IE443 - Multi-Objective Decision Analysis
Term Project

Zeynep Altıner - 21702167
Ayşe Beyza Çanakçı - 21901849
Simge Çınar - 21901465
Instructor: Assoc. Prof. Özlem Karsu

May 26, 2023

1 Introduction

The report focuses on the bi-objective programming model provided in the paper named "A multi-objective volleyball premier league algorithm for green scheduling identical parallel machines with splitting jobs"[1]. Throughout the report, we implement the ϵ - Constraint Algorithm to the presented model, post-process the weakly nondominated points on Python and examine the Pareto-Frontier.

2 Problem Definition and Mathematical Model

The paper that we worked on centers on solving a scheduling problem with parallel machines and splitting jobs. The scheduling problem is an NP-hard problem. In the literature, various meta-heuristic algorithms have been developed to solve NP-hard problems, several of which are Genetic Algorithm, Particle Swarm Optimization Algorithm, and Evolution Algorithm. To be utilized in the bi-objective optimization context, these algorithms have been modified into Non-dominated Sorting Genetic Algorithm II (NSGA II), Multi-Objective Particle Swarm Optimization (MOPSO), and Multi-Objective Evolutionary Algorithm. The paper runs the Multi-Objective Volleyball Premier League Algorithm on the proposed scheduling model and compares the results with the aforementioned algorithms. We, however, focus on running the ϵ-Constraint Algorithm to find an exact solution to a small-scale problem.
Before getting deep into the model, it should be noted that we had to modify the model by changing existing constraints and by adding new ones, as the original form of it resulted in infeasibility. Hence, the below explanations will follow the modified version of the model seen in Appendix B. The original form of the model can be viewed in Appendix A.
The first objective focuses on minimizing total tardiness, whereas the second one aims to minimize sequence-dependent waste during production. The rationale behind the combination of these objectives is that targeting to reduce delays (tardiness) leads to greater use of resources such as machines, labor, space, and materials. To give an example of a sequence-dependent waste, producing a white-colored product just after a black-colored one on the same machine could have reduced tardiness but would certainly increase the number of defectives. Constraint (1) assigns the sum of the number of job i processed in different machines to the total amount of job i to process. It is worth noting that the model assumes that when job i is assigned to a machine k, for example, there are 7 of job i's in total and 1 of them has been assigned to the 2^{nd} machine, then the whole process is done by this machine, and this 1 job cannot be fragmented to be processed on different machines. It also assumes that all of the machines are available throughout the time horizon. Constraint (2) ensures that when some amount of job i (e.g., 5 out of 10) is assigned to machine k, then y_{ik} must be 1. On the other hand, constraint (3) ensures that when there is no job i assigned to machine k, then y_{ik} must be 0. Constraints (4) and (5) are the modified version of the constraints (4) and (5) of the original problem. Constraint (4) ensures that if job j is processed in machine k, either it is the first job or there is another job that comes before job j in machine k. Similarly, constraint (5) ensures if job i is processed in machine k, either it is the last job, or there is another job that comes after job i in machine k.

Constraints (6) and (7) are the modified versions of the constraint (7) in the model. (6) considers the case when job i is the first job on machine k, then there is no setup and completion time based on a previous job. Thus, in this constraint, S_{ij} and C_{ij} are assigned to 0. On the other hand, (7) takes into account the rest of the cases, that is, when a job i is assigned to machine k, and job j comes after it, then there is a setup time S_{ij} and a completion time of job i, C_{ij}. The model assumes identical machines. Constraint (8) implies that tardiness is the difference between the completion time and the due date of job i. At this point, it is worth noting that the model assumes that all job i's are to be processed until the same deadline; that is, there are no customers waiting for the same products within different time periods. Constraint (9) stands for preventing the case when i and j are equal to each other, as a job cannot come after itself. It assumes that if n amount of job i is assigned to machine k, then all must be processed consecutively, as a batch. Constraint (10) ensures that the parameter a_k (amount of defective output of machine k) is the sum of sequence-dependent wastes if job j comes after job i on machine k. It is worth mentioning that at first glance, a_k being a parameter sounded counter-intuitive. However, it then seemed rational that the model assumes to limit the number of defective outputs a machine can produce to keep it working without a need for maintenance. It might also be the case that there has been a statistical analysis on the number of defectives a machine can produce. (11) guarantees that the total number of machines that could be used to process job i is adequate for the total number of machines job i is assigned to machine k. It is worth noting that to prevent discrepancy in results, we had to add constraints (12) and (13), which imply that there has to be exactly one last and first job on machine k. When we remove this constraint, the model assigns different jobs as the first or last jobs on the same machines. (14) is the modified version of constraint (6) as we had to prevent infeasibility. By this constraint, we had to make the assumption that each machine must be used to produce at least one job. As another discrepancy, the original version of the model resulted in assigning a completion time to job i on machine k even though it was not processed on machine k, that is, y_{ik} and C_{ik} were not consistent. Thus we needed to add constraints (15) and (16). By constraint (15), we ensured that C_{ik} takes value 0 as y_{jk} takes value 0, and by constraint (16), we guaranteed that C_{ik} takes a positive value as y_{ik} gets 1.

