PROGRAM EVALUATION AND REVIEW TECHNIQUE AND CRITICAL PATH METHOD

Nguyen Thi Thu Hoa

Hanoi Metropolitan University

Abstract: *In this paper, we start with a table of project tasks then use Python to create PERT chart and determine critical paths, critical activities, non-critical activities and slacks.*

Keywords*: PERT network diagram, PERT chart, CPM, critical task, non-critical task, project completion time, slack, AON, directed acyclic graph (DAG), arrow diagram (AND), networkx, matplotlib, Gantt chart, topological sort.*

Received 27 April 2023

Revised and accepted for publication 24 July 2023

Contact author: Nguyen Thi Thu Hoa; Email: hoantt@daihocthudo.edu.vn

1. INTRODUCTION

1.1. Basic terms

PERT (program evaluation and review technique) charts were first coined by U.S. Navy's Special Projects Office in 1957 to guide Polaris nuclear submarine. Then, for a long time, PERT has been used a graphical tool that breaks down project tasks for analysis. Using PERT, one can determine the interdependence of tasks, the time needed to complete each individual task and the minimum time to finish the project.

PERT charts are often associated with *CPM (critical path method)*. Being developed and put into practice in 1950s by Morgan R. Walker, CPM or CPA (critical path analysis) has now become one of the most significant algorithms for scheduling interdependent project tasks and identifying the longest path of critical activities (ones that cannot be delayed) to the endpoint.

The key concepts in PERT and CPM are defined as follows:

The *vertices* of a *directed acyclic graph* (DAG) represent *milestones* of a project.

PERT activity is the actual performance of a task which consumes time and requires resources (such as labor, materials, space, machinery).

Here, we only deal with *arrow diagram* or *activity on arrow diagram, (activity) network diagram (AND).* It is a diagram showing the sequence of interrelated tasks in a process or project and potential scheduling as well as the optimal schedule.

Each edge is labeled with the amount of performance time.

If an arrow (directed edge (i, j)) is drawn from ith event to jth event, then another arrow (directed edge (j, k)) shows that task (i, j) must be completed before working on task (j, k).

This type of activity relationship is *Start-to-Start (SS)*: the successor activity cannot start unless the predecessor one starts.

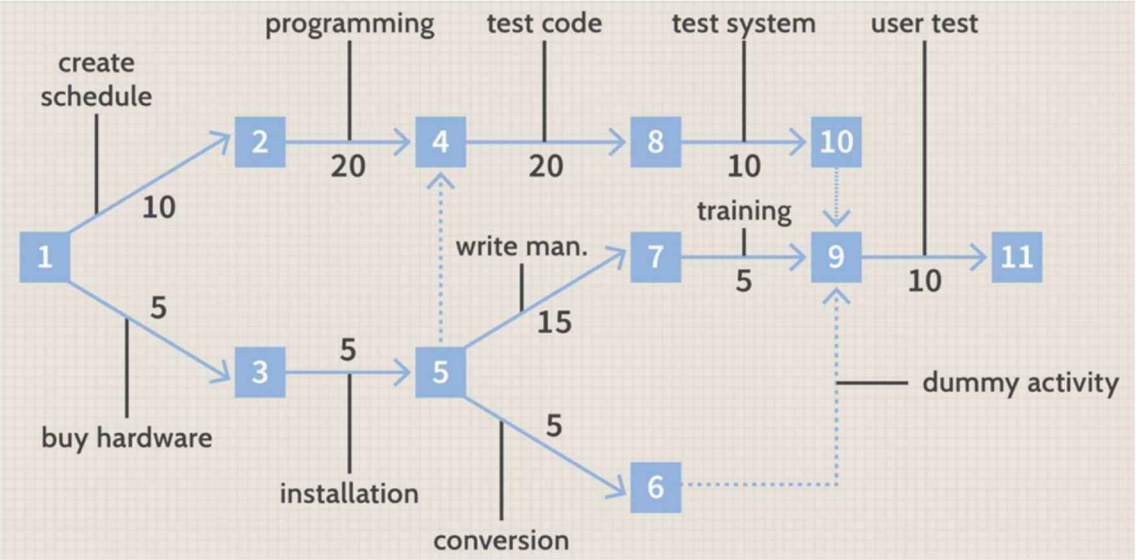


Figure 1: PERT chart (© Sabrina Jiang @Investopedia 2020)

Earliest start time (ES) is the earliest time at which the task may begin and the earliest time that all predecessor activities can be finished. *The latest start time (LS)* is the latest time the activity can begin. *Earliest finish time (EF)* is the earliest time for completion of a project activity. *Latest finish time (LF)* is the latest time the activity can end without delaying the entire project.

