactor(parentA.getSkillFactor());
    returnCrossChildren[0].setParentBSkillFactor(parentB.getSkillFactor());
    returnCrossChildren[1] = new Individual();
    returnCrossChildren[1].setChromosome(new ArrayList<>(Arrays.asList(childBChromosome)));
    returnCrossChildren[1].setParentNumber(2);
    returnCrossChildren[1].setParentASkillFactor(parentB.getSkillFactor());
    returnCrossChildren[1].setParentBSkillFactor(parentA.getSkillFactor());
    return returnCrossChildren;
}
```

Hình 2.3: Code toán tử lai ghép (1)

```
private void cross(int firstCutIndex, int secondCutIndex, Integer[] childChromosome,
                   Individual notAlikeParent, Individual alikeParent) {
    for (int i = firstCutIndex; i <= secondCutIndex; i++) {
        boolean alreadyThere = false;
        for (int j = firstCutIndex; j <= secondCutIndex; j++) {
            if (childChromosome[j] == notAlikeParent.getChromosome(i)) {
                alreadyThere = true;
                break;
            }
        }
        if (!alreadyThere) {
            int index = i;
            int notAlreadyThereValue = notAlikeParent.getChromosome(index);
            while (true) {
                int sameIndexValue = alikeParent.getChromosome(index);
                int sameValueIndex = notAlikeParent.getChromosome().indexOf(sameIndexValue);
                if ((sameValueIndex < firstCutIndex) || (sameValueIndex > secondCutIndex)) {
                    childChromosome[sameValueIndex] = notAlreadyThereValue;
                    break;
                } else {
                    index = sameValueIndex;
                }
            }
        }
    }
    for (int i = 0; i < unifiedTaskDimension; i++) {
        if ((childChromosome[i] == -1)) {
            childChromosome[i] = notAlikeParent.getChromosome(i);
        }
    }
}
```

Hình 2.4: Code toán tử lai ghép (2)

```
private Integer[] insertSegment(int firstCutIndex, int secondCutIndex, int[] segment) {
    Integer[] returnChildChromosome = new Integer[unifiedTaskDimension];
    int segmentIndex = 0;
    for (int i = 0; i < unifiedTaskDimension; i++) {
        if ((i >= firstCutIndex) && (i <= secondCutIndex)) {
            returnChildChromosome[i] = segment[segmentIndex];
            segmentIndex++;
        } else {
            returnChildChromosome[i] = -1;
        }
    }
    return returnChildChromosome;
}

private int[] createSegment(int firstCutIndex, int secondCutIndex, Individual parent) {
    int capacityOfSegment = (secondCutIndex - firstCutIndex) + 1;
    int[] returnSegment = new int[capacityOfSegment];
    ArrayList<Integer> parentChromosome = new ArrayList<>(parent.getChromosome());
    int segmentIndex = 0;
    for (int i = 0; i < unifiedTaskDimension; i++) {
        if ((i >= firstCutIndex) && (i <= secondCutIndex)) {
            returnSegment[segmentIndex] = parentChromosome.get(i);
            segmentIndex++;
        }
    }
    return returnSegment;
}
```

Hình 2.5: Code toán tử lai ghép (3)

2.1.4 Đột biến

Phương thức đột biến được thực hiện dựa trên một cá thể đầu vào:

C1	1	2	5	3	6	9	10	8	4	7

Thực hiện đột biến bằng cách tráo đổi bất kì hai vị trí khác nhau cho nhau.

Ví dụ với đột biến C1 bằng cách tráo đổi vị trí thành phố 1 với thành phố 5. Khi đó ta được chu trình mới:

C2	5	2	1	3	6	9	10	8	4	7

Cài đặt code:

```
private Individual mutation(Individual parent) {
    Individual mutantChild = new Individual();
    mutantChild.setChromosome(new ArrayList<>(parent.getChromosome()));
    int firstCity = new Random().nextInt(unifiedTaskDimension);
    int secondCity = 0;
    boolean done = false;
    while (!done) {
        secondCity = (new Random().nextInt(unifiedTaskDimension));
        if (secondCity != firstCity) {
            done = true;
        }
    }
    int temp = mutantChild.getChromosome(firstCity);
    mutantChild.setChromosome(firstCity,
            mutantChild.getChromosome(secondCity));
    mutantChild.setChromosome(secondCity, temp);
    mutantChild.setParentNumber(1);
    mutantChild.setParentASkillFactor(parent.getSkillFactor());
    mutantChild.setParentBSkillFactor(parent.getSkillFactor());
    mutantChild.setSkillFactor(parent.getSkillFactor());
    return mutantChild;
}
```

Hình 2.6: Code toán tử đột biến

2.1.5 Chọn lọc

Kích thước quần thể là cố định qua các thế hệ. Ở mỗi thế hệ sau tiến hóa lại xuất hiện các cá thể mới được sinh ra bằng phép lai ghép và đột biến do đó cần phải có sự chọn lọc để có thể đảm bảo tính cân bằng của quần thể cũng chính là tránh các lỗi phát sinh về bộ nhớ khi kích thước quần thể quá lớn.

Số cá thể ở một thế hệ = Kích thước mặc định + Số cá thể mới sinh

Cách thức chọn lọc cá thể được đánh giá dựa trên chi phí của mỗi chu trình. Cá thể được chọn làm lời giải tốt nhất sau cùng chính là cá thể có chi phí nhỏ nhất trong quần thể sau quá trình tiến hóa nhiều lần qua các thế hệ.

