

BỘ GIÁO DỤC VÀ ĐÀO TẠO
ĐẠI HỌC KINH TẾ TP. HỒ CHÍ MINH (UEH)
TRƯỜNG CÔNG NGHỆ VÀ THIẾT KẾ

UEH
UNIVERSITY

BÁO CÁO DỰ ÁN CUỐI KỲ

MÔN HỌC: TRÍ TUỆ NHÂN TẠO

**GIẢI BÀI TOÁN TỐI ƯU NETWORK FLOW
PROBLEM DÙNG GIẢI THUẬT GENETIC
ALGORITHM**

Giảng viên : TS. Đặng Ngọc Hoàng Thành

Mã học phần : 25D1INF50904201

Lớp : ST5 – B2.507

Thành viên nhóm : Thái Hoài An

Nguyễn Thị Thùy Dương

Nguyễn Duy Tân

Lê Vy

TP. Hồ Chí Minh - Tháng 05 năm 2025

LỜI CẢM ƠN

Nhóm chúng em xin gửi lời cảm ơn sâu sắc đến Thầy Đặng Ngọc Hoàng Thành vì đã luôn tận tình hướng dẫn và đồng hành cùng chúng em trong suốt quá trình học tập. Môn học Trí tuệ Nhân tạo không chỉ mang đến nhiều kiến thức mà còn giúp chúng em nhận thấy rõ giá trị thực tiễn và tiềm năng ứng dụng rộng lớn của lĩnh vực này.

Nhờ sự truyền đạt dễ hiểu, sát với thực tế và cách tiếp cận gần gũi với sinh viên của Thầy, chúng em đã có cơ hội tiếp cận những kiến thức nền tảng và chuyên sâu một cách vững chắc. Điều này là hành trang quý báu để chúng em tự tin tiếp tục hành trình học tập và phát triển bản thân trong tương lai.

Chúng em hiểu rằng trong quá trình thực hiện đề tài, do giới hạn về kinh nghiệm và kiến thức, chắc chắn vẫn còn những thiếu sót nhất định. Vì vậy, chúng em rất mong nhận được những nhận xét và góp ý từ Thầy để hoàn thiện bài tiểu luận này một cách tốt nhất. Một lần nữa, chúng em xin chân thành cảm ơn Thầy vì sự tận tâm trong giảng dạy và sự hỗ trợ quý báu đã dành cho chúng em trong suốt quá trình thực hiện đề tài.

Kính chúc Thầy luôn mạnh khỏe, nhiều niềm vui và đạt được nhiều thành tựu trong sự nghiệp giáo dục của mình.

TP. Hồ Chí Minh, tháng 5 năm 2025

Nhóm thực hiện

DANH MỤC BẢNG – HÌNH ẢNH

Bảng 4. 1 - Kết quả chạy thuật toán trên các loại đồ thị	29
Bảng 4. 2 - Kết quả thay đổi tham số	31
Hình 2. 1 - Sơ đồ giải thuật di truyền	9
Hình 3. 1 – Giao diện tổng thể ứng dụng	23
Hình 3. 2 – Giao diện khu vực chỉnh sửa đồ thị.....	24
Hình 3. 3 – Bảng điều khiển tham số	24
Hình 3. 4 - Khu vực bảng kết quả	25
Hình 3. 5 - Lựa chọn khi click chuột phải vào node	25
Hình 3. 6 - Bảng thay đổi giá trị capacity.....	25
Hình 3. 7 - Đồ thị được tạo ngẫu nhiên với 3 layers và 6 nodes trên mỗi layers (bên trái) vs 4 layers và 5 nodes trên mỗi layers (bên phải)	26
Hình 3. 8 - Nút chạy thuật toán GA và Nút dừng thuật toán.....	26
Hình 3. 9 - Nút chạy thuật toán FF và so sánh với GA	26
Hình 3. 10 - Lờ giải GA	27
Hình 3. 11 - Lờ giải FA	27
Hình 4. 1 – Biểu đồ tiến triển fitness đồ thị G506.....	30
Hình 4. 2 - Graph thực hiện đo lường thay đổi tham số.....	31

MỤC LỤC

LỜI CẢM ƠN	1
DANH MỤC BẢNG – HÌNH ẢNH.....	2
MỤC LỤC	3
CHƯƠNG 1: TỔNG QUAN VỀ BÀI TOÁN NETWORK FLOW PROBLEM.....	5
1.1 Giới thiệu bài toán.....	5
1.1.1 Định nghĩa	5
1.1.2 Các biến thể phổ biến của Network Flow	5
1.2. Phát biểu bài toán.....	6
1.3. Một số hướng tiếp cận giải quyết bài toán.....	7
1.3.1 Giải quyết bài toán bằng các giải thuật chính xác.....	7
1.3.2 Giải quyết bài toán bằng Genetic Algorithm	7
CHƯƠNG 2: GIẢI THUẬT GENETIC ALGORITHM.....	9
2.1. Thuật toán Genetic Algorithm.....	9
2.2. Các biến thể của thuật toán Genetic Algorithm.....	10
2.2.1. Steady-State Genetic Algorithm	10
2.2.2. Elitist Genetic Algorithm	10
2.2.3. Adaptive Genetic Algorithm (AGA)	10
2.2.4. Multi-objective Genetic Algorithm (MOGA)	10
2.2.5. Hybrid Genetic Algorithm	10
2.3. Áp dụng thuật toán Genetic Algorithm vào bài toán Maximun Network Flow	11
2.3.1. Biểu diễn cá thể.....	11
2.3.2. Khởi tạo quần thể.....	11
2.3.2. Tính giá trị thích nghi	13
2.3.3. Cơ chế chọn lọc.....	14
2.3.4. Cơ chế lai ghép.....	15
2.3.5. Cơ chế đột biến thích nghi	16
2.3.6. Cân bằng luồng.....	17
2.3.7. Tổng thể thuật toán.....	19
CHƯƠNG 3: THIẾT KẾ GIAO DIỆN	23
3.1. Mô tả ứng dụng	23

3.2. Thiết kế giao diện	23
3.3. Mô tả các chức năng	25
CHƯƠNG 4: THẢO LUẬN VÀ ĐÁNH GIÁ	28
4.1. Đánh giá thuật toán.....	28
4.1.1. <i>Độ sai lệch so và Hiệu năng khi mở rộng:</i>	29
4.1.2. <i>Khả năng thoát khỏi tối ưu cục bộ:</i>	30
4.1.3. <i>Ảnh hưởng của các tham số đến kết quả</i>	31
4.2. Ưu điểm và nhược điểm của thuật toán.....	32
4.3. So sánh Genetic Algorithm và các giải thuật khác trong việc giải quyết bài toán	33
CHƯƠNG 5: TỔNG KẾT ĐỀ TÀI.....	34
5.1. Hướng phát triển của đề tài	34
5.1.1. <i>Mở rộng bài toán luồng chuẩn sang các biến thể thực tế</i>	34
5.1.2. <i>Cải tiến và kết hợp với các chiến lược tiến hóa khác</i>	34
5.1.3. <i>Ứng dụng thực tiễn vào các hệ thống mạng phức tạp</i>	34
5.2. Hạn chế trong ứng dụng	35
5.3. Tổng kết đề tài	36
TÀI LIỆU THAM KHẢO.....	37
PHỤ LỤC	38
1. Hình ảnh kết quả thu được khi chạy thuật toán trong đo lường Chương 4 ..	38
2. Hướng dẫn chạy dự án	40
3. Phân công công việc	40

CHƯƠNG 1: TỔNG QUAN VỀ BÀI TOÁN NETWORK FLOW PROBLEM

1.1 Giới thiệu bài toán

1.1.1 Định nghĩa

Bài toán *Network Flow Problem* là một bài toán trong lý thuyết đồ thị và tối ưu hóa, mô tả luồng di chuyển trong một hệ thống mạng. Mạng (*network*) được biểu diễn như một đồ thị có hướng gồm:

- Đỉnh (*nodes*): đại diện cho các nút giao thông, trạm phát điện, điểm giao hàng, v.v.
- Cạnh (*edges*): đại diện cho các tuyến kết nối giữa các đỉnh, mang theo một đại lượng gọi là luồng (*flow*), trên mỗi cạnh đều sẽ có các giá trị dung lượng số.

Mục tiêu là xác định một luồng (*flow*), tức là các giá trị số trên mỗi cạnh, sao cho vừa tuân thủ các ràng buộc về dung lượng, vừa đảm bảo luồng đi vào bằng luồng đi ra tại tất cả các đỉnh (ngoại trừ các node đầu cuối đã được chỉ định) (Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin, 1993), ví dụ như:

- Tối đa hóa luồng tổng từ điểm xuất phát đến điểm đích;
- Tối thiểu hóa chi phí di chuyển;
- Phân phối luồng tối ưu nhằm tránh tắc nghẽn giao thông.

1.1.2 Các biến thể phổ biến của Network Flow

Maximum Flow Problem: Tìm luồng lớn nhất từ đỉnh nguồn đến đỉnh đích sao cho không vượt quá dung lượng (*capacity*) của các cạnh.

Minimum-Cost Flow Problem: Là mô hình cơ bản nhất trong số tất cả các bài toán *network flow*, mục tiêu tìm luồng thỏa mãn yêu cầu tại các đỉnh và có tổng chi phí thấp nhất.

Multi-commodity Flow Problem: Bài toán giải quyết việc nhiều loại hàng hóa khác nhau cùng luân chuyển trên một mạng lưới chung. Vấn đề cốt lõi là làm sao để phân bổ tối ưu dung lượng của các cung cho từng loại hàng hóa, nhằm giảm thiểu tổng chi phí luân chuyển khi mỗi loại hàng hóa có các điểm xuất phát/đích và ràng buộc cân bằng luồng riêng.

Nowhere-zero flow: Là một dạng luồng được nghiên cứu trong lĩnh vực tổ hợp, trong đó lượng luân chuyển bị giới hạn trong một tập hợp hữu hạn các giá trị khác không (Wikipedia, 2024).

1.2. Phát biểu bài toán

Bài toán luồng cực đại (Maximum Flow Problem) là một bài toán tối ưu trên đồ thị có hướng, trong đó mỗi cạnh được gán một giá trị gọi là dung lượng (capacity), biểu thị giới hạn tối đa của luồng có thể đi qua cạnh đó theo một hướng nhất định. Mục tiêu là tìm một phân bố luồng hợp lệ sao cho tổng luồng từ đỉnh nguồn (source) đến đỉnh đích (sink) là lớn nhất, đồng thời thỏa mãn các ràng buộc sau:

- Ràng buộc về dung lượng: Luồng trên mỗi cạnh phải nằm trong khoảng từ 0 đến dung lượng của cạnh đó.
- Ràng buộc bảo toàn luồng: Tại mỗi đỉnh trung gian (ngoại trừ source và sink), tổng luồng vào phải bằng tổng luồng ra, đảm bảo không có "tích trữ" hoặc "hao hụt" luồng tại đỉnh.

Cụ thể, cho một đồ thị có hướng $G = (V, E)$ với:

- $s \in V$: đỉnh nguồn
- $t \in V$: đỉnh đích
- $c(u, v)$: dung lượng của cạnh từ đỉnh u đến v , với $(u, v) \in E$

Tìm hàm luồng $f(u, v)$ sao cho:

1. $0 \leq f(u, v) \leq c(u, v)$ với mọi $(u, v) \in E$
2. $\sum_{v:(u,v) \in E} f(u, v) = \sum_{v:(v,u) \in E} f(v, u)$ với mọi $u \in V \setminus \{s, t\}$
3. Tổng luồng cực đại $F = \sum_{v:(s,v) \in E} f(s, v)$ được tối đa hóa

Đề tài này giải bài toán nêu trên bằng cách áp dụng giải thuật di truyền (Genetic Algorithm) – một phương pháp tối ưu heuristic mô phỏng quá trình tiến hóa tự nhiên. So với các giải thuật cổ điển như Ford–Fulkerson hoặc Edmonds–Karp, việc áp dụng GA cho bài toán Maximum Flow có tính thách thức cao hơn vì:

- Không gian lời giải rộng và liên tục (luồng có thể lấy giá trị bất kỳ từ 0 đến capacity).
- Ràng buộc bảo toàn luồng khó duy trì sau các bước lai ghép (crossover).

- Khó định nghĩa hàm thích nghi sao cho phản ánh đúng cả hai mục tiêu: tối đa hóa luồng và duy trì tính hợp lệ.

Do đó, mục tiêu chính của đề tài là thiết kế một mô hình GA hiệu quả có khả năng tạo ra lời giải hợp lệ (balanced flow) và tối ưu hóa tổng luồng, đồng thời khắc phục các điểm yếu phổ biến trong cách áp dụng GA cho bài toán này.

