BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM THÀNH PHỐ HỒ CHÍ MINH

SỐ PHÁCH

Phan Minh Tiến

KHMT835030

LẬP TRÌNH DI TRUYỀN TRONG HỌC MÁY
THIẾT KẾ VÀ ĐÁNH GIÁ MÔ HÌNH
TRÊN BÀI TOÁN PHÂN LOẠI IRIS

HỌC PHẦN: CÁC HỆ CƠ SỞ TRI THỨC

NGÀNH: KHOA HỌC MÁY TÍNH

KHÓA: 35 (2024-2026)

GIẢNG VIÊN GIẢNG DẠY: PGS.TS. LÊ HOÀNG THÁI

Thành phố Hồ Chí Minh, ngày 05 tháng 05 năm 2025

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐẠI HỌC SƯ PHẠM THÀNH PHỐ HỒ CHÍ MINH

TỔNG ĐIỂM	**Cán bộ chấm thi 1** *(Ký, ghi rõ họ tên)*	**Cán bộ chấm thi 2** *(Ký, ghi rõ họ tên)*	**SỐ PHÁCH**

LẬP TRÌNH DI TRUYỀN TRONG HỌC MÁY
THIẾT KẾ VÀ ĐÁNH GIÁ MÔ HÌNH
TRÊN BÀI TOÁN PHÂN LOẠI IRIS

HỌC PHẦN: CÁC HỆ CƠ SỞ TRI THỨC

NGÀNH: KHOA HỌC MÁY TÍNH

KHÓA: 35 (2024-2026)

GIẢNG VIÊN GIẢNG DẠY: PGS.TS. LÊ HOÀNG THÁI

Thành phố Hồ Chí Minh, ngày 05 tháng 05 năm 2025

MỤC LỤC

MỤC LỤC HÌNH ẢNH

LỜI CẢM ƠN

Trước tiên, em xin bày tỏ lòng biết ơn đến PGS. TS. Lê Hoàng Thái, giảng viên giảng dạy học phần Các Hệ Cơ Sở Tri Thức, thầy đã tận tình hướng dẫn, truyền đạt kiến thức và định hướng nghiên cứu trong suốt quá trình học tập và thực hiện đề tài. Những đóng góp chuyên môn và sự hỗ trợ tận tâm từ thầy là nguồn động lực quý báu giúp em hoàn thành tốt phần việc được phân công.

Em cũng xin chân thành cảm ơn Trường Đại học Sư phạm Thành phố Hồ Chí Minh đã tạo điều kiện thuận lợi về môi trường học thuật, cơ sở vật chất và tài liệu tham khảo, giúp tôi có cơ hội tiếp cận, nghiên cứu và triển khai các thuật toán tiên tiến một cách hiệu quả.

Mặc dù đã nỗ lực hết mình, nhưng do giới hạn về thời gian và năng lực, bài làm cá nhân của em chắc chắn vẫn còn những thiếu sót. Rất mong nhận được sự góp ý từ quý thầy cô để hoàn thiện hơn trong những lần nghiên cứu sau.

Trân trọng cảm ơn.

CHƯƠNG I. MỞ ĐẦU

1. Lý do chọn đề tài

Trong bối cảnh hiện nay, học máy và trí tuệ nhân tạo đang đóng vai trò quan trọng trong việc giải quyết nhiều bài toán thực tiễn, từ y tế, tài chính cho đến các lĩnh vực kỹ thuật cao như thiết kế anten và phân loại dữ liệu. Một trong những thách thức lớn trong học máy là việc thiết kế kiến trúc mô hình tối ưu và khả năng diễn giải của mô hình đối với người sử dụng.

Lập trình di truyền (Genetic Programming – GP) nổi lên như một phương pháp tiến hóa mạnh mẽ, mô phỏng quá trình chọn lọc tự nhiên để tự động tạo ra các chương trình máy tính tối ưu. Khác với nhiều mô hình hộp đen trong học máy, GP không chỉ cung cấp kết quả dự đoán mà còn cho phép sinh ra các biểu thức có thể diễn giải, từ đó giúp người dùng hiểu rõ hơn về cách mô hình đưa ra quyết định.

Với mong muốn tìm hiểu sâu hơn về khả năng ứng dụng thực tiễn của GP, đề tài này tập trung vào khảo sát và áp dụng GP để xây dựng các mô hình phân loại dữ liệu, cụ thể là tập dữ liệu nổi tiếng Iris – một bài toán kinh điển trong học máy..

2. Mục tiêu nghiên cứu

Tìm hiểu lý thuyết nền tảng và các đặc điểm nổi bật của lập trình di truyền (GP).

Triển khai GP để tiến hóa kiến trúc mô hình phân loại tự động trên tập dữ liệu Iris.

Đánh giá hiệu quả của GP trong việc tạo ra các mô hình phân loại có khả năng diễn giải và so sánh với các phương pháp truyền thống.

3. Phạm vi nghiên cứu

Nghiên cứu lý thuyết về lập trình di truyền trong khuôn khổ các bài toán học máy.

Tập trung triển khai và thử nghiệm GP trên một tập dữ liệu chuẩn là Iris, sử dụng các hàm số học cơ bản và các phép điều kiện đơn giản.

4. Phương pháp nghiên cứu

Nghiên cứu lý thuyết: Khảo sát các tài liệu học thuật, công trình nghiên cứu về GP và các phương pháp tiến hóa.

Thực nghiệm: Cài đặt thuật toán GP với cấu hình tùy chỉnh, huấn luyện mô hình trên tập dữ liệu Iris và đánh giá kết quả.

So sánh và phân tích: Đánh giá hiệu quả mô hình dựa trên độ chính xác, khả năng diễn giải và chi phí tính toán, từ đó rút ra ưu – nhược điểm của phương pháp GP.

