Modeling, Identification and Control, Vol. 40, No. 1, 2019, pp. 1–10, ISSN 1890–1328

Finding Clusters in Petri Nets An approach based on GPenSIM

R. Davidrajuh [1] D. Krenczyk [2] B. Skolud [2]

[1] *Electrical and Computer Engineering, University of Stavanger, N-4036 Stavanger, Norway.* *E-mail:* reggie.davidarjuh@uis.no

[2] *Faculty of Mechanical Engineering, Silesian University of Technology, 44-100 Gliwice, Poland.* *E-mail:* {damian.krenczyk,bozena.skolud}@polsl.pl

Abstract

Graph theory provides some methods for finding clusters in networks. Clusters reflect the invisible grouping of the elements in a network. This paper presents a new method for finding clusters in networks. In this method, the user can adjust a parameter to change the number of clusters. This method is newly added to the simulator General-purpose Petri Net Simulator (GPenSIM) as a function for network analysis. With this GPenSIM function, in addition to the usual performance analysis of a discrete-event system via a Petri net model, supplementary information about the grouping of the elements can also be found. Finding clusters in discrete-event systems provides valuable information such as the ideal location of the elements in a manufacturing network. This paper also presents an application example on a flexible manufacturing system.

Keywords: Clusters, peer-pressure method, Petri Nets, GPenSIM, Flexible Manufacturing System

1 Introduction

Finding clusters in a manufacturing network is a very useful thing to do. This is because clustering finds groups in a network where the elements within a group are connected with each other somehow. It is usual to group elements of a network into clusters based on their functional belonging. This technique is known as the logical clustering. For example, in an automobile manufacturing facility, the manufacturing elements can be grouped into painting cluster, engine assembly clusters, and electrical installation cluster. Apart from the logical clustering, graph theory provides some methods for finding clusters in networks by analyzing the static network. For example, Peer-pressure method (Kepner and Gilbert, 2011) and Markov clustering (Peterson, 1981) analyze the static network based on how the elements (nodes) in the graph are connected.

This paper focuses on finding clusters in Petri nets. Petri net is a bipartite graph which is widely used for modeling, simulation and (supervisor-based) control of discrete-event systems (Peterson, 1981). Finding clusters in Petri nets possesses an additional advantage: in addition to the usual performance analysis of Petri nets through simulations, static cluster analysis can also be done on the Petri net. Performing both static and dynamic analysis of a discrete-event system paves a deeper understanding of the system. For example, dynamic analysis can identify the elements that are the bottlenecks in the system, whereas static cluster analysis can show how these elements are grouped together.

In this paper, the software General-purpose Petri Net Simulator (GPenSIM) (Davidrajuh, 2018; Davidrajuh et al., 2018b) is used for both static and dynamic analysis of discrete-event systems (e.g., manufacturing networks); GPenSIM is developed by the first author of this paper. GPenSIM is used in the

doi:10.4173/mic.2019.1.1

literature for dynamic analysis of a variety of discrete-event systems (Abbaszadeh et al., 2018; Cameron et al., 2015; Jyothi, 2012; Mutarraf et al., 2018). This paper is the first attempt to perform static cluster analysis with GPenSIM.

The originality of this paper: This paper proposes an extension the Peer-pressure method so that the number of clusters in a network can be varied by the user. The theoretical work towards this extension and its implementation as a function of the GPenSIM software can be considered as the original work of this paper.

In this paper: Section-II introduces GPenSIM as a tool for modeling, simulation, and performance evaluation of discrete-event systems, based on Petri net models. Section-III is about static cluster analysis using the peer-pressure method. Section-IV describes the new extension to the peer-pressure method. Section-V presents the GPenSIM functions for cluster analysis. Section-VI presents an application example on flexible manufacturing system. The simulation results from the cluster analysis are discussed in section-VII.

2 Introduction To GPenSIM

General-purpose Petri net simulator (GPenSIM) is a toolbox on MATLAB platform which can be used for modeling, simulation, performance analysis, and control of discrete-event systems. GPenSIM is relatively new software as it was introduced in 2016 as version 9; the current version is v10. GPenSIM is already being used by some universities around the world, e.g., in Australia, China, Korea, and the USA. The reasons being the simplicity of learning and using, and its flexibility to incorporate newer functionality (Davidrajuh, 2018; Davidrajuh et al., 2018b; Abbaszadeh et al., 2018; Cameron et al., 2015; Jyothi, 2012; Mutarraf et al., 2018).

Implementing a Petri net model with GPenSIM usually happens via four M-files (Davidrajuh, 2018; Davidrajuh et al., 2018b):

1. Petri net Definition File (PDF): A PDF declares the static Petri net graph: the set of places, the set of transitions, and the set of arcs are declared in this file.