1.3. Một số hướng tiếp cận giải quyết bài toán

1.3.1 Giải quyết bài toán bằng các giải thuật chính xác

Các thuật toán chính xác (exact algorithms) truyền thống như *Ford-Fulkerson*, *Edmonds-Karp* và *Push-Relabel* là những công cụ phổ biến nhất trong việc giải bài toán luồng cực đại. Trong đó, Ford-Fulkerson sử dụng phương pháp tìm đường tăng luồng (augmenting path) dựa trên tìm kiếm tuyến tính như DFS hoặc BFS, và đạt kết quả tối ưu nếu trọng số luồng là số nguyên. Phiên bản Edmonds-Karp cải tiến bằng cách sử dụng BFS, giúp đảm bảo độ phức tạp thời gian là $O(VE^2)$ (cp-algorithms.com, n.d.). Trong khi đó, thuật toán Push-Relabel sử dụng cách tiếp cận khác hoàn toàn – thay vì tìm đường đi từ nguồn đến đích, nó thực hiện các phép "đẩy" và "gán nhãn" nội tại tại từng đỉnh để đẩy luồng qua mạng, với độ phức tạp thời gian tốt hơn cho một số loại đồ thị cụ thể.

Các thuật toán chính xác này có ưu điểm là đảm bảo tìm được lời giải tối ưu toàn cục với thời gian xác định, tuy nhiên chúng thường khó mở rộng cho các bài toán có ràng buộc phi tuyến, đa mục tiêu hoặc thay đổi theo thời gian.

1.3.2 Giải quyết bài toán bằng Genetic Algorithm

Bên cạnh các giải thuật truyền thống, một hướng tiếp cận khác là sử dụng thuật toán di truyền (Genetic Algorithm – GA), vốn là một phương pháp heuristic dựa trên cơ chế tiến hóa tự nhiên. GA hoạt động bằng cách khởi tạo một quần thể các lời giải ngẫu nhiên, sau đó lặp lại quá trình chọn lọc, lai ghép và đột biến để tiến hóa dần dần đến lời giải tối ưu hoặc gần tối ưu.

Trong bài nghiên cứu về A Parallel Genetic Algorithm for Maximum Flow Problem (Ola M. Surakhi, Mohammad Qatawneh, Hussein A. al Ofeishat, 2017), tác giả đã đề xuất một mô hình GA đặc biệt cho bài toán luồng cực đại, trong đó mỗi cá thể được biểu diễn bằng một ma trận luồng. Các phép toán như lai ghép (crossover) và đột biến (mutation) được định nghĩa dựa trên cấu trúc của mạng, với mục tiêu là tối đa hóa tổng luồng từ nguồn đến đích trong khi vẫn đảm bảo ràng buộc bảo toàn luồng tại các đỉnh trung gian. Một điểm nhấn đặc biệt trong nghiên cứu này là việc sử dụng "energy

function" và khái niệm "assimilation" để điều chỉnh và đánh giá cá thể sau mỗi bước tiến hóa, giúp hướng luồng về trạng thái cân bằng và khả thi.

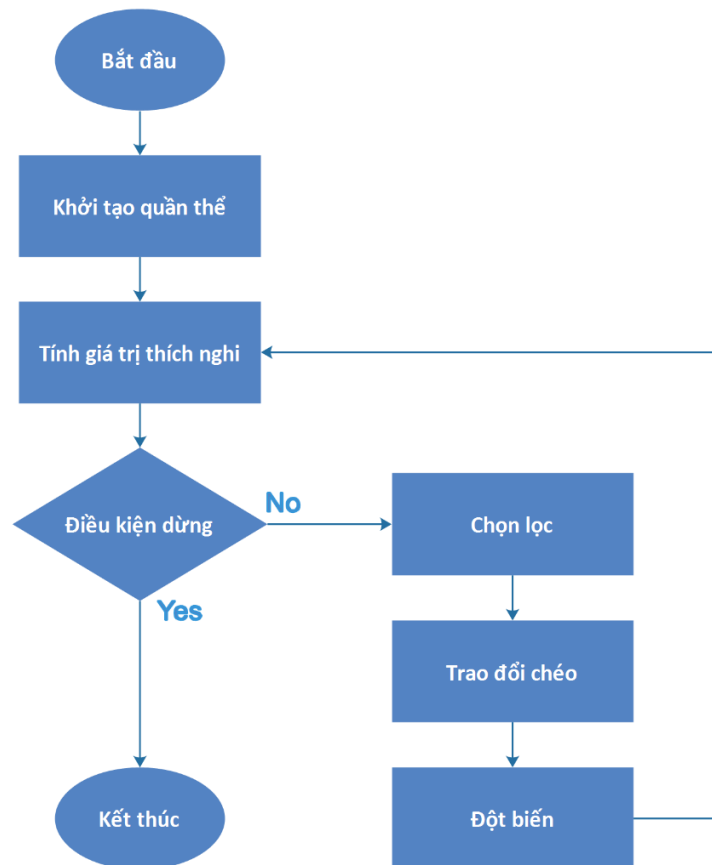
Kết quả trong nghiên cứu cho thấy GA có thể đạt được lời giải gần tối ưu trong số vòng lặp tương đối ít, đặc biệt hiệu quả trong các mạng phức tạp, không đồng đều hoặc có ràng buộc mở rộng. Tuy không đảm bảo tối ưu toàn cục như các thuật toán chính xác, nhưng GA lại linh hoạt, dễ mở rộng và có thể xử lý các biến thể khó mô hình hóa bằng giải tích, như mạng có nhiều nguồn – nhiều đích, mạng có chi phí hoặc độ trễ phi tuyến.

Từ đó, việc áp dụng GA vào bài toán luồng cực đại không chỉ mang tính chất học thuật mà còn mở ra nhiều khả năng ứng dụng trong thực tế, đặc biệt trong các hệ thống phức hợp, mạng động và các bài toán đa mục tiêu. Đây cũng là hướng tiếp cận mà nhóm lựa chọn để triển khai và mở rộng trong đề tài này.

CHƯƠNG 2: GIẢI THUẬT GENETIC ALGORITHM

2.1. Thuật toán Genetic Algorithm

Thuật toán di truyền (Genetic Algorithm – GA) là một kỹ thuật tính toán mô phỏng theo quá trình tiến hóa tự nhiên của Charles Darwin. Thuật toán này được John H. Holland giới thiệu và phổ biến từ những năm 1970, đặc biệt qua cuốn sách *Adaptation in Natural and Artificial Systems* (Holland, 1975). GA thuộc nhóm thuật toán tiến hóa, sử dụng cơ chế chọn lọc tự nhiên và di truyền để giải các bài toán tối ưu hóa và tìm kiếm (Genetic algorithm, n.d.). Nói một cách đơn giản, GA duy trì một *quần thể* các lời giải ứng viên và thực hiện “chọn lọc những cá thể thích nghi nhất” qua nhiều thế hệ để dần tiến tới lời giải tốt nhất. Quá trình tiến hóa lặp lại này bao gồm việc đánh giá độ thích nghi của từng cá thể, sau đó lai ghép và đột biến các cá thể *cha mẹ* để sinh ra *cá thể con*, tương tự như sinh sản tự nhiên. Nhờ khai thác thông tin lịch sử (các giải pháp tốt đã tìm được) và yếu tố ngẫu nhiên có định hướng, GA có khả năng tìm ra các giải pháp chất lượng cao cho những bài toán tối ưu phức tạp (genetic-algorithms, n.d.)



Hình 2. 1 - Sơ đồ giải thuật di truyền

Nguồn: (Duy, 2017)

2.2. Các biến thể của thuật toán Genetic Algorithm

2.2.1. Steady-State Genetic Algorithm

Khác với GA truyền thống (generational GA) – nơi toàn bộ quần thể được thay thế sau mỗi thế hệ, Steady-State GA chỉ tạo một số lượng nhỏ cá thể con (thường là 1 hoặc 2), sau đó chèn hoặc thay thế dần các cá thể kém trong quần thể hiện tại. Biến thể này giúp duy trì sự ổn định trong tiến hóa, giảm dao động lớn giữa các thế hệ và phù hợp với các bài toán yêu cầu cập nhật liên tục trong thời gian thực.

2.2.2. Elitist Genetic Algorithm

Biến thể Elitism cho phép giữ lại một số cá thể tốt nhất (top-k) từ thế hệ hiện tại và truyền thẳng vào thế hệ tiếp theo mà không qua crossover hay mutation. Kỹ thuật này đảm bảo rằng lời giải tốt không bị mất đi qua các phép biến đổi ngẫu nhiên, đồng thời giúp tăng độ ổn định và tốc độ hội tụ của thuật toán.

2.2.3. Adaptive Genetic Algorithm (AGA)

Trong AGA, các tham số như tỷ lệ đột biến (mutation rate) hay tỷ lệ lai ghép (crossover rate) được điều chỉnh động theo diễn tiến của quá trình tiến hóa. Ví dụ, nếu trong nhiều thế hệ liên tiếp không có sự cải thiện về fitness, tỷ lệ đột biến sẽ được tăng lên để thúc đẩy khám phá. Biến thể này đặc biệt hữu ích để tránh hiện tượng hội tụ sớm và cải thiện hiệu suất trên các bài toán có không gian lời giải rộng và không đồng đều.

2.2.4. Multi-objective Genetic Algorithm (MOGA)

Đối với các bài toán có nhiều mục tiêu tối ưu đồng thời (ví dụ: tối đa hóa luồng, tối thiểu hóa chi phí, cân bằng tải...), MOGA sử dụng khái niệm thống trị Pareto (Pareto dominance) để chọn lọc và đánh giá cá thể. Thay vì chỉ dựa trên một hàm fitness duy nhất, thuật toán tìm ra tập lời giải không thể cải thiện thêm về mọi mặt (Pareto-optimal set). Một đại diện nổi bật là thuật toán NSGA-II.

2.2.5. Hybrid Genetic Algorithm

Hybrid GA kết hợp GA với các phương pháp khác như hill climbing, local search hoặc tabu search để cải thiện lời giải sau mỗi vòng tiến hóa. Sau khi crossover và mutation tạo ra cá thể mới, một thuật toán local search có thể được áp dụng để tinh chỉnh lời giải đến điểm tối ưu cục bộ gần nhất. Cách tiếp cận này giúp kết hợp sức mạnh của khám phá toàn cục (global search) từ GA với khả năng khai thác cục bộ (local exploitation), mang lại kết quả tốt hơn trong thực tế.

2.3. Áp dụng thuật toán Genetic Algorithm vào bài toán Maximum Network Flow

Trong bài báo cáo này, nhóm đã có một vài thay đổi cải tiến cho thuật toán Genetic Algorithms trong việc giải quyết bài toán Maximum Network Flow.

2.3.1. Biểu diễn cá thể

Nhóm lựa chọn biểu diễn cá thể dưới dạng từ điển thay vì ma trận luồng giống bài của tác giả Surakhi (Ola M. Surakhi, Mohammad Qatawneh, Hussein A. al Ofeishat, 2017)

```
{(u, v): flow_value for (u, v) in edges}
```

Khi sử dụng biểu diễn ma trận, việc thực hiện phép **crossover** (lai ghép) giữa hai cá thể thường dẫn đến việc kết hợp tùy ý các dòng chảy giữa các cặp đỉnh, khiến tổng inflow và outflow tại các đỉnh không còn đồng nhất. Điều này làm phá vỡ tính hợp lệ của cá thể sau crossover, và gây khó khăn trong việc kiểm soát ràng buộc luồng theo Surakhi (Ola M. Surakhi, Mohammad Qatawneh, Hussein A. al Ofeishat, 2017), đặc biệt khi không có cơ chế kiểm tra hoặc cân bằng lại tức thời.

Ngược lại, biểu diễn bằng từ điển giúp dễ dàng lựa chọn các nhóm cạnh hợp lệ theo đường đi (path-based crossover) mà vẫn đảm bảo tính chất cân bằng tổng luồng tại từng đỉnh.

Do đó, biểu diễn từ điển không chỉ đơn giản hóa việc thực hiện phép crossover mà còn phù hợp hơn về mặt cấu trúc dữ liệu để duy trì các ràng buộc vật lý của bài toán.

2.3.2. Khởi tạo quần thể

Nhóm ở đây thay vì khởi tạo quần thể một cách ngẫu nhiên thì đã có thêm cải tiến là khởi tạo có thiên vị cho quần thể ban đầu:

Khởi tạo ngẫu nhiên chuẩn:

```
def initialize_individual(self):  
    individual = {}  
    for u, v, cap in self.graph_edges:  
        individual[(u, v)] = random.randint(0, cap)  
    return self.balance_flow(individual)
```

Khởi tạo có thiên vị có tham số bias:

```
def initialize_diverse_individual(self, bias_percentage):  
    # Ưu tiên luồng cao cho các cạnh từ nguồn và đến đích  
    for u, v, cap in self.graph_edges:
```

```

if u == self.source or v == self.sink:

    individual[(u, v)] = int(cap * random.uniform(bias_percentage, 1.0))

else:

    individual[(u, v)] = random.randint(0, cap)

return self.balance_flow(individual)

```

Trong thuật toán di truyền, cách khởi tạo quần thể ban đầu có ảnh hưởng lớn đến tốc độ hội tụ và chất lượng lời giải cuối cùng. Thay vì sử dụng cách khởi tạo hoàn toàn ngẫu nhiên như truyền thống – tức gán giá trị luồng bất kỳ trong giới hạn năng lực của cạnh – nhóm đã thực hiện một cải tiến bằng cách thiên vị luồng cao hơn cho các cạnh nối từ nguồn (source) hoặc đến đích (sink).

