

Prezentacja algorytmu WOA

Tomasz Murawski

1. Prezentacja algorytmu

symulacja

Symulacja to funkcja przeprowadzająca pełną symulację pracy dla danego rozwiązania. Do uruchomienia potrzebuje dane pracy (tabele problemu), rozwiązanie (kolejność rozwiązywania zadań) oraz liczbę maszyn. Zwraca czas pracy oraz dane do Diagramu Gantta.

```
def symulacja(dane_pracy_org, rozwiazanie, L_maszyny):
    dane_pracy = numpy.copy(dane_pracy_org)
    rozw = numpy.copy(rozwiæzanie)
    maszyny = []
    czas = 0
    dzis = datetime.date.today()
    harmonogram = []
    for i in range(L_maszyny):
        maszyny.append([True, 0, (0,0)])

    hamuj = True
    while(hamuj):
        for zad in rozw:
            x = zad[0]
            y = zad[1]
            if(dane_pracy[x][y][1] != 0):
```

harmo_plot

Funckja rysująca Diagram Gantta przy pomocy zewnętrznej biblioteki [plotly](#). Wynik wyświetla się w przeglądarce i jest interaktywny.

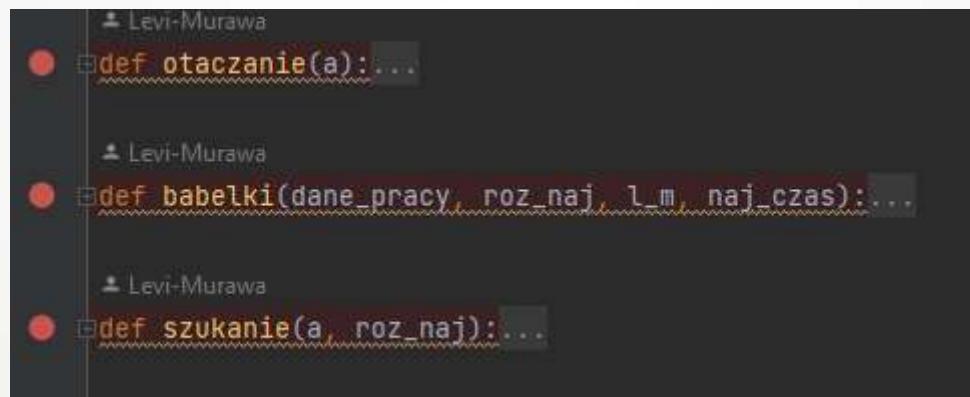
```
def harmo_plot(harmo):
    df = pd.DataFrame(harmo)
    fig = px.timeline(df, x_start="Start", x_end="Finish", y="Task", color="Praca")
    fig.update_yaxes(autorange="reversed") # otherwise tasks are listed from the bottom up
    fig.show()
```

harmo_plot



Otaczanie, babelki i szukanie

Te trzy funkcje pomocnicze są wykorzystywane przez każdego wieloryba do aktualizacji pozycji.



```
Levi-Murawa
def otaczanie(a):...
Levi-Murawa
def babelki(dane_pracy, roz_naj, l_m, naj_czas):...
Levi-Murawa
def szukanie(a, roz_naj):...
```

woa

Woa to połączenie wszystkich funkcji umożliwiająca realizację Optymalizacji Wieloryba dla JobShop problem

```
Levi-Murawa
def woa(dane, rozw, l_maszyn, l_iteracji):
    naj_rozw = rozw
    naj_czas, naj_ha = symulacja(dane, rozw, l_maszyn)

    for i in range(l_iteracji):
        a = 2 * ((l_iteracji - i)/l_iteracji)
        p = random.random()
        if(p>0.5):
            rozw_t = babelki(dane, naj_rozw, l_maszyn, naj_czas)
            czas, ha = symulacja(dane, rozw_t, l_maszyn)
            if(czas < naj_czas):
                naj_czas = czas
                naj_ha = ha
```

