Available online www.ejaet.com

European Journal of Advances in Engineering and Technology, 2022, 9(12):90-100

Research Article ISSN: 2394 - 658X

Matlab-Based Modelling and Dynamic Optimization of a Class of Sequential Decision Processes

Jean Mbihi[1], Paul Owoundi Etouké[2] and Arnaud Biyobo Obono[3]

[1,2,3]Research Laboratory of Computer Science Engineering and Automation
ENSET, University of Douala, P.O. Box 1872, Douala, Cameroon
[1]mbihidr@yahoo.fr, [2*]etouke_paul@yahoo.fr, [3]obonobiyo@yahoo.fr

---

ABSTRACT

In this paper, the discrete models of two types of sequential decision processes (i.e. open and closed graph topologies) are developed. Under the adopted counting policy of nodes, it is shown that a sequential open graph topology with n levels along the x-axis, involves a total of n (n+1)/2 states (nodes). However, for a closed graph topology with 2n-1 levels along the x-axis, it is shown also that the total number of state (nodes) is n2. In addition, for both types of open and closed graph processes, their dynamic state models are outlined, and the overall cost optimization problems are transformed into HJB (Hamilton-Jacobi-Bellman) Matrix equations, associated with state model constraints. Furthermore, a set of custom flowcharts are established in order to develop corresponding Matlab custom solvers. Finally, the main results obtained from the analysis of relevant case studies, show the simplicity and great conviviality of the proposed solvers for optimal sequential decision processes.

Key words: Sequential Decision processes, open and closed topology, dynamic models, dynamic programming principle, HJB Equations, custom Matlab solvers

---

INTRODUCTION

The sequential decision processes are widely encountered in applied science and engineering applications, e.g. management science, operations research, manufacturing management systems, communication networks and feedback control systems [1]. However, without lost of generality, the class of sequential decision processes studied in this paper, consist of two classes, i.e. open graph and closed graph topologies. Each sequential graph candidate under study consists of one root node (or state) at level 1, followed by N_k states of the same generation at stage k. Without lost of generality, a constant cost function is incurred from any single state to the next, and the relevant problem under consideration is to compute a target path (if any), associated with the optimum cost flow problem.

Many software tools are available in the scientific literature, for solving minimum cost flow problems. A few examples are Linear Network Optimization Software [2], Large Scale Nonlinear Network Optimizer [3], and even C-simPLEX [4]. Among these optimization tools there are numerous embedded Matlab solvers, e.g. [5] : *graph, treeplot, treelayout, addnode, addedge, addnode, subgraph, shortest*, etc. However, the great limitation of these embedded solvers, relies on the fact that they behave as input-output black boxes, with neither algorithmic details on their intrinsic code, nor possible opportunities for either their extension or improvement. Therefore, a more efficient solution approach for Matlab programmers, is to reinforce and extended the capabilities of current Matlab editions by suitable and custom functions, to be well checked and published for possible use by a wide community of Matlab users.

The scientific contribution to be outlined in this paper relies the following assumptions: a) The class of sequential decision process considered consists of sequential graphs; b) each graph topology can be either open or closed: c) the nodes of a graph are organized into x-axis layer levels with $x \in \{1, 2, \ldots, n, \ldots\}$, and y-axis levels with $y \in \{1, 2, \ldots, m, \ldots\}$; d) a constant forward cost is attached to any direct path between two successive nodes; e) A direct increasing path between 2 nodes i and j respectively, incurred a cost $C_{up}(i, j)$; f) A direct decreasing path between 2 nodes *k* and *p* respectively, incurred a cost $C_{up}(k, p)$. Given these basic assumptions, the next sections of this paper,

consist of : a) Development of well tested dynamic models and optimization schemes of open and closed graphs processes; b) Results and discussions when testing the proposed custom Matlab dynamic models and solvers on as sample of decision graph processes; c) Conclusion and future perspectives.

Dynamic Model and Optimization Scheme of Open Graph Decision Processes

Matlab matrix topology of open graph decision processes

Fig. 1 shows Matlab Matrix topology of an open graph process involving *n* levels along the x-axis, consists of $N_{op}(n)$ nodes, $R_{op}(n)$ rows and $C_{op}(n)$ columns. Using a recurrent reasoning, it is easy to show that $N_{op}(n)$, $R_{op}(n)$ and $C_{op}(n)$ can be explicitly defined by Equation (1).

$$\{(a)\ N_{op}(n) = n(n+1)/2 \quad (b)\ R_{op}(n) = 2n-1; \quad (c)\ C_{on}(n) = n \tag{1}$$

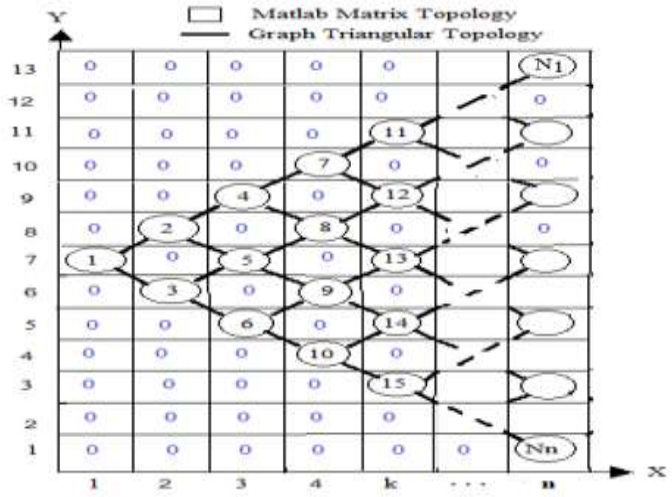

Fig. 1 Matlab Matrix topology of open graph processes

As an implication, given in Fig. 1 the nodes state-space, with index *k* defined by (2). Therefore, a straightforward analysis indicates that the resulting *n-stage* open-graph decision process, can be coded by a (2 *n* – 1) *x n* matrix.

