An Experimental Evaluation of Passage-Based Process Discovery

H.M.W. Verbeek and W.M.P. van der Aalst

Technische Universiteit Eindhoven
Department of Mathematics and Computer Science
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
{h.m.w.verbeek,w.m.p.v.d.aalst}@tue.nl

Abstract. In the area of process mining, the ILP Miner is known for the fact that it always returns a Petri net that perfectly fits a given event log. However, the downside of the ILP Miner is that its complexity is exponential in the number of event classes in that event log. As a result, the ILP Miner may take a very long time to return a Petri net. Partitioning the traces in the event log over multiple event logs does not really alleviate this problem. Like for most process discovery algorithms, the complexity is linear in the size of the event log and exponential in the number of event classes (i.e., distinct activities). Hence, the potential gain by partitioning the event classes is much higher. This paper proposes to use the so-called passages to split up the event classes over multiple event logs, and shows what the results are for seven large event logs. The results show that indeed the use of passages alleviates the complexity, but that much hinges on the size of the largest passage detected: The smaller this passage, the better the run time.

Key words: Process Discovery, Fitness, Petri nets, Passages

1 Introduction

In the process mining playground, the area of process discovery is concerned with discovering process models from event logs [1]. The most well-known process discovery algorithm is the α-algorithm [2], which discovers a Petri net from an event log. Under some assumptions, the resulting Petri net is known to fit the original event log, that is, the Petri net is able to replay all traces in the event log successfully. However, if these assumptions do not hold, the resulting Petri net may not even be free of deadlocks.

A process discovery algorithm that always discovers a perfectly fitting Petri net is the so-called ILP Miner [3]. Basically, this algorithm uses Integer Linear Programming (ILP) techniques to check whether or not adding a place with given inputs and given outputs would harm the perfect fitness, i.e., the ability to replay the entire event log. If it would not harm fitness, the place can be added; if it would harm fitness, the place cannot be added. However, the downside of this algorithm is that lots and lots of places need to be checked, as we basically

have to check all combinations of possible inputs and outputs. As a result, the ILP Miner works fine for event logs that contain only a few event classes, but it will take forever in case the number of event classes is even moderate (say, 20 or more).

Recently, the notion of *passages* was introduced to decompose process mining problems [4]. Passages are pairs of sets of event classes. The idea is that process discovery and conformance checking can be done per passage. Instead of having to mine the entire log for a perfectly fitting Petri net, we can mine every passage log for such a Petri net. At the end, we can simply combine the resulting passage Petri nets into a single Petri net for which we know that it perfectly fits the log we started with. As a result this passages technique allows for a possible solution to the complexity problem of the ILP Miner: First we split the log into smaller passage logs, second, we mine every passage log for a passage Petri net using the ILP Miner, third, we combine all passage Petri nets into a single Petri net. Clearly, this could outperform the standard ILP Miner.

This paper aims at evaluating the passage technique introduced in [4] using a number of logs with many event classes, among which the log used for this year's BPI Challenge [5]. First, Section 2 presents related work. Second, Section 3 introduces the concept of passages. Next, Section 4 describes the experimental setup and Section 5 presents the experimental evaluation The latter shows that although the passages alleviate the problem to a large extent, there may still be large passages that are to big for the ILP Miner to handle. Section 6 presents a possible solution to the problem of big passages, which allows the user of the ILP Miner to focus on the most obvious places and to simply ignore the less obvious ones. Finally, Section 7 concludes the paper by wrapping up the results and offering hints for future work.

2 Related Work

For an introduction to process mining we refer to [1]. For an overview of best practices and challenges, we refer to the Process Mining Manifesto [6].

Process discovery, i.e., discovering a process model from a multiset of example traces, is a very challenging problem and various discovery techniques have been proposed [7, 2, 8, 9, 10, 11, 12, 13, 14, 15, 16, 3]. Many of these techniques use Petri nets during the discovery process and/or to represent the discovered model. It is impossible to provide an complete overview of all techniques here. Very different approaches are used, e.g., heuristics [12, 16], inductive logic programming [13], state-based regions [7, 11, 15], language-based regions [9, 3], and genetic algorithms [14]. Classical synthesis techniques based on regions [17] cannot be applied directly because the event log contains only example behavior. For state-based regions one first needs to create an automaton as described in [7]. Moreover, when constructing the regions, one should avoid overfitting. Language-based regions seem good candidates for discovering transition-bordered Petri nets for passages [9, 3]. In fact, in this paper uses the ILP Miner described in [3].

