Algorithm of Critical Path Based on Dynamic Programming

[1]Qianchao PANG , [2]Xiaoli LIU
*[*1] Department of Information Engineering, Zhejiang Textile & Fashion College, Ningbo, 315211, China, mdj777@163.com*
[2] College of Computer and Information, Zhejiang Wanli University, Ningbo, 315100, China, liuxl62@163.com

Abstract

Directed acyclic graph (abbreviated as DAG) is an effective way to describe a project, such as planning process, construction work, production process and program flow etc., while critical path is used to estimate the shortest time to complete the whole project. AOE network refers to a directed graph in which vertex stands for events, directed edge for activities, weights on the edge of the cost are the said activities. In this paper the algorithm of solving the critical paths based on the philosophy of dynamic programming is presented, and the analysis and research on the algorithm is also furnished.

Keywords*: Directed Acyclic Graph (DAG); Activity on Edge (AOE) Network; Dynamic Programming; Critical Path*

1. Introduction

Construction industry is the leading industry that pushes forward the developing progress in China. However due to long construction term, large investment, high technical requirements and complex internal and external communication, people may encounter a lot of uncertainties in construction projects. The organization of construction will directly impact on the schedule of the project. So how to handle the issue of critical path becomes the important in the most importance in setting up the construction organization[1].

A directed graph without cycle is called Directed Acyclic Graph[2], abbreviated as DAG. DAG is an effective way to describe a project[3], such as planning process, construction engineering, production process and program flow. Generally, each project can be divided into several subprojects called Activities. Between subprojects, there exist certain constraints. Some subprojects should be started after others had been solved. Therefore a DAG can be used to represent a Project, among which the directed edges denote the constraints.

This kind of directed graph must be acyclic. If there appear cycles (directed cycle) then as the procedure goes on, any subprojects (activities) on the cycle must take the commencement of themselves as prerequisites. Obviously that is a paradox. If anyone presents such a project flow chart, it has no way to go.

To the whole project, people concern about two aspects, the first is whether the project could go on smoothly, and the second is to estimate the shortest time necessary to finish the whole project. This will involve two operations: topological ordering and critical path.

2. Definition of Critical Path

In the directed graph with weight, vertices denote events, the directed edges denote activities, the weight on edges denotes cost (such as duration of activities). So the directed graph with weight is called Activity on Edge Network, abbreviated as AOE network.

For instance, Figure 1 shows an AOE network, in which there are nine (9) events $v_1, v_2, \ldots, v_9,$ eleven (11) activities $a_1, a_2, \ldots, a_{11}$. Each event represents that the activity before it is finished, and the one after is to start. As we see, v_1 represents the commencement of the whole project, and v_9 the completeness of the whole project. V_5 shows that activity a_4 and a_5 are finished, and activity a_7 and a_8 can start. The weight associated with each activity represents the time necessary for finishing this activity. See that a_1 needs six (6) days to finish.

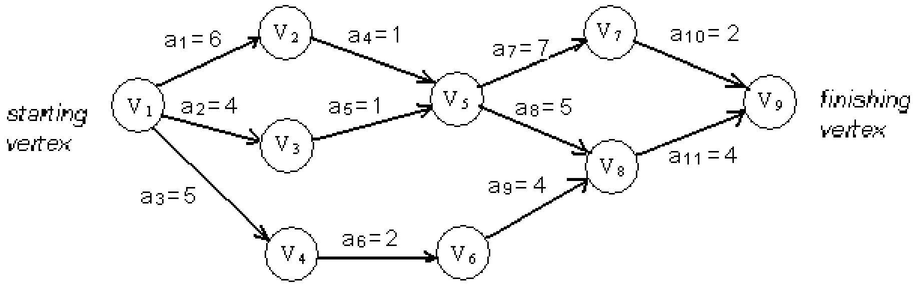

Figure 1. An Example of AOE Network

AOE network has following properties[4]:

(1) Only after one event represented by a vertex happened, may activities represented by each directed edges from this vertex start.

(2) Only when activities represented by each directed edges going into one vertex, can the event represented by this vertex happen subsequently.

(3) An AOE network representing a practical project should have no cycle, and there exists a starting vertex with a unique indegree 0, and a finishing vertex with an outdegree 0.

