N-Queens con algoritmo genetico

Matteo Migliarini 1886186

use\_module(library(lists)).

/\*

\* You can query with:

\* n\_queens(n,R).

\*

\* Note that each possible solution is represented by a list of size n: a permutation of [1. . . n]

\* \*/

n\_queens(N,Result):-

generatePopulation(N, N, InitialPopulation),

write('Initial Population: '),writeln(InitialPopulation),

genetic\_queens(InitialPopulation,Result).

genetic\_queens(Population, Result):-

evaluate(Population,Scores),

min\_list(Scores,Min),

Min = 0,

nth1(I,Scores,0),

nth1(I,Population,Result),

write('Solution: '),writeln(Result).

genetic\_queens(Population, Result):-

write('Population: '),writeln(Population),

evaluate(Population,Scores),

min\_list(Scores,Min),

Min =\= 0,

selection(Population,Scores,Survivors),

write('Survivors: '),writeln(Survivors),

crossover(Survivors, Cross),

write('Reproduction:' ),writeln(Cross),

mutation(Cross, NewPopulation),

%list\_to\_set(Mutants, NewPopulation),

genetic\_queens(NewPopulation,Result).

evaluate([],[]).

evaluate([QList|List], [R|Res]):-

% assigns to each individual a fitness value

fitness(QList, F), % List is an array of arrays

evaluate(List, Res),

R is F.

selection(QList,Scores, Res):-

% it sorts the individuals from the lowest to highest scoring in terms of fitness

pairs\_keys\_values(Pairs,Scores,QList),

keysort(Pairs, Sorted),

write('Fitness: '),writeln(Sorted),

proper\_length(QList,Len),

% then it extracts a random number X between 4 and Len

random\_between(4,Len, X),

% and takes only the best X scoring individuals

sublist(Sorted,X,Survivors),

pairs\_values(Survivors,Res).

crossover([],[]).

crossover([Q|List], Res) :-

% iterates through all the pairs of individuals

try\_crossing(Q,List,Crossings),

crossover(List, R2),

union([Q|R2],Crossings,Res).

try\_crossing(\_,[],[]).

try\_crossing(Q,[C|List],Res):-

% if crossover is possible between 2 individuals then it's done

proper\_length(Q,Len),

L is ceil(Len/2),

cross(Q,C,C1,C2,L),

try\_crossing(Q,List, Tail),

union([C1,C2],Tail,Res).

try\_crossing(Q,[C|List],Res):-

proper\_length(Q,Len),

L is round(Len/2),

\+ cross(Q,C,\_,\_,L),

try\_crossing(Q,List, Tail),

Res=Tail.

cross(Q1,Q2, R1,R2,P) :-

are\_crossable(Q1,Q2,P),

sublist(Q1,P, Q1s2),

sublist(Q2,P,Q2s2),

append(Q1s1,Q1s2,Q1),

append(Q2s1,Q2s2,Q2),

append(Q1s1,Q2s2,R1),

append(Q2s1,Q1s2,R2).

cross(Q1,Q2,R1,R2,P):-

\+are\_crossable(Q1,Q2,P),

C is P-1,

C>0,

cross(Q1,Q2,R1,R2,C).

are\_crossable(Q1,Q2, CrossingPoint) :-

% two individuals are crossable iff a section of them is a permutation

sublist(Q1, CrossingPoint, Q1s),

sublist(Q2, CrossingPoint, Q2s),

is\_permutation(Q1s,Q2s).

is\_permutation(Xs, Ys) :-

%checks if X and Y are permutation of each other

msort(Xs, Sorted),

msort(Ys, Sorted).

sublist([], \_, []).

sublist(\_,0, []).

sublist([L|List],N, [L|R]) :-

% returns the sublist from 0 to N-1

M is N-1,

sublist(List, M, R).

mutation([],[]).

mutation([Q|QList], [M1,M2|RList]):-

permute(Q,M1),

permute(Q,M2),

mutation(QList,RList).

permute(Q,M):-

%randomly swaps two elements of a list

proper\_length(Q,L),

random\_between(1,L,X),

random\_between(1,L,Y),

nth1(X,Q, Ex),

nth1(Y,Q, Ey),

replace(Q,Y,Ex,M1),

replace(M1,X,Ey,M).

% [1,2,3,4] -> [1,3,2,4]

replace([],\_,\_,[]).

replace([\_|List],Index,Element,[N|NewList]):-

Index = 1,

N = Element,

I is Index-1,

replace(List,I,Element,NewList).

replace([L|List],Index,Element,[N|NewList]):-

Index =\= 1,

N = L,

I is Index-1,

replace(List,I,Element,NewList).

fitness([], 0).

fitness([Q|Qlist], Res) :-

collisions(Q, Qlist, 1, Collisions),

fitness(Qlist, R2),

Res is R2 + Collisions.

collisions(\_,[], \_, 0).

collisions(Q,[Q1|Qlist],Xdist,Result) :-

Q =\= Q1, %not on the same row

Test is abs(Q1-Q),

Test =\= Xdist, %it means there is NO diagonal conflict

Xdist1 is Xdist + 1,

collisions(Q,Qlist,Xdist1, R2),

Result is R2.

collisions(Q,[Q1|Qlist],Xdist,Result) :-

Q =\= Q1, %not on the same row

Test is abs(Q1-Q),

Test = Xdist, %it means there IS diagonal conflict

Xdist1 is Xdist + 1,

collisions(Q,Qlist,Xdist1, R2),

Result is R2 + 1.

generatePopulation(0,\_,[]).

generatePopulation(N,D,[P|Pop]):-

n\_to\_one(D, Starter),

% generates a random population of size N

N>0,

random\_permutation(Starter, P),

N1 is N-1,

generatePopulation(N1, D, Pop).

n\_to\_one(0,[]).

n\_to\_one(N, [L|List]):-

% generates a list from n down to one

N>0,

L = N,

N1 is N-1,

n\_to\_one(N1, List).