Analysis of Input Data

Best & Worst Case Scenarios:

The first step taken was to evaluate the best case and worst case cycle time performance for each cell and each part type, by using the given process time distributions. All distributions were triangular with a given 'minimum', 'mode', and 'maximum' as parameters. The best case performance was computed by the sum of the max 'minimum' parameters across all sequences, for each cell and for each part type. The worst case was computed by the sum of the max 'maximum' parameters across all sequences, for each cell and for each part type. The results are shown below in Table 1, the units are in days. The results show that Cell 2 is expected to be the primary source of traveled work. This is because the current state cycle time is 4 days and the future state cycle time is 3.5 days.

Table 1: Worst & Best Case Cycle Times

Cell	PartType 1		PartType 2	
	Worst Case	Best Case	Worst Case	Best Case
1	3.68	2.92	3.11	2.31
2	4.84	3.62	4.85	3.51
3	3.20	2.06	3.32	2.54
4	3.53	2.75	3.19	2.43
5	0.29	0.24	0.29	0.24

Finalized Data Table:

The finalized data table included all the data given initially as well as other factors including: Task Event, Successor Task Event, Worker Node, and Work Station. All factors (ie. columns) in the Data Table 'Data' were key for each work process to run properly..

Modeling Approach

The intent of the approach was simplicity. This was primarily achieved by using two tokens as input arguments for generalized add-on processes.

Physical Layout & Objects:

The model includes 1 source to create plane entities. There are 83 sources to create each particular work entity after its processing time is complete. There is 1 source to create wing entities. There are 5 combiners to combine on-time entities with their plane entity at the end of each cell. There are 3 distinct networks of time paths and nodes, one for plane entities to ride on, one for on-time work/wing entities to ride on, and one for late work entities to ride on. There are 5 workers for each cell, with different population values. There is 1 resource to represent the crane required for Cell 3. Finally, there is one sink to destroy plane, work, and wing entities.

Processing Logic:

The processing logic can be found in two add-on processes: ‘WorkProcess’ and ‘LearningCurveEffect’. Starting with ‘WorkProcess’, this is assigned to each of the 83 sources that create work entities. This add-on process is run by a token ‘TaskToken’ which requires 4 input arguments: Task Number, System Sequence Number, Cell Number, and Cell Worker. The use of these input arguments is to access a particular row-column intersection of ‘Data’ or any of the various array based state variables. The first step is a seize that uses the Cell Worker input to seize that type of worker, sends the Cell Worker to a destination node that is extracted from ‘Data’ with the Task Number input as the row in the ‘WorkerNode’ column. The number of workers required is extracted the same way, but using the ‘Workers’ column. Once a worker is seized, ‘LearningCurveEffect’ is executed to compute the percent increase that will be applied to the processing time from that worker. This computation will be explained below in the Learning Curve Logic section. The next seize is for the crane, and it is seized the same way as the workers.

The work entity being created is assigned an identification number, and a part type. The next step will sum up the total learning curve effects of all workers that are seized, via the four state variables: ‘Resources’ (1) to increment how many workers have been summed, ‘CurrentWorker’ (2) to create the indices that extract the current learning curve effect found in ‘LearningCurve’ (3) and then added to ‘Effects’ (4) which will be the scalar multiplier for the upcoming processing time. The decide step acts as a loop that is broken once all of the workers required for this work entity have had their learning curve effects added to ‘Effects’. The counter ‘Resources’ is reset for the next time this process is triggered. The state variable ‘Effects’ is then divided by the number of workers required for this work entity to average the learning curve effects that will impact the upcoming process time.