$$k \in \{1, 2, \ldots, n, \ldots, n(n+1)/2\}, \tag{2}$$

In addition, unlike standard matrix schemes used in graph theory (Ex: node-arc incidence matrix, node-node adjacent matrix, adjacency lists, …, see [6] for details), an open graph *Gop* defined by Equations (3) can be built in such a way that its digital form maintains the structure as well as the states number of the original graph. Let consider *n* = 5 without lost of generality, then the digital code of G_{op} matrix is a 9 *x* 5 matrix given by (3).

$$Gop = \begin{pmatrix} 0 & 0 & 0 & 0 & 11 \\ 0 & 0 & 0 & 7 & 0 \\ 0 & 0 & 4 & 0 & 12 \\ 0 & 2 & 0 & 8 & 0 \\ 1 & 0 & 5 & 0 & 13 \\ 0 & 3 & 0 & 9 & 0 \\ 0 & 0 & 6 & 0 & 14 \\ 0 & 0 & 0 & 10 & 0 \\ 0 & 0 & 0 & 0 & 15 \end{pmatrix} \tag{3}$$

Matlab matrix costs of open graph decision processes

Let us consider for the sake of better understanding, a simple case with n = 5 (level on the open graph). Under these conditions, equations (4) show the structure of the resulting *dual costs Matrix* C_{up} (for all increasing paths) and C_{down} (for decreasing paths) from corresponding left starting nodes. It is worth nothing that the terminal colons of both dual matrix costs must be set to known identical cost values.

$$Cup = \begin{pmatrix} 0 & 0 & 0 & 0 & C11 \\ 0 & 0 & 0 & Cup\ 7,11 & 0 \\ 0 & 0 & Cup\ 4,7 & 0 & C12 \\ 0 & Cup\ 2,4 & 0 & Cup\ 8,12 & 0 \\ Cup\ 1,2 & 0 & Cup\ 5,8 & 0 & C13 \\ 0 & Cup\ 3,5 & 0 & Cup\ 9,13 & 0 \\ 0 & 0 & Cup\ 6,9 & 0 & C14 \\ 0 & 0 & 0 & Cup\ 10,14 & 0 \\ 0 & 0 & 0 & 0 & C15 \end{pmatrix} \quad Cdown = \begin{pmatrix} 0 & 0 & 0 & 0 & C11 \\ 0 & 0 & 0 & Cdown\ 7,12 & 0 \\ 0 & 0 & Cdown\ 4,8 & 0 & C12 \\ 0 & Cdown\ 2,5 & 0 & Cdown\ 8,13 & 0 \\ Cdown\ 1,3 & 0 & Cdown\ 5,9 & 0 & C13 \\ 0 & Cdown\ 3,6 & 0 & Cdown\ 9,14 & 0 \\ 0 & 0 & Cdown\ 6,10 & 0 & C14 \\ 0 & 0 & 0 & Cdown\ 10,15 & 0 \\ 0 & 0 & 0 & 0 & C15 \end{pmatrix} \tag{4}$$

Matrix models (3) and (4), can be generated for any open graph with length *n* on the x-axis. Fig. 2 shows the

flowcharts of the proposed custom Matlab functions to be implemented for automatic computing of matrix data associated with open graph topology and corresponding dual matrix costs.