3 Findings for the ϵ - Constraint Algorithm

The reason why we have chosen the ϵ-Constraint Algorithm as our scalarization approach is that, in an integer programming setting, with the Weighted Sum (WS) Algorithm, we could miss some of the points on the Pareto Frontier as its original form finds only the supported non-dominated points. We knew that integrating additional models into the WS Algorithm could solve the problem. However, as the ϵ - Constraint Algorithm, by nature, finds all of the nondominated points, including the unsupported ones, we found it more efficient to implement. You can find the modified model to be utilized in the algorithm below. We have chosen objective 2 to be minimized, and constrained objective 1.

$$\text{Min} \sum_{i \in I, j \in J, k \in K} x_{ijk} \cdot Wij$$

subject to

$$x \in X \tag{1}$$

$$\sum_{i \in I} Z_i \leq \epsilon_1, \quad i \in I \tag{2}$$

Firstly, it is worth noting that we had to be careful while initializing ϵ, as having a too-small value, in a minimization context, would result in infeasibility, and a too-high value would bring non-value added iterations until reaching all non-dominated points. The second case is of great importance as it would be more costly to have non-value added iterations in an integer programming context as it is already computationally more complex to solve compared to linear programming models. Hence, we have to do a pre-analysis before initializing the parameters. When we first ran the model with an ϵ value of 50, we observed that until the non-dominated point (32,2), the value of objective 1 has decremented by one to the value of 32. Considering this, initializing ϵ with a value of 50 would result in making 49 iterations, 18 of which turned out to be non-value added (resulting in 31 iterations). You can observe the non-value added iterations presented with points indicated in blue in Appendix C. Thus, we decided to initialize the ϵ with the value of 32 and prevented the above-mentioned iterations. In addition to this, we questioned whether increasing the step size to a value greater than 1 would prevent unnecessary iterations without missing a nondominated point. Increasing its value by 1 **did not result in losing** some of our nondominated points in the Pareto Frontier. Hence, we fixed the initial value of ϵ to 32 and the step size to 2. This was because the $z(1)$ values differed from each other by, at most, a value of 2. This helped us prevent 15 non-value added iterations and reduced the resulting number of iterations to 16. However, when we increased the step size to 3, we lost one of the points with coordinates (8,5) in the Pareto Frontier, which can be observed in Appendix D. Our next step was to define the model as a function of ϵ. Then we created a while loop via which we could reduce the value of ϵ by 2 (step size) and solve the model iteratively and return the values of objective 1 and objective 2 so as to plot the weakly nondominated points in the objective space, as seen in Appendix E. Hence, we needed to post-process our results via Python to achieve the non-dominated points indicated in red in Appendix F. By post-processing, we eliminated the points whose $z(x) - R_Z^2$ set was **not** empty and achieved the set of non-dominated points, that is, the Pareto Frontier, seen in Appendix G. It seems that all the points in the Pareto Frontier turned out to be extreme supported, except for the one indicated with coordinates (14,4).