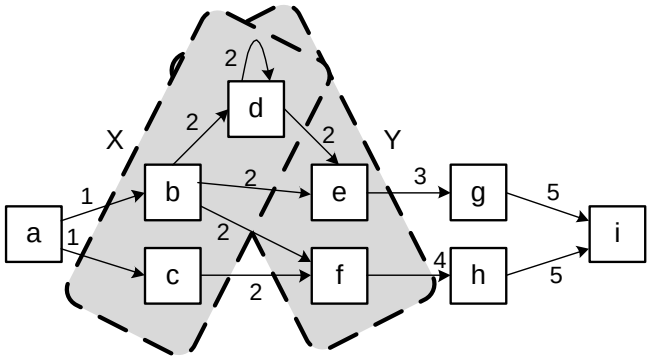

Fig. 1. An example passage in the example log

In [18] various approaches to decompose process mining problems (including passages) are discussed. Passages were introduced in [4] to decompose challenging process discovery and conformance checking problems into smaller problems. Most related to passages are the divide-and-conquer techniques presented in [19]. In [19] it is shown that region-based synthesis can be done at the level of synchronized State Machine Components (SMCs). Also a heuristic is given to partition the causal dependency graph into overlapping sets of events that are used to construct sets of SMCs. Passages provide a different (more local) partitioning of the problem and, unlike [19] which focuses on state-based region mining, we decouple the decomposition approach from the actual conformance checking and process discovery approaches.

In [4] only the theoretical notion of passages (i.e., definition and logical properties) was introduced without providing any experimental results or implementation. This paper describes the implementation of a passage-based discovery techniques and provides experimental results.

3 Passages

The concept of passages has been introduced in [4]. Basically, a passage corresponds to a pair of sets of event classes X and Y such that every event of X is immediately followed by some event of Y (denoted $X\bullet = Y$) and every event of Y is immediately preceded by some event of X (denoted $\bullet Y = X$). As such, the set of passages corresponds to a set of equivalence classes on the direct-follows relation in an event log. Fig. 1 shows an example passage where $X = \{\mathsf{b}, \mathsf{c}, \mathsf{d}\}$ and $Y = \{\mathsf{d}, \mathsf{e}, \mathsf{f}\}$, projected on the directly-follows relation as found in an example log. Note that $X\bullet = Y$ and $X = \bullet Y$, so (X, Y) is indeed a passage. Edges carry numbers to refer to the passage they belong to.

In [4] it has been shown that, provided that the log contains a unique start event and a unique end event, splitting up an event log and a Petri net using passages maintains a perfect fit between the event log and the Petri net. Hence,

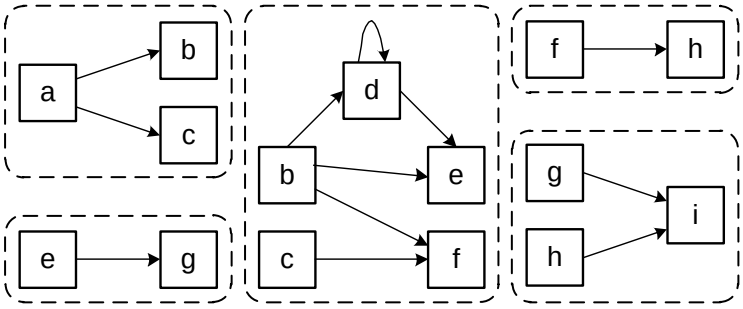

Fig. 2. The five passages found in the example log

if the Petri net perfectly fits the event log, then every passage Petri net perfectly fits the corresponding passage event log, and vice versa. As a result, discovery and conformance checking problems can be decomposed based on the notion of passages. Instead of solving a process mining problem for the whole event logs and model, five smaller problems can be solved (one for each passage), which is shown in Fig. 2.

This paper uses the vice-versa part for discovery:

> *If we split up the event log into passage event logs and discover for every passage event log a perfectly fitting passage Petri net, then we can combine these discovered passage Petri nets into one big Petri net that perfectly fits the original event log.*

Note, however, that to do so we need to extend the event log with two new event classes: one for the unique start event and another for the unique end event. So, initially we make the problem a little bigger, but we hope to be able to handle this little-bigger problem better than the original problem.

