MẠNG NORON NHÂN TẠO và GIẢI THUẬT DI TRUYỀN

Neural Network & Genetic Algorithm

Biên soạn: ThS.Vương Xuân Chí
vxchi@ntt.edu.vn
0903.270.567

CHƯƠNG 7

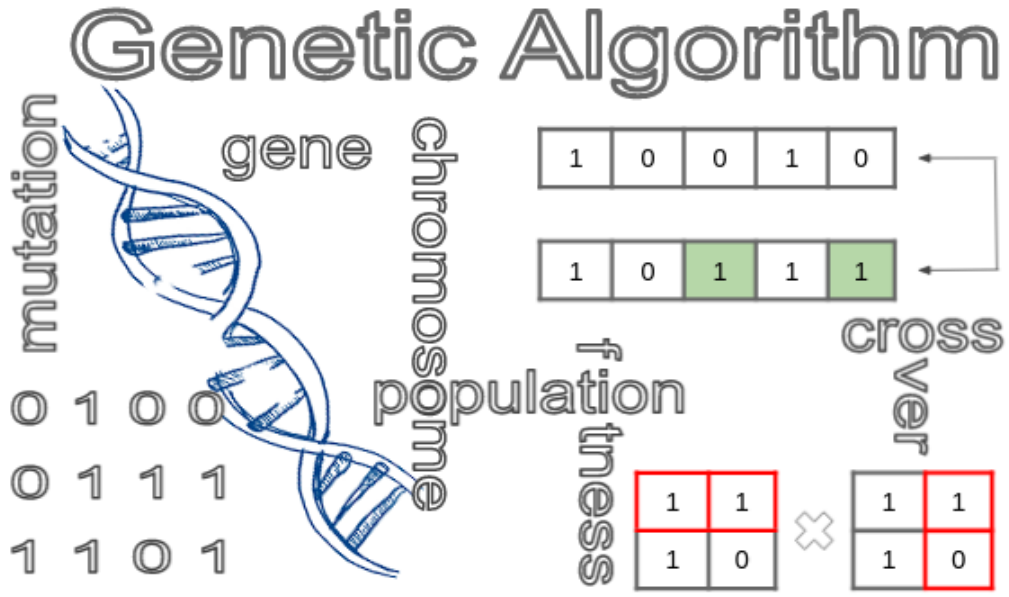
Ứng dụng GT di truyền (GAs)

MỤC TIÊU

- ✓ **Giới thiệu các ứng dụng của giải thuật di truyền**
- ✓ **Một số bài toán cụ thể**
- ✓ **Bài toán do tìm Password**

