Assegnamento 6 : Evolutionary Computation 2

January 24, 2022

Nicolò Toscani

1 Esercizio 3

Posizionare N regine sopra una scacchiera NxN in maniera che nessuna di essa possa catturare l'altra.
Procediamo con due distinte soluzioni.

1.1 Smart rapresentation

Questa soluzione utilizza come rappresenntazione una permutazione di valori da 0 a N-1.
Questa rappresentazione deriva dal fatto che due regine non possono essere allineate orizzontalmente o verticalmente, quindi ci deve essere una regina per riga e una per colonna.
Questa solizone puó essere codificata come una sequenza di numeri interi corrispondenti alla posizione di una regina in ogni riga.

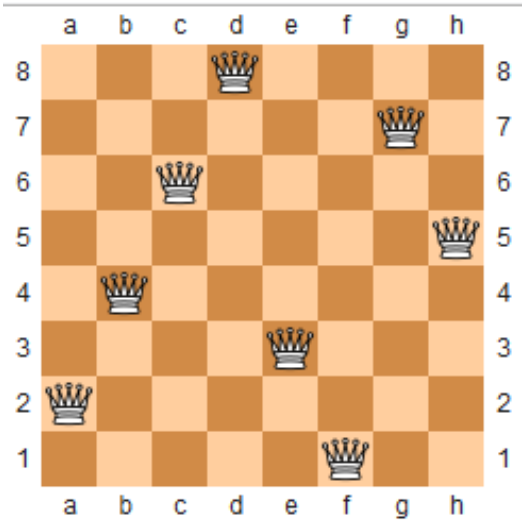

Figure 1: Possibile soluzione

Modificando i valori di N otteniamo diversi valori di *success rate*

```python
def evalNQueens(individual):
    """Evaluation function for the n-queens problem.
    The problem is to determine a configuration of n queens
    on a nxn chessboard such that no queen can be taken by
    one another. In this version, each queens is assigned
    to one column, and only one queen can be on each line.
    The evaluation function therefore only counts the number
    of conflicts along the diagonals.
    """

    """
    Il problema è quello di determinare la configurazione di n (8) regine
    in una scacchiera n x n (8x8) in modo che ogni regina non viene attaccata da un altra.
    Ogni regina viene assegnata a una colonna e una sola regina può essere assegnata ad una riga.
    Vengono contati il numero dei conflitti lungo la diagonale
    """
    size = len(individual) # size = 8

    # Conteggio collisioni lungo la diagonale in entrambi i lati della regina
    left_diagonal = [0] * (2*size-1)
    right_diagonal = [0] * (2*size-1)

    #Sum the number of queens on each diagonal:
    for i in range(size):
        left_diagonal[i+individual[i]] += 1
        right_diagonal[size-1-i+individual[i]] += 1

    #Count the number of conflicts on each diagonal
    sum_ = 0
    for i in range(2*size-1):
        if left_diagonal[i] > 1:
            sum_ += left_diagonal[i] - 1
        if right_diagonal[i] > 1:
            sum_ += right_diagonal[i] - 1
    return sum_,


creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", list, fitness=creator.FitnessMin)

#Since there is only one queen per line,
#individual are represented by a permutation
toolbox = base.Toolbox()
toolbox.register("permutation", random.sample, range(NB_QUEENS), NB_QUEENS)

#Structure initializers
#An individual is a list that represents the position of each queen.
#Only the line is stored, the column is the index of the number in the list.
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.permutation)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)

toolbox.register("evaluate", evalNQueens)
toolbox.register("mate", tools.cxPartialyMatched)
toolbox.register("mutate", tools.mutShuffleIndexes, indpb=2.0/NB_QUEENS)
toolbox.register("select", tools.selTournament, tournsize=3)

def main(seed=0):
    random.seed(seed)

    pop = toolbox.population(n=300)
    hof = tools.HallOfFame(1)
    stats = tools.Statistics(lambda ind: ind.fitness.values)
    stats.register("Avg", numpy.mean)
    stats.register("Std", numpy.std)
    stats.register("Min", numpy.min)
    stats.register("Max", numpy.max)

    algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=100, stats=stats,
                        halloffame=hof, verbose=True)

    return pop, stats, hof
```

