A Knowledge-Based Integrated Approach for Discovering and Repairing Declare Maps

Fabrizio M. Maggi, R.P. Jagadeesh Chandra Bose, and Wil M.P. van der Aalst

Eindhoven University of Technology, The Netherlands
{f.m.maggi,j.c.b.rantham.prabhakara,w.m.p.v.d.aalst}@tue.nl

Abstract. Process mining techniques can be used to discover process models from event data. Often the resulting models are complex due to the variability of the underlying process. Therefore, we aim at discovering *declarative* process models that can deal with such variability. However, for real-life event logs involving dozens of activities and hundreds or thousands of cases, there are often many potential constraints resulting in cluttered diagrams. Therefore, we propose various techniques to prune these models and *remove constraints that are not interesting or implied by other constraints*. Moreover, we show that *domain knowledge* (e.g., a reference model or grouping of activities) can be used to guide the discovery approach. The approach has been implemented in the process mining tool ProM and evaluated using an event log from a large Dutch hospital. Even in such highly variable environments, our approach can discover *understandable* declarative models.

Keywords: Process Discovery, Model Repair, Linear Temporal Logic, Declare.

1 Introduction

Imperative process models such as BPMN, UML ADs, EPCs, and Petri nets are often used to design and enact operational processes. The procedural nature of such models also helps to guide users by showing "What's next?". Unfortunately, imperative process models are less appropriate for "turbulent" environments characterized by the terms *variability* and *flexibility*. Consider, for instance, a physician in a hospital that requires flexibility to take into account individual characteristics of a patient. But, also physicians have to follow regulations and guidelines and may only deviate for good reasons. In such cases, *declarative* process models are more effective than the imperative ones [15,12,2].

Instead of explicitly specifying all possible sequences of activities in a process, declarative models implicitly specify the allowed behavior using constraints, i.e., rules that must be followed during execution. In comparison to imperative approaches, which produce *closed* models ("what is not explicitly specified is forbidden"), declarative languages are *open* ("everything that is not forbidden is allowed"). In this way, models offer flexibility and still remain compact.

Declarative languages have been successfully applied in the context of process discovery [7,5,6,11,10]. *Declare*, a declarative process modeling language based on LTL (Linear Temporal Logic) [13], was introduced in [2].[1] Declare is characterized by a user-friendly graphical representation and formal semantics grounded in LTL. A *Declare map* is a set of Declare constraints each one with its own graphical representation and LTL semantics (the constraints used in this paper are introduced in Table 1, see [2] for a full overview of Declare).

Declare maps are interesting in the context of process mining [1]. One can discover Declare maps from event logs (extracted from audit trails, transaction logs, and databases) without preexisting models and knowledge [10]. It is also fairly easy to check conformance of an event log with respect to Declare model and diagnose deviations and bottlenecks. The unstructured nature of many real-life processes —demonstrated by variability in event logs— suggests using declarative models for process mining. However, when discovering a Declare map from an event log, there are often too many candidate constraints. Showing all possible constraints often results in a cluttered Declare map. The number of constraints in a Declare map can be reduced by identifying only those constraints that are the most interesting for the user. As proposed in [10], the "interestingness" of a constraint can be defined using association rules metrics such as support and confidence. This paper proposes two sets of techniques to further improve the relevance of discovered Declare maps: (1) techniques to prune discovered process maps using various reduction and simplification rules and (2) techniques using apriori domain knowledge.

A constraint between two activities (e.g., A is eventually followed by B) is redundant if a stronger constraint holds (e.g., A is directly followed by B). There may also be constraints that are implied by other constraints (e.g., if A is followed by B and B is followed by C, then A is also followed by C). *By selectively removing such redundant constraints, we can simplify the model without losing information.*

Furthermore, using domain knowledge, activities in the log can often be grouped in different categories. Hence, it is possible to focus the discovery only on constraints involving activities belonging to the same group (*intra-group constraints*) or on constraints involving activities belonging to different groups (*inter-group constraints*). There may also be a *reference map* providing another source of domain knowledge. Instead of discovering a Declare map from scratch, it is possible to generate a new map by *repairing* the reference map. The reference map is modified using information retrieved from the event log, e.g., existing constraints are strengthened, weakened or refuted and important missing constraints are added.

The paper is structured as follows. In Section 2, we briefly introduce the Declare language and the Declare maps discovery approach proposed in [10]. In Section 3, we describe how we remove redundant constraints to create simpler Declare maps. Here, we also explain how to use domain knowledge for discovering and repairing Declare maps. The proposed techniques can be integrated in a general framework as shown in Section 4. In Section 5, we validate our approach using a case study in a Dutch hospital. Section 6 concludes the paper.

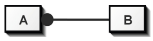
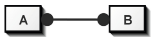
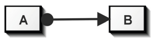
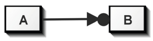
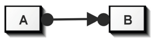
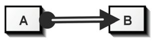
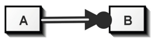
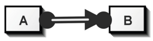
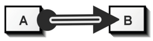
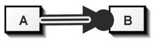
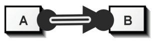
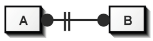
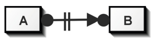
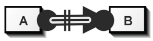
[1] In the remainder, LTL refers to the version of LTL tailored towards finite traces [9].

