Istnienie i stabilność klastra cząstek

Witold Gliwa

January 2023

Spis treści

1 Wprowadzenie

1.1 Treść zadania

Za pomocą algorytmu genetycznego i algorytmu inteligencji roju znaleźć minimum energetyczne układu cząstek (klastru) za pomocą sum potencjałów Morse'a wyrażonych wzorem:

$$V_m = \sum_{i<j} e^{p_0 \cdot (1-r_{ij})} (e^{p_0 \cdot (1-r_{ij})} - 2) \tag{1}$$

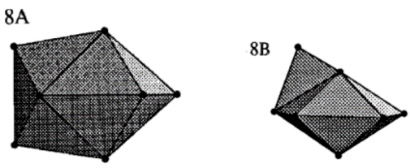
gdzie:

- r_{ij} to odległość w przestrzeni trójwymiarowej pomiędzy cząstkami i i j
- p_0 to siła przyciągania się cząsteczek w klastrze dla danego materiału

Minimum oznacza potencjalnie stabilne ułożenie cząstek w klastrze. Część już znalezionych wartości minimów w tabeli:

strength / size	3	6	10	14
5	-9.299500	-9.044930	-9.003565	-9.000283
6	-13.544229	-12.487810	-12.094943	-12.018170
7	-17.552961	-16.207580	-15.956512	-15.883113
8	-22.042901	-19.161862	-18.275118	-18.076248
9	-26.778449	-22.330837	-21.213531	-21.037957
10	-31.519768	-25.503904	-24.204958	-24.031994
11	-37.930817	-28.795153	-23.666072	NaN

Przykładowe przedstawienie graficzne 2 róznych minimów dla 8 czastek i $p_0 = 6$:



1.2 Zakres badań

Oba algorytmy zostały uruchomione na klastrach wielkości 5 - 11 i p_0 równego odpowiednio 3, 6, 10 i 14

2 Szukanie minimów

Oba algorytmy zostały uruchomione 10 razy dla każdego przypadku w celu unormowania średniego czasu pracy na populacji i ilości generacji równej 150. Wzorując się na poprzednich wynikach zakres współrzędnych punktów w przestrzeni został ograniczony do zakresu (-2.0, 2.0).
Do obliczania potencjału Morse'a dla danych współrzędnych zostały w obu algorytmach wykorzystane funkcje:

```
def calc3d(x):
    return np.linalg.norm(np.array([x[0], x[1], x[2]])
                          - np.array([x[3], x[4], x[5]]))

def fitness_func(x, solution_idx=0):
    x = np.array_split(x, len(x) / 3)
    sum = 0
    for i in range(0, len(x)):
        for j in range(i + 1, len(x)):
            sum += pow(e, strength *
                       (1 - calc3d(np.append(x[i], x[j])))) \
                   * (pow(e, strength *
                          (1 - calc3d(np.append(x[i], x[j])))) - 2)
    return +/-sum
```

Przy wywołaniu funkcji `fitness_func` do środka jest wrzucany dany chromosom w postaci listy współrzędnych. Lista jest podzielona na kawałki wielkości 3 zawierające x, y i z punktów. Następnie w pętli zostaje obliczony potencjał Morse'a dla każdej pary punktów. Funkcja pomocnicza `calc3d` służy do znalezienia odległości pomiędzy punktami. Każdy z potencjałów zostaje dodany do sumy potencjału całego klastra. Na koniec suma zostaje zwrócona jako wartość fitness, odpowiednio odwrócona w przypadku algorytmu genetycznego.

Cały algorytm zbierał dane z 10 uruchomień i zwracał listę: wielkości klastra, siły przyciągania, najmniejszego minimum, średniej czasów wykonania, czasu minimalnego, czasu maksymalnego i listy współrzędnych punktów najlepszego wyniku.

2.1 Algorytm genetyczny

Algorytm genetyczny został wykonany w 2 wersjach:

2.1.1 Bez specjalnych zasad krzyżowania

```
ga_instance = pygad.GA(gene_space=gene_space,
                       num_generations=150,
                       num_parents_mating=int(sol_per_pop / 2),
                       fitness_func=fitness_func,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       parent_selection_type="rws",
                       keep_parents=int(sol_per_pop / 10),
                       crossover_type="scattered",
                       mutation_type="swap",
                       mutation_percent_genes=20,
                       crossover_probability=0.5)
                       )
```