2. Main Simulation File (MSF): The MSF declares the initial dynamics (e.g., initial tokens in the places, firing times of the transitions, firing costs of the transitions) and runs the simulations. When the simulation terminates, the code for plotting and printing the simulation results are also coded in this file.

3. The pre-processor file (COMMON_PRE): If there are additional conditions for the enabled transitions to satisfy before firing, these conditions are coded in the COMMON_PRE file.

4. The post-processor file (COMMON_POST): If there are any post-firing actions to be performed after firing of transitions, these actions can be coded in the COMMON_POST file.

For static analysis of Petri nets, we need only the files MSF and PDF. This is because the pre- and post-processor files are for run-time simulations (dynamic analysis).

3 Cluster analysis: The basics

Finding clusters (*aka* graph clustering) in a network is determining groups of nodes that are highly connected with each other. Clusters are similar but different to components (or strongly connected components); in a strongly connected component, any two nodes within the component must be connected in both directions (Cormen et al., 2009). There are many methods for finding clusters, starting with older and less efficient random walks based methods (Křivánek and Morávek, 1986) to modern and efficient peer-pressure based (Gilbert et al., 2007) and flow-based methods (van Dongen, 2000).

3.1 Peer-pressure based clustering

The peer-pressure based method focuses on the pulling power (*aka* votes) of the neighbors. A node can be pulled by many clusters surrounding it. This node will join the cluster that possesses the highest pulling power.

The pull experienced by a node is the incoming edges to the node. The pulling strength of a cluster is the incoming edges from the nodes of that cluster to this particular node.

Peer pressure clustering is an iterative method, repeatedly calculating the pulls experienced by the individual nodes and allocating them to different clusters according to the pull strength. The method stops when the movements of nodes between the clusters converge (stabilize) meaning there is no movement in successive iterations.

3.2 The Peer-Pressure Algorithm

The peer-pressure algorithm is described in (Robinson, 2011). The algorithm is shown in Figure 1. The following matrices and vectors are involved in the algorithm:

```
1   iteration = 1;
2   while not(Old_belonging == New_belonging)))
3      Old_belonging = New_belonging;
4      T = C * A; % compute the pull on each node
5      for j = 1:N  % for each node
7         col = T(:,j);
8         [m, k] = max(col); % find the max pull
9      end
10     C = zeros(N,N); % new clusters
11     for j = 1:N   % place each node
12        i = New_belonging(j); % in the clutser
13        C(i,j) = 1; % that gave max pull
14     end
15     iteration = iteration + 1;
16   end
```

Figure 1: Peer pressure algorithm (based on (Robinson, 2011))

- The network is represented by its weighted adjacency matrix A.
- C is the matrix representing the current cluster formations; each row vector C_i of C is a cluster. If the element $c_{ik} == 1$ means node k is in the cluster i. Note that C_i need not be unique, as there can be $C_j == C_i$ for $i \neq j$.
- Computing the pulling strength is given by $T = CA$, where t_{ij} is the pulling strength of cluster i on node j. After computing T, node j will move to cluster l as t_{lj} is the maximum on the j-th column of T.
- *New_belonging* and *Old_belonging* are vectors in which $v_i == m$ means node i belongs to cluster m. *New_belonging* represents the cluster membership computed in the current iteration, and *Old_belonging* is from the previous iteration.

The algorithm starts with an initialization step (not shown in Figure 1) in which each node becomes an individual cluster (each node is in a cluster by itself). Thus, during the initialization $C == I$ (the identity matrix) as the node number and the cluster number will be equal.

After the initial step starts the iterations, as depicted in the lines 1-16 in Figure 1. The iterations terminate (line 2) when there is no change in the cluster composition (clusters are the same in two successive iterations).

During an iteration, the pulling strength experienced by each node is computed (line 4). In the lines 5-9, the maximum pull for each is node is computed. Finally, each node moves to the cluster that exerted the maximum pull (lines 10-16).

There are some enhancements to the peer-pressure algorithm. Some of these enhancements are democratic voting (all the nodes possess only one vote immaterial of their out-degrees) and Anti-bullying (just like individual nodes, all clusters possess the same pulling strength immaterial of their membership size). Another enhancement is favoring smaller cluster if a node is pulled by several clusters all with the same strength. For more details, the interested reader is referred to (Robinson, 2011).

3.3 Running Time and Space Complexity

Running Time: Peer pressure algorithm is not guaranteed to converge (Robinson, 2011). This means the loops have to be terminated after some iterations (usually equal to the number of nodes N). During an iteration, there are three code blocks that dominate the computations: The matrix multiplication $T = CA$ with running time $O(N^3)$, and two for-loops with running time of $O(N)$. Thus the running time for a single iteration will be $O(N^3)$. Considering $O(N)$ iterations, the running time of the algorithm becomes $O(N^4)$. Space complexity: the algorithm maintains two vectors (*Old_belonging* and *New_belonging*) of length N and three matrices (A, C, and T) of dimension $R^{N \times N}$. Thus, the space requirement is low, and hence the memory utilization is efficient.

