International Journal of Mathematics and Computer Research

ISSN: 2320-7167

Volume 13 Issue 06 June 2025, Page no. – 5285-5291

Index Copernicus ICV: 57.55, Impact Factor: 8.615

DOI: 10.47191/ijmcr/v13i6.02

Study of the Structure of Graphs in the Optimal Path Problem: Application of Richards BELLMAN Algorithms

Nkambu Ngoma, R.

Computer Science Department, Higher Institute of Applied Techniques, ISTA-LUKULA, DR Congo

ARTICLE INFO

Published Online:
02 June 2025

Corresponding author:
Nkambu Ngoma, R.

ABSTRACT

Data structures play an important role in problem modeling. Graphs model so many other in different domains, one always has to maximize or minimize the time or either the cost in a routing or scheduling problem.

Transport networks and scheduling problems are two areas where optimization plays a key role.

In this article, we are talking about how to adapt certain types of data by the graph structure. And we study the Shortest Path (SPC) and Longest Path (LPL) problem by applying BELLMAN's algorithms and see how these can be applied in scheduling when it comes to projects. An implementation of these algorithms, as well as an application on scheduling with the Metra Potential Method, in acronym MPM, enrich this article.

KEYWORDS : Connected graph, Transport network, algorithm

I. INTRODUCTION

Graph theory is a modeling and therefore problem-solving tool, used in a large number of disciplines (mathematics, physics, economics, etc.). Concretely, it applies in many fields: Modeling of electrical, computer and telecommunications networks; solving transport routing problems; solving cost minimization problems; solving assignment problems (timetables, transfers, team rotations, etc.). Graph problem solving is a field mainly developed by computer scientists for whom a graph is a data structure, just like a linked list or a table. we translate a concrete problem, which we think can be treated by this theory in terms of graphs; it then becomes a problem of graph theory that we try to solve by making it enter a category of known problems; If this is not the case, it will fuel research until it becomes a solved problem! As data structures, graphs are a generalization of structures not developed in this work (stack, queue, etc.), in that each element of a graph can have multiple predecessors and multiple successors.

In the context of our article, the aim is to implement some problems that arise with graphs: the problem of optimal routing in a transport network and the central problem of scheduling.

Our reflection will revolve around three points: the first point will concern the definitions of what is meant by graph, connected graph and transport network; the second point will present BELLMAN's algorithms as well as their implementations; and the third point will concern graph applications, and finally the conclusion will follow.

II. DEFINITIONS

II.1. GRAPHS

Pierre Lopez defines a graph G as a pair (X, u) where:

- X is a non-empty and at most countable set whose elements are called " vertices " of the graph G.
- u is a family of elements of the Cartesian product $XxX = \{(x,y)/\, x,y \in X\}$ the elements of u are called: bows or edges

A simple graph is loop free and has no multiple links. In a directed graph, we say that y is the successor of x if there is an arc that leads from x to y; we also say that y is adjacent to x. For a directed graph, we simply say that x and y are adjacent. A graph is complete if for every pair of vertices there is an arc (or an edge) joining them. In a directed graph, the exterior [interior] half-degree of a vertex x, which we denote by $d^+(x)$ [$d^-(x)$], is the number of arcs with x as their initial (final) endpoint. The degree of x is $d(x) = d^+(x) + d^-(x)$. For an undirected graph, we define only the degree of a vertex x. In a directed graph, a $d(x)$ path of length L is called a sequence of L+1 vertices

$(s_0, s_1 \cdots s_L)$ such that (s_i, s_{i+1}) forms an arc. For an undirected graph, we speak of a chain. In a directed [undirected] graph, a path [a chain] whose arcs [edges] are all distinct and such that the vertices at the ends coincide is a circuit [a cycle]. A directed graph is strongly connected if for any pair of distinct vertices s and s', there exists a path from s to s' and a path from s' to s. An undirected graph is connected, if for any pair of distinct vertices, there exists a chain joining them. A strongly connected [connected] component is a maximal strongly connected [connected] subgraph.

II.2. RELATED GRAPHS

1. RELATED GRAPH

The graph G = (X, u) is said to be a connected graph if $\forall$ there are distinct pairs of vertices i, j $\in X$, $\exists$ a chain joining i and j.

