Graph Learning for Process Discovery

TESI DI LAUREA MAGISTRALE IN

COMPUTER SCIENCE AND ANGINEERING

|  |  |
| --- | --- |
| Student ID:  Advisor:  Co-advisor:  Academic Year: | 010499057  Matteo Matteucci  Pietro Portolani  202122 |

Author: **Matteo Badenchini**

# Abstract

Here goes the Abstract in English of your thesis followed by a list of keywords. The Abstract is a concise summary of the content of the thesis (single page of text) and a guide to the most important contributions included in your thesis. The Abstract is the very last thing you write. It should be a self-contained text and should be clear to someone who hasn’t (yet) read the whole manuscript. The Abstract should contain the answers to the main scientific questions that have been addressed in your thesis. It needs to summarize the adopted motivations and the adopted methodological approach as well as the findings of your work and their relevance and impact. The Abstract is the part appearing in the record of your thesis inside POLITesi, the Digital Archive of PhD and Master Theses (Laurea Magistrale) of Politecnico di Milano. The Abstract will be followed by a list of four to six keywords. Keywords are a tool to help indexers and search engines to find relevant documents. To be relevant and effective, keywords must be chosen carefully. They should represent the content of your work and be specific to your field or sub-field. Keywords may be a single word or two to four words.

**Key-words:** here, the keywords, of your thesis.

# Abstract in lingua italiana

Qui va l’Abstract in lingua italiana della tesi seguito dalla lista di parole chiave.

**Parole chiave:** qui, le parole chiave, in italiano, della tesi.

# Contents

[Abstract i](#_Toc111325646)

[Abstract in lingua italiana iii](#_Toc111325647)

[Contents v](#_Toc111325648)

[SECTION I – Process Mining 7](#_Toc111325649)

[1. Process Mining 8](#_Toc111325650)

[1.1 Overview and objectives 8](#_Toc111325651)

[1.2 Data 10](#_Toc111325652)

[1.3 Models 11](#_Toc111325653)

[1.3.1 Directly Follows Graphs 11](#_Toc111325654)

[1.3.2 Petri Nets and Workflow Nets 11](#_Toc111325655)

[1.3.3 Transition System 13](#_Toc111325656)

[1.3.4 Dependency Graphs and Causal Nets 13](#_Toc111325657)

[1.3.5 Process Trees 14](#_Toc111325658)

[2. Process Discovery 16](#_Toc111325659)

[3. Bibliography 17](#_Toc111325660)

[4. List of Figures 19](#_Toc111325661)

SECTION I – Process Mining

# Process Mining

In this chapter we will introduce the fundamental concepts related to the filed of process mining, the goals it tries to achieve, the data, and the structures used to model real-life processes. The content of this chapter is of paramount importance since it defines the framework in which the project is developed. In the next chapter we will give a more in-depth explanation of the topic of process discovery, which denotes the main theme of the present work.

## Overview and objectives

Process mining can be defined as the bridge between data science and process science[[1]](#footnote-1) [1]. The goal of process mining is to discover, analyze and improve existing (business) processes by extracting the relevant information from event data stored inside information systems.

A business process is a collection of tasks and activities performed by people or (autonomous) systems that produces an outcome that contributes to the business goal. Processes are frequently described in form of graphs (e.g., Petri Nets), and this representation embodies the so-called model of the process. Event data is the result of the execution of such models.

Process mining represents the core of the Business Process Management (BPM) lifecycle, which describes the distinct phases of managing a particular business process.

Diagram

Description automatically generatedW. van der Aalst defines the BPM lifecycle as composed of three different phases [2]: *(re)design*, *implement/configure*, and *run and adjust*.

Figure 1

After being designed in the *(re)design* phase, the model is transformed into a running system in the *implementation/configuration* phase. Once an implementation of the model is available, the *run and adjust* phase starts: the model is executed, and event data is collected. This data can then be used to extract the actual behavior of the model, which is later analyzed and compared against the expected behavior of the blueprint. Finally, if necessary, the model is redesigned, in the attempt to optimize (or remove) bottlenecks, include observed behavior and remove unobserved one.

