Generation Scheduling

Prahlad Chaudhary (17085054), Sarthak Choudhary (17085066), Shashank Kumar Singh (17085068)

Generation scheduling is the analysis of the electric power system network operations and the economic dispatch of each power plant to optimise overall energy delivery under given constraints, such as carbon dioxide emissions or transmission stability limits. Scheduling tools support different planning horizons, such as hourly, daily, weekly, monthly and annually, as well as 10-year plans for long-term unit commitment and maintenance scheduling.

It involves the determination of start-ups and shut-downs, and production levels of all units in all hours of the optimisation period, considering unit characteristics and system restrictions. The unit characteristics and restrictions that are handled are Minimum and maximum production levels, fuel cost function, start-up costs, minimum uptime and minimum downtime. The system restrictions handled is the power balance (supply equal to demand in all hours). As the size of an electric power system grows, the economic significance of power generation scheduling becomes essential in terms of reliability and longer operational lives of the constituent units.

Important terms related to the subject are Actual Generation and Scheduled generation, hereafter referred to as AG and SG, Deviation, Penalty and Incentives, Deviation charges, Area clearing price (ACP), Fuel Costs, and Net Gain. The SG is the day-ahead data which is divided into blocks of 15 minutes each, i.e. 96 blocks in a day for a single power plant. Similarly, AG is the real-time generation from the power plant meter. The difference between the two, or (AG - SG) is the deviation.

The main purpose of Generation Scheduling is to minimise the deviation, or the SG must be equal to AG at all times. However, practically it is not possible. Depending upon the frequency and the sign of deviation (Overinjection in case of negative deviation and underinjection in the case of positive deviation), penalties or incentives for the powerplant are calculated.

Nowadays, SG is predicted through machine learning. The past values of AG are used as a database to train and test the model.

Generation Scheduling

Prahlad Chaudhary (17085054), Sarthak Choudhary (17085066), Shashank Kumar Singh (17085068)

Introduction

The purpose of generation scheduling is to optimise the generation and fuel cost in meeting the future load demands. The problem is effectively an optimization problem based on multiple constraints.

Several methods to solve the problem exist; these are based on the Lagrange multiplier, linear programming and network flow algorithm. However, these encounter problems when dealing with certain cases: the Lagrange multiplier is unable to deal with non-convexity which is inherent in the generation schedule problem; the network flow technique provides impractical solutions. Linear programming requires a large number of variables even for a short period of time, and consequently, is unsuitable. In recent years, new algorithms have been developed to overcome these difficulties and attain a more optimised solution to the problem. These rely on genetic algorithms and fuzzy set theory, and are capable of solving non-convex optimisation problems. To further increase efficiency, these are combined with the simulated annealing algorithm to develop a hybrid algorithm, combining the advantages of both the approaches.

The genetic algorithm is used to find a maxima or a minima over an interval and return the input of the function for that extremum. The input is first encoded into a chromosome or a string of 1s and 0s: one parameter can take ten of these bits and another can take five, consequently if these are the only parameters, then the length of the chromosome becomes fifteen. The algorithm works by promoting those solutions which takes it closer to an extremum and penalizing those which takes it further from an extremum.

This is similar to the biological process of evolution as described by Darwin wherein species with dominant traits are preferred over species with recessive traits. Over thousands of generations, the processes of generation, mutation and crossover take place, and species that are much more advanced than their ancestors emerge. The judging of a strong or weak solution is implemented through a cost function. Finally, after obtaining the extrema, the chromosome is decoded back into usable parameters that were needed in the first place.

Paper: Hybrid Genetic/Simulated Annealing Approach to Short-term Multiple-fuel-constrained Generation Scheduling

Link to the paper: Hybrid genetic/simulated annealing approach to short-term multiple-fuel-constrained generation scheduling

The paper develops a method for short-term multiple-fuel-constrained generation scheduling which takes care of parameters such as the power balance constraint, generator operation limits, fuel availability, efficiency factors of fuels, and their supply limits. The objective is to determine, over a period of time, the most economical generation schedule of T generators and their fuel schedules for K types of fuels. An approach by combining the genetic algorithm (GA) and simulated annealing (SA) is proposed.