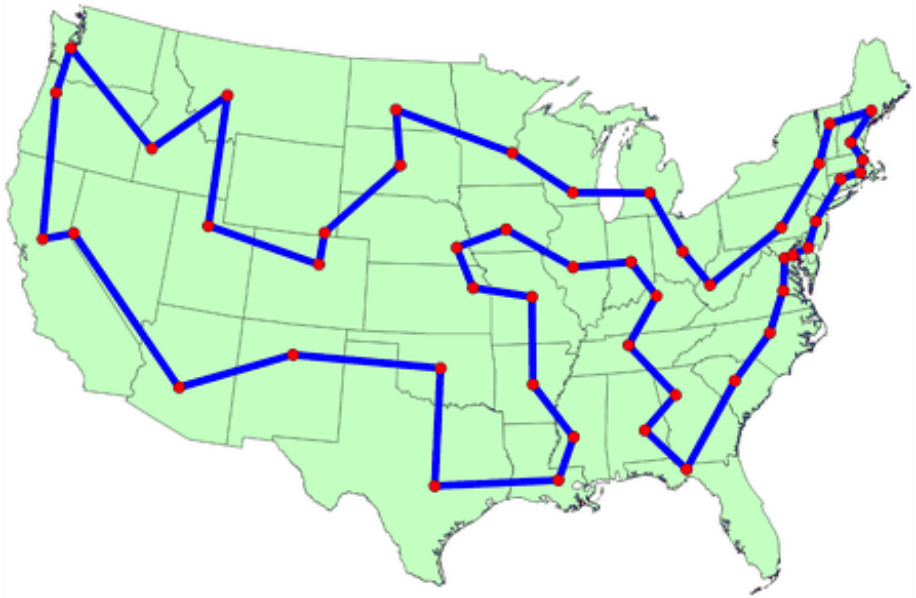
ỨNG DỤNG GIẢI THUẬT DI TRUYỀN

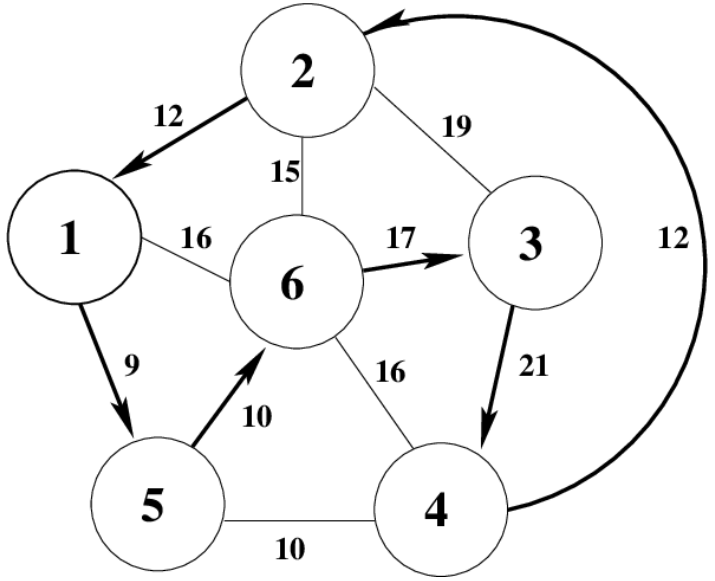
- GAs được sử dụng cho những bài toán khó, và đã được ứng dụng thành công cho một số bài toán như: lập kế hoạch, điều khiển tương thích, chương trình trò chơi, các bài toán vận tải, bài toán người đi du lịch,...



BÀI TOÁN NGƯỜI DU LỊCH (TSP)

- TSP được mô tả như sau: Một du khách muốn thăm những thành phố anh quan tâm; mỗi thành phố thăm qua đúng một lần; rồi trở về điểm khởi hành. Biết trước chi phí di chuyển giữa hai thành phố bất kỳ. Yêu cầu của bài toán là xây dựng một lộ trình thỏa các điều kiện trên với tổng chi phí nhỏ nhất.



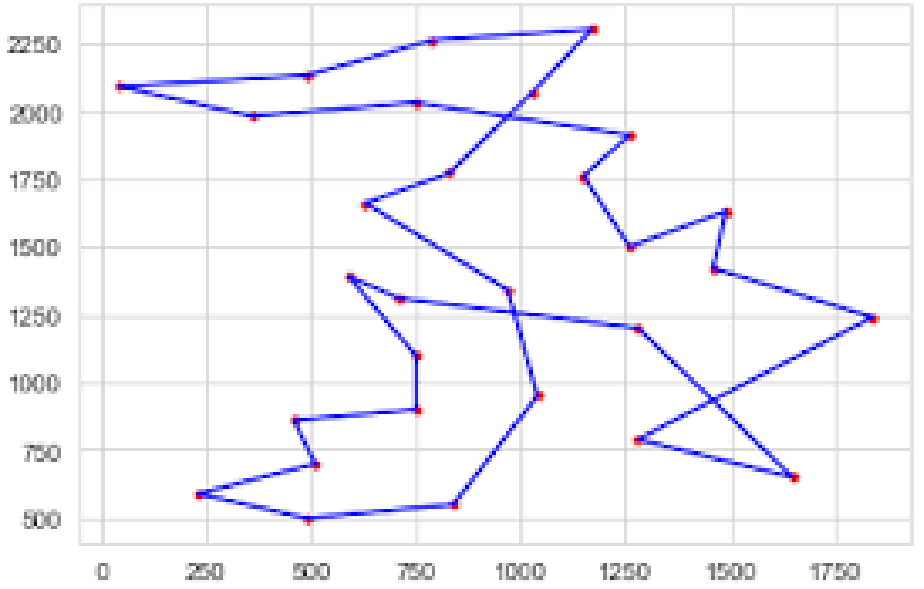


BÀI TOÁN NGƯỜI DU LỊCH (TSP)