If we use AOE network to represent a project, the consideration about the priority relationship between each subproject is not enough. What we concerns more is how much the shortest time to finish the whole project is, which delay of the activity impacts on the schedule of the whole project, and on which activity the acceleration may lead to the high efficiency of the project. Therefore, the topics to be discussed on AOE network are:

(1) How long at the minimum does it need to finish the whole project?

(2) Which activities are the critical ones to impact on the schedule?

As some activities in AOE network can proceed on in parallel, so the shortest time to complete the project is the length of the longest path from starting vertex to finishing vertex. Here the length of the path is the sum of time to finish each activity on this route. The route with the longest path is called Critical Path. For example, see AOE network in Figure 1, $(v_1, v_2, v_5, v_7, v_9)$ is the critical path [5]. The length of the critical path is 16. That is to say, we need at least sixteen (16) days to finish the whole project. Activities on critical path are called Key Activities. The length of the critical path is the shortest term necessary for the whole project. In another word, to shorten the whole term, we must speed up the progress of the key activities.

Using AOE network in project management is a systematic and common way in modern management and systematic engineering. This technique is called PERT (Program Evaluation Review Technique), which makes an overall balance on time, resource and technique from the point of view to finish the whole project in optimization. PERT is used to arrange the schedule of a plan reasonably by using network technique and system analysis, and then supervise and monitor the schedule during execution thereof to fulfill the prospective goal. The characteristics are embodied in its capability to forecast the key activities and critical paths in the network.

3. Traditional Algorithm and Performance

Traditional algorithm usually computes the possible very early happen time $ve(j)$ and very late happen time $vl(j)$ of each event V_j. $<V_j, V_k>$ denotes the computation of critical paths from the very early starting time $e(i)$ to the very late finishing time $l(i)$ of activity a_i[6,7]. This method will take large amount of calculation, and is very complex and hard to understand[8]. The time complexity of algorithm is $O(e)$, while in reference [9], the time complexity are both $O(n^2)$.

4. Algorithm of Critical Path Based on Dynamic Programming

4.1 Overview of Dynamic Programming

Dynamic Programming was coined by an American mathematician, Richard Bellman in 1950s. As a universal method to optimize the process of multistep solution, it brings up a principle in respect of the issue of optimization, and creates a new kind of algorithm to solve the problems in optimization – the Dynamic Programming, a very important tool of applied mathematics[10-12].

The characteristics of dynamic programming are summarized as follow:

(1) decision-making for separate steps, which is to optimize the problems by using optimization principle.

(2) analysis from the top to the bottom (scale descending), computation from the bottom to the top (scale ascending). The process of computation proceeds on grade by grade, step by step, till the last state.

The basic steps to solve the problems are:

(1) to divide the step;

(2) to select the state;

(3) to determine the decision and write down the state transition equation.

4.2 Solution Algorithm

In an AOE network, see Figure 1, the critical path of this project is the longest path from starting vertex (event) V_1 to finishing vertex (event) V_9. Dynamic programming can be used to solve the problem. Therefore, we can use topological ordering of vertice (events) $(V_1，V_2，V_3，V_4，V_5，V_6，V_7，V_8，V_9)$ to divide the steps as shown in Figure 2.

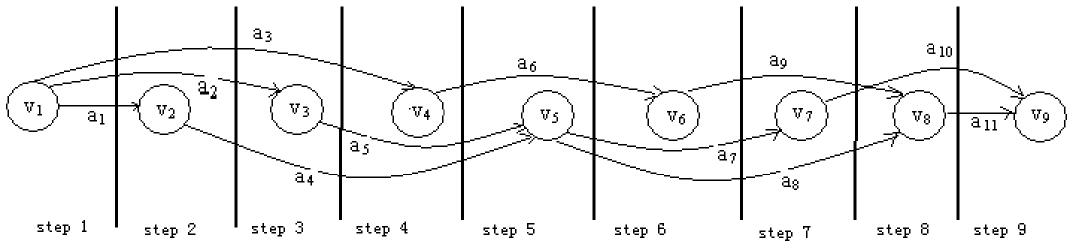

Figure 2. Sectional Diagram

One step has one vertex. The state of each vertex is defined as the length of the longest path from the starting vertex to this vertex. Here we use *LAST[j]* to represent the longest path from starting vertex V_1 to V_j. At the initial, *LAST[1]=0*, then state transition equation is:

LAST[1]=0；

LAST[j]=max{ LAST[k]+dut(<k,j>) }

in which, $Vk \in S$, *S* is the direct set of predecessors of vertex V_j , *dut(<k,j>)* is the duration of activity a_i between event V_k and V_j.