The procedure to solve the problem is as follows: a generator is selected randomly. Let this generator be ‘d’ or the dependent generator. The loadings and the fuel allocation factors of all other generators are determined using the hybrid algorithm or using GA. The parameters for the dependent generator r in all intervals are then calculated so that the power balance requirement is satisfied. Then the loading of the dependent generator is calculated; If it is not within the operational limits, the generation of another set of generator loadings takes place, and this process is repeated.

In adopting GA to the present problem, a chromosome represents a prospective solution which consists of the loadings and the fuel allocation factors of all the generators. The objective function is proportional to the total fuel cost function F, and consequently, the fitness function is proportional to 1/F.

In SA, in the high-temperature region, the loadings of the non-dependent generators are perturbed randomly in the range given by the operational limits, and provide a very high chance of generating feasible solutions; these may or may not be in the neighbourhood of the current solution. In the low-temperature region, the normal distribution (Gaussian function) is used to probabilistically generate new solutions in the vicinity of the current solution.

Finally, in the hybrid algorithm, The performance of GA is improved by introducing more diversity in the chromosomes in the early stages of the solution process by using the Gaussian function used in SA to replace fitter chromosomes by weaker child chromosomes. As the solution process progresses, the temperature level T in the Gaussian function is reduced as per the relation $T_k = r^{(k-1)}T_o$.

The hybrid approach ensures that the replacement probability is reduced slowly; thus, at later stages of the solution process, the chance of a fitter chromosome being replaced becomes less.

Paper: A Fuzzy-Optimization Approach for Generation Scheduling With Wind and Solar Energy Systems

Link to the paper: A Fuzzy-Optimization Approach for Generation Scheduling With Wind and Solar Energy Systems

This paper develops a fuzzy optimisation approach for generation scheduling for wind and solar energy systems. The fuzzy-set notation for various constrained parameters was used, which helps in representing the uncertainties inherent in practical optimisation. The optimised generation schedule is calculated with minimal total thermal unit fuel cost. The emission from the burning of fossil fuels is also considered in this approach. The effectiveness and application of the above method are tested using a simplified generation system.

The procedure to solve the problem is as follows: A wind and solar energy system is considered under an uncertain environment, i.e., under an uncertain load demand. Before performing the conventional generation scheduling methods, various parameters like available water, solar radiations, and wind speed, are forecasted to prevent errors. However, there is some error in the predicted values; consequently, they are represented as fuzzy set notations. A genetic algorithm is then employed to attain the desired generation schedule, and the total fuel cost is calculated with an optimal generation schedule value. For minimising the total cost, the experiment duration is divided into T time intervals.

The membership functions for the total fuel cost, load demand, available water, wind speed, and solar radiation are established to solve generation scheduling using this method. The modelling of the total fuel membership function is done so that high cost is given to low membership value. After the completion of the modelling of the membership function, the optimisation method is implemented, and the genetic algorithm is deployed for the optimisation update process.

The chromosome for applying GA uses the wind speed and solar radiation for wind power plants and solar power plants, respectively. The control variables for chromosomes in GA are designed, and a fitness function is developed for assigning the quality value to each iterative solution. Since there exists an error in the forecasted costs of the parameters, the membership value for each was obtained from its membership function. The parent population is obtained using the two strategies based on the existing chromosome's performance in offspring. The mutation and crossover are performed using simple roulette-wheel selection and defined probabilities.

On testing the above-stated method, the results reveal that it is effective in optimal generation scheduling with good performance, and the total fuel cost is optimised.

Paper: Genetic Aided Scheduling of Hydraulically-coupled Plants in Hydrothermal Coordination

Link to the paper: Genetic aided scheduling of hydraulically coupled plants in hydro-thermal coordination

Hydroelectric scheduling assumes a significant part within the operation planning of a power system. The problem is predominantly focused on both hydro units scheduling and thermal units dispatching. However, when multiple hydro plants are situated on a similar waterway, the water outflow from one plant may be a very significant portion of the inflow to other plants which are located downstream. Scheduling for these coupled hydro plants has been a difficult task. The electric and hydraulic couplings create a multidimensional, nonlinear programming problem.

