Module 5 - Tuần 3 - Giải thuật di truyền (Genetic Algorithm)

Time-Series Team

Ngày 20 tháng 10 năm 2025

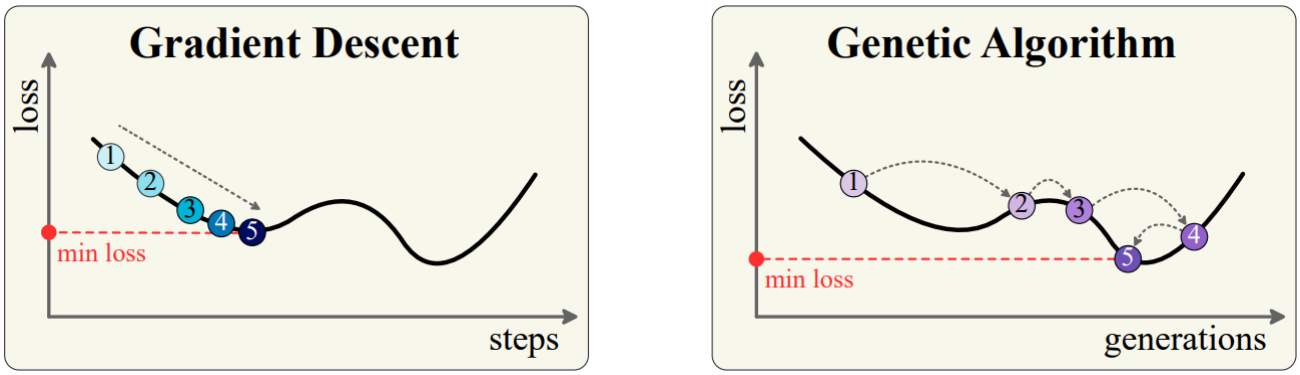
1 Giới thiệu

1.1 Tại sao dùng GA?

Trong những bài toán tối ưu cổ điển, khi biết rõ hàm mục tiêu và có thể tính đạo hàm, Gradient Descent (GD) là lựa chọn tự nhiên: ta cập nhật tham số theo hướng âm của gradient và kỳ vọng dần chạm tới điểm tối ưu.

Tuy nhiên, thực tế thường phức tạp hơn: hàm mục tiêu có thể không khả vi, nhiễu, hoặc chỉ có thể đánh giá như một "hộp đen" thông qua mô phỏng hoặc thử nghiệm. Khi đó, GD dễ bị kẹt ở cực trị cục bộ hoặc thậm chí không áp dụng được.

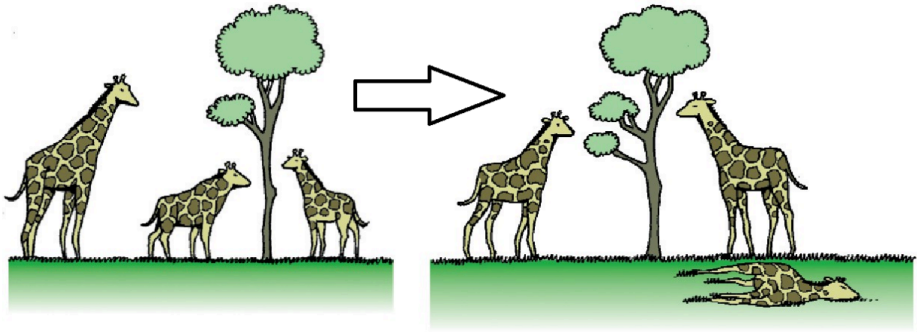
Một hướng tiếp cận không cần gradient là **Genetic Algorithm** (GA). Thay vì lần theo một quỹ đạo đơn lẻ như GD, GA duy trì quần thể nghiệm, đánh giá độ thích nghi, rồi cải thiện chúng qua việc chọn lọc, lai ghép và đột biến. Nhờ giữ đa dạng trong quần thể, GA có khả năng vượt qua các "thung lũng cục bộ" để tiếp tục khám phá không gian nghiệm.



Hình 1: Minh họa về quá trình tối ưu giữa Gradient Descent (GD) và Genetic Algorithm (GA).(Ảnh: AIVIETNAM)

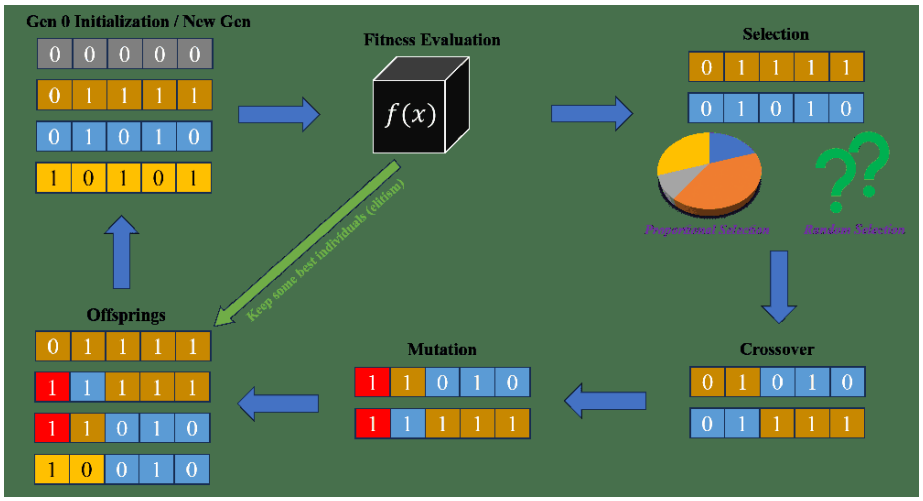
1.2 Từ tiến hóa tự nhiên đến giải thuật di truyền

Để hiểu giải thuật này một cách dễ dàng, chúng ta có thể liên hệ với quá trình tiến hóa của sinh vật trong tự nhiên qua nhiều thế hệ. Hiểu một cách đơn giản, trong một quần thể sinh vật, các cá thể buộc phải tiến hóa liên tục qua từng thế hệ để thích nghi với môi trường và tồn tại.



Hình 2: Minh họa về quá trình chọn lọc tự nhiên giữa loài hươu cao cổ. (Ảnh: creationscience4kids)

Giải thuật di truyền (GA) là một phương pháp tối ưu hóa ngẫu nhiên mô phỏng quá trình chọn lọc tự nhiên và di truyền sinh học thuật toán khởi đầu bằng một **quần thể** các cá thể (mỗi cá thể là một giải pháp ứng viên) được khởi tạo ngẫu nhiên. Sau đó, các cá thể này được đánh giá độ **thích nghi** (fitness) theo mục tiêu bài toán. Quá trình lặp của GA gồm các bước: chọn lọc các cá thể ưu tú, **lai ghép** (crossover) và **đột biến** (mutation) để tạo ra thế hệ mới.



Hình 3: Minh họa quy trình cơ bản của GA

Yếu tố **ngẫu nhiên** trong GA thể hiện ở khâu khởi tạo ban đầu, chọn cặp lai ghép và đột biến; nó giúp GA khám phá rộng không gian giải pháp và có khả năng thoát khỏi các cực tiểu cục bộ. Đặc điểm của GA là xử lý song song nhiều giải pháp (quần thể), điển hình với hình ảnh một quần thể các nhiễm sắc thể đồng thời phát triển qua các thế hệ

2 Cơ sở lý thuyết

Giải thuật di truyền (GA) xem bài toán tối ưu (ví dụ: tìm x để tối thiểu hóa hàm $f(x)$) như một quá trình chọn lọc tự nhiên. Thay vì làm việc với một "điểm" x duy nhất (như trong Gradient Descent), chúng ta làm việc với một **"quần thể"** (population) gồm nhiều giải pháp tiềm năng $P = \{x_1, x_2, \ldots, x_N\}$.

Mục tiêu của chúng ta là tìm x trong một không gian tìm kiếm S (tập ràng buộc) sao cho hàm mục tiêu (objective function) $f(x)$ đạt giá trị tối ưu (giả sử là *tối đa hóa* – maximization):

$$\max_{x \in S} f(x)$$

Giải thuật Di truyền không làm việc trực tiếp với x hay $f(x)$. Thay vào đó, nó "dịch" bài toán này sang ngôn ngữ của sinh học.

Chú thích:

- **Không gian tìm kiếm** (S) (tập ràng buộc của bài toán tối ưu) là không gian chứa mọi nghiệm có thể có của bài toán.
- **Quần thể** (P) là một nhóm nhỏ các nghiệm đang được xem xét trong tiến trình tiến hóa. Sau mỗi thế hệ, quần thể thay đổi — nhưng vẫn nằm trong cùng không gian tìm kiếm S

2.1 Các Định nghĩa Nền tảng

2.1.1 Nhiễm sắc thể (Chromosome)

Định nghĩa: Một **Nhiễm sắc thể** (Chromosome) là một cấu trúc dữ liệu đại diện cho một *giải pháp tiềm năng* (một cá thể – individual) trong không gian tìm kiếm S.

