UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS)

Tesis doctoral

MINING COMPLETE, PRECISE AND SIMPLE PROCESS MODELS

Presentada por:

Borja Vázquez Barreiros

Dirigida por:

Manuel Lama Penín
Manuel Mucientes Molina

Santiago de Compostela, septiembre de 2016

Manuel Lama Penín, Profesor Titular del Área de Ciencias de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

Manuel Mucientes Molina, Profesor Contratado Doctor del Área de Ciencias de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

HACEN CONSTAR:

Que la memoria titulada **MINING COMPLETE, PRECISE AND SIMPLE PROCESS MODELS** ha sido realizada por **Borja Vázquez Barreiros** bajo nuestra dirección en el Centro Singular de Investigación en Tecnoloxías da Información de la Universidade de Santiago de Compostela, y constituye la Tesis que presenta para obtar al título de Doctor.

Santiago de Compostela, septiembre de 2016

Manuel Lama Penín
Director de la tesis

Manuel Mucientes Molina
Director de la tesis

Borja Vázquez Barreiros
Autor de la tesis

Manuel Lama Penín, Profesor Titular del Área de Ciencias de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

Manuel Mucientes Molina, Profesor Contratado Doctor del Área de Ciencias de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

como Director/res de la tesis titulada:

MINING COMPLETE, PRECISE AND SIMPLE PROCESS MODELS

Por la presente INFORMAN:

Que la tesis presentada por Don **Borja Vázquez Barreiros** es idónea para ser presentada, de acuerdo con el artículo 41 del *Regulamento de Estudos de Doutoramento*, por la modalidad de compendio de ARTÍCULOS, en los que el doctorando ha tenido participación en el peso de la investigación y su contribución fue decisiva para llevar a cabo este trabajo. Y que está en conocimiento de los coautores, tanto doctores como no doctores, participantes en los artículos, que ninguno de los trabajos reunidos en esta tesis serán presentados por ninguno de ellos en otras tesis de Doctorado, lo que firmamos bajo nuestra responsabilidad.

Santiago de Compostela, septiembre de 2016

Manuel Lama Penín
Director de la tesis

Manuel Mucientes Molina
Director de la tesis

I love deadlines. I like the whooshing sound they make as they fly by.

Douglas Adams

You know it is. You pick up a book, flip to the dedication, and find that, once again, the author has dedicated a book to someone else and not to you.
Not this time.

- Neil Gaiman

Agradecimientos

En primer lugar, mi más sincero agradecimiento a mis directores de tesis, Manuel Mucientes y Manuel Lama, por depositar su confianza en mi para llevar a cabo esta tesis. Gracias por vuestros consejos y ayuda brindada a lo largo de todos estos años.

Dar las gracias al Departamento de Electrónica y Computación, así como al Centro singular de Investigación en Tecnoloxías da Información (CiTIUS), de la Universidad de Santiago de Compostela por los medios y recursos proporcionados para la realización de la tesis.

Aprovecho para mostrar mi agradecimiento al Dr. Wil van der Aalst por acogerme en su grupo durante mi estancia predoctoral, y permitirme disfrutar de una experiencia inolvidable tanto personal, como profesionalmente. Gracias a toda la gente que conocí allí durante ese tiempo y que me hizo sentir *coma na casa*.

También me gustaría expresar mi gratitud a las entidades que han financiado el desarrollo de esta investigación. Concretamente, agradecer el soporte económico recibido por parte del Ministerio de Ciencia e Innovación a través del proyecto "SoftLearn: Soft computing para minería de procesos en e-learning" (TIN2011-22935) y del Ministerio de Economía y Competitividad, cofinanciado por el Fondo Europeo de Desarrollo Regional - FEDER, a través del proyecto "BAI4SOW: Soft Computing for supporting Business Intelligence in Social Workflows" (TIN2014-56633-C3-1-R).

Imposible olvidarse de los irremplazables compañeros de laboratorio. Gracias por estar ahí durante todos estos años.

Gracias también a toda esa gente que por un motivo u otro forma parte de anécdotas y momentos inolvidables de mi vida durante estos años.

Y, por supuesto, a mi familia: a mis padres y a mis hermanos, gracias; gracias por vuestro continuo apoyo en los buenos y malos momentos. Sin vosotros no podría ser quién soy ni estar donde estoy.

Santiago de Compostela, septiembre de 2016

Resumen

En los últimos años ha habido una increíble inversión en el desarrollo de tecnologías para automatizar las diferentes tareas que se desarrollan en una organización, así como para almacenar toda la información posible generada durante estas tareas. En particular, en relación con los procesos de negocio, esto ha dado lugar a un increíble crecimiento en la cantidad de datos relacionados con los procesos de negocio, es decir, información relacionada con la ejecución de las actividades empresariales. Es evidente que la explosión de este tipo de datos ha abierto una puerta para proporcionar información sobre la forma real de trabajar en una organización, para predecir el rendimiento mediante la simulación, para la detección de desviaciones en el proceso o para mejorar la manera en que ciertas actividades comerciales se ejecutan.

En este contexto, un proceso de negocio o modelo de proceso, se entiende como un conjunto de actividades estructuradas y relacionadas entre sí que producen un resultado específico, por ejemplo, un producto o un servicio. Estas actividades son llevadas a cabo por un conjunto de agentes para lograr el propósito del proceso. Por ejemplo, en el ámbito educativo, el diseño de un curso de aprendizaje es un proceso en el que los alumnos deben realizar una serie de actividades educativas, por ejemplo, publicar en foros, hacer ejercicios y exámenes, etc., con el fin de alcanzar los objetivos pedagógicos del curso. Por lo general, estos procesos tienen una descripción detallada, es decir, hay un diseño del proceso en el que se describen claramente sus actividades y los actores que participan en estos pasos. En esencia, los procesos se diseñan generalmente para obligar a las personas o máquinas a trabajar de una manera particular. Desafortunadamente, puede haber diferencias entre el modelo de proceso diseñado, y cómo el proceso se está ejecutando en la realidad. Por ejemplo, siguiendo con el ejemplo anterior, en el ámbito educativo, los estudiantes pueden llevar a cabo actividades de aprendizaje adicionales, como revisar la bibliografía o interactuar con otros estudiantes, a mayores de las que se especifica explícitamente en el proceso de aprendizaje diseñado por un profesor.

Así, en este escenario, la minería de procesos ha surgido como una manera de analizar el comportamiento de una organización mediante la extracción de conocimiento a partir de los registros de eventos, es decir, los datos relacionados con el proceso, y ofreciendo técnicas para descubrir, monitorizar y mejorar los procesos reales. En otras palabras, la minería de procesos permite entender *lo que realmente está sucediendo en un proceso de negocio, y no lo que creemos que está sucediendo*. Así, la minería de procesos tiene como objetivo modelar el *mundo real* que está soportado y controlado por varios tipos de sistemas de información. Para ello, se toman como punto de partida los datos de eventos almacenados en forma de registros de sucesos por los sistemas de información durante la ejecución de los procesos definidos.

Por lo general, las técnicas de minería de procesos se clasifican en tres grupos principales. El primer grupo es el de *Descubrimiento*, cuyo objetivo es recuperar los diferentes tipos de modelos a partir de la información de un registro de eventos. Por ejemplo, a través de técnicas de descubrimiento es posible recuperar un modelo en formato de red de Petri que represente y explique el comportamiento almacenado en un registro de eventos. El segundo grupo se conoce como *Análisis de conformidad*, donde un modelo de proceso se compara con un registro de eventos del mismo proceso para analizar y cuantificar las diferencias entre el comportamiento esperado y el real. Por ejemplo, en la fabricación de un teléfono móvil, éste siempre se ha de revisar en las últimas etapas del montaje. A través de los análisis de conformidad es posible detectar si esta regla se cumple o no. Por lo tanto, este tipo de técnicas se utilizan generalmente para detectar, localizar, explicar y medir la gravedad de las desviaciones entre el proceso real y lo esperado. Por último, el tercer grupo de técnicas de minería de proceso es el de *Extensión o mejora*. En este grupo, un modelo de proceso se mejora de forma dinámica, enriqueciéndolo o extendiéndolo utilizando un registro de eventos nuevo, u otro modelo de proceso. Mientras que el análisis de conformidad tiene por objetivo detectar y cuantificar las desviaciones, la mejora de los procesos trata de extender y/o cambiar el modelo de proceso a priori. Un ejemplo de mejora es la *reparación*, es decir, modificar un modelo de proceso para representar mejor la realidad. Por ejemplo, modificar un paso obligatorio en un proceso que se modeló inicialmente como opcional. Otro tipo de mejora es la *extensión*, es decir, añadir nueva información a un modelo de proceso. Por ejemplo, la adición de la dimensión temporal a un proceso para mostrar cuellos de botella, tiempos de ejecución, o para las predicciones de tiempo. Otro tipo de extensiones puede implicar la adición de información sobre recursos, reglas de decisión, métricas de calidad, etc.

Como se puede ver, los tres grupos en los que se divide la minería de procesos comparten

un componente clave: un registro de eventos. Una secuencia de eventos, por ejemplo, la ejecución de las actividades relacionadas con una instancia de proceso, se conoce como *traza*. De este modo, en esencia, un registro de eventos es un grupo de trazas que consisten en eventos. Desde el punto de vista de la minería de procesos, es un requisito mínimo que los eventos en el registro de eventos tengan asignados *i)* un *caso*, *ii)* una *actividad*, y *iii)* un *punto en el tiempo*. Teniendo esta información disponible, es posible, por ejemplo, descubrir un modelo de proceso, es decir, cuál es el orden entre las actividades de un proceso. Este tipo de técnicas que permite descubrir las dependencias entre tareas se conoce como *descubrimiento de procesos*, y es el área de interés de la presente Tesis Doctoral.

El descubrimiento de procesos se utiliza para descubrir el proceso subyacente que se ha seguido para alcanzar un objetivo. En general, los algoritmos que buscan descubrir este modelo de proceso subyacente parten de un registro de eventos y no tienen en cuenta el conocimiento de dominio para derivar el modelo de proceso, lo que permite aplicarlos de manera general. Sin embargo, dependiendo de la técnica utilizada, es posible obtener diferentes modelos de proceso, ya que cada técnica tiene sus fortalezas y debilidades —por ejemplo, la expresividad de la notación utilizada. Por lo tanto, es importante tener en cuenta los requisitos del dominio al decidir qué algoritmo se ha de utilizar, ya que la correcta selección puede dar lugar a modelos de proceso más ricos.

Por ejemplo, de entre los diferentes ámbitos de aplicación de la minería procesos, es posible identificar varios campos que comparten un requisito interesante acerca de cómo deben ser los modelos de proceso. Por ejemplo, en las auditorías de seguridad, los modelos descubiertos deben cumplir requisitos estrictos. Esto significa que los modelos de proceso descubiertos deben reproducir todo el comportamiento posible recogido en el registro de eventos, o de lo contrario es posible que algunas desviaciones pasen desapercibidas (*completitud*). Por otra parte, con el fin de evitar falsos positivos, los modelos de proceso deben reproducir solamente el comportamiento registrado en el registro de eventos (*precisión*). Por último, los modelos de procesos deben ser fácilmente legibles para detectar más fácilmente las desviaciones (*simplicidad*). Otro ejemplo claro de este tipo de requisitos en un modelo de proceso lo podemos encontrar en el ámbito educativo, ya que para que un modelo de proceso sea de valor para los profesores y estudiantes, éste debe de satisfacer los requisitos antes mencionados. Es decir, para garantizar unas correctas evaluaciones, los profesores tienen que acceder a todas las actividades realizadas por los alumnos, con lo que el proceso de aprendizaje debe ser capaz de reproducir todo el comportamiento posible (*completitud*). Por otra parte, el proceso de apren-

dizaje debe centrarse únicamente en el comportamiento almacenado en el registro de eventos (*precisión*), es decir, *centrarse únicamente en lo que hicieron los estudiantes, y no en lo que podrían haber hecho*. Por último, los modelos de proceso deben ser fácilmente interpretables por los profesores (*simplicidad*).

Uno de los requisitos anteriores se relaciona con la legibilidad de los modelos de proceso: la *simplicidad*. En la minería de procesos, uno de los problemas identificados es la adecuada visualización de los modelos de proceso, es decir, cómo representar los resultados del descubrimiento de procesos de tal manera que se pueda obtener información del mismo de forma clara y concisa. Esto se deba a que los modelos de proceso que son innecesariamente complejos, pueden dificultar la correcta lectura del modelo, en lugar de proporcionar una intuición de lo que realmente ocurre en una organización. Dentro de los diferentes enfoques centrados para reducir la complejidad de un modelo de proceso, el interés de esta Tesis Doctoral se centra en dos técnicas. Por un lado, mejorar la legibilidad de un modelo de proceso ya descubierto a través de la inclusión de etiquetas duplicadas. Por otro lado, la *jerarquización* de un modelo de proceso, es decir, proporcionar al modelo de proceso una estructura bien conocida dentro del dominio. Sin embargo, con respecto a este último objetivo, este tipo de técnicas requiere tener en cuenta conocimiento del dominio, ya que diferentes dominios pueden depender de diferentes requisitos de la hora de mejorar la legibilidad del modelo de proceso. En otras palabras, con el fin de mejorar la comprensibilidad de un modelo de proceso, la jerarquización tiene que ser impulsada por el dominio.

Para resumir, podemos identificar dos temas principales de interés en la presente Tesis Doctoral. Por un lado, el descubrimiento de modelos de proceso que puedan reproducir todo el comportamiento posible almacenado en un registro de eventos, sin introducir comportamiento extra. Por otro lado, se pretende reducir la complejidad de los modelos extraídos con el fin de mejorar su interpretabilidad. De este modo, el objetivo principal de la presente tesis se puede resumir en: descubrir modelos de procesos teniendo en cuenta la completitud, precisión y simplicidad, prestando especial atención en la recuperación de modelos de procesos altamente interpretables.

Dentro del área de investigación del descubrimiento de modelos de proceso, se han abordando una gran cantidad de soluciones desde diferentes puntos de vista. Sin embargo, la revisión del estado del arte muestra que muchos algoritmos de descubrimiento de procesos se centran en una o dos de las dimensiones previamente mencionadas, o que la notación utilizada no permite representar todo el comportamiento posible. En otras palabras, las técnicas

actuales tienen dificultades para recuperar modelos con altos niveles de completitud, pero siendo lo más precisos y sencillos posible. Considerando este escenario, el Capítulo 2 aborda la problemática del descubrimiento de procesos bajo la hipótesis: ¿Es posible descubrir modelos de proceso de alta calidad, en dominios generales, a través de un *criterio jerárquico de búsqueda que se base en la completitud, precisión y simplicidad*? Así, en este capítulo se presenta ProDiGen, un algoritmo genético para el descubrimiento de procesos guiado por la completitud, precisión y simplicidad. El algoritmo utiliza una función de evaluación jerárquica que tiene en cuenta la completitud, la precisión y la simplicidad (con nuevas definiciones tanto para la precisión y la simplicidad). Además, el algoritmo utiliza heurísticas para optimizar los operadores genéticos: *i)* un operador de cruce que selecciona el punto de cruce a partir de una función de densidad de probabilidad (PDF) generada a partir de los errores del modelo extraído, y *ii)* un operador de mutación guiado por las dependencias causales del registro de eventos. La validación del algoritmo se ha realizado con 39 modelos diferentes y los resultados se han comparado con cuatro algoritmos del estado del arte, demostrando, estadísticamente, que el uso de una evaluación jerárquica permite obtener modelos con una gran calidad en términos de completitud, precisión y simplicidad.

Por otro lado, la visualización de un modelo de proceso juega un papel importante a fin de obtener correctamente conocimientos sobre el modelo de proceso. Es decir, los modelos de procesos que son demasiado complejos, pueden obstaculizar el comportamiento real del proceso en lugar de proporcionar información de lo que realmente está sucediendo. Dentro de este contexto, la presente Tesis Doctoral se centra en el concepto de *tareas duplicadas*. Dentro de este área de estudio de las tareas de duplicadas se han obtenido resultados muy valiosos, sin embargo, el estado del arte presenta diferentes debilidades: algunos algoritmos obtienen soluciones peores que sin tareas duplicadas; otras técnicas permiten duplicar cualquier actividad en el registro eventos; y otras propuestas utilizan heurísticas que no consideran las actividades duplicadas en algunos patrones de flujo de trabajo, tales como bucles. Así, dentro de este contexto, el Capítulo 3 trata este área de investigación bajo la hipótesis: ¿Es posible mejorar la claridad estructural de un modelo de proceso mediante la duplicidad de actividades de un modelo de proceso *después* del proceso de descubrimiento? Con este fin se ha desarrollado SLAD, un algoritmo que toma como punto de partida un modelo ya minado y, a través de la información local del registro de eventos, trata de mejorar la completitud, la precisión y la simplicidad del modelo duplicando aquellos nodos con una mayor densidad de arcos. Con el fin de validar el algoritmo, se presenta una evaluación detallada con 54 modelos extraídos de

tres algoritmos de descubrimiento de procesos. Además, los resultados se han comparado con ocho algoritmos diferentes del estado del arte, demostrando, estadísticamente, que añadiendo tareas duplicadas tras descubrir un modelo de proceso presenta mejores resultados, en términos de precisión y simplicidad, que las técnicas del estado del arte actual.

Siguiendo con la idea de mejorar la visualización de un modelo de proceso, la última sección de la presente Tesis Doctoral busca mejorar este aspecto con la inclusión de conocimiento de dominio utilizando técnicas de jerarquización. En otras palabras, la jerarquización de procesos tiene como objetivo utilizar el conocimiento del dominio para *traducir* un modelo de proceso ya descubierto a una representación más específica dentro de un dominio particular y, por lo tanto, obtener un modelo de proceso más interpretable. Esta idea de *jerarquización* de procesos utilizando conocimiento del dominio es, dentro de nuestro conocimiento, una aproximación novedosa, sin trabajo previo directamente relacionado en el campo de la minería proceso. No obstante, sigue las mismas ideas que los métodos que tienen como objetivo descomponer modelos de proceso. Sin embargo, las técnicas de descomposición se centran en la partición de modelos de proceso y de registros de eventos en partes más pequeñas que se pueden analizar de forma independiente. Por lo tanto, no devuelven una representación jerárquica del modelo de proceso, sino una representación todavía aplanada del modelo de proceso original.

De este modo, la hipótesis que se presenta en el Capítulo 4 queda definida como: ¿Es posible jerarquizar *automáticamente* un modelo de proceso ya descubierto, utilizando el conocimiento de dominio y, por lo tanto, mejorar la interpretabilidad del modelo de proceso? En este capítulo se describe cómo se consigue este objetivo en tres pasos diferentes. En primer lugar, el modelo de proceso se extrae automáticamente de las secuencias registrados a través de los algoritmos de minería de procesos desarrollados. A continuación, se aplica un algoritmo basado en conocimiento sobre la estructura de control de destino para determinar qué componentes deben ser creados. Por último, se extraen automáticamente las reglas de adaptación a partir de los registros de eventos —más específicamente, a partir de los valores de las variables de los registros de eventos— a través de un algoritmo de aprendizaje de tipo árbol de decisión, y se integran en la estructura del lenguaje destino. En este campo, a pesar de que buscamos una aproximación general independiente de cualquier dominio en particular, la validación e implementación de este proceso de jerarquización de modelos de proceso se ha llevado a cabo en el campo educativo. El catalizador de esta decisión se debe a la existencia de IMS LD, un estándar de metadatos que describe todos los elementos del diseño de un pro-

ceso de enseñanza-aprendizaje. Por lo tanto, los diferentes componentes de este proceso de jerarquización se han analizado utilizando un conjunto de nueve cursos reales con diferentes grados de complejidad. Los resultados experimentales mostraron que la minería proceso es una buena solución para recuperar una estructura de proceso de un conjunto de archivos de registro de eventos. Concretamente, en todos los casos se extrajo correctamente la estructura IMS LD original.

Finalmente, la tesis termina con las conclusiones y el trabajo futuro presentados en el Capítulo 5.

Contents

CHAPTER 1

INTRODUCTION

1.1 Motivation

In the last years, there has been an incredible investment to develop technologies to automate all the different tasks carried out in an organization and to store all the information generated during these processes. In particular, regarding business processes, the area of interest in this dissertation, this has lead to an incredible growth on the amount of process-related data, i.e, execution traces of business activities. Hence, with this flood of information in today's organizations, there has raised a special interest, or even a need, for extracting valuable knowledge from such process-related information [68], and thereby to understand what data really say. Clearly, the explosion of this kind of information has opened the door to provide insights into the actual way of working in an organization, to detect deviations in the process or to improve the way certain business activities are executed.

In this context, a business process, henceforth a process or process model, is understood as a collection of related structured activities that produce a specific outcome, e.g., a product or a service [132]. These activities are performed by a set of actors to achieve the purpose of the process. For instance, in education, the learning design of a course is a process where learners must undertake a sequence of learning activities, e.g., posting in forums, making exercises and exams, etc., in order to achieve the pedagogical objectives of the course. Typically, these processes have a detailed description, i.e., there is a design of the process where its activities and the actors participating in these steps are clearly described. In essence, processes are usually designed to force people or machines to work in a particular way. Unfortunately, there might be differences between the designed process model, and how the process is being executed

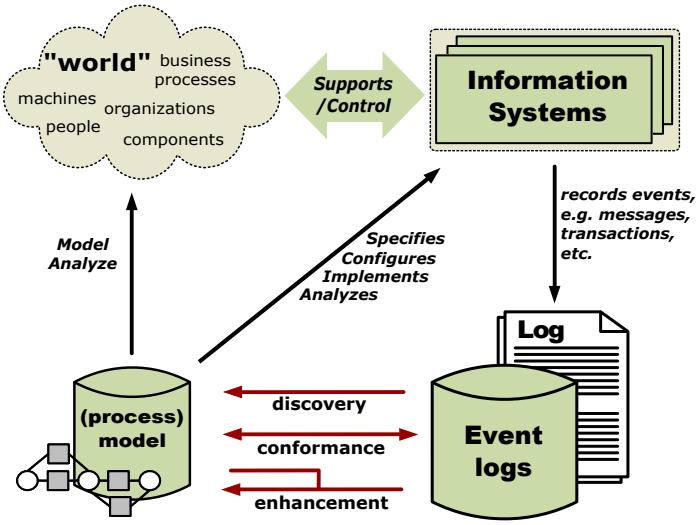
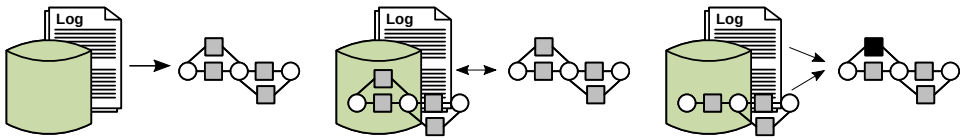


Figure 1.1: Process mining framework (Adapted from [122]).

in reality [119, 122]. For instance, following with the previous example in the educational domain, learners can undertake additional learning activities, like check the bibliography or interact with other learners, apart from those that were explicitly specified in the learning process designed by a teacher. The reasons behind this gap are manifold. It might be that there were human errors while implementing the process model in the information system. Maybe, the system allows to skip an activity, to change the order in which some activities must be executed, or to execute activities not defined in the process. Perhaps a particular instance of a process needs to be restarted or prematurely canceled with no reasons. In a nutshell, creating process models is a difficult and error-prone task, and there is an evident gap between *what we think is going on* (the designed process model) and *what is really happening* (the real process model). This deprecates the actual value of process models in organizations [122].

In this scenario, process mining has emerged as a way to analyse the behavior of an organization by extracting knowledge from event logs, i.e., process-related data, and by offering techniques to discover, monitor and enhance real processes. In other words, process mining allows us to understand *what is really happening in a business process, and not what we think is going on* [122]. Figure 1.1 depicts the process mining framework. The general view is that process mining aims to model the *real world* which is supported and controlled by various



(a) Discovery. (b) Conformance checking. (c) Enhancement.

Figure 1.2: Types of tasks in process mining (Adapted from [16]).

types of information systems. To do that, it takes as starting point the event data stored in the form of event logs by the information systems while executing the defined process. Hence, process mining provides links between the actual observed process execution and the modeled process behavior. Typically, process mining techniques have been categorized in three different main groups (Figure 1.2):

- The first group is *Discovery* (Figure 1.2(a)), which aims to retrieve different kinds of models based on the information of an event log. For instance, through discovery techniques it is possible to retrieve a Petri net model explaining the behavior recorded in the log. In general, discovery techniques do not take into account any domain knowledge to derive models, although there are a few techniques that assume a-priori knowledge to retrieve a model [10, 38, 48, 97].

- The second group is *Conformance Checking* (Figure 1.2(b)), also known as *conformance analysis*, where a process model is compared with a log of the same process to analyse and quantify the differences between the expected and the real behavior as recorded in the log. For instance, when manufacturing a mobile phone, it has always to be checked at the end of the assembly process. Through conformance checking it is possible to detect if this rule is followed or not. Hence, these kind of techniques are generally used to detect, locate, explain and measure the severity of the deviations.

- The third group of process mining techniques is *Enhancement* (Figure 1.2(c)). In this group, a process model is dynamically improved, enriched or extended based on the information of a (new) log and/or a defined process model. Whereas conformance checking aims to detect and quantify the deviations, process enhancement tries to extend and/or change the a-priori process model. An example of enhancement is *repair*, i.e., to modify a process model to better depict reality. For example, correcting a process where a particular activity was modeled as optional, i.e., it could be skipped, but in

Table 1.1: A fragment of an event log loosely based on a fictional loan application process [40], where each individual line corresponds to an event.

Case	Activity	Resource	Time-stamp	···
⋮	⋮	⋮	⋮	⋮
3554	*Check application form*	*John*	*2016-10-08T09:45:37+00:00*	···
3555	*Check application form*	*Lucy*	*2016-10-08T10:12:37+00:00*	···
3554	*Check credit history*	*Harold*	*2016-10-08T10:14:25+00:00*	···
3555	*Check credit history*	*Harold*	*2016-10-08T10:31:02+00:00*	···
3554	*Appraise property*	*Pete*	*2016-10-08T10:45:22+00:00*	···
3554	*Assess loan risk*	*Harold*	*2016-10-08T10:49:52+00:00*	···
3555	*Assess loan risk*	*Harold*	*2016-10-08T11:01:51+00:00*	···
3553	*Return application to client*	*John*	*2016-10-08T11:03:18+00:00*	···
3556	*Check application form*	*Lucy*	*2016-10-08T11:05:10+00:00*	···
3555	*Assess eligibility*	*Harry*	*2016-10-08T11:06:22+00:00*	···
3554	*Assess eligibility*	*Harry*	*2016-10-08T11:33:42+00:00*	···
3554	*Reject application*	*Harry*	*2016-10-08T11:45:42+00:00*	···
3557	*Check application form*	*Lucy*	*2016-10-08T13:48:12+00:00*	···
3555	*Prepare acceptance pack*	*Sue*	*2016-10-08T14:02:22+00:00*	···
⋮	⋮	⋮	⋮	⋮

reality is a mandatory step. Another type of enhancement is *extension*, i.e., add new information to a process model, for instance, adding the timestamps to a process to show bottlenecks, throughput times, or for time prediction [130]. Other type of extensions can involve adding information about resources, decision rules, quality metrics, etc.

As can be seen, all the different groups in which process mining is divided share one component: an *event log*. Consider Table 1.1 showing a snapshot of an event log of a fictional loan application handling process. We can see that each event, i.e., each individual row, refers to an *activity*. Furthermore, it refers to one *case*, i.e., a process instance, it has a *timestamp*, and it shows which *resource* executed the task. For instance, let us consider all activities related to the *case 3554*. First, John *Checks the application form*, after which Harold *Checks the applicant's credit history*. Pete *Appraises the property*, after which Harold performs a *Loan risk assessment*. Finally, Harry *assesses the eligibility of the client for the loan* and, at the end, he decides to *Reject the application*. A sequence of events, e.g., the execution of the activities related to a process instance, e.g., the case *3554*, is referred to as a *trace*. Thereby, in essence, an event log is a group of traces that consist of events.

From the point of view of process mining, it is a minimum requirement that the events in the log refer to *i)* a *case*, *ii)* an *activity*, and *iii)* a *point in time* (timestamp). Having this information available, it is possible, for instance, to discover a process model from an event log. In practice, the actual data stored within a company's information system might not necessarily be of the form presented in Table 1.1. Hence, the extracted data often needs to be transformed. Anyway, such type of required data is present in almost any information system.

Often, a wealth of additional attributes are available in an event log, e.g., *customer id*, *credit balance*, etc. In situations where the log provides a rich source of data, analysts can decide upon various log perspectives [132] to drive the execution of a process mining technique. We can distinguish the following perspectives [106, 107, 109, 108, 110] regarding the analysis of an event log:

- *Control flow perspective*. The most common perspective. It captures aspects related to control-flow dependencies between various tasks, i.e., the ordering of activities in the event log. For example, *"check the credit history always after checking the application form"*.

- *Resource perspective*. It focuses on how the resources, i.e., human (e.g., people) or non-human (e.g., equipment) actors, are involved and related to the process. *"Assessing the loan risk has to be made by the manager"* is an example handled by this perspective.

- *Data perspective*. It deals with the flow of data through the process, i.e., the passing of information between activities, the scope of variables, etc. An example of this perspective would be *"if the credit balance is negative, reject the application"*.

- *Exception handling perspective*. This perspective focuses on problems and failures. In other words, it deals with the various causes of exceptions and the various actions that need to be taken as a result of exceptions, e.g., *"if the application has an error, call the client for an amendment"*.

- *Time perspective*. This perspective is related to time and performance aspects, e.g., bottlenecks, monitor the utilization of resources, etc. For example, *"in January, checking the application takes more than one day"*.

Within this dissertation we primarily focus on the control-flow perspective, i.e., the sequential ordering of *activities* w.r.t. *cases*. Thus, from the *control-flow perspective*, case *3554* can be

written as ⟨*Check application form*, *Check credit history*, *Appraise property*, *Assess loan risk*, *Assess eligibility*, *Reject application*⟩.

Note that the different perspectives presented so far are somewhat overlapped. For instance, when detecting an error (exception handling) it may be interesting to check the time related to that failure (time perspective). Furthermore, there exists an orthogonal relation between the process mining tasks previously described in Figure 1.1, i.e., discovery, conformance checking and enhance, and the different perspectives regarding the analysis of an event log [128]. For instance, discovery combined with the control-flow perspective can give as a result a process model represented as a Petri Net or a BPMN model [26, 79, 85, 112, 126]. But it can also be combined with the resource perspective to discover a social network to check, for instance, the *handover of work* between resources [128, 131]. On the other hand, enhancement [64, 84] can be associated with the time perspective, extending a process model to show the overall efficiency, i.e., bottlenecks, throughput times, etc. Also, conformance checking and control-flow can lead to replay or alignment techniques [6, 15, 86]. Concerning this PhD Thesis, our interest relies on the *discovery* group combined with the *control-flow* perspective, i.e., we are interested in *process discovery* [7, 27].

Topics of interest

Process discovery algorithms are generally used to discover the underlying process that has been followed to achieve an objective. In general, these algorithms do not take into account any domain knowledge to derive process models, allowing to apply them in a general manner. However, as we will see through Section 1.2, depending on the selected approach, a different kind of process models can be discovered, as each technique has its strengths and weaknesses, e.g., the expressiveness of the used notation [120]. Hence, it is important to take into account the requirements of the domain when deciding which algorithm to use, as the correct assumptions can lead to richer process models.

For instance, among the different domains of application of process mining [80, 115, 117, 142, 145], we can identify several fields that share an interesting requirement about the discovered process models. In security audits [3, 4], discovered processes have to fulfill strict requisites. This means that the process model should reproduce as much behavior as possible, otherwise some violations may go undetected (*replay fitness*). On the other hand, in order to avoid false positives, process models should reproduce only the recorded behavior (*precision*). Finally, process models should be easily readable to better detect deviations (*simplicity*). An-

other clear example concerns the educational domain [146], as in order to be of value for both teachers and learners, a discovered learning process should satisfy the aforementioned requirements [100]. That is, to guarantee feasible and correct evaluations, teachers need to access to all the activities performed by learners, thereby the learning process should be able to reproduce as much behavior as possible (*replay fitness*). Furthermore, the learning process should focus on the recorded behavior seen in the event log (*precision*), i.e., *show only what the students did, and not what they might have done*, while being easily interpretable by the teachers (*simplicity*). Section 1.2 provides a more detailed view on these characteristics.

One of the previous requirements is related to the readability of process models: *simplicity*. In process mining, one of the identified challenges is the appropriate visualization of process models [135], i.e., to present the results of process discovery in such a way that people actually gain insights about the process. Process models that are unnecessary complex, can hinder the real behavior of the process rather than to provide an intuition of what is really happening in an organization. However, achieving a good level of readability is not always straightforward, for instance, due the used representation [122]. Within the different approaches focused to reduce the complexity of a process model [34, 44], the interest in this PhD Thesis relies on two techniques. On the one hand, to improve the readability of an already discovered process model through the inclusion of duplicate labels. Section 1.3 provides a more detailed view on this idea. On the other hand, the *hierarchization* of a process model, i.e., to provide a well known structure to the process model. However, regarding the latter, this technique requires to take into account domain knowledge, as different domains may rely on different requirements when improving the readability of the process model. In other words, in order to improve the interpretability and understandability of a process model, the hierarchization has to be driven by the domain. Section 1.4 provides the intuition, in the educational domain, on how a discovered process model can be *hierarchized* to better describe the structure of a *learning process*.

To sum up, concerning the aim of this PhD Thesis, we can identify two main topics of interest. On the one hand, we are interested in retrieving process models that reproduce as much behavior recorded in the log as possible, without introducing unseen behavior. On the other hand, we try to reduce the complexity of the mined models in order to improve their readability. Henceforth, the aim of this PhD Thesis is to:

> *Discover process models considering replay fitness, precision and simplicity, while paying special attention in retrieving highly interpretable process models.*

Throughout the next sections we will review the state of the art related with these objectives.

1.2 Process discovery

The goal of process discovery is to obtain a process model that specifies the relations between tasks or activities in an event log, i.e., to describe the ordering and flow of events that occur in a process [132, 134]. From the control-flow perspective, this ordering and flow of the activities can be represented with several workflow patterns, e.g., activities can be executed sequentially, activities can be skipped, activities can be concurrent or mutual exclusive, or the same activity can be executed multiple times.

