INTRODUCTION to ARTIFICIAL INTELLIGENCE

Written by: Trần Ngọc Bảo

Subject information:
Trang web để học course lấy điểm cộng: MIT

Mục lục

1 Giới thiệu Trí tuệ Nhân tạo và Máy học

1.1 Quá trình tư duy

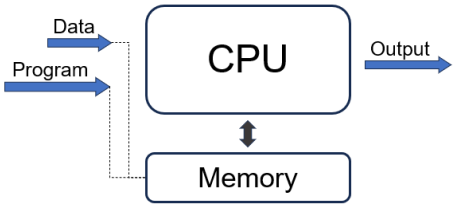
Máy	Người
tính (định lượng, xử lý con số cụ thể)	toán (định tính, xử lý dữ liệu liên tục, có tính tương đối)
ghi	nhớ
hành (thực hành)	học
kiếm (từ những gì đã có)	tìm (chưa chắc đã có, có thể là những thứ mới)
suy (từ lý thuyết, cơ sở, bổ đề)	luận (rút ra được kiến thức từ thông tin)
tư duy logic (não trái)	*tư duy trực quan (não phải)*

Áp dụng 2 cách tư duy của con người cho máy, ta xây dựng được 2 mô hình:
Von Neumann: (tư duy logic, ký hiệu, tuần tự) xử lý chính xác các bước được biết trước. Kết quả như được tái sản xuất. Hướng ngôn ngữ. Độ chính xác cao. ⇒ *Xử lý số. Xử lý CSDL.*
Neuron-computer: (tư duy trực quan, trực giác, nhận dạng, song song) thông tin phân tán, mơ hồ và xử lý dữ liệu song song. Hướng mẫu. Dư thừa. ⇒ *Dự báo số. Nhận dạng.*

1.2 Tiến hóa thuật toán

Một chương trình - program (P) có bản chất là những thuật toán - algorithm (A) được mã hóa thông qua coding để xử lý bài toán của chương trình (P). Một thuật toán bao gồm 6 đặc trưng (3 đặc trưng đầu là bắt buộc cho A):

1. Tính xác định (đơn định)
2. Tính dừng (hữu hạn)
3. Tính đúng (kết quả)
4. Tính hiệu quả
5. Tính phổ dụng
6. I/O



Với mô hình Von Neumann, dữ liệu và chương trình được xử lý tại CPU nhưng lại lưu trữ ở Memory, vậy nên quá trình xử lý gặp *nút thắt cổ chai* giữa CPU và Memory, điều khá tốn thời gian. Người ta mong muốn xây dựng một mô hình mà chương trình và dữ liệu được xử lý song song. Nói cách khác là kết hợp CPU và Memory để xử lý nhiều luồng cùng lúc.

- Mở rộng thuật toán (A) ta sẽ có được kĩ thuật mới là heuristic (H). Kỹ thuật này giúp máy tính có thể giải được những bài toán không có thuật toán.
- Khi kết hợp cả 2 kĩ thuật: thuật toán (A) và heuristic (H) trong một program (P) thì program (P) được gọi là có khả năng học máy (ML) (cấp độ 3 của "thực hành"). Vậy ta có thể hiểu: (A) → (H) → (ML).

Lập trình truyền thống: Là đưa data và program vào máy tính (computer) để xử lý và cho ra output.

Machine Learning: Là đưa data và output vào máy tính (computer) để xử lý và học, từ đó tạo ra chương trình để xử lý những yêu cầu (data & output) tương tự.

- Các mức độ của **data mining**: hiện tại đang ở *

(raw data) → (enhanced data) → (information) → (knowledge)* → (meta-knowledge)

> Mục tiêu của Trí tuệ nhân tạo là không chỉ hiểu mà còn xây dựng nên các thực thể thông minh.

1.3 Trí tuệ nhân tạo

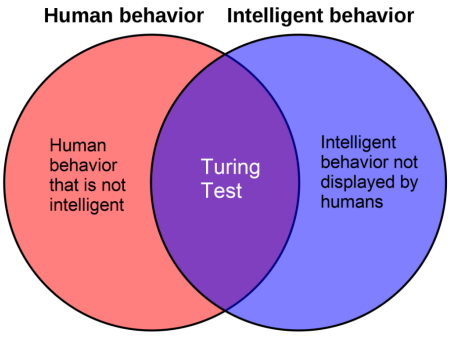
Các định nghĩa về TTNT thường được chia thành 4 nhóm:

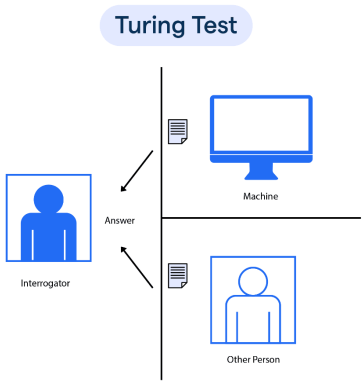
Suy nghĩ như người	Suy nghĩ hợp lý
Hành động như người	Hành động hợp lý

Trong đó:

- Nửa trên: Quá trình suy nghĩ, lập luận.
- Nửa dưới: Hành vi.
- Nửa trái: Con người.
- Nửa phải: Sự hợp lý.