The *run and adjust* phase is especially important, since it allows to close the cycle, and therefore to incorporate into the design information about the actual execution of the process in a systematic way.

In the field of process mining, we can identify three different branches (or type of tasks) that conduct the distinct phases of the BPM lifecycle: play-in, play-out and replay:

* Play-in, also known as discovery, refers to the practice of building the model of a process, starting from its observed behavior (event data). It represents the starting point of the BPM lifecycle, since it generates the model that is then put into production and subsequently analyzed and refined.
* Play-out consists instead in generating the behavior of a known process starting from the model that describes it. In this sense it reaches the opposite goal of play-in: it tries to produce event data (usually in form of logs) by simulating the model of the process.
* Replay is responsible for testing the model of a given process against the actual behavior observed. It serves different purposes: to spot bottlenecks, to find deviations from the blueprint and to try to predict problems that can arise from using such a model.

## Data

The starting point of any process mining task is event data, which can come, for example, from information and ERP systems, csv files, APIs, emails, or transaction logs. Data is usually gathered from different sources, and then refined and assembled into event logs, each one expressing the observed behavior of a different process.

Inside an event log, each entry represents a single event, which corresponds to the name of an activity (label) inside the process model. Besides the activity name, an event is usually characterized by other attributes, the timestamp and the case ID being among the most important ones. The case id denotes the unique identifier of a single run of the process model (case): a sequence of activities that, if performed, allows to go from the start of the process to the end of it. Each event belongs to a single case, but different cases may contain the same activity/event (an example of this are the start and end activity, that are always executed for every case).

Other attributes (e.g., resource availability, cost, user/patience name) are possible, and usually present, and are used to better characterize the execution of the process. Attributes can be at case level, if they are shared by all the events of the case (e.g., the patient’s name in case of medical records), or at event level, if they are unique for the event. Figure 2 shows an example of a real-life event log, in which besides case id, activity name and timestamp (*case item*, *event concept: name* and *event time: timestamp* respectively) we can see many more attributes that characterize the process.

A sequence of activities belonging to the same case is also known as a trace, and a log can be seen as a collection (or multiset) of traces. A trace, taken without its case, can appear multiple times inside a log, since different runs of a model can produce the same outcome reached by means of the same sequence of activities. A variant, instead, is a unique trace inside a log.

Let us clarify these definitions with an example. Suppose to have a log written in form of a multiset of traces (so a set in which each element is reported with its multiplicity), and be L = {<a, b, c, d>3, <a, c, b, d>2, <a, e, d>} such a log. Inside L we can identify three different variants, namely <a, b, c, d>, <a, c, b, d> and <a, e, d>, for a total of six traces.

