

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | | | |
|  | | Nurse Scheduling Algorithm using MATLAB Genetic Algorithm | | | | |  | |
|  |  | | | | | | |  |
|  | | | |  |  | | | |
|  | | | | Mohamed Saber |  | | | |
|  | | | | Optimization TechniquesOptimization Final Project |  | | | |
|  | | |  | | |  | | |

1.What is the set cover problem?

* Idea: “You must select a minimum number [of any size set] of these sets so that the sets you have picked contain all the elements that are contained in any of the sets in the input (Wikipedia).” Additionally, you want to minimize the cost of the sets.

Input:

Ground elements, or Universe = { }

Subsets

Costs

Goal:

Find a set I {1,2,…..,m} that minimizes , such that

(Note: in the un-weighted Set Cover Problem,= 1 for all j)

The MIN-SET-COVER is a Combinatorial Optimization Problem (COP) that frequently shows up in real-world scenarios, typically when you have collections of needs (e.g., tasks, responsibilities, or capabilities) and collections of resources (e.g., employees or machines) and you need to find a minimal set of resources to satisfy your needs. In fact, MIN-SET-COVER generalizes MIN-VERTEX-COVER that we have seen in the earlier lectures. In vertex cover, each vertex (i.e., set) covers the adjacent edges (i.e., elements); in set cover, each set can cover an arbitrary set of elements

2.Application of the minimal subset cover problem

Set-cover heuristics are used in random testing ("fuzz testing") of programs. Suppose we have a million test cases, and we're going to test a program by picking a test case, randomly modifying ("mutating") it by flipping a few bits and running the program on the modified test case to see if it crashes. We'd like to do this repeatedly. If we do this naively, it will be less effective than it could be typically, many test cases cover basically the same code paths, but a small minority of test cases cover unusual code paths that would be interesting to test more intensively. So, here's one solution that is used in industry. We use a coverage measurement tool to instrument the program and record which lines of code are covered by each of the million test cases. Then, we choose a small subset S  of those million test cases that has maximal coverage: every line of code covered by one of the million test cases will be covered by some test case in S . S  is called a reduced test suite. We then apply random mutation & testing to the reduced test suite S . Empirically, this seems to make random testing more effective. The smaller S  is, the more effective and efficient testing becomes. So, how do we choose a reduced test suite S  that is as small as possible, while still achieving maximal coverage? Answer: that's a set cover problem, so we use standard heuristics/approximation algorithms for the set cover problem. The standard greedy approximation algorithm is typically used for this purpose. By far the most relevant, large size, important application of set covering is in personnel shift planning (mainly in large airline companies). There, elements to be covered are the single shifts (or single flights), and sets are legal combinations of work/no work schedules. These easily go to millions or even billions of variables, as the number of combinations is huge.

3. Review of recent population-based metaheuristics.

Diagram

Description automatically generated

Genetic algorithm (GA) is an optimization algorithm that is inspired from the natural selection. It is a population-based search algorithm, which utilizes the concept of survival of fittest . The new populations are produced by iterative use of genetic operators on individuals present in the population. The chromosome representation, selection, crossover, mutation, and fitness function computation are the key elements of GA. The procedure of GA is as follows. A population (Y) of n chromosomes are initialized randomly. The fitness of each chromosome in Y is computed. Two chromosomes say C1 and C2 are selected from the population Y according to the fitness value. The single-point crossover operator with crossover probability (Cp) is applied on C1 and C2 to produce an offspring say O. Thereafter, uniform mutation operator is applied on produced offspring (O) with mutation probability (Mp) to generate O′. The new offspring O′ is placed in new population. The selection, crossover, and mutation operations will be repeated on current population until the new population is complete. The mathematical analysis of GA is as follows :

GA dynamically change the search process through the probabilities of crossover and mutation and reached to optimal solution. GA can modify the encoded genes. GA can evaluate multiple individuals and produce multiple optimal solutions. Hence, GA has better global search capability. The offspring produced from crossover of parent chromosomes is probable to abolish the admirable genetic schemas parent chromosomes and crossover formula is defined as :

R=(G+2√g)/3G

where g is the number of generations, and G is the total number of evolutionary generations set by population. It is observed from the equation that R is dynamically changed and increase with increase in number of evolutionary generations. In initial stage of GA, the similarity between individuals is very low. The value of R should be low to ensure that the new population will not destroy the excellent genetic schema of individuals. At the end of evolution, the similarity between individuals is very high as well as the value of R should be high.

According to Schema theorem, the original schema must be replaced with modified schema. To maintain the diversity in population, the new schema keeps the initial population during the early stage of evolution. At the end of evolution, the appropriate schema will be produced to prevent any distortion of excellent genetic schema . Algorithm shows the pseudocode of classical genetic algorithm.

4. Flowchart of the implemented meta-heuristic technique

Diagram

Description automatically generated

1. Initialize Population

* What is population?
  + Population is a collection of genes
* What is gen?
  + it’s an individual in the population

1. Selection

* Selection is a process to choose 2 best from a population

1. Crossover

* crossover is doing exchanging activities between 2 parent which chosen in selection process. the idea is getting last half total character from parent1 and combine with first half total character from parent2 and vice versa.

