


## Generalized Policy Iteration (GPI) là gì?

- **Khái niệm cốt lõi:**
  - GPI là khung chung **hợp nhất nhiều thuật toán** trong RL, dựa trên nguyên lý “lặp lại hai bước: ước lượng giá trị → cải tiến chính sách” cho đến khi hội tụ.
  - Trong mọi thuật toán thuộc GPI, luôn tồn tại hai quá trình song song:
    1. **Policy Evaluation:** Đánh giá (ước lượng) hàm giá trị của chính sách hiện tại.
    2. **Policy Improvement:** Dựa trên giá trị vừa ước lượng, **cải tiến** (thường là chọn hành động greedy) để sinh ra chính sách mới tốt hơn.
  - Quá trình này có thể thực hiện cho đến khi không còn cải thiện được nữa, tức khi chính sách đạt tối ưu. 
- **Điểm khác biệt so với Policy Iteration “thuần túy”:**
  - Trong Policy Iteration chuẩn, ta thường khóa **hoàn toàn** ước lượng giá trị cho đến khi hội tụ “gần đúng” rồi mới làm cải tiến.
  - Trong GPI, **không nhất thiết** chờ policy evaluation hoàn chỉnh. Ví dụ: Value Iteration chỉ chạy **một lần quét** (sweep) để ước lượng giá trị và ngay lập tức “greedify” → lặp lại liên tục.

## 3. Các bước cơ bản trong GPI

GPI không gán cứng số lượt cho mỗi bước; thay vào đó, hai quá trình đánh giá và cải tiến diễn ra một cách **linh hoạt**:

### 1. Policy Evaluation (Ước lượng giá trị)

- Mục đích: Với policy  $\pi$  cho trước (có thể là policy được khởi tạo ngẫu nhiên hoặc đã cải tiến từ bước trước), tính hoặc xấp xỉ hàm giá trị  $V^\pi$  (hoặc  $Q^\pi$ ).
- Cách làm:
  - **Đầy đủ (Full):** Giải hệ Bellman Expectation cho  $V^\pi$  (như trong iterative policy evaluation).
  - **Một phần (Partial):** Chỉ cập nhật một số state nhất định (hoặc một quét “non-sweep”) rồi dừng, để chuyển sang bước cải tiến ngay.
  - **Ví dụ:** Trong Value Iteration, mỗi quét qua toàn bộ state để cập nhật

$$V(s) \leftarrow \max_a \sum_{s', r} P(s', r | s, a) [r + \gamma V(s')]$$

chính là kết hợp đồng thời việc tính tiếng nói của policy cải tiến và ước lượng giá trị.




### 2. Policy Improvement (Cải tiến chính sách)

- Mục đích: Dựa trên  $V^\pi$  (hoặc  $Q^\pi$ ) vừa ước lượng, chọn hành động tốt nhất (greedy) ở mỗi state để tạo ra policy mới  $\pi'$ .
- Công thức mẫu (dùng giá trị trạng thái):


$$\pi'(s) = \arg \max_a \sum_{s',r} P(s', r | s, a) [r + \gamma V^\pi(s')].$$

- Trường hợp dùng trực tiếp  $Q^\pi$ :

$$\pi'(s) = \arg \max_a Q^\pi(s, a).$$

- Trong GPI, sau một lượt “partial” evaluation, ta có thể ngay lập tức cải tiến để tránh phải ước lượng quá chính xác ở đầu. 

### 3. Lặp lại cho đến khi hội tụ


- Các bước trên liên tục “chạy song song”: không phải đợi evaluation hoàn chỉnh mới cải tiến, mà có thể xen kẽ nhiều lần hơn.
- Khi policy không thay đổi qua bước improvement, GPI đã tìm được **optimal policy**  $\pi^*$  và **optimal value function**  $V^*$  hoặc  $Q^*$ . 

## 4. Value Iteration như một trường hợp GPI

- **Khái niệm:** Value Iteration thực chất là GPI ở mức “nhanh gọn”: mỗi lần chỉ làm **một bước cập nhật giá trị rồi ngay lập tức – tức thì** giả định policy mới (greedy) dựa trên giá trị vừa cập nhật, mà không thực hiện Policy Evaluation đến khi hội tụ.