- **Tư duy Toán học:** Thay vì nói "một điểm x trong không gian S", chúng ta nói "một nhiễm sắc thể h".
- **Mã hóa (Encoding):** Đây là quá trình "dịch" từ x (toán học) sang h (GA). Đây là bước thiết yếu.
  - **Mã hóa Nhị phân (Binary):** Cổ điển nhất. x được biểu diễn dưới dạng một chuỗi bit.
  - **Mã hóa Số thực (Real-valued):** Hiện đại hơn. x được biểu diễn bằng một vector các số thực (thường dùng khi S là $\mathbb{R}^n$).

2.1.2 Gen và Allele

Định nghĩa: Một **Gen** là một thành phần cơ bản của nhiễm sắc thể, đại diện cho một phần của giải pháp. Giá trị mà một gen có thể nhận được gọi là **Allele**.

- **Tư duy Toán học:** Nếu nhiễm sắc thể là một vector, thì gen là một *phần tử* (element) của vector đó. Allele là *giá trị* của phần tử đó.

Ví dụ Cụ thể

Bài toán (Toán học): Tìm x là số nguyên trong đoạn $[0, 31]$ để tối đa hoá hàm $f(x) = x^2$.

- Không gian tìm kiếm $S = \{0, 1, \ldots, 31\}$.
- Hàm mục tiêu $f(x) = x^2$.

Mã hoá (Giải thuật Di truyền): Ta cần biểu diễn các số từ 0 đến 31.

Nhận thấy $2^5 = 32$. Vì vậy, chúng ta có thể dùng **mã hoá nhị phân 5-bit.**

- **Nhiễm sắc thể:** Một chuỗi 5 bit (ví dụ: $h = b_4b_3b_2b_1b_0$).
- **Gen:** Một bit tại một vị trí cụ thể (ví dụ: b_2 là một gen).
- **Allele:** Các giá trị mà gen có thể nhận. Ở đây là $\{0, 1\}$.

Hãy xem sự "phiên dịch" này:

Giải pháp (Toán học) x	**Mã hoá (GA)** h
$x = 13$	Nhiễm sắc thể: `01101`
$x = 21$	Nhiễm sắc thể: `10101`
$x = 31$	Nhiễm sắc thể: `11111`

Trong ví dụ này, nhiễm sắc thể `01101` có 5 gen. Gen thứ 3 (từ trái sang, giả sử) có allele là 1.

2.1.3 Hàm Thích nghi (Fitness Function)

Đây là định nghĩa quan trọng nhất, là cầu nối giữa bài toán và quá trình tiến hóa.

Định nghĩa: Một **Hàm Thích nghi** (Fitness Function), ký hiệu là $fitness(h)$, là một hàm gán một giá trị (thường là số thực không âm) cho mỗi nhiễm sắc thể h, đo lường "chất lượng" hay "độ tốt" của giải pháp mà nó đại diện.

Ý nghĩa: GA không trực tiếp tối ưu $f(x)$. Nó tối ưu $fitness(h)$. Quá trình chọn lọc (Selection) của GA dựa hoàn toàn vào hàm thích nghi: *Thích nghi càng cao, xác suất được chọn để "sinh sản" càng lớn.*

Liên hệ giữa Hàm Mục tiêu $f(x)$ và Hàm Thích nghi $fitness(h)$:

- **Trường hợp 1: Tối đa hóa (Maximization)**
  Nếu $f(x)$ luôn không âm (hoặc có thể được điều chỉnh để không âm), ta có thể đặt:

  $$fitness(h) = f(\text{decode}(h))$$

  (Trong đó decode(h) là hàm giải mã, ví dụ: "dịch" `01101` thành 13).

  Ví dụ: $f(x) = x^2$.

  - $h_1 =$ `01101` (tức là $x = 13$) $\Rightarrow fitness(h_1) = 13^2 = 169$.
  - $h_2 =$ `10101` (tức là $x = 21$) $\Rightarrow fitness(h_2) = 21^2 = 441$.

  Rõ ràng, cá thể h_2 "mạnh hơn", "tốt hơn" h_1 và sẽ có cơ hội được chọn cao hơn.

- **Trường hợp 2: Tối thiểu hóa (Minimization)**
  Đây là trường hợp phổ biến trong học máy (ví dụ: tối thiểu hóa hàm mất mát – loss function $L(\theta)$). GA về bản chất luôn là tối đa hóa (tìm cá thể "khỏe nhất"), do đó ta phải biến đổi (transform) hàm mục tiêu.

  Giả sử ta muốn $\min g(x)$ (với $g(x) \geq 0$). Ta có thể định nghĩa:

  $$fitness(h) = \frac{1}{1 + g(\text{decode}(h))}$$

  (Thêm 1 để tránh chia cho 0).

  Ý nghĩa: Khi $g(x)$ (mất mát) tiến về 0 (tối ưu), $fitness(h)$ sẽ tiến về 1 (tối đa). Cá thể có mất mát thấp nhất sẽ có độ thích nghi cao nhất.

2.2 Lý thuyết Lược đồ (Schema Theory)

Đây là lý thuyết do John Holland (cha đẻ của GA) phát triển vào những năm 1970 để giải thích một cách toán học tại sao và làm thế nào GA lại hoạt động.

2.2.1 Lược đồ (Schema)

Câu hỏi đầu tiên: GA thực sự xử lý cái gì?

Câu trả lời trực quan là "nhiễm sắc thể". Nhưng Holland cho rằng GA không chỉ xử lý N nhiễm sắc thể trong quần thể, mà nó còn *xử lý song song* một số lượng lớn các "mẫu" (pattern) hay "đặc điểm" (feature) mà các nhiễm sắc thể đó đại diện.

Định nghĩa: Một **Lược đồ** (Schema), ký hiệu là H, là một chuỗi (template) mô tả một tập hợp các nhiễm sắc thể có các đặc điểm chung tại một số vị trí gen nhất định.

Ta định nghĩa lược đồ bằng cách mở rộng bảng chữ cái của mình. Nếu ta dùng mã hoá nhị phân $\{0, 1\}$, thì bảng chữ cái của Lược đồ là $\{0, 1, *\}$.

- **0:** Gen phải là 0 tại vị trí này.
- **1:** Gen phải là 1 tại vị trí này.
- *** (wildcard – "không quan tâm"):** Gen có thể là 0 hoặc 1 tại vị trí này.

Ví dụ: Giả sử chiều dài nhiễm sắc thể là $L = 5$.

- Lược đồ $H_1 = 10 * **$
  Nó đại diện cho tập hợp tất cả các nhiễm sắc thể bắt đầu bằng `10`.
  Ví dụ các nhiễm sắc thể "khớp" (match) với H_1: `10000`, `10101`, `10111`, `10010`, ...

- Lược đồ $H_2 = *1 * 0*$
  Nó đại diện cho tập hợp tất cả các nhiễm sắc thể có `1` ở vị trí thứ 2 và `0` ở vị trí thứ 4.
  Ví dụ: `01000`, `11101`, `01100`, ...

Ý nghĩa: Một lược đồ thực chất là một *siêu phẳng* (hyperplane) trong không gian tìm kiếm $\{0, 1\}^L$. GA, bằng cách xử lý các nhiễm sắc thể, thực chất đang lấy mẫu và so sánh hiệu suất của hàng tỷ các siêu phẳng này cùng một lúc. Điều này được gọi là **Tính song song ngầm (Implicit Parallelism)**.

2.2.2 Các thuộc tính của Lược đồ

Để phân tích một lược đồ, chúng ta cần 2 thước đo:

Bậc (Order), $o(H)$

- **Định nghĩa:** Là số lượng các vị trí *cố định* (không phải *) trong lược đồ.
- **Ví dụ:**
$$o(10 * **) = 2, \quad o(*1 * 0*) = 2, \quad o(10101) = 5.$$
- **Ý nghĩa:** Bậc càng thấp, lược đồ càng chung chung. Bậc càng cao, lược đồ càng cụ thể.

Chiều dài (Defining Length), $d(H)$

- **Định nghĩa:** Là khoảng cách giữa vị trí cố định đầu tiên và vị trí cố định cuối cùng.
- **Ví dụ:**
  - $H_1 = 10 * ** \Rightarrow$ Vị trí cố định: 1 và 2. $\Rightarrow d(H_1) = 2 - 1 = 1$.
  - $H_2 = *1 * 0* \Rightarrow$ Vị trí cố định: 2 và 4. $\Rightarrow d(H_2) = 4 - 2 = 2$.
  - $H_3 = 1 * * * 1 \Rightarrow$ Vị trí cố định: 1 và 5. $\Rightarrow d(H_3) = 5 - 1 = 4$.
- **Ý nghĩa:** $d(H)$ càng nhỏ (schema *ngắn*), các gen định nghĩa nó càng gần nhau. $d(H)$ càng lớn (schema *dài*), các gen định nghĩa nó càng xa nhau.