4 Extension to Peer pressure algorithm

In this section, we propose a simple yet powerful extension to the peer pressure algorithm. The extension is to allow the user to determine the sizes of the clusters. For example, a user may prefer a large number of small clusters or a small number of large clusters. The proposed extension allows the user to vary the size and

the number of clusters, through the introduction of the parameter called "self-loop strength".

4.1 Parameter "Self-loop Strength"

The peer pressure algorithm adds self-loop to each node to make the algorithm converge; self-loop (*aka* self-edge) means there is an arc emerging from each node to itself. The addition of self-loops is done by making the diagonal elements of the incidence matrix A become one. $A = A + I$.

As an extension, the self-loop strength (represented by β) of the nodes need not be one but can be a real value: $\beta \in R^+$. Thus, the extension becomes:

$$A = A + (\beta \times I). \tag{1}$$

4.2 The influence of "Self-Loop"

It is a fundamental assumption in the peer-pressure algorithms that each node can exert a total pulling pressure of singleton (1). For example, a node P that has only one outgoing arc and a node Q that has ten outgoing arcs would possess the same total pulling pressure of 1. However, the sole outgoing arc of P will take the total pressure of 1, whereas, in Q, every ten outgoing arcs (assuming that all the ten arcs have a unit arc weight) have to share the total pulling pressure Q; thus each outgoing arc of Q will exert a pressure of $1/10 = 0.1$.

Let us consider the nodes X and Y as shown in Figure 2. Let us assume that the nodes have outdegree (number of outgoing arcs) of m and n, respectively.

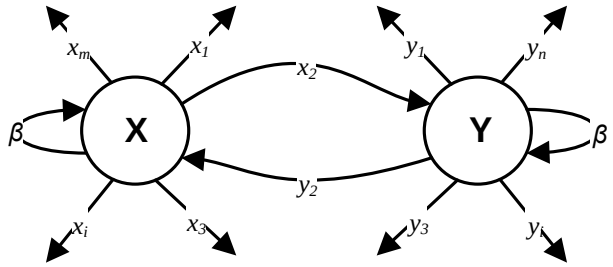

Figure 2: Peer-pressure dynamics between two nodes

Since X has m number of outgoing arcs, each with arc weight of a_i, the total arc weights is equal to $\sum_{i=1}^{m} a_i + \beta$. The share of pulling strength of arc x_i of X (except self-loop) is $a_i/(\sum_{i=1}^{m} a_i + \beta)$. Whereas, the share of pulling strength of self-loop of X is $\beta/(\sum_{i=1}^{m} a_i + \beta)$. Similarly, the share of pulling strength of arc y_j of Y (except the self-loop) is $a_j/(\sum_{j=1}^{n} a_j + \beta)$, and for the self-loop of Y it is $\beta/(\sum_{j=1}^{n} a_j + \beta)$.

Considering the peer-pressure dynamics, X will resist the pulling pressure from Y, as long as the strength of remaining in the current state (in other words, the self-loop strength) is greater than the pulling pressure from Y. Thus, X will remain in its current cluster (not join with Y) as long as:

$$\beta / \left(\sum_{i=1}^{m} a_i + \beta \right) \geq a_{yx} / \left(\sum_{j=1}^{n} a_j + \beta \right), \tag{2}$$

where a_{yx} is the arc weight of the outgoing arc y_2 of Y towards X.

Otherwise, X will succumb to the pulling pressure from Y and will join Ys cluster.

Generally, by assigning a larger value to self-loop strength, say $\beta = 2.0$, the nodes will become more independent and will resist pull from the neighboring cluster. In the worst case, all nodes can remain as stand-alone clusters (each node become a cluster on its own). Whereas, assigning a smaller value to self-loop strength, say $\beta = 0.5$, the nodes will become less-independent thus succumb to the pull from the neighboring cluster and will readily join it.

The side effect of assigning a smaller value to self-loop strength is that smaller the value the more unstable the iterations become. In other words, the number of iterations before the algorithm converges is inversely proportional to the self-loop strength (β).

5 GPenSIM Functions for Cluster Analysis

Table 1 summarizes the GPenSIM functions that are used for static cluster analysis in Petri Net models.

Table 1: GPenSIM functions for Network Cluster analysis

GPenSIM function	*Description*
clusterana	Finds the clusters of a network.
prnclusters	Prints the clusters on the screen.

The function clusterana takes one input parameter named self-loop strength; self-loop strength is a real-valued parameter, and the assigned value usually varies between 0 2. As discussed in the previous section, higher the value of self-loop strength, stronger (more independent) individual nodes become, thus larger number of clusters will result.