CHƯƠNG II. CƠ SỞ LÝ THUYẾT

1. Khái quát lập trình di truyền

Lập trình di truyền (Genetic Programming – GP) là một phương pháp học máy bắt nguồn từ cảm hứng sinh học, mô phỏng quá trình tiến hóa và chọn lọc tự nhiên nhằm tự động tạo ra các chương trình máy tính thực hiện một nhiệm vụ được xác định trước. Cơ chế hoạt động của GP tương tự quá trình sinh học: bắt đầu từ một quần thể chương trình ngẫu nhiên, các cá thể sẽ được đánh giá dựa trên mức độ hoàn thành nhiệm vụ (fitness), sau đó trải qua các phép lai ghép (crossover), đột biến (mutation) và chọn lọc để dần tiến hóa thành các lời giải tối ưu hơn theo từng thế hệ.

Trong thời gian gần đây, GP đã được ứng dụng hiệu quả vào các bài toán phân loại trong nhiều lĩnh vực đa dạng như nhận dạng ký tự, phân tích sinh học, tài chính và y học. Một trong những minh chứng nổi bật cho tiềm năng của GP là khả năng thiết kế các kiến trúc anten dạng dây có hiệu suất cao, mang tính phi trực giác, và trong một số trường hợp đã đạt được kết quả tương đương hoặc vượt qua các thiết kế đã được cấp bằng sáng chế – ngay cả khi không có sẵn quy trình thiết kế tiêu chuẩn.

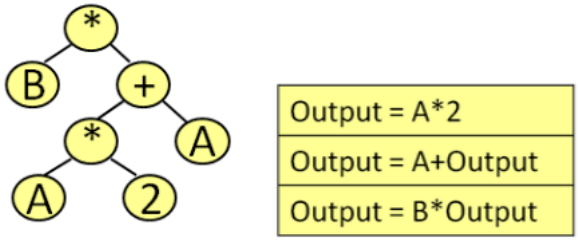
Tuy nhiên, phạm vi ứng dụng hiện tại của GP trong thiết kế cấu trúc điện từ vẫn chủ yếu giới hạn ở các hình thái đơn giản như cấu trúc dây. Điều này phần nào phản ánh sự phức tạp của bài toán và yêu cầu cao về khả năng tổng quát hóa của mô hình.

Đối với con người, việc giải quyết những bài toán này thường đòi hỏi kiến thức chuyên sâu trong lĩnh vực liên quan, cùng với nhiều năm kinh nghiệm tích lũy để đưa ra một giải pháp có ý nghĩa. Ngược lại, GP – với khả năng tự động hóa quá trình học và khám phá – không những giúp rút ngắn thời gian tiếp cận mà còn cung cấp một chương trình cụ thể để hoàn thành nhiệm vụ, đồng thời hé lộ những hiểu biết có giá trị về cấu trúc của lời giải.

2. Lập trình di truyền và ứng dụng trong học máy

Lập trình di truyền (Genetic Programming – GP) là một nhánh của học máy, khai thác cơ chế tiến hóa tự nhiên theo thuyết Darwin để phát triển tự động các chương trình máy tính có khả năng thực hiện một nhiệm vụ cụ thể. Trong GP, mỗi chương trình máy tính được xem như một cá thể trong quần thể, mang cấu trúc di truyền riêng biệt – tương tự như bộ gen ở sinh vật – quy định hành vi và hiệu quả thực hiện nhiệm vụ của chương trình đó.

Về mặt biểu diễn, nhiễm sắc thể của một chương trình trong GP có thể được mã hóa dưới dạng cây cú pháp (tree-based GP) hoặc chuỗi lệnh tuyến tính (linear GP). Trong đó, các nút bên trong thường là tập hợp các hàm toán học hoặc logic (ví dụ: +, ×, so sánh, điều kiện...), còn các nút lá là các đầu vào hoặc hằng số – được gọi chung là tập điểm cuối (terminals). Hình dưới minh họa hai biểu diễn khác nhau của cùng một nhiễm sắc thể thực hiện công thức số học đơn giản:



Hình 1. Ví dụ của GP structure

B là các biến đầu vào. Trong cấu trúc cây, mỗi nhánh đại diện cho một gen, còn toàn bộ cây biểu thị nhiễm sắc thể; tương tự, trong GP tuyến tính, mỗi dòng lệnh là một gen cấu thành chuỗi nhiễm sắc thể.

Mỗi chương trình máy tính trong quần thể sẽ trải qua quá trình đánh giá độ phù hợp (fitness evaluation), trong đó kết quả đầu ra của chương trình được so sánh với giá trị mục tiêu của bài toán. Một hàm độ phù hợp – do người dùng xác định – được sử dụng để tính toán điểm số cho từng cá thể. Các cá thể có điểm số cao nhất sẽ được chọn lọc để tham gia vào các phép toán di truyền như sinh sản (reproduction), lai chéo (crossover) và đột biến (mutation), qua đó tạo ra thế hệ kế tiếp. Quy trình này được lặp lại cho đến khi đạt tiêu chí dừng nhất định.

Trong GP, phép sinh sản đơn thuần sao chép chương trình cha mẹ sang thế hệ sau mà không thay đổi cấu trúc, đảm bảo duy trì các lời giải tốt. Lai chéo kết hợp các đoạn nhiễm sắc thể từ hai cá thể để tạo ra cá thể mới với sự pha trộn đặc điểm di truyền. Ngược lại, đột biến đóng vai trò tạo ra sự đa dạng di truyền bằng cách thay đổi ngẫu nhiên một phần của nhiễm sắc thể, cho phép khám phá các vùng lời giải mới ngoài phạm vi tiến hóa hiện có.

