The Scheduling Problem in Manufacturing

Hanny Fauzia (13509042)[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13509042@std.stei.itb.ac.id

***Abstract*—Nowadays, manufacturing setting is very complex, involving multiple lines of product, each requiring many different machines and multiple stages for completing many tasks. These facts must be considered by the decision maker for the manufacturing plan to successfully manage resources in order to produce products as efficient as possible. This problem is often called Scheduling Problem, which involves many problems concerning minimizing or maximizing job completion aspect (completion time, number of processors needed to complete jobs, etc.). This paper will focus mainly on discussing how to minmize job completion time, and minimize number of processors required for completng the job. We will see how we can use graphs to solve those two problems.**

***Index Terms*—Graphs Application in Scheduling Problem, Job Completion Time Minimization, Manufacturing Setting, Number of Processor required for Job Completion Minimization.**

I. INTRODUCTION

These days, industry world has grown very rapidly. This fact cause many manufacturing plants are build worldwide. The advancement of technology also contibute to increase the efficiency of jobs completion in manufacturing plants. Although that advancement no mistakingly make job completion in manufacturing industry easier, it also makes other problems arise. The main problem is the growing complexity of manufacturing setting that will make it really difficult to successfully manage resources in order to produce products as efficient as possible. There will be countless job assignment variation because there are so many aspects to be considered, such as number of tasks, number of processors, kind of tasks each processors can complete, and many more.

That kind of problem is widely known as scheduling problem. Scheduling problems, just like its name, consists of deciding how tasks should be scheduled. The tasks are performed by a number of identical *processors*, which can be machines, humans, etc. The goals in solving this scheduling problem are rich in variety, such as achieving as short time as possible to complete a job, minimizing the total "lateness" of jobs, minimizing the maximum waiting time, or optimizing performance according to some other criterion. In this paper, author will only focus on these two problems: minimizing job completion time, and minimizing number of processors needed to complete the job.

Those two problems seem easy to solve, but they are really difficult actually, because there are so many aspects that should be taken into account. In this paper, we shall disuss how to solve that problems using critical path method (for minimizing job completion time), and First-Fit algorithm (for minimizing number of processors required for job completion), as we will see in chapter II.

A. Job Completion Time Minimization (Scheduling Problem)

Manufacturing is basically an activity to create goods or products. In order to create a product (a job), several tasks need to be completed by using various processors (machines, humans, etc.). Each task has an accosiative length of time required for its completion. The faster a product is created, the more products quantity that can be achieved, and the more profitable it will be for the manufacturer. Therefore, it is often of interest to find the optimal task assignments for each processors available in order to finish the job as soon as possible. The naive approach to solve the problem is to list all of possible task arrangements, and choose one arrangement that has the shortest job completion time, which is known as exhaustive search method. But, as we will see in the next paragraphs, this method is not preferable and rarely used in the real industry world.

Now, we will see an example that use exhaustive search for the optimal task assignments. In any manufacturing plant, where there are many tasks and various processors capable of completing certain tasks, there will be also many variety of task assignments in order to complete a job. For example, there are 8 tasks that should be done for completing a job. Each task has various completion time, for example, task A required 10 minutes to complete, whereas task B required only 4 minutes to complete, and so on. Those tasks may have dependence toward other tasks, for example, task A

should be done first before doing task B, and task C should be done before doing task D, etc. Then, if in a manufacturing plant there consists only 3 processors that able to complete those tasks, how many work assignments are there for such criterias (8 tasks (each has various dependencies toward other tasks), and 3 processors)? The answer is: It is so many that it is very inefficient to list all possible assignments. We can assign all tasks completion to the processor 1 while other processors are idle, or we can use all processors together with various work assignments for each processor, and many more.

That's why, it is very inefficient to use exhaustive search method as described in previous paragraphs to find the best work assignment in order to complete a job as fast as possible. The more preferable method to solve this problem is critical path method (CPM) that will be discussed further in chapter II.

B. Number of Processor required for Job Completion Minimization (Bin Packing Problem)

Other than minimizing the time required for completing a job, it is also of interest to minimize the number of processors required for completing a job. Just imagine it, if by using 8 processors, a job can be completed in 1 day, but the cost for using each processors is 40$/day and the profit for that day is only 200$, is it worth it? But of course, by minimizing the numbers of processors available to complete a job, the job completion time may also takes longer than before. Each decision have its negative and positive effects, so the decision on deciding what aspects should be minimized depends on what problems that a decision maker in manufacturing plant face.