- TSP là bài toán tối ưu tổ hợp, không gian tìm kiếm là tập các hoán vị của n thành phố. Bất cứ hoán vị nào của n thành phố cũng là một lời giải chấp nhận được.
- Lời giải tối ưu là một hoán vị với chi phí tối thiểu của hành trình. Không gian tìm kiếm là n!. Có thể giải bài toán này bằng nhiều phương pháp: phương pháp nhánh cận, phương pháp gần đúng hay những phương pháp tìm kiếm ***heuristic***. Phương pháp nhánh cận đã được chứng minh đạt sự tối ưu về lời giải, tuy nhiên phương pháp này lại mất khá nhiều thời gian khi số đỉnh của đồ thị lớn.

BÀI TOÁN NGƯỜI DU LỊCH (TSP)

- Trong những năm gần đây, đã xuất hiện nhiều thuật toán đạt gần đến lời giải tối ưu của bài toán **TSP**: láng giềng gần nhất, đảo gần nhất, đảo xa nhất...và **TSP** cũng trở thành một đích ngắm của cộng đồng **GAs**.
- Với bài toán này chúng ta sẽ đánh số các thành phố và dùng một vector nguyên để biểu diễn một NST lộ trình $v=<i_1,i_2,...,i_n>$ biểu diễn một lộ trình: từ i_1 đến i_2..., từ i_{n-1} đến in và trở về i_1 (v là một hoán vị của vector <1,2,...,n>), hàm lượng giá chính là chi phí của lộ trình.



BÀI TOÁN LẬP LỊCH

- Lập lịch là bài toán tổ chức sản xuất. Một công ty cần sản xuất nhiều loại hang hóa; những hàng hóa này có thể được sản xuất theo những kế hoạch khác nhau. Mỗi kế hoạch xử lý gồm một chuỗi thao tác; những thao tác này sử dụng một số tài nguyên và cần thời gian chạy máy. Một lịch sản xuất là một kế hoạch thực hiện các đơn đặt hàng. Trong đó, một số đơn đặt hàng có thể được thực hiện với cùng những thao tác tương đương. Nhiệm vụ là lên kế hoạch, lập lịch sản xuất, để thực hiện các đơn đặt hàng này nhanh nhất có thể.

LẬP THỜI KHÓA BIỂU CHO TRƯỜNG HỌC

- Bài toán thời khóa biểu là một bài toán kết hợp nhiều ràng buộc không tầm thường thuộc nhiều loại. Có nhiều phiên bản của bài toán thời khóa biểu, một trong những bài toán này có thể được mô tả như sau: Có một danh sách các giáo viên, một danh sách các khoảng thời gian, một danh sách các lớp. Bài toán cần tìm thời khóa biểu tối ưu (giáo viên – thời gian – lớp); hàm mục tiêu phải thỏa những mục tiêu này (các ràng buộc mềm) gồm: Có một số giờ được xác định trước cho mỗi giáo viên và mỗi lớp.

LẬP THỜI KHÓA BIỂU CHO TRƯỜNG HỌC

- Chỉ một giáo viên trong một lớp vào một giờ nhất định; Một giáo viên không thể dạy hai lớp cùng lúc; Đối với mỗi lớp được xếp thời khóa biểu vào một khoảng thời gian, phải có một giáo viên...Ngoài ra còn có các mục tiêu sư phạm như trải một số lớp ra nguyên tuần, những mục tiêu thuộc cá nhân như những giáo viên hợp đồng không phải dạy buổi chiều, và các mục tiêu về tổ chức như mỗi giờ có một giáo viên bổ sung sẵn sàng chỗ dạy tạm thời

PHÂN HOẠCH ĐỐI TƯỢNG VÀ ĐỒ THỊ

- Bài toán phân hoạch là cần chia n đối tượng thành k loại. Lớp bài toán này gồm nhiều bài toán nổi tiếng như bài toán đóng thùng (gán các mặt hàng vào các thùng), bài toán tô màu đồ thị (gán các nút của đồ thị vào các màu cụ thể...).
- Bài toán đóng thùng **(Bin Packing – BP)** được phát biểu như sau: Cho danh sách gồm n đồ vật **$L=a_1,a_2,a_3,...,a_n$** và các thùng giống nhau với cùng sức chứa **B**. Kích thước của đồ vật **a_i** là **s_i** thỏa mãn. Vấn đề đặt ra là tìm cách xếp các đồ vật vào các thùng sao cho số lượng thùng cần phải sử dụng là ít nhất.