Một điểm khác biệt then chốt giữa GP và thuật toán di truyền cổ điển (Genetic Algorithm – GA) nằm ở cách biểu diễn lời giải. GA hoạt động trên các vector có độ dài cố định, phù hợp với các bài toán tối ưu hóa tham số có cấu trúc xác định. Ngược lại, GP cho phép kích thước và cấu trúc của lời giải tiến hóa linh hoạt, dẫn đến khả năng khám phá

không gian lời giải phong phú hơn. Chẳng hạn, trong bài toán tối ưu hóa thiết kế anten Yagi-Uda, GA chỉ có thể tối ưu các khoảng cách giữa các phần tử trong một cấu trúc cố định, trong khi GP có thể tiến hóa toàn bộ thiết kế anten, thậm chí tạo ra cấu trúc hoàn toàn mới phù hợp hơn với yêu cầu đầu ra.

Khả năng thoát khỏi cực tiểu cục bộ và tìm kiếm giải pháp tối ưu toàn cục là một trong những thế mạnh nổi bật của GP và GA so với các phương pháp học máy khác như mạng nơ-ron nhân tạo (ANN). Nhờ vào đặc tính ngẫu nhiên và cơ chế đột biến, GP có thể thực hiện những "bước nhảy" vượt ra khỏi ràng buộc cấu trúc ban đầu để phát hiện các lời giải sáng tạo và hiệu quả hơn.

Trong những năm gần đây, GP đã được ứng dụng thành công trong nhiều lĩnh vực như: chẩn đoán y tế, phân tích tài chính, phân loại văn bản, thiết kế anten dây, thiết kế mạch điện tử, và nhiều bài toán tối ưu hóa phức tạp khác. Trong khuôn khổ luận văn này, GP được nghiên cứu và triển khai cho hai bài toán chính: (1) phân loại mục tiêu vật thể nổ chưa phát nổ (UXO) sử dụng dữ liệu radar xuyên đất, và (2) thiết kế các cấu trúc siêu vật liệu có băng thông rộng tối ưu. Thông qua hai ứng dụng này, tiềm năng của GP trong việc giải quyết các bài toán khó học, nơi mà phương pháp học máy truyền thống còn hạn chế, sẽ được làm rõ và phân tích chuyên sâu.

3. So sánh lập trình di truyền và các phương pháp khác

Lập trình di truyền là một phương pháp giải quyết vấn đề độc lập với lĩnh vực cụ thể, trong đó các chương trình máy tính được tiến hóa để tìm ra giải pháp. Kỹ thuật này dựa trên nguyên tắc "chọn lọc tự nhiên" của Darwin và có liên quan chặt chẽ với thuật toán di truyền (Genetic Algorithm - GA). Tuy nhiên, có ba điểm khác biệt quan trọng giữa GA và GP:

- Cấu trúc: GA sử dụng chuỗi mã hóa cố định (chuỗi nhị phân hoặc số thực), trong khi GP sử dụng cấu trúc dạng cây. Điều này cho phép GP biểu diễn các chương trình phức tạp hơn, bao gồm các hàm lồng nhau và các biểu thức toán học phức tạp.
- Cơ chế tiến hóa: GA chủ yếu dựa trên toán tử di truyền để điều chỉnh các giá trị trong chuỗi mã hóa, trong khi GP có thể tạo ra và thay đổi cấu trúc chương trình bằng cách lai ghép hoặc đột biến các nhánh cây. Nhờ đó, GP có khả năng biểu diễn các thuật toán và chiến lược phức tạp hơn mà không cần xác định trước cấu trúc của chúng.
- Độ linh hoạt: Trong GA, độ dài chuỗi mã hóa thường cố định, hạn chế khả năng biểu diễn những giải pháp có kích thước thay đổi. GP có thể tạo ra các chương trình có

độ dài thay đổi, giúp hệ thống thích nghi tốt hơn với các bài toán cần tối ưu hóa cấu trúc chương trình.

Ngoài ba điểm chính này, một lợi thế quan trọng khác của GP là khả năng tìm ra các chương trình có thể được thực thi trực tiếp mà không cần quá trình giải mã như trong GA. Điều này giúp GP dễ dàng áp dụng vào các bài toán liên quan đến tự động hóa lập trình, tối ưu hóa thuật toán, và trí tuệ nhân tạo.

4. Các thành phần cơ bản của lập trình di truyền

Mỗi lời giải do GP tiến hóa được xây dựng từ hai tập hợp thành phần cơ bản:

- Tập hợp đầu cuối (Terminal set): Đây là các biến đầu vào hoặc hằng số, đóng vai trò như dữ liệu cung cấp cho chương trình. Tập hợp đầu cuối có thể bao gồm biến số, tham số hoặc dữ liệu thô cần xử lý.
- Tập hợp hàm số (Function set): Bao gồm các toán tử và hàm số được sử dụng để thao tác trên các biến đầu cuối. Các hàm số có thể là các toán tử số học (+, -, *, /), toán tử logic (AND, OR, NOT) hoặc các hàm điều kiện như IF-THEN-ELSE.

Lập trình di truyền sử dụng ba toán tử tiến hóa chính để biến đổi chương trình qua các thế hệ:

- Sao chép (Reproduction): Lựa chọn một cá thể trong quần thể và sao chép nó vào thế hệ tiếp theo mà không thay đổi.
- Lai ghép (Crossover): Kết hợp hai chương trình cha mẹ bằng cách trao đổi một phần của cây chương trình để tạo ra con cái. Điều này giúp chương trình con thừa hưởng đặc điểm từ cả hai cha mẹ, tạo ra sự đa dạng trong quần thể.
- Đột biến (Mutation): Thay đổi ngẫu nhiên một phần của chương trình, chẳng hạn như thay thế một toán tử hoặc một biến số bằng một giá trị khác trong tập hợp có sẵn. Điều này giúp duy trì sự đa dạng di truyền và tránh bị mắc kẹt trong cực tiểu cục bộ.