Set of Weakly Nondominated Points

Z_{WN} = {(32.0, 2.0), (30.0, 3.0), (28.0, 3.0), (26.0, 3.0), (24.0, 3.0), (22.0, 3.0), (20.0, 3.0), (18.0, 4.0), (16.0, 4.0), (14.0, 4.0), (12.0, 5.0), (10.0, 5.0), (8.0, 5.0), (6.0, 8.0), (4.0, 8.0), (2.0, 11.0)}

Set of Nondominated Points

$Z_N = \{(32.0, 2.0), (20.0, 3.0), (14.0, 4.0), (8.0, 5.0), (4.0, 8.0), (2.0, 11.0)\}$

In addition to this, to determine if these non-dominated points were supported or unsupported, we utilized the *scipy.spatial* library's [2] *ConvexHull* module and drew a convex hull with these points on Python. Even though we are aware that the whole points on the objective space were not present to draw the whole convex hull, it helped us in making a convenient analysis of the points in terms of supportedness. It resulted in all of the points in the Pareto Frontier being supported as they were on the convex hull, as seen in Appendix H. In this choice of parameters, the section from the convex hull and Pareto Frontier were the same; yet, we observed that for a different choice of parameters, the section of the convex hull would not be the same as the Pareto Frontier and all of the points in the Pareto Frontier would not result in being supported, as seen in Appendix I.

4 Discussions on Results

In Appendix J, you can observe the efficient solutions corresponding to the image vectors indicated in the Pareto Frontier. You can also see the ϵ values attained through the iterations at which the solution vectors were obtained. Primarily, it is worth noting that as tardiness reduces, the model tends to split jobs as much as possible; that is, in Figure 11, job 4 is split into machine 2 and machine 3 with a tardiness value of 8. In Figure 12, the tardiness value reduces to 4, and job 4 is split into **3** machines. Consequently, a decrease in tardiness amplifies the impact of splitting jobs, which in turn, results in increasing the total number of defective products. Considering Figure 8, to reduce the total number of defective products, the model tends not to split any jobs. Hence, the Q amount of each job is processed in the same machine. As a result, the target of decreasing the total waste results in preventing the splitting of jobs into different machines. These trade-offs can also be observed in the Pareto Frontier.

4.1 Computational Time Analysis

You can observe that the run time increases exponentially as the number of jobs and machines increases. 50 jobs and 15 machines caused the algorithm to run above 1 hour, where we expect a far higher number of jobs and machines in a realistic production environment. Hence, it might be interpreted that running the ϵ - Constraint Algorithm on an NP-hard problem is limited by the computational time/load constraint. You can refer to Appendices K and L. This result goes parallel with the main intention of the paper, which is implementing adjusted meta-heuristic algorithms to multiple-objective settings in case of scaling up the problem.

5 Appendices

5.1 Appendix A: Model in the Paper

Indices:
M : Set of machines indexed by $k = \{1, .., m\}$
N : Set of jobs indexed by i and $j = \{1, .., n\}$

Parameters:
MO_i: Total number of machines that can be used to perform job i
P_i : Processing time of job i
Q_i : The amount of jobs
$S_{i,j}$: Setup time in case job j comes after job i
d_i : Due data of job i
c_{0k} : Completion time of first job on the machine k.
Wij: Sequence dependent setup defective output of job j if it comes after job i.
a_k: Amount of defective output of machine k.

Decision Variables:

$$x_{0jk} = \begin{cases} 1, & \text{if job } j \text{ is the first job on machine } k \\ 0, & \text{otherwise} \end{cases}$$

$$y_{ik} = \begin{cases} 1, & \text{if job } i \text{ is processed on machine } k \\ 0, & \text{otherwise} \end{cases}$$

$$x_{i0k} = \begin{cases} 1, & \text{if job } i \text{ is the last job to be processed on machine } k \\ 0, & \text{otherwise} \end{cases}$$

$$x_{ijk} = \begin{cases} 1, & \text{if job } j \text{ comes after job } i \text{ on machine } k \\ 0, & \text{otherwise} \end{cases}$$