1.3.1 Hành động như người





Turing test (Alan Turing, 1950): Dựa vào tiêu chí không thể phân biệt được giữa người và máy. Một người (bên A) đặt câu hỏi cho hệ thống (bên B gồm 1 người và một máy). Nếu sau một số câu hỏi, bên A không phân biệt 2 câu trả lời của bên B, đâu là của người, đâu là của máy thì hệ thống (máy của B) "đậu".

1.3.2 Suy nghĩ như người

- Nếu hệ thống có đầu vào/đầu ra và hành vi thời gian khớp với hành vi của con người, hệ thống được gọi là "suy nghĩ như người".
- **General Problem Solver** (Newell and Simon, 1961): Mục tiêu không phải là giải quyết vấn đề đúng mà so sánh các bước lập luận của hệ thống với hành vi thực tế của con người.
- Ngành Khoa học Nhận thức (Cognitive Science) và Trí tuệ Nhân tạo bổ sung lẫn nhau cùng phát triển.

1.3.3 Suy nghĩ hợp lý

- Luật suy nghĩ đúng đắn là cấu trúc lập luận luôn đưa ra kết luận đúng khi cho trước tiền đề đúng.
- Aristotle là một trong những người đầu tiên đưa ra khái niệm suy nghĩ đúng đắn. Eg. tam đoạn luận "Socrates là người. Mọi người phải chết. Vậy Socrates cũng chết."
- Những luật này điều khiển hoạt động trí tuệ và là khởi nguồn cho lĩnh vực logic.
- Vào thế kỉ 19, kí hiệu logic ra đời. Năm 1965, xuất hiện chương trình giải bất kì bài toán nào có thể giải được bằng kí hiệu logic.
- Khó khăn: Biểu diễn tri thức tổng quát theo dạng chuẩn logic không dễ dàng. Khả năng giải được của bài toán theo lý thuyết và thực tế khác nhau rất xa.

1.3.4 Hành động hợp lý

- Hành động sao cho đạt được đầu ra tốt nhất, hoặc tốt nhất theo mong đợi khi có sự không chắc chắn.
- Muốn hành động hợp lý cần lập luận đúng đắn, nhưng lập luận đúng đắn chưa chắc đã cho hành động hợp lý.

Eg: Phản xạ rút tay ngay khi chạm nước nóng cho kết quả tốt hơn việc suy nghĩ kĩ xem nhiệt độ nước là bao nhiêu, có đạt mức nóng không?.

- Ưu điểm: Tổng quát hơn hướng "Luật suy nghĩ" ví có nhiều cơ chế khác nhau để đạt được tính hợp lý. Phù hợp với sự phát triển khoa học vì chuẩn hợp lý dễ dàng được định nghĩa và hoàn toàn tổng quát hơn so với chuẩn hành vi và suy nghĩ của con người.

1.4 Nền tảng xây dựng

Triết học:

- Ý thức được phát sinh từ bộ não như thế nào?
- Tri thức đến từ đâu? Làm thế nào từ tri thức dẫn đến hành động?
- Có thể dùng luật hình thức để rút ra kết luận hợp lệ được không?
- Lam đoạn luận (Aristotle) và luật qui nạp (David Hume) trong logic cổ điển → áp dụng vào các hệ thống suy diễn tự động.

Toán học:

- Logic mệnh đề (George Boole), Logic bậc nhất (Gottlob Frege), và Lý thuyết tham chiếu (Tarski).
- Định lý không đầy đủ (Kurt Godel), Lý thuyết NP-đầy đủ (Steven Cook và Richard Karp).
- Lý thuyết xác suất.

Tâm lý học:

- Não xử lý thông tin như thế nào?

Khoa học thần kinh:

- Con người suy nghĩ và hành động như thế nào?
- Hiện tượng cảm nhận và điều khiển vận động.
- Các kĩ thuật thực nghiệm.

Ngôn ngữ học:

- Ngôn ngữ liên hệ với suy nghĩ như thế nào?
- Biểu diễn tri thức, văn phạm?

Kinh tế học:

- Chúng ta quyết định như thế nào để tối đa hóa lợi nhuận?
- Lý thuyết quyết định và Lý thuyết trò chơi (John von Neumann và Oskar Morgenstern).

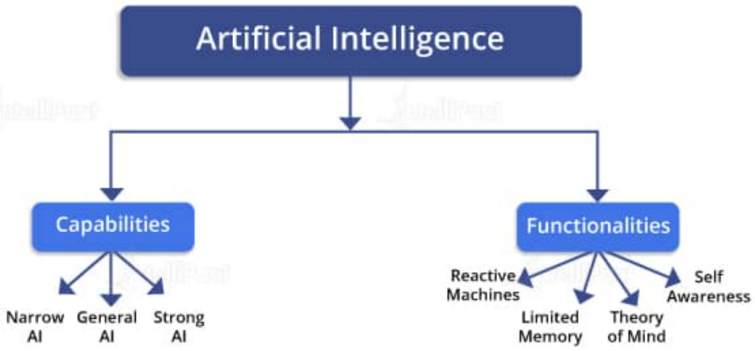
Công nghệ máy tính:

- Làm sao để có thể xây dựng máy tính hiệu quả?
- Xây dựng hệ thống ngày càng mạnh mẽ.

Điều khiển học:

- Các công cụ điều khiển hoạt động như thế nào?
- Thiết kế các hệ thống tối đa hóa hàm mục tiêu theo thời gian?