2.2.3 Định lý Lược đồ (Schema Theorem)

Bây giờ là phần cốt lõi.

Định lý này cho ta một *giới hạn dưới* về số lượng nhiễm sắc thể thuộc một lược đồ H ở thế hệ tiếp theo $t + 1$, dựa trên số lượng của nó ở thế hệ t.

Ký hiệu:

- $m(H, t)$: Số lượng nhiễm sắc thể trong quần thể tại thế hệ t khớp với lược đồ H.
- $\bar{f}(H, t)$: Độ thích nghi *trung bình* của tất cả các nhiễm sắc thể khớp với H tại thế hệ t.
- $\bar{f}(t)$: Độ thích nghi *trung bình* của toàn bộ quần thể tại thế hệ t.
- $p_d(H)$: Xác suất lược đồ H bị *phá vỡ (disrupted)* bởi các toán tử di truyền (Lai ghép và Đột biến).

Định lý Lược đồ (Holland, 1975)

Số lượng *kỳ vọng* của H ở thế hệ $t + 1$ (chỉ xét chọn lọc, lai ghép 1-điểm, và đột biến) thỏa mãn:

$$E[m(H, t + 1)] \geq m(H, t) \cdot \frac{\bar{f}(H, t)}{\bar{f}(t)} \cdot [1 - p_d(H)]$$

Ý nghĩa: Định lý này cho thấy rằng những lược đồ có:

- Độ thích nghi trung bình cao hơn mức trung bình chung ($\bar{f}(H, t) > \bar{f}(t)$),
- Và có xác suất bị phá vỡ thấp ($p_d(H)$ nhỏ),

sẽ có xu hướng **phát triển số lượng** qua các thế hệ — tức là "những mẫu gen tốt" sẽ được khuếch đại trong quần thể.

2.2.4 Phân tích Định lý (Ý nghĩa thực sự)

Công thức này trông có vẻ phức tạp, nhưng nó mô tả chính xác cuộc "đấu tranh sinh tồn" của các lược đồ. Hãy chia nó thành hai phần:

Phần 1: Toán tử Chọn lọc (Selection)

$$m(H,t) \cdot \frac{\bar{f}(H,t)}{\bar{f}(t)}$$

Đây là tác động của **chọn lọc tự nhiên** (giả sử dùng chọn lọc *Roulette Wheel* – tỷ lệ với fitness).

- $\frac{\bar{f}(H,t)}{\bar{f}(t)}$ là *độ thích nghi tương đối* của lược đồ H.
- **Ý nghĩa:**
  - Nếu $\bar{f}(H,t) > \bar{f}(t)$ (tức là H là một lược đồ "tốt", có độ thích nghi trên trung bình), thì tỷ số này > 1. Số lượng của H sẽ tăng theo cấp số nhân qua các thế hệ (giống như lãi suất kép).
  - Nếu $\bar{f}(H,t) < \bar{f}(t)$ (tức là H là một lược đồ "tệ", dưới trung bình), tỷ số này < 1. H sẽ biến mất dần khỏi quần thể.

Đây chính là động lực chính của GA: **Các mẫu tốt được nhân lên.**

Phần 2: Toán tử Phá vỡ (Disruption)

$$[1 - p_d(H)]$$

Đây là **xác suất sống sót** (*survival probability*) của lược đồ H khi đối mặt với Lai ghép và Đột biến.

Các toán tử này rất quan trọng để tạo ra cái mới, nhưng chúng cũng *phá vỡ* các mẫu tốt đang tồn tại.

Hãy ước tính $1 - p_d(H)$:

a. Phá vỡ do Lai ghép (Crossover):

- Giả sử ta dùng Lai ghép 1-điểm với xác suất p_c.
- Một lược đồ H bị phá vỡ *chỉ khi* điểm lai ghép rơi vào **bên trong** chiều dài định nghĩa $d(H)$ của nó.
- Độ dài vùng "nguy hiểm" này là $d(H)$. Tổng số điểm cắt là $L - 1$.
- Xác suất bị phá vỡ $\approx p_c \cdot \frac{d(H)}{L-1}$ (đây là một ước tính chặn trên).
- Xác suất *sống sót* qua lai ghép $\geq 1 - p_c \cdot \frac{d(H)}{L-1}$.
- **Ý nghĩa:** Lược đồ càng ngắn ($d(H)$ nhỏ), càng ít bị lai ghép phá vỡ.

b. Phá vỡ do Đột biến (Mutation):

- Giả sử xác suất đột biến mỗi gen là p_m (rất nhỏ).

- Lược đồ H bị phá vỡ nếu bất kỳ gen cố định nào trong $o(H)$ gen của nó bị đột biến.
- Xác suất 1 gen *không* bị đột biến là $(1 - p_m)$.
- Xác suất tất cả $o(H)$ gen *không* bị đột biến là $(1 - p_m)^{o(H)}$.
- Vì $p_m \ll 1$, ta có thể xấp xỉ $(1 - p_m)^{o(H)} \approx 1 - o(H) \cdot p_m$.
- **Ý nghĩa:** Lược đồ có bậc càng thấp ($o(H)$ nhỏ), càng ít bị đột biến phá vỡ.

Kết hợp lại, ta có Định lý Lược đồ (dạng đầy đủ):

$$E[m(H, t+1)] \geq m(H, t) \cdot \frac{\bar{f}(H, t)}{\bar{f}(t)} \cdot \left[1 - p_c \frac{d(H)}{L-1}\right] \cdot \left[(1 - p_m)^{o(H)}\right]$$

—

Ý nghĩa tổng quát: Những lược đồ *ngắn* ($d(H)$ nhỏ), *bậc thấp* ($o(H)$ nhỏ), và *độ thích nghi cao* ($\bar{f}(H, t)$ lớn) sẽ được nhân lên theo cấp số nhân trong quần thể qua các thế hệ.

2.2.5 Ý nghĩa của Định lý: Giả thuyết Khối Xây dựng (Building Block Hypothesis)

Đây là kết luận quan trọng nhất từ định lý này.

Định lý Lược đồ cho chúng ta biết loại lược đồ nào sẽ "thịnh vượng" và được nhân lên trong quần thể:

1. **Thích nghi cao:** $\bar{f}(H, t) \gg \bar{f}(t)$ (có độ thích nghi trên trung bình).
2. **Ngắn:** $d(H)$ nhỏ (ít bị lai ghép phá vỡ).
3. **Bậc thấp:** $o(H)$ nhỏ (ít bị đột biến phá vỡ).

Những lược đồ thỏa mãn 3 điều kiện này được gọi là **Khối Xây dựng (Building Blocks)**.

Giả thuyết Khối Xây dựng: Giải thuật Di truyền hoạt động bằng cách tìm ra các "khối xây dựng" ngắn, bậc thấp, có độ thích nghi cao ở các thế hệ đầu. Sau đó, nó *kết hợp* (combine) các khối xây dựng nhỏ này lại (thông qua toán tử Lai ghép) để tạo ra các khối xây dựng lớn hơn, bậc cao hơn, và tốt hơn, cho đến khi hội tụ đến giải pháp tối ưu.

Ví dụ Cụ thể

Quay lại bài toán $\max f(x) = x^2$ trên $[0, 31]$. Mã hoá 5-bit $b_4\, b_3\, b_2\, b_1\, b_0$. Giải pháp tối ưu là $x = 31$ (`11111`).

- Xét 2 lược đồ tại thế hệ t:
  - $H_1 = 1****$ (Schema "tốt", $x \geq 16$).
  - $H_2 = *0***$ (Schema "kém", $x < 16$ hoặc $24 \leq x < 28$).
- Giả sử tại t, quần thể có $\bar{f}(t) = 200$.
- Các cá thể trong H_1 ($1****$) có $x \geq 16$, $f(x) \geq 16^2 = 256$. Vậy $\bar{f}(H_1, t)$ chắc chắn > 200 (ví dụ $\bar{f}(H_1, t) = 400$).
- Các cá thể trong H_2 ($*0***$) có giá trị $f(x)$ rải rác. $\bar{f}(H_2, t)$ có thể nhỏ hơn 200 (ví dụ $\bar{f}(H_2, t) = 150$).

Phân tích H_1 (Khối xây dựng):

- $o(H_1) = 1$ (bậc thấp).
- $d(H_1) = 0$ (siêu ngắn).
- $\frac{\bar{f}(H_1, t)}{\bar{f}(t)} = \frac{400}{200} = 2.$
- Xác suất sống sót ≈ 1 (lai ghép không phá vỡ vì $d = 0$, đột biến rất ít ảnh hưởng vì $o = 1$).
- **Kết quả:** $m(H_1, t+1) \approx m(H_1, t) \cdot 2$. Số lượng cá thể bắt đầu bằng bit `1` sẽ **tăng gấp đôi** ở thế hệ sau.