Przykładowe dane

Biblioteka zawiera również cztery przykładowe tablice z danymi, od tablicy 1 na 2 po tablicę 10 na 10.

```
jobs_data4 = [
    [[2, 44], [3, 5], [5, 58], [4, 97], [0, 9], [7, 84], [8, 77], [9, 96], [1, 58], [6, 89]],
    [[4, 15], [7, 31], [1, 87], [8, 57], [0, 77], [3, 85], [2, 81], [5, 39], [9, 73], [6, 21]],
    [[9, 82], [6, 22], [4, 10], [3, 70], [1, 49], [0, 40], [8, 34], [2, 48], [7, 80], [5, 71]],
    [[1, 91], [2, 17], [7, 62], [5, 75], [8, 47], [4, 11], [3, 7], [6, 72], [9, 35], [0, 55]],
    [[6, 71], [1, 90], [3, 75], [0, 64], [2, 94], [8, 15], [4, 12], [7, 67], [9, 20], [5, 50]],
    [[7, 70], [5, 93], [8, 77], [2, 29], [4, 58], [6, 93], [3, 68], [1, 57], [9, 7], [0, 52]],
    [[6, 87], [1, 63], [4, 26], [5, 6], [2, 82], [3, 27], [7, 56], [8, 48], [9, 36], [0, 95]],
    [[0, 36], [5, 15], [8, 41], [9, 78], [3, 76], [6, 84], [4, 30], [7, 76], [2, 36], [1, 8]],
    [[5, 88], [2, 81], [3, 13], [6, 82], [4, 54], [7, 15], [8, 29], [9, 40], [1, 78], [0, 75]],
    [[9, 88], [4, 54], [6, 64], [7, 32], [0, 52], [2, 6], [8, 54], [5, 82], [3, 6], [1, 26]]]
```

2. Prezentacja algorytmu

Plik cwiczenie.py

Plik zawiera wytłumaczenie działania biblioteki oraz dwa badania mające ustalić efektywność implementacji algorytmu

```
Podstawy biblioteki:
```

```
Najmniejszy znaleziony czas: 12
```

```
Wektor rozwiązań: [[0, 0], [1, 2], [1, 1], [2, 0], [0, 1], [2, 2], [2, 1], [0, 2], [1, 0]]
```

```
Rysowanie Diagramu Gantta
```

```
Badanie skuteczności w zależności od ilości testów:
```

```
Czas rozwiązania dla pojedynczego przebiegu: 1046
```

```
Czas rozwiązania dla dziesięciu przebiegów: 981
```

```
Czas rozwiązania dla stu przebiegów: 897
```

```
Badanie stabilności wyników dla różnych ilości przebiegów:
```

```
Process finished with exit code 0
```

2.1. podstawy biblioteki

W pierwszym paragrafie zostały przedstawione podstawowe działania biblioteki. Wykonany zostaje jedna optymalizacja i wyświetcone zostają wyniki.

```
if __name__ == '__main__':
    ##----- podstawy korzystania z biblioteki
    print('\n', "Podstawy biblioteki:")
    przyklad1 = WOA_TM.jobs_data2
    liczenie_maszyn = 1 + max(task[0] for job in przyklad1 for task in job)
    rozw = WOA_TM.pod_rozw(przyklad1)
    min_czas, naj_har, naj_roz = WOA_TM.woa(przyklad1, rozw, liczenie_maszyn, 10)
    print("Najmniejszy znaleziony czas:", min_czas)
    print("Wektor rozwiązania:", naj_roz)
    print("Rysowanie Diagramu Gantta ")
    WOA_TM.harmo_plot(naj_har)
```

2.1. podstawy biblioteki

Otrzymywana odpowiedź w terminalu:

```
Podstawy biblioteki:  
Najmniejszy znaleziony czas: 12  
Wektor rozwiazania: [[0, 0], [1, 2], [1, 1], [2, 0], [0, 1], [2, 2], [2, 1], [0, 2], [1, 0]]  
Rysowanie Diagramu Gantta
```

2.2. badanie wpływu liczby iteracji

Pierwszym sprawdzeniem biblioteki jest wpływ liczby iteracji na jakość znajdowanych rozwiązań. W tym celu uruchamiamy algorytm dla najbliższego rozwiązania przy ustawieniu 1, 10 oraz 100 cykli.

```
if __name__ == '__main__':
    ##----- podstawy korzystania z biblioteki
    print('\n', "Podstawy biblioteki:")
    przyklad1 = WOA_TM.jobs_data2
    liczenie_maszyn = 1 + max(task[0] for job in przyklad1 for task in job)
    rozw = WOA_TM.pod_rozw(przyklad1)
    min_czas, naj_har, naj_roz = WOA_TM.woa(przyklad1, rozw, liczenie_maszyn, 10)
    print("Najmniejszy znaleziony czas:", min_czas)
    print("Wektor rozwiązania:", naj_roz)
    print("Rysowanie Diagramu Gantta ")
    WOA_TM.harmo_plot(naj_har)
```