As mentioned in [4], the passage-based technique uses two abstract algorithms γ_c and γ_p. The γ_p algorithm is used to discover a Petri net from a passage event log, i.e., the ILP Miner introduced in [3] and implemented in ProM. The γ_c algorithm is used to extract the causal relations from the the event log, that is, a directly-follows relation. Based on these causal relations, the log will be split into passage event logs. To obtain as many small passages as possible (which would alleviate our problem with the ILP Miner the most), we should aim for a causal relations structure that is as sparse as possible while still being connected. For this reason, we will use the Heuristics Miner in such a way that it yields such a structure. Hence, we will accept the default settings for this miner, except for the `Relative-to-best` (which we will set to 0 instead of 5) and the `Dependency` (which we will set to 100 instead of 90) settings.

4 Experimental Setup

The experimental setup contains 7 event logs (A to G), which vary in size. Event log A contains test events for the deployment of high-tech equipment,

Event log	Event classes	Events	Traces
A	720	154,966	24
B	36	262,200	13,087
C	615	53,874	2713
D	96	124,862	130
E	255	67,271	2076
F	5415	612,340	2246
G	15	119,021	14,279

Table 1. Characteristics for the different event logs

Key	Value
Computer	Dell Precision T5400
Processor	Intel® Xeon® CPU, E5430 @ 2.66Ghz (2 processors)
Installed memory (RAM)	16.0 GB
System type	64-bit Windows 7 Enterprise SP 1
JRE	64-bit jdk1.6.0_24
VM arguments	-ea -Xmx4G

Table 2. Basic information on the system used

which contains both in-factory tests and on-site test events. Event log B is the BPI Challenge 2012 event log. Event log C contains diagnostic and treatment events from a hospital department. Event logs D and E contain events from a municipality, where D contains events that correspond to citizens objecting to the valuation of their houses, and E contains events that correspond to citizens that request for building permits. Event log F contains events from a web server. Finally, event log G contains event related to invoices at a provincial office of the Dutch national public works. Table 1 shows the characteristics of these logs.

Each of these logs will be mined using both the standard ILP Miner and the passage-enhanced ILP Miner, where the latter miner uses the Heuristics Miner as described earlier for determining the direct-follows relations. Table 2 shows the characteristics of the system we used to run both miners on.

5 Experimental Evaluation

Table 3 shows the run times we obtained from the standard ILP Miner. For sake of completeness, we mention that the run times have been rounded to the nearest number containing only two relevant digits. For example, the run time in case of log B is rounded from 4620.97 to 4600. Using the standard ILP Miner we did not obtain any results on the A and F logs, as the miner ran out of memory.

Fig. 3 shows the resulting Petri net for the B event log. Clearly, this Petri net is beyond comprehension by the human eye: There are way too many edges to be able to see any structure in this net.

