TECHNICAL RESEARCH REPORT

A Tool Optimization Interface for a Semiconductor Manufacturing System

By Ryan Thomas
Advised by Dr. Jeffrey Herrmann

August 18, 2000

<u>Contents</u>

A Tool Optimization Interface for a Semiconductor Manufacturing System

Ryan Thomas
August 16, 2000

Introduction / Abstract

This paper will serve as the documentation for the Tool Optimization code of the HSE software. The purpose of the software is, simply, to enable a user to optimize a factory's tool selection. This will be added to the existing Factory Administrator which enables users to understand the effects of changes in many parts of the manufacturing process (i.e. Temperatures, Pressures, etc.).

To accomplish this an interface was designed via the DELPHI programming language that can take inputs from a user as well as factory details from an Excel spreadsheet, run simulations, determine an optimal tool configuration, and output this data as easily as possible to the user.

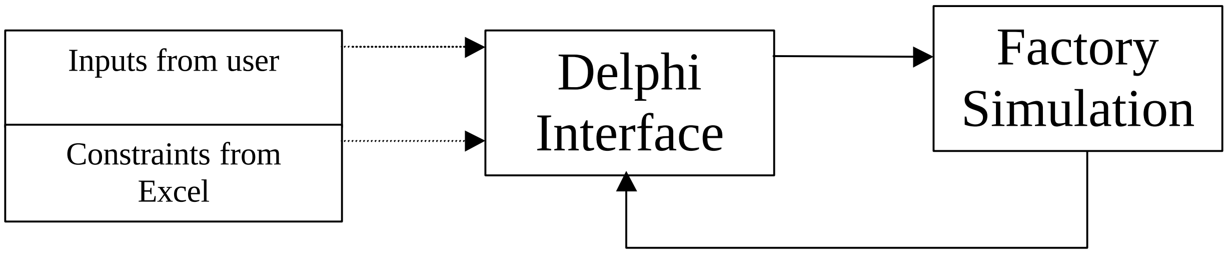


Figure 1

This process is shown above in Figure 1. The Interface will guide the Simulation as many times as needed to perform its gradient analysis. After the program is complete, it determines a best-case tool configuration that meets the user's throughput while maintaining to his budget. The interface will output how many of each tool to purchase as well the best possible tool allocation (usage) for each tool.

Algorithm

The algorithm will be explained in two parts. The first part will be the initial heuristic. During the heuristic, the optimization program will determine if a system exists that can meet throughput demands while performing under a maximum cost (budget). If an acceptable system exists, the heuristic will determine the minimum number of each tools and how each tool should be allocated The second part will be a "search" algorithm to determine the best use of any money remaining after the initial purchases are made by the heuristic. The second part will entail controlling a simulation loop to find exact cycle times of the system.

Part 1
Step 1: Read in all inputs

The interface takes direct inputs for throughput (λ, wafers per hour) and budget. In addition, it will read processing times, tool costs, and existing tools from an Excel Spreadsheet. Each tool and its characteristics will be stored based on its type. For example, the number of each type of tool is necessary and stored in an array called NumCount[i]. The following is a list of variable initialized by this step and their notations.

NumCount[i] : The number of tools for each step. i = 1 represents clean tools, and so on.
NumProc[i] : The number of processes that a group of tools need to perform.
Cap[i,j,k] : Capacity of a tool for a process. Represents the capacity of tool j for the k^{th} process of type i.

Step 2: Determine capacities, U, and ranks
Step 2a: Finding U

Each tool will have a known capacity for each process. For example, if there are three clean processes for which a certain tool could be used, the tool would have three capacities representing the number of wafers that it could process for each of the cycles. Using this capacity, we can find a cost-efficiency ratio, U.

$$U[i,j,k] = Cap[i,j,k] / Cost[i,j]$$

Note that cost is independent of process and only takes on a two dimensional array. After this calculation, we have three matrices (one for Clean, Ti Liner, and W CVD) of cost-efficiencies for each tool at each process.

Step 2b: Finding HighU

It is obvious that in trying to optimize tool selection we want the most bang for the buck, or the most production for each dollar. Assuming this to be true it is important to know which, of the possible tools for each process, can complete the process's throughput at the lowest cost. To do this, we will find the highest U for each process, and the respective tool(s) that can produce that U.

Once this is complete, we have three lists (not matrices this time) of Uhigh[i,k]. That is, when i = 1, we have a list of x (where x = NumProc[1]) entries that represent the highest possible U for that process. Additionally, we have a corresponding list x entries long of UhighTool[i,k] that represent the high tool number for the j^{th} process of type i.

Step 2c: Ordering HighU

As stated above, the purpose of the heuristic is find the minimum solution to the problem: the least expensive configuration that still meets the throughput constraint. In order to this, we may need tools to operate on wafers during more than one cycle. A tool that "overlaps" (that is, used for more than one cycle) cycles will have more capacity than it was originally needed. However, this extra capacity may create a situation where less of another tool is needed. In order to select which tool(s) may overlap, we want to know which efficiency is the most efficient. Therefore, the program will rank the efficiencies of all of the HighU's, and this will be the order in which tool are purchased.

