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. Like for most process discovery algorithms, its complexity is linear in the size of the event log and exponential in the number of event classes (i.e., distinct activities). As a result, the potential gain by partitioning the event classes is much higher than the potential gain by partitioning the traces in the event log over multiple event logs. This paper proposes to use the so-called passages to split up the event classes over multiple event logs, and shows the results are for seven large real-life event logs and one artificial event log: The use of passages indeed alleviates the complexity, but much hinges on the size of the largest passage detected.

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

1 Introduction

Process discovery, i.e., learning a process model from an event log, remains the most challenging process mining task [1]. The best-known process discovery algorithm is the α-algorithm [2]. Under some assumptions, this algorithm results in a Petri net that fits the original event log. However, if these assumptions do not hold, this Petri net may not even be free of deadlocks.

An example of a process discovery algorithm that always discovers a perfectly fitting Petri net is the so-called ILP Miner [3]. This algorithm uses Integer Linear Programming (ILP) techniques to check whether or not adding a place would harm perfect fitness. However, the downside of this algorithm is that lots of places need to be checked, as we 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 may require excessive computation time 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]. Instead of having to mine the entire log for a perfectly fitting Petri net, we can partition the log into several smaller passage logs and mine every passage log for 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 entire log. As a result, passage-based decomposition may help to address the complexity problem of the ILP Miner.

This paper aims at evaluating the passage technique introduced in [4] using a number of large real-life logs with many event classes. For details on the passages, we refer to [4]. For a more detailed version of this paper, please see [5]. Here, it suffices to mention that the passage technique requires the addition of a unique start event and a unique end event, and that it allows for two abstract algorithms γ_c and γ_p, where γ_c is used to detect causal relations (which underly the different passage logs) between event classes, and γ_p is used to mine a Petri net from a single passage log.

The remainder of this paper is organized as follows. First, Section 2 describes the experimental setup and Section 3 presents the experimental evaluation. The latter shows that although the passages alleviate performance problems to a large extent, there may still be large passages that are to big for the ILP Miner to handle. Section 4 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 5 concludes the paper by wrapping up the results and offering hints for future work.

2 Experimental Setup

The initial experimental setup contains 7 real-life event logs (A to G). Table 1 shows the characteristics of these logs. A contains test events for the deployment of high-tech equipment, which contains both in-factory tests and on-site test events. B is the BPI Challenge 2012 event log [6]. C contains diagnostic and treatment events from a hospital department. 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. F contains events from a web server. Finally, G contains events related to invoices at a provincial office of the Dutch national public works.

Each of these logs will be mined using both the standard ILP Miner[1] and the passage-enhanced ILP Miner[2] as they are implemented in ProM 6.2[3]. Table 2 shows the characteristics of the system we used to run both miners on.

3 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

[1] The `ILP Miner` plug-in from the `ILPMiner` package with default settings.

[2] The `Mine Petri net using Passages` plug-in from the `Passage` package with γ_c set to `Heuristics Miner` (`Relative-to-best` set to 0 and `Dependency` set to 100) and γ_p set to `ILP Miner`.

[3] ProM 6.2 can be downloaded from `www.processmining.org`.

number containing only two relevant digits. For example, the run time for B is rounded from 4620.97 to 4600. The standard ILP Miner ran out of memory for A and F and ran out of time (more than a week) for C.

In contrast, Table 5 shows (among other things) the run times for the passage-enhanced ILP Miner. For example, it shows that the run time for B 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 we do 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 if several other passages are about as large as the largest passage, then using different computers might help. Finally, note that this miner also cannot handle F: 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 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, G 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.

Of course, one could argue that the passage-enhanced ILP Miner takes less run time because it produces different results, i.e., different Petri nets. Therefore, we also conducted experiments where we used the *proper completion* ILP Miner as both the γ_c algorithm and the γ_p algorithm[4]: If we split up the net as obtained from the proper completion 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 ILP Miner translates noise into causal dependencies, 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.

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

[4] The `Mine Petri net using Passages` plug-in from the `Passage` package with γ_c set to `Flower and ILP Miner with Proper Completion` and γ_p set to `ILP Miner with Proper Completion`.

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

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

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

Table 4. Run times obtained for the passage-enhanced ILP Miner, restricted to 20 event classes

Event log	**Run time**	**Passages**	**Largest passage**	**Run time largest passage**
	in seconds	#	in event classes	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 5. Run times (and other characteristics) obtained for the passage-enhanced ILP Miner

ILP Miner can produce the same Petri net as the standard ILP Miner, but using only a fraction of the time.

4 A Possible Relaxation

Obviously, large passages can pose as much as a problem as a large collection of event classes in a log can. 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 (641 for A), 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 are small enough.

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 maximum passage size of 20: A passage-enhanced ILP Miner that only uses the ILP Miner in case the passage at hand contains less than 20 event classes (based on the earlier experiments, 20 seems to be still reasonable)[5]. 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 4 shows the results. F still contains too many event classes to be handled, while A contains a trace that results in more than 10,0000 events for some passages, which explains the exceptional long run time for this log. The other run times are quite acceptable: in a matter of minutes, the process is discovered.

5 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. For two logs (A and C), the passage-enhanced ILP Miner finds a Petri net, where the standard ILP Miner did not. For three logs (B, D, and E), the passage-enhanced ILP Miner performed

[5] The `Mine Petri net using Passages` plug-in from the `Passage` package with γ_c set to `Heuristics Miner` (`Relative-to-best` set to 0 and `Dependency` set to 100), γ_p set to `ILP Miner`, and `max size` set to 20.

way better than the standard ILP Miner. For one log (F), both the passage-enhanced ILP Miner and the standard ILP Miner ran out of memory because of the huge number of event classes. For one log (G), the passage-enhanced ILP Miner performed worse than the standard ILP Miner. This is explained by the fact that size of the largest passage exceeds the size of the original net, which is possible as the passage technique requires the addition of a unique start event and a unique end event.

We also 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. In some cases, this 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. To do so, we can consider the causal structure produced by the Heuristics Miner, which provides 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. Verbeek, H.M.W., Aalst, W.M.P.v.d.: An experimental evaluation of passage-based process discovery. BPM Center Report BPM-12-14, BPMcenter.org (2012)
6. Dongen, B.F.v.: BPI challenge 2012. Dataset. http://dx.doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f (2012)