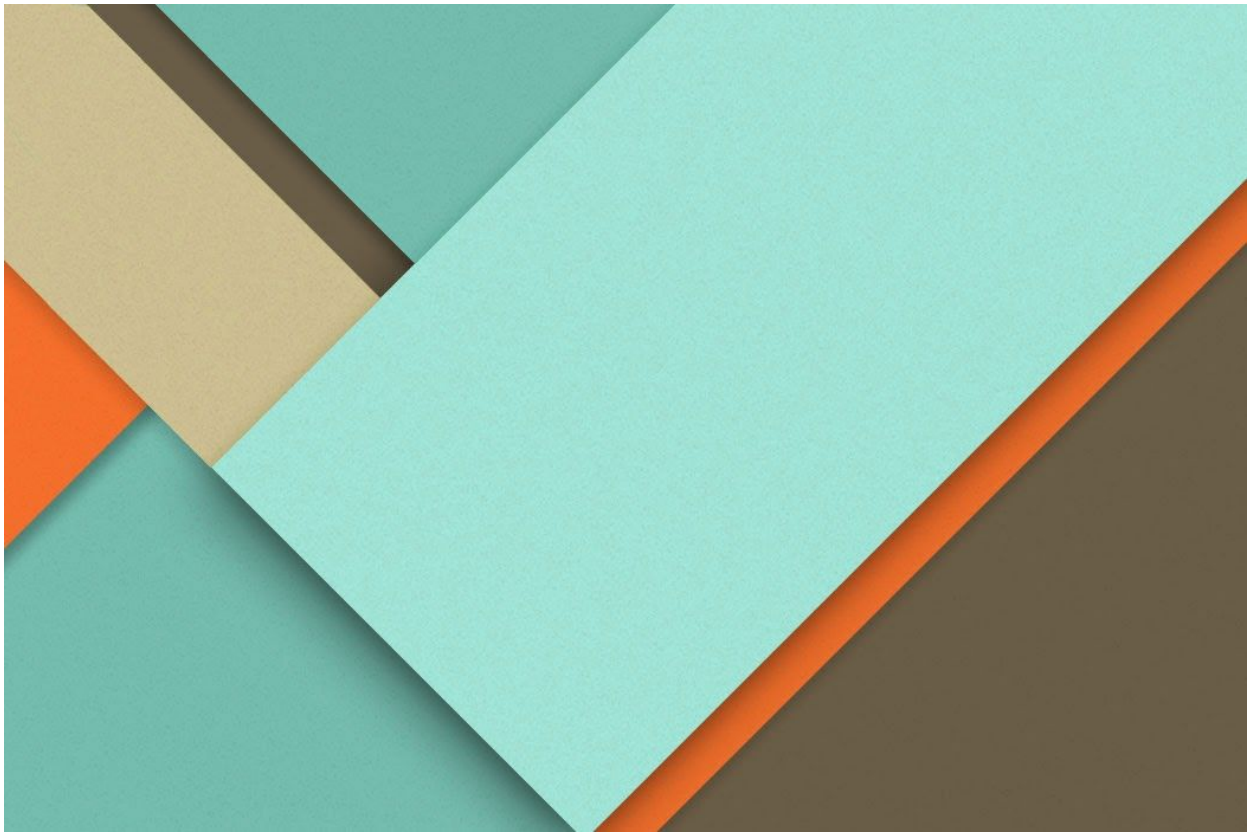


Scheduling Freight Trains on Complex Track Network

~ Genetic Algorithms Term Project

Rishi Jain ~ 18IM10023

Vidit Gupta ~ 18IM10038

Department of Industrial and Systems Engineering

Indian Institute of Technology, Kharagpur - 721302, India

Overview

This Project focuses on the scheduling of trains. The objective of operational scheduling for freight trains is to safely move each train from its origin to its destination as fast as possible so that the total time all the trains is minimized, keeping in mind the deadlock constraint. To improve the railway transit system and service, it is necessary to build optimal train scheduling. Here we propose a method to find an near optimal solution where the train's path is decided by the minimum total time it takes to reach from origin to destination. To solve the problem using genetic algorithms, the actual rail network needs to be translated into nodes. A pair of adjacent nodes denotes a train track segment.

