Discovering health-care processes using DeciClareMiner

Steven Mertens , Frederik Gailly and Geert Poels

Faculty of Economics and Business Administration, Department of Business Informatics and Operations Management, Ghent University, Ghent, Belgium

ABSTRACT

Flexible, human-centric and knowledge-intensive processes occur in many service industries and are prominent in the health-care sector. Knowledge workers (e.g., doctors or other health-care personnel) are given the flexibility to address each process instance (i.e., episode of care) in the way that they deem most suitable. As a result, the knowledge of these processes is generally of a tacit nature, with many stakeholders lacking a clear view of a process. In this paper, we propose an algorithm called DeciClareMiner that combines process and decision mining to extract a process model and the corresponding knowledge from past executions of these processes. The algorithm was evaluated by applying it to a realistic health-care case and comparing the results to a complete search benchmark. In a relatively short time (10 min), DeciClareMiner was able to produce a DeciClare model that represents 93% of episodes of care with atomic constraints. Compared to the 50 h required to calculate the 100%-episode model via an exhaustive search approach, our result is considered a major improvement.

ARTICLE HISTORY

Received 7 June 2017
Revised 9 October 2017
Accepted 8 November 2017

KEYWORDS

Process mining; decision mining; health-care modelling; knowledge extraction

1. Introduction

Business process modelling has been successfully adopted in many different sectors. However, health-care is one of the few sectors that have yet to fully embrace the process-driven approach (Lenz & Reichert, 2007), even though some of the main concerns trending in eHealth are very similar to those of other sectors, namely, cost reduction, efficiency, and patient orientation (Payton, Paré, LeRouge, & Reddy, 2011). The primary causes of this are the specific characteristics of health-care processes (Di Ciccio, Marrella, & Russo, 2015) and, more generally, of flexible, human-centric and knowledge-intensive processes. These processes typically have a set of possible activities that can be executed in many different ways, leading to situations where deviations and variations are the norm rather than the exception. The knowledge workers executing a process (e.g., doctors or other health-care personnel) decide which activities to perform and when they are to be performed. To make these decisions, they often use what is called tacit knowledge: they have an implicit idea of the appropriate actions to perform when certain conditions apply (Lenz & Reichert, 2007). This idea might be partially based on explicit knowledge (e.g., medical or hospital guidelines), but experience usually factors in heavily. Of course, it would be beneficial for an organisation to not only transform these implicit ideas into explicit rules as

much as possible to be able to provide a more consistent and transparent service but also allow making explicit changes to these rules to potentially improve the outcomes of a process.

Many currently available process modelling languages focus solely on the explicit sequencing of the activities (e.g., activity X is executed first, then Y, and finally Z) of these processes (i.e., imperative control flow). However, exposing the decisions that govern the sequencing of activities (e.g., activity Y has to be executed because activity X has been executed and data attribute “d” has value 19) can be just as important, if not more important, for health-care processes. As a result, these languages cannot be used to model the knowledge governing these processes. As part of our ongoing research project, we have developed a declarative process modelling language, called DeciClare, designed specifically for modelling these types of processes (Mertens, Gailly, & Poels, 2017). DeciClare uses decision logic as a complement to the declarative modelling concepts to provide the means to describe processes from the data perspective. A new tier of applications is unlocked by modelling the processes with this higher level of detail. Consider the following example applications:

- The knowledge contained by models will make simulations much more realistic and flexible. For example, how would a department address an

increase in a group of pathologies by 5%? Would this create new bottlenecks? Can these be eliminated by changing, if possible, some decision criteria?

- The decision logic of certain treatments in models can be compared to the available clinical guidelines. Do the doctors follow the guidelines? If not, what are the characteristic and decision criteria of the cases in which they diverge from the guidelines?
- The knowledge of the mined models can be used to more quickly bring interns up to speed or even to educate future nurses/doctors as part of medical courses.

However, the manual creation of decision and process models of real-life health-care processes remains a problem. Parts of the knowledge that must be captured in these models is typically missing due to its tacit nature. This greatly hinders the adoption of many process modelling, analysis, and redesign techniques for health-care processes. Therefore, the research question addressed by this paper is as follows: *“How can we automatically and efficiently discover DeciClare models, keeping in mind the computational challenge due to the vast amounts of data that need to be processed in realistic use cases?”*

The process mining domain provides ways to (semi-) automatically discover, monitor, and improve real processes by extracting knowledge from the data logged by the IT systems of organisations during their day-to-day operations (van der Aalst et al., 2011). A simple example would be a production line company, where the employees execute their daily activities in a strict sequence while the IT system of the company logs the start and end times of these activities. Process mining techniques can detect this pattern and automatically compose a process model that reflects the actual process correctly. The DeciClare language can represent the relevant information of health-care processes, but a corresponding process mining technique for (semi-)automatically discovering these models is currently missing.

In this paper, we propose a process mining technique called DeciClareMiner that is specifically developed for health-care processes. This technique can mine a declarative process model and its corresponding decision logic (i.e., a DeciClare model). In essence, it attempts to offer a more complete picture of the (possibly tacit) knowledge found in health-care processes. This is achieved by mining the rules that have been applied, implicitly and/or explicitly, in past executions of the process, which are contained within a given event log. The two-phase solution strategy used by the miner to efficiently discover the models is the most important contribution of this paper. The strategy intelligently aligns an Apriori-inspired phase with a custom genetic algorithm to considerably reduce the vast search space.

The remainder of this paper is structured as follows. Section 2 gives an overview of the related literature. In Section 3, a brief description of the used modelling

language is provided. This is followed by a description of the proposed algorithm in Section 4. In Section 5, the algorithm is evaluated by applying it to a realistic health-care case and comparing the results to a benchmark for different levels of complexity. Subsequently, the limitations of the presented algorithm are discussed in Section 6. Section 7 concludes the paper and presents directions for future research.

2. Related research

The health-care domain turned to process management due to the constant pressure to improve the quality of their services while at the same time reducing the associated costs (Cardoso, Miller, & Kochut, 2003). Processes in health-care can be classified as either organisational or medical treatment processes (Lenz & Reichert, 2007). Medical treatment processes can be further categorised as either standard, routine, or non-routine processes according to the level of variability and complexity (Lillrank & Liukko, 2004). This paper is focused on routine and non-routine medical treatment processes. To manage the variability and complexity of these processes, several approaches have been proposed. For example, Birkmeyer et al. advocated for the enforcement of hospital volume standards for high-risk procedures (Birkmeyer, Finlayson, & Birkmeyer, 2001). The basic idea is that the large-scale repetition makes these hospitals more capable of managing the variability and complexity. However, Gawande stated that relying on process workers' memory alone is not enough when faced with the occasional extreme complexity and specialisation of health-care. He proposed making extensive use of checklists, similar to in the aviation sector, to reduce human mistakes due to forgetfulness (Gawande, 2010). Another important way to manage medical treatment processes is the use of clinical pathways (Campbell, Hotchkiss, Bradshaw, & Porteous, 1998; De Bleser et al., 2006; Rotter et al., 2010; Vanhaecht, De Witte, Depreitere, & Sermeus, 2006), which model the essential steps in the care of patients with specific clinical problems. Traditional business process management (Amorim Lopes, Soares, Almeida, & Almada-Lobo, 2016; Faber, Konrad, Tang, & Trapp, 2016; Helfert, 2009; Hofmann & Perry, 2010; Keel, Savage, Rafiq, & Mazzocato, 2017; Mazzocato, Savage, Brommels, Aronsson, & Thor, 2010) and modelling techniques (Aguilar et al., 2008; Braun, Schlieter, Burwitz, & Esswein, 2014; Noumeir, 2006; Putman, Anstey, Harper, & Knight, 2016; Rojo et al., 2008; Svagård & Farshchian, 2009) have also been applied in different health-care scenarios.

In the process modelling literature, there are two modelling approaches (three when counting hybrid approaches): imperative and declarative modelling. The former approach explicitly models every possible sequence of activities in the process model, while the latter approach models only the rules that determine if

a sequence of activities is (in)valid (van der Aalst, Pesic, & Schonenberg, 2009). The latter is preferred when modelling non-standard health-care processes, as they do not require enumerating the potentially large number of valid process variations (Goedertier, Vanthienen, & Caron, 2015; Rovani, Maggi, de Leoni, & van der Aalst, 2015; van der Aalst, 2013). Many process variations are modelled implicitly, allowing the choice of a suitable variation to be made at runtime for each individual process instance (i.e., episode of care).