The next decide step directs the entity, based on its part type, to its process time. The delay step acts as the processing time which is scaled based on state variable ‘Effects’. The release step relieves all resources. The next assign step resets ‘Effects’ for the next time this process is triggered. The next assign step increments state variable ‘Tasks’ to count how many tasks within this particular sequence have been completed. The decide step evaluates to TRUE when all tasks for the particular sequence have been completed. The next assign step resets ‘Tasks’ for the next time this process is triggered. The next decide step evaluates to TRUE for all tasks, unless it is the last task in the system (because it is the last sequence). The next assign step gives the next sequence its entities’ part type in the state variable ‘PartType’ which allowed the second assign step in this add-on process to give the work entity its proper part type. Finally the successor event is fired to trigger the next sequence of sources to run their own version of this add-on process based on their unique set of input arguments.

Plane Logic:

The plane logic can be found in the ‘PlaneArrives’ add-on process which is run by the plane source, and each of the combiners. The first assign step increments how many arrivals there have been in the particular cell. The decide step evaluates to FALSE for all planes unless it is the (n + 1) plane. The value in creating (n + 1) planes is required for the manner in which traveled work is evaluated, this will be explain below in “Verification & Validation’. The next decide step evaluates to TRUE for a plane that arrives in Cell 1 to then decide it's part type which is shown by the following decide step and assign steps. The next assign step gives the particular cell’s first sequence its entities’ part type in the state variable ‘PartType’. The next fire step fires an event to trigger the sources in the cell’s first sequence, to run the ‘WorkProcess’ add-on process.

Traveled Work Logic:

The traveled work logic is based on the global state variable ‘Arrivals’ and model entity state variable ‘JobNum’. When a work entity travels its network, approaching its end of cell combiner, it evaluates whether or not it’s ‘JobNum’ is equivalent to the cells’ current ‘Arrival’ value via link weights. An example from the work entity point of view: if 3 planes have arrived in the cell, and your ‘JobNum’ value is 3 then your plane is still in the cell and you’re on time, so you go wait in the member input queue of the combiner. If more than 3 planes have arrived in the cell, and your ‘JobNum’ value is 3, then your plane must be in the next cell, so you bypass the combiner and head straight for the next end of cell combiner. The ‘JobNum’ value can never be less than the ‘Arrival’ value because the plane entity is created first, and increments ‘Arrival’ before the work entity can increment its ‘JobNum’. The ‘Arrival’ value can never be greater than ‘JobNum’ by more than a value of 1 because the processing time of work entities in terms of hours, whereas the travel time of planes is in terms of days. Therefore, traveled work will only travel to the next cell and make it to its correct plane.

Learning Curve Logic:

The learning curve logic can be found in three add-on processes: ‘IncrementUnitsMade’, ‘InitializeLaborRate’, and ‘LearningCurveEffect’, which are all applied to each worker. The equation used to compute learning curve is shown below in Figure 1.

$$T_n = T_1 n^r$$

T_n = time required to complete the n^{th} unit
T_1 = time required to complete the first unit
r = log(learning rate)/log(2)

Figure 1: Learning Curve Equation for Wright’s Model

When the simulation run is initialized the process ‘InitializeLaborRate’ is run. The decide step evaluates whether or not the worker is new (ie. a population index greater than 8). The assign step for the TRUE path assigns an labor rate value based on a triangular distribution with parameters 1.5, 1.75, and 2 to indicate a new worker will perform a cell task 50%, 75%, or 100% slower. This ‘LaborRate’ value represents ‘T1’ in Figure 1. The assign step for the FALSE path assigns a labor rate value of 1 because current workers (ie. population index <= 8) don’t scale up the processing time of any task in their cell.

When a worker is released the ‘IncrementUnitsMade’ process is run. The decide step is TRUE only for new workers, and then uses the assign step to increment the state variable ‘UnitsMade’ for that particular new worker.

When a worker is seized in the ‘WorkProcess’, the ‘LearningCurveEffect’ process is executed as stated in ‘Processing Logic’. In the assign step, the state variable ‘LearningCurve’ accesses an entry corresponding to the particular worker in ‘LaborRate’ and ‘UnitsMade’ which correspond to ‘T1’ and ‘n’ respectively in Figure 1. The ‘learning rate’ in Figure 1 is 0.90, to compute ‘r’, as that was given in the data. Finally, the computation is the maximum value between 1 and the computation of ‘Tn’ shown in Figure 1. The maximum value computation is necessary because the value of ‘Tn’ will eventually drive below a value of 1 after enough units have been made.