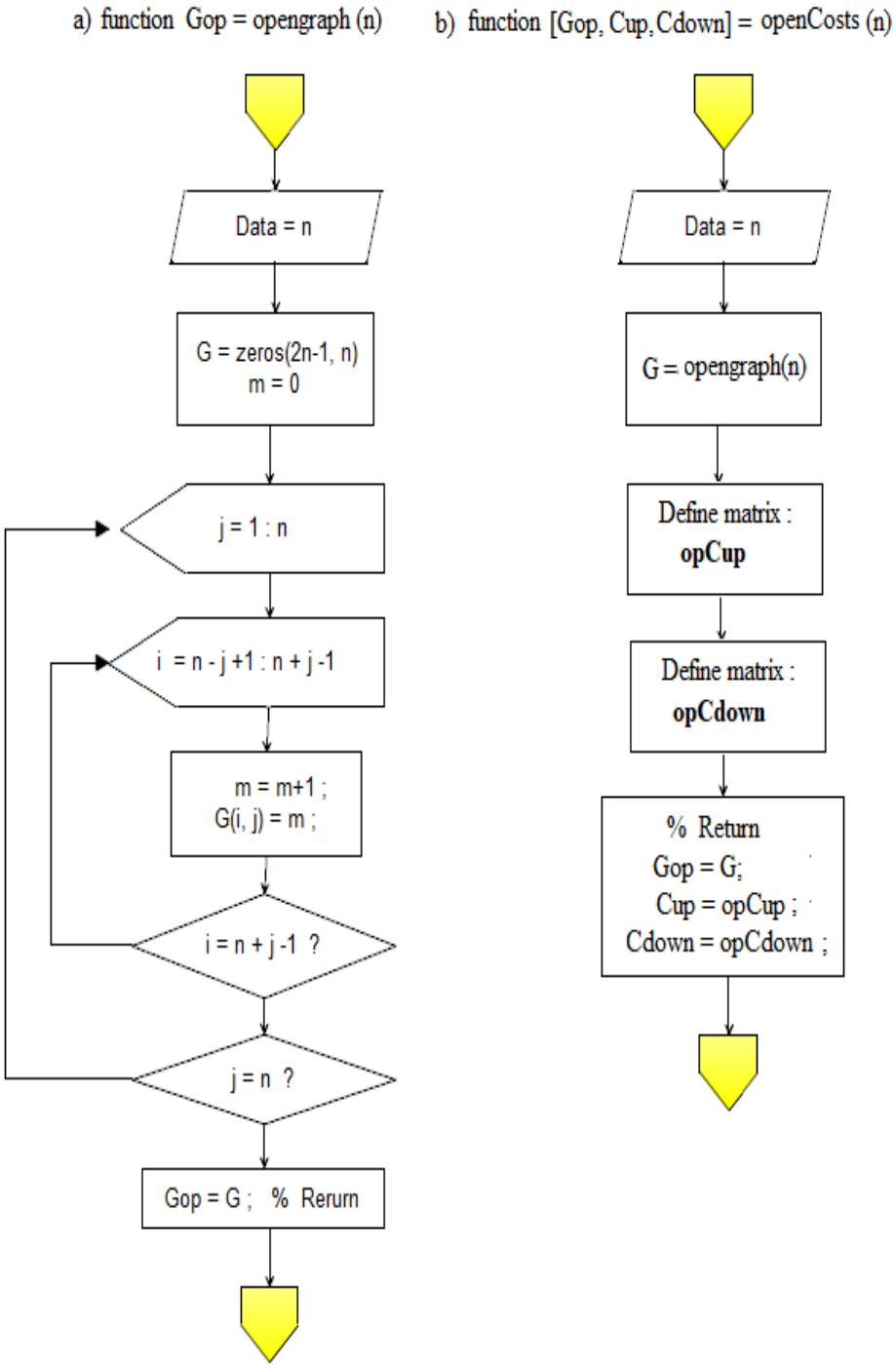

Fig. 2 Flowcharts of custom Matlab functions *opengraph.m* and *opencost.m*

Dynamic model of open graph decision processes

Consider the state space $\chi = \{1, 2, 3, \ldots, n\ (n+1)/2\}$ adopted in Figure 1, and a decision space U = {-1, 0, 1}. For each $u \in U$ and from any possible state $x \in X$, u = -1 if the downward path is chosen from x, and u = 1 if an upward path is chosen, and u = 0 for any right side terminal node. Subsequently, a dynamic model resulting from the matrix model described by (5). In the notation $x_{i,j}$, *i* stands for any appropriate row of the matrix *Gop*.

$$x_{i,k+1} = \begin{cases} x_{i+1,k} + k \quad if \quad u_{i+1,k} = 1 \ \ (upward \quad path) \\ x_{i-1,k} + k + 1 \ \ si \ \ u_{i-1,k} = -1 \ \ (downward \quad path \ \ ) \end{cases} \quad with \quad k = 1,2,\ldots,n-1 \qquad (5)$$

HJB equations open graph decision processes

For an open graph topology, the analysis of the minimum cost flow problem relies on a straightforward application of the dynamic programming principle. Therefore, the optimal solution can be computed by solving the following HJB (*Hamilton-Jacobi-Bellman*) equation [6].

$$J(x_{i,k})) = \underset{ui,k}{Min}\left(Cup_{i,k} + J^*(x_{i-1,k+1}),\ Cdown_{i,k} + J^*(x_{i+1,k+1})\right) \quad (6)$$

In (6), k = n-1, n-2, …, 1 (processing backward index), whereas *i* stands for an appropriate raw of Matlab cost matrix, i.e. *Gop* (3). The state $x_{i,k}$ is defined as in (5), J*(.) being the cost-to-go function. Therefore, given the dynamic state model (5), the resulting optimal outcomes (i.e. controls, states and associated costs) of the , can be computed according to a backward recursive strategy using a custom Matlab function, .

Flowchart of the Open graph optimization scheme

Fig. 4 shows a simple custom flowchart of the optimal closed graph decision processes. It has been easily developed and tested using Matlab script programming language.