1.5 Giai loại TTNT



1.5.1 Khả năng

a) Artificial Narrow Intelligence - ANI (AI yếu hoặc AI hẹp)

- ANI là giai đoạn của TTNT liên quan đến máy móc chỉ có thể thực hiện 1 tập hợp các nhiệm vụ cụ thể được xác định trong phạm vi hẹp. Ở giai đoạn này, máy móc không có bất kì khả năng tư duy nào, nó chỉ thực hiện một tập hợp các chức năng được xác định trước.

- Eg: Siri, Alexa, Xe tự lái, Alpha-Go, Sophia,... Hầu hết tất cả các hệ thống dựa trên AI được xây dựng cho đến ngày nay đều thuộc AI yếu.

b) Artificial General Intelligence - AGI (AI tổng quát hay AI mạnh)

- AGI là giai đoạn của TTNT trong đó máy móc sẽ sở hữu khả năng suy nghĩ và đưa ra quyết định giống như con người.
- Hiện tại vẫn chưa có ví dụ nào về AI mạnh. Tuy nhiên, người ta tin rằng chúng ta sẽ sớm tạo ra được những cỗ máy thông minh như con người.
- AGI được coi là mối đe dọa đối với sự tồn tại của con người bởi nhiều nhà khoa học bao gồm cả Stephen Hawking tuyên bố: "Sự phát triển của TTNT đầy đủ có thể đánh dấu sự kết thúc của loài người... Nó sẽ tự cất cánh và tự thiết kế lại với tốc độ ngày càng tăng. Con người, những người bị giới hạn bởi quá trình tiến hóa sinh học chậm chạp, không thể cạnh tranh và sẽ bị thay thế.

c) Artificial Super Intelligence - ASI (Siêu AI)

- ASI là giai đoạn của TTNT khi khả năng của máy tính sẽ vượt qua con người. ASI là một tình huống giả định được mô tả trong phim và khoa học viễn tưởng, nơi máy móc đã chiếm linh thế giới.
- Máy móc không còn bao xa để đạt đến giai đoạn này nếu xét đến tốc độ hiện tại của chúng ta.

1.5.2 Chức năng

Khi ai đó yêu cầu giải thích các loại hệ thống TTNT khác nhau, chúng ta phải phân loại chúng dựa theo chức năng:

a) Reactive Machines AI (Máy Phản ứng AI)

- Loại AI này bao gồm các máy hoạt động chỉ dựa trên dữ liệu hiện tại, chỉ tính đến tình hình hiện tại. Máy AI phản ứng không thể hình thành các suy luận từ dữ liệu để đánh giá các hành động trong tương lai của chúng . Họ có thể thực hiện một loạt các nhiệm vụ được xác định trước. Một ví dụ Reactive AI là chương trình cờ vua nổi tiếng IBM đánh bại nhà vô địch thế giới.

b) Limited Memory AI (Bộ nhớ Giới hạn AI)

- Có thể đưa ra quyết định sáng suốt và cải thiện bằng cách nghiên cứu dữ liệu quá khứ từ bộ nhớ của nó. Một AI như vậy có bộ nhớ ngắn hạn hoặc tạm thời có thể sử dụng để lưu trữ các trải nghiệm trong quá khứ và do đó đánh giá các hành động trong tương lai.
- Xe tự lái là AI của bộ nhớ ngắn hạn, sử dụng dữ liệu được thu thập trong quá khứ gần đây để đưa ra quyết định tức thì. Ví dụ, ô tô tự lái sử dụng cảm biến để xác định người sang đường, đường đốc, tín hiệu giao thông,...để đưa ra quyết định lái xe tốt hơn, giúp ngăn ngừa mọi tai nạn trong tương lai.

c) Theory of Mind (Thuyết về Tâm trí AI)

- Là một loại TTNT tiên tiến hơn. Loại máy móc này được suy đoán là đóng một vai trò quan trọng trong tâm lý học. Tập trung vào trí tuệ cảm xúc để con người tin tưởng và suy nghĩ có

thể được thấu hiểu tốt hơn.

- Lý thuyết về Tâm trí AI vẫn chưa được phát triển đầy đủ nhưng các nghiên cứu nghiêm ngặt đang diễn ra trong lĩnh vực này.

d) Self-aware AI (AI Tự nhận thức)

- Máy móc có ý thức riêng của nó và trở nên tự nhận thức. Loại AI này hơi xa rời với hoàn cảnh hiện tại. Tuy nhiên, trong tương lai, có thể đạt được một giai đoạn siêu trí tuệ.

Lịch sử phát triển

1943	McCulloch & Pitts: Mô hình hóa mạch bool của bộ não
1950	Bài báo "Computing Machinery and Intelligence" - Turing
1956	Hội nghị Dartmouth: "Artificial Intelligence" khai sinh
1950-59	Những chương trình AI đầu tiên, bao gồm trò chơi checker của Samuel's, Logic Theorist của Netwell & Simon, máy Hình học của Gelernter
1965	Thuật toán đầy đủ của Robinson cho lập luận logic
1966-79	AI khám phá ra độ phức tạp thuật toán Nghiên cứu về mạng Neuron hầu như biến mất
1969-79	Những bước phát triển ban đầu của các hệ dựa tri thức
1980	AI trở thành ngành công nghiệp
1986	Mạng neuron lại trở nên thông dụng
1987	AI trở thành một ngành khoa học
1990	Sự nổi lên của **Machine Learning**
1995	Sự nổi bật của các agent thông minh
2010s	Công nghiệp nặng đầu tư vào **Deep Learning**

Sự khác biệt AI - ML - DL: (AI (ML (DL))).

