Modelling, Simulation and Optimisation TABA

1st Steven Travers - X23299525
NCIRL Modelling, Simulation & Optimisation TABA
Postgraduate Diploma in Science in Data Analytics (PGDDA)

***Abstract*—The aim of this project is to identify which of two algorithms, the Greedy and Monte-Carlo algorithm, provide an optimal solution to a job scheduling problem. The job scheduling problem involves a set number of jobs (orders) that must be completed. The optimal solution is the solution in which minimises the total time to complete all the jobs. Given 4 machines, algorithms must be tested to identify which algorithm gives the optimal / closest to optimal solution. To do this, these algorithms will be tested for various workloads (job counts) and the results will be compared with an integer programming optimal solution. To find the optimal solution, the variation of these algorithms will be implemented in Python and several evaluation techniques will be utilised. The final algorithm that will be provided, should show as close to optimal as possible results and variations for future work will be provided to further optimise the algorithm / the process.**

***Index Terms*—Optimisation Algorithms, Scheduling, Python Optimisation Implementation**

I. Introduction

NCI Woodwork Limited is a small carpentry company that specialises in producing different bespoke household furniture such as tables, chairs & frames. A recent expansion, which has allowed for the purchase of 4 new machines, gives rise to a new issue for NCI Woodwork. Previously NCI Woodwork have handmade each order and managed the workload manually. They now want to find a way to utilise the machinery to minimise the time taken to manufacture each customers orders. To do this, several algorithms will be tested against a LP(Linear Programming) solution and a algorithm will be provided to NCI Woodwork. This algorithm should aim to minimise the total time taken to produce a customers full order. Throughout this project, there are a small number of assumptions that are made. These assumptions include:

- Every specific order must go through each respective machine in the correct order. i.e. From machine 1, to machine 2, to machine 3, finishing with machine 4.
- Each job must be completed for the given machine before progressing to the next machine, i.e. machines cannot run concurrently.

Given that these orders are bespoke, the time taken for each order varies for each machine. In order to generate a simulation of this data, a "GenerateData" function in python was implemented. This function was provided by NCI woodwork and can be downloaded from Moodle [1]. This function generates an array of data that can be used to replicate a set of N customer orders to NCI Woodwork. The output of this function returns a set number of orders and time taken for each machine to produce each part of an order. Throughout this project, various statistics and methods of evaluation will be used. Given that the task is to find a suitable algorithm which will help NCI Woodwork to minimise the length of time to produce daily orders, the first comparison that will be made is to identify the integer programming solution for the respective orders. From here, the various algorithms used will be compared to this number as this is the "Optimal" solution. To ensure a valid simulation is conducted, various values for customer orders will be produced. For this project, the amount of machines will be limited to 4. This is because each machine plays a unique role in the manufacturing process and each machine is needed for each order. Other variations will be discussed in a "Future Work" Section.

II. Algorithms

A. Greedy Algorithm

The first algorithm that will be discussed is the Greedy Algorithm. The premise for the greedy Algorithm is that it chooses the locally optimal solution at each stage. This means that it ignores information from the future and only thinks of the optimal solution locally.

There were several reasons for deciding to go to the greedy algorithm as the first comparison algorithm. These reasons include that the greedy algorithm is extremely easy to implement, easy to understand and it is not very resource intensive. The version of the greedy algorithm that was implemented for this project was very easy to implement. It involves getting the sum of time each job (order) takes to complete and using this as the basis for the greedy calculation. Since the greedy algorithm chooses the locally optimal solution, the "optimal" had to be defined. For this, the minimum time for a job was used. To implement this solution, a simple pandas dataframe was used to get the sum of each jobs time. From here, they were sorted in ascending order from shortest to longest and the index of the dataframe was taken as the sequence which was provided to the "TotalTime()" function. This ensures that the order in which the jobs are ran are "locally optimal" meaning the shortest job is ran first, then the next shortest and so on. Another reason that the greedy algorithm was chosen was its simplicity to understand. Understanding some concepts of optimisation can be difficult. This is one of the easiest algorithms to understand and therefore was used as the basis of the first algorithm to compare.

job #	Mach. 1	Mach. 2	Mach. 3	Mach. 4	Total Time
0	3	5	5	6	19
1	4	8	5	4	21
3	6	5	9	2	22

TABLE I
EXAMPLE DATAFRAME FROM GREEDY

As can be seen in Table I, the order of the jobs generated (Job #) is gotten by sorting the values from smallest to largest on the total time. There is the option to also sort each machine smallest to largest either. The output of this algorithm is to perform the jobs in order [0,1,3,4,2,5,6] for the case with 7 jobs. When this is complete, the "TotalTime()" function is called, passing the job order. This returns the time in units that the entire job list takes to run. For this algorithm, given an input of 7 jobs, the total time to run is 65 Hours.

