Exploring Task Execution Patterns in Event Graphs

Eva L. Klijn, Felix Mannhardt, Dirk Fahland
Eindhoven University of Technology, The Netherlands
{e.l.klijn, f.mannhardt, d.fahland}@tue.nl

***Abstract*—Classical process mining aims to capture the behavior of a process based on a single dimension: the sequence of activities grouped by process cases. This viewpoint fails to capture how individual actors are organizing their work across multiple cases. We present a tool that uses the graph database Neo4j to model actor behavior over different cases as an event graph. We then use Neo4j queries to detect task execution patterns in the graph describing how multiple actors collaborate across multiple cases. Exploring and visualizing these patterns enables the data driven analysis of tasks, routines, and habits as studied in organizations research.**

I. Introduction

Process mining focuses on improving processes by analyzing event data. Classically, recorded events are grouped in an event log under the viewpoint of one (or more) *case identifiers* and ordered by time. The resulting event log describes which *tasks* were performed in which process execution viz. *case*. Process discovery identifies behavioral patterns and information along each case and aggregates them into a process model describing the *control-flow perspective* [1] of a process, which can consist of multiple data objects or entities [2].

Each *task* is performed by an *actor* (or resource) working on the case, which is studied under the *resource-perspective* of the process [1]. An actor moving from one task in a case to a task in another case introduces behavior along the resource perspective and dependencies between tasks of different cases. In [3] we showed that the control-flow and resource-perspective can be studied together as an *event graph* [2] where each event is part of two paths, a case path and a resource path, modeling event dependencies over two behavioral dimensions. One or more case and resource paths synchronizing form a *task execution pattern* which describes work habits of an actor or routines, i.e., how one or more actors collaborate over multiple cases.

In this paper, we present a command-line tool for analyzing such task execution patterns as described in [3]. The tool is realized in Python 3.7 and publicly available[1]. It connects to a Neo4j (neo4j.com) database instance to execute queries (1) for constructing an event graph over the case and resource dimension from a classical event log; (2) for detecting various forms of *task execution patterns* in the event graph and aggregating them to high-level events which can be queried, visualized, and explored using Neo4j. A screencast[2] and a detailed instruction manual[1] explain usage of the tool.

[1]https://github.com/multi-dimensional-process-mining/event-graph-task-pattern-detection
[2]https://vimeo.com/630382325

II. Task Execution Pattern Detection

The configurable end-to-end workflow is implemented in *main.py* and shown in Fig.1.

A. Input & Parameters

The input is a classical event log in CSV format. The CSV file must contain columns for the event classifier, timestamp, case identifier and resource identifier. The columns and details on the used CSV format need to be provided as parameters, e.g., its filename, column keys, column separator and the timestamp format used. We assume a Neo4j database has already been set up and the credentials are provided as parameters. The graph labels assigned for the entities and relationships can be customized if wanted.

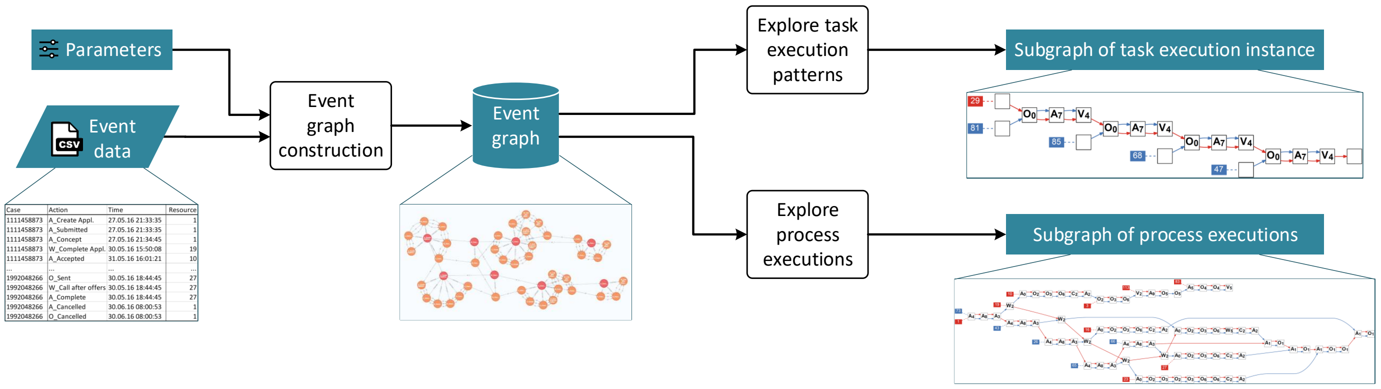
B. Event Graph Creation

The tool creates the event graph in three steps:

1. Preprocessing. Preprocess the event data (*PreprocessSelector.py*) to make it suitable to import to a Neo4j Database instance by standardizing the name and formatting of the event classifier and timestamp column.

