Leveraging Super-Scalarity and Parallelism to Provide Fast Declare Mining without Restrictions

Michael Westergaard[1,2⋆] and Christian Stahl[1]

[1] Department of Mathematics and Computer Science,
Eindhoven University of Technology, The Netherlands
[2] National Research University Higher School of Economics, Moscow, 101000, Russia
m.westergaard@tue.nl, c.stahl@tue.nl

Abstract. UnconstrainedMiner is a tool for fast and accurate mining Declare constraints from models without imposing any assumptions about the model. Declare models impose constraints instead of explicitly stating event orders. Constraints can impose various choices and ordering of events; constraints typically have understandable names, but for details, refer to [5]. Current state-of-the-art mining tends to fail due to a computational explosion, and employ filtering to reduce this. Our tool is not intended to provide user-readable models, but instead to provide all constraints satisfied by a model. This allows post-processing to weed out uninteresting constraints, potentially obtaining better resulting models than making filtering beforehand out of necessity. Any post-processing (and complexity-reducing filtering) possible with existing miners is also possible with the UnconstrainedMiner; our miner just allows more intelligent post-processing due to having more information available, such as interactive filtering of models. In our demonstration, we show how the new miner can handle large event logs in short time, and how the resulting output can be imported into Excel for further processing. Our intended audience is researchers interested in Declare mining and users interested in abstract characterization of relationships between events. We explicitly do not target end-users who wish to see a Declare model for a particular log (but we are happy to demonstrate the miner on other concrete data).

Processes that are not perfectly understood or have little structure, are often easier modeled using a declarative rather than a classical imperative approach. In a declarative approach, constraints between tasks are described rather than for each task specifying the next task to execute. Declarative modeling is a more recent and less matured approach, which has so far not found widespread application in industry yet. Declare [7] is an emerging language studied in academia over the last decade.

Existing miners. Typically the complexity of checking a constraint is $O\left(m \cdot \binom{n}{k} \cdot k!\right)$, where m is the number of traces in the log, n is the number of different events in the entire log, and k is the number of parameters to the constraint (number of ways to assign n events to k ordered parameters – ${}_nP_k = \binom{n}{k} \cdot k!$ – times the number of traces k).

⋆ Support from the Basic Research Program of the National Research University Higher School of Economics is gratefully acknowledged.

For the BPI challenge log from 2012[3] [1] the parameters are $m = 13,087$ and $n = 24$, yielding from $314,088$ ($k = 1$) to $66,749,981,760$ ($k = 5$) checks. ProM 6 has a Declare miner which systematically checks each constraint and returns a model comprising constraints satisfied for a certain percentage of traces. Due to the complexity, this miner employs a couple of tricks, including forcing users to pick among interesting constraints [3], employing a priori reduction to avoid considering rarely occurring events [3], and considering the relationships between constraints to avoid mining some constraints [4]. Even so, the ProM Declare miner did not mine a single constraint such as succession in 24 hours.

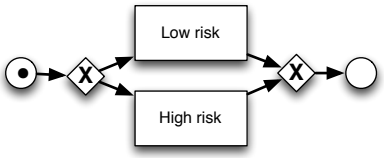

Fig. 1: Model with a branch.

The assumption that rarely occurring events need not be checked is crucially dependent on the notion of *support*, i.e., that a constraint is only interesting if it is triggered. This is problematic in a case such as Fig. 1. There we have a hypothetical XOR-split and -join. If the top path is chosen, constraints in the bottom part are never triggered, so their support is low, even though those are arguably more interesting. If we want to mine without enforcing high support, we cannot use a priori reduction. Exploitation of relationships between constraints is problematic if we want to display a simpler model, e.g., by removing constraints redundant due to transitivity. The problem here is that, generally, support of a removed constraint cannot be derived from support of the remaining constraints.

The MINERful miner [2] uses regular expressions and global computations to mine constraints. First statistics are computed for the log and subsequently constraints are mined from these. The MINERful miner can mine all constraints for the 2012 BPI challenge log in approximately 26 seconds, but only supports a subset of all constraints. Computation of constraints from statistics makes it difficult to add new constraints, as it is necessary to develop and prove rules for doing so. For example, it is far from obvious how to extend this approach to also mine choices.

Removing all assumptions. Instead, we prefer to mine first and, subsequently, filter with full information. Not only does this avoid the problem of support and branches, it allows us full knowledge of all constraints, so we can remove redundant constraints more intelligently using simple patterns. Finally, this allows us to provide users with a slider and let them interactively (and locally) balance between simplicity and support.

Efficient preprocessing. We first make sure that our base mining algorithm is efficient. We transform the log into a simple array of arrays of identifiers instead of actual events. Declare constraints have a (finite) LTL semantics, which can be represented as a finite automaton. Using a precomputed mapping between event identifiers and automaton labels, we obtain a fast and memory-efficient replay of the log. We can mine most of Declare's 34 constraints (excluding 2 with four or more parameters) on the BPI challenge log in 249 seconds using 12 MB of memory (including storing the log and all

[3] We use the 2012 log as the 2013 log is much simpler with $819 \leq m \leq 7554$ and $3 \leq n \leq 4$.

results). We mine the same set of constraints as MINERful in 32 seconds (vs 26). This is done automatically, making it easy to add constraints by entering their LTL semantics.