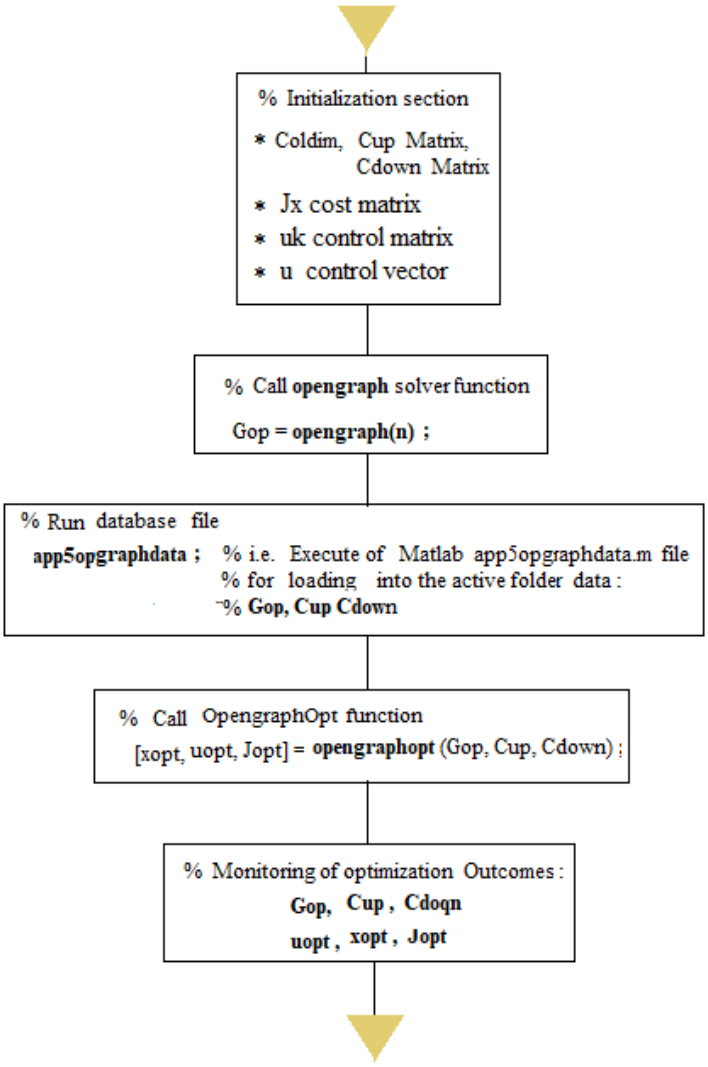

Fig. 3 Flowchart of the open graph optimization scheme

Dynamic Model and Optimization Scheme of Closed Graph Decision Processes

Matlab Matrix topology and digital map of closed graph decision processes

Fig. 4 shows Matlab Matrix topology of a closed graph process involving 2*n*-1 levels along the x-axis. For this more intricate case however, the required design parameters, i.e. $N_{cl}(n)$ nodes, $R_{cl}(n)$ rows and $C_{cl}(n)$ columns, are given by the set of Equations given by (7).

(a) $N_{cl}(n) = \frac{n(n+1)}{2} + \frac{n(n+1)}{2} - n = n^2$; (b) $R_{cl}(n) = 2n - 1$; (c) $C_{cl}(n) = 2n - 1$ (7)

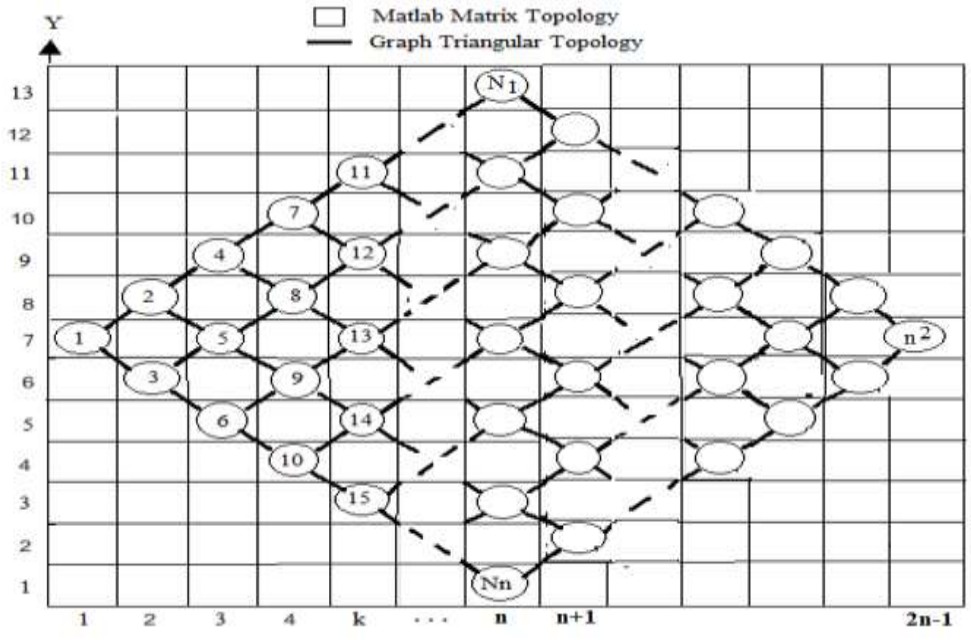

Fig. 4 Matlab matrix topology and digital map of closed graph processes

Matlab matrix topology of closed graph decision processes

Given the state-space {1, 2, …, n, n+1, …, 2n-1} considered in Fig. 4, and following Equations (8), it becomes obvious that any (2n-1)-*stage* closed graph, can be digitally coded by a square (2n-1) x (2n-1) matrix G_{cl}. As an example, for n = 5, then (2n-1) = 9, in which case the G_{cl} matrix is given as by (8).

$$Gcl = \begin{pmatrix} 0 & 0 & 0 & 0 & 11 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 7 & 0 & 16 & 0 & 0 & 0 \\ 0 & 0 & 4 & 0 & 12 & 0 & 20 & 0 & 0 \\ 0 & 2 & 0 & 8 & 0 & 17 & 0 & 23 & 0 \\ 1 & 0 & 5 & 0 & 13 & 0 & 21 & 0 & 25 \\ 0 & 3 & 0 & 9 & 0 & 18 & 0 & 24 & 0 \\ 0 & 0 & 6 & 0 & 14 & 0 & 22 & 0 & 0 \\ 0 & 0 & 0 & 10 & 0 & 19 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 15 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (8)$$

Matlab Matrix costs of closed graph decision processes

From Fig. 4 where n = 5, the associated dual matrix costs C_{up} and C_{down} are defined by (9) and (10) respectively. In this case, the total number of involve nodes is $n^2 = 25$ states (nodes). Even in this case, Matrix terms (8), (9) and (10) can be automatically generated for any arbitrary size of x-axis, using simple custom Matlab functions.