2 Preliminaries

Table 1 shows the graphical notation and the meaning of the Declare constraints used in this paper. Consider, for example, the *response* constraint. This constraint indicates that if A occurs, B must eventually follow. Therefore, this constraint is satisfied for traces such as $t_1 = \langle A, A, B, C \rangle$, $t_2 = \langle B, B, C, D \rangle$, and $t_3 = \langle A, B, C, A, B \rangle$, but not for $t_4 = \langle A, B, A, C \rangle$ because, in this case, the second A is not followed by a B.

In [10], the authors use the seminal *apriori* algorithm introduced in [3] for discovering Declare maps. In this way, only constraints involving frequent activities are taken into consideration. The authors show that the apriori-based approach significantly improves the computational complexity and accuracy of the uncovered Declare maps with respect to the brute force approach where all

Table 1. Graphical notation and textual description of some Declare constraints

Constraint	**Meaning**	**Graphical representation**
responded existence	if A occurs then B occurs before or after A	
co-existence	if A occurs then B occurs before or after A and vice versa	
response	if A occurs then eventually B occurs after A	
precedence	if B occurs then A occurs before B	
succession	for A and B both precedence and response hold	
alternate response	if A occurs then eventually B occurs after A without other occurrences of A in between	
alternate precedence	if B occurs then A occurs before B without other occurrences of B in between	
alternate succession	for A and B both alternate precedence and alternate response hold	
chain response	if A occurs then B occurs in the next position after A	
chain precedence	if B occurs then A occurs in the next position before B	
chain succession	for A and B both chain precedence and chain response hold	
not co-existence	A and B cannot occur together	
not succession	if A occurs then B cannot eventually occur after A	
not chain succession	if A occurs then B cannot occur in the next position after A	

activities in the log are considered. In the same approach, the constraints in the discovered map with a support lower than a given threshold are removed. The *support* of a constraint is evaluated based on the number of traces where the constraint is *non-vacuously satisfied* [8]. We explain vacuity using an example. Consider again the *response* constraint and the traces mentioned above. The constraint is satisfied in t_1, t_2 and t_3. However, in t_2, the *response* constraint is satisfied in a trivial way because A never occurs. In this case, we say that the constraint is *vacuously satisfied.* The validity of a constraint is, therefore, more significant if there is a high percentage of traces where the constraint is non-vacuously satisfied.

3 Pruning Discovered Maps

Declare maps can be discovered by smartly testing all possible constraints. However, as Section 3.1 shows this may result in an explosion of discovered constraints. In Section 3.2, we illustrate how redundant and "less relevant" constraints can be pruned out to reduce the number of constraints in a map and improve its readability. In Section 3.3, we explain how domain knowledge can help guiding the discovery process towards the identification of the most interesting constraints.

3.1 The Problem of Discovering Too Many Constraints

The *support* measure assessing the significance of a constraint (i.e., the fraction of traces satisfying the constraint non-vacuously) is not robust enough to prune constraints. We have discovered Declare maps on several event logs (both synthetic as well as real-life logs) and our experiences show that the number of discovered constraints is often beyond human comprehension. Table 2 depicts the number of Declare constraints discovered for several logs and varying degrees of support. We see that even for very high support values, the number of discovered constraints is too high to be visualized in a single understandable diagram.

Moreover, several of these constraints are considered to be trivial by domain experts. Analysts are more interested in finding patterns that are surprising and interesting. There is a need for robust methods for discovering such non-trivial

Table 2. Number of discovered constraints for varying support thresholds across different event logs (log D is described in Section 5)

Log	**#Cases**	**#Event classes**	**#Events**	**Support/#Discovered Constraints (DC)**					
				Supp	**#DC**	**Supp**	**#DC**	**Supp**	**#DC**
A	1.104	12	11.855	60	**515**	80	**439**	100	**174**
B	221	37	15.643	60	**2.647**	80	**2.248**	100	**1.767**
C	223	74	10.664	60	**13.625**	80	**8.885**	100	**453**
D	289	152	10.215	60	**9.570**	80	**7.178**	100	**195**

constraints. In this paper, we address this through two directions: (1) pruning discovered constraints to filter out redundant ones and (2) exploiting domain knowledge in the form of reference maps and grouped activities.

3.2 Mine Only for Surprising (Non-trivial) Constraints

When discovering a Declare map, there are many constraints that are redundant thus cluttering the map. Therefore, we propose various techniques to remove redundant constraints.

Removing Weaker Constraints Implied by Stronger Constraints. A constraint between two activities is redundant if a stronger constraint holds between the same activities according to the constraint hierarchy shown in Fig. 1. For example, if a *response* constraint and a *responded existence* constraint hold between the same activities A and B, the *responded existence* constraint is redundant and can be discarded. Note that if a *chain succession* constraint and a *response* constraint hold between activities A and B, then the *response* constraint is redundant because there is a directed path of solid arrows from *chain succession* to *response* in Fig. 1.

Transitive Reduction for Declare Maps. Redundancy may also be caused by the interplay of three or more constraints. Removing such redundancies greatly improves the readability of discovered Declare maps.