This kind of problem, rather than called one of scheduling problems, are widely known as Bin Packing Problems. Bin packing problems mainly discuss how objects of different volumes must be packed into a finite number of bins of capacty V in a way that minimizes the number of bins used. In manufacturing schduling problems, those *bins* are processors, *volume* is the deadline for completing the job, and objects with different volumes are tasks with various completion times. In other words, we want to find task assignment for each processor such that it will minimize number of processors needed to complete a job, given a deadline time to finish all tasks.

This problem, again, can be solved by using exhaustive search method as described in point A. There is another method that can solve that problem, which is called First-Fit Algorithm. Although it doesn't always give the best optimal result, the result is fairly well, and much more efficient than exhaustive search method. We will see the use of this algorithm in order to solve bin packing problems in the next chapter.

II. SOLVING THE PROBLEMS

In this chapter, author will give one example for each problem in previous chapters. Each example will have a solution by using one method. First example is about minimizing job completion time by using CPM (Critical Path Method), and next example is about solving bin packing problem by using First-Fit Algorithm.

A. Critical Path Method

The critical path method (CPM) is a mathematically based algorithm for scheduling a set of project activities. This method, although seems easy and trivial as we will see later, is very useful and flexible because it can be used in all forms of projects, such as construction, software development, plant maintanace, product manufacturing, and many more.

The essential requirement before we can use CPM is to construct a model of the project that includes the following:

1. A list of all activities required to complete the project
2. The time (duration) that each activity will take to completion
3. The dependencies between each activities

We called an activity depend on other activities if that activity cannot start until the other activities that they *depend on* has been completed. For example, task U which consists of transporting all products to distributor cannot start before task T, which is testing for the quality of the products. In this case, task T is a precedent or a predecessor of task U. This kind of dependence is very common in real manufacturing world, and should be taken into account.

Actually, we can easily construct this model by using direected graph structure to easily conduct this critical path method. Duration for each activity take to completion is stored in a vertex which named after their task name. The dependencies between two tasks are represented by directed edge that connect two nodes (activities/tasks).

To fully understand how to construct such graph, which is often called order requirements digraph, let's see an example of such graphs (Figure 1) derived on information described in Table 1.

Table I. Tasks Information for Figure 1

Task	Completion Time (in minutes)	Precedence(s)
A	1	none
B	1	none
C	1	none
D	1	none
E	10	none
F	10	none

G	10	none
H	3	A
I	3	A
J	3	A
K	3	A
L	10	H,I,J,K

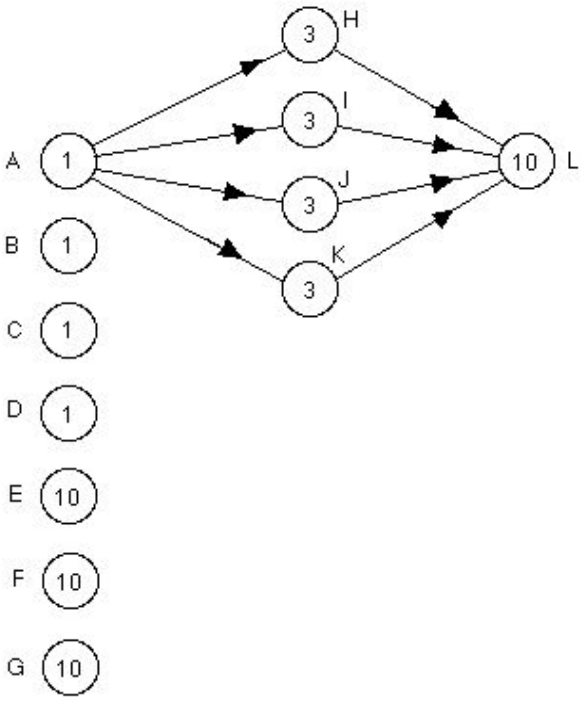


Figure 1. Order requirements digraph derived from Table I

As we can see, it is easier to read task information from Figure 1 rather than from Table I. The graph can be simplified (optionally), by eliminating an edge from node J to K if there is a different directed path from J to K, so that the graph will be less complicated and easier to understand. We won't lose any precedence information because the dependencies between J and K can still be implied by the alternative directed path from J to K.