AI: Human Itelligence Exhibited by Machines

ML: An Approach to Achieve Artificial Intelligence

DL: A Technique for Implementing Machine Learning. Thanks to DL, AI has a bright future.

Cách thức máy học:

Từ [mẫu huấn luyện] đưa vào [thuật toán học] tạo ra được một [chương trình đơn giản] chứa các tham số của mạng học (neuron networks), chương trình này có thể hiểu là model AI. Model này nhận đầu vào và cho ra kết quả như các chương trình khác.

2 Chương trình Tiến hóa và Ứng dụng

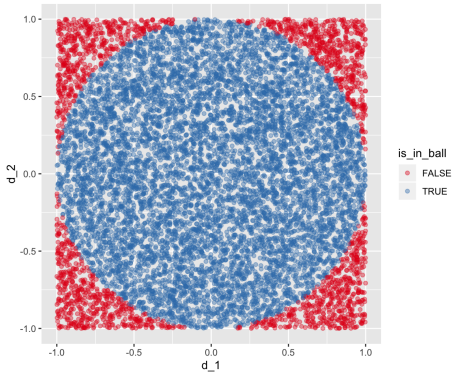
2.1 Sự tiến hóa của thuật toán

$$5x^2 - 8x - 4 = 0$$

TS - Thử sai: Vét cạn, duyệt toàn bộ; mê cung, mê lộ; ngẫu nhiên (genetic) eg. ~~$x = 0.1$~~, ~~$x = 0.2$~~ x=2 !!!

He - Heuristic: Nguyên lý vét cạn thông minh; nguyên lý Greedy; nguyên lý sắp thứ tự, nguyên lý hướng đích, Hàm Heuristic, các hệ GPS (General Problem Solver) eg. thu hẹp vùng nghiệm
TrT - Trực tiếp: Chính xác, gần đúng, đệ quy, quy hoạch động,... eg. công thức nghiệm của phương trình bậc 2 theo định lý Vi-et
K (Knowledge) - Trí tuệ nhân tạo gồm: các hệ cơ sở TTNT, Hệ chuyên gia, Mạng Neural nhân tạo
Sử dụng A - Algorithm (gồm TrT + TS)
Sử dụng PP Gián tiếp (TS + H:Heuristic + K:Knowledge & Trí tuệ nhân tạo).

Phương pháp Monte-Carlo



Trong lĩnh vực máy tính, thuật toán Monte Carlo là thuật toán ngẫu nhiên có đầu ra có thể không chính xác với xác suất nhất định (thường nhỏ). Hai ví dụ về các thuật toán như vậy là thuật toán Karger–Stein và thuật toán Monte Carlo cho tập hợp cung phản hồi tối thiểu.
Phát biểu bài toán: Cho hình đa giác M cần tính diện tích. Ta cho M nội tiếp trong hình vuông S có cạnh là 1 đơn vị. Sau đó lấy 1 điểm ngẫu nhiên P_i trong hình vuông S, kiểm tra điểm P_i có nằm trong M hay không. Lặp lại cho đến P_n. Ta có:

$$\frac{M}{N_{\square}} \to \lim_{N_{\square}\to\infty} \frac{M}{N_{\square}} = \frac{dtM}{dtN_{\square}} \approx dtM$$

với $N_{\square}$ là tổng số điểm P nằm trong hình vuông S; M là tổng số điểm P nằm trong hình M
Áp dụng tính số pi:

```
import random as rd
niter = int(input("Number of iterations:"))
count = 0

for i in range(niter):
    x = rd.random()
    y = rd.random()
    if(x*x + y*y <= 1):
        count += 1
pi = count/niter*4
print("Pi after ", niter, " iterations is: ",pi)
```

Result:

Enter the number of iterations in the first quadrant of unit circle: 10.000.000.000

Pi after 10.000.000.000 iterations is: 3.1415979652

Code in Cpp:

```cpp
#include <stdio.h>
#include <iostream>
using namespace std;

int main() {
    double niter;
    cout << "Number of iterations: "; cin >> niter;

    int count = 0;
    for (int i = 0; i < niter; i++) {
        double x = ((double)rand() / (RAND_MAX));
        double y = ((double)rand() / (RAND_MAX));
        if (x * x + y * y <= 1)
            count++;
    }
    double pi = (double)count / niter * 4;
    cout << pi;
    return 0;
}
```

Result:

Enter the number of iterations in the first quadrant of unit circle: 10.000.000.000

Pi after 10.000.000.000 iterations is: 3.1415979652

2.2 Thuật toán di truyền (Genetic Algorithm)

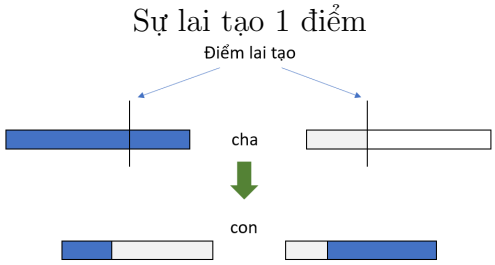
Sự tiến hóa của chương trình:

- Chương trình truyền thống = cấu trúc dữ liệu + thuật toán.
- Chương trình tiến hóa = cấu trúc dữ liệu + thuật toán Genetic. (Zbigniew Michalewicz, 1992 and John Holland, 1975)

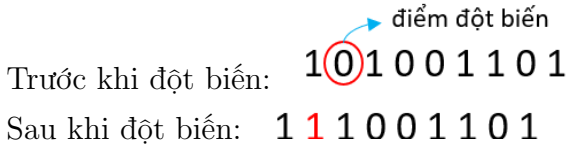
2.2.1 Các thành phần

- Nhiễm sắc thể.
- Quần thể.
- Các phép toán:
    a. Lai ghép
    b. Đột biến
- Hàm thích nghi (chọn lọc tự nhiên).

a) Lai ghép



b) Đột biến Sự đột biến 1 điểm:



2.2.2 Procedure

$t \leftarrow 0$
khởi tạo lớp P(t)
đánh giá lớp P(t)
while not (điều kiện kết thúc) **do**
  $t = t + 1$
  chọn lọc P(t) từ P($t-1$)
  kết hợp các cá thể của P(t)
  đánh giá lớp P(t)

2.2.3 Nguyên tắc cơ bản

- Được giới thiệu bởi John Holland năm 1975, cho phép thực hiện tìm kiếm ngẫu nhiên.
- Mã hóa các lời giải tiềm năng của bài toán bằng các nhiễm sắc thể.
- Đánh giá độ tốt của các lời giải qua độ thích nghi của các nhiễm sắc thể.
- Lưu trữ một quần thể qua các lời giải tiềm năng.
- Thực hiện các phép toán di truyền để phát sinh các cá thể mời, đồng thời áp dụng chọn lọc tự nhiên trên các lời giải.

a. Tính chất đặc thù

b. Các bước cơ bản

1. *Chọn mô hình cho giải pháp của vấn đề:* chọn một số tượng trưng cho toàn bộ các giải pháp có thể có cho vấn đề.

2. Chỉ định cho mỗi giải pháp một kí hiệu. Kí hiệu có thể là một dãy số 1 và 0 thuộc hệ nhị phân, hay dãy số thập phân, dãy chữ hay hỗm hợp của số và chữ.

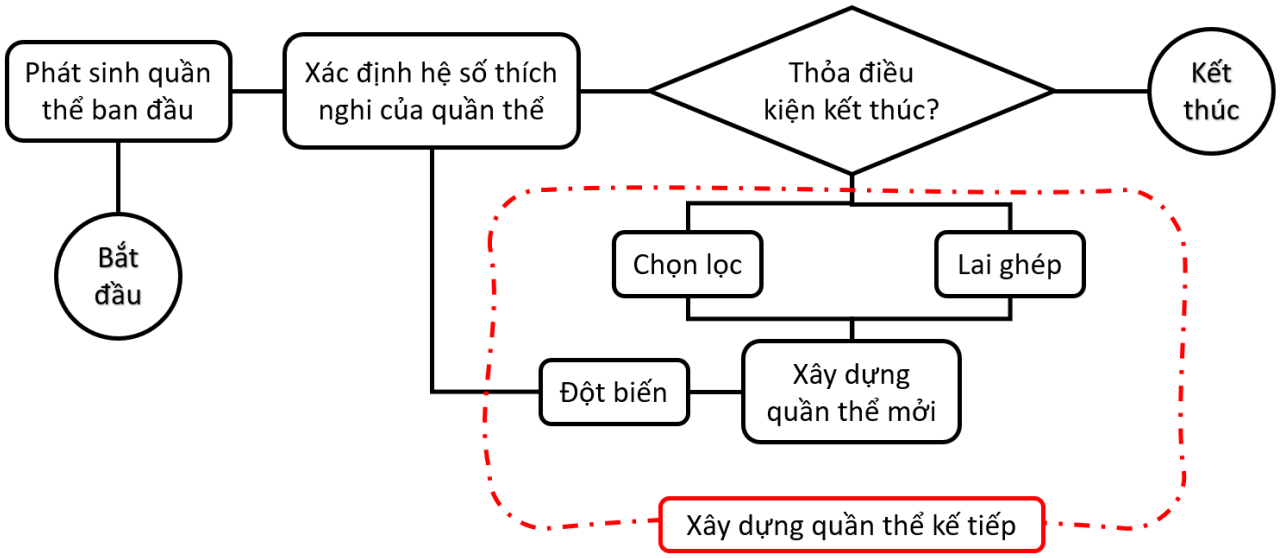
3. Tìm hàm số thích nghi cho vấn đề và tính hệ số thích nghi cho từng giải pháp.

4. Dựa trên hệ số thích nghi của các giải pháp để thực hiện sự sinh sản (Reproduction) và biến hóa các giải pháp: Lai ghép (Cross over), Đột biến (Mutation).

5. Tính các hệ số thích nghi cho các giải pháp mới và loại bỏ những giải pháp kém nhất, chỉ giữ lại một số nhất định các giải pháp.
6. Nếu chưa tìm được giải pháp tối ưu hay tương đối khá nhất hay chưa hết hạn thời gian, trở lại bước thứ 4 để tìm thêm các giải pháp mới.
7. Tìm được giải pháp hay hết thời gian cho phép, báo cáo được.