Figure 2: n = 8

gen	nevals	Avg	Std	Min	Max	
0	300	3.94333	1.25437	1	9	
1	176	3.4	1.23288	0	7	
2	149	3.09667	1.20581	0	7	
3	181	2.98333	1.32025	0	7	
4	175	2.84	1.35931	0	7	
5	163	2.74	1.48965	0	7	
6	204	2.82	1.51468	0	7	
7	183	2.57667	1.48687	0	7	
8	170	2.29333	1.46764	0	6	
9	186	2.35333	1.51278	0	7	
10	176	2.21667	1.63597	0	8	
11	170	2	1.7282	0	8	
12	198	1.94	1.69009	0	6	
13	184	1.68667	1.72099	0	6	
14	174	1.3	1.65429	0	6	
15	173	0.953333	1.45756	0	6	
16	167	0.996667	1.61555	0	6	
17	177	0.82	1.54087	0	6	
18	182	0.666667	1.42439	0	7	
19	183	0.62	1.33502	0	5	
20	168	0.583333	1.40821	0	6	
21	180	0.603333	1.46492	0	8	
22	170	0.546667	1.29916	0	6	
23	163	0.456667	1.16681	0	6	
24	198	0.573333	1.25351	0	6	
25	189	0.523333	1.32519	0	8	
26	183	0.54	1.2813	0	6	
27	177	0.45	1.19199	0	6	
28	169	0.59	1.30967	0	6	
29	184	0.656667	1.39718	0	6	
30	189	0.62	1.41972	0	6	
31	160	0.533333	1.25255	0	5	
32	168	0.52	1.27917	0	6	
33	170	0.536667	1.29692	0	6	
34	175	0.51	1.32032	0	7	
35	202	0.63	1.42118	0	7	
36	167	0.64	1.4412	0	6	
37	168	0.626667	1.43781	0	7	
38	188	0.676667	1.50957	0	7	
39	169	0.686667	1.48161	0	6	
40	173	0.423333	1.0476	0	5	
41	183	0.67	1.44491	0	8	
42	173	0.64	1.37492	0	6	
43	192	0.626667	1.33939	0	7	
44	190	0.433333	1.14261	0	5	
45	179	0.656667	1.44872	0	7	
46	177	0.68	1.41336	0	7	
47	158	0.52	1.29471	0	6	
48	184	0.663333	1.41067	0	6	
49	173	0.64	1.42726	0	6	
50	183	0.58	1.27682	0	5	
51	176	0.613333	1.30019	0	6	
52	188	0.513333	1.18736	0	5	
53	161	0.426667	1.16531	0	6	
54	186	0.596667	1.36406	0	6	
55	175	0.67	1.43797	0	7	
56	179	0.626667	1.40735	0	6	
57	179	0.556667	1.35405	0	7	
58	187	0.59	1.32987	0	6	
59	161	0.526667	1.24738	0	6	
60	196	0.593333	1.32964	0	7	
61	186	0.573333	1.3849	0	6	
62	171	0.6	1.27279	0	6	
63	168	0.696667	1.47354	0	7	
64	183	0.523333	1.27389	0	6	
65	175	0.52	1.28177	0	9	
66	170	0.626667	1.39067	0	6	
67	179	0.73	1.51782	0	7	
68	180	0.496667	1.23423	0	6	
69	192	0.616667	1.29475	0	5	
70	163	0.55	1.3016	0	7	
71	179	0.596667	1.35178	0	6	
72	155	0.493333	1.26621	0	6	
73	201	0.723333	1.5341	0	7	
74	178	0.613333	1.37979	0	6	
75	177	0.563333	1.29846	0	8	
76	195	0.49	1.23958	0	5	
77	166	0.556667	1.32166	0	5	
78	190	0.566667	1.32372	0	6	
79	186	0.606667	1.38514	0	6	
80	177	0.603333	1.36601	0	7	
81	186	0.593333	1.31705	0	7	
82	168	0.58	1.27943	0	5	
83	174	0.593333	1.36185	0	6	
84	170	0.586667	1.3326	0	6	
85	161	0.586667	1.35984	0	6	
86	177	0.66	1.44375	0	5	
87	174	0.51	1.24762	0	5	
88	183	0.74	1.47391	0	7	
89	170	0.513333	1.26616	0	6	
90	187	0.62	1.41972	0	6	
91	176	0.693333	1.40924	0	6	
92	176	0.68	1.36782	0	5	
93	184	0.74	1.56388	0	8	
94	175	0.633333	1.32119	0	6	
95	178	0.68	1.51358	0	7	
96	186	0.433333	1.16285	0	6	
97	184	0.583333	1.35759	0	6	
98	176	0.573333	1.36307	0	6	
99	197	0.603333	1.28296	0	5	
100	170	0.623333	1.38617	0	6	

Figure 3: Esecuzione n = 8