VẠCH ĐƯỜNG CHO ROBOT DI CHUYỂN

- Tìm đường là hướng dẫn robot di chuyển đến đích mà không bị lạc hay va vào những đối tượng khác. Trong bài toán này, một lộ trình được lập trước để robot đi theo, lộ trình này có thể dẫn dắt robot đi tới đích một cách hoàn hảo. Tuy nhiên, các nhà khoa học muốn cải tiến hơn bằng cách vạch lộ trình nội tại, phụ thuộc vào tri thức thu được từ việc cảm nhận môi trường cục bộ để xử lý các chướng ngại chưa biết.
- Bộ tìm đường tiến hóa **(EN)** được đề xuất. Phần đầu của thuật giải là tìm lộ trình toàn cục tối ưu từ điểm khởi đầu đến đến đích, phần thứ hai có trách nhiệm xử lý những va chạm có thể xảy ra hay những vật cản chưa được biết trước bằng cách thay một phần của lộ trình toàn cục gốc bằng một lộ trình con tối ưu

BÀI TOÁN ĐOÁN PASSWORD

- Về bài toán đoán password (**Guessing Password**), sử dụng những ký tự trong bảng chữ cái để tái tạo lại mật khẩu. Cụ thể, ở bài này chúng ta sẽ bắt đầu với những ký tự: **a-z, A-Z, !** . để tái tạo nên mật khẩu: **"Hello World!".**

BÀI TOÁN ĐOÁN PASSWORD

- **Các bước giải bài toán:**
  - ***Khởi tạo quần thể:*** *Sinh ra ngẫu nhiên một đoạn dài 11 ký tự (dài bằng với mục tiêu* ***"Hello World".***
  - ***Tính giá trị thích nghi****: Đếm số ký tự trùng giữa đoạn tạo ra và* ***"Hello World".***
  - ***Điều kiện dừng****: Kiểm tra xem số ký tự trùng bằng 11 hay không.*
  - ***Chọn lọc:*** *Chọn hai cá thể bố mẹ từ quần thể cũ theo độ thích nghi của chúng (cá thể có độ thích nghi càng cao thì càng có nhiều khả năng được chọn).*
  - ***Trao đổi chéo*** *(Không thực hiện ở bài toán này): Với một xác suất được chọn, trao đổi chéo hai cá thể bố mẹ để tạo ra một cá thể mới.*
  - ***Đột biến:*** *Với một xác suất đột biến được chọn, biến đổi cá thể mới.*
  - ***Chọn kết quả:*** *Khi số ký tự trùng bằng 11 thì dừng giải thuật.*

BÀI TOÁN ĐOÁN PASSWORD

- **Chromosomes (nhiễm sắc thể)**
  - Tất cả những trình tự đoạn 11 ký tự (độ dài bằng với **"Hello World")** được gọi là Chromosomes. Và mỗi loại ký tự sẽ đại diện cho 1 loại Gene khác nhau.

```
geneSet = " abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
target = "Hello World"
```

BÀI TOÁN ĐOÁN PASSWORD

- **Chromosomes (nhiễm sắc thể)**
  - Tất cả những trình tự đoạn 11 ký tự (độ dài bằng với **"Hello World")** được gọi là Chromosomes. Và mỗi loại ký tự sẽ đại diện cho 1 loại Gene khác nhau.

```
geneSet = " abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
target = "Hello World"
```

  - Đầu tiên chúng ta khai báo biến **geneSet** chứa các ký tự trong bảng chữ cái, mỗi ký tự đại diện cho một loại **Gene**. Cùng với đó, khai báo biến target là đoạn ký tự mục tiêu **"Hello World".**

BÀI TOÁN ĐOÁN PASSWORD

Bước 1: Tạo tập các thể ban đầu

- *Sử dụng random.sample để tạo ngẫu nhiên từ geneSet theo độ dài cho trước. Chúng ta có thể có test bằng initialize_chromosomes(11). Kết quả sẽ có dạng:* ***'yTtumWSgfkM'***
- *Chú ý: Vòng* ***While*** *sinh ra nhằm xử lý vấn đề nếu độ dài của* ***Chromosomes*** *dài hơn số lượng loại* ***Gene*** *ban đầu*

```
import random

def initialize_chromosomes(length):
    chromosomes = []
    while len(chromosomes) < length:
        sampleSize = min(length - len(chromosomes), len(geneSet))
        chromosomes.extend(random.sample(geneSet, sampleSize))
    return ''.join(chromosomes)
```

