Solving Air Crew Scheduling Problem Using Genetic Algorithms

Yi Liu - 1605486

In this Assignment, I was using MATLAB as programming language to implement the genetic algorithm for solving travelling air crew scheduling problem. For each benchmark problem, I did 30 independent runs for 30 times of generations. The number of generation is changeable, for more information about coding, please read the README file inside the folder.

1.1 Introduction

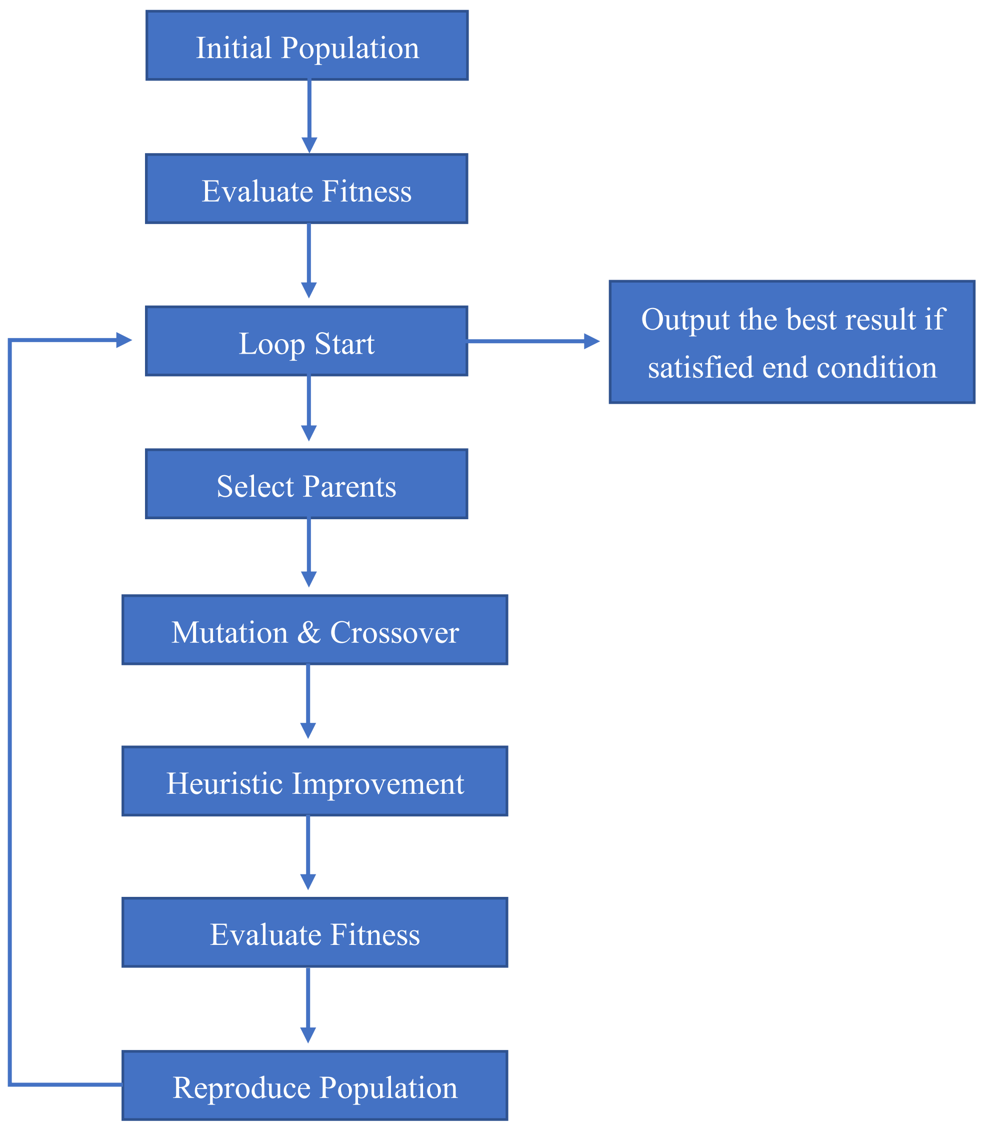
Genetic Algorithm

Genetic algorithm (GA) is a stochastic local search inspired by evolution of natural organisms The genetic algorithm could avoid the drawbacks of the randomized search and local search. The randomized search is good at exploration but bad at exploitation, especially bad for problems where good solutions are a very small portion of all possible solutions. The local search is good at exploitation but bad at exploration, it often gets stuck into local optimum. However, the genetic algorithm can automatically acquire and accumulate situations of the search space in the search process, and adaptively control the search process to obtain the high quality solution.

The genetic algorithm contains the following main sections:

1. Selection: Select parents from population based on their fitness.
2. Mutation: Mutation operator would randomly change some parts in individual to achieve mutation operation. In binary string, it will randomly flip selected bits. The probability of mutation rate often is between ‘1/string length’ to ‘1/2’.
3. Crossover: Two individuals will be cut off at the same position(s), exchanged the parts and combined as two new individuals.
4. Heuristic improvement Operator: It will improve the results after mutation and crossover. The data after mutation and crossover would often contain over-covered and under-covered rows, it would remove the columns that cause over-covered and add columns that cause under-covered.
5. Reproduction: update the population base with the new offspring for the next generation. usually by replacing the least-fit individuals (or combining old population and offspring, then applying stochastic ranking and selecting top part of data as the new population).