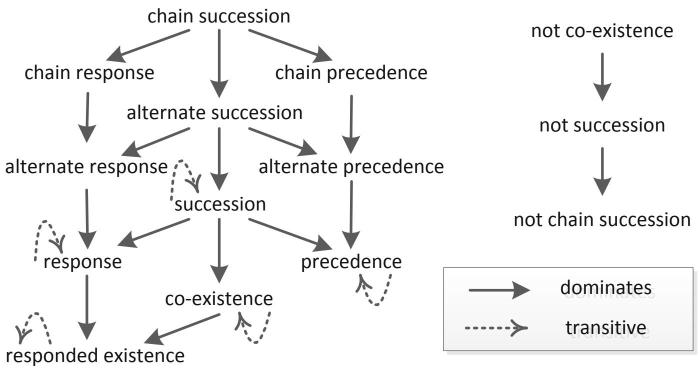
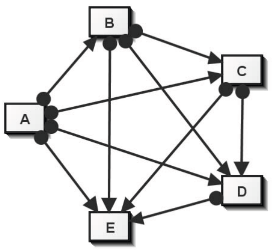
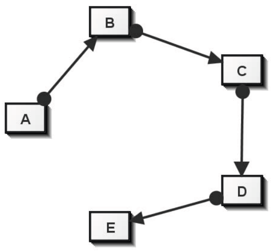


Fig. 1. The solid arcs indicate which constraints dominate other constraints, e.g., *succession* implies *response*. A constraint can be removed if there exists a directed path to it from another constraint involving the same activities, e.g., if a *succession* constraint holds, then the corresponding *response*, *precedence*, *co-existence*, and *responded existence* constraints are all redundant. The dashed arcs indicate constraints that are transitive. For example, using transitive reduction we may remove a redundant *precedence* constraint between A and C provided that the same constraint holds between A and B and B and C.

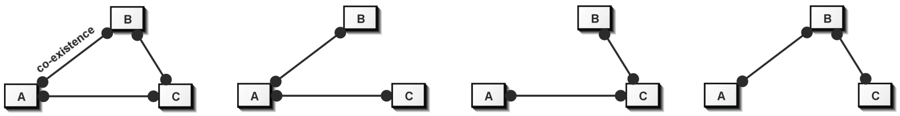


(a) Declare map composed of *response* constraints before the transitive reduction



(b) Declare map composed of *response* constraints after the transitive reduction

Fig. 2. Declare map composed of *response* constraints before and after the transitive reduction



(a) (b) (c) (d)

Fig. 3. Transitive reduction for *co-existence* constraints: the original Declare map (a) can be pruned in three different ways using translative reduction (b, c, and d)

Consider the example in Fig. 2(a). In this case, the *response* constraint between A and C is redundant, because it belongs to the transitive closure of the set composed of the *response* constraints between A and B and between B and C. For the same reason, several other constraints are also redundant in this map. To prune these redundant constraints, we use techniques for transitive reduction of a directed graph. In particular, we have implemented the algorithm presented in [4], which can deal with cyclic graphs. Applying this algorithm on the input map in Fig. 2(a) yields the pruned map in Fig. 2(b).

Not all constraints can be pruned using transitive reduction. For instance, if we take the example in Fig. 4(d), none of the constraints in the map can be pruned out because the *not co-existence* is not transitive and none of the constraints in the map is redundant.[2] The dashed arcs in Fig. 1 show for which Declare constraints transitive reduction can be applied.

The co-existence constraint is "bidirectional" and can be considered as a special case for transitive reduction. For example, for the map in Fig. 3(a), there are three possible reductions (indicated in Fig. 3(b)–(d)).

[2] If A is not co-existent with B, and B is not co-existent with C, then we cannot conclude that A is also not co-existent with C. For example, consider traces $\langle A, C\rangle$, $\langle B, D\rangle$. In these traces, the first two constraints hold, whereas the third one does not.

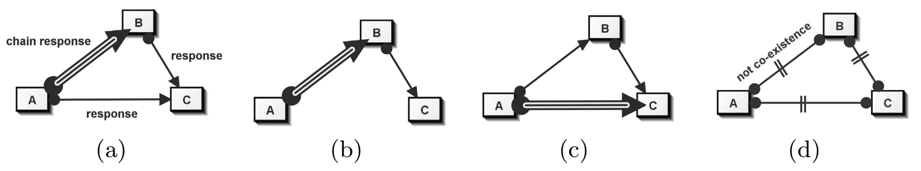


Fig. 4. Examples illustrating transitive reduction: (a) Declare map composed of heterogeneous types of constraints before the transitive reduction, (b) Declare map after the transitive reduction, (c) map for which the *chain response* constraint cannot be pruned, and (d) map illustrating that transitive reduction cannot be applied to *not co-existence* constraints

The transitive reduction can also be used in case some of the constraints involved are stronger. Consider for example Fig. 4(a). The *response* constraint between A and C belongs to the transitive closure of the set composed of the *chain response* constraint between A and B and the *response* constraint between B and C. The *chain response* between A and B, implies the weaker *response* constraint. Hence, we can indeed apply transitive reduction and remove the *response* constraint between A and C. Fig. 4(b) shows the resulting map. In contrast, in the example in Fig. 4(c), the chain response constraint cannot be pruned out because it indicates that the relation between A and C is stronger than a simple response.