The function prnclusters gets the clusters found by clusterana and prints these clusters on the screen.

5.1 Other GPenSIM functions for network analysis

Table 2 summarizes the other GPenSIM functions that are useful for static network analysis (Davidrajuh, 2008). These functions measure network centrality of the elements, measuring how central (important) each element in the network.

Table 2: GPenSIM functions for cluster analysis

GPenSIM function	*Description*
degCentrality	Measure normalized degree centrality of all the nodes.
betCentrality	Measure normalized betweenness centrality of all the nodes.
cloCentrality	Measure normalized closeness centrality of all the nodes.
matrixD	Returns the incidence matrix of a Petri net.
sssp	Finding single-source shortest paths, using Dijkstras algorithm.
apsp	Finding all-pairs shortest paths, using Johnsons algorithm. algorithm.

6 Application Example

A simple Flexible Manufacturing System (FMS) is given in this section as the application example. This example is taken from the authors earlier work (Davidrajuh et al., 2018a).

The example is shown in the Figure 3 is to make only one type of product. In this FMS:

- The input raw material of type-1 arrives on the conveyor belt C1. Robot R1 picks the raw material type-1 and places into the machine M1. Similarly, robot R2 picks the raw material from conveyor belt C2 and places into the machine M2.
- Machine M1 makes the part P1, and M2 makes the part P2. When the parts are made by the machine M1 and M2, they are placed on the assembly station by the robots R1 and R2, respectively.
- An assembly station AS to join the two parts P1 and P2 together to form the product. The robot R2 does the part assembly at AS.
- Robot R3 picks the product from the assembly station and places it on the painting (and polishing) station PS. Robot R3 performs the painting and puts the completed product into the output buffer (cartridge) OB.

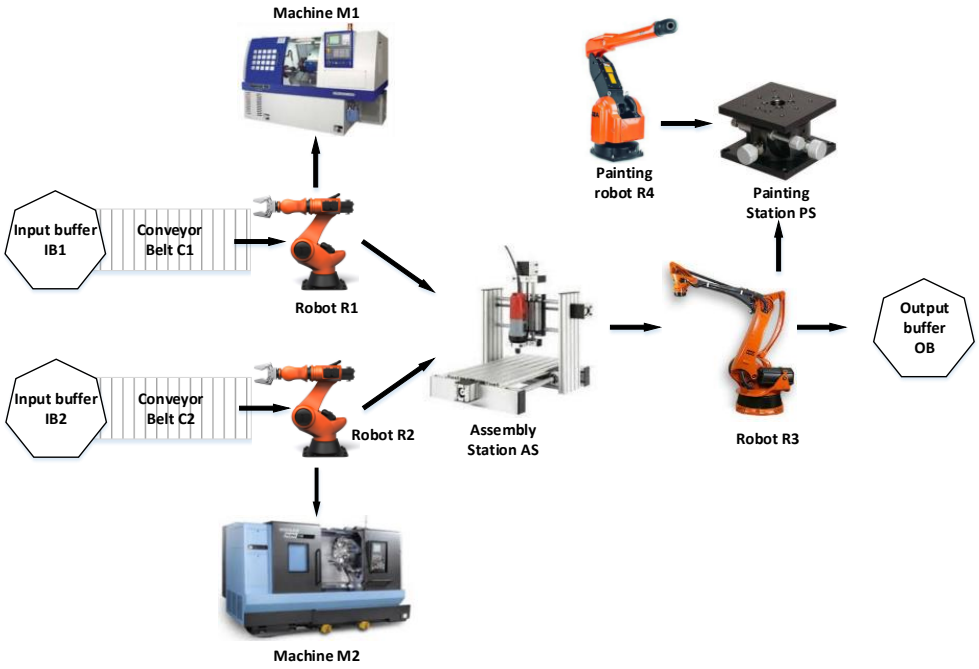

Figure 3: Flexible manufacturing System (Davidrajuh et al., 2018a)

The following activities explain the FMS operations (t stand for transition):

- tC1: conveyor belt C1 brings the input material type-1 into the FMS.
- tC2: conveyor belt C2 brings the input material type-2 into the FMS.
- tC1M1: robot R1 moves raw material from conveyor belt C1 to M1.
- tC2M2: robot R2 moves raw material from conveyor belt C2 to M2.
- tM1: machining of Part-1 at machine M1.
- tM2: machining of Part-2 at machine M2.
- tMA: robot R1 moves part-1 from M1 and R2 moves part-2 from M2 into Assembly Station AS.
- tAS: robot R2 assembles parts P1 and P2 together at the assembly station AS.
- tAP: robot R3 picks the product from the assembly station and places on the painting station PS.
- tPS: robot R3 performs painting and surface polishing on the product. When the job is finished, R3 places the product into the output buffer OB.