Phân tích H_2 (Schema "kém"):

- $o(H_2) = 1$ (bậc thấp).
- $d(H_2) = 0$ (siêu ngắn).
- $\frac{\bar{f}(H_2, t)}{\bar{f}(t)} = \frac{150}{200} = 0.75.$
- **Kết quả:** $m(H_2, t+1) \approx m(H_2, t) \cdot 0.75$. Số lượng cá thể có bit `0` ở vị trí thứ 2 sẽ **giảm 25%**.

GA đang tự động "học" rằng bit `1` ở vị trí đầu tiên ($1 * * * *$) là một "khối xây dựng" tốt và nó nhân bản khối đó lên. Tương tự, nó "học" được $*1 * **$ cũng là khối xây dựng tốt. Bằng cách lai ghép $1 * * * *$ và $*1 * **$, nó tạo ra `11***`, một khối xây dựng tốt hơn nữa.

2.3 Lý thuyết Chuỗi Markov

Ta đã đi qua phần "động lực học" (dynamics) với Định lý Lược đồ. Bây giờ, ta sẽ sang phần "phân tích hội tụ" (convergence analysis) để trả lời câu hỏi: **"Liệu GA có chắc chắn tìm ra giải pháp tối ưu toàn cục không?"**

Để làm điều này, ta cần một công cụ toán học mạnh mẽ hơn: **Lý thuyết Chuỗi Markov** (Markov Chain Theory).

2.3.1 Tại sao Định lý Lược đồ là chưa đủ?

Định lý Lược đồ rất tốt để giải thích *tại sao GA hiệu quả* — nó cho thấy các "khối xây dựng" (*building blocks*) tốt được nhân lên. Tuy nhiên:

1. Nó là một **giới hạn dưới**, không phải một mô hình chính xác. Nó không mô tả đầy đủ sự phức tạp, ví dụ như làm thế nào các toán tử *tạo ra* các lược đồ tốt mới.
2. Nó **không chứng minh được sự hội tụ**. Chỉ vì các khối tốt được nhân lên, không có nghĩa là chúng sẽ được kết hợp đúng cách để tạo ra giải pháp tối ưu toàn cục.

Để chứng minh sự hội tụ, chúng ta phải mô hình hóa toàn bộ quá trình GA như một hệ thống ngẫu nhiên (*stochastic system*).

—

2.3.2 Chuỗi Markov

Về bản chất, **Chuỗi Markov** là một công cụ toán học dùng để mô hình hóa các hệ thống ngẫu nhiên (stochastic) thay đổi theo thời gian.

Một **Chuỗi Markov** (rời rạc, hữu hạn) được định nghĩa bởi 3 thành phần cốt lõi:

Không gian Trạng thái (State Space) **Không gian Trạng thái**, ký hiệu là S, là một tập hợp *hữu hạn* gồm tất cả các trạng thái có thể có của hệ thống. Hệ thống phải ở *một và chỉ một* trạng thái tại mỗi bước thời gian.

- **Ký hiệu:** $S = \{s_1, s_2, \ldots, s_k\}$, trong đó k là tổng số trạng thái.
- X_t là một biến ngẫu nhiên đại diện cho trạng thái của hệ thống tại thời điểm t. $X_t \in S$.

Ví dụ (Thời tiết):

- $S = \{\text{Nắng}, \text{Mưa}\}$
- $X_0 = \text{Nắng}$ (Hôm nay trời nắng)
- $X_1 = \text{Mưa}$ (Ngày mai trời mưa)

Tính chất Markov (The Markov Property) Đây là định nghĩa quan trọng nhất, còn được gọi là **"Tính chất Không nhớ" (Memorylessness)**.

Phát biểu: Trạng thái *tương lai* của hệ thống (X_{t+1}) chỉ phụ thuộc duy nhất vào trạng thái *hiện tại* (X_t), và *hoàn toàn độc lập* với tất cả các trạng thái trong *quá khứ* ($X_{t-1}, X_{t-2}, \ldots$).

Phát biểu Toán học: Với mọi $t \geq 0$ và mọi chuỗi trạng thái $i, j, i_{t-1}, \ldots, i_0 \in S$:

$$P(X_{t+1} = j \mid X_t = i, X_{t-1} = i_{t-1}, \ldots, X_0 = i_0) = P(X_{t+1} = j \mid X_t = i)$$

Diễn giải: "Biết được hiện tại, quá khứ không còn cung cấp thêm thông tin gì về tương lai."

Xác suất Chuyển (Transition Probabilities) Đây là các quy tắc điều khiển cách hệ thống di chuyển.

Một Chuỗi Markov được gọi là **"đồng nhất theo thời gian"** (*time-homogeneous*) nếu các xác suất chuyển này không thay đổi theo thời gian. (Đây là trường hợp hầu hết chúng ta quan tâm.)

Định nghĩa: Xác suất chuyển từ trạng thái i đến trạng thái j, ký hiệu là p_{ij}, là xác suất mà hệ thống sẽ ở trạng thái j tại bước $t+1$, *biết rằng nó đang ở trạng thái i tại bước t.*

$$p_{ij} = P(X_{t+1} = j \mid X_t = i)$$

Các xác suất này phải thỏa mãn hai điều kiện:

1. $p_{ij} \geq 0$ (Chúng là các xác suất)
2. $\sum_{j \in S} p_{ij} = 1$ (Từ trạng thái i, hệ thống phải đi đến một trạng thái nào đó trong S, kể cả chính nó)

Ma trận Chuyển (Transition Matrix) Đây là một cách ngắn gọn để biểu diễn tất cả các quy tắc của hệ thống.

Định nghĩa: Ma trận Chuyển, ký hiệu là P (hoặc M), là một ma trận $k \times k$ (với $k = |S|$) mà phần tử ở hàng i, cột j chính là xác suất chuyển p_{ij}.

$$P = \begin{bmatrix} p_{11} & p_{12} & \cdots & p_{1k} \\ p_{21} & p_{22} & \cdots & p_{2k} \\ \vdots & \vdots & \ddots & \vdots \\ p_{k1} & p_{k2} & \cdots & p_{kk} \end{bmatrix}$$

Tính chất của P:

- Tất cả các phần tử ≥ 0.
- Tổng của mỗi hàng **bằng 1**. (Đây được gọi là *ma trận ngẫu nhiên* – stochastic matrix).

Ví dụ (Thời tiết):

- $S = \{1 : \text{Nắng},\ 2 : \text{Mưa}\}$
- Giả sử:

$$\begin{aligned} P(\text{mai Mưa} \mid \text{nay Nắng}) &= p_{12} = 0.2 \\ P(\text{mai Nắng} \mid \text{nay Nắng}) &= p_{11} = 0.8 \\ P(\text{mai Mưa} \mid \text{nay Mưa}) &= p_{22} = 0.6 \\ P(\text{mai Nắng} \mid \text{nay Mưa}) &= p_{21} = 0.4 \end{aligned}$$

- Khi đó ma trận chuyển là:

$$P = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix}$$

(Hàng 1: từ "Nắng"; Hàng 2: từ "Mưa"
Cột 1: tới "Nắng"; Cột 2: tới "Mưa")

2.3.3 Giải thuật Di truyền như một Chuỗi Markov

Hãy xem xét một GA "chuẩn" (*Canonical GA*) có kích thước quần thể N và chiều dài nhiễm sắc thể L.

1. **Trạng thái (State):** Một "trạng thái" của hệ thống *không phải* là một nhiễm sắc thể. Một trạng thái S_t chính là *toàn bộ quần thể* tại thế hệ t.

$$S_t = P_t = \{h_1, h_2, \ldots, h_N\}$$

2. **Không gian Trạng thái (Ω):**
   - Đây là tập hợp của tất cả các quần thể có thể có.
   - Số lượng nhiễm sắc thể có thể có là $K = 2^L$.
   - Không gian trạng thái Ω là tập hợp tất cả các cách chọn N nhiễm sắc thể từ K loại. Kích thước của không gian này là hữu hạn nhưng *siêu khổng lồ*: $|\Omega| = K^N = (2^L)^N$.
3. **Tính chất Markov (Markov Property):**

- Quần thể ở thế hệ tiếp theo P_{t+1} được tạo ra *chỉ dựa trên* quần thể hiện tại P_t thông qua các toán tử ngẫu nhiên (chọn lọc, lai ghép, đột biến).

$$P(P_{t+1}|P_t, P_{t-1}, \ldots, P_0) = P(P_{t+1}|P_t)$$

- Quá trình này không "nhớ" lịch sử (trừ những gì đã được lưu trữ trong P_t).
- Do đó, GA "chuẩn" là một **Chuỗi Markov hữu hạn**.

4. **Ma trận Chuyển (Transition Matrix), M:**

- Đây là một ma trận khổng lồ, trong đó M_{ij} là xác suất để chuyển từ trạng thái (quần thể) i sang trạng thái (quần thể) j trong một thế hệ.