Prune a Declare Map through Reduction Rules. Another way to remove redundant constraints is by using reduction rules. For example, reduction rules can be used to prune out redundant *not co-existence* constraints from a map. For the reduction rules defined here, we use the concept of *co-existence path*. A *co-existence path* is a sequence of activities in a Declare map connected through *co-existence* constraints or through constraints stronger than *co-existence* (e.g., *succession*, *alternate succession*, and *chain succession*). For example, $\langle A, C, E\rangle$ is a co-existence path in Fig. 6(a)–(d), and $\langle E, D, C\rangle$ is a co-existence path in Fig. 7(a) and (b).

We illustrate the triangular reduction rule for *not co-existence* using the example map in Fig. 5(a). The *co-existence* constraints ensure that both A and C are in or out. Hence, one *not co-existence* constraint is sufficient to exclude B in case A and C occur. The resulting maps are shown in Fig. 5(b) and (c).

Another reduction rule is shown in Fig. 6. Here, instead of having one activity connected through *not co-existence* constraints to a coexistence path, we have the elements of two coexistence paths connected pair-wise through *not co-existence* constraints. Also in this case, only one of the *not co-existence* constraints is enough (Fig. 6(b)–(d)) to imply all the original constraints in Fig. 6(a).

As shown in Fig. 7, the reduction rules in Fig. 5 and in Fig. 6 can be applied in combinations and, also, with co-existence paths composed of heterogeneous types of Declare constraints. In Fig. 7(a), the original map contains three parallel co-existence paths: $\langle A, B\rangle$, $\langle E, D, C\rangle$ and $\langle F, G\rangle$. In the pruned map depicted in Fig. 7(b), the *not co-existence* constraint between A and D and the

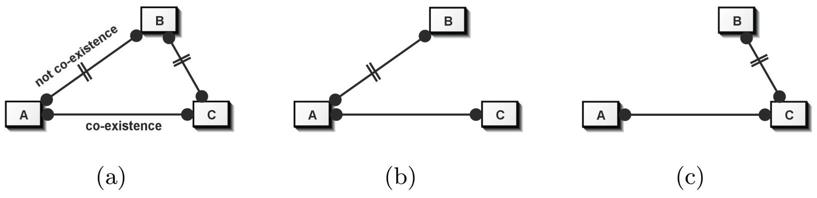


Fig. 5. Triangular reduction rule for *not-coexistence* constraints. The reduced map is not unique: (a) original map, (b) pruned map, and (c) another pruned map.

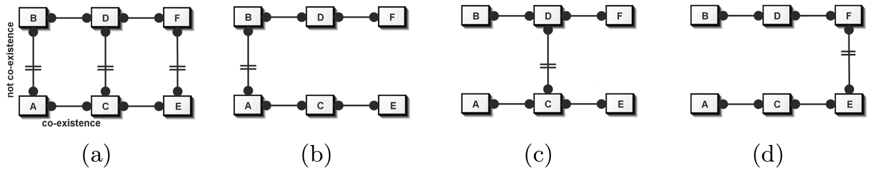


Fig. 6. Reduction rule on parallel co-existence paths for *not-coexistence* constraints. The original map (a) can be pruned in three different ways (b, c, and d).

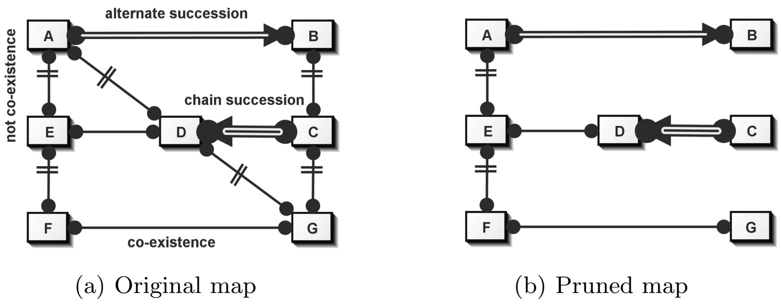


Fig. 7. Four *not-coexistence* constraints can be removed using the two reduction rules

not co-existence constraint between D and G can be removed through the triangular reduction rule. Then, the *not co-existence* constraint between B and C and the *not co-existence* constraint between C and G can be removed through the reduction rule on parallel co-existence paths.

Note that these rules are merely examples. Currently, we are developing a much larger set of reduction rules.

3.3 Guiding the Discovery Process through Domain Knowledge

The techniques provided above may still fail to single out interesting constraints as they do not consider any domain knowledge. Therefore, we propose several means of integrating domain knowledge during the discovery process. The first

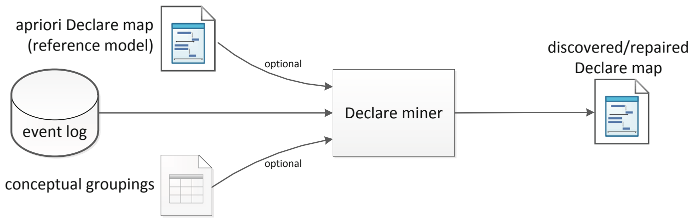


Fig. 8. Using additional domain knowledge to discover a meaningful Declare map

class of techniques deals with repairing an apriori Declare map (called a *reference map*) provided by a domain expert while the second class of techniques deals with discovering constraints based on *conceptual groupings of activities*. Fig. 8 depicts the process of Declare map discovery/repair based on apriori domain knowledge. Note that reference map and the groupings are optional.