The next step in this analysis is to compare this to the integer programming (IP) solution. To do this, there were 2 comparisons completed. The first analysis done was to generate both the IP solution and the greedy solution for a variety of job counts. Due to resource and time constraints, the maximum job counts that was analysed is 50.

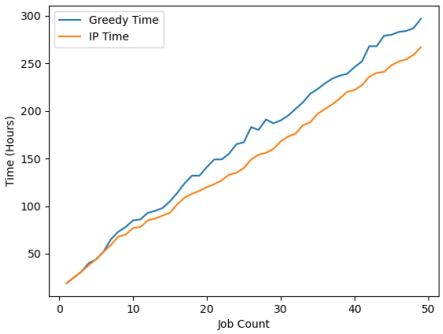

Fig. 1. Greedy vs IP Solution on job counts varying from 1-50

As can be seen in Figure 1, it appears that the greedy and IP solutions are quite different. Using the Greedy Comparison excel, it can be seen that the difference is approx 11% more in the greedy solutions time to complete the jobs. This variance also looks to increase over time. From this graph, the next, and similar analysis done was to test the difference in IP and Greedy over time.

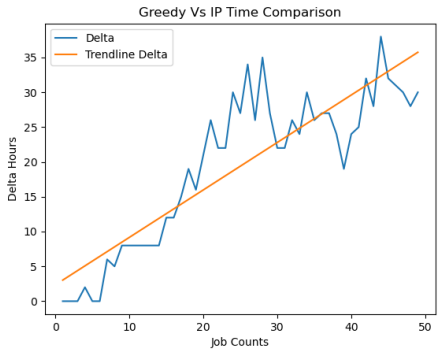

Fig. 2. Delta Comparison Greedy vs IP

Figure 2 indicates the delta between IP and greedy. An important point to note here is the increase in difference as the job count increases. This is important as if higher job count simulations were ran, the greedy algorithm is insufficient to use. A Line of best fit was added to Figure 2, this was to give a very rough approximate value for higher job counts. "greedy_implementation.ipynb", cell 30 indicates that if this line of best fit was extrapolated to 1,000 job counts, the estimated delta between greedy and IP solution would be approximately 683 hours.

This leads to one of the cons of using greedy algorithms, they may not always provide the globally optimal solution [2]. Greedy algorithms in general prioritise local optimal solutions over global optimal solutions. For this reason, there are potentials for more optimal solutions to be found by using another algorithm.

B. Stochastic Algorithm

The next algorithm that will be investigated is the Monte-Carlo Stochastic algorithm. The Monte-Carlo algorithm relies on repeated random sampling to obtain results [3]. This method was chosen over other algorithms such as the Brownian Motion version of the Random Walk algorithm due to the fact that there is no "cost in travelling", i.e. there are no constraints to say that testing x amount of job combinations is unfeasible.

There were two versions of this algorithm implemented. The main difference between Monte-Carlo Version 1 and Version 2 is the shuffling algorithm used. The pseudo code for the Monte-Carlo algorithm implemented is as follows:

1) Initialise starting order - A random list of jobs equal to the length of the total jobs where no values repeat.
2) Calculate the initial time taken to complete all jobs with this order
3) Generate a new job order by shuffling the list*
4) Calculate time to complete jobs with this new sequence
5) Compare this new time with the time from step 2
6) Take the smaller of times from the calculation in step 5**
7) repeat steps 3-6 N times, returning smallest time found

The code for implementing this can be seen in block 6 of "Monte-Carlo-Optimisation.ipynb". Step 3, highlighted with

an asterisk above, is the step in which there is a difference between Monte-Carlo Version 1 & 2. For version 1, the shuffling algorithm that is used is the "Fisher–Yates shuffle Algorithm". This shuffling algorithm is unbiased - meaning that each possible outcome is equally likely [4]. This method was chosen from a variety of shuffling techniques due to its time and space complexity. The code for this shuffling algorithm is simple and can be found on geeksforgeeks [4]. This shuffling algorithm uses the current order and swaps the index of some of the list elements to create a new list. The method used in V2 comes from the sorted function which allocates all the list elements randomly.

