



Prolog-based Expert System

Reasoning about The Disruption Patterns for
Train System Using Bayesian Network and
Prolog

User Manual

Version 1.0

07/06/2018

Table of Contents

1. Introduction	1
2. Hardware and Software Requirements	2
2.1 Hardware Requirements	2
2.2 Software Requirements	2
3. Getting Started	3
4. Program Functionality and Query	6

List of Figures

Figure 1. Downloading github repository	3
Figure 2. Extracting .zip file	3
Figure 3. Running the program user interface	4
Figure 4. Prolog interpreter	4
Figure 5. Consulting necessary files.....	5
Figure 6. Prolog-based Expert System main menu for reasoning the disruption patterns in train system.....	5
Figure 7 . An example of the query implementation of specification 1 to calculate $Pr(fallen_tree \mid fire)$	6
Figure 8. An example of the the query implementation od specification 2 to calculate $Pr(fallen_tree \mid fire)$	6
Figure 9. An example of the query implementation for specification 3 to find the number of children (directly triggered disruption) and parent (directly triggering disruption) of a disruption.	7
Figure 10. Disruption that categorize as <i>safety criticalness</i>	7
Figure 11. Disruption that categorize as <i>mission criticalness</i>	8
Figure 12. . Disruption that categorize as <i>need immediate criticalness</i>	8
Figure 13. Reasoning about disruption path, a path contains a particular disruption and a path that does not contain a particular disruption	9
Figure 14. Disruption that does not lead to any other disruptions.	9
Figure 15. Disruption that is commonly triggered by two disruptions.	10

Figure 16. Disruption that is common triggers by the same disruption.	10
Figure 17. The most triggering disruption.....	11
Figure 18. The most triggered disruption.....	11

1. Introduction

We construct a Prolog-based expert system to reason about the disruption patterns for train system using Bayesian network and Prolog. The disruptions dependencies are modeled using Bayesian network and the reasoning is carried on using Prolog. We choose Bayesian network because it is one of the most efficient and elegant framework to represent and reason using probabilistic model. The causative relationship among disruptions is represented through Directed Acyclic Graph (DAG). We use Prolog to improve efficiency of the reasoning process by defining Bayesian network and its probabilistic information into a knowledge base. The causative relationship among disruptions are also modeled in terms of Prolog rules. Our Prolog-based expert system combines the statistical reasoning using Bayesian network and logic programming efficiency. The system provides comprehensive reasoning regarding the causative probability of events, the causative relationship among disruptions, as well as the most triggering.

2. Hardware and Software Requirements

2.1 Hardware Requirements

To make sure this Prolog-based Expert System working properly, please complete this following minimum hardware specification to run this program:

Operating System : Windows 7 or higher.

Processor : Intel Processor with clock speed 1.9 GHz or higher

RAM : 2 GB of RAM or higher

2.2 Software Requirements

Please kindly install SWI-Prolog version 7.6.1

3. Getting Started

Here we show you the step by steps of how to run this program:

1. Please download all the files in the following repository :

<https://github.com/yunitarp/ReasoningTrainDisruptionWithProlog>

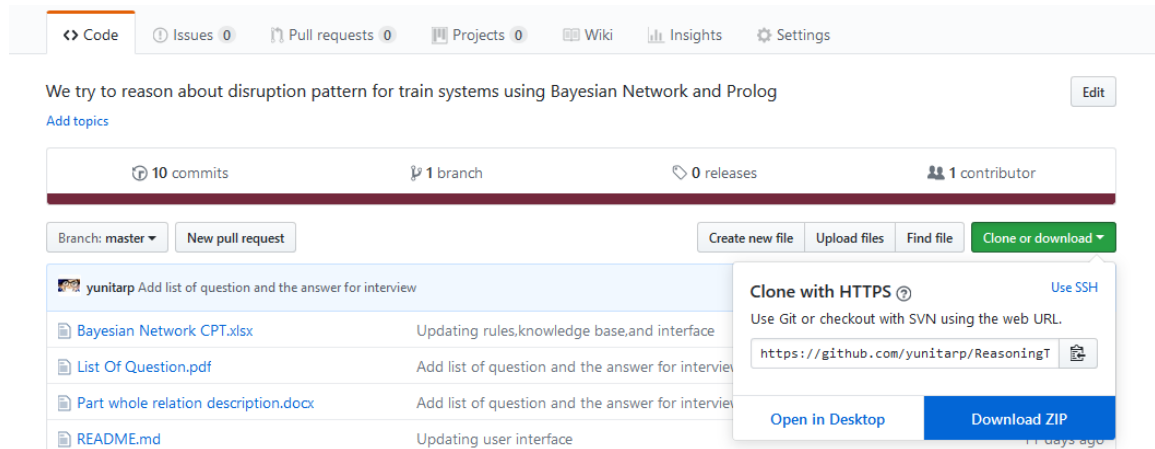


Figure 1. Downloading github repository

2. Extract the .zip file.

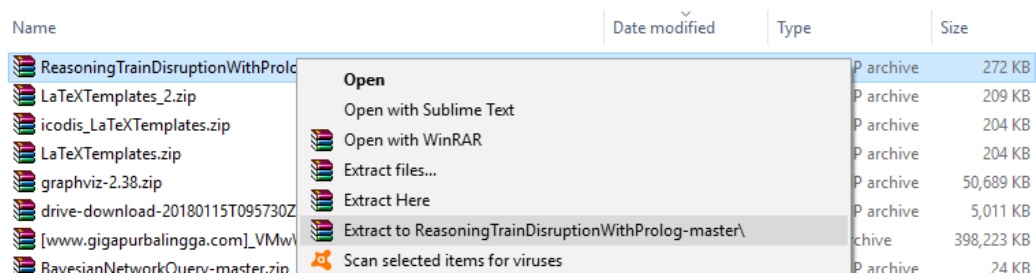


Figure 2. Extracting .zip file

3. Go to the directory where the files is extracted, double click/run the "user_interface.pl" files.

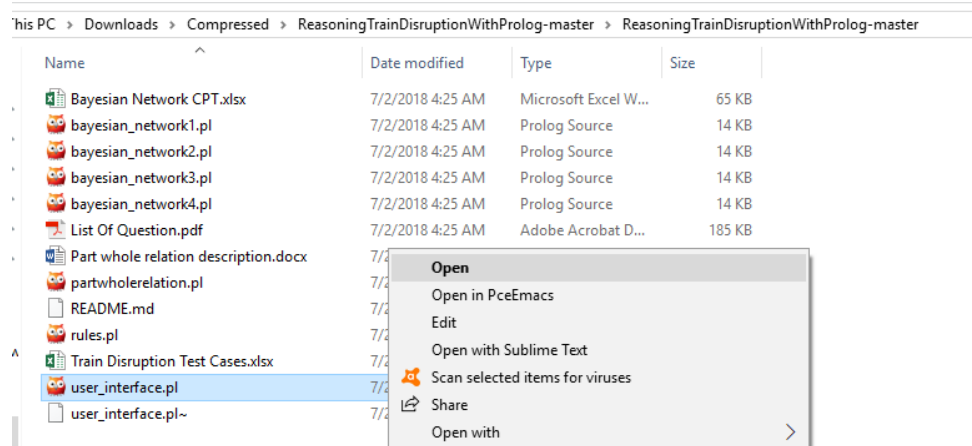


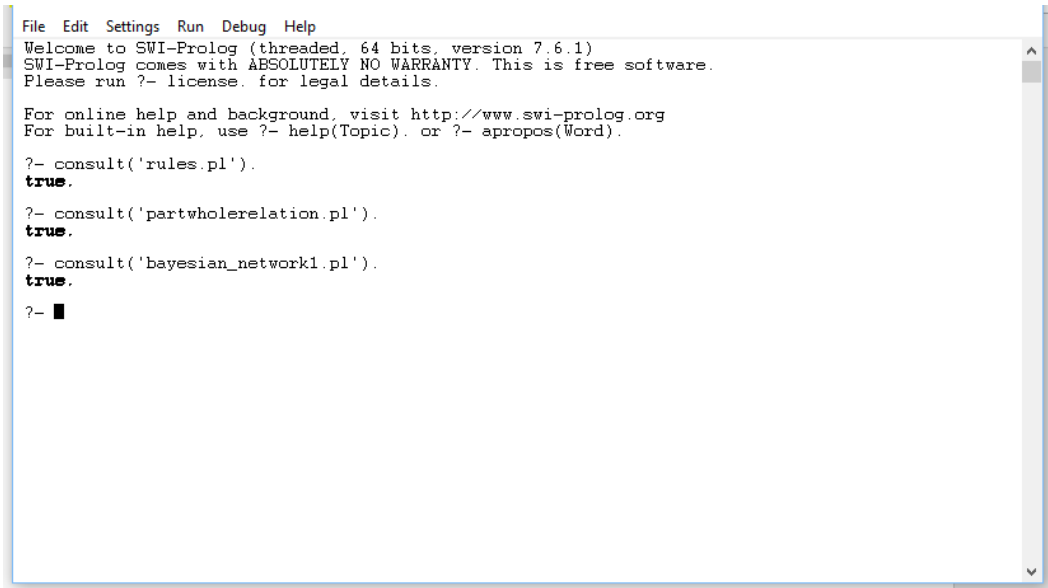
Figure 3. Running the program user interface

4. Prolog-based Expert System main program will be opened



Figure 4. Prolog interpreter

5. Consult the "rules.pl", "partwholerelement.pl", and "bayesian_networkN" that you want try to reason using the following command:
 - `consult('rules.pl').`
 - `consult('partwholerelement.pl').`
 - `consult('bayesian_network1.pl').` example if we want try to reason the Bayesian network 1.



```
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult('rules.pl').
true.

?- consult('partwholerelement.pl').
true.

?- consult('bayesian_network1.pl').
true.

?-
```

Figure 5. Consulting necessary files.