Repair a Declare Map. Given an apriori Declare map, it can be repaired in several ways:

– We can use the set of templates (i.e., constraint types) and activities provided in the initial map as a reference and discover constraints pertaining to those templates and activities from the event log.
– We can try to repair the provided Declare map based on the log by *strengthening* constraints or removing constraints that no longer hold.
– One of the challenges when discovering Declare maps from event data is to choose suitable thresholds for objective measures such as support and confidence [10]. Given an apriori Declare map, we can learn the thresholds from the log and use them to enrich the map by discovering additional constraints. One can distinguish between two classes of thresholds, (a) global thresholds and (b) local thresholds. Global thresholds hold for all constraints in the apriori map (e.g., the minimum support for all constraints), while local thresholds vary based on the constraint type (e.g., the minimum support for *response* might be different from the minimum support for *succession*).

Use Activity Ontologies to Discover Declare Maps. Activities in a log can typically be grouped (clustered) based on their functionality, e.g., using an ontology. For example, a log pertaining to the treatment procedures of cancer patients in a hospital contains activities related to patient administration, diagnosis, surgery, therapy, etc. Given such a grouping of activities, we can distinguish between two classes of constraints:

– *intra-group constraints:* this refers to the class of constraints where the activities involved in a constraint all emanate from a single group (see Fig. 9(a)). In many scenarios, analysts would be interested in finding constraints between activities pertaining to a functionality, to a particular department in an organization, etc. For example, in a hospital event log, an analyst would

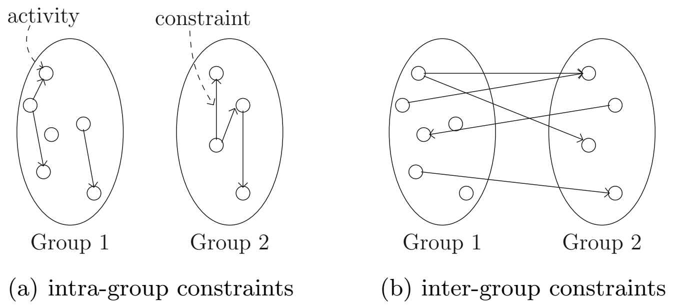


Fig. 9. Inter- and intra-group constraints

be interested in finding relationships/constraints between the various administration activities.

- *inter-group constraints:* this refers to the class of constraints where the activities involved in a constraint belong to two different groups (see Fig. 9(b)). For example, in a hospital log, an analyst would be interested in constraints between activities involved in surgery and therapy.

4 Framework for Discovering/Repairing Declare Maps

We have developed and implemented a comprehensive framework for discovering/repairing Declare maps. Fig. 10 shows that the framework consists of the following building blocks (all implemented in ProM).

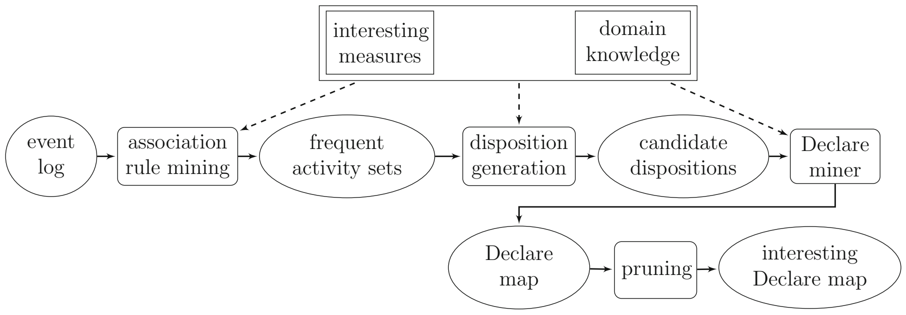


Fig. 10. Framework for discovering/repairing Declare maps. The ellipses depict input/output objects and the rounded rectangles depict steps.

- *Association Rule Mining:* Using the apriori association rule mining algorithm, we first identify the frequent activity sets (of size at most 2).[3] This

[3] This is due to the fact that the Declare templates considered in this paper comprise one or two activities.

process can be assisted through interestingness measures such as support and confidence as well as through domain knowledge such as conceptual grouping of activities and an initial map. The result of this step is a set of frequent activity sets satisfying selected criteria.
– *Disposition Generation:* This step corresponds to the generation of candidate dispositions based on the frequent activity sets uncovered in the previous step. The candidate dispositions are the permissible permutations of the activity sets and are necessary to instantiate Declare constraints. For example, if $\{a, b\}$ is a frequent activity set, the candidate dispositions are (a, b) and (b, a). One can use additional interestingness criteria such as diversity to filter out some candidate dispositions.
– *Declare Miner:* This step corresponds to instantiating the filtered dispositions with constraints and assessing their significance based on the event log using metrics such as support and confidence as well as domain knowledge such as an initial map.
– *Pruning:* This step corresponds to pruning the discovered Declare map using the concepts presented in Section 3.2 (removing weaker constraints, transitive reduction, reduction rules, etc.).