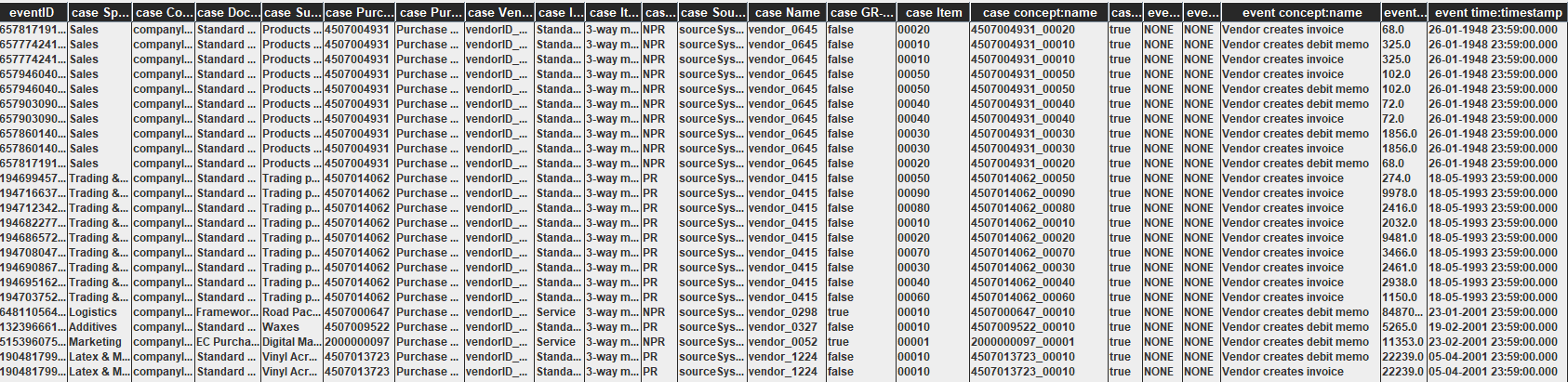


Figure 2

## Models

Processes can be represented in a variety of forms, depending on one’s needs. In this paragraph we will discuss some of the most popular models, be they the final representation that can be presented to the end user or the starting point (or the intermediate representation) of a discovery algorithm.

### Directly Follows Graphs

The directly follows graph is one of the most straightforward representations of a process and the easiest to build starting from event logs. It consists of a directed graph whose set of nodes corresponds to the set of unique activity names in the event log. Arcs connect activities that directly follow each other inside a trace. The result is a model that captures the temporal relations among activities, but not necessarily causal dependencies (e.g., in case of parallel activities). Logs can be filtered beforehand (by removing infrequent activities and/or traces) to avoid building so-called spaghetti models, keeping only the most interesting information.

### Petri Nets and Workflow Nets

A petri net is a bipartite and directed graph composed of places, transitions and arcs connecting places to transitions and vice versa. A place contains a given number of tokens, which represent the resources that can be used to carry out the execution of the process. Places are usually displayed as circles, transitions as boxes and tokens as black dots inside a place.

Diagram

Description automatically generatedIn the process mining context, transitions are associated with the activities of the log, while places are the nodes that control the order of execution of the activities by receiving tokens from (a subset of) the transitions in their backward star and passing them to (a subset of) the transitions in their forward star.

Figure 3

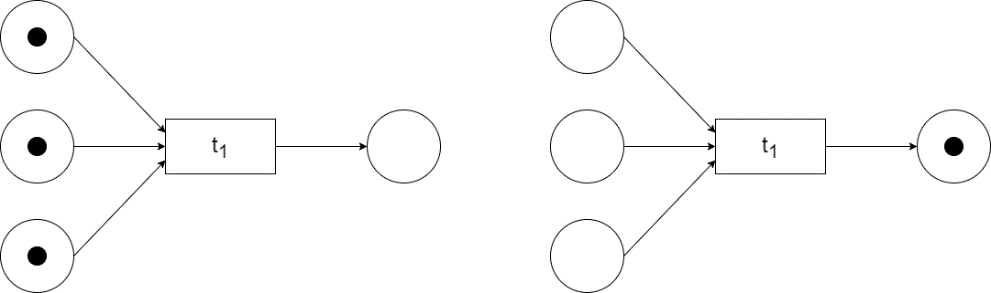


Figure 4

A transition is said to be enabled if each input place (those in its backward star) contains at least a token. When a transition is enabled, it can *fire* by consuming a token from each input place and producing a token for each output place. As you can in Figure 3, transition t1 is not enabled, since one of its input paces does not contain any token. In Figure 4, instead, we can see how the transition t1, once enabled, consumes all input tokens, and produces a new token for each output place (in this example just the one).

The distribution of the tokens inside a net is called marking and it defines the state of the net. A marking can be reachable or unreachable, depending on whether it is possible for the net to reach such a configuration starting from the initial state.

In principle, a place can contain an infinite number of tokens. We can however define an upper bound to such a number and say that a place is k-bounded if there is no reachable marking for which said place holds more than k tokens. A petri net is defined k-bounded if all its places are k-bounded. When k is equal to one, the place (or the net, if the property holds for all its places) is said to be safe.

Petri nets can suffer from deadlock problems: these can arise whenever there is a reachable marking in which no transition is enabled[[2]](#footnote-2). The complete absence of such problems can be verified in live petri nets, nets such that, for every transition, it is possible to reach a marking that enables it starting from the initial reachable marking [3].