After constructing such graph from tasks information, we shall search the critical path of the graph. A critical path in an order requirements digraph is a directed path such that the sum of the completion times for the vertices in the path is the largest possible. For Figure 1, as we can see, the critical path is A-H-I-J-K-L, not A-H-L, nor A-I-L, A-J-L, and A-K-L, because L's precedendes are H,I,J,K. That's why, L cannot start before H,I,J,K are done. Other than A-H-I-J-K-L, A-I-J-K-H-L, A-K-H-I-J-L, are also critical paths, because sum of times for their vertices are the same (1+3+3+3+3+10 = 23 minutes). The total sum of times for their vertices are the shortest time possible for completing the job.

Why is it so? Note that the earliest starting time for any task T is the largest sum of times on a order requirements digraph to that task. Every precedent of T will be on a directed path to T. Seeking the longest such directed path means that all precedents of T will be finished before we start T. Thus, critical path gives us a lower bound for the completion time for the whole project. But, this method only applies when the number of processors available is unlimited—which means that it is not necessary to wait until a processor is available. After understanding this method, we shall see an example how to apply this CPM into a case.

For simplicity in our discussion, lets make some assumptions. We will assume that any processors can complete any tasks, no processor can be idle if there is some task it can be doing, and once a processor has begun a task, it alone must continue to process that task until it is finished. If those assumptions are not considered, then this discussion will be far more complex and not appropriate for beginners.

Example I : Suppose in a manufacturing plant, in order to complete a job, 5 tasks must be completed, with tasks information as described in Table II below. Find the shortest time possible for this job to be completed by using CPM!

Table II. Tasks Information for Example I

Task	Completion Time (in minutes)	Precedence(s)
A	30	none
B	40	A
C	10	A
D	30	A,C
E	50	A,B,C,D

From information derived from Table I, we make order requirements digraph in Figure 2. This figure is the simplified version (one edge that connects 2 nodes can be erased if there is another alternative path that still connect those edges).

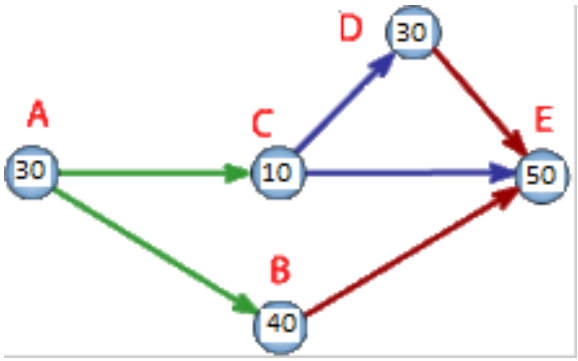


Figure 2. Order requirements digraph derived from Table II

After creating such graphs, now we have to find its critical path. The possible path available from figure 2 is A-B-E, A-C-E, and A-C-D-E. Then, we found that critical paths from graph in figure 2 are A-C-D-E and A-B-E which each completion time is 120 minutes. Then, from that two paths, we can make the optimal schedule

in the form of Table as we can see in Table III. This optimal schedule isn't unique because there is other possible arrangements, such as in Table IV. But one criteria that isn't unique is that each optimal schedule always required 120 minutes for completing all tasks. One thing to be considered again, this method only applies if there is unlimited processors available for completing the job, which means it won't be necessary to wait for other processors to be available in order to complete a task. For example, if available processor for completing the job is only one, then this optimal schedule is not valid, because for example in Table III, in order to start task B, it has to wait until all tasks in processor 1 are done, and so does for task E. This will make the required job completion time much longer (160 minutes).

Table III. One possible optimal schedule for example I

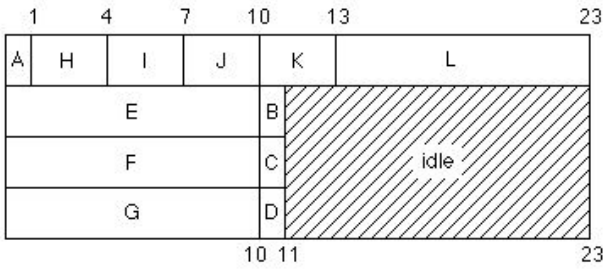
Time (minutes)	0	30	40	70	120
Processor 1	A	C	D	Idle	
Processor 2	Idle	B		E	Idle