Symmetry reduction. We observe that some Declare constraints are symmetric in their parameters; this is, for example, the case for the choice 1 of 2 constraint. By only checking these constraints for one ordering of parameters, we halve the number of checks (and time needed). In general, this reduces the checking complexity to $O\left(m \cdot \binom{n}{k} \cdot k! \div \prod_{sg \in SG} |sg|!\right)$, i.e., divided by the faculty of the size of each symmetry group $sg \in SG$. The number we divide by tends to get larger exactly when the traditional approach has problems (i.e., when k is large). For example, the most complex standard Declare constraint is choice 1 of 5, which yields the aforementioned 67 billion checks for the BPI challenge log. As this constraint is symmetric in all parameters, this is reduced to $556,249,848$ checks. Our implementation can mine *all* 34 Declare constraints for the BPI challenge log in 287 seconds (so the reduction allows us to check all constraints in the same amount of time we could only check the easy ones before). We now tie with MINERful at 26 seconds (the reason for less reduction is that MINERful does not handle any constraints where the reduction is large). For the standard constraints, we never have more than two symmetry groups, which means that as number of parameters go up, so does the reduction. We compute the symmetry reduction automatically, making this compatible with any added constraint.

Parallelization. As our miner is simple, we do not need any global computations so far. This means our miner is easy to parallelize. We can parallelize in the number of constraints or the number of traces. With 34 constraints, we can mine each individually. Declare constraints are not equal however, and indeed the choice 1 of 5 constraint is responsible for more than 50% of the total time spent on the BPI challenge log, making this strategy less compelling. As we generate the parameters for the mining deterministically, we can have each computation thread take care of this on its own, alleviating this (but making mining more complex). A simpler approach, and the one we have chosen to go with, is to split the log and have each computation thread handle part of the log. As we have abolished a priori reduction, we do not need any central coordination in this case, and our overhead is so small (< 1 second) we can just run the full mining process for each part of the log. Our results can be directly added, making mining scalable nearly infinitely (with constant overhead for preprocessing and aggregation). This has the added benefit of allowing each thread to use less memory (it only needs to store its own part of the log). Employing this allows us to mine all constraints of the BPI challenge log in 174 seconds (using two cores; the reason scalability is not completely linear is due to the feature Turbo Boost on modern CPUs allowing one core on a multi-core system to run at extra speed if other cores are inactive). We beat MINERful at 19 seconds (vs 26). We have implemented this in a parallel setting (i.e., multiple CPU cores on a single machine), but it is equally applicable in a distributed setting as synchronization is only necessary for aggregating results at the end.

Super-scalar mining. Using the automaton representation from [6], it is possible to check more than one constraint at a time. This comes at a cost of memory as the combined automaton is larger than the individual automata, sometimes even exponentially

so, though in practise the size is always less than 1,000 states. We call this super-scalar mining, using the term from CPUs where it refers to the ability to execute multiple instructions simultaneously. We do not wish to break our symmetry reduction from before. Thus, we build a directed acyclic graph with constraints for nodes and arcs from one constraint to another if the first implies the second holds. We then group by taking each leaf and iteratively grouping it with all predecessors and successors not extending the set of parameters and not having incompatible symmetry groups (e.g., splitting them up so $[[A], [B]]$ is not compatible with $[[A, B]]$—it would split up the symmetry group—but the other way around does hold—the symmetry group is already split). The rationale is that adding new parameters increases complexity as does splitting symmetry groups. Adding a constraint with larger symmetry groups does not increase complexity, while the constraint with larger symmetry groups could be checked more efficient on its own, adding to the super-scalar group comes at no extra cost. We check all such maximal groups. We compute this automatically and tailor it to any constraints selected and any new constraints added. Using super-scalar mining with parallelism, we can mine the entire BPI log in only 57 seconds and even without parallelism, we can mine the entire log in just 92 seconds. We beat MINERful on their constraints at just over 4 seconds, an improvement of a factor 6 over 26 seconds. On a server with 8 slower processing cores, MINERful still runs in 26 seconds, whereas UnconstrainedMiner runs in 30 seconds on the entire set of constraints. Using hyper-threading, the UnconstrainedMiner and MINERful tie in running time with the UnconstrainedMiner mining more and more complex constraints. The gain obtained by super-scalar mining is independent of the log.