Another variable that was decided was the number of random samples / shuffles to take, marked with a double asterisk (**) in the pseudo code above. This was hard coded at 10,000 to ensure that the resources needed were not extravagant while also ensuring there was a relatively large sample size. This can be seen in Variable N in the Monte-Carlo function. A similar analysis was done on the Monte-Carlo simulation as was done on the Greedy simulation.

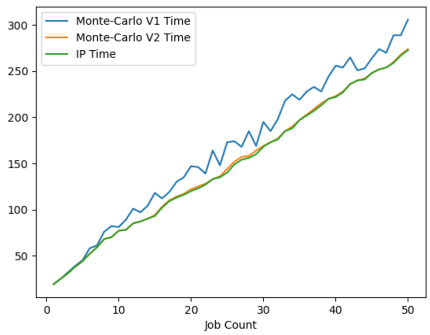

Fig. 3. Monte-Carlo Variants vs IP Solution

Figure 3 and Figure 4 both highlight the variance between both versions of the Monte-Carlo algorithm and the Integer programming solution. It is extremely clear that the variants of the Monte-Carlo problem both show vastly different optimal solutions for the various job lengths. An important note is that the line of best fit for both algorithms, which although it is not an exact value, does show that when the job count increases, version 1 gets further away from the ip solution, while version 2 stays quite consistent, with some small variance for some job counts. This small variance is to be expected when using random numbers. This could change when ran for different seed values.

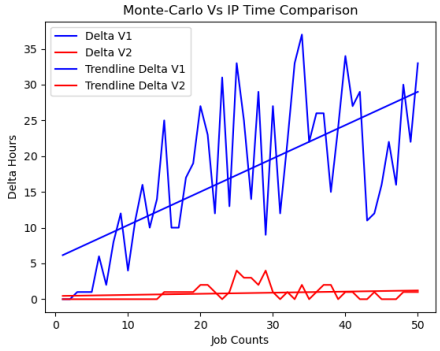

Fig. 4. Greedy vs IP Solution

III. Evaluation

In order for these algorithms to be tested, as mentioned previously, an Integer Programming solution was created. This gives the optimal solution for the generated job/order schedule. The function "TotalTime()" provided from previous class notes [1], generates the total time taken for each algorithm. This will be used to compare the times for the Integer programming solution and the two tested algorithms.

Another metric that is needed when comparing algorithms is how well each algorithms scales in respect to job counts. In order to calculate this, each algorithm was extrapolated using the line of best fit. Due to resource constraints, the time taken to run larger simulations with larger job counts increases exponentially. The optimal time found for each algorithm can be seen in Table II.

Job Count	Δ Greedy	Δ MC 1	Δ MC 2
1,000	683	471	16

TABLE II
Extrapolated results using Line of Best Fit for all Algorithms used, showing the expected relative Δ's to the IP solution for each algorithm

These extrapolated results are important as they give an indication of how well the algorithms will perform at higher counts. Although this may not be necessary for NCI Woodwork currently, with future expansion this is where the significance of the variance at higher counts will be relevant. A key reason that an algorithm such as the MC2 algorithm can be utilised is trying to find solutions to larger problems. For example, when choosing 1,000 jobs, the IP code was allowed to run for nearly 2 hours. In this time the code was still running. However, when running the MC2 algorithm, with 10,000 random samples, the time taken was 3 minutes. Using the extrapolation table above, it can be noted that the expected delta is approx 16 hours difference to the IP solution. For a resource saving of circa 2+ hours, this approximation would be a better alternative to calculating the optimal solution via IP. [Note: laptop spec may impact actual run time so pictures included in .zip file (output_figures folder)]

To summarise these findings, there is significant difference between greedy and random algorithms. The greedy is the

worst performing algorithm, with a steep increasing trend in Δ to IP solution with increasing job counts. It would have been respectable to assume that this algorithm would find a better solution than the Monte-Carlo methods due to their randomness, but with a large enough sample, the MC methods are superior. Not only did the Monte-Carlo v2 method provide superior results, it has also given the results in a fraction of the time.