Checklists are a form of declarative process model, as they state only that each activity on the list needs to be performed and not how or when. However, they offer limited support to process workers regarding in which situations a certain checklist should be used. Early declarative languages, such as Declare (Pesic, 2008), tried to improve on this by allowing general relations between possible activities to be described (e.g., if activity X is performed, then activity Y will need to be performed somewhere in the future). This allows for a checklist, of which activities need to be performed, could be performed, and should not be performed, at any point during the execution of a process to be created dynamically. DeciClare (Mertens et al., 2017) takes this a step further by allowing the models to specify the actual decision logic, if available, regarding the time at which certain activities must be performed, could be performed, or should not be performed. In principle, this allows for the dynamic creation of an intelligent checklist that is tailored to a specific situation.

The automated approach of process mining offers an alternative to manually creating health-care process models, which is hard and tedious work due to the vast amount of domain knowledge that is required. Process mining techniques have been applied to different aspects of health-care processes in the past (Huang et al., 2014; Huang, Dong, Bath, Ji, & Duan, 2015; Lismont et al., 2016; Mans, Schonenberg, Song, Van Der Aalst, & Bakker, 2009; Rashid, Naeem, Aamir, Ali, & Ahmed, 2017; Rismanchian & Lee, 2017; Rojas, Munoz-Gama, Sepúlveda, & Capurro, 2016; Rovani et al., 2015; Wang, Tian, Yu, Qi, & Yang, 2017). However, these applications typically target more standardised processes while using imperative modelling languages as a foundation. Using process mining in combination with declarative modelling languages is actually a relatively new research topic. Maggi et al. (Maggi, Mooij, & van der Aalst, 2011) were the first to describe an automated declarative process mining approach, called DeclareMiner. Based on automata, it generates and verifies candidate constraints expressed in the Declare language. By choosing the standard Declare language, the authors limited the approach to discovering only rather simple control flow constraints (i.e., no timed, branched, decision-dependent, data or resource constraints). Later work by Maggi et al. (Maggi, Jagadeesh Chandra Bose, & van

der Aalst, 2012) introduced a two-phase approach to improving the performance using an Apriori-based first phase to reduce the search space. Maggi, Dumas, García-Bañuelos, and Montali (2013) took their approach a step further by adding support for data conditions in a couple of templates of Declare. More recently, a declarative process miner, called TB-MINERful, which can mine branched constraints, was proposed (Di Ciccio, Maggi, & Mendling, 2016). It introduced subsumption and set dominance, two techniques that have also been implemented in DeciClareMiner to reduce the sizes of the mined models. Schönig, Di Ciccio, Maggi, and Mendling (2016) also proposed a framework for the discovery of multiperspective Declare (MP-Declare) models. The work of Schönig et al. (2016) focused on an SQL-like on-demand query system. It does not mine a model on its own; rather, an analyst can manually define queries to examine the event log and find useful rules.

DeciClareMiner has the ambitious goal of discovering the declarative control flow of non-standard medical treatment processes while at the same time also uncovering the medical knowledge governing them. The utilisation of Apriori to mine decision-independent rules in the first phase of DeciClareMiner differs slightly from its application in (Maggi et al., 2012). We moved away from the basic Apriori approach by differentiating between minimal and non-minimal rules. This customisation makes it possible to search for decision-independent rules more efficiently in the second phase of DeciClareMiner. The application of data conditions to a couple of templates of Declare in (Maggi et al., 2013) is similar to the activation decisions of DeciClare, although DeciClare supports them for all templates. The miners themselves essentially differ regarding the use of a heuristic search strategy for the second phase, allowing our technique to scale better with input using increasing amounts of data values, as shown in Section 5.3. The MP-Declare language is even more comparable to DeciClare: both are multiperspective declarative languages based on Declare, but DeciClare distinguishes itself by its additional support from the resource perspective and deactivation decisions. The MP-Declare mining approach requires a significant amount of domain knowledge from the analyst to define useful queries manually, narrowing the application area compared to the automatic mining approach of DeciClareMiner.

3. Modelling health-care processes: DeciClare

DeciClare is a mixed-perspective declarative process and decision modelling language that specifically targets health-care processes (Mertens et al., 2017). It can be used to create detailed models of the control flow, data, and resource aspects of health-care processes. The starting point for the development of the language was a set of requirements extracted from literature and practice. To satisfy the aforementioned requirements, parts of several

existing modelling languages were reused, combined, and integrated with newly introduced solutions. The result was an abstract syntax of this new language, which was made available as an Ecore model (see <https://ars.els-cdn.com/content/image/1-s2.0-S0957417417304414-mm3.zip>). It was subsequently evaluated by domain experts, who applied the conceptual modelling quality framework (CMQF) (Nelson, Poels, Genero, & Piattini, 2012). This paper uses a direct instantiation (i.e., a temporary visual syntax in a textual format) of this abstract syntax, as a real visual syntax has not yet been developed at this time. DeciClare incorporates some concepts of the Declare language and its extensions (Jiménez-Ramírez & Barba, 2013; Pesic, 2008; Westergaard & Maggi, 2012) as well as the DMN language (Object Management Group, 2014). The language supports 36 constraint templates (i.e., types of constraints) spanning four template classes: existence, relation, resource, and data. Twenty-six of these templates have time parameters expressing the time interval in which the constraint applies. The templates are branched, which means that instead of taking just one element as input, they accept a logical expression using multiple elements and logical operators (e.g., “ e_1 or e_2 must be executed at least twice”). For a complete explanation of DeciClare, we refer the reader to (Mertens et al., 2017).

The current implementation of DeciClareMiner supports the nine most important constraint templates of DeciClare. Support for more templates will be added in the near future; however, most of the data and resource templates of DeciClare cannot be mined automatically using a traditional event log because these logs do not have the ability to store the required data (e.g., the number of instances available of a certain resource at a certain time). Table 1 presents the constraint templates currently

supported by DeciClareMiner, each of which is accompanied by an example that illustrates its meaning and parameters. The parameters I and B define the logical operators that can be used in the parameter expressions of the template as a result of branching. The letter used refers to the option of using either the inclusive disjunction (I) or both the inclusive disjunction and conjunction (B).

Table 1 omits the timing parameters required for each of the templates, with the exception of the untimed RespondedPresence template, to avoid overloading the table. These are two parameters expressing the time interval in which the constraint applies. For example, `Response(AtLeast( $e_1$ , 1), AtLeast( $e_2$ , 1), 2, 7)` expresses that if e_1 is executed, then e_2 must be executed at least once after at least 2 time units and at most 7 time units after e_1 was executed. The lower and upper bound time parameters are allowed to be 0 and infinity, respectively, making them reduce to the examples presented in Table 1.

Finally, DeciClare uses concepts from the DMN decision modelling language to express decision logic. This decision logic is used to define when a constraint can be activated and deactivated. Therefore, a constraint does not have to be satisfied until the activation decision evaluates as true (i.e., decision-dependent). The conditions of a decision can consist of any combination of values of the available data attributes, chained using the logical conjunctive and disjunctive operators (e.g., “ $a = \text{True}$ ” and (“ $b = \text{False}$ ” or “ $c = \text{clavicle}$ ”)). When a constraint has a trivial decision (i.e., one that always evaluates as true), it applies generally (i.e., decision-independent). To illustrate the concept of activation decisions, consider the decision-dependent health-care rule for the reaction to trauma cases with internal bleeding. The rule would consist of a constraint `Response(AtLeast(Doctor examination, 1), AtLeast(Surgery, 1), 0, 10)` and an activation

Table 1. DeciClare constraint templates used in this paper.

Class	Template	Example
Existence	AtLeast(I, n)	AtLeast(Apply bandage or Apply cast, 2): the sum of the occurrences of Apply bandage and the occurrences of Apply cast has to be at least 2
	AtMost(B, n)	AtMost(Examine patient and Take X-ray, 3): the number of co-occurrences of Examine patient and Take X-ray must be at most 3
	First(I)	First(Register patient, Examine patient): either Register patient or Examine patient has to be executed before any other event can be executed
	Last(I)	Last(Apply sling or Examine patient): either Apply sling or Examine patient has to be executed as the final event
Relation	RespondedPresence(B_1 , B_2)	RespondedPresence(AtLeast((Take X-ray and Apply sling) or Apply bandage, 1), AtLeast(Examine patient or Perform surgery, 1)): if Take X-ray and Apply sling or just Apply bandage are executed at least once, then Examine patient or Perform surgery must also be executed at least once somewhere in the trace (before, during or after)
	Response(B_1 , B_2)	Response(AtLeast((Take X-ray and Apply sling) or Apply bandage, 1), AtLeast(Examine patient or Perform surgery, 1)): if Take X-ray and Apply sling or just Apply bandage are executed at least once, then Examine patient or Perform surgery must also be executed at least once at some time after Take X-ray and Apply sling or after Apply bandage
	ChainResponse(B_1 , B_2)	ChainResponse(AtLeast((Take X-ray and Apply sling) or Apply bandage, 1), AtLeast(Examine patient or Perform surgery, 1)): if Take X-ray and Apply sling or just Apply bandage are/is executed at least once, then Examine patient or Perform surgery must also be executed at least once immediately after Take X-ray and Apply sling or after Apply bandage
	Precedence(B_1 , B_2)	Precedence(AtLeast((Take X-ray and Apply sling) or Apply bandage, 1), AtLeast(Examine patient or Perform surgery, 1)): Take X-ray and Apply sling or just Apply bandage can be executed only after Examine patient or Perform surgery has already been executed at least once
	ChainPrecedence(B_1 , B_2)	ChainPrecedence(AtLeast((Take X-ray and Apply sling) or Apply bandage, 1), AtLeast(Examine patient or Perform surgery, 1)): Take X-ray and Apply sling or just Apply bandage can be executed only immediately after Examine patient or Perform surgery has been executed at least once

decision that evaluates as true when it is a trauma case and when the patient exhibits signs of shock and/or heavy local inflammation or pain. This implies that when a patient is a trauma victim and is showing signs of shock and/or heavy local inflammation or pain, surgery has to be performed within 10 time units after examination by a doctor. The constraint will be activated during such an examination, prompting emergency surgery.