Problem Definition

Objective

1. Minimise the total time taken by all the trains to reach from its origin to destination.

$$\text{MIN} \sum_i T_i \quad , i \in \text{N}$$

Constraints and Assumptions

- Safety distance maintained (no collision over segments)
- No train can go in reverse direction, only single direction movement allowed
- All trains are identical and are significantly smaller than any length of an segment
- All trains move with the same constant speed and distance between each pair of nodes is identical.
- Once the train starts, it will stop directly at its destination, nowhere in the middle of the path.

Therefore time taken by any train $T_i \propto$ No. of nodes train passes through

NOTE : These assumptions are not rigid. They can be relaxed by giving vital information regarding these in numeric form. The railway freight transportation system inspires all these physical constraints.

Our Approach

The following pseudocode illustrates our approach to use the different components of Genetic Algorithm to minimize our objective.

Algorithm Find minimum total time paths of trains	
INPUT	Set the Parameters: pop_size, no_of_gen,Pc,Pm,N
Output	Minimized Total time and path of all trains

1.
2.
3.
4.
5.
5.1
5.2
5.3
5.4
6.
7.
8.
	Generate the initial population according to population size
	gen ←1
	while (gen <= gen_max) do{
	Apply Genetic operations to obtain new population
	Apply selection operator to population
	Apply crossover according to Pc parameter (Pc >= 0.8)
	Apply Mutation according to Pm parameter (Pm <=0.1)
	Compute the total time(fitness) of the child path
	Set gen = gen +1
	}
	sort(final_population) on basis of fitness value
	Select the paths which has minimum fitness value i.e minimum total time

Description of Our Approach

I. Network Construction

The problem needs to be described with all its initial parameters to be passed to the program like no. of trains, the railway network etc.

Nomenclature

N	No. of trains
O	Set of all origins
D	set of all destinations
S	Set of all nodes
A_S□□	Set of all nodes adjacent to node j ,j∈S
P	Set of paths of a train, i.e. population for an individual
Seg	Set of all segments, i.e every pair of adjacent nodes joined

This variable definition will help introduce any kind of problem structure in numbers , so that we can apply genetic operators.

II. Population initialization

The first task is to initialize the population for the starting generation. There are many ways to guide the initial population, but the quickest possible way is randomly choosing paths from origin to destination. We traversed through adjacent nodes without turning back until we reached the destination. If the individual does not reach the destination, that individual gets discarded.

Initial Population Generation Algorithm	
INPUT	Set the parameters : origin, destination, pop_count, 2d array of adjancent_stations
OUTPUT	2-d matrix path of trains - individuals

1.
2.
3.
4.
5.
6.
7.
 	
8.
 	
9.
	i ⟵0
	while(i < pop_count) do{
	  set temp = [ ]
	  j ⟵origin
	while(j != destination) do{
	  append j in temp
	  now remove all j from adjacent_stations[j]
	// so that train won't go from stations it has already passed
	
	  Set j = random choice from adjacent_stations of j
	}
	
	Save the temp in population array
	
	Reintialise temp = [ ]

10. 11.	i ⟵ i + 1 }

III. Genetic operators

Now to run over generations to attain a desirable set of train paths, we defined our genetic operators as follows:

- **Selection**

  For selection after every generation, we used absolute elitism, i.e., selecting the best individuals from the population for the next generation. We are not worried about the diversity of our solutions because of the operators we are using, and secondly, we can make any of the trains wait on their respective sources for some time.

Elitism Algorithm	
INPUT	Set the parameters : population, pop_count
OUTPUT	Updated population

1.
2.
3.
4.
5.
6.
 	