In other words, a graph is connected if any vertex can be reached from any vertex by traversing different edges.

The relation $\Re$ defined on X by:

$$\forall i, j \in X : i \Re j \Leftrightarrow \begin{cases} - \text{ soit } i = j \\ ou \\ - \text{ soit } i \neq j \text{ et } \exists \text{ une chaîne joignant } i \text{ et } j \end{cases}$$

Is an equivalence relation. The equivalence classes modulo $\Re$ (induced by $\Re$ on X) of x:x1, x2, …, xr form a partition of X.

2. Related component

A connected component of a graph G = (X, u) is a subgraph G A = (A, u A) with A $\subset$X, maximally connected (for inclusion): it is not possible to add other vertices to A while preserving the connectivity of the subgraph.

The subgraphs G1, G2, …, G r generated by the subsets x_1, x_2, …, x_r are called the r-connected components of the graph G.

Strongly connected graph and strongly connected components

Path and circuit

1°) A path of length q is a sequence of q arcs such that the initial end of the next one coincides with the terminal end of the previous arc except the initial end of the path and the terminal end of the path.

2°) A circuit of length q is a closed path (whose 2 initial and terminal ends) coincide.

Strongly connected graph

G = (X, u) is said to be a strongly connected graph if $\forall i, j \in X, avec\ i \neq j, \exists$ a path connects them.

Strongly connected components and the strong connectivity number

Consider the relation $\Re$ defined on x as follows:

$$\forall i, j \in X : i \Re j \Leftrightarrow \begin{cases} - \text{ soit } i = j \\ ou \\ - \text{ soit } i \neq j \text{ et } \exists \text{ un circuit passant par } i \text{ et } j \end{cases}$$

It goes without saying that $\Re$ is an equivalence relation. The module equivalence classes $\Re$ form a partition of X into X1, X2, .., X n .

The subgraphs G_1, G_2, …, G_n, generated by the equivalence classes are strongly connected graphs, we call them the strongly connected components of G = (X, u).

Transport network

Let G = (X, u)

We say that G is weighted if at any arc (i, j)∈ u we associate a real number C ij ;

The matrix $C = (C_{ij})_{(i,j)\in u}$ is called the "weighting matrix."

The triplet G = (X, u, C) is a weighted graph.

Knowing that ∀(i,j)∈ u, the weighting C ij is a numerical value resulting from a measurement (which can be: a distance from vertex i to vertex j, a cost on the arc (i, j), a penalty, example: Payment associated with the arc (i, j); a transition probability from vertex i to vertex j; a duration (time) between i and j; etc.). This measurement is a real number, i.e. it can be positive or negative.

A transport network is a weighted connected graph without loops (networks) or negative-valued circuits.

Otherwise, if we consider a network:

$\Re$ = (X,u , ζ): a weighted graph without loops or negative-valued circuits

$\Re$ = (X, u , ζ) is a transport network ⇔(1) and $C_{ij} \geq 0, \forall (i, j) \in$ u ⇔ (1) and C ij $\geq 0, \forall i, j \in$ X ; with C ij ≡M if (i,j) $\notin$ u . Where M>>0.

M can be the sum of all the weights of the transport network.

3. Bellman algorithms

Richards' Optimality Principal BELLMAN

Any optimal path (minimum or maximum) in a weighted graph can only consist of optimal partial paths. In general, this principle is stated as follows: "Any optimal policy can only consist of optimal sub-policies."

NOTE:

The demonstration of this principle is trivially done by the absurd.

This principle includes the direct statement and its converse.

This principle gives rise to 2 types of analyses:

Prospective analysis: it is this analysis which gives the value of the optimal path.

Retrospective analysis: this is the analysis that allows the optimal path to be constructed, i.e. it allows all the arcs in

the tree structure that are part of the optimal path to be found retrospectively.

This principle is formalized by the Richards BELLMAN equation.

let us consider $\Re$ = (x, u, C) connected and without circuit (quasi-strongly connected).

Let us assume that the vertex considered as entered (root) is number 1 and that the other vertices are arbitrarily numbered 2,3,… , $n = |X|$. $\forall j \in X$ with $j \neq 1, \exists$ some final arc (k, j) in the optimal path from 1 to j $\Rightarrow \forall k \neq j : \ell_j = \ell_k + C_{kj}$.