2.3.4 Định lý Hội tụ Toàn cục (Global Convergence Theorem)

Câu hỏi là: *Chuỗi Markov này có hội tụ không? Và nó hội tụ về đâu?*

Câu trả lời phụ thuộc vào một chi tiết cực kỳ quan trọng: **Ta có dùng Elitism (Giữ lại tinh hoa) hay không?**

Elitism là kỹ thuật *đảm bảo* cá thể tốt nhất của thế hệ t được sao chép sang thế hệ $t + 1$.

Định lý Hội tụ Toàn cục

Một Giải thuật Di truyền sử dụng **Elitism** và có **toán tử Đột biến** ($p_m > 0$) sẽ hội tụ **chắc chắn (với xác suất 1)** đến trạng thái chứa giải pháp tối ưu toàn cục khi $t \to \infty$.

2.3.5 Chứng minh (Tại sao Elitism + Mutation = Hội tụ)

Để chứng minh, ta sử dụng các khái niệm từ chuỗi Markov:

Trạng thái Hấp thụ (Absorbing State):

- Gọi h^* là giải pháp tối ưu toàn cục (*global optimum*).
- Hãy xem xét bất kỳ trạng thái quần thể P^* nào có chứa ít nhất một bản sao của h^* (ví dụ $P^* = \{h^*, h_2, \ldots, h_N\}$).
- Vì chúng ta dùng **Elitism**, cá thể tốt nhất (h^*) sẽ luôn được giữ lại.
- Do đó, một khi quần thể đã đi vào một trạng thái P^* có chứa h^*, nó sẽ *không bao giờ rời khỏi* nhóm các trạng thái đã chứa h^* nữa.

Kết luận 1: Tập hợp tất cả các trạng thái có chứa h^* là một *tập hấp thụ* (*absorbing set*).

Khả năng đi đến (Reachability):

- Bây giờ, giả sử chúng ta đang ở một trạng thái P_j *không chứa* h^*.
- Câu hỏi: Liệu có khả năng đi từ P_j đến một trạng thái P_i có chứa h^* không?
- Câu trả lời là **CÓ**, miễn là $p_m > 0$. Vì **Đột biến** ($p_m > 0$) đảm bảo rằng bất kỳ nhiễm sắc thể nào cũng có thể được tạo ra từ bất kỳ nhiễm sắc thể nào khác (chỉ cần đủ số lần đột biến).

- Do đó, từ mọi trạng thái P_j (dù tệ đến đâu), luôn có một xác suất (dù rất nhỏ) để tạo ra h^* và đi vào tập hấp thụ.

Kết luận 2: Tập hấp thụ h^* có thể *đạt tới được* từ mọi trạng thái khác.
Kết luận cuối cùng (Từ Lý thuyết Chuỗi Markov): Một Chuỗi Markov hữu hạn mà có (các) trạng thái hấp thụ, và từ mọi trạng thái không hấp thụ đều có thể đi đến một trạng thái hấp thụ, thì **chắc chắn sẽ hội tụ về một trong các trạng thái hấp thụ đó với xác suất 1 khi** $t \to \infty$.

2.3.6 So sánh với Schema Theorem

Ta nên phân biệt rõ hai định lý này:

Định lý Lược đồ:

- **Mục đích:** Giải thích *tại sao GA hiệu quả* (*efficient*).
- **Ý nghĩa:** GA khai thác các "khối xây dựng" tốt và nhân chúng lên theo cấp số nhân. Nó nói về **động lực học (dynamics)** và **tốc độ** trong thời gian ngắn.

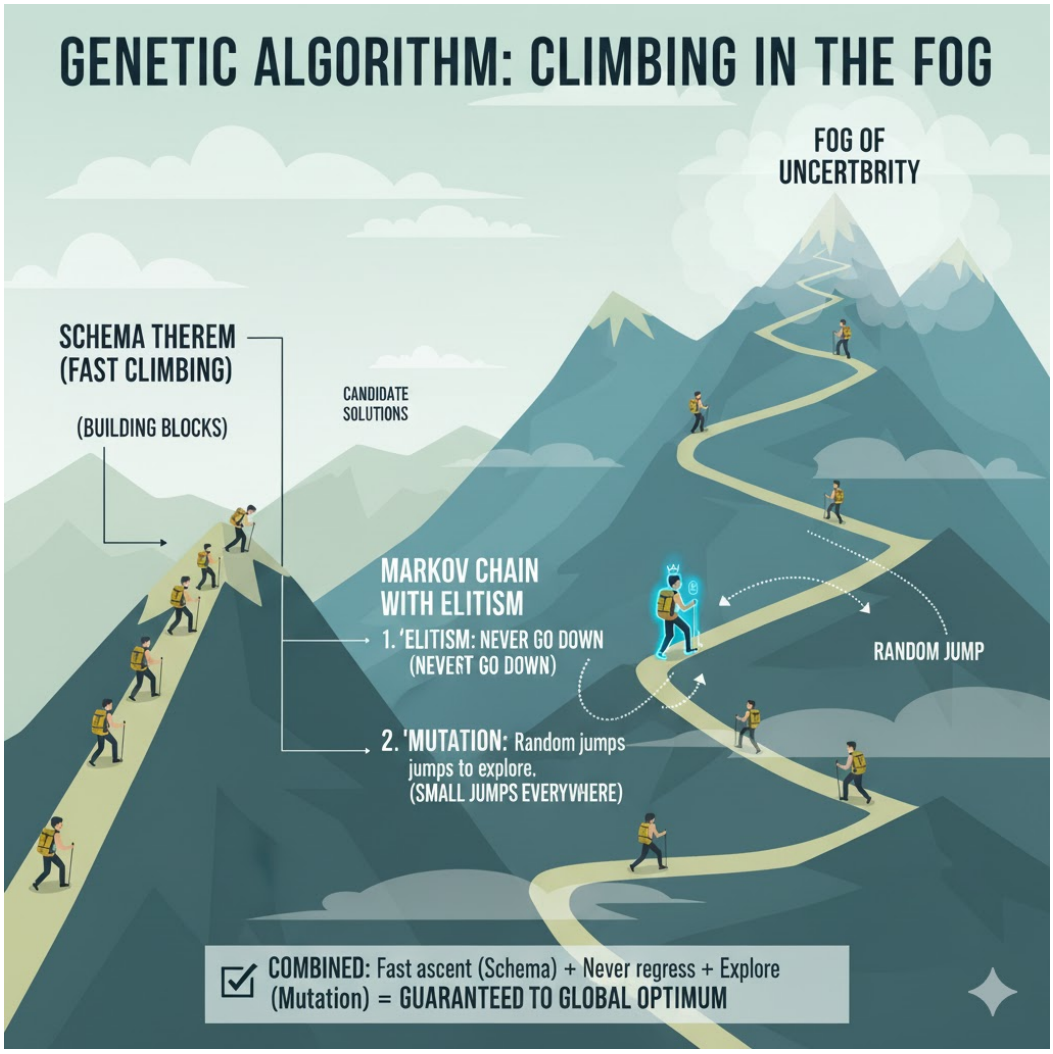
Định lý Hội tụ Chuỗi Markov:

- **Mục đích:** Chứng minh rằng *GA chính xác (correct / convergent)* (với Elitism).
- **Ý nghĩa:** GA được đảm bảo (về mặt lý thuyết) sẽ tìm ra giải pháp tối ưu toàn cục. Nó nói về **hành vi giới hạn (limit behavior)** khi $t \to \infty$.

Ví dụ: Leo núi trong sương mù, tìm đỉnh cao nhất (tối ưu toàn cục).

- **Định lý Lược đồ** giống như việc ta có một tấm bản đồ (fitness) và một chiến lược (crossover) cho phép bạn nhanh chóng kết hợp các đoạn đường dốc (*building blocks*) mà ta tìm thấy. Nó giúp ta leo lên *rất nhanh*.
- **Định lý Chuỗi Markov (với Elitism)** giống như một lời đảm bảo rằng:
  1. Ta có một cái neo (**Elitism**): sẽ không bao giờ bị trượt xuống thấp hơn điểm cao nhất ta đã từng đạt được.
  2. Có khả năng dịch chuyển tức thời (**Mutation**): Ta *luôn có một cơ hội* (dù rất nhỏ) để "nhảy" ngẫu nhiên đến bất kỳ điểm nào trên bản đồ, bao gồm cả đỉnh núi.

Kết hợp lại: Ta sẽ leo nhanh (Schema Theorem) và được đảm bảo sẽ không bao giờ tụt dốc (Elitism), đồng thời luôn có cơ hội "nhảy" trúng đỉnh (Mutation). Sự kết hợp này đảm bảo cuối cùng sẽ tìm thấy đỉnh.