Các chương trình trong lập trình di truyền thường được biểu diễn dưới dạng cây, trong đó các nút lá là các đầu cuối (Terminal), còn các nút trong là các hàm số (Function). Việc sử dụng cấu trúc cây giúp chương trình có thể thay đổi kích thước và cấu trúc một cách linh hoạt trong quá trình tiến hóa.

5. Thuộc tính của lập trình di truyền

Lập trình di truyền có 16 thuộc tính quan trọng, còn được gọi là lập trình tự động, tổng hợp chương trình hoặc suy diễn chương trình. Những thuộc tính này giúp hệ thống có thể tạo ra các chương trình máy tính một cách tự động từ mô tả cấp cao của bài toán.

Dưới đây là danh sách 16 thuộc tính chính của lập trình di truyền:

1. Bắt đầu từ câu hỏi "Cần làm gì?" - Hệ thống khởi đầu từ một mô tả cấp cao về yêu cầu của bài toán.
2. Xác định "Làm thế nào để thực hiện" - Kết quả được tạo ra dưới dạng một chuỗi các bước có thể thực thi trên máy tính.
3. Tạo ra chương trình máy tính - Đầu ra của quá trình tiến hóa là một thực thể có thể chạy trên máy tính.
4. Xác định tự động kích thước chương trình - Hệ thống có thể xác định số bước cần thực hiện mà không cần người dùng định trước kích thước giải pháp.
5. Tái sử dụng mã lệnh - Tự động tổ chức các nhóm bước có ích để tái sử dụng.
6. Tái sử dụng có tham số - Có thể tái sử dụng nhóm bước với các giá trị tham số khác nhau.
7. Lưu trữ nội bộ - Có khả năng sử dụng bộ nhớ nội bộ như biến đơn, vectơ, ma trận, danh sách, hàng đợi...
8. Lặp, vòng lặp và đệ quy - Có thể thực hiện các phép lặp, vòng lặp và đệ quy.
9. Tổ chức thứ bậc một cách tự động - Có thể tổ chức các nhóm bước thành một cấu trúc thứ bậc.
10. Xác định tự động kiến trúc chương trình - Hệ thống có thể tự quyết định việc sử dụng hàm con, vòng lặp, đệ quy và số lượng tham số.
11. Hỗ trợ nhiều cấu trúc lập trình - Có thể mô phỏng các cấu trúc lập trình phổ biến như macro, thư viện, kiểu dữ liệu, con trỏ, điều kiện, toán tử logic, số nguyên, số thực...
12. Xác định rõ ràng nhiệm vụ của hệ thống - Hệ thống hoạt động một cách có quy tắc, phân biệt rõ ràng giữa đầu vào của người dùng và đầu ra của hệ thống.
13. Không phụ thuộc vào bài toán cụ thể - Hệ thống có thể giải quyết nhiều bài toán khác nhau mà không cần thay đổi quy trình thực thi.
14. Ứng dụng rộng rãi - Có thể áp dụng để giải quyết nhiều loại bài toán khác nhau trong các lĩnh vực đa dạng.
15. Khả năng mở rộng - Có thể mở rộng để giải quyết các bài toán có quy mô lớn hơn.

16. Kết quả có thể so sánh với con người - Hệ thống có thể tạo ra kết quả cạnh tranh với những gì do con người lập trình thủ công.

Những thuộc tính này giúp lập trình di truyền trở thành một phương pháp mạnh mẽ để tự động tạo ra các chương trình máy tính, cung cấp một cách tiếp cận mới trong việc giải quyết các bài toán phức tạp. Lập trình di truyền có một số thuộc tính quan trọng giúp nó trở thành một phương pháp hiệu quả trong việc tìm kiếm giải pháp tối ưu:

- Tính tổng quát (Generality): Các chương trình được tạo ra có thể áp dụng cho nhiều bài toán khác nhau mà không cần thay đổi đáng kể cấu trúc. Điều này giúp GP trở thành một công cụ mạnh mẽ trong nhiều lĩnh vực như trí tuệ nhân tạo, tối ưu hóa, mô phỏng hệ thống phức tạp và tự động hóa quy trình.
- Tính tự động (Automation): GP có thể tự động khám phá, tối ưu hóa và tinh chỉnh chương trình mà không cần sự can thiệp của con người. Điều này đặc biệt hữu ích trong các lĩnh vực mà không có thuật toán giải quyết sẵn có hoặc nơi mà các thuật toán truyền thống gặp khó khăn.
- Tính mở rộng (Scalability): GP có thể xử lý các bài toán từ đơn giản đến phức tạp bằng cách mở rộng tập hợp hàm số và đầu cuối. Nhờ vào khả năng biểu diễn chương trình dưới dạng cây, GP có thể mở rộng hoặc thu gọn kích thước chương trình một cách linh hoạt theo yêu cầu của bài toán.
- Tính mạnh mẽ (Robustness): Các chương trình tiến hóa có thể thích nghi với các thay đổi của bài toán và vẫn hoạt động hiệu quả. GP không bị giới hạn bởi các giả định cứng nhắc về mô hình dữ liệu hay không gian tìm kiếm, giúp nó có thể làm việc tốt với dữ liệu nhiễu hoặc môi trường không ổn định.
- Tính tái sử dụng (Reusability): Do cấu trúc GP dựa trên các hàm và toán tử, các thành phần của một chương trình tiến hóa có thể được tái sử dụng trong nhiều bài toán khác nhau. Điều này giúp giảm thời gian phát triển và tối ưu hóa hiệu suất của GP.
- Tính khả giải thích (Interpretability): Không giống như các mô hình hộp đen trong học máy, GP tạo ra các chương trình có thể được đọc và phân tích bởi con người. Điều này rất quan trọng trong các lĩnh vực như y tế, tài chính và khoa học, nơi mà sự minh bạch trong quá trình ra quyết định là cần thiết.