1.2 Flow Chart



1.3 Pseudo Code

Main Function

```
begin
    P = pseudo_random(NATIVE_DATA) // Initial solutions.
    for(t = 1; t <= NUMBER_OF_GENERATION; t++) // Generation Loop
        Select parents from Pt by fitness
        crossovered_values = crossover(parent1,parent2)
        mutated_values = mutation(crossovered_values)
        improved_values = improveOperator(mutated_values)
        offsprings_fitness = fitness(improved_values)
        // ----- Reproduce Population ----- //
        mixed_Pt = [Pt; mutated_values]
        ranked_Pt = stochasticRanking(mixed_Pt , offsprings_fitness)
        Pt+1 = ranked_Pt(1:POPULATION_SIZE, : ) //Recap the population
    end
end
```

Crossover Function

```
// Only summarized ‘n Point’ crossover, for ‘one Point’ and ‘uniform’
// Please check the crossover.m
Function crossovered = crossover(parent1,parent2)
    N = 10 // Set your N of ‘n Point’ crossover.
    cut_points = RANDOMLY SELECT 10 POINTS ON STRING
    sol = []
    for( i = 1; i <= N; i++ )
        if i == N
            sol = [sol parent2(first_point : end)]
        elseif mod(i,2) == 1
            second_point = cut_points(i)
            sol = [sol parent1(first_point : second_point-1)]
            first_point = second_point;
        else
            second_point = cut_points(i)
            sol = [sol parent2(first_point : second_point-1)]
            first_point = second_point;
        end
    end
    crossovered = sol
end
```

Mutation Function

```
Function mutated = mutation(F)
    P = 1/LENGTH_OF_F // Set your mutation rate here (1/length : 1/2)
    for( i = 1; i <= LENGTH_OF_F ; i++ )
        c = F(i) // c is current value in F
        if rand <= P // Should we mutate this value?
            if( c == 1 )
                F(i) == 0
            elseif( c == 0 )
```

```
                F(i) == 1
            end
        end
    end
    mutated = F
end
```

2. Average Results & Standard Deviations

The following are results of 30 independent runs, each of which takes 30 generations and gives the best result. If you set higher numbers of generation, the average results will be better, and the standard deviation will be lower.

Sppnw41.txt

11307	11307	11307	11307	11307	11307
11307	11307	11307	11307	11307	11307
11307	11307	11307	11307	11307	11307
11307	11307	11307	11307	11307	11307
11307	11307	11307	11307	11307	11307

Average (30 runs) = 11307

Standard Deviation (30 runs) = 0

Sppnw42.txt

8114	7938	8968	7674	9716	7722
7938	7674	7656	9406	7666	7674
7666	7722	7666	7930	7674	7722
7674	7930	8432	7674	8114	7666
9808	7938	7930	7722	8766	7938

Average (30 runs) = 8154

Standard Deviation (30 runs) = 624

Sppnw43.txt

9216	9504	9422	9422	9462	9590
9704	9422	9532	9012	8974	9504
9430	9012	9586	9622	9560	9670
9504	9422	9216	9462	8974	9216
9290	9590	9216	9586	9704	9504

Average (30 runs) = 9411

Standard Deviation (30 runs) = 216

3. Comparison

Stochastic Ranking Method

In this assignment, I implemented stochastic ranking method in my programme. It is the ranking method which introduced a random probability P_f. If the we set a higher value of P_f (e.g. greater than 0.5), more comparisons are focus on fitness, but less focus on constraint violation, and infeasible solutions are likely to occur. If we set a lower value of P_f (e.g. less than 0.5), more comparisons are focus on constraint violation, but less focus on fitness, and feasible solutions are likely to occur. The essence of P_f is that allows infeasible solutions (with higher fitness value) has a higher ranking than feasible solutions (with lower fitness value), therefore P_f is often set between 0.45 to 0.5. It will improve the diversity of search space and give a better quality of solution. One of important point is that solution from infeasible region may the only solution that can lead to the global optima, however the global optima may never be found if the solutions from infeasible region be ignored at first. Essentially, the stochastic ranking method is helping the algorithm escape from local optimal. We can use this method to rank the combined population base (old population + new offspring), then select the top part (same size as original population) as the new population for the next generation.

Ranking Replacement Method

This is another method to balance the trade-offs between fitness and unfitness (constraint violation). Different from the stochastic ranking method, it considers fitness and unfitness without a probability like P_f, but a fixed scheme that divide the population as four subgroups with respect to the child. It takes one offspring each time, and replaces the least-unfit individual of the first non-empty subgroups. The order of subgroups are defined as the following:
G_1 = Higher fitness value and higher unfitness value.
G_2 = Lower fitness value and higher unfitness value.
G_3 = Higher fitness value and lower unfitness value.
G_4 = Lower fitness value and lower unfitness value.
According to those features, the stochastic ranking method has higher probability than ranking replacement method to escape the local optima by accepting an adjustable parameter.

How to Run The Programme

Please read the README file inside the folder. Thanks for your reading.