Hình 4: Minh hoạ leo núi

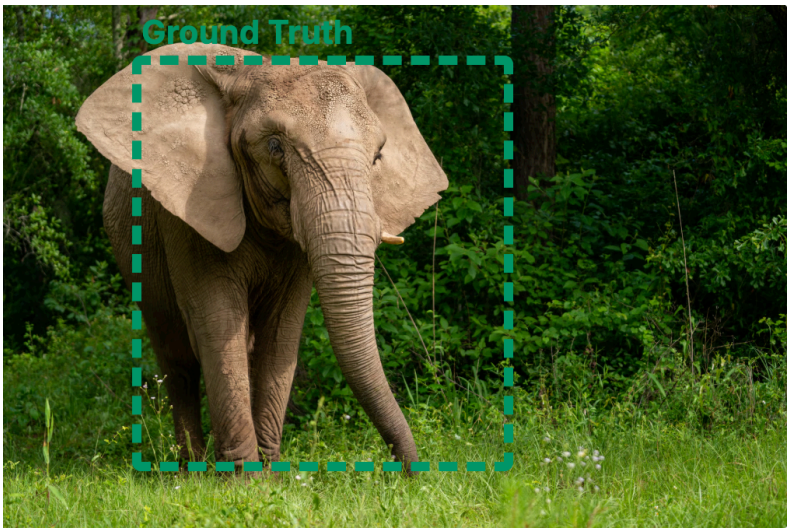
3 Quy trình cơ bản của GA: Bài toán tìm đối tượng

Mục tiêu của ta là tìm vị trí chính xác của **con voi**. Vị trí này là một *khung chứa* (bounding box) duy nhất.

Định nghĩa câu trả lời hoàn hảo là **"Ground Truth" (GT)**. Đây là khung màu xanh lá mà thuật toán sẽ cố gắng bắt chước.

Ground Truth (GT) được biểu diễn dưới dạng toạ độ trung tâm:

$$[x_{\text{tâm}}, y_{\text{tâm}}, \text{rộng}, \text{cao}] = [350, 270, 400, 480]$$



Hình 5: Minh họa vị trí Ground Truth (khung màu xanh lá) trên ảnh gốc.

Bước 1: Khởi tạo Quần thể (Thế hệ 1) Bắt đầu bằng cách tạo ra một *"quần thể"* (ví dụ: 100 cá thể). Mỗi cá thể là một **Chromosome** – một giải pháp ngẫu nhiên $[x, y, w, h]$.

5 Cá thể đầu tiên (Ví dụ)

```
Cá thể 1 (C1): [70, 70, 100, 100]
Cá thể 2 (C2): [100, 50, 200, 200]
Cá thể 3 (C3): [300, 300, 50, 50]
Cá thể 4 (C4): [180, 100, 300, 350]
Cá thể 5 (C5): [400, 400, 80, 80]
```

Hình 6: Thế hệ 1 – Khung đỏ ngẫu nhiên (quần thể khởi tạo ban đầu).

Bước 2: Đánh giá (Fitness Function = IoU) Chấm điểm (*độ thích nghi*) cho từng cá thể bằng cách so sánh nó với **Ground Truth**. Điểm số chính là **Intersection over Union (IoU)**:

$$\text{Fitness} = \text{IoU} = \frac{\text{Diện tích Chồng lắp}}{\text{Diện tích Hợp nhất}}$$

Tính toán chi tiết Fitness cho Cá thể 1 (C1)

GT (Target): [x_min: 150, y_min: 30, x_max: 550, y_max: 510] **C1 (Candidate)**: [x_min: 70, y_min: 70, x_max: 170, y_max: 170]

1. Tính Diện tích Chồng lắp (Overlap):

$$\begin{aligned}
x_{\min}^{(overlap)} &= \max(150, 70) = 150 \\
y_{\min}^{(overlap)} &= \max(30, 70) = 70 \\
x_{\max}^{(overlap)} &= \min(550, 170) = 170 \\
y_{\max}^{(overlap)} &= \min(510, 170) = 170 \\
\text{Overlap_Width} &= 170 - 150 = 20 \\
\text{Overlap_Height} &= 170 - 70 = 100 \\
\text{Area(Overlap)} &= 20 \times 100 = 2{,}000
\end{aligned}$$

2. Tính Diện tích Hợp nhất (Union):

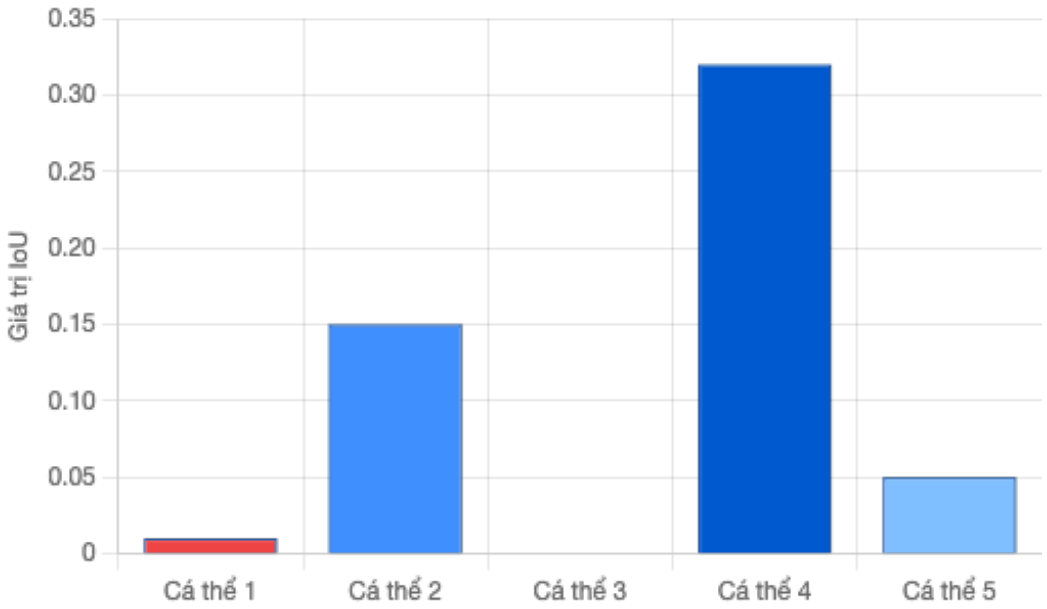
$$\begin{aligned}
\text{Area(GT)} &= 400 \times 480 = 192{,}000 \\
\text{Area(C1)} &= 100 \times 100 = 10{,}000 \\
\text{Area(Union)} &= \text{Area(GT)} + \text{Area(C1)} - \text{Area(Overlap)} \\
&= 192{,}000 + 10{,}000 - 2{,}000 = 200{,}000
\end{aligned}$$

3. Kết quả Fitness (IoU):

$$\text{IoU}(\text{C1}) = \frac{2{,}000}{200{,}000} = 0.01$$

Bảng điểm Fitness (Thế hệ 1)

Cá thể	Chromosome [x, y, w, h]	Fitness (IoU)
C1	[70, 70, 100, 100]	0.01
C2	[100, 50, 200, 200]	0.15
C3	[300, 300, 50, 50]	0.00
C4	[180, 100, 300, 350]	0.32
C5	[400, 400, 80, 80]	0.05



Hình 7: Biểu đồ minh họa giá trị Fitness (IoU) cho 5 cá thể đầu tiên trong Thế hệ 1.

Bước 3: Chọn lọc (Selection) Chọn "cha mẹ" cho thế hệ sau. Các cá thể có **Fitness (IoU)** cao hơn sẽ có cơ hội được chọn cao hơn. Phương pháp phổ biến được sử dụng là **Chọn lọc Bánh xe Roulette** (*Roulette Wheel Selection*).

Tính toán Xác suất Chọn

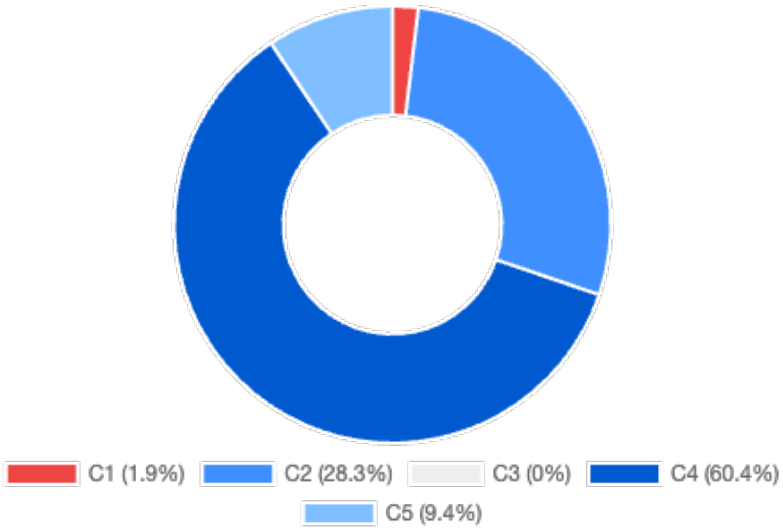
Tổng Fitness:

$$0.01 + 0.15 + 0.00 + 0.32 + 0.05 = 0.53$$

Xác suất chọn từng cá thể:

$$\text{Prob(C1)} = \frac{0.01}{0.53} = 1.9\%$$
$$\text{Prob(C2)} = \frac{0.15}{0.53} = 28.3\%$$
$$\text{Prob(C3)} = \frac{0.00}{0.53} = 0\%$$
$$\text{Prob(C4)} = \frac{0.32}{0.53} = 60.4\%$$
$$\text{Prob(C5)} = \frac{0.05}{0.53} = 9.4\%$$

Như ta thấy, **Cá thể 4** có khả năng được chọn làm "cha mẹ" cao nhất, trong khi Cá thể 1 và Cá thể 3 gần như không có cơ hội.



