Scientific Research Publishing

Journal of Applied Mathematics and Physics, 2025, 13(5), 1948-1959
https://www.scirp.org/journal/jamp
ISSN Online: 2327-4379
ISSN Print: 2327-4352

Petri Nets Representation Approach through Euler Graph

Hilaire Nkunzimana[1], Emile Niyongabo[2], Egide Ndayizeye[1], Elie Mushengezi Zihindula[3], Jérémie Ndikumagenge[1]

[1]Center for Research in Infrastructure, Environment and Technologie, University of Burundi, Bujumbura, Burundi
[2]Doctoral School of the University of Burundi, Bujumbura, Burundi
[3]Institut Superieur Pédagogique de Bukavu, Bukavu, Democratic Republic of the Congo
Email: hilaire.nkunzimqnq@ub.edu.bi, niyemi2014@gmail.com, ndayegide2003@gmail.com, eliezihindula@yahoo.fr, jeremie.ndikumagenge@ub.edu.bi

How to cite this paper: Nkunzimana, H., Niyongabo, E., Ndayizeye, E., Zihindula, E.M. and Ndikumagenge, J. (2025) Petri Nets Representation Approach through Euler Graph. *Journal of Applied Mathematics and Physics*, **13**, 1948-1959.
https://doi.org/10.4236/jamp.2025.135108

Received: March 22, 2025
Accepted: May 27, 2025
Published: May 30, 2025

Abstract

For decades, engineers have utilized Petri nets to develop automated systems with specific functional requirements or characteristics. Nonetheless, the existing formalism prevents the use of classic traversal techniques to examine and analyze a system's functional behavior. This paper provides a novel technique to represent Petri nets as directed Euler graphs. It enables design engineers to examine and traverse the various states of a system utilizing the various methodologies and traversal processes applicable to Euler graphs. In this study, we present an iterative approach for determining an optimal path in terms of the least number of edges (vertices) required to cover and contain the system's states, which are represented as Petri nets. The goal is to leave one vertex and return to the same vertex of the graph in a finite number of steps. This is a new method for determining the attribute of the system's reset, which is represented by Petri nets and allows the system to return to its starting state, the resting state. The goal is to broaden the ontological basis of Petri nets by displaying linkages or relationships between Petri nets that are akin to directed Euler graphs.

Keywords

Petri Nets, Euler Graphs, Depth-First and Breadth-First Traversal Search, Iterative Algorithm

1. Introduction

The goal of mathematically modeling a situation is to make it easier to manage and thus analyze.

The modeling of automated systems with Petri nets begins and is based on a bipartite graph [1] [2]. Such a representation does not offer the possibility of traversing the system in order to optimize its multiple states. Current Petri net representation models suffer from optimization challenges induced by the near-impossibility of traversing the functional structure's various states [1]. Furthermore, the lack of iterative techniques for identifying the shortest path in terms of the number of vertexs in the Euler graph model demonstrates problem in determining the ideal path.

In Petri nets, there are no techniques to traverse the states/nodes of the modeled system. For this reason, we have established links between Petri nets and Euler graphs, highlighting the pragmatism of duality and dualism between Petri nets and unweighted directed Euler graphs. Establishing links between the above mathematical tools allows the use of classical graph algorithms in the study and processing of Petri nets for modeling purposes. Thus, the developed iterative algorithm helps in determining the shortest path required for the modeled system to return to the initial state, the resting state.

This paper presents a new strategy based on Petri net representation models in the form of Euler graphs. This type of schematization, both visual and structural, tries to tackle the problem of the inability of traversing the various states of the system that make up the structure of a Petri net as it is now represented [1]. In this work, an iterative algorithm will be proposed for determining the minimum number of vertexs that allow us to leave the top of the graph and return to the same top. The algorithm will serve as a foundation for the study and analysis of one of the modeled system's behavioral properties—system reinitialization.

Designing an iterative approach based on the adjacency matrix produced from directed Euler graphs would allow us to explore a comparable Petri net. The iterative approach will assist us in determining the quickest path for the system to return to the resting state, taking into account the number of edges in the Petri net.

2. Materials, Tools and Methods

2.1. Material and Tools

The work on this topic acts upon on a generalized Petri net and directed Euler graph. Many of the techniques used for graph manipulation, such as Breadth First Search (BFS) and/or Breadth First Search (DFS), will extend to Petri networks.