2.2. badanie wpływu liczby iteracji

Otrzymane rezultaty wykazują znaczną poprawę przy wzroście liczby iteracji (choć nie jest to zawsze prawda).
Poniżej przykładowe dane:

```
Badanie skuteczności w zależności od ilości testów:  
Czas rozwiązania dla pojedyńczego przebiegu: 1046  
Czas rozwiązania dla dziesięciu przebiegów: 981  
Czas rozwiązania dla stu przebiegów: 897
```

2.3. badanie stabilności wyników i cech algorytmu

Ostatni fragment kodu przedstawia bardziej rozbudowane badanie. Staramy się w nim wykazać cechy algorytmu Wieloryba. Jak każdy algorytm losowy powinien on wpadać w ekstrema lokalne przy za małej liczbie iteracji, a wraz z ich wzrostem wyniki powinny być coraz bardziej zbliżone. Otrzymane wyniki wyświetlimy przy pomocy biblioteki `matplotlib`.

2.3. badanie stabilności wyników i cech algorytmu

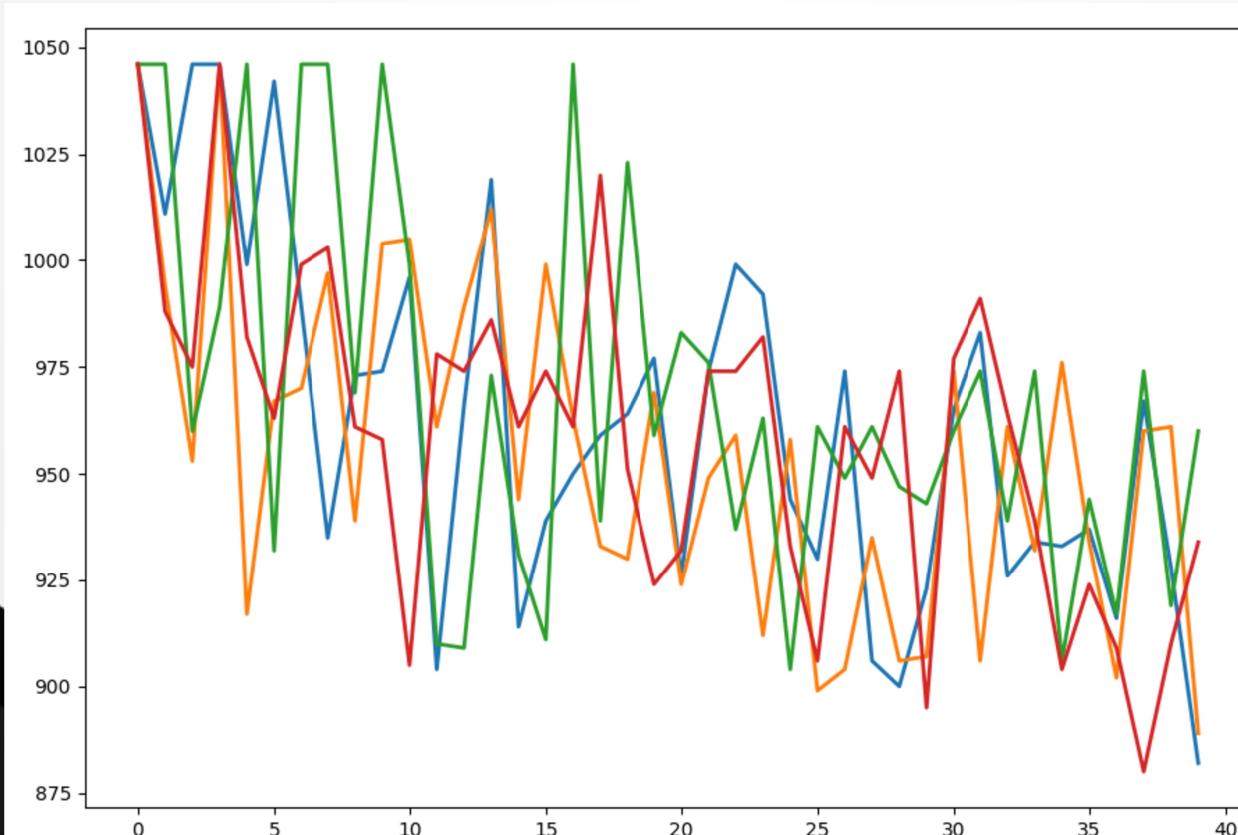
```
##----- Badanie stabilności wyników dla różnych ilości przebiegów
print('\n', "Badanie stabilności wyników dla różnych ilości przebiegów:")
przyklad3 = WOA_TM.jobs_data4
liczenie_maszyn = 1 + max(task[0] for job in przyklad3 for task in job)
rozw = WOA_TM.pod_rozw(przyklad3)
wektor_z_czasami = []
wektor_sredni = []
zakres_badania = range(30)
for i in zakres_badania:
    wynik1 = WOA_TM.woa(przyklad3, rozw, liczenie_maszyn, i)
    wynik2 = WOA_TM.woa(przyklad3, rozw, liczenie_maszyn, i)
    wynik3 = WOA_TM.woa(przyklad3, rozw, liczenie_maszyn, i)
    wynik4 = WOA_TM.woa(przyklad3, rozw, liczenie_maszyn, i)
    wynik_s = (wynik1[0]+wynik2[0]+wynik3[0]+wynik4[0])/4
    wektor_z_czasami.append([wynik1[0], wynik2[0], wynik3[0], wynik4[0]])
    wektor_sredni.append(wynik_s)

fig, ax = plt.subplots()
ax.plot(zakres_badania, wektor_z_czasami, linewidth=2.0)
plt.show()

fig, ax = plt.subplots()
ax.plot(zakres_badania, wektor_sredni, linewidth=2.0)
plt.show()
```