Z_i: Tardiness for job i
q_{ik}: Amount of job i on machine k

Model:

$$\text{Min } z(1) = \sum_{i \in I} Z_i$$

$$\text{Min } z(2) = \sum_{i \in I, j \in J, k \in K} x_{ijk} \cdot Wij$$

subject to

$$\sum_{k \in K} q_{ik} = Q_i, \quad i \in N \tag{1}$$

$$y_{ik} \geq \frac{q_{ik}}{Q_i}, \quad i \in N,\ k \in M \tag{2}$$

$$y_{ik} \leq q_{ik}, \quad i \in N,\ k \in M \tag{3}$$

$$\sum_{j \in N} x_{ijk} = y_{ik}, \quad i \in N,\ k \in M \tag{4}$$

$$\sum_{i \in N} x_{ijk} = y_{ik}, \quad i \in N,\ k \in M \tag{5}$$

$$\sum_{i \in N} y_{ik} \leq 1, \quad k \in M \tag{6}$$

$$(x_{ijk} \cdot S_{i,j}) + C_{ik} + (q_{jk} \cdot P_j) \leq C_{jk}, \quad i \in N,\ j \in N,\ k \in M \tag{7}$$

$$C_{ik} - d_i \leq Z_i \quad i \in N,\ k \in M \tag{8}$$

$$c_{0k} = 0 \quad k \in M \tag{9}$$

$$x_{ijk} \in \{0, 1\} \quad j \in N,\ k \in M \tag{10}$$

$$\sum_{i \in N, j \in N} x_{ijk} \cdot Wij \leq a_k \quad k \in M \tag{11}$$

$$\sum_{k \in M} y_{ik} \leq MO_i \quad i \in N \tag{12}$$

$$x_{iik} = 0; \quad i \in N,\ k \in M, \tag{13}$$

$$q_{ik} \geq 0; C_{ik} \geq 0; Z_i \geq 0; y_{ik} \geq 0; \tag{14}$$

5.2 Appendix B: Modified Model

Sets:
I: $\{1, .., 5\}$ defined as the set of jobs.
J: $\{1, .., 5\}$ defined as the set of job.
K: $\{1, 2, 3\}$ defined as the set of machines.

Parameters:
a_k= [20,50,40]
M = 100000
MO_i = [2,2,1,3,1]
P_i = [4,7,5,6,1]
Q_i= [3,4,5,7,9]
d_i = [20, 60, 50, 20, 30]

$$S_{ij} = \begin{bmatrix} 3 & 2 & 5 & 1 & 5 \\ 4 & 1 & 5 & 3 & 5 \\ 1 & 1 & 6 & 1 & 3 \\ 1 & 3 & 3 & 1 & 5 \\ 3 & 1 & 6 & 1 & 3 \end{bmatrix}$$

$$W_{ij} = \begin{bmatrix} 0 & 1 & 3 & 2 & 5 \\ 3 & 0 & 8 & 1 & 5 \\ 2 & 1 & 0 & 1 & 3 \\ 3 & 1 & 5 & 0 & 4 \\ 7 & 1 & 5 & 1 & 0 \end{bmatrix}$$

Decision Variables:

They are the same as the original problem.

Model:

$$\text{Min } z_1 = \sum_{i \in I} Z_i$$

$$\text{Min } z_2 = \sum_{i \in I, j \in J, k \in K} x_{ijk} \cdot Wij$$

subject to

$$\sum_{k \in K} q_{ik} = Q_i, \quad i \in I \tag{1}$$

$$y_{ik} \geq \frac{q_{ik}}{Q_i}, \quad i \in I,\ k \in K \tag{2}$$

$$y_{ik} \leq q_{ik}, \quad i \in I,\ k \in K \tag{3}$$

$$x_{0jk} + \sum_{i \in I} x_{ijk} = y_{jk}, \quad j \in J,\ k \in K \tag{4}$$

$$x_{j0k} + \sum_{j \in J} x_{ijk} = y_{ik}, \quad i \in I,\ k \in K \tag{5}$$

$$x_{0jk} \cdot (q_{ik} \cdot P_i) \leq C_{ik} \cdot y_{ik}, \quad i \in I,\ k \in K \tag{6}$$

$$x_{ijk} \cdot (S_{i,j} + C_{ik}) + (q_{jk} \cdot P_j) \leq C_{ik} \cdot y_{jk}, \quad i \in I,\ j \in J,\ k \in K \tag{7}$$