Hình 8: Biểu đồ xác suất chọn lọc (Roulette Wheel) cho 5 cá thể trong Thế hệ 1.

Bước 4 & 5: Lai ghép (Crossover) & Đột biến (Mutation) Từ các "cha mẹ" đã chọn (giả sử chọn **C4** và **C2**), ta tạo ra "con cái" (thế hệ mới).

Lai ghép (Crossover) – 2 điểm

Trộn "gen" $[x, y, w, h]$ của cha mẹ để tạo ra con.

$$\text{Cha Mẹ 1 (C4)} : [180,\ 100,\ 300,\ 350]$$
$$\text{Cha Mẹ 2 (C2)} : [100,\ 50,\ 200,\ 200]$$
$$\Downarrow$$
$$\text{Con 1 (Mới)} : [180,\ 50,\ 200,\ 350]$$

Đột biến (Mutation)

Thay đổi ngẫu nhiên một gen của "Con 1" để tạo ra sự đa dạng.

Trước đột biến:

$$[180,\ 50,\ 200,\ 350]$$

(Thêm một số ngẫu nhiên nhỏ, ví dụ: -15 vào gen thứ 2)

Sau đột biến (sẵn sàng cho Gen 2):

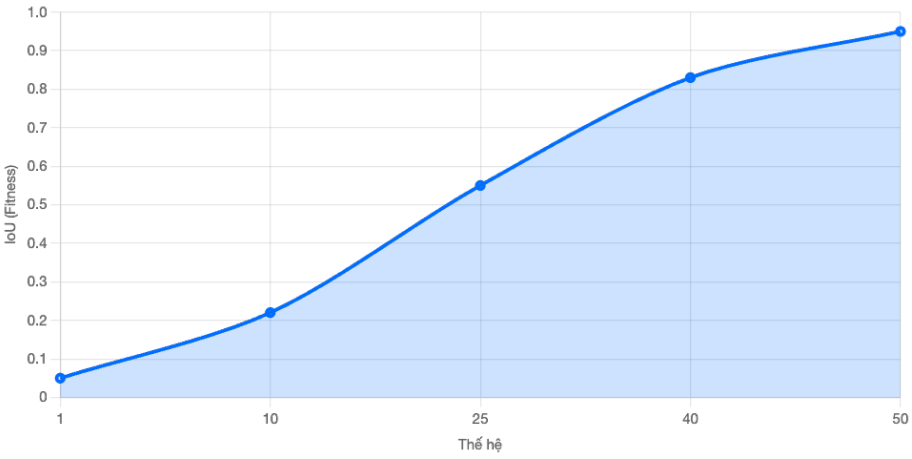
$$[180,\ 35,\ 200,\ 350]$$

Bước 6: Hội tụ qua các Thế hệ Lặp lại các bước 2, 3, 4 và 5 nhiều lần. Quần thể mới sẽ thay thế quần thể cũ. Trong đó, áp dụng **Elitism (Tinh hoa)** – luôn giữ lại cá thể tốt nhất (ví dụ: C4) và sao chép nó thẳng sang thế hệ sau để đảm bảo giải pháp không bị tệ đi.

Hãy xem sự "tiến hóa" trực quan của quần thể qua các thế hệ:

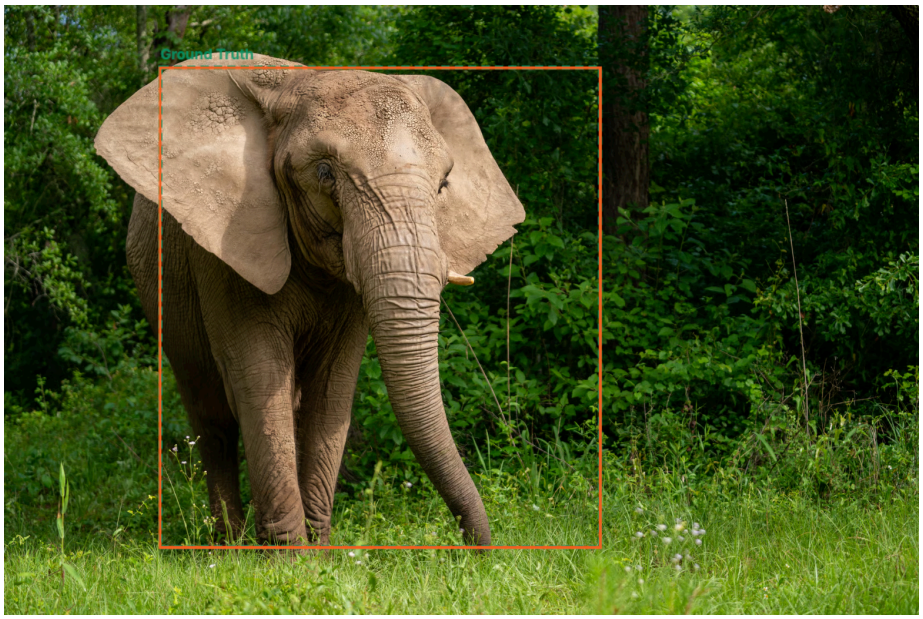
Hình 9: Các giai đoạn hội tụ

- **Thế hệ 1**: Max IoU ≈ 0.32
- **Thế hệ 5**: Max IoU ≈ 0.47
- **Thế hệ 10**: Max IoU ≈ 0.72
- **Thế hệ 20**: Max IoU ≈ 0.88
- **Thế hệ 50**: Max IoU ≈ 0.95



Hình 10: Biểu đồ hội tụ qua các thế hệ – giá trị IoU tối đa tăng dần theo thời gian.

Kết quả Cuối cùng (Sau 50 Thế hệ) Sau khi dừng thuật toán (ví dụ: sau 50 thế hệ), ta chọn cá thể có **Fitness** cao nhất từ quần thể cuối cùng. Đây chính là **Giải pháp GA tốt nhất**, tương ứng với **Ground Truth** được mô phỏng.



Hình 11: Ảnh minh họa đối tượng được phát hiện với khung chứa cuối cùng (Gen 50).

Cá thể Thắng cuộc (Gen 50)

$$\text{Chromosome} : [348,\ 270,\ 396,\ 476]$$

$$\textbf{Fitness (IoU)} = 0.95$$

4 Thực hành code

1. Định nghĩa Hàm Thích nghi (IoU)

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image

def calculate_iou(boxA, boxB):
    """
    Tính toán Intersection over Union (IoU) ữgia hai bounding box.
    box: [x_min, y_min, width, height]
    """
    xA, yA, wA, hA = boxA
    xB, yB, wB, hB = boxB
    xA_max, yA_max = xA + wA, yA + hA
    xB_max, yB_max = xB + wB, yB + hB

    x_intersect_min = max(xA, xB)
    y_intersect_min = max(yA, yB)
    x_intersect_max = min(xA_max, xB_max)
    y_intersect_max = min(yA_max, yB_max)
```

```
20
21      inter_width = max(0, x_intersect_max - x_intersect_min)
22      inter_height = max(0, y_intersect_max - y_intersect_min)
23      inter_area = inter_width * inter_height
24
25      union_area = wA * hA + wB * hB - inter_area
26      return 0.0 if union_area == 0 else inter_area / union_area
27
28  def fitness_function(chromosome, gt_box):
29      return calculate_iou(chromosome, gt_box)
```

Code Listing 1: Hàm tính toán IoU giữa hai bounding box

2. Các Toán tử của Giải thuật Di truyền