6.1 The Petri Net model

The Petri net model of the FMS is given in Figure 4. The Petri net model is obtained by serially connecting the activities listed in the preceding subsection. In Figure 4, the passive places are represented by circles. The black spot with the circle (e.g., pR1) represents the availability of the resource (robot R1). The active transitions are represented by the rectangular boxes. The number shown inside the rectangular box is the firing time (machining time) of the transition. Finally, the most important transitions (machining in M1 and M2, assembly, and painting) are shown in shaded (grey) rectangles, and the utility transitions (involving robots and conveyor belts) are given as plain rectangles.

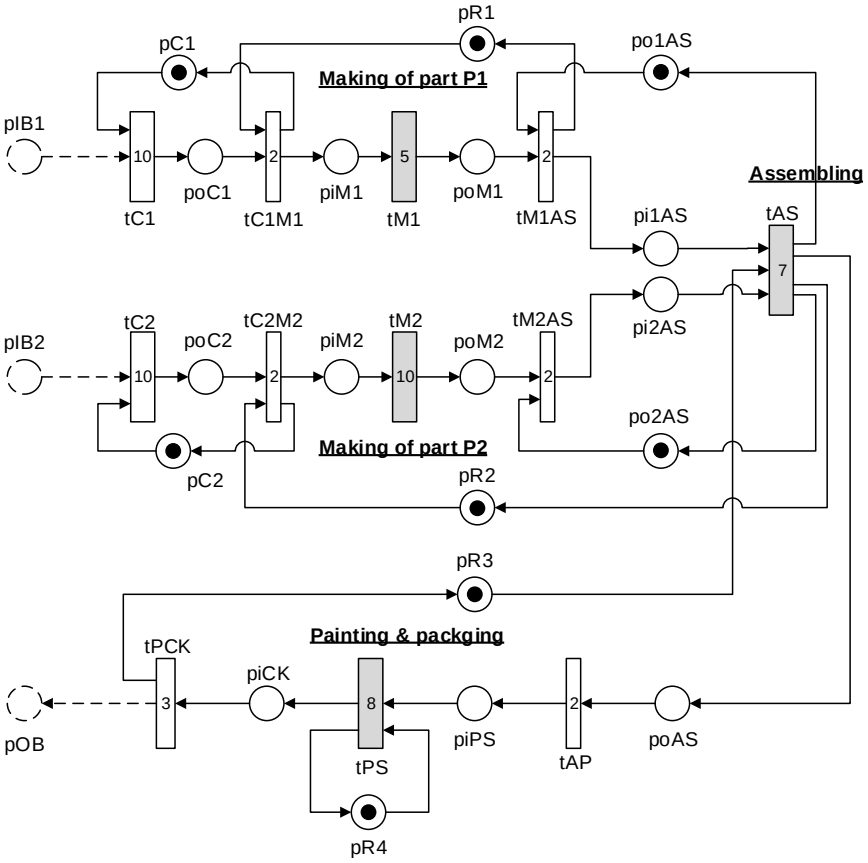

Figure 4: Petri Net Model of the FMS

6.2 GPenSIM Implementation

For finding the clusters in the Petri net model with GPenSIM, two M-files are needed: 1) the Main Simulation File (MSF), and 2) the Petri net Definition File (PDF). In the MSF (shown below), first, the static Petri net structure is created from the declarations given in the PDF. Then, the function clusterana is called for finding the clusters and then the function prnclusters is called for printing the clusters found.

```
% MSF: this is the main simulation file
% firstly, create Petri net structure
% from the PDF file 'nc_pdf.m'
pns = pnstruct('nc_pdf');

% analyze the static Petri net
% parameter beta (self-loop strength)
beta = 0.8; % 0 <= beta <= 2
clusterana(beta); % find the clusters
prnclusters(); % print the clusters
```

Due to brevity, the PDF file and the two functions (clusterana and prnclusters) are not shown in this paper. The interested reader is encouraged to visit the webpage (Complete Code for the example) to download the code and experiment with it. The software GPenSIM can be downloaded from the website (GPenSIM: A General Purpose Petri Net Simulator).

7 Simulation Results

The Table 3 and Figures 5-6 given below show how the number of clusters varies with the value of self-loop strength (β). Table 3 shows the number of resulting clusters and the number of iterations taken when β was varied from 2 to 0.