$$C_{ik} - d_i \leq Z_i \quad i \in I,\ k \in K \tag{8}$$

$$x(iik) = 0 \quad i \in I,\ k \in K \tag{9}$$

$$\sum_{i \in I, j \in J} x_{ijk} \cdot Wij \leq a_k \quad k \in K \tag{10}$$

$$\sum_{k \in K} y_{ik} \leq MO_i \quad i \in I \tag{11}$$

$$\sum_{j \in J} x_{0jk} = 1 \quad k \in K \tag{12}$$

$$\sum_{j \in J} x_{j0k} = 1 \quad k \in K \tag{13}$$

$$\sum_{i \in I} y_{ik} \geq 1 \quad k \in K \tag{14}$$

$$C_{ik} \leq M \cdot y_{ik} \quad i \in I,\ k \in K \tag{15}$$

$$C_{ik} \geq y_{ik} \quad i \in I,\ k \in K \tag{16}$$

$$x_{0jk} \in \{0,1\}, x_{j0k} \in \{0,1\} \quad j \in J,\ k \in K \tag{17}$$

$$x_{ijk} \in \{0,1\} \quad i \in I,\ j \in J,\ k \in K, y_{jk} \in \{0,1\} \quad i \in I,\ k \in K \tag{18}$$

$$q_i \geq 0 \quad i \in I,\ k \in K, Z_i \geq 0 \quad i \in I, C_{ik} \geq 0 \quad i \in I,\ k \in K \tag{19}$$