Step 3: Purchasing the minimum number of tools

Using the Process order found in Step 2c, the program will purchase the necessary tools for the first process. This is an easy calculation since the number of tools (X) simply equals:

$$\text{For each Process, where } j = HighTool,\ X[i,j,k] = l / Cap[i,j,k]$$

Where l represents desired throughput and X is rounded up to the next integer. This expression then tells us that the number of tools j needed of type i to complete the k^{th} process is throughput divided by the capacity for that tool for that process.

In case the capacity of these purchased tools exceed the required throughput, any extra time that it may have (after completing the cycle it was just purchased for) is dedicated to the cycle of next highest rank. The throughput that needs to be met for this cycle is now the original throughput minus any production that this last purchase can accomplish.

$$l_{new} = l - \{[\ 1 - (X[i,j,k] * l / Cap[i,j,k])] * Cap[i,j,k+1]\}$$

where k + 1 represents the cycle of next highest rank. From this, the program is able to purchase the next set of tools using the above equation for X, but l is now l_{new}. This is repeated for all processes.

Step 4: Tool Allocation

Now that we know the minimum number of tools to meet throughput, the program has to configure them to optimize cycle time. In order to run the simulation, Factory Explorer (FX) needs to be told what percentage of each process each tool will complete. For example, if tools 1 and 2 are going to be used for a given process tool 1 will clean twice as many wafers in this cycle than tool 2, the FX times are 66.7% and 33.3%, respectively. If these tools are then used for multiple other cycles, it can be very complicated for the user to determine what percentage of a tool's time is actually being spent on each process. For this reason, the program exports time allocations in both formats. The interface reads the percentage of a tools time to be allocated per process and the spreadsheet takes the percentages that represent a ratio of the cycle's throughput. Additionally, the program must account for the cases where a tool has the highest efficiencies for more than one process or assumes the role of an "overlap" tool. Therefore, we are able to find a percentage of time of the tools that are purchased for a cycle to do that job.

$$Time[i,j,k] = l / X[i,j,k] * Cap[i,j,k]$$
And
$$1 - Time[i,j,k] = MoreTime[i,j,k+1]$$

Once we have purchased tools for every process, we must sum the number of the tools of each type. For a given process, Time is the same as above multiplied by the ratio of tools dedicated to that process over the sum of the tools. The following is the conversion for the Time variable.

Putting all of this together, we can calculate an actual time based on the sum of the tools of a certain type, an MoreTime that was allocated to the process, and the leftover time from the last process which is divided into all processes, again where X[i,j] is the sum of all tools j.

$$ActualTime[i,j,k] = \{ Time[i,j,k] * X[i,j,k] / X[i,j]\} + \{MoreTime[i,j,k-1] * X[i,j,k-1] / X[i,j]\} + \{MoreTime[i,j,lastprocess] * X[i,j,lastprocess] /(NumProc[i] * X[i,j])\}$$

Where Σ X[i,j,k] is the sum of X for a given tool over all j processes. This number is then converted to a time in terms of each process for the spreadsheet. To do this, we must know a potential sum of all capacities (how many wafers could be processed by all tools in their current dedications). We can then multiply the Time from above by the ratio of the capacity for that process (and tool) over the sum of capacities:

$$Available\ Capacity[i,j,k] = ActualTime[i,j,k] * X[i,j] * Cap[i,j,k]$$
and
$$FXFlow\%[i,j,k] = 100\% * \{Available\ Capacity[i,j,k] / S\ Available\ Capacity[i,j,k] \}$$