This would lead to the conclusion that for the algorithms tested, the Monte-Carlo V2 method would be suggested to NCI Woodwork as the job scheduling algorithm. However, with more time, other algorithms could also be tested but for the scope of this project, only 2 were tested.

IV. Future Work

Due to time constraints, there were many areas of this project that could not be further analysed/ scenarios that would have been particularly interesting to test and see where further improvements could be made. This would have led to the opportunity to go back to NCI Woodwork with potential further improvements to their current systems / workflows.

An area which was not explored throughout this project is the opportunity to potentially buy another machine for NCI Woodwork. As mentioned previously, for this project, the main focus was on keeping the number of machines to 4. However, an area of possible exploration is to increase the number of machines. A possible reason for increasing the machine would be to find out which machine produces a "bottleneck" in production and to potentially buy another one of these machines. In the "GenerateData()" function, randomness was used to simulate the time taken for each job. In the real world, there is a higher possibility that a certain machine takes longer than the rest. In order to show an example of this, the generate data function was modified and the time for each machine was generated for 1,000 orders.

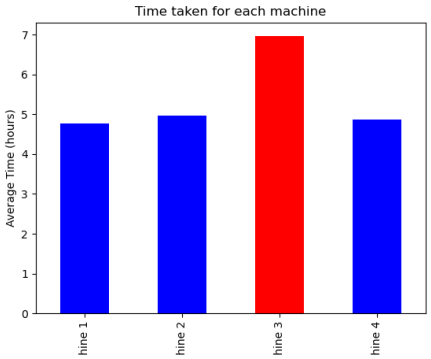

Fig. 5. Real world example of machine 3 taking longer to produce order

As can be seen in Figure 5, combined with Table III, Machine 3 (red) takes nearly 2 hours longer to produce the order. This may lead to NCI Woodwork either upgrading / buying another machine 3. These are key insights that can be provided by running simulations. Code used for this simulation can be found in "future_work_data_example.ipynb".

Machine	Mean Time (hours)
1	4.76
2	4.97
3	6.96
4	4.86

TABLE III
MEAN PRODUCTION TIME REAL WORLD EXAMPLE

Testing the Monte-Carlo variations highlighted that there is a significant difference attained by using different sorting algorithms. This was unexpected, and mainly highlights that there can be a disadvantage to using certain sorting algorithms to go through random solutions. However, this also leads to the possibility of using Machine learning to identify other opportunities to improve the order in which the sequences are shuffled. This could return a potentially better solution / at worst case, a similar solution to variant 2, using a smaller N value and thus decreasing computational power which is needed for higher job counts.

Another area worth future exploration is the idea of a new manufacturing process where NCI Woodwork can work on different parts of a customers order concurrently. This means while machine 1 is running for the first order, NCI Woodwork could potentially begin working on machine 2 for the same order. This will help to reduce the wait time for each machine. This in turn, will help reduce the overall time taken for each order significantly. Due to time constraints, this process could not be implemented for this project, however, it would be interesting to analyse what impact this has on overall manufacturing times. There would need to be new variables added such as "Time taken to combined piece from Machine 1 and 2". Where, for example, another small piece must be manufactured in order to join two previous pieces that were made one after the other in the current process.

V. Other

Link to code and pdf in GitHub

References

[1] C. Horn, "Christian horn mso class notes." [Online]. Available: https://moodle2023.ncirl.ie/course/view.php?id=1999

[2] S. Khuller, B. Raghavachari, and N. E. Young, "Greedy methods," in *Handbook of Approximation Algorithms and Metaheuristics*. Chapman and Hall/CRC, 2018, pp. 55–69.

[3] GeeksforGeeks, "Pros and cons of decision tree regression in machine learning," Feb 2024. [Online]. Available: https://www.geeksforgeeks.org/pros-and-cons-of-decision-tree-regression-in-machine-learning/

[4] ——, "Python: Ways to shuffle a list," Jul 2023. [Online]. Available: https://www.geeksforgeeks.org/python-ways-to-shuffle-a-list/