$$\text{all dv's are integer} \tag{20}$$

5.3 Appendix C: Non-Value Added Iterations

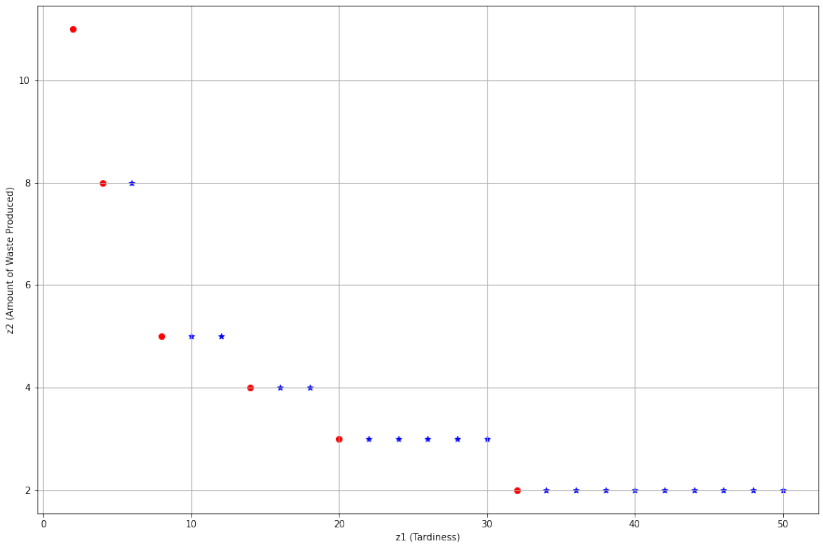

Figure 1: Non-Value Added Iterations

5.4 Appendix D: Missed Point

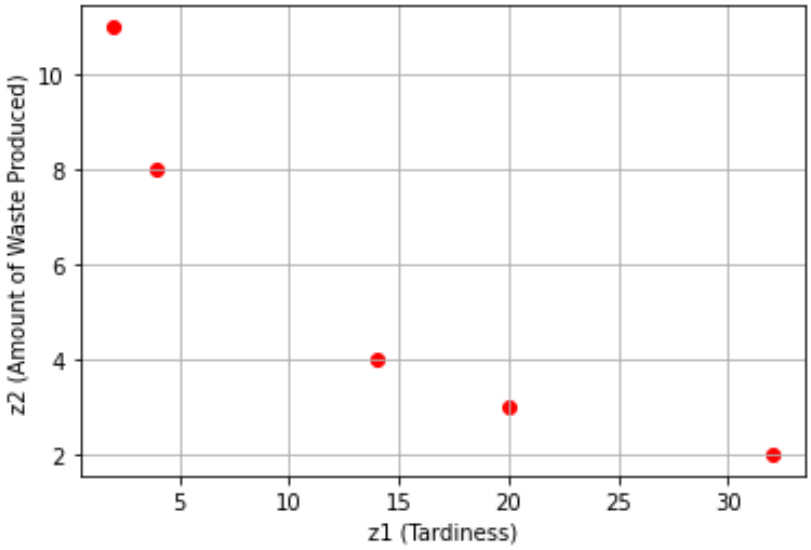

Figure 2: Missed point (8,5) when step size is 3

5.5 Appendix E: Weakly Nondominated Points

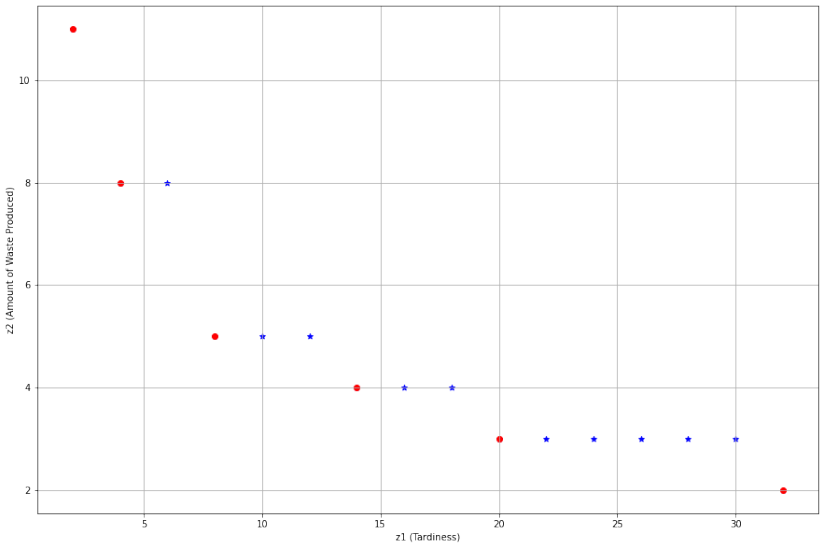

Figure 3: Weakly Nondominated Points