Lý do cho lựa chọn này là vì trong bài toán luồng cực đại, các cạnh xuất phát từ nguồn và đi vào đích đóng vai trò quyết định đến giá trị tổng luồng. Nếu các cạnh này bị gán luồng thấp ngay từ đầu, cá thể sẽ có tổng fitness thấp, khó vượt qua vòng chọn lọc tự nhiên.

Chiến lược khởi tạo quần thể sẽ kết hợp cả hai phương thức là một nửa quần thể đầu tiên được khởi tạo bằng phương pháp ngẫu nhiên hoàn toàn và nửa còn lại sử dụng phương pháp khởi tạo có thiên vị.

```

def initialize_population(self) -> List[Dict[Tuple[int, int], int]]:

    population = []

    # Khởi tạo một nửa số cá thể với khởi tạo ngẫu nhiên

    standard_count = self.pop_size // 2

    for _ in range(standard_count):

        population.append(self.initialize_individual())

    # Tạo các cá thể với bias hướng về đường trực tiếp từ nguồn đến đích

    for i in range(self.pop_size - standard_count):

        # Thay đổi tỷ lệ bias để tạo đa dạng

        bias = 0.5 + (i / (self.pop_size - standard_count)) * 0.4 # Bias
        từ 0.5 đến 0.9

        population.append(self.initialize_diverse_individual(bias))

    return population

```

Mức độ thiên vị (**bias_percentage**) được điều chỉnh tăng dần từ 0.5 đến 0.9, tạo ra sự đa dạng định hướng: một số cá thể được gán luồng cao vừa phải, trong khi một số khác gần đạt mức cực đại theo giới hạn năng lực cạnh.

Chiến lược này vừa đảm bảo đa dạng ban đầu về mặt cấu trúc, vừa tăng xác suất sinh ra những cá thể có fitness cao, từ đó thúc đẩy khả năng hội tụ nhanh hơn của thuật toán mà không làm mất cân bằng ràng buộc luồng.

2.3.2. Tính giá trị thích nghi

Hàm `compute_fitness` là thành phần then chốt trong thuật toán di truyền, dùng để đánh giá chất lượng (fitness) của từng cá thể. Cách đánh giá này phải được thiết kế sao cho vừa phản ánh mục tiêu bài toán, vừa duy trì ràng buộc hợp lệ về bảo toàn luồng.

```
def compute_fitness(self, flow: Dict[Tuple[int, int], int]) -> int:
    """
    Tính độ thích nghi của một cá thể (luồng)
    Độ thích nghi = tổng luồng ra từ nguồn (hoặc vào đích)
    Với điều kiện: luồng phải bảo toàn tại các đỉnh trung gian
    """
    # Tính tổng luồng ra từ nguồn
    source_outflow = sum(f_val for (u, v), f_val in flow.items() if u == self.source)

    # Tính tổng luồng vào đích
    sink_inflow = sum(f_val for (u, v), f_val in flow.items() if v == self.sink)

    # Kiểm tra bảo toàn luồng tại các đỉnh trung gian
    for node in self.intermediate_nodes:
        inflow = sum(flow.get(edge, 0) for edge in self.incoming_edges[node])
        outflow = sum(flow.get(edge, 0) for edge in self.outgoing_edges[node])

        if inflow != outflow:
            # Phạt cá thể không bảo toàn luồng
            return -1

    # Trả về giá trị nhỏ hơn giữa luồng ra từ nguồn và luồng vào đích
    # Đảm bảo không tạo luồng "từ hư không"
    return min(source_outflow, sink_inflow)
```

Nguyên lý hoạt động của hàm là tính tổng luồng ra từ nguồn (`source_outflow`) và tổng luồng vào đích (`sink_inflow`). Hơn nữa còn phải kiểm tra ràng buộc bảo toàn luồng tại tất cả các đỉnh trung gian. Nếu phát hiện vi phạm ($\text{inflow} \neq \text{outflow}$), trả về `fitness = -1` như một cơ chế phạt nặng. Nếu hợp lệ, fitness của cá thể là giá trị nhỏ hơn giữa

source_outflow và sink_inflow, nhằm tránh các cấu hình sinh luồng “ảo” không được truyền hết qua mạng.

Với cách tính fitness như vậy ta thấy rằng nó rõ ràng, dễ tính toán và phản ánh đúng mục tiêu tối ưu hóa tổng luồng từ source đến sink. Việc dùng min(source_outflow, sink_inflow) giúp tránh đánh giá sai lệch do mất cân bằng tạm thời ở đầu–cuối mạng.

Tuy nhiên cơ chế phạt trả về -1 mang tính mạnh tay dẫn đến hệ quả là có thể khiến mất đi nhiều cá thể tiềm năng chỉ vì sai lệch nhỏ do đột biến. Trong một số tình huống, cá thể có luồng cao nhưng mất cân bằng nhỏ vẫn bị loại bỏ hoàn toàn – điều này ảnh hưởng đến khả năng khám phá lời giải mới. Do đó ở dưới nhóm có sử dụng các cơ chế để khắc phục yếu điểm này.

2.3.3. Cơ chế chọn lọc

Để tăng hiệu quả tiến hóa và hạn chế hiện tượng hội tụ sớm vào cực trị cục bộ, nhóm đã triển khai một cơ chế chọn lọc lai ghép kết hợp nhiều chiến lược với nhau để thử nghiệm.

Đầu tiên là cơ chế Elitism – Giữ lại cá thể ưu tú nhất, mỗi thế hệ, nhóm giữ lại top_k cá thể có độ thích nghi cao nhất để đưa thẳng vào thế hệ tiếp theo. Cơ chế này đảm bảo rằng những lời giải tốt nhất sẽ không bị mất đi qua các thao tác crossover hay mutation, giúp tăng ổn định và duy trì chất lượng tiến hóa liên tục.

Các cá thể còn lại trong quần thể mới được tạo ra thông qua lai ghép giữa các cặp cha mẹ được chọn bằng chiến lược tournament selection.

```
def tournament_selection(self, population, fitness_scores, tournament_size):  
    """Select an individual using tournament selection"""  
  
    # Select tournament_size individuals randomly  
  
    tournament_indices = random.sample(range(len(population)),  
min(tournament_size, len(population)))  
  
    # Find the best individual in the tournament  
  
    best_idx = tournament_indices[0]  
    best_fitness = fitness_scores[best_idx]  
  
    for idx in tournament_indices[1:]:  
        if fitness_scores[idx] > best_fitness:  
            best_idx = idx
```



```

        best_fitness = fitness_scores[idx]

    return population[best_idx]

```

Nguyên lý hoạt động của hàm là chọn ngẫu nhiên một nhóm cá thể từ quần thể hiện tại, với số lượng là `tournament_size` được người dùng nhập vào. Sau đó so sánh độ thích nghi (fitness) của các cá thể trong nhóm được chọn, cá thể có fitness cao nhất trong nhóm sẽ được đem đi lai ghép.

Vậy ta có thể thấy khi `tournament_size` nhỏ sẽ tăng độ ngẫu nhiên lên và duy trì được sự đa dạng. Ngược lại với `tournament_size` lớn thì thuật toán sẽ ưu tiên cá thể tốt và tốc độ hội tụ nhanh hơn.

Các cơ chế này đảm bảo tính ngẫu nhiên có kiểm soát: việc chọn nhóm là ngẫu nhiên, nhưng chiến thắng lại dựa vào chất lượng cá thể. Điều này giúp thuật toán duy trì áp lực chọn lọc phù hợp: cá thể tốt hơn có cơ hội sinh sản cao hơn, nhưng cá thể yếu vẫn có thể được chọn nếu may mắn, tránh việc hội tụ sớm tại cục bộ.

2.3.4. Cơ chế lai ghép

Như đã nói ở trên lai ghép thông thường trên ma trận luồng gặp vấn đề với ràng buộc bảo toàn luồng. Nên nhóm đề xuất phương pháp "Path-based Crossover":

```

def crossover_path_based(self, F1, F2):
    paths_F1 = self.find_augmenting_paths(F1, max_paths=2)
    paths_F2 = self.find_augmenting_paths(F2, max_paths=2)

    child_flow = defaultdict(int)

    # Kết hợp các đường tăng luồng
    for path, bottleneck in paths_F1 + paths_F2:
        for i in range(len(path) - 1):
            u, v = path[i], path[i + 1]
            if (u, v) in self.capacity_map:
                child_flow[(u, v)] += bottleneck

    # Giới hạn bởi capacity
    for edge in list(child_flow.keys()):
        if edge in self.capacity_map:

```

```

child_flow[edge] = min(child_flow[edge], self.capacity_map[edge])

return self.balance_flow(child_flow)

```

Phương pháp này hoạt động như sau:

- Bước 1: Từ hai cá thể cha F1 và F2, trích xuất một số lượng giới hạn các đường tăng luồng (augmenting paths), mỗi đường đi kèm với giá trị nút thắt (bottleneck capacity).
- Bước 2: Các đường này được gộp lại để hình thành cá thể con – bằng cách cộng dồn luồng qua từng cạnh trên các đường đi đã chọn.
- Bước 3: Mỗi luồng tại cạnh được giới hạn không vượt quá năng lực (capacity) ban đầu.
- Bước 4: Cá thể con được cân bằng lại luồng bằng hàm `balance_flow` để đảm bảo tuân thủ ràng buộc $\text{inflow} = \text{outflow}$ tại các đỉnh trung gian.

Với việc tìm kiếm các đường tăng luồng và thực hiện chỉnh sửa cân bằng luồng nên thuật toán này có thể đảm bảo được ràng buộc luồng vào bằng luồng ra. Tuy nhiên một vấn đề của phương pháp trên gặp phải đó là dễ mắc kẹt tại cực trị cục bộ (local optimum) trong quá trình tiến hóa.

Nguyên nhân chính đến từ việc thuật toán lai ghép chỉ sử dụng các đường tăng luồng đã tồn tại trong cá thể cha mẹ. Khi quá trình này lặp lại nhiều lần mà không có đột phá về cấu trúc, quần thể dần hội tụ quanh một vùng lời giải "tốt vừa phải" nhưng không phải tối ưu toàn cục. Điều này đặc biệt xảy ra khi các cá thể trong quần thể ngày càng trở nên giống nhau, làm giảm tính đa dạng di truyền – một yếu tố then chốt để duy trì khả năng khám phá của thuật toán di truyền.

Ngoài ra, hàm chọn đường đi (`find_augmenting_paths`) thường ưu tiên các đường có bottleneck cao, vô tình khiến thuật toán bị ràng buộc bởi những đường đi “an toàn” và khó thoát khỏi cấu trúc cũ, càng làm tăng nguy cơ rơi vào bẫy tối ưu cục bộ.

2.3.5. Cơ chế đột biến thích nghi

Để xử lý yếu điểm ở trên, thay vì chỉ sử dụng cơ chế đột biến bình thường nhóm thay đổi thành cơ chế đột biến thích nghi.

```

def update_mutation_rate(self, current_best_fitness):
    # Tăng tỷ lệ đột biến nếu không cải thiện

```

```

if self.no_improvement_count > 10:
    self.current_mutation_rate = min(0.3, self.current_mutation_rate * 1.5)
else:
    # Giảm tỷ lệ đột biến nếu đang cải thiện
    self.current_mutation_rate = max(0.001, self.current_mutation_rate *
0.95)

```

Cơ chế đột biến thích nghi được thiết kế dựa trên nguyên lý điều chỉnh tỷ lệ đột biến động (adaptive mutation rate) tùy theo mức độ cải thiện fitness trong quá trình tiến hóa:

- Khi không có cải thiện trong nhiều thế hệ (**no_improvement_count > 10**), tỷ lệ đột biến được tăng lên, nhằm kích hoạt lại tính khám phá (exploration) của quần thể. Điều này giúp cá thể thoát khỏi vùng lân cận local optimum bằng cách thử các cấu hình mới lạ hơn.
- Ngược lại, khi thuật toán đang cải thiện, tỷ lệ đột biến được giảm dần, để tránh phá vỡ các cấu trúc tốt đã tìm được và duy trì ổn định tiến hóa.

Cơ chế này có vai trò tương tự như một bộ điều chỉnh "nhiệt độ tiến hóa", giúp cân bằng giữa việc Khai thác (exploitation) khi tiến gần tối ưu và Khám phá (exploration) khi bị "mắc kẹt".