Total slack or float (TF) is the maximum amount of time a task can be delayed before the project finishing date is delayed. *Free slack (FS)* of an activity is the time this activity can be delayed without impact on the following activity.

Critical path is the longest possible continuous path starting from the initial event to the terminal event. *Critical task* is a task with total float is 0. All tasks on the critical path are critical tasks while some zero free float activities might not be on the critical path. *Non-critical task* is a task with non-zero total float.

1.2. Network rules

1.2.1. Drawing AND

To draw an AND, we carry out two steps.

Step 1: List all necessary tasks and the correct sequence of the tasks in the process or project and estimate the time that each task should require. This result would be saved on tasks.csv.

Step 2: Create a network diagram satisfying the following network rules:

- Time flow from left to right

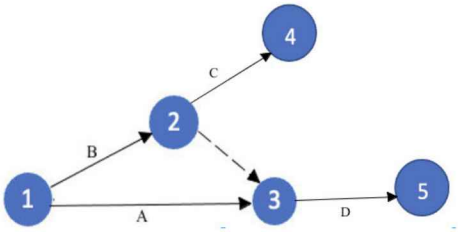


Figure 2: Dummy task separating simultaneous tasks

- Vertices represents *events*.

An event marks the beginning or end of a task.

- A task or activity is represented by a directed edge, connecting two events that mark the beginning and completion of the activity.
- *Dummy tasks* (not real tasks) and *extra events* are used to separate tasks to avoid circular dependency and to show logical sequence.

A and B is represented by the dashed line from event 2 to event 3. Here, the arrow diagram shows that task C starts after task B is completed and task D starts after both tasks A and B are finished.

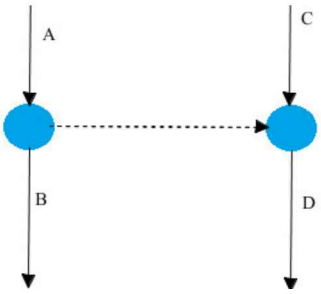


Figure 3. Dummy task between the end of task A and the beginning of task D helps keeping sequence correct

Here, the arrow diagram shows that task B starts after task A is finished and need not wait for task C, task D starts after task C is completed and task D cannot start until both tasks A and task C are finished.

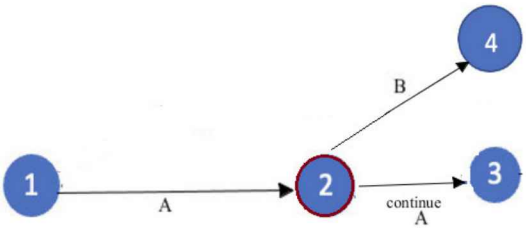


Figure 4. Here, the arrow diagram shows that task B can be started before part of task A is done. An extra event 2 is added where task B can start and task A is split into two subtasks

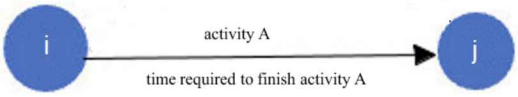


Figure 5: An example of topological ordering. Events i and j mark the beginning and the end of task A, respectively with i < j

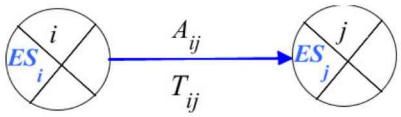
The vertices are numbered according to *topological sort* i.e., for every directed edge ij from vertex i to vertex j, i comes before j in the ordering (i < j). ***1.2.2. Forward and Backward Pass Computations*** Earliest start (ES)	Earliest finish (EF)
Latest start (LS)	Latest finish (LF)

Figure 6. Arrow diagram time box

Let D be the duration of that activity, then we have the following formulae

$$LS + D = LF\ ; ES + D = EF$$

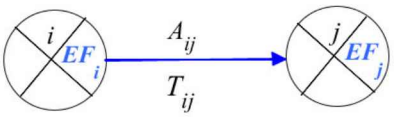
a) Forward pass computations



$$ES_1 = 0; \quad ES_j = \max_{(i,j) \in E} \{ES_i + T_{ij}\}$$

Figure 7. Earliest start times are computed from left to right, beginning from the initial node

b) Backward pass computations



$$EF_n = ES_n; \quad EF_i = \min_{(i,j) \in E} \{EF_j - T_{ij}\}$$

Figure 8. Earliest finish times are calculated from right to left, starting from the terminal node

1.2.3. Critical Path Method (CPM)

Here are the steps to calculate the CP in project management:

Step 1: *Collect activities using work breakdown structure*.