1. Mutation

* mutation process is genetic operator used to maintain genetic diversity from one generation of a population of genetic algorithm chromosomes to the next. the idea is we need to choose mutation rate and looping for every character in a gen to get random number. If the random number is bigger than mutation rate, in that looping character will be replaced with random character

1. Evaluation

* evaluation process is check what fitness value. if the fitness of mutation process is equal to 100% so the guess is same with target word and the process will be stopped

1. Regeneration of Population

* regeneration is insert gen of mutation process into a population. in this process, the worst gen in a population will be dropped and replace with new gen from mutation process

5. Description of test examples.

Our test example is nurse scheduling problem and to generate the minimum number of days needed from the nurses while achieving specific constraints. The schedule is a full week schedule from Monday till Sunday, Saturday and Sundays are weekends, so we need to hire the minimum number of nurses to minimize the cost and at the same time achieve the optimized efficiency in their schedule. We do have specific constraints we need to follow while designing the schedule

1. Each nurse should work for 3 consecutive days and have 4 days off
2. At least half of the day-shift nurses should have a day from their weekends off.

Our fitness function should be Number of Nurses = Monday + Tuesday + Wednesday + Thursday + Friday + Saturday + Sunday should have the minimum number of nurses.

Saturday + Sunday + Monday >= 16

Sunday + Monday + Tuesday >=12

Monday + Tuesday + Wednesday >= 18

Tuesday + Wednesday + Thursday >= 13

Wednesday + Thursday + Friday >= 15

Thursday + Friday + Saturday >= 9

Friday + Saturday + Sunday >= 7

At least half the nurses should have a day off from their weekend.

Number Of Nurses = Monday + Tuesday + Wednesday + Thursday + Friday + Saturday + Sunday

Total Number of Nurses Don’t work in weekends / Total Nurses >=0.5

Monday + Tuesday + Wednesday / Total Nurses >=0.5

2(Monday + Tuesday + Wednesday)>= Total Nurses

Monday + Tuesday + Wednesday >= Thursday + Friday + Saturday + Sunday

Our Lower Bound is having 0 nurses working everyday and we don’t have upper bound.

6. Code

function NumberOfNurses = fitnessFunction(X)

Monday = X(1);

Tuesday = X(2);

Wednesday = X(3);

Thursday = X(4);

Friday = X(5);

Saturday = X(6);

Sunday = X(7);

NumberOfNurses=Monday+Tuesday+Wednesday+Thursday+Friday+Saturday+Sunday;

end

function [LinearConstraint, EqualityConstraint] = Constraint(X)

Monday = X(1);

Tuesday = X(2);

Wednesday = X(3);

Thursday = X(4);

Friday = X(5);

Saturday = X(6);

Sunday = X(7);

%Each nurse should work 3 consecutive days shifts and then has 4 days off

%At least half of the nurses should have atleast one day weekend off

EqualityConstraint=[-Monday-Saturday-Sunday+16;

-Monday-Tuesday-Sunday+12;

-Monday-Tuesday-Wednesday+18;

-Tuesday-Wednesday-Thursday+13;

-Wednesday-Thursday-Friday+15;

-Thursday-Friday-Saturday+9;

-Friday-Saturday-Sunday+7;

-Monday-Tuesday-Wednesday+Thursday+Friday+Saturday+Sunday+0.5];

LinearConstraint=[];

end

%-------------------------------------------------------------------------%

% N U R S E S C H E D U L I N G P R O B L E M %

% A S i m u l a t e d A n n e a l i n g %

%-------------------------------------------------------------------------%

clc;

%-------------------------------------------------------------------------%

% Genatic Algorithm %

%-------------------------------------------------------------------------%

objectFunction = @fitnessFunction;

numberOfVariables = 7;

lowerBound = [0 0 0 0 0 0 0];

upperBound = [];

constraintFunction = @Constraint;

options = optimoptions('ga','MaxStallGenerations',50);

options = optimoptions('ga','PlotFcn',@gaplotbestf,'Display','iter');

% For Reproducibility

rng default;

[X,finalValue] = ga(objectFunction,numberOfVariables,[],[],[],[],lowerBound,upperBound,constraintFunction,options);

X = ceil(X);

finalValue=ceil(finalValue);

bar(X);

xlabel('Days');

ylabel('Nurses');

title('Optimal Number Of Nurses/Day');

fprintf("-------------------------------------------------------------------------------------------\n");

fprintf("| Number of Nurses | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday | Sunday |\n");

fprintf("-------------------------------------------------------------------------------------------\n");

fprintf("| %d | %d | %d | %d | %d | %d | %d | %d |\n",finalValue,X(1),X(2),X(3),X(4),X(5),X(6),X(7));

fprintf("-------------------------------------------------------------------------------------------\n");

Table

Description automatically generated

Graphical user interface, application

Description automatically generated

Chart

Description automatically generatedChart, bar chart

Description automatically generated