# PROCESS MINING

## Lab Session 6

## Hands-on Multi-perspective Discovery

**Content**

## Laboratory Session

We will learn dotted chart, performance analysis including time between transitions, discovery of routing probabilities, decision guard mining and social network analysis including hands over of work and similarity of profiles.

## 1. Dotted Chart

We use the same log in the file, `2016.xes.gz`, as in the laboratory session on conformance checking. In particular, we focus on the *Offer sub-process*. After filtering and visualizing the filtered event log, the statistics should be as follows.


Figure 1. Selecting visualization type as dotted chart



Change the visualization to the *Dotted Chart (LogProjection)* view from the drop-down list highlighted in Figure 1. Then you obtain Figure 2.

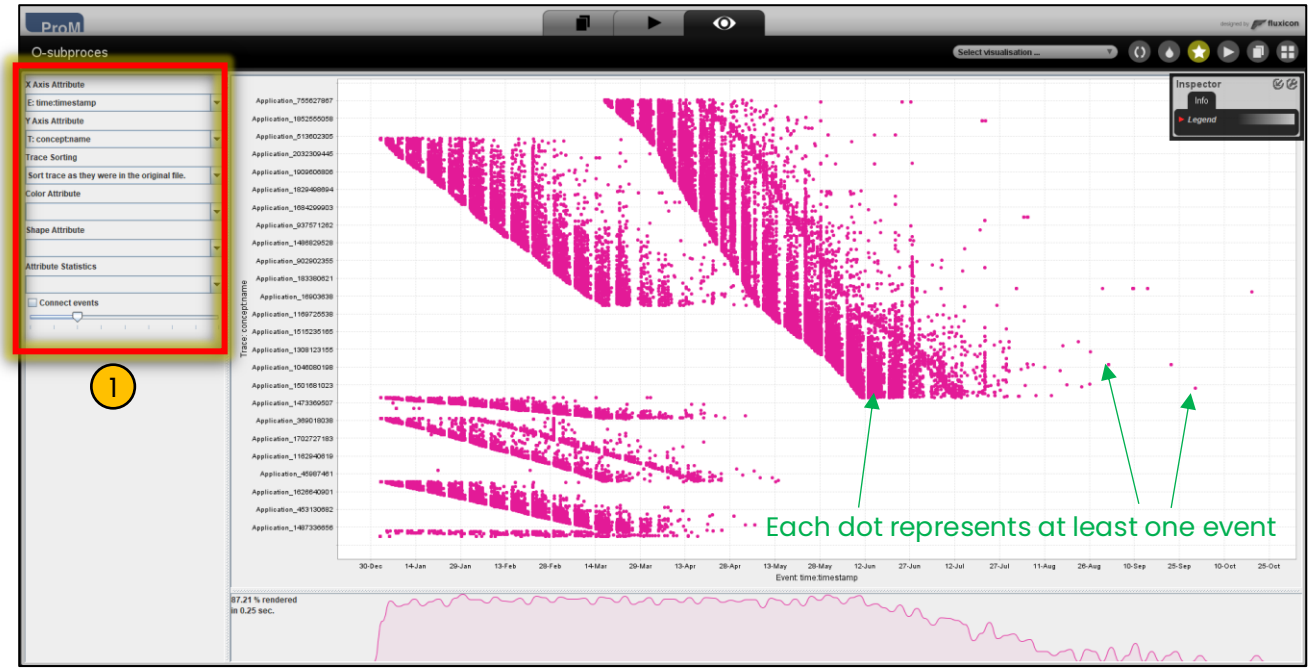



Figure 2. Dotted chart (log projection)

In the dotted chart, each "*dot*" corresponds to a different event of the log. When some dots are located at the same position, they are aggregated into one single dot. When passing with the mouse over one dot, one can see the list of events aggregated to the dot. Clearly, if the dot is not the result of any aggregation, the list is composed of one element. To keep the visualization neat, if more than ten dots are aggregated, only ten (random) events are visualized.

By default, the x-axis attribute is the time at which an event occurs (if the timestamp is available in the log), and the y-axis is the names of the trace, which are sorted as they are in the original file. The trace names are not helpful here as the axis to discover the relevant pattern.

In situations like this, the dotted-chart axes can be configured, as well as the colours and the axis sorting through the options in the left-hand side panel (in *Area* ①). The X- and Y-axis can be customized according to the value of several attributes in the log. Both axes can include some attributes related to *traces (**T**)*, *events (**E**)*, and *classifiers (**C**)*.

**(T)race attributes** are defined on the level of traces. Only those attributes are shown that are declared "global" or have a value for each trace in the event log.

**(E)vent attributes** are defined on the level of events. Only those attributes are shown that have a value for each event in the event log. If the time extension is used in the log, then also new attributes are calculated which relate to relative time, i.e., for each event, the time since the case started is recorded, as well as the time since the day or week started.

**(C)lassifier attributes** refer to event classes specified in the log. They are combinations of event attributes that determine an event class (often, the event name is defined as a combination of the *concept:name* of an event and the *lifecycle:transition* of that same event).



The following are the option present in the configuration panel on the left-hand side:

The **X-axis attribute** changes the attribute for the X-axis. Suppose the trace/event attribute is a continuous variable (e.g., integer). In that case, the axis will be set to the range of [min, max] where min and max are, respectively, the smallest and largest value observed for the chosen attribute. The values are sorted alphabetically if the attribute is a discrete variable (e.g., string). Exceptionally, the days of the week are sorted from Sunday to Saturday.

The **Y-axis attribute** changes the attribute for the y-axis. The considerations for the X-axis are also valid for the Y-axis.

**Trace Sorting** determines the sorting of traces. It only has an effect if the attribute "*T: concept:name*" is used for one of the axes. As mentioned, the default is that the events are sorted according to the order in the log file. Typically, the original file has events ordered by timestamps, which is what one would expect; however, this is not an imposed constraint (although the algorithms for discovery and conformance checking will typically assume it – see the lecture and instruction of next week).

**Color Attributes** specify which attribute determines the color. All dots are the same color by default, as no attribute is selected. The attribute selected should not have more than 1024 values. If an attribute is selected with more values, a message appears explaining this.

**Shape Attribute** specifies which attribute determines the shape of each dot. A*ll dots are circles by default, as no attribute is selected.* The attribute selected should not have more than eight values. If an attribute is selected with more values, a message appears explaining this.

**Attribute Statistics** enables to visualization of the statistics of a chosen attribute if the choice falls on a continuous attribute. The statistics consist of the average, the standard deviations, the minimum/maximum value, and the number of values observed.

**Connect Events** option connects the events referring to the same traces connected through a line.

**Dot-size slider** is used to configure the size of the dots.

Various visuals can be produced by changing the axis settings and colors. As it is generally impossible to draw all events (often, there are more events than pixels on the screen), the tool automatically adjusts the actual number of events rendered. The tool also considers the computer's performance when choosing which dots to show to allow for a pleasant user experience.

Changing the y-axis to *T:time:startTime*, dots are located on the y-axis according to the timestamp of the first event in the trace. We can also colour the dots according to the event's concept name (*E:concept:name*) to obtain Figure 3.



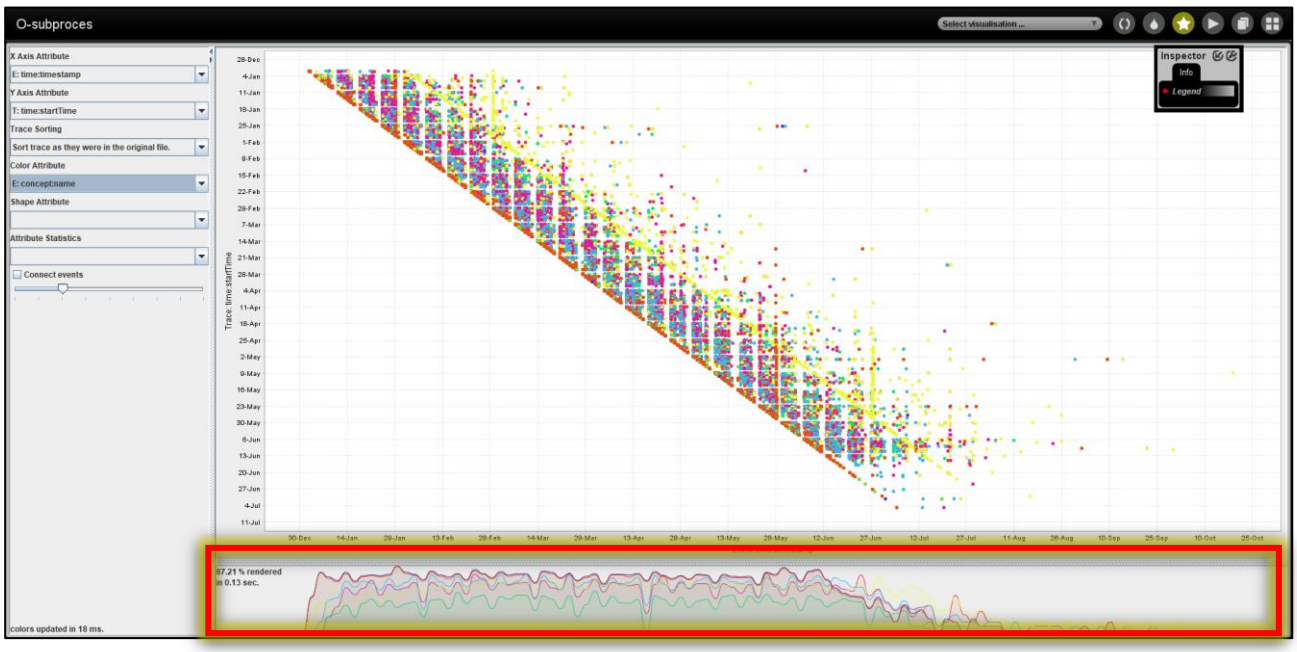



Figure 3. Dotted chart with event timestamp (x-axis) and trace start name (y-axis)

*Area* ① in Figure 3, in any configuration, shows the distribution of events over the x-axis. All the events associated with dots with the same value for the x coordinate are summed up and visualized in the distribution below.