Z zasadami

- Ilość krzyżujących się rodziców - połowa populacji (75)
- Zasadą wybierania rodziców - ruletka
- Ilością rodziców do zatrzymania do kolejnej generacji - 10% (15)
- Wymiennym typem mutacji
- Prawdopodobieństwem mutacji genu 20%
- Prawdopodobieństwem krzyżowania się rodziców 50%

2.1.2 Z dostosowanymi zasadami krzyżowania

```
ga_instance = pygad.GA(gene_space=gene_space,
                       num_generations=150,
                       num_parents_mating=int(sol_per_pop / 2),
                       fitness_func=fitness_func,
                       sol_per_pop=sol_per_pop,
                       num_genes=num_genes,
                       parent_selection_type="tournament",
                       keep_parents=int(sol_per_pop / 10),
                       crossover_type=crossover,
                       mutation_type="random",
                       mutation_percent_genes=20,
                       )
```

Z zasadami

- Ilość krzyżujących się rodziców - połowa populacji (75)
- Zasadą wybierania rodziców - turniej

- Ilością rodziców do zatrzymania do kolejnej generacji - 10% (15)
- Losowym typem mutacji
- Prawdopodobieństwem mutacji genu 20%

```
def crossover(parents, offspring_size, pygad):
    offspring = []
    idx = 0
    while len(offspring) != offspring_size[0]:
        parent1 = parents[idx % parents.shape[0], :].copy()
        parent2 = parents[(idx + 1) % parents.shape[0], :].copy()
        random_split_point = 3 * round(np.random.choice(range(
    offspring_size[1])) / 3)
        parent1[random_split_point:] = parent2[random_split_point:]
        offspring.append(parent1)
        idx += 1
    return np.array(offspring)
```

Powyższa zasada `crossover` krzyżowania się niewiele odstaje od krzyżowania się w 1 punkcie z dodaną zasada że punkt krzyżowania nie może być w innym miejscu niż wielokrotność 3. Zasada została dodana z powodu typu organizacji danych - wymieniane geny pomiędzy rodzicami będą tutaj zawsze pełnymi punktami a nie ewentualnie jedna z współrzędnych punktu.

2.2 Inteligencja roju

Algorytm inteligencji roju został uruchomiony w 2 wersjach:

- GlobalBest - cząsteczki są przyciągane do najlepiej sprawującej się cząsteczki z calego roju (topologia gwiazdy)
  Z ustawieniami: $(\phi_1 = 0,5, \phi_2 = 0,3, \omega = 0.9)$
- LocalBest - cząsteczki są przyciągane do swoich najbliższych sąsiadów (topologia pierścienia)
  Z ustawieniami: $(\phi_1 = 0,5, \phi_2 = 0,3, \omega = 0.9, k = 3, p = 2)$

Gdzie:

- ϕ_1 - współczynnik dążenia do najlepszego indywidualnego rozwiązania
- ϕ_2 - współczynnik dążenia do najlepszego lokalnego rozwiązania
- ω - współczynnik bezwładności
- k - ilość sąsiadów do rozpatrzenia
- p - rodzaj znajdowania sąsiadów, w tym przypadku suma bezwzględnych różnic wartości