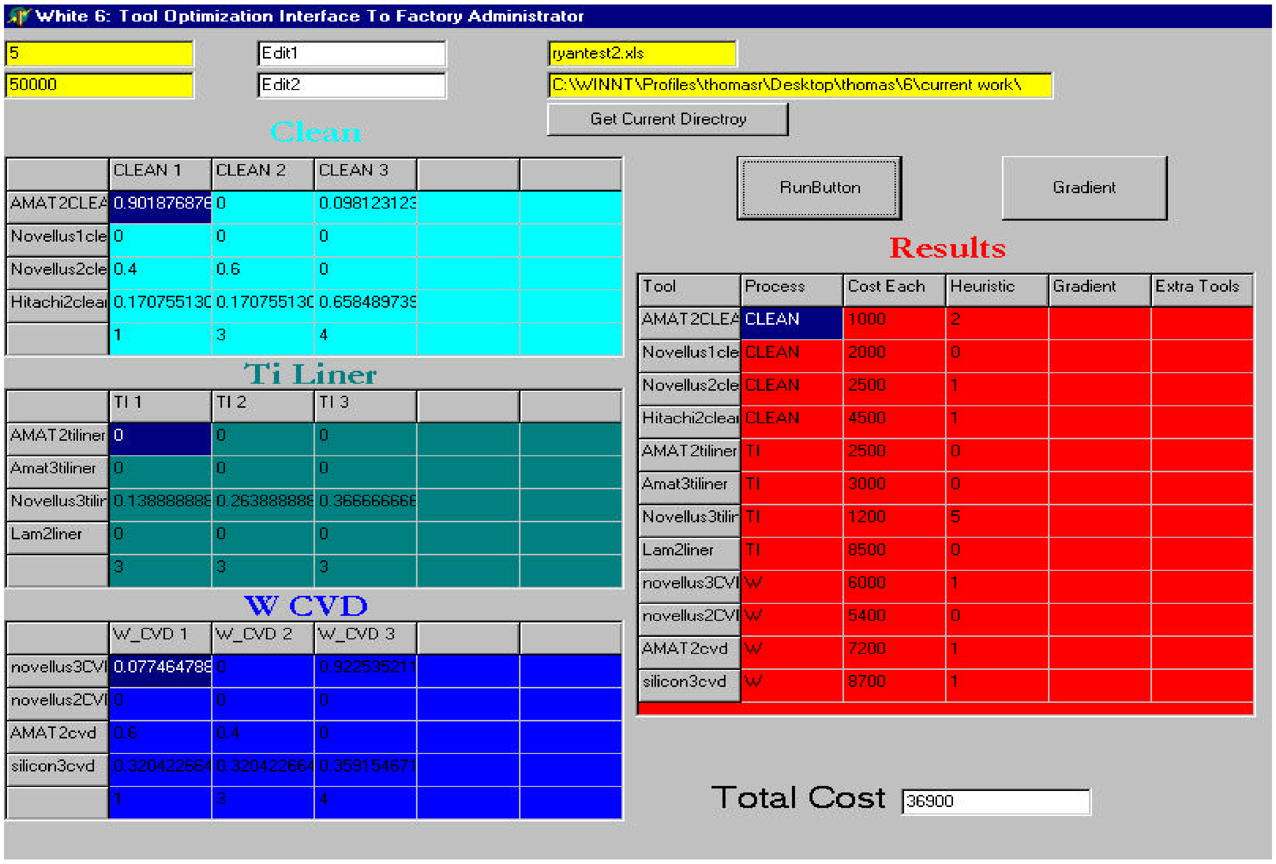
Step 5: Output and Error Check

Having determined the minimum combinations of tools to purchase and the best allocation for these tools, this data is now output to the user via the programs interface. A copy of this data is also sent to the Excel Spreadsheet for future use in Factory Explorer. This information in automatically saved so a user could access it later or compare its results to another tool configuration. Additionally, this info, along with the tool costs allows us to find a Minimum Cost for the system.

$$TotalCost = S \{X[i,j,k] * Costi[i,j,k]\}$$

Should the minimum configuration cost exceed the budget, then it can be determined that there does not exist a possible tool configuration that can meet throughput. If this is the case, the program recognizes this fact and alerts the user to his error.

After the Heuristic is run, the interface will appear as shown below. The red "Results" section on the right displays the given information of a tool (name, use, and cost) as well as the total number of tools purchased during the heuristic. The matrices on the left represent each tool group (Clean, Ti Liner, W CVD). Each number is ratio of the tool's total time spent on each process. If a tool is not used for a process, the number reads zero. The yellow boxes on top of the interface represent locations for the user's input. As discussed earlier, a user will enter a desired throughput, a maximum budget and the name and location of the Excel file where tool data can be found.



White 6: Tool Optimization Interface To Factory Administrator

5	Edit1
50000	Edit2
ryantest2.xls	
C:\WINNT\Profiles\thomasr\Desktop\thomas\6\current work\	

Clean

	CLEAN 1	CLEAN 2	CLEAN 3		
AMAT2CLEA	0.90187687€	0	0.09812312:		
Novellus1cle	0	0	0		
Novellus2cle	0.4	0.6	0		
Hitachi2clea	0.17075513(	0.17075513(	0.65848973§		
	1	3	4		

Ti Liner

	TI 1	TI 2	TI 3		
AMAT2tiliner	0	0	0		
Amat3tiliner	0	0	0		
Novellus3tilir	0.13888888€	0.26388888€	0.36666666€		
Lam2liner	0	0	0		
	3	3	3		

W CVD

	W_CVD 1	W_CVD 2	W_CVD 3		
novellus3CVI	0.07746478€	0	0.922535211		
novellus2CVI	0	0	0		
AMAT2cvd	0.6	0.4	0		
silicon3cvd	0.320422664	0.320422664	0.359154671		
	1	3	4		