Cases are ordered by the timestamp of the first event in Figure 4. Because the first dots in each row correspond to the first event in the log, we can derive the arrival rate by looking at the timestamp of the first dot of each row. We see that Offer sub-processes are given out at constant rates. The above figure also shows a line of yellow dots parallel to the hypothetical line of case arrival (see the highlighted area). It refers to offers that are automatically cancelled after 30 days. This can be better seen this if the x-axis is changed to refer to the relative time with respect to the case start instead of the absolute time (see Figure 5).

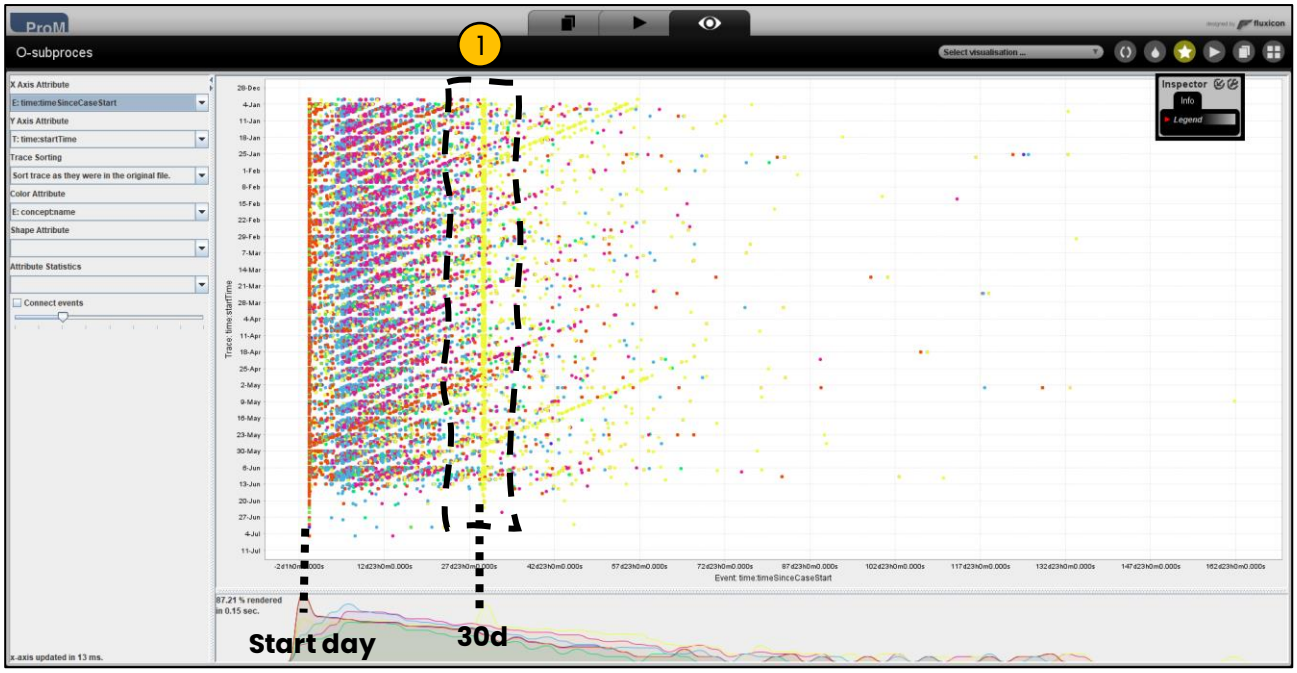



Figure 4. Dotted chart with case start time (x-axis) and trace start time (y-axis)

Compare *Area* ① in Figure 4 (the line of yellow dots) and Figure 5.

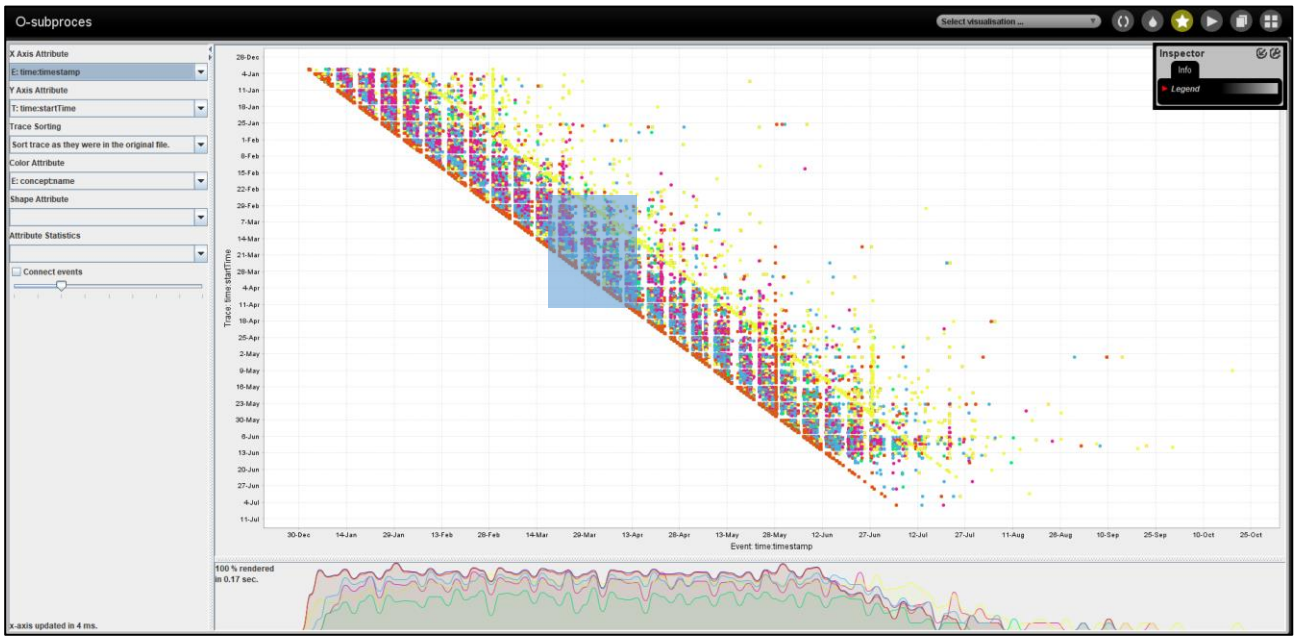



Figure 5. Dotted chart with event timestamp (x-axis) and trace start time (y-axis)

Figure 6 presents events over the day of the week. We can observe a few events on Sundays, which makes sense given that the financial institute is closed on those days. On Sundays, only activities (*O_ACCEPTED*, *O_CANCELLED*, *O_DECLINED*) related to ending the process are performed. Offers are only accepted or declined by applicants through a website or automatically cancelled by the system.




Figure 6. Dotted chart with day of week (x-axis) and activity name (y-axis)

Other insights can also be observed. For instance, one can choose the x-axis to represent the amount of offers from the financial institute associated with the trace (attribute *T: Amount*). The

y-axis can be the timestamp of the first event of traces to which the event belongs (attribute *T: time:startTime)*—these yield Figure 7. No dots in the highlighted area demonstrates that the financial institute was not offering more than *50000 Euros* in the first 90 days of the year (until *February 15th*). Also, by observing the picks in the distribution at the bottom, one can observe that offers are typically in multiple of *5000 Euros* because the start activity shown in red (*O_SELECTED*) is performed amounts in *5000 Euros*. Offers are usually less frequent for larger amounts because the number of dots over *50000 Euros* is less than those below *50000 Euros*.

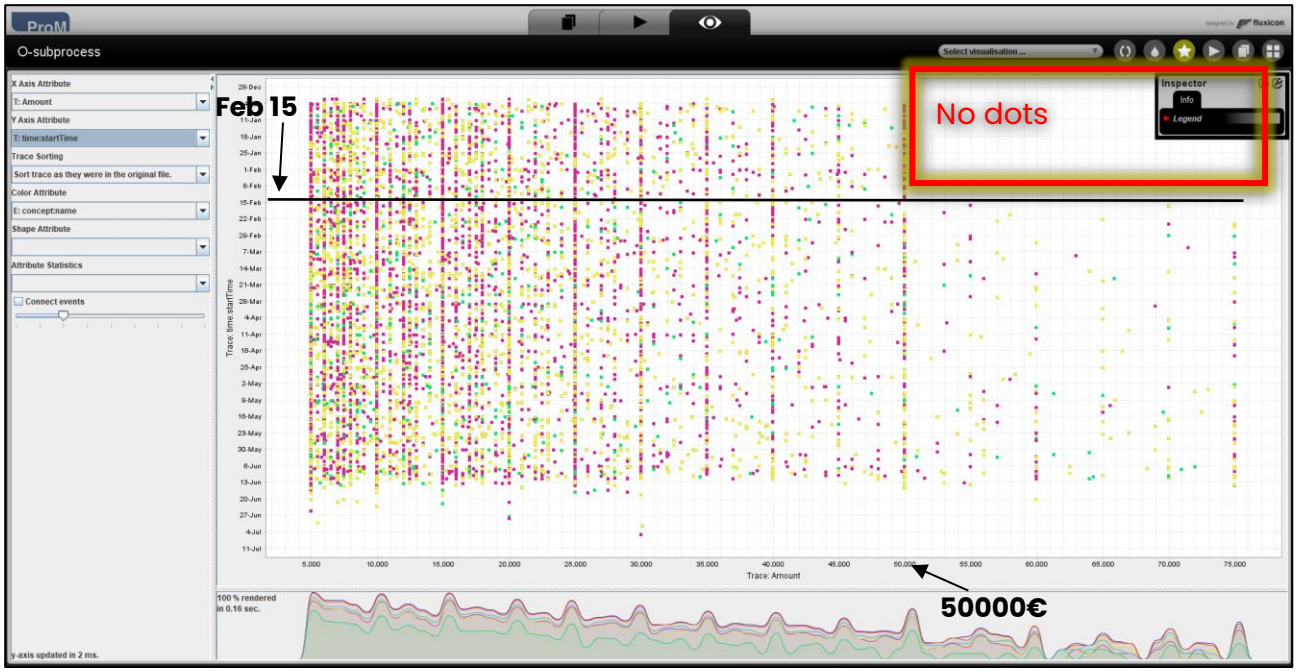



Figure 7. Dotted chart with amount (x-axis) and trace start time (y-axis)

## 2. Performance Analysis

We start from the alignment results given in Figure 8 that were already discussed in the laboratory session on Conformance Checking.