Table 3: Varying the value of Self-Loop Strength

Self-loop Strength (β)	Number of resulting clusters	Number of iterations taken to converge
2	34	2
1.5 1.9	33	3
1.1 1.4	28	3
1	21	5
0.7 0.9	16	5
0 0.6	-	(did not converge)

Initially, every element in the Petri net is an individual cluster on its own. Thus, there were 34 clusters in

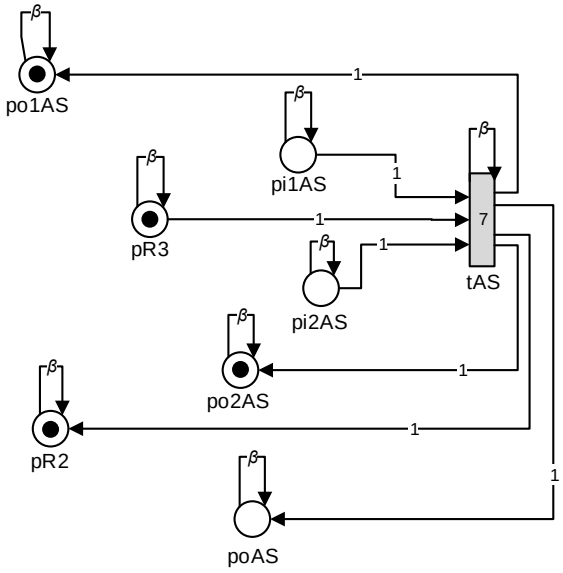

Figure 5: Peer-pressure dynamics between two nodes

the beginning. Also, β was given the value of 2.0. This value was strong enough for the individual elements to hold on to their own cluster.

For example, focusing on the element tAS, it has five outgoing arcs (outdegree is equal to 5), see Figure 5. One of the outgoing arcs is to itself (the self-loop) with the strength of β. The other four outgoing arcs (to poAS, po1AS, po2AS, and pR2) are with unit arc weights of one. Thus, for tAS, the strength of holding on to its own cluster is $\beta/(\beta+4)$. tAS is also pulled by three nodes such as pi1AS, pi2AS, and pR3. The pulling strength of these three nodes is equal to $1/(\beta+1)$ per arc. When $\beta = 2$, the strength of tAS to hold on to its own cluster ($= 2/(2+4) = 1/3$) is equal to the pulls from the other elements (e.g., the strength of the pull from pR3 $= 1/(2+1) = 1/3$), thus tAS remain in its cluster.

However, when β became 1.9, the pull experienced

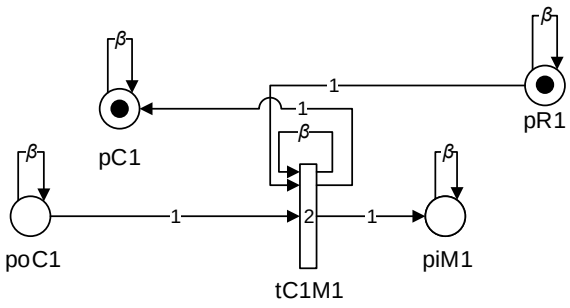

Figure 6: Peer-pressure dynamics between two nodes

Table 4: The 16 Clusters when self-loop strength = 0.7 0.9 (standalone clusters are not shown)

Cluster No.	Elements of the cluster
1	tC1, tC1M1, pC1, poC1
2	tC2, tC2M2, pC2, poC2
3	tAS, pR3
4	tPS, pR4
5	tPCK, piCK
6	tM1, piM1, poM1
7	tM2, piM2, poM2
8	tM1AS, po1AS
9	tM2AS, pi2AS, po2AS
10	tAP, piPS, poAS

by tAS from pR3 ($= 1/2.9 = 0.34$) becomes larger than the self-loop strength ($= 1.9/5.9 = 0.32$). Thus, tAS joined with pR3. Thereby, the number of clusters goes from 34 to 33. The formation of 33 clusters remained intact as β was changed from 1.9 to 1.5.

When β was reduced to 1.4, more of the elements were pulled into clusters. Thus, the number of clusters drops further from 33 to 28. For example, tC1M1 and poC1 remained as independent clusters when β was in the region 2 1.5. When β became 1.4, tC1M1 is pulled by poC1 to become one cluster. This is because, poC1 has an outdegree of two and the pulling strength of the arc towards tC1M1 is equal to $1/(\beta+1)$, see Figure 6. Whereas tC1M1 has an outdegree of three, and its self-loop strength is equal to $\beta/(\beta+2)$. tC1M1 will resist the pull from poC1 as long as:

$$\text{tC1M1 self-loop strength} \geq \text{poC1pulling strength}$$
$$\beta(\beta+2) \geq 1/(\beta+1)$$
$$\beta \geq \sqrt{2}$$

Hence, when β became 1.4, tC1M1 joined poC1.