Ban đầu khi khởi tạo quần thể thì các cá thể sẽ được đánh giá và sắp xếp theo thuật toán merge sort theo mức chi phí tăng dần. Tuy nhiên khi số thế hệ đạt đến một mức nhất định, việc tìm ra chu trình có chi phí nhỏ hơn càng ngày càng khó dẫn đến việc tập cá thể trong quần thể gần như không biến đổi, điều này làm giảm sự đa dạng nguồn gen cho tiến hóa ở các thế hệ sau.

Do đó em đã chọn lọc như sau:

- Sắp xếp quần thể theo chi phí tăng dần
- Lấy ra số lượng cá thể có chi phí nhỏ nhất nhất định bằng kích thước mặc định
- Lấy ngẫu nhiên trong các cá thể còn lại theo một chỉ số ngẫu nhiên (random) để cho vào quần thể đạt đến kích thước cố định.

Với cách thức chọn lọc này em nghĩ sẽ đảm bảo sự phong phú của quần thể sau nhiều thế hệ.

2.1.6 Tiến hóa

Với quần thể khởi tạo ngẫu nhiên ban đầu, chúng sẽ được tiến hóa và thích nghi với các điều kiện chọn lọc, các cá thể thích nghi kém sẽ bị loại bỏ và các cá thể tốt nhất sẽ được giữ lại chọn làm lời giải tốt nhất.

Việc tiến hóa diễn ra qua nhiều thế hệ. Mỗi thế hệ ta sẽ thực hiện các công việc như lai ghép và đột biến ngẫu nhiên trên toàn quần thể.

- Lai ghép: các cá thể cha mẹ được chọn lai ghép có chi phí đáp ứng một tiêu chí nhất định. Đảm bảo cho tập các cá thể cha mẹ được chọn lai ghép luôn là số chẵn. Sau đó sẽ lựa chọn ngẫu nhiên hai cá thể cha mẹ riêng biệt để lai ghép tạo ra các cá thể con
- Đột biến cho toàn bộ quần thể

Sau một số thế hệ định trước chương trình sẽ dừng lại và xuất ra kết quả về lời giải tốt nhất cho từng nhiệm vụ.

2.2 Chương trình

```
package com.ngm.mfea;

import java.util.ArrayList;

public class Main {
    public static void main(String[] args) {
        double[] totalTimes = new double[(int)Population.MAX_ITERATIONS];

        ArrayList<Task> taskList = new ArrayList<>();
        taskList.add(new Task("input\\berlin52.tsp"));
        taskList.add(new Task("input\\st70.tsp"));
        taskList.add(new Task("input\\bier127.tsp"));
        UnifiedMultitaskingEnvironment ume = new UnifiedMultitaskingEnvironment(taskList);
        for (int i = 0; i < Population.MAX_ITERATIONS; i++) {
            System.out.println("Lần #" + (i + 1) + ":");
            long startTime = System.nanoTime();
            ume.MFEAlgorithm();
            long endTime = System.nanoTime();
            long totalTime = endTime - startTime;
            totalTimes[i] = (double) totalTime / 1000000000;
            System.out.println("Total Time: " + totalTimes[i] + " second(s).");

            for (Task task : taskList) {
                ume.printBestSolution(task);

            }
        }
    }
}
```

Hình 2.7: Code hàm main của chương trình

Hàm main của chương trình có cấu trúc như hình. Người dùng sẽ đưa luôn các bộ dữ liệu chuẩn như đã đề cập trong mục mã hóa đồ thị để chương trình xử lý. Nếu file đầu vào không đúng định dạng thì chương trình sẽ không chạy.

Kết quả trả về sẽ bao gồm chu trình đường đi của lời giải tốt nhất và cụ thể chi phí cho đường đi đó là bao nhiêu của quần thể sau mỗi lần tiến hóa. Chương trình còn hiển thị thời gian chạy theo giây.

2.3 Kết quả chạy các bộ dữ liệu

2.3.1 Môi trường thực nghiệm

Máy tính chạy giải thuật:

- Dell Gaming G7
- CPU: Intel(R) Core (TM) i5-8300H CPU @ 2.30GHz 2.30 GHz
- RAM: 8GB
- Hệ điều hành: Windows 10 Pro 64 bit

Đánh giá: Môi trường máy tính trung bình nên ảnh hưởng tới tốc độ xử lý và truy xuất kết quả đầu ra.

2.3.2 Dữ liệu thực nghiệm

❖ Ý nghĩa thông số

NAME: Tên bộ dữ liệu

COMMENT: Thông tin về thông số của bộ dữ liệu, nơi nghiên cứu và người đóng góp.