- **Công thức chính:**

$$V_{k+1}(s) = \max_a \sum_{s',r} P(s', r | s, a) [r + \gamma V_k(s')].$$

- Ở mỗi bước  $k$ , với giá trị cuống  $V_k$ , ta chọn hành động  $a$  sao cho kỳ vọng “reward tức thì + chiết khấu giá trị trạng thái kế tiếp” là lớn nhất.
- Việc này vừa tương đương với bước **Giá trị (Evaluation)**—thử nghiệm một giả định policy greedy—vừa tương đương với bước **Cải tiến**—cf. greedy action.
- **Đặc điểm:**
  1. Không giữ policy tường minh; suy ngầm rằng policy luôn là “greedy đối với  $V_k$ ”.
  2. Lặp liên tục cho đến khi  $\max_s |V_{k+1}(s) - V_k(s)| < \theta$ .
  3. Cuối cùng thu được  $V^*$ , suy ra  $\pi^*(s) = \arg \max_a \sum_{s',r} P(s', r | s, a) [r + \gamma V^*(s')]$ .
- **Ưu điểm:** Thường hội tụ nhanh hơn Policy Iteration thuần túy, vì chỉ một bước cập nhật giá trị là có thể cải tiến policy luôn. 

## 5. Dynamic Programming: Đồng bộ (Synchronous) vs. Không đồng bộ (Asynchronous)

### 1. Synchronous Dynamic Programming (DP đồng bộ)

- **Định nghĩa:** Trong mỗi vòng lặp (iteration), **cập nhật đồng thời** hết tất cả các giá trị (state-value hoặc action-value) dựa trên giá trị cũ ngay tại bước đó.
- **Quy trình cơ bản:**
  1. Lấy ra bản copy của  $V_k$ .
  2. Với mọi  $s$ , tính  $V_{k+1}(s)$  dựa trên công thức Bellman (expectation hay optimality).
  3. Sau khi duyệt hết các state, gán  $V_k \leftarrow V_{k+1}$ .
- **Ví dụ:**
  - Value Iteration triển khai như trên: mỗi sweep qua toàn bộ state để cập nhật đồng loạt.
  - Policy Iteration khi thực hiện Policy Evaluation: dùng iterative sweeps, cập nhật mỗi state trên bản copy cũ, sau khi hết sweep thì cập nhật chung.
- **Ưu điểm:**
  - Dễ phân tích và chứng minh hội tụ.
  - Thích hợp khi **số lượng state không quá lớn**, dễ tổ chức tính toán song song (parallel).
- **Nhược điểm:**
  - Cần **quét toàn bộ state** trong mỗi iteration, khá tốn kém nếu không gian trạng thái lớn. 

### 2. Asynchronous Dynamic Programming (DP không đồng bộ)

- **Định nghĩa:** Cập nhật **từng giá trị một**, có thể cập nhật theo thứ tự tùy chọn (ngẫu nhiên, ưu tiên...). Không đòi hỏi phải quét hết toàn bộ state mỗi iteration.
- **Các chiến lược cập nhật:**
  - **Random Scan:** Chọn ngẫu nhiên một state để cập nhật tại mỗi bước.
  - **Prioritized Sweeping:** Định mức "độ quan trọng" của mỗi state (ví dụ: độ chênh giữa hai lần cập nhật liên tiếp), ưu tiên cập nhật trước những state có độ thay đổi lớn.
  - **TD( $\lambda$ ):** Xử lý tín hiệu lỗi (temporal-difference) lan đến nhiều state khác nhau, sau đó học giá trị dựa trên trải nghiệm.
  - **Online Q-learning:** Mỗi khi agent quan sát sample  $(s, a, r, s')$ , ngay lập tức cập nhật  $Q(s, a)$  dựa trên

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)].$$

- **Ưu điểm:**
  - **Linh hoạt:** Có thể tập trung cập nhật các state quan trọng đầu tiên, tránh cập nhật state ít liên quan.

- **Scalable:** Phù hợp cho **không gian trạng thái rất lớn** hoặc liên tục (continuous), nơi không thể lưu trữ và quét toàn bộ.
- **Online Learning:** Có thể vừa thu thập sample vừa cập nhật, không cần đợi offline.
- **Nhược điểm:**
  - Cần **chính sách chọn thứ tự** thích hợp; nếu chọn ngẫu nhiên hoàn toàn, có thể hội tụ chậm.
  - Phải cẩn trọng với tiêu chí hội tụ: vì sử dụng dữ liệu “mới” ngay lập tức, đôi khi khó kiểm soát ổn định. 📄

## 6. Ví dụ minh họa Asynchronous DP 📄

- **Prioritized Sweeping:**
  1. Khi một state  $s'$  vừa được cập nhật giá trị, xác định những state  $s$  “tiền lệ” (predecessor)—tức state có thể dẫn đến  $s'$  qua một hành động.
  2. Tính **mức độ ưu tiên** cho mỗi  $s$  dựa trên “độ chênh”  