BÀI TOÁN ĐOÁN PASSWORD

- **Bước 2: Tính giá trị thích nghi**
  - Chúng ta thấy rằng, Chromosomes có khả năng là những trường hợp như:
    1. **Sekmo xosmd**
    2. **Seklo Wocle**
    3. **Fello Wosld**
    4. **Gello World**
    5. **Hello World**
  - Có thể nhận ra ngay đoạn cuối cùng là đoạn chính xác. Nhưng làm sao chúng ta có thể "đo" được độ tối ưu của **Chromosomes** này
  - Hàm lỗi (**error function/ cost function/loss function)** là một hàm số giúp đo độ tối ưu của một Chromosome. Ở hàm này, chúng ta đang hướng tới giá trị càng nhỏ càng tốt.

BÀI TOÁN ĐOÁN PASSWORD

- Với bài toán **“Đoán mật khẩu”**, để đo độ tối ưu, chúng ta đếm số kí tự sai khác trên cùng vị trí giữa **Chromosome** và đoạn kí tự mục tiêu **"Hello World".** Vì vậy, hàm lỗi ở đây sẽ tối ưu ở số kí tự sai khác bằng 0.
- Dùng quy luật trên để tính sai khác cho các trường hợp **Chromosomes** trên:

1. **Sekmo xosmd(6)**
2. **Seklo Wocle(4)**
3. **Fello Wosld(2)**
4. **Gello World(1)**
5. **Hello World(0)**

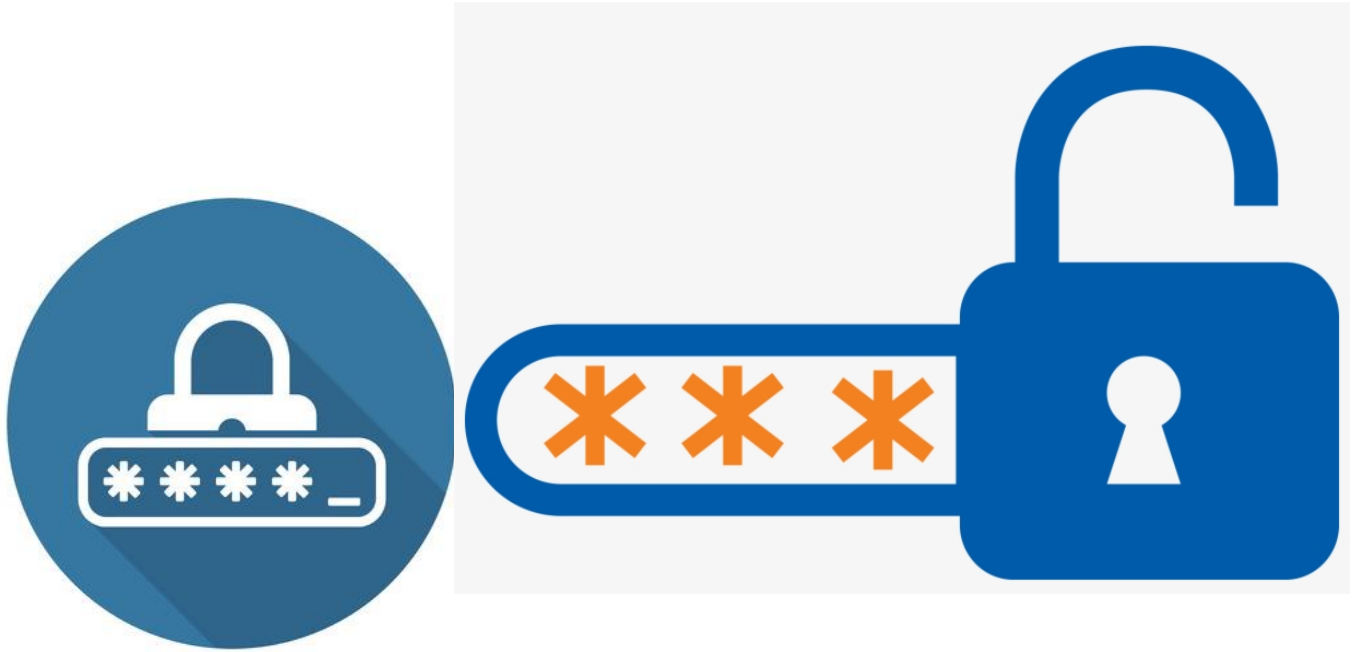
BÀI TOÁN ĐOÁN PASSWORD

```
def error_function(chromosome):
    return len(target) - sum(1 for expected, actual \
                             in zip(target, chromosome)  \
                             if expected == actual)
```