5.6 Appendix F: Nondominated Points

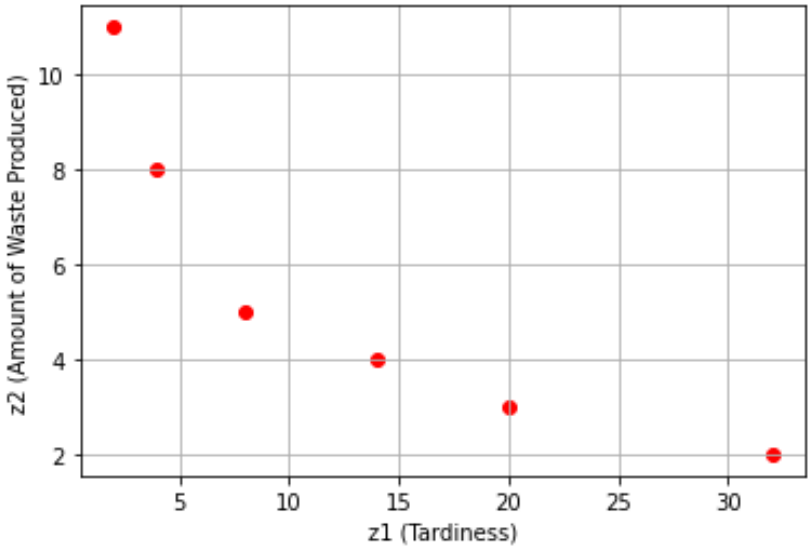

Figure 4: Nondominated Points

5.7 Appendix G: Pareto Frontier

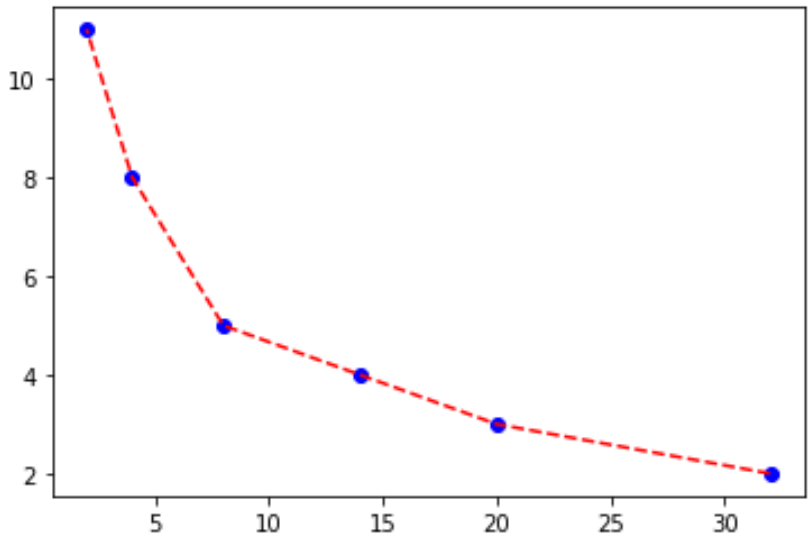

Figure 5: Pareto Frontier

5.8 Appendix H: A Section from the Convex Hull (Line on the Right Excluded)

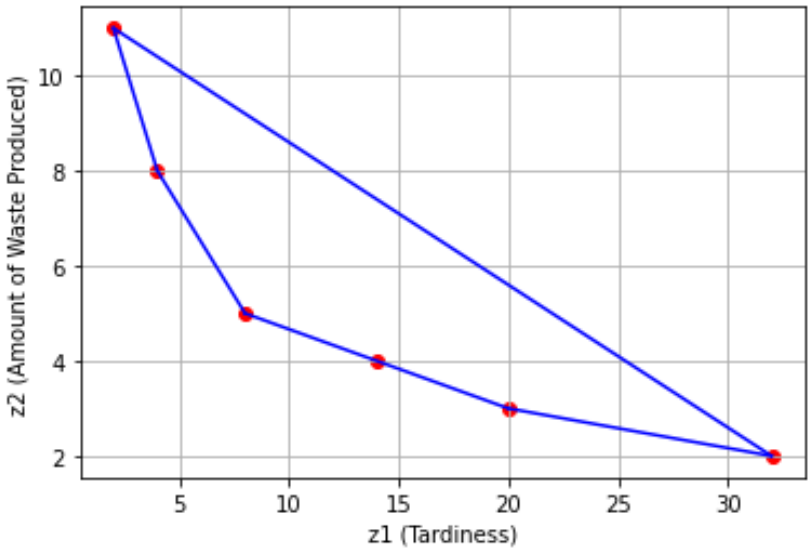

Figure 6: A Section from the Convex Hull (Line on the Right Excluded)