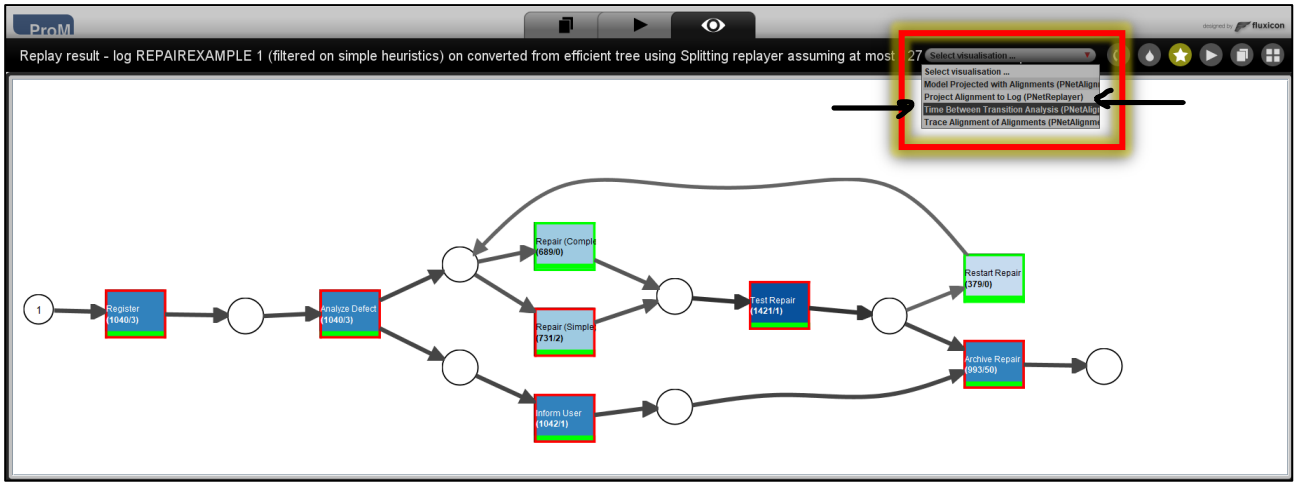



Figure 8. Alignment result created in conformance checking lab session

### Analysis of the time between transitions

Looking at alternative visualizations of the alignment in the combo box on the top highlighted in Figure 8, you observe visualization "*Time Between Transitions Analysis (PNetAlignmentAnalysis)*". It shows a matrix containing as many rows and columns as the model transitions. Given two transitions ***a*** and ***b***, the matrix element ***(a,b)*** indicates the time elapsed between the completion of transition ***a*** and the completion of transition ***b***. Not only does the matrix consider the direction succession, but also does the "eventual" succession: between ***a*** and ***b***. In other words, other activities can occur between ***a*** and ***b***.



When such visualization is chosen, we are confronted with the following question. The question is concerned with visualizing the invisible transitions as rows/columns of the matrix. Since there are no invisible transitions, we can answer either way. Then you see Figure 10.

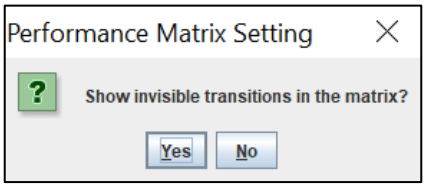



Figure 9. Performance matrix setting

By default, the values indicate the average elapsed time between each pair of transitions. When the execution of a transition/activity is never eventually followed by the execution of a certain transition/activity, the corresponding matrix's cell is black. For instance, in *Area ①*, activity *Repair (Complex)* is never eventually followed by *Repair (Simple)*; also, *Archive Repair* is never

eventually followed by any activity/transition, which is expected since it is the last activity of the process.

Analyzing the results, you can observe that *Repair (Simple)* takes 25.44 minutes and *Repair (Complex)* takes 40.33 minutes. It is the time between the activity *Analyzed Defect*, which precedes these two activities, and *Repair (Simple)* and *Repair (Complex)*, respectively.

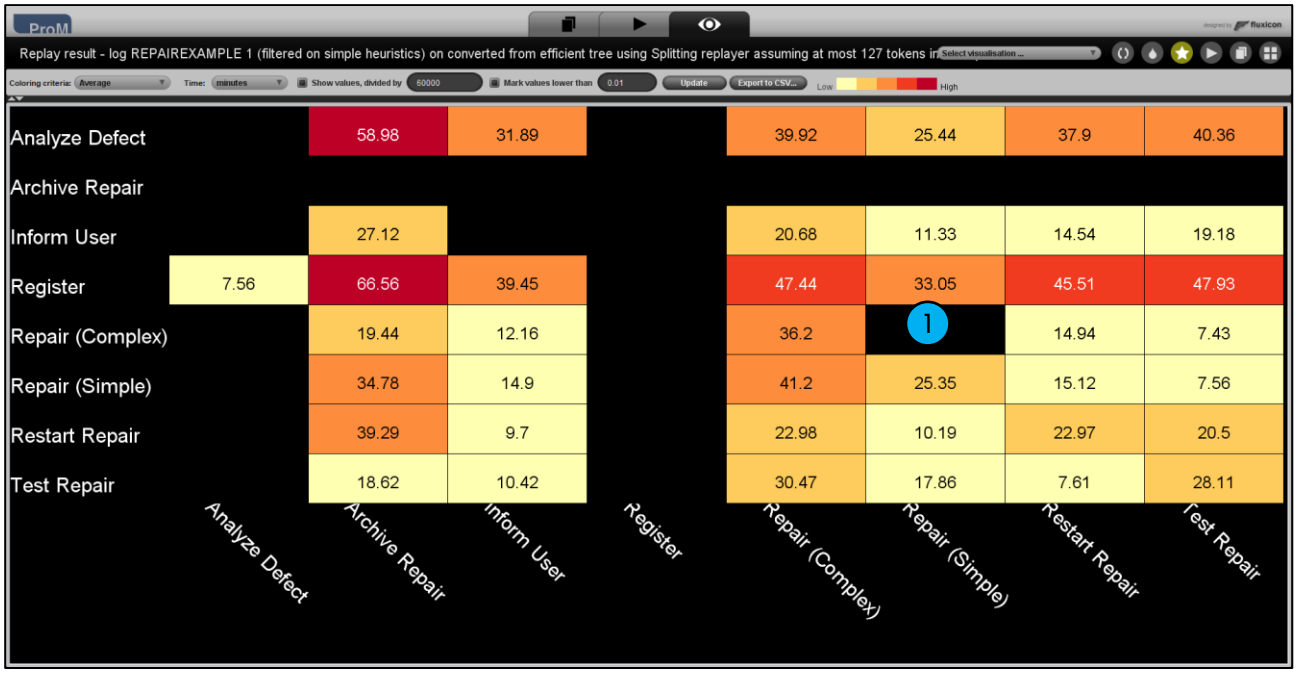





Figure 10. Time-based performance matrix

It is worth highlighting that the elapsed time between any two transitions, ***a*** and ***b***, only considers the first occurrence of ***b*** that follows ***a***. This means that if the trace is $\langle a, b, b \rangle$, the second occurrence of ***b*** is discarded in the computation. For instance, including timestamps ***t***, the log contains a single trace: $\langle a(\mathrm{t}=1), b(\mathrm{t}=2), b(\mathrm{t}=3) \rangle$, the average elapsed time is ***1***; it is *not 1.5*, which would be if we also considered the second b. Conversely, if the trace were $\langle a(\mathrm{t}=1), b(\mathrm{t}=2), \mathrm{a}(\mathrm{t}=3), \mathrm{b}(\mathrm{t}=5) \rangle$ the second ***b*** would be considered because the two occurrences of ***b*** are interleaved by one ***a***.

Alignments are computed before computing the elapsed time between each pair of activities. When a trace is not compliant with the process model, the alignment contains moves in the log or model. The plug-in only considers the activities involved in both moves (a.k.a. synchronous moves) while computing elapsed times to ensure the results' reliability,

## 3. Discovery of the routing probabilities

Here, we use the plug-in *Multi-perspective Process Explorer* (MPE). This plug-in integrates several techniques to discover different perspectives into a single interface. For the entire repertoire of offered techniques, you can refer to the following publication:

> *Felix Mannhardt, Massimiliano de Leoni, Hajo A. Reijers. The Multi-perspective Process Explorer. BPM (Demos) 2015: 130-134*

The plug-in *Multi-perspective Process Explorer* takes an event log and a model as input. We used the model already used above, along with event log ***repairExample2.xes***, and limited the functionalities to illustrate branching probabilities. You obtain Figure 11.