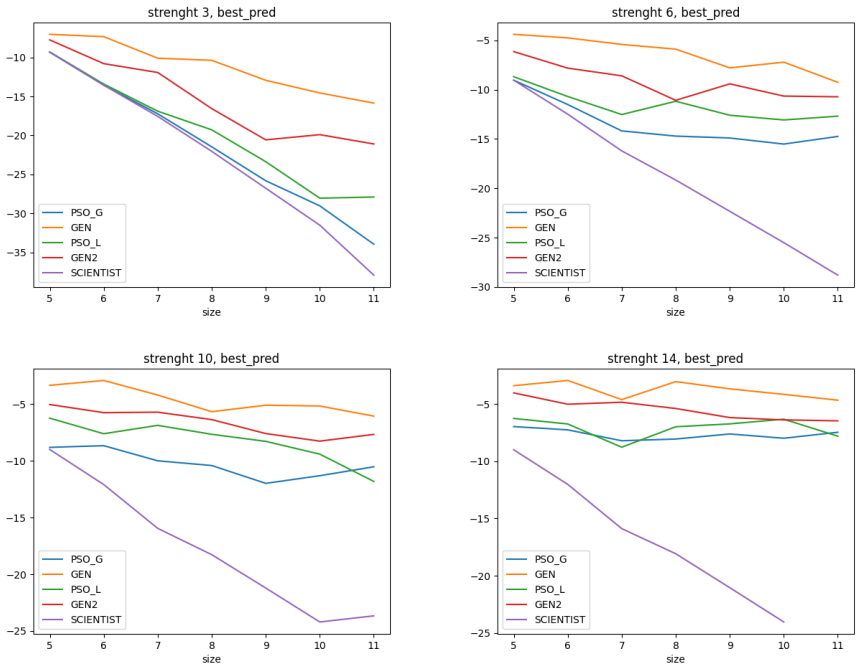
3 Wyniki

3.1 Wykresy

- PSO_G - GlobalBest PSO
- PSO_L - LocalBest PSO
- GEN - AlgorytmGenetyczny
- SCIENTIST - Wyniki naukowców?
- GEN2 - Algorytm genetyczny z niestandardową regułą krzyżowania

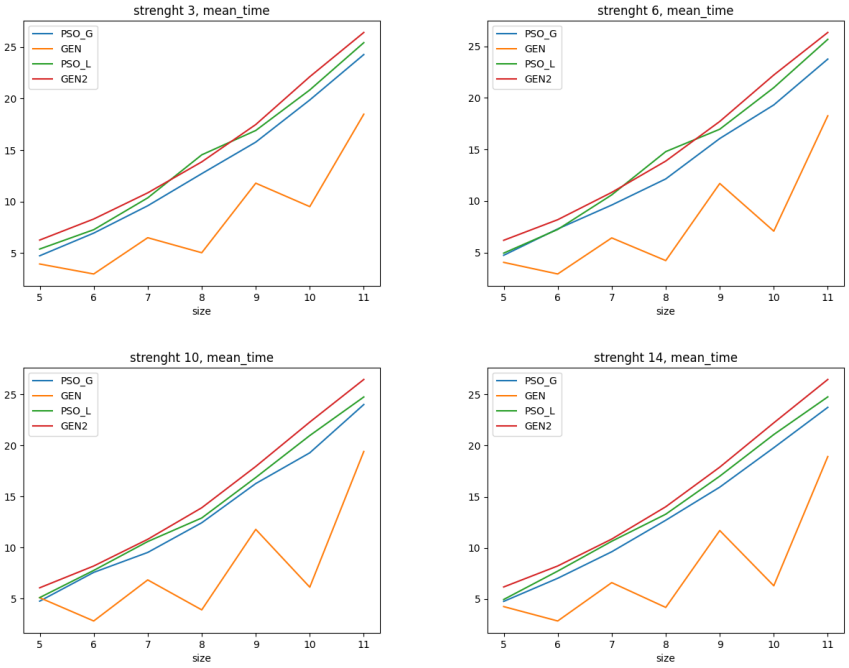
3.1.1 Najlepszy wynik

Poniżej zostały przedstawione 4 wykresy dla każdej z sił przyciągania i ich odpowiadające wyniki



3.1.2 Średni czas

Poniżej zostały przedstawione 4 wykresy dla każdej z sił przyciągania i ich odpowiadające czasy wykonania



3.2 Tabele

3.2.1 Algorytm Genetyczny

strength / size	3	6	10	14
5	-7.019020	-4.397432	-3.351963	-3.407595
6	-7.320105	-4.757172	-2.926122	-2.953002
7	-10.086451	-5.423284	-4.209104	-4.621444
8	-10.358348	-5.900441	-5.677644	-3.053355
9	-12.922751	-7.795724	-5.106053	-3.683217
10	-14.547740	-7.216919	-5.175406	-4.165866
11	-15.846790	-9.259666	-6.067997	-4.673248

(a) Najlepsze wyniki

strength / size	3	6	10	14
5	-2.280480	-4.647498	-5.651602	-5.592688
6	-6.224124	-7.730638	-9.168821	-9.065168
7	-7.466510	-10.784296	-11.747408	-11.261669
8	-11.684553	-13.261421	-12.597474	-15.022893
9	-13.855698	-14.535113	-16.107478	-17.354740
10	-16.972028	-18.286985	-19.029552	-19.866128
11	-22.084027	-19.535487	-17.598075	NaN

(b) Róznica z naukowcami

3.2.2 PSO GlobalBest

strength / size	3	6	10	14
5	-9.299497	-9.044771	-8.820133	-6.978604
6	-13.544056	-11.507746	-8.671867	-7.261647
7	-17.224570	-14.185112	-10.004062	-8.213925
8	-21.463751	-14.710338	-10.419723	-8.064092
9	-25.818440	-14.901252	-11.990945	-7.620893
10	-29.038532	-15.517282	-11.317686	-7.995260
11	-33.959362	-14.745030	-10.531785	-7.473845