$$\Delta = |r + \gamma V(s') - V(s)|.$$
  3. Đưa  $s$  vào hàng đợi ưu tiên (priority queue) theo  $\Delta$  lớn.
  4. Mỗi bước, lấy ra state có độ ưu tiên cao nhất để cập nhật trước.
  5. Mỗi lần cập nhật lại, lặp lại cho các predecessor.
- **Kết quả:** Cập nhật tập trung vào những state có thay đổi lớn, đẩy nhanh hội tụ.
- **TD( $\lambda$ ) – Asynchronous mix giữa Monte Carlo và TD(0):**
  1. Khi agent trải nghiệm  $(s, a, r, s')$ , ngay lập tức tính **temporal-difference error**  $\delta = r + \gamma V(s') - V(s)$ .
  2. Lan tín hiệu  $\delta$  về các state trước thông qua **eligibility traces** (những state đã “đóng góp” vào việc chọn action lúc đó).
  3. Cập nhật ngay  $V(s)$  cho từng state kéo dài.
  4. Luôn kết hợp “nhanh” (mỗi sample cập nhật ngay) với “chậm” (làm lan qua nhiều state trước).
- **Online Q-learning:** Mỗi sample  $(s, a, r, s')$  dẫn đến:
 
$$Q(s, a) \leftarrow Q(s, a) + \alpha [r + \gamma \max_{a'} Q(s', a') - Q(s, a)].$$
  - Không cần biết toàn bộ MDP trước, có thể học dần dần khi tương tác với môi trường.
  - Cập nhật **một cặp**  $(s, a)$  mỗi lần, phù hợp với Asynchronous DP. 📄

## 7. Khi nào nên dùng Synchronous vs. Asynchronous

### 1. Synchronous DP:

- Dùng khi:
  - Mô hình hoàn toàn biết trước (các hàm  $P(s', r | s, a)$  có sẵn).
  - Không gian trạng thái vừa phải (dưới hàng chục nghìn state), có thể quét toàn bộ.
  - Cần kết quả chính xác (tính toán offline).
- Ví dụ: Value Iteration hay Policy Iteration cho môi trường Grid World nhỏ.

### 2. Asynchronous DP:

- Dùng khi:
  - Không gian trạng thái lớn hoặc liên tục (ví dụ: robot định vị trong không gian thực).
  - Không biết trước mô hình; chỉ có thể truy vấn sample bằng cách tương tác (model-free).
  - Cần học online hoặc ưu tiên cập nhật trạng thái "quan trọng" trước (ví dụ: gần khu vực reward cao).
- Ví dụ: Q-learning trong trò chơi Atari,  $TD(\lambda)$  cho bài toán dự báo (prediction) trên chuỗi thời gian.

---

## 8. Tóm tắt (Summary)

1. **Generalized Policy Iteration (GPI):** Là khung chung hợp nhất nhiều thuật toán RL, dựa trên hai bước lặp song song:
  - **Policy Evaluation:** Ước lượng giá trị của policy hiện tại (có thể toàn phần hoặc một phần).
  - **Policy Improvement:** Cải tiến policy (thường là greedy) dựa trên giá trị vừa ước lượng.
  - GPI cho phép linh hoạt: không nhất thiết phải hoàn thiện policy evaluation trước khi cải tiến (ví dụ Value Iteration).
2. **Value Iteration** là một trường hợp đặc biệt của GPI:
  - Kết hợp tức thì (one-step) giữa cập nhật giá trị và cải tiến (implicitly greedy) mà không đòi hỏi policy evaluation hoàn chỉnh.
3. **Dynamic Programming – Synchronous vs. Asynchronous:**
  - **Synchronous DP:** Mỗi vòng lặp quét qua tất cả state để cập nhật đồng loạt. Thích hợp khi không gian state nhỏ, mô hình biết trước.
  - **Asynchronous DP:** Cập nhật từng giá trị (state hoặc state-action) một cách không đồng bộ theo thứ tự bất kỳ (ngẫu nhiên, ưu tiên...). Phù hợp khi không gian lớn hoặc khi cần học online không cần biết mô hình.
4. **Ứng dụng thực tế:**
  - Khi xây dựng các thuật toán RL cho môi trường có mô hình biết trước và số trạng thái vừa phải, dùng Synchronous DP (Value Iteration, Policy Iteration).

- Khi không biết mô hình hoặc không gian trạng thái rất lớn (ví dụ: robot, trò chơi lớn), dùng Asynchronous DP ( $TD(0)$ ,  $TD(\lambda)$ , Q-learning, Prioritized Sweeping).

#### 5. Lời khuyên áp dụng:

- Luôn đặt GPI làm khung tư duy: xác định rõ vòng ước lượng giá trị và vòng cải tiến chính sách, rồi tùy bài toán mà chọn mức độ “partial/full” cho policy evaluation.
- Với bài toán càng phức tạp (không gian lớn, mô hình không rõ), ưu tiên Asynchronous DP và kết hợp hàm xấp xỉ (approximation) để đảm bảo hiệu năng và khả năng mở rộng.