Wing Inventory Control Logic:

The wing inventory control logic can be found in the add-on process ‘WingInventoryControl’. This is a simple add-on process that tells a wing entity to wait at the node ‘WingStorage’ which has a traveler capacity of 1. The wing entity is waiting for the firing of the event for the first sequence in cell 3. The reasoning for such a simple piece of logic is because the arrival rate of wings within 6 normal standard deviations is lower than the 4 day cycle time. Also, the wing arrival rate within 4.6 normal standard deviations is lower than the 3.5 day cycle time. This shows that the statistical possibility of multiple wings arriving quicker than their corresponding planes is significantly impractical. Furthermore, the first plane must go through two cycle times before the first wing entity moves from the ‘WingStorage’ node, guaranteeing that there will always be wings available for the first 6 tasks of Cell 3.

Verification & Validation

The verification and validation of the model was accomplished via status plots. The contents of the member input buffers for work entities in each of the 5 combiners were plotted in a time series. The contents of the parent input buffer for plane entities in the last combiner were plotted in a time series. A global state variable ‘TotalWork’ extracts the number of batched members on each plane after it leaves each combiner, were plotted in time series. A global state variable ‘TravelWork’ counts cumulatively, the number of traveled work entities for Cells 1 - 4, were plotted in time series. The utilization of each worker in each cell were plotted in time series.

The usage of these plots allowed for abnormalities to be seen, and for patterns to be seen. Observations that were necessary to see was that a plane never entered the parent input buffer of the last combiner because this would indicate a late plane. Another observation to see was that the work entities in the member input buffers of the combiners followed a clear visible pattern without any extreme jumps. An observation to see was that the state variables ‘TotalWork’ and ‘TraveledWork’ reacted to changes at the same time instances in the time series. For example, if at a particular point there was a jump in traveled work, then there should be a dip in total work on that plane. The last observation to see was that the utilizations of each worker converge to a particular window of percentages, in each of the cells.

The most valuable abnormality found was in the parent input buffer of the last combiner. The very last plane would always go into the queue because the remaining entities to complete the plane weren’t there. Once the simulation ended, there were some work entities waiting in the member input buffer of cell 2, these were the same work entities meant for the plane waiting in cell 5. This is because the decision for a work entity being late or on time for its plane, was determined by the arrival of the next plane. There was no (n + 1) plane in place to arrive and tell these late work entities to travel to the next cell. The production requirements call for (n) planes, therefore an (n + 1) dummy plane that doesn’t trigger work processes must be made. Hence the first decide step in the add-on process ‘PlaneArrival’.

Once all of the desired observations were set in place, experimentation and goal setting could begin.

Experiments & Goals

The time series of the contents in the parent input buffer for plane entities in the last combiner, for current state and future state, showed that 8 workers in each cell always meets the desired cycle time. This observation set the first goal for our proposed process improvement: minimize the number of workers while still meeting the desired cycle times. The next goal chosen is to minimize the total traveled work, as to reduce the risk of a late plane. Therefore the overall goal was to find a balance between worker reduction and traveled work reduction while guaranteeing all planes meet the desired cycle times for the current state and future state.

The experiment method of choice was a general factorial design. The factors in this experiment would be the 5 different cell worker types, and the levels would be the number of workers for each cell worker type. The levels for each cell worker type were defined by a range [the max workers for a single task, 8 workers]. Table 2 below shows these factors and levels.

Table 2: General Factorial Parameters

Factor	Min	Max
Cell1Workers	5	8
Cell2Workers	4	8
Cell3Workers	5	8
Cell4Workers	4	8
Cell5Workers	3	8

Analysis of Results

The entire design and results that lead to our final solution can be found in the spreadsheet DOE.xslx.