Therefore *LAST[n]* is the longest path from starting vertex (event) to the finishing vertex (event). All paths with such length are called critical paths. In order to get the critical path, we can use a two-dimensional array *PRIOU[n+1][n+1]* to store all these critical paths. If *PRIOU[n][i]* $\neq 0$, then the direct predecessor of finishing vertex *Vn* in one critical path is stored in *PRIOU[n][i]* . The starting vertex V_1 has no predecessor, therefore *PRIOU[1][i] =0*. So through the array *PRIOU[n+1][n+1]* we can get all critical paths. For instance for project shown in Figure 1, *LAST* and *PRIOU* array has the value separately as follow:

LAST:

1	2	3	4	5	6	7	8	9
0	6	4	5	7	7	14	12	16

PRIOU:

	1	2	3	4	5	6	7	8	9
1	0	0	0	0	0	0	0	0	0
2	1	0	0	0	0	0	0	0	0
3	1	0	0	0	0	0	0	0	0
4	1	0	0	0	0	0	0	0	0
5	2	0	0	0	0	0	0	0	0
6	4	0	0	0	0	0	0	0	0
7	5	0	0	0	0	0	0	0	0
8	5	0	0	0	0	0	0	0	0
9	7	8	0	0	0	0	0	0	0

LAST[9]=16 represents the longest path from starting point to finishing point, also the length of critical path, or we may see that is the shortest time to complete the project. From array *PRIOU* we may find that this project has two critical paths, as shown in Figure 3, one is: $V_9 \leftarrow V_7 \leftarrow V_5 \leftarrow V_2 \leftarrow V_1$, the other is $V_9 \leftarrow V_8 \leftarrow V_5 \leftarrow V_2 \leftarrow V_1$.

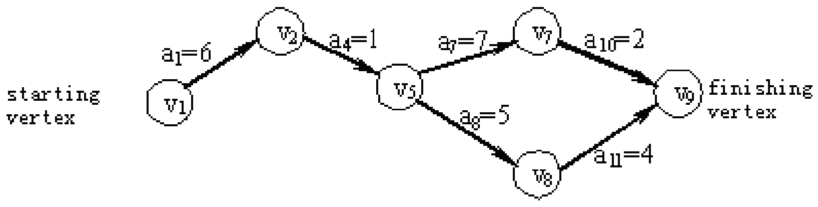

Figure 3. Critical Path in AOE Network Shown in Figure 1

4.3 Fulfillment and Analysis of Algorithm

In order to achieve the said philosophy, we use adjacency list of the reverse graph as the storage structure of AOE network[13,14]. In the adjacency list of the reverse graph, a *dut* field is set on edge node to store the weight of the edge; that is to say the directed edge represents the duration of the activity. The adjacency list of the reverse graph generated from AOE network in Figure 1 is shown in Figure 4.