Finally, when β was in the region 0.9 to 0.7, there were 16 clusters formed. When β was reduced further, the iterations did not converge. Thus, the Petri net shown in Figure 4 can be composed of 34 clusters at most (every element is in its own cluster) when $\beta = 2$, and 16 clusters at least when $\beta = 0.70.9$. Figure 7

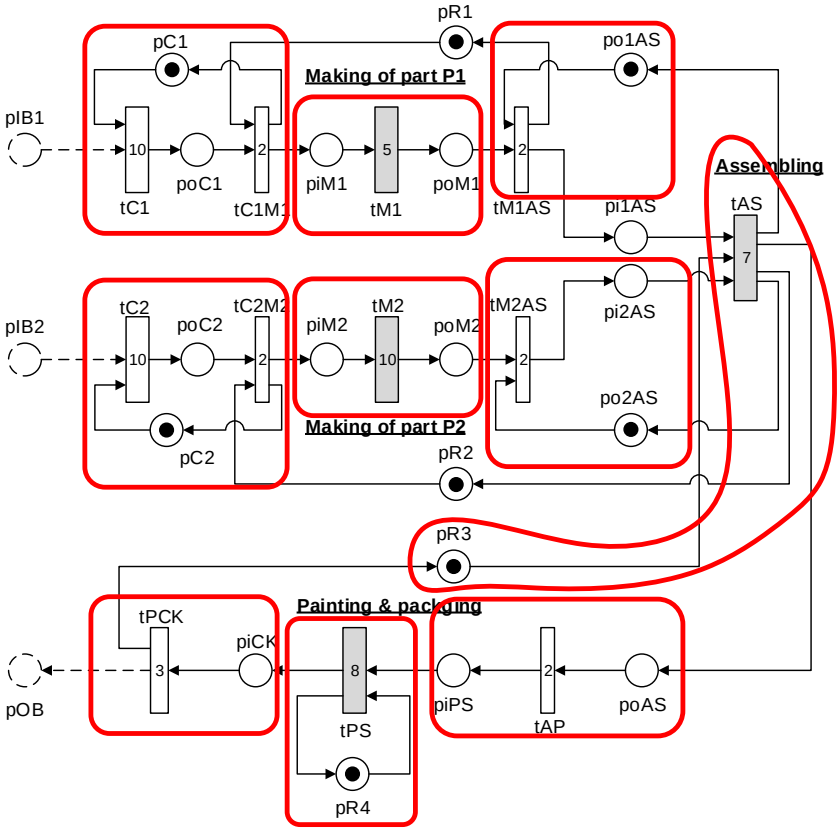

Figure 7: Petri Net Model of the FMS

and Table 4 show the 10 clusters of the Petri net that possess more than a single element; the six standalone clusters (clusters possessing only one element) are not shown.

8 Discussion

This section discusses the implication of the simulation result shown in the previous section, in other words why finding clusters in a manufacturing network is important. Cluster creation is widely used in many aspects related to manufacturing. Clusters are used at various levels, from production cells to virtual production networks (*aka* clusters), which is especially popular among small and medium-sized enterprises (SMEs). Clusters are central to lean manufacturing and just-in-time (JIT) production concepts. The cellular manufacturing is an application of principles of group technology in manufacturing, both in medium variety and volume mix of production. It concerns the processing of a group of similar parts (part families) on dedicated clusters of machines or manufacturing processes (manufacturing cells) (Kioon et al., 2009; Bi et al., 2008). Compared to the other manufacturing systems, cellular systems allow producing a higher variety of customized products together with the simplification of material flow and material handling, reduction of work in progress and setup times. Research on cellular systems paves further development of highly flexible, reconfigured production systems, especially in the context of changes that are associated with the emergence of Industry 4.0 (Hermann et al., 2016; Krenczyk et al., 2018).

The networked manufacturing systems described in the literature refer to two different levels - the clusters that group specific production resources and devices, as well as to bring together manufacturers belonging to different SMEs into producers' networks. In this context, multi-scale methods, in which it would be possible to configure and plan the production flow for individual levels, starting from larger clusters, gradually going to a division with fewer elements. Multi-scale modeling means determining the characteristics or behavior of the system on one level by using information or a model from a different level (Neumann et al., 2012).

In this paper, the authors propose a method for configuring Petri Net models of production systems into a variable number of clusters, which leads to the use of multi-scale modeling in the production planning and scheduling process for reconfigured production systems.

9 Conclusion

This paper presents the realization of a method for finding clusters in networks. The method is newly added as auxiliary functions to the software GPenSIM. GPenSIM is a software for modeling, simulation, performance analysis, and control of discrete-event systems. With these new functions, it is now possible to perform cluster analysis of a Petri net, in addition to the (usual) dynamic analysis.

For finding clusters, this paper presents an extension of the peer-pressure method. The extension is the introduction of the parameter Self-loop Strength with which the user can tune the number of clusters in the network.

Though this paper highlights manufacturing systems, the theory, idea, and the GPenSIM functions that are presented in this paper apply to other engineering systems as well.