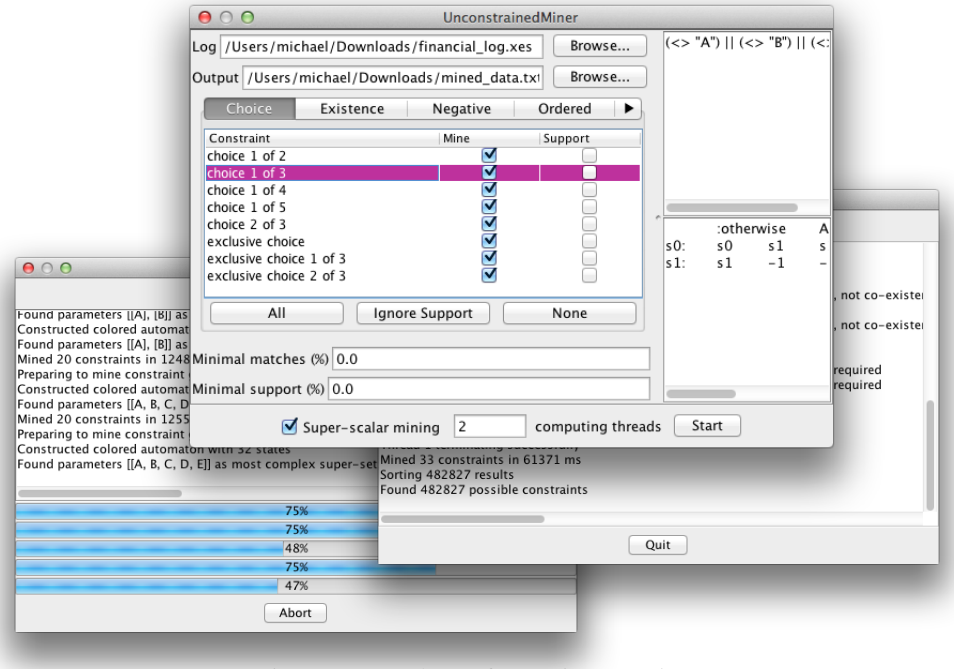

Fig. 2: Screen shots of our miner running.

Usage. In Fig. 2 (top) we see a screen-shot from the configuration of the mining process. All options are saved between runs, making mining from scratch a breeze; the number of threads is computed automatically as the number of cores in the local computer. We can manually inspect the LTL semantics and constraint automaton at the right. In Fig.2 (bottom left) we see the tool in the process of mining. The top progress shows overall progress, and the next progress bars show overall and individual constraint progress for each computation thread. The text area is continuously updated with information about what each thread is doing. At Fig. 2 (bottom right), we see the end result along with total processing time. The end result is a list of all mined constraints, which can be imported in Excel for further processing, as shown in Fig. 3.

Maturity, availability, screencast. Our miner is very new but so far very robust, and uses the same underlying library as the Declare tool (for representing automata) and ProM (for reading logs). The tool is written in Java and runs on all platforms supported by Java. We are currently employing the tool to construct hybrid models containing both declarative constraints and imperative modeling in a block-structured manner.

Our miner is available from `tinyurl.com/declareminer` along with source code. That page also contains a screen cast of simple use of the tool.

References

1. DOI: 10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f
2. Di Ciccio, C., Mecella, M.: A two-step fast algorithm for the automated discovery of declarative workflows. In: CIDM 2013. IEEE (2013)
3. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: Efficient discovery of understandable declarative process models from event logs. In: CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer (2012)
4. Maggi, F.M., Bose, R.P.J.C., van der Aalst, W.M.P.: A knowledge-based integrated approach for discovering and repairing declare maps. In: CAiSE 2013. LNCS, Springer (2013)
5. Pesic, M.: Constraint-Based Workflow Management Systems: Shifting Controls to Users. Ph.D. thesis, Beta Research School for Operations Management and Logistics, Eindhoven (2008)
6. Westergaard, M.: Better Algorithms for Analyzing and Enacting Declarative Workflow Languages Using LTL. In: Proc. of BPM. LNCS, vol. 6896, pp. 83–98. Springer (2011)
7. Westergaard, M., Maggi, F.M.: Declare: A Tool Suite for Declarative Workflow Modeling and Enactment. In: Business Process Management Demonstration Track (BPMDemos 2011). CEUR Workshop Proceedings, vol. 820. CEUR-WS.org (2011)

	A	C	D	E	F	G	H	I	J	K	L	M
1	constraint	matches	support	left initial	entered una	returned to i	is initial at er	parameter1	parameter2	parameter3	parameter4	parameter5
2	response	13087	13087	13087	13087	0	13087	A_SUBMITTE	A_PARTLYSUBMITTED			
3	alternate suc	13087	13087	13087	13087	0	13087	A_SUBMITTE	A_PARTLYSUBMITTED			
4	chain succes	13087	13087	13087	0	13087	13087	A_SUBMITTE	A_PARTLYSUBMITTED			
5	alternate pre	13087	13087	13087	13087	0	0	A_SUBMITTE	A_PARTLYSUBMITTED			
6	alternate pre	13087	7635	13087	7635	5452	0	A_SUBMITTE	A_DECLINED			
7	alternate pre	13087	7635	13087	7635	5452	0	A_PARTLYSU	A_DECLINED			
8	alternate pre	13087	7367	13087	7367	5720	0	A_SUBMITTE	A_PREACCEPTED			

Fig. 3: The mining results imported into Excel.