Results

Tool	Process	Cost Each	Heuristic	Gradient	Extra Tools
AMAT2CLEA	CLEAN	1000	2		
Novellus1cle	CLEAN	2000	0		
Novellus2cle	CLEAN	2500	1		
Hitachi2clea	CLEAN	4500	1		
AMAT2tiliner	TI	2500	0		
Amat3tiliner	TI	3000	0		
Novellus3tilir	TI	1200	5		
Lam2liner	TI	8500	0		
novellus3CVI	W	6000	1		
novellus2CVI	W	5400	0		
AMAT2cvd	W	7200	1		
silicon3cvd	W	8700	1		

Total Cost 36900

Part II
Step 1: Find heuristic's cycle time[1]

The heuristic being complete, the program has found the absolute minimum combination of tools that can meet throughput. Additionally, it has alerted the user that there is extra money to be used to improve cycle time. Currently (with such a low number of tools) it is common to have extremely high cycle times. In the example that follows, we will see that the heuristic provides a minimum solution that takes a cycle time of almost 200 days!

To find the "initial" cycle time, the program runs Factory Explorer with the tool configurations that it found. This "initial" time will be referred to as the Lower Cycle Time. The program will then run the simulation with the same figures except one more of the first tool, and then one more of the second tool, and so on for every tool. From these simulations the program will read in the cycle times of the system if

[1] It should be noted that there are several methods for gradient estimation. The one used in the Tool Optimization software is the Forward Difference Method. These methods were researched by Praveen V. Mellacheruvu and the FD Method was chosen as the most practical for this application. For more information regarding the Method, reference *Optimization of Tool Configurations for a semiconductor Manufacturing System*, Mellacheruvu, Fu, Herrmann, June 2000.

one more of each tool were added to the system. Using these cycle times, we are able to calculate a gradient using the equation for the Forward Difference Gradient Method.

For our purposes, and since the Lower Cycle Time is constant, we can simplify the equation to

$$g(Q) = 1 / N * S \text{ (Higher Cycle Time – Lower Cycle Time)}$$

N represents the number of repetitions for the simulation. This makes the gradient an average of the differences between the Lower Cycle Time and the Higher Cycle time for each tool. These gradients can then be used to add tools to the system under the budget constraint.

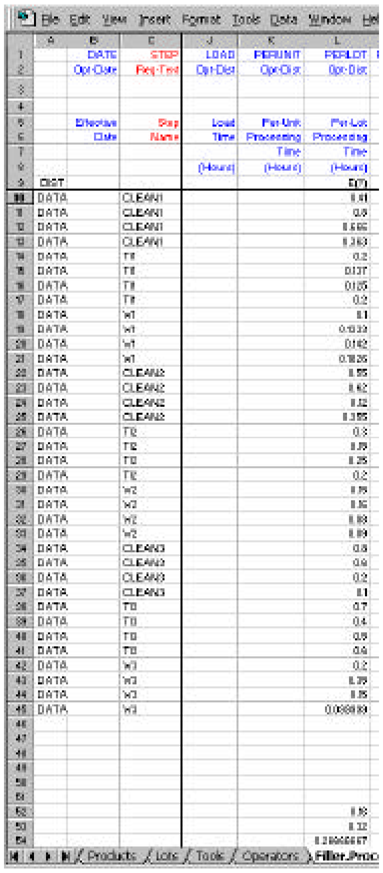
Step 2: Adding Tools Based on Gradient Analysis

The Program will now have a matrix containing the gradient for each tool if it were added to the system. The program will add one tool of the type with the highest gradient. If there is room under the budget, the program will add one tool for a different process (again with the highest gradient). If there is still room in the budget, a third tool (of the third type) is added.

If the budget is large enough that there is still room in the budget to add tools, the gradient process is repeated. It is too difficult to estimate how the gradient analysis will react after one tool of each type is added. This loop is continued until there is no longer room under the budget to purchase new tools. If the first tool is too expensive, the program will substitute a less expensive tool of the same type if one exists with a positive gradient. If no more tools can be added, the simulation is run once more for a final cycle time.

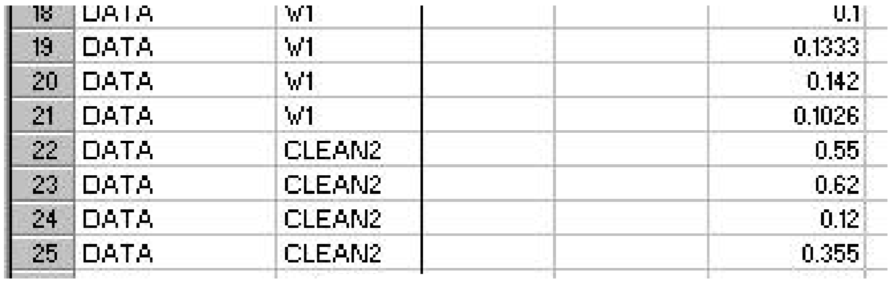
Example

The example we will consider consists of 12 possible tools, 4 of each type. They will be used in a manufacturing set-up that performs each operation (Clean, Ti Liner, and W CVD) three times.



