

# YOLO + ALGORYTMY GENETYCZNE

NAJPIERW TROCHĘ GRAFIKI, POTEM TROCHĘ OPTYMALIZACJI  
(PROBLEM PODZIAŁU, PLECAKOWY, STOP METALI, LABIRYNT)

## ZADANIE 1: DETEKCIJA OBIEKTÓW Z UŻYCIEM YOLO

W załączniku otrzymasz jedno zdjęcie oraz jeden krótki film (np. kilka sekund). Użyj wybranego przez siebie modelu YOLO (dowolna implementacja) do detekcji obiektów:

- raz na pojedynczym zdjęciu,
- oraz klatka po klatce na filmie.

Szczegóły do zrobienia:

- a) Postaraj się dowiedzieć coś o wybranym YOLO: jakie klasy wykrywa? Ile ich jest? Na jakim zbiorze był treowany? (np. COCO?) Jaki model CNN jest w środku?
- b) Dla każdej detekcji ze zdjęcia zapisz do pliku (np. JSON):

- nazwę / identyfikator klasy,
- confidence (pewność detekcji),
- pozycję bounding boxa (współrzędne w wybranym przez Ciebie formacie).

- c) Poeksperytmuj z minimalnymi progami confidence (np. 0.1, 0.3, 0.5, 0.7) dla zdjęcia. Jak zmieniają się wykrycia? Czy obiekt wykrywa wszystko (nawet mniejsze i zamazane obiekty)? Czy wykrywa obiekty tam gdzie ich nie ma (false positive)? Możesz wygenerować kilka plików json, które w nazwie mają minimalny próg confidence.

- d) W przypadku filmu zrób podobnie ale:

- zapisuj wyniki dla każdej klatki (np. z numerem klatki lub czasem),
- zadbaj, aby struktura pliku była spójna (łatwa do późniejszej analizy).
- Czyli kilka plików json z różnymi programi, powinno zawierać informacje o wszystkich klatkach, a w każdej klatce o wszystkich wykryciach (trocę nam się pogeneruje!).

- e) Zapisz:

- obraz/film z narysowanymi bounding boxami i podpisami klas, w nazwie pliku dodaj jaki próg confidence był użyty.
- krótkie statystyki, np. liczba wykryć każdej klasy w całym video.



## PROBLEM PODZIAŁU

Przypomnijmy sobie wiedzę z wykładu. W problemie podziału (Partition problem) pytamy, czy da się podzielić zbiór liczb  $S$  na dwa zbiory  $S_1$  i  $S_2$  w taki sposób, że liczby w jednym i drugim podzbiorze sumują się do tej samej liczby.

Rozpatrzmy poniższą listę z 15 liczbami:

$S = [1, 2, 3, 6, 10, 17, 25, 29, 30, 41, 51, 60, 70, 79, 80]$

### IMPLEMENTACJA W PYTHONIE - PYGAD

Przedstawiony powyżej problem podziału został rozwiązyany za pomocą paczki `pygad` (co omówiono na wykładzie) <https://pygad.readthedocs.io/en/latest/>