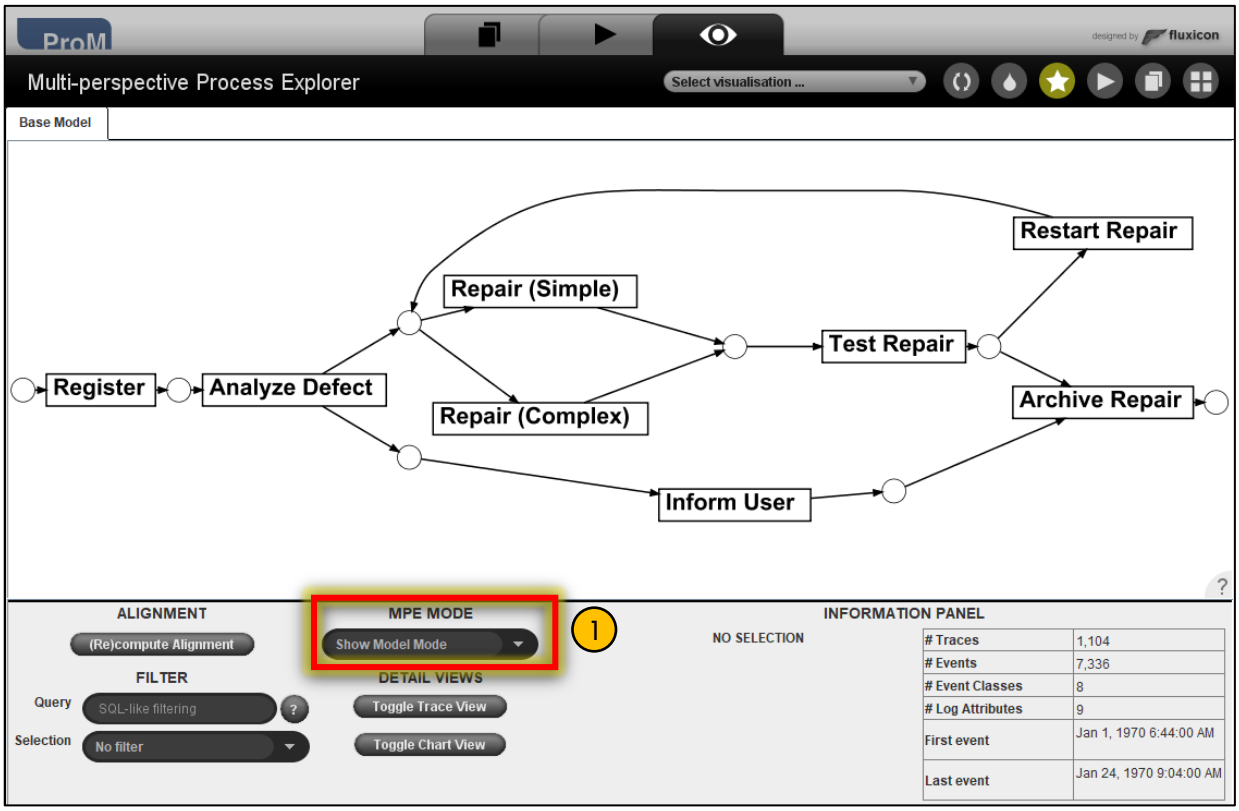



Figure 11. MPE Model Mode

The routing probabilities can be seen by changing the MPE MODE to "*Show Performance Mode*" from *Area* ① in Figure 11. Most of the modes require to compute alignments, including the performance mode. This plug-in has a simplified version of computing alignments, which tries to guess the activity-event mapping with the initial and final marking. It uses the cost function that assigns 3 to log moves and 2 to model moves. Figure 12 will be shown and click *Simple Configuration*.

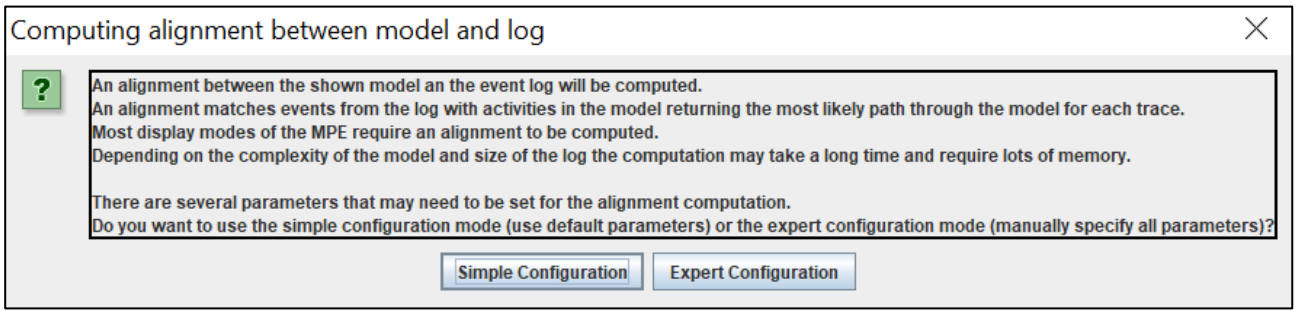



Figure 12. Configuration for performance mode

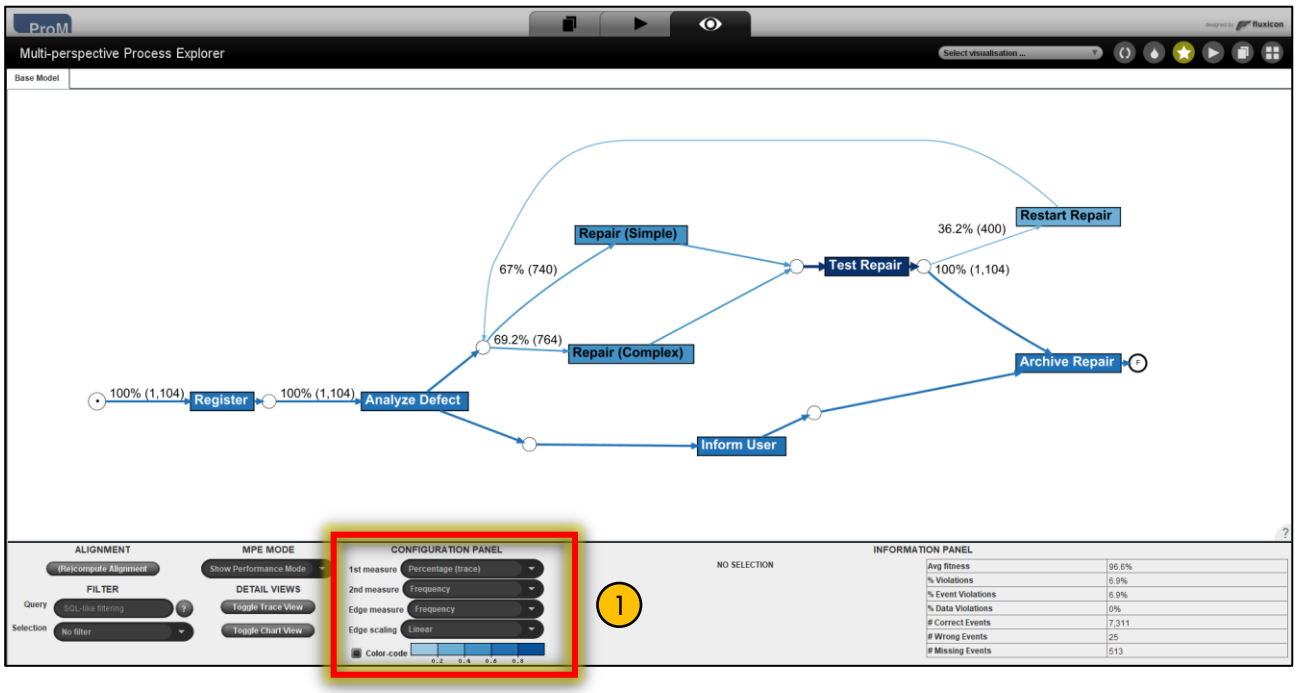



Figure 13. MPE Performance Mode by default values

The result contains the branching probabilities at the trace level. When probabilities are computed at the trace level, the probability of an arc between *place p* and *transition t* indicates the percentage of traces that traverse that arc, namely the percentage of traces in which *transition t* fires consuming a token from *place p*. In this example, 69.2% of traces perform *Repair (Complex)*, and 67.7% perform *Repair (Simple)*. Note that the percentage's sum is larger than 100% because the same trace can fire both in subsequent loop executions to restart repair. It should be obvious that *Archive Repair* is executed in 100% of the traces, given that it is the only possible last activity to be executed. The second value is the trace's frequency; namely, the number of traces traverse that arc.

*Area* ① in Figure 13 includes some configuration settings. *1st measure* is the value before the brackets, while *2nd measure* refers to the value within the brackets. *Edge measure* defines how edge thickness will be determined for a transition colour. The value shown above is not the actual routing probability of a place *p*, which is intended as the probability of traversing the arc when a token is present in place *p*. To compute this value, you need to change the measure to *Percentage (local)* in the *1st measure* like in Figure 14. To better explain, let us consider the place in *Area* ① in

in Figure 14. It is an *XOR split* between *Restart Repair* and *Archive Repair*. Figure 14 indicates that when a token is present in that place, it is consumed *26.6%* of the time by *Restart Repair*, while it is consumed *73.4%* of the time by *Archive Repair*. This is different for the percentage of traces in which the token is consumed by one or the other transition. Indeed, every execution will eventually execute *Archive Repair*.

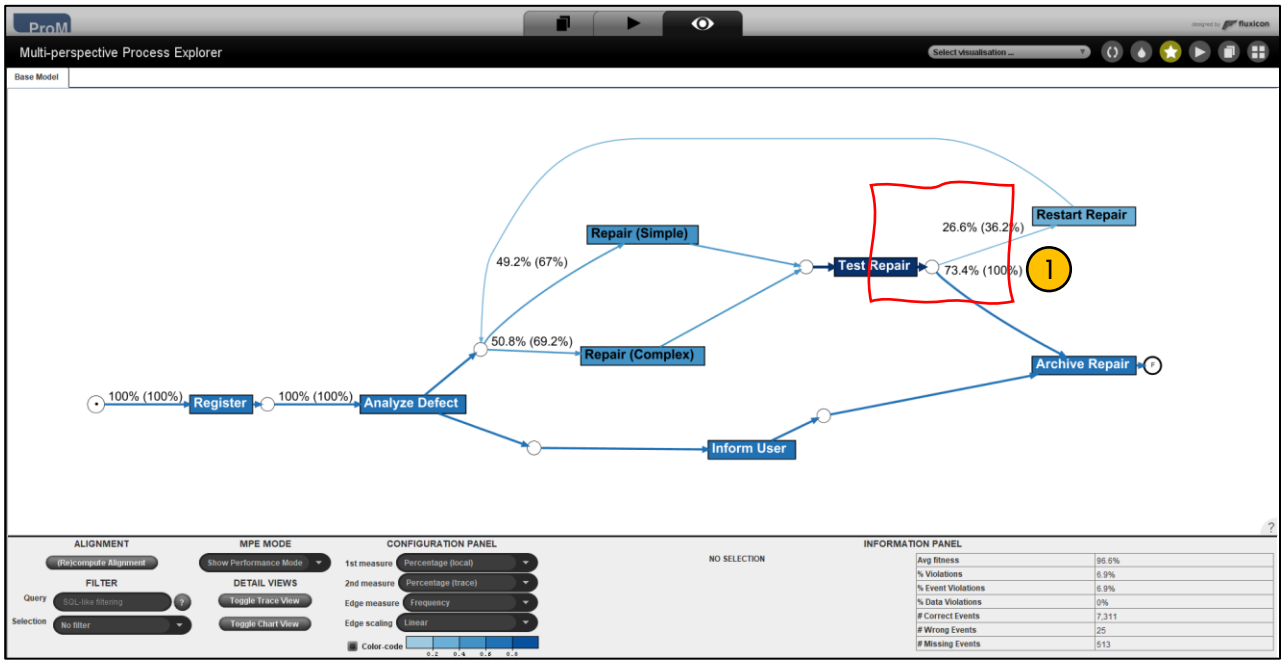



Figure 14. MPE Performance Mode by percentage (local) in the 1st measure

## 4. Decision Guard Mining

This part shows how ProM supports the discovery of the activity guards at decision points. ProM supports it through the decision miner, which is implemented as a plug-in "*Discovery of the Process Data-flow (Decision-Tree Miner)*", which requires an event log and a Petri-net model as input. We again use the repairing process model, created the event log filtered out by start event, along with the event log ***repairExample.xes***.

