Dec. 7, 23

ISE708 Final Project Report

Job Assignment and Scheduling Optimization

Project Team: Tsung-Yang Chang & Vasuki Garg

Abstract

This project delves into the optimization of job assignment and scheduling for a conventional factory specializing in outdoor apparel production. With the basic model being incorporated from Jin et al.'s paper on integer programming techniques for makespan minimizing workforce assignment models, the research extends and refines the model to address challenges such as workload fairness, the learning effect of workers, and job precedence.

The primary objectives encompass minimizing total production time, determining optimal assignments considering worker intricacies, and generating fair schedules. The methodology involves formulating an Integer Programming model, enhancing robustness with constraints for fairness and learning dynamics. Valid inequalities are introduced, and the model is implemented in Python using the Gurobi library. Various constraints ensure a structured framework, including those governing makespan, worker and job assignment, and cumulative worker experience. Learning constraints link productivity with predefined parameters, aligning experiential levels with productivity values. Energy consumption constraints safeguard against excessive usage. Precedence constraints maintain job order, and valid inequalities refine the model. In conclusion, this project provides a comprehensive optimization model for workforce assignment, considering real-world complexities and offering avenues for future refinement and extension.

1. Introduction

A conventional factory specializing in outdoor apparel relies on senior management's decision-making for job allocation among workers. This research project focuses on optimizing job assignment and scheduling, with inspiration drawn from the paper "Integer Programming Techniques for Makespan Minimizing Workforce Assignment Models that Recognize Human Learning" by Jin et al. [2] The model proposed in the paper serves as the groundwork for our extended version, which incorporates considerations such as fairness in workload distribution, the learning effect of workers, and job precedence. Within the manufacturing domain, minimizing makespan time holds utmost importance. Our objective is to create a comprehensive solution that tackles the intricacies of the problem.

For the scope of the project, we consider a production line manufacturing a product through a sequence of processes. The objective is to assign processes to each worker during designated time slots over the manufacturing horizon, aiming to minimize the total processing time. This assignment takes into account process precedence, fairness considerations, and the constraints imposed by workers' learning curves.

2. Objectives

The primary objectives of this project are multifaceted. First and foremost, we aim to minimize the total production time required for a single item. In pursuit of efficiency, we seek to determine the optimal assignment for each worker and process, carefully considering the intricacies of task distribution. Our goal extends to generating an optimal schedule that encompasses all processes, taking into account the sequential order/precedence between them. Moreover, we emphasize the importance of fairness in the allocation of tasks among workers, ensuring an equitable distribution of workload. Additionally, the project incorporates the dynamic element of the learning effect, acknowledging its influence on worker productivity and integrating it into the overall optimization framework.

3. Methodology

The methodology for this project involves a systematic approach to address the assignment and scheduling challenges. Initially, we articulate the assignment and schedule problem through the formulation of an Integer Programming model. To enhance the model's robustness, we introduce constraints focused on ensuring fairness in task allocation, emphasizing equitable workload distribution among workers. Furthermore, constraints related to process precedence and the learning dynamics of the workforce are incorporated to capture real-world complexities.

Valid inequalities are introduced to refine the model, enhancing its accuracy and practicality. Subsequently, the mathematical formulation is translated into executable code using Python, leveraging the Gurobipy library. This step allows for the seamless integration of the formulated model into a computational framework. The Gurobi solver is then applied to effectively tackle and solve the optimization problem, leveraging its capabilities to find an optimal solution to the intricacies posed by the assignment and scheduling requirements.

4. Modeling

4.1 Assumptions

In the modeling phase, certain assumptions are established to frame the optimization framework effectively. We assume that the process operates within discrete-time slots, where each slot represents a unit of time. It is assumed that a worker can engage in only one process within a given time slot, reflecting the singular focus of labor. Similarly, each process is exclusive to one worker in a specific time slot, ensuring individualized task assignments. The assumption of uniformity extends to the workforce, where all workers share the same initial experience and learning rate. This standardization further applies to the maximum productivity of all workers, emphasizing a consistent baseline for capabilities across the workforce. These foundational assumptions provide a structured basis for the subsequent development and analysis of the optimization model.