4. Discovering health-care process models: DeciClareMiner

The process mining algorithm that can be used to discover DeciClare models is called DeciClareMiner. Figure 1 presents a high-level flow diagram of the proposed DeciClareMiner algorithm. The input, i.e., an event log, is first entered into the phase 1 algorithm (described in Section 4.1). In the first phase, all rules that always apply according to the log, independent of decisions, are mined (e.g., $\text{AtLeast}([\text{Clinical examination of patient}], 1)$). These rules have a trivial activation decision (i.e., the decision always evaluates as true). The outcome of this phase is twofold: a set of decision-independent rules and a set of rules that did not make the cut but satisfy certain criteria (e.g., $\text{Response}(\text{AtLeast}(\text{Perform surgery}, 1), \text{AtLeast}(\text{Apply cast}, 1))$ is a minimal rule that, with a trivial activation decision, is correct for 493 traces and

violated in 4018 traces). The former will be stored temporarily pending the completion of the rest of the algorithm. The latter, together with the original event log, will serve as the input for the phase 2 algorithm (described in Section 4.2). The second phase mines rules that only apply in specific situations and thus are dependent on certain (activation) decisions (e.g., $\text{Response}(\text{AtLeast}(\text{Perform surgery}, 1), \text{AtLeast}(\text{Apply cast}, 1)) \text{ activateIf}[[\text{Fractured bone} = \text{Wrist}, \text{More than 16 yrs old} = \text{true}], [\text{Fractured bone} = \text{Wrist}, \text{Extensive damage to arteries or nerves} = \text{true}], [\text{Open or complex fracture} = \text{true}, \text{Fractured bone} = \text{Wrist}], [\text{Fractured bone} = \text{Forearm}], [\text{Fractured bone} = \text{Wrist}, \text{Periosteum torn} = \text{false}]]$). The outcome of this phase will complement the already mined decision-independent rules with decision-dependent rules to form the eventual outcome of DeciClareMiner: a DeciClare model of the process contained by the given event log (with a given minimal confidence and support level). Table 2 presents the definitions of important concepts that are used throughout this paper.

4.1. Phase 1: mining decision-independent rules

The algorithm used to mine rules that always apply is largely inspired by the Apriori algorithm (Agrawal & Srikant, 1994). The Apriori algorithm was originally designed to mine frequent itemsets in the context of

Figure 1. High-level flow diagram of the proposed DeciClareMiner algorithm.

Table 2. Definitions of important concepts.

Concept	Definition
Event log	The data logged by the IT systems of an organisation that are used as input for process mining algorithms. Typically, these event logs are preprocessed to format data, remove unneeded data, and combine information from different data sources
Trace	Each individual execution of the process, as logged by the IT systems. It consists of a sequence of activities/events performed in the execution and other relevant data. An event log is a collection of many traces. In health-care, a trace aligns with what is called an episode of care for one patient
Constraint template	A type of constraint (e.g., AtLeast). A template has a specific interpretation and can have certain parameters
Constraint	A (mandatory or optional) limitation of the universe of discourse represented by the process model (e.g., $\text{AtLeast}(A, 1, 0, 5)$). A constraint is an instantiation of a template, with values given for all necessary parameters
Constraint activation	A constraint for which the activation decision evaluates as true and, when it is a relation constraint, the condition side of the constraint is satisfied. This says nothing about the satisfaction of the actual constraint
Conforming activation	A constraint that is activated <i>and</i> satisfied by a certain (partial) trace (e.g., consequence side of an activated relation constraint is satisfied)
Violating activation	A constraint that is activated <i>and</i> violated by a certain (partial) trace (e.g., consequence side of an activated relation constraint is not satisfied)
Vacuous satisfaction	A constraint that is not activated and thus holds trivially (i.e., it is not violated)
Rule	A constraint in the context of a certain event log. A rule contains a constraint and the corresponding statistics related to the application of the constraint in the given event log (e.g., confidence and support of the constraint)
Confidence	The number of conforming activations of a rule (i.e., vacuous satisfactions are ignored) divided by the total number of activations of the rule
Support	The total number of traces in which a rule is activated (i.e., vacuous satisfactions are ignored) divided by the total number of traces in the event log

association rule mining. An itemset is called frequent if it occurs at least a certain number of times (i.e., support is higher than the given threshold). Apriori uses a breadth-first complete-search approach that starts from the smallest frequent itemsets and works its way up to the largest available frequent itemsets. The key principle used by the Apriori algorithm is that any subset of a frequent itemset has to also be a frequent itemset. Thus, for example, a set containing items A and B cannot be frequent if either (or both) item A or item B on its own is not frequent. Instead of evaluating every possible combination of items, only those that have a chance of being a frequent itemset are evaluated.

Although declarative rules can also be seen as more complex association rules, identifying declarative rules in an event log is quite different from identifying frequent itemsets in a transactional database. First, the evaluation of a rule is much more complex compared to simply counting occurrences in transactions. Each constraint template has a different logic that needs to be evaluated. Additionally, we are interested in not only the support of a rule in the event log but more so in its corresponding confidence. Second, DeciClare does not have only templates connected to one logical expression but also templates that are connected to two logical expressions (i.e., relation templates). Finally, several templates also have an integer parameter (i.e., a bound) in addition to one or two logical expressions. Therefore, to use the Apriori algorithm, a mapping of its core concepts needs to be defined (see Table 3). The pseudocode that results from the mapping of the Apriori pseudocode is presented in Listing 1. In the remainder of this section, the non-trivial mapping steps will be explained in more detail.

Listing 1. Pseudocode of the first phase of the algorithm

Input: set F_1 containing all minimal rules with logical expressions of size 1, a support threshold, a confidence threshold, and a maximal branching level m

Output: set F containing the union of sets $F_1, F_2, \dots$

For $k = 1$ to m or until F_k is empty

1. Generate candidates set C_{k+1} containing rules with logical expressions of size $k + 1$ by joining set V_k , which contains all rules in set F_k with at least one violation, with itself
2. Prune C_{k+1} by removing all rules that have a logical expression that contains the logical expression of a non-minimal rule of size k based on the same constraint template
3. Evaluate the rules in C_{k+1} , and put the rules with at least one conforming occurrence into set F_{k+1}

4.1.1. Leveraged property

In the mapping of the goal of the algorithm, we introduced the notion of a minimal rule. A rule is minimal if no subset of the connected logical expression results in an equal or lower number of violations. In terms of the

goal of the algorithm, discovering a non-minimal rule gets us no closer to a rule with a 100% confidence level than at least one of the previously discovered rules, with a logical expression that is a subset of the non-minimal rule, already did. The non-minimal rule thus offers no additional value over a subset rule that has at most an equal number of violations; at the same time, it is more complex. Consider, for example, an event log with three traces: A, AB, and ACD. Evaluating response constraints with only event A in the condition side logical expression results in Figure 2. The first step calculates the confidence level of rules with a consequence-side logical expression size of 1 (e.g., “+1” = one conforming occurrence, and “-2” = two violating occurrences, resulting in a 33% confidence level). In the following step, these are joined pairwise to create rules with a consequence-side logical expression size of 2. $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(B \text{ or } C, 1))$ is a minimal rule because it has one less violation than any of its subsets, in this case, than its parent rules $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(B, 1))$ and $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(C, 1))$. The same can be said of $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(B \text{ or } D, 1))$. However, $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(C \text{ or } D, 1))$ is not a minimal rule because it has the same number of violations, namely, 2, as both of its parents $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(C, 1))$ and $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(D, 1))$. Based on the property of Table 3, we can already conclude that the rule $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(B \text{ or } C \text{ or } D, 1))$ is non-minimal, as it contains the non-minimal subset rule $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(C \text{ or } D, 1))$.