	A	B	C	J	K	L
1		DATE	STEP	LOAD	PERUNIT	PERLOT
2		Opt-Date	Req-Text	Opt-Dist	Opt-Dist	Opt-Dist
3						
4						
5		Effective	Step	Load	Per-Unit	Per-Lot
6		Date	Name	Time	Processing	Processing
7					Time	Time
8				(Hours)	(Hours)	(Hours)
9	DIST					E(7)
10	DATA		CLEAN1			0.41
11	DATA		CLEAN1			0.9
12	DATA		CLEAN1			0.666
13	DATA		CLEAN1			0.263
14	DATA		TI1			0.2
15	DATA		TI1			0.137
16	DATA		TI1			0.125
17	DATA		TI1			0.2
18	DATA		W1			0.1
19	DATA		W1			0.1333
20	DATA		W1			0.142
21	DATA		W1			0.1026
22	DATA		CLEAN2			0.55
23	DATA		CLEAN2			0.62
24	DATA		CLEAN2			0.12
25	DATA		CLEAN2			0.355
26	DATA		TI2			0.3
27	DATA		TI2			0.19
28	DATA		TI2			0.25
29	DATA		TI2			0.2
30	DATA		W2			0.15
31	DATA		W2			0.16
32	DATA		W2			0.08
33	DATA		W2			0.09
34	DATA		CLEAN3			0.8
35	DATA		CLEAN3			0.6
36	DATA		CLEAN3			0.2
37	DATA		CLEAN3			0.1
38	DATA		TI3			0.7
39	DATA		TI3			0.4
40	DATA		TI3			0.9
41	DATA		TI3			0.6
42	DATA		W3			0.2
43	DATA		W3			0.39
44	DATA		W3			0.15
45	DATA		W3			0.008889
46						
47						
48						
49						
50						
51						
52						0.16
53						0.12
54						0.28666667

Figure 2a



18	DATA	W1			0.1
19	DATA	W1			0.1333
20	DATA	W1			0.142
21	DATA	W1			0.1026
22	DATA	CLEAN2			0.55
23	DATA	CLEAN2			0.62
24	DATA	CLEAN2			0.12
25	DATA	CLEAN2			0.355

Figure 2b

Figure 2a shows the process sheet with the order of the processes. Factory Explorer and the Tool Optimization Program read in each process until another is listed. In the blown-up picture (Figure 2b) it is easy to see that there are 4 tools available for the CLEAN2 Process. Since Factory Explorer assumes that they are listed in the same order as the tool sheet (shown below in Figure 3), the Program makes the same assumption. This means that for the example, the first tool takes .55 hours to process one wafer, the second takes .62 hours, and so on. Their capacities, then, are the inverse of these numbers.

Both Factory Explorer and the Tool Optimization Program use this sheet to determine the number of cycles for each process.

From the above data, the Program calculates 3 matrices of capacities and cost-efficiencies. Additionally, we are going to assume the user has entered a through put of 5 wafers per hour and a maximum budget of $50,000.

$\lambda = 5$
Money = 50,000

Step1: Read in Inputs

TOOL	COST	TOOL	COST	TOOL	COST
AMAT2CLEAN	1000	AMAT2tiliner	2500	novellus3CVD	6000
Novellus1clean	2000	Amat3tiliner	3000	novellus2CVD	5400
Novellus2clean	2500	Novellus3tiliner	1200	AMAT2cvd	7200
Hitachi2clean	4500	Lam2liner	8500	silicon3cvd	8700

PER LOT PROCESSING TIME

TOOL NAME	CLEAN 1	CLEAN 2	CLEAN 3
AMAT2CLEAN	0.41	0.55	0.8
Novellus1clean	0.8	0.62	0.6
Novellus2clean	0.666	0.12	0.2
Hitachi2clean	0.363	0.355	0.1

TOOL NAME	TI 1	TI 2	TI 3
AMAT2tiliner	0.2	0.3	0.7
Amat3tiliner	0.137	0.15	0.4
Novellus3tiliner	0.125	0.25	0.5
Lam2liner	0.2	0.2	0.6

TOOL NAME	W 1	W 2	W 3
novellus3CVD	0.1	0.15	0.2
novellus2CVD	0.1333	0.16	0.35
AMAT2cvd	0.142	0.08	0.15
silicon3cvd	0.1026	0.09	0.099999