4.2 Parameters

$W: set\ of\ workers, \quad i = 1 \dots |W|$

$J: Set\ of\ jobs\ , \quad j = \cdots |J|$

$T: Set\ of\ time\ slots, \quad t = 1 \dots |T|$

$v_j = Volume\ of\ each\ job\ represented\ by\ the\ number\ of\ time\ slots, \quad \forall j$

$p_{ij} = Initial\ experience\ of\ worker\ i\ on\ job\ j, \quad \forall i, j$

$r_{ij} = learning\ rate\ of\ worker\ i\ on\ job\ j, \quad \forall i, j$

$K_{ij} = maximum\ worker's\ performance\ on\ job\ j, \quad \forall i, j$

$M_{ijt} = big\ M\ number\ matrix, \quad \forall i, j, t$

$l = acumulated\ experience\ of\ worker\ i\ upto\ time\ t\ on\ job\ j, 0 \dots t-1\ \forall t$

$\hat{\varphi}_{ijt}^{l} = productivity\ when\ having\ i\ accumulated\ experience\ upto\ time\ t, \quad \forall i, j, t$

$\bar{\varphi}_j = \max_i \hat{\varphi}_{ijt}^{T-1}$

$E_j = Energy\ consumption\ on\ job\ j, \quad \forall j$

$ME = maximum\ energy\ consumption\ per\ job, \quad \forall j$

4.3 Decision Variables

$T_{max} = completion\ time\ of\ the\ last\ job$

$$x_{ijt} = \begin{cases} 1, & if\ job\ j\ is\ done\ by\ worker\ i\ at\ time\ slot\ t \\ 0, & otherwise \end{cases}, \quad \forall i, j, t$$

$c_{ijt} = cumulated\ experience\ \ \forall i, j, t$

$\varphi_{ijt} = productivity\ \ \forall i, j, t$

$$z_{ijt}^{l} = \begin{cases} 1, & if\ worker\ i\ has\ done\ l\ periods\ of\ job\ j\ upto\ time\ t \\ 0, & otherwise \end{cases}, \quad \forall i, j, t, l$$

$$y_{ijt} = \begin{cases} 0, & if\ x_{ijt}\ is\ the\ end\ period\ of\ process\ j \\ 1, & otherwise \end{cases}, \quad \forall i, j, t$$

$$u_{ij} = \begin{cases} 1, & if\ job\ j\ is\ assinged\ by\ worker\ i \\ 0, & otherwise \end{cases}, \quad \forall i, j$$

$st_j = start\ period\ of\ job\ j, \quad \forall j$

$ed_j = end\ period\ of\ job\ j, \quad \forall j$

4.4 Objective Function

$Min\ T_{max},$ (0)

Eq. (0) is the objective minimizing the makespan time T_{max}.

4.5 Constraints

$$t * x_{ijt} \leq T_{max}, \quad \forall\, i,j,t, \quad (1)$$

$$\sum_{j}^{J} x_{ijt} \leq 1, \quad \forall\, i,t, \quad (2)$$

$$\sum_{i}^{W} x_{ijt} \leq 1, \quad \forall\, j,t, \quad (3)$$

$$c_{ijt} = \sum_{k=1}^{t-1} x_{ijk}\,, \quad \forall\, i,j,t = 2 \ldots T \quad (4)$$

These constraints have been taken directly from "Integer Programming Techniques for Makespan Minimizing Workforce Assignment Models that Recognize Human Learning" by Jin et al. [2]

The constraints within the optimization model are designed to establish a structured framework. Firstly, Constraint (1) introduces the concept of makespan, defining it as the completion time of the last job, denoted as T_{max}. Moving on to Constraint (2), it explicitly ensures that a worker cannot undertake more than one job within a given period. Similarly, Constraint (3) imposes the restriction that only a single worker can be assigned to a specific job during any given period, avoiding parallel execution by multiple workers on the same task.

Finally, Constraint (4) introduces the variable c_{ijt}, representing the accumulated experience of worker i for job j up to time t. Together, these constraints provide a comprehensive set of rules that guide the basic optimization model.

4.6 Learning Model Extension

$$\varphi_{ijt} = \sum_{l=0}^{t-1} \hat{\varphi}_{ijt}^{l} z_{ijt}^{l}\,, \quad \forall\, i,j,t, \quad (5)$$

$$\sum_{l=0}^{t-1} l z_{ijt}^{l} \leq c_{ijt}, \quad \forall\, i,j,t, \quad (6)$$

$$\sum_{l=0}^{t} z_{ijt}^{l} \leq 1, \quad \forall\, i,j,t, \quad (7)$$

$$\varphi_{ijt} \leq M_{ijt} x_{ijt}, \qquad \forall\, i,j,t, \qquad (8)$$

$$\sum_{i=1}^{W} \sum_{t=1}^{T} \varphi_{ijt} \geq v_j, \qquad \forall\, j, \qquad (9)$$

$$x_{ijt} \in \{0,1\}, \qquad i \in W, \qquad j \in J, \qquad t \in T, \qquad (10)$$

The learning model extension constraints has been taken directly from "Integer Programming Techniques for Makespan Minimizing Workforce Assignment Models that Recognize Human Learning" by Jin et al. as well. [2]

Here, we first calculate $\hat{\varphi}_{ijt}^{l}$ as a priori parameter using the equation (11) given the maximum productivity, learning rate, initial experience and accumulated experience l for each combination of i,j,t.