4.1.2. The iteration principle: optimisations

The three steps of the algorithm in Listing 1 are repeated until the constraints are branched up to the maximal branching level or until no new constraints are found in the previous execution of these steps. The leveraged property allows for three optimisations to be applied in this iteration: the join criteria, the join step, and the prune step. The latter does not differ from the corresponding step in Apriori, but the former two require slightly more attention.

4.1.2.1. The join criteria. Similar to Apriori, a join can be applied to two rules if the logical expressions differ by exactly one element. However, additional criteria states that only rules with a confidence level lower than 100% will be joined. This is because joining two rules, of which at least one has a confidence level of 100%, will not result in anything better than the two original rules. For example, joining rules $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(B, 1))$, and $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(C, 1))$ will result in a rule $\text{Response}(\text{AtLeast}(A, 1), \text{AtLeast}(B \text{ or } C, 1))$, which will be explained below. If it is already known that event A is always followed by

Table 3. Mapping of the Apriori algorithm to our problem domain.

	Apriori for frequent itemsets	Apriori for declarative rules
Input	A transactional database and a minimal support threshold	An event log, a minimal support threshold, and a minimal confidence threshold
Element	Itemsets	Rules connected to logical expressions consisting of activities, events, resources, or data elements
Goal	Identify all itemsets with a support level at least equal to a given value (= frequent itemset)	Identify all minimal declarative rules with confidence and support levels at least equal to given values (= frequent minimal rules)
Leveraged property	Any subset of a frequent itemset is also frequent	Any rule connected to a logical expression that is a subset of the logical expression connected to a minimal rule (same template) is also minimal
Iteration principle: initialisation	Start from all possible itemsets with size 1	For each type of constraint template, start from constraints with logical expressions of size 1 (i.e., no logical operators)
Iteration principle: joining	Increase the sizes of the itemsets by joining all sets that differ by one element	Increase the size of the connected logical expressions by joining all rules based on the same constraint template that do not have a 100% confidence level and have logical expressions that differ by just one element
Iteration principle: optimisation	Only evaluate itemsets that can have a support level at least equal to a given value	Only evaluate rules that can reduce the number of violations while still having a support level at least equal to a given value

Figure 2. Example of the evaluation of the response template.

event B (confidence level of 100%), then additionally stating that A is always followed by event B or C (i.e., the child rule) offers no new information.

4.1.2.2. The join step. Rules with a logical expression of size k are joined to create rules with a logical expression of size $k + 1$ in the iteration steps of the algorithm. However, the elements of the logical expressions can be connected with either the inclusive disjunction or the logical conjunction operator. Therefore, the influence of joining these logical expressions using either operator on the number of violations and the support level is investigated for each template when the leveraged property from Table 3 is applied (see “https://github.com/stevmert/DeciClareMinerV11/blob/master/Health_Systems_-_Phase_1.pdf” for the details). As a result, certain rules (e.g., OR join for the condition side of relation rules) will never be joined, as they will never lead to a new minimal rule being found.

However, the AtLeast and AtMost templates have an additional numerical limit associated with them. This makes it very inefficient to apply the Apriori candidate generation step to these templates, as this would require the generation of all possible event sets of size k with all possible limits (= a candidate set of infinite size). Instead, our algorithm makes an exception regarding the Apriori-inspired iteration for these templates and will generate all possible logical expressions and, for each logical expression, calculate the associated limit based on the event log directly. To increase the efficiency, these templates are evaluated together, which makes it possible to generate all rules with a meaningful limit of both templates for some logical expression in one pass of the

event log with no performance hit compared to evaluating one rule of these templates (similar optimisation is also used for the other templates). For example, suppose A occurs twice and three times, respectively, in an event log with two traces. With a support level of more than 50%, two meaningful rules can be mined in one step: AtLeast(A, 2) and AtMost(A, 3).

4.1.3. Running example

To illustrate how the first phase works, let us consider the following example for the consequence side of the positive Response template (Figure 3). We have an event log of a process with the following traces: AB, AAB, CAC, and ABABC. The confidence threshold will be 80%, and the support 50%. The algorithm starts by creating every Response constraint possible with a condition and consequence expression of size 1. The candidates are subsequently evaluated by calculating their confidence and support levels. The first candidate rule has a confidence and support level above the given thresholds and will therefore be added to the result set. Note that the first five rules are minimal rules, as they have no subset rules, but only the first rule is a frequent minimal rule. Next, a new set of candidate rules is generated by joining all compatible candidate rules from the previous candidate set. Joining the consequence side of the positive Response template will be done using only the OR operator (see “Health_Systems_-_Phase_1.pdf”). Additionally, the last two rules from the candidate set from iteration 1 are not joined since the latter rule has a confidence of 0% (i.e., non-minimal). The result is a new candidate set with two new candidate rules. Since all rules from the candidate set of the previous iteration are minimal, no

Figure 3. Running example of the first phase (minimal rules in black; non-minimal rules in grey).

candidates can be pruned at this point. The subsequent evaluation of the candidates yields one frequent minimal rule and one non-minimal rule (i.e., the second candidate has as many violations as the fourth candidate rule, which is one of the parent rules, from iteration 1). The first candidate will be added to the result set to give us two frequent minimal rules. The algorithm stops at this point, as no candidates of size 3 can be generated.

4.1.4. The output of the first phase

This first phase results in a set of decision-independent rules with a confidence and support level equal or higher than the given thresholds and logical expressions of a size equal to or less than the given maximal branching level (top output in Figure 1). As a waste product, it also generates all minimal rules no matter the confidence and support levels. This set of “waste” rules will be used to optimise the second phase of the algorithm (bottom output in Figure 1). With the exception of the AtLeast, AtLeastChoice, AtMost, and AtMostChoice templates, all decision-dependent rules can be represented by one of these waste rules combined with some decision. Therefore, the constraints of all these minimal waste rules with a confidence level of less than 100% and a support level higher than the support threshold (as the support level of a rule can only drop by adding an activation decision) that were generated during this first phase will be used as seed constraints for the second phase. In the running example, this would be the first five candidate rules from the first iteration. As a result, the second phase will not have to generate any constraints and can focus on only the decision logic. This will significantly reduce the complexity of defining a search strategy for finding decision-dependent rules and substantially reduce the size of the search space.

For the AtLeast and AtMost templates, only the AtMost rules with 100% confidence levels are used as input, as these express the boundaries within which it is meaningful to search for the decision-dependent rules of these templates. For example, from the rule AtMost(A, 3), with a confidence level of 100%, we can deduce that any meaningful decision-dependent AtMost rule with the same logical expression has to have a bound of 0, 1, or 2 and that any meaningful decision-dependent

AtLeast rule with the same logical expression has to have a bound of 1, 2, or 3.

4.2. Phase 2: mining decision-dependent rules

In the second phase, decision-dependent rules are generated. Our assumption is that for health-care processes, the number of possible activities will be much lower than the number of possible combinations of data elements on which the activation decisions are based. Therefore, while the first phase uses a complete search algorithm, the second phase uses a heuristic approach to search for meaningful rules from the vast set of possible combinations of data values and seed constraints from the first phase.

For problems with a search space that is too large to go over completely, heuristics can offer “good enough” solutions by sampling the search space. A subset of these heuristics, called genetic algorithms, mimics evolutionary processes that utilise natural selection. These processes are non-deterministic but have been proven and tested by nature to be very flexible and efficient. The heuristic that we use belongs to the class of genetic algorithms that use the crossover, mutation, and selection operators to solve constrained or unconstrained optimisation problems (Mitchell, 1996). The approach requires the problem to be redefined in terms of finding the best genetic codes (i.e., individuals) according to some fitness function. An individual consists of a subset of the available genes, which are the features that can make the individual a better or worse fit as a solution for a problem compared to other individuals. A population of these individuals is recombined and mutated over multiple iterations to allow the fitness of the population to evolve by sharing the genes of the better individuals. Table 4 presents a mapping of the general genetic approach to our problem domain.

The search problem of finding the best decision-dependent rules can be redefined as finding activation decisions for the seed constraints from the first phase. An individual is a combination of a seed constraint and a non-trivial activation decision. The decision values, which can form an activation decision, can be regarded as the genes of the genetic code of individuals. The basic approach is illustrated in Figure 4. The algorithm

Table 4. Mapping of the genetic approach to our problem domain.