We have implemented the functionalities described in Fig. 10 as part of the *Declare Miner*, a plug-in of the process mining tool ProM (`www.processmining.org`).

5 Experiments and Validation

We evaluated the approach presented in this paper using a real-life event log originating from a large Dutch academic hospital. The log contains events related to the treatment of patients diagnosed for bladder cancer. Bladder cancer is a life-threatening disease and each patient must be treated in a different way, depending on the patient's characteristics (e.g., age, gender), their histology, stage of cancer and depth of invasion. There are several possible tests and treatments the doctors can decide to use. Due to the high variability of the treatment process, it is impossible to create or discover an imperative model. Therefore, we apply the *Declare Miner*.

Here, we report results based on an event log with 289 log traces (patients) containing 152 event classes and 10.215 events. In Table 2, we have already shown how many constraints would be discovered from this log (log D in the table) by pruning the model only based on support. Table 3 shows the number of discovered constraints after removing redundant constraints using the pruning techniques presented in Section 3.2. We can see a significant reduction (84%, 81% and 35% for support equal to 60, 80 and 100 respectively) in the number of constraints. Nevertheless, as shown in Table 3, even for the reduced maps the number of constraints is still too high for the maps to be readable.

To reduce the number of discovered constraints, it is possible to use domain knowledge as a criterion to "guide" the discovery task and to discriminate between constraints that are more interesting and others that are less relevant from

Table 3. Effect of applying reduction techniques of Section 3.2

Support	60	80	100
Discovered constraints without reduction	9.570	7.178	195
Discovered constraints with reduction	1.522	1.316	127
Reduction (percentage)	84%	81%	35%

Table 4. Sample activity groups

G1: Administration	**G2: Surgery**	**G3: Therapy**
First Outpatient Visit	Cysto-urethroscopy	Intravesical Chemo-/Immunothe
Order Fee	Transurethral Resection	Exercise therapy
Contact After-Pa Result	Urethrot.Int.Blind	Endocervical Electr.
Phone Consult	T.U.Proefexcisie Bladder Wall	Peripheral Infusion Insertion
Information/Education	T.U.R.Prostaat.	Urological Beh. And dilatations
Dbc Registration Code		
Coordination Activities		
Short-Out Map Cost anesthesia		
Emergency Assistance		

the domain point of view. To illustrate the conceptual grouping of activities presented in Section 3.3, we define three groups of activities (shown in Table 4). The first group includes administrative activities ($G1$), in the second group there are surgery-related activities ($G2$), and the last group includes therapy-related activities ($G3$).

Fig. 11 depicts the discovered *inter-group constraints*. The map has been discovered with a minimum support equal to 60 and the constraints in the map are sorted by confidence. Their colors range from purple (indicating the highest confidence) to white (indicating the lowest confidence). In Fig. 11, the three activity groups are highlighted with a dashed line. Note that the constraints in the map connect activities belonging to different groups and that there are no connections between activities belonging to the same group. This way, we can focus our analysis on inter-group relations. For example, if we consider the connections between group $G1$ and group $G2$, we can see that after a cysto-urethroscopy and a transurethral resection the patient is registered and the order fee is payed (*alternate response*). Also, activity short-out map cost for anesthesia is preceded by cysto-urethroscopy (*alternate precedence*) and is followed by transurethral resection (*alternate response*). On the other hand, if we consider the connections between group $G2$ and group $G3$, we can see that an intravesical chemotherapy is preceded by a transurethral resection (*alternate precedence*) and by a cysto-urethroscopy (*precedence*).

Fig. 12 shows the discovered *intra-group constraints*. In this case, the constraints in the map connect activities belonging to the same group, whereas connections between activities belonging to different groups are not shown. Therefore, we focus our process analysis on intra-group relations. For example, in group $G2$, we can see that, in most of the cases, activity cysto-urethroscopy is followed by transurethral resection and, vice versa, transurethral resection is preceded by cysto-urethroscopy (*alternate succession*). In group G1, the first