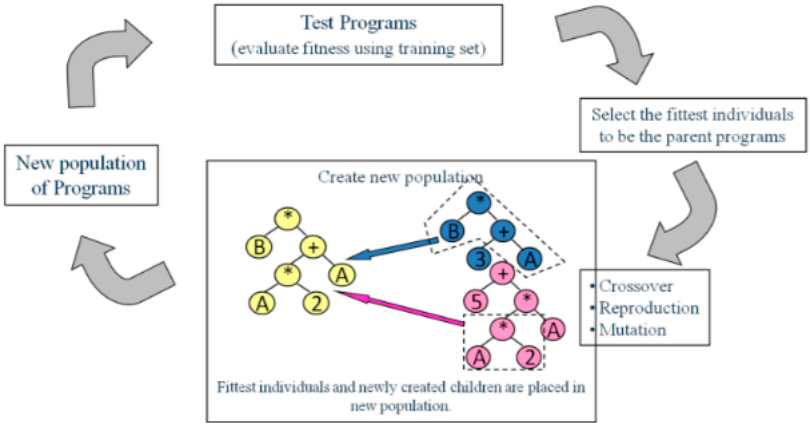
Các thuộc tính này giúp lập trình di truyền trở thành một công cụ hữu ích trong nhiều lĩnh vực, từ trí tuệ nhân tạo đến tối ưu hóa hệ thống phức tạp, giúp giải quyết các bài toán mà các phương pháp truyền thống có thể gặp khó khăn. Lập trình di truyền có một số thuộc

tính quan trọng giúp nó trở thành một phương pháp hiệu quả trong việc tìm kiếm giải pháp tối ưu:

- Tính tổng quát (Generality): Các chương trình được tạo ra có thể áp dụng cho nhiều bài toán khác nhau mà không cần thay đổi đáng kể cấu trúc.
- Tính tự động (Automation): GP có thể tự động khám phá và tinh chỉnh chương trình mà không cần sự can thiệp của con người.
- Tính mở rộng (Scalability): GP có thể xử lý các bài toán từ đơn giản đến phức tạp bằng cách mở rộng tập hợp hàm số và đầu cuối.
- Tính mạnh mẽ (Robustness): Các chương trình tiến hóa có thể thích nghi với các thay đổi của bài toán và vẫn hoạt động hiệu quả.

Các thuộc tính này giúp lập trình di truyền trở thành một công cụ hữu ích trong nhiều lĩnh vực, từ trí tuệ nhân tạo đến tối ưu hóa hệ thống phức tạp.

6. Thuật toán lập trình di truyền



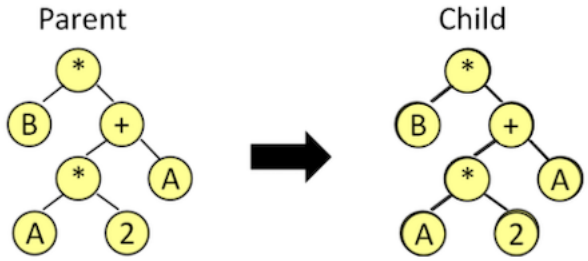
Hình 2. Thuật toán lập trình di truyền

Thuật toán lập trình di truyền (Genetic Programming – GP) khởi đầu với một quần thể ban đầu gồm các chương trình máy tính, trong đó mỗi chương trình mang một nhiễm sắc thể – tức mã chương trình – được tạo thành bằng cách kết hợp các biến đầu vào, hằng số và tập các hàm được định nghĩa trước. Số lượng chương trình trong quần thể khởi tạo cũng như danh sách các đầu vào, hằng số và hàm khả dụng đều do người dùng thiết lập.

Sau khi được khởi tạo, từng chương trình trong quần thể sẽ trải qua giai đoạn đánh giá độ phù hợp (fitness evaluation), trong đó đầu ra của chương trình được so sánh với đầu ra mong muốn của bài toán. Một hàm đánh giá độ phù hợp hoặc chi phí (fitness/cost

function), do người dùng cung cấp, sẽ được sử dụng để gán điểm số độ phù hợp cho từng cá thể.

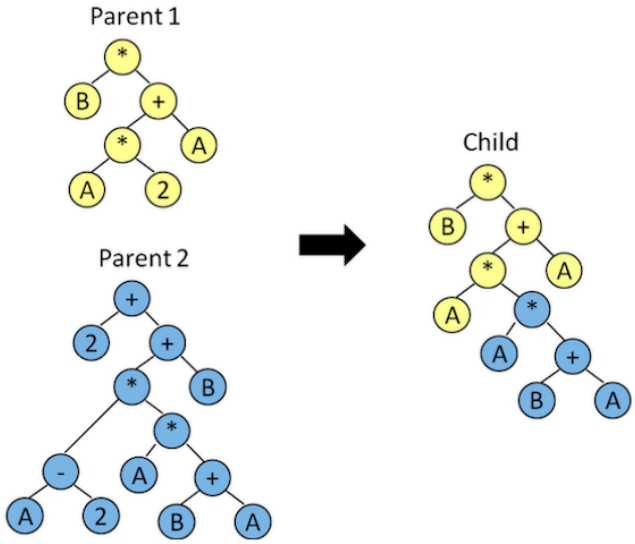
Các chương trình có điểm số cao nhất – đại diện cho các cá thể "thích nghi" tốt nhất – sẽ được chọn lọc để tham gia vào quá trình tạo ra thế hệ kế tiếp thông qua các phép toán di truyền. Tương tự như trong tiến hóa sinh học, các cá thể phù hợp nhất sẽ truyền lại đặc điểm di truyền của mình để sinh ra thế hệ con với khả năng thực hiện nhiệm vụ ngày càng hiệu quả hơn. Toàn bộ quá trình này mô phỏng quá trình chọn lọc tự nhiên trong sinh giới, với ba phép toán di truyền chính được sử dụng bao gồm: sinh sản (reproduction), lai chéo (crossover) và đột biến (mutation).