Similarly, to the conformance checker, the first configuration window is concerned with providing a mapping between activity names in the model and the event log. Again, since the model has been discovered from the event log, the activity names in the model coincide with those in the event log.

Initially, the plug-in internally computes an alignment of the process model and the event log. After the alignments are computed, the decision miner needs to be configured using the window given in Figure 15. For further details of the meaning of the different parameters, the box at the end of this section will provide a detailed description. For now, there is no need to understand every parameter: it is sufficient to press on *Continue.*

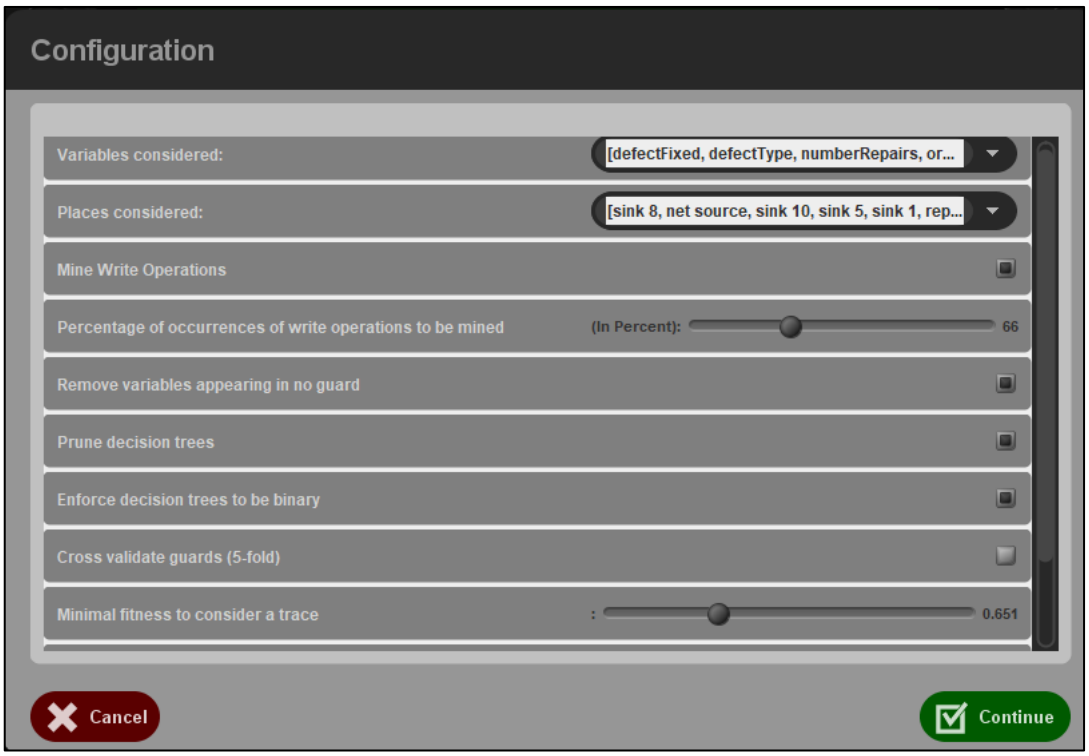



Figure 15. Decision guard mining configuration settings

If you do not change the value of any configuration, you will obtain Figure 16.

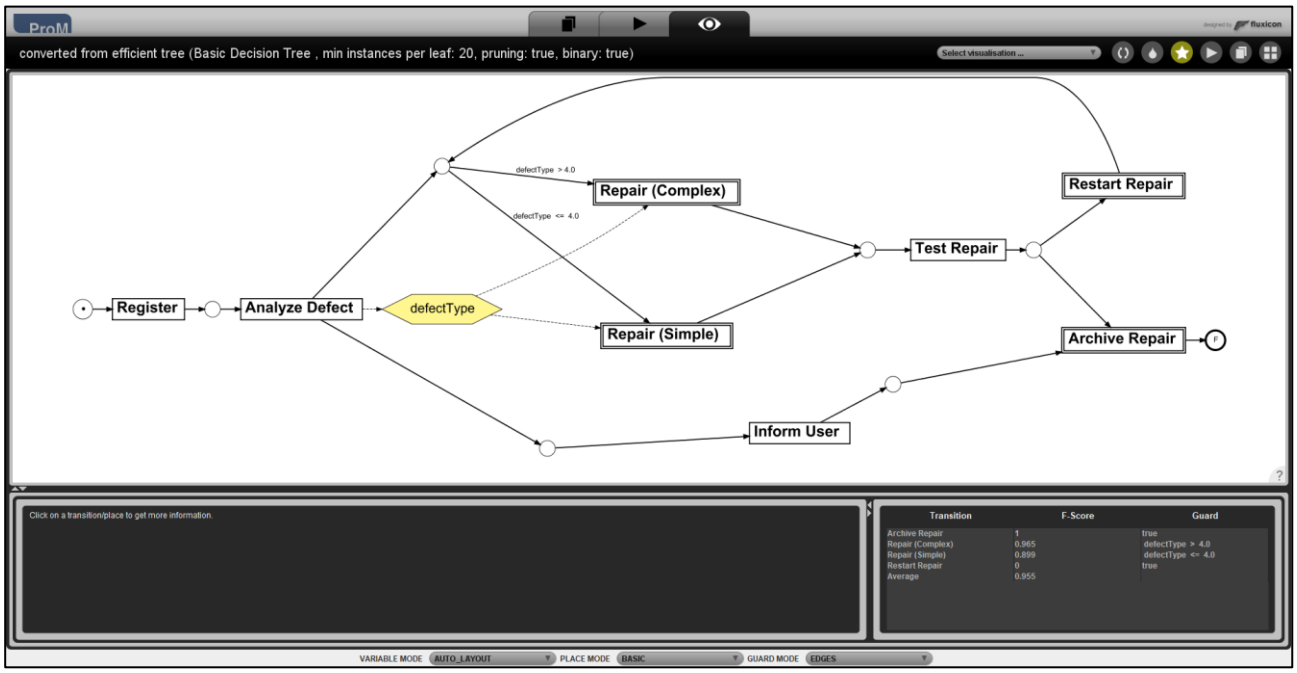
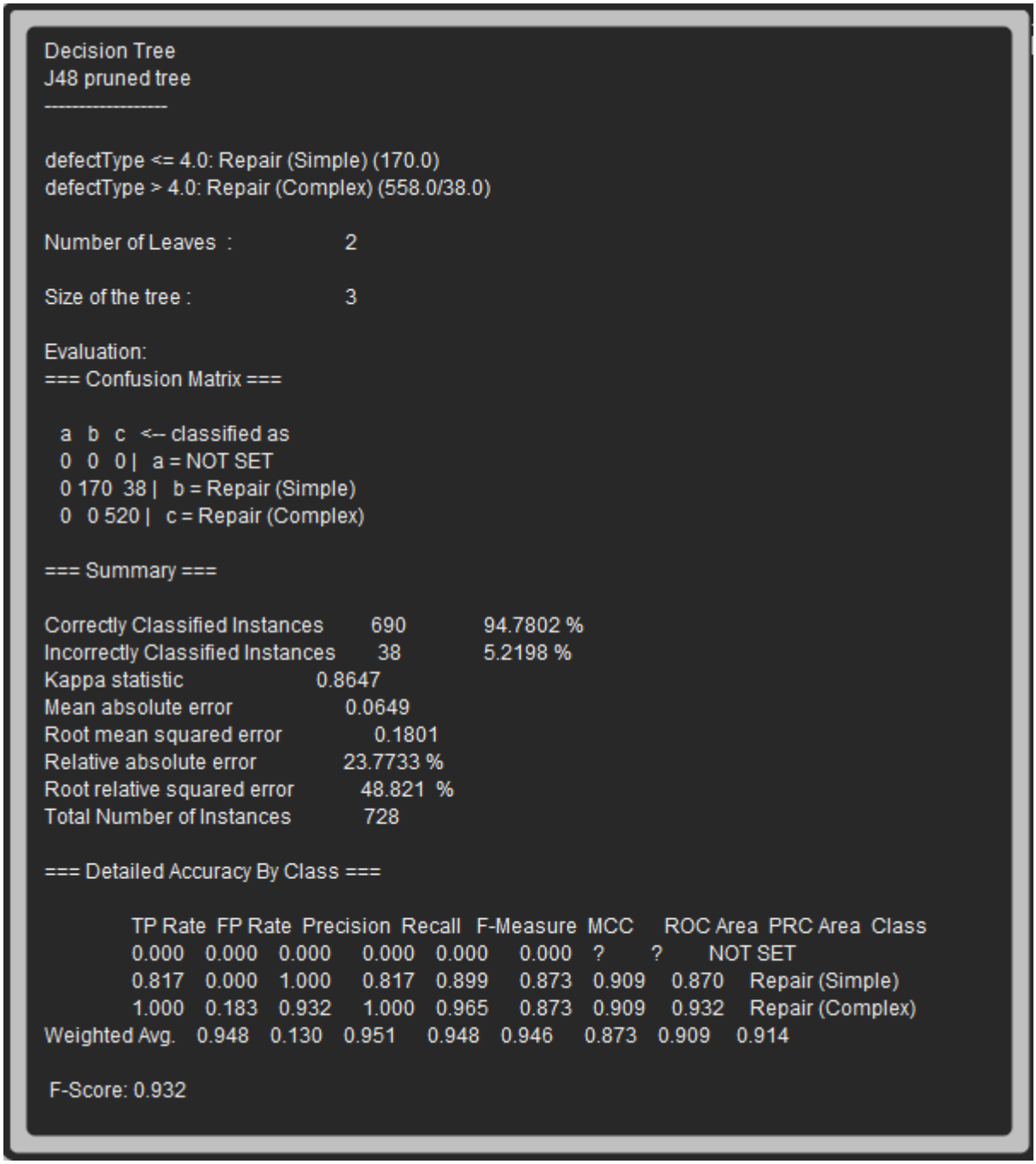



Figure 16. Discovery of the process data flow

The arcs show the guards of the transitions attached to the arc. For instance, the guard of *Repair (Simple)* is $defectType \leq 4.0$. The hexagon represents the variables read and written by the transitions. For instance, variable *defectType* is written by transition *Analyze Defect* and used to determine whether the reparation is simple or complex.