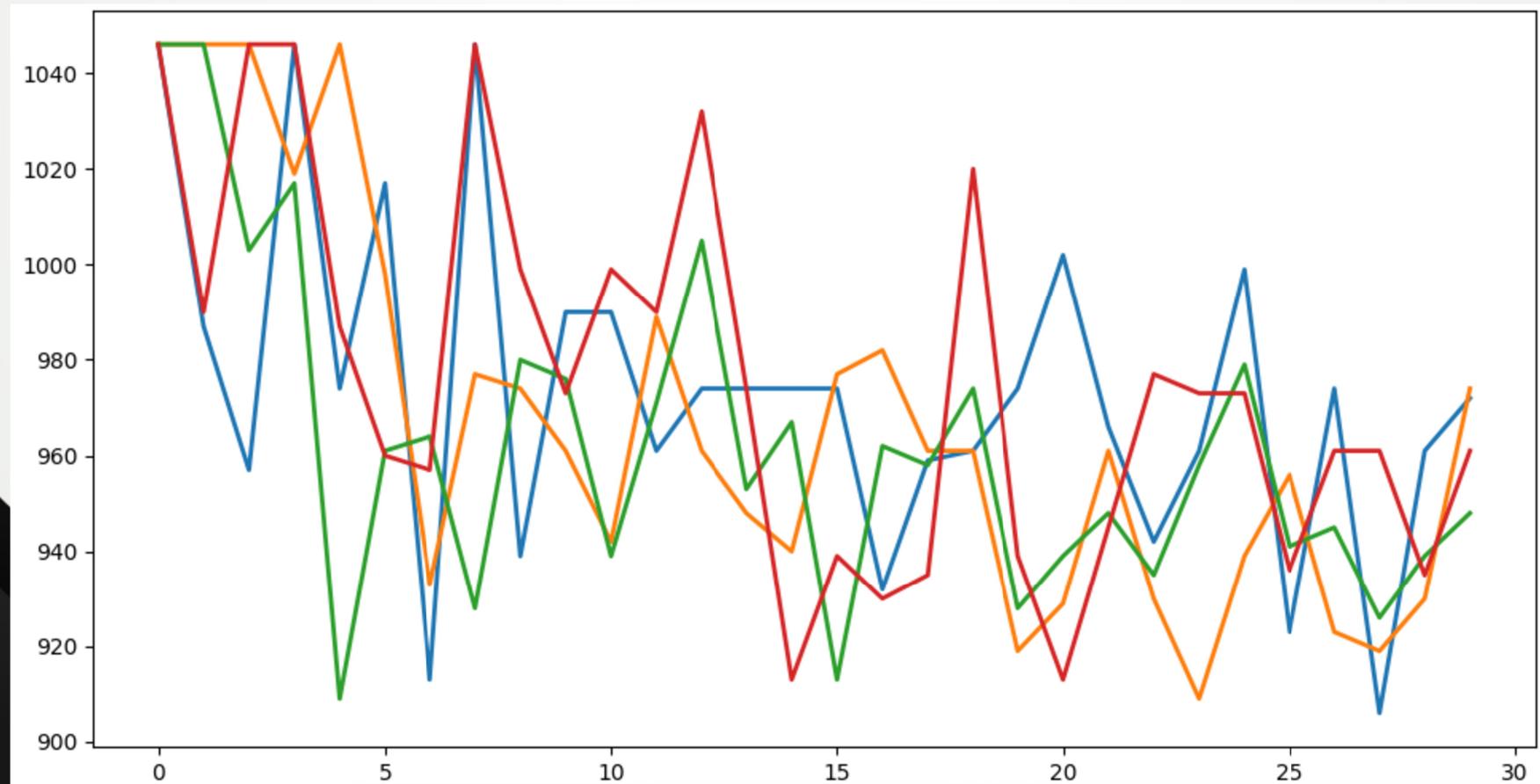
2.3. badanie stabilności wyników i cech algorytmu

