Faculty of Engineering and Technology

Computer Science Department

Artificial intelligence
ENCS 3340

Project 1 report

Optimizing Job Shop Scheduling in a Manufacturing Plant using Genetic Algorithm

Prepared by:
Abdel Rahman Shahen 1211753
Mahmoud Awad 1212677

Instructor:
Dr. Aziz Qaroush

Section: 3
Date: 5/19/2024

algorithm

"A genetic algorithm (GA) is an optimization technique inspired by natural selection. It evolves solutions to problems through selection, crossover, and mutation. Starting with a population of random solutions, GA iteratively improves them, selecting the fittest individuals to create the next generation, leading to an optimal or near-optimal solution.". [1]

Project idea

Develop a job shop scheduling system using a genetic algorithm. The system accepts a list of jobs and available machines, with each job defined as a sequence of operations specifying the machine and processing time. The output is a schedule for each machine, showing start and end times for each job, visualized using a Gantt Chart.

Problem formulation

The genetic algorithm requires chromosomes, Genes and a metric to evaluate fitness. And generation operations. (crossover and random mutation)

Assuming the input:

Job_1: M1[10] -> M2[5] -> M4[12]

Job_2: M2[7] -> M3[15] -> M1[8]

chromosome

Then one solution (chromosome) is represented as the set of all operations to execute

e.g. chromosome1 = (M1[10], M2[5], M1[8], M3[15], M2[7], M4[12])

Gene

As you can see the Genes are represented like the operations inside each solution

chromosome1 Gens: Gen1: M1[10], Gen2: M2[5], Gen3: M1[8], Gen4: M3[15], Gen5: M2[7], Gen6: M4[12]

metric

And then, the metric used is the makespan.

which is the total time required to complete all jobs. To evaluate the efficiency of a schedule (solution). the goal is to minimize the makespan. It's done by initializing tracking variables. Then scheduling operations considering machines availability. Then Determine the makespan as the maximum time at which any machine completes its last operation. The makespan is used as the fitness value, with a lower makespan indicating a more efficient schedule (solution). This fitness value guides the selection, crossover, and mutation processes in the genetic algorithm to evolve better solutions over successive generations.

To know more about makespan please follow the link below:

https://www.sciencedirect.com/topics/computer-science/minimum-makespan

In generation operations crossover and mutation is used:

selection

But before generation new population we need to select parents to do crossover on. The selection uses the fitness values calculated to select best two parents (solutions) to do crossover on.

Crossover

Crossover: the crossover is done by choosing random crossover points let's say for our example p1 = 2 and p2 = 4 and let's say the 2 selected parents are:

Parent1: (M1[10], M2[5], M1[8], M3[15], M2[7], M4[12])

Parent2: (M1[8], M2[5], M1[10], M3[15], M4[12], M2[7])

Then the two new children will start with values:

Child1: (None, None, M1[8], M3[15], None, None)

Child2: (None, None, M1[10], M3[15], None, None)

Then it fills the remaining from the other parent making sure to check if the value exists in child or not. If the operation already exists it continues until the children is filled.

Final children:

Child1: (M2[5], M1[10], M1[8], M3[15], M4[12], M2[7])

Child2: ((M1[8], M2[5], M1[10], M3[15], M2[7], M4[12])

mutation

Finally, the mutation is simply implemented to exchange gens in the chromosome for example if we have the child solution: (M1[8], M2[5], M1[10], M3[15], M4[12], M2[7]) and the mutation was decided to be done then it selects 2 random indices and exchanges values between them if index 1 = 2 and index 2 = 5 then the new solution is: (M1[8], M2[5], M2[7], M3[15], M4[12], M1[10]).

Deciding if doing mutation is done by defining a constant mutation rate in our code 0.1 Then a random number x is generated if x is less than the mutation rate then do the mutation else do not do the mutation.