2. Initial Graph Creation. Invoke Cypher queries to construct an event graph (*EventGraphConstructor.py*) by limiting the original event graph construction approach [2] to the resource and case entities. In the resulting event graph, each event is an *Event* node that is part of two paths of *directly-follows* edges: the path of all events correlated to the same *case* entity, and the path of all events correlated to the same *resource* entity. For event data over multiple case entities, a user can also choose to construct a custom event graph following the original approach [2] and may then skip step 1 and 2.

3. High Level Event Construction. Detect task execution patterns in the event graph and materialize them in the graph as "high-level event" nodes (*HighLevelEventConstructor.py*). For the most basic pattern type, we query for sub-graphs of event nodes that are all part of the same case path and the same resource path (i.e., the resource works on the case over one or more consecutive events); sub-graphs of other pattern types [3] are found through variations of this query. For each found sub-graph, we create a new *HLEvent* node linked to the events in the sub-graph. We lift the *directly-follows* edges from *Event* nodes to *HLEvent* nodes. This allows to query for larger task execution patterns as patterns of *HLEvent* nodes along case and resource *directly-follows* edges, see [3] for details.


Parameters
Event data
CSV
Event graph construction
Event graph
Explore task execution patterns
Subgraph of task execution instance
Explore process executions
Subgraph of process executions

Case	Action	Time	Resource
1111458873	A_Create Appl.	27.05.16 21:33:35	1
1111458873	A_Submitted	27.05.16 21:33:35	1
1111458873	A_Concept	27.05.16 21:34:45	1
1111458873	W_Complete Appl.	30.05.16 15:50:08	19
1111458873	A_Accepted	31.05.16 16:01:21	10
...	...	...	...
1992048266	O_Sent	30.05.16 18:44:45	27
1992048266	W_Call after offers	30.05.16 18:44:45	27
1992048266	A_Complete	30.05.16 18:44:45	27
1992048266	A_Cancelled	30.06.16 08:00:53	1
1992048266	O_Cancelled	30.06.16 08:00:53	1

Fig. 1. Design of the toolkit (1) taking event data (CSV) and a set of parameters as input, (2) creating a Neo4j event graph and (3) providing subgraph visualizations of task execution pattern instances as output.

C. Event Graph Exploration

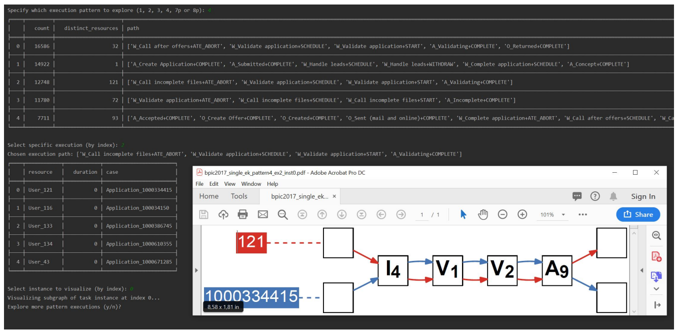
Once the graph and *HLEvents* are constructed, the tool prompts the user to explore the graph for (1) task patterns of a particular type or (2) patterns occurring in a subset cases.

1. Exploring task execution patterns. In [3], several task pattern types have been identified and assigned numbers from 1 to 16. *GraphExplorer.explore_patterns()* prompts the user to specify which pattern type they wish to explore, e.g., type 4 in Fig. 2. It then returns a list of all distinct task execution patterns of this type ordered by frequency. The user can select a specific task pattern to explore further, which will return a list of all instances (sub-graphs) of that particular task pattern. The user can select to visualize a specific instance of the pattern, which is shown in a separate window as a PDF. Fig. 1 (top-right) shows an example of a task execution instance that shows pattern 8', i.e., an actor performing the same sequence of steps for a number of cases one after the other. Other patterns such as resource interruptions (pattern 2, see [3]) and case interruptions (pattern 3, see [3]) can also be explored.

2. Exploring subsets of process executions. GraphExplorer.explore_cases() lets the user specify the case identifiers for which they want to explore task execution patterns. The resulting subgraph of those process executions and task patterns is then output in a separate window in PDF as shown at the bottom right part of Fig. 1.