To solve this problem, the dynamic programming with successive approximation (DPSA) is conventionally used. For the successive approximation (SA) procedure, it is necessary to specify a starting feasible schedule for every reservoir first. Then, one reservoir is scheduled while keeping the others schedules fixed, changing from one reservoir to the other until the stopping rule is fulfilled. The stopping rule might be the cost difference between the last two iterations within a specified tolerance limit or once a specified iteration number is exceeded. In this paper a global optimization technique known as genetic algorithm (GA) has been used due to its flexibility and efficiency and final results are compared with DPSA in a case study. Genetic Algorithm is a stochastic searching algorithm. The complex water balance constraints due to hydraulic coupling are embedded in the encoding chromosome string and are fulfilled throughout the proposed decoding algorithm. To make the outcomes more viable, the impacts of net head and water travel time delay are additionally considered.

Most methods for solving the hydro-thermal coordination problem are based on decomposition approaches that involve a hydro and a thermal sub-problem. These two sub-problems are typically planned by Lagrange multipliers and then the optimal generation Schedules of both hydro and thermal units are acquired by means of repetitive hydro-thermal iterations. A very much perceived difficulty is that the solutions to these two sub-problems may oscillate between maximum and Minimum generations with small changes of the multipliers. In the proposed GA approach, the thermal sub-problem can be tackled entirely independently. No multiplier coordination is required in the solution procedure, and subsequently the oscillation problem can be totally blocked. In the thermal sub-problem, the unit commitment is performed for the remaining thermal load profile (load profile minus HCP's generations), and the subsequent thermal expense is returned as the fitness of this HCP's schedule. The optimal solutions of both units are simultaneously obtained within the evolutionary cycle of "fitness."

The scheduling of HCPs deals with the problem of getting the optimal generations for both the hydro and thermal units. It focuses on minimizing the production costs of thermal units while

satisfying various constraints. With discretization of the absolute total scheduling time into sets of shorter time intervals (say, one hour as one time interval), the scheduling of HCPs can be mathematically formulated as a constrained nonlinear optimization problem.This gives us objective function for GA. The fitness function adopted here consists of thermal production cost plus the penalty cost. In order to use the best chromosomes and speed up the convergences, fitness is normalized between 0 and 1. When the fitness of each chromosome is calculated and it is sorted in descending order, the roulette wheel parent selection technique is used to select the best parents for the crossover and the mutation stages according to their calculated fitness. Mathematical outcomes show the appealing properties of the GA approach in useful application, in particular, a highly optimal solution cost and more robust convergence behavior.

Generation Scheduling: Design and Simulation

By: Prahlad Chaudhary (17085054), Sarthak Choudhary (17085066), Shashank Kumar Singh (17085068)

***Abstract*— This paper presents a method for short-term multiple-fuel-constrained generation scheduling which consists of parameters such as the power balance constraint, generator operation limits, fuel availability, efficiency factors of fuels, and their supply limits. . The objective of the paper is to determine, over a certain period of time, the best economical generation schedule of T generators and their fuel schedules for K types of fuels. An approach is proposed by combining the genetic algorithm (GA) and simulated annealing (SA) method.**

I. INTRODUCTION

THE motive of generation scheduling is to optimise the generation and fuel costs in meeting the future load requirements. The problem is mainly an optimization problem which has multiple constraints. The genetic algorithm is effectively used to locate a maxima or a minima over an defined interval and return the inputs of the function for that extremum. The input is first encoded into a chromosome or a string which consists of only 1s and 0s, one parameter of which can take ten of these bits and another parameter can take five of these bits, therefore if these are the only parameters, then the total length of the chromosome comes fifteen. The algorithm works by taking those solutions in account which takes it closer to an extremum and penalizing those solutions which takes it further from an extremum.

This very much resembles the biological process of evolution as described by Darwin, wherein species with dominant traits are preferred over species with recessive traits or say the fittest traits survive. Over the thousands of generations during the processes of generation the phenomenons of mutation and crossover take place and species that are much more advanced or fit than their ancestors emerge. The calculation of a strong solution or weak solution is implemented through a cost function. Finally, after finding the extrema, the chromosome is decoded back into usable parameters that were required in the first place.