TYPE: Tên bài toán (VD: TSP)

DIMENSION: Kích thước dữ liệu

EDGE_WEIGHT_TYPE: Loại trọng lượng cạnh (2D hoặc 3D)

❖ Bộ dữ liệu: berlin52.tsp

Bảng 2.1: Bộ dữ liệu berlin52.tsp

• NAME : berlin52 • TYPE: TSP • COMMENT : 52 locations in Berlin (Groetschel) • DIMENSION : 52 • EDGE_WEIGHT_TYPE: EUC_2D • NODE_COORD_SECTION		
1	565.0	575.0
2	25.0	185.0
3	345.0	750.0
…	…	…
EOF		

❖ Bộ dữ liệu: vm1084.tsp

Bảng 2.2: Bộ dữ liệu vm1084.tsp

• NAME : vm1084 • COMMENT : 1084-city problem (Reinelt) • TYPE: TSP • DIMENSION : 1084 • EDGE_WEIGHT_TYPE: EUC_2D • NODE_COORD_SECTION		
1	4.08000e+03	5.236000e+03

2	4.68800e+03	6.46800e+03
3	5.29600e+03	9.21800e+03
…	…	…
EOF		

❖ Bộ dữ liệu: st70.tsp

Bảng 2.3: Bộ dữ liệu st70.tsp

• NAME : st70 • TYPE: TSP • COMMENT : 70-city problem (Smith/Thompson) • DIMENSION : 70 • EDGE_WEIGHT_TYPE: EUC_2D • NODE_COORD_SECTION		
1	64	96
2	80	39
3	69	23
…	…	…
EOF		

❖ Bộ dữ liệu: lin318.tsp

Bảng 2.4: Bộ dữ liệu lin318.tsp

• NAME: lin318 • TYPE: TSP • COMMENT: 318-city problem (Lin/Kernighan) • DIMENSION: 318 • EDGE_WEIGHT_TYPE: EUC_2D • NODE_COORD_SECTION		
1	63	71
2	94	71
3	142	370

…	…	…
EOF		