Na początku otrzymujemy wykres z kilkoma kolorowymi liniami. Każda z tych linii to wyniki innego algorytmu. Widzimy, iż ze wzrostem liczby cykli linie zbliżają się do siebie, co zgadza się z naszymi przewidywaniami



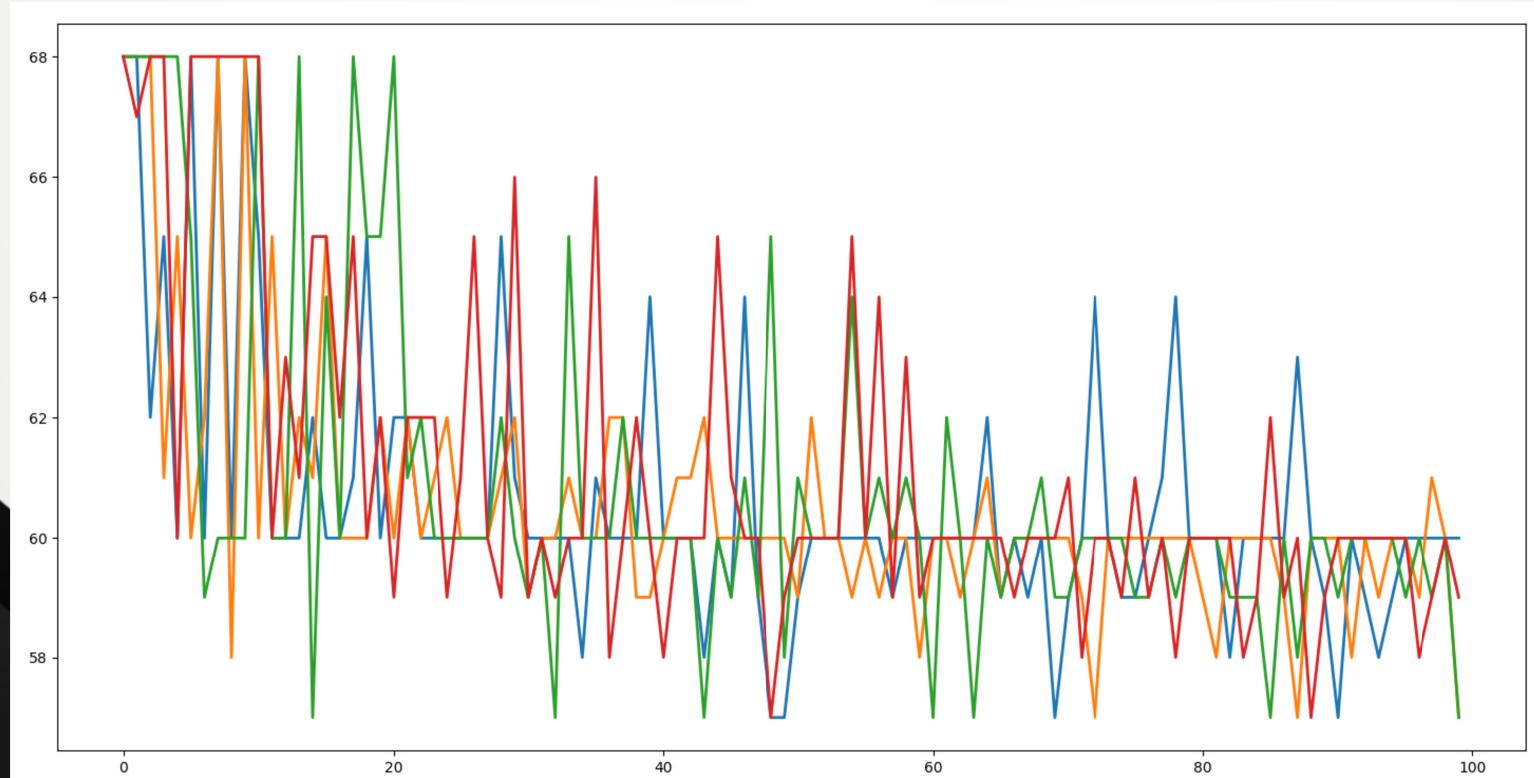
2.3. badanie stabilności wyników i cech algorytmu