2.3.6. Cân bằng luồng

Sau quá trình lai ghép hoặc đột biến, các cá thể thường không còn giữ được ràng buộc bảo toàn luồng - tổng luồng vào phải bằng tổng luồng ra. Để đảm bảo lời giải hợp lệ, nhóm đã xây dựng hàm **balance_flow**, với chức năng điều chỉnh luồng tại các đỉnh trung gian sau khi thực hiện lai ghép nhằm khôi phục lại tính bảo toàn.

```

def balance_flow(self, flow: Dict[Tuple[int, int], int]) -> Dict[Tuple[int,
int], int]:
    """Cân bằng luồng tại các đỉnh trung gian để đảm bảo tính bảo toàn"""
    # Khởi tạo và áp dụng ràng buộc về capacity
    balanced_flow = {edge: min(flow.get(edge, 0), cap) for edge, cap in
self.capacity_map.items()}

    # Lặp để lan truyền thay đổi qua mạng
    for _ in range(3):
        for node in self.intermediate_nodes:
            # Tính luồng vào và ra

```

```

        inflow = sum(balanced_flow.get(edge, 0) for edge in
self.incoming_edges[node])

        outflow = sum(balanced_flow.get(edge, 0) for edge in
self.outgoing_edges[node])

        if inflow == outflow:

            continue # Đỉnh đã cân bằng

        imbalance = inflow - outflow

        if imbalance > 0: # Lượng vào > lượng ra

            self._adjust_outgoing_flow(balanced_flow, node, imbalance)

        else: # Lượng ra > lượng vào

            self._adjust_incoming_flow(balanced_flow, node, -imbalance)

    return balanced_flow

```

Bước đầu tiên là áp dụng ràng buộc capacity trực tiếp cho tất cả các cạnh, đảm bảo không có luồng nào vượt quá năng lực. Sau đó, thuật toán thực hiện 3 vòng cân bằng (passes) cho toàn bộ các đỉnh trung gian. Tại mỗi đỉnh, nếu phát hiện mất cân bằng:

- Gọi `_adjust_outgoing_flow()` nếu $\text{inflow} > \text{outflow}$.
- Gọi `_adjust_incoming_flow()` nếu $\text{outflow} > \text{inflow}$.

Hai hàm phụ xử lý từng trường hợp bằng cách tăng luồng tại cạnh còn dư hoặc giảm luồng ngược lại theo tỉ lệ, nếu không thể tăng đủ.

Tuy nhiên, nếu cân bằng luồng được thực hiện quá mạnh tay hoặc quá nhiều vòng lặp, sẽ gây ra hệ quả tiêu cực làm suy yếu vai trò nguyên gốc của thuật toán GA. Rõ ràng nhất là sau khi crossover hoặc mutation sinh ra cá thể con, nếu cá thể bị chỉnh sửa quá nhiều bởi `balance_flow`, thì các đặc trưng di truyền từ cha mẹ có thể bị làm mờ hoặc mất hoàn toàn.

Điều này giảm tác dụng của chọn lọc tự nhiên, khiến GA gần giống một thuật toán sửa lỗi hơn là tiến hóa. Hơn nữa nếu mọi cá thể đều bị “nắn chỉnh” về cùng một kiểu cân bằng, quần thể sẽ trở nên đồng nhất theo thời gian, tăng nguy cơ hội tụ sớm vào tối ưu cục bộ.

Để đảm bảo được ý nghĩa của thuật toán GA và đảm bảo được tính hợp lệ của đáp án, nhóm đã giới hạn số vòng của cân bằng để đảm bảo rằng tạo ra cá thể hợp lệ nhưng vẫn giữ lại đủ thông tin di truyền để phục vụ tiến hóa. Cân bằng này là yếu tố cốt lõi để giữ hiệu quả cho toàn bộ hệ thống.

2.3.7. Tổng thể thuật toán

Hàm run() là hàm chính điều phối toàn bộ quá trình tiến hóa của thuật toán di truyền để giải bài toán Maximum Flow. Cấu trúc thuật toán được tổ chức rõ ràng theo từng giai đoạn trong vòng đời tiến hóa:

```
def run(self):  
    # Khởi tạo các biến cần thiết  
    self.graph_edges_keys_only = list(self.capacity_map.keys())  
    self.current_mutation_rate = self.mutation_rate  
    self.best_fitness_history = []  
    self.no_improvement_count = 0
```

2.3.7.1 Khởi tạo quần thể

Gọi hàm initialize_population() để sinh ra quần thể ban đầu bao gồm các cá thể được khởi tạo ngẫu nhiên và có thiên hướng định hướng theo cấu trúc đồ thị. Mỗi cá thể đại diện cho một phân bố luồng trên mạng, được biểu diễn dưới dạng từ điển cạnh – luồng.

```
# Khởi tạo quần thể ban đầu  
population = self.initialize_population()  
best_solution = None  
best_fitness = float('-inf')  
fitness_history = []  
# Theo dõi top 5 cá thể tốt nhất  
top_solutions = []
```

2.3.7.2 Tiến hóa các thế hệ

Lặp lại trong self.generations vòng lặp sau đó đánh giá độ thích nghi (fitness). Mỗi cá thể được tính điểm dựa trên hàm compute_fitness. Cá thể tốt nhất hiện tại được truy xuất từ quần thể và so sánh với **best_fitness** toàn cục để cập nhật nếu cần.

```
# Lặp qua các thế hệ  
for generation in range(self.generations):
```

```

# Tính độ thích nghi cho mỗi cá thể trong quần thể

fitness_scores = [self.compute_fitness(ind) for ind in population]

# Tìm cá thể tốt nhất trong thế hệ hiện tại

current_max_fitness = float('-inf')

current_best_individual = None

if fitness_scores:

    current_max_fitness = max(fitness_scores)

    current_best_individual =

population[fitness_scores.index(current_max_fitness)]

```

Ghi nhận `best_fitness` qua từng thế hệ để phân tích hiệu suất hội tụ. Cập nhật biến `no_improvement_count` để theo dõi số thế hệ không cải thiện — dùng cho cơ chế đột biến thích nghi.

```

# Cập nhật lời giải tốt nhất

if current_max_fitness > best_fitness:

    best_fitness = current_max_fitness

    best_solution = current_best_individual.copy()

    self.no_improvement_count = 0

else:

    self.no_improvement_count += 1

```

Nếu không có cải thiện sau nhiều thế hệ, thuật toán sẽ tự động tăng tỷ lệ đột biến để thoát khỏi local optimum. Ngược lại, nếu đang cải thiện tốt, tỷ lệ đột biến sẽ giảm dần để ổn định tiến hóa.

```

# Ghi lại lịch sử độ thích nghi tốt nhất

fitness_history.append(best_fitness)

# Cập nhật tỷ lệ đột biến nếu kích hoạt chế độ thích ứng

if self.adaptive_mutation:

    self.update_mutation_rate(current_max_fitness)

# Kiểm tra điều kiện dừng sớm

if not fitness_scores or not population:

    break

# Sắp xếp quần thể theo độ thích nghi

sorted_population_with_scores = sorted(zip(fitness_scores, population),

                                         key=lambda x: x[0], reverse=True)

```

```

        # Cập nhật top 5 sau mỗi thế hệ

        top_solutions = [(score, ind.copy()) for score, ind in
sorted_population_with_scores[:5]]

```

Giữ lại top_k cá thể tốt nhất để đảm bảo không mất lời giải chất lượng. Cha mẹ được chọn qua tournament selection nhưng nếu tournament_size = 0 thì chọn ngẫu nhiên. Áp dụng Path-based Crossover và mutation để sinh ra cá thể mới. Quá trình này lặp lại cho đến khi quần thể đạt kích thước yêu cầu.

```

        # Chọn lọc: giữ lại top_k cá thể tốt nhất (elitism)

        new_population = [ind for _, ind in
sorted_population_with_scores[:self.top_k]]

        # Tạo phần còn lại của quần thể thông qua lai ghép và đột biến

        while len(new_population) < self.pop_size:

            # Chọn lọc: Tournament selection

            if self.tournament_size > 0 and len(population) >
self.tournament_size:

                parent1 = self.tournament_selection(population, fitness_scores,
self.tournament_size)

                parent2 = self.tournament_selection(population, fitness_scores,
self.tournament_size)

            else:

                # Hoặc chọn ngẫu nhiên nếu không dùng tournament

                parent1 = random.choice(population)

                parent2 = random.choice(population)

            # Lai ghép: Path-based crossover

            child = self.crossover_path_based(parent1, parent2)

            # Đột biến

            child = self.mutate(child)

            # Thêm vào quần thể mới

            new_population.append(child)

        # Cập nhật quần thể

        population = new_population[:self.pop_size]

```

2.3.7.3 Trả về kết quả cuối cùng

```

        # Đảm bảo trả về ít nhất một cá thể khi top_solutions rỗng

        if not top_solutions and best_solution is not None:

            top_solutions = [(best_fitness, best_solution)]

```

```
# Đảm bảo có đúng 5 phần tử

while len(top_solutions) < 5:

    # Điền các phần tử giả nếu thiếu

    top_solutions.append((0, {}))

# Trả về kết quả: cá thể tốt nhất, độ thích nghi, lịch sử, top 5 cá thể

return best_solution, best_fitness, fitness_history, top_solutions
```

Hàm sẽ trả về các kết quả:

- `best_solution`: cá thể có fitness cao nhất toàn bộ quá trình.
- `best_fitness`: giá trị fitness tương ứng.
- `fitness_history`: biểu đồ lịch sử tiến hóa.
- `top_solutions`: danh sách các cá thể tốt nhất từng được ghi nhận.

CHƯƠNG 3: THIẾT KẾ GIAO DIỆN

3.1. Mô tả ứng dụng

GA – *Maximum Flow Solver* là một ứng dụng trực quan nhóm xây dựng để hỗ trợ minh họa cho bài toán luồng cực đại trong đồ thị có hướng, bằng cách sử dụng thuật toán di truyền (Genetic Algorithm – GA). Ứng dụng cho phép người dùng trực tiếp thiết kế và chỉnh sửa đồ thị, hỗ trợ người dùng tự điều chỉnh tham số GA, theo dõi quá trình tiến hóa, và so sánh kết quả với thuật toán Ford–Fulkerson. Tất cả kết quả được trực quan hóa sinh động, giúp người dùng đánh giá hiệu quả thuật toán một cách dễ dàng.

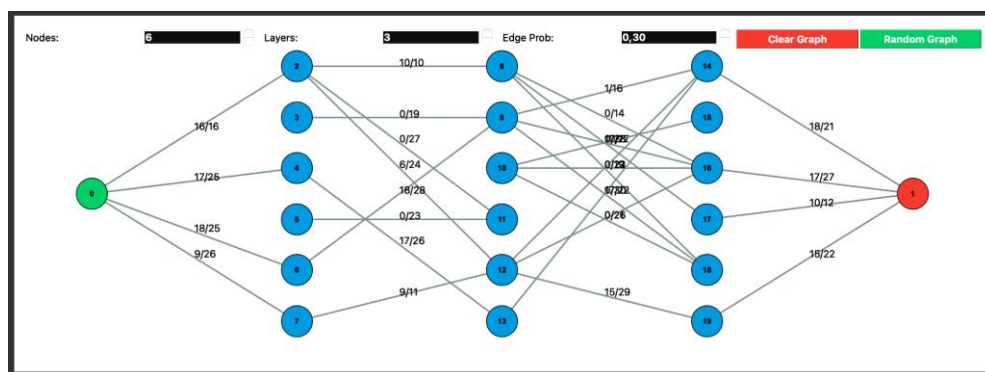
3.2. Thiết kế giao diện

Giao diện của ứng dụng được chia thành ba khu vực chính, tạo nên một bộ cục hợp lý giữa chỉnh sửa, điều khiển và quan sát kết quả.



Hình 3. 1 – Giao diện tổng thể ứng dụng

Bên trái màn hình là khu vực chỉnh sửa đồ thị – nơi người dùng có thể trực tiếp tương tác để thêm, xóa, và định cấu trúc cho đồ thị. Đồ thị được vẽ dưới dạng trực quan với các node và cạnh, trong đó node nguồn và node đích được đánh dấu bằng màu sắc khác biệt để dễ nhận diện.



Hình 3. 2 – Giao diện khu vực chỉnh sửa đồ thị

Phía bên phải là bảng điều khiển tham số, nơi người dùng có thể cấu hình toàn bộ các biến số của giải thuật di truyền. Giao diện này hỗ trợ nhập liệu cho các thông số như kích thước quần thể, số thế hệ, tỷ lệ đột biến, tỷ lệ lai ghép, số cá thể giữ lại ở mỗi thế hệ, số đường đi lai ghép, kích thước đấu chọn và tùy chọn đột biến thích nghi. Nút “Run GA” và “Stop GA” dùng để chạy và dừng thuật toán cùng với dòng trạng thái hiển thị quá trình thực thi để người dùng có thể theo dõi.