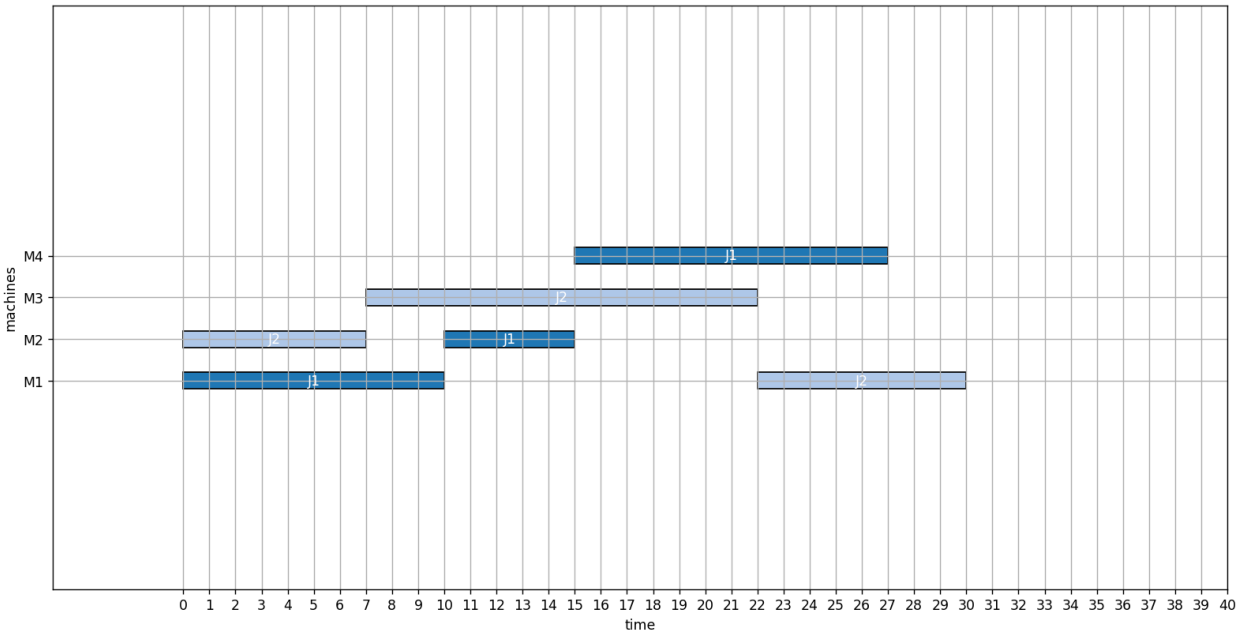
Test case 1

Sample input:

[“M1[10] -> M2[5] -> M4[12]”,

”M2[7] -> M3[15] -> M1[8]”]

Sample output:



As declared In the image the machines as presented on the y-axis and the processing time for each operation is presented on the x-axis.

TEST CASE 2

Sample input:

"M1[10] -> M2[5] -> M4[12] -> M3[8] -> M5[15] -> M6[7] -> M7[9] -> M8[6]",

"M2[7] -> M3[15] -> M1[8] -> M4[10] -> M5[6] -> M6[12] -> M7[8] -> M8[10]",

"M4[11] -> M1[9] -> M2[14] -> M3[10] -> M5[8] -> M6[11] -> M7[10] -> M8[12]",

"M3[6] -> M4[13] -> M5[9] -> M1[12] -> M2[11] -> M6[10] -> M7[7] -> M8[9]",

"M5[8] -> M1[15] -> M2[6] -> M4[14] -> M3[12] -> M6[8] -> M7[11] -> M8[10]",

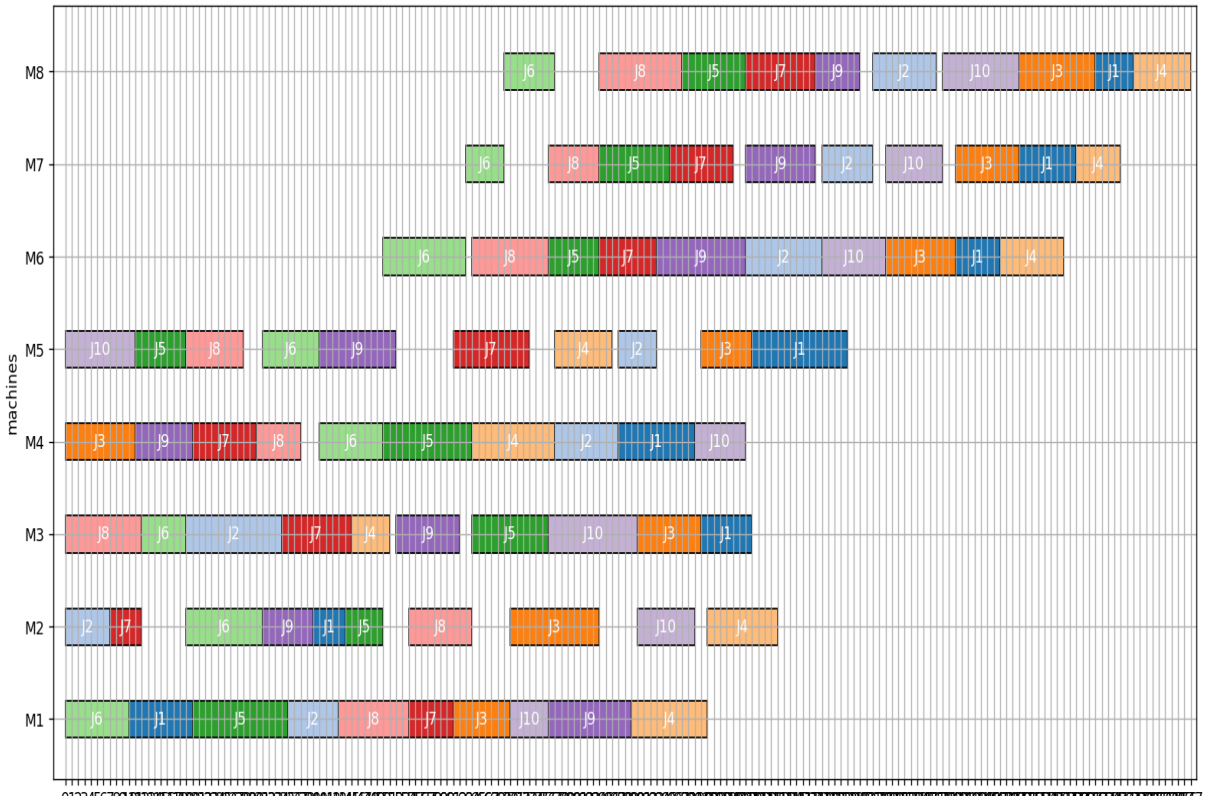
"M1[10] -> M3[7] -> M2[12] -> M5[9] -> M4[10] -> M6[13] -> M7[6] -> M8[8]",

"M2[5] -> M4[10] -> M3[11] -> M1[7] -> M5[12] -> M6[9] -> M7[10] -> M8[11]",

"M3[12] -> M5[9] -> M4[7] -> M1[11] -> M2[10] -> M6[12] -> M7[8] -> M8[13]",

"M4[9] -> M2[8] -> M5[12] -> M3[10] -> M1[13] -> M6[14] -> M7[11] -> M8[7]",

"M5[11] -> M1[6] -> M3[14] -> M2[9] -> M4[8] -> M6[10] -> M7[9] -> M8[12]"



The minimum makespan among all makespan is 177 so it is the optimal solution

Additional explanations. this is how we figured out the project you can consider not reading this:

Let's start dividing the problem into smaller tasks:

we have

- operations
- jobs
- Machines

operation	Job	Machines
• joblD • machinelD • processingTime	• joblD • operations	• machinelD • Schedule

➢ **Now we need to link these parts together we'll use a function to do this job.**

Let's assume these inputs:

- M1[10] → M1[2] → M1[10]
- M2[3] → M3[1] → M1[8]

This function will take these descriptions and turn them into objects, we need it to create the machines used and generate the jobs.

- **1st_loop:**

Let's start with a loop to iterate over descriptions:

This loop will have the first element of it as: **M1[10] → M1[2] → M1[10]**

- **2nd_loop:**

We create a new job using the data gathered from the 2nd_loop, and add it to the jobs list. Finally move to the next job by incrementing the job_id by 1.

Then we need another loop to iterate over operations which are jobs assignments and processing times splinted by **(→)**. Assuming the top jobs has its first element: **M1[10]**

2nd_loop: of **job[i]**

We need to take the machine which is machine 1 and the time which is 10, let's split them by ' **[**' we will have ['M1', '10]'], Then let's take the first index of M1 which is 1 then we subtract it by 1 to start indices from 0 **(machine_id).**

After that we need to take the time here:

All index before the last one, since the output would be 10 **(processing_time)**, Then we have all data we need to create an Operation and a Machine, we create these 2 objects and we add them to our lists, Finally after the 2 loops finishes their work we take the machines from the set of the machines to the list of the machines and return the generated jobs and machines