7.
8.
9.
10.
11.
	set c2 = population.copy()
	i ⟵ 0
	while (i <= length(c2)) do{
	  set pop_fitness = [ ]
	  set x = Call fitness function on c2[i]
	  append x in pop_fitness
	}
	
	sort pop_fitness values in ascending order
	j ⟵ 0
	while (j != pop_count) do{
	  delete maximum value from pop_fitness
	  j ⟵ length(pop_fitness)
	}

- **Crossover**

  The crossover method employed here is straightforward. We find common nodes between two individuals and select any two of them. Then we swap the segment of the path between these two nodes, including them. It ensures that there is always a complete path from source to destination in the child gene.

Crossover Algorithm	
INPUT	Set the parameters : parent_1, parent_2
OUTPUT	child_1, child_2
1.	Set C_s = common nodes between parent_1 and parent_2
2.	Set node_1, node_2 = different random nodes from C_s
3.	Set h1 = path segment from node_1 to node_2 in parent_1
4.	Set h2 = path segment from node_1 to node_2 in parent_2
5.	child_1,child_2 = swap(h1,h2) in parent_1 and parent_2
6.	Save child_1, child_2 for next generation selection

- **Mutation**

  The method used here uses the initialization function. If an individual is selected, then we pick one of its nodes as a mutation point and redefine the path from this point to the destination node, using the initialization function.

Mutation Algorithm	
INPUT	Set the parameter : individual
OUTPUT	new_individual

1.	set temp = random int(0, length(individual))
2.	set new_origin = node at individual[temp]
3.	Set new_destination = same as destination of individual
4.	delete individual from node temp till end node
5.	Call an individual generator to generate the individual from new_origin to new_destination and append it with the given individual.

IV. Fitness Function

A well-defined fitness function is required considering our objectives and constraints. Selection will be carried out based on this fitness value for every individual.

Fitness Evaluation Algorithm	
INPUT	2-d array with one complete path of all trains - individual
OUTPUT	Fitness value of individual
1. 2. 3.	Fitness = 0 Fitness += length of every path in the individual For every pair of paths in the individual: If they share any common nodes(largest common segment) then either of the trains needs to wait for some time for no collision. Waiting time = length(largest common segment) For i in range(Waiting time): path.forward_append(first node) Set fitness = fitness + 1

Results

We optimised the problem for the following test cases shown.

FIRST

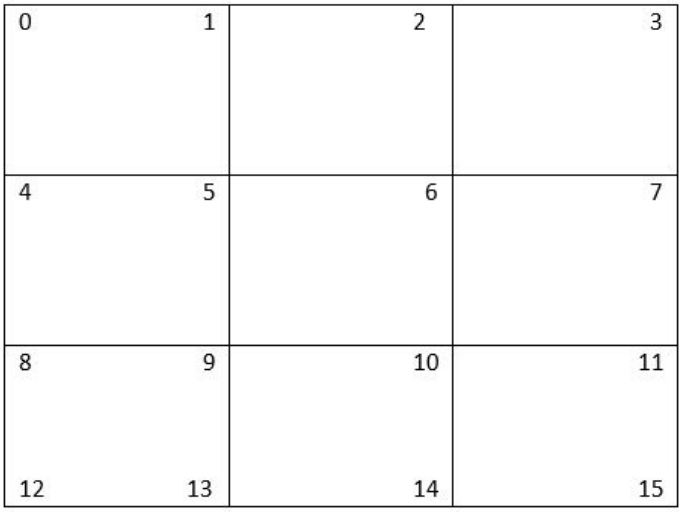
Info \ Trains	1	2	3
Source node	0	15	11
Destination node	15	0	3

SECOND

Info \ Trains	1	2	3
Source node	0	15	2
Destination node	15	0	14

Railway Network Diagram:

We chose a small sophisticated network because the computation increases exponentially with the addition of every node. It may take a large number of generations as well as a huge population to achieve a near-optimal solution for an extensive network, thereby demolishing the practical advantage of genetic algorithms over other optimization methods.