II. MATHEMATICAL MODELLING

The power grid is assumed to have T types of generator and K types of fuels having a unique cost function with constraints of minimum value as Q_k. The goal of the basic short-term multiple fuel-constrained generation scheduling problem is to minimise the fuel cost, i.e. to determine the most economical dispatch available.

$$F = \sum_{k=1}^{k} (C_k * F_k) \quad \textbf{(1)}$$

A. Variables Used

Symbol	Meaning
T	Types of generators
K	Types of fuels
J	Total number of intervals in schedule horizon
F	Fuel cost function
F_k	Fuel cost function of k_{th} type
C_k	Fuel price of k_{th} type
Q_k	Minimum of fuel cost function
n_j	Number of hours in j_{th} interval
P_{tj}	Loading generator t in interval j
δ_{tjk}	Fuel allocation factor
f_{tk}	Heat rate function
ρ_{tjk}	Fuel availability factor
β_{kt}	Efficiency factor of fuel

D_j	Total load demand at interval j

Table 1: Variable used

B. Inputs

Number of generators (t) = 10
Number of fuel types (k) = 4
Number of intervals (j) = 96
Number of hours in intervals (nj) = 0.25
Fuel costs per unit = [2.75, 2.25, 2.00, 3.00]
Lower lim (load) = [11, 36, 36, 47, 47, 60, 60, 43, 43, 43]
Upper lim (load) = [16, 114, 114, 97, 97, 190, 190, 60, 60, 60]

C. Dependent Generators

The loading and the fuel allocation factor for the schedule horizon are committed so that power balance requirement is fulfilled in all the defined generation intervals and the generation schedule and the fuel schedule for the generators can be formed.

D. Dependent Fuels

For any interval, the total sum of fuel allocation factor of all the fuels to generators t must be 1. The constraint mentioned above is fulfilled by selecting arbitrary fuel and adjusting the allocation factor according to the below equation.

$$\delta_{t'fk'} = 1 - \sum_{k=k1,k\neq K}^{K} \delta_{t'fk} \quad \textbf{(2)}$$

E. Mathematical Equations

The cumulative consumption of fuel k in MBtu by all the generators is given by the double summation term in the equation given below. When the calculated value of F_k is below Q_k (the agreed minimum values) then the value is taken as Q_k.Therefore, the expression for F_k is given by equation (3).

$$F_k = \sum_{j=1}^{J}\sum_{t=1}^{T}(1/\beta_{kt})n_j P_{tjk} f_{tk}(\delta_{tjk} * P_{tj}) \quad \textbf{(3)}$$

The total load demand at an interval j is given by the summation of each generator demand at interval j as shown in the below equation (4).

$$D_j = \sum_{t=1}^{T} P_{tj} \quad \textbf{(4)}$$

There is also a constraint on maximum and minimum operation limit of the generators. For the t^{th} generator, it is given by equations (5) and (6).

$$P_{t,min} \leq P_t \leq P_{t,max} \quad \textbf{(5)}$$

$$F_{t,min} \leq F_t \leq F_{t,max} \quad \textbf{(6)}$$

III. FORMULATION FOR SOLVING PROBLEM

In order to solve the multiple-fuel-constrained generation scheduling problem we have applied the genetic algorithm formulation. Firstly, a generator was selected randomly, say r, which is referred to as a dependent generator. The loading and the fuel allocation factor for the schedule horizon are committed so that power balance requirements are fulfilled in all the generation intervals, and the generation schedule and the fuel schedule for the generators can be formed. The loading P_{rj} of the dependent generator r can be calculated using the below equation (7).

$$P_{rj} = D_j - \sum_{t=1,t\neq T}^{T} P_{tj} \quad \textbf{(7)}$$

If the value of P_{rj} is not within the defined operation limits then the assumed loadings, P_{tj}, are invalid and another set of loadings is to be assumed again by the optimisation techniques. To find out the fuel allocation factors for the dependent generator 'r' in any time interval j, first select arbitrarily an available fuel k'. The allocation factor of fuel k', hjk', to dependent generator r is represented as 1 minus the sum of the allocation factors of other fuels as shown in the below equation (8).

$$\delta_{rfk'} = 1 - \sum_{k=k1,k\neq K'}^{K} \delta_{rjk} \quad \textbf{(8)}$$

IV. STEPS FOR APPLYING GENETIC ALGORITHM

The genetic algorithm (GA) was applied to reach the optimal situation iteratively. The following steps were taken for implementation of GA.

Step 1: Initialize the control variable for GA chromosome.

Step 2: Define the fitness function for assigning the quality value to every solution produced.

Step 3: Calculate the fitness value at each iterative step.

Step 4: Create parent population using the below replacement rules in every iteration.

1) If all offspring are dominant over every existing chromosome in the parent population, then all of the offspring will replace all of the existing parent chromosomes in the new population.