	Genetic decision-dependent rule miner
Input	Seed rules, the available data elements, an event log, an intermediate population size, a minimal support threshold, and a minimal confidence threshold, the number of new seeds to be added in each iteration, a fitness function, and a recombination and mutation frequency
Element	A decision-dependent rule, with its activation decision serving as its genetic code
Fitness	A measure of how well a decision-dependent rule reflects the underlying process of the event log
Goal	Find the best decision-dependent rules with a confidence and support above the thresholds
Operations	To find better decision-dependent rules, changes are introduced by recombining and mutating candidates; this allows good decision logic to be shared across the population
Iteration principle	The best decision-dependent rules of the previous iteration and some new seed rules with random decisions form the population of the next iteration; this offers a balance between searching in a specific direction (i.e., of previously found good decision-dependent rules) versus in all directions (i.e., new good decision-dependent rules)

Figure 4. Overview of the second phase of the algorithm.

starts by combining seed constraints, produced in the first phase, with random activation decisions (e.g., “Has fever?” = false and “headache” = true) to create the initial population of individuals. The individuals in the initial population are then evaluated and sorted according to the fitness function. The recombination operator is subsequently used to create a population of children of this population. The higher the fitness of a rule, the higher the chances of being selected as one of the two parent rules of a new child rule. The child gets half of the genes of each parent, with some random mutations being applied to the genes based on the mutation rate. In the next step, this child population is evaluated and sorted according to the fitness function. The best children are used to create a new edit population, together with a given percentage of new seeds in each iteration. This edit population is used to start the next iteration until no new individuals are added to the result population during a given number of iterations or the given maximal amount of search time has been exceeded. Individuals that satisfy the confidence threshold are integrated into the result population during the two evaluation and sort steps of each iteration. Note that the support threshold is ignored at this point and will only be introduced during the post-processing steps.

4.2.1. Fitness function

To measure the fitness of an individual, we use four partial scores (all expressed as percentages):

- (1) A score that expresses the increase of the trace conformance (i.e., number of conforming traces divided by the total number of traces for which the rule was activated) of an individual compared to the decision-independent seed rule on which the individual was based.
- (2) A score that expresses the generality of the activation decision of an individual. The score is higher for individuals with an activation decision that uses less decision elements.

- (3) A score that punishes individuals that stay dominant in the population for too many iterations. Each individual has a generation, i.e., the number of recombination operations it has undergone. An individual created from a seed constraint and a random decision has 0 as generation and an individual created in a recombination step has one plus the highest generation of its parents as generation. The punishment is negligible for younger generations but rises exponentially. This ensures that the search direction stays sufficiently broad.
- (4) A score that expresses the potential of the individual to increase the diversity of the result population. A frequency map, which stores the number of times a certain constraint (with its specific parameter values, ignoring the activation decision) has already been found with satisfactory confidence and support levels, is kept. An individual with a constraint that is part of an already found individual will progressively receive a lower and lower score as the frequency of found individuals with the constraint rises. This ensures that other search directions will be explored more frequently as frequent search directions are being exhausted.

4.2.2. Optimisation

The obvious choice would be to run the second phase algorithm for all templates at once. However, because we grouped certain templates together to increase the efficiency of the evaluation step, the fitness score of such a group of constraints cannot be fairly compared to that of other groups based on other templates. For example, a relation group evaluates the RespondedPresence, Response, ChainResponse, Precedence, and ChainPrecedence templates as well as their negative counterparts for a certain set of required parameters, while the bounded group does the same for the AtLeast and AtMost templates. Its score for the partial fitness

measure I is a weighted combination, based on the subsumption hierarchy (Di Ciccio & Mecella, 2015), of the scores of all rules found for this group. However, for the bounded group, there will always be at least one rule found with a confidence of 100% because of how they are evaluated: $\text{AtMost}(\text{AE}, x)$, with x being the maximum number of times the activity expression “AE” occurred in any trace of the given log (see end of Section 4.1). Therefore, the bounded group fitness will generally be higher than the relation group fitness. Hence, the second phase algorithm will be executed separately for each group of templates. The size of the intermediate populations for each group is set relative to the number of available seed rules for that group. Each iteration of the global algorithm for the second phase executes one iteration for each group, unless the group has already converged. Finally, the result populations of each group are merged.

4.2.3. Running example

Figure 5 presents a running example of how the second phase would be performed for the AtLeast template (i.e., ignoring the AtMost template of the group). The confidence (c) and support (s) thresholds are set to 100 and 50%, respectively. For simplicity, we set the size of the intermediate populations to just two individuals, with four children, and consider only one seed constraint, namely, $\text{AtLeast}(A, 1)$ (i.e., “A is executed at least once”), with its corresponding decision-independent rule having a confidence of 60% and a support of 100%. One can assume that the bound parameter does not change. Three data attributes are used in this example: x (Boolean), y (Boolean), and z (categorical with values: “1”, “2”, and “3”). In each iteration, one new seed is added to the new edit population.

In the first step of the second phase, an initial population is created by combining the seed constraint with random decisions based on the available data attributes. The constraints in the initial population are then evaluated and added to the edit population, which in turn is sorted according to the fitness function. In the next step, the child population is created by recombining the rules in the sorted edit population with random splits of their genes. This process also introduces random mutations in the genes of the children (grey). The child population is then sorted, with one rule making it into the result population. The best rule of the sorted child population is also transferred to the new edit population for the second iteration, along with a new seed constraint combined with another random decision. The process continues until it converges or the time limit is reached.

4.2.4. The output of the second phase

After the iteration steps of the genetic algorithm, three post-processing steps are performed to improve the combined results from both phases. The first step collects all decision-independent rules that satisfy the confidence and support thresholds from the results of the first phase while also removing subsumed rules according to the subsumption hierarchy (e.g., keep $\text{ChainResponse}(A, B)$ and remove $\text{Response}(A, B)$, as the former is a stricter version of the latter). The second step merges the decision logic for rules from the results of the second phase with the same constraint but different activation decisions. This step also removes all subsumed rules from these results. The last step combines both results and removes redundant rules (e.g., $\text{First}(A)$ makes $\text{AtLeast}(A, 1)$ redundant).

Figure 5. Running example of the second phase.

5. Evaluation

The DeciClareMiner process discovery algorithm is implemented (fully single-threaded) to conduct an evaluation. The implementation of DeciClareMiner used in this paper can be found at <https://github.com/stevmert/DeciClareMinerV11/>. The event log and case description (.../models/) as well as a summary of the results of the experiments (.../results/) can also be found at the above link. For the actual results of each of the experiments (ca. 1.13 GB of compressed zip-data, mostly in .CSV format), please contact the first author.

5.1. Data-set

In previous work (Mertens et al., 2017), the health-care process for treating arm-related fractures was modelled using the direct instantiation of the DeciClare abstract syntax metamodel. The process entails the registration, diagnosis, and treatment phases for patients with one or more (possible) fractures of a finger, hand, wrist, forearm, upper arm, shoulder, and/or collarbone. This DeciClare process model was created based on the available literature and by interviewing doctors and nurses that participate in this health-care process in the emergency and orthopaedic department of several hospitals in Ghent, Belgium. This fictional (i.e., this is not an isolated process in real-life) but realistic health-care case will be used for the evaluation of the DeciClareMiner process discovery algorithm.

We first generated an event log based on the DeciClare model of the arm fracture case. We identified 2072 unique realistic combinations of control flow and data that fit the model correctly (i.e., no noise). This is of course a simplification, as the model theoretically allows an infinite number of unique combinations. Additionally, we chose to exclude some of the activities from the original model (i.e., all prescriptions of medication and physiotherapy) for simplicity. The resulting event log uses 9 data attributes, 1 categorical (7 categories) and 8 Boolean, and 9 activities and was created by adding each unique combination at least once, but possibly multiple times (at random), to the log until it contained exactly 5000 traces.

As an example of the kind of rules that corresponds to the arm fracture case, consider the following two relatively complex examples. First, one rule yields all situations in which surgery will need to be performed at least once sooner or later: `AtLeast(Perform surgery, 1, 0, -1) activateIf[[Open or complex fracture = true], [Fractured bone = Upper arm, No improvement in 3 months = true], [Extensive damage to arteries or nerves = true], [Fractured bone = Upper arm, Both arms broken = true], [Periosteum torn = true, Fractured bone = Wrist]]`. The first part represents the constraint (i.e., surgery must be performed at least once), while the second part describes its activation decision (i.e.,

all the possible data combinations for which the constraint applies). This rule is activated in 4511 of the 5000 traces in the arm fracture event log and thus is regarded as frequent behaviour. The second example represents less frequent behaviour contained by the event log, i.e., in 493 of the 5000 traces. It describes the situations in which a cast needs to be applied after surgery is performed: `Response(AtLeast(Perform surgery, 1, 0, -1), AtLeast(Apply cast, 1, 0, -1), 0, -1) activateIf[[More than 16 yrs old = true, Fractured bone = Wrist], [Extensive damage to arteries or nerves = true, Fractured bone = Wrist], [Open or complex fracture = true, Fractured bone = Wrist], [Fractured bone = Forearm], [Periosteum torn = false, Fractured bone = Wrist]]`. The decision logic associated with both of these example rules is presented in its most general and precise form. Of course, subsets of this decision logic exist that describe only parts of the general behaviour (i.e., missing disjunctive decision logic or overfitting in a few specific cases) and can therefore uniquely identify only a subset of the traces of these rules.