6. Type “menu.” on the Prolog interpreter, all of menu will appear. Then, just choose the menu number for the query you want to see.

```
File Edit Settings Run Debug Help
Welcome to SWI-Prolog (threaded, 64 bits, version 7.6.1)
SWI-Prolog comes with ABSOLUTELY NO WARRANTY. This is free software.
Please run ?- license. for legal details.

For online help and background, visit http://www.swi-prolog.org
For built-in help, use ?- help(Topic). or ?- apropos(Word).

?- consult('bayesian_network1.pl').
true.

?- consult('partwholerelement.pl').
true.

?- consult('rules.pl').
true.

?- menu.
Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggered Disruption
9. Common Triggering Disruption
10. Not lead to any other disruptions
11. Exit
Choice:
```

Figure 6. Prolog-based Expert System main menu for reasoning the disruption patterns in train system.

4. Program Functionality and Query

Here we describe the program functionality and the example of Prolog query for each specification:

1. Calculate the probability of certain disruption given other disruptions.

```
?- menu.  
Main Menu  
1. Safety Criticalness  
2. Mission Criticalness  
3. Need Immediate Action  
4. Disruption Probability  
5. Trace Disruption  
6. The most-triggering disruptions  
7. The most-triggered disruptions  
8. Common Triggged Disruption  
9. Common Triggering Disruption  
10. Exit  
Choice: 4.  
Disruption: |: fallen_tree.  
Conditional: |: fire.  
Path: []  
Probability: [0.008613618]
```

Figure 7 . An example of the query implementation of specification 1 to calculate $Pr(\text{fallen_tree} | \text{fire})$.

From query shown in Figure 7, we obtain the probability if *fallen tree* disruption happened given *fire* disruption happened is 0.00861.

2. Find paths from one disruption to other disruptions from given conditional probability problem.

```
?- menu.  
Main Menu  
1. Safety Criticalness  
2. Mission Criticalness  
3. Need Immediate Action  
4. Disruption Probability  
5. Trace Disruption  
6. The most-triggering disruptions  
7. The most-triggered disruptions  
8. Common Triggged Disruption  
9. Common Triggering Disruption  
10. Exit  
Choice: 4.  
Disruption: |: series.  
Conditional: |: fire.  
Path: [fire,overhead_line,static_inverter,ac,series]  
Probability: 0.0866614752888476
```

Figure 8. An example of the the query implementation od specification 2 to calculate $Pr(\text{fallen_tree} | \text{fire})$.

From the query shown in Figure 8, we obtain a disruption path from *fire* disruption to *series*, namely *fire* \rightarrow *overhead line* \rightarrow *static inverter* \rightarrow *AC* \rightarrow *series*, with the probability $Pr(\text{series} | \text{fire}) = 0.08666$.

3. Find the number of disruption children (directly triggered disruption) and parent (directly triggering disruption), then show the list of disruptions children.

```

?- menu.
Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Exit
Choice: 4.
Disruption: |: ac.
Conditional: |: no.
Probability: 0.02994254825056266
Number of Children: 1
[series]
Number of Parents: 1
[static_inverter]

```

Figure 9. An example of the query implementation for specification 3 to find the number of children (directly triggered disruption) and parent (directly triggering disruption) of a disruption.

From the query shown in Figure 9, we obtain that *ac* disruption has a parent (i.e., *series* disruption) and a child (i.e., *static_inverter* disruption). We also obtain the probability of *ac* disruption happens is 0.02994.

4. Find disruptions that categorize as *safety criticalness* disruption.

```

Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Exit
Choice: |: 1.
Safety Criticalness: [door,horn,switch,signal,wiper,speedometer]

```

Figure 10. Disruption that categorize as *safety criticalness*.

From query shown in Figure 10, we obtain disruption that categorize as *safety criticalness*, namely: *door*, *horn*, *switch*, *signal*, *wiper*, and *speedometer*.

5. Find disruptions that categorize as *mission criticalness* disruption.

```
Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Exit
Choice: |: 2.
Mission Criticalness: [railway,switch,signal,traction,overhead_line]
```

Figure 11. Disruption that categorize as *mission criticalness*.

From query shown in Figure 11, we obtain disruptions that categorize as *mission criticalness*, namely: *railway, switch, signal, traction, and overhead line*.