References

Abbaszadeh, A., Abedi, M., and Doustmohammadi, A. General stochastic petri net approach for the estimation of power system restoration duration. *International Transactions on Electrical Energy Systems*, 2018. 28(6):e2550. doi:10.1002/etep.2550.

Bi, Z. M., Lang, S. Y. T., Shen, W., and Wang, L. Reconfigurable manufacturing systems: the state of the art. *International Journal of Production Research*, 2008. 46(4):967–992. doi:10.1080/00207540600905646.

Cameron, A., Stumptner, M., Nandagopal, N., Mayer, W., and Mansell, T. Rule-based peer-to-peer framework for decentralised real-time service oriented architectures. *Science of Computer Programming*, 2015. 97:202 – 234. doi:10.1016/j.scico.2014.06.005.

Complete Code for the example. 2019. URL `http://www.davidrajuh.net/gpensim/Pub/2019/MIC/`. Accessed on 07 January 2019.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009.

Davidrajuh, R. Developing a new petri net tool for simulation of discrete event systems. In *2008 Second Asia International Conference on Modelling Simulation (AMS)*. pages 861–866, 2008. doi:10.1109/AMS.2008.13.

Davidrajuh, R. *Modeling Discrete-Event Systems with GPenSIM*. Springer International Publishing, Cham, 2018. doi:10.1007/978-3-319-73102-5.

Davidrajuh, R., Skolud, B., and Krenczyk, D. Gpensim for performance evaluation of event graphs. In *Advances in Manufacturing*, number 201519 in Lecture Notes in Mechanical Engineering. Springer International Publishing, Cham, pages 289–299, 2018a. doi:10.1007/978-3-319-68619-6_28.

Davidrajuh, R., Skolud, B., and Krenczyk, D. Performance evaluation of discrete event systems with gpensim. *Computers*, 2018b. 7(1):8. doi:10.3390/computers7010008.

van Dongen, S. *Graph Clustering by Flow Simulation*. Ph.D. thesis, University of Utrecht, 2000.

Gilbert, J. R., Reinhardt, S., and Shah, V. B. High-performance graph algorithms from parallel sparse matrices. In *Proceedings of the 8th International Conference on Applied Parallel Computing: State of the Art in Scientific Computing*, PARA'06. Springer-Verlag, pages 260–269, 2007. doi:10.1007/978-3-540-75755-9_32.

GPenSIM: A General Purpose Petri Net Simulator. 2019. URL `http://www.davidrajuh.net/gpensim`. Accessed on 07 January 2019.

Hermann, M., Pentek, T., and Otto, B. Design principles for industrie 4.0 scenarios. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, volume 2016-March. pages 3928–3937, 2016. doi:10.1109/HICSS.2016.488.

Jyothi, S. D. Scheduling flexible manufacturing system using petri-nets and genetic algorithm. *Department of Aerospace Engineering, Indian Institute of Space Science and Technology: Thiruvananthapuram, India*, 2012.

Kepner, J. and Gilbert, J. *Graph algorithms in the language of linear algebra*. SIAM, 2011. doi:10.1137/1.9780898719918.

Kioon, S. A., Bulgak, A. A., and Bektas, T. Integrated cellular manufacturing systems design with production planning and dynamic system reconfiguration. *European Journal of Operational Research*, 2009. 192(2):414 – 428. doi:10.1016/j.ejor.2007.09.023.

Krenczyk, D., Skolud, B., and Herok, A. A heuristic and simulation hybrid approach for mixed and multi model assembly line balancing. In *Intelligent Systems in Production Engineering and Maintenance – ISPEM 2017*, volume 637 of *Advances in Intelligent Systems and Computing*. Springer International Publishing, Cham, pages 99–108, 2018. doi:10.1007/978-3-319-64465-3_10.

Křivánek, M. and Morávek, J. Np-hard problems in hierarchical-tree clustering. *Acta Informatica*, 1986. 23(3):311–323. doi:10.1007/BF00289116.

Mutarraf, U., Barkaoui, K., Li, Z., Wu, N., and Qu, T. Transformation of business process model and notation models onto petri nets and their analysis. *Advances in Mechanical Engineering*, 2018. 10(12):1687814018808170. doi:10.1177/1687814018808170.

Neumann, M., Constantinescu, C., and Westkmper, E. Method for multi-scale modeling and simulation of assembly systems. *Procedia CIRP*, 2012. 3:406 – 411. doi:10.1016/j.procir.2012.07.070.

Peterson, J. L. *Petri net theory and the modeling of systems*. Prentice Hall PTR, 1981.

Robinson, E. 6. complex graph algorithms. In J. Kepner and J. Gilbert, editors, *Graph Algorithms in the Language of Linear Algebra*, pages 59–84. Society for Industrial and Applied Mathematics, 2011. doi:10.1137/1.9780898719918.ch6.