In general, the representation used for most process discovery algorithms are Petri nets [88], as they are simple and graphical, while allowing for modeling concurrency. Nevertheless, there is a plethora of process models notations in the literature (oftentimes referred as the *Tower of Babel* [122]), each one of them with a different level of expressiveness and popularity [15, 122]: BPMN, C-nets, Process Trees, EPCs, etc[1]. In [120] the authors discuss that the notation for discovering a process should be driven by its properties, and not by its graphical representation. Concerning this thesis, we are interested in Heuristic nets [154, 155] and Causal nets [114, 153]. Both of these notations are especially tailored towards process discovery [122, 123], allowing to produce rich and high quality results. Furthermore, these two kind of models are easily translated to both Petri nets and BPMN.

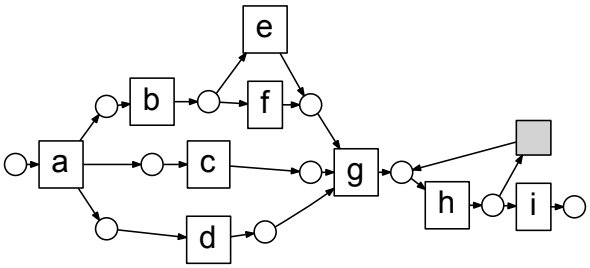
Figure 1.3 shows a simplified event log for a phone production. This log contains nine different labeled activities, and 1,134 cases represented by nine different traces. In this process, when producing a mobile phone, three different parts are manufactured: *i)* the frame (d = *Produce Frame*), *ii)* the keyboard (c = *Produce Keyboard*), and *iii)* the cover (b = *Produce Cover*), which is painted in black (e = *Paint Black*) or white (f = *Paint White*). After these three parts are completed, the phone is assembled (g = *Assemble Phone*). Finally, the mobile phone goes through a quality control (h = *Check Phone*), multiple times if necessary, before finishing its production.

Based on the event log shown in Figure 1.3(a), a discovery algorithm could retrieve the Petri net model depicted in Figure 1.3(b). This process model can replay all traces in the log, but it can also reproduce more behavior than the recorded. For instance, the firing sequence $\langle a, c, b, f, d, g, h, i \rangle$ is not in the event log. Moreover, there are infinitely many firing sequences

[1] In general, all these notations can be translated to a Petri net.

Trace	#
a b c d e g h i	212
a c b d e g h h i	193
a b d c f g h i	185
a d c b f g h i	146
a b d f c g h h i	105
a b e d c g h i	94
a b d c e g h i	90
a d b c f g h h h i	61
a d c b f g h i	48

(a) Event log.



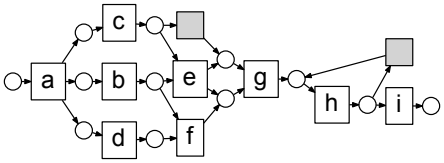
(b) A Petri net discovered for the left log.

Figure 1.3: A log and a process model for the production of a mobile phone. (a = Start Production, b = Produce Cover, c = Produce Keyboard, d = Produce Frame, e = Paint Black, f = Paint White, g = Assemble Phone, h = Check Phone, i = End Production).

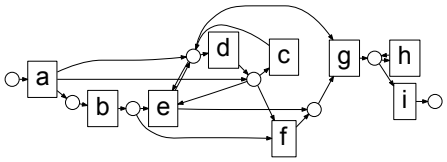
due to the loop construct in *h* (*Check Phone*). At this point, and without more information, this solution appears to be a *representative* process model of the behavior seen in the event log. However, this process model is not the only possible solution. For instance, Figure 1.4 shows four different solutions depicting the behavior of the same log (Figure 1.3(a)). Thereby, more elaborate criteria are necessary to quantify the *goodness* of a process model.

When deciding how good a process model depicts the recorded behavior, i.e., how representative is a process for the behavior seen in the log, we can distinguish four quality dimensions or metrics. The first one is *replay fitness*. It quantifies how much of the behavior observed in the log can be reproduced by a process model. A process model has a perfect replay when it can reproduce all the behavior in the event log. For instance, the process model in Figure 1.4(a) cannot replay all the traces of the log, e.g., it is not possible to correctly reproduce $\langle a, b, e, d, c, g, h, i \rangle$. All the remaining processes in Figure 1.4 have a perfect replay fitness. The second dimension is *precision*, which assesses the ability of the model to disallow unwanted behavior. Clearly, the *flower model*[2] is an example of lack of precision, i.e., it is an overly general process model. The process model in Figure 1.4(d) is an example of a solution with a very high precision, as it almost only reproduces the behavior recorded in the log. The third dimension is *generalization* which estimates the extent to which a process model will be

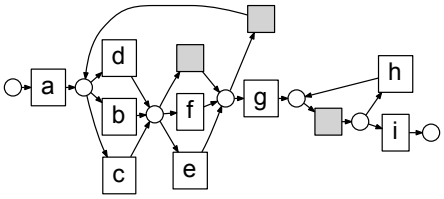
[2]The *flower model* allows the execution of any possible combination of activities in the log; hence it always has a perfect replay fitness.



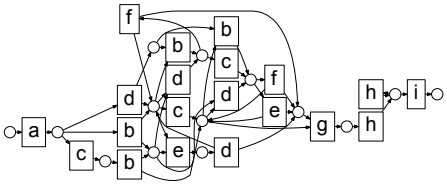
(a) A Petri net discovered with Heuristics Miner [155] using the default settings.



(b) A Petri net discovered with ILP [138] miner using the default settings, ensuring an empty net after completion.



(c) A Petri net discovered with Inductive Miner [70], with the 0% noise threshold.



(d) A Petri net discovered with State-based Region Theory [9, 113] with no limit in the set size and the inclusion of all activities.

Figure 1.4: Solutions retrieved with different algorithms using the log shown in Figure 1.3(a).

able to reproduce new behavior. The *trace model*[3] is an example of lack of generalization, i.e., it is an overly precise process model. For example, the model shown in Figure 1.4(c) allows to execute more than the behavior seen in the log due to the loop before an activity *g* (*Assemble Phone*). The last dimension is *Simplicity*, that follows the Occam's Razor principle, that is, process models should be as simple and easily readable as possible, while reflecting the behavior of the log. As can be seen, the process models in Figure 1.4(b) and 1.4(d) (due to the number of connections) are more complex than the process models in Figure 1.4(c) (which is more general than the other process models) and Figure 1.4(a) (which does not correctly replay all the log). In general, the *holy grail* in process discovery is to find a trade-off among these four dimensions: replay fitness, precision, generalization and simplicity. However, as shown with the process models in Figure 1.4, this turns out to be a challenging balance, e.g., usually a very simple process model is likely to have a low fitness or a lack of precision, or, on the other hand, guaranteeing a perfect replay fitness can lead to overly general or overly specific process models.

[3] A *trace model* creates a path for each trace of the log. This kind of process model has a perfect precision as it only allows the specific behavior recorded in the log, but it is not a desirable solution.

State of the art

Since Cook and Wolf [27] coined the term process discovery, and later on, Agrawal et al. [7] applied this idea in the context of workflow management systems, a plethora of process discovery algorithms have been proposed. Although we can classify these algorithms based on different criteria [122, 15], e.g., the notation used to represent the process models, or which dimensions they focus the search on, we arrange the different techniques based on the approach they follow [140]:

- **Abstraction-based algorithms**. All of them are derived from the α-algorithm [137], and address some of its drawbacks. On the one hand, the α-algorithm can discover a large class of process models under the assumption that the event log contains all the behavior and it is free of noise. But even with this assumption, the α-algorithm has some problems. The limitations solved with the extensions of this algorithm are: short loops (α^{+}-algorithm [32]), non-free-choice constructs (α^{++}-algorithm [156]), invisible tasks ($\alpha^{\#}$-algorithm [157, 158]) and duplicate tasks (α^{*}-algorithm [75]). A recent approach to mine invisible tasks in non-free choice[4] constructs, ($\alpha^{\$}$-algorithm) was presented in [56]. Despite the different extensions, none of the abstraction-based algorithms can tackle all the complex constructs at once, and are based on a complete and noise free-log. In general, this type of algorithms focuses on simplicity, retrieving very simple process models but with poor replay fitness.

- **Heuristics-based algorithms**. In [154, 155], Weijters et al. presented the Heuristics Miner, an extension of the α-algorithm but taking into account the frequency of ordering relations. One of its main advantages is its ability to handle noise based on a set of thresholds. Thus, this method is appropriate for identifying the main behavior registered in the log, excluding duplicate tasks and some non-free-choice constructs. DWS [49] is an extension of Heuristics Miner that identifies different variants of a process model by clustering similar log traces. Another extension of the Heuristics Miner was presented in [21]. It takes into account the timestamp of the activities, expressing the activity as time intervals instead of single events. In [22] the authors present different stream-aware versions of the Heuristics Miner for mining process models from event data streams [143]. Fodina [15] is a recent extension of the Heuristics Miner with

[4] A non-free choice (NFC) construct is a special kind of choice, where the selection of a task depends on what has been executed before in the process model.

different improvements such as mining duplicate activities. Although these heuristics-based algorithms use replay fitness as their guiding principle, they do not guarantee optimal results as they only focus on the main behavior of the event log. On the other hand, Inductive Miner [70] is an approach that produces block-structured [120] process models able to replay the whole event log, i.e., it guarantees a perfect replay fitness. However, it is quite sensitive to incompleteness (although different extensions to face this issue are proposed in [71, 72, 73]), as well as duplicate labels and invisible activities, falling back to a flower model when it cannot find a strong relation between activities. Hence, the Inductive Miner (and extensions) can lead to process models with a lack of precision and very good generalization.

- **Search-based algorithms**. These techniques are based on the paradigm of evolutionary algorithms [8, 43]. In a nutshell, an evolutionary algorithm is a heuristic search that mimics the process of natural selection, using techniques like crossover, mutation and selection to generate a set of possible solutions which are optimized through several iterations until a convergence criterion is reached. The major characteristics of a genetic algorithm are its capability to explore large search spaces, and its flexibility to incorporate prior knowledge virtually in any part of the algorithm. Within this context, the first approach presented following this paradigm was Genetic Miner [30]. Alves de Medeiros et al. proved that it is possible to mine all common constructs and be robust to noise, all at once, but it cannot ensure simple process models as some of the mined solutions have implicit places or needless arcs. Another recently proposed approach, so called Evolutionary Tree Miner (ETM) [18, 141], guides its search taking into account a balance between the four objectives previously described, but considering only block structured solutions. However, the expressiveness of block structured process models is limited as they do not allow certain behavior, such as arbitrary loops, NFC, unbalanced split/join points, or certain patterns such as the *milestone* [144]. Moreover, many real life processes are not block structured [122]. Additionally, although alignments[5] are the de facto standard instrument for conformance checking [6, 124], computing them is a combinatorial problem and hence, extremely costly [149]. Another search-based algorithm is presented in [16], where the authors present the initial results of a new evolutionary algorithm for discovering *declarative* process models.

[5] Basically, alignments map as many events as possible from a trace with activities of a process model [5].

- **Algorithms based on theory of regions**. These algorithms can be classified in two groups based on their behavioral process specification: *state-space* and *language* based.

  - The *state-space* algorithms perform two steps: first, they build a transition system [129], i.e., a set of states and transitions between states, and then they construct a Petri net according to that transition system [9, 25, 42, 113]. This group of algorithms focuses on the synthesis of a Petri net whose reachability graph is similar to the transition system. As discussed in [129] and [138] the main problem of this solution is the non-trivial construction of the state information from a log, because usually logs almost never carry state information. Other problem is that it usually results in overfitting process models that can only replay the log without any form of generalization, hence being very sensitive to noise and incompleteness, leading to very complex processes.

  - In contrast, *language-based* algorithms assume that the log contains words (traces) of a specific language (the activities are the letters), whereas the target net allows just words of this language. In [12], the authors distinguish between two methods to derive Petri nets from event logs: *i)* using a *basis representation*, which cannot tackle duplicate tasks or non-free-choice constructs; and *ii)* using *separating representation* [12, 78] to mine duplicate tasks but not non-free-choice constructs. These algorithms usually guarantee a perfect replay fitness. However, a problem of language-based regions is that, in order to construct a Petri net, it is necessary to solve a linear inequation system which, unfortunately, has many solutions. Moreover, this approach usually leads to overfitted process models due to the restrictive assumptions about process logs. Additionally, the number of places introduced by both approaches is theoretically high. To overcome these drawbacks, in [138], authors propose to use Integer Linear Programming to avoid overffited process models and minimize the upper bound number of places. Nevertheless, pure region-based techniques still have problems with incomplete behavior, resulting in overly complex processes. Extensions such as [144] try to avoid this overffitting through heuristics and filtering feedback from the user.

- **Machine learning**. These algorithms apply different machine learning techniques to discover the control-flow of the activities. An example of this technique, based on *inductive logic programming*, was presented by Goedertier et al. [47]. They designed

AGNES, an algorithm that introduces artificially generated negative events, i.e., traces that describe a particular path that is not allowed for a process. Unfortunately, event logs hardly ever contain information about disallowed behavior. This idea of introducing negative events was latter applied in [91]. Another approach recently proposed is the algorithm NPM [76]. This technique, instead of mining the relationship between two events, mines a set of patterns that could cover all the traces seen in an event log. Another approach is presented in [159], where the authors describe a technique to discover declarative process models based on regular expressions and (finite) automata.

- **Partial approaches**. All the aforementioned techniques focus their search on retrieving an end-to-end process model. However, in the state of the art we can also distinguish different process discovery algorithms focused on producing *rules* or *frequent patterns* from an event log. For instance, there are approaches based on *sequential pattern mining* [116] and *episode mining* [69], as well as approaches to learn declarative (based on temporal properties) languages [127].

In summary, a large amount of work has been done in this specific area by addressing solutions from different points of view. Unfortunately, the review of the state of the art shows that many end-to-end process discovery algorithms focus on one or two of the aforementioned dimensions, or the used notation does not allow to represent all possible behavior. In other words, current techniques have difficulties to retrieve models with high levels of replay fitness, but being as precise and simple as possible. Considering this scenario, the hypothesis of this PhD Thesis regarding process discovery can be stated as follows:

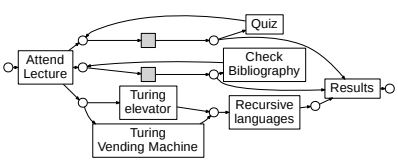
H_1. Is it possible to retrieve high quality process models, in general domains, through a *hierarchical criteria of replay fitness, precision and simplicity*?

1.3 Enhancing process models

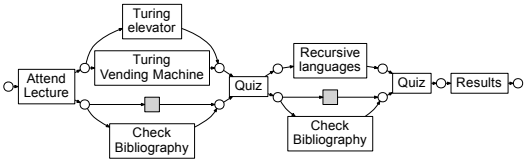
The visualization of a process model plays an important role in order to correctly gain insights about the process [135]. That is, process models that are too much complex, can hinder the real behavior of the process rather than to provide insights of what is really happening. Within this context, this dissertation focuses on the concept of *duplicate tasks*. The notion of duplicate tasks, or activities, refers to situations in which multiple tasks in the process have the same label. Concerning the visualization of a process model, this kind of behavior is useful *i)* when

	Sequence of events
$case_1$	Attend lecture, Turing Elevator, Check Bibliography, Quiz, Recursive Languages, Quiz, Results.
$case_2$	Attend lecture, Turing Elevator, Check Bibliography, Quiz, Check Bibliography, Recursive Languages, Quiz, Results.
$case_3$	Attend lecture, Turing Vending Machine, Check Bibliography, Quiz, Recursive Languages, Check Bibliography, Quiz, Results.
$case_4$	Attend lecture, Check Bibliography, Turing Vending Machine, Check Bibliography, Quiz, Recursive Languages, Quiz, Results.
⋮	⋮
$case_n$	...

(a) Traces extracted from a synthetic event log.



(b) Process model without duplicate labels.



(c) Process model with duplicate labels.

Figure 1.5: A log and two process models (Petri nets) exemplifying a lecture of Automata Theory and Formal Languages.

a particular task is used in different contexts in a process and *ii)* to enhance the readability of a model by reducing overly connected tasks.

Figure 1.5 shows an example on how the addition of duplicate tasks to a process model improves its readability and structural clarity. In this example, considering the sample log of Figure 1.5(a), the events *Quiz* and *Check Bibliography*, are executed at most twice in each trace. Between the multiple possibilities of modeling the behavior of the log, we can assume *i)* an injective relation between the events in the log and the activities in the process model (Figure 1.5(b)); or *ii)* that multiple activities can share the same label, i.e., a process model with duplicate activities (Figure 1.5(c)). In this example, although both process models perfectly reproduce all the behavior recorded in the log, i.e., both have a perfect replay fitness, the process model depicted in Figure 1.5(b) allows to execute both *Quiz* and *Check Bibliography* as many times as we want at any time in the process, hence, this process model is not a rigorous picture of the recorded behavior of the log: its precision is lower. On the other hand, if both activities are duplicated, the resulting process model (Figure 1.5(c)) is more suitable, i.e., more precise w.r.t. the recorded behavior in the log, as it does not allow, for example, to check

the bibliography (*Check Bibliography*) during the exam (*Quiz*), as the model in Figure 1.5(b) does. Hence, the ability to discover these duplicate tasks *may* greatly enhance, not only the readability of the final solution, but the precision of the process model by disallowing unseen behavior.

From the perspective of process discovery, including duplicate tasks in the mining process is a well known challenge [136] as, usually, different tasks can be recorded with the same label in the log, hindering the discovery of the process model that better fits the log. Thus, algorithms have to find out which events of the log belong to which tasks. In the state of the art of process discovery, many techniques [33, 71, 96, 138, 147, 153, 155] assume an injective relation between tasks and events in the log, considering that there cannot be two different activities with the same label. Therefore, these algorithms, when discovering processes that can generate logs with duplicate labels, usually give as a result process models with overly connected nodes or needless loops, decreasing the precision and simplicity of the process model.

Further, there are techniques that do not make such a restrictive assumption [15, 20, 23, 24, 30, 47, 75]. Typically, all these techniques identify the potential duplicate activities in a pre-mining step, or during the mining process. One example is the α^*-algorithm [75], an extension of the α-algorithm [137] to mine duplicate tasks. However, the heuristic rules used in this algorithm require a noise-free and complete log [140]. Fodina [15] is an algorithm based on heuristics that infers the duplicate tasks transforming the event log into a *task log* following the heuristics defined in [30]. Other solutions, like DGA [30] and ETM [20] (based on evolutionary algorithms), or AGNES [47] (an approach based on inductive programming) include the possibility to mine duplicate tasks. However, these techniques do not allow to *unfold* loops [30], or they are very permissive allowing to duplicate any activity in the log [20]. Region-based algorithms [23, 24] are also able to mine duplicate tasks, but, when searching for regions, they usually allow to split any label in the log without any bound. In [94] the authors present an approach for discovering duplicate tasks specifically based on BPMN models using the Heuristics Miner. On the other hand, besides these process discovery techniques, the closest works that follow the idea of improving the readability of an already mined process model are presented in [34, 44]. However, these techniques do no try to enhance a process model with the inclusion of duplicate labels. A recent technique [118] introduces the concept of label refinements, i.e., the authors perform the label splitting based on data attributes of the recorded events, for instance, using the time attribute of events.

To sum up, although very valuable results have been achieved in this field, the state of the art algorithms have different weaknesses. Some obtain, in specific logs, worse solutions than without duplicated tasks [15]. Others allow to duplicate any activity in the log [20], or generate solutions with a lower simplicity [24], i.e., more complex solutions. Finally, other proposals use heuristics that do not consider duplicate activities in some workflow patterns such as loops [30, 75]. Therefore, based on the above overview, the hypothesis of this PhD Thesis regarding enhancing process models can be stated as follows:

H_2. Is it possible to improve the structural clarity of a process model by duplicating the activities of a process model *after* the discovery process?

1.4 Process hierarchization

In the previous section, we showed that taking into account duplicate labels can drastically change how a process depicts the behavior recorded in the log. However, it is possible to reach a further level of readability of a process model by applying hierarchization techniques, through the use of domain knowledge. The exploitation of domain knowledge to retrieve richer process models is still in an incipient phase with some recent studies [10, 38, 48, 97], mostly because domain knowledge is, in most of the cases, very difficult to obtain. However, there is one particular difference with these approaches and the idea of *process hierarchization*. Instead of using domain knowledge during the discovery phase, process hierarchization aims to use domain knowledge to *translate* an already discovered process model to a more specific representation within a particular domain, and, thus, retrieve a more interpretable process model.

This idea of *process hierarchization* using domain knowledge is, to the extent of our knowledge, a rather new approach, with no previous work directly related within the field of process mining. Nonetheless, it follows the same idea as methods that aim to decompose process models [74, 87, 121]. However, decomposition techniques focus on the reduction of the complexity of the models by *slicing* them, i.e., to partition larger process models and event logs into smaller parts that can be analyzed independently. Thereby, the resultant process model using decomposition techniques is still a flatten representation of the original process model, i.e., through these techniques they do not completely *hierarchize* a process model. Another notable technique worth mentioning is the Fuzzy Miner [54, 55], an approach that can construct hierarchical process models, by moving less frequent activities to subprocesses

or clusters of activities. Hence, this hierarchization is driven by the frequency of use of the different activities and arcs. In [14], the authors present an extension of the Fuzzy Miner, called Fuzzy Map Miner, that follows a two-phase approach, by first simplifying the log to a desired level of granularity, and later discovering the models from this simplified log. More specifically, the discovery of hierarchical process models is enabled through the automated discovery of abstractions (of activities), during the first step of the approach. These abstractions are defined through the discovery of common execution patterns in the log [13]. Another approach based on this idea of hierarchical process models is presented in [50, 51]. In this approach, the idea is to produce hierarchical views of the process that satisfactorily capture the behavior of the log at different levels of detail. In other words, it aims to discover different *variants* (different usage scenarios) of the process by means of clustering. Hence, instead of a single, possible intricate and complex process model of the whole process, the aim is to retrieve a collection of more compact and easier to understand process models.

Concerning this PhD Thesis, although we seek for a general approach independent of any particular domain, we focus the process hierarchization in the educational field. The catalyst behind this decision is due the existence of IMS LD [46], a meta-data standard that describes all the elements of the design of a teaching-learning process. In this specification, one of the main components is the learning design. This component is understood as the coordination of the learning activities to be performed by the participants to achieve the pedagogical objectives, i.e the learning design describes the learning flow, or learning path, to be followed by learners. To describe this learning design, the IMS LD specification follows a theater metaphor where there are a number of *plays*, that can be interpreted as the runscripts for the execution of the course and that are *concurrently* executed, being independent of each other. Each one of these *plays* is composed by a set of *acts*, which can be understood as a module or chapter in a course. *Acts* are performed in *sequence* and define the activities that participants must do. This model also allows the assignation of *roles* to the participants and the partitioning of the activities of an act according to those *roles*, which are called *role-parts*. In this case, each one of the partitions can run *in parallel*. Finally, in each of these partitions, activities or activity structures are selected. The latter may consist of a *sequence* or a *selection* of activities.

Taking this structure into account, the hierarchization process would consist of identifying the different IMS LD elements from a discovered process model, e.g., a Petri net, where these elements are also represented as Petri nets. However, a Petri net is a flatten process, while

IMS LD is a hierarchical structure in which each layer is composed of a different type of elements, i.e., each layer of the tree corresponds to a layer of IMS LD. Specifically, the first layer represents plays, the second acts, the third role-parts, and the remaining layers activities or activity structures. Moreover, as IMS LD also allows to specify adaptive learning strategies, through this process it would also be necessary to extract the adaptive rules that constrict the learning flow. Hence, the the problem to be solved is how to *hierarchize* an IMS LD course from a process model and an event log, i.e., how to transform a process model into the IMS LD standard to depict a more readable process model. In this way we can provide a more interpretable process model to teachers, as they can work directly with learning elements. For instance, a teacher can directly change a sequence of learning activities, or even a role part, without further knowledge about Petri nets. Furthermore, thanks to this standardization of a process model, the learning processes become reusable between different platforms. That is, teachers would be able to take the reconstructed course, change it, and use it as a new course in a different environment.

In the state of the art, we can find several approaches related to this idea of reconstructing an IMS LD course. In [82], the authors focus on the reconstruction of an IMS LD course based on a visual language, which hides the complexity of the process model. Moreover, this approach requires a close collaboration between developers and teachers to simplify the gap between the technical and pedagogical point of view of the course. In [2], and later in [1], the authors present a four-step approach for process reenginering in higher education. However, the reengineering process is also not fully automatic, as it requires the participation and feedback from all the appropriate personnel and users. In [67], the authors present PETRA, a system to extract new knowledge rules about transitions and learning activities in processes from previous platform executions. However, this tool is not oriented towards process reconstruction and discovery, but on process extension, i.e., it requires an already defined process model in order to enrich such process. In summary, the main drawback of the state of the art approaches is that the course is not automatically reconstructed from scratch and needs the supervision of teachers and even developers. In other words, teachers need to provide feedback to map the discovered model to the IMS LD representation.

Therefore, considering the aforementioned state of the art, the hypothesis of this PhD dissertation on this scenario can be stated as follows:

H_3. Is it possible to *automatically hierarchize* an already discovered process model, using domain knowledge, and, thus, retrieve a more interpretable process model?

1.5 Objectives

As previously stated in Section 1.1, we defined two main objectives in this PhD Thesis: *i)* to design a process discovery algorithm focusing its search towards replay fitness, precision and simplicity; and *ii)* to enhance the interpretability of the discovered process models. To achieve that, different specific objectives have been pursued:

O.1 Process discovery algorithm guided by replay fitness, precision and simplicity

The first objective in this PhD thesis is to design a process discovery algorithm. Thus, from the control-flow perspective, the approach should be able to tackle the usual workflow patterns. Furthermore, it should be able to mine process models with high levels of replay fitness and precision, i.e., the discovered process models should represent, as best as possible, the main behavior recorded while retrieving the simplest process model as possible. Additionally, the algorithm should be able to deal with different levels of noise in the event log.

O.2 Enhance models through the inclusion of duplicate activities

The next objective is to extend the mined process models with more behavior, in particular, with duplicate labels. This is of particular interest from the point of view of readability and structural clarity of a process model, as with duplicate labels it is possible to disengage two activities with the same name that take place in different contexts of the process. The main problem behind this idea is that including duplicate activities as part of the semantics of a discovery algorithm can significantly increase the search space, hindering the discovery process. Furthermore, if any activity can be duplicated without any bound, this can result in overly-specific process models. Thus, the main objective is to enhance an already mined process model by duplicating the overly connected activities, improving the readibility of the resultant process model without adversely affecting its quality in terms of replay fitness and precision.

O.3 Hierarchization of process models

The last objective relates to the improvement of the interpretability and structural clarity of a process model. More specifically, the idea is to hierarchize a process model using domain knowledge. In other words, the pursued objective is to use domain knowledge to *translate* an already discovered process model to a more specific representation, within a particlar domain, and, thus, retrieve a more interpretable process model. This idea of

process hierarchizarion is of particular interest in domains such as education, due the existence of the IMS LD standard, a meta-data model that describes all the elements of a learning process. Thus, taking as an example the educational scenario, the idea is to use the event logs generated during a course to extract the learning flow structure using the previously developed algorithms, then to obtain the underlying rules that control the adaptive learning of students, and finally to combine them into an educational modeling language standard.

1.6 Research Contributions

The main contributions of this PhD dissertation are as follows:

C.1 Process discovery through a genetic algorithm

We developed a genetic process discovery algorithm (ProDiGen) that automatically searches for process models with high levels of replay fitness, precision and simplicity, while being robust to noise. Furthermore, the algorithm is able to face all the common workflow patterns at once. Thus, through this algorithm we are able to retrieve high quality process models regardless the domain. More specifically, the novelties of ProDiGen are:

- A hierarchical fitness function that takes into account replay fitness, precision and simplicity.
- A new definition of precision based on the log and the mined process model.
- A new definition of simplicity based on the mined process model.
- A crossover operator that selects the crossover point from a Probability Density Function (PDF) generated from the errors of the mined process model.
- A mutation operator guided by the causal dependencies of the log.

C.2 Mining duplicate labels from discovered process models

SLAD (Splitting Labels After Discovery) is a novel algorithm that enhances an already discovered process model by splitting the behavior of its activities. Through this algorithm, we analyse the possibility of tackling duplicate tasks *after* mining a process model, without adversely affecting the quality of the initial solution. Before the execution of SLAD, a process discovery technique mines a log without considering duplicate

tasks, generating a causal net or a heuristic net. Then, SLAD, using the local information of the log and the retrieved process model, tries to improve the quality of the process model by performing a local search over the tasks that have more probability to be duplicated in the log. The contributions of this proposal are:

- The discovering of the duplicate activities is performed *after* the discovery process, in order to *unfold* the overly connected nodes than may introduce extra behavior not recorded in the log.
- New heuristics to focus the search of the duplicated tasks on those activities that better improve the process model.
- New heuristics to detect potential duplicate activities involved in loops.

C.3 Process hierarchization

We present an approach to *automatically hierarchize* a process model using domain knowledge. This objective is achieved in three different steps. Firstly, the process model is automatically extracted from the logged sequences through the developed process mining algorithms. Then, an algorithm based on the knowledge about the target language control structure is applied to determine which components should be created. Finally, the adaptive rules are automatically extracted form the event logs (more specifically, from the variable values of the logs) by a decision tree learning algorithm, and integrated into the target language structure. Furthermore, we have implemented and validated this hierarchization process into the educational domain, enabling the hierarchization of process models into a standardized learning process model more suitable for teachers, i.e., IMS LD. The main contributions of this proposal are:

- A new framework to hierarchize process models using domain knowledge regardless the target language.
- A new framework to make process models more interpretable based on domain knowledge.
- The automatic identification of the adaptive rules from event logs.
- The automatic discovery of learning processes from event logs and its recompilation to IMS LD.
- A new framework to facilitate the reuse of designed courses between different virtual learning environments.

C.4 Software Tools

Additionally, in this PhD Thesis, the following software tools have been developed together with the aforementioned research contributions:

Tool **ProDiGen Web**[6]

A web platform to mine, visualize and analyse both event logs and process models. This platform provides automated process discovery for logs deriving from different sources, allowing to apply different kinds of filters to retrieve much richer process models. On the other hand, it provides a dynamic visualization of the process as it happened, allowing to instantly spot bottlenecks or deviations. Related to data, it provides both statistics for the information of the event log and the discovered process model, ranging from how many times an activity was executed in the process, or which case was the slowest/fastest in the process, etc. Additionally, this web platform also provides access to other algorithms, such as mining the frequent paths of a process model (WoMine), or the algorithm to mine duplicate labels (SLAD).

Tool **SoftLearn**[7]

SoftLearn [146, 148] is a process mining-based tool which automatically discovers and represents the learning flows that the students have followed in the development of the tasks of a specific course. SoftLearn allows teachers to assess the performance of the students, providing information about their learning process and behavior throughout the course, and facilitating the evaluation of the learning activities carried out by learners during the course. This tool is currently been used by five different teachers in three different courses in the Degree in Pedagogy at the Faculty of Education of the Universidade de Santiago de Compostela: more than 150 students are being evaluated through this platform.

[6] http://tec.citius.usc.es/processmining
[7] http://tec.citius.usc.es/SoftLearn

1.7 Publications

All the contributions of this PhD dissertation are included in the following publications:

Journal Papers:

Inf Sci B. Vázquez-Barreiros, M. Mucientes, and M. Lama. ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. *Information Sciences*, 294:315–333, 2015.
(DOI: 10.1016/j.ins.2014.09.057).

- Impact Factor (JCR 2015): 3.364. Category: COMPUTER SCIENCE, INFORMATION SYSTEMS. Order 8/143. Q1.

KBS J.C. Vidal, B. Vázquez-Barreiros, Manuel Mucientes, and Manuel Lama. Recompiling Learning Processes from Event Logs. *Knowledge-Based Systems*, 100:160–174 2016.
(DOI: 10.1016/j.knosys.2016.03.003).

- Impact Factor (JCR 2015): 3.325. Category: COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE. Order 17/130. Q1.

Inf Sci B. Vázquez-Barreiros, M. Mucientes, and M. Lama. Enhancing Discovered Processes with Duplicate Tasks. *Information Sciences*, 373:369–387, 2016.
(DOI: 10.1016/j.ins.2016.09.008)).

- Impact Factor (JCR 2015): 3.364. Category: COMPUTER SCIENCE, INFORMATION SYSTEMS. Order 8/143. Q1.

Int J Intell Syst A. Ramos-Soto, B. Vázquez-Barreiros, A. Bugarín, A. Gewerc, S. Barro. Evaluation of a Data-To-Text System for Verbalizing a Learning Analytics Dashboard. *International Journal of Intelligent Systems*, 2016.
(DOI: 10.1002/int.21835).

- Impact Factor (JCR 2015): 2.050. Category: COMPUTER SCIENCE, ARTIFICIAL INTELLIGENCE. Order 37/130. Q2.

International Conferences:

BPM B. Vázquez-Barreiros, M. Mucientes, and M. Lama. *A Genetic Algorithm for Process Discovery Guided by Completeness, Precision and Simplicity*. In S.W. Sadiq, P. Soffer, and H. Völzer, editors, *Proceedings of 12th International Conference on Business Process Management BPM*, volume 8659 of *Lecture Notes in Computer Science*, pages 118–133, Eindhoven, The Netherlands, 2014.

- Conference Ranking (CORE 2014): A.

ICALT B. Vázquez-Barreiros, M. Mucientes, M. Lama, and J.C. Vidal . *SoftLearn: A Process Mining Platform for the Discovery of Learning Paths*. In D.G. Sampson and J.M. Spector and N.S. Chen and R. Huang and Kinshuk, editors, *Proceedings of 14th IEEE International Conference on Advanced Learning Technologies, ICALT*, pages 373–375, Athens, Greece, 2014.

- Conference Ranking (CORE 2014): B.

EC-TEL J.C. Vidal, M. Lama, B. Vázquez-Barreiros, and M. Mucientes. *Reconstructing IMS LD Units of Learning from Event Logs*. In C. Rensing, S. de Freitas, T. Ley, P.J.M. Merino, editors, *Proceedings of 9th European Conference on Technology Enhanced, EC-TEL*, volume 8719 of *Lecture Notes in Computer Science*, pages 345–358, Graz, Austria, 2014.

FIE A. Rodríguez Groba, B. Vázquez-Barreiros, M. Lama, A. Gewerc, and M. Mucientes. *Using a Learning Analytics Tool for Evaluation in Self-Regulated Learning*. In M. Castro, E. Tovar, editors, *Proceedings of 44th IEEE Frontiers in Education Conference, FIE*, pages 2484–2491, Madrid, Spain, 2014.

