Evolutionary Algorithm

```
% clear variables
clear;
```

Algorithm Parameters

```
population_size = 18;
%number_of_genes = 8;
crossover_probability = 0.9;

maximum_generations = 100;
% Keep record of last valid best individual
last_best_valid_individuals_fitness = 0;
```

Read data

Reads the data and stores weight and values to variables

```
data = importdata('items.csv')
weights = data.data(:,1);
values = data.data(:,2);
number_of_genes = length(weights);
% Read names without header
names = data.textdata(2:length(data.textdata),1);

mutation_probability = 1./number_of_genes;
```

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

```
    for index = 1:population_size
        individual = population(index,:);
        value = individual*values;
        weight = individual*weights;
        %
```

```
        %if weight>=400
        %    fitness(index) = 0;
        %else
        %    fitness(index) = value;
        %end
        %using soft constraints
        fitness(index) = value - (weight - 400);
        if weight < 400 && fitness(index) > last_best_valid_individuals_fitness
            last_best_valid_individual = individual;
            last_best_valid_individuals_fitness = fitness(index)
        end
    end

    % Dont vary last generation
    if generation < maximum_generations
        [best_fitness(generation), index] = max(fitness);
        best_individuum = population(index,:);
        median_fitness(generation) = median(fitness);
        best_fitness(generation)
```

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_wo
        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_wo
        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(ind
        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('Parer
        subplot(1,3,2);imagesc(second_mates);xlabel('Genes');ylabel('Individuals');title('Pare
        subplot(1,3,3);imagesc(population);xlabel('Genes');ylabel('Individuals');title('Childr
        pause(0.1);
    end
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')

% Output the best valid individual
best_value = last_best_valid_individual * values
best_weight = last_best_valid_individual * weights
best_pack = names(last_best_valid_individual)
```

Results

Best results using soft constraints: best_value =

```
1030
```

```
%best_weight =
%
%   396
%
%
%best_pack =
%or
%    'map'
%    'compass'
%    'water'
%    'sandwich'
%    'glucose'
%    'banana'
%    'suntan cream'
%    'waterproof trousers'
%    'waterproof overclothes'
%    'note-case'
%    'sunglasses'
%    'socks'
```