CAP

CLEAN 1	CLEAN 2	CLEAN 3
2.439024	1.818182	1.25
1.25	1.612903	1.666667
1.501502	8.333333	5
2.754821	2.816901	10

TI 1	TI 2	TI 3
5	3.333333	1.428571
7.29927	6.666667	2.5
8	4	2
5	5	1.666667

W 1	W 2	W 3
10	6.666667	5
7.501875	6.25	2.857143
7.042254	12.5	6.666667
9.746589	11.11111	10.0001

Step 2a: Find U's

COST CAPACITY

TOOL NAME	CLEAN 1	CLEAN 2	CLEAN 3
AMAT2CLEAN	0.002439	0.001818	0.00125
Novellus1clean	0.000625	0.000806	0.000833
Novellus2clean	0.000601	0.003333	0.002
Hitachi2clean	0.000612	0.000626	0.002222

TOOL NAME	TI 1	TI 2	TI 3
AMAT2tiliner	0.002	0.001333	0.000571
Amat3tiliner	0.002433	0.002222	0.000833
Novellus3tiliner	0.006667	0.003333	0.001667
Lam2liner	0.000588	0.000588	0.000196

TOOL NAME	W 1	W 2	W 3
Novellus3CVD	0.001667	0.001111	0.000833
Novellus2CVD	0.001389	0.001157	0.000529
AMAT2cvd	0.000978	0.001736	0.000926
Silicon3cvd	0.00112	0.001277	0.001149

Step 2b: Rank U's

The Cost-Capacities are ranked for each process-cycle.

	HIGH TOOL	HIGH U
CLEAN 1	1	0.002439
CLEAN 2	3	0.003333
CLEAN 3	4	0.002222
TI 1	3	0.006667
TI 2	3	0.003333
TI 3	3	0.001667
W 1	1	0.001667
W 2	3	0.001736
W 3	4	0.001149

Step 2c: Order HighU

ORDERS

	CLEAN	TI	W
First	2	1	2
Second	1	2	1
Third	3	3	3

Step 3: Purchase minimum tools

Going in the order found in Step 2c, we start with the clean tools. The first process is CLEAN 2 where tool number 3 is the High Tool. For the 2nd Process, the 3rd tool has a capacity of 8.33. Therefore, we only need to purchase one tool to complete the throughput of 5.

Before we go to the next Cycle (CLEAN 1), we need to find the next throughput.

$$l_{new} = l - \{[1 - l/(Cap[i,j,k] * X[i,j,k])] * Cap[i,j,k+1]\}$$
$$l_{new} = 5 - \{[1-(5/8.33*1)] * 1.50\} = 4.40$$

The high tool for the second Clean Process (CLEAN 1) is tool 1 with a capacity of 2.44.

4.40 / 2.44 = 1.80, which we round up to two. This cycle continues for all cycles of all processes. Note how the program has already cut costs by "overlapping" the 3rd tool purchase for CLEAN 2. If we were to fill a throughput of 5, then we would have needed to purchase 3 of the 1st tool for CLEAN 2. This would have wasted at least $1000. After completing the process, we see that the heuristic finds the following tools:

TOOL PURCHASES

	Tool #	X		Tool #	X		Tool #	X
AMAT2CLEAN	1	2	AMAT2tiliner	1	0	novellus3CVD	1	1
Novellus1clean	2	0	Amat3tiliner	2	0	novellus2CVD	2	0
Novellus2clean	3	1	Novellus3tiliner	3	5	AMAT2cvd	3	1
Hitachi2clean	4	1	Lam2liner	4	0	silicon3cvd	4	1

Step 4: Tool Allocations

Once again, we use the order as found in Step 2c. Beginning with CLEAN 2, we find the minimum time for the tool that was originally dedicated to that process. This is simply a ratio of Potential capacity under needed throughput. For the first tool, of which we purchased one, the time needed to complete CLEAN 2 is:

*5.00 / (1 * 8.33) = .60024*, which means that 60.024% of its operations, this tool is working on CLEAN 2. The other 1 **-** .60024 is allocated to the next cycle. When we reach the last process (such that there is no next cycle for the remaining time to do) the remainder is divided by the number of cycles and shared between them. This will help alleviate the cycle time at all cycles in a process. The process is completed for all processes. For this example we arrive at the following Times:

		Clean1	**Clean2**	**Clean 3**
AMAT2CLEAN	1	0.90187687	0	0.092123123
Novellus1clean	2	0	0	0
Novellus2clean	3	0.4	0.6	0
Hitachi2clean	4	0.17075513	0.17075513	0.65848973
		Ti 1	**Ti2**	**Ti3**
AMAT2tiliner	1	0	0	0
Amat3tiliner	2	0	0	0
Novellus3tiliner	3	0.167	0.292	0.542
Lam2liner	4	0	0	0
		W1	**W2**	**W3**
novellus3CVD	1	0.077464788	0	0.8225321
novellus2CVD	2	0	0	0
AMAT2cvd	3	0.6	0.4	0
silicon3cvd	4	0.3204226	0.3204226	0.35915467

For the clean process, the time allocations were fairly simple since each process had a unique tool with a HighU. This was obvioulsy not the case with the Ti Processes since the same tool actually has the HighU for all three cycles. In this case an initial time was calculated based on the total capacity bought at that cycle divided by the thoughput for that cycle. In other words, each percentage was a percentage of the time of the tools bought by that cycle, not of all of the tools. To convert this to the later we multiply this percentage by the ratio of the number of tools bought by that process over the total number of that tool type.