Kích thước quần thể (Population Size):	30	▲▼
Số thế hệ (Max Generations):	100	▲▼
Tỷ lệ đột biến (Mutation Rate):	0,030	▲▼
Tỷ lệ lai ghép (Crossover Rate):	0,80	▲▼
Số cá thể giữ lại (Elitism (Top-K)):	3	▲▼
Số đường đi lai ghép (Max Paths Crossover):	2	▲▼
Kích thước đấu chọn (Tournament Size):	3	▲▼
Đột biến thích nghi (Adaptive Mutation):	<input checked="" type="checkbox"/>	

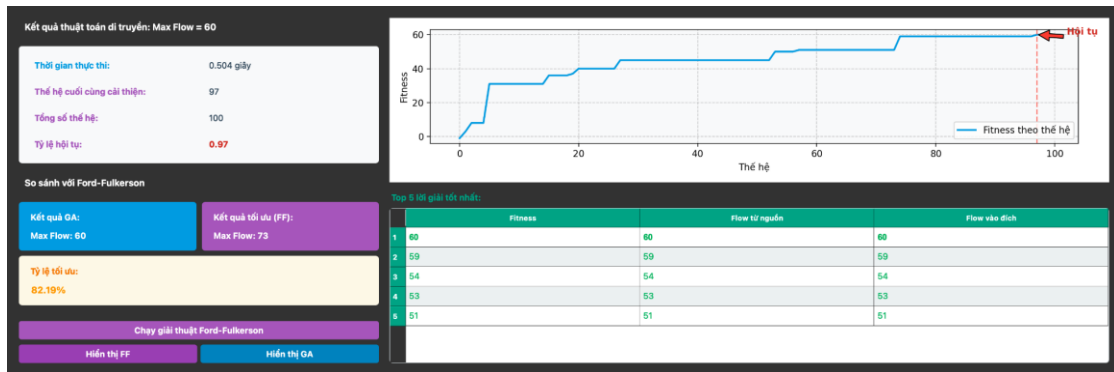
Thuật toán đã hoàn thành

Run GA
Stop GA

Hình 3. 3 – Bảng điều khiển tham số

Cuối cùng, khu vực phía dưới màn hình là bảng kết quả – nơi hiển thị các số liệu thống kê và biểu đồ đánh giá kết quả của giải thuật. Phần này bao gồm bảng thông số so sánh giữa GA và Ford–Fulkerson, biểu đồ tiến triển fitness theo thế hệ, và bảng

xếp hạng top 5 lời giải tốt nhất.

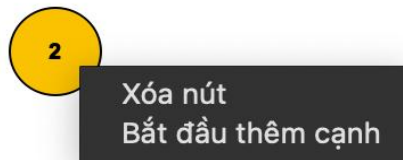


Hình 3. 4 - Khu vực bảng kết quả

3.3. Mô tả các chức năng

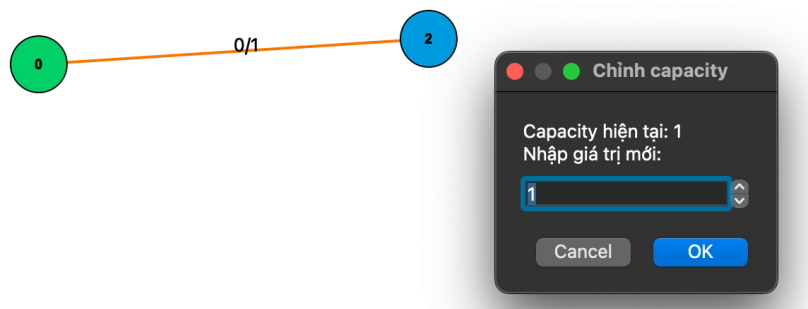
Ứng dụng hỗ trợ đầy đủ các chức năng cần thiết để người dùng không chỉ thao tác với đồ thị mà còn điều khiển và phân tích kết quả giải thuật.

Trong quá trình chỉnh sửa đồ thị, người dùng có thể click chuột trái vào vùng trống để tạo node mới, kéo thả để di chuyển node, và click chuột phải để tạo hoặc xóa cạnh.



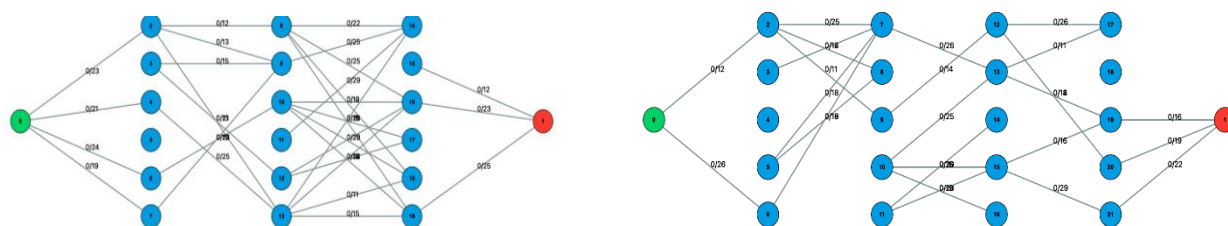
Hình 3. 5 - Lựa chọn khi click chuột phải vào node

Việc nhập giá trị capacity cho mỗi cạnh được thực hiện trực tiếp bằng cách click vào cạnh đó.



Hình 3. 6 - Bảng thay đổi giá trị capacity

Ngoài ra, người dùng có thể tạo đồ thị ngẫu nhiên theo tham số đầu vào như số layers, số node nằm trên layer và tỷ lệ tạo cạnh (Edge Prob). Người dùng có thể xóa toàn bộ đồ thị với nút Clear Graph.



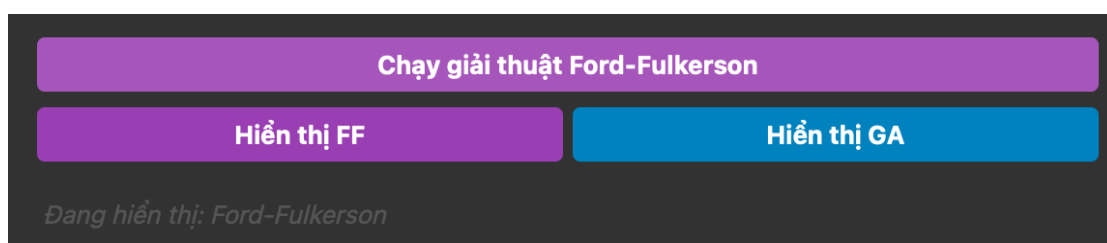
Hình 3. 7 - Đồ thị được tạo ngẫu nhiên với 3 layers và 6 nodes trên mỗi layers (bên trái) vs 4 layers và 5 nodes trên mỗi layers (bên phải)

Sau khi thiết lập đồ thị và cấu hình tham số, người dùng có thể khởi động giải thuật bằng nút “Run GA”. Quá trình thực thi sẽ được hiển thị theo thời gian thực, cho phép người dùng theo dõi sự thay đổi về độ thích nghi qua từng thế hệ. Nếu cần, người dùng có thể dừng thuật toán bất cứ lúc nào thông qua nút “Stop GA”.



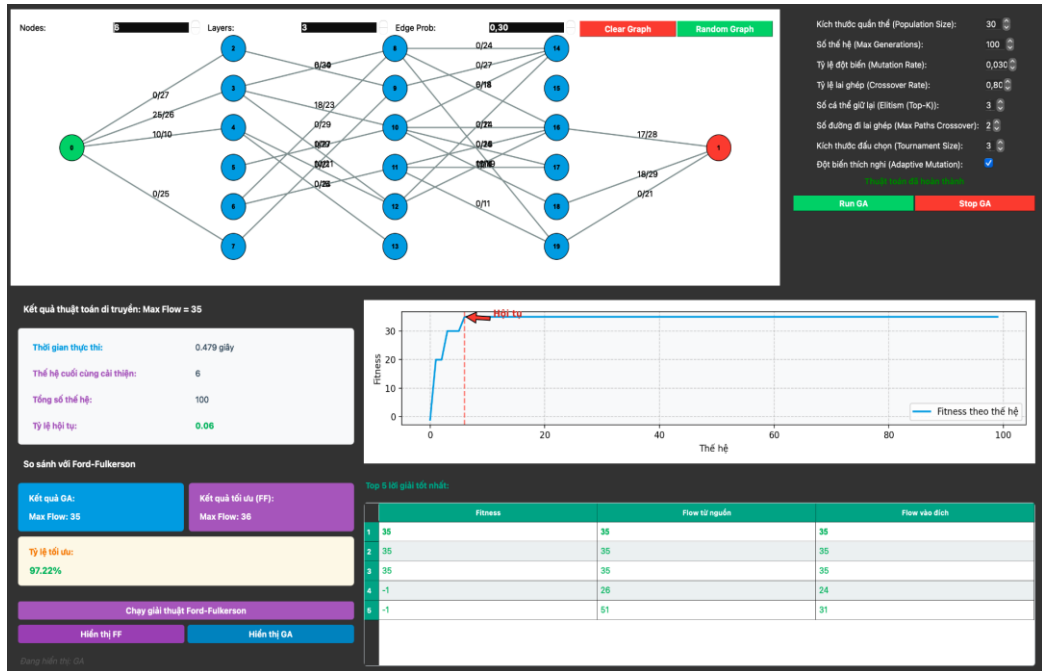
Hình 3. 8 - Nút chạy thuật toán GA và Nút dừng thuật toán

Kết quả cuối cùng được trình bày trực quan trong bảng kết quả. Tại đây, ngoài việc xem biểu đồ tiến triển fitness, theo dõi top các lời giải tốt nhất, người dùng có thể so sánh hiệu suất giải thuật GA với Ford–Fulkerson bằng cách nhấn nút Giải thuật Ford–Fulkerson. Đặc biệt, giao diện còn cho phép chuyển đổi hiển thị lời giải giữa hai thuật toán trên cùng một đồ thị, giúp người dùng đánh giá trực quan sự khác biệt giữa thuật toán heuristic và thuật toán tối ưu truyền thống.

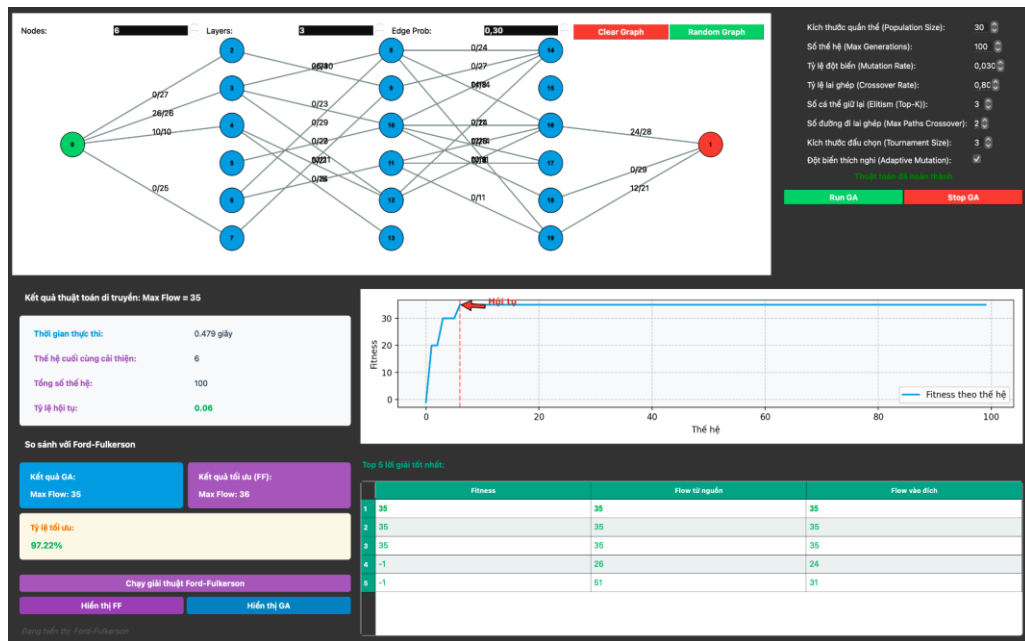


Hình 3. 9 - Nút chạy thuật toán FF và so sánh với GA

Ví dụ một graph lời giải của Genetic Algorithm so với Ford–Fulkerson Algorithm:



Hình 3. 10 - Lời giải GA



Hình 3. 11 - Lời giải FA

CHƯƠNG 4: THẢO LUẬN VÀ ĐÁNH GIÁ

4.1. Đánh giá thuật toán

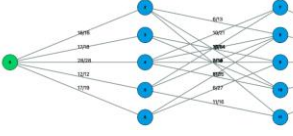
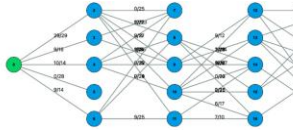
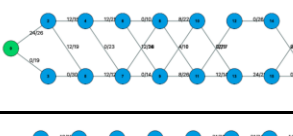

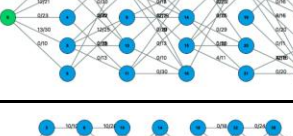
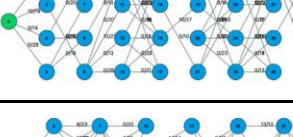
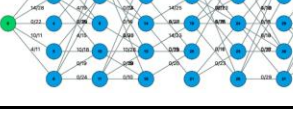
Để đánh giá hiệu quả của giải thuật di truyền trong việc giải bài toán luồng cực đại, nhóm đã tiến hành đo lường theo nhiều khía cạnh: độ chính xác so với giải pháp tối ưu Ford-Fulkerson, khả năng thoát khỏi tối ưu cục bộ, khả năng mở rộng và hiệu quả tính toán.