- Conference Ranking (CORE 2014): B.

FIE M. Fernandez-Delgado, M. Mucientes, B. Vázquez-Barreiros, and M. Lama. *Learning Analytics for the Prediction of the Educational Objectives Achievement*. In M. Castro, E. Tovar, editors, *Proceedings of 44th IEEE Frontiers in Education Conference, FIE*, pages 2500–2503, Madrid, Spain, 2014.

- Conference Ranking (CORE 2014): B.

ATAED B. Vázquez-Barreiros, M. Mucientes, and M. Lama. *Mining Duplicate Tasks from Discovered Processes*. In W.M.P. van der Aalst, R. Bergenthum, J. Carmona, editors, *Proceedings of 2015 International Workshop on Algorithms & Theories for*

the Analysis of Event Data, ATAED, volume 1371 of *CEUR Workshop Proceedings*, pages 78–82, Brussels, Belgium, 2015.

AIED B. Vázquez-Barreiros, A. Ramos-Soto, M. Lama, M. Mucientes, A. Bugarin, and S. Barro. *Soft Computing for Learner's Assessment in SoftLearn*. In C. Conati, N. Heffernan, A. Mitrovic, M.F. Verdejo, editors, *Proceedings of 17th International Conference on Artificial Intelligence in Education, AIED*, volume 9112 of *Lecture Notes in Computer Science*, pages 925–926, Madrid, Spain, 2015.

- Conference Ranking (CORE 2014): A.

ICALT A. Ramos-Soto, M. Lama, B. Vázquez-Barreiros, A. Bugarín, M. Mucientes, S. Barro:. *Towards Textual Reporting in Learning Analytics Dashboards* . In N.S. Chen and Kinshuk and C.C. Tsai, editors, *Proceedings of 15th IEEE International Conference on Advanced Learning Technologies, ICALT*, pages 260–264 , Hualien, Taiwan, 2015.

- Conference Ranking (CORE 2014): B.

ATAED B. Vázquez-Barreiros, D. Chapela, M. Mucientes, and M. Lama. *Process Mining in IT Service Management: A Case Study*. In W.M.P. van der Aalst, R. Bergenthum, J.Carmona, editors, *Proceedings of 2016 International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED*, volume of *CEUR Workshop Proceedings*, pages 16–30, Toruń, Poland, 2016.

BPMDS B. Vázquez-Barreiros, S.J. van Zelst, J.C.A.M. Buijs, M. Lama, M. Mucientes *Repairing Alignments: Striking the Right Nerve*. In , editors, *Proceedings of 17th International Conference on Business Process Modeling, Development, and Support, BPMDS*, volume of *Lecture Notes in Business Information Processing*, pages 266–281, Ljubljana, Slovenia, 2016.

- Conference Ranking (CORE 2014): C.

National Conferences:

JCIS B. Vázquez-Barreiros, M. Mucientes, and M. Lama. ProDiGen: minando modelos completos, precisos y simples con un algoritmo genético. In *Jornadas de Ciencia e Ingeniería de Servicios, JCIS*, Santander, Spain, 2015.

UNIVEST A. Rodríguez Groba, A. Gewerc, B. Vázquez-Barreiros, and M. Lama. SoftLearn: Una herramienta para evaluar y acompañar la creación de e-portflio. In *V Congreso Internacional UNIVEST*, Girona, Spain, 2015.

JCIS J.C. Vidal, B. Vázquez-Barreiros, M. Mucientes, and M. Lama. Recompilación de procesos de educación a partir de registros de eventos. In *Jornadas de Ciencia e Ingeniería de Servicios, JCIS*, Salamanca, Spain, 2016.

JCIS B. Vázquez-Barreiros, S.J. van Zelst, J.C.A.M. Buijs, M. Lama, M. Mucientes. Reparación de alignments. In *Jornadas de Ciencia e Ingeniería de Servicios, JCIS*, Salamanca, Spain, 2016.

1.8 Thesis Outline

Figure 1.6 shows the main structure of this thesis:

- Chapter 2 presents ProDiGen, a genetic algorithm for process discovery guided by replay fitness, precision and simplicity. The algorithm uses a hierarchical fitness function that takes into account completeness, precision and simplicity (with new definitions for both precision and simplicity) and uses heuristics to optimize the genetic operators: *i)* a crossover operator that selects the crossover point from a Probability Density Function (PDF) generated from the errors of the mined model, and *ii)* a mutation operator guided by the causal dependencies of the log. Furthermore, ProDiGen was compared with four state of the art algorithms using 39 process models.

- Chapter 3 focuses on improving the structural claririty and readability of discovered process models through the inclusion of duplicate labels. For this purpose, we developed SLAD (Splitting Labels After Discovery), an algorithm that takes as starting point an already mined model, and using the local information of the log, tries to improve the replay fitness, precision and simplicity of the model by splitting the overly connected nodes into two or more activities. In order to validate the performance of the approach, we present a detailed evaluation with 54 mined models from three process discovery algorithms. Furthermore, the results have been compared with eight different algorithms from the state of the art.

- Chapter 4 presents a novel framework for the hierarchization of process models through the inclusion of domain knowledge. More specifically, we show how the hierarchization

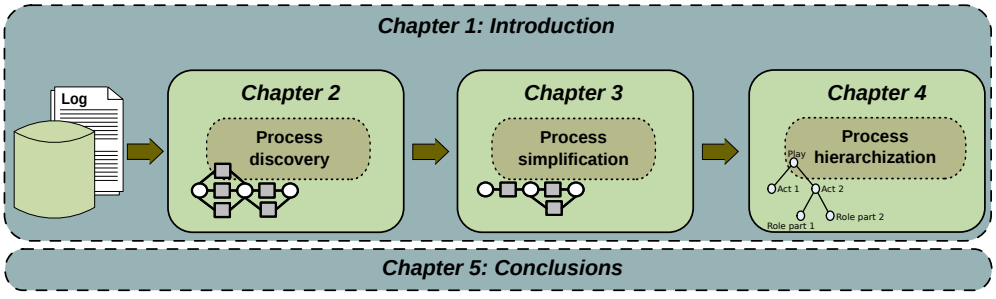


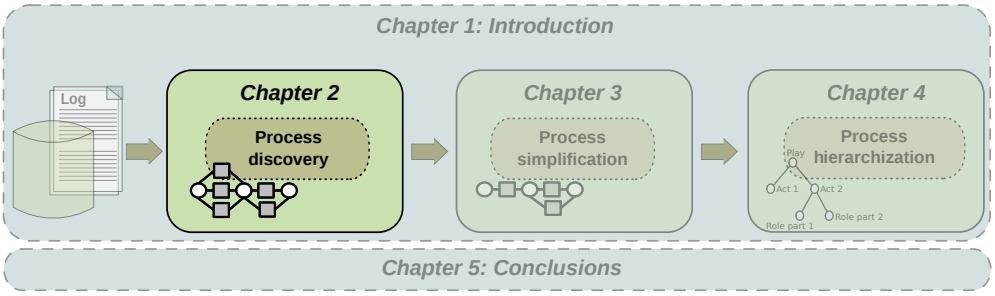
Figure 1.6: Dissertation structure.

of a process model using domain knowledge can enhance its interpretability within tan specific domain. Furthermore, we have implemented this framework in the educational domain, enabling the hierarchization of a process model into the standard IMS LD. The main components of this framework are: *i)* the automatic discovery of learning processes from event logs and its automatic hierarchization to IMS LD; *ii)* the automatic identification of the adaptive rules from event logs; *iii)* a new framework to facilitate the reuse of courses between different virtual learning environments. Moreover, an important feature of the described approach is its independence from any target language. Each one of the three parts of the reengineering approach has been analyzed separately using a set of nine real courses with different degrees of complexity.

- Chapter 5 presents the main conclusions and future work.

CHAPTER 2

PRODIGEN: MINING COMPLETE, PRECISE AND MINIMAL STRUCTURE PROCESS MODELS WITH A GENETIC ALGORITHM



Process discovery aims to obtain a process model that specifies the relations between activities in an event log, by combining different control structures such as sequences, choices, loops, among others, that are used to coordinate the control-flow of the activities in the workflow. As indicated in Chapter 1 very valuable results have been achieved in this field. Unfortunately, many process discovery algorithms focus on one or two quality dimensions or the used notation does not allow to properly represent all possible behavior. For instance, algorithms that *only* are able to retrieve block-structured process models cannot tackle non-free-choice constructs, arbitrary loops, or other common patterns in a real life process model. In other words, current techniques have difficulties to retrieve models with high levels of replay fitness,

but being as precise and simple as possible.

To tackle this scenario, in this chapter we present ProDiGen (Process Discovery through a Genetic algorithm), a process discovery algorithm that guides its search towards replay fitness, precision, and simplicity. More specifically, the algorithm uses a hierarchical fitness function that takes into account replay fitness, precision and simplicity (with new definitions for both precision and simplicity) and uses heuristics to optimize the genetic operators: (i) a crossover operator that selects the crossover point from a Probability Density Function (PDF) generated from the errors of the mined model, and (ii) a mutation operator guided by the causal dependencies of the log. In order to validate the performance of ProDiGen, we have used 39 models from the literature with several levels of noise and different degrees of complexity, giving a total of 111 different logs. Moreover, we have compared our approach with four state of the art algorithms using a collection of conformance checking metrics.

This chapter includes a full copy of the following journal paper that describes in detail the proposed approach:

> B. Vázquez-Barreiros[1], M. Mucientes[1], and M. Lama[1]. ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. *Information Sciences*, 294:315–333, 2015.
> (DOI: 10.1016/j.ins.2014.09.057).

2.1 Abstract

Process discovery techniques automatically extract the real workflow of a process by analyzing the events that are collected and stored in log files. Although in the last years several process discovery algorithms have been presented, none of them guarantees to find complete precise and simple models for all the given logs. In this paper we address the problem of process discovery through a genetic algorithm with a new fitness function that takes into account both fitness replay, precision and simplicity. ProDiGen (Process Discovery through a Genetic algorithm) includes new definitions for precision and simplicity, and specific crossover and mutation operators. The proposal has been validated with 39 process models and several noise levels, giving a total of 111 different logs. We have compared our approach with the state of

[1]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela. Santiago de Compostela, Spain.

the art algorithms; non-parametric statistical tests show that our algorithm outperforms the other approaches, and that the difference is statistically significant.

2.2 Introduction

In the last decade, a great effort for developing technologies to automate the execution of processes has been made in different application domains such as industry, education or medicine [41]. In this context, a process is understood as a collection of tasks —or activities— with coordination requirements among them [132]. These tasks are performed by a set of actors to achieve the purpose of the process. For instance, in education the learning design of a course is a process where learners must undertake a sequence of learning activities, e.g., posting in forums, making exercises and exams, etc., in order to achieve the pedagogical objectives of the course. Typically, these processes have a detailed description, i.e., there is a design of the process where its activities and the actors participating in these steps are clearly described. However, even in this situation there might be differences between what is actually happening and what is predefined in the process. For instance, following with the example of the education domain, learners can undertake additional learning activities —like check the bibliography or interact with other learners— apart from those that were explicitly specified in the learning process designed by a teacher.

At this point, Process Mining (PM) techniques are needed to get information about *what is really happening* in the execution of a process, and *not what the people think it is happening* [137]. Typically these techniques use the log files that collect information about the events detected and stored by the information system in which the process has been performed. PM techniques can be classified in three different groups [122]. The first one is *process discovery*, which aims to retrieve the process model that represents the behavior recorded in an event log. These algorithms are used to discover the underlying process that has been followed by users to achieve an objective. The second class of process mining techniques is *conformance checking*, where a process model is compared with a log of the same process to analyze and quantify the deviations between the observed and the real behavior, as recorded in the log. These techniques are focused on providing an *understanding* of the real processes that take place in an organization. The third group is *enhancement*, where a process model is dynamically modified or extended based on the information from the log. In this paper, we address the problem of process discovery. More specifically, our interest lies in the control-flow of the

recorded events, i.e., the ordering of the activities.

Over the last decade, several papers have dealt with the discovery problem in process mining [19, 30, 32, 137, 138, 155, 156, 157]. Unfortunately, existing techniques may produce models that are unable to replay the log, may produce complex and unreadable models, or may retrieve erroneous models. For instance, those approaches based just on the local information provided by the log [137] are only capable to overcome specific weak points in the field under some conditions —completeness and noise-free logs—, like short loops [32], non-free-choice constructs [156] or invisible tasks [157], but not all at once. Other papers solve the process discovery problem with search-based approaches, based on heuristics or on theory of regions [19, 30, 138, 155]. Some techniques guarantee sound models [19], others guarantee the rediscoverability of the main behavior of the log [155], some guarantee perfect fitness replay [138] and others can tackle all the different and main pattern constructs at once [30] but leaving simplicity aside. Nevertheless, there is no discovery algorithm that can tackle all the different structures at once, and that can find complete, precise and simple models. Furthermore, many of them have problems while dealing with noise.

In this paper we present ProDiGen[2] (Process Discovery through a Genetic algorithm), a process discovery algorithm that searches complete, precise and simple models. The contributions of this proposal are:

1. A hierarchical fitness function that takes into account completeness, precision and simplicity.
2. A new definition of precision based on the log and the mined model.
3. A new definition of simplicity based on the mined model.
4. A crossover operator that selects the crossover point from a Probability Density Function (PDF) generated from the errors of the mined model.
5. A mutation operator guided by the causal dependencies of the log.

The proposal has been tested with 39 models with several noise levels and different degrees of complexity, giving a total of 111 different logs. Moreover, we have compared our approach with four of the state of the art algorithms using a collection of conformance checking metrics. The results of the comparison have been validated with non-parametric statistical tests.

[2] http://tec.citius.usc.es/processmining/prodigen

The remainder of this paper is structured as follows. Section 2.3 introduces the process discovery problem, and Section 2.4 describes the different approaches that have already been proposed. Then, Section 2.5 presents the proposed genetic algorithm for process discovery. Section 2.6 shows the obtained results and the comparison with other approaches, and, finally, Section 2.7 points out the conclusions.

2.3 Process Discovery

The goal of process discovery is to obtain a process model that specifies the relations between tasks —or activities— in an event log. The basic assumption is that there is a process model that generates the log with the following rules: *i)* each event is a well-defined step in some process; *ii)* each event is related to a particular case; and *iii)* the events are sequentially saved no matter the type of pattern behind. Figure 2.1(a) represents a simple log with 18 events, 6 different activities and performed by three different users.

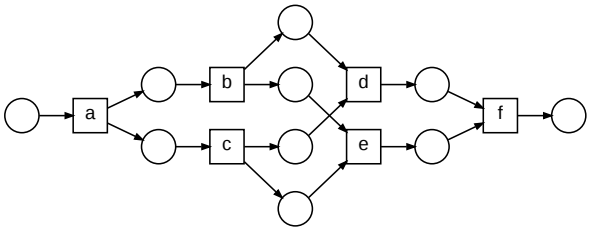
To discover the underlying process, the proposal presented in this paper only needs the list of events in the log and their corresponding case identifiers. Other process discovery techniques can use more information, like the timestamp [21, 133] or the data attributes that affect the routing of the cases [104]. The event log of Figure 2.1(a) shows the process instances of three different students during the course *Automata Theory and Formal Languages*. In this log, all the students started with an introductory class, and then they attended to four different lessons (*finite automaton*, *regular grammar*, *context-free grammar* and *pushdown automaton*). Once all these four lessons are finished, they make an exam. Using the information provided by the log, the discovery of a process model can take place with different objectives [102]:

- **Completeness**. Measures how much of the behavior observed in the log can be reproduced by the mined model. A model is complete when it can reproduced all the behavior in the log.

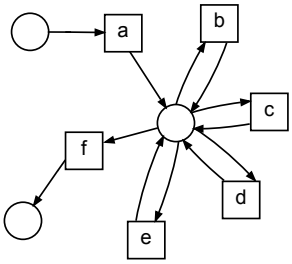
- **Precision**. The objective of this metric is to avoid overly general models. The model in Figure 2.1(c) is able to parse all the traces in the log. However, it is an overly general model, as it allows more behavior than the contained in the log. For example, one valid trace for this model is the possibility to make the exam after attending only to the introductory class.

User	Event	Lifecycle	Timestamp
Pablo	Introductory class	complete	07-03-2013:10:00
Borja	Introductory class	complete	09-03-2013:16:00
Pablo	Finite Automaton	complete	11-03-2013:19:30
Pablo	Regular Grammar	complete	17-03-2013:15:28
Borja	Finite Automaton	complete	20-03-2013:10:12
Manuel	Introductory class	complete	20-03-2013:11:42
Borja	Regular Grammar	complete	21-03-2013:14:34
Manuel	Regular Grammar	complete	23-03-2013:09:21
Pablo	Context-Free Grammar	complete	01-04-2013:12:36
Manuel	Finite Automaton	complete	01-04-2013:15:54
Borja	Pushdown Automaton	complete	04-04-2013:17:20
Manuel	Context-Free Grammar	complete	06-04-2013:20:00
Pablo	Pushdown Automaton	complete	21-04-2013:11:02
Borja	Context-Free Grammar	complete	22-04-2013:15:09
Manuel	Pushdown Automaton	complete	28-04-2013:17:45
Pablo	Exam	complete	06-05-2013:18:45
Manuel	Exam	complete	06-05-2013:19:01
Borja	Exam	complete	06-05-2013:19:22

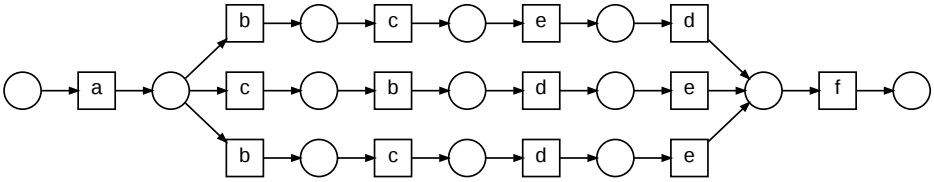
(a) An event log



(b) A complete, specific and general model: Completeness +, Precision +, Generalization +, Simplicity +



(c) A complete over-general model: Completeness +, Precision -, Generalization +, Simplicity +



(d) A complete over-specific model: Completeness +, Precision +, Generalization -, Simplicity -

Figure 2.1: Discovery of a process model prioritizing different objectives. The models are represented as Petri nets. The name of the activities are: *Introductory class* (a), *Finite Automaton* (b), *Regular Grammar* (c), *Context-free grammar* (d), *Pushdown automaton* (e) and *Exam* (f)).

- **Generalization**. This metric tries to prevent overly precise models. For instance, although the model in Figure 2.1(d) is complete, it is overly specific because it merely recreates each one of the traces of the log, not allowing extra behavior. When mining models, there is a tradeoff between precision an generalization, which can be compared to the bias-variance tradeoff.

- **Simplicity**. Indicates to models with a minimal structure that reflects the behavior in the log. For example, the model in Figure 2.1(d) creates one path for every possible trace of the log, i.e., it is not simple as it contains several duplicated tasks, which make the model difficult to read.

Figure 2.1(b) shows a model that takes into account the four objectives. This model is well-structured, providing not only the behavior shown in the log, but allowing also the trace *"Intro, Regular Grammar, Finite Automaton, Pushdown Automaton, Context-Free Grammar, Exam"*. Moreover, the model does not allow random behavior.

2.4 Related Work

Since Cook and Wolf [27] coined the term process discovery, and later on, Agrawal et al. [7] applied this idea in the context of workflow management systems, dozens of process discovery methods have been proposed [122]. Although some mining techniques use a specific target model for control-flow discovery [55], most of the process discovery algorithms are based on Petri nets [88]. These algorithms can be classified in four groups [140]:

- **Abstraction-based algorithms**. This type of algorithms are based on a complete and noise free-log. All of them are derived from the α-algorithm [137] to address some of its drawbacks. The limitations solved with these extensions are: short loops (α^{+}-algorithm [32]), non-free-choice constructs (α^{++}-algorithm [156]), invisible tasks ($\alpha^{\#}$-algorithm [157]) and duplicate tasks (α^{*}-algorithm [75]). Despite the different extensions, none of the abstraction-based algorithms can tackle all the complex constructs at once. In general, this type of algorithms focuses on simplicity, retrieving very simple models but with poor completeness.

- **Heuristics-based algorithms**. In [154, 155], Weijters et al. presented the Heuristics Miner, an extension of the α-algorithm but taking into account the frequency of ordering relations. One of its main advantages is its ability to handle noise based on a

set of thresholds. Thus, this method is appropriate for identifying the main behavior registered in the log, excluding duplicate tasks and some non-free-choice constructs. DWS [49] is an extension of Heuristics Miner that identifies different variants of a process model by clustering similar log traces. Another extension of the Heuristics Miner was presented in [21]. It takes into account the timestamp of the activities, expressing the activity as time intervals instead of single events. Heuristics-based algorithms use replay fitness (completeness) as their guiding principle, but do not guarantee optimal results in terms of completeness, as they only focus on the main behavior.

- **Search-based algorithms**. So far, the previously described algorithms are based on local information and therefore, they cannot discover some constructs like non-free-choices. To overcome this situation, the search-based algorithms perform a global search based on an abstraction from local properties like ordering relations. With Genetic Miner, Alves de Medeiros et al. [30] proved that it is possible to mine all common constructs and be robust to noise, all at once, but it cannot ensure simple models as some of the mined solutions have implicit places or needless arcs. Another approach recently proposed [19] guides its search taking into account a balance between the four objectives described in Section 2.3, considering only block-sructured solutions.

- **Algorithms based on theory of regions**. They can be classified in two groups based on their behavioral process specification: *state-spaced* and *language* based. The *state-based* algorithms perform two steps: first, they build a transition system [129] —a set of states and transitions between states—, and then they construct a Petri net according to that transition system [9, 25, 113, 42]. This group of algorithms focuses on the synthesis of a Petri net whose reachability graph is similar to the transition system. As discussed in [129] and [138] the main problem of this solution is the non-trivial construction of the state information from a log, because usually logs almost never carry state information. In contrast, *language-based* algorithms assume that the log contains words (traces) of a specific language —the activities are the letters—, whereas the target net allows just words of this language. For *language-based* algorithms, in [12], the authors distinguish between two methods to derive Petri nets from event logs, *i)* using a *basis representation*, which cannot tackle duplicate tasks or non-free-choice constructs; and *ii)* using *separating representation* [12, 78] to mine duplicate tasks but not non-free-choice constructs. Both approaches lead to a model overfitting the log due to the restrictive assumptions about process logs. Additionally, the number of places

introduced by both approaches is theoretically high. To overcome these drawbacks, in [138], authors propose to use Integer Linear Programming to avoid overffited models and minimize the upper bound number of places. However, because no assumptions are made about the completeness of the log, the solution might be an underfitted model, allowing for much more extra behavior. These algorithms usually guarantee a perfect replay fitness, but, unfortunately, these techniques still have problems with incomplete behavior.

A method that does not fit in any of these categories and is based on **inductive logic programming** was presented by Goedertier et al. [47]. They designed an algorithm that introduces artificially generated negative events, i.e., traces that describe a particular path that is not allowed for a process. Unfortunately, event logs hardly ever contains information about disallowed behavior.

In summary, a large amount of work has been done in this specific area by addressing solutions from different points of view. Unfortunately, none of the techniques can retrieve models with high levels of completeness, but being as precise and simple as possible. Furthermore, none of these techniques can handle all the different control constructs and noise all at once, but ensuring completeness, precision and simplicity. Hence, we propose a more elaborate approach based on the idea of Genetic Miner [30] to overcome these drawbacks.

2.5 ProDiGen: Process Discovery through a Genetic algorithm

Our proposal (ProDiGen) is inspired in the Genetic Miner algorithm [30], which can tackle all the different constructs at once. However, Genetic Miner has several drawbacks: *i)* it is not able to mine complete and precise models when they have many interleaving situations; *ii)* the mined models are usually hard to interpret and unnecessarily complex; and *iii)* it needs many generations to converge to a solution.

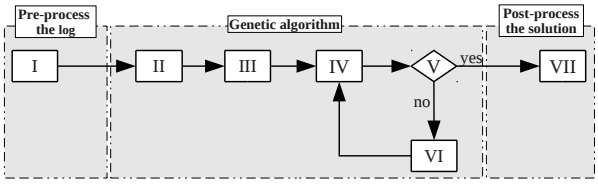
The drawbacks of Genetic Miner are caused by: *i)* a weighted fitness function that combines completeness and precision in an inadequate way; *ii)* the precisions of the models are not very informative as they depend on the precisions of the other individuals of the population; *iii)* the fitness function does not take into account the simplicity of the model; and *iv)* the genetic operators are executed in a completely random way, without taking advantage of the information of the log and the errors of the mined model during the parsing of the traces.

Table 2.1: Differences between ProDiGen and Genetic Miner.

Fitness	The fitness is hierarchical and takes into account the completeness, precision and simplicity of the mined model.
Precision	Definition of a new method to measure the precision of a model.
Simplicity	Definition of a new method to measure the simplicity of a model.
Initialization	ProDiGen incorporates the result of the Heuristics Miner [155] into the initial population.
Selection	Binary tournament selection.
Replacement	It selecs the best individuals of a joint population of parents and offspring. The reinitialization criterium is based on the improvement of the population.
Crossover	The crossover operator is guided by a Probability Density Function (PDF) generated from the errors of the mined model.
Mutation	The mutation operator is guided by the causal dependencies of the log.

The differences between ProDiGen and Genetic Miner are summarized in Table 2.1. As can be seen, almost all the mains steps of the genetic algorithm have been changed in order to overcome the previously discussed drawbacks. In particular, one of the major changes takes place in the hierarchical fitness function, where completeness, precision and simplicity are considered for the evaluation of an individual. Additionally, we introduce heuristics to guide the genetic operators, focusing the search on those parts of the mined model that have errors and, also, looking for new models that are supported by the information in the log.

The main steps of ProDiGen are shown in Figure 2.2. The algorithm has three phases: *i)* a *pre-processing* of the log, which groups and filters out the noise in the log; *ii)* the core of ProDiGen is the *genetic algorithm* phase; and *iii)* a *post-processing* of the mined model, to prune unused and infrequent arcs. The genetic algorithm is described in 2.5.1 and both the pre-processing and post-processing steps are described in Section 2.5.2.



Step	Description
I	Pre-process the log
II	Compute dependencies
III	Build the initial population
IV	Evaluate each solution
V	Stopping conditions fulfilled?
VI	Generate the new population
VII	Post-process the best individual

Figure 2.2: Main steps of ProDiGen.

Algorithm 2.1: Genetic algorithm for process discovery.

```
1  Initialize population
2  Evaluate population
3  t = 1, timesRun = initialTimesRun, restarts = 0
4  while t ≤ maxGenerations && restarts < maxRestarts do
5      Selection
6      Crossover
7      Mutation
8      Evaluate new individuals
9      Replace population
10     t = t + 1
11     if bestInd(t) == bestInd(t − 1) then
12         timesRun = timesRun − 1
13     if none of the individuals of the population have been replaced then
14         timesRun = timesRun − 1
15     if timesRun < 0 then
16         Reinitialize population
17         Evaluate population
18         timesRun = initialTimesRun, restarts = restarts + 1
```

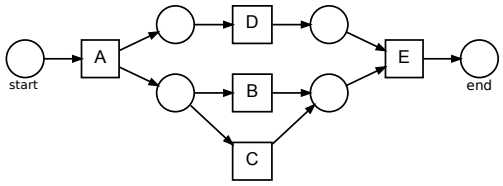
2.5.1 Genetic algorithm

Algorithm 2.1 describes the genetic algorithm. The first three steps correspond to an initialization, where t represents the number of iterations, *timesRun* is used to detect situations in which the search gets stuck, and *restarts* counts the number of executed reinitializations. The evolution cycle of the algorithm starts at Algorithm 2.1:4. This part will be repeated until the stopping criterion is fulfilled. The main steps of the iterative part are the selection of the individuals, the crossover and mutation operations to generate new individuals, their evaluation, the replacement of the population, and the analysis of the population to detect blockages in the search process. All of these steps are described in detail in the next sections.

Internal representation

Each individual of the population codifies a workflow using the causal matrix representation [30], which can map any Petri net in terms of causal dependencies. The causal matrix has a row for each task t in the log, and two columns corresponding to the inputs —I(t)—

and outputs —O(t)— of each task t (see Figure 2.3 for an example). Those tasks in the same subset of I(t) have an OR-join relation, and those on different subsets an AND-join relation. On the other hand, tasks in the same subset of O(t) have an OR-split relation and those in different subsets an AND-split relation.



(a) Example of a Petri net.

Task	I(Task)	O(Task)
A	{}	{{D},{C B}}
B	{{A}}	{{E}}
C	{{A}}	{{E}}
E	{{D},{B,C}}	{}
D	{{A}}	{{E}}

(b) Causal matrix of the Petri net.

Figure 2.3: Mapping of a petri net into a causal matrix.

Initialization

The initialization follows the heuristic approach described in [30], which is based on the causality relations between tasks. Moreover, we also add to the initial population an individual mined with the Heuristics Miner approach [155]. It is important to notice that the inclusion of the Heuristics Miner individual does not modify the best mined model of ProDiGen in any of the 111 logs tested in the results section. Nevertheless, the inclusion of this individual in the initial population *speeds up* the iteration at which the best individual is found, as the main dependency relations are captured by Heuristics Miner —these dependencies are more robust than the ones defined in [30]— and then, with ProDiGen, the different inputs and outputs bindings are optimized.

Evaluation

Individuals of the population are evaluated with a hierarchical fitness function that takes into account completeness, precision and simplicity.

Completeness

A natural definition for completeness would be the number of properly parsed traces divided by the total number of event traces . However, this definition is not able to distinguish

between two individuals that cannot process a trace; e.g., one of them because an arc is missing and the other one because the model is totally incorrect. For this reason, we use the definition of completeness (C_f) described in [30], which takes into account the number of correctly parsed tasks[3], but also the number of missing and not consumed tokens of the Petri net encoded in the individual —each missing or not consumed token represents a failure.

Precision

The measurement of the precision of a mined model is difficult, as precision has to detect the extra behavior, i.e., paths in the model that are not represented in the log. Therefore, our definition of precision considers all the activities that are enabled while an individual parses the log:

$$P_f(L, CM) = \frac{1}{allEnabledActivities\,(L, CM)} \tag{2.1}$$

where *allEnabledActivities* is the number of enabled activities when a log L is processed by an individual CM. *allEnabledActivities* is evaluated by counting the number of enabled activities after firing each activity of a trace. This process is performed for every trace in the log. Thereby, with this definition for the precision ProDiGen punishes those models with too many enabled activities, as each enable activity represents a possible path for extra behavior. Contrary to [30], we do not consider the rest of the population in order to compute the precision of each individual, which can evolve without taking into account the precision of the rest of the population.

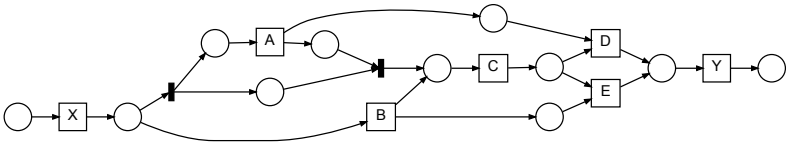
Simplicity

The third dimension of the fitness is simplicity, which measures the complexity of a mined model based on the number of causal relations of an individual:

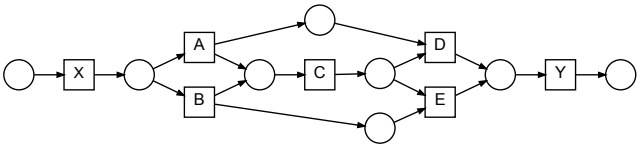
$$S_f(CM) = \frac{1}{\sum_{t \in CM} \left(\sum_{\Phi \in I(t)} |\Phi| + \sum_{\Psi \in O(t)} |\Psi| \right)} \tag{2.2}$$

where t is a task of the causal matrix CM, Φ is an element of I(t) —the tasks in Φ have an OR-join relation—, and Ψ is an element of O(t) —the tasks in Ψ have an OR-split relation. Therefore, the simplicity counts the number of causal relations of the model using the cardinality of the input and output subsets of the causal matrix.

[3] If a task from an individual does not have the proper input arcs, that task will be incorrectly parsed when reproducing the log, as its input conditions are not fulfilled.



(a) Mined model with needless branches.



(b) Simplest mined model.

Figure 2.4: Two possible solutions with the same completeness and precision.

We illustrate the relevance of simplicity to mine the original model with a simple example of two traces repeated three times: $<<X,B,C,E,Y>^3, <X,A,C,D,Y>^3>$. Figure 2.4 shows two mined models that discover the non-free-choice[4] construction, and have the same completeness and precision: *i)* both can parse exactly the same tasks, i.e., $completeness = 1.0$; and *ii)* they enable exactly the same number of tasks during the parsing (36), thus $precision = 1/36$. However, the model in Figure 2.4(a) has a $simplicity = 1/24$ while the model in Figure 2.4(b) has a $simplicity = 1/20$ and, therefore, the second one is a better model. The difference between these two solutions —in terms of simplicity— is caused by the output function of the task X and the input function of the task C: *i)* the causal matrix of the model in Figure 2.4(a) has $O(X) = \{\{A,B\},\{B,C\}\}$, and $I(C) = \{\{A,B\},\{B,X\}\}$ which increases the complexity of the model by 8; *ii)* the causal matrix of the model in Figure 2.4(b) has $O(X) = \{\{A,B\}\}$, and $I(C) = \{\{A,B\}\}$ which increases the complexity of the model by 4.

