1 Technical report

During this project, several tools were used. In this section, it will be described how these tools were used. The encountered technical difficulties and their solutions or workarounds will also be reported.

1.1 ProM

ProM is an extensible framework that supports a wide variety of process mining techniques in the form of plug-ins[1]. During this project, ProM is used to analyse the event logs. Since it might be that not every process mining technique that is useful for this project is already implemented in ProM, I figured it would be useful to be able to create my own.

1.1.1 Plug-in development

To learn how to make such plug-ins, I found the following tutorial by M. Westergaard very helpful:
`https://westergaard.eu/category/prom-tutorial/`
Naturally, this short tutorial did not cover all information necessary to be able to deploy ProM plug-ins. Hence, I also had to occasionally consult ProM's Javadoc and forum. I used the Eclipse IDE to write the Java code for the plugin. There was some difficulty in being able to load the plugins into ProM since the ProM package manager gets the available plugins from a number of repositories on the `promtools.org` website and I was not able to add my own plugins to those repositories. After some research, trial and error, I came to the following workaround: The ProM package manager keeps a list of repositories in an xml-file that is locally stored in the 'packages'-folder. This file ('packages.xml') directs to a number of remote xml-files (also named 'packages.xml') which refer to zip-files which contain the jar-file of the plugin and a folder 'lib' which contains the jar-files of the libraries that the plug-in needs to be able to run. After downloading one of those remote xml-files, I figured out the format of these files and created a local packages.xml-file for my own plugins to which I referred to in the first packages.xml file as a new repository. I also exported my project as a jar-file and zipped it together with the required libraries. This allowed me to install my own package using the ProM package manager. The downside of this method is that it is very time-consuming to do all of this each time the plug-in is changed. Luckily, Eclipse also allows for running the plug-ins in debug mode. The downside of using debug mode, however, is that other plug-ins are not available unless you have imported their containing packages as libraries in the Eclipse project, and for certain plug-ins (mostly export and import plug-ins) it is difficult to find out to which package they belong. To workaround this issue, I created my own xes-importer so that I could test the plug-ins in debug mode. But since the full event-log of this project is rather large (over 3GB) and my basic xes-importer was not disk-buffered, I was only able to test the plug-ins on a small subset of the data in debug mode. Another problem that I encountered, is that some basic functions (for instance putIfAbsent() from java.util.HashMap) only worked in debug mode. This was hard to find out, but could easily be worked around by simply not using this function.

1.1.2 Plug-ins created

Below is a list of the various plug-ins that I created during this project together with a short description:

- **XesImportPlugin**: This plug-in imports xes-files, it was made to work around the fact that the existing importers were not available in debug mode.
- **DiscreteTimeMarkovChainPlugin**: This plug-in takes a XLog and produces a DiscreteTimeMarkovChain, using the events from the log as states.
- **StationaryDistributionPlugin**: This plug-in takes a DiscreteTimeMarkovChain and computes the eigenvalue-decomposition of the transition-matrix, from which the stationary distribution can be read. The output is a StationaryDistribution object.
- **StationaryDistributionExportPlugin**: This plug-in exports a StationaryDistribution object to a txt-file.
- **TraceLifetimesPlugin**: This plug-in takes an XLog as input and returns a list with the lifetimes of the traces. This list is encapsulated by an TraceLifetimes object.
- **TraceLifetimesExportPlugin**: This takes a TraceLifetimes object as input and stores the lifetimes in a csv-file so that they can easily be analysed using an R-script.

1.2 R

R is a scripting language for statistics. During this project, it has been used to perform statistical analysis on data extracted from the event logs.

References

[1] Prom tools.