Based on the Petri net that models the processes of two users on an ATM, as shown in **Figure 1**, we aim to express Petri nets in the form of Euler graphs using oriented arrows. Graphs are excellent for representing natural conditions or events. This representation allows you to browse and optimize the routes between the many states of the represented system. The operations on graphs include, for example, algorithms for exploring these structures by decreasing or maximizing the number of edges or vertices. These processes include depth-first browsing, breadth-first browsing, Eulerian browsing, and Hamiltonian browsing, which provide support for assessing behavioral aspects such as the system's resting state [3] [4].

2.2. Methods

The implementation of the new model of representing generalized Petri nets using Euler graphs with oriented vertexs must be fixed and based on the conservation of the latter's major properties. The current study aims to preserve the formalisms provided by current tools and methods while proposing a set of corrective and novel strategies, as well as an iterative algorithm, for traversing the functional structure of a Petri net and hence the future system. As in the case of Petri net models in the form of bipolar/bipartite Euler graphs, the systems of matrix equations and linear algebras will be sought as techniques and materials for determining the shortest path required that retains the considered property [5].

3. Results and Discussion

3.1. A Case Study on a Petri Net That Models Client Operations at an ATM

Consider the Petri net below as a specific scenario for simulating the user experience in an ATM (Figure 1). We handled the problem of two consumers using a single ATM.

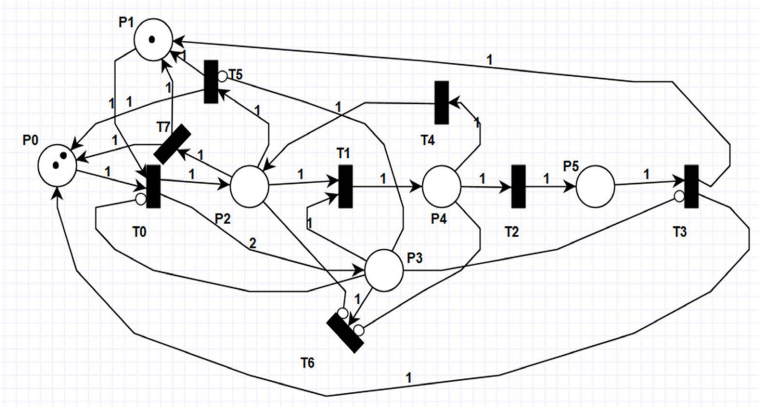

Figure 1. Petri nets modeling the journey of two customers at the ATM.

The ATM Petri Net model has two vertex categories or classes—places and transitions. As shown in Table 1 below, there are six (6) places and eight (8) transitions which model the ATM functional processes.

Table 1. Places and transitions modeling respectively the objects and events of Figure 1.

Places	Description of the role played in the model	Transit ions	Description of the role played in the model
P_0	Two customers waiting to enter the ATM.	T_0	Customer entry at the ATM.
P_1	ATM availability.	T_1	Insertion of the credit card by a customer into the ATM and entry of identification parameters.

Continued

P_2	A customer enters the ATM.	T_2	Credit card acceptance and sufficient balance.
P_3	Number of attempts to insert card or enter authentication parameters.	T_3	Operation completed successfully.
P_4	Credit card inserted into ATM.	T_4	Card refusal or error entering authentication parameters or insufficient balance.
P_5	Client successfully completed the operations	T_5	Cancellation of a customer's presence at the counter when the number of attempts given to a customer has been exhausted or a customer's card is blocked after having completed all the number of attempts granted to him;
	-----	T_6	Cancellation of a number of attempts assigned to a customer when: 1) A customer has successfully completed the withdrawal transaction for the first attempt; 2) A customer leaves the counter without completing the banking transaction.
	-----	T_7	The customer left the counter without doing the banking operations.