❖ Bộ dữ liệu: bier127.tsp

Bảng 2.5: Bộ dữ liệu bier127.tsp

• NAME: bier127.tsp • COMMENT: 127 Biergaerten in Augsburg (Jueger/Reinelt) • TYPE: TSP • DIMENSION: 127 • EDGE_WEIGHT_TYPE: EUC_2D • NODE_COORD_SECTION		
1	9860	14152
2	9396	14616
3	11252	14848
…	…	…
EOF		

2.3.3 Kết quả chạy các bộ dữ liệu chuẩn

Dữ liệu trong mỗi bộ sẽ bao gồm thông tin về tên bộ dữ liệu, số đỉnh, danh sách đỉnh cùng với tọa độ mỗi đỉnh.

Chúng em lựa chọn kết hợp các bộ sau để thử nghiệm giải thuật:

Bảng 2.6: Tổng hợp kết quả chạy các bộ dữ liệu

Bộ kết hợp	Tên bộ dữ liệu	Số đỉnh	Chi phí của lời giải tối ưu đã tìm được
1	lin318.tsp	318	379652.9592874837
2	vm1084.tsp	1084	5615598.467350487
3	berlin52.tsp	52	12696.503764379411
4	st70.tsp	70	1734.0913005309856
5	bier127.tsp	127	387178.3939543789
6	lin318.tsp	318	169307.27657614453
	vm1084.tsp	1084	5824661.122539781

7	lin138.tsp	138	393267.0494292692
	berlin52.tsp	52	14460.22746909632
8	st70.tsp	70	2679.540255802813
	bier127.tsp	127	520313.5063664525
	berlin52.tsp	52	21695.2343075326

2.3.3.1 Bộ dữ liệu lin318.tsp

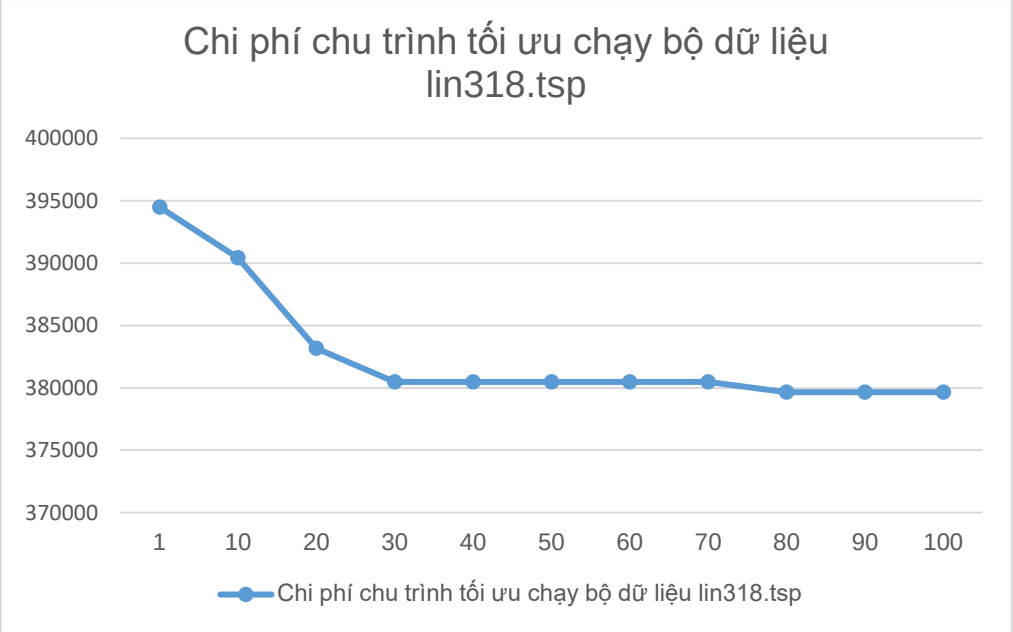
- Tên bộ dữ liệu: lin318.tsp (Lin/Kernighan)