5.9 Appendix I: Convex Hull Indicating an Unsupported Point for a Different Scenario

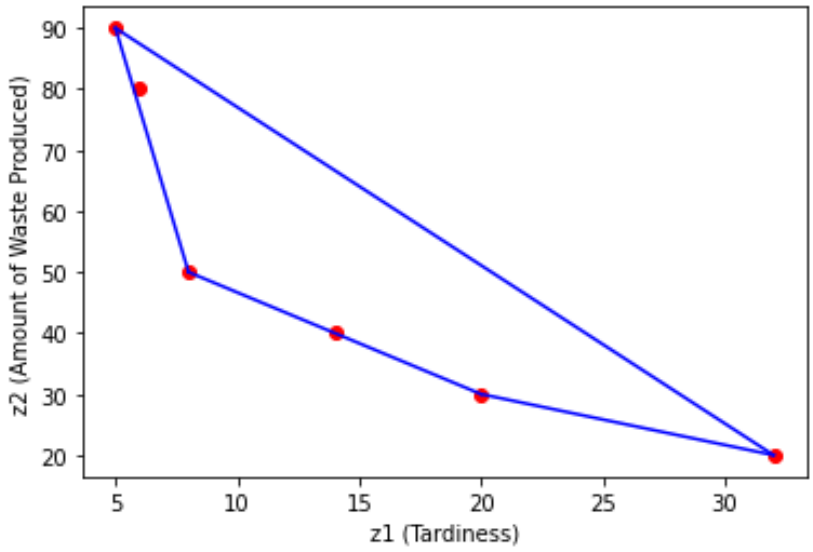

Figure 7: Convex Hull Indicating an Unsupported Point for a Different Scenario

5.10 Appendix J: Efficient Solutions

Machine 1		
Job	Amount	Comp. time
3	5	25

Machine 2		
Job	Amount	Comp. time
5	9	9
4	7	52

Machine 3		
Job	Amount	Comp. time
1	3	12
2	4	60

Figure 8: epsilon = 32, tardiness = 32, defective product = 2

Machine 1		
Job	Amount	Comp. time
3	5	25
4	2	40

Machine 2		
Job	Amount	Comp. time
1	3	12
2	4	60

Machine 3		
Job	Amount	Comp. time
5	9	9
4	5	40

Figure 9: epsilon = 20, tardiness = 20, defective product = 3

Machine 1		
Job	Amount	Comp. time
5	9	9
4	4	34

Machine 2		
Job	Amount	Comp. time
3	5	25
2	4	60

Machine 3		
Job	Amount	Comp. time
1	3	12
4	3	34

Figure 10: epsilon = 14, tardiness = 14, defective product = 4

Machine 1		
Job	Amount	Comp. time
1	3	12
3	5	42

Machine 2		
Job	Amount	Comp. time
4	4	28
2	4	60

Machine 3		
Job	Amount	Comp. time
5	9	9
4	3	28

Figure 11: epsilon = 8, tardiness = 8, defective product = 5

Machine 1		
Job	Amount	Comp. time
4	4	24

Machine 2		
Job	Amount	Comp. time
4	1	6
1	3	19
3	5	49

Machine 3		
Job	Amount	Comp. time
5	9	9
4	2	22
2	4	60

Figure 12: epsilon = 4, tardiness = 4, defective product = 8

Machine 1		
Job	Amount	Comp. time
4	1	6
1	3	20
3	5	50

Machine 2		
Job	Amount	Comp. time
4	3	20
2	4	51

Machine 3		
Job	Amount	Comp. time
4	3	18
5	9	32

Figure 13: epsilon = 2, tardiness = 2, defective product = 11

5.11 Appendix K: Computational Time Table

Number of Jobs	**Number of Machines**	**Run Time (sec)**
5	3	10.56
25	8	125.97
32	10	372.6
50	15	3877.8

5.12 Appendix L: Computational Time Graph

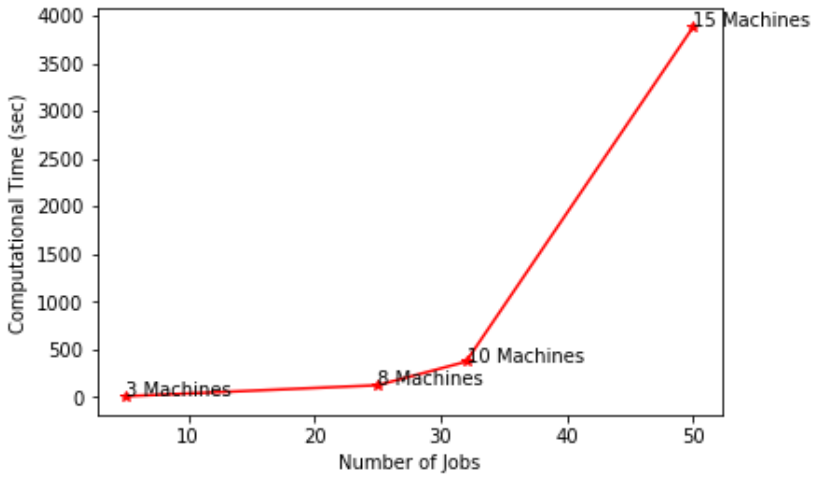

Figure 14: Exponentially Increasing Computational Time wrt. Increasing Number of Jobs and Machines

References

[1] Khodakaram GSalimifard, Jingpeng Li, Davood Mohammadi, and Reza Moghdani. A multi objective volleyball premier league algorithm for green scheduling identical parallel machines with splitting jobs. *Applied Intelligence*, 51(7):4143–4161, 2021.

[2] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.