When a petri net has a well-defined start (source) and end (sink) place, it is called workflow net. Workflow nets are typically preferred when modeling a process, since the presence of a well-defined source and sink leads to models that are more easily interpretable by humans.

### Transition Systems

A transitions system is a type of model that tries to represent a process as a set of states connected to one another. As already discussed, a log can be seen as a set of traces, each of which specifies a possible execution of a model. In the context of a transitions system, the position of the execution inside a trace defines a state of the process, and the set of all possible states generated in this way (considering all the traces available for a given process) represents the set of nodes in the transition system.

Given two states, if it is possible to transition from the first state to the other by means of a single action, meaning by executing a single activity, the nodes corresponding to such states are connected. The arc connecting two states has a label that corresponds to the name of the activity that needs to be executed to transition from a state to the other.

### Dependency Graphs and Causal Nets

A dependency graph is a directed graph that embeds the causal dependencies among activities in an event log. In this graph, each node corresponds to a unique activity in the event log, while the arcs connect nodes that are causally dependent from one another. Differently from directly follows graphs, two nodes are connected if and only if the corresponding activities are such that one enables the other, that is, given two activities A and B, A must be concluded before B starts, and the execution of B depends on whether A is executed in the first place. Another way to picture such a structure is to think of a petri net from which the places have been removed.

Causal nets are strictly related to dependency graphs: nodes and arcs have the same meaning. However, in this model, each activity (node) has a set of possible input bindings and a set of possible output bindings. Given an activity A, an input binding represents a set of activities that enable A, while an output binding represents the set of activities that activity A can enable [4].

|  |  |
| --- | --- |
| Figure 5 | Figure 6 |

Figure 5 depicts an example of output bindings: activity *a* can enable either the set {*b, d*} or {*c, d*}, but not both. An output binding creates an obligation: once a binding is chosen, all the activities in it must be executed. Figure 6 describes an example of input bindings: activity *e* can be enabled by either the couple {*b, d*} or {*c, d*}, but not both.

Let us consider the bindings in Figure 5 and Figure 6 as belonging to the came causal net: if activity *a* enables *b* and *d*, then activity *e* will be enabled by *b* and *d*, and not *c* and *d*. In other words, an obligation created by an output binding must be consumed by a (set of) input bindings for the activities involved.

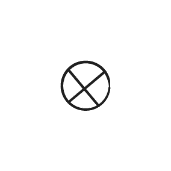
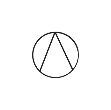
In a causal net the start activity has a no input bindings, while the end activity has no output bindings.

### Process Trees

Process trees are another powerful, yet easy to interpret, way of modeling a process. It describes a process as a tree whose leaves are the activities of process, and the internal nodes are instead operators that connect the different activities in a hierarchical way.

There are four possible operators, each of which defines a specific relation among its children:

* A picture containing shape

  Description automatically generatedSequence : n-ary operator; all the child nodes are are executed one after the other
* Exclusive choice : n-ary operator; only one of the child nodes is executed
* Parallel : n-ary operator; all the child nodes are executed
* Loop A picture containing icon

  Description automatically generated : binary operator; the left child defines the condition, while the right one represents the body of the loop.

# Process Discovery

# Bibliography

|  |  |
| --- | --- |
| [1] | W. v. d. Aalst, Process Mining: Data Science in Action, Springer, 2016. |
| [2] | W. v. d. Aalst, "Business Process Management: A Comprehensive Survey," 2012. |
| [3] | M. Tadao, "Petri Nets: Properties, Analysis and Applications," 1989. |
| [4] | W. v. d. Allst, "Causal Nets: A Modeling Language Tailored towards Process Discovery," 2011. |

# List of Figures

[Figure 1 BPM lifecycle 9](file:///C:\Users\matte\Documents\GitHub\GLPD\documents\thesis_10499057.docx#_Toc111209438)

1. The discipline that focuses on process modeling and the implementation of processes (not necessarily in a data driven way). [↑](#footnote-ref-1)
2. Such markings are called dead markings. [↑](#footnote-ref-2)