Hình 3. Reproduction

Như minh họa trong Hình 1.4, phép toán di truyền sinh sản (reproduction) tạo ra một bản sao chính xác của chương trình cha mẹ. Cơ chế này cho phép các chương trình máy tính có độ phù hợp cao nhất từ các thế hệ trước tiếp tục tồn tại và được duy trì trong quần thể của thế hệ hiện tại.

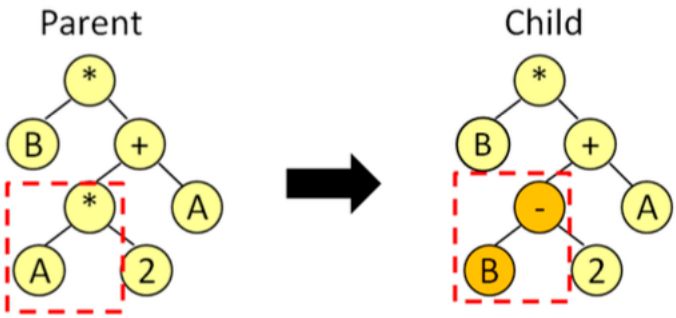
Phép toán di truyền tiếp theo, được thể hiện trong Hình 1.5, là phép lai chéo (crossover). Trong phép lai chéo, hai chương trình máy tính đóng vai trò là cha mẹ, mỗi chương trình truyền lại một phần gen của mình để hình thành nhiễm sắc thể của chương trình con. Cụ thể, như minh họa trong Hình 1.5, cha mẹ 1 (màu vàng) truyền cấu trúc nhiễm sắc thể cơ bản của nó cho chương trình con (màu vàng), trong khi đó, bốn gen từ cha mẹ 2 (màu xanh lam) sẽ thay thế một phần gen tương ứng trong nhiễm sắc thể của cha mẹ 1. Cách kết hợp này cho phép chương trình con kế thừa và tái tổ hợp các đặc điểm di truyền từ cả hai nguồn, góp phần tạo ra sự đa dạng và cải thiện độ phù hợp trong thế hệ tiếp theo.



Hình 4. Cross Over

Phép toán di truyền cuối cùng được sử dụng để tạo ra các chương trình cá thể mới từ thế hệ trước là phép đột biến (mutation). Khác với hai phép toán sinh sản và lai chéo, đột biến cho phép xuất hiện các gen mới không bắt nguồn từ bất kỳ cá thể cha mẹ cụ thể nào. Sự thay đổi hoặc xuất hiện ngẫu nhiên của gen – như minh họa trong Hình 5 – giúp một số chương trình trong thế hệ mới có thể đi lệch khỏi quỹ đạo tiến hóa chung của quần thể, từ đó mở ra cơ hội khám phá những vùng lời giải mới chưa được khảo sát.

Tương tự như trong sinh học, đột biến đôi khi mang lại những đặc điểm vượt trội, giúp chương trình con có khả năng thực hiện nhiệm vụ tốt hơn. Ngược lại, nếu không mang lại lợi ích, những biến thể này sẽ dần bị loại bỏ khỏi quần thể qua cơ chế chọn lọc tự nhiên.



Hình 5. Mutation

Xác suất sử dụng mỗi phép toán di truyền trong quá trình tạo ra thế hệ kế tiếp có thể được người dùng thiết lập tùy theo yêu cầu cụ thể của bài toán. Sau khi quần thể chương trình máy tính mới được sinh ra, chúng sẽ tiếp tục trải qua quy trình GP – bao gồm kiểm

tra độ phù hợp và áp dụng các phép toán di truyền – lặp lại cho đến khi điều kiện dừng được thỏa mãn.

Tổng thể quy trình GP thực chất là sự ứng dụng của thuật toán di truyền (Genetic Algorithm – GA) vào miền chương trình máy tính, với các tham số hoạt động do người dùng định nghĩa, bao gồm: tập hàm (functions), tập biến đầu vào (inputs), các hằng số liên quan, kích thước quần thể khởi tạo, các phép toán di truyền cùng với tỷ lệ áp dụng tương ứng, và tiêu chí dừng (termination criteria).

CHƯƠNG III. KIẾN TRÚC MÔ HÌNH

1. Thiết kế kiến trúc mô hình vận dụng GP

Thay vì phải tự thử thủ công các kiến trúc như:

```
model = nn.Sequential(
    nn.Linear(4, 32),
    nn.ReLU(),
    nn.Linear(32, 1),
    nn.Sigmoid()
)
```

1.1. Tổng quan quá trình tiến hoá

Giai đoạn	Vai trò
1. Khởi tạo quần thể (Population)	Tạo ra nhiều kiến trúc mạng ngẫu nhiên
2. Đánh giá độ phù hợp (Fitness Evaluation)	Huấn luyện từng mô hình và đánh giá độ chính xác
3. Chọn lọc (Selection)	Giữ lại các mô hình tốt nhất để "sinh sản"
4. Lai ghép (Crossover)	Kết hợp hai mô hình để tạo mô hình con
5. Đột biến (Mutation)	Thay đổi ngẫu nhiên một phần trong mô hình
6. Lặp lại (Loop)	Lặp lại các bước trên qua nhiều thế hệ

1.2. Mã hoá kiến trúc

Mỗi cá thể là một chuỗi gene, biểu diễn kiến trúc mạng. Ví dụ:

["Dense(64)", "ReLU", "Dense(32)", "Sigmoid"]

Tức là:

- Lớp Linear (input → 64 node)
- Activation: ReLU
- Lớp Linear (64 → 32 node)
- Activation: Sigmoid
- Cuối cùng có thêm output layer cố định: Linear(32, 1) + Sigmoid()

Ta sử dụng các building blocks như:

- Lớp: Dense(16), Dense(32), Dense(64)
- Activation: ReLU, Sigmoid, Tanh

1.3. Biến đổi

Genotype = danh sách ["Dense(64)", "ReLU", ...]