Remember that decision guard mining is based on decision-tree discovery. A decision tree is learned for each Petri-net place with multiple ongoing arcs. If you click on the place highlighted by a green circle, the statistics related to decision-tree discovery is shown in Figure 17.



```
Decision Tree
J48 pruned tree
------------------

defectType <= 4.0: Repair (Simple) (170.0)
defectType > 4.0: Repair (Complex) (558.0/38.0)

Number of Leaves  :          2

Size of the tree :          3

Evaluation:
=== Confusion Matrix ===

   a   b   c   <-- classified as
   0   0   0 |   a = NOT SET
   0 170  38 |   b = Repair (Simple)
   0   0 520 |   c = Repair (Complex)

=== Summary ===

Correctly Classified Instances         690               94.7802 %
Incorrectly Classified Instances        38                5.2198 %
Kappa statistic                          0.8647
Mean absolute error                      0.0649
Root mean squared error                  0.1801
Relative absolute error                 23.7733 %
Root relative squared error             48.821  %
Total Number of Instances              728

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure  MCC      ROC Area  PRC Area  Class
               0.000    0.000    0.000      0.000    0.000      0.000    ?         ?         NOT SET
               0.817    0.000    1.000      0.817    0.899      0.873    0.909     0.870     Repair (Simple)
               1.000    0.183    0.932      1.000    0.965      0.873    0.909     0.932     Repair (Complex)
Weighted Avg.  0.948    0.130    0.951      0.948    0.946      0.873    0.909     0.914

F-Score: 0.932
```

Figure 17. Statistics of decision-tree discovery

The high F-score value indicates that the guards are reliable, with just 38 misclassified cases. Note that k-folder validation has been used.

The meaning of the configuration parameters is as follows:

- **Variables considered**: This can be used to exclude certain variables from the analysis. For instance, we exclude certain variables because those variables are trivially involved in guards at decision points. By default, every variable is taken into consideration.
- **Place considered**: This can be used to exclude certain places that are potential decision points from the decision tree mining.
- **Mine Write Operation**: When this option is enabled, in addition to discovering the guards, the plug-in also takes care of discovering the process variables written by each process activity. Keeping this option on allows for discovering a coherent model where guards are defined over process variables that take on a value given by some previous activity in the same process instance. This would make the model less readable. It can be deactivated if only guards are interested. It will lead more readable model where one can easily analyze the guards, but the model will not be sound because variables never take on values.
- **Percentage of occurrence of write operations to be mined**: This parameter is used to configure the support for a write operation to be mined. If value X is chosen, this means that a write operation for variable V is mined for a certain activity A if the log attribute V is associated with at least X% of the events referring to A. The default value is 66%.
- **Remove variables appearing in no guard**: If an attribute appears in the event log but not in any guard, this means that such an attribute does not influence choices at any decision point. Therefore, adding such an attribute increases the complexity of the model without being of any actual use. By deselecting this option, every attribute appearing in the log becomes a process variable, even if they are not distinguishing at any decision point.
- **Prune decision trees**: As discussed during the lecture and the instruction, the decision miner builds a decision tree for each decision point. If this opinion is enabled, decision trees are pruned. If decision trees are pruned, guards are small in size. Therefore this opinion is used to reduce the length of guards, even if this can have a negligible effect on the confidence of the guards.
- **Enforce decision trees to be binary**: If this option is enabled, decision trees are enforced to be binary. Activating such an option is useful when there are many literal attributes (i.e. strings). Without this opinion to be activated, when there is a split on a literal attribute A, every value is separately enumerated: $if\ A = value\_1\ then\ SOMETHING\_1\ else\ if\ A = value\_2\ then\ SOMETHING\_2\ else\ if\ A = value\_3\ then\ SOMETHING\_3\ \ldots\ if\ = value\_N\ then\ SOMETHING\_N$. When this option is activated, the most discriminating value of the attribute is enumerated, and the others are merged: $if\ A = value\_J, then\ SOMETHING\ else\ SOMETHING\_ELSE$.

- **Cross validate guards (5-fold)**: This option, which is de-activated by default, enables cross-validation. Cross-validation involves:
  - Partitioning the data set into complementary subsets.
  - Performing the analysis on one subset (called the *training set*).
  - Validating the analysis on the other subset (called the *testing set*).

  Multiple rounds of cross-validation are performed using different partitions to reduce variability, and the validation results are averaged over the rounds. When activating this option, the 5-fold validation is used. The original sample is randomly partitioned into *five* equal-sized subsamples. A single subsample is used as *a testing set* of the five subsamples, and the remaining *four* subsamples are used as *a training set*. The cross-validation process is then repeated *five* times (the *folds*), with each of the five subsamples used exactly once as the validation data. The five results from the folds can then be averaged (or otherwise combined) to produce a single estimation.
- **Minimal fitness to consider a trace**: This option is concerned with fixing a threshold for the traces used to construct the decision trees. Every trace with fitness lower than the threshold is discarded. Initially, the proposed value is the average fitness over all log traces.
- **Minimal numbers of instances per decision-tree leaf**: This value is also used to constrain the maximum depth of the tree and, hence, to keep the size of the guards reasonable. The choice of a value X indicates that any decision tree is such that each tree leaf is associated with at least X% (per mil) of the instances.
- **Algorithm to mine guards**: There are several algorithms available. For the course, students can focus on the three below:
  - *Basic Decision Tree*: The algorithm illustrated in the class.
  - *True/False Decision Tree*: A rather naïve algorithm that is capable of discovering overlapping guards. The basic algorithm learns XOR-split at the decision point where the guards of the different activities are mutually exclusive. For any execution that reaches the decision point, one and exactly one activity is enabled for performance. This algorithm removes this constraint: in many case studies, when reaching a decision point in a certain state, more than one activity should be enabled even with the same assignment of values to variables. In similar cases, the choice of which activity will fire is entirely non-deterministic. Please note that this is still an XOR split and differs from an OR split: still, only one of the activities at the decision point will be executed.
  - *Overlapping Decision Tree*: The first step corresponds to the algorithm for overlapping guards, which was briefly illustrated in the class. However, there is a second step that involves all executions that are not compliant with the rules just discovered: the executions that are compliant are filtered out. In a nutshell,

- *Overlapping Decision Tree*: The first step corresponds to the algorithm for overlapping guards, which was briefly illustrated in the class. However, there is a second step that involves all executions that are not compliant with the rules just discovered: the executions that are compliant are filtered out. In a nutshell, the executions that are retained are then used again as input for the algorithm. The new rules discovered on the retained executions are considered alternatives. Namely, suppose that f is the rule discovered for activity a at the first step and g is the rule at the second step. The overall rule for activity a will be (a or b). It is likely that, even after the second step, some executions are still not compliant. The second step is iterated until their number is smaller than a given threshold. For further information, please refer to the publication:

**F. Mannhardt, M. de Leoni, H.A. Reijers, W.M.P. van der Aalst** "Data-driven Process Discovery - Revealing Conditional Infrequent Behavior from Event Logs". In Proc. of the 29th International Conference on Advanced Information Systems Engineering (CAiSE 2017), 12-16 June 2017, Essen, Germany.

## 5. Social-Network Analysis

Here, we illustrate the ProM functionalities for analyzing the social network of the resources involved in the repairing process. In particular, we focus on the resource's work hand-over and the similarity of profiles. The latter allows for discovering the roles of the resources involved in the process.

### Hands-over of work

This part shows how the social network based on the handover of work can be derived using ProM. For social network analysis, no process model is necessary. We continue using log *repairExample2.xes*. The discovery of social networks based on the handover of work is implemented in the ProM plug-in "*Mine handover-of-Work Social Network*". After launching the plug-in on the event log, a configuration window is opened. Please, let us ignore the parameters and click *Continue*.

The plug-in computes the social network and visualizes it as given in Figure 18.