- Hàm **error_function** sẽ tính số ký tự sai khác ở cùng một vị trí giữa Chromosome và **"Hello World".** Có thể test hàm trên như sau: **error_function("Hello World")** sẽ cho kết quả bằng 0.

BÀI TOÁN ĐOÁN PASSWORD

- **Bước 3: Điều kiện dừng**
  - Kiểm tra điều kiện điểm dừng là **error_function** có bằng **0** hay không.



BÀI TOÁN ĐOÁN PASSWORD

- **Bước 4: Tiến hóa**
  - Trong bước này, chúng ta sẽ gộp chung cả 3 kỹ thuật: ***chọn lọc, trao đổi chéo và đột biến***. Mục tiêu của bước này là tạo ra thế hệ mới tối ưu hơn thế hệ cũ. Độ tối ưu sẽ được đo theo hàm lỗi. Chúng ta có thể mường tượng ra rằng, đây là một vòng lặp, mỗi vòng lặp sẽ thực hiện 3 kỹ thuật trên, sau đó check điều kiện dừng. Nếu không đáp ứng được thì vòng lặp sẽ tiếp tục cho đến khi đáp ứng được thì thôi.

```
def mutate(parent):
    index = random.randrange(0, len(parent))
    childGenes = list(parent)
    newGene, alternate = random.sample(geneSet, 2)
    childGenes[index] = alternate \
    if newGene == childGenes[index] \
    else newGene
    return ''.join(childGenes)
```

- index sẽ là một số được chọn ngẫu nhiên tức là đột biến sẽ xảy ra một cách ngẫu nhiên trên Chromosome. newGene và alternate cũng sẽ được ngẫu nhiên tạo ra từ geneSet.
- Vì sao lại phải tạo ra tận 2 đột biến? Câu trả lời là đột biến có thể vô nghĩa tại 1 điểm nên cần 1 đột biến khác thay thế.

BÀI TOÁN ĐOÁN PASSWORD

- **Hàm hỗ trợ**
  - Xây dựng hàm **display** giúp hiển thị sự thay đổi qua từng thế hệ và thời gian chạy tương ứng

```
import datetime

def display(guess):
    timeDiff = datetime.datetime.now() - startTime
    fitness = error_function(guess)
    print("{0}\t{1}\t{2}".format(guess, fitness, str(timeDiff)))
```

BÀI TOÁN ĐOÁN PASSWORD

- **Main:**

```
random.seed(1)
startTime = datetime.datetime.now()
bestParent = initialize_chromosomes(len(target))
bestFitness = error_function(bestParent)
display(bestParent)
while True:
    child = mutate(bestParent)
    childFitness = error_function(child)

    if bestFitness <= childFitness:
        continue
    display(child)
    if childFitness == 0:
        break
    bestFitness = childFitness
    bestParent = child
```

```
hJYVdpgEBDO 11 0:00:00
HJYVdpgEBDO 10 0:00:00
HJYVdpgEBDd 9 0:00:00.001003
HJYVopgEBDd 8 0:00:00.001003
HJYVopgErDd 7 0:00:00.004010
HJYVopgErld 6 0:00:00.005012
HJlVopgErld 5 0:00:00.008021
HJlVo gErld 4 0:00:00.008021
HelVo gErld 3 0:00:00.009036
HelVo gorld 2 0:00:00.010026
Hello gorld 1 0:00:00.017045
Hello World 0 0:00:00.020052
```

https://github.com/handcraftsman/GeneticAlgorithmsWithPython

Thank you !