Event log	**Run time** in seconds
A	-
B	4600
C	-
D	45,000
E	110,000
F	-
G	56

Table 3. Run times obtained for the standard ILP Miner

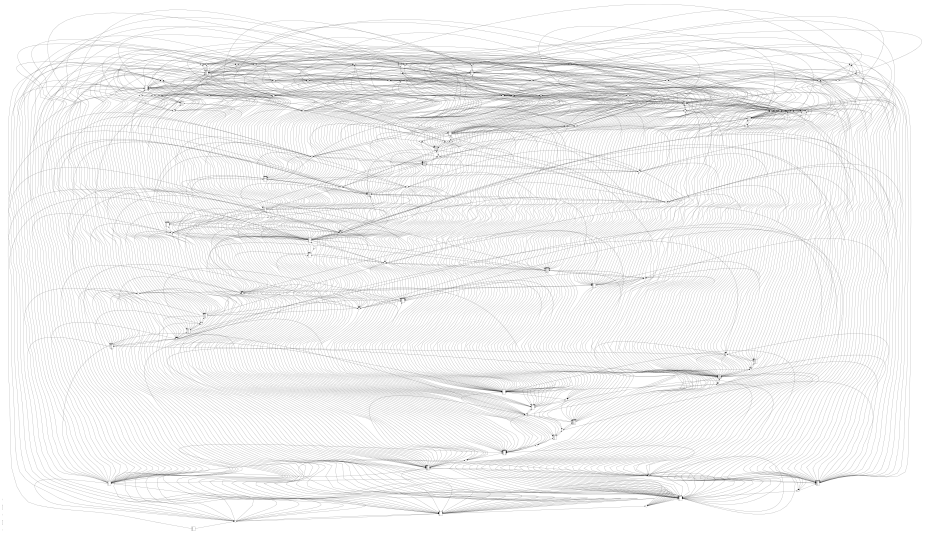

Fig. 3. The Petri net mined from the B event log with the standard ILP Miner.

In contrast, Table 4 shows (among other things) the run times for the passage-enhanced ILP Miner. For example, it shows that the run time in case of the B log has decreased from 4600 to 290 seconds, that 16 passages were detected, that the largest passage contains 13 event classes, and that it took the ILP Miner 92 seconds to mine the Petri net for the largest passage. The run times for the largest passages show that by one does not gain much by running the ILP Miner on different computers for different passages, as the overall run time mainly depends on the run time for the largest passage. Only in case several other passages are about as large as the largest passage, than using different computers might help. Finally, note that this miner also cannot handle the F event log: It simply contains too many event classes to be kept in memory.

These results show that splitting up the event log into many event logs using passages typically helps in reducing the run times, while still resulting in a Petri net that perfectly replays the original event log. It also shows, that the better

Event log	**Run time** in seconds	**Passages** #	**Largest passage** in event classes	**Run time largest passage** in seconds
A	220,000	382	641	210,000
B	290	16	13	92
C	300,000	113	337	230,000
D	15,000	36	45	14,000
E	16,000	94	83	15,000
F	-	-	-	-
G	84	2	16	72

Table 4. Run times (and other characteristics) obtained for the passage-enhanced ILP Miner

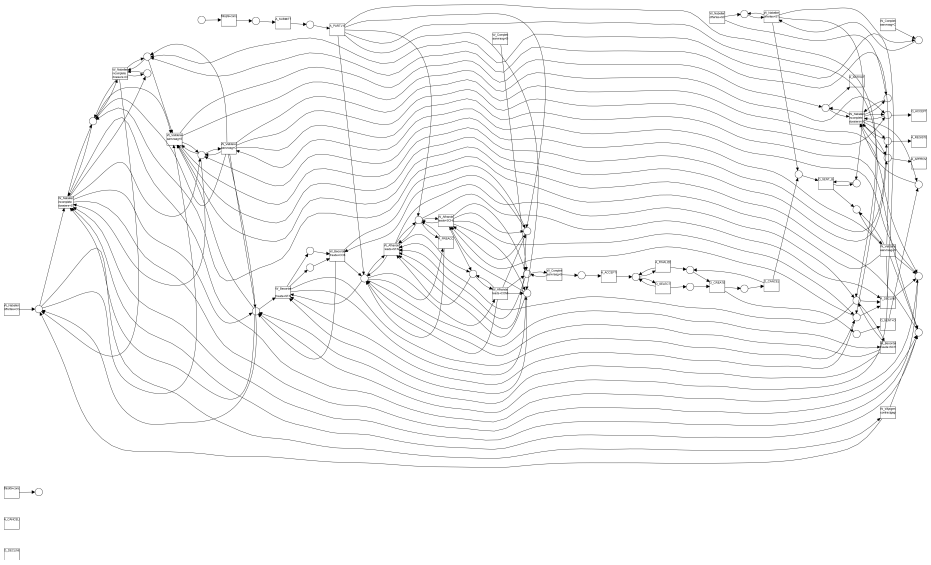

Fig. 4. The Petri net mined from the B event log with the passage-enhanced ILP Miner.

the distribution among the passages is, the better the reduction will be: If some passage is still large in size, the run time will still be large as well. Finally, the case of the G log shows that the passage-enhanced ILP Miner comes with a little overhead (the additional start and end event classes), which may result in worse run times in case the log cannot really be split up into passages.

Fig. 4 shows the resulting Petri net for the B event log. If we compare this Petri net to the Petri net as obtained from the standard ILP Miner (see Fig. 3), we see that comprehension of this Petri net is way better, at the cost (so it seems) of a number of transitions being disconnected. Apparently, trying to connect these transitions to the remainder of the net causes the ILP Miner to create so many edges that the result becomes incomprehensible.