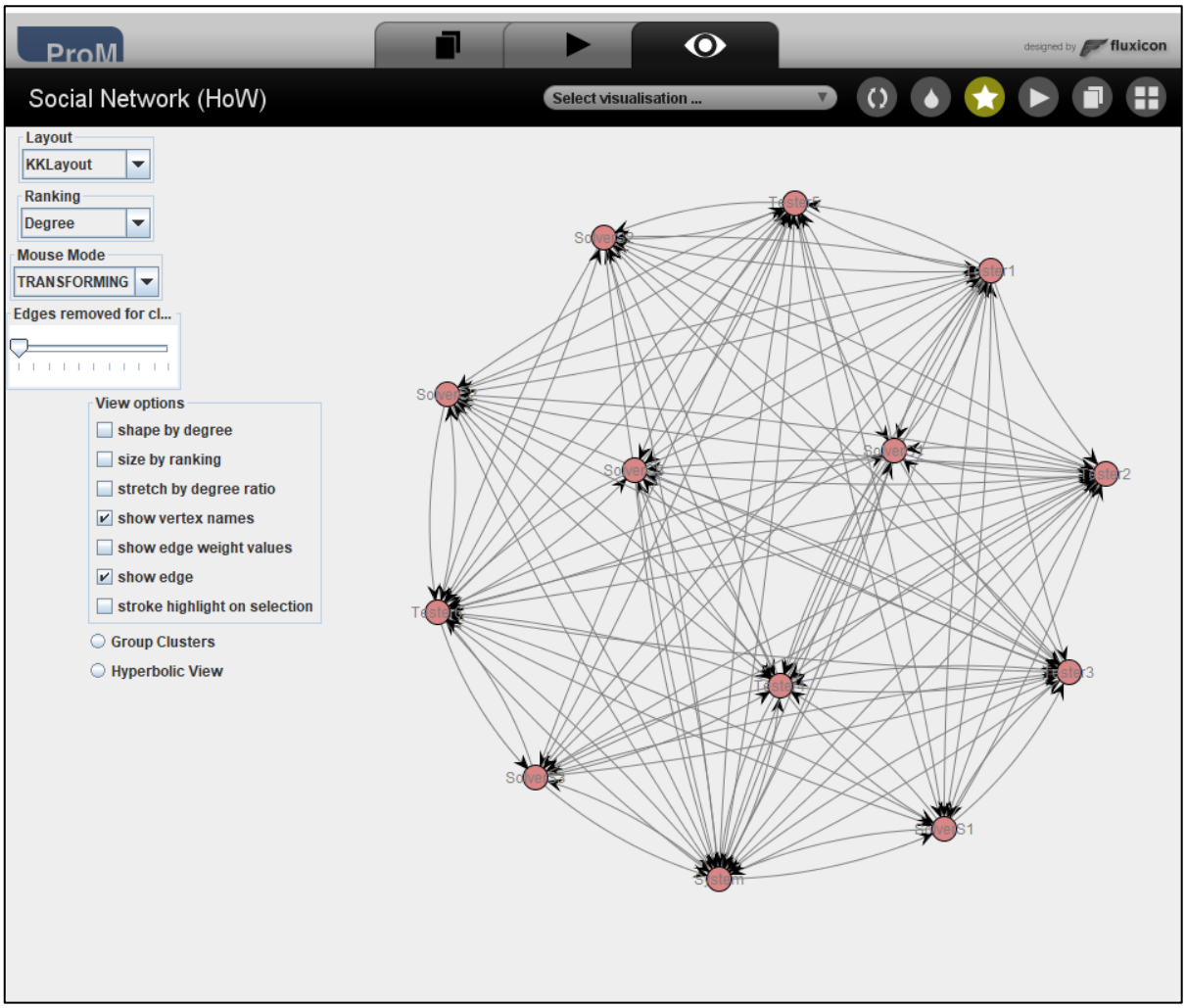



Figure 18. Social network (handover of work)

Now, we can aim to discover roles by clustering nodes. For this purpose, we need to filter out the arcs with low weight (i.e., with few cases of hand-over) and activate the clustering. This is achieved by moving the slider *"Edge removed for cl..."* towards the right. For instance, if we move it to the middle, we obtain Figure 19.

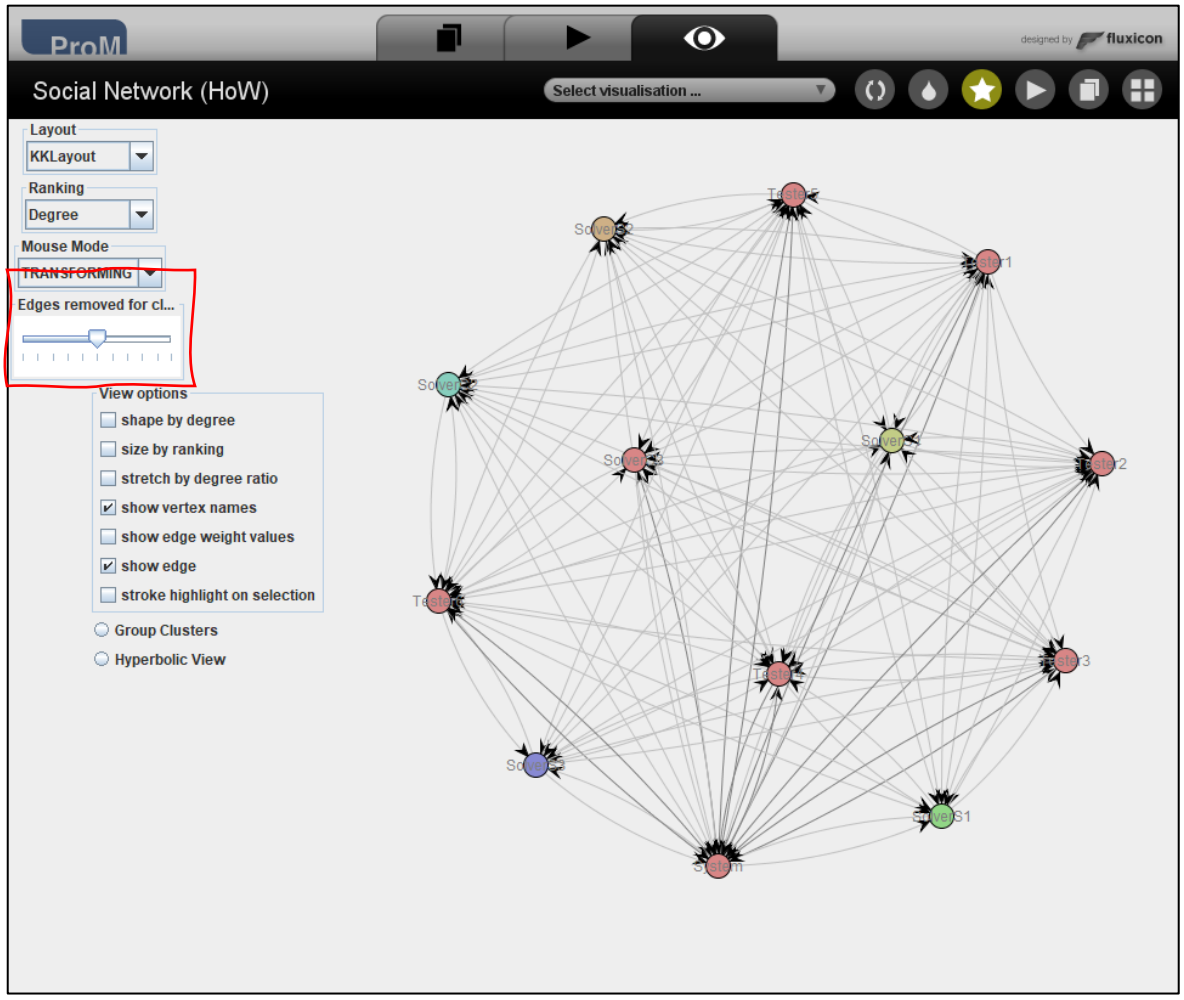



Figure 19. Social network with filtered low-weight arcs

The arcs that have been removed are coloured light grey. The different colors indicate the different clusters of resource nodes (i.e., nodes connected to each other). To better visualize the clusters, we can activate the clustering by selecting the option *"Group Clusters"*, thus obtaining the following results given in Figure 20.

Nodes belonging to the same cluster are put close. It shows that the various solver's hands relatively little work over each other (please remember that low-weight arcs have been removed). Therefore, they have been put in different clusters.

In the brown cluster, there are several resources. The first question is which resource is more involved in that cluster. To answer it, we activate the option *"size by ranking"* like in Figure 21. Now, nodes have a diameter that is proportional to the sum of the number of ingoing and outgoing arcs.