Ở đây nhóm thử với 9 loại đồ thị được chia làm 2 tập. Tập 1 là với các node ở mỗi layer đều là 5 và số layer từ 1 đến 5, tập 2 là số layer là 10 và số node ở mỗi layer chạy từ 2 đến 6. Và cấu hình các tham số không đổi là:

- Tỷ lệ sinh cạnh: 0,5
- Kích thước quần thể: 40
- Số thế hệ: 100
- Tỷ lệ lai ghép: 0,8
- Số cá thể giữ lại: 3
- Số đường đi lai ghép: 2
- Kích thước đầu chọn: 3
- Có bật đột biến thích nghi

Kết quả thu được (Bảng 4. 1 bên dưới):

Bảng 4. 1 - Kết quả chạy thuật toán trên các loại đồ thị

Loại đồ thị	Số đỉnh trên mỗi lớp	Số lớp	Tổng số node	Dạng đồ thị	Thời gian thực thi (s)	Tỷ lệ hội tụ	Tỷ lệ tối ưu
G502	5	2	10		0,517	0,91	96,77%
G504	5	4	20		0,811	0,84	80,28%
G210	2	10	20		0,947	0,07	100%
G310	3	10	30		1,714	0,14	100%
G506	5	6	30		27,329	0,08	48,48%
G410	4	10	40		3,737	0,13	64,52%
G508	5	8	40		200,835	0,08	51,11%

4.1.1. Độ sai lệch so và Hiệu năng khi mở rộng:

Một trong những tiêu chí đánh giá quan trọng là độ gần đúng của lời giải GA so với giải pháp tối ưu, được tính bằng công thức:

$$Tỷ\ lệ\ tối\ ưu = \frac{GA\ flow}{Ford-Fulkerson} \times 100\%$$

Qua các thử nghiệm, có thể thấy rằng trong các cấu hình đồ thị nhỏ (tổng số đỉnh dưới 30), GA đạt trên 95% độ chính xác, thậm chí có nhiều trường hợp tìm ra đúng lời giải tối ưu như G210, G310. Tuy nhiên, khi quy mô đồ thị tăng lên, đặc biệt từ 30 đỉnh

trở lên, hiệu suất thuật toán giảm rõ rệt. Các ví dụ như G506, G508, G510, và G610 cho thấy tỷ lệ tối ưu tụt xuống chỉ còn 33–50%, thậm chí có trường hợp bị timeout khi thực thi quá 5 phút.

Nguyên nhân đến từ chi phí hàm đánh giá fitness, cân bằng luồng, và không gian tìm kiếm tăng nhanh, khiến GA khó vượt qua các cực trị cục bộ và không đủ thời gian để hội tụ. Kết quả này phản ánh rằng thuật toán GA của nhóm xây dựng phù hợp cho các bài toán quy mô vừa và nhỏ, nhưng cần tối ưu hóa hoặc sử dụng tính toán song song để áp dụng cho đồ thị lớn.

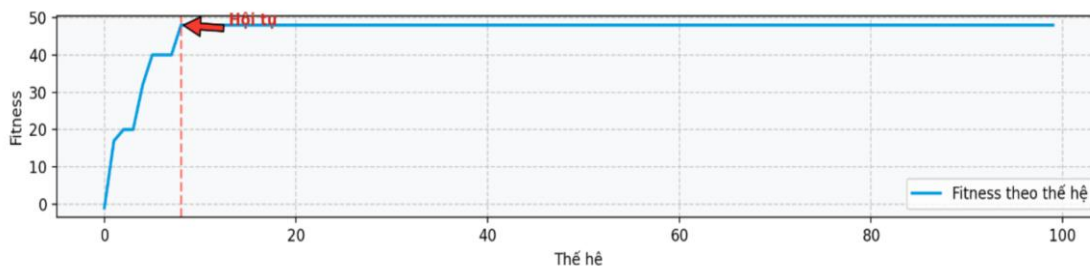
4.1.2. Khả năng thoát khỏi tối ưu cục bộ:

Để kiểm tra khả năng tránh bị mắc kẹt ở cực trị cục bộ, nhóm theo dõi biểu đồ tiến triển fitness qua các thế hệ. Các biểu đồ cho thấy thuật toán đôi lúc gặp hiện tượng "bình nguyên" (plateau), nơi fitness không tăng trong một số thế hệ liên tiếp. Tuy nhiên, nhờ cơ chế đột biến thích nghi và thay thế định kỳ cá thể kém, GA thường vượt qua được bình nguyên sau một số thế hệ, tiếp tục tìm ra lời giải tốt hơn.

Để định lượng khả năng hội tụ, nhóm sử dụng chỉ số tỷ lệ hội tụ, được định nghĩa là:

$$\text{Tỷ lệ hội tụ} = \frac{\text{Số thứ tự của thế hệ cuối cùng tìm thấy giải pháp tốt hơn}}{\text{Tổng số cá thể}}$$

Khi điểm hội tụ thấp thì có nghĩa là thuật toán nhanh chóng tìm thấy lời giải tốt và dừng cải thiện đáng kể từ rất sớm. Đây có thể chứng tỏ cho việc thuật toán hội tụ nhanh, sớm tìm ra vùng lời giải tốt. Nhưng mặt xấu là có thể là dấu hiệu hội tụ sớm, bị kẹt ở cực trị cục bộ, mất đa dạng quần thể quá nhanh như ta thấy ở G506.



Hình 4. 1 – Biểu đồ tiến triển fitness đồ thị G506

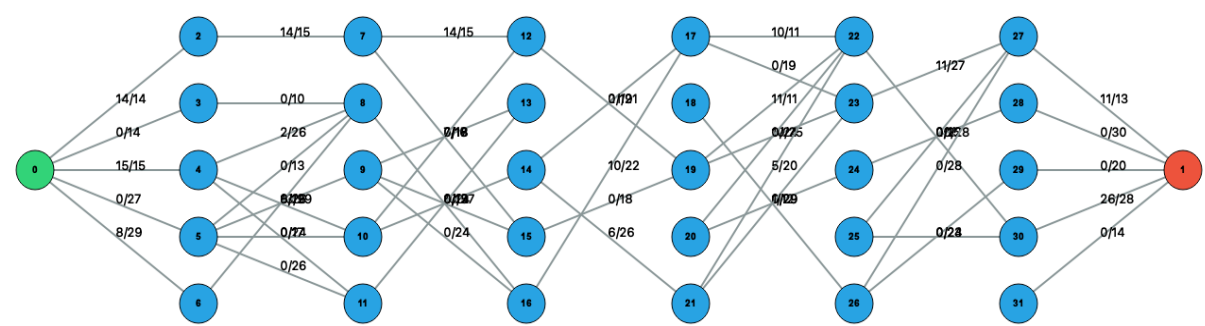
Ngược lại nếu điểm hội tụ cao thì chứng tỏ thuật toán liên tục tìm được cải thiện cho đến gần cuối quá trình chạy. Đây có thể là dấu hiệu hội tụ chậm. Nếu dừng đúng tại

max_generations, có thể thuật toán chưa khai thác hết tiềm năng, và thuật toán cần thêm nhiều thế hệ nữa để hoàn tất.

Tỷ lệ hội tụ phù hợp nhất là nằm ở trung bình, lúc này thuật toán có giai đoạn khám phá ban đầu, sau đó hội tụ ổn định. Đây là biểu hiện của sự cân bằng giữa khai thác và khám phá.

4.1.3. Ảnh hưởng của các tham số đến kết quả

Trong phần này, nhóm tiến hành khảo sát tác động của các tham số trong giải thuật di truyền (GA) đến chất lượng lời giải trên bài toán đồ thị. Đồ thị được sử dụng có cấu trúc gồm 6 lớp (layer), mỗi lớp chứa 5 node, với tỷ lệ sinh cạnh (edge generation rate) là 0.5.



Hình 4. 2 - Graph thực hiện đo lường thay đổi tham số

Nhóm tiến hành thử nghiệm với các cấu hình tham số khác nhau, bao gồm: kích thước quần thể, tỷ lệ đột biến, top-k được chọn lọc mỗi thế hệ. Các cấu hình được đánh giá dựa trên thời gian thực thi, tỷ lệ hội tụ (convergence rate), và tỷ lệ tìm ra lời giải tối ưu.

Bảng 4. 2 - Kết quả thay đổi tham số

Cấu hình	Kích thước quần thể	Tỷ lệ đột biến	Top-k	Thời gian	Tỷ lệ hội tụ	Tỷ lệ tối ưu
C1	30	0,05	3	1,629	0,1	92,68%
C2	30	0,1	3	1,688	0,14	92.68%
C3	50	0,1	5	2,882	0,06	95,12%
C4	50	0,2	5	3,008	0,13	92,68%
C5	50	0,05	10	1,805	0,14	97,56%

C6	50	0,1	10	1,790	0,18	97,56%
C7	50	0,2	10	2,170	0,07	100%

4.2. Ưu điểm và nhược điểm của thuật toán

Thuật toán di truyền (Genetic Algorithm – GA) được nhóm triển khai cho bài toán luồng cực đại thể hiện nhiều ưu điểm nổi bật cả về khả năng tối ưu hóa và tính mở rộng linh hoạt. Trước hết, thuật toán cho phép dễ dàng tùy biến và áp dụng cho nhiều biến thể của bài toán mạng luồng nhờ kiến trúc mô-đun rõ ràng, từ biểu diễn cá thể đến các bước chọn lọc, lai ghép và đột biến. Thêm vào đó, thông qua cơ chế `balance_flow`, mọi cá thể được sinh ra đều đảm bảo tuân thủ ràng buộc bảo toàn luồng tại các đỉnh trung gian, từ đó duy trì tính hợp lệ của lời giải trong suốt quá trình tiến hóa.

Một điểm mạnh đáng kể khác là khả năng tránh hội tụ sớm nhờ tích hợp cơ chế đột biến thích nghi – tự động tăng tỷ lệ đột biến khi không có cải thiện – kết hợp với chiến lược thay thế định kỳ các cá thể kém nhất trong quần thể. Điều này giúp thuật toán duy trì sự đa dạng và có thể thoát khỏi các vùng cực trị cục bộ. Trong thực nghiệm, thuật toán cho kết quả tốt, đạt được tỷ lệ tối ưu cao (lên đến 100%) ở các cấu hình hợp lý, đồng thời hội tụ ổn định qua các thế hệ. Sự kết hợp giữa elitism và tournament selection cũng góp phần giữ lại các cá thể tốt nhất trong khi vẫn thúc đẩy quá trình khám phá không gian lời giải mới.

Tuy nhiên, bên cạnh các ưu điểm, thuật toán cũng tồn tại một số hạn chế nhất định. Đáng chú ý là chi phí tính toán tăng đáng kể khi mở rộng kích thước quần thể, gây ảnh hưởng đến tốc độ thực thi nếu không có cấu hình phần cứng đủ mạnh. Ngoài ra, hiệu quả của GA phụ thuộc lớn vào việc lựa chọn và tinh chỉnh các tham số như kích thước quần thể, tỷ lệ đột biến hay số lượng cá thể elitism. Nếu không được cấu hình phù hợp, thuật toán có thể hội tụ chậm hoặc dễ mắc kẹt tại các lời giải chưa tối ưu. Một điểm cần lưu ý khác là việc cân bằng luồng – tuy cần thiết để đảm bảo ràng buộc – đôi khi có thể làm thay đổi cấu trúc cá thể quá mức, khiến cá thể con mất đi các đặc điểm di truyền quan trọng từ cha mẹ. Cuối cùng, như đa số các phương pháp heuristic, thuật toán không đảm bảo tìm được lời giải tối ưu toàn cục, đặc biệt trong trường hợp không biết trước giá trị cực đại lý thuyết, nên vẫn cần được so sánh với các thuật toán truyền thống như Edmonds-Karp để đánh giá chất lượng tương đối.