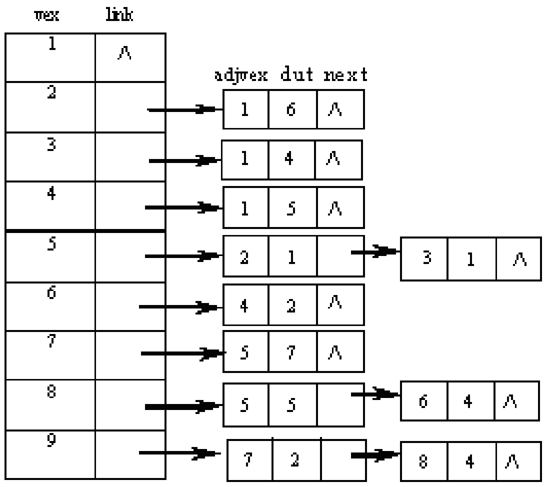

Figure 4. Adjacency List of the Reverse Graph in AOE Network Shown in Figure 1

For convenience of fulfillment, topological order needs to be stored during topologically ordering. If one topological order has already been stored in *tpord[n+1]*, we need to figure out the storage structure first:

```
#define    MaxVertices    20


/*edge node structure information*/
typedef    struct    edgenode1
   {    int adjvex;                          /*field of adjacency vertex*/
        float    dut;                        /*weight (duration of activity)*/
       struct    edgenode1    *next;         /*link field (point to the next arc with
                                               same heads )*/
   } edgenode1;

/* vertex node structure information of header vector */
typedef    struct    vexnode1
   {    int    vex;                        /*vertex information*/
        edgenode1    *link;                /*pointer to the node of first edge* /
   } vexnode1;

/* structure information of adjacency list of the reversed graph */
typedef    struct
{ vexnode1    vertex [MaxVertices];        /* header vector */
  int    vexnum, arcnum;                   /* number of current vertice and arcs of the Figure */
}NAdjList;

float LAST[MaxVertices];            /*array storing the length of the longest path*/
int   PRIOU[MaxVertices]   [MaxVertices];          /*information about storing
                                                     critical paths in PRIOU*/
```

The concrete algorithm to get the critical paths is as follow:

```
void criticalpath (NAdjList G，int tpord[ ])
/* G is the adjacency list of the reverse graph with weight in AOE Network, tpord is for
   storing a topological order */
{
    float LAST[MaxVertices];    /*array storing the length of the longest path*/
    int PRIOU[MaxVertices];     /* information about storing critical paths in
                                   PRIOU */
   edgenode1    *p;
   n= G.vexnum;                      /*number of vertice in the Figure*/
   for (i=1;    i<=n;    i++)    LAST[i]=0.0;    /* LAST set initial value 0*/
   for (i=2;    i<=n;    i++)    /*get the critical paths per the sequence of topological order*/
     {
        j=tpord[i]; l=1;
        p=G. vertex [j].link;
        while (p)
          {
             k=p->adjvex;    /*k is the subscript of one predecessor of v_j*/
             if(p->dut+LAST[k]>=LAST[j])
              if(p->dut+LAST[k]>LAST[j])
                {
                   for (m=2;    m<=l;    m++)    PRIOU[j][m]=0;
                   LAST[j]=p->dut+LAST[k];    /*revise LAST[j]* /
                   l=1;  PRIOU[j][l++]=k;  /*note down the information of
                                             predecessors*/
```

```
                    }
                    else    PRIOU[j][ l++]=k;
                p=p->next;    /*sort out the next of arcs with head $v_j$*/
            }
        }
}
```

/*output the critical paths. Starting from each nonzero term in *PRIOU[n][i],i=1…n* to conduct deep-first search, we can get all critical paths, or output all key activities as follow */

```
    n enqueue;    /*n is the serial number of finished vertice */
    while    (array not empty)
     {
        k= dequeue ;
      for(i=1; PRIOU[k][i];i++)
          {
            PRIOU[k][i]    enqueue ;
           printf("%d←%d,",k, PRIOU[k][i]);
          }
      }
```

5. Result Analysis

In conventional algorithm, it adopts the computation to get the possible very early time *ve(j)* and very late time *vl(j)* of each event, $< v_j, v_k>$ denotes the activity a_i, and the critical path is from its most early starting time *e(i)* to the most late starting time *l(i)*, and the complexity of the algorithm is *O(n+e)*. While in the algorithm presented in this paper, it's only necessary to use the method of dynamic programming to get the critical path in regard to AOE network, which is used to figure out the longest path from the original vertex to others in sequence by one topological order. The longest path from the original vertex to the finishing vertex is the critical path we are trying to get. Getting the longest path to one vertex is only related to its direct predecessor, while in AOE network the edge represents the activity just finished before this event happened. Therefore the total time spending on the longest path from the original vertex to others will also be computed as *O(n+e)* (where *e* is the number of edge, also represents the number of activities, *n* is the number of vertex.). As we can see in an AOE network with a number of activities *e*, a number of events (vertice) *e+1* will happen at the most, see Figure 5.

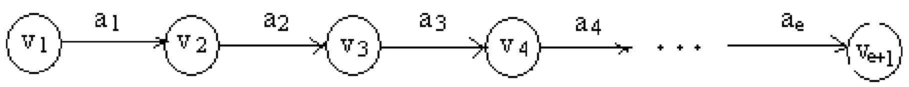

Figure 5. AOE Network with *e+1* Edges

The events are at least 2, see Figure 6.