Of course, one could argue that the passage-enhanced ILP Miner takes less run time *because* it produces different results, i.e., a different Petri net. For this sake, we also did a little experiment where we used the standard ILP Miner as both the γ_c algorithm and the γ_p algorithm: If we split up the net as obtained from the standard ILP Miner into passages, run the same miner on every passage, and glue the resulting net into a single Petri net, then we expect that the end result is identical to the result of the initial ILP Miner. However, this requires that the event log at hand contains almost no noise, as the standard ILP Miner translates however little noise into a causal dependency, which typically results in a net that contains only a single passage.

Therefore, we created a model for a paper review system and used the model to create a noise-free event log, called H, which contains 54 event classes, 71,800 events, and 2500 traces. We ran the miner as mentioned above on this event log, and we compared both the resulting Petri nets (both the end result as the result of the γ_c algorithm) and the execution times. For sake of completeness, we do mention that in this case we did not use the default settings for the ILP Miner, as these settings do not punish for improper completion. Instead, we used the default setting with the variant changed to "Petri net (Empty after Completion)". This variant prevents the γ_p ILP Miners from introducing implicit places that contain tokens after completion of the net.

The γ_c ILP Miner took 1300 seconds. From the resulting Petri net, 30 passages were derived of which the largest passage contains 7 event classes. In total, filtering the log for every passage, running the γ_p ILP Miner for every resulting sublog, and synthesizing all 30 subnets into a single Petri net took 140 seconds, and resulted in the same Petri net. This clearly shows that the passage-enhanced ILP Miner can result in the same Petri net as the standard ILP Miner, but using only a fraction of the time.

6 A Possible Relaxation

Obviously, large passages can pose as much as a problem as a large collection of event classes in a log can: No guarantee can be given that we can successfully split the collection of event classes into passages which are small enough. However, we could use the fact that we know how many event classes there will be before we start the ILP Miner on a passage. In case we think that the collection of event classes is still too large (like 641 in case of the A log), we can simply decide not to use the ILP Miner for such a passage, but just to return a Petri net that contains a transition for every event class. This means that we overgeneralize the behavior of large passages, as we allow for any combination of the transitions present in large passage. Intuitively, one could argue that these large passages correspond to difficult causal structures that are hard to comprehend in the first place, so why would the user want to see these complex structures? Instead, it just might be better for the user to see the more simple structures, which can be easily obtained by running the ILP Miner only on those passages that small enough.

Event log	**Run time** in seconds
A	11,000
B	320
C	650
D	420
E	650
F	-
G	85

Table 5. Run times obtained for the passage-enhanced ILP Miner, restricted to 20 event classes

Please note that the standard ILP Miner does not offer this possibility: Either the collection of event classes in the event log is small enough and we will get a connected Petri net, or the collection is too big and we will get a Petri net containing only transitions. The fact that the passage-enhanced ILP Miner can check the number of event classes per passage is obviously useful here.

For this reason, we have extended the experiment with a third miner: A passage-enhanced ILP Miner that only uses the ILP Miner in case the passage at hand contains fewer as 20 event classes (based on the earlier experiments, 20 seems to be still reasonable). Possibly, this miner results in a Petri net that is disconnected, but it is very likely that it will also contain connected parts, and it will still fit the original event log perfectly. Table 5 shows the results. Event log F still contains too many event classes to be handled, while event log A contains a single trace that result in more than 10,0000 event classes for some passages, which explains the exceptional long run time for this log. The remainder of the run times are quite acceptable: in a matter of minutes, the miner is done.

Fig. 5 shows a detail from the resulting Petri net for the A event log, whereas Fig. 6 shows such a detail for the D event log. In contrast, the standard ILP Miner fails to produce a Petri net for the A event log, and it produces a Petri net like shown in Fig. 3 for the D event log. Clearly, the latter does not provide any information at all, whereas the former does provide some information (some relations between event classes are present in the Petri nets). As such, the restricted passage-enhanced ILP Miner can provide useful information within minutes, whereas the standard ILP Miner would take days and provide something that is (almost) useless.

