Introduction:

The simulation part of the PADME project was set out to aid in reducing the production cycle time at the robotic factory at ABB. Moreover, we aimed to connect the simulation model to live streams of data from the factory floor, to enable real-time control and adjustments to planned production based on new information. The latter is known as a dynamic simulation model or a digital twin.

Several simulation environments were explored, including ABB's own simulation system, but we concluded that utilizing the full potential of any simulation system required more data than available. Hence, we focused on reducing the cycle time by improving the scheduling process of incoming jobs.

Improving the scheduling process:

The current scheduling process relies on a heuristic approach that accounts for material availability, current products being produced, due dates, and machine failures to name a few. The scheduling process also assumes a fixed takt time to process different products on different machines or workstations. This assumption reduces the blockage and starvation phenomena that occurs on a serial production line with different processing times. It also allows products to move in a synchronized fashion resulting in a seemingly steady throughput. We wanted to investigate if the cycle time can be reduced if we schedule the jobs in a structured manner, using mathematical programming.

A mathematical model for job sequencing:

The mathematical model represents the following scenario: a number of n jobs $j = 1,2, \dots, n$ are to be sequenced on a number of m machines $k = 1,2, \dots, m$. Each job has a specific sequence, or route, to follow r_j that specifies the order of machines to visit. Each job has a deterministic processing time on each machine $p_{j,k}$. Each machine has a capacity of 1 and can process one job at a time. We do not allow a job to move ahead of another job even if the target machine is available. We also do not allow any buffer space between machines.

Parameters:

Parameter	Representation
$p_{j,k}$	A deterministic processing time of job j on machine k
r_j	A sequence of machines that job j must visit
List of jobs	A number of n jobs $j = 1,2, \dots, n$ to sequence
List of machines	A number of m machines $k = 1,2, \dots, m$ to process the jobs

Decision variables:

Variable	Type and domain	Representation
$s_{j,k}$ $\forall j, k$	Continuous [Earliest start time, latest finish time]	start time of job j on machine k. Note that since $p_{j,k}$ are all deterministic, the domain for variable cab be reduced to $[EST, LFT]$
$y_{k,j,j'}$ $\forall j, k; j \neq j'$	Binary $\{0,1\}$	a binary variable that is equal to 1 if job j is to be processed before job j' on machine k.

		Note that since we do not allow for buffers or one job taking over another, the sequence of jobs that go into the first machine should be maintained throughout the entire route r_j.
f	Continuous ≥ 0	The makespan time, or the finish time of the last job to be produced. It is used in the objective function.

Constraints:

1. Precedence constraints: jobs must follow their specified routes; that is, if for job j we have machine k preceding machine k' ($k < k'$), then j must start and finish on k *immediately* before it starts on k'. Note that the word immediately implies that there is no buffer space between succeeding machines.

$$s_{j,k} + p_{j,k} = s_{j,k'} \ \forall j, k$$

2. Capacity constraints: each machine can process only one job at a time; that is, if two jobs j and j' compete for machine k, then either j starts and finishes before j' starts, or vice versa.

$$s_{j,k} + p_{j,k} \leq s_{j',k} + M_1(1 - y_{k,j,j'}) \ \forall j, k$$

$$s_{j',k} + p_{j',k} \leq s_{j,k} + M_2(y_{k,j,j'}) \ \forall j, k$$

Where $M_1$and M_2 are numbers large enough to dominate the constraints. Recommended values are

$$M_1 = LFT_j - EST_{j'}$$

$$M_2 = LFT_{j'} - EST_j$$

Where the Latest Finish Time and the Earliest Start Time are derived from the deterministic durations.

3. Fixed sequence constraints: jobs must maintain the same order on all machines; that is, if job j is sequenced before j' on the first machine that they both use, $k = 1$ in this case, then they must maintain that order on all subsequent machines they visit along their routes.

$$y_{1,j,j'} = y_{k,j,j'} \ \forall k > 1; k \in r_j$$

4. Variable constraints:

$$EST \leq s_{j,k} \leq LFT$$

$$y_{k,j,j'} = \{0,1\}$$

Objective function:

The objective is to minimize the makespan time, which is the finish time of the last job produced.

$$min. f$$

$$f \geq s_{j,k} + p_{j,k} \ \forall j; k = k_{last}$$

Where $k_{last}$is the last machine job j visits in its route.