3.2. Coverage Graph of the Petri Net Shown Above

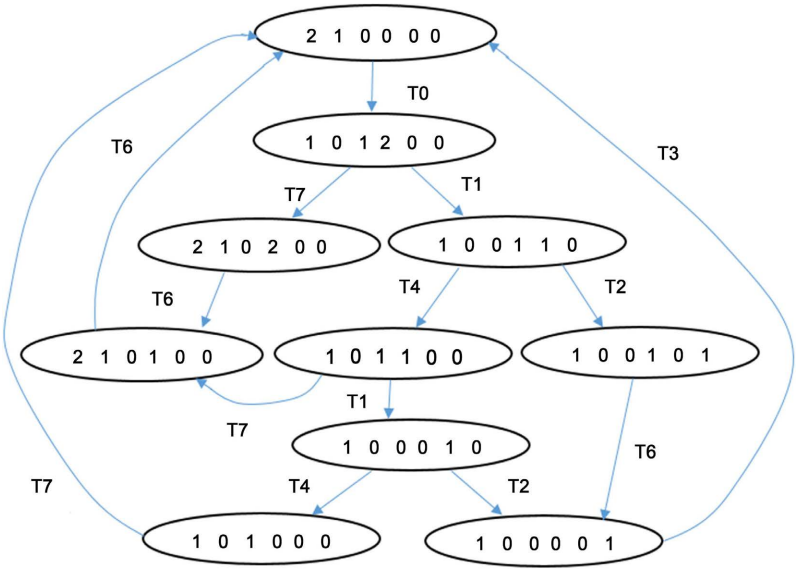

Figure 2. Petri net marking graph models the path of two customers on an ATM.

Figure 2 depicts the dynamic evolution of the system's many states, as depicted in Petri nets in Figure 1. This diagram depicts the modeled system, which is the

path of two consumers on a bank counter represented by Petri nets, having nine states. Drawing the various transitions of the Petri net of the above-mentioned system reveals the system's many states. When we sketch the transition T0 from the beginning state (2 1 0 0 0 0), we obtain the state (1 0 1 2 0 0).

Figure 2 depicts the many states of the system, which are described below:

- $\mathbf{M_0}$ = (2 1 0 0 0 0): This is an initial state. This state illustrates two customers who are in the waiting state and the ATM. The ATM is available to accommodate one customer only;
- $\mathbf{M_1}$ = (1 0 1 2 0 0): One of two customers enters the ATM and the machine offers him the opportunity to take two tries;
- $\mathbf{M_2}$ = (2 1 0 2 0 0): A customer who entered the ATM and left without completing banking transactions;
- $\mathbf{M_3}$ = (1 0 0 1 1 0): A customer inserts his card into the machine and enters the identification parameters. He uses up his first chance to try. He has only one chance to try left;
- $\mathbf{M_4}$ = (2 1 0 1 0 0): One of the two chances given to a customer who was able to enter the ATM and leave without doing banking operations is canceled; When the second chance is canceled, the system returns to the initial state or the idle state;
- $\mathbf{M_5}$ = (1 0 1 1 0 0): A card and/or customer identification parameters are not accepted for the first insertion of the card into the machine. Or a customer's balance may be insufficient. So the operation is failed and a second chance is given to a customer;
- $\mathbf{M_6}$ = (1 0 0 1 0 1): An inserted card is accepted for the first trial chance, the identification parameters are accepted and the balance is sufficient. There is only one trial chance left that the customer has not consumed;
- $\mathbf{M_7}$ = (1 0 0 0 1 0): A customer who has failed the first chance of trying, he inserts his card again into the machine and enters the identification parameters. He consumes his second or last chance of trying;
- $\mathbf{M_8}$ = (1 0 0 0 0 1): An inserted card is accepted for the second try chance, the identification parameters are accepted and the balance is sufficient. There are no try chances left for the customer. When a customer finishes, the system returns to the idle state;
- $\mathbf{M_9}$ = (1 0 1 0 0 0): A customer's card and/or identifying characteristics are not accepted for the second time the card is inserted into the machine. Alternatively, a customer's balance may not be sufficient. So the procedure has failed, and there is no way for a consumer to try. When a consumer exits the counter, the system returns to an idle state.

Each state of the system represents a node. The connection between the nodes is possible thanks to the transitions that allow to transit one state to another state to obtain the dynamic behavior of the system. In this case, the arrow is oriented and the graph is unweighted.

Thus, we obtain the unweighted oriented graph of Figure 3.