2) If only some of the offspring are better than the existing parent population, they replace an equal number of existing parent chromosomes which
3) are lowest in order of fitness with offspring arranged in more fittest first order.

Step 5: Based on the fitness values, two chromosomes are selected from the given parent population. The selection is done according to the roulette-wheel selection method. The selected chromosomes are combined to form offspring chromosomes that inherit segments of information stored in parent chromosomes. The crossover operator is generally used with a higher probability and the mutation operator is generally used with a relatively smaller probability. The probabilities for crossover and mutation are set as 0.6 and 0.1, respectively.

Step 6: The fitness function evaluation and genetic evolution take part in an iterative process cycle which stops when a maximum number of defined generations is reached. Otherwise the procedure returns to step 2.

V. SIMULATION & OUTPUTS

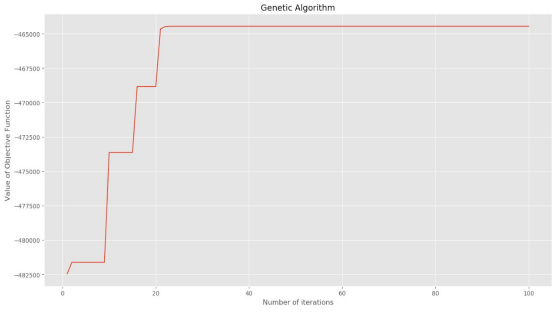

Figure 1: Output Plot

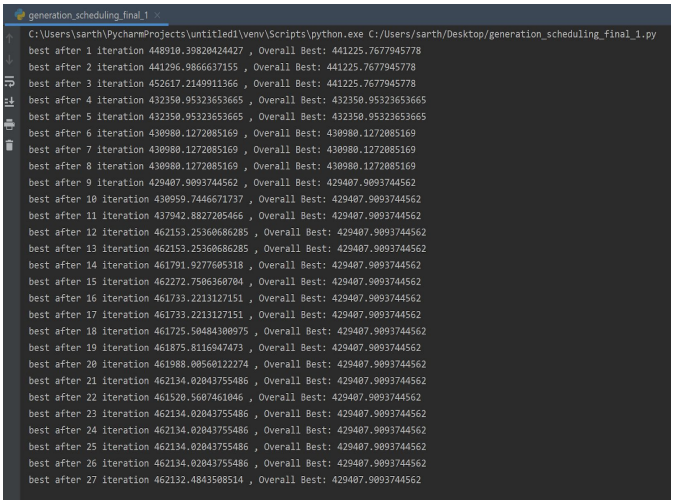

Figure 2 : overall best output values of cost function

VI. APPLICATION PARAMETERS

Algorithm	Values
Population size	10
Chromosome length	9
Number of iterations	100
Selection size	20
Probability Of crossover	0.6
Probability of mutation	0.1

Table 2: Parameters settings of GA

Fuel	Type 1 (coal)	Type 2 (coal)	Type 3 (Gas)	Type 4 (Gas)
Prices($/M Btu)	2.75	2.25	2.00	3.00
Q_k(MBtu)	-	-	-	3000
F_{min}(MBtu)	500	500	500	0
F_{max}(MBtu)	-	-	130000	-
Available to gen	1	8-15	2-7, 16-25	16-25

Table 3: Fuel types and data

VII. REFERENCES

[1] K.P. Wong and Y. W. Wong, "Hybrid genetic/simulated annealing approach to short-term multiple-fuel-constrained generation scheduling," in IEEE Transactions on Power Systems, vol. 12, no. 2, pp. 776-784, May 1997, doi: 10.1109/59.589681

[2] R. Liang and J. Liao, "A Fuzzy-Optimization Approach for Generation Scheduling With Wind and Solar Energy Systems," in IEEE Transactions on Power Systems, vol. 22, no. 4, pp. 1665-1674, Nov. 2007, doi: 10.1109/TPWRS.2007.907527.