Phenotype = mô hình thực tế PyTorch nn.Sequential(...)

```
# --- ĐỊNH NGHĨA THÀNH PHẦN KIẾN TRÚC CÓ THỂ TIẾN HÓA ---
layer_options = ["Dense(16)", "Dense(32)", "Dense(64)"]
activation_options = ["ReLU", "Sigmoid", "Tanh"]
building_blocks = layer_options + activation_options
```

Ta có hàm build_model_from_genome() để ánh xạ genotype thành một mô hình mạng thực sự.

```python
# --- MAPPING CÁC THÀNH PHẦN KIẾN TRÚC ---
def build_model_from_genome(genome):
    layers = []
    input_dim = 4
    for gene in genome:
        if gene.startswith("Dense"):
            out_dim = int(gene.split("(")[1][:-1])
            layers.append(nn.Linear(input_dim, out_dim))
            input_dim = out_dim
        elif gene == "ReLU":
            layers.append(nn.ReLU())
        elif gene == "Sigmoid":
            layers.append(nn.Sigmoid())
        elif gene == "Tanh":
            layers.append(nn.Tanh())
    # output layer
    layers.append(nn.Linear(input_dim, 1))
    layers.append(nn.Sigmoid())  # binary classification
    return nn.Sequential(*layers)
```

1.4. Hàm đánh giá

Để biết mô hình tốt hay dở, ta phải đo độ chính xác. Trong ví dụ:

- Mỗi mô hình được huấn luyện nhanh (10 epochs)
- Tính accuracy trên tập test
- Accuracy đó chính là fitness của cá thể

```python
# --- FITNESS FUNCTION ---
def eval_nn(individual):
    model = build_model_from_genome(individual)
    optimizer = torch.optim.Adam(model.parameters(), lr=0.01)
    loss_fn = nn.BCELoss()

    # train trong 10 epochs
    try:
        for epoch in range(10):
            model.train()
            optimizer.zero_grad()
            output = model(X_train)
            loss = loss_fn(output, y_train)
            loss.backward()
            optimizer.step()

        # evaluate
        model.eval()
        with torch.no_grad():
            pred = (model(X_test) > 0.5).float()
            acc = (pred == y_test).float().mean().item()
        return acc,
    except Exception:
        return 0.0,  # nếu model lỗi thì return fitness = 0
```

1.5. Lai ghép

Chọn 2 cá thể tốt → kết hợp một phần chuỗi gene của mỗi cá thể để tạo "con":

Parent 1: ["Dense(64)", "ReLU", "Dense(32)", "Sigmoid"]

Parent 2: ["Dense(16)", "Tanh", "Dense(16)", "ReLU"]

Crossover → Offspring:

["Dense(64)", "ReLU", "Dense(16)", "ReLU"]

Crossover tạo ra biến dị có thể tốt hơn tổ tiên.

1.6. Đột biến

Thay ngẫu nhiên 1 vài gene

Before: ["Dense(32)", "ReLU", "Dense(16)", "Sigmoid"]

Mutation: ReLU → Tanh

After: ["Dense(32)", "Tanh", "Dense(16)", "Sigmoid"]

Tiến hoá

Tiếp tục lặp lại:

Chọn lọc → Lai ghép → Đột biến → Đánh giá → Sinh thế hệ mới

Mỗi thế hệ hy vọng tiến hóa tốt hơn thế hệ trước.

1.7. Kết quả

gen	nevals	avg	max
0	10	0.788889	1
1	5	1	1
2	7	1	1
3	9	0.997778	1
4	5	1	1
5	6	0.991111	1
6	6	1	1
7	3	1	1
8	10	1	1
9	8	1	1
10	7	1	1

```
Best Architecture: ['Tanh', 'Dense(16)', 'Dense(16)', 'Tanh']
Best Accuracy: 1.0
```

Link thực nghiệm: Link

CHƯƠNG IV. ỨNG DỤNG TRONG MỤC TIÊU PHÂN LOẠI

1. Giới thiệu

Bài toán phân lớp loài hoa Iris là một ví dụ kinh điển trong lĩnh vực học máy, thường được sử dụng như một bài toán kiểm thử tiêu chuẩn để đánh giá hiệu quả của các thuật toán phân loại. Do tính đơn giản về mặt cấu trúc nhưng vẫn đảm bảo tính phân biệt giữa các lớp dữ liệu, bài toán Iris đã trở thành nền tảng cho nhiều nghiên cứu và ứng dụng trong lĩnh vực trí tuệ nhân tạo và khai phá dữ liệu.

Trong nghiên cứu này, bài toán phân lớp Iris được sử dụng như một trường hợp điển hình để kiểm nghiệm khả năng áp dụng của lập trình di truyền (Genetic Programming – GP) vào các tác vụ phân loại.