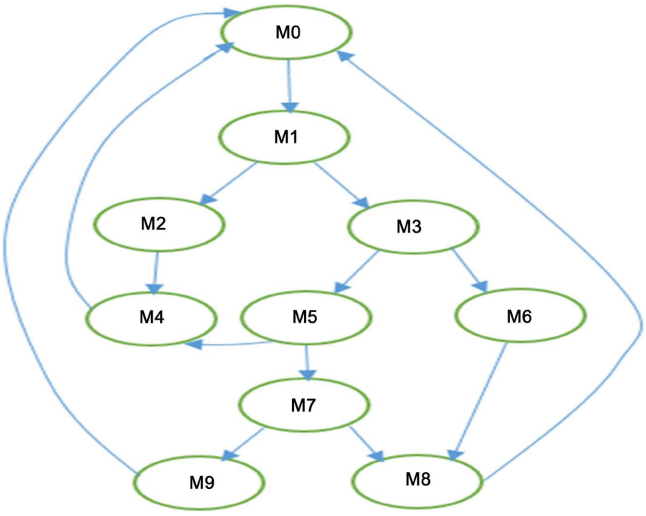

Figure 3. Directed graph of the Petri net modeling the path of two customers.

3.3. Adjacency Matrix of the Directed Graph of the Petri Net

The adjacency matrix represents the number of directed arrows that leave a node to go to another node. We can define the coefficient a_{ij} of the adjacency matrix as the number of links/arrows that connect the vertices i to j. The vertices are numbered from 0 to $n-1$. $\forall n, i, j \in \mathbb{N}$, we have $(0 \le i < n; 0 \le j < n)$. The coefficient $a_{ij}^{(p)}$ defines the number of paths of length p that connect i to j [6].

In our case of an unweighted graph, the relationship between two nodes by a directed arrow is binary. Either there is a relationship or there is no relationship. So there are only two possibilities, respectively 1 and 0. Since it is a directed graph, the relationship between two nodes of a graph is one-way. There is not necessarily a return relationship as in the undirected graph.

Listing 1 below gives the sequence of instructions for a function that determines an adjacency matrix.

```
Function AdjacenceMatrix(): int [ ][ ]
    int i = 0, j;
        int numberLineCol ;
        lire(numberLineCol) ;
        int M[ ][ ] = new [numberLineCol][numberLineCol];
        while(i < numberLineCol)
            j = 0;
            while(j < numberLineCol)
          if(there is an arrow that connects Mi to Mj) then.
              M[i][j] = 1;
          else
              M[i][j] = 0;
                  EndIf
                  j + +;
      EndWhile
      i + +;
    EndWhileI
return M.
```

Listing 1. Algorithm for adjacency matrix determination.

Let us present the nodes of our system represented in Figure 3.

We have ten nodes: M_0, M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8 and M_9, which are connected by directed arrows. The adjacency matrix will be the square matrix of order 10. The rows are called from M_0, M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8 up to M_9 and the columns from M_0, M_1, M_2, M_3, M_4, M_5, M_6, M_7, M_8 up to M_9. Thus we have a 10×10 square matrix.

$$A = \begin{array}{c} \\ M_0 \\ M_1 \\ M_2 \\ M_3 \\ M_4 \\ M_5 \\ M_6 \\ M_7 \\ M_8 \\ M_9 \end{array} \begin{array}{c} \begin{array}{cccccccccc} M_0 & M_1 & M_2 & M_3 & M_4 & M_5 & M_6 & M_7 & M_8 & M_9 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \end{array}$$

If one of the row elements in the adjacency matrix is not zero, it means that the node corresponding to this row, ii^{th} row has a successor of the ji^{th} column, which corresponds to this same row. This node is not a leaf of the graph, indicating that there is no blocking and that the system is alive. This confirms the modeled system's liveliness.

3.4. Liveness Properties and System Reinitialization

Lemmas 1 and 2, formulated and established on the basis of a Petri net, which is represented in the form of an Euler graph, allow us to establish the liveness property (Lemma 1) and the reinitialization property (Lemma 2) of the Petri net, two fundamental properties of the Petri net. The system is alive when the Petri net satisfies the following necessary and sufficient condition: On each row of the adjacency matrix,

Lemme 1:

$$\text{If } \exists a_{ij} \neq 0 \Rightarrow A_j \text{ is Successor of } A_i$$

where:

a_{ij} *is an element of the adjacency matrix*;

A_i *is a starting node*;

A_j *is an arrival node*;

So no blocking, the system is alive.

The system respects the reset property when:

The system respects the reinitialization property when we have the following formula from the adjacency matrix A.