Fitness

ProDiGen uses completeness, precision and simplicity to evaluate the mined models. However, instead of combining these three objectives in a weighted sum —which requires the definition of a new weight parameter for each criteria— it defines a hierarchical fitness

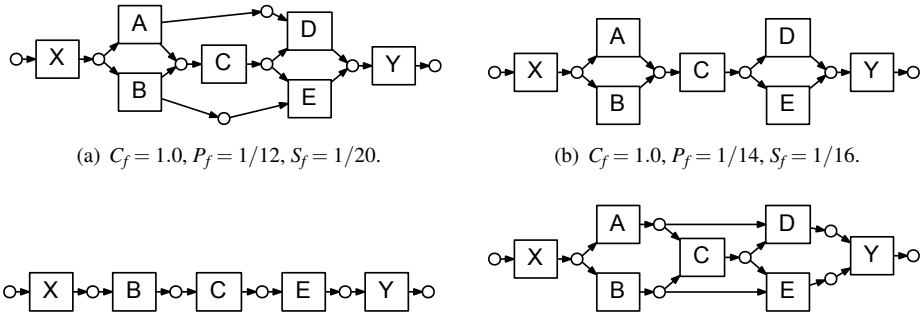
[4] A non-free-choice construction is a mixture of a synchronization and a choice [30]. For example, in Figure 2.4, the execution of D or E depends on whether the task A or B has been executed.

function that establishes priorities among the objectives:

$$F(a) > F(b) \iff \{C_f(a) > C_f(b)\} \vee \{C_f(a) = C_f(b) \wedge P_f(a) > P_f(b)\} \vee \{C_f(a) = C_f(b) \wedge P_f(a) = P_f(b) \wedge S_f(a) > S_f(b)\} \quad (2.3)$$

where $F(a)$, $C_f(a)$, $P_f(a)$ and $S_f(a)$, are respectively the *fitness*, *completeness*, *precision* and *simplicity* of a process model a. The advantage of using this hierarchical fitness function over a weighted fitness function is that, during the first stage of the evolutionary process, the GA focuses the search on those individuals that are complete. Once these individuals become representative in the population, the second level of the hierarchy takes the control, modifying the models that are complete in order to improve their precision. Finally, in the third stage, the fitness function guides the GA to improve the simplicity of those models that are both complete and precise.

If we change the hierarchical order of the fitness measure, the algorithm may find a different solution, as completeness, precision and simplicity are three opposed objectives. Figure 2.5 shows four different models that can be found mining two traces: $<< X,B,C,E,Y >$, $< X,A,C,D,Y >>$ but prioritizing different objectives. For example, both the models of Figure 2.5(b) and Figure 2.5(a) have the same completeness but, in order to achieve a better simplicity, the solution of Figure 2.5(a) retrieves a lower precision than the solution of Figure 2.5(b). If we want to retrieve a simple model (Figure 2.5(c)) the solution will have a lower completeness. On the other hand, if we want to retrieve a model with a better precision (Figure 2.5(d)), both the completeness and the simplicity will be lower.



(a) $C_f = 1.0$, $P_f = 1/12$, $S_f = 1/20$.

(b) $C_f = 1.0$, $P_f = 1/14$, $S_f = 1/16$.

(c) $C_f = 0.2$, $P_f = 1/18$, $S_f = 1/8$.

(d) $C_f = -0.2$, $P_f = 1/6$, $S_f = 1/20$.

Figure 2.5: Four different models prioritizing the different search criteria.

Genetic operators

The process to create new individuals starts with the selection phase. ProDiGen uses as selection mechanism a binary tournament selection, in which two individuals of the population are randomly picked —with replacement—, and the best of them is selected. Then, each pair of individuals in the selected population is crossed and mutated.

Crossover

The crossover operator replaces causality relations of an individual with causality relations of another individual. As the process models are represented through causal matrices, and the size of the causal matrix increases with the number of activities in the log, the number of possible crossover points could be really large. Therefore, picking the crossover point at random produces a poor performance of the crossover operator, as most of *the offspring have a fitness lower than their parents*. ProDiGen makes the selection of the activity that is going to be crossed using a non uniform PDF. This PDF assigns a null probability of being selected to those activities that have been correctly fired during the parsing of the traces in the log. On the other hand, those activities that were incorrectly fired receive a uniform probability, inversely proportional to the number of incorrectly parsed activities, of being crossed.

Algorithm 2.2: Crossover operator.

```
1  r ← getRandomNumber() // returns a random number between [0,1)
2  if r < crossoverRate then
3      incorrectlyFiredActivities ← ∅
4      if fitness(parent_1) >= fitness(parent_2) then
5          incorrectlyFiredActivities ← set of incorrectly fired activities of parent_1
6      else
7          incorrectlyFiredActivities ← set of incorrectly fired activities of parent_2
8      if incorrectlyFiredActivities ≠ ∅ then
9          crossoverPoint ← randomly select an activity t from incorrectlyFiredActivities
10     else
11         crossoverPoint ← randomly select an activity t from the bag of all possible
             tasks in the log
12     offspring_1, offspring_2 ← doCrossover(parent_1, parent_2, crossoverPoint)
13     Repair offspring_1 and offspring_2
```

Algorithm 2.2 summarizes the behavior of the crossover operator. By incorrectly fired activities we mean *i)* activities that need extra tokens in their inputs to be fired, i.e., tasks that do not have the correct input arcs, and *ii)* activities that have left tokens in their outputs after the parsing, i.e., activities that do not have the correct output arcs. This process generates, for each individual, a bag of *incorrectlyFiredActivities*. Thereby, the crossover point is selected from the set of *incorrectlyFiredActivities* of the fittest parent (Algorithm 2.2:4). Note that if the fittest individual has a completeness equal to 1, the set of *incorrectlyFiredActivities* of the fittest individual is empty (Algorithm 2.2:8); thereby the crossover point is randomly chosen from the bag of all the possible tasks in the log (Algorithm 2.2:11).

After the crossover point is selected, the crossover (Algorithm 2.2:12) is performed as defined in [30]. The following example illustrates how the crossover operator works. Let's suppose that the crossover point is task A, and the input of the individual 1 for that task is $I_1(A) = \{\{D\},\{B\}\}$ and for individual 2 is $I_2(A) = \{\{D,B\},\{C\}\}$. First, both input sets are split randomly. This process generates four sets, two for each input function:

- $I_1^1(A) = \{\{D\}\}$ and $I_1^2(A) = \{\{B\}\}$.
- $I_2^1(A) = \{\{D,B\}\}$ and $I_2^2(A) = \{\{C\}\}$.

Then, these four sets are swapped — $I_1^1(A)$ with $I_2^2(A)$ and $I_2^1(A)$ with $I_1^2(A)$— in order to generate the new input sets for the task A of the offspring. During this process, each subset of these sets can be added as a new subset into the other set or merged with another subset of the other set. For instance, the subset of $I_1^1(A)$ can be added as a new subset of $I_2^2(A)$, resulting in $I_1'(A) = \{\{D\},\{C\}\}$; and the subset of $I_2^1(A)$ can be joined with the existing subset of $I_1^2(A)$, resulting in $I_2'(A) = \{\{D,B\}\}$ —subsets do not allow duplicate tasks. Finally, this process is repeated for the output set of the crossover task A.

Note that when adding/removing causal relations from an input/output (I/O) set of a task t —being t the *crossoverPoint*— there may be inconsistencies. For instance, a task t' does not appear in the output set of the task t, but the input set of t' contains the task t. Therefore, after an I/O set is modified, we have to check the consistency of the individual (Algorithm 2.2:13). This process first checks those relations that were removed during the modification of the individual, and after that, it checks those relations that were created. This prevents to create again a relation that was previously removed.

Mutation

The mutation operator modifies the causality dependencies of the individual by adding or removing relations. The mutation operator may perform one of the following three actions to the input/output set of a task: *i)* randomly add a task t' to input or output sets of a task t; *ii)* randomly remove a task t' from the I/O sets of a task t; and *iii)* randomly redistribute the elements from the I/O sets of a task t.

Algorithm 2.3: Mutation operator.

```
1  while the individual does not change do
2      Randomly choose one task t in the individual
3      mutationType ← getRandomNumber() // returns a random number
         between [0,1)
4      if mutationType < 1/3 then
5          Randomly add a new task t' to I(t), being t' a task from inputDependencies(t)
6          if getRandomNumber() < 1/2 then
7              Randomly choose one subset X ∈ I(t) and add the task t' to X
8          else
9              Create a new subset X, add the task t' to X, and add X to I(t)
10     else if mutationType < 2/3 then
11         Randomly choose one subset X ∈ I(t) and remove a task t' from X, where t' ∈
             X. If X is empty after this operation, exclude X from I(t)
12     else
13         Randomly redistribute the elements from I(t)
14     Repeat from line 3, but using O(t) instead of I(t) and outputDependencies(t)
         instead of inputDependencies(t)
15     Repair the individual
```

There are four differences between our mutation operator and the one used in Genetic Miner [30]: *i)* the individual is iteratively mutated until it is different from its parent —a mutation could generate an individual equal to its parent due to a reparation; *ii)* only one task is affected by the mutation operator; *iii)* individuals are always forced to mutate —the mutation probability is 1; and *iv)* the task t' added to the I/O set of a task t must belong to the set of tasks that have an input/output dependency with t. The major goal of these modifications is to avoid duplicate individuals within the same population, or at least minimize its duplicates. Hence, although the offspring are equal to their parents after the crossover, we force each

offspring to mutate until it changes, creating different individuals with new features. With these modifications, we have a more diverse population.

Algorithm 2.3 describes in detail the mutation operator. It uses two sets for the addition of a new task: *outputDependencies(t)* and *inputDependencies(t)*. Both sets are created when calculating the dependencies between tasks at the first stages of the algorithm. ProDiGen uses these sets to reduce the set of tasks that are appropriate to be inserted in an I/O set, preventing the inclusion of a new task that never appears in a trace of t within the log. A first approach could be to include in the dependencies sets those tasks that have a dependency with t as calculated in the initialization phase. However, if we only take into account these dependencies, there will be not enough new material to discover, for instance, the non-free-choice constructs. Therefore, *inputDependencies(t)* will be the set of tasks appearing before t in any trace of the log and, in the same way, *outputDependencies(t)* will be the set of activities that appear after t in any trace of the log. In this way, the mutation operator focuses only on those regions of the search space that represent information contained in the log. As a result, the success of the mutation operator increases, finding better offspring. Again, after each mutation the individual has to be repaired (Algorithm 2.3:15) following the same strategy as explained with the crossover.

Replacement

At each iteration, the algorithm generates N offspring, being N the size of the population. These offspring and the parent population —current population— are joined and sorted —using the fitness— generating a $2N$-size population, and then the replacement operator selects the N best individuals. In order to maintain a diverse population, those repeated individuals are placed at the bottom of the ranking, keeping one representative in the original ranking position.

Reinitialization

A reinitialization takes place when the value of *timesRun* goes under 0 (Algorithm 2.1:15), which indicates that the search process was not improving in the last iterations. This situation is detected in two ways. The first one (Algorithm 2.1:11) is when the new population of an iteration has no new individuals —in comparison with the initial population of that iteration. The second indicator (Algorithm 2.1:13) is the fact that the best individual does not improve. Each time that one of these situations is detected, *timesRun* decreases. The initial population

after a reinitialization is generated in the same way as in the initialization stage. Moreover, ProDiGen also includes in the new population a mutation of the best individual of the last iteration. The maximum number of reinitializations is limited, and when it reaches the threshold (*maxRestarts*) ProDiGen ends.

2.5.2 Pre-processing and post-processing steps

Noise can be defined as a low-frequent incorrect behavior in the log [140]. The main difficulty to deal with noise is that the logs —usually— contain only positive examples, i.e., there is no explicit information about the characteristics of the noisy traces. Additionally, there is low-frequent correct behavior in the log that cannot be easily distinguished from the low-frequent incorrect cases.

ProDiGen explicitly handles the noise in two phases: *i)* a pre-processing of the log; and *ii)* a post-processing of the mined model. In the pre-processing of the log, ProDiGen groups all the traces that are equal, and calculates the normal distribution of the frequency of the traces $\mathcal{N}(\mu,\sigma)$, where μ and σ are respectively the weighted mean and the weighted standard deviation. Finally, all those traces with *frequency(trace)* $< \mu - \xi\sigma$ will be removed, where ξ is a parameter. Therefore, ProDiGen eliminates those traces that are infrequent, and the threshold depends on the characteristics of the log.

The post-processing stage of ProDiGen consists in a post-pruning over the mined model. It removes those arcs that are used less frequently than a certain threshold, being this threshold a percentage of the frequency of the most used arc. This post-pruning was also applied in other process discovery techniques [30, 52, 58].

2.6 Experimentation

ProDiGen has been validated with 111 different logs. We have classified these tests in two different groups: *i)* 18 process models to generate logs with five noise levels —0%, 1%, 5%, 10%, and 20%—, which results in 90 logs; and *ii)* 21 unbalanced logs, i.e., logs that contain traces with very different frequencies, which correspond with 21 process models that contain many interleaving situations.

Moreover, we have also compared the performance of ProDiGen with four of the state of the art process discovery algorithms, using non-parametric statistical tests. The selected algorithms are: *i)* α^{++}-algorithm [156]; *ii)* Heuristics Miner (HM) [155]; *iii)* Genetic Miner

(GM) [30]; and *iv)* ILP [138]. These are well-known algorithms for process models discovery from event logs, and they are available in the ProM framework [139], a very complete and excellent tool for process mining and analysis.

2.6.1 Logs

The algorithms have been tested with 39 process models, and a total of 111 different logs[5]. From those models, 18 out of 39 were used to generate 90 synthetic logs with different degrees of noise. The rest of the logs —21 out of 111— come directly from [19] and [30] with their corresponding original models.

Balanced logs

We have conducted an experiment with 18 different process models with increasing degrees of complexity. Table 2.2 summarizes the structural complexity of these models ranging from 5 to 16 tasks that contain sequences, choices, parallelism, loops and non-free-choice constructs. For each of these models, a *synthetic* log was randomly generated with all the possible paths represented in the model. Table 2.2 also shows the characteristics of the logs, where column *#traces* indicates the number of traces and the column *#events* the number of total activities in the event log.

Afterwards, each noise-free log was used to generate another four logs with 1%, 5%, 10% and 20% of noise. Four different types of noise were used [83]: *i) missing head*; *ii) missing body*; *iii) missing tail*; and *iv) swap tasks*. We followed the same strategy as [30] to generate the noisy traces. Assuming that each trace is defined as $\sigma = t_1 \cdots t_n$, these noise types behave as follows. The first three types, respectively, randomly remove sub-traces of events from the head, body and tail. The head goes from t_1 to $t_{n/3}$, the body goes from $t_{(n/3)+1}$ to $t_{2n/3}$ and the tail goes from $t_{(2n/3)+1}$ to t_n. Swap noise interchanges two random chosen events.

To incorporate noise, the traces of the original noise-free logs were randomly selected and then one of the four noise types was applied —each one with an equal probability of 0.25. This combination of noise types is called *mixed noise*. We focus our tests on this noise type, as it is the typical noise in real logs. Note that the number of traces of each log is the same after applying the noise, but the number of events may change —some type of noise may remove events from cases, reducing the number of total events in the log.

[5]The reader can found all the logs and models used in our experimentation in `http://tec.citius.usc.es/processmining`.

Table 2.2: Process models used in the experimentation. Balanced logs.

	Activity structures										*Log content*	
Model	**#Tasks**	**Sequence**	**Choice**	**Parallelism**	**Length-One Loop**	**Length-Two Loop**	**Arbitrary Loop**	**Structured Loop**	**Non-local NFC**	**Local NFC**	**Invisible tasks**	**#traces**	**#events**
Caminatas	12	✓	✓	✓								700	4,200
A8	7	✓	✓	✓								300	1,200
D2	6	✓		✓								300	1,200
Ml1lSkip [30]	6	✓	✓	✓	✓							500	4,757
Ma5 [30]	7	✓	✓	✓	✓							300	2,178
Ml2l [30]	6	✓	✓			✓						300	4,668
MDriverLL [30]	11	✓	✓	✓				✓	✓		✓	700	13,303
allLoops	5	✓	✓		✓	✓		✓			✓	300	1,035
l2la	6	✓	✓	✓			✓					300	2,264
Ma7 [30]	9	✓	✓	✓								500	2,427
Herbst6p37 [30]	16	✓		✓								700	12,600
MexampleL [30]	8	✓	✓	✓								300	1,645
Ma6nfc [30]	8	✓	✓							✓		300	2,006
MParallel5 [30]	10	✓		✓								700	12,600
NC	7	✓	✓	✓						✓		300	1,704
L2LP	7	✓	✓	✓	✓	✓		✓			✓	300	5,476
NCB	7	✓	✓	✓			✓			✓		300	2,950
DWS [49]	12	✓	✓	✓							✓	500	4,033

Unbalanced logs

The second type of logs consisted in 21 more complex case scenarios without noise. These models and logs[6] are summarized in Table 2.3. Some of the models used in this experimentation contain *unbalanced AND-split/join points*, i.e., there is not a one-to-one relation between the AND-split points and the AND-join points. Moreover, all the logs are imbalanced, i.e., they contain traces with very different frequencies, as it is unrealistic to assume that, from a model with many interleaving situations, all the possible paths are equally executed. Hence, with this experiment, we can check whether an algorithm overfits or underfits the data due to the unbalanced frequencies of the traces in the log.

[6] In this experiment, both the process models and the logs were taken from other papers [19, 30].

Table 2.3: Process models used in the experimentation. Unbalanced logs.

	Activity structures									*Log content*	
Model	**#Tasks**	**Sequence**	**Choice**	**Parallelism**	**Length-One Loop**	**Length-Two Loop**	**Arbitrary Loop**	**Structured Loop**	**Invisible tasks**	**Unbalanced AND-join/split**	**#traces**	**#events**
g2 [30]	22	✓	✓	✓		✓	✓		✓		300	4501
g3 [30]	29	✓	✓	✓			✓	✓	✓		300	14599
g4 [30]	29	✓	✓	✓		✓				✓	300	5975
g5 [30]	20	✓	✓	✓				✓	✓		300	6172
g6 [30]	23	✓	✓	✓		✓			✓		300	5419
g7 [30]	29	✓	✓	✓			✓		✓		300	14451
g8 [30]	30	✓	✓	✓		✓	✓		✓	✓	300	5133
g9 [30]	26	✓	✓	✓		✓	✓		✓		300	5679
g10 [30]	23	✓	✓	✓				✓	✓		300	4117
g12 [30]	26	✓	✓	✓		✓		✓	✓		300	4841
g13 [30]	22	✓	✓	✓	✓	✓			✓	✓	300	5007
g14 [30]	24	✓	✓	✓			✓		✓	✓	300	11340
g15 [30]	25	✓	✓		✓	✓			✓		300	3978
g19 [30]	23	✓	✓	✓		✓			✓	✓	300	4107
g20 [30]	21	✓	✓		✓	✓		✓	✓		300	6193
g21 [30]	22	✓	✓					✓	✓		300	3882
g22 [30]	24	✓	✓	✓			✓		✓	✓	300	3095
g23 [30]	25	✓	✓	✓		✓				✓	300	9654
g24 [30]	21	✓	✓	✓				✓	✓	✓	300	4130
g25 [30]	20	✓	✓	✓	✓				✓		300	6312
ETM[19]	7	✓	✓	✓					✓		100	790

2.6.2 Metrics

The performance of the process discovery algorithms over the different logs has been measured with two different sets of metrics: *i)* metrics based on the original model; and *ii)* metrics based on the event log.

Metrics based on the original model

We use the metrics defined in [30] to compare the original and mined models. *Behavioral precision* (B_p) and *Behavioral recall* (B_r) detect, respectively, if the mined model can process traces that cannot be parsed by the original model, and if the original model can parse traces that cannot be processed in the mined model. On the other hand, *Structural precision* (S_p) and *Structural recall* (S_r) checks, respectively, if there are causality relations of the mined model

that are not defined in the original model, and if there are causality relations of the original model that are not defined in the mined model.

The mined model is as precise as the original one if $B_p = 1$ and $B_r = 1$: the closer the values of B_p and B_r to 1, the higher the similarity between the original and the mined models. Although two models could be equal from the *behavioral point of view*, their structures may be different. S_p and S_r measure the similarity from the *structural point of view*. When the original model has connections that do not appear in the mined model, S_r will take a value smaller than 1, and, in the same way, when the mined model has connections that do not appear in the original model, S_p will take a value lower than 1.

Metrics based on the log

Additionally to the four previously described metrics, we have also used three metrics to measure the completeness, precision and simplicity taking into account the information of the log. To measure the completeness (C), we use the *proper completion* metric [105], which is the fraction of properly completed process instances. *Proper completion* takes a value of 1 if the mined model can process all the traces without having missing tokens or tokens left behind. Also, the precision (P) is evaluated as follows:

$$P = 1 - max\{0, P'_o - P'_m\} \tag{2.4}$$

where P'_o and P'_m are, respectively, the precision of the original model and the precision of the mined model, both calculated with the *alignment precision* defined in [124]. As the original model is the optimal solution, we use it to normalize the precision. Therefore, P will be equal to 1 if the mined model has a precision (P'_m) equal or higher than the original model (P'_o). When the precision of the mined model is worse than that of the original model, P will take a value under 1 —the lower the precision of the mined model, the closer the value of P to 0. Finally, for the simplicity (S) we use:

$$S = \frac{1}{1 + max\{0, S'_m - S'_o\}} \tag{2.5}$$

where S'_m and S'_o are, respectively, the simplicity of the mined model and the simplicity of the original model, both calculated with the *weighted P/T average arc degree* defined in [111] —the higher the value of S' the lower the simplicity. As explained with the precision, we use the original model —which is the optimal model— to normalize the simplicity. S takes a value of 1 if the simplicity of the mined model is equal or higher than that of the original

model, i.e., $S'_m \leq S'_o$. If the simplicity of the mined model is worse than that of the original model ($S'_m > S'_o$), S will take a value under 1 —the worse the simplicity of the mined model, the closer the value of S to 0. To measure the metrics C, P' and S' we have used the tool CoBeFra [17].

2.6.3 Settings

The settings of the different algorithms were mostly kept to the default options of ProM 5.2 and ProM 6.3. However, some modifications were made for Heuristics Miner and Genetic Miner to keep the configurations specified by the authors of the algorithms. To be more specific, the settings used are:

- α^{++}*-algorithm*: the algorithm has no settings (ProM 5.2).
- *Heuristics Miner*. We used the default settings established in ProM 6.3: relative-to-best = 0.05, dependency = 0.9, length-one-loops = 0.9, lenght-two-loops = 0.9, long distance = 0.9. Additionally, *mine long distance dependencies* was enabled. These parameters were set for both the balanced and unbalanced logs.
- *Genetic Miner*. We set the values of the parameters equal to those established in [30], as 29 out of the 39 models used in this experimentation were also tested in [30]:
  - Settings for the balanced logs: iterations = 1,000, population size = 100, elitism rate = 0.2, crossover probability 0.8, mutation probability = 0.2 (per task), elitism rate = 0.02, selection type = tournament 5, extra behavior punishment = 0.025 and prune threshold = 0.1%.
  - Settings for the unbalanced logs: the same as for the balanced logs, except iterations = 5,000, population size = 10, elitism rate = 0.2, selection type = binary and prune threshold = 0%.
- *ProDiGen*:
  - Settings for the balanced logs: iterations = 1,000, population size = 100, crossover probability 0.8, initialTimesRun = 35, maxRestarts = 5, prune threshold = 0.1. For the pre-processing parameter $\xi = 2$, i.e., two standard deviations.
  - Settings for the unbalanced logs: the same as for the balanced logs, except prune threshold = 0%.

- *ILP*: we selected the default settings defined in ProM 6.3: ILP Solver = Java-ILP & LPSolve 5.5, ILP Variant = Petri net (Empty net after completion), number of places = Per Causal Dependency and *search for separate initial places* was enabled.

For ProDiGen, we kept exactly the same settings as Genetic Miner, except for the new parameters related with the reinitialization process, the pre-processing parameter ξ, and the mutation probability that in ProDiGen is always 1, as explained in Sec 13. The values of maxRestarts and initialTimesRun do not affect the results of ProDiGen —provided that they are not drastically reduced; increasing those values only augments the execution time of ProDiGen. Moreover, we did not modify ProDiGen parameters when dealing with more complex logs, contrary to Genetic Miner which requires to increase the maximum number of iterations to converge when the complexity of the log increases. Additionally, we did not apply any post-pruning process for the unbalanced logs for both ProDiGen and Genetic Miner, as these are noise-free logs.

2.6.4 Results on balanced logs

Tables 2.4-2.6 present the results of the algorithms with the balanced logs for the different percentages of noise. The rows show the values of the 4 metrics based on the original model and the 3 metrics based on the log. As both ProDiGen and Genetic Miner are non-deterministic algorithms, the results shown in the tables are the average results of 10 executions.

For the noise-free logs (Table 2.4) ProDiGen obtains the original model in all the cases, as the values for both the model and log metrics are 1, proving that it can handle all the main workflow constructs at once. On the other hand, Genetic Miner is not able to get the original model for several logs. For instance, for logs *Ma6nfc* and *Ma5*, Genetic Miner always gets a complete model but, on average —ten executions for each log—, the mined model is different from the original model. This is closely related to the simplicity of the mined model as there are needless relations that increase the complexity of the model. Heuristics Miner shows a great performance, but it has difficulties to mine some non-free-choice constructs and short loops. ILP has problems tackling invisible tasks and, finally, the α^{++}-algorithm fails when dealing with local non-free-choices and invisible tasks.

For the noisy logs (Tables 2.4-2.6) we have used the same metrics. However, to evaluate the seven metrics we did not consider the noisy traces, i.e., a completeness of 1 means that the model is able to process all the noise-free traces of the noisy log. ProDiGen shows again

Table 2.4: Results on the balanced logs with a 0% and 1% of noise.

			Logs with 0% of noise																		Logs with 1% of noise																	
			Caminatas	A8	D2	M11Skip	Ma5	M121	MDriverLL	allLoops	Ma7	12a	Mexample1	Herbst6p37	Ma6nfc	MParallel5	NC	L2LP	NCB	DWS	Caminatas	A8	D2	M11Skip	Ma5	M121	MDriverLL	allLoops	Ma7	12a	Mexample1	Herbst6p37	Ma6nfc	MParallel5	NC	L2LP	NCB	DWS
ProDiGen	Model metrics	B_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		B_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
GM	Model metrics	B_p	0.84	1.0	1.0	1.0	0.96	1.0	0.9	0.99	0.95	1.0	0.94	1.0	0.98	0.74	1.0	1.0	1.0	1.0	0.68	1.0	0.95	0.99	0.97	1.0	0.92	0.99	0.82	1.0	1.0	1.0	0.98	0.8	1.0	1.0	1.0	0.86
		B_r	0.98	1.0	1.0	1.0	1.0	1.0	0.99	0.99	1.0	0.97	1.0	1.0	0.97	1.0	1.0	1.0	1.0	1.0	0.99	0.99	1.0	0.96	0.99	0.99	1.0	1.0	0.99	1.0	1.0	0.99	1.0	0.98	1.0	1.0	1.0	0.81
		S_p	0.85	1.0	1.0	1.0	1.0	1.0	0.93	0.98	0.94	1.0	0.86	1.0	0.96	0.08	1.0	1.0	1.0	1.0	0.79	0.8	1.0	1.0	0.91	0.88	0.94	1.0	0.75	1.0	1.0	0.95	0.96	0.85	1.0	1.0	1.0	0.83
		S_r	0.73	1.0	1.0	1.0	0.91	1.0	0.84	0.96	0.97	1.0	0.88	1.0	0.99	0.98	1.0	1.0	1.0	1.0	0.67	1.0	0.87	0.92	0.91	1.0	0.85	0.96	1.0	1.0	1.0	1.0	0.99	0.93	1.0	1.0	1.0	0.95
	Log metrics	C	0.36	1.0	1.0	1.0	1.0	1.0	0.48	0.15	1.0	0.52	1.0	1.0	0.81	1.0	1.0	1.0	1.0	1.0	0.23	1.0	1.0	1.0	1.0	1.0	1.0	0.49	1.0	1.0	1.0	1.0	0.14	1.0	1.0	1.0	1.0	0.82
		P	0.81	1.0	1.0	1.0	0.97	1.0	0.93	0.92	0.68	1.0	0.97	1.0	0.81	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.63	0.96	0.98	0.88	0.88	0.82	0.76	1.0	1.0	0.94	0.92	0.75	1.0	1.0	1.0	0.93
		S	1.0	1.0	1.0	1.0	1.0	1.0	0.93	1.0	1.0	1.0	1.0	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.72	0.88	0.73	1.0	0.85	1.0	0.9	1.0	1.0	1.0	1.0	0.92	0.93	0.67	1.0	1.0	0.88
HM	Model metrics	B_p	1.0	1.0	1.0	0.97	1.0	1.0	1.0	0.96	1.0	1.0	1.0	1.0	0.91	1.0	0.44	0.82	0.88	0.89	1.0	1.0	1.0	0.97	1.0	1.0	0.93	0.96	1.0	1.0	1.0	1.0	0.9	1.0	0.87	0.82	0.88	0.89
		B_r	1.0	1.0	1.0	0.95	1.0	1.0	1.0	0.92	1.0	1.0	1.0	1.0	1.0	1.0	0.82	0.93	1.0	0.94	1.0	1.0	1.0	0.92	1.0	1.0	1.0	0.92	1.0	1.0	1.0	1.0	1.0	1.0	0.93	1.0	0.94	
		S_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.81
		S_r	1.0	1.0	1.0	0.82	1.0	1.0	1.0	0.83	1.0	1.0	1.0	1.0	0.9	1.0	0.91	0.9	0.88	1.0	1.0	1.0	1.0	0.76	1.0	1.0	0.9	0.83	1.0	1.0	1.0	1.0	0.91	1.0	0.91	0.9	0.88	1.0
	Log metrics	C	1.0	1.0	1.0	0.84	1.0	1.0	1.0	0.73	1.0	1.0	1.0	1.0	0.69	1.0	0.84	0.24	0.0	0.51	1.0	1.0	1.0	0.84	1.0	1.0	1.0	0.73	1.0	1.0	1.0	1.0	0.69	1.0	0.84	0.24	0.0	0.51
		P	1.0	1.0	1.0	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.88	1.0	0.83	1.0	0.14	1.0	1.0	1.0	1.0	0.89	1.0	1.0	0.91	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.87	1.0	0.14	1.0
		S	1.0	1.0	1.0	1.0	1.0	1.0	0.75	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.88	1.0	0.98		1.0	1.0	1.0	0.98	1.0	1.0	1.0	0.79	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.89	1.0	0.98
α^{++}	Model metrics	B_p	1.0	1.0	1.0	1.0	1.0	1.0	0.87	0.77	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.59	0.88	0.75	0.76	0.82	1.0	0.73	0.84	0.76	0.77	0.94	0.83	0.73	0.4	0.41	0.6	0.87	0.76	0.77	0.88	0.46
		B_r	1.0	1.0	1.0	1.0	1.0	1.0	0.89	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.72	1.0	0.63	0.87	0.87	1.0	0.63	0.82	0.69	0.75	0.83	0.85	0.6	0.7	0.52	0.78	0.99	0.86	0.56	1.0	0.45
		S_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.72	1.0	0.81	0.93	0.77	1.0	0.81	0.83	1.0	0.79	0.81	0.83	0.69	0.69	0.46	0.68	0.68	0.84	1.0	1.0	0.74
		S_r	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.88	1.0		1.0	0.87	1.0	0.69	0.83	0.75	0.9	0.75	0.83	0.81	0.75	0.61	0.91	0.86	0.91	0.6	0.88	0.9
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	0.63	0.25	1.0	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	1.0	1.0	0.0	0.25	1.0	0.0	0.0	0.0	0.0	0.0	1.0	0.0	0.0	0.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.97	1.0	1.0	1.0	1.0	1.0	1.0	0.44	0.14	0.23		0.0	1.0	1.0	0.19	0.96	0.83	0.11	0.96	0.0	0.05	0.05	0.05	0.0	0.01	0.0	0.45	0.14	0.23
		S	1.0	1.0	1.0	1.0	1.0	1.0	0.81	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.52	1.0	0.71		0.83	0.86	1.0	0.75	0.63	1.0	0.19	0.72	1.0	0.75	0.8	0.72	0.49	0.75	0.83	0.52	0.83	0.35
ILP	Model metrics	B_p	1.0	1.0	1.0	1.0	1.0	1.0	0.86	0.81	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.63	1.0	0.82	0.3	0.39	0.64	0.58	0.84	0.63	0.32	0.31	0.42	0.48	0.47	0.33	0.47	0.3	0.52	0.62	0.63	0.42
		B_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.92	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.97		0.68	0.56	0.83	0.75	0.89	0.9	0.68	0.53	0.69	0.7	0.75	0.89	0.78	0.74	0.84	0.9	0.99	0.56
		S_p	1.0	1.0	1.0	1.0	1.0	1.0	0.86	0.54	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.66	1.0	0.66	0.28	0.53	0.63	0.69	0.75	0.76	0.32	0.53	0.43	0.34	0.41	0.87	0.49	0.45	0.45	0.66	0.66	0.29
		S_r	1.0	1.0	1.0	1.0	1.0	1.0	0.93	0.71	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0		1.0	0.83	1.0	1.0	0.87	1.0	1.0	1.0	0.95	0.92	0.88	0.96	1.0	0.98	1.0	1.0	1.0	1.0
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	0.93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.91	1.0	0.95		0.47	0.5	0.72	0.82	0.96	0.7	0.64	0.79	0.57	0.05	0.79	0.6	0.62	0.74	0.63	0.92	0.67	0.74
		S	1.0	1.0	1.0	1.0	1.0	1.0	0.99	0.61	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.63		0.3	0.64	0.59	0.56	0.63	0.88	0.22	0.34	0.42	0.34	0.53	0.56	0.49	0.38	0.39	0.9	0.88	0.2

a great performance, getting the original models for all the logs with a noise percentage up to 5%, and all the original models but one for a percentage of 10%. For the logs with a percentage of the 20% of noise the performance decreases, although ProDiGen finds 6 out of 18 of the original models and is the best algorithm. These results are related with the fitness definition of ProDiGen, that tries to better fit the behavior of the log. This proves that the pre-processing —filtering the less used traces— and post-processing —pruning the infrequent arcs— improves the results of ProDiGen.

With respect to the results of the other algorithms, Heuristics Miner obtains the same results as for the noise-free logs with very low noise levels (1%), but, the higher the frequency of noise, the worse the results. Heuristics Miner is known to be good retrieving the main behavior of the log, hence, since these logs contain a mixture of all noise types, some noise dependencies can be as frequent as the correct ones, resulting in incorrect models —excluding those noise-free logs that the Heuristics Miner cannot correctly mine. Nevertheless, with high levels of noise, it obtains very competitive results. On the other hand, ILP usually gets a complete model, but it produces very complex and underfitted models because of its inability

Table 2.5: Results on the balanced logs with a 5% and 10% of noise.