$$\hat{\varphi}_{ijt}^{l} = K_{ij}\left(1 - e^{-\left(\frac{l+p_{ij}}{r_{ij}}\right)}\right), l = 0 \ldots t-1, \qquad \forall\, i,j,t, \qquad (11)$$

Further, Constraint (5) establishes the link between productivity and the predefined parameter value $\hat{\varphi}_{ijt}^{l}$, assigning the productivity φ_{ijt} the corresponding value $\hat{\varphi}_{ijt}^{l}$ when the binary variable z_{ijt}^{l} is set to 1. In a complementary fashion, Constraint (6) guarantees that the index l in the assigned predefined value $\hat{\varphi}_{ijt}^{l}$ is equivalent to the accumulated experience c_{ijt}, aligning experiential levels with productivity parameters.

Furthermore, Constraint (7) is designed to maintain consistency by ensuring that each individual is assigned only one productivity value within each period, avoiding ambiguities in the allocation of productivity. Lastly, Constraint (9) enforces that the volume of work associated with job j, must be completed by the conclusion of the planning horizon. Together, these constraints govern the assignment and alignment of productivity parameters, contributing to the overall coherence and feasibility of the optimization model.

4.7 Fairness Model Extension

$$\sum_{j=1}^{J} \sum_{t=1}^{T} E_j\, \varphi_{ijt} \leq ME, \qquad \forall\, i, \qquad (12)$$

Referring to earlier definitions, where E_j represents the energy consumption associated with job i and ME is the maximum allowable energy consumption per job, Constraint (12) ensures that, for each worker, the cumulative energy consumption across all jobs throughout the entire time horizon remains below the stipulated maximum energy consumption threshold. In essence, it safeguards against excessive energy utilization, establishing a critical boundary to manage and control the overall energy of the workforce across the defined planning horizon.

The estimation of the energy consumption requirement for each job E_j derived from the research on the maximum physical loading for female textile workers [4]. According to the paper, the maximum allowable physical exertion among the female textile workers should not exceed 3.5 kcal/min, which we assumed that every work should take the workers' energy within that up bound. To define fairness [3] considering different skill levels of job requiring different effort, we assigned different energy consumption to them. For instance, all the jobs have been classified into 4 levels, namely A, B, C, and D, and we made an assumption that for level A, it requires 3 kcal/min, B requires 2.8 kcal/min, C takes 2.5 kcal/min, and D consumes 2.3 kcal/min.

Also, ME for each worker is estimated by assuming that the typical total energy consumption for a female is 2100 kcal per day. If they work for 8 hours in a shift and only 80% of their energy could be used for the work, it means that only 560 kcal ($= 2100 * \frac{8}{24} * 0.8$)can be utilized for working. Next, assume that the target of productivity per day is 40 items, which implies that each work can only consume 18.6 kcal ($= 560/30$) for a piece of product to avoid from being completely exhausted during their shift.

4.8 Precedence Model Extension

$$st_p - ed_q \geq 1, \qquad \forall\, (p,q) \in J, \qquad (13)$$

$$ed_q \leq t * x_{ijt} + T * y_{ijt}, \qquad \forall\, j, \qquad (14)$$

$$\sum_{i=1}^{W}\sum_{t=1}^{T} y_{ijt} \leq w * T - 1, \qquad \forall\, j, \qquad (15)$$

$$ed_j \geq t * x_{ijt}, \qquad \forall\, i,j,t, \qquad (16)$$

$$ed_j \leq T_{max}, \qquad \forall\, j, \qquad (17)$$

$$ed_j - st_j = \sum_{i=1}^{W}\sum_{t=1}^{T} x_{ijt} - 1, \qquad \forall\, j, \qquad (18)$$

$$st_j \geq 1, \qquad \forall\, j, \qquad (19)$$

$$st_j \leq \sum_{i=1}^{W}\sum_{t=1}^{T} t * x_{ijt}\,, \qquad \forall\, j, \qquad (20)$$

The idea of constructing the constraints for the precedence is inspired by the paper [1], which discusses the job assignment problem for an Assembly Line Balancing Problem.

In order to maintain precedence of jobs, constraint (13) ensures that the start time of job p is at least one time unit later than the end time of job q for all pairs (p, q) in the set of jobs J. As the procedures discussed during the lectures to linearise the maximum function, with the aim to calculate $ed_j = \max\,(t * x_{ijt})$, Constraint (14) imposes a limit on the end time of job q based on the binary variables x and y and constraint (15)

ensures that only for one value of y_{ijt} is 0 which corresponds to the maximum while the rest are summed up to give $w * T - 1$. Constraint (16) provides a sanity check in ensuring our ending period must be greater than all operating periods for a job. Constraint (17) ensures that the ending period for a job is earlier than the completion of the last job, obeying the definition of T_{max}.