Experiments and results:

1. Cycle time reduction:

    1.1. Experimental setup:
    To assess whether the proposed model can produce a shorter cycle time, we collected data from different work weeks showing batches of products to be produced, and the sequence in which they were dispatched to the shop floor. We refer to this as the basic sequence seq_{basic}. Since the real makespan time depends on process randomness and distributions that occurred during production, we cannot compare it to the result from the model; however, we can use the model to calculate the makespan time *if* jobs were dispatched according to seq_{basic} and compare it to the optimal makespan time and the optimal job sequence seq_{opt}. Production mixes from 6 different weeks were collected. Each week was considered as an instance of the problem.

    1.2. Results:
    The results in Table 1 show scheduling jobs using a mathematical model that minimizes the makspane time, compared to scheduling them heuristically, as is done right now. Each record represents 12 jobs that were produced on different dates, along with its makespan time. Clearly, a mathematical model approach to scheduling is superior in all cases, especially that the search algorithm terminated when reaching optimal solutions.

Table 1: makespan time reduction using optimal job dispatching compared to heuristic dispatching

Instance	Makespan (min) with optimal dispatching	Makespan (min) with heuristic dispatching
19-03-19	446	598
21-03-19	449	577
28-03-19	438	493
01-04-19	440	522
02-04-19	266	364

Another measure of interest is the utilization of machines for a given scheduling solution. Though this is not the objective for either scheduling methods, it is worth noting how it differs. As seen in Table 2, using a mathematical model to schedule jobs increases the utilization of all machines, across the different instances considered. Overall, a random machine can expect about 17% increase in its utilization, for any given day. The minimum and maximum observed increase in utilization were 9% and 22% across instances, not machines.

Table 2: utilization improvement using a mathematical model to schedule jobs over a heuristic approach.

Instance	**19-Mar**		**21-Mar**		**28-Mar**		**01-Apr**		**02-Apr**	
Machine	Bas.	Opt.	Bas.	Opt.	Bas.	Opt.	Bas.	Opt.	Bas.	Opt.
Axis1-2_0	38%	49%	39%	47%	42%	49%	43%	53%	31%	35%
Axis1-2_1	39%	50%	43%	52%	44%	50%	41%	50%	46%	50%
Axis1-2_2	42%	53%	44%	52%	48%	54%	45%	54%	51%	54%
Axis1-2_3	36%	44%	35%	41%	36%	41%	35%	43%	44%	45%

Axis1-2_4	28%	35%	30%	35%	33%	38%	31%	39%	30%	29%
Axis1-2_5	26%	32%	36%	42%	31%	36%	24%	30%	38%	39%
Axis1-2_6	63%	76%	60%	68%	62%	72%	62%	76%	68%	69%
Axis3-4_0	53%	83%	55%	82%	75%	91%	69%	83%	51%	60%
Axis3-4_1	35%	55%	33%	51%	43%	53%	43%	52%	42%	54%
Axis3-4_2	35%	57%	37%	56%	47%	57%	44%	55%	38%	52%
Axis3-4_3	32%	53%	32%	49%	41%	51%	39%	50%	37%	54%
Axis3-4_4	37%	61%	39%	60%	50%	61%	46%	61%	38%	57%
Axis5_0	32%	54%	30%	48%	37%	47%	37%	54%	30%	60%
Axis5_1	30%	51%	28%	46%	35%	43%	35%	51%	29%	61%
Axis5_2	31%	53%	30%	50%	37%	45%	35%	52%	30%	66%
Axis5_3	29%	51%	30%	50%	36%	44%	33%	49%	31%	72%
Axis5_4	35%	61%	36%	60%	43%	52%	40%	61%	32%	76%
Axis6_0	71%	95%	72%	91%	80%	82%	76%	84%	66%	78%
BalCy_0	36%	61%	30%	62%	42%	63%	43%	61%	33%	64%
BalCy_1	36%	61%	30%	62%	42%	63%	43%	61%	33%	64%
BalCy_2	36%	61%	30%	62%	42%	63%	43%	61%	33%	64%
finalAssembly_0	36%	44%	36%	41%	38%	44%	36%	44%	41%	40%
finalAssembly_1	26%	45%	27%	41%	32%	42%	30%	44%	28%	59%
finalAssembly_2	30%	52%	29%	45%	37%	47%	35%	52%	29%	65%
finalAssembly_3	26%	45%	27%	43%	33%	42%	30%	44%	26%	62%
finalAssembly_4	34%	60%	33%	53%	41%	52%	40%	60%	31%	71%
finalAssembly_5	31%	53%	29%	47%	35%	43%	35%	53%	29%	70%
finalAssembly_6	29%	51%	30%	49%	34%	42%	32%	50%	28%	72%
finalAssembly_7	29%	51%	29%	48%	34%	40%	32%	50%	28%	76%