Step 2: *Identify dependencies.*

Step 3: *Create a network diagram*

Step 4: *Estimate timeline* by figuring out the duration of each activity

Step 5: Use *forward and backward pass calculations* to calculate $ES, EF, LS,$ LS.

Step 6: Identify the *float of each activity*

$(Total\ slack = latest\ start - earliest\ start)\ TF_{ij} = LS_j - ES_i$

$(Free\ float = ES\ of\ next\ activity - EF\ of\ current\ activity)\ \ FS_{ij} = ES_j - EF_i$

Step 7: *Identify the critical path*.

The activities with 0 float make up the critical path.

Step 10: *Revise during execution*: Continue to update the critical path network diagram as you go through the execution phase.

2. CONTENT

2.1. Creating AND using csv file data

The PERT chart can be drawn in 3 steps as follows:

Step 1: Creating pandas (pd) DataFrame of project tasks.

The sequence of interrelated project tasks is given in a csv file.

Then, we use Pandas Python library to deal with data analysis.

The csv project tasks file with semicolon delimiter is read as Pandas DataFrame df and shape attribute gives us its dimension.

Step 2: Creating the DAG using networkx (nx)

We initialize our DAG as empty DAG, then add nodes from a sequence. For each row in our Pandas DataFrame, node1, node2 and the weight are determined from the corresponding DataFrame columns. The .add_edge and .directed_edge_labels are used to label directed edges.

In our desired DAG, arrows must show forward direction either left to right or upward. So, we specify the node positions in order to adjust the graph layout. These node positions can be obtained by trials and errors.

Then, .draw_networkx and .draw_networkx_edge_labels are used to draw the labelling DAG.

Step 3: Displaying the DAG using matplotlib.pyplot (plt)

2.2. Calculating the critical path

The criticalpath module comes in handy when we only want to calculate the critical path and the minimum time to complete the project. So, we first import criticalpath. Node, add node names, edges; then use update_all, get_critical_path method and duration attribute.

2.3. Numerical results

2.3.1. Creating a PERT chart in Python

Eg2_GTVT.csv

Activities	Duration	Dependencies
a1	3	
a2	7	
a3	9	
a4	5	a2
a5	8	a1
a6	10	a2 a3 a5
a7	9	a1
a8	8	a4
a9	12	a7
a10	6	a4
a11	15	a6 a8
a12	11	a6 a8 a10
a13	8	a9
a14	19	a7

Eg2_GTVT_AND.csv

n1	n2	Weight
1	2	A1(3)
1	3	A2(7)
1	4	A3(9)
2	6	A7(9)
2	4	A5(8)
3	4	Dummy
3	5	A4(5)

4	7	A6(10)
5	7	A8(8)
5	9	A10(6)
6	8	A9(12)
6	10	A14(19)
7	10	A11(15)
7	9	Dummy
8	10	A13(8)
9	10	A12(11)

Eg2_GTVT.py
(Python 3.9, pandas 1.2.3, network 2.5, matplotlib 3.3.4)

```
import networkx as nx
import matplotlib.pyplot as plt
import pandas as pd

df = pd.read_csv('Eg2_GTVT_AND.csv', sep=";")
n_rows = df.shape[0]
g = nx.DiGraph()
g.add_nodes_from([_ for _ in range(1, 11)])
directed_edge_labels = dict()

for _ in range(0, n_rows):
   node1, node2, weight1 = df['n1'][_], df['n2'][_], df['weight'][_]
   g.add_edge(node1, node2, label=str(weight1))
   directed_edge_labels[(node1, node2)] = weight1

pos = {1: (0, 20),
       2: (10, 35), 3: (10, 0),
       4: (20, 20), 5: (25, 0), 6: (25, 40),
       7: (35, 20), 8: (43, 40),
       9: (42, 0),
       10: (55, 20)}
nx.draw_networkx(g,
                 pos=pos,
                  node_color='y')
nx.draw_networkx_edge_labels(g,
                             pos=pos,
                             edge_labels=directed_edge_labels)
plt.savefig('Eg2_GTVT_AND.png')
plt.show()
```