Uruchom i przeanalizuj poniższe rozwiązanie (załączone jest też jako osobny plik `partition_ga.py`)

```
import pygad
import numpy

S = [1, 2, 3, 6, 10, 17, 25, 29, 30, 41, 51, 60, 70, 79, 80]

#definiujemy parametry chromosomu
#geny to liczby: 0 lub 1
gene_space = [0, 1]

#definiujemy funkcję fitness
def fitness_func(solution, solution_idx):
    sum1 = numpy.sum(solution * S)
    solution_invert = 1 - solution
    sum2 = numpy.sum(solution_invert * S)
    fitness = -numpy.abs(sum1-sum2)
    #lub: fitness = 1.0 / (1.0 + numpy.abs(sum1-sum2))
    return fitness

fitness_function = fitness_func

#ile chromosomów w populacji
#ile genów ma chromosom
sol_per_pop = 10
num_genes = len(S)

#ile wylaniamy rodziców do "rozmanazania" (około 50% populacji)
#ile pokolen
#ilu rodziców zachować (kilka procent)
num_parents_mating = 5
num_generations = 30
keep_parents = 2

#jaki typ selekcji rodziców?
#sss = steady, rws=roulette, rank = rankingowa, tournament = turniejowa
parent_selection_type = "sss"

#w ilu punktach robić krzyzowanie?
crossover_type = "single_point"

#mutacja ma działać na ilu procent genów?
#trzeba pamiętać ile genów ma chromosom
mutation_type = "random"
mutation_percent_genes = 8
```

```

#inicjacja algorytmu z powyzszymi parametrami wpisanymi w atrybuty
ga_instance = pygad.GA(gene_space=gene_space,
                        num_generations=num_generations,
                        num_parents_mating=num_parents_mating,
                        fitness_func=fitness_function,
                        sol_per_pop=sol_per_pop,
                        num_genes=num_genes,
                        parent_selection_type=parent_selection_type,
                        keep_parents=keep_parents,
                        crossover_type=crossover_type,
                        mutation_type=mutation_type,
                        mutation_percent_genes=mutation_percent_genes)

#uruchomienie algorytmu
ga_instance.run()

#podsumowanie: najlepsze znalezione rozwiazanie (chromosom+ocena)
solution, solution_fitness, solution_idx = ga_instance.best_solution()
print("Parameters of the best solution : {solution}".format(solution=solution))
print("Fitness value of the best solution =
{solution_fitness}".format(solution_fitness=solution_fitness))

#tutaj dodatkowo wyswietlamy sume wskazana przez jedynki
prediction = numpy.sum(S*solution)
print("Predicted output based on the best solution :
{prediction}".format(prediction=prediction))

#wyswietlenie wykresu: jak zmieniala sie ocena na przestrzeni pokolen
ga_instance.plot_fitness()

```

## PROBLEM PLECAKOWY

Na wykładzie mówiliśmy o problemie „Złodziej”, który jest w istocie tak zwanym problemem plecakowym (Knapsack problem). W problemie plecakowym dana jest lista przedmiotów o wartościach i wagach. Chcemy do plecaka zabrać najcenniejsze rzeczy. Pytanie brzmi: jaki zestaw przedmiotów (o łącznej maksymalnej wadze  $n$  kg) ma największą wartość?

Rozpatrzmy instancję problemu:

$n = 25$  kg (limit wagi), lista przedmiotów:

	przedmiot	wartosc	waga
1	zegar	100	7
2	obraz-pejzaż	300	7
3	obraz-portret	200	6
4	radio	40	2
5	laptop	500	5
6	lampka nocna	70	6
7	srebrne sztućce	100	1
8	porcelana	250	3
9	figura z brązu	300	10
10	skórzana torba	280	3
11	odkurzacz	300	15

## ZADANIE 2: PROBLEM PLECAKOWY



Rozwiąż powyższy problem plecakowy w Pythonie z użyciem paczki pygad. Możesz skorzystać z kodu z partition\_ga.py, który trzeba rozsądnie zmodyfikować.

Aspekty do zastanowienia:

- Jak zakodować problem w Pythonie? Słownik, lista, kilka list?
- Jak napisać sensownie funkcję fitness? Wskazówki były na wykładzie.
- Jak sensownie dobrać parametry algorytmu dla naszego problemu?  
(wielkość populacji, mutacja, itp.)
- Odpowiedz na pytania: Jakie jest najlepsze rozwiązanie? Które przedmioty powinniśmy zabrać? Jaką mają wartość? Jak przebiegała optymalizacja (wykres)?

Chcielibyśmy też sprawdzić skuteczność i szybkość algorytmu genetycznego. Odpowiedz dodatkowo na dwa pytania:

- Na 10 odpaleń algorytmu genetycznego, ile z nich znalazło najlepsze rozwiązanie (wartość 1630)? Podaj w procentach. Uwaga: za każdym razem trzeba na nowo tworzyć instancję modelu (ga\_instance).
- Jaki jest średni czas działania algorytmu w przypadku znalezienia najlepszego rozwiązania? Tutaj trzeba uwzględnić następujące aspekty:
  - Algorytm genetyczny powinien zatrzymać ewolucję gdy znajdzie rozwiązanie 1630 zł. Dalsze szukanie rozwiązań sztucznie przedłuża jego działanie. Jak to zrobić?  
[https://pygad.readthedocs.io/en/latest/README\\_pygad\\_ReadTheDocs.html#stop-criteria](https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html#stop-criteria)
  - Mierzenie czasu można zrobić z pomocą modułu time:  
`import time`  
  
`start = time.time()  
print("hello")  
end = time.time()  
print(end - start)`
    - Uśrednij czas z 10 udanych prób.