			Logs with 5% of noise																Logs with 10% of noise														
			Caminatas	A8-	D2	MIISkip	Ma5	MI2I	MDriverLicensed	allLoops	Ma7	I2la	MexampleLog	MherbstFitGp37	MaioHc	MParallel5	NC	L2LP	NCB	DWS	Caminatas	A8-	D2	MIISkip	Ma5	MI2I	MDriverLicensed	allLoops	Ma7	I2la	MexampleLog	MherbstFitGp37	MaioHc	MParallel5	NC	L2LP	NCB	DWS
ProDiGen	Model metrics	B_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.57	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		B_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.73	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.78	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.86	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
GM	Model metrics	B_p	0.8	0.91	0.94	0.99	0.84	1.0	0.88	0.99	0.94	0.94	0.87	0.79	1.0	0.86	0.85	0.95	1.0	0.86	0.92	0.83	0.8	1.0	0.87	1.0	0.92	0.99	0.77	0.87	0.99	0.92	0.88	0.86	0.89	1.0	0.82	0.84
		B_r	0.99	0.99	0.99	0.96	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.99	0.99	0.99	1.0	1.0	0.81	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	0.99	1.0	0.99	0.99	0.99	0.99	0.99	0.99	0.83
		S_p	0.66	0.77	0.77	0.92	0.66	0.88	0.8	1.0	0.8	0.9	0.68	0.76	0.92	0.66	0.69	0.9	1.0	0.83	0.84	0.6	0.75	0.92	0.9	0.88	0.9	0.98	0.57	0.75	0.92	0.95	0.83	0.66	0.91	0.9	0.72	0.79
		S_r	0.5	0.87	0.87	0.92	0.66	1.0	0.76	0.96	1.0	0.9	0.91	0.95	1.0	0.66	0.75	0.9	1.0	0.95	0.78	0.75	0.75	1.0	0.83	1.0	0.85	0.96	0.66	0.81	1.0	0.95	0.83	0.66	0.91	1.0	0.88	0.95
	Log metrics	C	0.22	1.0	1.0	1.0	0.69	1.0	0.38	1.0	0.86	1.0	1.0	0.76	1.0	1.0	0.55	0.77	1.0	0.27	0.23	0.5	1.0	1.0	1.0	1.0	1.0	0.61	0.59	1.0	0.53	1.0	1.0	1.0	0.17	1.0	1.0	0.57
		P	0.68	0.63	0.86	0.86	0.75	0.81	0.64	0.75	0.77	0.9	0.58	0.65	0.91	0.47	0.62	0.99	1.0	0.82	0.78	0.55	0.45	0.86	0.6	0.58	0.73	0.65	0.51	0.48	0.71	0.79	0.42	0.58	0.75	1.0	0.7	0.76
		S	1.0	0.69	0.88	0.64	0.92	0.85	0.94	0.85	0.76	1.0	0.85	0.81	0.76	0.78	0.87	1.0	1.0	0.89	1.0	0.69	0.91	0.73	0.87	0.71	0.95	0.79	0.85	0.84	0.77	0.88	0.74	0.77	0.83	0.83	0.93	0.84
HM	Model metrics	B_p	0.96	1.0	1.0	0.97	0.92	0.91	0.92	0.93	1.0	1.0	1.0	1.0	0.9	1.0	0.87	0.77	0.88	0.77	0.96	1.0	0.91	1.0	0.92	0.86	0.92	0.93	1.0	0.94	1.0	1.0	0.9	1.0	0.87	0.96	0.88	0.86
		B_r	1.0	1.0	1.0	0.92	0.79	0.99	0.95	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.95	1.0	0.94	1.0	1.0	1.0	0.89	0.79	0.99	0.95	0.91	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.81
		S_p	1.0	1.0	1.0	1.0	1.0	0.88	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	0.83	1.0	1.0	1.0	1.0	1.0	0.8	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	1.0	0.83
		S_r	0.92	1.0	1.0	0.76	0.91	1.0	0.9	0.83	1.0	1.0	1.0	1.0	0.91	1.0	0.91	0.8	0.88	1.0	0.93	1.0	0.75	0.84	1.0	1.0	0.9	0.83	1.0	0.9	1.0	1.0	0.91	1.0	0.91	0.9	0.88	0.95
	Log metrics	C	0.66	1.0	1.0	0.0	0.0	1.0	0.63	0.26	1.0	1.0	1.0	0.69	1.0	0.85	0.77	0.0	0.0		1.0	1.0	1.0	0.9	0.0	1.0	0.62	0.34	1.0	0.3	1.0	1.0	0.69	1.0	0.84	0.0	0.0	0.0
		P	0.92	1.0	1.0	0.19	0.1	0.72	0.98	1.0	1.0	1.0	1.0	0.9	1.0	0.87	0.99	0.14	0.23		1.0	1.0	0.91	0.19	0.1	0.71	0.99	0.96	1.0	0.93	1.0	1.0	0.9	1.0	0.87	0.45	0.14	0.23
		S	1.0	1.0	1.0	0.91	1.0	0.91	1.0	0.79	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.92		1.0	1.0	1.0	0.88	1.0	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.96	1.0	0.91	
α^{++}	Model metrics	B_p	0.73	0.81	0.69	0.34	0.75	0.63	0.38	0.68	0.83	0.72	0.55	0.38	0.8	0.45	0.75	0.76	0.23	0.57	0.85	0.84	0.84	0.86	0.64	0.61	0.35	0.74	0.84	0.65	0.63	0.33	0.54	0.63	0.62	0.68	0.32	0.43
		B_r	0.69	0.99	0.83	0.32	0.88	0.68	0.49	0.67	0.97	0.65	0.6	0.47	0.88	0.54	0.99	0.56	0.24	0.66	0.95	0.99	0.99	0.66	0.64	0.65	0.45	0.77	0.99	0.68	0.69	0.41	0.6	0.75	0.97	0.52	0.36	0.56
		S_p	0.62	0.41	0.57	0.31	0.46	0.3	0.41	0.5	0.47	0.43	0.36	0.4	0.55	0.37	0.5	0.6	0.5	0.48	0.61	0.46	0.71	0.46	0.3	0.35	0.35	0.41	0.44	0.58	0.37	0.36	0.31	0.29	0.25	0.45	0.66	0.31
		S_r	0.82	0.62	0.5	0.38	0.5	0.37	0.57	0.58	0.66	0.63	0.66	0.56	0.91	0.6	0.83	0.6	0.66	0.72	0.75	0.75	0.62	0.53	0.5	0.62	0.61	0.41	0.66	0.63	0.75	0.45	0.58	0.66	0.33	0.5	0.44	0.42
	Log metrics	C	0.0	0.49	1.0	0.0	1.0	0.0	0.0	0.0	0.86	0.0	0.0	0.0	0.69	0.0	0.0	0.0	0.0	0.0	0.0	1.0	1.0	0.0	0.0	1.0	0.0	0.0	0.57	0.82	0.0	0.0	0.0	0.0	0.84	1.0	0.0	0.0
		P	0.01	0.68	0.94	0.19	0.84	0.16	0.11	0.16	0.84	0.05	0.05	1.0	0.0	0.01	0.0	0.45	0.14	0.23	0.0	0.84	0.84	0.19	0.1	0.15	0.11	0.15	0.76	0.75	0.05	0.99	0.0	0.01	0.75	1.0	0.14	0.23
		S	0.36	0.64	1.0	0.28	0.81	0.36	0.15	0.26	0.68	0.28	0.36	1.0	0.55	0.55	0.4	0.39	0.3	0.25	0.52	0.6	1.0	0.3	0.32	0.36	0.19	0.34	0.64	0.41	0.35	1.0	0.37	0.37	0.59	0.81	0.99	0.22
ILP	Model metrics	B_p	0.13	0.15	0.24	0.48	0.31	0.14	0.07	0.27	0.05	0.14	0.18	0.1	0.24	0.34	0.28	0.36	0.26	0.25	0.16	0.25	0.2	0.33	0.32	0.2	0.12	0.28	0.21	0.12	0.15	0.05	0.13	0.17	0.1	0.41	0.24	0.15
		B_r	0.3	0.48	0.83	0.33	0.62	0.42	0.21	0.81	0.17	0.44	0.36	0.45	0.59	0.64	0.4	0.69	0.75	0.49	0.25	0.72	0.58	0.47	0.64	0.5	0.19	0.64	0.38	0.42	0.29	0.15	0.43	0.29	0.32	0.72	0.47	0.24
		S_p	0.23	0.23	0.32	0.26	0.26	0.17	0.16	0.33	0.23	0.21	0.13	0.09	0.16	0.16	0.23	0.3	0.27	0.13	0.24	0.19	0.34	0.27	0.18	0.18	0.15	0.36	0.19	0.25	0.16	0.06	0.16	0.13	0.18	0.28	0.28	0.12
		S_r	1.0	1.0	0.8	0.95	0.93	1.0	1.0	1.0	1.0	0.92	0.92	0.94	1.0	0.78	1.0	1.0	0.91	0.96	1.0	0.83	0.8	1.0	0.87	1.0	1.0	0.95	1.0	1.0	1.0	1.0	0.93	1.0	1.0	0.92		
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0.32	0.4	0.41	0.49	0.1	0.42	0.28	0.6	0.57	0.34	0.43	0.05	0.41	0.42	0.47	0.77	0.44	0.58	0.35	0.4	0.43	0.52	0.1	0.4	0.27	0.59	0.42	0.41	0.41	0.05	0.36	0.3	0.31	0.8	0.41	0.5
		S	0.14	0.27	0.74	0.16	0.27	0.24	0.1	0.52	0.26	0.21	0.19	0.13	0.18	0.19	0.22	0.38	0.33	0.13	0.17	0.31	0.44	0.22	0.29	0.26	0.1	0.4	0.23	0.24	0.21	0.07	0.2	0.12	0.19	0.35	0.28	0.15

to handle infrequent behavior. Finally, α^{++} is not robust to noise and, therefore, almost all of its mined models are not even complete. In summary, ProDiGen correctly mines, i.e., finds the original model, the 85% (77 out of 90) of the cases. For the other algorithms, the percentage of correctly mined models was: GM in the 16% (16 out of 90), HM in the 35% (32 out of 90), α^{++} in the 15% (14 out of 90), and ILP in the 15% (14 out of 90).

We have compared the results of the algorithms by means of non-parametric statistical tests. We first applied the Friedman test [45] that computes the ranking of the results of the algorithms, and rejects the null hypothesis —which states that the results of the algorithms are equivalent— with a given confidence or significance level (α). Then we applied the Holm's post-hoc test [62] for detecting significant differences among the results. However, as we are using 7 different metrics, the comparison must be done in a multi-objective way. In order to perform a fair comparison, we have used the criterion of Pareto dominance. We have applied the *fast-non-dominated-sort* [35] in order to rank the solutions of the algorithms for each log. With this method, a mined model a dominates other mined model b, i.e., $a \succ b$, if the model a is not worse than the model b in all the objectives —the 7 metrics— and better in at least one

Table 2.6: Results on the balanced logs with a 20% of noise.

			Logs with 20% of noise																	
			Caminatas	A8	D2	MIIISkip	Ma5	MI2I	MDriverLL	allLoops	Ma7	I2la	MexampleL	Herbst6p37	Ma6nfc	MParallel5	NC	L2LP	NCB	DWS
ProDiGen	Model metrics	B_p	0.57	1.0	0.91	1.0	1.0	1.0	0.62	1.0	0.78	0.8	0.75	1.0	0.75	0.92	0.72	0.76	0.8	0.5
		B_r	0.99	1.0	1.0	1.0	1.0	1.0	0.99	1.0	0.99	0.99	0.99	1.0	0.99	0.99	0.99	0.99	0.97	0.99
		S_p	0.75	1.0	0.87	1.0	1.0	1.0	0.77	1.0	0.79	0.81	0.76	1.0	0.76	0.78	0.76	0.78	0.8	0.47
		S_r	0.85	1.0	0.87	1.0	1.0	1.0	0.89	1.0	1.0	1.0	0.83	1.0	0.83	0.84	0.75	0.85	1.0	0.95
	Log metrics	C	0.34	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.52	1.0	0.69	0.72	0.62	1.0	1.0	0.37
		P	0,63	1.0	0,95	1.0	1.0	0,15	1.0	1.0	1.0	1.0	0,95	1.0	0,68	0,8	0,7	0.76	0.89	0,98
		S	1.0	1.0	1.0	1.0	1.0	0,22	1.0	1.0	1.0	1.0	1.0	1.0	0,97	0,91	0,98	0.95	0.91	0,92
GM	Model metrics	B_p	0.56	0.67	0.66	0.85	0.82	1.0	0.44	0.9	0.67	0.77	0.73	0.59	0.65	0.83	0.57	0.95	0.88	0.78
		B_r	0.99	0.99	0.99	0.99	0.99	0.99	0.98	0.98	0.99	0.99	0.99	0.97	0.99	0.99	0.99	0.99	0.99	0.81
		S_p	0.56	0.46	0.6	0.69	0.72	0.88	0.52	0.88	0.47	0.69	0.64	0.44	0.61	0.52	0.5	0.9	0.8	0.6
		S_r	0.6	0.75	0.75	0.69	0.66	1.0	0.52	0.95	0.66	0.81	0.91	0.65	0.66	0.66	0.66	0.9	0.88	0.95
	Log metrics	C	0.0	1.0	1.0	1.0	1.0	1.0	0.0	1.0	0.0	1.0	0.0	0.0	0.3	1.0	0.16	1.0	1.0	0.54
		P	0,4	0,42	0,45	0,84	0,45	0,8	0,11	0,61	0,42	0,47	0,43	0,05	0,38	0,6	0,37	0,76	0,65	0,76
		S	1.0	0,64	0,91	0,77	0,75	0,85	1.0	0,58	0,81	0,76	0,65	0,63	0,68	0,88	0,84	0,68	0,88	0,84
HM	Model metrics	B_p	0.97	0.7	0.95	0.9	0.9	0.67	0.9	0.93	1.0	0.94	1.0	1.0	0.89	1.0	0.75	0.77	0.88	0.7
		B_r	1.0	0.85	1.0	0.8	0.89	0.91	0.92	0.89	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.95	1.0	0.6
		S_p	1.0	0.77	1.0	1.0	1.0	0.6	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.8	1.0	0.76
		S_r	0.95	0.87	0.87	0.76	0.83	0.75	0.9	0.83	1.0	0.9	1.0	1.0	0.91	1.0	0.83	0.8	0.88	0.95
	Log metrics	C	0.34	0.0	1.0	0.0	0.65	0.0	0.63	0.58	1.0	0.3	1.0	1.0	0.69	1.0	0.0	0.0	0.0	0.0
		P	0,9	0.0	0,95	0,19	0,94	0,15	0,11	1.0	1.0	0,93	1.0	1.0	0,9	1.0	0.0	0,45	0,14	0,23
		S	1.0	0,9	1.0	0,75	1.0	0,81	1.0	0,93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,96	1.0	0,93
α^{++}	Model metrics	B_p	0.83	0.88	0.84	0.39	0.61	0.4	0.29	0.73	0.76	0.42	0.4	0.29	0.44	0.75	0.53	0.62	0.29	0.26
		B_r	0.91	0.99	0.99	0.55	0.56	0.49	0.4	0.82	0.87	0.57	0.75	0.42	0.76	0.99	0.89	0.54	0.4	0.24
		S_p	0.59	0.46	0.62	0.23	0.26	0.26	0.18	0.2	0.27	0.25	0.17	0.21	0.23	0.22	0.26	0.15	0.27	0.29
		S_r	0.67	0.75	0.62	0.23	0.5	0.5	0.28	0.16	0.41	0.27	0.33	0.3	0.41	0.46	0.33	0.2	0.33	0.42
	Log metrics	C	0.0	0.49	1.0	0.0	0.0	1.0	0.0	0.24	1.0	0.0	0.52	1.0	0.0	1.0	0.83	0.0	0.0	0.0
		P	0,01	0,72	0,84	0,19	0,09	0,69	0,11	0,75	0,65	0,05	0,64	1.0	0.0	0,76	0.0	0,45	0,14	0,23
		S	0,5	0,58	1.0	0,22	0,29	0,88	0,11	0,86	0,63	0,24	0,41	1.0	0,4	0,42	0,62	0,27	0,36	0,22
ILP	Model metrics	B_p	0.1	0.09	0.27	0.35	0.31	0.11	0.24	0.29	0.17	0.1	0.2	0.05	0.07	0.2	0.17	0.35	0.41	0.14
		B_r	0.19	0.23	0.83	0.46	0.57	0.27	0.19	0.63	0.34	0.33	0.31	0.06	0.31	0.35	0.49	0.75	0.84	0.21
		S_p	0.23	0.23	0.45	0.28	0.21	0.17	0.19	0.26	0.2	0.22	0.14	0.05	0.14	0.1	0.19	0.29	0.25	0.12
		S_r	1.0	0.64	0.8	1.0	1.0	1.0	0.93	0.7	0.97	1.0	1.0	1.0	0.77	0.76	1.0	1.0	1.0	0.96
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0,32	0,4	0.0	0,49	0,37	0,35	0,27	0,59	0,38	0,32	0,35	0,18	0,28	0,27	0,31	0,71	0,36	0,45
		S	0,11	0,2	0,77	0,22	0,18	0,16	0,11	0,5	0,23	0,18	0,16	0,05	0,18	0,14	0,27	0,33	0,26	0,1

objective. Thereby, for each log, all the solutions in the first non-dominated front will have a rank equal to 1 —these are the solutions more similar to the original model and, therefore, the best ones—, the solutions in the second non-dominated front will have a rank equal to 2, and the process continues until all fronts are identified[7]. Therefore, to perform the non-parametric statistical tests, we first ranked all the solutions for each log based on the Pareto

[7]Note that there can be as many fronts as possible solutions. Hence, as we are comparing five algorithms, there can be a maximum of 5 possible fronts, being the solutions in the first front the best ones.

Table 2.7: Non-parametric test for the balanced logs.

(a) Friedman ranking.

Algorithm	Ranking
ProDiGen	1.63
HM	2.52
GM	3.27
ILP	3.5
α^{++}	4.1
Friedman p-value: 5.34E-11	

(b) Holm post-hoc, $\alpha = 0.05$.

i	Comp.	z	p	α/i	Hypothesis
4	α^{++}	10.4	9.73E-26	0.013	**Rejected**
3	ILP	7.9	2.87E-15	0.017	**Rejected**
2	GM	6.95	3.57E-12	0.025	**Rejected**
1	HM	3.77	1.62E-4	0.05	**Rejected**

dominance and then we used these ranks as input for the Friedman test. Table 2.7 summarizes the results of the tests. As can be seen, ProDiGen has the best ranking, whereas Heuristics Miner gets the second position, as it retrieves very competitive results when dealing with noisy logs, but it still has problems with non-free-choice constructs and some short loops. On the other hand, Genetic Miner has the third position, closely followed by ILP miner. In general Genetic Miner retrieves better solutions than ILP, as ILP cannot handle invisible tasks and infrequent behavior. Finally, α^{++}-algorithm has the worst ranking. This is due the inability of this algorithm to tackle invisible tasks and handle infrequent behavior in the log, giving as a result incomplete and very underfited models. The p-value of the Friedman test (Table 2.7(a)) is really low, indicating a high level of confidence. Furthermore, based on the results of the Friedman test, we performed a Holm's post-hoc test (Table 2.7(b)), starting with the initial hypothesis that all the tested algorithms are equal to ProDiGen. The test rejects the null hypothesis in all the cases for a confidence level of $\alpha = 0,05$ —the *p-value* of each algorithm has to be lower than α/i in order to reject the hypothesis. This means that ProDiGen outperforms all the other algorithms, and that the difference is statistically significant for that confidence level. We have repeated this test considering only the model metrics, and also taking into account only the log metrics —the results of the tests do not change.

2.6.5 Results on unbalanced logs

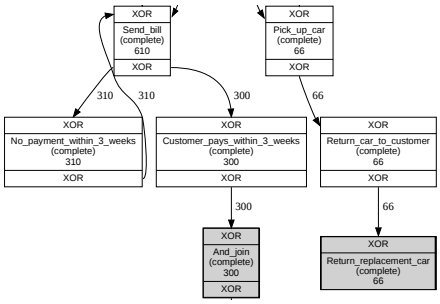
Table 2.8 shows the results on the 21 unbalanced logs. ProDiGen mines the original model in 17 of the logs —the values of the four model metrics are 1—, while in the other 4 logs the mined model is very similar to the original one. The difficulties in these 4 logs arise when *i)* mining logs with parallel constructs with more than two branches and with two or

Table 2.8: Results on the unbalanced logs.

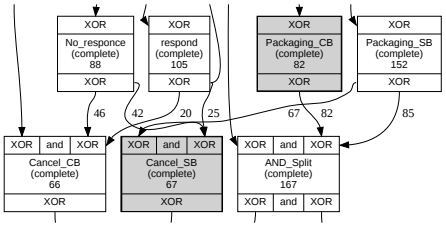
			Logs																				
			g2	g3	g4	g5	g6	g7	g8	g9	g10	g12	g13	g14	g15	g19	g20	g21	g22	g23	g24	g25	EMT
ProDiGen	Model metrics	B_p	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.96	1.0
		B_r	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	1.0
		S_p	1.0	1.0	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.91	1.0
		S_r	1.0	1.0	0.97	1.0	1.0	1.0	0.94	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.98	0.91	1.0
	Log metrics	C	1.0	1.0	0.78	1.0	1.0	1.0	0.52	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.98	1.0
		P	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		S	1.0	1.0	1.0	1.0	1.0	1.0	0,93	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,94	1.0
GM	Model metrics	B_p	1.0	0.61	0.78	1.0	1.0	1.0	0.84	0.96	0.99	1.0	0.98	0.61	0.8	0.98	1.0	1.0	0.97	0.57	0.83	0.81	1.0
		B_r	1.0	0.97	0.97	1.0	1.0	1.0	1.0	1.0	0.97	1.0	0.99	1.0	0.97	0.9	1.0	1.0	1.0	0.88	0.88	0.96	0.83
		S_p	1.0	0.81	0.81	1.0	1.0	1.0	1.0	0.97	0.9	1.0	0.95	0.95	0.88	0.95	1.0	1.0	0.85	0.76	0.75	0.76	0.85
		S_r	1.0	0.81	0.81	1.0	1.0	1.0	0.94	0.98	0.92	1.0	0.94	0.94	0.87	0.89	1.0	1.0	0.85	0.74	0.75	0.74	0.85
	Log metrics	C	1.0	0.31	0.59	1.0	1.0	1.0	0.26	0.48	0.48	1.0	0.75	1.0	0.15	0.2	1.0	1.0	0.43	0.2	0.72	0.41	0.3
		P	1.0	0,6	1.0	1.0	1.0	1.0	0,15	1.0	1.0	1.0	1.0	0,81	0,14	0,08	1.0	1.0	0,96	0,42	0,99	0,75	0,94
		S	1.0	1.0	0,97	1.0	1.0	1.0	0,72	0,96	0,88	1.0	0,95	0,99	0,88	0,95	1.0	1.0	1.0	0,88	1.0	0,82	1.0
HM	Model metrics	B_p	1.0	1.0	0.94	1.0	0.9	0.97	0.87	1.0	0.96	1.0	1.0	0.97	0.96	0.97	1.0	1.0	0.99	0.6	0.92	0.76	0.81
		B_r	1.0	0.98	0.92	1.0	0.98	0.97	0.99	0.98	0.95	1.0	1.0	0.97	0.98	1.0	1.0	1.0	0.99	1.0	0.88	0.94	0.96
		S_p	1.0	0.97	0.96	1.0	0.93	0.97	0.95	1.0	0.96	1.0	1.0	0.96	1.0	1.0	1.0	1.0	0.97	0.91	0.89	0.85	0.76
		S_r	1.0	0.97	0.86	1.0	0.97	1.0	0.86	1.0	0.96	1.0	1.0	0.92	0.86	0.9	1.0	1.0	0.91	0.94	0.81	0.85	0.74
	Log metrics	C	1.0	1.0	0.78	1.0	0.66	1.0	0.52	0.74	0.78	1.0	1.0	0.91	0.87	0.85	1.0	1.0	0.9	0.0	0.93	0.23	0.37
		P	1.0	1.0	1.0	1.0	0,99	1.0	1.0	1.0	1.0	1.0	1.0	0,99	1.0	1.0	1.0	1.0	1.0	0,42	0,97	0,97	0,98
		S	1.0	1.0	1.0	1.0	0,99	0,98	0,93	0,96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0,93	1.0	0,94	1.0
α^{++}	Model metrics	B_p	0.8	0.81	0.95	1.0	0.88	0.92	0.87	0.96	0.94	0.95	0.93	0.58	0.85	0.9	0.64	0.84	0.93	0.71	0.81	0.94	0.99
		B_r	0.85	0.91	0.94	1.0	0.99	0.9	0.94	0.94	0.9	1.0	0.94	0.92	0.85	0.94	0.94	0.89	0.85	0.67	0.7	0.92	0.93
		S_p	0.89	0.94	0.98	1.0	0.79	0.98	0.83	0.98	0.81	0.96	0.88	0.86	0.9	0.9	0.92	0.93	0.82	0.66	0.85	0.91	1.0
		S_r	1.0	0.98	0.97	1.0	1.0	1.0	0.94	1.0	1.0	1.0	0.97	1.0	0.97	1.0	0.87	1.0	0.97	1.0	1.0	0.98	0.91
	Log metrics	C	0.33	0.0	1.0	1.0	0.45	0.0	0.35	0.48	0.563	1.0	0.48	0.0	0.05	0.25	0.46	0.68	0.43	0.0	0.0	0.97	0.89
		P	0,96	0,18	0,97	1.0	1.0	0,12	1.0	1.0	1.0	0,97	1.0	0,82	0,14	0,98	0,86	0,09	1.0	0,42	0,11	0,26	1.0
		S	0,78	0,79	1.0	1.0	0,76	0,93	0,74	0,79	0,76	0,99	0,79	0,79	0,97	0,75	1.0	0,83	0,32	0,34	0,42	0,89	1.0
ILP	Model metrics	B_p	0.9	0.89	1.0	1.0	0.96	0.96	0.9	0.94	0.92	0.95	0.93	0.85	0.87	0.91	0.72	0.87	0.9	0.58	0.78	0.66	0.98
		B_r	1.0	0.98	0.99	1.0	0.99	0.99	0.99	0.99	0.97	1.0	0.99	0.99	1.0	0.99	0.95	1.0	0.99	0.88	0.94	0.96	1.0
		S_p	0.83	0.85	0.98	1.0	0.78	0.94	0.76	0.89	0.73	0.96	0.78	0.78	0.67	0.85	0.85	0.92	0.72	0.5	0.77	0.64	0.91
		S_r	1.0	0.98	0.97	1.0	1.0	1.0	0.96	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.97	1.0	1.0	0.98	1.0
	Log metrics	C	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0,97	0,97	1.0	1.0	0,99	1.0	0,98	0,98	0,95	0,97	0,07	0,95	0,96	0,98	0,96	0,96	0,93	0,83	0,89	0,8	1.0
		S	0,93	0,92	0,96	1.0	0,74	0,93	0,66	0,9	0,68	0,99	0,97	0,69	0,76	0,79	0,79	0,97	0,52	0,28	0,7	0,59	0,93

more tasks in each branch, and *ii)* when mining logs that came from models with unbalanced AND-join/split points. These type of patterns are even more difficult to mine considering that not all the possible combinations admitted by the original model are represented in the log, and not all the traces have the same frequency. Therefore, ProDiGen tries to better fit the most frequent behavior of the log, overfitting the data.

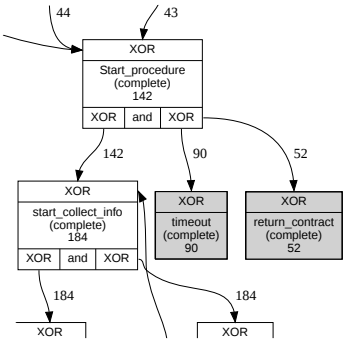
- The mined model for log $g8$ (Figure 2.6(c)) has a behavioral precision and recall equal to 1. However, the model is not complete because it cannot tackle the output dependencies of the tasks *timeout* and *return-contract*, considering them as final tasks. This results in a incomplete mined model, because all the traces involving these two tasks will have an extra token at the end of the parsing.



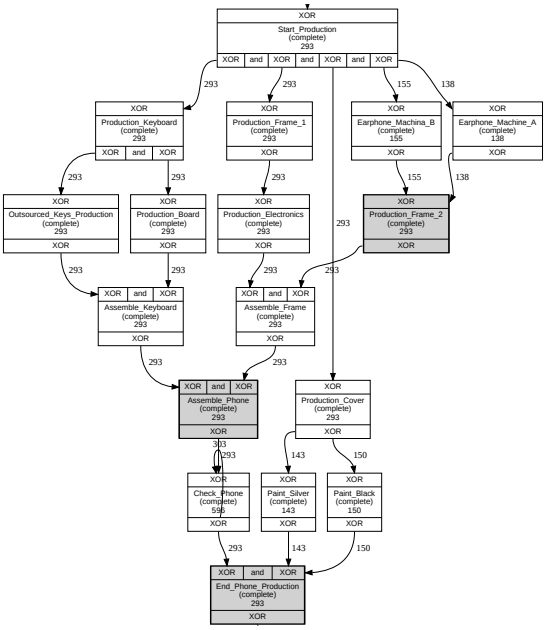
(a) Detail of the mined model for the log *g4*.The tasks highlighted in grey are involved in a unbalanced AND-join/split point, making very difficult to correctly mine their inputs and outputs.



(b) Detail of the mined model for the log *g24*. In the original model, the missing relation is never used, therefore it is impossible to mine it.



(c) Detail of the mined model for the log *g8*. Both tasks hihglighted in grey are involved in an unbalanced AND-join/split.



(d) Heuristic net of the mined model for the log g25. The mined model incorrectly finds the AND-join point at the task *"End_phone_product"*. This leads to other incorrect relations trying to better fit the log.

Figure 2.6: Heuristics nets of the mined models for the unbalanced logs: *g4*, *g8*, *g24* and *g25*.

- The results for log $g24$ (Figure 2.6(b)) shows that the mined model is almost equal to the original one, except in only one relation between two tasks (tasks in grey, Figure 2.6(b)).
- The mined model for log $g4$ (Figure 2.6(a)) has a behavioral precision and recall of 1, i.e., the mined model allows the same behavior as the original one w.r.t the information contained in the log. The difference between the mined and original models is that the mined model cannot find —as with log $g8$— the output dependencies of the task *return_replacement_car*, therefore it considers that task as final, generating an extra final token every time the model parses a trace involving this task.
- For log $g25$ (Figure 2.6(d)) the behavioral recall and precision are closer to 1. This means that, even when the model is not as precise as the original, it does not allow for too much extra behavior than the original one. The mined model cannot mine the original AND-join point.

Analyzing the results of the other algorithms, HM focuses its search on the main behavior of the log —finding solutions with high levels of simplicity. Hence, it cannot find the original model on those logs that came from models with many interleaving situations, as it tries to better fit the most frequent behavior recorded in the log —as the logs are unbalanced, *not all the possible relations* have the same frequency. On the other hand, ILP tends to retrieve complete models, but they usually are very general and, therefore, very different from the original model. Additionally, although ILP retrieves complete models when possible, it cannot tackle invisible tasks, giving poor results with the models containing this kind of constructs. With respect to Genetic Miner, based on its fitness definition —always benefits the individuals that portrait the most frequent behavior in the log—, it has problems to obtain complete and precise models when dealing with logs with many interleaving situations. This results in solutions with poorly precision values and very complex —with many silent tasks. Finally, the α^{++}-algorithm gets the worse results since it cannot deal with logs with many interleaving situations, giving as a result very poor values of completeness and precision for almost all the logs. Comparing the results of the five algorithms: ProDiGen correctly mines, i.e., finds the original model, the 81% (17 out of 21) of the cases. For the other algorithms, the percentage of correctly mined models was: GM in the 33% (7 out of 21), HM in the 28% (6 out of 21), and both α^{++} and ILP in the 5% (1 out of 21). Tables 2.9(a) and 2.9(b) show the results for the non-parametric tests for the unbalanced logs. Again, the p-value of the Friedman test is really low, indicating that there are significant differences among the algorithms with a

Table 2.9: Non-parametric test for the unbalanced logs.

(a) Friedman ranking.

Algorithm	Ranking
ProDiGen	1.55
HM	3.17
GM	3.31
ILP	3.43
α^{++}	3.55
Friedman p-value: 3.58E-7	

(b) Holm post-hoc, $\alpha = 0.05$.

i	Comp.	z	p	α/i	Hypothesis
4	α^{++}	4.1	4.15E-5	0.013	**Rejected**
3	ILP	3.85	1.15E-4	0.017	**Rejected**
2	GM	3.61	3.05E-4	0.025	**Rejected**
1	HM	3.31	9.06E-4	0.05	**Rejected**

Table 2.10: Average runtimes of the algorithms on the 21 unbalanced logs.

	ProDiGen			*GM*[8]	*HM*	α^{++}	*ILP*
	Initial Sol.	*Best Sol.*	**Total**	**Total**	**Total**	**Total**	**Total**
Average Time	2.5s	4.7m	32m	65m	606ms	128ms	33.8s

high level of confidence. Holm's test also rejects the null hypothesis between ProDiGen and each of the algorithms used in the comparison. Therefore, we can conclude that ProDiGen also outperforms all the algorithms with unbalanced logs, and the difference is statistically significant. We also did the test with only the model metrics and only the log metrics —with no differences.

Table 2.10 shows the average runtimes of all the algorithms over the unbalanced logs of Section 2.6.1. As can be seen, Heuristics Miner, α^{++} and ILP have on average very fast runtimes, being the quality of the results of Heuristics Miner the highest of the three. On the other hand, ProDiGen improves the execution time of Genetic Miner. The comparison between the runtimes of ProDiGen and Heuristics Miner requires to take into account that ProDiGen is an iterative algorithm —it obtains thousands of solutions per execution—, while HM only gets one solution per execution. The *initial solution runtime* (Table 2.10) shows the time that ProDiGen needs, on average, to pre-process the log and retrieve de best solution of the initial population. The quality of that solution is at least as good as the solution of Heuristics Miner —as ProDiGen uses the Heuristics Miner solution as part of the initial population. Both the

[8]Genetic Miner is an iterative algorithm and, therefore, it would be possible to measure the runtimes of the first and best solutions and the total runtime. We only show the total runtime as the implementation of Genetic Miner (ProM 6.3) does not measure other runtimes. Nevertheless, the runtime for the initial solution is very similar to that of ProDiGen, although the quality of that solution for Genetic Miner is usually very low.