Table IV. Another possible optimal schedule for example I

Time (minutes)	0	30	40	70	120
Processor 1	A	B		E	Idle
Processor 2	Idle	C	D	Idle	

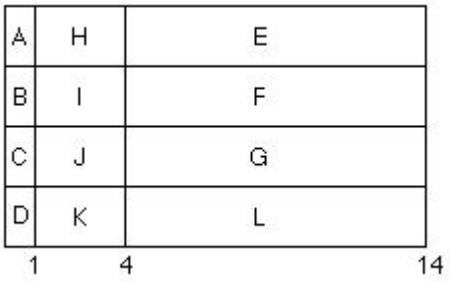
Simple, but very useful are some words that can describe this method. But, unfortunately, this method doesn't always give the best result. For example, we've found that one of critical paths from figure I is A-H-I-J-K-L which gives 23 minutes for job completion. For realizing this schedule, we need 4 processors, and the schedule is described in Table V. The numbers above and below the table represents time in minutes.

Table V. Critical Path Schedule from Figure 1



But unfortunately, in this case, that table is not the optimal schedule for figure 1. The optimal schedule is the one in Table VI. Again, the numbers above and below the table represents time in minutes.

Table VI. Optimal Schedule from Figure 1



This one example shows the flaw of finding optimal schedule with critical path method. Even so, this method is fairly useful and easy to use, so, this method is often used in the real manufacturing world to find the optimal schedule for completing a job.

B.First-Fit Algorithm

One of heuristic approach to solve bin packing problem is by using First-Fit Algorithm. This Algorithm, though doesn't always give the best result, is very easy to conduct and fast to be done, with $\Theta(n \log n)$, where n is the number of elements to be packed. The basic idea of this algorithm in solving bin packing problem concerning manufacturing scheduling problem, is to assign each tasks available into the first processors that still can complete the task before the deadline. Thus, there is limitation in order to use this algorithm in scheduling problem, which is, the tasks must be independent. Meaning, there is no dependencies between any task, and the order requirements digraph shouldn't have any edges. So, the graph should only consists of nodes. The algorithm in pseudo-code notation is listed in figure 3.

Fig. 3.First-fit Algorithm

First-fit Algorithm.
procedure *FF*(*W*: an arbitrary list *w*1, . ,*wn*)
*{b*1, . . . is the list of bins; object *Oi* has weight *wi*; *L*(*bj*) = total weight placed in *bj}*
k := 1
{k is number of bins opened*}*
for *i* := 1 **to** *n*
begin
*{*find first available bin*}*
j := 1
while *Oi* not packed **and** $j \leq k$
begin
if *L*(*bj*) + *wi* ≤ *d* **then** pack *Oi* in *bj*
j := *j* + 1
end
if j=k+1 **then** open *bj* and pack *Oi* in *bk*+1
end

Before we can apply this algorithm to solve previous problem, we should make a list that consists of each completion time for all tasks. For example, for tasks

information on Table VII, the list for first-fit algorithm is 8 7 6 5 2 3.

Table VII. Tasks Information

Task	Completion Time (in minutes)	Precedence(s)
A	8	none
B	7	none
C	6	none
D	5	none
E	2	none
F	3	none

The reason why this algorithm doesn't always give the best result is that the algorithm depends heavily on the order of the weights (completion time for each task) in list such as in Table VII, as we can see in Example II.

Example II : Suppose in a manufacturing plant, in order to complete a job, 25 tasks must be completed, with tasks information as described in Table VII below. There is no dependency between each task. Find the minimum number of processor required for this job to be completed by using First-Fit Algorithm, given the job must be completed in 100 minutes!

Table VIII. Tasks Information for Example II

Task	Completion Time (in minutes)	Precedence(s)
A	7	none
B	7	none
C	7	none
D	7	none
E	7	none
F	12	none
G	12	none
H	12	none
I	12	none
J	12	none
K	15	none
L	15	none
M	15	none
N	36	none
O	36	none
P	36	none
Q	36	none
R	36	none
S	52	none
T	52	none
U	52	none
V	52	none
W	52	none
X	52	none
Y	52	none

First, we have to make a completion time list for all tasks. One arbitrary choice to make this list is:
7 7 7 7 7 12 12 12 12 12 15 15 15 36 36 36 36 36 52 52 52 52 52 52 52

The first-fit algorithm yields the following schedule, stored in Table IX (produced on next page), which needs 11 processors. This is not the optimal result that we want. The actual minimum number of processors required is 7, just like the schedule in Table X (also produced on next page) shows. This is the optimal result, because the sum for all tasks completion time is 684 minutes, and, in order to complete all tasks in only 100 minutes, means that at least 684/100 = 6.84 or 7 processors are required.