## PROBLEM INŻYNIERYJNY: STOP METALI



W pewnym zakładzie badawczym inżynierowie próbowali stworzyć bardzo trwały stop sześciu metali. Ilości wszystkich 6 metali w stopie oznaczone zostały symbolami  $x, y, z, u, v, w$  i są to liczby z przedziału  $[0, 1]$ . Okazało się, że wytrzymałość stopu określona jest przez funkcję:

$$\begin{aligned} \text{endurance}(x, y, z, v, u, w) \\ = e^{-2 \cdot (y - \sin(x))^2} + \sin(z \cdot u) + \cos(v \cdot w) \end{aligned}$$

Obliczenie maksymalnej wytrzymałości (endurance) było dla inżynierów problematyczne. Poproszono Ciebie, eksperta od sztucznej inteligencji, o rozwiązanie problemu.

### ZADANIE 3: STOP METALI

W tym zadaniu rozwiążemy problem inżynierijny za pomocą algorytmu genetycznego. Naszym celem jest znalezienie odpowiedzi na dwa pytania:

- Jaka jest najlepsza wytrzymałość stopu metali?
- Jakie ilości metali trzeba zmieszać, by uzyskać najbardziej wytrzymały stop?

Przydatny fragment kodu (po importie biblioteki math):

```
def endurance(x, y, z, u, v, w):  
    return math.exp(-2*(y-math.sin(x))**2)+math.sin(z*u)+math.cos(v*w)
```

Zadanie to jest inne niż te z poprzednich laboratoriów, gdyż musimy się zmierzyć z chromosomami, które nie mają zer i jedynek jak genów. Zamiast nich, mają liczby rzeczywiste (zmiennoprzecinkowe) z przedziału [0, 1].

Przykładowy chromosom:  $A = [0.09, 0.06, 0.99, 0.98, 0.1, 0.15]$

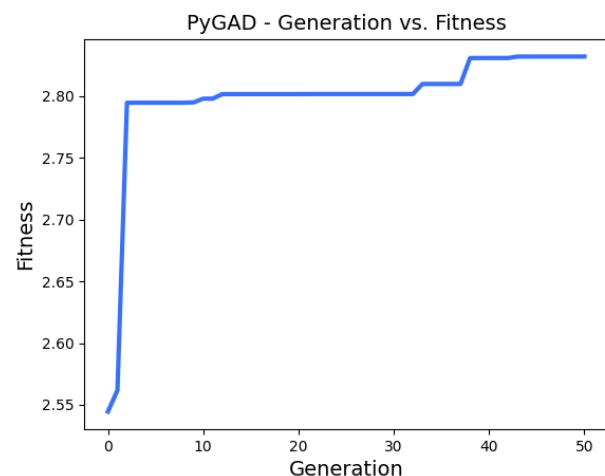
Rozwiąż ten problem z użyciem paczki pygad i odpowiedz na pytania zadane na początku tego zadania.

- Otwórz dokumentację i korzystaj z niej w razie problemów:  
[https://pygad.readthedocs.io/en/latest/README\\_pygad\\_ReadTheDocs.html](https://pygad.readthedocs.io/en/latest/README_pygad_ReadTheDocs.html)
- Zdefiniuj poprawnie gene\_space.
- Zdefiniuj sensowną funkcję fitness (to jest u nas banalnie proste 😊)
- Ustaw sensowne parametry związane z populacją i długością chromosomu (6!).
- Chromosom jest krótki, mutację trzeba zwiększyć do kilkunastu procent, żeby nie dostawać czerwonego warninga.  
**UserWarning: The percentage of genes to mutate ...**
- Uruchom algorytm genetyczny. Powtórz kilka razy, by zobaczyć czy wyniki i wykresy się zmieniają.