4.3. So sánh Genetic Algorithm và các giải thuật khác trong việc giải quyết bài toán

Trong quá trình triển khai và thực nghiệm, nhóm đã tiến hành so sánh giữa thuật toán di truyền (Genetic Algorithm – GA) và thuật toán Ford-Fulkerson (FF) – một trong những giải thuật cổ điển phổ biến nhất để giải bài toán luồng cực đại. Kết quả cho thấy rằng FF có khả năng tìm ra lời giải tối ưu với thời gian rất nhanh trong các đồ thị có cấu trúc rõ ràng và dung lượng cạnh nguyên. Đây là điều dễ hiểu bởi FF là thuật toán chính xác (exact algorithm) và đã được tối ưu hóa cho các bài toán mạng tuyến tính, không có nhiễu. Trong khi đó, thuật toán GA cho kết quả tiệm cận với lời giải tối ưu, có thể dao động nhẹ tùy thuộc vào tham số và quá trình tiến hóa. Tuy vậy, GA lại tỏ ra linh hoạt hơn trong các tình huống phức tạp như đồ thị có ràng buộc phi tuyến, các cạnh biến đổi theo thời gian, hay yêu cầu tối ưu đa mục tiêu (multi-objective).

So với các giải thuật truyền thống khác như Edmonds-Karp (phiên bản BFS của FF) hay Push-Relabel, GA có độ chính xác thấp hơn nhưng lại có lợi thế ở khả năng xử lý các biến thể phi chuẩn của bài toán. Trong khi các thuật toán cổ điển thường yêu cầu biểu diễn đồ thị cụ thể và có cấu trúc dữ liệu tương thích (như danh sách kề hoặc ma trận năng lực), GA có thể linh hoạt với bất kỳ cách biểu diễn nào miễn là xây dựng được hàm fitness phù hợp. Đặc biệt, với những bài toán luồng có thêm các yếu tố ràng buộc phức tạp như chi phí, độ trễ, hoặc yêu cầu ưu tiên luồng, các thuật toán cổ điển phải sửa đổi đáng kể hoặc không áp dụng được trực tiếp, trong khi GA vẫn có thể hoạt động tốt nhờ tính chất mở rộng theo hướng mô-đun.

Tuy nhiên, cũng cần nhìn nhận một cách khách quan rằng trong các trường hợp đồ thị nhỏ hoặc chuẩn hóa đơn giản, việc sử dụng GA là không cần thiết và không hiệu quả so với các thuật toán chính xác. GA phát huy thế mạnh rõ rệt khi bài toán trở nên khó mô hình hóa theo hướng quy hoạch tuyến tính hoặc khi ta cần một tập lời giải thay vì một kết quả duy nhất.

Tóm lại, trong khi Ford-Fulkerson và các thuật toán tuyến tính tỏ ra vượt trội về tốc độ và độ chính xác trong các trường hợp chuẩn hóa, thì Genetic Algorithm lại thể hiện sự linh hoạt, khả năng thích nghi cao và hiệu quả trong các bài toán phức tạp, phi tuyến hoặc khó xác định hàm mục tiêu rõ ràng. Do đó, lựa chọn giữa GA và các giải thuật khác cần căn cứ vào tính chất cụ thể của bài toán, mục tiêu thực tế và yêu cầu mở rộng trong ứng dụng.

CHƯƠNG 5: TỔNG KẾT ĐỀ TÀI

5.1. Hướng phát triển của đề tài

Mặc dù đề tài đã triển khai thành công một phiên bản thuật toán di truyền (Genetic Algorithm – GA) để giải bài toán luồng cực đại trên đồ thị có cấu trúc cố định, thực tế cho thấy rằng các bài toán mạng trong các hệ thống thật thường phức tạp và động hơn rất nhiều. Do đó, việc mở rộng và phát triển đề tài là cần thiết nhằm tăng tính thực tiễn, khả năng ứng dụng cũng như hiệu quả giải quyết các biến thể đa dạng của bài toán.

5.1.1. Mở rộng bài toán luồng chuẩn sang các biến thể thực tế

Một trong những hướng phát triển quan trọng là mở rộng mô hình bài toán luồng chuẩn sang các biến thể thực tế. Cụ thể, nhóm có thể tiếp tục nghiên cứu bài toán luồng chi phí thấp (Minimum Cost Maximum Flow), nơi mục tiêu không chỉ là tối đa hóa luồng mà còn tối thiểu hóa tổng chi phí đi qua mạng. Ngoài ra, bài toán với nhiều nguồn – nhiều đích (Multi-Source Multi-Sink) hoặc bài toán luồng thay đổi theo thời gian (dynamic flow) cũng là các trường hợp ứng dụng phổ biến trong mạng giao thông, logistics, hoặc truyền tải dữ liệu, đòi hỏi phải xây dựng lại cách biểu diễn cá thể và hàm đánh giá thích hợp.

5.1.2. Cải tiến và kết hợp với các chiến lược tiến hóa khác

Bên cạnh việc mở rộng bài toán, thuật toán cũng có thể được cải tiến bằng cách kết hợp với các chiến lược tiến hóa khác để tăng tốc độ hội tụ và chất lượng lời giải. Việc tích hợp local search sau crossover và mutation có thể giúp tinh chỉnh lời giải để đạt chất lượng tốt hơn trong thời gian ngắn. Ngoài ra, nhóm có thể nghiên cứu meta-GA hoặc Bayesian Optimization để tối ưu hóa tự động các tham số như tỷ lệ đột biến, kích thước quần thể, số thế hệ... nhằm giảm sự phụ thuộc vào việc tinh chỉnh thủ công. Đồng thời, việc thiết kế và thử nghiệm các toán tử crossover/mutation mới – như theo cụm, theo chu trình hoặc theo định tuyến ngẫu nhiên – cũng hứa hẹn mang lại sự đa dạng di truyền cao hơn và khả năng khám phá tốt hơn.

5.1.3. Ứng dụng thực tiễn vào các hệ thống mạng phức tạp

Một hướng phát triển đáng chú ý khác là ứng dụng mô hình vào các hệ thống thực tiễn, chẳng hạn như mạng giao thông đô thị, hệ thống phân phối năng lượng hoặc điều phối hàng hóa. Trong các hệ thống này, việc phân bổ luồng tối ưu có thể giúp giảm tắc nghẽn, tiết kiệm chi phí vận hành và nâng cao hiệu quả. Thuật toán GA có thể được tùy chỉnh để xử lý dữ liệu đầu vào thực, thậm chí triển khai trên các hệ thống tính toán phân

tán hoặc chạy trong môi trường thời gian thực, nơi đòi hỏi khả năng thích ứng và phản hồi nhanh.

Tổng kết lại, thuật toán di truyền là một hướng tiếp cận giàu tiềm năng trong việc giải quyết các bài toán tối ưu trên mạng luồng. Những hướng phát triển nêu trên không chỉ giúp nâng cao chất lượng giải pháp mà còn mở rộng khả năng ứng dụng thực tiễn, từ đó mang lại giá trị cả về mặt học thuật lẫn ứng dụng công nghệ trong tương lai.

5.2. Hạn chế trong ứng dụng

Mặc dù thuật toán di truyền (Genetic Algorithm – GA) nhóm xây dựng đã thể hiện được nhiều ưu điểm trong việc giải bài toán luồng cực đại, đặc biệt là khả năng thích nghi và xử lý bài toán có ràng buộc phức tạp, tuy nhiên vẫn tồn tại một số hạn chế nhất định trong quá trình ứng dụng thực tiễn.

Trước hết, GA không phải là một giải thuật chính xác, do đó không đảm bảo luôn tìm được lời giải tối ưu toàn cục. Trong các bài toán yêu cầu tính chính xác tuyệt đối – chẳng hạn trong các hệ thống tài chính, điều phối năng lượng quy mô lớn, hoặc trong môi trường có yêu cầu nghiêm ngặt về tính toàn vẹn dữ liệu – việc sử dụng GA có thể không phù hợp hoặc cần được kết hợp với các phương pháp kiểm chứng bổ sung. Thay vào đó, GA thường chỉ đưa ra lời giải gần đúng và có thể dao động nhẹ giữa các lần chạy, đặc biệt khi cấu hình tham số không được điều chỉnh tốt.

Thứ hai, hiệu suất và độ ổn định của GA phụ thuộc mạnh vào việc lựa chọn tham số như tỷ lệ đột biến, kích thước quần thể, số thế hệ và chiến lược chọn lọc. Việc tinh chỉnh những tham số này thường phải thực hiện thử công hoặc thử nghiệm lặp lại nhiều lần, gây tốn kém thời gian và tài nguyên, đặc biệt khi bài toán có quy mô lớn hoặc cấu trúc phức tạp.

Thứ ba, chi phí tính toán của GA có thể cao, nhất là khi kích thước quần thể lớn hoặc số thế hệ tăng lên để đạt độ hội tụ tốt hơn. Trong môi trường thực tế như các hệ thống nhúng, thiết bị IoT hoặc ứng dụng thời gian thực, việc sử dụng một giải pháp tiến hóa như GA có thể gặp khó khăn về mặt tài nguyên và thời gian phản hồi.

Bên cạnh đó, các bước hậu xử lý như cân bằng luồng (balance_flow) tuy cần thiết để đảm bảo tính hợp lệ của cá thể, nhưng có thể làm mất đi các đặc điểm di truyền quan trọng từ cha mẹ, ảnh hưởng đến hiệu quả tiến hóa và làm mờ tác dụng của các toán tử crossover và mutation.

Cuối cùng, GA cũng chưa thật sự mạnh trong các bài toán đơn giản, tuyến tính hoặc có cấu trúc rõ ràng, nơi các thuật toán cổ điển như Ford-Fulkerson, Edmonds-Karp hoặc Push-Relabel có thể cho kết quả tối ưu với độ phức tạp thấp hơn rất nhiều.

5.3. Tổng kết đề tài

Đề tài đã tập trung nghiên cứu và triển khai một giải pháp dựa trên thuật toán di truyền (Genetic Algorithm) nhằm giải quyết bài toán luồng cực đại (Maximum Flow) trên đồ thị. Thông qua việc xây dựng mô hình biểu diễn luồng dưới dạng từ điển, thiết kế các toán tử lai ghép theo đường đi (path-based crossover), cơ chế đột biến thích nghi và kiểm soát ràng buộc bằng hàm cân bằng luồng, nhóm đã hoàn thiện một hệ thống giải thuật vừa đảm bảo tính linh hoạt vừa duy trì được tính hợp lệ của lời giải.

Kết quả thực nghiệm cho thấy thuật toán đạt được chất lượng lời giải cao, gần tiệm cận với tối ưu, đặc biệt là khi được cấu hình tham số hợp lý. Các thử nghiệm thay đổi kích thước quần thể, tỷ lệ đột biến và số lượng elitism cũng giúp nhóm đánh giá được rõ ràng ảnh hưởng của từng yếu tố đến hiệu quả tiến hóa. Bên cạnh đó, việc so sánh với các thuật toán chính xác như Ford-Fulkerson cho thấy GA có ưu thế trong những bài toán mở rộng, có ràng buộc phức tạp hoặc khó mô hình hóa tuyến tính.

Mặc dù vậy, nhóm cũng nhận thức rõ một số hạn chế còn tồn tại, như thời gian tính toán phụ thuộc vào kích thước quần thể, sự nhạy cảm với cấu hình tham số, cũng như việc GA không đảm bảo tìm ra lời giải tối ưu tuyệt đối. Tuy nhiên, với tính mở rộng cao, khả năng kết hợp linh hoạt và tính thích nghi mạnh mẽ, thuật toán di truyền vẫn là một hướng tiếp cận hiệu quả và tiềm năng trong việc giải quyết các bài toán tối ưu hóa mạng.

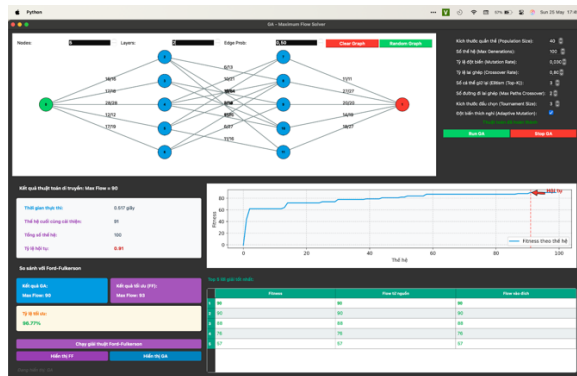
Từ những kết quả đạt được, đề tài không chỉ giúp nhóm củng cố kiến thức về trí tuệ nhân tạo, lập trình tối ưu và mô hình hóa bài toán, mà còn mở ra nhiều hướng phát triển tiếp theo trong nghiên cứu và ứng dụng thực tiễn. Đây là một trải nghiệm học thuật quý giá và có ý nghĩa sâu sắc đối với toàn nhóm.