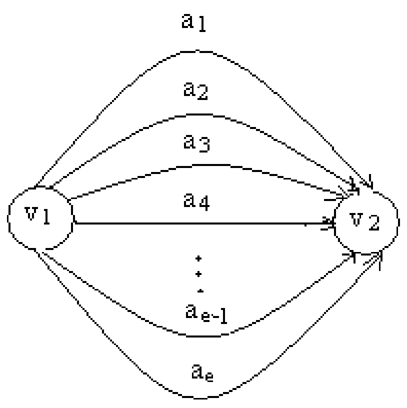

Figure 6. AOE Network with 2 Edges

As *n+e<=2e+1*, so *O(n+e)= O(2e+1)= O(e)*, therefore we may get the time complexity of critical path as *O(e)*.

6. Conclusion

This paper has presented an algorithm to compute the critical path in AOE network by using the philosophy of dynamic programming. This algorithm is simple, intuitive, easy to understand and highly in efficiency. But, it is more reasonable to present a project with the AOV network (vertex for activity, the vertex information for the duration of the activity, directed edge for relationship between the said activities). By combining the judgment of whether the project goes smoothly (topological sort or not) and the sum the shortest completion time of the whole process of the project, the executor of the project can improve the efficiency, but also save space. And then critical path problem in the AOV net is worthy of further research.

7. Acknowledgments

I’d like to give my special thanks to my friend Kehan LI who gave me help on translation of my paper. This research is financially supported by the Education Department of Zhejiang Province (No.9(2011) & No.209(2007)).

8. References

[1] John S. Liu, Louis Y.Y. Lu, “An Integrated Approach for Main Path Analysis: Development of the Hirsch Index as an Example”, Journal of the American Society for Information Science and Technology, Vol.63, No.3, pp.528~542, 2012

[2] Main. Michael, Savitch. Walter, Data Structures and Other Objects Using C++ (4th Edition) , Addison Wesley, 2010

[3] Weifeng Sun, Danchuang Zhang, Yiyang Jia, etc, "A DAG-based Scheduling Algorithm for Dependent Tasks in Grid", JDCTA, Vol. 6, No. 15, pp. 347 ~ 356, 2012

[4] Jyh-Bin. Yang, Chih-Kuei. Kao, “Critical path effect based delay analysis method for construction projects”, International journal of project management, Vol.30, No.3, pp.385~397, 2012

[5] Huahai. Yan, Yinfeng. Xu, “Issue of Critical Edge in Shortest Path in Communication Network with Incomplete Information”, Systems Engineering, Vol.24, No.2, 2006

[6] Kruse. Robert L., Tondo. Clovis L., Leung. Bruce, Data Structures and Program Design in C, Prentice Hall, 1996

[7] Weiss. Mark Allen, Data Structures and Algorithm Analysis in C, Addison-Wesley Educational Publishers Inc, 1996

[8] Q. Duan, T. Warren Liao, “Improved ant colony optimization algorithms for determining project critical paths”, Automation in construction, Vol.19, No.6, pp.676~693, 2010

[9] F. Harary, Graph Theory, Reading, Mass.: Addison-Wesley, 1969

[10] Jiadong. REN, Shiyuan.CAO, Changzhen. HU，“A Hierarchical Clustering Algorithm Based on Dynamic Programming for Categorical Sequences”, Journal of Computational Information Systems, Vol.7, No.5, 2011

[11] Chongguang. REN, Tingwei. WANG, Xiangrui. LI, Guangyi. BAI, “An Improved Adaptive Dynamic Programming Algorithm for Cloud Storage Resource Allocation”, Journal of Computational Information Systems, Vol.7, No.15, 2011

[12] Yi Jiangang, "Research on Stochastic Dynamic Programming Model in the Optimal Scheduling Process of Sinter Conveyors", IJACT, Vol. 3, No. 8, pp. 57 ~ 63, 2011

[13] Kaijian. Liang, Linfeng. Bai, Quan. Liang, etc, “Efficient Resource Reservation Scheme in Service-Oriented Network Environment”, Journal of Networks, Vol.7, No.3, pp.568~575, 2012

[14] L. Chen, “Optimal Computation of Shortest Paths on Doubly Convex Bipartite Graphs”, Computers & Mathematics with Applications, Vol.38, No.3/4, pp.1~12, 1999