We then have the following equations:

PCC

PLC

$$(1)\begin{cases} \ell_1 = 0 \\ \ell_j = \min\limits_{k \neq j}(\ell_k + C_{kj}) \end{cases}$$

Note:

At each iteration we only consider the vertices which can be improved at this iteration, that is to say we only consider the vertex whose labels are declared provisional.

System (1) for the PCC or system (1b) for the PLC constitutes the Richards BELLMAN equations. All Tree algorithms builder uses its equations to find the final labels to declare. Once the final labels are found, we note it ℓ_j^* or $\boxed{\ell_j}$.

After the prospective analysis, the labels are ordered as follows: $\ell_j^* < \ldots < \ell_{i_2}^* < \ell_{i_1}^* < \ell_j^*$.

FORD algorithm

Principle

Let $\Re$ = (x, u, C) be a network of order n verifying the classical hypotheses. The algorithm is as follows:

1 Number the vertices in any order; but the input (origin or source) being numbered 1 (or s or even 0).

2 Poser (initialization step):

PLC

$$(1b)\begin{cases} \ell_1 = 0 \\ \ell_j = \max\limits_{k \neq j}(\ell_k + C_{kj}) \end{cases}$$

j=2, 3, …, n

3 At each step (iteration) k+1, (k=0, 1, 2, m) with m finite.

For $(i, j) \in u$.

PCC

PLC

$-si\ \ell_j^{(k)} - \ell_i^{(k)} > C_{ij}$,

Then replace

$\ell_j^{(k+1)}\ par\ \ell_j^{(k+1)} = \ell_i^{(k)} + C_{ij}$

$-\sin on\ \ell_j^{(k+1)} = \ell_j^{(k)}$

4 The iterations (procedures) are stopped when none ℓ_j can be improved further, that is, when all the labels are declared definitive.

5 Check: If there are arcs $(i, j)\ avec\ i > j\ et\ i \neq 1$, all operations must be restarted from j. This return allows nothing to be forgotten and all operations to be made ℓ_j definitive (optimal).

Note:

The FORD algorithm proceeds by prospective analysis to find all the definitive (optimal) labels, i.e. to find the values of optimal partial paths.

Once all the labels are declared definitive, we use the retrospective analysis to construct the branches of the optimal tree, that is to say, to construct the optimal partial paths starting each time from the vertex i k (PCC or PLC) having the largest label.

Noticed:

At each iteration, the FORD algorithm and its variants, among others: BELLMAN KALABA, DIJKSTRA, BELLMAN-FORD, … only consider the vertices whose labels can be improved at this stage, i.e. only consider the successors of vertices whose labels were declared definitive at the previous iteration.

III. APPLICATIONS OF GRAPHS

Two problems are to be solved in this part. These are the optimal routing problem in a transport network covering the strategic environments of the city of Kananga and the task scheduling problem in the construction project of the library.

1. Optimal Path Problem

Using its Tree algorithms builder based on the Richards BELLMAN principle, two trees are to be determined (the Shortest Path and the Longest Path) in a transport network.

Tree algorithms builder as the word indicates allow to build the tree with optimal value between the root (source) and the other vertices of the network. Matrix algorithms on the other hand look for the path with optimal value between all pairs of vertices (x,y) $\in XxX$.

2. ***Scheduling problem***

Scheduling problems initially appeared in the Planning large projects. The goal was to save time on their completion. Such projects consist of many steps, also called tasks. There are temporal relationships between these. For example:

- A stage must start on a specific date;
- A number of tasks must be completed before another can be started;
- Two tasks cannot be performed at the same time (they use the same machine for example);
- Each task requires a certain amount of labor. Therefore, it is important to avoid exceeding the total available labor capacity at any time.

Not all of these constraints need to be taken into account when solving the problem. Here, we will focus only on the types of constraints.

We will seek to determine a schedule, a sequence of steps that minimizes the total time taken to complete the project. From this schedule, we will see that the time of certain steps can possibly be modified without causing a delay to the project, while others, the so-called "critical" steps, delay the project entirely at the slightest local delay.

In all generality, the scheduling problem is posed as follows: Given an objective to be achieved and the achievement of which supposes the execution of a multiple task, which tasks are subject to numerous constraints determining the order and the schedule of execution of these multiple tasks.