2. Dữ liệu và đặc trưng

Tập dữ liệu Iris ban đầu được công bố bởi Ronald A. Fisher vào năm 1936 và bao gồm 150 mẫu dữ liệu, được chia đều cho ba loài hoa: Iris Setosa, Iris Versicolor, và Iris Virginica. Mỗi mẫu được mô tả bởi bốn thuộc tính:

- Chiều dài đài hoa (sepal length)
- Chiều rộng đài hoa (sepal width)
- Chiều dài cánh hoa (petal length)
- Chiều rộng cánh hoa (petal width)

Tập dữ liệu có tính cân bằng giữa các lớp và đã được chuẩn hóa tốt, do đó thường được sử dụng mà không cần xử lý tiền xử lý phức tạp. Tuy nhiên, giữa ba lớp loài hoa, chỉ có Iris Setosa là dễ tách tuyến tính hoàn toàn, trong khi hai lớp còn lại có mức độ chồng lấn tương đối cao.

3. Mô hình và phương pháp áp dụng

Thuật toán lập trình di truyền được áp dụng để xây dựng mô hình phân loại, trong đó mỗi cá thể trong quần thể GP là một biểu diễn chương trình máy tính có khả năng ánh xạ các đặc trưng đầu vào thành một nhãn lớp cụ thể.

Các tham số chính của GP bao gồm:

- Tập hàm: {+, −, ×, /, if-then-else, >, <}
- Tập đầu cuối: {sepal length, sepal width, petal length, petal width, hằng số ngẫu nhiên}
- Kích thước quần thể: 100

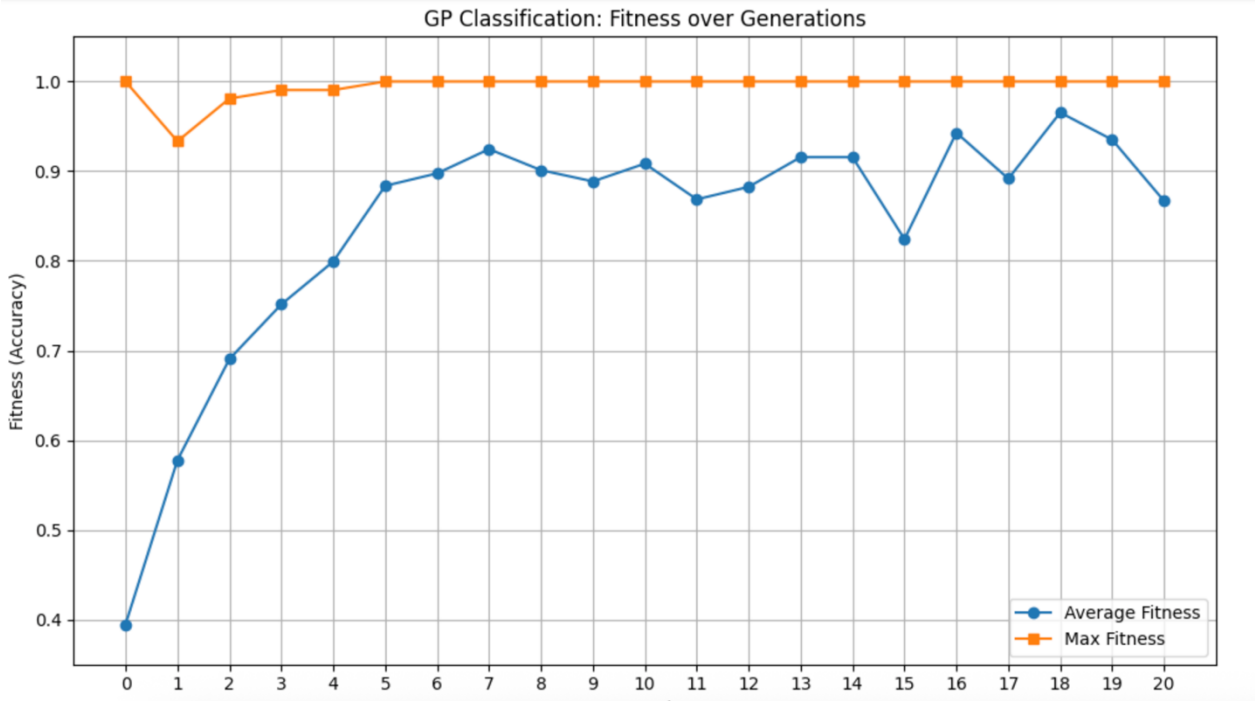
- Số thế hệ: 50
- Tỷ lệ lai chéo: 0.8
- Tỷ lệ đột biến: 0.1

Hàm đánh giá độ phù hợp (fitness): tỷ lệ phân loại chính xác trên tập huấn luyện

Dữ liệu được chia thành hai phần: 70% dùng để huấn luyện và 30% còn lại dùng để kiểm thử. Mỗi cá thể GP sau khi được huấn luyện sẽ được đánh giá dựa trên độ chính xác khi phân loại đúng loài hoa trong tập kiểm thử.

4. Đánh giá và nhận xét

Thực nghiệm trên tập dữ liệu Iris cho thấy GP là một phương pháp phân loại hiệu quả, đặc biệt trong những bài toán có kích thước dữ liệu vừa và nhỏ, với đặc trưng rõ ràng. Ưu điểm nổi bật của GP so với các thuật toán khác như mạng nơ-ron hoặc SVM là khả năng sinh ra biểu thức có thể diễn giải được, giúp người sử dụng hiểu rõ cách ra quyết định của mô hình.



Tuy nhiên, GP có một số hạn chế như chi phí tính toán cao hơn do phải đánh giá một số lượng lớn cá thể qua nhiều thế hệ, và kết quả phụ thuộc nhiều vào cấu hình tham số cũng như yếu tố ngẫu nhiên ban đầu.

CHƯƠNG V. TÀI LIỆU THAM KHẢO

Springer, (2008). Chương 6. Introduction to Genetic Algorithms