General Factorial Design of Experiment:

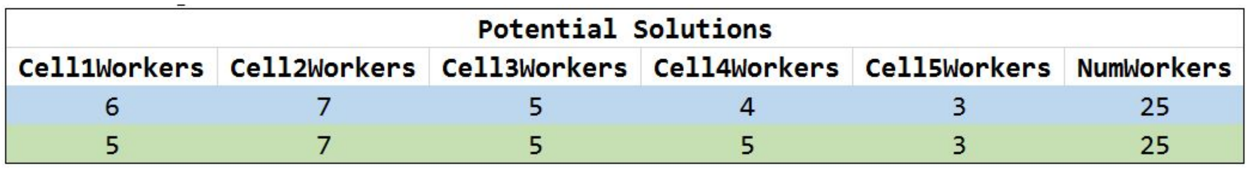
This experiment required 2400 runs, due to this expensive demand each run was run just once. The value in doing one replication immediately allows many factorial combinations to be eliminated quickly, because it only takes one replication with at least one late plane to be removed from consideration.

The experiment was run and a valuable pattern was found, each combination that created at least one late plane for the current state also created at least one late plane for the future state. The converse is true where each combination that didn't create at least one late plane for the current state also did not create at least one late plane for the future state. After filtering out the combinations that violated the late plane constraint, there were 1920 combinations left over. An additional filter based on the total number of workers was applied initially for at most 30 total workers, and ultimately a filter of at most 25 total workers was chosen. This created 56 combinations to evaluate.

The next experiment was run with a focus on total traveled work while not creating late planes, and 10 replications were run for each of the remaining 56 combinations. The results of this experiment also offered another convenient pattern between current state and future state traveled work. The sorted current state traveled work values showed a significant break between two combinations, a traveled work value of 243.8 units to a traveled work value of 419.6 units. The sorted future state traveled work values showed a significant break between two combinations, a traveled work value of 1106.9 units to a traveled work value of 1966.3 units. The combinations for current state, with a traveled work value of 243.8 units or less, and the combinations for future state, with a traveled work value of 1106.9 units or less, were the same combinations. These two breaks created 21 combinations to evaluate.

The next experiment was run with a focus on total traveled work while not creating late planes, and 50 replications were run for each of the remaining 21 combinations. The results of this experiment offered no significant breaks in the current state sorted traveled work values. The results of this experiment offered a significant break in the future state sorted traveled work values between 717.5 units and 833.58 units. The combinations for future state, with a traveled work value of 717.5 units or less created 2 combinations to evaluate. These two combinations differed slightly, as shown below in Table 3.

Table 3: Potential Solutions



Potential Solutions					
Cell1Workers	Cell2Workers	Cell3Workers	Cell4Workers	Cell5Workers	NumWorkers
6	7	5	4	3	25
5	7	5	5	3	25

Graphical Analysis:

The graphics explained above in ‘Verification & Validation’ were analyzed and compared between the two potential solutions shown above in Table 3. The following figures below were the determining factors in choosing potential solution 2, represented by the green row in Table 3. These graphics show the number of work entities waiting in the queue for their plane to arrive at the end of cell 5 combiner to then be sent to finished goods inventory. The less work entities in this queue means the more work entities already on the plane. The more work entities on the plane as it approaches the end of cell 5 combiner, the less risk of there being a late plane. The trends in Figure 3 are more promising, therefore potential solution 2 was chosen as the final solution.