- Kích thước đồ thị: 318 đỉnh
- Chi phí chu trình tối ưu: 42029

Kết quả chạy bộ dữ liệu lin318.tsp:

Bảng 2.7: Kết quả chạy bộ dữ liệu lin318.tsp

Pop Size	Max Gen	Min cost	Optimal found
200	10	471893.5280223076	Có
	100	**379652.9592874837**	Có

Biểu đồ 2.1: Chi phí chu trình tối ưu chạy bộ dữ liệu lin318.tsp (100 lần tiến hóa)



2.3.3.2 Bộ dữ liệu vm1084.tsp

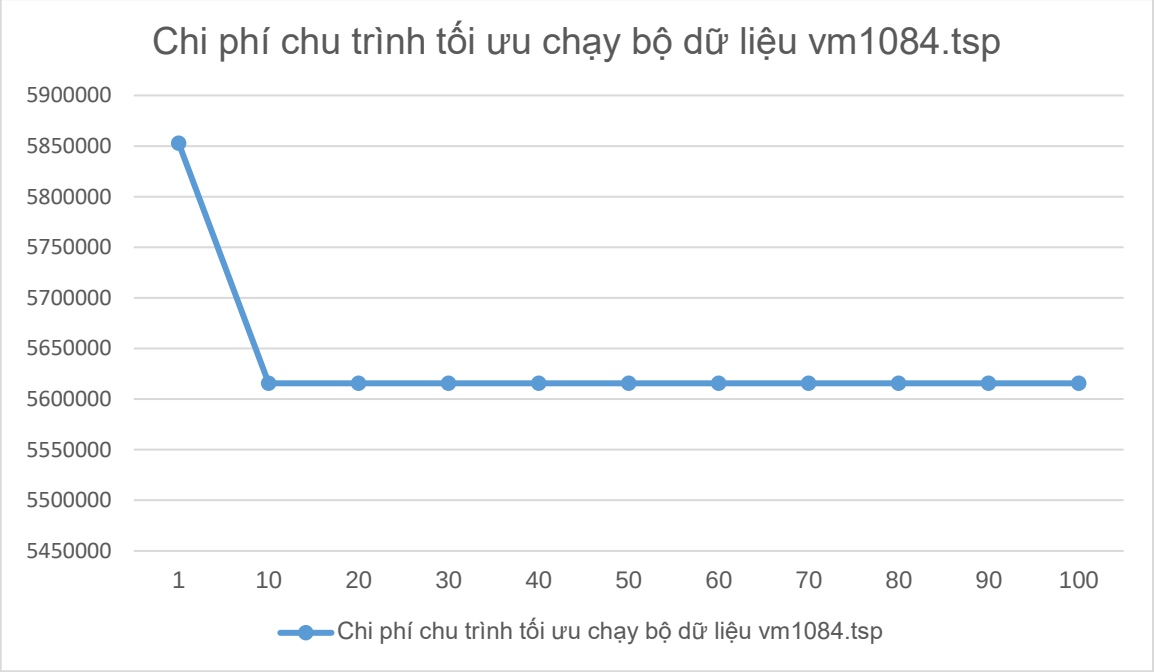
- Tên bộ dữ liệu: vm1084.tsp
- Kích thước đồ thị: 1084 đỉnh
- Chi phí chu trình tối ưu: 239297

Kết quả chạy bộ dữ liệu vm1084.tsp:

Bảng 2.8: Kết quả chạy bộ dữ liệu vm1084.tsp

Pop Size	Max Gen	Min cost	Optimal found
200	10	6413907.796103962	Có
	100	**5615598.467350487**	Có

Biểu đồ 2.2: Chi phí chu trình tối ưu chạy bộ dữ liệu vm1084.tsp (100 lần tiến hóa)



2.3.3.3 Bộ dữ liệu berlin52.tsp

- Tên bộ dữ liệu: berlin52.tsp
- Kích thước đồ thị: 52 đỉnh
- Chi phí chu trình tối ưu: 7544.3659

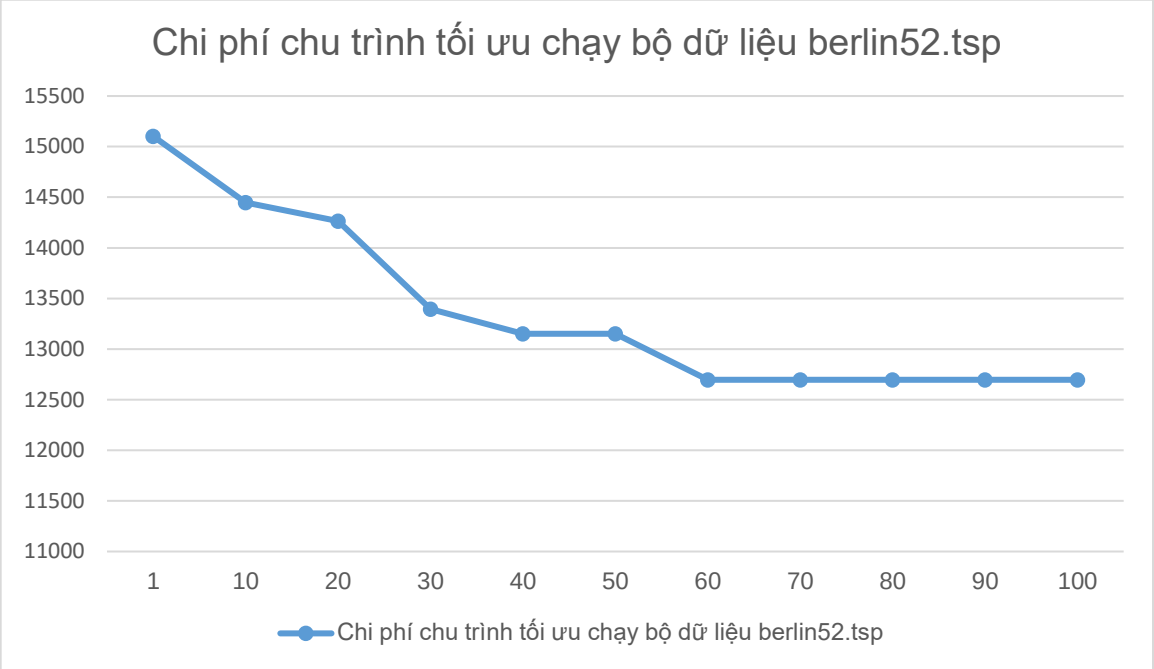
Kết quả chạy bộ dữ liệu berlin52.tsp:

Bảng 2.9: Kết quả bộ dữ liệu berlin52.tsp

Pop Size	Max Gen	Min cost	Optimal found
200	10	21110.76976792551	Có

	100	**12696.503764379411**	Có