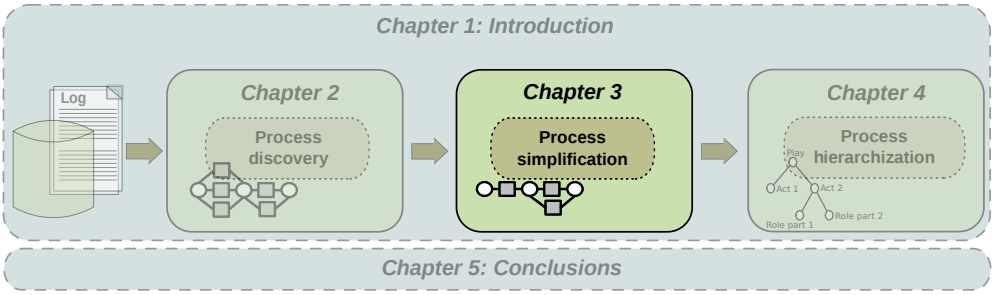
runtimes of the initial solution and HM are fast —as mining algorithms do not have real time requirements. Moreover, the extra-time inverted by ProDiGen in finding the *best solution* (Table 2.10) is worth, as ProDiGen improves the solutions of Heuristics Miner in 58 out of 111 logs —15 out of 21 unbalanced logs— in a fast way for process mining requirements.

2.7 Conclusions

We have presented ProDiGen, a genetic algorithm for process mining that can tackle all the different constructs at once, and obtains models that are complete, precise, and simple. ProDiGen uses a new hierarchical fitness function that includes new definitions for precision and simplicity. Moreover, the proposal uses genetic operators that focus the search on specific parts of the model: *i)* the crossover operator selects the crossover point based on the errors of the mined model; and *ii)* the mutation operator is guided by the causal dependencies of the log. ProDiGen has been validated with 111 different logs with all kind of workflow patterns, noise, and unbalanced logs. Also, we have compared ProDiGen with 4 of the state of the art process discovery algorithms using non-parametric statistical tests. Results conclude that using a hierarchical fitness based on completeness, precision and simplicity shows a great performance when retrieving the original model. Moreover, ProDiGen outperforms the other process mining algorithms, as it is able to retrieve the original model in the 84% of the tested logs.

CHAPTER 3

ENHANCING DISCOVERED PROCESSES WITH DUPLICATE TASKS



In Chapter 2 we presented an algorithm for the automatic discovery of process models. Here, we introduce a new algorithm to support the mining of additional behavior, in particular, duplicate labels. How to face this type of behavior is of particular interest in process discovery as, usually, duplicate events are recorded with the same label in the log, hindering the discovery of the model that better fits the recorded behavior. Taking into account duplicate activities can enhance the comprehensibility of the mined model. For instance, in the genetic algorithm presented in the previous chapter, introducing duplicate activities before or during the mining process can lead to a search space explosion. Furthermore, the incorrect identification of duplicate labels can lead to an incorrect representation of the behavior recorded in the event log. In order to keep the search space within bounds, one possible solution is to duplicate the most convenient activities *after* mining the model.

To tackle this issue, in this chapter we present SLAD (Splitting Labels After Discovery), an algorithm that takes as starting point an already mined model, and using the local information of the log, tries to improve the comprehensibility and understandability of the model by splitting the overly connected nodes into two or more activities. More specifically, the contributions presented in this chapter are: i) the discovering of the duplicate activities is performed *after* the discovery process, in order to *unfold* the overly connected nodes than may introduce extra behavior not recorded in the log; ii) new heuristics to focus the search of the duplicated tasks on those activities that better improve the model; and iii) new heuristics to detect potential duplicate activities involved in loops. This proposal has been validated with 54 different mined models from three process discovery algorithms. Furthermore, the results have been compared with eight different algorithms from the state of the art.

This algorithm, as well as the aforementioned comparison, is described in the following publication:

B. Vázquez-Barreiros[1], M. Mucientes[1], and M. Lama[1]. Enhancing Discovered Processes with Duplicate Tasks. *Information Sciences*, 373:369–387, 2016.
(DOI: 10.1016/j.ins.2016.09.008)).

3.1 Abstract

Including duplicate tasks in the mining process is a challenge that hinders the process discovery, as it is also necessary to find out which events of the log belong to which transitions. To face this problem, we propose SLAD (Splitting Labels After Discovery), an algorithm that uses the local information of the log to enhance an already mined model, by performing a local search over the tasks that have more probability to be duplicated in the log. This proposal has been validated with 54 different mined models from three process discovery algorithms, improving the final solution in 45 of the cases. Furthermore, SLAD has been tested in a real scenario.

[1]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela. Santiago de Compostela, Spain.

3.2 Introduction

In the recent years, a lot of work has been made for developing technologies to automate the execution of processes in different application domains such as industry, education or medicine [80]. In particular, for business processes, there has been an incredible growth on the amount of process-related data, i.e, execution traces of business activities. Within this context, process mining has emerged as a way to analyze the behavior of an organization based on these data —event logs— offering techniques to discover, monitor and enhance real processes, i.e., to understand *what is really happening in a business process* [122].

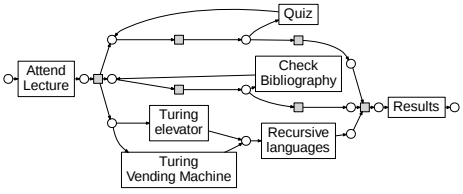
Based on this idea, and in order to model what is really happening in an organization, *process discovery* techniques aim to find the process model that *better* portraits the behavior recorded in an event log. There are four quality dimensions to measure *how good* is a model and, hence, identify which model is the *best*: fitness replay, precision, generalization and simplicity. *Fitness replay* measures how much of the behavior recorded in the log can be reproduced in the process model. On the other hand, *precision* and *generalization* measure if the model overfits —it disallows for new behavior not recorded in the log— or underfits —it allows additional behavior not recorded in the log— the data, respectively. Finally, *simplicity*, quantifies the complexity of the model, for instance, the number of arcs and tasks. Hence, the idea behind process discovery is to maximize these metrics in order to obtain an *optimal* solution that better describes the flow of the events that occur within the process.

In order to discover the optimal solution, one key question is *how to describe the ordering and flow of events that occur in a process* [134]. From the control-flow perspective, a model can be represented with many different workflow patterns, such as *sequences*, *parallels*, *loops*, *choices*, etc. Furthermore, to improve the quality of the solution, models can be extended with more behavior: *duplicate activities*, *non-free-choice constructs*[2], etc. Within the scope of this paper, the notion of duplicate tasks —or activities— [136] refers to situations in which multiple tasks in the process have the same label, i.e., they can appear more than once in the process. As previously said, the inclusion of duplicate tasks is useful to improve the precision and simplicity of a model, and, hence, enhance its comprehensibility [15, 31]. Figure 3.1 shows an example on how the addition of duplicate tasks to a model improves its understandability and structural clarity. In this example, considering the sample log of Fig. 3.1(a), the events *Quiz* and *Check Bibliography*, are executed multiple times —

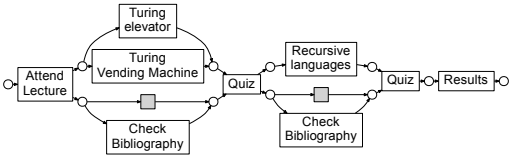
[2] A non-free choice (NFC) construct is a special kind of choice, where the selection of a task depends on what has been executed before in the process model.

	Sequence of events
$case_1$	Attend lecture, Turing Elevator, Check Bibliography, Quiz, Recursive Languages, Quiz, Results.
$case_2$	Attend lecture, Turing Elevator, Check Bibliography, Quiz, Check Bibliography, Recursive Languages, Quiz, Results.
$case_3$	Attend lecture, Turing Vending Machine, Check Bibliography, Quiz, Recursive Languages, Check Bibliography, Quiz, Results.
$case_4$	Attend lecture, Check Bibliography, Turing Vending Machine, Check Bibliography, Quiz, Recursive Languages, Quiz, Results.
⋮	⋮
$case_n$	...

(a) Traces extracted from a synthetic event log.



(b) Model without duplicate tasks.



(c) Model with duplicate tasks.

Figure 3.1: A log and two process models —Petri nets— exemplifying a lecture of Automata Theory and Formal Languages.

twice in each trace. Between the multiple possibilities of modeling the behavior of the log, we can assume i) an injective relation between the events in the log and the activities in the model (Fig. 3.1(b)); or ii) that multiple activities can share the same label, i.e. a model with duplicate activities (Fig. 3.1(c)). In this example, although both models perfectly reproduce all the behavior recorded in the log —both have a perfect replay fitness—, the model depicted in Fig. 3.1(b) allows to execute both *Quiz* and *Check Bibliography* as many times as we want at any time in the process, hence, this model is not a precise picture of the recorded behavior of the log —its precision is lower. On the other hand, if both activities are duplicated, the resulting model (Fig. 3.1(c)) is more suitable, i.e., more precise with respect to the recorded behavior in the log, as it does not allow, for example, to check the bibliography —*Check Bibliography*— during the exam —*Quiz*. Hence, the ability to discover these duplicate tasks *may* greatly enhance the comprehensibility of the final solution, and create a more specific process model.

From the perspective of process discovery, including duplicate tasks in the mining process is a well known challenge [134, 136] as, usually, duplicate tasks are recorded with the same

label in the log, hindering the discovery of the model that better fits the log —algorithms need to find out which events of the log belong to which tasks. To face this issue, handling duplicate tasks is usually considered as a pre-mining step, i.e., the potential duplicate tasks are identified and accordingly labeled *before* mining the log, or as part of the process discovery algorithm. Within this context, there are several techniques in the state of the art that allow to mine duplicate tasks [15, 20, 24, 30, 47, 57, 58, 75]. However, some approaches can retrieve worse solutions than without duplicate activities [15], others can duplicate any activity of the log without imposing any limit to the number of activities that may be duplicated [20, 24, 47], or they have problems when dealing with duplicate activities involved in certain constructs, such as loops [30, 75].

In this paper we analyze the possibility of tackling duplicate tasks *after* mining a process model, in particular, after mining a causal net [123] or heuristic net [155], without adversely affecting the quality of the initial solution. Hence, we present SLAD (Splitting Labels After Discovery) a novel algorithm that enhances an already discovered process model by splitting the behavior of its activities. Firstly, a process discovery technique mines a log without considering duplicate tasks, generating a causal net or a heuristic net. Then, SLAD, using the local information of the log and the retrieved model, tries to improve the quality of the model by performing a local search over the tasks that have more probability to be duplicated in the log. The contributions of this proposal are: i) the discovering of the duplicate activities is performed *after* the discovery process, in order to *unfold* the overly connected nodes than may introduce extra behavior not recorded in the log; ii) new heuristics to focus the search of the duplicated tasks on those activities that better improve the model and iii) new heuristics to detect potential duplicate activities involved in loops.

The remainder of this paper is structured as follows. Section 3.3 describes the current state of the art of process discovery algorithms dealing with duplicate tasks. Then, Section 3.4 describes in detail the SLAD algorithm to tackle duplicate tasks. Section 3.5 shows the obtained results with 54 different mined models from three process discovery algorithms as well as a comparison with other state of the art approaches. Finally, Section 3.6 points out the conclusions.

3.3 State of the art

In the state of the art of process discovery, many techniques [33, 71, 138, 147, 153, 155, 156, 158] assume an injective relation between tasks and events in the log, considering that there cannot be two different activities with the same label. Therefore, these algorithms, when dealing with logs with duplicate tasks, usually give as a result models with overly connected nodes or needless loops, decreasing the precision and simplicity of the model. On the other hand, there are techniques that do not make such a restrictive assumption [15, 20, 23, 24, 30, 47, 57, 58, 75]. Typically, all these techniques identify the potential duplicate activities in a pre-mining step, or during the mining process. One example is the α^*-algorithm [75], an extension of the α-algorithm [137] to mine duplicate tasks. However, the heuristic rules used in this algorithm require a noise-free and complete log [140]. Fodina [15] is an algorithm based on heuristics that infers the duplicate tasks transforming the event log into a *task log* following the heuristics defined in [30]. Other solutions, like DGA [30] and ETM [20] —based on evolutionary algorithms—, or AGNES [47] —an approach based on inductive programming— include the possibility to mine duplicate tasks. However, these techniques do not allow to *unfold* loops [30], or they are very permissive allowing to duplicate any activity in the log [20]. State-based region theory algorithms [23, 24] are also able to mine duplicate tasks, but, when searching for regions, they usually allow to split any label in the log without any bound. Herbst et al. also developed a set of algorithms [57, 58] that infer the duplicate activities in a pre-mining step. However, the algorithms developed by Herbst et al. only focus their search on block-structured representations of a process model, constraining the expressiveness of a process model. For example, he only way to represent a non-free-choice construct, if possible, is through duplicate labels, which can lead to a more complex model.

In summary, although very valuable results have been achieved in this field, the state of the art algorithms have different weaknesses. Some obtain, in specific logs, worse solutions than without duplicated tasks [15]. Others allow to duplicate any activity in the log [20], or generate solutions with a lower simplicity [24] —more complex solutions. Finally, other proposals use heuristics that do not consider duplicate activities in some workflow patterns such as loops [30, 75]. Within this context, we propose SLAD, an algorithm to tackle duplicate tasks *after* mining a model, with the objective of improving its quality, i.e., its replay fitness, precision and simplicity. SLAD combines the actual dependencies of a model mined by a process discovery technique, and a set of heuristics that use the information of the behavior recorded in the log, in order to enhance the precision and simplicity —and hence its

comprehensibility— of the already mined model, trying to split its overly connected tasks that are more suitable to be duplicated.

Definition 1 (Trace, Event log)**.** *Let T be a set of tasks. A trace $\sigma \in T^*$ is a sequence of tasks. Let $\mathbb{B}(A)$ denote the set of all multisets over some set A. An event log $L \in \mathbb{B}(T^*)$ is a multiset of traces.*

3.4 Splitting Labels After Discovery

Algorithm 3.1 describes SLAD[3], an algorithm to tackle duplicate tasks on an already mined causal matrix (Definition 2). Usually, when applying a process mining technique, duplicate events in the event log (Definition 1) are represented as i) overly connected nodes where all the behavior from different contexts of the model piles up, and with ii) needless loops to allow the execution of the same label multiple times. Therefore, with the presented approach, we try to reduce the density of these nodes by delegating some of their inputs and outputs to other activities with the same label. First, using heuristics, the algorithm detects which activities may be split into multiple tasks. Then, based on the local information of the event log and the causal dependencies of the input model, the algorithm splits the behavior of the original tasks among the new tasks with the same label. Finally, the original model is replaced with the new one if its quality —*how good* is the solution— is better than the previous solution. Note that we measure the quality of a process based on three criteria: *fitness replay*, *precision* and *simplicity*. Hence, the presented algorithm tries to improve the quality of an already mined solution by unfolding the overly-connected activities —through duplicate activities— and, therefore, enhancing the comprehensibility of the model.

Definition 2 (Causal matrix)**.** *A Causal matrix is a tuple (T, I, O) where:*

T is a finite set of tasks,

$I : T \to \mathbb{P}(\mathbb{P}(T))$ is the input condition function, where $\mathbb{P}(X)$ denotes the powerset of some set X. Hence, I represents a set of sets of the tasks T.

$O : T \to \mathbb{P}(\mathbb{P}(T))$ is the output condition function.

If $e \in T$ then $I(e)$ denotes the input tasks of e, i.e., a set o sets of tasks, and $O(e)$ denotes the output tasks of e.

[3] http://tec.citius.usc.es/processmining/SLAD

Algorithm 3.1: Local search Algorithm.

input: A log L

$ind_0 \leftarrow$ initial_solution(L) `// Causal matrix retrieved by a process discovery technique.`
$T \leftarrow$ finite set of tasks of L
$potentialDuplicates \leftarrow \emptyset$
foreach *activity* $t \in T$ **do**
  if $\max(\min(|t >_L t'|, |t' >_L t|), 1) > 1$ **then**
    $potentialDuplicates \leftarrow potentialDuplicates \cup \{t\}$
$ind_0 \leftarrow$ `localSearch` (ind_0, L, $potentialDuplicates$, $true$)
Function `localSearch` (ind_0, L, $potentialDuplicates$, $unfoldL2L$)
  $ind_{best} \leftarrow ind_0$
  $potentialDuplicatesL2L \leftarrow \emptyset$
  foreach *activity* $t \in potentialDuplicates$ **do**
    subsequences $\leftarrow$ Retrieve all the subsequences ($t_1 t t_2$) where $t_1 \in \mathrm{I}(t)$ and $t_2 \in \mathrm{O}(t)$ from parsing ind_0
    combinations $\leftarrow$ `calculateCombinations` ($subsequences$, t)
    foreach *combination* $c \in combinations$ **do**
      $t' \leftarrow$ activity t from ind_0
      $t.inputs = (t.inputs \setminus c.inputs) \cup c.sharedInputs$
      $t'.inputs = c.inputs$
      $t.outputs = (t.outputs \setminus c.outputs) \cup c.sharedOutputs$
      $t'.outputs = c.outputs$
      if $(I(t') \neq \emptyset$ && $O(t') \neq \emptyset$ && $I(t) \neq \emptyset$ && $O(t) \neq \emptyset)$ **then**
        Add task t' to ind_0 and update t in ind_0
        Repair ind_0
        Prune unused arcs
        Evaluate ind_0
        if $ind_0 < ind_{best}$ **then**
          $ind_0 \leftarrow ind_{best}$
        else
          $ind_{best} \leftarrow ind_0$
          $potentialDuplicatesL2L = potentialDuplicatesL2L \cup \{\bigcup O(t)\}$
            `/*` $\bigcup O(t)$ `is the union of the subsets in` $O(t)$ `*/`
      else
        $ind_0 \leftarrow ind_{best}$
  if $potentialDuplicatesL2L \neq \emptyset$ && $unfoldL2L$ **then**
    $ind_{best} \leftarrow$ `localSearch` (ind_{best}, L, $potentialDuplicatesL2L$, $unfoldL2L$)
  return ind_{best}

For the sake of the argument, we will use the example in Figure 3.2 to illustrate the behavior of the presented approach. Fig. 3.2(a) shows a log with three traces and 6 *different* tasks. On the other hand, Fig. 3.2(b) shows the initial solution —a causal matrix and its respective Petri net— mined by the process discovery algorithm ProDiGen [147] without considering duplicate tasks. Fig. 3.2(c) shows the model obtained after the execution of SLAD to the model of Fig. 3.2(b). Finally, Fig. 3.2(d) shows each of the steps —described in the next sections— involved in the process.

3.4.1 Discovering duplicate tasks

The first step of the algorithm is the discovery of the potential duplicate tasks. One naive solution to detect if a task is a potential duplicate is to set the upper bound for that task to the number of times it appears in the log. This makes the search space finite, covering all the possible solutions with duplicate tasks, i.e., *all the tasks* are identified as potential duplicates. The problem with this solution is that within this search space is also included the overly-specific *trace-model*[4]. A variant to this approach is to set the upper bound to the maximum number of times a task is repeated in a trace, instead of considering the complete log. This will reduce the search space, but at the expense of dismissing the possible duplicity of a task between traces.

Definition 3 (Follows relation)**.** *Let T be a set of tasks. Let L be an event log over T, i.e., $L \in \mathbb{B}(T^*)$. Let $t, t' \in T$:*

$t >_L t'$ iff: there is a trace $\sigma = t_1 t_2 \ldots t_n$ and $i \in \{1, \ldots, n-1\}$ such that $\sigma \in L$, $t_i = t$ and $t_{i+1} = t'$.

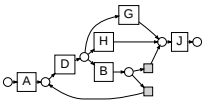
In SLAD, instead of going through a blind search over all the tasks of the input model —ind_0—, we decided to apply heuristics to identify and retrieve more information about the duplicate tasks of the event log. This strategy follows the heuristics defined in [30], where the duplicate activities can be distinguished based on their local context, reducing the search space by stating that two tasks with the same label cannot share the same input and output dependencies. Within this context, the duplicate tasks are locally identified based on the *follows relation* ($>_L$) —Definition 3 [137]. Thus the heuristics to detect potential duplicates can be formalized as described in Definition 4 [30]. In summary, this definition states that if

[4] A *trace-model* creates a path for each trace of the log. This kind of model has a perfect precision and replay fitness as it only allows the specific behavior recorded in the log, but it is not a desirable solution.

$case_1$	$case_2$	$case_3$
A D G J	A D B D H J	A D B D B J

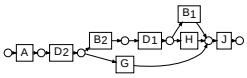
(a) Log.

t	I(t)	O(t)
A	{}	{{D}}
D	{{A,B}}	{{G,B,H}}
G	{{D}}	{{J}}
J	{{G,B,H}}	{}
B	{{D}}	{{D,J}}
H	{{D}}	{{J}}



(b) Initial solution.

t	I(t)	O(t)
A	{}	{{D_2}}
D_1	{{B_2}}	{{B_1,H}}
G	{{D_2}}	{{J}}
J	{{G,B_1,H}}	{}
B_1	{{D_1}}	{{J}}
H	{{D_1}}	{{J}}
D_2	{{A}}	{{G,B_2}}
B_2	{{D_2}}	{{D_1}}



(c) Final solution.

Step	t	Local variables	Unrepaired Solution	Repaired Solution
1		*potentialDuplicates* = [D]		
2	D	**Subsequences from parsing** ind_0: ADG, ADB, BDB and BDH		
3	D	**Combinations**: $c[0].inputs = [A]$ $c[0].outputs = [G,B]$ $c[0].sharedOutputs = [B]$; $c[0].sharedInputs = []$; $c[1].inputs = [B]$ $c[1].outputs = [B,H]$ $c[1].sharedOutputs = [B]$; $c[1].sharedInputs = []$;		
4	D	**Duplicate D into D_1 and D_2 using the previous combinations and repair the causal dependencies**	→ t: A, D_1, G, J, B, H, D_2 / I(t): {}, {{B}}, {{D_1}}, {{G,B,H}}, {{D_1}}, {{D_1}}, {{A}} / O(t): {{D_1}}, {{B,H}}, {{J}}, {}, {{D_1,J}}, {{J}}, {{G,B}}	→ t: A, D_1, G, J, B, H, D_2 / I(t): {}, {{B}}, {{D_2}}, {{G,B,H}}, {{D_1,D_2}}, {{D_1}}, {{A}} / O(t): {{D_2}}, {{B,H}}, {{J}}, {}, {{D_1,J}}, {{J}}, {{G,B}}
5		**After duplicating** D: *potentialDuplicates* = [] *potentialDuplicatesL2L* = [B,G,H]		
6	B	**Subsequences from parsing** ind_0: D_2BD_1 and D_1BJ		
7	B	**Combinations**: $c[0].inputs = [D_2]$ $c[0].outputs = [D_1]$ $c[0].sharedInputs = []$ $c[0].sharedOutputs = []$ $c[1].inputs = [D_1]$ $c[1].outputs = [J]$ $c[1].sharedInputs = []$ $c[1].sharedOutputs = []$		
8	B	**Duplicate B into B_1 and B_2 using the previous combinations and repair the causal dependencies**	→ t: A, D_1, G, J, B_1, H, D_2, B_2 / I(t): {}, {{B_1}}, {{D_2}}, {{G,B_1,H}}, {{D_1}}, {{D_1}}, {{A}}, {{D_2}} / O(t): {{D_2}}, {{B_1,H}}, {{J}}, {}, {{J}}, {{J}}, {{G,B_1}}, {{D_1}}	→ t: A, D_1, G, J, B_1, H, D_2, B_2 / I(t): {}, {{B_2}}, {{D_2}}, {{G,B_1,H}}, {{D_1}}, {{D_1}}, {{A}}, {{D_2}} / O(t): {{D_2}}, {{B_1,H}}, {{J}}, {}, {{J}}, {{J}}, {{G,B_2}}, {{D_1}}

(d) Steps of SLAD.

Figure 3.2: An example on how SLAD works.

for a task t the upper bound is greater than 1, then t is considered as a potential task for being duplicated and, hence, it is added to *potentialDuplicates* (Alg.3.1:4-6).

Definition 4 (Duplicate task)**.** *Let L be an event log over T. Let* $t,t' \in T$, $|t >_L t'|$ *the total number of times that task* t' *follows t, and* $|t' >_L t|$ *the total number of times that task* t' *precedes t. A task t is considered as a duplicate task iff:*

$max(min(|t >_L t'|, |t' >_L t|), 1) > 1$.

We use this strategy as a first step to detect the potential duplicates, and as a way to retrieve the local context of the activities. Step 1 in Fig. 3.2(d) shows the potential duplicate tasks detected after applying the described heuristic over the log of Fig. 3.2(a). In this case, only D is detected as a potential duplicate task: task D is *preceded* by tasks A and B and directly followed by B, G and H. Hence, as $max(min(3,2),1) > 1$, activity D is a potential candidate for label splitting. In the next steps of the algorithm, we extend this heuristic in order to get more information about the duplicate activities and perform the adequate split of the tasks (Section 3.4.2). Moreover, we improve this heuristic to detect potential duplicate activities in loops (Section 3.4.3).

One particular scenario of this initial approach to detect the potential duplicate tasks is related to the *start* and *end* activities of a trace. For instance, if we consider the trace $\sigma = \{A,B,C,A\}$, the activity A is never going to be selected as a potential duplicate task through this method, as A is only followed by B and preceded by C, i.e., $max(min(1,1),1) = 1$. To overcome this situation, one possible solution is to add a *dummy* start and end activity to each trace (Definition 5). Thus, by changing σ to $\sigma' = \{start,A,B,C,A,end\}$, it is possible to add the activity A into *potentialDuplicates*, as $max(min(2,2),1) > 1$.

Definition 5 (Dummy tasks)**.** *Let L be an event log over T. Let 'start' be a dummy task with no predecessors, i.e.* $|t' >_L start| = 0$*, and 'end' be a dummy task with no successors, i.e.,* $|end >_L t'| = 0$*. Then* L^+ *is an event log over* T^+ *with the dummy activities such as:*

$T^+ = T \cup \{start, end\}$,

$L^+ = \{start\ \sigma\ end \mid \sigma \in L\}$.

3.4.2 Extending the model with duplicate tasks

Once all the potential duplicates are identified, the algorithm splits the input and output dependencies of these activities of the model into multiple tasks with the same label through

the function *localSearch* (Alg. 3.1:8). Within this function, the algorithm calculates the input and output combinations for each activity in *potentialDuplicates* (Alg. 3.1:11-13), in order to split their behavior among new tasks. To compute these combinations, the algorithm firstly finds all the subsequences —Definition 6— (Alg. 3.1:12) in the log L that match the pattern $t_1 t t_2$ —window of size 1— where $t_1 \in \mathrm{I}(t)$ and $t_2 \in \mathrm{O}(t)$ in the model —being $\mathrm{I}(t)$ and $\mathrm{O}(t)$ the inputs and outputs of task t, respectively. We add this input/output constraint to focus only on those patterns that can be reproduced by the initial model. Following with the example shown in Figure 3.2, when iterating over *potentialDuplicates*, the algorithm must find the subsequences that match the pattern $t_1 \mathrm{D} t_2$ in the traces of the log L. The resultant subsequences are shown in the Step 2 in Fig. 3.2(d). Note that all these subsequences satisfy the input and output dependencies of the task D in the initial model of Fig. 3.2(b).

Definition 6 (Subsequences)**.** *Let CM be a causal matrix* (T, I, O)*. Let L be an event log. Let* $\sigma = t_1 t_2 \ldots t_n$ *and* $i \in \{1, \ldots, n-1\}$ *be a trace such that* $\sigma \in L$*. The subsequences of a task* $t \in T$ *are defined by:*

$$S = \{(t_{i-1} t_i t_{i+1}) \mid t_{i-1}, t_i, t_{i+1} \in \sigma \wedge t_{i-1} >_L t \wedge t >_L t_{i+1} \wedge t_{i-1} \in I(t) \wedge t_{i+1} \in O(t)\}.$$

Algorithm 3.2: Compute the combinations of a task.

```
1  Function calculateCombinations(subsequences, t)
2      combinations ← ∅
3      forall ((t1 t t2) ∈ subsequences) do
4          c ← ∅
5          Create a set c.inputs with the subsequences that share the same t1 and add in c.outputs
             their respective t2
6          Add c to combinations
7      foreach c ∈ combinations do
8          if c.outputs = c'.outputs where c' ∈ combinations then
9              c.inputs = c.inputs ∪ c'.inputs and c.outputs = c.outputs ∪ c'.outputs
10             combinations = combinations \ c'
11         if c.outputs shares an element e with another c'.outputs then
12             c.sharedOutputs ← c.sharedOutputs ∪ {e}
13         if c.inputs shares an element e with another c'.inputs then
14             c.sharedInputs ← c.sharedInputs ∪ {e}
15     return combinations
```

After obtaining such subsequences, the algorithm creates the combinations —through the

function *calculateCombinations* (Alg.3.2)— which will serve as the basis to split the original task into multiple activities. These combinations —Definition 7— represent the *context* in which the original task is involved —in terms of relations— with the other tasks of the log. *calculateCombinations* builds such combinations of a task t following three rules (Alg.3.2:7-14). First, given two subsequences $t_1 t t_2$ and $t_3 t t_4$, if $t_1 = t_3$, then SLAD merges both subsequences into a new combination (Alg.3.2:3-6). Second, and after identifying all the combinations, given two different combinations c and c', if they share the same $outputs$, i.e., $c.outputs = c'.outputs$, these two combinations are merged (Alg.3.2:8-10). With these two rules SLAD can split the behavior of the task into different contexts combining the mined causal dependencies and the local information of the log. Finally, if the intersection between two combinations is not the empty set, SLAD records which inputs and outputs are shared by both combinations (Alg.3.2:11-14) —this information is later used in the repairing step. Step 3 in Fig. 3.2(d) shows the combinations generated from the subsequences in which task D is involved. For instance, one of the combinations is generated based on the subsequeces ADG and ADB; and the other one is based on the subsequences BDB and BDH. In this case, task B is shared in the $outputs$ of both combinations, therefore $c[0].sharedOutputs = [B]$ and $c[1].sharedOutputs = [B]$; in the same way, none of the combinations share an element in their inputs, therefore both $sharedInputs$ subsets are empty.

Definition 7 (Combinations)**.** *Let S be a set of subsequences. We define a combination c as a group of subsequences such as:*

$C = \{c \in \mathbb{P}(S) \mid \forall (xyz), (ijk) \in S : x = i \wedge y = j\}$.

Let n be the total number of combinations $c \in C$ and m the total number of subsequences $s \in c$. Hence, for a task t, each $c \in C$ represents a set of k grouped subsequences $(t' t t'')$ where:

$c.inputs = \{\bigcup_{i=1}^{m} t' \mid t' >_L t \in s_m\}$

$c.outputs = \{\bigcup_{i=1}^{m} t'' \mid t >_L t'' \in s_m\}$

$c.sharedInputs = \{\bigcap_{i=1}^{n} c_i.inputs \mid c_i \in C\}$

$c.sharedOutputs = \{\bigcap_{i=1}^{n} c_i.outputs \mid c_i \in C\}$

After building all the possible combinations, the next steps in Alg. 3.1 are straightforward. For each combination c (Alg.3.1:14), SLAD creates a new task t' equal to the original task t of the current model (Alg.3.1:15). Then, it removes from $Input(t)$ all the tasks shared with $c.inputs$, but keeping the tasks that are in $c.sharedInputs$ (Alg.3.1:16). On the other hand, for the new task t', it retains only the elements in $Input(t')$ that are contained in $c.inputs$

(Alg.3.1:17). The same process is applied for the outputs of both t and t' but with *c.outputs* and *c.sharedOutputs* (Alg.3.1:18-3.1:19). With this process the algorithm redistributes the inputs and outputs of the original task among the new task. If these two tasks are compliant with the model, i.e. both their inputs and outputs are not empty[5] (Alg.3.1:20), the new task is included in ind_0 —the actual model—, and the original task of ind_0 is updated (Alg. 3.1:21). Otherwise the model goes back to its previous state and tries a new combination. Following with the example, SLAD, using the combinations previously calculated (Step 3 of Fig. 3.2(d)), splits the inputs and outputs of the original task D among two new tasks D_1 and D_2 —for the sake of the argument we label the duplicate tasks with different subscripts. This process gives as a result the unrepaired model shown in Step 4 of Fig. 3.2(d).

Definition 8 (Repairing process)**.** *Being t the original task, t' a new task split from t, a Causal Matrix (T,I,O) is repaired as follows:*

$$\forall t'' \in O(t') \rightarrow I(t'') = I(t'' : t \rightarrow t')$$
$$\forall t'' \in I(t') \rightarrow O(t'') = O(t'' : t \rightarrow t')$$

When including this new task, the model has to be repaired (Alg.3.1:22), as some of the dependencies of the original task t are now in a new task t' with the same label, resulting in an inconsistent model, e.g., the activity D_2 has the activity B as output, but activity B has no activity D_2 as input, i.e., it still has the activity D_1. The repairing process works as stated in Definition 8. In summary, being t the original task and t' the new task, for each task t" that was eliminated from O(t), the process checks if $t \in$ I(t''). If that is true, t has to be replaced *in each subset* of I(t'') with t'. This process is repeated also for the input sets. Then, the algorithm performs a post-pruning of the model removing the unused arcs, i.e., arcs whose frequency of use is zero (Alg.3.1:23). In order to evaluate the models (Alg.3.1:24), we based their quality on three criteria: *fitness replay*, *precision* and *simplicity*. To measure these criteria we used the hierarchical metric defined in [147], that first compares the fitness replay, then the precision and last the simplicity of the process models:

Definition 9 (Process models dominance)**.** *Let x, x' be two process models. Let $F(x)$, $P(x)$ and $S(x)$ be, respectively, the replay fitness, precision and simplicity of the process model x.*

[5]Based on Definition 5, only the dummy activities *start* and *end* should have an empty input and output set, respectively, as they are the only initial and final activities of the event log L.

The process model x is better than the process model x′ iff:

$$\begin{aligned} x \succeq x' \iff \quad & [F(x) > F(x')] \\ & \vee [F(x) = F(x') \wedge P(x) > P(x')] \\ & \vee [F(x) = F(x') \wedge P(x) = P(x') \wedge S(x) > S(x')] \end{aligned}$$

Thus, replay fitness is the primary ordering criteria. When two solutions have the same replay fitness, precision is used to decide the best solution. When two solutions have the same replay fitness and precision, simplicity is the decisive criterion. Finally, if the new model with duplicate tasks is better, it means that the task t was correctly duplicated. Therefore, the best solution ind_{best} is replaced with ind_0 (Alg.3.1:28). Otherwise, the model goes back to its previous state and repeats the process with a new combination.