Widzimy także wypłaszczenia, szczególnie na początku, oznacza to wpadanie w ekstrema lokalne, co również pasuje do naszych rozważań.



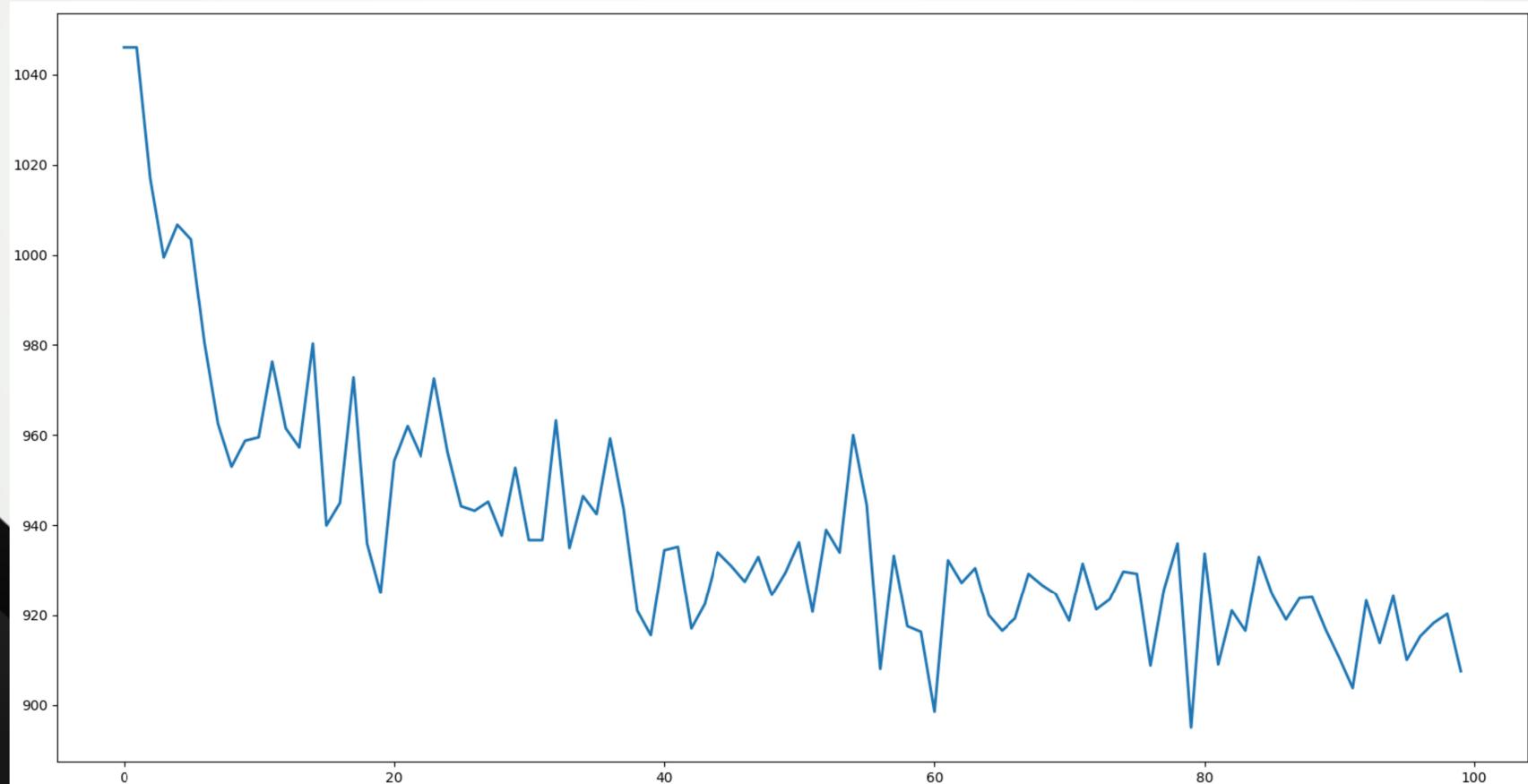
2.3. badanie stabilności wyników i cech algorytmu