(a) Najlepsze wyniki

strength / size	3	6	10	14
5	-0.000003	-0.000159	-0.183432	-2.021679
6	-0.000173	-0.980064	-3.423076	-4.756523
7	-0.328391	-2.022468	-5.952450	-7.669188
8	-0.579150	-4.451524	-7.855395	-10.012156
9	-0.960009	-7.429585	-9.222586	-13.417064
10	-2.481236	-9.986622	-12.887272	-16.036734
11	-3.971455	-14.050123	-13.134287	NaN

(b) Róznica z naukowcami

3.2.3 PSO LocalBest

strength / size	3	6	10	14
5	-9.281168	-8.694257	-6.247303	-6.265649
6	-13.401194	-10.695470	-7.618372	-6.748812
7	-16.888065	-12.523764	-6.881275	-8.779186
8	-19.279571	-11.180575	-7.668592	-6.993494
9	-23.372072	-12.593219	-8.290908	-6.741712
10	-28.052590	-13.074072	-9.416607	-6.332480
11	-27.906037	-12.689516	-11.820280	-7.814554

(a) Najlepsze wyniki

strength / size	3	6	10	14
5	-0.018332	-0.350673	-2.756262	-2.734634
6	-0.143035	-1.792340	-4.476571	-5.269358
7	-0.664896	-3.683816	-9.075237	-7.103927
8	-2.763330	-7.981287	-10.606526	-11.082754
9	-3.406377	-9.737618	-12.922623	-14.296245
10	-3.467178	-12.429832	-14.788351	-17.699514
11	-10.024780	-16.105637	-11.845792	NaN

(b) Róznica z naukowcami

3.2.4 Genetyczny niestandardowy

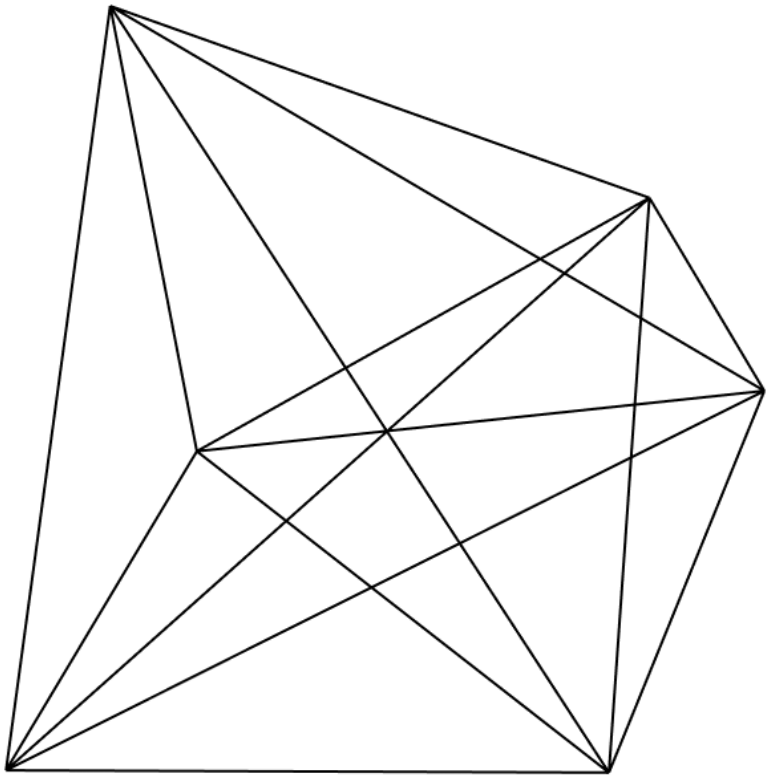
strength / size	3	6	10	14
5	-7.734039	-6.148184	-5.044700	-4.037571
6	-10.782255	-7.821588	-5.761345	-5.025583
7	-11.920447	-8.601497	-5.719240	-4.853453
8	-16.558138	-11.085490	-6.373818	-5.398010
9	-20.567594	-9.416199	-7.604074	-6.190765
10	-19.898133	-10.654868	-8.269752	-6.393808
11	-21.094210	-10.727311	-7.678765	-6.476099