$$Cup = \begin{pmatrix} 0 & 0 & 0 & 0 & Cup11 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Cup7,11 & 0 & Cup16 & 0 & 0 & 0 \\ 0 & 0 & Cup4,7 & 0 & 12 & 0 & Cup20 & 0 & 0 \\ 0 & Cup2,4 & 0 & Cup8,12 & 0 & Cup17,20 & 0 & Cup23 & 0 \\ Cup1,2 & 0 & Cup5,8 & 0 & Cup13,17 & 0 & Cup21,23 & 0 & Cup25 \\ 0 & Cup3,5 & 0 & Cup9,13 & 0 & Cup18,21 & 0 & Cup24,25 & 0 \\ 0 & 0 & Cup6,9 & 0 & Cup14,18 & 0 & Cup22,24 & 0 & 0 \\ 0 & 0 & 0 & Cup10,14 & 0 & Cup19,22 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Cup15,19 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (9)$$

$$Cdown = \begin{pmatrix} 0 & 0 & 0 & 0 & Cdown\,11,16 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Cdown\,7,12 & 0 & Cdown\,16,20 & 0 & 0 & 0 \\ 0 & 0 & Cdown\,4,8 & 0 & Cdown\,12,17 & 0 & Cdown\,20,23 & 0 & 0 \\ 0 & Cdown\,2,5 & 0 & Cdown\,8,13 & 0 & Cdown\,17,21 & 0 & Cdown\,23,25 & 0 \\ Cdown\,1,3 & 0 & Cdown\,5,9 & 0 & Cdown\,13,18 & 0 & Cdown\,21,24 & 0 & Cdown\,25 \\ 0 & Cdown\,3,6 & 0 & Cdown\,9,14 & 0 & Cdown\,18,22 & 0 & Cdown\,24 & 0 \\ 0 & 0 & Cdown\,6,10 & 0 & Cdown\,14,19 & 0 & Cdown\,22 & 0 & 0 \\ 0 & 0 & 0 & Cdown\,10,15 & 0 & Cdown\,19 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Cdown\,15 & 0 & 0 & 0 & 0 \end{pmatrix} \quad (10)$$

Fig. 5 shows the flowcharts of the aforementioned custom Matlab functions. According to their declared syntaxes, they can be either executed from Matlab command Windows, or called from a target Matlab subroutine.

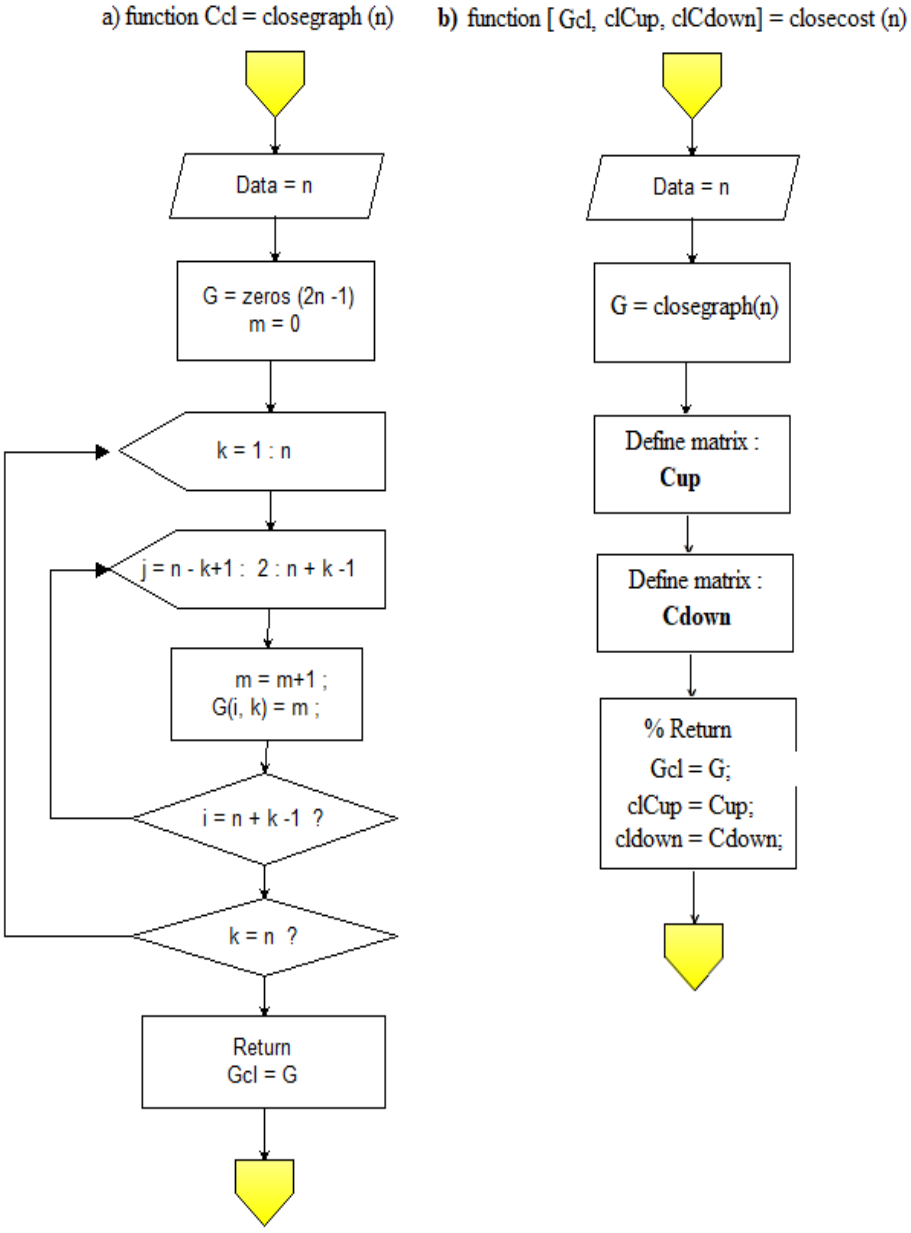

Fig. 5 Flowcharts of custom Matlab functions *closegraph.m* and *closecost.m*

Dynamic model of closed graph decision processes

Given the state space $\chi = \{1, 2, 3, \ldots, N = n^2\}$ adopted for a closed-graph decision process, as well as a control decision space U = {-1, 0, 1}, our analysis indicates that a complete dynamic model of the resulting sequential process, can be exactly defined according to the recursive law (11).