The program will also calculate percentages for each tool as a ratio of time for each process. To do this it simply takes the above matrix and multiplies each by the capacity dedicated to that process over the total capacity for that process.

For example, for the first cycle of clean, there are three tools that help complete the process: Tool 1, 3, and 4. We know the percentage of each tool's time that it is going to dedicate to this process, the total number of that tool, and the capacity of that tool do complete CLEAN 1. Using this info, we can sum the possible capacity for each tool (and add them to find the sum).

*Sum = (0.90187687 * 2 * 2.439024) + (.4 * 1 * 1.501502) + (0.17075513 * 1 * 2.754821) = 5.4683*

*Capacity for Tool 1 = (0.90187687 * 2 * 2.439024) = 4.39939*

FXFlow% for Tool 1 = 4.39939 / 5.4683 = .8042

Therefore, the FXFlow% for the first tool for the first process is 80.42%. This process is again completed for every cycle. For this example we arrive at the following FXFlow%:

FXFlow %

TOOL NAME	**CLEAN 1**	**CLEAN 2**	**CLEAN 3**
AMAT2CLEAN	80.4219	0	3.591514
Novellus1clean	0	0	0
Novellus2clean	10.9791	91.22422	0
Hitachi2clean	8.599002	8.775777	96.40849

TOOL NAME	**TI 1**	**TI 2**	**TI 3**
AMAT2tiliner	0	0	0
Amat3tiliner	0	0	0
Novellus3tiliner	100	100	100
Lam2liner	0	0	0

TOOL NAME	**W 1**	**W 2**	**W 3**
novellus3CVD	9.536443	0	56.22295
novellus2CVD	0	0	0
AMAT2cvd	52.01696	58.4095	0
silicon3cvd	38.4466	41.5905	43.77705

Step 5: Output and Error Check

All of the above data is outputted to the interface and to Excel. The total cost is

*TotalCost = (2 * 1000) + (1 * 2500) + (1 * 4500) + (5 * 1200) + (1 * 6000) + (1 * 7200) + (1 * 8700)*
Total Cost = $36,900

This is under the budget of $50,000, so no error will appear. There is a total of $13,100 extra.

Part II
Step 1

The Lower Cycle Time for this system was found to be 4315.6 hours, almost 200 days.

Step 2
The following chart displays the Cycle Times and respective gradients for each tool.

	Tool	**Upper Cycle Time (Hours)**	**Lower Cycle Time (Hours)**	**Gradient**	**Cost ($)**
AMAT2CLEAN	1,1	2849.3	4315.6	1466.3	1000
Novellus1clean	1,2	3077.6	4315.6	1238	2000
Novellus2clean	1,3	1831.3	4315.6	2484.3	2500
Hitachi2clean	1,4	1236.1	4315.6	3079.5	4500
AMAT2tiliner	2,1	4300.2	4315.6	15.4	2500
Amat3tiliner	2,2	4315.6	4315.6	0	3000
Novellus3tiliner	2,3	4304.7	4315.6	10.9	1200
Lam2liner	2,4	4308	4315.6	7.6	8500
novellus3CVD	3,1	4310.5	4315.6	5.1	6000
novellus2CVD	3,2	4316.6	4315.6	-1	5400
AMAT2cvd	3,3	4308.8	4315.6	6.8	7200
silicon3cvd	3,4	4384.4	4315.6	-68.8	8700

From this data, the program sees that the greatest gradient (by far) is that of the **Hitachi2clean** tool. It will add one since the cost for one of these is less than the money available. The available money is now

Money = Money – Cost
Money = $13,100 - $4,500 = 8,600

The next tool to purchase is an **AMAT2tiliner** for $2,500: hence we have 6,100. Since the next tool is an **AMAT2cvd**, the program wants to purchase it. However the cost exceeds the remaining money. Because of this condition, the program will look for a less expensive CVD tool with a positive gradient. The next highest gradient belongs to the **Novellus3CVD** which costs $6,000, which is affordable. Since there is only $100 remaining (which is less than the cost for any tool), no more gradients are calculated.

The final number of tools is as follows:

TOOL PURCHASES

	Tool #	X		Tool #	X		Tool #	X
AMAT2CLEAN	1	2	AMAT2tiliner	1	1	novellus3CVD	1	2
Novellus1clean	2	0	Amat3tiliner	2	0	novellus2CVD	2	0
Novellus2clean	3	1	Novellus3tiliner	3	5	AMAT2cvd	3	1
Hitachi2clean	4	2	Lam2liner	4	0	silicon3cvd	4	1

Algorithm Selection

A lot of time and effort was spent trying to optimize both parts of the algorithm. The heuristic has proven to work quite effectively. Obviously there are cases where a solution exists that is better than the one the heuristic would provide. However, the simplicity of the heuristic versus the results it provides is very effective.