In the previous example, after splitting D into two new activities, SLAD ends up with an inconsistent model (Step 4 of Fig. 3.2(d)). Therefore, the dependencies of the model have to be repaired as explained. Note the task B is shared by both outputs of tasks D_1 and D_2. This has to be taken into account when repairing the process model, as B must now contain both tasks as inputs. When this situation occurs, the repairing process must add a new relation into the model. In other words, if O(t) and O(t') share an activity t'' when repairing the model, t' should be added *in the same subset* as t in $I(t'')$. Thus in the repaired solution shown in the Step 4 of Fig. 3.2(d), I(B) contains as input both tasks D_1 and D_2. Finally, in this example, the new repaired model after duplicating task D has a better precision than the initial solution, and therefore the best model is updated.

3.4.3 Handling length-two-loops

One of the novelties of SLAD is its ability to duplicate tasks in loops, specially in Length-two-Loops (L2L). In process mining, there is a well known relation between duplicate tasks and loops [136]. For example, in Fig. 3.2(a) the sequence of activities *BD* is executed multiple times, i.e., twice, but, as shown in the model depicted in Fig. 3.2(c), it can be considered that it is not a loop. Depending on the perspective, modeling this behavior as a loop could be a valid solution —for instance, by generalizing the model and hence allowing more executions of the loop than the recorded number in the log. With SLAD, we are going to consider that this type of behavior, i.e., situations where a sequence of activities is executed multiple times —twice as maximum—, will be modeled with duplicate tasks with the aim of improving the *precision* of the model. Otherwise, these situations —more than two repetitions— will be modeled as

a Length-one-Loop (L1L) or Length-two-Loop (L2L) —we do not want to unfold a loop by converting it into a large sequence. Detecting if the behavior can be modeled as a L1L or with two tasks with the same label can be detected with the previous process (Section 3.4.2). The problem arises when deciding if a L2L can be modeled with duplicate tasks.

The main limitation of the heuristic followed to detect the possible duplicate tasks of the log —Definition 4— is that it does not cover all the search space, particularly with tasks involved in a Length-two-Loop situation, as it breaks the rule of two tasks sharing the same input and output dependencies (Definition 3). For instance, considering the log of Fig. 3.2(a), only the activity D is identified as a potential duplicate activity, making it impossible to include in the search space the model depicted in Fig. 3.2(c), as B is not identified as a duplicate task: based on the log, B is only preceded by D and followed by D and J, hence $max(min(2,1),1) = 1$.

Making this process recursive can solve this drawback: when a task t is detected as a duplicate activity, the upper bound for all the tasks t' that directly follow t must be updated, because these tasks will now have multiple tasks with the same label as input. Note that updating only the tasks t' that directly follow t is sufficient, as this situation only occurs when loops of length two are involved in the computation of this heuristic —a task in a L2L usually have the same activity as input and output. To avoid a infinite recursion —Theorem 1—, the maximum number of times a task t can be modified by this recursive operation is twice. This bound is set also to avoid ending with a potential trace-model by unfolding a loop into a large sequence. In order to mimic this behavior in the proposed algorithm, if a task t is correctly duplicated in the model (Alg.3.1:28), we add the tasks that directly follow t into $potentialDuplicatesL2L$ (Alg.3.1:29). Following with the previous example, when duplicating D, the algorithm also needs to include B as a potential duplicate task. Therefore, after splitting the behavior of D among the new tasks D_1 and D_2, the algorithm adds to $potentialDuplicatesL2L$ the outputs of D, i.e. $potentialDuplicatesL2L = [B,G,H]$ as shown in the Step 5 of Fig. 3.2(d).

Theorem 1 (Infinite recursion)**.** *Given a trace with a very long length-two-loop, recursively applying Definition 2 after splitting an activity can lead to a trace-model.*

Proof. Let T be a set of tasks. Let L be a log over T and $\sigma = \ldots t_1 t_2 t_3 t_4 t_5 \ldots$ be a fragment of trace where label(t_2) = label(t_4) = a, and label(t_3) = b, i.e., $\sigma = \ldots t_1 abat_5 \ldots$. Based on Definition 4, the task a is selected to be duplicated. Assuming that this task is correctly split, this results in $\sigma = ...t_1 a_1 ba_2 t_5 ....$ As $b \in O(a)$, task b is selected as a potential duplicate

task. If the task b is correctly split, as $a_2 \in O(b)$, a_2 is going to be added again as a potential duplicate task. Let's consider now that label(t_5) = b_2, i.e., $\sigma = ...t_1a_1b_1a_2b_2....$ As $b_2 \in O(a_2)$, b_2 is selected again as a duplicate task. If the L2L between tasks a and b in σ is infinite, i.e., $\{...abababa...\}$, this could lead to an infinite recursion between these two tasks as long as they are correctly split. □

Once the algorithm ends its iteration over the main loop (Alg.3.1:11-31), it checks if there are tasks that were affected by the duplication of other activities, i.e., it checks if *potentialDuplicatesL2L* is empty (Alg.3.1:32). If this condition is false, the algorithm makes a recursive call but considering *potentialDuplicatesL2L* as input (Alg.3.1:33) —following with the example, now the algorithm has to iterate over $[B,G,H]$. In this new iteration, the process is the same as explained before.

Considering that the new task to be duplicated in this new iteration is B, the algorithm first retrieves the context in which this task is involved. In this new particular case, we have to take into account that the original task D from the log was duplicated in the model, generating D_1 and D_2. Therefore, when reproducing the solution over the log, we can check which activities with the same label $D \in$ I(B) were executed just before B and which activities $t \in$ O(B) were executed after B. With this process, the algorithm creates the sequences needed to generate the combinations to split B: when D_2 is executed before B, then D_1 is always executed, and when D_1 is executed before B, J is always executed later —where $J, D_1 \in$ O(B). These subsequences are shown in the Step 6 of Fig. 3.2(d).

Then, with these subsequences the algorithm builds the combinations considering that the tasks with the same label are different (Step 6 in Fig. 3.2(d)), and create two different combinations as explained in Section 3.4.2. The resultant combinations are shown in the Step 7 in Fig. 3.2(d). Note that, as these combinations do not share neither *inputs* nor *outputs*, both *sharedInputs* and *sharedOutputs* are empty. With this process, the algorithm is able to split B into two tasks B_1 and B_2, obtaining the model shown in the Step 8 in Fig. 3.2(d). After repairing the solution —for instance, I(D_1) has to be updated—, this model achieves a better quality than the best solution so far (Step 8 in Fig. 3.2(d)), allowing to unfold the loop into a sequence of tasks, and retrieving the model depicted in Fig. 3.2(c). If we would have extended the log with a new trace repeating the sequence *BD* more than twice, i.e., a case like <*A*,*D*,*B*,*D*,*B*,*D*,*H*,*J*>, the obtained model would have a lower fitness replay than the actual best solution —it would be impossible to execute the tasks more than twice— and therefore this behavior would be modeled as a L2L instead of unfolding the loop.

At this point the algorithm will continue to check if the other tasks can be duplicated — and even make another recursive call with the outputs of the already split task B. However, the model cannot be further improved by SLAD, as it perfectly models the behavior of the log.

3.5 Experimentation

The validation of SLAD has been done with a set of synthetic models from [30, 75]. Table 3.1 summarizes the original known models on the basis of their activities and the workflow patterns that each net contains[6], For example, the *FlightCar* model has eight *different tasks* structured in sequences, choices and parallel constructs. It also has duplicate tasks in sequence[7], meaning that, although the model contains eight different labels, it can be represented with more than eight activities. For each of these models, there is log with 300 traces. Table 3.1 shows the total number of events in each log. Note that, in these logs, we also included two additional dummy activities —a start and end activity—, as some algorithms used in the experimentation, including SLAD, are very sensitive when handling event logs with more than one start and/or end points [15, 30, 147, 155].

3.5.1 Metrics

The quality of the models retrieved by the proposed approach were measured by taking into account three objectives: fitness replay, precision and simplicity. To measure the fitness replay (F), we use the *proper completion* metric [105]:

$$F = \frac{PPT}{|L|} \tag{3.1}$$

where PPT is the number of properly parsed traces, and $|L|$ is the total number of traces in the event log. Hence, *proper completion* takes a value of 1 if the mined model can process all the traces without having missing tokens or tokens left behind. Also, the precision (P) is evaluated as follows:

$$P = 1 - max\{0, P'_o - P'_m\} \tag{3.2}$$

[6] All the datasets and experiments can be found in `http://tec.citius.usc.es/processmining/SLAD`.

[7] Models with duplicate tasks in parallel mean that the activities with the same label are executed in different branches, whereas duplicate tasks in sequence are executed in the same branch.

Table 3.1: Process models used in the experimentation.

	Activity structures											Log content	
Model Name	#Labels	Sequence	Choice	Parallelism	Length-One Loop	Length-Two Loop	Structural Loop	Non-local NFC	Invisible tasks	Duplicates in Sequence	Duplicates in Parallel	#traces	#events
betaSimpl [30]	13	✓	✓				✓	✓	✓			300	4,209
FlightCar [30]	8	✓	✓	✓						✓		300	2,385
Fig5p1AND [30]	5	✓		✓					✓			300	2,400
Fig5p1OR [30]	5	✓	✓						✓			300	2,100
Fig5p19 [30]	8	✓	✓	✓				✓	✓			300	2,428
Fig6p9 [30]	7	✓	✓					✓	✓			300	2,592
Fig6p10 [30]	11	✓	✓	✓				✓	✓			300	4,376
Fig6p25 [30]	21	✓	✓			✓	✓		✓	✓		300	5,661
Fig6p31 [30]	9	✓	✓						✓			300	2,400
Fig6p33 [30]	10	✓	✓						✓			300	2,504
Fig6p34 [30]	12	✓	✓	✓			✓		✓	✓		300	5,406
Fig6p38 [30]	7	✓		✓							✓	300	3,000
Fig6p39 [30]	7	✓		✓					✓		✓	300	2,684
Fig6p42 [30]	14	✓	✓	✓					✓	✓		300	3,420
RelProc [30]	16	✓	✓	✓						✓		300	4,155
Alpha [75]	11	✓	✓	✓						✓	✓	300	3,978
Loop	7	✓	✓		✓					✓		300	2,175
FoldedOr	6	✓	✓							✓		300	2,200

where P'_o and P'_m are, respectively, the precision of the original model and the precision of the mined model, both evaluated through *alignments*. More specifically, we used the metric defined in [6], considering all possible optimal alignments. It takes a value of 1 if all the behavior allowed by the model is observed in the log. As the original model is the optimal solution, we use it to normalize the precision. Therefore, P will be equal to 1 if the mined model has a precision (P'_m) equal or higher than the original model (P'_o). When the precision of the mined model is worse than that of the original model, P will take a value under 1 — the lower the precision of the mined model, the closer the value of P to 0. Finally, for the simplicity (S) we use:

$$S = \frac{1}{1 + max\{0, S'_m - S'_o\}} \tag{3.3}$$

where S'_m and S'_o are, respectively, the simplicity of the mined model and the simplicity of the original model, both calculated with the *weighted P/T average arc degree* defined in [111] —the higher the value of S' the lower the simplicity. As explained with the precision, we use the original model —which is the optimal model— to normalize the simplicity. S takes a value of 1 if the simplicity of the mined model is equal or higher than that of the original model, i.e., $S'_m \leq S'_o$. If the simplicity of the mined model is worse than that of the original model ($S'_m > S'_o$), S will take a value under 1 —the worse the simplicity of the mined model, the closer the value of S to 0.

We used the tool CoBeFra [17] to compute the different metrics. Note that the model input representation for this tool is a Petri net, therefore we had to map each heuristic net retrieved by SLAD into its equivalent Petri net.

Table 3.2: Results for the 18 logs.

			Logs																	
			Alpha	**Folded**	**Loop**	**Fig6p25**	**betaSimpl.**	**flightCar**	**Fig5p19**	**Fig5p1AND**	**Fig5p1OR**	**Fig6p10**	**Fig6p31**	**Fig6p33**	**Fig6p34**	**Fig6p38**	**Fig6p39**	**Fig6p42**	**Fig6p9**	**RelProc**
1	*ProDiGen*	*F*	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0.85	0.75	0.8	0.78	0.92	0.81	0.9	0.76	0.73	0.82	0.61	0.65	0.8	0.93	0.93	0.7	0.79	0.94
		S	0.82	0.84	0.86	0.88	0.92	0.82	0.88	0.69	0.67	0.82	0.61	0.73	0.82	0.84	0.92	0.79	0.79	0.94
2	*HM*	*F*	1.0	1.0	1.0	1.0	1.0	0.32	1.0	0.67	1.0	1.0	1.0	1.0	0.41	0.0	0.0	0.07	0.21	1.0
		P	0.77	0.75	0.8	0.76	0.92	0.81	0.9	0.75	0.67	0.82	0.56	0.6	0.83	0.57	0.6	0.64	0.95	0.94
		S	0.85	0.84	1.0	0.86	0.99	0.81	0.94	1.0	0.8	0.82	0.63	0.68	0.78	0.62	0.82	0.82	0.99	0.94
3	*Fodina*	*F*	1.0	1.0	1.0	1.0	0.23	0.32	0.28	0.32	1.0	1.0	1.0	1.0	1.0	1.0	0.52	0.31	0.21	0.72
		P	0.77	0.75	0.8	0.76	0.97	0.88	0.82	0.73	0.67	0.82	0.56	0.6	0.8	0.93	1.0	0.93	0.86	0.96
		S	0.85	0.84	1.0	0.86	1.0	1.0	1.0	1.0	0.8	0.82	0.63	0.68	0.82	0.91	1.0	1.0	1.0	0.98
4	*IMi*	*F*	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0.78	0.83	0.73	0.79	0.68	0.81	0.73	0.83	0.7	0.66	0.63	0.67	0.54	0.74	0.95	0.34	0.7	0.76
		S	0.84	0.88	0.98	0.81	0.98	0.89	0.87	0.92	0.83	0.77	0.69	0.73	0.82	0.86	1.0	0.9	0.84	0.85
5	*ProDiGen + SLAD*	*F*	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0.97	1.0	1.0	1.0	1.0	1.0	1.0	0.98	1.0	1.0	1.0	1.0	1.0	1.0	0.94	1.0	1.0	1.0
		S	0.91	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.89	1.0	1.0	1.0
6	*HM + SLAD*	*F*	1.0	1.0	1.0	1.0	1.0	0.32	1.0	0.67	1.0	1.0	1.0	1.0	0.72	1.0	0.53	0.36	0.21	1.0
		P	0.86	1.0	1.0	1.0	0.99	0.82	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.94	0.95	0.95	1.0
		S	0.96	1.0	1.0	1.0	1.0	0.84	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.99	1.0
7	*Fodina + SLAD*	*F*	1.0	1.0	1.0	1.0	1.0	0.45	0.28	0.32	1.0	1.0	1.0	1.0	1.0	1.0	0.52	0.31	0.21	0.72
		P	0.86	1.0	1.0	1.0	1.0	0.88	0.82	0.73	1.0	1.0	1.0	1.0	1.0	0.93	1.0	0.93	0.86	1.0
		S	0.85	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.91	1.0	1.0	1.0	0.99
8	*Fodina + D*	*F*	1.0	1.0	1.0	1.0	1.0	1.0	0.52	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.22	0.75	1.0	0.62
		P	1.0	0.75	0.74	1.0	1.0	1.0	1.0	1.0	1.0	0.91	1.0	1.0	0.85	0.93	0.91	0.97	1.0	0.98
		S	1.0	0.84	0.84	1.0	1.0	1.0	1.0	1.0	1.0	0.92	1.0	1.0	0.85	0.91	0.92	1.0	1.0	0.98
9	*DGM*	*F*	1.0	0.67	0.33	0.38	1.0	1.0	1.0	1.0	1.0	0.43	1.0	0.66	0.41	1.0	0.23	0.38	1.0	0.28
		P	1.0	0.8	0.87	0.98	1.0	1.0	1.0	1.0	1.0	0.67	1.0	0.85	0.7	0.93	0.78	0.62	1.0	0.78
		S	1.0	0.78	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.91	1.0	0.84	0.99	0.91	0.81	0.88	1.0	0.83
10	*ETM*	*F*	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	1.0	0.9	0.83	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.9	0.73	0.91	1.0	0.95	0.11	0.92	0.14
		S	1.0	0.93	0.92	0.47	1.0	1.0	1.0	1.0	1.0	1.0	0.87	0.7	0.9	1.0	0.58	0.41	0.93	0.41
11	*TS*	*F*	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0
		P	0.51	0.62	0.87	0.89	0.95	1.0	0.82	0.78	0.75	1.0	0.82	0.82	1.0	0.84	0.94	0.91	0.77	0.84
		S	0.35	0.74	0.62	0.46	0.55	1.0	0.41	0.7	0.83	0.37	0.59	0.61	0.35	0.61	0.61	0.55	0.46	0.36
12	*Fodina + D + SLAD*	*F*	1.0	1.0	1.0	1.0	1.0	1.0	0.52	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.36	0.75	1.0	0.62
		P	1.0	1.0	0.98	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.88	0.93	1.0	1.0	1.0	1.0
		S	1.0	1.0	0.98	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.91	1.0	1.0	1.0	1.0

3.5.2 Setup

As explained in Section 3.4, SLAD takes as starting point a model and a log. Therefore, the first step in the validation process is to apply a process discovery technique over the previously described logs (Table 3.1) to retrieve a heuristic net or causal net as input. In order to test how the solutions of different algorithms affect the behavior of SLAD, we repeated the experiments taking as starting point the solutions of three different algorithms that retrieve solutions in the aforementioned format: ProDiGen [147], Heuristics Miner (HM) [155], and Fodina [15] —all these algorithms, by default, do not handle label splitting. To identify the solutions retrieved by these algorithms after applying SLAD, we have added the suffix "+*SLAD*". Additionally, we also used Inductive Miner [71] (IMi) in the experimentation[8].

On the other hand, we wanted to compare the results of mining the duplicate activities *before* or *after* the discovery process. Therefore, we also repeated the experiments with Duplicate Genetic Miner (DGM) [30], Evolutionary Tree Miner (ETM) [20], a state-based region theory algorithm (TS) [9, 113] and Fodina [15], as it can mine duplicate activities if enabled —we use the name Fodina+D as a way to reference the algorithm with this parameter enabled. Additionally, we also tested if it is possible to apply SLAD to an already mined solution with duplicate activities (with Fodina and DGM). Note that for all these algorithms we used the default settings specified by the authors. More specifically: i) for Inductive Miner we guarantee a perfect replay fitness; ii) for DGM we set the *maximum number of iterations* to 5,000, and a *population size* of 50; iii) for the ETM we generate the pareto front for each log, retrieving the solution with the highest fitness replay and precision; and iv) for the state-based region theory algorithm, when discovering the transition system, we set *no limit* in the *set* size, and the inclusion of all activities. We used the ProM framework [139] to execute each of these algorithms.

3.5.3 Results

Table 3.2 shows the results —in terms of fitness replay (F), precision (P) and simplicity (S)— retrieved for each algorithm over each log. Moreover, Table 3.2 also shows information about which algorithm retrieves better results for each metric and log —highlighted in grey. In Section 3.5.3 we prove that applying SLAD to the mined models improves the solutions retrieved by the algorithms that do not take into account duplicate tasks. In Section 3.5.3 we show that

[8] We did not apply SLAD over IMi and ETM, as these algorithms retrieve process trees as a solution, and, in some situations, this kind of models cannot be easily translated to heuristic nets without changing its internal behavior.

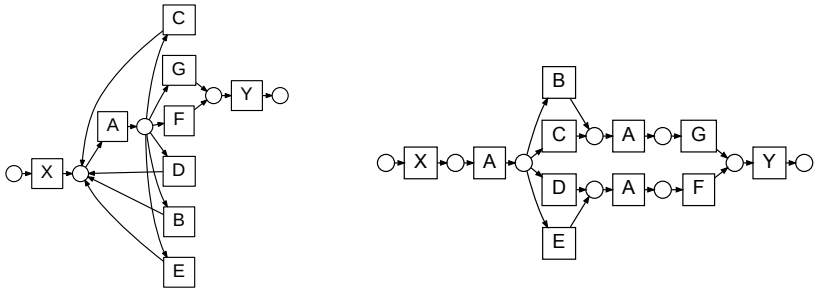
the models obtained with SLAD, considering this case study, are better than those mined with other algorithms that consider duplicated tasks —Fodina+D, ETM, DGM.

Improvement of the mined models through SLAD.

First, we analyze the results of the algorithms used to retrieve the initial solution for SLAD: ProDiGen, HM, Fodina and IMi —rows 1-4 in Table 3.2. With ProDiGen and IMi, all the solutions have a perfect fitness replay, whereas HM and Fodina do not achieve a perfect value for this quality dimension in some of the cases. In general, for all the algorithms, the mined nets contain overly connected nodes that make the models hardly readable and complex, i.e, models with a poor precision and activities with a high density of incoming/outgoing arcs, needless loops, or too many invisible activities.

Then, we applied SLAD to the models mined by ProDiGen, HM and Fodina. Our proposal was able to enhance the results in 45 out of 54 solutions —rows 5-7 in Table 3.2. One example of this improvement can be seen in Figure 3.3, where Figure 3.3(a) shows the original Petri net mined by HM for the log *Fig6p31*, and Figure 3.3(b) shows the Petri net after finding the duplicate tasks through SLAD —task *A* in this case. In this example, the process model of Figure 3.3(a) has a very low precision (0.56) as the number of different traces it can generate is infinite due to the loop with the activity A. However, through the inclusion of duplicate labels (Figure 3.3(b)), we can limit its behavior to only four different paths, i.e., we create a more specific process model.

Furthermore, for those initial models mined by ProDiGen, SLAD was able to improve



(a) Mined model by Heuristics Miner without duplicate tasks.

(b) Mined model by Heuristics Miner and SLAD.

Figure 3.3: A Petri net mined for the log *Fig6p31* before and after SLAD.

the precision in all the cases. Taking as starting point the initial solutions retrieved by HM, 17 out of 18 solutions were improved not only in terms of precision, but also the fitness replay achieves a higher value in four of those models. After applying SLAD over the results retrieved by Fodina, 12 out of 18 solutions were improved in terms of both precision and fitness replay. Analyzing the simplicity of the models, we have to take into account that when performing the label splitting over the initial models, there is a reduction in the number of invisible activities —used as control structures— as we are reducing most of the needless loops and overly connected nodes. For example, with the log *Fig6p10*, ProDiGen, Fodina and HM retrieve a solution —the three algorithms retrieve the same solution— with 46 arcs, and 22 tasks —including the invisible activities. After SLAD, the final solution has, in total, 36 arcs and 17 tasks.

Table 3.3: Wilcoxon test for each algorithm with and without SLAD.

Comparison	*p-value*	**Hypothesys**
ProDiGen+SLAD vs ProDiGen	0.0002	**Rejected**
HM+SLAD vs HM	0.0002	**Rejected**
Fodina+SLAD vs Fodina	0.0019	**Rejected**
Fodina+D+SLAD vs Fodina+D	0.014	**Rejected**

We have compared the results of applying SLAD by means of non-parametric statistical tests —through the web platform STAC [99]—, checking if each algorithm with SLAD improves its correspondent algorithm without SLAD. The Wilcoxon test [160] has been applied, using as null hypothesis that the medians of the quality of the solutions are equal with a given significance level (α). However, as we are using 3 different metrics, the comparison must be done in a multi-objective way. In order to perform a fair comparison, we have used the criterion of Pareto dominance. We have applied the *fast-non-dominated-sort* [35] in order to rank the solutions of the algorithms for each log. With this method, a mined model a dominates other mined model b, i.e., $a \succ b$, if the model a is not worse than the model b in all the objectives —the 4 metrics— and better in at least one objective. Thereby, for each log, all the solutions in the first non-dominated front will have a rank equal to 1 —these are the solutions more similar to the original model and, therefore, the best ones—, the solutions in the second non-dominated front will have a rank equal to 2, and the process continues until all fronts are identified. Therefore, to perform the non-parametric statistical tests, we first ranked all the solutions for each log based on the Pareto dominance and then we used these ranks as input

for the tests.

Table 3.3 summarizes such test. The hypothesis —which states that the solutions retrieved before and after applying SLAD are equal— is rejected in all the cases, as the *p-value* for each comparison is lower than the given confidence level (α = 0,05). This means that SLAD was able to i) *significantly* improve the precision of the models, and ii) enhance their structural clarity by splitting the behavior of the overly connected activities. In summary, SLAD significantly enhances the solutions of the process discovery algorithms.

Comparison of process discovery algorithms with duplicate activities.

Rows 8-11 in Table 3.2 show the results of the algorithms that take into account duplicate activities in the mining process. Fodina, when mining duplicate activities, is able to enhance the solution on 13 out of 18 solutions, however, this algorithm also retrieves a worse solution in three of the cases —*Loop*, *Fig6p39* and *RelProc*— than without duplicate activities. On the other hand, DGM is able to retrieve the original model in eight of the 18 cases, and ETM in seven of the 18 cases. In particular, these algorithms were able to retrieve to original solution in cases where two activities with the same label are executed in different branches of a parallel construct. For instance, the three algorithms were able to retrieve the original model with the *Alpha* log. On the other hand, TS was only able to retrieve the original solution in one case. In all the cases the state-based region theory algorithm results process models with a guaranteed perfect replay fitness. However, this type of algorithms tend to overfit the event log, resulting in solutions with a very low simplicity.

Comparing the solutions of performing the label splitting after —rows 5-7 in Table 3.2— and before —rows 8-11 in Table 3.2— we can extract that with SLAD, the most difficult scenario is related to those situations where two different activities with the same label are executed in different branches —log *Alpha* or *Fig5p1AND*. To achieve this duplicity of a label in different branches of a parallel construct, it is necessary to perform the label splitting *before* or during the mining step, otherwise the process mining techniques usually isolate the affected activity into only one branch, precluding the label splitting in a post-processing step. Additionally, SLAD is highly dependant on the input solution, more specifically, on the fitness replay of the initial model. As can be seen in the results after applying SLAD over Fodina and HM —rows 5-6, respectively, of Table 3.2—, if the input model avoids the overly connected nodes by means of reducing its fitness replay, it is more difficult to perform the label splitting through SLAD.

Table 3.4: Friedman ranking for all the algorithms with SLAD.

Algorithm	Ranking
ProDiGen+SLAD	25.528
Fodina+D+SLAD	35.583
HM+SLAD	40.611
Fodina+SLAD	44.278
p-value: 0.0941	

We also compared in more detail the solutions retrieved with Fodina detecting the duplicate activities *after* (Fodina+SLAD) and *before* (Fodina+D) mining the model —rows 7-8 of Table 3.2. Within this context, detecting the duplicate activities through SLAD improves 14 out of 18 solutions, whereas detecting the duplicate activities before the mining process, improves 13 out of 18 solutions. In particular, there are some solutions that were improved performing the label splitting before —for instance, log *Fig6p9*—, and after —for instance, log *Fig5p19*— the process mining. On the other hand, as previously indicated, in three of the solutions Fodina+D retrieved a worse model than the one without duplicate activities, whereas with SLAD the final solution is always the same or better than the original.

It should be noted that performing the label splitting before and after the mining process is not exclusive, i.e. it is possible to apply SLAD to a solution that already contains duplicate activities. Therefore, we carried out this experiment with the solutions retrieved by Fodina+D (Fodina+D+SLAD) and DGM (DGM+SLAD). Fodina+D+SLAD enhances a total of 7 out of 18 models —row 12 of Table 3.2— of the solutions retrieved by Fodina+D. On the other hand, SLAD was unable to enhance any of the models mined by DGM, therefore we omitted this row in the table, as the results where the same as the achieved by the DGM —row 9 of Table 3.2.

Finally, we compared the algorithms that include the duplicate activities in the mining process with the best algorithm with SLAD. In order to select the best algorithm with SLAD, we applied the Friedman Aligned Ranks test [61], using the rank solutions based on the Pareto dominance. This test computes the ranking of the results of the algorithms rejecting the null hypothesis —which states that the results of the algorithms are equivalent— with a given confidence or significance level (α). Then we applied the Holm's post-hoc test [62] for detecting significant differences among the results. Table 3.4 summarizes this test, ranking ProDiGen+SLAD as the best algorithm. We omitted the Holm's post-hoc test, as based on the

p-value of the previous test, the differences are not significant.

Table 3.5: Non-parametric test.

(a) Friedman ranking.

Algorithm	Ranking
ProDiGen+SLAD	22.527
Fodina+D	40.833
ETM	44.666
DGM	50.277
TS	69.194
p-value: 0.0248	

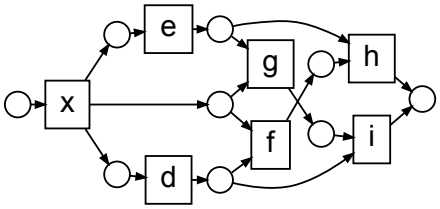
(b) Holm post-hoc, $\alpha = 0{:}05$.

Comparison	*Adj. p-value*	Hypothesis
ProDiGen+SLAD vs TS	0.0000	**Rejected**
ProDiGen+SLAD vs DGM	0.0043	**Rejected**
ProDiGen+SLAD vs ETM	0.0220	**Rejected**
ProDiGen+SLAD vs Fodina+D	0.0355	**Rejected**

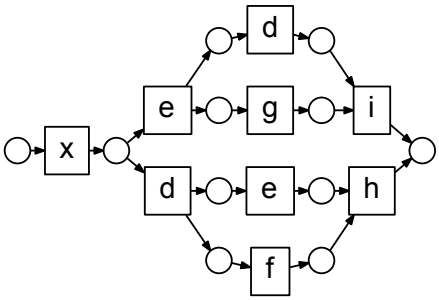
The comparison of Fodina+D, ETM, DGM, TS and ProDiGen+SLAD is summarized in Table 3.5. After applying the Friedman Aligned Ranks test ProDiGen+SLAD has the best ranking (Fig. 3.5(a)). Based on the results of the Friedman Aligned Ranks, we performed a Holm's post-hoc test (Fig. 3.5(b)), starting with the initial hypothesis that all the tested algorithms are equal to ProDiGen+SLAD. The test rejects the null hypothesis in all the cases for a confidence level of $\alpha = 0.05$ —the p-value of each algorithm has to be lower than α in order to reject the hypothesis. This means that ProDiGen+SLAD outperforms all the other algorithms, and that the difference is statistically significant for that confidence level.

ProDiGen+SLAD obtains the original model in 15 out of 18 logs. The most difficult situations for SLAD are the duplicates in parallel. Nevertheless, as previously said, this problem is more related to the input solution, rather than to the performance of the proposed algorithm (Figs. 3.4 and 3.5) as it cannot create new behavior in the mined model. More specifically:

- For log *Alpha* (Fig. 3.4), although the algorithm correctly detects the tasks e and d as potential duplicates, due to the original model retrieved by ProDiGen (Fig. 3.4(a)), it cannot construct a model with those tasks repeated in the different branches (Fig. 3.4(b)). This is because ProDiGen finds a non-free-choice pattern that mimics the same behavior as the model with duplicate tasks disabling the possibility to split the causal dependencies of these activities. In other words, both tasks d and e are always executed in the same trace. Therefore, ProDiGen mines a parallel construct involving these two tasks. Then, depending on the execution of g or f, the next task to be executed can be h or i, respectively.



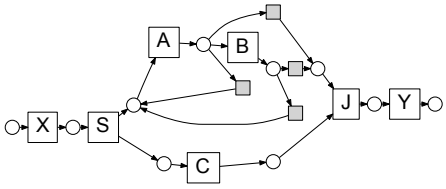
(a) Detail of the mined model by ProDiGen without duplicate tasks. SLAD does not improve this part of the model.



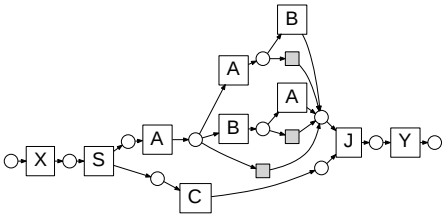
(b) Detail of the original model with duplicate tasks.

Figure 3.4: Two models depicting the same behavior for the log *Alpha*, but with and without duplicate tasks.

- For model *Fig6p39* (Fig. 3.5) the algorithm cannot correctly obtain the parallelism with the task *A*, which is again executed multiple times in different branches. Based on this, ProDiGen tries to reproduce as much behavior as possible overly connecting the same task *A* in only one branch (Fig. 3.5(a)). For this reason, SLAD tries to split the task *A*(Fig. 3.5(b)), improving the precision but still allowing for more behavior than the recorded in the log. This is one example of how duplicate activities executed in different branches are isolated when mining the log without considering the duplicity.



(a) Initial solution retrieved by ProDiGen without duplicate tasks.



(b) The same model after the local search.

Figure 3.5: Two models from the log *Fig6p39* with and without duplicate tasks.

One of the objectives of SLAD is to retrieve high values of precision, i.e., to create a more specific process model and, therefore, to reduce the generalization that overly-connected

Table 3.6: Generalization values for 18 logs before and after SLAD.