[3] Po-Hung Chen and Hong-Chan Chang, "Genetic aided scheduling of hydraulically coupled plants in hydro-thermal coordination," in IEEE Transactions on Power Systems, vol. 11, no. 2, pp. 975-981, May 1996, doi: 10.1109/59.496183.

```
import random
import matplotlib.pyplot as plt
import numpy as np
import math


# Global Variables:
number_of_generators_t = 10
number_of_fuels_k = 4  # Types of fuels
number_of_intervals_j = 96
number_of_hours_in_interval_nj = 0.25
fuel_costs_per_unit = np.array([2.75, 2.25, 2.00, 3.00])

P_array_low = [11, 36, 36, 47, 47, 60, 60, 43, 43, 43]
P_array_high = [16, 114, 114, 97, 97, 190, 190, 60, 60, 60]

# Fuel allocation factor for 24
R_array = np.random.rand(number_of_generators_t, number_of_fuels_k)

# N_c = Number of bits used to describe one load, N_rlc = number of loads, N_l = number of
locations for one load
N_loadings_tj, N_fuel_allocation_factor_tj = 9, 9

# N: number of iterations, M: Selection Size
N, M = 100, 20

# # Range of output values
# low, high = 0, pow(10, 10)

# Size of initial population and chromosome length
pop_size = 10

# Chromosome length has 1 extra bit -> for series or parallel
chr_length_P = N_loadings_tj * number_of_intervals_j * number_of_generators_t
chr_length_RHO = N_fuel_allocation_factor_tj * number_of_generators_t *
number_of_intervals_j * number_of_fuels_k

# crossover and mutation rates
crossover_rate, mutation_rate = 0.6, 0.1

# Number of bits to be mutated in one instance of the mutation function
B = 1
```

```
# convert a chromosome(string of binary characters) to its real value
def binary_to_real_p(c):
    global P_array_high, P_array_low, N_loadings_tj
    j = 0
    count = 0
    op = []
    for i in range(N_loadings_tj, len(c) + 1, N_loadings_tj):
        op.append(P_array_low[j] + ((P_array_high[j] - P_array_low[j]) / (pow(2, N_loadings_tj - 1))
* int(c[i - N_loadings_tj:i], 2)))
        count += 1
        if count == 96:
            count = 0
            j += 1

    return op


def binary_to_real_r(c):
    global N_fuel_allocation_factor_tj, number_of_generators_t
    op = []
    for i in range(N_fuel_allocation_factor_tj, len(c) + 1, N_fuel_allocation_factor_tj):
        op.append((1 / pow(2, N_fuel_allocation_factor_tj - 1)) * int(c[i -
N_fuel_allocation_factor_tj:i], 2))

    return op


def heat_rate(p):
    global P_array_low
    a = [0.0586, 0.01172, 0.0069, 0.0069, 0.0114, 0.0035, 0.0016, 0.0009, 0.000052, 0.000052]
    b = [10.32, 10.32, 6.7378, 6.7378, 5.3519, 7.2038, 6.4352, 6.5651, 11.1222, 11.1222]
    c = [36.53, 18.27, 94.7055, 94.7055, 148.8907, 83.1773, 222.9258, 233.4006, 15.723,
15.723]
    e = [0, 0, 100, 100, 120, 120, 150, 150, 0, 0]
    f = [0, 0, 0.084, 0.084, 0.077, 0.077, 0.063, 0.063, 0, 0]

    op = []
    for i in range(len(p)):
        op.append((a[i % 10] * (p[i] ** 2)) + (b[i % 10] * p[i]) + c[i % 10] + abs(e[i % 10] * math.sin(f[i
% 10] * (p[i] - P_array_low[i % 10]))))
    return op


# returns the objective function value for a given chromosome
```

```
def obj_function(c_p, c_r):
    global number_of_hours_in_interval_nj
    array_p = binary_to_real_p(c_p)
    array_r = binary_to_real_r(c_r)
    # if sum(array_r) > 1:
    #     return -math.inf
    # else:
    array_heat_rate = [array_p[i] * array_r[i] for i in range(len(array_p))]
    array_heat_rate = heat_rate(array_heat_rate)
    op = sum(array_heat_rate)
    return -op


# generates a chromosome of a given length(k)
def c_generator(k):
    c = ''
    for _ in range(k):
        if random.random() > 0.5:
            c += '1'
        else:
            c += '0'
    return c


# generates population (1-D array of chromosomes) given total size(n) and individual length(k)
respectively
def pop_generator(n, k):
    pop = []
    for _ in range(n):
        pop.append(c_generator(k))
    return pop