(a) Najlepsze wyniki

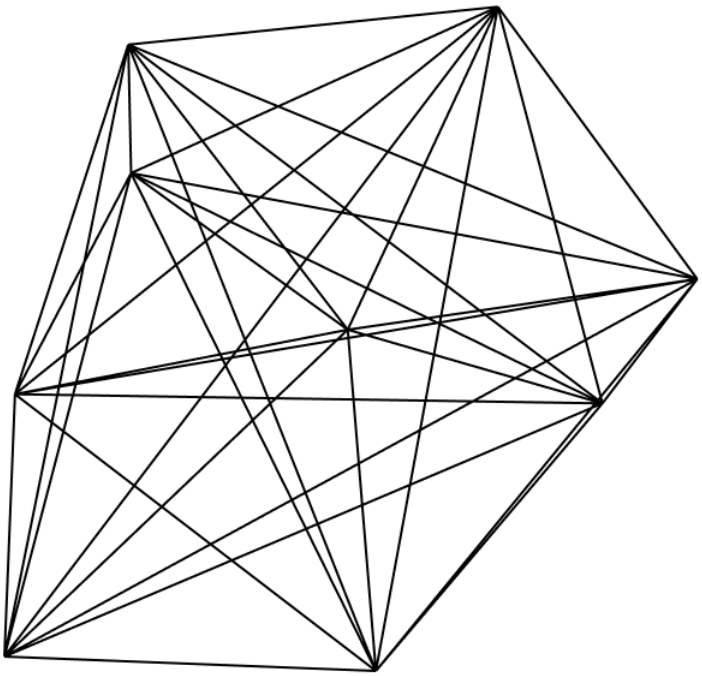
strength / size	3	6	10	14
5	-1.565461	-2.896746	-3.958865	-4.962712
6	-2.761974	-4.666222	-6.333598	-6.992587
7	-5.632514	-7.606083	-10.237272	-11.029660
8	-5.484763	-8.076372	-11.901300	-12.678238
9	-6.210855	-12.914638	-13.609457	-14.847192
10	-11.621635	-14.849036	-15.935206	-17.638186
11	-16.836607	-18.067842	-15.987307	NaN

(b) Róznica z naukowcami

3.3 Obrazy



Rysunek 7: Przykładowy wynik Global Best PSO dla wielkości 6 i siły 3



Rysunek 8: Przykładowy wynik Global Best PSO dla wielkości 9 i siły 3

Z powodu konieczności wykonania syzyfowej pracy potrzebnej do umieszczenia animacji w pliku PDF wszystkie klastry o sile 3 są animowane w plikach GIF na githubie.

4 Wnioski

Analizując dane z wykresów i tabel wyraźnie widać ze algorytm inteligencji roju radzi sobie lepiej z szukaniem minima od algorytmu genetycznego, osiągając niemalże te same wyniki co naukowcy dla niskiej wartości p_0, jednocześnie osiągając niewiele gorsze wynik czasowo niż algorytm genetyczny. Wyjątkiem jest pierwszy algorytm genetyczny, który działa znacznie szybciej, co było testowane na 2 próbach, więc nie jest to wina czynników zewnętrznych. Jest to prawdopodobnie spowodowane ustawianiami algorytmu, prawdopodobnie system wybierania rodziców ma niewielką złożoność czasową. Nadal algorytm ten osiąga zbyt słabe wyniki żeby miało sens badać go dalej.

Wszystkie algorytmy osiągają znacznie gorsze wyniki w przypadku wyższych wartości p_0. Jest to prawdopodobnie spowodowane składnią równania Morse'a

i znalezienie dobrych minimów wymagałoby większej ilości pokoleń i populacji. Istotą tego dokumentu było jednak badanie różnych typów algorytmów a nie pobicie wyników naukowców. Cytując wpis ze strony o klastrach:

> The location of all the global minimum at ρ_0=3, 6, 10 and 14 for clusters with up to 80 atoms would be a significant achievement and one which no unbiased global optimization algorithm has yet managed.

Nie pomaga fakt że w badaniach zostały użyte zwykłe Pythonowe floaty więc ekstremalnie dokładne wyniki i tak nie były do osiągnięcia. *Osobiście* uważam że przy większej dokładności i mocy obliczeniowej jest możliwe pobicie tych wyników algorytmem PSO ale *osobiście* nie wiem jaką miałoby to wartość.

5 Źródła i użyte paczki

- Dokument z projektami
- Strona o klastrach
- The effect of the range of the potential on the structures of clusters
- Algorytm Genetyczny - paczka PyGAD
- Algorytm Inteligencji Roju - paczka PySwarms
- Obliczenia, zbieranie danych - Numpy, Pandas
- Generowanie wykresów i tabel - Matplotlib
- Generowanie klastrów w 3D - Plotly