Such result is possible to attain with a bit modification in before using First-Fit-Algorithm. This modification leads to the new algorithm, which is called First-Fit-Decreasing Algorithm. The Algorithm is exactly like the First-Fit Algorithm. The difference is, before we apply the algortihm to the list, we should sort it in decreasing weights order. Then, let's apply this new Algorithm to the example II.

The list for example II, if it's sorted in decreasing weights order is:
52 52 52 52 52 52 52 36 36 36 36 36 15 15 15 12 12 12 12 12 7 7 7 7 7

If we apply the same algorithm (First-Fit Algorithm) to the list above, then the schedule it results will be the one in Table X, the optimal result which only need 7 processors for job completion. But actually, this algorithm also doesn't always give the best result. The method that will guarantee us to give the best result is still the exhaustive search method. But the results derived from First-Fit Algorithm and First-Fit-Decreasing Algorithm are satisfying concerning the easy steps that should be completed.

As we can see in example II, First-Fit Decreasing algorithm gives more satisfying results than First-Fit Algorithm. But actually, in real manufacturing process, sometimes First-Fit Algorithm is more preferable than the First-Fit Decreasing Algorithm. This is true mainly in case of loading goods to the trucks. It is often impossible to weight all of goods first before loading it into the truck. Thus, sorting the goods in decreasing weights order is also not possible in that case. That's why, the more efficient and fast algorithm in is First-Fit Algorithm, in case we want to minimize number of processors required faster. If we want to get the minimum number of processors required, where time is not of concern, then the First-Fit Decreasing algorithm might be more preferable.

Table IX. Schedule Derived from First-Fit Algorithm for Example II

Time(min)	0	7	14	21	28	35	47	59	71	83	95	100
Proc. 1	A	B	C	D	E	F	G	H	I	J	idle	
Proc. 2	K		L		M		N			idle		
Proc. 3	O					P				idle		
Proc. 4	Q					R				idle		
Proc. 5	S							idle				
Proc. 6	T							idle				
Proc. 7	U							idle				
Proc. 8	V							idle				
Proc. 9	W							idle				
Proc. 10	X							idle				
Proc. 11	Y							idle				

Table X. Schedule Derived from First-Fit Decreasing Algorithm for Example II

Time(min)	0	52	59	63	70	77	88	100
Proc. 1	S	N					F	
Proc. 2	T	O					G	
Proc. 3	U	P					H	
Proc. 4	V	Q					I	
Proc. 5	W	R					J	
Proc. 6	X	K		L			M	
Proc. 7	Y	A	B	C	D	E		

III. CONCLUSION

In order to solve the main problem in a manufacturing process—scheduling problem, concerning on minimizing job completion time or minimizing number of processors required for job completion—, we can use Critical Path Method (CPM) and First-Fit Algorithm (FF). CPM is used when we are interested in finding the optimal schedule with the shortest possible job time completion, while FF is used when we want to minimize number of processors required for completing a job. Those methods, although doesn't always give the best result, is far more efficient than the exhaustive search method.

IV. ACKNOWLEDGMENT

In the Name of Allah, the Most Gracious, the Most Merciful.

The Author would like to give sincere thanks to all the people who have contributed to the creating of this paper in this section. First praise is to Allah, the Almighty, on whom ultimately the author depended on for showing inner peace and for all my blessings. A special word of gratitude to my mother, father, and sister. Without their support, this paper could not have been achieved. A word of thanks must also go to Mr. Rinaldi Munir for his guidance during the course work for this degree Discrete Mathematic Education. The last, but not the least, thank you to all of my friends who helped me work my way through the academic and technical dimensions of this paper.

REFERENCES

[1] M. Robert, "Applications of Discrete Mathematics,", updated ed., M.G. John and H.R. Kenneth, Ed. New York: McGraw-Hill, 2007 , pp.187-199.

[2] Bin Packing Problem. (2010, December 14). In Wikipedia, The Free Encyclopedia. Retrieved 04:27, December 14, 2010, from http://en.wikipedia.org/wiki/Bin_packing_problem

[3] Critical Path Method. (2010, December 13). In Wikipedia, The Free Encyclopedia. Retrieved 10:30, December 13, 2010, from http://en.wikipedia.org/wiki/Critical_path_method

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 14 Desember 2010

Hanny Fauzia (13509042)