# returns the 2nd element of a 1d array
def sort_key(array_1d):
    return array_1d[1]


# returns a crossed chromosomes, given 2 chromosomes and a crossover point
def cross_one_pt(chr1, chr2):
    p = random.randint(0, len(chr1) - 1)
    o1 = chr1[:p] + chr2[p:]
    return o1
```

```
# returns a crossed chromosome, given 2 chromosomes
def cross_two_pt(chr1, chr2):
    p1, p2 = random.randint(0, len(chr1) - 1), random.randint(0, len(chr1) - 1)
    p1, p2 = min(p1, p2), max(p1, p2)
    o1 = chr1[:p1] + chr2[p1:p2] + chr1[p2:]
    return o1


# returns a crossed chromosomes after uniform crossover
def cross_uni(chr1, chr2):
    o1 = ''
    for i in range(len(chr1)):
        if random.random() > 0.5:
            o1 += chr1[i]
        else:
            o1 += chr2[i]
    return o1


# takes chromosome and mutates b bits in it
def mutation(c):
    o = ''
    mut_b = [random.randint(0, len(c) - 1) for i in range(B)]
    for i in range(len(c)):
        if i in mut_b:
            o += str(1 - int(c[i]))
        else:
            o += c[i]
    return o


# takes population of chromosome (with their fitness scores) and tournament size, returns a
single individual
def sel_tournament(p_obj, n):
    return max([p_obj[random.randint(0, len(p_obj) - 1)] for i in range(n)], key=sort_key)[0]


# takes population (with fitness) and returns a single individual
def sel_roulette(p_obj):
    total = sum(i[1] for i in p_obj)
    p = [i[0] for i in p_obj]
    w = [i[1] / total for i in p_obj]
    op = (random.choices(p, weights=w))[0]
```

```
    return op


def sel_ranking(p_obj):
    n = len(p_obj)
    total = n * (n + 1) / 2
    p1 = sorted(p_obj, key=sort_key)
    p = [i[0] for i in p1]
    w = [(i + 1) / total for i in range(len(p1))]
    op = (random.choices(p, weights=w))[0]
    return op


# takes population (1-d array of strings) and returns 2-d array (string, obj fn value)
def pop_obj(pop):
    half = len(pop[0]) // 2
    return [[i, obj_function(i[:half], i[half:])] for i in pop]


# Combine two arrays
def combine(pop1, pop2):
    return pop1 + pop2


# Split array into two
def split(pop):
    half = len(pop) // 2
    return pop[:half], pop[half:]


mating_pool_p = pop_generator(pop_size, chr_length_P)
mating_pool_r = pop_generator(pop_size, chr_length_RHO)
# mating_pool = [mating_pool_p[i] + mating_pool_r[i] for i in range(len(mating_pool_r))]


best_p_1 = (max([i for i in pop_obj(mating_pool_p)], key=sort_key))
output_graph_x = []
output_graph_y = []



for i1 in range(N):
    mating_pool_p_1 = []
    mating_pool_p_2 = []
```

```
    mating_pool_p_3 = []
    # selection
    for _ in range(M):
        mating_pool_p_1.append(sel_roulette(pop_obj(mating_pool_p)))
    # cross-over (rate = 0.6)
    for _ in range(M):
        if random.random() <= crossover_rate:
            mating_pool_p_2.append(cross_two_pt(mating_pool_p_1[random.randint(0, M - 1)],
mating_pool_p_1[random.randint(0, M - 1)]))
        else:
            mating_pool_p_2.append(mating_pool_p_1[random.randint(0, M - 1)])
    # mutation (rate = 0.1)
    for _ in range(M):
        k = random.randint(0, M - 1)
        if random.random() <= mutation_rate:
            mating_pool_p_3.append(mutation(mating_pool_p_2[k]))
        else:
            mating_pool_p_3.append(mating_pool_p_2[k])
    mating_pool_p = mating_pool_p_3
    best_p_2 = (max([i for i in pop_obj(mating_pool_p)], key=sort_key))
    if best_p_2[1] > best_p_1[1]:
        best_p_1 = best_p_2
    output_graph_x.append(i1+1)
    output_graph_y.append(best_p_1[1])
    print('best after ' + str(i1+1) + ' iteration ' + str(abs(best_p_2[1])), ', Overall Best: ' +
str(abs(best_p_1[1])))

plt.style.use('ggplot')
plt.plot(output_graph_x, output_graph_y)
plt.grid(True)
plt.title("Genetic Algorithm")
plt.xlabel("Number of iterations")
plt.ylabel("Value of Objective Function")
plt.show()
```