Biểu đồ 2.3: Chi phí chu trình tối ưu chạy bộ dữ liệu berlin52.tsp (100 lần tiến hóa)



2.3.3.4 Bộ dữ liệu st70.tsp

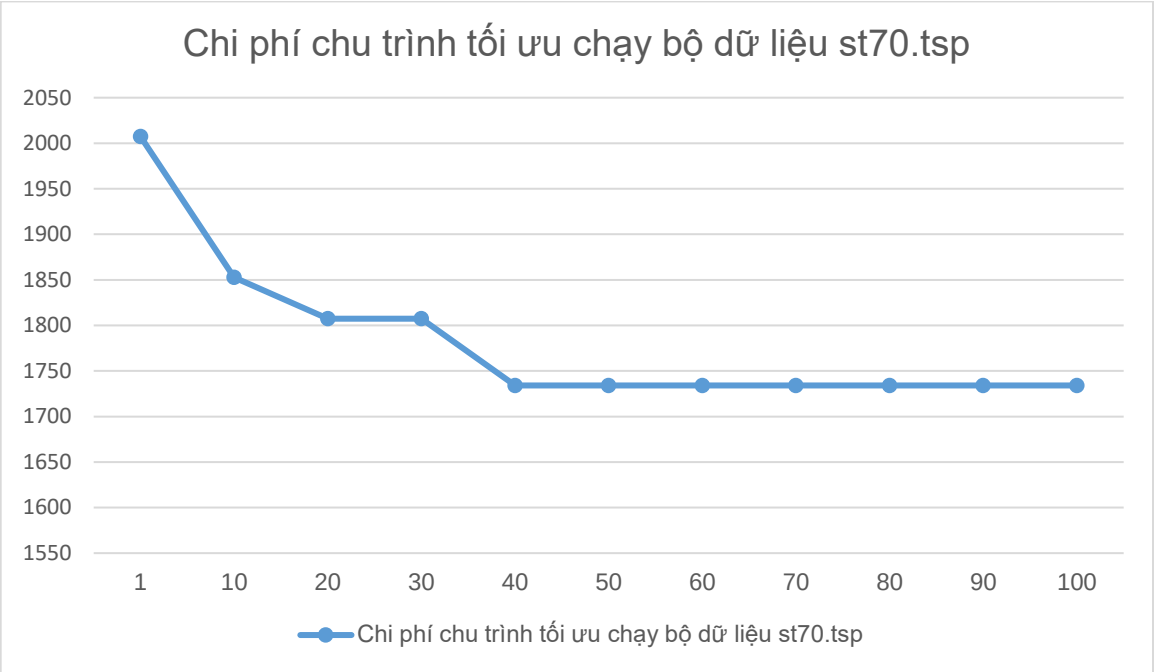
- Tên bộ dữ liệu: st70.tsp
- Kích thước đồ thị: 70 đỉnh
- Chi phí chu trình tối ưu: 675

Kết quả chạy bộ dữ liệu st70.tsp:

Bảng 2.10: Kết quả chạy bộ dữ liệu st70.tsp

Pop Size	Max Gen	Min cost	Optimal found
200	10	2782.5554710918973	Có
	100	**1734.0913005309856**	Có

Biểu đồ 2.4: Chi phí chu trình tối ưu chạy bộ dữ liệu st70.tsp (100 lần tiến hóa)



2.3.3.5 Bộ dữ liệu bier127.tsp

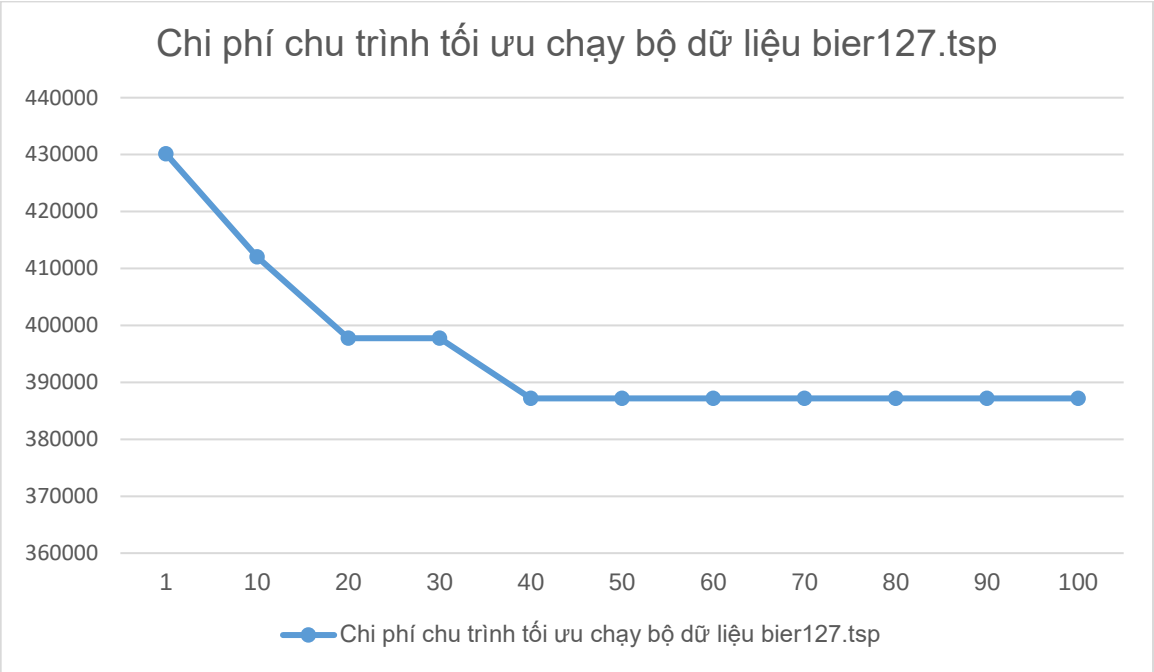
- Tên bộ dữ liệu: bier127.tsp
- Kích thước đồ thị: 127 đỉnh
- Chi phí chu trình tối ưu: 124651.524

Kết quả chạy bộ dữ liệu bier127.tsp:

Bảng 2.11: Kết quả chạy bộ dữ liệu bier127.tsp

Pop Size	Max Gen	Min cost	Optimal found
200	10	522747.6251442227	Có
	100	**387178.3939543789**	Có

Biểu đồ 2.5: Chi phí chu trình tối ưu chạy bộ dữ liệu bier127.tsp (100 lần tiến hóa)



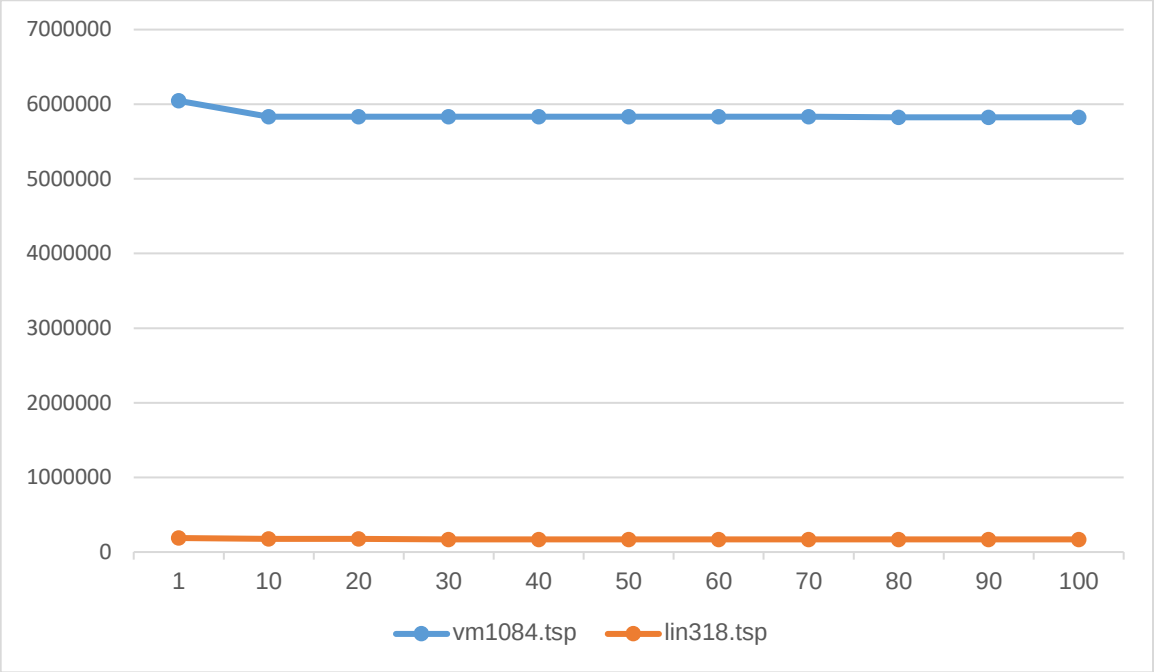
2.3.3.6 Bộ dữ liệu vm1084.tsp và lin318.tsp

Chạy chương trình với hai bộ dữ liệu vm1084.tsp và lin318.tsp cùng một lúc.