```
1   def initialize_population(pop_size, img_width, img_height, min_box_size, max_box_size):
2       population = []
3       for _ in range(pop_size):
4           x_min = np.random.randint(0, img_width - min_box_size)
5           y_min = np.random.randint(0, img_height - min_box_size)
6           width = np.random.randint(min_box_size, max_box_size)
7           height = np.random.randint(min_box_size, max_box_size)
8           width = min(width, img_width - x_min)
9           height = min(height, img_height - y_min)
10          width = max(width, 1)
11          height = max(height, 1)
12          population.append(np.array([x_min, y_min, width, height]))
13      return np.array(population)
14
15  def selection(population, fitness_scores, num_parents):
16      parents = []
17      for _ in range(num_parents):
18          i1, i2 = np.random.choice(len(population), 2, replace=False)
19          parents.append(population[i1] if fitness_scores[i1] > fitness_scores[i2] else
        population[i2])
20      return np.array(parents)
21
22  def crossover(parents, crossover_rate, img_width, img_height):
23      children = []
24      for i in range(0, len(parents) - 1, 2):
25          p1, p2 = parents[i], parents[i + 1]
26          if np.random.rand() < crossover_rate:
27              swap_idx = np.random.choice(4, 2, replace=False)
28              c1, c2 = np.copy(p1), np.copy(p2)
29              c1[swap_idx] = p2[swap_idx]
30              c2[swap_idx] = p1[swap_idx]
31              children.extend([c1, c2])
32          else:
33              children.extend([p1, p2])
34      return np.clip(np.array(children), 0, None)
35
36  def mutation(children, mutation_rate, img_width, img_height, mutation_strength=20):
37      mutated = np.copy(children)
38      for i in range(len(mutated)):
39          if np.random.rand() < mutation_rate:
40              j = np.random.randint(4)
41              delta = np.random.randint(-mutation_strength, mutation_strength + 1)
42              mutated[i][j] += delta
43      for i in range(len(mutated)):
```

```
        mutated[i][0] = np.clip(mutated[i][0], 0, img_width - 1)
        mutated[i][1] = np.clip(mutated[i][1], 0, img_height - 1)
        mutated[i][2] = np.clip(mutated[i][2], 1, img_width - mutated[i][0])
        mutated[i][3] = np.clip(mutated[i][3], 1, img_height - mutated[i][1])
    return mutated
```

Code Listing 2: Khởi tạo, chọn lọc, lai ghép và đột biến

3. Hàm chính chạy GA

```
def run_ga_bounding_box_optimization(gt_box, img_width, img_height,
                                     pop_size=50, generations=100,
                                     crossover_rate=0.7, mutation_rate=0.1,
                                     elitism_count=2, min_box_size=10, max_box_size=100):
    population = initialize_population(pop_size, img_width, img_height, min_box_size,
    max_box_size)
    best_fitness_history, avg_fitness_history = [], []

    print(f"Ground Truth Box: {gt_box}")

    for gen in range(generations):
        fitness_scores = np.array([fitness_function(chromo, gt_box) for chromo in
    population])
        best_fitness = np.max(fitness_scores)
        avg_fitness = np.mean(fitness_scores)
        best_fitness_history.append(best_fitness)
        avg_fitness_history.append(avg_fitness)

        best_chromo = population[np.argmax(fitness_scores)]
        if gen % 10 == 0 or gen == generations - 1:
            print(f"Gen {gen:4d} | Best IoU: {best_fitness:.4f} | Avg IoU: {avg_fitness:.4f
    } | Best Box: {best_chromo}")

        if best_fitness >= 0.999:
            print(f"Converged at generation {gen} with IoU: {best_fitness:.4f}")
            break

        elites = population[np.argsort(fitness_scores)[-elitism_count:]]
        parents = selection(population, fitness_scores, pop_size - elitism_count)
        children = crossover(parents, crossover_rate, img_width, img_height)
        mutated_children = mutation(children, mutation_rate, img_width, img_height)
        population = np.concatenate((elites, mutated_children[:pop_size - elitism_count]))

    final_scores = np.array([fitness_function(c, gt_box) for c in population])
    final_best = population[np.argmax(final_scores)]
    print(f"\nFinal Best Box: {final_best} with IoU: {np.max(final_scores):.4f}")

    return final_best, best_fitness_history, avg_fitness_history, gen
```

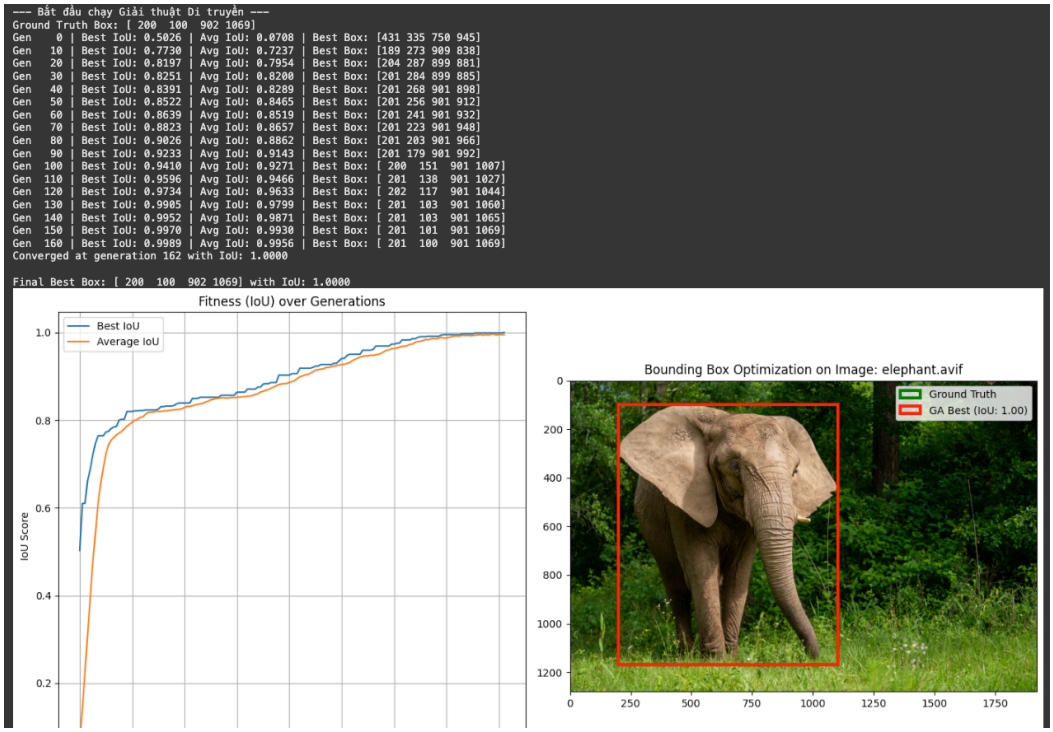
Code Listing 3: Thuật toán chính cho tối ưu Bounding Box

4. Thực nghiệm và Trực quan hóa

```
if __name__ == "__main__":
    image_path = "elephant.avif"
    try:
        image = Image.open(image_path)
    except FileNotFoundError:
```

```
        print(f"ỗLi: Không tìm ấthy ảnh ạti '{image_path}'.")
        exit()

    img_width, img_height = image.size
    img_array = np.array(image)
    ground_truth_box = np.array([200, 100, 902, 1069])
    ground_truth_box[0] = np.clip(ground_truth_box[0], 0, img_width - ground_truth_box[2])
    ground_truth_box[1] = np.clip(ground_truth_box[1], 0, img_height - ground_truth_box[3])
    ground_truth_box[2] = np.clip(ground_truth_box[2], 1, img_width - ground_truth_box[0])
    ground_truth_box[3] = np.clip(ground_truth_box[3], 1, img_height - ground_truth_box[1])

    print("\n--- ắBt đầu ạchy ảGii ậthut Di ềtruyn ---")
    final_best_box, best_fitness_history, avg_fitness_history, generations_run =
    run_ga_bounding_box_optimization(
        gt_box=ground_truth_box,
        img_width=img_width, img_height=img_height,
        pop_size=100, generations=300,
        crossover_rate=0.8, mutation_rate=0.2, elitism_count=5,
        min_box_size=30, max_box_size=int(min(img_width, img_height) * 0.9)
    )

    fig, axes = plt.subplots(1, 2, figsize=(14, 7))
    axes[0].plot(range(generations_run + 1), best_fitness_history, label='Best IoU')
    axes[0].plot(range(generations_run + 1), avg_fitness_history, label='Average IoU')
    axes[0].set_title('Fitness (IoU) over Generations')
    axes[0].set_xlabel('Generation')
    axes[0].set_ylabel('IoU Score')
    axes[0].legend()
    axes[0].grid(True)

    axes[1].imshow(img_array, origin='upper', extent=[0, img_width, img_height, 0])
    rect_gt = patches.Rectangle((ground_truth_box[0], ground_truth_box[1]),
                                ground_truth_box[2], ground_truth_box[3],
                                linewidth=3, edgecolor='g', facecolor='none', label='Ground
     Truth')
    rect_ga = patches.Rectangle((final_best_box[0], final_best_box[1]),
                                final_best_box[2], final_best_box[3],
                                linewidth=3, edgecolor='r', facecolor='none',
                                label=f'GA Best (IoU: {calculate_iou(final_best_box,
    ground_truth_box):.2f})')
    axes[1].add_patch(rect_gt)
    axes[1].add_patch(rect_ga)
    axes[1].set_title(f'Bounding Box Optimization on Image: {image_path}')
    axes[1].set_xlim(0, img_width)
    axes[1].set_ylim(img_height, 0)
    axes[1].set_aspect('equal', adjustable='box')
    axes[1].legend()
    plt.tight_layout()
    plt.show()
```

Code Listing 4: Chạy GA trên ảnh và hiển thị kết quả



Hình 12: Kết quả code