Lemme 2:

$$\begin{cases} 0 \text{ if there is one and only one arrow that directly connects } A_0 \text{ to } A_0. \text{ In this case } p \text{ takes the value 1} \\ \text{Otherwise, we apply the expanded formula, } \prod_{k=1}^{p} M, \text{ until } a_{00} \neq 0 \end{cases}$$

$\forall k$ and $\forall p \in \mathbb{N}_0$ and $p \geq k$

p *is the number of arrows that make up the shortest path*;

a_{00} *is the number of paths*;

M *is the adjacency matrix*;

The multiplication process is stopped when the first element of the matrix, a_{00}, is different from zero. The variable p is the number of links/vertex that make up the optimal length of a chain to return to the initial state.

Listing 2 below gives the optimal number of vertex/links that form the length of a chain allowing one to leave a vertex and return to the same vertex.

```
Algorithm Calculation_OptimalChainLength
    int coeff = 0;
    int nbreLigneCol ;
    lire(nbreLigneCol)
        int matriceAdjacence[ ][ ] = FunctionAdjacenceMatrix ( ) ;
        int prod [ ][ ] = new int[nbreLigneCol][nbreLigneCol];
        int matrr [ ][ ] = new int [nbreLigneCol][ nbreLigneCol] ;
        int n = 1, i, j, k;
      for(i = 0; i < nbreLigneCol; i + +){
            for(j = 0; j < nbreLigneCol; j + +){
                for(k = 0; k < nbreLigneCol; k + +)
                  {
                    matrr[i][j] = matrr[i][j] + matriceAdjacence[i][k] *
matriceAdjacence[k][j];
                  }
            }
        }
        n + +;
        while(coeff ! = 0)
        {
            for(i = 0; i < nbreLigneCol; i + +){
                  for(j = 0; j < nbreLigneCol; j + +){
                      for(k = 0; k < nbreLigneCol; k + +)
                      {
                        prod[i][j] = prod[i][j] + matrr[i][k] *
matriceAdjacence[k][j];
                      }
                  }
            }
            n + +;
            coeff = prod[0][0];
            for(i = 0; i < nbreLigneCol; i + +){
            for(j = 0; j < nbreLigneCol; j + +){
                matrr[i][j] = prod[i][j];
            }
        }

            for(i = 0; i < nbreLigneCol; i + +){
            for(j = 0; j < nbreLigneCol; j + +){
                prod[i][j] = 0;
            }
        }
        }

Display ("Number of links that allow you to leave the rest state and return to the same state:" +
n);
          Display ("Number of paths : ");
        for(i = 0; i < 1; i + +){
            for(j = 0; j < 1; j + +){
                Display("" + matrr[i][j]);
            }
        }
    }

End Algorithm
```

Listing 2. Algorithm for optimal chain length determination.

The algorithm in Listing 2 helps us find the optimal number of links/arrows, or the length of the chain leaving the vertex and returning to it. It determines the best path from M_0 back to M_0. It is the minimum number of arrows/links required to establish the system reset property. This is conceivable if the coefficient a_{00}, the first member of the adjacency matrix, is greater than zero. We can draw the following conclusion: if the last nodes have a successor, and that successor is a vertex, then the modeled system is alive and resettable.

When we apply the technique described above to the adjacency matrix of the graph illustrated in Figure 3, the calculation of matrix multiplication ends when the initial coefficient of the matrix is not zero. Thus, we get the matrix shown below.

$$M^4 = \begin{pmatrix} 0&1&0&0&0&0&0&0&0&0\\ 0&0&1&1&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&0&0&0\\ 0&0&0&0&0&1&1&0&0&0\\ 1&0&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&0&0&1&0&0\\ 0&0&0&0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0&0&1&1\\ 1&0&0&0&0&0&0&0&0&0\\ 1&0&0&0&0&0&0&0&0&0 \end{pmatrix}^4 = \begin{pmatrix} 1&0&0&0&1&0&0&1&1&0\\ 2&1&1&1&0&0&0&0&1&1\\ 0&0&1&1&0&0&0&0&0&0\\ 2&2&0&0&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&0&0&0\\ 0&2&1&1&0&0&0&0&0&0\\ 0&0&1&1&0&0&0&0&0&0\\ 0&0&2&2&0&0&0&0&0&0\\ 0&0&0&0&1&1&1&0&0&0\\ 0&0&0&0&1&1&1&0&0&0 \end{pmatrix}$$

The coefficient $a_{00}^{(4)} = 1$ 1 defines the number of paths of length 4 that connect M_0 to M_0.