Bảng 2.12: Kết quả chạy bộ dữ liệu vm1084.tsp và lin318.tsp

Pop Size	**Max Gen**	**Task**	**Min cost**	**Optimal found**
200	10	vm1084	6429730.383152495	Có
		lin318	240158.6519685901	
	100	vm1084	**5824661.122539781**	Có
		lin318	**169307.27657614453**	

Biểu đồ 2.6: Chi phí chu trình tối ưu chạy bộ dữ liệu vm1084.tsp và lin318.tsp (100 lần tiến hóa)



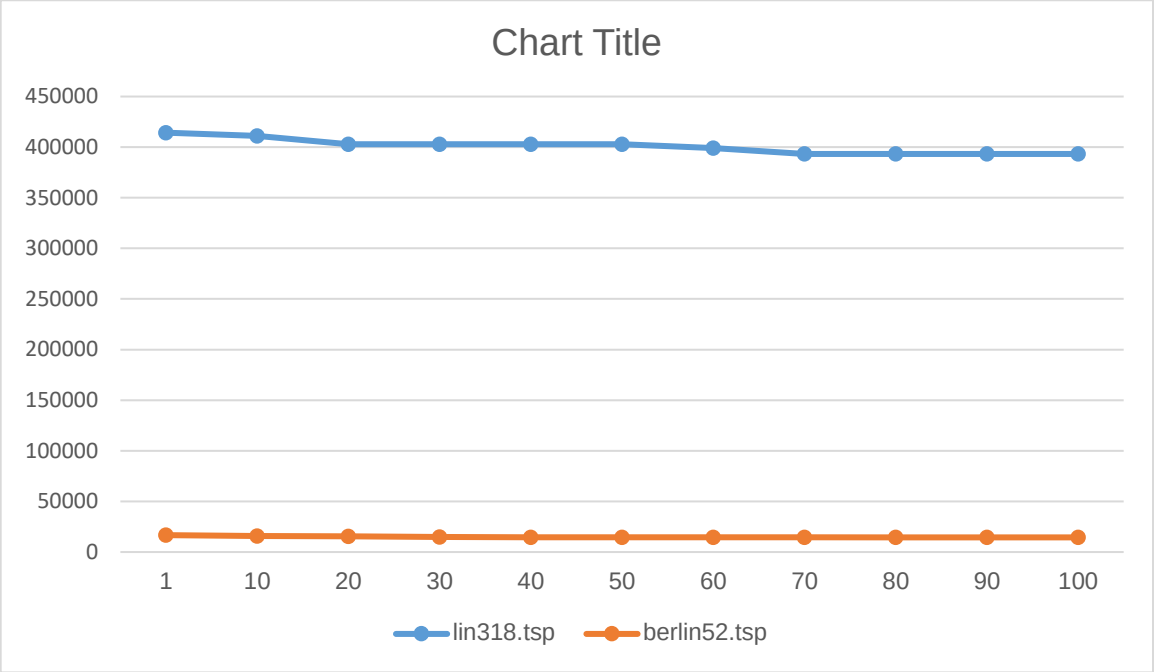
2.3.3.7 Bộ dữ liệu lin318.tsp và berlin52.tsp

Chạy chương trình với hai bộ dữ liệu vm1084.tsp và lin318.tsp cùng một lúc.

Bảng 2.13: Kết quả chạy bộ dữ liệu vm1084.tsp và lin318.tsp

Pop Size	**Max Gen**	**Task**	**Min cost**	**Optimal found**
200	10	berlin52	21577.744445523374	Có
		lin318	469504.3406587931	
	100	berlin52	**14460.22746909632**	Có
		lin318	**393267.0494292692**	

Biểu đồ 2.7: Chi phí chu trình tối ưu chạy bộ dữ liệu berlin52.tsp và lin318.tsp (100 lần tiến hóa)



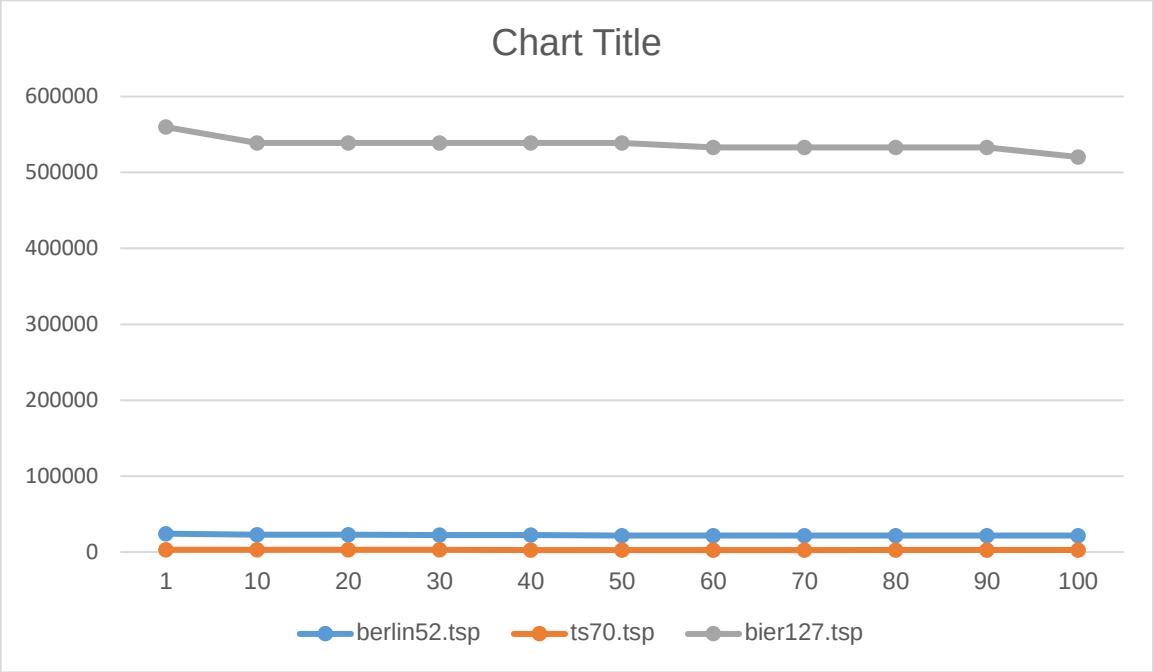
2.3.3.8 Bộ dữ liệu st70.tsp, bier127.tsp và berlin52.tsp

Chạy chương trình với ba bộ dữ liệu st70.tsp, bier127.tsp và berlin52.tsp cùng một lúc.