Resolution methods

There are two scientific and technical methods based on graph theory to solve the Central Scheduling Problem:

The French method: Metra Potential Method (***MPM***) in 1958.

The American method : ***PERT*** : " Program Evaluation and Research Task " in 1958.

3. The problems to be solved

THE interfaces following have are obtenu es and n executing THE program with the example digital presented in appendix.

Reading of the graph

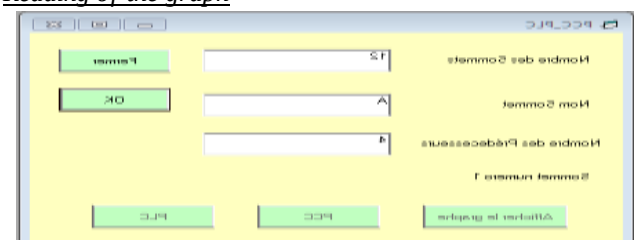

Figure 3.1. Input of vertices and edges

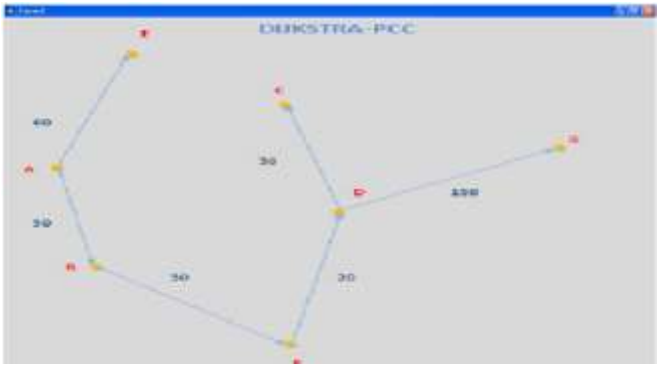

Figure 3.2. Graph PCC

```
Const N = 5
Const INF = 9999
Dim graph(N - 1, N - 1) As Integer
Dim distance(N - 1) As Integer
Dim visited(N - 1) As Boolean
' Initialize the graph with sample data
Sub InitGraph()
    Dim i As Integer, j As Integer
    For i = 0 To N - 1
        For j = 0 To N - 1
            If i = j Then
                graph(i, j) = 0
            Else
                graph(i, j) = INF  ' Default: no direct edge
            End If
        Next j
    Next i
    ' Example edges with weights
    graph(0, 1) = 10
    graph(0, 3) = 30
    graph(0, 4) = 100
    graph(1, 2) = 50
    graph(2, 4) = 10
    graph(3, 2) = 20
    graph(3, 4) = 60
End Sub
' Dijkstra algorithm from a given source
Sub Dijkstra(source As Integer)
    Dim i As Integer, j As Integer
    Dim minDist As Integer, u As Integer
    ' Initialize distances and visited array
    For i = 0 To N - 1
        distance(i) = INF
        visited(i) = False
    Next i
    distance(source) = 0

    ' Loop to find shortest paths
    For i = 0 To N - 1
        minDist = INF
        u = -1
        ' Find the unvisited node with the smallest distance
        For j = 0 To N - 1
```

```
        If Not visited(j) And distance(j) < minDist Then
            minDist = distance(j)
            u = j
        End If
    Next j
    If u = -1 Then Exit For
    visited(u) = True
    ' Update distances of u's neighbors
    For j = 0 To N - 1
        If graph(u, j) < INF Then
            If distance(j) > distance(u) + graph(u, j) Then
                distance(j) = distance(u) + graph(u, j)
            End If
        End If
    Next j
  Next i
End Sub
' Print the shortest distances from the source
Sub ShowDistances()
  Dim i As Integer
  For i = 0 To N - 1
    Debug.Print "Distance from 0 to "; i; " = "; distance(i)
  Next i
End Sub
```