Constraint (18) allows us to enforce that the ending period plus one time period is equivalent to the sum of the starting period and the periods the job is run for. Constraint (19) ensures a lower bound on the starting period of 1 for each job to ensure that each job is started throughout the horizon. While for the case of constraint (20), incorporates a check to ensure the for all the time periods a particular job is run for, must be greater that the starting period for that job.

4.9 Valid Inequality

The valid inequality and its procedural methodology have been directly incorporated from "Integer Programming Techniques for Makespan Minimizing Workforce Assignment Models that Recognize Human Learning" by Jin et al. [2]

We introduce a new parameter $\bar{\varphi}_j$ as defined below:

$\bar{\varphi}_j = parameter\ introduced\ for\ Valid\ Inequality, where$:

$$\bar{\varphi}_j = max_i\, \hat{\varphi}_{ijt}^{t-1}, and, \qquad (21)$$

$$\hat{\varphi}_{ijt}^{T-1} = \max\{\hat{\varphi}_{ijt}^{l}, \mathrm{l} = 0,1,\dots \mathrm{T}-1\}, \qquad (22)$$

$\hat{\varphi}_{ijt}^{T-1}$ represents the maximum production rate of worker i on job j throughout the entire time horizon.

By ensuring that the work required for job j must be completed by the end of the time horizon, we incorporate our newly defined parameter as:

$$\sum_{i=1}^{W}\sum_{t=1}^{T} \bar{\varphi}_j x_{ijt} \geq v_j, \qquad \forall\, j, \qquad (23)$$

Using the integer rounding techniques introduced during the lectures, we divide both sides by $\bar{\varphi}_j$ and take the ceiling of the right-hand side. Consequently, the cover inequality is formulated as:

$$\sum_{i=1}^{W}\sum_{t=1}^{T} x_{ijt} \geq \left\lceil \frac{v_j}{\bar{\varphi}_j} \right\rceil, \qquad \forall\, j, \qquad (24)$$

4.10 Lower Bound Model

The lower bound model and its procedural methodology have been directly incorporated from "Integer Programming Techniques for Makespan Minimizing

Workforce Assignment Models that Recognize Human Learning" by Jin et al.

A formulation's robustness can be enhanced by introducing a lower bound on the optimal objective value. To establish such a bound, we address an integer program closely resembling our problem but with simpler solvability, denoted as the Makespan Minimizing Workforce Assignment with Maximum Learning ($MwML$) model. In this simplified version, we assume that all workers operate at their maximum productivity for all jobs across all time periods $\hat{\varphi}_{ijt}^{t-1}$, this model serves as a useful reference.

The ($MwML$) model seeks to minimize the makespan while adhering to constraints below. It incorporates the assumption of maximum productivity for all workers i, time periods t, and jobs j, subject to the constraint (LB.5). ($MwML$) is a relaxation of original problem, with certain constraints removed or weakened.

To establish a global bound on the optimal objective function value before solving our original model, we introduce the constraint (LB.6).

$$Min\ T_{max}^{MwML}, \qquad (LB.0)$$

$$\sum_{j}^{J} x_{ijt} \leq 1, \qquad \forall\, i,t, \qquad (LB.1)$$

$$\sum_{i}^{W} x_{ijt} \leq 1, \qquad \forall\, j,t, \qquad (LB.2)$$

$$c_{ijt} = \sum_{k=1}^{t-1} x_{ijk}\,, \qquad \forall\, i,j,t = 2 \ldots T \quad (LB.3)$$

$$x_{ijt} \in \{0,1\}, \qquad i \in W, \qquad j \in J, \qquad t \in T, \qquad (LB.4)$$

$$\sum_{i=1}^{W}\sum_{t=1}^{T} \hat{\varphi}_{ijt}^{t-1} x_{ijt} \geq v_j, \qquad \forall\, j \in J, \qquad (LB.5)$$

$$T_{max}^{MwML} \leq T_{max}, \qquad (LB.6)$$

$$st_p - ed_q \geq 1, \qquad \forall\, (p,q) \in J, \qquad (LB.7)$$

$$ed_q \leq t * x_{ijt} + T * y_{ijt}, \qquad \forall\, j, \qquad (LB.8)$$

$$\sum_{i=1}^{W}\sum_{t=1}^{T} y_{ijt} \leq w * T - 1, \qquad \forall\, j, \qquad (LB.9)$$

$$ed_j \geq t * x_{ijt}, \qquad \forall\, i,j,t, \qquad (LB.10)$$

$$ed_j \leq T_{max}, \qquad \forall\, j, \qquad (LB.11)$$

$$ed_j - st_j = \sum_{i=1}^{W}\sum_{t=1}^{T} x_{ijt} - 1, \qquad \forall\, j, \qquad (LB.12)$$

$$st_j \geq 1, \qquad \forall j, \qquad (LB.13)$$

$$st_j \leq \sum_{i=1}^{W} \sum_{t=1}^{T} t * x_{ijt}, \qquad \forall j, \qquad (LB.14)$$