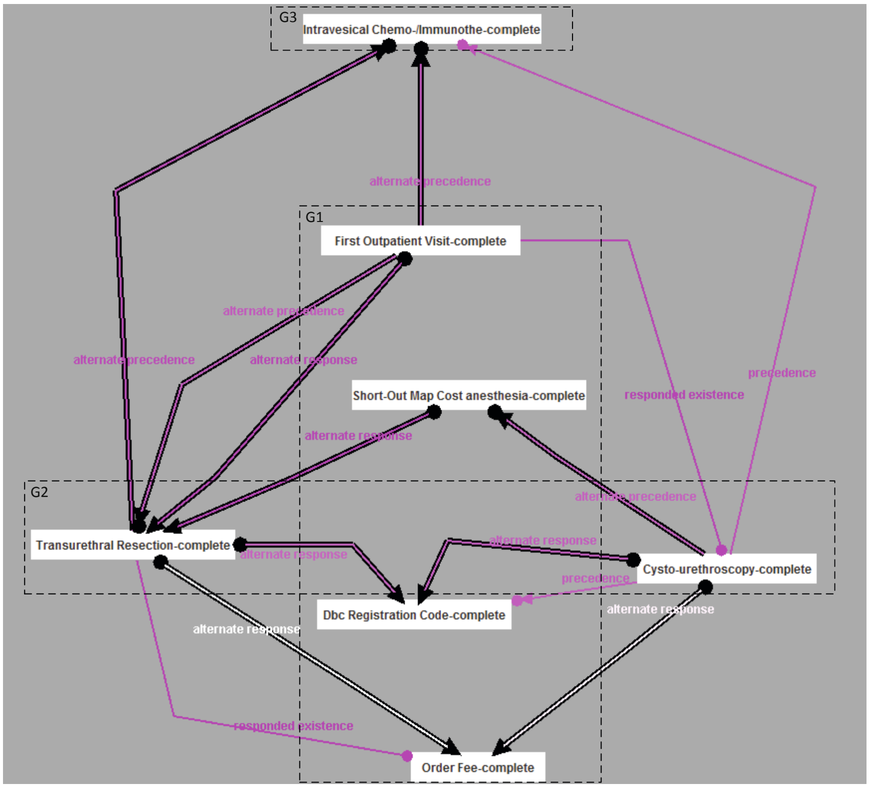


Fig. 11. Inter-group constraints

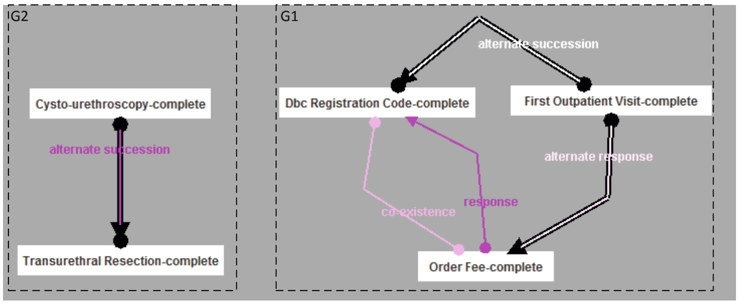

G2
Cysto-urethroscopy-complete
alternate succession
Transurethral Resection-complete
G1
alternate succession
Dbc Registration Code-complete
First Outpatient Visit-complete
alternate response
response
co-existence
Order Fee-complete

Fig. 12. Intra-group constraints

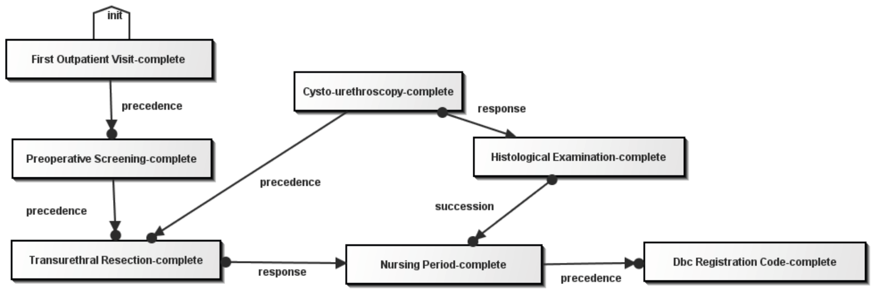

init
First Outpatient Visit-complete
precedence
Preoperative Screening-complete
precedence
Transurethral Resection-complete
Cysto-urethroscopy-complete
precedence
response
Histological Examination-complete
succession
response
Nursing Period-complete
precedence
Dbc Registration Code-complete

Fig. 13. Reference map made using the Declare designer

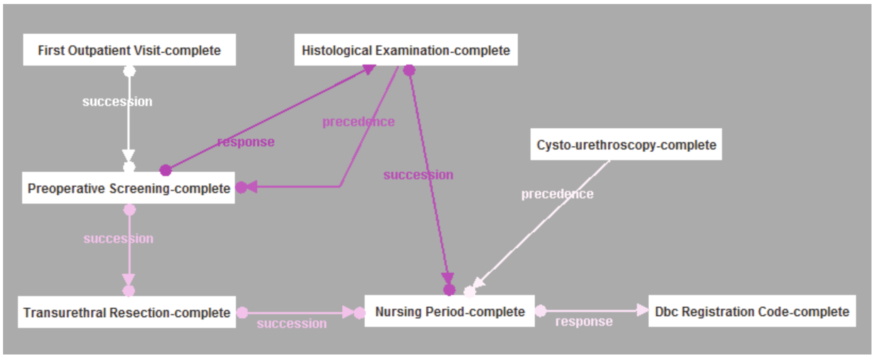

First Outpatient Visit-complete
Histological Examination-complete
succession
precedence
response
Cysto-urethroscopy-complete
succession
precedence
Preoperative Screening-complete
succession
Transurethral Resection-complete
succession
Nursing Period-complete
response
Dbc Registration Code-complete

Fig. 14. Repaired map using the approach in Section 3.3

visit is followed by order fee payment (*alternate response*) and the payment is followed by the patient registration (*response*).