Now let's get started with the GENETIC ALGORITHM

❖ GENETICALGORITHM

The genetic algorithm needs the jobs and the machines to work with, also there are some parameters to consider:

- A limit for the number of generations
- Number of individuals in the population
- Mutation rate

As known in the genetic algorithm we need an initial population so let's implement a function to generate the initial population:

❖ generateInitialPopulation

This function needs the number of individuals as a limit for the generation and the set of jobs to work with. We need a loop that runs numberOfIndividuals times. In each run, we need to create an individual list. For each individual, we need to iterate over the jobs and append all of the operations of that job to this individual list. After that, we will move the operations in that individual list, which is the same thing to do in the next population. Then, we need to use the shuffle function from the random library. Don't forget that the shuffle gives unique random mixes. Finally, we append that individual to the population list that we want to return.

Now we have the initial population ready. Next, we need to assume the best individual is to be reset and the fitness to be infinity. We have to get the number of solutions to keep as the top number of solutions. After that, a loop running for the number of generation times is required.

In each run, we need to evaluate the fitness using a function.

❖ getFitness

Let's get started with this function, the function takes the **job_id** and the list of machines as input.

- First thing first we need to clear the schedule for all the machines to make new schedules.
- Now we need to initialize some lists to track the objects.
- Then we need to set the list of operations we have.
- After that, we need a loop to schedule all operations in that list:
    - This loop needs to have a flag to know if the operation is scheduled or not. It will be false initially.

- Then, for each operation, we need to check if the operation is the next in the sequence for the current job. If so, declare the operation as scheduled.
- If this operation's index of the current job is less than the number of operations for the current job, which means there are still operations to be scheduled, and if the next available operation of the current job is the same as the operation we are working on:

    - If this is the next sequence, we need to now check the start time. We take the larger one of the current machine available time and the times when each operation stops.
    - We also need to decide the end time, which is simply start time + processing time of the current operation.
    - Then, if the start time is the same as the available time of the current machine, we need to update that machine by appending the operation, end time, and start time to the schedule of that machine.

- Now we update the tracking lists of when machine is available and the end times of the current job.
- Then we move to the next operation in the job and remove the old one and update the operation scheduling flag to be true and finally we leave the loop\
- If there were no operations scheduled, an extra consideration is to ensure we process machine idle. To do that:
    - We need to check the available time which will be the least available time of the machines.
    - Now we fill all the machines if the machine available time is the same as the next available time we get. We add a time unite to that machine available time.

- Finally, we return the largest available time from all the machines.

We can use this function to evaluate the fitness for all the individuals, now we need to create a new population using crossover and random mutation, we need to make a loop that runs the half of the number of individuals. Which will select 2 parents according to the fitness values.

❖ Selection:

The function which selects the 2 parents is very simple. We need to sum all fitness's and make a list of probabilities each one is made by inversion of dividing each fitness over the sum of all fitness's then we select 2 individuals using the choices function tie selection depends on the probabilities calculated before.

Now that we've selected the parents, we need to do the crossover between them.

❖ Crossover:

The crossover functions will be designed like this:

- We need to take the number of operations each (parent) solution has.
- Then we need to select 2 random different crossover points within the size range to ensure that the first point is less than the second point.
- Now we will initialize 2 zoning children, which will have the same size and have null values.
- The first child will have values from parent one inserted into indices between the points, and the second child will have values from parent two inserted into indices between the points.
- To fill the remaining positions, we will initialize filling positions as the point we stopped at.
- We will iterate over all the operations:
    - For child 1, if the element from parent 2 is not already in the copied segment, it is placed in the next available position.
    - For child 2, if the element from parent 1 is not already in the copied segment, it is placed in the next available position.
    - Then move in the filling positions.

- Finally, it returns the new children.

And then the random mutation for the new children, we make the mutation using a function:

❖ Mutation:

- This function will generate a random number. If that number is less than the mutation rate, we take 2 random operations of the child and swap their order.
- Then we consider the new population to work with, which is the new population generated via crossover and mutation.
- We will take the best fitness of the current population. If this is better than the old best fitness of the old population:
    - We will take it as the new best fitness.
    - We will consider the best individual as the one with that best fitness.

- Finally, we return the best fitness and the best individual.

Now we have the schedules filled successfully, we need to plot the results. Let's use pyplot and patches libraries.