5.2. Benchmark creation

To analyse the DeciClare models mined using DeciClareMiner, benchmarks were created for several branching levels. A *complete search* algorithm was used to mine reference models that could serve as benchmarks for branching levels 1 and 3 and for unlimited branching. Table 5 highlights some important numbers from the application of the complete search algorithm, which were executed one by one on a single core of an Intel E5-2680v3 with the confidence threshold set to 100% (as there is no noise in the log) and no support threshold (as most of the 2072 variations occur only once or twice due to limiting the log to 5000 traces). The reference models do not contain any subsumed or redundant rules, as these are filtered out automatically. Each seed rule needed to be checked for all 279,935 possible data combinations for both branching levels. This vast search space is the result of the 9 data attributes used in the arm fracture event log and is still very limited compared to real-life cases. Compared to branching level 1, the search space increases by an additional 736% for a branching level of 3 and by 1475% for unlimited branching. This supports the assumption made at the

Table 5. Application of the complete search algorithm for different branching levels.

Branching level	Time	# of seed rules for Phase 2	# of rules mined in Phase 2	Total # of rules mined in both Phase 1 and 2
1	49 h 39 min	56	129	268
3	353 h 24 min	412	973	1177
Unlimited	734 h 42 min	826	3989	4205

start of Section 4.2, based on which a heuristic for the second phase of the algorithm was adopted instead of a complete search approach. The increased search space for branching levels higher than 1 make it infeasible to calculate a complete search model in real-life cases.

To calculate how the models mined using DeciClareMiner stack up to the benchmarks, two criteria are used: the percentage of traces uniquely identified and the percentage of constraints identified by the model as a whole compared to the corresponding reference model. For example, an overfitted version of the first example rule from Section 5.1 might only be able to uniquely identify 3000 traces. This means that for this rule, the mined model would score 3000 out of a possible 4511 for the first criteria and 1 out of 1 for the second criteria. However, if the mined model does not contain any version of this rule, it would score 0 out of a possible 4511 for the first criteria and 0 out of 1 for the second criteria for this rule. The scores are calculated for each rule in the corresponding reference model, are compared to the rules in the mined model, and together serve as the score for the mined model.

5.3. Results

Using the arm fracture case event log as input, DeciClareMiner was executed using the same single core of an Intel E5–2680v3 with the same confidence and support thresholds as for the benchmark creation (100% and 0%, respectively), with different settings for the branching level and amount of search time for the second phase. For each branching level and search time combination, DeciClareMiner was executed many times (i.e., # of experiments column in Table 6). This was necessary because the second phase of the algorithm is not deterministic, which means that the results of the miner can vary. Based on many runs of the miner on the same

settings, we were able to calculate the average model mined using the predefined branching level and search time.

The power of the chosen heuristic is illustrated by the fact that DeciClareMiner found 99% of all the constraints and 93% of the total decision logic in just 10 min (see Table 6). The benchmark needed 49 h and 38 min to calculate the complete search model. If we assume that the complete search algorithm mined the results evenly over this runtime (i.e., 0.03% per minute), it would have mined 93.32% of the total decision logic after 46 h and 20 min. Then, DeciClareMiner speeds this up by a factor of 277 (i.e., 9.33% per minute). Figure 6 illustrates how the intermediate populations evolve during the execution of the algorithm (please note that the results in Figure 6 are a bit worse than the actual performance of the algorithm due to the added overhead of having to store the intermediate population after each iteration). The second phase of DeciClareMiner does most of the work early on due to the innate characteristics of the genetic approach. After just 13 iterations, 88% of all the constraints and 61% of the total decision logic had already been found. The scores further approach the model solution with each additional iteration, though the speed drops significantly at the same time. Almost 60% of the model had already been found during the first minute of execution (speed-up factor of 1740). However, although the results further improve with a longer runtime, to 98 and 99% for 20 and 40 min, respectively, the speed-up factor drops significantly at the same time to 145 (4.88% per minute) and 74 (2.48% per minute), respectively.

For branching levels 3 and unlimited branching, the mined models still obtain respectable averages of 87% and 68% of the benchmark model in 40 min. Although the absolute speed declines for higher branching levels (2.18 and 1.70% per minute, respectively, during the 40 min search), the relative speed-up factor remains high compared to the runtimes for a complete search approach. DeciClareMiner speeds up the discovery of 87% of the model by a factor of 462 for branching level 3 and of 68% of the model by a factor of 752 for unlimited branching.

5.4. Discussion

As illustrated in the evaluation, the miner presented in this paper is a promising step towards modelling and capturing the knowledge of real-life health-care processes. DeciClareMiner can efficiently mine declarative process models containing decision-independent and decision-dependent constraints. It was able to produce a model that represents 93% of episodes of care with atomic constraints in a relatively short time (10 min). This is considered a major improvement compared to the 50 h it took to calculate the 100%-episode model using an exhaustive search approach. The

Table 6. Results from the mining experiments for different settings.

Branching level	Search time P2	% of traces identified	% of constraints identified	# of experiments	Speed-up factor
1	10 min	93.32% ($\sigma = 1.42\%$)	98.72% ($\sigma = 0.84\%$)	271	278
	20 min	97.64% ($\sigma = 0.52\%$)	99.56% ($\sigma = 0.48\%$)	409	145
	40 min	99.05% ($\sigma = 0.19\%$)	99.82% ($\sigma = 0.34\%$)	117	74
3	10 min	69.22% ($\sigma = 1.13\%$)	81.38% ($\sigma = 1.27\%$)	270	1466
	20 min	78.32% ($\sigma = 1.71\%$)	89.39% ($\sigma = 1.28\%$)	410	830
	40 min	87.29% ($\sigma = 1.15\%$)	94.33% ($\sigma = 0.70\%$)	117	462
Unlimited	10 min	50.57% ($\sigma = 1.22\%$)	58.89% ($\sigma = 1.23\%$)	364	2229
	20 min	59.55% ($\sigma = 1.59\%$)	67.85% ($\sigma = 1.78\%$)	572	1312
	40 min	68.2% ($\sigma = 1.04\%$)	77.41% ($\sigma = 1.16\%$)	307	752

Figure 6. Evolution of the intermediate population scores.

main building blocks of the DeciClareMiner algorithm, i.e., the Apriori-inspired first phase algorithm and the genetic algorithm of the second phase, can address rising complexity very well. This behaviour was expected due to the selection of a genetic algorithm as the basis for the second phase. Similar to the natural processes on which genetic algorithms are based, genetic algorithms offer good performance when faced with high complexity. In addition, although not a part of this evaluation, they should also be able to handle varying levels of noise in the event log, as these techniques are considered relatively noise-resistant.

The downside of using a heuristic to search for decision-dependent rules is that we cannot guarantee that every detail of the complete search model will ever be found. As Figure 6 shows, the algorithm will quickly find most of the complete search model but eventually slows down more and more as it approaches 100%. We believe that this is a necessary trade-off to apply this kind of technique to the domain of health-care in real-life environments. In real-life settings, the cases are much more complex than the arm fracture case used in this evaluation. The event log used in the evaluation has a size of just 26 MB, whereas event logs for real-life cases can easily be more than 10 GB (e.g., the MIMIC-III clinical database (Johnson et al., 2016) contains 43 GB of data). Clearly, the available processing power and time are going to be the limiting factors. Additionally, the domain is very dynamic, as medical knowledge evolves continuously. In this setting, it is better to have a very good approximation of the complete search model in reasonable time than a complete search model that is already at least partially obsolete by the time it is available. We believe that the efficiency offered by DeciClareMiner is necessary for the adoption of process mining in real-life health-care organisations and, more specifically, in non-standard medical treatment processes.

DeciClareMiner can also be applied when the event log is not complete, as in our evaluation and in almost every practical case. However, this requires a subsequent evaluation of the resulting set of rules by domain experts to relax possibly overfitted rules. Because DeciClareMiner calculates the whole subsumption

hierarchy, this relaxation is just a matter of selecting an already available rule that is lower in the hierarchy (e.g., if the event log only contains episodes of care that execute activity B immediately after A, which is not that strict in real life, the mined rule $\text{ChainResponse}(A, B)$ can be relaxed to $\text{Response}(A, B)$). The use of a declarative language makes the evaluation of the rules by domain experts relatively easy, as each rule can be evaluated in isolation.