TÀI LIỆU THAM KHẢO

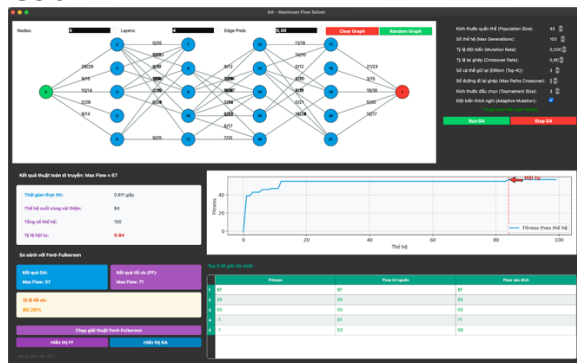
- cp-algorithms.com*. (n.d.). From *cp-algorithms.com*: https://cp-algorithms.com/graph/edmonds_karp.html
- Duy, P. T. (2017). *Genetic Algorithms (GAs) - Giải thuật di truyền*. From *duyphamdata.blogspot.com*:
<https://duyphamdata.blogspot.com/2017/11/genetic-algorithm-giai-thuat-di-truyen.html>
- Genetic algorithm*. (n.d.). From *en.wikipedia.org*:
https://en.wikipedia.org/wiki/Genetic_algorithm
- genetic-algorithms*. (n.d.). From *www.geeksforgeeks.org*:
<https://www.geeksforgeeks.org/genetic-algorithms/>
- Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*.
- Ola M. Surakhi, Mohammad Qatawneh, Hussein A. al Ofeishat. (2017). A Parallel Genetic Algorithm for Maximum Flow Problem. (*IJACSA*) *International Journal of Advanced Computer Science and Applications*.
- Ravindra K. Ahuja, Thomas L. Magnanti, James B. Orlin. (1993). *Network Flows Theory, Algorithms, and Applications*.
- Wikipedia. (2024, November 16). *Network flow problem*. From Wikipedia:
https://en.wikipedia.org/wiki/Network_flow_problem

PHỤ LỤC

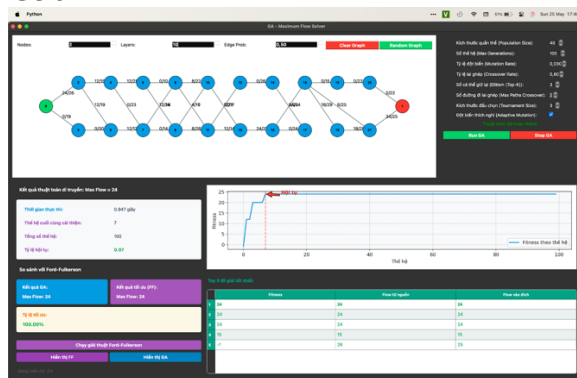
1. Hình ảnh kết quả thu được khi chạy thuật toán trong đo lường Chương 4



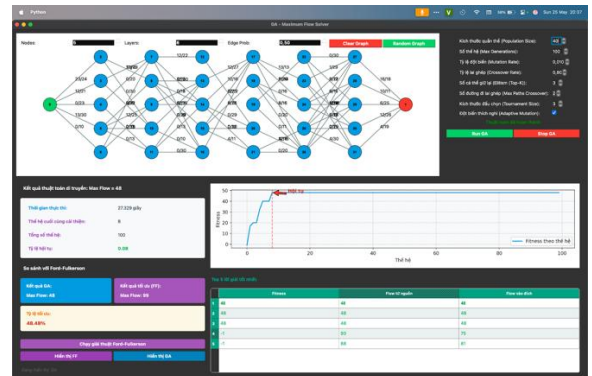
G502



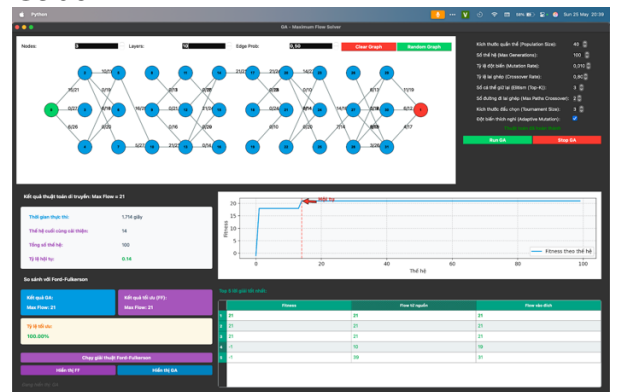
G504



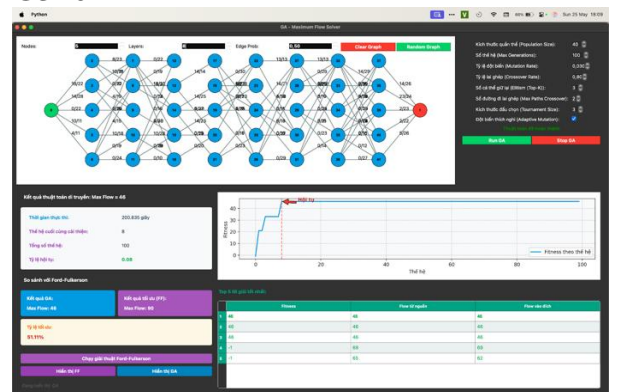
G210



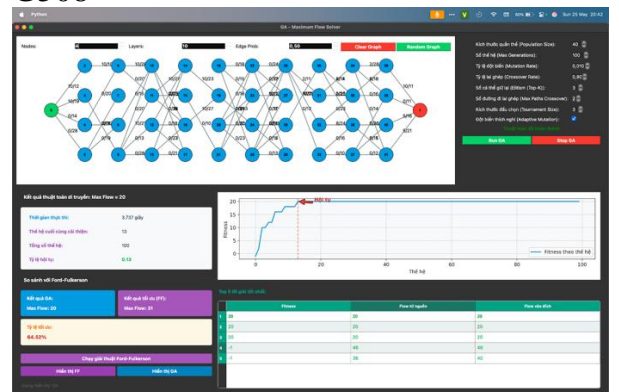
G506



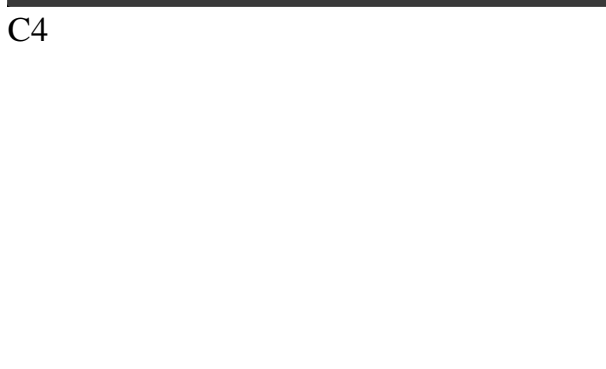
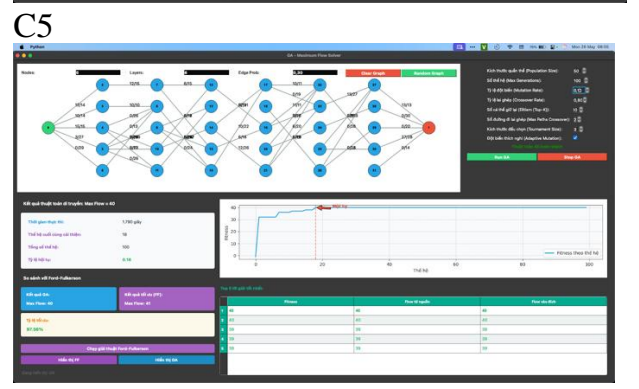
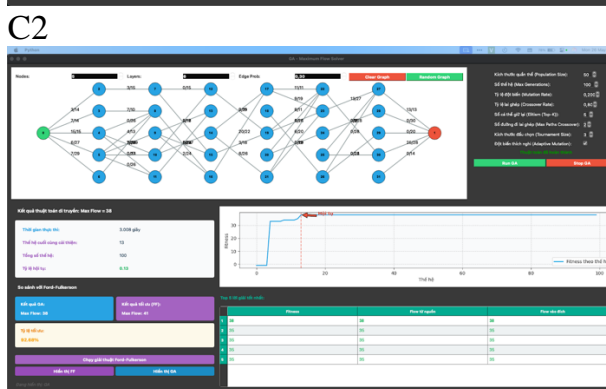
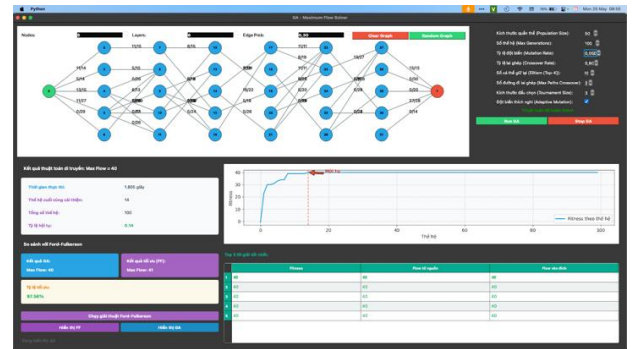
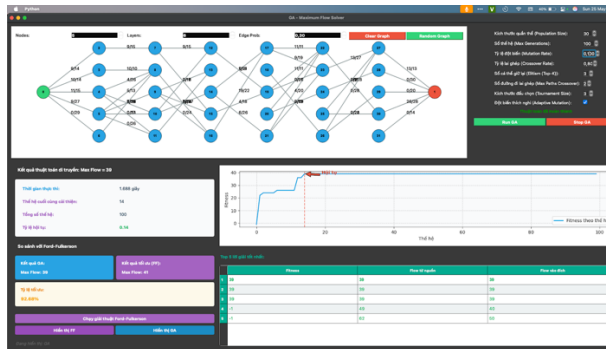
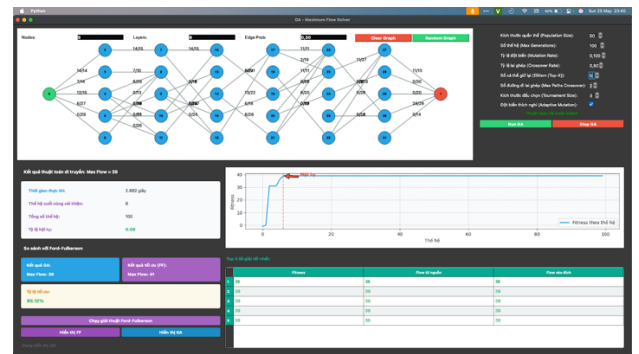
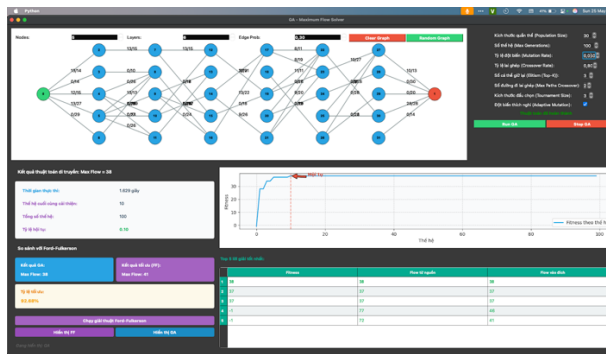
G310



G508



G410



2. Hướng dẫn chạy dự án

- Source code toàn bộ dự án: [Link GitHub](#)

- Cài đặt:

2.1. Clone repository:

```
git clone
https://github.com/hoaianthai345/Genetic_Algorithm_for_Maximum_Flow_Problem.git
```

2.2. Cài đặt các thư viện:

```
pip install -r requirements.txt
```

2.3. Chạy ứng dụng:

```
python main.py
```

- Cấu trúc dự án:

- main.py: Điểm khởi đầu của ứng dụng
- logic/: Các thuật toán.
 - ga_solver.py: Thuật toán di truyền để giải bài toán luồng cực đại
 - ford_fulkerson.py: Thuật toán Ford-Fulkerson
- ui/: Các thành phần giao diện người dùng
 - main_window.py: Cửa sổ chính của ứng dụng
 - graph_editor.py: Trình soạn thảo đồ thị tương tác
 - control_panel.py: Bảng điều khiển với các tham số thuật toán
 - result_panel.py: Hiển thị kết quả và biểu đồ

3. Phân công công việc

Thành viên nhóm	Nội dung báo cáo	Code
Thái Hoài An (Nhóm trưởng)	Chương 3: Thiết kế giao diện Tổng hợp, chỉnh sửa toàn văn	Thiết kế giao diện, Graph editor
Nguyễn Duy Tân	Chương 1: Tổng quan về bài toán Network Flow Problem	-Thiết kế cấu trúc dữ liệu cho đồ thị -Thuật toán FF
Nguyễn Thị Thùy Dương	Chương 2: Giải thuật Genetic Algorithm	Thiết kế thuật toán GA
Lê Vy	Chương 4: Thảo luận và đánh giá	Thiết kế đo lường kết quả, Result panel