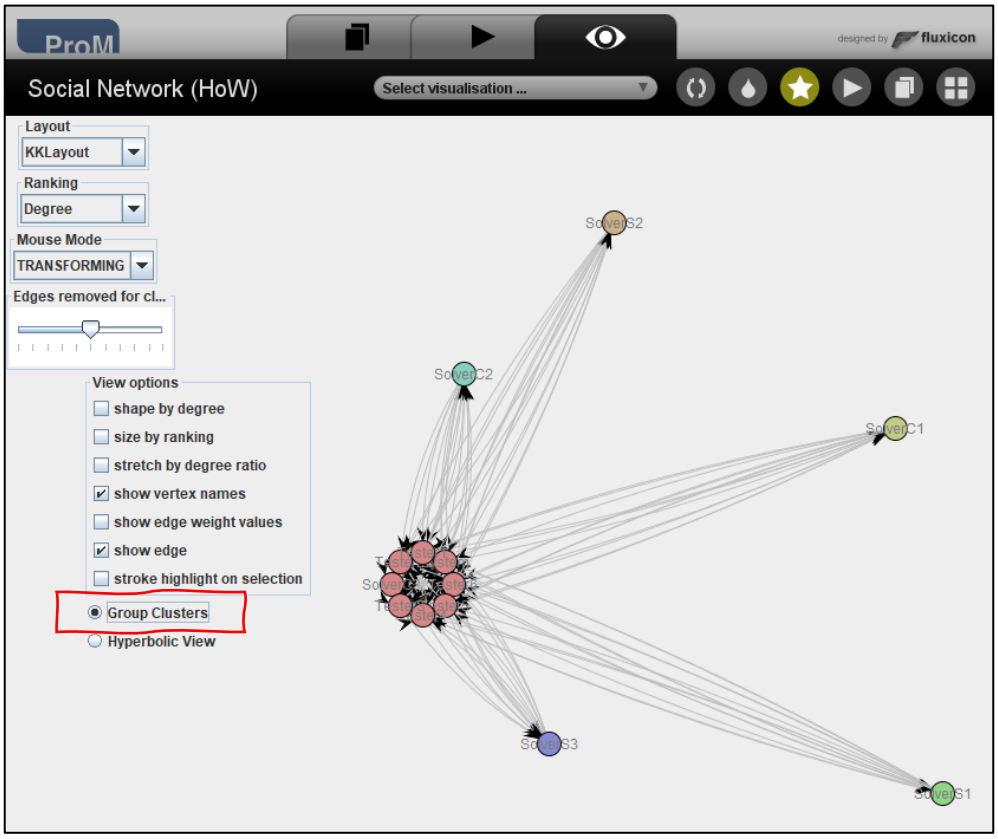



Figure 20. Social network with clustered groups



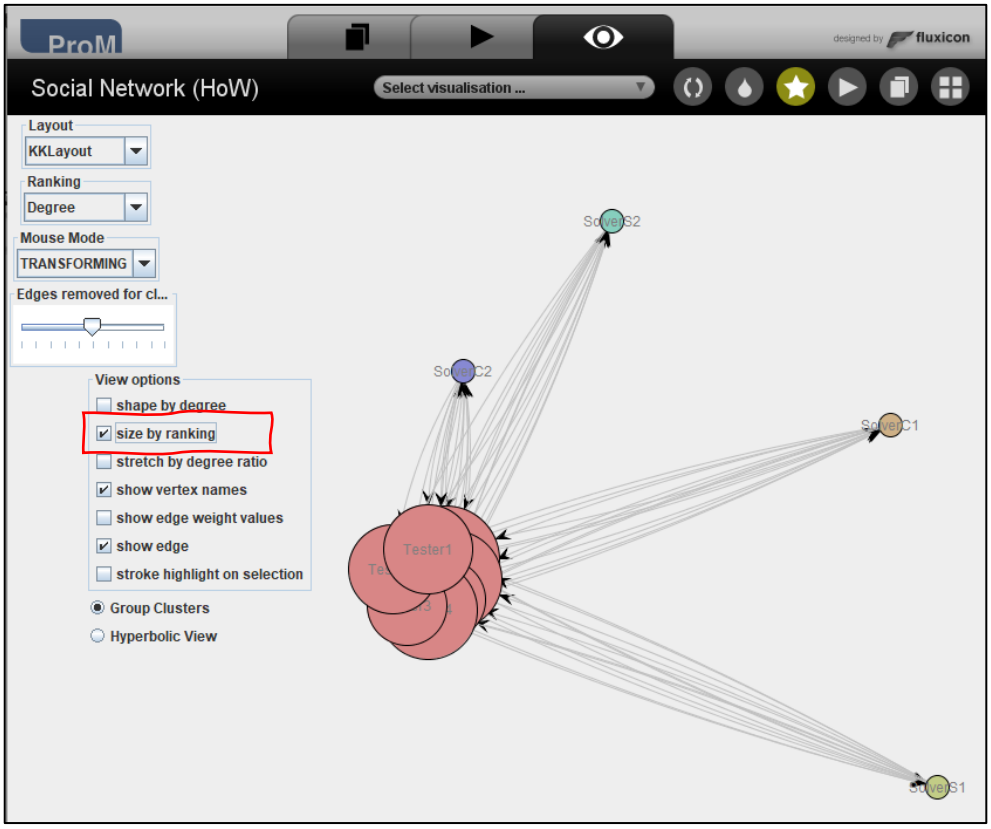



Figure 21. Social network with size by ranking

In the brown cluster, nodes are overlapping. We should remove such an overlapping by moving the node. We need to change the mouse mode to *PICKING* to pick and move nodes. Then, we can manually rearrange nodes like in Figure 22. The resource *"System"* has a predominant role in the brown cluster. Some resources *SolverC3* and *SolverS3* are obviously not much involved in the process.

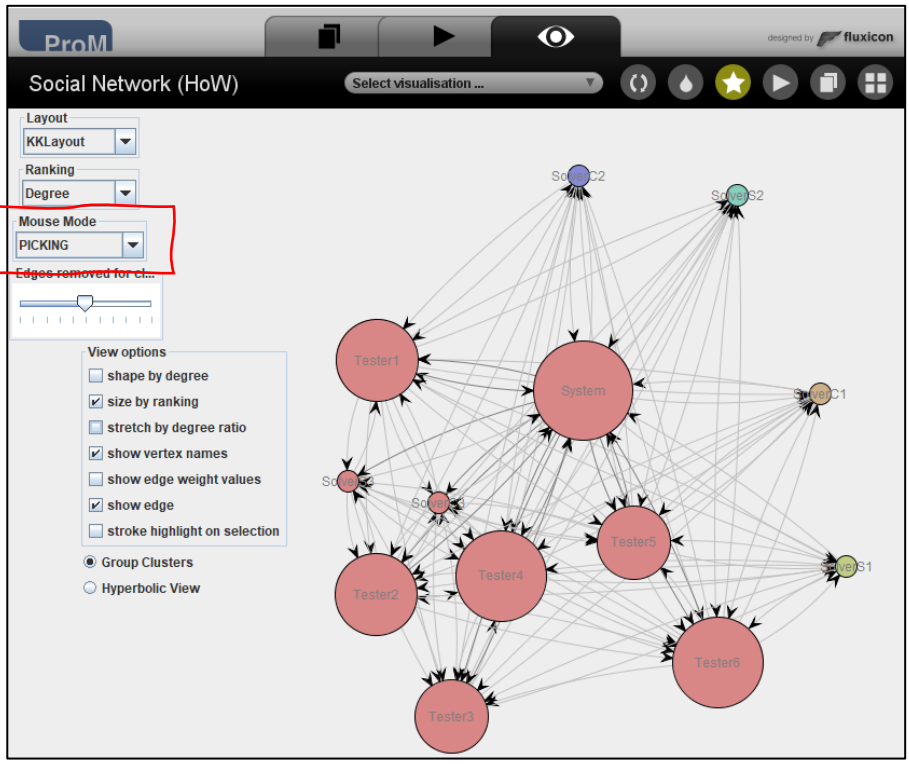



Figure 22. Social network with picking mouse mode

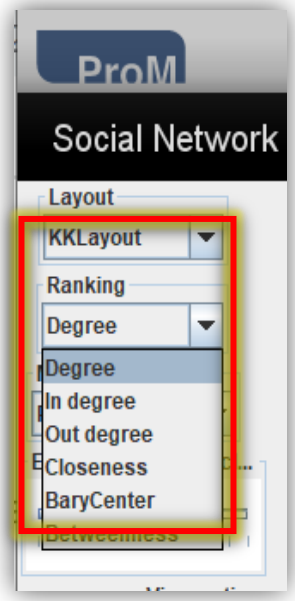



Figure 23. Ranking Types

The size of the node depends on the ranking (given in Figure 23), which can be customized with different options:

- **Degree, In Degree, Out Degree** shows the number of arcs entering a node (In Degree), exiting a node (Out Degree), or the sum of both (Degree). For social network analysis, the In- and Out-Degree indicate the amount of work the resource receives from or hands over to others, respectively (and the Degree is the amount of work that receives from and hands over to others).
- **Closeness** measures a node's proximity to all other nodes in the graph. The closeness of a node in a graph (such as a social network) is the inverse of the average shortest-path distance from the node to any other node in the graph. In the social network analysis, it can also be viewed as the efficiency of each resource in handing work over to all other resources.
- **Betweenness** indicates the number of shortest paths from all nodes to all others that pass through that node. A node with high

betweenness centrality greatly influences the transfer of items through the network, if the item transfer follows the shortest paths. For social network analysis, it can also be seen the importance of a resource in the process of executions. The higher betweenness means that the resource plays an intermediary role when the other two resources aim to hand over work to each other.

The node with a cyan border is SolverS2, and it is the node selected. The nodes with a thicker black border are the nodes with which SolverS2 is connected. The fact that no solver is shown with a thicker border makes us conclude that solvers never hand work over to each other. We obtain the same result if we click on any other Solver resource. The solver nodes are those in charge of simple repairing (namely those named "SolverS…") or complex repairing (namely those named "SolverC…").

We can easily discover more information if we activate the *"stroke highlight on selection"* option (see Figure 24). When this option is enabled, if we click on a node, the border of the nodes becomes thicker. The node with a cyan border is *SolverS2*, and it is the node selected. The nodes with a thicker black border are the nodes with which *SolverS2* is connected. The fact that no solver is shown with a thicker border makes us conclude that solvers never hand work over to each other. We obtain the same result if we click on any other Solver resource. The solver nodes are those in charge of simple repairing (namely those named "*SolverS…*") or complex repairing (namely those named "*SolverC…*").

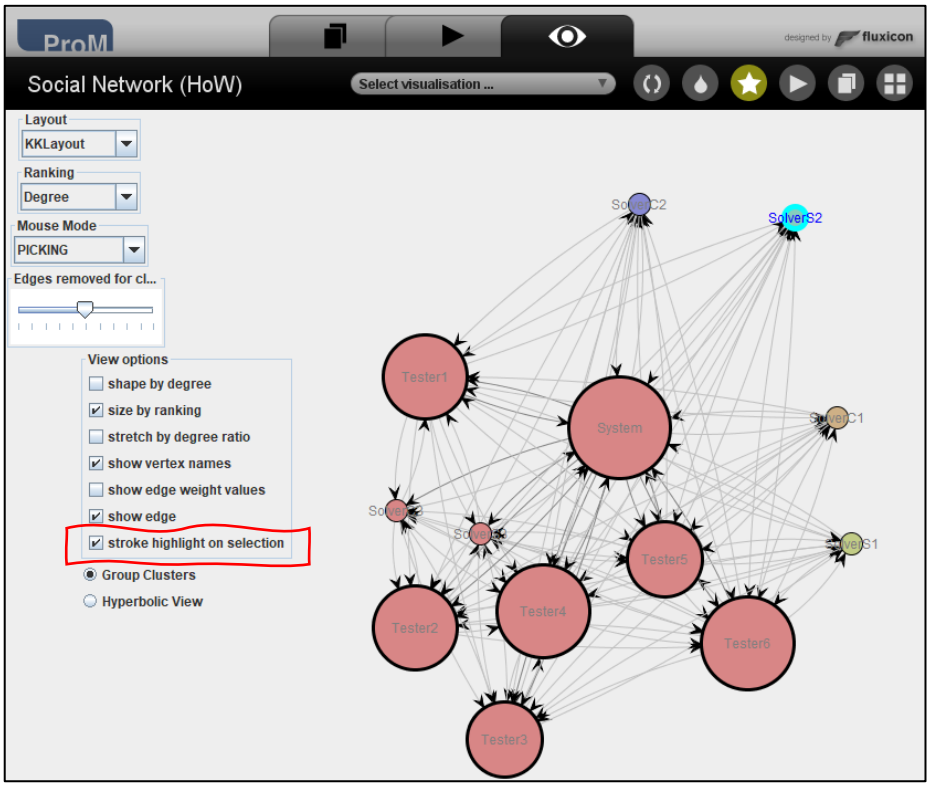



Figure 24. Social network with stroke highlight on selection

## 6. Exercises

This exercise focuses on the road-traffic fine management. Use the Petri net stored in the file ***Complete-RoadTrafficManagementProcess.pnml***, which corresponds to one solution of the exercise for conformance checking and event log ***RoadFineLog_30000_traces.xes.gz***.

In particular, answer the following questions:

1. What is the probability that a fine is paid before being sent?
2. Fines need to be sent within 90 days. Is it usually happening?
3. The penalty must be added 60 days after the offender is notified (i.e., after Insert Fine Notification). Is this happening?
4. On average, how long does it take for a fine to go for credit collection?
5. Do event attributes drive any decisions? If so, which decisions and what attributes do they influence?
6. Inspect the log using dotted charts[1]. Do you see any seasonal pattern? Are activities executed in batches at certain points of the process?



[1]Traces are not ordered by the timestamp of the first event. This causes trouble in the default visualization. It is necessary to explicitly indicate that "*trace sorting*" is "*Sort on time:timestamp of the first event*".