The availability of DeciClare models of health-care processes can enable many new applications at the management level. For example, consider the checklists mentioned in Section 2. They are already being used throughout the health-care sector at an operational level to reduce the number of human errors that occur due to the innate and overwhelming complexity of non-standard medical treatment processes. Declarative process models can be used to generate these checklists on-demand during the execution of a process. The dynamic checklists can give an overview of the activities that still need to be performed, those that should not be performed, and those that are optional. The decision logic in a mined DeciClare can help produce tailor-made checklists for patients currently being treated, taking into account the specific patient characteristics and progression of treatment. Of course, it would be advisable to have the mined model validated by experts first. The treating physicians could also perform this validation on-the-fly, as the knowledge mined by DeciClareMiner represents only a small portion of their extensive domain knowledge.

6. Limitations

The current implementation is already rather efficient, as the first phase and each iteration of the second phase of the algorithm each have a linear asymptotic time complexity (i.e., $O(n)$) in relation to the number of traces in the given event log. However, it should still be considered a proof of concept. There is still significant room for optimisation (e.g., better parameter tuning and the introduction of advanced techniques such as parallelisation and caching). Additionally, the DeciClare language support will need to be expanded for future use. First, the current

implementation can mine nine of the available templates (e.g., no data constraint templates). These nine are considered the most important templates. Support for other templates will be added in the future, although some templates cannot be mined automatically (e.g., authorisation templates), and several others require information that is not typically available in an event log. Second, the implementation only supports relation templates using the “at least once” interpretation for the logical expressions of the condition and consequence sides of the constraints (e.g., if A is executed at least once, then B must be executed at least once) and the “at most zero times” interpretation for the logical expression of the consequence side (e.g., if A is executed at least once, then B can be executed at most zero times). This is similar to the interpretation of the corresponding branched templates in Declare for the positive and negative versions of the relation templates. Third, DeciClare allows constraints to be not only activated by way of a decision but also deactivated. However, from experience, we know that deactivations are required much less frequently compared to activations; thus, including these in the second phase in a similar way would not be very efficient. How to add support for constraint deactivations is still an open research question and might require the introduction of a third phase. Fourth, the current implementation only supports data attributes with a finite domain. A data attribute can have a Boolean (no overlap) or categorical (allowed to overlap) domain, but continuous data attributes (e.g., temperature = 39.6°) have not yet been considered. These present a tougher challenge, as it is difficult to process their infinite domains efficiently. An extra preprocessing step by a domain expert is currently required to convert infinite domains to well-defined finite domains until support for these types of data attributes is added. Lastly, the time parameters present in many of the constraint templates are not yet supported. These were included in a previous version of the miner but were only outlined by the minimal and maximal time parameters for each rule and not mined actively. However, this subject will also be revisited as part of future research.

Finally, it is also worth noting that the current lack of a visual syntax for DeciClare does not greatly impact DeciClareMiner. When a real concrete syntax becomes available, no changes to the miner itself are required. A visual representation of the mined model can be easily created separately, as the results of DeciClareMiner are a direct instantiation of the abstract syntax of the language, which will have to be the foundation of the visual syntax anyway.

7. Conclusions and directions for future research

In this paper, we proposed a process discovery approach for health-care processes. The process discovery algorithm of DeciClareMiner can mine DeciClare process

models in the form of decision-independent and decision-dependent rules. The algorithm attempts to automatically extract a complete picture of the (possibly tacit) knowledge of a process contained within a given event log.

The algorithm consists of two phases to mine both decision-independent rules and decision-dependent rules. The first phase involves a complete search algorithm inspired by the Apriori association rule mining algorithm, while the second phase involves a heuristic belonging to the class of genetic algorithms. This difference in approach is rooted in the assumption that in the context of health-care processes, the number of possible activities will be much lower than the number of combinations of available data elements on which decisions are based. This means that using a complete-search algorithm for the first phase is acceptable, while for the second phase, a heuristic is more appropriate. Although the concept of mining decision-dependent rules is the main contribution of this paper, the alignment of the strategies for mining decision-independent and decision-dependent rules is also a novelty. Using the waste generated in the first phase, we can turn the second phase into a much more targeted search. The process discovery algorithm was evaluated using a realistic event log of an arm fracture health-care process and by comparing the results to a benchmark set obtained via an exhaustive search approach. DeciClareMiner considerably speeds up the discovery of a process model compared to a complete search algorithm. A trade-off was made to prioritise good results in reasonable time over a guaranteed complete model. This makes DeciClareMiner more applicable to real-life scenarios, where time and processing power are limited.

In future research, we will first expand the support of DeciClareMiner for the DeciClare language. Section 6 reports a number of limitations regarding this support, each of which must be resolved. Additionally, a visualisation layer will need to be added when a visual syntax for DeciClare becomes available. The performance of the miner with event logs containing noise will also need to be investigated. DeciClareMiner should theoretically be able to successfully handle the presence of noise in event logs, as stated in Section 5. Nevertheless, this paper does not evaluate this property. The main questions are related to the impact of noise on the efficiency of the miner. Finally, DeciClareMiner will be further improved and evaluated in future iterations of this design science project using it to create a model of the entire emergency department health-care process of a large Belgian hospital.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Steven Mertens <http://orcid.org/0000-0001-8013-8322>
 Frederik Gailly <http://orcid.org/0000-0003-0481-9745>
 Geert Poels <http://orcid.org/0000-0001-9247-6150>