This 16 node network(every corner is a node) can be optimised real quick, with a few trains on it. After many test cases, we observed that a near optimal practically implementable solution can be achieved in less than 10 generations(which is real fast compared to other methods searching over complete feasible space).

Genetic Variables:

Population Size	10
No. of generations	10
Probability of crossover	0.8
Probability of mutation	0.1

Results and Paths obtained:

Below shown is a snap of *spyder* IDE console after running program on our test cases-

FIRST:

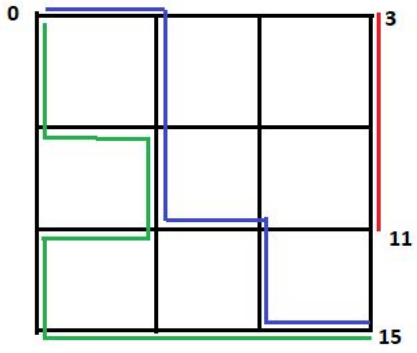
```
In [2]: runfile('C:/Users/Rishi Jain/Desktop/train_scheduling_GA.py', wdir='C:/Users/Rishi
Jain/Desktop')

 Minimized total running time of all trains (in units)  =  16
 for the train paths:

train  0  follows the path-  [0, 1, 5, 9, 10, 14, 15]
train  1  follows the path-  [15, 14, 13, 12, 8, 9, 5, 4, 0]
train  2  follows the path-  [11, 7, 3]
```

SECOND:

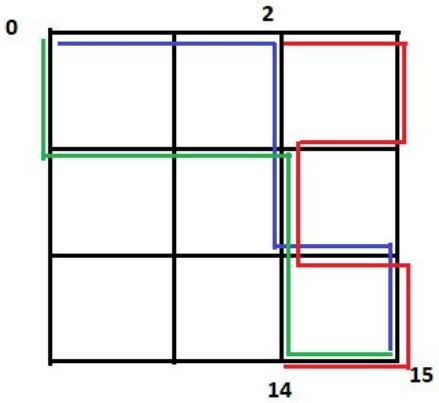
```
In [4]: runfile('C:/Users/Rishi Jain/Desktop/train_scheduling_GA.py', wdir='C:/Users/Rishi
Jain/Desktop')

 Minimized total running time of all trains (in units)  =  20
 for the train paths:

train  0  follows the path-  [0, 0, 1, 2, 6, 10, 11, 15]
train  1  follows the path-  [15, 14, 10, 6, 5, 1, 0]
train  2  follows the path-  [2, 3, 7, 6, 10, 11, 15, 14]
```

The multiple occurrences of the source node in the path suggest that the train waits for those many unit times until it starts its journey. By waiting, we avoid any possible collisions as paths share common segments.

This path array for every train can also be interpreted as a node position with indexing as time. Once a train starts its journey, it stops directly at its destination.



Test case 1



Test case 2

Conclusion

Train scheduling and dispatching is one important sub-problem of the freight railroad management problem. In this project, we propose GA based optimization methods for the scheduling of freight trains. The traditional problem-solving method has limitations due to the complexity of selecting the upper or lower bounds of variables and parameters when the sub-objective functions are being constructed. The complexity arises due to the extensive computation and necessary assumptions and simplification. The solution procedures of the proposed GA-based optimization methods do not involve any such assumption or simplification, and the quality of the result is guaranteed. As we can see from the above-obtained results, this sixteen node network can be optimized real quick, with a few trains on it. We optimized this network in ten generations of our definition of genetic operators suggesting supremacy of Genetic Algo over other optimisation heuristics(at least upto a scale of sophistication). This GA-based heuristic optimization approach is a flexible approach that can be extended to find solutions for various types of operation programming scenarios. It can also be used as an all-purpose algorithm for economic optimizations with some modifications.

*****