4.11 Cuts induced in the model

We employed Gurobi's built-in *Cuts* parameter to generate cutting planes, where this parameter governs the quantity of cutting planes generated and incorporated to enhance the linear programming (LP) relaxation. An increased number of cuts, denoted by *Cuts = 2 or 3* in our case, strengthens the LP relaxation but comes with the drawback of elevated computational costs for solving individual node LPs. Moreover, a surplus of cuts may introduce numerical challenges. Conversely, constraining the number of cuts, specified as *Cuts = 1 or 0*, weakens the LP relaxation, potentially leading to a greater number of nodes being evaluated before achieving optimality. Through experimentation, we observed that setting *Cuts = 1 or 2* did not result in convergence, while *Cuts = 3* exhibited convergence, hence our choice for the model.

5. Case Study

5.1 Instance

Consider a production line manufacturing outdoor outfit, let's say jackets. For the left sleeve of a single jacket, it takes 27 processes to produce one piece, and 8 workers are assigned to it. Refers to the Appendix for the standard processing time, precedence, skill level requirement.

Assume the time length of each time slot is 1 minute. Also, all the workers are identical, which means that they have the same initial experience, learning rate, and maximum performance in terms of productivity.

5.2 Results and Analysis

Then we imported the data into Python and applied Gurobi Solve to obtain the optimal solution. To begin with, the optimal solution to the Lower Bound Model is obtained and shown below:

Optimal Assignment and Schedule for Lower Bound

Time\Job\Worker	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	5	9	26	6	4	7	24	10
2	9	24	4		1	26	2	22
3	12	26	8	16	7	11	3	18
4	23	27	15	9	13	17	26	8
5	27	20		14	19		21	25
Energy Consumption	12.8	12.5	10.6	10.3	12.8	10.3	12.6	12.8

The table above shows the assignment of each job as well as the schedule in different time slots. Note that some of the jobs which require multiple periods are distributed into multiple workers, which is allowed in our model. The allowance depends on instances, and it is feasible in the case of sewing garments since everyone is identical in our assumptions. Also, all the precedence constraint and fairness constraints are satisfied.

Next, let's look at the result of our model considering the learning effect. The assignment and the schedule are listed below:

Optimal Assignment and Schedule Considering Learning

Time\Job\Worker	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
1	24	6	2	9	26	8	4	1
2	24	7	2	9	26	8	21	1
3	27	4	3	10	26	8	7	5
4	27	7	16	20	26	11	22	13
5	15	23	21	18	22	14	16	11
6	12	19	14	20	21	17	25	18
Energy Consumption	15	15.6	15.2	15	15.3	16.2	15.9	15

Notice that the precedence constraint is violated only for processes 21 and 9. Process 21 should be performed after the completion of process 9 but in this case the start of process 21 and the end of process 9 is at the same time slot, which leads to the imperfection of our model. Other than that, the outcome satisfies fairness constraints and provides a clear assignment and scheduling to follow.

Moreover, after the case study, we performed some numerical experiments to determine the impact of the Cuts. The comparison table is shown below:

Model	**Time (sec)**
Lower Bound	<1
Regular Model without Cuts, LB	1075
Regular Model with Cuts, LB	75

We can see that the lower bound is solved instantly since the model is relatively small in terms of the number of variables and constraints. Also, it shows that the Cuts generated by Gurobi Solver poses a significant impact on the running time of the model. The running time is increased by 13 times after removing the Cuts, which justifies the benefit of adding Cuts to the model.

5.3 Limitations of the Model

According to the result of the model mentioned in the previous section, the precedence constraint is violated. After exhaustively examining our model, we realized that the precedence constraints can't prevent this from occurring since the relationship between

the decision variable st_j and x_{ijt} can't be captured in the current settings. Logically, the smallest $t * x_{ijt}\ \forall i,j,t$ when $x_{ijt} = 1$ is the start period of each job, however, if we also consider the cases when $x_{ijt} = 0$, then the minimum value of them is 0, which is not similar to the end period case. Furthermore, the reason why the precedence constraints are satisfied in the lower bound model is that most of the jobs in lower bound scenario only takes a single period to complete, which the precedence constraints work well in that case. To address the issue, the definition of decision variables x_{ijt} should be modified as if the job j is assigned to worker i and starting at time t. Due to the change of the basis, the whole model should be changed as well and it would be counted as our future work.