$$x_{i,k+1} = \begin{cases} \left.\begin{matrix} x_{i+1,k} + k & if \;\; u_{i+1,k} = 1 \;\; (upward \;\; path) \\ x_{i-1,k} + k + 1 & if \;\; u_{i-1,k} = -1 \;\; (downward \; path \;\;) \end{matrix}\right\} & \text{for } k = 1, 2, \ldots, n-1 \\ \left.\begin{matrix} x_{i+1,k} + 2n - 1 - k & if \;\; u_{i+1,k} = 1 \;\; (upward \;\; path) \\ x_{i-1,k} + 2n - k & if \;\; u_{i-1,k} = -1 \;\; (downward \; path \;\;) \end{matrix}\right\} & \text{for } k = n+1, n+2, \ldots, 2n-1 \end{cases} \quad (11)$$

with x (n, 1) = 1, and x (2n –1, 2n-1) = n^2.

HJB equation of closed graph decision processes

The HJB Equations of a closed graph is structurally similar to that obtained in the open topology case, with similar boundary conditions. These HJB equations are defined by (12), where k = 2n-1, 2n-2, …, 1 (backward sequence index).

$$J(x_{i,k}) = \underset{ui,k}{Min}\left(Cup_{i,k} + J^*(x_{i-1,k+1}),\; Cdown_{i,k} + J^*(x_{i+1,k+1})\right) \quad (12)$$

As a conclusion, the set of equations {(11), (12)} can be organized as an overall Matlab file, e.g. ***closegraphopt.m***, with input argument (G, Cup, Cdown) and outputs .

Flowchart of the closed graph optimization scheme

Fig. 6 stands for the flowchart of closed graph optimization processes. Although it seems long, it has been easily implemented and well tested using Matlab script programming language.