Notes: Tìm 1 lời giải hay mọi lời giải? Quay lại tìm tiếp lời giải (Backtracking).

c. Sơ đồ thuật giải di truyền



d. Các phương thức biến hóa GA

d.1 Sinh sản: Là dùng những thành phần của thế hệ trước để tạo thêm thành phần cho thế hệ sau. Giống trong tự nhiên, thành phần nào có hệ số thích nghi lớn hơn sẽ có cơ hội được chọn để thực hiện việc sinh sản.

d.2 Lai ghép: Dùng thông tin thế hệ trước có sẵn trong thành phần và truyền lại thế hệ sau.

Eg.

Thế hệ 1	Thế hệ 2
A 0 1 1 0 0	A' 1 1 0 0 0
B 1 1 0 1 0	B' 0 1 1 1 0

- Phát ngẫu nhiên: lai ghép 1 vị trí 2 bit (bit 3 bit 4).

d.3 Đột biến: Thay đổi trị số 1 số trong dãy.

Theo Kenneth De Jong, có bảng tần số sử dụng các phương thức trong thuật toán GA.

Phương thức	Lai ghép	Đột biến	Sinh sản
Tần suất	0.6	0.001	0.399

Bài tập Giải phương trình bậc 2: $ax^2 + bx + c = 0$

- Input: a, b, c
- Output: nghiệm
- Phương pháp: di truyền

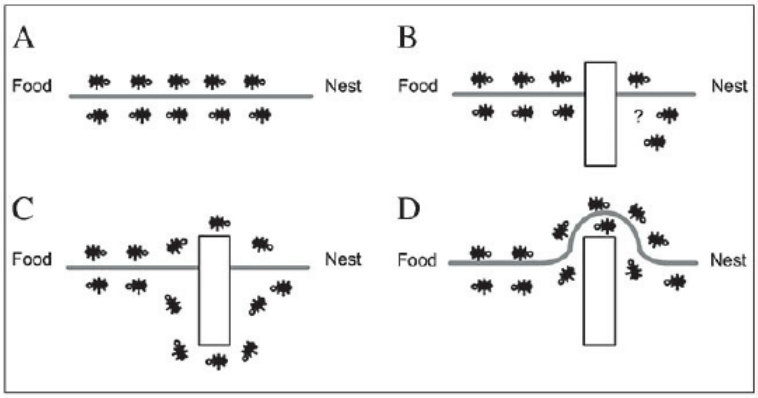
Lời giải: Cpp Ideone or Cpp Drive or Python Ideone

e. Swarm Intelligence (SI): Bầy đàn tốt hơn cá nhân. AI mô tả mục tiêu thiết kế các hệ thống đa tác nhân thông minh bằng cách lấy cảm hứng từ hành vi tập thể của các xã hội động vật như đàn kiến, đàn cá, đàn ong,...

- Tính chất:
  - Bao gồm một tập hợp các thực thể đơn giản.
  - Tính phân tán: Không có kiểm soát toàn cục.
  - Tự tổ chức giao tiếp: Direct (tiếp xúc trực quan / hóa chất) & Indirect (Stigmergy - Grass'e, 1959).
- Kết quả: Các nhiệm vụ, hành vi phwucs tạp có thể được hoàn thành, thể hiện với sự hợp tác.
- Đặc tả:
  - Từng cá thể riêng biệt có khả năng hạn chế.
  - Không có sự tác động điều khiển hay điều hành từ bên ngoài.
  - Không có hệ thống quản lý tập trung.
  - Thực hiện được nhiệm vụ khó khăn mà từng cá thể không làm được.

⇒ Mô hình để giải các bài toán phân tán.

Đường đi của đàn kiến (Sử dụng Pheromone): Từ tổ kiến, các con kiến sẽ đi tìm thức ăn ở nhiều hướng khác nhau. Trong quá trình di chuyển, kiến sẽ tiết ra pheromone như một tín hiệu hóa học để đánh dấu đường đi. Theo thời gian, nó sẽ bay mùi dần. Khi kiếm được thức ăn, kiến sẽ dựa vào pheromone đã tiết ra để quay về tổ. Đường đi nào ngắn hơn (tốn ít thời gian để đi) thì lượng pheromone lưu lại sẽ nhiều và dày hơn, ngược lại, đường đi dài hơn lượng pheromone sẽ ít hơn. Những con kiến khác sẽ dựa vào nồng độ pheromone để đi theo. Dần dần lượng kiến đi theo con đường đó sẽ nhiều hơn và dần hình thành con đường di chuyển của đàn kiến.



Thuật giải ACO (Ant Colony Optimization)

- Do Marco Dorigo đưa ra 1991, mô phỏng cách tìm đường đi của bầy kiến trong tự nhiên → giải các bài toán tối ưu tổ hợp. - Thuật giải:
  - While (điều kiện lặp):
    - Tạo đàn kiến: Điểm bắt đầu tùy theo bài toán. Khởi tạo pheromone bằng nhau tại các vị trí.

- Tìm lời giải: Xác suất chuyển vị trí

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}(t)]^\alpha [\eta_{ij}]^\beta} \qquad \forall j \in N_i^k$$

- Cập nhật pheromone: Tăng pheromone tại các cạnh có kiến đi qua. Giảm pheromone tại tất cả các cạnh (mô phỏng sự bay hơi).