7 Conclusions

In this paper, we showed that passages can help the ILP Miner in finding a Petri net that perfectly fits a given event log. First of all, the run time of the ILP Miner is typically reduced dramatically when the passages are small. Only in a single case the run time was increased, which was caused by the fact that one two passages were detected: a simple one containing only two event classes,

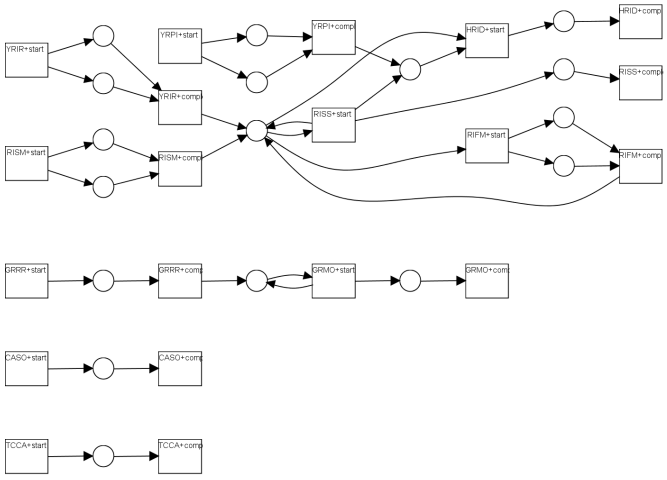

Fig. 5. A detail from the Petri net mined from the A event log with the restricted passage-enhanced ILP Miner.

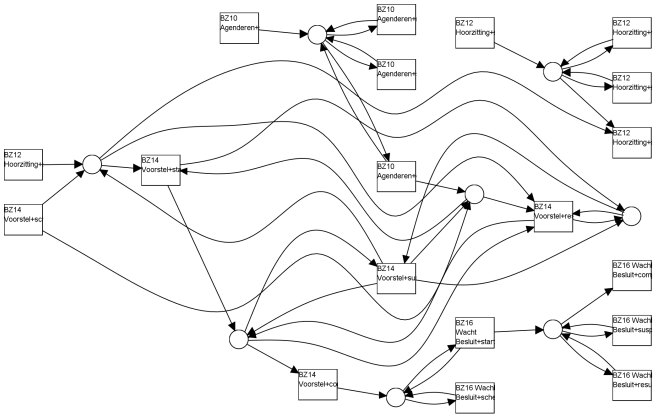

Fig. 6. A detail from the Petri net mined from the D event log with the restricted passage-enhanced ILP Miner.

and a complex one containing 16 event classes, which include all 15 original event classes and the artificial end event class. In this case, the use of passages only made the problem slightly bigger. Nevertheless, four out of seven event logs show that the use of passages typically reduce the run times by factors. Note that in the remaining two event logs, nor the standard ILP Miner nor the passage-enhanced ILP Miner were able to find a Petri net. In one case, the number of event classes was simply too big to handle, in the size of the largest passage simply prevented the ILP Miner from finding a Petri net for this passage.

Moreover, we showed that the quality of the resulting Petri net may also improve greatly. In one out of seven event logs, the standard ILP Miner returns an incomprehensible Petri net, while the passage-enhanced returns a comprehensible one.

Finally, we showed that by adding a restriction on the size of the passages, the run times of the passage-enhanced ILP Miner can even be reduced further, although his typically results in disconnected Petri nets. Where this latter can be seen as a downside, it also can be regarded as positive, as the disconnected parts might offer the domain expert the information he needs. As a result, there seems to be a possible trade-off between run time and precision, while keeping the fitness at a perfect level: The further we restrict the number of event classes in passages (which means that passages that exceed this restriction will not be mined for a Petri net), the more disconnected (and the less precise) the resulting Petri net will be, but the faster the ILP Miner will finish. We could even think of extending the passage-enhanced ILP Miner with a certain time limit: It will only consider the smallest passages, and while the time limit still permits, it will also consider the smallest of the unconsidered passages as well.

Another option for future research is to remove causal relations while turning the causal structure into passages. For example, if the removal of a single, infrequent, causal relation would cause the largest passage to break apart into multiple passages, then it might be worthwhile to indeed remove this relation. As such, we need to consider the entire causal structure that results from the Heuristics Miner, which fortunately comes with such frequencies.