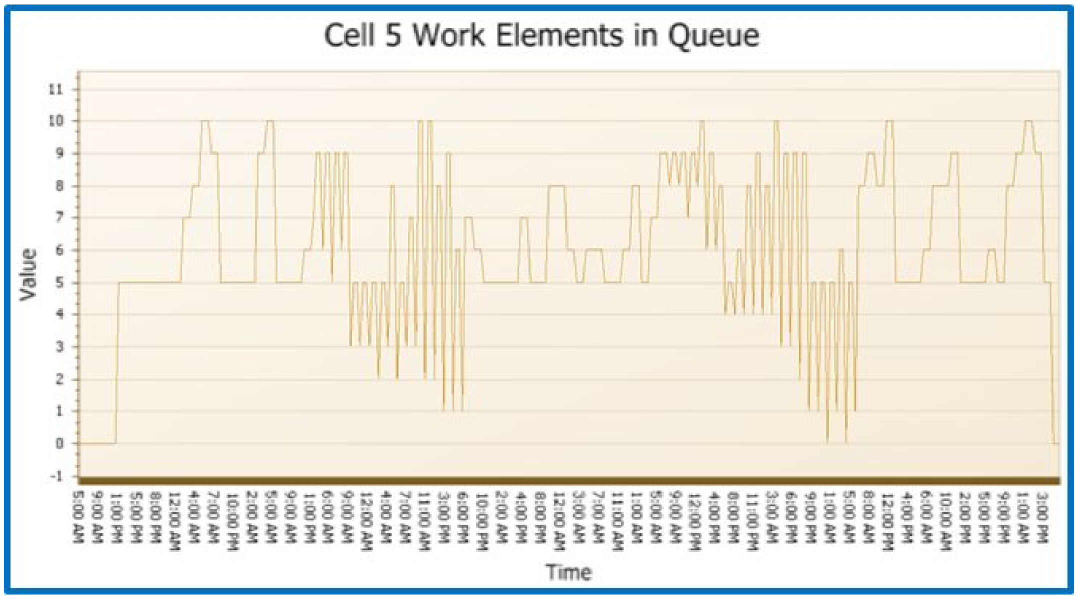


Figure 2: Potential Solution 1 - Work Elements in Queue Time Series

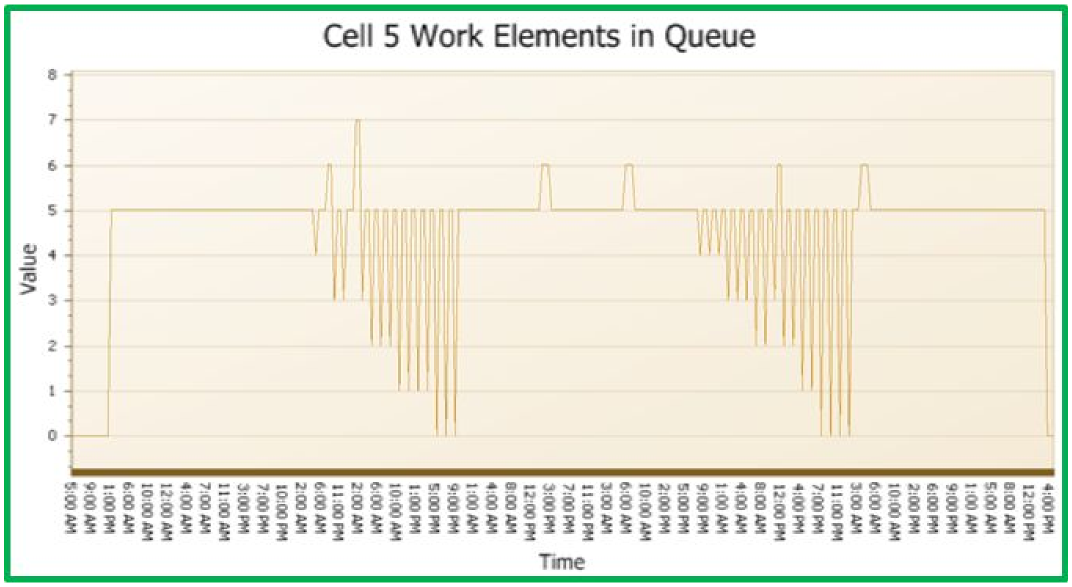


Figure 3: Potential Solution 2 - Work Elements in Queue Time Series