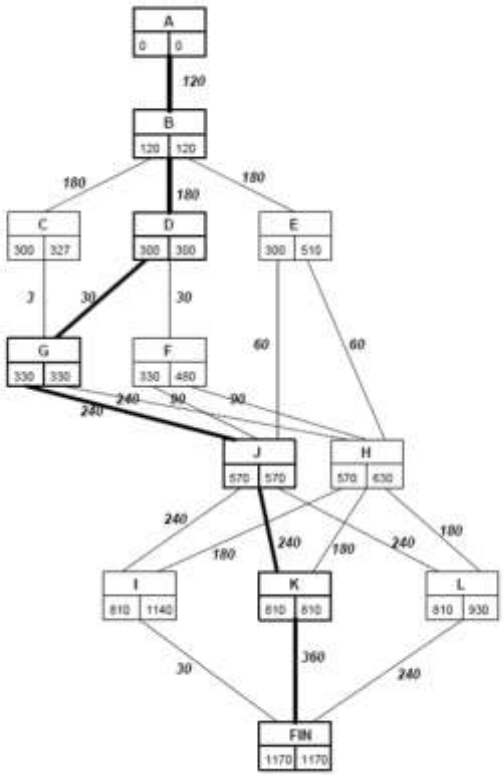

Figure 3.4. Graph MPM

4. Programmation of the a method MPM and PERT

For this program, it is assumed that there can be only one task without a predecessor.

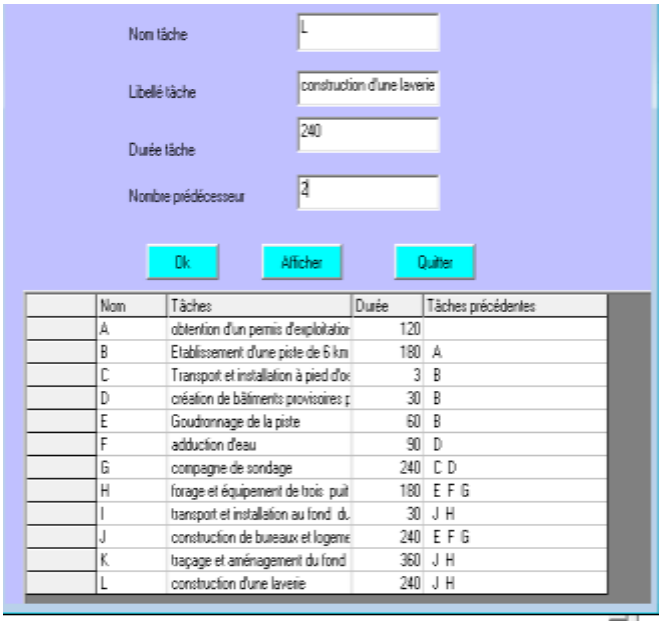

Figure 3.3. Input interface for the predecessor table

```
Const N = 6
Const INF = 9999
Dim duration(N - 1, N - 1) As Integer
Dim TE(N - 1) As Integer ' Earliest time
Dim TL(N - 1) As Integer ' Latest time
Dim visited(N - 1) As Boolean
' Initialize graph with durations
Sub InitGraph()
  Dim i As Integer, j As Integer
  For i = 0 To N - 1
    For j = 0 To N - 1
      duration(i, j) = -1 ' -1 means no arc
    Next j
  Next i
  ' Example graph (activity durations)
  duration(0, 1) = 3
  duration(0, 2) = 2
  duration(1, 3) = 4
  duration(2, 3) = 1
  duration(3, 4) = 2
  duration(4, 5) = 3
End Sub
' Compute earliest times (TE) with forward pass
Sub ForwardPass()
  Dim i As Integer, j As Integer
  For i = 0 To N - 1
    TE(i) = 0
  Next i
  For i = 0 To N - 1
    For j = 0 To N - 1
      If duration(i, j) > -1 Then
```

```
            If TE(j) < TE(i) + duration(i, j) Then
                TE(j) = TE(i) + duration(i, j)
            End If
        End If
    Next j
  Next i
End Sub
' Compute latest times (TL) with backward pass
Sub BackwardPass()
  Dim i As Integer, j As Integer
  Dim maxTime As Integer

  ' Set all TL to project duration
  maxTime = TE(N - 1) ' project total duration = max TE of last node
  For i = 0 To N - 1
    TL(i) = maxTime
  Next i

  For i = N - 1 To 0 Step -1
    For j = 0 To N - 1
      If duration(i, j) > -1 Then
        If TL(i) > TL(j) - duration(i, j) Then
          TL(i) = TL(j) - duration(i, j)
        End If
      End If
    Next j
  Next i
End Sub

' Display TE, TL and Marges
Sub ShowResults()
  Dim i As Integer
  Debug.Print "Node", "TE", "TL", "Marge"
  For i = 0 To N - 1
    Debug.Print i, TE(i), TL(i), TL(i) - TE(i)
  Next i
End Sub
' Main routine
Sub Main()
  InitGraph
  ForwardPass
  BackwardPass
  ShowResults
End Sub
```