3 Chủ đề 1: Các giải pháp Tìm kiếm Mù (Blind Search)

Trong suốt lịch sử, các câu đố và trò chơi yêu cầu khám phá các lựa chọn thay thế đã được coi là một thách thức đối với trí thông minh của con người:

- Cờ (chess) bắt nguồn từ Ba Tư và Ấn Độ khoảng 4000 năm trước.
- Cờ (checkers) xuất hiện trong các bức tranh Ai Cập 3600 năm tuổi.
- Cờ vây (Go) bắt nguồn từ Trung Quốc hơn 3000 năm trước.

Vì vậy, không có gì ngạc nhiên khi AI sử dụng các trò chơi để thiết kế và kiểm tra các thuật toán.

Eg. Bài toán 15-puzzle.

3.1 Dẫn nhập

- Cho bài toán/vấn đề: A⇒ B

A: giả thiết — S_i: trạng thái (state)

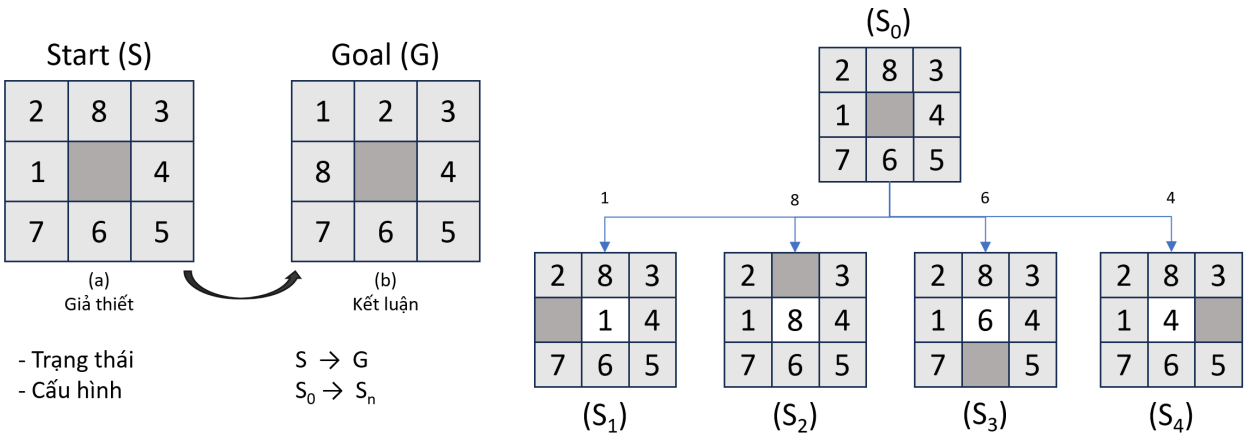
B: kết luận — $\{S_i\} = \{S_0, S_1, ..., S_n\}$: không gian trạng thái (space state)

⇒: lời giải — Biểu diễn SS bằng đồ thị (đỉnh = state, cung: ⇒)

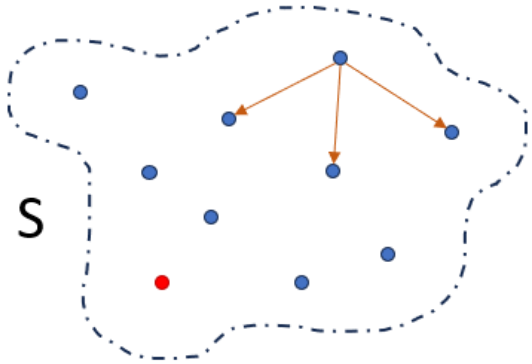
Có A, B: tìm ⇒ — $S_0 \Rightarrow S_1 \Rightarrow ... \Rightarrow S_n$

Có A, ⇒: tìm B (tiến)

Có ⇒, B: tìm A (lùi)



3.1.1 Nêu vấn đề như một bài toán tìm kiếm



- Không gian trạng thái S
- Hàm Successor:
$x \in S \rightarrow \text{SUCCESSORS}(x) \in 2^S$
- Trạng thái ban đầu s_0
- Trạng thái mục tiêu:
$x \in S \rightarrow \text{GOALS}(x) = T \text{ or } F$
- Chi phí Arc

3.1.2 Đồ thị trạng thái

- Mỗi trạng thái được biểu diễn bởi một đỉnh riêng biệt.
- Một cung (hoặc cạnh) kết nối một đỉnh s đến một đỉnh s' nếu s' = SUCCESSORS(s).
- Đồ thị trạng thái có thể chứa nhiều hơn 1 thành phần liên thông.
- *Một lời giải là một đường đi kết nối nút ban đầu với nút mục tiêu (bất kì đường đi nào.*
- Chi phí của một đường đi là tổng chi phí các cạnh trên đường đi này.
- Một lời giải tối ưu là một đường đi có chi phí tối thiểu.
- Có thể không có lời giải nào!

Không gian trạng thái của trò chơi (n^2-1)-puzzle

- Thông thường không khả thi (hoặc quá tốn kém) để xây dựng một biểu diễn hoàn chỉnh của đồ thị trạng thái. (Ví dụ bên dưới áp dụng tốc độ 100 triệu trạng thái/giây).