The number of pathways is 1 and the ideal length is 4. This indicates that an optimal path composed of four arrows connects M_0 to M_0.

3.5. Iterative Algorithm for Shortest Path Determination and Its Complexity

3.5.1. Iterative Algorithm for Shortest Path Determination

Listing 3 three below provides the sequence of instructions for an algorithm that determines the shortest path, in terms of the number of vertexs, that allows a return to the starting state, the initial state.

```
Algorithme Calcul_LongueurOptimaleChaine
          int coeff=0;
          int nbreLigneCol ;
          lire(nbreLigneCol)
          int matriceAdjacence[][] = FunctionAdjacenceMatrix() ;
          int prod [][]= new int[nbreLigneCol][nbreLigneCol];
          int matrr [][]=new int [nbreLigneCol][nbreLigneCol] ;
          int n=1,i=0,j,k,m=0;
          while(i<nbreLigneCol) do
               j=0;
               while(j<nbreLigneCol) do
                    k=0;
                    while(k<nbreLigneCol) do
                         matrr[i][j] = matrr[i][j]+matriceAdjacence[i][k]*matriceAdjacence[k][j];
                         k++;
                    endWhileK
                    j++;
               endWhileJ
               i++;
          endWhileI
          n++;
          i=0;
          coeff=matrr[0][0];
```

```
24          while(coeff==0) do
25              while(i<nbreLigneCol) do
26                    j=0;
27                    while(j<nbreLigneCol) do
28                          k=0;
29                          while(k<nbreLigneCol) do
30                                prod[i][j] = prod[i][j] + matrr[i][k] * matriceAdjacence[k][j];
31                                k++
32                          endWhileK;
33                          j++;
34                    endWhileJ
35                    i++;
36              endwhileI
37              n++;
38              coeff = prod[0][0];
39              i=0;
40              while(i<nbreLigneCol) do
41                    j=0;
42                    while(j<nbreLigneCol) do
43                          matrr[i][j]=prod[i][j];
44                          j++;
45                    endWhileJ
46                    i++;
47              endwhileI
48            i=0;
49            while(i<nbreLigneCol) do
50                j=0;
51                while(j<nbreLigneCol) do
52                   prod[i][j]=0;
53                   j++;
54                endWhileJ
55                i++;
56            endWhileI
57            m++;
58        endWhile_Coeff_Diff_Zero
59        Afficher ("Nombre de lien qui permet de quitter  l'etat de repos et retourner a cet meme etat: ",n);
60        Afficher ("Nombre de chemin(s) : ",coeff);
61
62  Fin Algorithme
```

Listing 3. Algorithm for the determination of shortest required for system's state initialization.

3.5.2. Determination of the Complexity of the Developed Algorithm

For the algorithm below, let (n) be the execution time as a function of the argument n and c_i be the time cost of line i. The calculation of the worst-case complexity by executing the lines is determined as follows. For simplicity, we will use instructions where there are four (4) nested loops.

Thus, to determine the worst-case complexity of the algorithm, we start from line 24 and stop at line 30.

The time complexity of all instructions executed in four nested loops, from C_{24} to C_{30}, is given by the expression or equation. Let n be equal to nbLineCol.

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\sum_{i=0}^{n-1}\left(C_{25}+C_{26}+\sum_{j=0}^{n-1}\left(C_{27}+C_{28}+\sum_{k=1}^{n-1}\left(C_{29}+C_{30}\right)\right)\right)\right)$$

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\sum_{i=0}^{n-1}\left(C_{25}+C_{26}+\sum_{j=0}^{n-1}\left(C_{27}+C_{28}+\left(C_{29}+C_{30}\right)\sum_{k=1}^{n-1}1\right)\right)\right)$$

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\sum_{i=0}^{n-1}\left(C_{25}+C_{26}+\sum_{j=0}^{n-1}\left(C_{27}+C_{28}+\left(C_{29}+C_{30}\right)n\right)\right)\right)$$

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\sum_{i=0}^{n-1}\left(C_{25}+C_{26}+\sum_{j=0}^{n-1}\left(C_{27}+C_{28}\right)+\sum_{j=0}^{n-1}\left(C_{29}+C_{30}\right)n\right)\right)$$