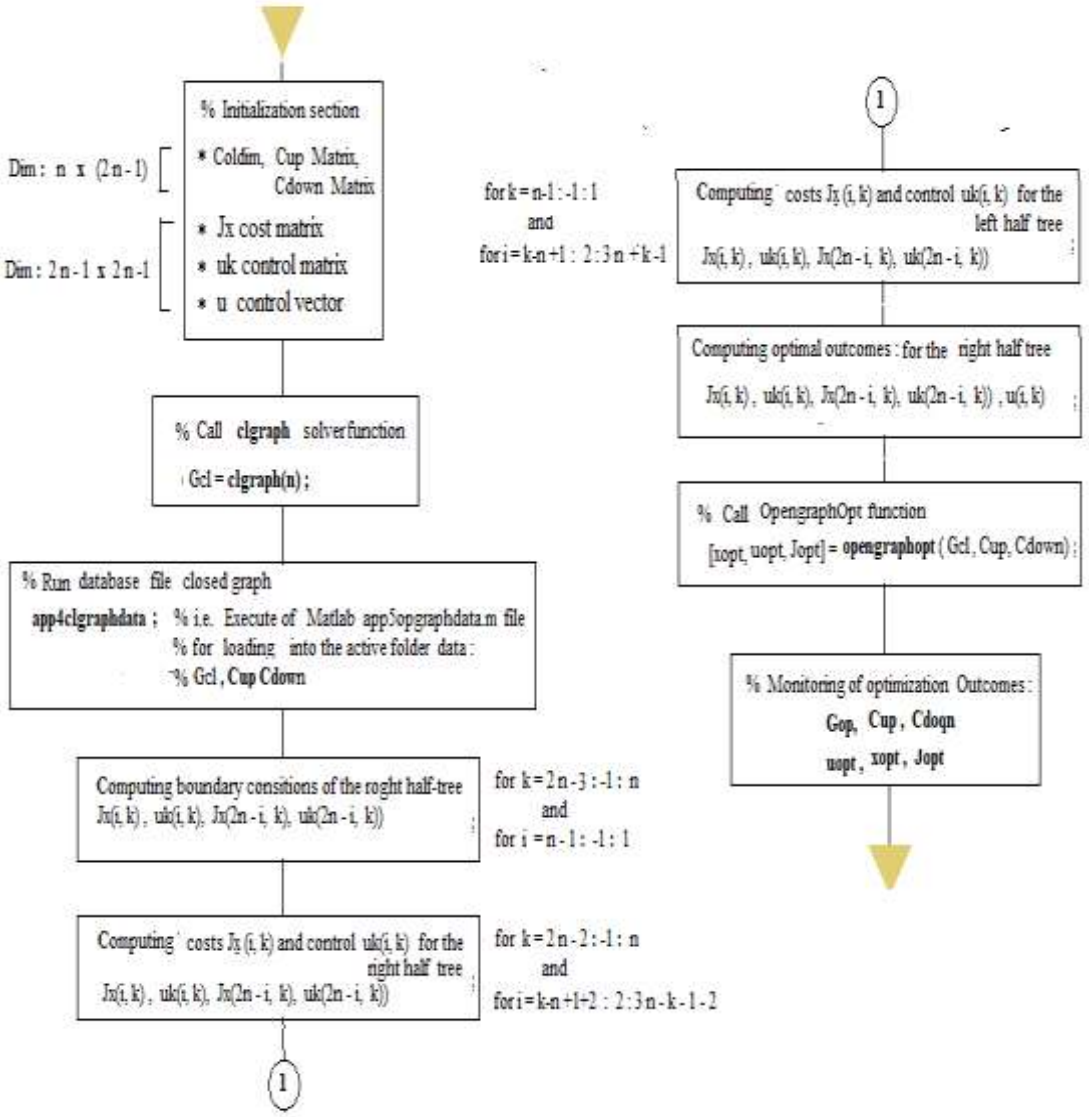

Fig. 6 Flowchart of the closed graph optimization scheme

RESULTS AND DISCUSSIONS

A few relevant technical details are required for a better understanding of the strategy used to obtain the simulation results to be presented in this section. It is worth noting that the whole analytical developments and flowcharts presented in this paper, have been implemented as Matlab Script components, before been organized into a custom Matlab toolox consisting of embedded solvers for : a) Dynamic models; b) Associated HJB equations; c) Optimal state feedback control problems and more.

In addition, for the sake of better simplicity and conviviality when using the proposed custom Matlab toolbox, the user only needs is to create and save a data file of a target application into Matlab active folder, and then to open Matlab application main frame, for updating the name of existing data file. As illustrative examples, Fig. 7 shows Matlab application main frame for *open graph decision processes* with n = 5 levels and n (n+1)/2 nodes, in which the actual Matlab data file (*app5opengraphdata.m*) is called in line 6. Similarly, in Fig. 8 corresponding to closed graph processes, the corresponding Matlab data file (*app4closegraphdata.m*) is called in line 6 too. Then, using the proposed Matlab application main frame is simple, friendly and convivial for Matlab users.