Instead of discovering a Declare map from scratch, it is also possible to provide domain knowledge in the form of a reference Declare map (cf. Section 3.3). This map is repaired based on the information from the event log. Fig. 13 shows a hand-made Declare map. The map was created using the Declare designer [14] and describes the expected process behavior. This map can be given as input to the *Declare Miner*. We repair the map by discovering constraints of the same type as the ones in the map in Fig. 13 and by using the same set of activities. We search for constraints with a support greater or equal to 80.

The repaired map is shown in Fig. 14. In the repaired map, some constraints are the same as in the initial map (e.g., the *succession* constraints between histological examination and nursing period). Some constraints have been strengthened (e.g., the *precedence* constraint between preoperative screening and transurethral resection becomes a *succession* and the *response* constraint between transurethral resection and nursing period also becomes a *succession*). Moreover, the resulting Declare map shows that constraints can be removed (e.g., the *init* constraint), added (e.g., the *response* constraint between preoperative screening and histological examination) or replaced by another constraint (e.g., the *precedence* constraint between nursing period and dbc registration code has been replaced by a *response* constraint).

6 Conclusion

As shown in this paper, one may easily find thousands of constraints for event logs with only a moderate number of activities (see Table 2). This results in incomprehensible Declare maps despite the assumed ability of declarative languages to deal with highly variable processes. However, most constraints in such maps are not interesting or redundant. We developed new techniques to remove such constraints and only conserve the most interesting ones. Moreover, we showed that domain knowledge in the form of reference models or activity groupings can be used to further guide the discovery process. The approach has been implemented as a ProM plug-in and experiments show that it is indeed possible to produce simple, yet informative, Declare maps. In this paper, we reported on a case study involving a bladder cancer treatment process of a Dutch hospital.

References

1. van der Aalst, W.M.P.: Process Mining: Discovery, Conformance and Enhancement of Business Processes. Springer (2011)
2. van der Aalst, W.M.P., Pesic, M., Schonenberg, H.: Declarative Workflows: Balancing Between Flexibility and Support. Computer Science - R&D, 99–113 (2009)
3. Agrawal, R., Srikant, R.: Fast Algorithms for Mining Association Rules. In: VLDB 1994, pp. 487–499 (1994)
4. Case, M.L.: Online Algorithms to Mantain a Transitive Reduction, Department of EECS, University of California, Berkeley, CS 294-8 (2006)
5. Chesani, F., Lamma, E., Mello, P., Montali, M., Riguzzi, F., Storari, S.: Exploiting Inductive Logic Programming Techniques for Declarative Process Mining. In: Jensen, K., van der Aalst, W.M.P. (eds.) ToPNoC II. LNCS, vol. 5460, pp. 278–295. Springer, Heidelberg (2009)
6. Di Ciccio, C., Mecella, M.: Mining constraints for artful processes. In: Abramowicz, W., Kriksciuniene, D., Sakalauskas, V. (eds.) BIS 2012. LNBIP, vol. 117, pp. 11–23. Springer, Heidelberg (2012)
7. Goedertier, S., Martens, D., Vanthienen, J., Baesens, B.: Robust process discovery with artificial negative events. Journal of Machine Learning Research 10, 1305–1340 (2009)
8. Kupferman, O., Vardi, M.Y.: Vacuity Detection in Temporal Model Checking. International Journal on Software Tools for Technology Transfer, 224–233 (2003)
9. Lichtenstein, O., Pnueli, A., Zuck, L.: The Glory of the Past. In: Parikh, R. (ed.) Logic of Programs. LNCS, vol. 193, pp. 196–218. Springer, Heidelberg (1985)

10. 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: Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (eds.) CAiSE 2012. LNCS, vol. 7328, pp. 270–285. Springer, Heidelberg (2012)
11. Maggi, F.M., Mooij, A.J., van der Aalst, W.M.P.: User-Guided Discovery of Declarative Process Models. In: IEEE Symposium on Computational Intelligence and Data Mining, vol. 2725, pp. 192–199. IEEE Computer Society (2011)
12. Pichler, P., Weber, B., Zugal, S., Pinggera, J., Mendling, J., Reijers, H.A.: Imperative Versus Declarative Process Modeling Languages: An Empirical Investigation. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM Workshops 2011, Part I. LNBIP, vol. 99, pp. 383–394. Springer, Heidelberg (2012)
13. Pnueli, A.: The Temporal Logic of Programs. In: Annual IEEE Symposium on Foundations of Computer Science, pp. 46–57 (1977)
14. Westergaard, M., Maggi, F.M.: Declare: A tool suite for declarative workflow modeling and enactment. In: Proceedings of the Demo Track of the Ninth Conference on Business Process Management 2011, Clermont-Ferrand, France, August 31. CEUR Workshop Proceedings, vol. 820. CEUR-WS.org (2011)
15. Zugal, S., Pinggera, J., Weber, B.: The impact of Testcases on the Maintainability of Declarative Process Models. In: Halpin, T., Nurcan, S., Krogstie, J., Soffer, P., Proper, E., Schmidt, R., Bider, I. (eds.) BPMDS 2011 and EMMSAD 2011. LNBIP, vol. 81, pp. 163–177. Springer, Heidelberg (2011)