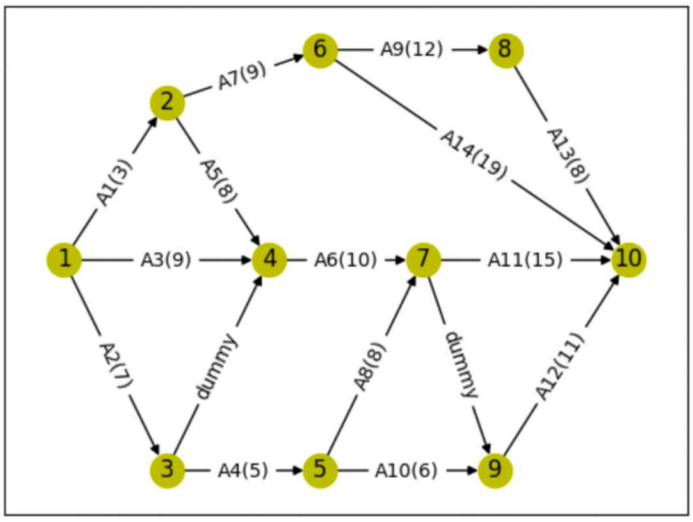


Figure 9. PERT chart for the numerical example 1

2.3.2. Numerical example 2: Determining the critical path

Eg2_CPM2.py
(Python 3.9, pandas 1.2.3, criticalpath 0.1.5)

```
from criticalpath import Node
import pandas as pd

df = pd.read_csv('Eg2_GTVT_CPM1.csv', sep=';')
d = df['Duration']

g = Node('project')

a1, a2, a3, a4, a5, a6, a7, a8, a9, a10, a11, a12, a13, a14 \
   = g.add(Node('A' + str(1), duration=d[0])), \
   g.add(Node('A2', duration=d[1])), \
   g.add(Node('A3', duration=d[2])), \
   g.add(Node('A4', duration=d[3])), \
   g.add(Node('A5', duration=d[4])), \
   g.add(Node('A6', duration=d[5])), \
   g.add(Node('A7', duration=d[6])), \
   g.add(Node('A8', duration=d[7])), \
   g.add(Node('A9', duration=d[8])), \
   g.add(Node('A10', duration=d[9])), \
   g.add(Node('A11', duration=d[10])), \
   g.add(Node('A12', duration=d[11])), \
   g.add(Node('A13', duration=d[12])), \
   g.add(Node('A14', duration=d[13]))
```

```
edges = [(a1, a7), (a1, a5),
         (a2, a4), (a2, a6),
         (a3, a6),
         (a4, a8), (a4, a10),
         (a5, a6),
         (a6, a11), (a6, a12),
         (a7, a9), (a7, a14),
         (a8, a11), (a8, a12),
         (a9, a13),
         (a10, a12)]
for e in edges:
   g.link(*e)
g.update_all()
print(g.get_critical_path())
print(g.duration)
```

Results: The critical path is [A1, A5, A6, A11]
and the minimum time to complete the project is 36.

3. CONCLUSION

In this paper, we have created PERT chart and find the critical path using Python pandas, matplotlib, pyplot, network, criticalpath modules.

REFERENCES

1. Joseph J.Moder, Cecil R.Phillips (1970). *Project management with CPM and PERT*. Van Nostrand Reinhold Co., ISBN: 0-442-15666-9.
2. Nancy R. Tague (1955). *The Quality Toolbox*. William A.Tony, ISBN: 978-0-87389-639-9.
3. U.S.Dept of the Navy (1958). *Program Evaluation Research Task, Summary Report, Phase 1.* Washington, D.C., Government Printing Office,
4. Harold Kerzner (2003). *Project Management: A systems approach to planning, scheduling, and controlling* (8th ed.). Wiley, ISBN 0-471-22577-0.

KỸ THUẬT ƯỚC LƯỢNG VÀ ĐÁNH GIÁ DỰ ÁN VÀ PHƯƠNG PHÁP ĐƯỜNG GĂNG

Tóm tắt: *Bài viết này đề cập đến cách tạo sơ đồ mạng và tính toán các công việc găng, đường găng, các công tác có thời gian dự trữ, thời gian tối thiểu để hoàn thành dự án.*

Từ khoá: *Sơ đồ mạng PERT, đường găng, công việc găng, thời gian hoàn thành dự án, thời gian dự trữ, đồ thị có hướng không chu trình (DAG), networkx, matplotlib, sắp xếp tôpô trên đồ thị.*