III. Maturity & Performance

The tool was successfully evaluated in [3], where it was used on two real-life event data sets (BPIC'14 and BPIC'17). These experiments were run on an Intel i7 CPU @ 2.2GHz machine with 32GB RAM. For the larger of the two data sets (BPIC'17, 237MB), the tool was able to construct all graph related constructs in $\sim$ 140 seconds and return lists of executions and instances of all pattern types in under 4 seconds. The subgraph visualizations are generally retrieved in under 2 seconds, but we have also seen various instances that take almost 60 seconds. The tool's performance on this aspect highly depends on the complexity of the subgraph to be visualized. The preprocessing scripts and label settings for the graph output are easily adaptable. We provide example scripts for BPIC'14 and BPIC'17.



```
Specify which execution pattern to explore (1, 2, 3, 4, 7p or 8p): 4

|   |   count |   distinct_resources | path
| 0 |   16586 |                   32 | ['W_Call after offers+ATE_ABORT', 'W_Validate application+SCHEDULE', 'W_Validate application+START', 'A_Validating+COMPLETE', 'O_Returned+COMPLETE']
| 1 |   14922 |                    1 | ['A_Create Application+COMPLETE', 'A_Submitted+COMPLETE', 'W_Handle leads+SCHEDULE', 'W_Handle leads+WITHDRAW', 'W_Complete application+SCHEDULE', 'A_Concept+COMPLETE']
| 2 |   12748 |                  121 | ['W_Call incomplete files+ATE_ABORT', 'W_Validate application+SCHEDULE', 'W_Validate application+START', 'A_Validating+COMPLETE']
| 3 |   11780 |                   72 | ['W_Validate application+ATE_ABORT', 'W_Call incomplete files+SCHEDULE', 'W_Call incomplete files+START', 'A_Incomplete+COMPLETE']
| 4 |    7711 |                   93 | ['A_Accepted+COMPLETE', 'O_Create Offer+COMPLETE', 'O_Created+COMPLETE', 'O_Sent (mail and online)+COMPLETE', 'W_Complete application+ATE_ABORT', 'W_Call after offers+SCHEDULE', 'W_Ca

Select specific execution (by index): 2
Chosen execution path: ['W_Call incomplete files+ATE_ABORT', 'W_Validate application+SCHEDULE', 'W_Validate application+START', 'A_Validating+COMPLETE']

|   | resource |   duration | case                   |
| 0 | User_121 |          0 | Application_1000334415 |
| 1 | User_116 |          0 | Application_100034150  |
| 2 | User_133 |          0 | Application_1000386745 |
| 3 | User_134 |          0 | Application_1000610355 |
| 4 | User_43  |          0 | Application_1000671285 |

Select instance to visualize (by index): 0
Visualizing subgraph of task instance at index 0...
Explore more pattern executions (y/n)?
```

bpic2017_single_ek_pattern4_ex2_inst0.pdf - Adobe Acrobat Pro DC
File Edit View Window Help
Home Tools bpic2017_single_ek... ×
Sign In
1 / 1
101%
Share
121
I4
V1
V2
A9
1000334415
8,58 x 1,81 in

Fig. 2. Example of the CLI and PDF output for exploring task execution pattern 4 in the BPIC'17 data (for sake of space, only a condensed version of the intermediate output is shown).

IV. Conclusion

We developed an open-source command-line tool for exploring task execution patterns in event graphs. In [3], we have shown that the exploration of specific execution patterns that include the behavioral dimensions of both cases and resources, can reveal a complex interplay of cases and actors engaging in recurrent patterns of work, i.e., routines and habits. This makes our tool applicable in any process mining use case where resource information is recorded. Future work is to extend the subgraph visualization feature for all task execution patterns introduced in [3]. Still, we covered a core set of patterns that occur in public datasets. Furthermore, we plan to build an interactive and graphical user interface to enable a seamless interaction with the tool.

References

[1] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management*. Springer, 2 ed., 2018.
[2] S. Esser and D. Fahland, "Multi-dimensional event data in graph databases," *J. Data Semant.*, vol. 10, no. 1, pp. 109–141, 2021.
[3] E. L. Klijn, F. Mannhardt, and D. Fahland, "Classifying and detecting task executions and routines in processes using event graphs," in *BPM Forum 2021*, vol. 427 of *LNBIP*, pp. 212–229, Springer, 2021.