6. Find disruptions that categorize as *need immediate criticalness* disruption.

```
Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Exit
Choice: |: 3.
Need Immediate Action: [switch,signal,overhead_line,pantograph]
```

Figure 12. . Disruption that categorize as *need immediate criticalness*.

From query shown in Figure 12, we obtain disruptions that categorize as *need immediate action criticalness*, namely: *switch, signal, overhead line, and pantograph*.

7. Find disruption path from a disruption to other disruption that contains and does not contains certain disruption.

```

Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Disruptions that not lead any other disruptions
11. Exit
Choice: |: 5.
Disruption: |: series.
Cause: |: compressor.
List Contain Disruption: |: brake.
List Not Contain Disruption: |: door.
Path containing brake:
[[compressor,brake,series]]
Path not containing door:
[[compressor,brake,series],[compressor,horn,series]]

```

Figure 13. Reasoning about disruption path, a path contains a particular disruption and a path that does not contain a particular disruption

From the query shown in Figure 13, we obtain that there is a *disruption path* when *series* disruption happened and caused by *compressor* disruption. *Brake* disruption may become one of the intermediate disruptions. However, there also disruptions path from *compressor* disruption to the *series* disruption that does not contain *door* disruption in between.

8. Find disruption that does not lead to any other disruptions.

```

?- menu.
Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Disruptions that not lead any other disruptions
11. Exit
Choice: 10.
Disruption that not lead any other disruptions :
[server_and_application,railway,suspension,emergency,mg]

```

Figure 14. Disruption that does not lead to any other disruptions.

From the query shown in Figure 14, we obtain disruption that does not lead to any other disruptions, namely: *server and application, railway, suspension, emergency, and mg*.

9. Find common triggered disruptions from two other disruptions. This specification aims to find disruptions that directly led by two other disruptions. For any disruption

D_i , D_1 is the common triggered disruption of D_2 and D_3 , if both D_2 and D_3 have direct edge to D_1 .

```
Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Disruptions that not lead any other disruptions
11. Exit
Choice: |: 8.
Disruption 1 : |: fallen_tree.
Disruption 2 : |: fire.
Common Triggged Disruption by fallen_tree and fire :
[ac,overhead_line,pantograph,series,static_inverter]
```

Figure 15. Disruption that is commonly triggered by two disruptions.

From query shown in Figure 15, we reason that *fallen_tree* and *fire* disruptions can lead to other disruptions, namely: *AC*, *overhead line*, *pantograph*, *series*, and *static inverter*.

10. Find common triggering disruptions that lead to other disruption. This specification aims to find disruptions that may trigger two other disruptions. For any disruption D_i , D_1 is the common triggering disruption of D_2 and D_3 , if D_1 has an edge to D_2 and D_3 .

```
Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Not lead to any other disruptions
11. Exit
Choice: |: 9.
Disruption 1: |: overhead_line.
Disruption 2 :|: pantograph.
Common Triggering Disruption by overhead_line and pantograph :
[[fallen_tree],[fire]]
```

Figure 16. Disruption that is common triggers by the same disruption.

From the query that shown in Figure 16, we obtain that *overhead line* and *pantograph* disruption are the common triggering of *fallen tree* and *fire* disruption.

11. Find the most triggering disruption in the network.

```

Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Disruptions that not lead any other disruptions
11. Exit
Choice: |: 6
Disruption: fallen_tree
Trigger Disruptions: [ac,overhead_line,pantograph,series,static_inverter]
Total: 5

```

Figure 17. The most triggering disruption

From the query shown in Figure 17, we reason that the most triggering disruption using bayesian_network1.pl file is *fallen tree* disruption. It can lead to five other disruptions, namely: *AC*, *overhead line*, *pantograph*, *static inverter*, and *series* disruption.

12. Find the most triggered disruption in the network

```

Main Menu
1. Safety Criticalness
2. Mission Criticalness
3. Need Immediate Action
4. Disruption Probability
5. Trace Disruption
6. The most-triggering disruptions
7. The most-triggered disruptions
8. Common Triggged Disruption
9. Common Triggering Disruption
10. Disruptions that not lead any other disruptions
11. Exit
Choice: |: 7
Disruption: series
Triggered by Disruptions: [ac,brake,compressor,door,fallen_tree,fire,horn,overhead_line,speedometer,static_inverter,traction,wiper]
Total: 12

```

Figure 18. The most triggered disruption

From the query shown in Figure 18, we reason that the most triggered disruption using bayesian_network1.pl file is *series* disruption. It can be triggered by 12 other disruptions, namely: *AC*, *break*, *compressor*, *door*, *fallen tree*, *fire*, *horn*, *overhead line*, *speedometer*, *static inverter*, *traction*, and *wiper*.