- 8-puzzle $\Rightarrow 9! = 362.880$ trạng thái. (0.036 giây)
- 15-puzzle $\Rightarrow 16! \approx 2.09 \times 10^{13}$ trạng thái. ($\approx$ 55 giờ)
- 24-puzzle $\Rightarrow 25! \approx 10^{25}$ trạng thái. ($\approx 10^9$ năm)

Nhưng chỉ có 1 nửa của những trạng thái này có thể tiếp cận được từ bất kì trạng thái khác (nhưng có thể không biết trước điều đó).

- Do đó, người giải quyết vấn đề phải xây dựng giải pháp bằng cách khám phá một phần nhỏ của đồ thị.

3.2 Bài toán tìm kiếm

3.2.1 Phát biểu

Gồm 5 thành phần: Q, S, G, succs, cost

- Q: tập hữu hạn các trạng thái.
- S $\subseteq$ Q: tập khác rỗng gồm các trạng thái ban đầu.
- G $\subseteq$ Q: tập khác rỗng gồm các trạng thái đích.

- succs(s): hàm nhận một trạng thái s làm đầu vào và trả về một tập các trạng thái có thể đến từ s trong một bước.

- cost(s, s'): là hàm nhận 2 trạng thái s và s' làm đầu vào và trả về chi phí di chuyển một bước từ s đến s'. Hàm chi phí chỉ được định nghĩa khi s' là trạng thái con của s.

Eg. Bài toán du lịch ở Romania

- Q = không gian trạng thái: các thành phố.
- S: Arad.
- G: Bucharest.
- succs(s): Đường đi đến thành phố liền kề.
- cost(s, s'): Khoảng cách giữa 2 thành phố (tính bằng đơn vị đo chiều dài).

Cây tìm kiếm

- Mỗi nút trong cây tìm kiếm là một đường đi trong đồ thị không gian trạng thái.
- Thực hiện: Mở rộng các nút tiềm năng. Duy trì một bộ nhớ fringe. Cố gắng mở rộng càng ít nút càng tốt.

3.2.2 Tìm kiếm chiều rộng

- Gán nhãn mọi trạng thái có thể đến được từ S trong 1 bước, nhưng không thể đến được trong ít hơn 1 bước.
- Tiếp đó gán nhãn mọi trạng thái có thể đến được từ S trong 2 bước, nhưng không thể đến được trong ít hơn 2 bước.
- Tiếp đó gán nhãn mọi trạng thái có thể đến được từ S trong 3 bước, nhưng không thể đến được trong ít hơn 3 bước.
- Cứ tiếp tục cho đến khi tới được trạng thái Goal.

Ghi nhớ đường đi: Khi gán nhãn một trạng thái, cần ghi nhận trạng thái trước đó. Ghi nhận này được gọi là con trỏ quay lui (backpointer). Lịch sử các trạng thái trước dùng để phát sinh lời giải đường đi khi đã tìm thấy đích.

- Với trạng thái s bất kì đã gán nhãn

- *previous*(s) là trạng thái trước đó trên đường đi ngắn nhất từ trạng thái START đến s.
- Tại vòng lặp thứ k của thuật toán, bắt đầu với V_k là tập các trạng thái mà đường đi ngắn nhất từ START đi đến chúng có đúng k bước.
- Tiếp đó, trong vòng lặp, tính V_{k+1} là tập các trạng thái mà đường đi ngắn nhất từ START có đúng $k+1$ bước.
- Ta bắt đầu với $k = 0$, $V_0 = \{\text{START}\}$ và định nghĩa *previous*(START) = NULL.
- Tiếp đó ta thêm vào những trạng thái đi một bước từ START vào V_1 và cứ thế tiếp diễn.

Mô tả thuật toán

V_0 := S (tập các trạng thái khởi đầu)

previous(START) := NULL
k := 0
while (không có trạng thái đích trong V_k và V_k không rỗng) **do**
V_{k+1} := tập rỗng
Với mỗi trạng thái s trong Vk
Với mỗi trạng thái s' trong **succcs**(s)
Nếu s' chưa được gán nhãn
Đặt *previous*(s') := s
Thêm s' vào V_{k+1}
k := k+1
If V_k rỗng **Then** thông báo THẤT BẠI
Else xây dựng lời giải đường đi: Gọi S_i là trạng thái thứ i trong đường đi ngắn nhất. Định nghĩa S_k = GOAL và với mọi $i \leq k$ | S_{i-1} = *previous*(S_i).
Một cách khác **Đi lui**:
- Gán nhãn mọi trạng thái có thể đi đến G trong 1 bước nhưng không thể đi đến trong ít hơn 1 bước.
- Tiếp theo gán nhãn mọi trạng thái có thể đi đến G trong 2 bước nhưng không thể đi đến trong ít hơn 2 bước.
- ... cho đến khi đến trạng thái START.
- Các nhãn "số bước đến đích" xác định đường đi ngắn nhất, không cần thêm thông tin lưu trữ.

3.2.3 Nhận xét

- Vẫn chạy tốt khi có nhiều hơn một trạng thái đích hoặc nhiều hơn một trạng thái ban đầu.
- Thuật toán thực hiện tiến tới từ trạng thái bắt đầu. Những thuật toán thực hiện tiến tới từ trạng thái bắt đầu gọi là *suy diễn tiến*.
- Thuật toán này rất giống với thuật toán Dijkstra's.
- Có thể thực hiện lùi dần từ đích. Thuật toán thực hiện lùi dần từ đích gọi là *suy diễn lùi*.
- Lùi và tiến. Phương pháp nào tốt hơn?