- *Complexity : the algorithm implemented to solve this problem is of exponential order: O(n $^{3)}$*

IV. CONCLUSION

Graphs can be used to represent various situations such as: road networks, computer networks, electrical networks, and more. Once the network is represented, we can determine the **longest path (PLC)** or the **shortest path (PCC)**. The PCC can, for example, represent the minimum-cost path in a transport network or a computer network (for packet routing). To determine the PLC or PCC, we referred to several algorithms which we implemented in **Visual Basic**.

In this work, we also highlighted the importance of graphs in **task scheduling**, such as in the construction of a building where elementary tasks are well defined. The **critical path** is determined using the **MPM** or **PERT** methods, which we also implemented in **Visual Basic**.

Recognizing that graphs are data structures like any others (though not fully explored in this article), we presented **tree structures** and **graph structures**, while also analyzing their **computational complexities**.

REFERENCES

1. Pierre Lopez. Graph Course. 2012. Available online at: http://www.laas.fr/~lopez/cours/graphes/spedago.html
2. Pierre Arnoux et al. Graphs for the ES terminal . 2002. Available on the Internet at the following address: http://www.irem.univmrs.fr/productions/nouveautes.php
3. Eric Sigward . Introduction to Graph Theory. 2022. Available online http://www.ac-nancy-metz.fr/ensign/maths/m2002/institut/cadre_institut.html
4. Claude Berge. Graphs. Gauthiers -Villars: Paris. 2013.
5. Pierre Arnoux et al. Graphs in specialist teaching. Appendix to the accompanying document for the ES final year program. 2011. Available online at the following address: www.cndp.fr/lycee/maths/
6. Claudine Robert & Olivier Cogis . Graph theory. Beyond the bridges of Königsberg: problems, theorems, algorithms. Paris: Vuibert, 2013.
7. Roy, B. (1962). *Graphes et ordonnancement: Application de la théorie des graphes à l'ordonnancement.* Revue Française de Recherche Opérationnelle. This is the original publication describing the MPM method developed by Bernard Roy. It's a foundational reference in project scheduling and operations research.
8. Wiest, J. D., & Levy, F. K. (1977). *A Management Guide to PERT/CPM with GERT/PDM/DCPM and Other Networks.* Prentice-Hall. This book includes comprehensive chapters on network-based project scheduling techniques, including MPM and its comparison with PERT and CPM.
9. Gutiérrez-Benítez, J., et al. (2022). *Project Scheduling Using Graph-Based Techniques: A Review.* Journal of Modern Project Management. A modern review of graph-based scheduling methods, including MPM applications and extensions.

10. Dijkstra, E. W. (1959). *A note on two problems in connexion with graphs.* Numerische Mathematik, 1(1), 269–271.
    The classic paper introducing Dijkstra's algorithm, one of the most used for shortest path problems.
11. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.
    Chapter 24 provides detailed analysis of shortest path algorithms including Dijkstra and Bellman-Ford.
12. Geisberger, R., Sanders, P., Schultes, D., & Delling, D. (2008). *Contraction Hierarchies: Faster and Simpler Hierarchical Routing in Road Networks.* In Proceedings of the 7th International Conference on Experimental Algorithms (WEA 2008).
13. Skiena, S. S. (2008). *The Algorithm Design Manual* (2nd ed.). Springer.
14. Ahuja, R. K., Magnanti, T. L., & Orlin, J. B. (1993). *Network Flows: Theory, Algorithms, and Applications.* Prentice Hall.
15. Coursera – Graph Search, Shortest Paths, and Data Structures (Stanford University) https://www.coursera.org/learn/shortest-paths
16. GeeksforGeeks – Shortest Path Algorithms https://www.geeksforgeeks.org/shortest-path-algorithms/
17. Brilliant.org – Graph Theory
    https://brilliant.org/courses/graph-theory/