U mnie wyniki wyszły takie (najlepszy stop miał 2.83 endurance). U Ciebie powinno wyjść podobnie, choć proporcje metali mogą wyjść inne:

Parameters of the best solution : [0.20431118 0.20483286 0.99940818 0.98341734 0.21199392 0.01554426]

Fitness value of the best solution = 2.832060419993197

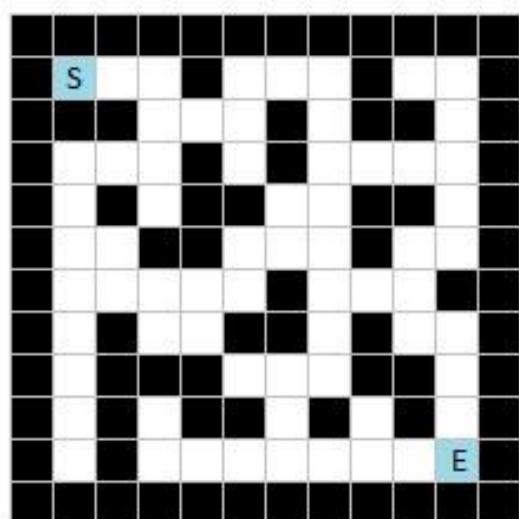


## PROBLEM DROGI W LABIRYNCIE

Dany jest prostokątny labirynt składający się z kwadratowych pól. Niektóre pola są puste i można po nich chodzić (kratki białe), a niektóre pola są ścianami i nie można na nie wchodzić (kratki czarne). Można założyć, że labirynt jest otoczony ramką ścianek, tak aby nie dało się wyjść poza jego granice.

Po labiryncie chodzimy przeskakując z pola na jedno z sąsiednich pól (po lewej, prawej, u góry lub na dole). Nie są legalne skoki na ukos.

W lewym górnym rogu labiryntu znajduje się pole startu (S), a w prawym dolnym rogu pole wyjścia (exit, E). Czy istnieje ścieżka z S do E o określonej maksymalnej długości?



Żeby było łatwiej, weźmy konkretny przykład (instancję problemu).

Mamy labirynt **12x12 pól**, uwzględniając ramkę ze ścianek, lub 10x10 pól, nie uwzględniając tej ramki. Rysunek powyżej.

W labiryncie tym szukamy drogi o maksymalnej liczbie **30 kroków**. Czy istnieje taka droga? Jeśli tak, jak ona wygląda?

Oczywiście, ponieważ labirynt jest łatwy to od razu znajdziemy rozwiązanie. Z bardzo skomplikowanymi labiryntami byłoby jednak gorzej. A jak poradzi sobie algorytm genetyczny z labiryntem?

### ZADANIE 4: LABIRYNT

Stosując paczkę pygad, rozwiąż problem szukania drogi w labiryncie za pomocą algorytmu genetycznego.

Aspekty organizacyjne do rozważenia (były omówione dość obszernie na wykładzie)

- Jak zakodować labirynt?
- Jak zakodować chromosomy? Jaką mają długość? Jakie mają geny?
- Jaką wybrać funkcję fitness? Jakie oceny będzie wystawiała?
- Problem wydaje się być cięższy niż poprzednie. Czy warto zwiększyć populację?

Gdy rozstrzygniesz aspekty organizacyjne, przejdź do rozwiązania zadania:

- a) Zakoduj labirynt jako macierz.
- b) Ustaw odpowiednie gene\_space i długości chromosomów.
- c) Ustaw sensowne parametry związane z populacją i rodzicami. Możesz przetestować różne konfiguracje.
- d) Ustaw sensowny procent na szansie na mutację. Najniższy możliwy, żeby nie wyskakiwał warning.
- e) Stwórz funkcję fitness. (Jest to najczęstszy punkt tego zadania: sporo programowania, pętle).
- f) Jeśli algorytm znajduje rozwiązanie, to wprowadź warunek stopu i zmierz średni czas z 10 uruchomień.