The search part of the program, however, is much more difficult to evaluate. Obviously the most effective way to add tools would be to run the gradient analysis after each tool is added. This would eliminate making small jumps where large ones are possible. Additionally, the search algorithm does not take into account the raw cost of the tool. This means that no matter how much more tool A costs than any tool B, as long as it provides a slightly higher gradient, it will be purchased over tool B. This makes initially makes little sense. This strategy (Mellacheruvu) is based on the idea that we want to make the largest leaps in gradients: even though a combination of another tool(s) may create a higher gradient, we do not have data for this. In other words, to ensure that this would be better than purchasing the highest gradient, we would have to simulate every possible combination. This would be incredibly calculation intensive. A few alternate search algorithms are listed below. Pros and Cons for each are discussed.

Current Method: The current method employs performing a gradient analysis and purchasing up to one of each type of tool, then repeating the process until budget is met. The benefit of this method is that for realistic cases it provides relatively accurate outputs and keeps computation times to a minimum.

Every Case Method: The best possible algorithm would be one that considers every tool combination that is able to produce a given throughput while remaining under budget. The program would simulate each of these cases and alert the user of the cost and cycle time of each case. The dilemma for this case is computation. The algorithm for such a program would be fairly simple, however for the example used above with 10 repetitions per case, the program could take several days to run. As the factory becomes larger, this could take weeks, months, or years. A possible solution to this problem is to perform a much more complicated algorithm to narrow the field to, say, the best possible 100 cases. We could run one simulation for each of these, choose the top 10 and run 10 repetitions for each of these. This would mean that there would be 200 runs, which could be completed in approximately 2 hours.[2]

Gradient Every Tool Method: This method is similar to the current method except that it would perform a new gradient analysis for every tool purchased. This may not cause too large of a problem unless the budget is far greater than the cost of the heuristic's solution. In some of these cases, this method could also take up to a day or longer to run.

Cost Based Method: Again, this method is very similar to the current method. The improvement here is that the gradient for each tool is divided by the cost of that tool. In essence the program will purchase tools on a gradient per dollar basis instead of a gradient one. The problem with this solution is that case where two of a tool causes a slower cycle time than does one. This is the problem addressed by Praveen Mellacheruvu in that we are uncertain of the data beyond the current analysis and have no basis to buy any tool other than the one with the greatest gradient.

Summary / Future Suggestions

The program is effective if used properly. Some ideas to improve the program include building on the robustness of the code and researching the effects of other search algorithms. Overall, the program provides a very good base for professionals designing a factory as well as a learning tool for students. The interface provides a link to the Factory Administrator that allows a user to perform sensitivity analyses on several different variables. These two programs are able to guide users through design and optimization of use of a semi-conductor factory. For students, the programs allow classroom simulation and numerical data to compare and demonstrate classroom theory.

REU

This paper is a summary of the work done by Ryan Thomas from June to August 2000 at the University of Maryland. Throughout this time I was enrolled in the Research Experience for Undergraduates Program. This is a program developed and guided by the Institute for Systems Research (ISR) and funded by the National Science Foundation (NSF).

[2] Note that processing times are those of a 500 MHz Intel Processor.

The program allows students to gain experience in systems engineering at the University of Maryland while earning a stipend. Students are advised by a faculty member in ISR and assist them with their research throughout the summer.

My work was done under advisement from Dr. Jeffrey Herrmann, with assistance from Dr. Gary Rubloff and Dr. Laurent Henn-Lecordier. The project was Integrating Product Dynamics and Process Models (IPDPM) which studied a very broad scope of aspects affecting semi-conductor manufacturing and optimization. The work I have done was applying the research of a former graduate student (see References) and developing it into a practical interface that was able to handle more realistic situations.

I am currently a senior Mechanical Engineer at the Carnegie Mellon University in Pittsburgh, Pennsylvania. I will graduate in May 2001 and will pursue a career in the Washington DC metro area. The REU program has provided a great deal of insight and experience into my field as well as a practical understanding of simulation. Additionally, I was part of weekly meetings with the entire IPDPM staff (consisting of full-time faculty, research faculty, and graduate students). These meetings created an excellent "team" atmosphere and both benefited my research and insured the well-being of the IPDPM project.

References

The Tool Optimization Program was based on the work of Praveen Mellacheruvu (University of Maryland, 1998-2000). Two of his papers were directly referenced in this paper:

Mellacheruvu, Praveen V; Fu, Michael C; and Herrmann, Jeffrey W., *Comparing Gradient Estimation Methods to Stochastic Manufacturing Systems*, 2000.

Mellacheruvu, Praveen, *Sensitivity Analysis and Discrete Stochastic Optimization for Semiconductor Manufacturing Systems*, pp. 67-85, 2000.