Evolutionary Algorithm

```
% clear variables
clear;
```

Algorithm Parameters

```
population_size = 30;
number_of_genes = 80;
crossover_probability = 0.9;
mutation_probability = 1./number_of_genes;
maximum_generations = 200;
```

Initialize Population

```
population = randi([0 1], population_size, number_of_genes);

figure(1);
imagesc(population);
xlabel('Genes');
ylabel('Individuals');
title('Children');
```

Evolution Loop

```
for generation=1:maximum_generations
```

Evaluate Population

Simply count the number of ones in every gene

```
    fitness = sum(population,2);
    [best_fitness(generation), index] = max(fitness);
    best_individuum = population(index,:);
    median_fitness(generation) = median(fitness);
```

Early Termination

```
    if best_fitness(generation) == number_of_genes
        break;
    end
```

Selection

Tournament

```
    competitors = randi(population_size, population_size, 2);
    first_competitor_won = fitness(competitors(:,1)) > fitness(competitors(:,2));
    winner_indizes = [competitors(first_competitor_won,1);competitors(~first_competitor_won,2)
    first_mates = population(winner_indizes,:);

    competitors = randi(population_size, population_size, 2);
    first_competitor_won = fitness(competitors(:,1)) > fitness(competitors(:,2));
    winner_indizes = [competitors(first_competitor_won,1);competitors(~first_competitor_won,2)
    second_mates = population(winner_indizes,:);
```

Generate Next Generation

Crossover Determine if crossover will be done for each pair of mates

```
    do_crossover = (rand(population_size, 1) < crossover_probability);

    % Combine mate's genes
    index = [1:population_size]';
    crossover_point = randi([1 number_of_genes-1], population_size, 1) .* do_crossover(index);
    next_generation = [first_mates(:,1:crossover_point)...
        second_mates(:,crossover_point+1:number_of_genes)];

    % Elitism
    next_generation(1,:) = best_individuum;

    % Mutate
    % For each gene check if mutation shall appear
    mutate = (rand(population_size, number_of_genes) < mutation_probability);
    % XOR exactly changes 1s to 0s and vice versa if mutate equals 1 and
    % leaves genes as are if mutate equals 0
    % gene  mutate  result
    %  0      0       0
    %  0      1       1
    %  1      0       1
    %  1      1       0
    next_generation = xor(next_generation, mutate);

    population = next_generation;

    % Plot Parents and Children
    subplot(1,3,1);imagesc(first_mates);xlabel('Genes');ylabel('Individuals');title('ParentsA'
    subplot(1,3,2);imagesc(second_mates);xlabel('Genes');ylabel('Individuals');title('ParentsB
    subplot(1,3,3);imagesc(population);xlabel('Genes');ylabel('Individuals');title('Children')
    pause(0.1);
end
```

Plot Result

```
figure(2);
clf;
plot(best_fitness);
hold on;
plot(median_fitness);
xlabel('Generations');
ylabel('Fitness');
legend('Max Fitness', 'Median Fitness','Location','SouthEast')
```

```
best_individuum
```