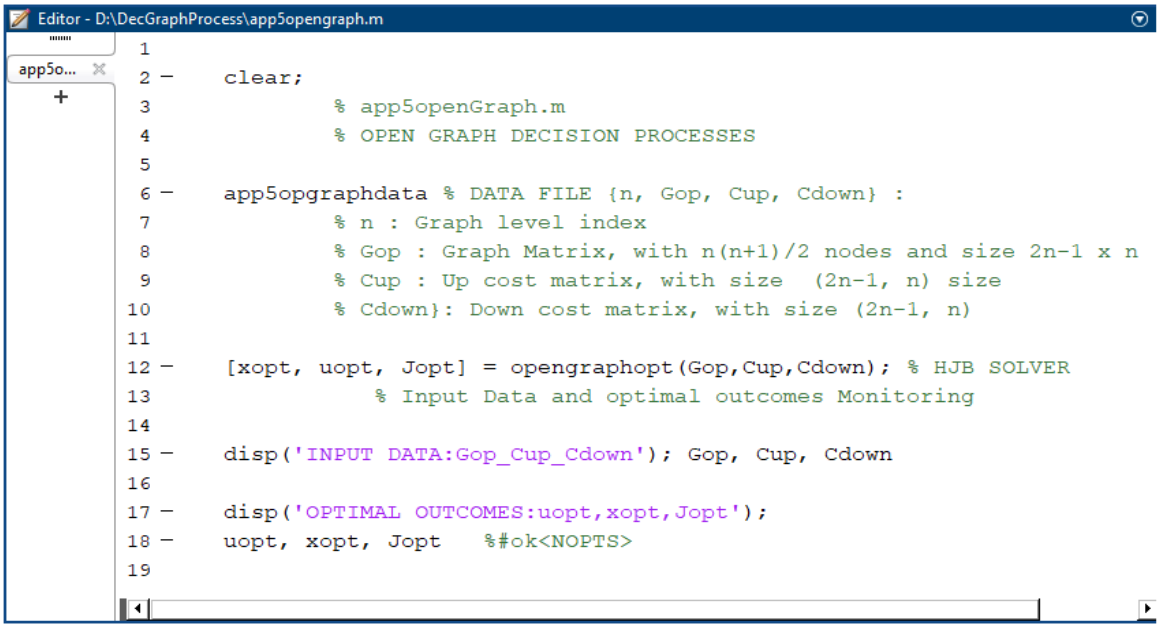

Fig. 7 Main frame of Matlab *app5openGraph.m*

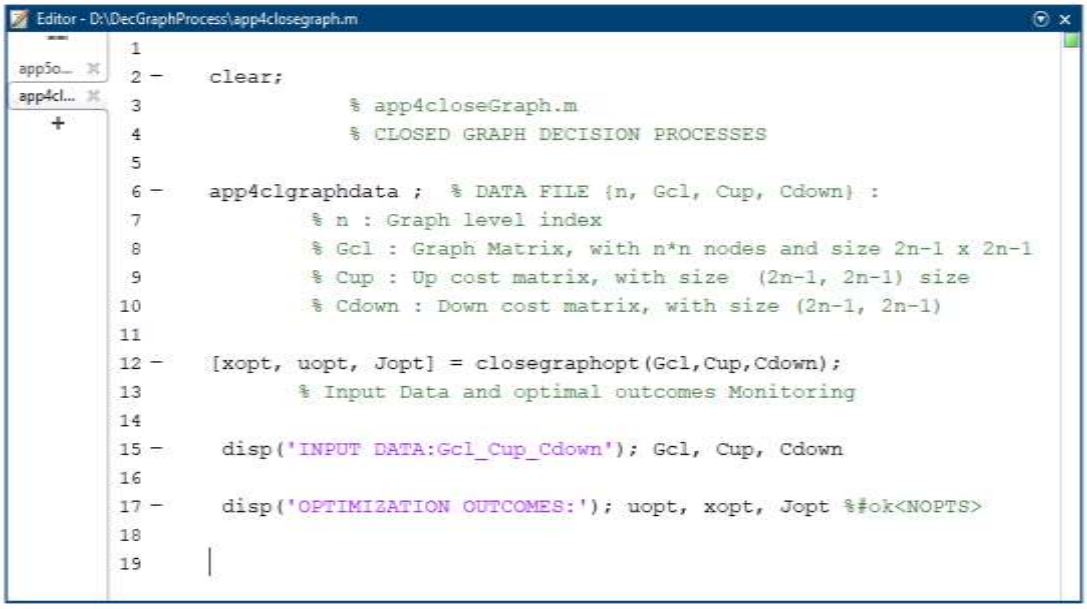

Fig. 8 Main frame of Matlab *app4closeGraph.m*

The following text shows the results monitored on Matlab command, when execution Matlab main frame in Fig. 7. They have been transferred here by copy and paste action.

```
>>
>> app5opengraph

INPUT DATA From app5opgrapdata.m :
n =
     5
INPUT DATA:Gop_Cup_Cdown
Gop =
     0     0     0     0    11
     0     0     0     7     0
     0     0     4     0    12
     0     2     0     8     0
     1     0     5     0    13
     0     3     0     9     0
     0     0     6     0    14
     0     0     0    10     0
     0     0     0     0    15

Cup =
     0     0     0     0    10
     0     0     0     1     0
     0     0     2     0     5
     0     2     0     2     0
     2     0     1     0     1
     0     2     0     3     0
     0     0     2     0     7
     0     0     0     1     0
     0     0     0     0     3

Cdown =
     0     0     0     0    10
     0     0     0     2     0
     0     0     1     0     5
     0     1     0     4     0
     1     0     2     0     1
     0     1     0     2     0
     0     0     2     0     7
     0     0     0     1     0
     0     0     0     0     3

OPTIMAL OUTCOMES : uopt, xopt, Jopt
uopt =
    -1    -1     1     1     0
xopt =
     1     3     6     9    13

Jopt =
     0     0     0     0    10
     0     0     0     7     0
     0     0     6     0     5
     0     7     0     5     0
     8     0     6     0     1
     0     7     0     4     0
     0     0     6     0     7
     0     0     0     4     0
     0     0     0     0     3

>>
```

The following result also shows the outcomes monitored on Matlab command, when execution Matlab main frame in Fig. 7.

>> **app4closegraph**

INPUT DATA From app4clgrapdata.m :

n =

4

INPUT DATA:Gcl_Cup_Cdown

Gcl =

0 0 0 7 0 0 0
0 0 4 0 11 0 0
0 2 0 8 0 14 0
1 0 5 0 12 0 16
0 3 0 9 0 15 0
0 0 6 0 13 0 0
0 0 0 10 0 0 0

Cup =

0 0 0 2 0 0 0
0 0 5 0 0 0 0
0 10 0 9 0 0 0
7 0 8 0 8 0 4
0 5 0 9 0 6 0
0 0 10 0 5 0 0
0 0 0 11 0 0 0

Cdown =

0 0 0 7 0 0 0
0 0 7 0 9 0 0
0 10 0 7 0 7 0
12 0 12 0 7 0 4
0 5 0 9 0 0 0
0 0 8 0 0 0 0
0 0 0 0 0 0 0

OPTIMIZATION OUTCOMES:

uopt =

1 1 -1 -1 -1 1 0

xopt =

1 2 4 8 12 15 16

Jopt =

0 0 0 23 0 0 0
0 0 **27** 0 16 0 0
0 **37** 0 **20** 0 7 0
44 0 28 0 **13** 0 0
0 33 0 20 0 **6** 0
0 0 30 0 11 0 0
0 0 0 22 0 0 0

>>

A set of our custom Matlab solvers initiated in this paper with successful implementation and tests, are summarized in Table 1. Depending on the type of decision process under study, the optimal control decisions of the minimum cost flow problem, can be backwardly solved off-line, and then saved for real processing application needs of dynamic programming strategy.

CONCLUSION

The custom software solvers implemented in this research paper, could be used to extend the use of Matlab optimization toolbox to the considered class of sequential decision processes. As an implication, the minimum cost flow problems could be solved using the set of well tested Matlab scripts initiated in this research. They are more straightforward and easier to handle modelling and optimization tools compared to compiled MEX-files requiring external solving environment. However, a few technical singularities, have not been explicitly outlined in our research methodology, e.g. a) What would happen under two (or more) paths with the same overall cost value ? b) What would happen if some internal nodes of the sequential decision graph involves more than 2 input cost flows, or more than 2 output cost flows, or a single input or single output cost flow ?. Furthermore, in order to capture the attention of a maximum community of Matlab users, it would be interesting to migrate the proposed custom Matlab script solvers, into Matlab Advanced development technologies, e.g. Matlab Live Script, Matlab GUIDE, Matlab Application Designer, Matlab Deployed App, etc. These numerous scientific perspectives will be investigated in Future research Works.

REFERENCES

[1]. J. A. Bondy and U. S. R. Murty, *Graph Theory and applications*, © Elsevier Science Ltd, 264 pages, 1984.
[2]. Bertsekas D. P. (1991). *Linear Network Optimization: Algorithms and codes* , MIT Press, USA.
[3]. Toint P. L. and Tuyttens D (1992). LSNNO: *A fortran Subroutine for solving large scale nonlinear network optimization problem, ACM trans. Math. Software, Vol* 18 , 308-328.
[4]. ILOG (2002), A*MPL CPLEX System: Version 8.0, User's manual,* ILOG *P*ress.
[5]. Mathworks, Matlab documentation Home, version 2020a (9.8.0.1323502).
[6]. Ravindra K. et al (1993). *Network flows*, Chap. 2, pp 23-46, Prentice-Hall.