Przy prostych problemach następuje wypłaszczenie przy zbyt dużej liczbie cykli. Możemy na tej podstawie wyznaczyć optymalną liczbę cykli dla danej wielkości tabeli



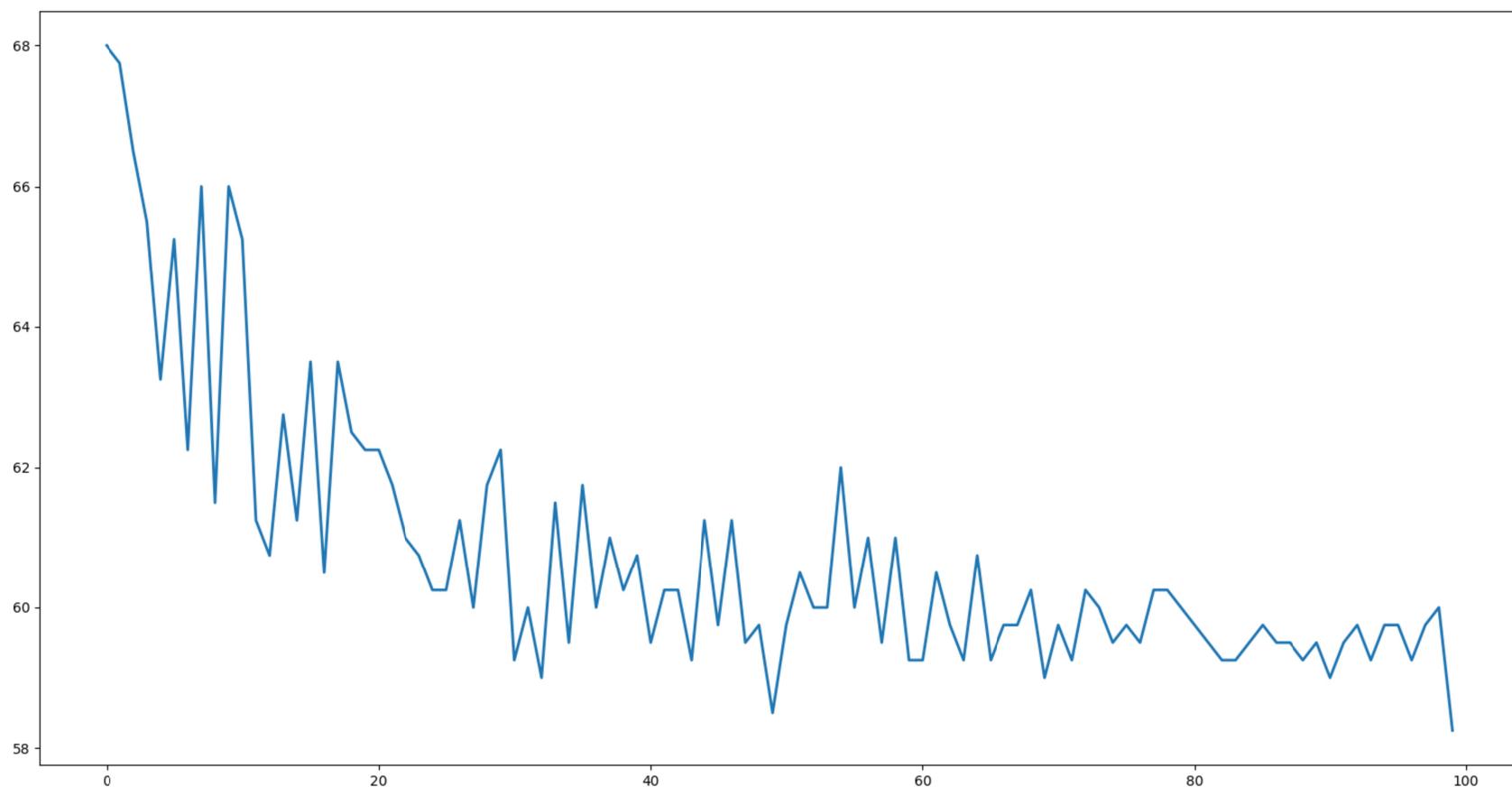
2.3. badanie stabilności wyników i cech algorytmu

Następnie wyświetla się wykres z wartościami uśrednionymi. Dzięki temu widzimy dokładniej jak stabilnie pracuje algorytm.