6. Conclusion

The optimal assignment and scheduling are able to be generated through our model. Even though there is an imperfection in our model, the results can still be considered as the basis of the assignment and then the manual adjustment by the leader of production line comes to play. They don't have to do the assignment from scratch. Also, we successfully incorporate the learning effect and fairness which are also important in real environment but easy to be omitted. Lastly, as the complexity of precedence goes high and the time it takes to reach optimality or convergence will be rocketed. Similarly, the number of jobs, workers, and time horizon play critical roles to impact the running time. Therefore, more VIs and Cuts should be created and implemented to reduce the running time and make the model more implementable in job shops.

7. Future Work

In the realm of future work, the optimization problem at hand can be extended and refined in various dimensions. One avenue involves tackling a larger-scale challenge by solving the problem with a substantial workload, specifically, 127 processes, compounded by the involvement of 35 workers over a time horizon of 10 minutes. To enhance the model's realism, the future exploration incorporates diversity among workers by introducing distinct initial experience levels, learning rates, and maximum productivity values for each individual.

Expanding the scope of evaluation, an innovative approach is proposed to measure the oxygen consumption of workers. This metric is intended to provide a more accurate and nuanced estimation of energy consumption, offering valuable insights into the physiological aspects of labor.

Furthermore, the future work entails the integration of additional constraints to further refine the optimization model. These include considerations such as machining constraints, and skill-related constraints. Each of these factors contributes to a more comprehensive and realistic representation of the manufacturing environment.

Moreover, to address the issue of the precedence constraints violation, the decision variable x_{ijt} should be redefined and the whole mode should be changed as well. Also,

we can assume that the work has to be done in the consecutive periods to avoid any additional setup cost since it is a little bit counter-intuitive to anyone.

In pursuit of improved solution quality, manual creation of valid inequalities is proposed. This step involves introducing additional constraints that capture specific characteristics of the problem, contributing to a more tailored and effective optimization process. Collectively, these future works aim to advance the applicability of the optimization model in addressing real-world manufacturing scenarios.

Reference

[1] Koltai, T., Tatay, V. Formulation of workforce skill constraints in assembly line balancing models. Optim Eng 14, 529–545 (2013). https://doi.org/10.1007/s11081-013-9230-x

[2] Jin, H., Thomas, B. W., & Hewit, M. (2016). Integer programming techniques for makespan minimizing workforce assignment models that recognize human learning. Computers & Industrial Engineering, 97, 202–211. https://doi.org/10.1016/j.cie.2016.03.027

[3] Yi, W., & Wang, S. (2016). Multi-objective mathematical programming approach to construction laborer assignment with equity consideration. Computer-Aided Civil and Infrastructure Engineering, 31(12), 954–965. https://doi.org/10.1111/mice.12239

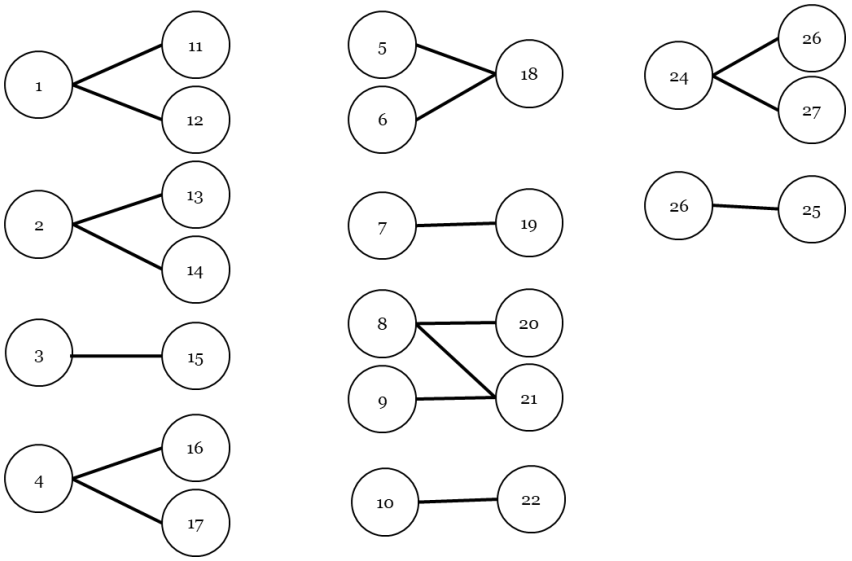
[4] Dobrovolski I, Dincheva E, Stoianova N. Opredeliane optimuma na fizicheskoto natovarvane v zavisimost ot maksimalnite vuzmozhnosti na organizma [Determination of the optimum of physical loading in relation to the maximum potentialities of the body]. Probl Khig. 1975;1:11-5. Bulgarian. PMID: 1235987.