2. Job sequencing policy:
   Several jobs have long processing times on the assembly line. The current scheduling policy is set to dispatch such jobs at random times in between other jobs that have normal processing times.

   2.1. Experimental setup:
   For a given batch of products to schedule, we label the jobs with high processing times (60 minutes) on the assembly line with the subscript *long*, and the jobs with normal processing times with the subscript *normal*. Since we know that the current policy is to mix both types of jobs, we wanted to see if the optimal schedule will choose to grope "*long*" jobs together, or scatter them in between "*normal*" jobs.

   2.2. Results:
   The numbers in Table 3 represent the optimal dispatching sequence of "long" jobs. Other jobs with "normal" processing times are not shown. The total number of jobs scheduled in each instance is a random number between 10 and 12. Note that these numbers are all consecutive, it indicates that grouping jobs with high processing times on the assembly line is a better

scheduling policy, compared to dispatching them at random in between other jobs with “normal” processing times.

Table 3: optimal dispatching sequence of jobs with high processing times at the assembly line, for various days in weeks 10-47. Note that all sequences are consecutive.

Week / Product	W-10	W-24	W-47	W-12	W-13	W-13	W-14	W-14
IRB460_rep0					17			14
IRB6700_rep2	15		16					
IRB6620_rep0	11		14		14			
IRB6650S_rep0			13			10		
IRB6700_rep0	14		15		15		13	12
IRB7600_rep0	10		12		13			11
IRB6700_rep5		12		12				
IRB6700_rep6		13		11				
IRB6700_rep4	12	14	18	13				
IRB6700_rep3		15	17	14		11	12	
IRB6700_rep1	13	16				12		
IRB660_rep0		17	19	15	16		14	13

3. Processing times variation:
   Due to the variation in the design processing times of different products on different machines, we observed high waiting times for products due to blockage, even when the jobs are dispatched in an optimal manner. To put things into perspective, producing a batch of 18 products can expect to have about 55% of workstation idle time due to process time variation. This is based on all the test cases and assumes an optimal dispatch of products. Such an insight motivated the team to investigate the potential gain from reducing the variation in processing times of different products on different workstations. Due to engineering restrictions, some processing times cannot be changed, so we only propose guidelines to what these times can be, and the corresponding gain in workstation utilization, and hence throughput, if these guidelines are followed. Due to time limitations, this last avenue is still be explored and no results are available at this time, though the production team agreed to continue developing this even after the project ends.

Implementation:

The above model and experiments were implemented in C++ with CPLEX as the main modeling and optimization package. The code is submitted with this report and can be used for further development.

Operating the code is simple, it reads an *.csv file that stores the dependency relations between jobs and all the parameters needed for the model. The file is named: `dependencyMatrix`. Refer to the model section for more details. It then outputs a solution detailing the optimal schedule as a Gantt chart, along with some performance measures like the makespan and idle time.

All experiments are controlled from the `main` file and are activated by changing the corresponding Boolean variables to true instead of their default false values. For example, if we are to evaluate a given job dispatching sequence to determine its makespan and workstation utilization, we set the `expEvalSol` variable to true and the code will take care of the rest. For this specific experiment, one needs to provide the sequence to be evaluated; it needs to be in the same format another input file that is provided with this report named: `solutionAsInput`.

The code itself is well documented and guides the user as to how to use it. Still, if any questions arise, please contact the developer at jawad.elomari@ri.se.