References

1. Aalst, W.M.P.v.d.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. Aalst, W.M.P.v.d., Weijters, A.J.M.M., Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering **16**(9) (2004) 1128–1142
3. Werf, J.M.E.M.v.d., Dongen, B.F.v., Hurkens, C.A.J., Serebrenik, A.: Process discovery using integer linear programming. Fundam. Inform. **94**(3-4) (2009) 387–412
4. Aalst, W.M.P.v.d.: Decomposing process mining problems using passages. In Haddad, S., Pomello, L., eds.: Applications and Theory of Petri Nets 2012. Volume 7347 of Lecture Notes in Computer Science., Springer-Verlag, Berlin (2012) 72–91
5. Dongen, B.F.v.: BPI challenge 2012. Dataset. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f (2012)
6. IEEE Task Force on Process Mining: Process mining manifesto. In Daniel, F., Dustdar, S., Barkaoui, K., eds.: Business Process Management 2011 Workshops. Volume 99 of Lecture Notes in Computer Science., Springer-Verlag, Berlin (2011) 169–194
7. Aalst, W.M.P.v.d., Rubin, V., Verbeek, H.M.W., Dongen, B.F.v., Kindler, E., Günther, C.W.: Process mining: A two-step approach to balance between underfitting and overfitting. Software and Systems Modeling (SoSyM) **9**(1) (2010) 87–111

8. Agrawal, R., Gunopulos, D., Leymann, F.: Mining Process Models from Workflow Logs. In: Sixth International Conference on Extending Database Technology. (1998) 469–483
9. Bergenthum, R., Desel, J., Lorenz, R., Mauser, S.: Process Mining Based on Regions of Languages. In Alonso, G., Dadam, P., Rosemann, M., eds.: International Conference on Business Process Management (BPM 2007). Volume 4714 of Lecture Notes in Computer Science., Springer-Verlag, Berlin (2007) 375–383
10. Carmona, J., Cortadella, J.: Process mining meets abstract interpretation. In: Proceedings of the 2010 European conference on Machine learning and knowledge discovery in databases: Part I. ECML PKDD'10, Berlin, Springer-Verlag, Berlin (2010) 184–199
11. Carmona, J., Cortadella, J., Kishinevsky, M.: A region-based algorithm for discovering petri nets from event logs. In Dumas, M., Reichert, M., Shan, M.C., eds.: Business Process Management. Volume 5240 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2008) 358–373 10.1007/978-3-540-85758-7_26.
12. Cook, J.E., Wolf, A.L.: Discovering Models of Software Processes from Event-Based Data. ACM Transactions on Software Engineering and Methodology **7**(3) (1998) 215–249
13. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. J. Mach. Learn. Res. **10** (June 2009) 1305–1340
14. Medeiros, A.K.A.d., Weijters, A.J.M.M., Aalst, W.M.P.v.d.: Genetic Process Mining: An Experimental Evaluation. Data Mining and Knowledge Discovery **14**(2) (2007) 245–304
15. Solé, M., Carmona, J.: Process mining from a basis of state regions. In: Applications and Theory of Petri Nets 2010, Berlin, Springer-Verlag, Berlin (2010) 226–245
16. Weijters, A.J.M.M., Aalst, W.M.P.v.d.: Rediscovering Workflow Models from Event-Based Data using Little Thumb. Integrated Computer-Aided Engineering **10**(2) (2003) 151–162
17. Darondeau, P.: Unbounded Petri Net Synthesis. In Desel, J., Reisig, W., Rozenberg, G., eds.: Lectures on Concurrency and Petri Nets. Volume 3098 of Lecture Notes in Computer Science., Springer-Verlag, Berlin (2004) 413–438
18. Aalst, W.M.P.v.d.: Distributed process discovery and conformance checking. In de Lara, J., Zisman, A., eds.: Fundamental Approaches to Software Engineering. Volume 7212 of Lecture Notes in Computer Science. Springer-Verlag, Berlin (2012) 1–25 10.1007/978-3-642-28872-2_1.
19. Carmona, J., Cortadella, J., Kishinevsky, M.: Divide-and-conquer strategies for process mining. In: Proceedings of the 7th International Conference on Business Process Management. Business Process Management 2009, Berlin, Springer-Verlag, Berlin (2009) 327–343