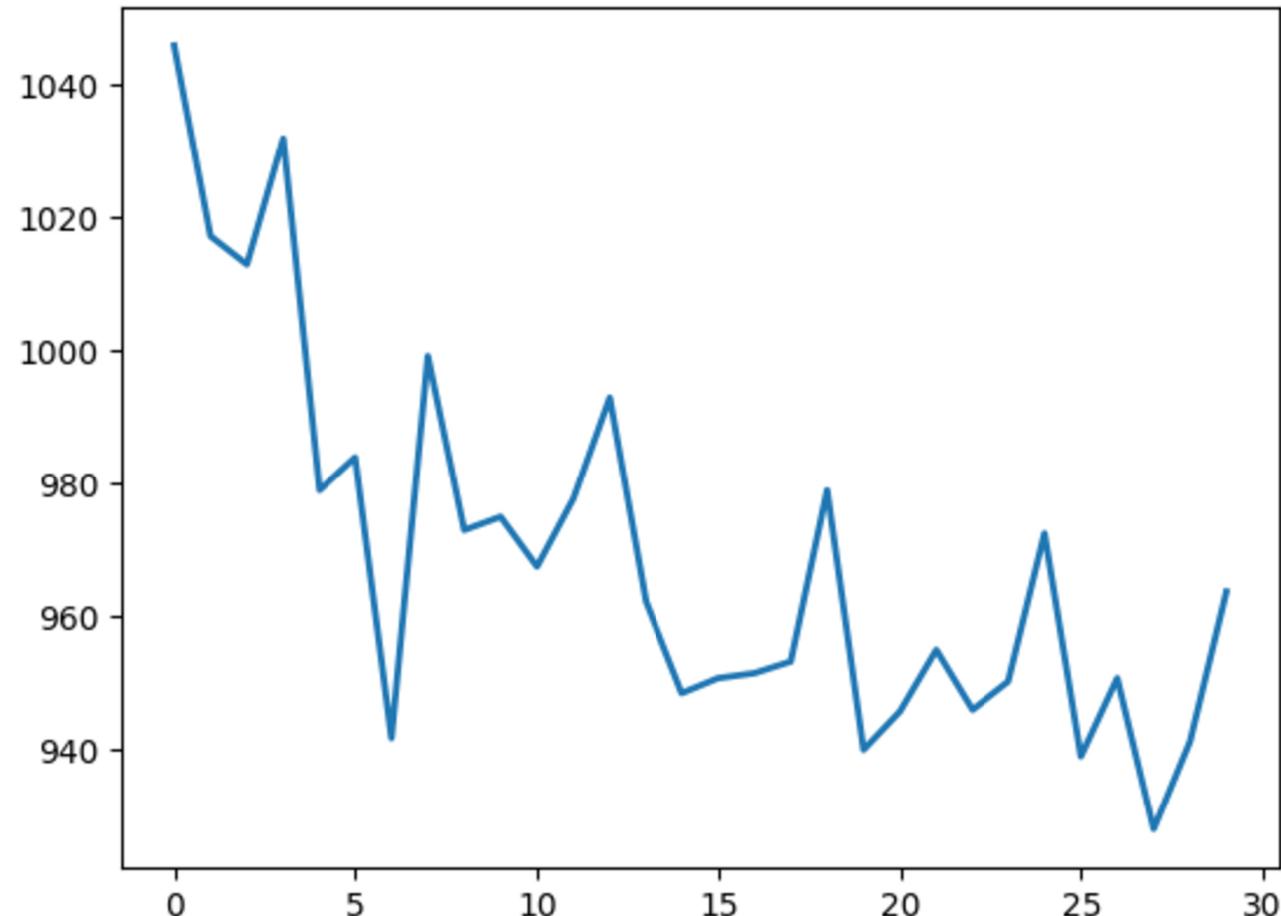


2.3. badanie stabilności wyników i cech algorytmu

Widzimy te same zjawiska co poprzednio, jednak teraz precyzyjniej możemy wyznaczyć jak zachowuje się algorytm.



2.3. badanie stabilności wyników i cech algorytmu



3. Podsumowanie

3. podsumowanie

Dzięki implementacji Pythona możemy swobodnie analizować skuteczność algorytmu jak również wykorzystywać go do innych celów.

Dziękuję za uwagę