References

- Agrawal, R., & Srikant, R. (1994). Fast algorithms for mining association rules. In *VLDB* (pp. 487–499). doi:10.1.1.40.7506
- Aguilar, E. R., García, F., Ruiz, F., Piattini, M., Calahorra, L., García, M., & Martin, R. (2008). Process modeling of the health sector using BPMN: A case study. In *HEALTHINF 2008* (Vol. 2, pp. 173–178).
- Amorim Lopes, M., Soares, C., Almeida, Á., & Almada-Lobo, B. (2016). Comparing comparables: An approach to accurate cross-country comparisons of health systems for effective health-care planning and policy guidance. *Health Systems*, 5(3), 192–212. doi:10.1057/s41306-016-0001-8
- Birkmeyer, J. D., Finlayson, E. V. A., & Birkmeyer, C. M. (2001). Volume standards for high-risk surgical procedures: Potential benefits of the Leapfrog initiative. *Surgery*, 130(3), 415–422. doi:10.1067/msy.2001.117139
- Braun, R., Schlieter, H., Burwitz, M., & Esswein, W. (2014). BPMN4CP: Design and implementation of a BPMN extension for clinical pathways. *IEEE BIBM 2014* (pp. 9–16). doi:10.1109/BIBM.2014.6999261
- Campbell, H., Hotchkiss, R., Bradshaw, N., & Porteous, M. (1998). Integrated care pathways. *BMJ*, 316(7125), 133–137. doi:10.1136/bmj.316.7125.133
- Cardoso, J., Miller, J. A., & Kochut, K. J. (2003). Health-care enterprise process development and integration integration. *Journal of Research and Practice in Information Technology*, 35(September), 83–98.
- De Bleser, L., Depreitere, R., De Waele, K., Vanhaecht, K., Vlayen, J., & Sermeus, W. (2006). Defining pathways. *Journal of Nursing Management*, 14(7), 553–563. doi:10.1111/j.1365-2934.2006.00702.x
- Di Ciccio, C., Maggi, F. M., & Mendling, J. (2016). Efficient discovery of target-branched declare constraints. *Information Systems*, 56, 258–283. doi:10.1016/j.is.2015.06.009
- Di Ciccio, C., Marrella, A., & Russo, A. (2015). Knowledge-intensive processes: Characteristics, requirements and analysis of contemporary approaches. *Journal on Data Semantics*, 4(1), 29–57. doi:10.1007/s13740-014-0038-4
- Di Ciccio, C., & Mecella, M. (2015). On the discovery of declarative control flows for artful processes. *ACM Transactions on Management Information Systems*, 5(4), Article No: 24. doi:10.1145/2629447
- Faber, B., Konrad, R. A., Tang, C., & Trapp, A. C. (2016). Examining the impact of regular physician visits on heart failure patients: A use case with electronic health data. *Health Systems*, 5(2), 132–139. doi:10.1057/hs.2015.13
- Gawande, A. (2010). *The checklist manifesto*. New Delhi: Penguin Books.
- Goedertier, S., Vanthienen, J., & Caron, F. (2015). Declarative business process modelling: Principles and modelling languages. *Enterprise Information Systems*, 9(2), 161–185. doi:10.1080/17517575.2013.830340
- Helfert, M. (2009). Challenges of business processes management in health-care: Experience in the Irish health-care sector. *Business Process Management Journal*, 15(6), 937–952. doi:10.1108/14637150911003793
- Hofmann, P. B., & Perry, F. (2010). *Management mistakes in health-care: Identification, correction, and prevention*. Cambridge: Cambridge University Press.
- Huang, Z., Dong, W., Ji, L., Gan, C., Lu, X., & Duan, H. (2014). Discovery of clinical pathway patterns from event logs using probabilistic topic models. *Journal of Biomedical Informatics*, 47, 39–57. doi:10.1016/j.jbi.2013.09.003
- Huang, Z., Dong, W., Bath, P., Ji, L., & Duan, H. (2015). On mining latent treatment patterns from electronic medical records. *Data Mining and Knowledge Discovery*, 29(4), 914–949. doi:10.1007/s10618-014-0381-y
- Jiménez-Ramírez, A., & Barba, I. (2013). Generating multi-objective optimized business process enactment plans. In C. Salinesi, M. C. Norrie, & Ó. Pastor (Eds.), *Advanced Information Systems Engineering. CAiSE'13. LNCS* (Vol. 7908, pp. 99–115). Springer.
- Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L. H., Feng, M., Ghassemi, M., ... Mark, R. G. (2016). MIMIC-III, a freely accessible critical care database. *Scientific Data*, 3, 160035. doi:10.1038/sdata.2016.35
- Keel, G., Savage, C., Rafiq, M., & Mazzocato, P. (2017). Time-driven activity-based costing in health care: A systematic review of the literature. *Health Policy*, 121(7), 755–763. doi:10.1016/j.healthpol.2017.04.013
- Lenz, R., & Reichert, M. (2007). IT support for health-care processes - premises, challenges, perspectives. *Data and Knowledge Engineering*, 61, 39–58. doi:10.1016/j.datak.2006.04.007
- Lillrank, P., & Liukko, M. (2004). Standard, routine and non-routine processes in health care. *International Journal of Health Care Quality Assurance*, 17(1), 39–46. doi:10.1108/09526860410515927
- Lismont, J., Janssens, A. S., Odnoletkova, I., vanden Broucke, S., Caron, F., & Vanthienen, J. (2016). A guide for the application of analytics on health-care processes: A dynamic view on patient pathways. *Computers in Biology and Medicine*, 77, 125–134. doi:10.1016/j.compbiomed.2016.08.007
- Maggi, F. M., Dumas, M., García-Bañuelos, L., & Montali, M. (2013). Discovering data-aware declarative process models from event logs. In F. Daniel, J. Wang, & B. Weber (Eds.), *Business Process Management BPM'13, LNCS, BPM'13* (Vol. 8094, pp. 81–96). Beijing: Springer. doi:10.1007/978-3-642-40176-3_8
- Maggi, F. M., Jagadeesh Chandra Bose, R. P., & van der Aalst, W. M. P. (2012). Efficient discovery of understandable declarative process models from event logs. In J. Ralyté, X. Franch, S. Brinkkemper, & S. Wrycza (Eds.), *Advanced Information Systems Engineering. CAiSE'12. LNCS* (Vol. 7328, pp. 270–285). Springer. doi:10.1007/978-3-642-31095-9_18
- Maggi, F. M., Mooij, A. J., & van der Aalst, W. M. P. (2011). User-guided discovery of declarative process models. *CIDM 2011: IEEE Symposium on Computational Intelligence and Data Mining*, 192–199. doi:10.1109/CIDM.2011.5949297
- Mans, R. S. S., Schonenberg, M. H. H., Song, M. S., Van Der Aalst, W. M. P., & Bakker, P. J. M. J. M. (2009). Application of process mining in health-care – A case study in a Dutch hospital. In *BIOSTEC 2009* (Vol. 25, pp. 425–438). Springer. doi:10.1007/978-3-540-92219-3_32
- Mazzocato, P., Savage, C., Brommels, M., Aronsson, H., & Thor, J. (2010). Lean thinking in health-care: A realist review of the literature. *BMJ Quality & Safety*, 19(5), 376–382. doi:10.1136/qshc.2009.037986
- Mertens, S., Gailly, F., & Poels, G. (2017). Towards a decision-aware declarative process modeling language

- for knowledge-intensive processes. *Expert Systems with Applications*, 87, 316–334. doi:10.1016/j.eswa.2017.06.024
- Mitchell, M. (1996). *Introduction to genetic algorithms*. Cambridge, MA: MIT Press.
- Nelson, H. J., Poels, G., Genero, M., & Piattini, M. (2012). A conceptual modeling quality framework. *Software Quality Journal*, 20(1), 201–228. doi:10.1007/s11219-011-9136-9
- Noumeir, R. (2006). Radiology interpretation process modeling. *Journal of Biomedical Informatics*, 39(2), 103–114. doi:10.1016/j.jbi.2005.07.001
- Object Management Group. (2014). *Decision model and notation (DMN1.0)*. Retrieved from <http://www.omg.org/spec/DMN/Current/>
- Payton, F. C., Paré, G., LeRouge, C., & Reddy, M. (2011). Health care IT: Process, people, patients and interdisciplinary considerations. *Journal of the Association for Information Systems*, 12(2).
- Pesic, M. (2008). *Constraint-based workflow management systems: Shifting control to users*. Eindhoven: Technische Universiteit Eindhoven.
- Putman, K., Anstey, A., Harper, P. R., & Knight, V. A. (2016). Modelling of psoriasis patient flows for the reconfiguration of secondary care services and treatments. *Health Systems*, 5(1), 13–20. doi:10.1057/hs.2015.4
- Rashid, M., Naeem, H., Aamir, M., Ali, W., & Ahmed, W. (2017). A multi-level process mining framework for correlating and clustering of biomedical activities using event logs. *International Journal of Advanced Computer Science and Applications*, 8(3), 393–401. doi:10.14569/IJACSA.2017.080354
- Rismanchian, F., & Lee, Y. H. (2017). Process mining-based method of designing and optimizing the layouts of emergency departments in hospitals. *Health Environments Research and Design Journal*, 10(4), 105–120. doi:10.1177/1937586716674471
- Rojas, E., Munoz-Gama, J., Sepúlveda, M., & Capurro, D. (2016). Process mining in health-care: A literature review. *Journal of Biomedical Informatics*, 61, 224–236. doi:10.1016/j.jbi.2016.04.007
- Rojo, M. G., Rolón, E., Calahorra, L., García, F. O., Sánchez, R. P., Ruiz, F., ... Espartero, R. M. (2008). Implementation of the business process modelling notation (BPMN) in the modelling of anatomic pathology processes. *Diagnostic Pathology*, 3(Suppl 1), S22. doi:10.1186/1746-1596-3-S1-S22
- Rotter, T., Kinsman, L., James, E. L., Machotta, A., Gothe, H., Willis, J., ... Kugler, J. (2010). Clinical pathways: Effects on professional practice, patient outcomes, length of stay and hospital costs. *Cochrane Database of Systematic Reviews*, 3, 1–170. doi:10.1002/14651858.CD006632.pub2
- Rovani, M., Maggi, F. M., de Leoni, M., & van der Aalst, W. M. P. (2015). Declarative process mining in health-care. *Expert Systems with Applications*, 42(23), 9236–9251. doi:10.1016/j.eswa.2015.07.040
- Schönig, S., Di Ciccio, C., Maggi, F. M., & Mendling, J. (2016). Discovery of multi-perspective declarative process models. In *Service-oriented computing* (Vol. 9936). doi:10.1007/978-3-319-46295-0
- Svagård, I., & Farshchian, B. A. (2009). Using business process modelling to model integrated care processes: Experiences from a European project. In *IWANN 2009* (Vol. 2, pp. 922–925). doi:10.1007/978-3-642-02481-8_140
- van der Aalst, W. M. P. (2013). Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013, 1–37. doi:10.1155/2013/507984
- van der Aalst, W. M. P., Pesic, M., & Schonenberg, H. (2009). Declarative workflows: Balancing between flexibility and support. *Computer Science - Research and Development*, 23(2), 99–113. doi:10.1007/s00450-009-0057-9
- van der Aalst, W. M. P., Adriansyah, A., Alves de Medeiros, A. K., Arcieri, F., Baier, T., Blickle, T., & Wynn, M. (2011). Process mining manifesto. In *Business Process Management Workshops. BPM'11. LNBIP* (Vol. 99, pp. 169–194). Clermont-Ferrand: Springer. doi:10.1016/j.is.2011.10.006
- Vanhaecht, K., De Witte, K., Depreitere, R., & Sermeus, W. (2006). Clinical pathway audit tools: A systematic review. *Journal of Nursing Management*, 14(7), 529–537. doi:10.1111/j.1365-2934.2006.00705.x
- Wang, T., Tian, X., Yu, M., Qi, X., & Yang, L. (2017). Stage division and pattern discovery of complex patient care processes. *Journal of Systems Science and Complexity*, 30(5), 1136–1159. doi:10.1007/s11424-017-5302-x
- Westergaard, M., & Maggi, F. M. (2012). Looking into the future about timed declarative process models. In *OTM 2012* (Vol. 7565, pp. 250–267). doi:10.1007/978-3-642-33606-5_16