Appendix-I: Data for Instance

Available Data

Index	Process Name	Required Skill Level	Processing Time (Sec)	Predecessor in index	Processing Time (1 min)
1	J1	C	58	N/A	0.967
2	J2	D	55	N/A	0.917
3	J10	C	32	N/A	0.525
4	J11	B	61	N/A	1.008
5	J12	C	23	N/A	0.385
6	J13	C	19	N/A	0.315
7	J14	C	100	N/A	1.663
8	J15	B	138	N/A	2.292
9	J16	C	42	N/A	0.700
10	J17	C	15	N/A	0.245
11	J18	C	59	1	0.980
12	J19	C	21	1	0.350
13	J20	C	15	2	0.245
14	J21	C	60	2	0.998
15	J22	C	28	3	0.473
16	J23	B	66	4	1.100
17	J24	B	33	4	0.550
18	J25	C	47	5, 6	0.788
19	J26	C	19	7	0.315
20	J27	C	63	8	1.050
21	J28	B	77	8, 9	1.283
22	J29	C	58	10	0.963
23	J30	B	29	24	0.477
24	J31	C	68	N/A	1.138
25	J32	C	25	26	0.420
26	J33	C	200	N/A	3.325
27	J34	C	84	24	1.400

Precedence Network of jobs



Appendix-II: Python code used for Analysis

```
from gurobipy import Model, GRB, quicksum

import gurobipy as gp

import math

import pandas as pd


# read information of the process

Process_Dataset_df = pd.read_excel("Process_Dataset.xlsx", sheet_name="Part 1")

Volume_Job = Process_Dataset_df["Processing Time (1 min)"]

Volume_Job.tolist

Skill_Level = Process_Dataset_df["Required Skill Level"]

Skill_Level.tolist


# convert the skill lsevel to the required energy consumption per 10 seconds (length of
period)

EnergyConsm = []

for letter in Skill_Level:

    if letter == "A":

        EnergyConsm.append(3)

    elif letter == "B":

        EnergyConsm.append(2.8)

    elif letter == "C":

        EnergyConsm.append(2.5)

    elif letter == "D":

        EnergyConsm.append(2.3)


# create a list of precedence (first one is job i and the second is precedence)

Precedence =  [[11, 1], [12, 1], [13, 2], [14, 2], [15, 3], [16, 4],

               [17, 4], [18, 5], [18, 6], [19, 7], [20, 8], [21, 8],

               [21, 9], [22, 10], [23, 24], [25, 26], [27, 24]]


# Parameters

jWj = 8  # Number of Workers

jJj = len(Volume_Job)  # Number of Jobs

jTj = 8  # Number of Time Periods (each period represent i minute)

Kij = 2  # Learning Asymptote

pij = 1  # Initial Experience

rij = 3  # Learning rate

#M = 2  # Random Big M

v = Volume_Job  # Processing time for the each job
```

```
ME = 18.6  # Max work in the time horizon
E = EnergyConsm

# Model
model = gp.Model("Job_Assignment_with_Learning")

# Sets
W = range(1, jWj + 1)
J = range(1, jJj + 1)
T = range(1, jTj + 1)

# Variables Sets
x = {}
c = {}
y = {}
st = {}
ed = {}
phi = {}
zl = {}
phi_l_cap = {}
M = {}
Phi_l_CapT = {}
Phi_l_bar_CapT = {}
Tmax = model.addVar(vtype=GRB.CONTINUOUS, name="Tmax")

# Variables Creation
for i in W:
    for j in J:
        for t in T:
            production_rates = []

            for l in range(t):
                phi_l_cap[i, j, t, l] = Kij * (1 - math.exp(-(l + pij) / rij))
                production_rates.append(phi_l_cap[i, j, t, l])
               M[i, j, t] = max(production_rates)

for i in W:
    for j in J:
        for l in range(jTj):
            productivity = []
```

```
            productivity.append(phi_l_cap[i, j, jTj, l])
                Phi_l_CapT[i, j] = max(productivity)

for j in J:
    for i in W:
        productivity = []
        productivity.append(Phi_l_CapT[i, j])
        Phi_l_bar_CapT[j] = max(productivity)


for i in W:
    for j in J:
        for t in T:
            x[i, j, t] = model.addVar(vtype=GRB.BINARY,  name=f"x_{i}_{j}_{t}")
            c[i, j, t] = model.addVar(vtype=GRB.CONTINUOUS, name=f"c_{i}_{j}_{t}")
            phi[i, j, t] = model.addVar(vtype=GRB.CONTINUOUS, name=f"phi_{i}_{j}_{t}")
            y[i, j, t] = model.addVar(vtype=GRB.BINARY, name=f"y_{i}_{j}_{t}")

for i in W:
    for j in J:
        for t in T:
            for l in range(t + 1):
               zl[i, j, t, l] = model.addVar(vtype=GRB.BINARY, name=f"zl_{i}_{j}_{t}_{l}")

for j in J:
    st[j] = model.addVar(vtype=GRB.INTEGER, name=f"st_{j}")
    ed[j] = model.addVar(vtype=GRB.INTEGER, name=f"ed_{j}")

# Objective function
model.setObjective(Tmax, GRB.MINIMIZE)

# Constraints
# Constraint 1
for i in W:
    for j in J:
        for t in T:
            model.addConstr(t * x[i, j, t] <= Tmax, f"constraint_1_{i}_{j}_{t}")

# Constraint 2
for i in W:
    for t in T:
```

```
        model.addConstr(quicksum(x[i, j, t] for j in J) <= 1, f"constraint_2_{i}_{t}")


# Constraint 3
for j in J:
    for t in T:
        model.addConstr(quicksum(x[i, j, t] for i in W) <= 1, f"constraint_3_{j}_{t}")


# Constraint 4
for i in W:
    for j in J:
        for t in range(2, jTj +1):
            model.addConstr(c[i, j, t] == quicksum(x[i, j, k] for k in range(1, t)),
                            f"constraint_4_{i}_{j}_{t}")


# Constraint 5
for i in W:
    for j in J:
        for t in T:
            model.addConstr(
                phi[i, j, t] == quicksum(phi_l_cap[i, j, t, l] * zl[i, j, t, l] for l in
                range(0, t)),f"constraint_5_{i}_{j}_{t}")


# Constraint 6
for i in W:
    for j in J:
        for t in T:
            model.addConstr(
                quicksum(l * zl[i, j, t, l] for l in range(0, t)) <= c[i, j, t],
                f"constraint_6_{i}_{j}_{t}")


# Constraint 7
for i in W:
    for j in J:
        for t in T:
            model.addConstr(
                quicksum(zl[i, j, t, l] for l in range(0, t + 1)) <= 1,
                f"constraint_7_{i}_{j}_{t}")


# Constraint 8
for i in W:
    for j in J:
```

```
        for t in T:
            model.addConstr(
                phi[i, j, t] <= M[i, j, t] * x[i, j, t],f"constraint_8_{i}_{j}_{t}")


# Constraint 9
for j in J:
    model.addConstr(
        quicksum(phi[i, j, t] for i in W for t in T) >= v[j - 1],
        f"constraint_9_{j}")


# Constraint 10
for i in W:
    model.addConstr(
        quicksum(E[j - 1] * x[i, j, t] for j in J for t in T) <= ME,
        f"constraint_10_{i}")


# Constraint 11: V.I
for j in J:
    model.addConstr(
        quicksum(x[i, j, t] for i in W for t in T) >= math.ceil(v[j-1]/Phi_l_bar_CapT[j]),
        f"constraint_11_{j}")


# Constraint 12: Jobs Precedence
for combination in Precedence:
    model.addConstr(st[combination[0]] - ed[combination[1]] >= 1,
                f"constraint_12_{combination}")


# Constraint 12-1
for j in J:
    for i in W:
        for t in T:
                    model.addConstr(ed[j] <= t*x[i, j, t] + jTj*y[i, j, t],
f"constraint_14_{i}_{j}_{t}")


# Constraint 12-3
for j in J:
      model.addConstr(quicksum(y[i, j, t] for i in W for t in T) == jWj*jTj-1,
f"constraint_15")


# Constraint 12-5
for i in W:
```

```
    for t in T:
        for j in J:
            model.addConstr(ed[j] >= t*x[i, j, t], f"constraint_16{j}")


# Constraint 12-6
for j in J:
    model.addConstr(ed[j] <= Tmax, f"constraint_17")


# Constraint 12-7
for j in J:
    model.addConstr(ed[j] - st[j] == quicksum(x[i, j, t] for i in W for t in T) - 1,
f"constraint_18_{j}")


# Constraint 12-8
for j in J:
   model.addConstr(st[j] >= 1, f"constraint_19_{j}")


# Constraint 12-9
for j in J:
    model.addConstr(st[j] <= quicksum(t*x[i, j, t]for i in W for t in T),
f"constraint_20_{j}")


# Constraint 13: Lower Bound
model.addConstr(Tmax >= 5)


# Implement Cuts Created by Gurobi
model.setParam('Cuts', 3)  # Enable Gurobi to generate Chvátal-Gomory cuts


# Solving model
model.optimize()


# Print the values of decision variable x
print("Objective value with Chvátal-Gomory cuts:", model.objVal)
print("Solution with Chvátal-Gomory cuts:")
for var in model.getVars():
    if var.varName.startswith("x") and var.x == 1:
        print(f"Worker {var.varName.split('_')[1]} performs job
        {var.varName.split('_')[2]} at period {var.varName.split('_')[3]}: {var.x}")
```