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\sum_{i=0}^{n-1}\left(C_{25}+C_{26}+\left(C_{27}+C_{28}\right)\sum_{j=0}^{n-1}1+\left(C_{29}+C_{30}\right)\sum_{j=0}^{n-1}n\right)\right)$$

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\sum_{i=0}^{n-1}\left(C_{25}+C_{26}+\left(C_{27}+C_{28}\right)\sum_{j=0}^{n-1}1+\left(C_{29}+C_{30}\right)n^2\right)\right)$$

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\sum_{i=0}^{n-1}\left(C_{25}+C_{26}+\left(C_{27}+C_{28}\right)n+\left(C_{29}+C_{30}\right)n^2\right)\right)$$

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\left(C_{25}+C_{26}\right)\sum_{i=0}^{n-1}1+\left(C_{27}+C_{28}\right)\sum_{i=0}^{n-1}n+\left(C_{29}+C_{30}\right)\sum_{i=0}^{n-1}n^2\right)$$

$$T(n)=\sum_{p=1}^{m}\left(C_{24}+\left(C_{25}+C_{26}\right)n+\left(C_{27}+C_{28}\right)n^2+\left(C_{29}+C_{30}\right)n^3\right)$$

$$T(n)=C_{24}\sum_{p=1}^{m}1+\left(C_{25}+C_{26}\right)\sum_{p=1}^{m}n+\left(C_{27}+C_{28}\right)\sum_{p=1}^{m}n^2+\left(C_{29}+C_{30}\right)\sum_{p=1}^{m}n^3$$

$$T(n)=C_{24}m+\left(C_{25}+C_{26}\right)n\times m+\left(C_{27}+C_{28}\right)n^2\times m+\left(C_{29}+C_{30}\right)n^3\times m$$

$$T(n)\in\theta\left(n^3\times m\right)$$

The worst-case complexity namely the real complexity of the developed algorithm is $\theta\left(n^4\right)$.

3.6. Discussions

Petri nets can be represented as an Euler graph, which offers several advantages. It enables for the study and optimization of the paths of the system's various states, which are represented by Petri nets. In this paper, we present an iterative technique for determining the lowest number of arrows required to leave a vertex and return to the same vertex in a finite number of edges, or steps. The purpose is to establish and implement the reinitialization property for the system's states, which are represented by Petri net models.

This algorithm can be used to determine the minimal number of vertexs in a graph, identify a route from the initial state to the same starting state while making more feasible chess moves, and so on. The concept of the shortest path is well defined in terms of the number of vertexs traveled, if there exists a path from vertex s to vertex t [5].

4. Conclusion

The depiction of Petri nets as directed Euler graphs allows for the extension of the Petri nets' ontological basis. With the necessary specifications for a new representation of the latter as directed Euler graphs, we were able to develop an iterative algorithm that allows to determine a shortest path in a graph, by making fewer possible moves in chess, to be able to confirm that the system modeled as Petri nets maintains the stable stationary state on its future system's different states [3] [4]. The directed Euler graph also helped us establish the qualities of liveliness and boundedness.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Peterson, J.L. (1981) Petri Net Theory and the Modeling of Systems. Prentice Hall PTR.

[2] Piétrac, L. and Denis, B. (1999) Une approche de méta-modélisation formelle des méthodes de conception des systèmes automatisés de production. *Journées Doctorales d'Automatique* (*JDA*'99), hal-03745632.

[3] Schwarzentruber, F. (2021) ALGO1-Pvertexours en largeur. https://people.irisa.fr/Francois.Schwarzentruber/algo1/06parcourslargeur.pdf

[4] Bordat, J.P. (1991) Calcul des idéaux d'un ordonné fini. *RAIRO-Operations Resevertexh*, **25**, 265-275. https://doi.org/10.1051/ro/1991250302651

[5] Maquin, D. (2003) Eléments de Théorie des graphes. Ecole Nationale Supérieur d'Electricité et de Mécanique.

[6] Attali, H., Buscaldi, D. and Pernelle, N. (2024) Matrice d'adjacence courbée: Intégration de la courbure des arêtes dans la transmission des messages. 24*ème conférence francophone sur l'Extraction et la Gestion des Connaissances EGC* 2024, Dijon, hal-04568325, 70-82.