Bảng 2.14: Kết quả chạy bộ dữ liệu vm1084.tsp và lin318.tsp

Pop Size	**Max Gen**	**Task**	**Min cost**	**Optimal found**
200	10	berlin52	21310.771708086297	Có
		bier127	539494.4484143468	
		st70	3031.361530176967	
	100	berlin52	**21695.2343075326**	Có
		bier127	**520313.5063664525**	
		st70	**2679.540255802813**	

Biểu đồ 2.8: Chi phí chu trình tối ưu chạy bộ dữ liệu berlin52.tsp, st70.tsp và bier127.tsp (100 lần tiến hóa)



2.4 Đánh giá giải thuật và dự đoán hướng đi trong tương lai

Giải thuật tiến hóa đa nhiệm vụ MFEA đã đáp ứng cơ bản các bước và chạy ra kết quả. Kết quả sau chạy chương trình cho thấy kết quả tối ưu trong các trường hợp tiến hóa 100 lần "tối ưu" hơn kết quả cho ra trong trường hợp tiến hóa 10 lần.

Giải thuật tiến hóa đa nhiệm vụ MFEA là một giải pháp tốt và hiệu quả cho vấn đề đa tác vụ MFO. Trong vấn đề này, em đang tập trung đi xử lý cùng một bài toán người du lịch với hai hoặc nhiều bộ dữ liệu khác nhau cùng một lúc. Hay nói cách khác là tổ chức chung một quần thể lời giải cho nhiều tác vụ khác nhau. Trong tương lai, em sẽ tìm hiểu và cài đặt giải thuật MFEA cho nhiều bài toán khác nhau như bài toán người du lịch (TSP), bài toán cái túi (Knapsack), …

KẾT LUẬN VÀ KIẾN NGHỊ

Kết quả đạt được

Về mặt lý thuyết, bài báo cáo đã trình bày được các nội dung sau:

- Tìm hiểu về bài toán tối ưu, bài toán Người du lịch TSP.
- Tìm hiểu về tiến hóa, thuật toán di truyền, giải thuật tiến hóa đa nhiệm vụ MFEA.
- Áp dụng thuật toán MFEA vào giải bài toán Người du lịch với nhiều bộ dữ liệu khác nhau cùng một lúc.

Về mặt thực nghiệm, bài báo cáo đã thu được một số kết quả:

- Cài đặt thành công giải thuật tiến hóa đa nhiệm vụ MFEA.
- Thực nghiệm giải thuật với các tham số và đưa ra đánh giá, nhận xét.

Tuy nhiên, do còn nhiều hạn chế về kiến thức, kinh nghiệm của bản thân và thời gian thực nghiệm, bài báo cáo còn một số thiếu sót sau:

- Thực nghiệm còn chưa đầy đủ
- Bài báo cáo dừng ở mức tiếp cận vấn đề, chưa đi sâu giải quyết vấn đề.

Hướng phát triển trong tương lai

Bài báo cáo đã mang đến cách tiếp cận gần hơn với phương pháp giải quyết vấn đề đa nhiệm vụ MFO bằng giải thuật MFEA. Vì vậy trong tương lai, tác giả sẽ tiếp tục phát triển, đi sâu vào việc thử nghiệm với nhiều bài toán tối ưu khác và đồng thời, lên ý tưởng về việc giải quyết nhiều bài toán có hàm mục tiêu khác nhau để tạo ra một cách giải quyết cho yêu cầu giải đồng thời nhiều bài toán, nhiều tác vụ cùng một lúc của khoa học máy tính hiện đại.

TÀI LIỆU THAM KHẢO

Tài liệu Tiếng Việt

[1] Nguyễn Đình Thúc, 2001. *Trí tuệ nhân tạo-Lập trình tiến hóa*, NXB Giáo Dục.

[2] Nguyễn Thanh Thủy, 1996. *Trí tuệ nhân tạo: Các phương pháp giải quyết vấn đề*, Nxb giáo dục.

[3] Lại Thị Nhung, Nguyễn Thị Hòa, Phạm Văn Hạnh, Lê Đăng Nguyên, Lê Trọng Vĩnh, 2018. *ĐA TÁC VỤ TIẾN HÓA: KỸ THUẬT TỐI ƯU HÓA MỚI.*

[4] Các tài liệu liên quan và bộ dữ liệu mẫu tại: http://www.tsp.gatech.edu/

Tài liệu Tiếng Anh

[1] Abhishek Gupta, Yew-Soon Ong*, and Liang Feng, 2007. *Multifactorial Evolution: Towards Evolutionary Multitasking*.

[2] Mitchell Melanie, 1999, *An Introduction to Genetic Algorithms*. A Bradford Book The MIT Press.

[3] ∗ Kavitesh Kumar Bali, ∗ Yew-Soon Ong, † Abhishek Gupta, † Puay Siew Tan ∗ School of Computing Science and Engineering, Nanyang Technological University, Singapore. † Singapore Institute of Manufacturing Technology (SIMTech), A*STAR, Singapore, 2019. *Multifactorial Evolutionary Algorithm with Online Transfer Parameter Estimation: MFEA-II*.