		Logs																
		Alpha	Folded	Loop	Fig6p25	betaSimpl	BigCar	Fig5p19	Fig5p1AND	Fig5p1OR	Fig6p10	Fig6p31	Fig6p33	Fig6p34	Fig6p38	Fig6p39	Fig6p42	Fig6p9	RelProc
ProDiGen	*G*	0.99991	0.99997	0.99991	0.99945	0.70556	0.51282	0.75333	0.0	0.0	0.9992	0.41667	0.38462	0.99986	0.6375	0.94324	0.96336	0.62308	0.99796
ProDiGen + SLAD	*G*	0.99991	0.99995	0.99991	0.99945	0.38889	0.51282	0.75333	0.0	0.0	0.99887	0.41667	0.38462	0.99982	0.6375	0.9301	0.95229	0.62308	0.99591
HM	*G*	0.99993	0.99997	0.99991	0.9995	0.79722	0.47619	0.75333	.44444	0.33333	0.9992	0.68254	0.69597	0.99984	0.7	0.97596	0.96493	0.725	0.99796
HM + SLAD	*G*	0.99992	0.99995	0.99991	0.99945	0.72639	0.2381	0.75333	0.0	0.0	0.99887	0.41667	0.38462	0.99976	0.5875	0.92241	0.96183	0.725	0.99591
Fodina	*G*	0.99993	0.99997	0.99991	0.9995	0.7911	0.40000	0.5297	0.1363	0.33333	0.9992	0.68254	0.69597	0.99986	0.6375	0.93467	0.97421	0.83334	0.99797
Fodina + SLAD	*G*	0.99991	0.99995	0.99991	0.99945	0.72639	0.40000	0.5297	0.1363	0.0	0.99887	0.41667	0.38462	0.99982	0.6375	0.93467	0.97421	0.83334	0.99797
Fodina + D	*G*	0.99993	0.99997	0.99993	0.99945	0.72639	0.51282	0.85278	0.0	0.0	0.99901	0.41667	0.38462	0.99987	0.6375	0.9783	0.96115	0.62308	0.9976
Fodina + D + SLAD	*G*	0.99993	0.99995	0.99993	0.99945	0.72639	0.51282	0.85278	0.0	0.0	0.99887	0.41667	0.38462	0.99987	0.6375	0.96799	0.96014	0.62308	0.9971
Original Model	*G*	0.99993	0.99995	0.99991	0.99945	0.38889	0.51282	0.75333	0.0	0.0	0.99887	0.41667	0.38462	0.99982	0.6375	0.9246	0.95229	0.62308	0.99591

nodes and unnecessary loops can introduce to the process model. As precision and generalization are opposed objectives [19, 122], and SLAD is trying to increase the precision and reduce the generalization, we evaluated the generalization separately to analyse how much this dimension decreases when SLAD introduces duplicate labels. Table 3.6 presents the values for the generalization —*G*—, for the models retrieved by ProDiGen, HM, and Fodina, before and after SLAD, using the *Alignment Based Probabilistic Generalization* metric [124]. Table 3.6 also shows the values of generalization for the original model as a baseline. Note that a generalization value closer to 1 means that the process model is too general —a less specific process model— whilst a value closer to 0 means that there is no room for reproducing unseen behavior —a more specific process model. Furthermore, for each pair of algorithms, we have shadowed those cases where the generalization changes after applying SLAD. As can be seen, for ProDiGen the generalization was lower after applying SLAD in 7 out of 18 cases. The cases where SLAD retrieved the same generalization is because ProDiGen, through its hierarchical fitness function —Definition 9—, already tries to reduce this dimension of a process model by focusing in precise process models —generally through the inclusion of non-free-choice constructs. On the other hand, for HM, in 15 out of 18 cases the label splitting process retrieved a more specific process model. With Fodina, SLAD reduced the generalization in 9 of the cases —both Fodina and HM usually introduce a high number of unnecessary loops and overly connected nodes that can be executed any time in the process. Finally, with the Fodina extension to mine duplicate labels —Fodina+D—, SLAD retrieved a more specific process model in 6 out of 18 cases —Fodina+D already mines models with duplicate labels, therefore it retrieves solutions reducing the generalization. It should be noticed that in all the cases SLAD retrieved equal or lower generalization than its respective process model without duplicate labels.

In summary, SLAD was able to detect the duplicate tasks and *significantly* improve the precision in 45 out of 54 solutions —considering the initial solutions of ProDiGen, HM and Fodina— and also the fitness replay in some of the cases. More specifically, after applying SLAD, 35 out of 54 solutions were equal to the original model. Furthermore, the algorithm was able to unfold those situations detected as a loop, i.e., situations with repeated sequences —at most two times— that can be represented with duplicate tasks in order to improve the precision of the model. Additionally, in *none of the cases* the presented approach retrieved the trace-model after the label splitting process, avoiding the overfitted models with one path per trace. Also, none of the resulting models after applying SLAD were worse than its respective initial solution.

Complexity analysis.

With respect to the runtime of SLAD, we can identify two main time consuming parts in the algorithm: *discovering the duplicate tasks* (Alg. 3.1:4-6), and *evaluating* the solutions (Alg. 3.1:24). On the one hand, discovering the duplicate tasks is a costly function as, in order to detect the follows relation of each activity, it is necessary to build a dependency graph, which requires one pass through the log and, for each trace, one scan through the trace. This translates to a complexity $O(n)$, where n represents the total number of events in the log. Fortunately, most process discovery algorithms [15, 30, 147, 155] already compute this dependency graph from an event log in order to mine a process model. Therefore SLAD can reuse this information without the need to recompute it, which reduces this process to check only *once* —at the beginning of the algorithm— each of the different activities (T) of the log, i.e $O(T)$. On the other hand, evaluating each solution is the main *bottleneck* of SLAD. Each time SLAD generates a new potential solution after splitting a task, the algorithm needs to evaluate its replay fitness, precision and simplicity. Unfortunately, with current state of the art conformance checking metrics, such as alignments or token replay techniques, this is a very time consuming process when dealing with large event logs. Nevertheless, we can approximately relate the complexity of a *greedy* conformance checking technique to the number (t) and length (l) of the traces in an event log, hence $O(t \cdot l)$. In fact, this is equal to traverse all the events of the log, therefore we can set $O(t \cdot l)$ to $O(n)$. Regardless the complexity of the conformance checking metric, SLAD has to check and split each one of the activities detected as a potential duplicate. Thereby, the complexity of this part can be set as $O(d \cdot c)$ where d is the total number of potential duplicate activities and c is the number of times the activities

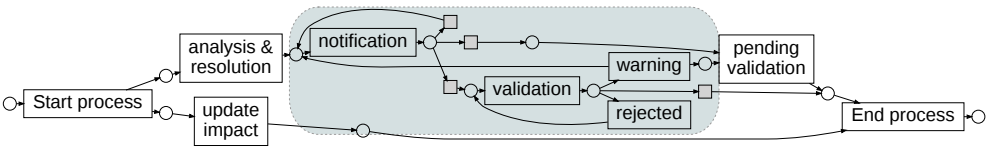
can be split. In summary, the complexity of SLAD is $O(d \cdot c \cdot n)$. This does not include the complexity of building the initial dependency graph.

Regarding the logs used in the experimentation, the runtime of SLAD was, on average, 9 seconds —considering the time needed to compute the dependency graphs. In particular, the highest time was achieved when applying SLAD over the solution retrieved by ProDiGen with the log *Fig6p42*, that took 12 seconds.

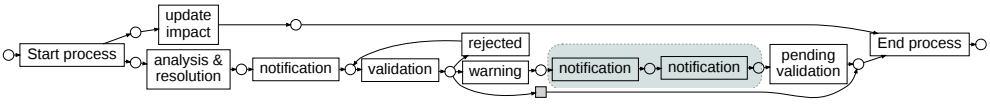
Experiments in a real-life scenario.

Finally, we have tested ProDiGen+SLAD in a real-life scenario: a process model within an IT Service Management platform [145]. This process is related to handling incidents and requests, henceforth tickets, in a *service desk*, i.e., a central point of communications between users and staff in an organization. The process model has 7 different activities. However one of these activities, *notification*, is executed multiple times during the process. In particular, albeit its purpose is to notify different involved staff during the process of handling a ticket, it is always recorded with the same label. Hence, using this process model we generated an even log containing 300 traces to check if it is possible to obtain again the original process.

Figure 3.6(a) shows the process model discovered with ProDiGen, and Figure 3.6(b) displays the same process model after applying SLAD. In this example, both process models have a perfect replay fitness. However, the precision of the model without duplicate labels is lower —0.58— than the precision of the process model after applying SLAD —0.87. The low precision of the model without duplicate labels (Fig. 3.6(a)) is due the overly-connected



(a) Initial solution retrieved by ProDiGen without duplicate tasks.



(b) The same model after applying SLAD.

Figure 3.6: Process models mined for a real event log before and after SLAD.

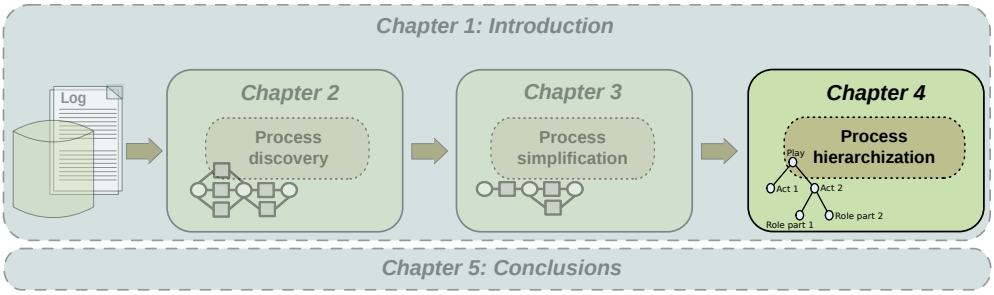
activity *notification*, that enables to repeat the shadowed part of the model without any limit. On the other hand, the model retrieved after applying SLAD (Fig. 3.6(b)) removes this loop by duplicating *notification* three times. Furthermore, the only difference between the designed process model, and the one retrieved by SLAD, is the shadowed part in Fig. 3.6(b). In reality, there are always two notifications that are performed in parallel: one of the notifications is for the *Incidents manager* and the other one is for the *Senior manager*. This two activities always happen as a sequence sharing the same label. As we do not have any other information regarding this situation, it is impossible to distinguish them, thus its representation as a sequence.

3.6 Conclusions

We have presented SLAD, an algorithm to tackle duplicate tasks in an already discovered model. Our proposal takes as starting point a model without duplicate tasks and its respective log and, based on heuristics, the local information of the log, and the causal dependencies of the input mined model, it improves the fitness replay, precision and simplicity of the model. SLAD has been validated with 18 different logs and 54 different initial solutions from three different process mining algorithms. Results show that the algorithm was able to enhance the initial solutions in 45 of the 54 tested scenarios. Moreover, we have compared SLAD with the state of the art process discovery algorithms with duplicate tasks. Statistical test have shown that the best SLAD algorithm (ProDiGen+SLAD) is better, and that the differences are statistically significant. As a future work, and based on the obtained results, we want to extend SLAD with the possibility to not only redistribute the already mined dependencies of the model, but introduce new relations based on the combinations of the potential duplicate activities, in order to enhance those models with a lower fitness.

CHAPTER 4

RECOMPILING LEARNING PROCESSES FROM EVENT LOGS



The visualization of a process model plays an important role in order to correctly gain insights about the process. That is, models that are unnecessary complex, can hinder the real behavior of the process rather than to provide insights of what is really happening. As shown in Chapter 3 taking into account duplicate labels can drastically change how a process depicts the behavior recorded in the log. But this simplification of a process model can be achieved even further through its hierarchization. However, to get a proper level of hierarchization, it is necessary to introduce domain knowledge within the process.

To this end, in this chapter we present the insights about how to use domain knowledge to *translate* an already discovered process model to a more specific representation, within a particular domain and, thus, retrieve a more interpretable process model. Specifically, we present a framework to automatically hierarchize a process model using domain knowledge.

This objective is achieved in three different steps. Firstly, the process model is automatically extracted from the logged sequences through the developed process mining algorithms. Then, an algorithm based on the knowledge about the target language control structure is applied to determine which components should be created. Finally the adaptive rules are automatically extracted form the event logs (more specifically, from the values of the variables in the logs) by a decision tree learning algorithm, and integrated into the target language structure. Hence, an important feature of the presented approach is its independence from any target modelling language. We have implemented and validated this framework within the educational domain, enabling the translation of a discovered process model into a standardized learning process model more suitable for teachers, i.e., the standard IMS LD. For a proper evaluation, the three parts of the presented framework for the hierarchization process models have been analyzed separately using a set of nine real courses with different degrees of complexity.

All these contributions are encompassed in the following publication:

J.C. Vidal[1], B. Vázquez-Barreiros[1], Manuel Mucientes[1], and Manuel Lama[1]. Recompiling Learning Processes from Event Logs. *Knowledge-Based Systems*, 100:160–174 2016.
(DOI: 10.1016/j.knosys.2016.03.003).

4.1 Abstract

In this paper a novel approach to reuse units of learning (UoLs) —such as courses, seminars, workshops, and so on— is presented. Virtual learning environments (VLEs) do not usually provide the tools to export in a standardized format the designed UoLs, making thus more challenging their reuse in a different platform. Taking into account that many of these VLEs are legacy or proprietary systems, the implementation of a specific software is usually out of place. However, these systems have in common that they record the events of students and teachers during the learning process. The approach presented in this paper makes use of these logs *i)* to extract the learning flow structure using process mining, and *ii)* to obtain the underlying rules that control the adaptive learning of students by means of decision tree learning. Finally, *iii)* the process structure and the adaptive rules are recompiled in IMS

[1]Centro Singular de Investigación en Tecnoloxías da Información (CiTIUS), Universidade de Santiago de Compostela. Santiago de Compostela, Spain.

Learning Design (IMS LD) —the *de facto* educational modelling language standard. The three steps of our approach have been validated with UoLs from different domains.

4.2 Introduction

While designing a course, there are two main concerns that worsen the realization of an educational scenario: *i)* how to model a practical pedagogical scenario to achieve the educational objectives, and *ii)* how to reuse this scenario in another context than the original. Teachers do not only define the learning content to be consumed by the learners, but they also include the different educational objectives, the order in which the learning activities must be undertaken to achieve these objectives, the evaluation methods, etc. Hence, to reuse and better validate an educational scenario, it should be explicitly written. Although these learning designs, i.e the descriptions of the educational process, are usually portrayed with documents that use natural language, they can be formally described through Educational Modelling Languages (EMLs). Moreover, when interacting with a virtual learning environment (VLE), learners also perform additional activities than the specifically defined by the teachers, such as interacting in the forum, checking the bibliography, etc. This information should be also highlighted to enable teachers to improve the *learning flow*, i.e the *real* workflow of learning activities, as well as the *evaluation process* [101]. Therefore, the defined educational scenario is more complex than the learning design explicitly documented by teachers.

In the last decade a great effort has been made for developing EMLs. The main idea underlying these languages is to describe, from a pedagogical point of view, the learning process of the course, i.e., the *sequence of steps* the learners should undertake to achieve the educational objectives of the course, by using the available educational resources and services. Regarding the wide variety of specifications for representing learning designs, one standard *de facto* has jumped into e-learning panorama: the IMS Learning Design (IMS LD) specification [46]. IMS LD enables the formal description of learning processes for a wide range of pedagogical contexts in a VLE. Although there is some controversy about whether IMS LD is too complex to be understood by teachers from a practical point of view [37] —especially with the levels B and C—, most of authors highlight this complexity as a barrier for adopting IMS LD [82].

To deal with this issue, a number of user-friendly authoring tools have appeared [1, 28, 53,

60, 66, 81, 152], but even with these tools, authoring process of IMS LD units of learning[2] (UoLs) is not easy for teachers when these UoLs are complex or require to use advanced features of this standard. The *automatic* reconstruction of UoLs could relieve this issue [82], promoting the use of IMS LD by teachers and instructors. Taking as starting point the event log files, which stores all the events generated by the learners, it is possible to mine the *real behavior* undertaken by the students during the UoL, i.e., what the learners really did, and the rules that constraint the behavior of the model. Then, by combining these two models, it is possible to reconstruct the UoL to a specific target language. Therefore, this process facilitates the reuse of defined UoLs no matter the VLE that has been used, as the techniques used for both mining the variable values and the formal model are totally independent of the domain. Therefore, teachers can design their courses within their VLE, avoiding the need to use an authoring tool with a specific EML notation, and still be possible to reconstruct the UoLs from the scratch to a target language — such as IMS LD.

In this paper, we present an approach to *automatically* reconstruct the IMS LD representation of an UoL from the events generated by the learners in the VLE. This objective is achieved in three different steps. Firstly, the learning flow of the UoL is automatically extracted from the logged sequences through a *process discovery algorithm*. Then an algorithm based on the knowledge about the IMS LD control structure is applied to determine which IMS LD components should be created. Finally the adaptive rules of the UoL are automatically extracted form the event logs —more specifically, from the variable values of the logs— by a decision tree learning algorithm, and integrated into the IMS LD structure. The contributions of this proposal are: *i)* a new framework to facilitate the reuse of UoLs between different VLEs; *ii)* the automatic discovery of learning processes from event logs and its recompilation to IMS LD; and *iii)* the automatic identification of the adaptive rules from event logs.

Notice that IMS LD has a high expressiveness to allow the definition and orchestration of complex activity flows in a multi-role setting, but at the expense of complexity. In fact, current IMS LD research seems to accept the assumption that specification's conceptual complexity hiders the authoring process [37]. Taking this into account, another objective of this paper is to reduce this barrier and facilitate the adoption of UoLs specified in IMS LD by instructors. Specifically, the proposed semi-automatic approach hides the complexity of the EML language, so instructors only have to decide which one of the recompiled processes fits better

[2] An UoL represents a variety of prescribed activities, assessments and services provided by teachers, in a course or lesson, which is the result of the learning design.

with the learning objectives of the UoL, in terms of structure and adaptive criteria. Henceforth, the main research question addressed in this paper is the automatic reconstruction of UoLs from scratch, as the state of the art heavily relies in the participation and feedback from all appropriate personnel and users during the whole process, hindering the reuse of UoLs in different platforms.

The remainder of this paper is structured as follows. Section 4.3 briefly introduces the main features of the IMS LD specification. Section 4.4 describes the different approaches that have already been proposed and that motivated our approach. Then, Section 4.5 presents the framework that supports the mining of log files and the reconstruction of IMS LD. Sections 4.6 and 4.7 detail, respectively, how the learning flow —through a process discovery algorithm— and the adaptive rules —through a decision tree algorithm— are mined from the logs. Then, Section 4.8 details the transformation from these two models to the actual target language (IMS LD). Section 4.9 shows the results and, finally Section 4.10 points out the conclusions and future work.

4.3 IMS Learning Design

IMS LD specification is a meta-data standard that describes all the elements of the design of a teaching-learning process [46]. This specification is based on: *i)* a well-founded *conceptual model* that describes the vocabulary and the functional relations between the concepts of the learning design; *ii)* an *information model* that details in natural language the semantics of every concept and relation introduced in the conceptual model; and *iii)* a *behavioral model* that specifies the constraints imposed to the software system when a given learning design is executed in run-time. In other words, the behavioral model defines the semantics during the execution phase. Furthermore, IMS LD defines three levels of implementation depending on whether the learning design is adaptive or not:

- *Level A*. This first level contains the main components of a UoL: participants (roles), pedagogical objectives, resources (services and contents), and learning design. This last component is understood as the coordination of the learning activities to be performed by the participants to achieve the pedagogical objectives, i.e the learning design describes the learning flow –or learning path– to be followed by learners in a UoL. To describe this learning design, the IMS LD specification follows a theater metaphor where there are a number of plays, that can be interpreted as the runscripts for the exe-

cution of the UoL and that are *concurrently* executed, being independent of each other. Each one of these plays is composed by a set of acts, which can be understood as a module or chapter in a course. Acts are performed in *sequence* and define the activities that participants must do. This model also allows the assignation of roles to the participants and partitioning the activities of an act according to those roles. In this case, each one of the partitions can run *in parallel*. Finally, activities can be simple or complex, the latter may consist of a *sequence* or *selection* of activities (simple or complex).

- *Level B*. This level adds *properties* and *conditions* to level A. It also adds monitoring services and global elements which allow users to create more complex structures. The properties store information about people (preferences, outcomes, roles, etc.), personal information, or even about the learning design itself. Level B also establishes *i)* the visibility of the elements of the learning flow; *ii)* if properties are transient or should persist across multiple sessions; and *iii)* the set of operators and expressions that may transform the value of properties and the visibility of elements. For instance, adaptation is usually based on the visibility of the activities of the learning flow, since IMS LD does not have control structures such an *if-then-else*. Therefore, the adaptation rules use properties, such as a test score, an answer to a specific exercise, and so on, to decide the learning path of the student through the visibility of the activities.

- *Level C*. The last level incorporates notifications to level B. Notifications fire automatically in response to events triggered in the learning process. For example, if a student submits a job, an email to report the event could be automatically sent to the teacher.

Taking this into account, the objective of this paper can be defined more precisely as recompiling the structure of the learning process defined at level *A* and the properties and adaptation rules at level *B* from event log files.

4.4 State of the Art

We have focused our analysis of the state of the art in the topics and fields that motivated our approach: *i)* IMS LD authoring tools; *ii)* the reconstruction of IMS LD; and *iii)* the applications of process mining in education.

For the last years, a number of IMS LD authoring tools have been developed. These solutions allow a better analysis of the related educational design approaches by trying to relieve

the complexity of this standard to teachers and instructors. ASK-LDT [66] provides a graphical interface that allows to hide the complexity of the IMS LD control structure and adaptive components. The authors define an abstract high-level architecture for designing pedagogical scenarios that can be reused in different virtual learning environments. In [59], the authors offer visual templates or patterns for creating learning designs based on pedagogical strategies such as collaborative learning or project based learning. Within this idea, they present Collage, a specialised high-level collaborative learning editor that guides teachers to create their own collaborative learning design from the existing patterns. However, this tool is not intended to edit templates, so it is restricted to the predefined set. Other tools such as ReCourse [53] or Prolix GLM [60] made a great effort by providing the user-friendly graphical notation, however they do not have wide support for level B, i.e they have difficulties providing a visual notation for level B properties and conditions [39]. Other tools, such as LAMS [28] and MOT+ [89], although they are not based on IMS LD, can handle this standard by making a translation to their internal EML representation. Nonetheless they can lack of flexibility —being difficult to interpret by non-experts— or face difficult importation/exportation issues in order to interoperate with IMS LD. In summary, although these approaches try to hide the complexity of IMS LD by using a more user-friendly notation, they share the same drawback: real users —teachers and instructors— do not have the technical skills needed for a practical use of these tools, as they still require to know about IMS LD to use the advanced features of this standard. In fact, this drawback works as a barrier for the practical use of IMS LD, as teachers are forced to transform their educational scenario into machine-oriented notation [98].

The automatic reconstruction of UoLs is a novel approach to hide the complexity of the authoring process of IMS LD. In [82], the authors present an approach that makes a conceptual distinction between exchange EMLs and authoring EMLs. In this approach, the authors provide graphical notations —authoring EMLs— to the teachers, which are more user-friendly than the raw XML format. Then, these notations are translated, via an exportation process. With this classification, they focus on the reengineering of IMS LD UoLs based on a visual language, which hides the complexity of the model. On the other hand, this approach also requires a close collaboration between developers and teachers to simplify the gap between the technical and pedagogical point of view of the UoL. In [2], and later in [1], the authors present a four-step approach for process reenginering in higher education. However, the reengineering process is not fully automatic, as it requires the participation and feedback from all appropri-

ate personnel and users. In summary, very valuable results have been achieved in this field, however, the main drawback of these approaches is that the UoL is not automatically reconstructed from the scratch and needs the supervision of teachers and even developers. From the point of view of mining the control flow of processes, a recent approach [29] tries to mine the adaptive rules of the log by using decision trees. In this approach, the authors define a framework for deriving and correlating process characteristics.

Process mining has emerged as a way to analyze the behavior of an organization based on event logs. Specifically, process mining focus is on concurrent processes, trying to discover the underlying control flow of the behavior recorded in a event log: *sequences*, *parallelisms*, *loops*, etc. On the other hand, a second important distinction concerns the unit of analysis, which can be variables or events. Most of the approaches found in the literature use events for automatically mining the processes; however, this second dimension depends directly on the information available in the log files. Taking these features into account, and although process mining from log files has been widely applied [36, 77, 117, 122], its application in education is a relatively new topic with some recent studies. In [67], the authors present PETRA, a system to extract new knowledge rules about transitions and learning activities in processes from previous platform executions. However, this tool is not oriented towards process reconstruction and discovery, but on process extension, i.e it requieres an already defined process model in order to enrich such process. In [63] a process mining approach in adult informal learners is presented. Informal learning situations often exhibit high variability within the learner population, especially when learning experiences and environments offer broad availability. However, this freedom of navigation has sometimes negative effects of learning experiences, particularly when prior domain knowledge or learning skills are weak. The experience is developed in two steps: firstly, a process discovery using hidden Markov models is performed [65], and then a process analysis is done using a standard process mining approach [125]. A similar approach is presented in [92] but for improving on the objectivity of the assessment as well as to provide the learners with prompt feedback. A set of software repositories are preprocessed with the FRASR tool [93], to produce event logs and then mine them using ProM [139]. The first step of this approach is particularly interesting and differs from previous proposals. The authors do not start the derivation from a log file but from a set of repositories that have different structures. Another process mining approach for providing students feedback is described in [90]. In this case, the objective is to discover a process from the behavior of the students during the course of a MCQ test. They analyzed the navigation

flow of each student when answering the questions, to determine what kind of feedback is more preferable and more effective for the students. Authors use the α-algorithm [137] for the process discovery. This three approaches are very similar. In fact they almost use the same process mining techniques. The main drawback is that the process discovery techniques that they use do not always guarantee feasible results.

To sum up, although a wide variety of approaches have tried to provide a IMS LD user-friendly notation by means of graphical tools, its complex specification, particularly the levels B and C, entails a barrier for adopting this standard by teachers and instructors. The automatic reconstruction of IMS LD UoLs could relieve this issue, but it is necessary to deep in the automation of this process from the scratch, avoiding the supervision by teachers and developers during the whole process, and thereby facilitating the reuse of UoLs in different platforms.

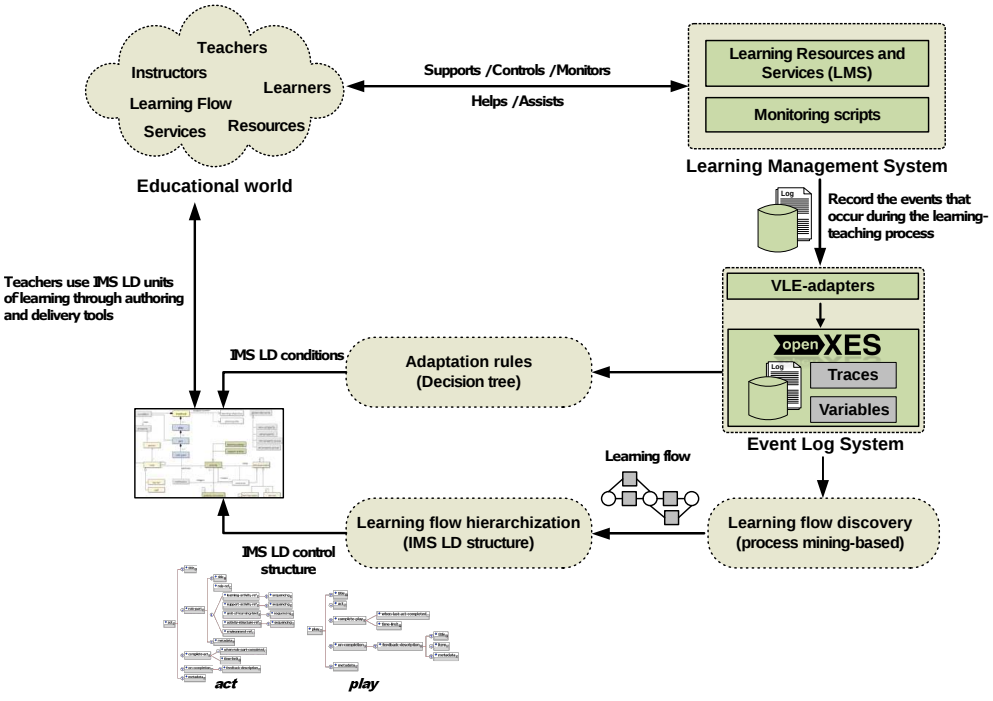


Figure 4.1: Framework for IMS LD UoLs reconstruction.

4.5 Framework

Figure 4.1 depicts the conceptual framework for reconstructing IMS LD UoLs from the events generated by learners. It shows all the main parts involved in the reconstruction of the IMS LD UoLs. The first component of this framework is the *educational world*. Teachers and learners are the typical participants in any learning activity. On the one hand, *teachers* design the learning flows based on some educational methodology, and support the learning activities of the course. On the other hand, *learners* are the core of the educational world since they undertake the learning flow activities by using the resources and services available in the virtual learning environment. The rest of the components are: the *virtual learning environment*, the *event log system*, the *learning flow discovery*, the *the learning flow hierarchization*, and the *adaption rules*.

- *Virtual Learning Environment*. From an educational point of view, virtual learning environments (VLEs) provide the means to carry out the learning activities planned for an UoL, allowing learners to access to the learning contents and executing the services required to facilitate those activities such as interacting with other learners or looking for information in libraries. Furthermore, VLEs detect and register all the relevant events generated by learners when undertake the learning activities. These events are stored in an event log database that contains data about the activities execution, including who participates in an activity, when it has been performed, the properties values of the UoL such as a mark of a test, and so forth.

- *Event Log System*. When a learner undertakes a learning activity, such as answering a test, uploading an exercise or even asking a question in the forum, the VLE stores in a database the information generated as result of performing this learning activity. However, in order to be able to extract the needed information for reconstructing the IMS LD, it is necessary to translate this information into an input format for both the process mining and the decision tree algorithm. As Figure 4.2 depicts, this translation is carried out by VLE-specific *adapters* that generate an event log register in a standard format for process mining, so-called *eXtensible Event Stream*[3] (XES) format [150]. XES is a well-known logging standard for process mining, and we have adapted it with user-defined extensions in order to include in the same file all the outstanding information to both obtain the learning flow —by storing the sequence of steps undertaken by

[3] www.xes-standard.org

each learner— and the adaptive rules —by storing the information generated as result of performing the learning activities. Figure 4.2 shows an example of the framework used to translate the event log from any VLE to an event log in XES format[4] through specific VLE-*adapters*. These adapters and the use of XES standard provide the independence of reconstructing the IMS LD with the particular VLE in which the learning-teaching process takes place, enabling furthermore to generalize the proposed architecture to any virtual environment. The only restriction required for the VLE is that it must register the user activity —along with the variable modifications— that takes place in the learning context.

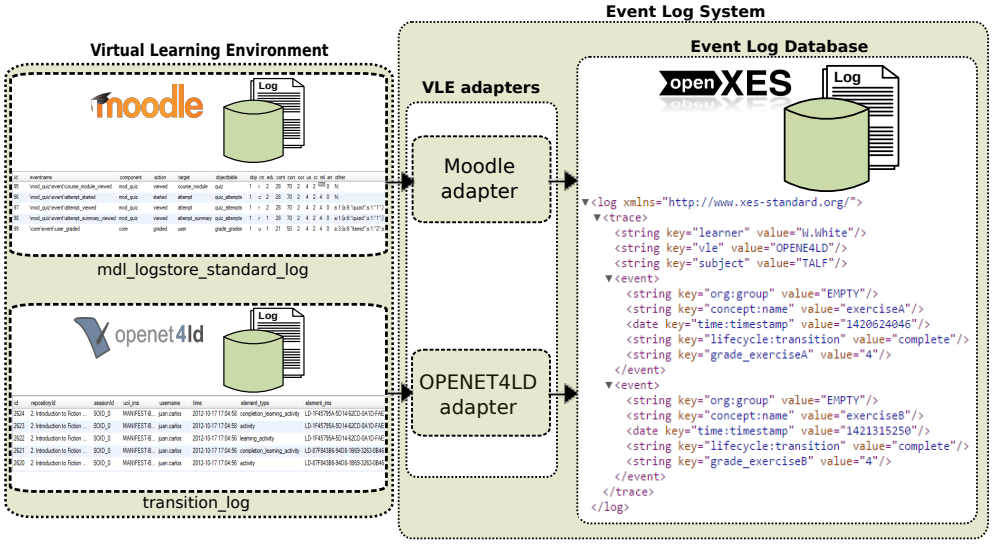


Figure 4.2: Event log infrastructure. Each VLE accesses the event log system through a specific adapter to transform its log format to XES, the standard format for process mining.

- *Learning Flow Discovery*. This component implements the algorithm whose aim is to discover the workflow of learning activities, i.e the learning flow, that learners undertake during a UoL. Note that this algorithm must guarantee that *all the learning paths* followed by learners are represented in the discovered learning flow. Furthermore, this

[4]Note that we have omitted some of the standard metadata of XES in the event log shown in Figure 4.2 to make the image more readable.

discovered learning flow should be as simple as possible in order to facilitate the hierarchization of the learning flow into the IMS LD control structure. This component is explained in detail in Section 4.6.

- *Learning Flow Hierarchization*. Once the learning flow has been extracted from the event logs, an algorithm will translate this learning flow to the control structure defined by the IMS LD specification. This algorithm starts by detecting sequences and selections of learning activities and then uses the knowledge about IMS LD to create activity structures, acts , and plays. The learning flow hierarchization is explained in detail in Section 4.8.

- *Adaptation Rules*. The IMS LD specification has an extensive set of adaptation conditions, but this framework is focused on extracting the conditions related to the selection of learning activities — show and hide mechanism — based on the changes in the properties values of the UoL. A decision tree technique was implemented to automatically obtain these conditions, since it is an effective approach to deal with this kind of problems. This process is explained in Section 4.7.

It is important to emphasize that, although we are depicting this framework for IMS LD UoLs reconstruction, the framework described can be applied to reconstruct any UoL to any particular target EML. Both the process mining step and the adaptive rules mining from variable values are totally independent from the target language. In fact, process mining retrieves graph-based structures independent from IMS LD. On the one hand, graphs are based on Petri nets, which is the most used formalism for modelling processes. On the other hand, a decision tree algorithm is used to get the adaptive rules that will guide the students along the UoL. The learned knowledge is represented through binary trees and, thus, not tied to the specificity of IMS LD rules.

The last step of this framework, i.e., the combination of these two models into the target language, is the only dependency between the reenginering process and the target EML. Nevertheless, the algebra that we are using —explained in Section 4.8— in the reconstruction of the IMS LD is not so different from any particular EMLs —*if-then-else*, *AND*, *OR*, *sums*, etc—. Hence, it is possible to modify the knowledge applied in the last step of the reeinginering process in order to reconstruct an UoL *from any particular VLE to any particular EML*.

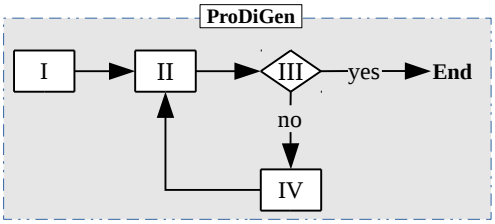
4.6 Mining the learning flow from event logs

The goal of process mining is to automatically obtain a process model that specifies the relations between activities from *concurrent* processes. Therefore, the aim of the process discovery algorithm in the reconstruction of the IMS LD is to identify the workflow that represents the learning flow followed by the learners during the UoL [11]. To achieve this objective, the process discovery algorithm only needs to consume the log in XES format —as this is the standard format for process mining— and it will retrieve a learning flow representing the behavior recorded in the event log. From the perspective of process mining [103], the quality of a learning flow is measured taking into account the following criteria:

- *Fitness replay*, which indicates how much of the behavior observed in the event log can be reproduced by the discovered learning path. A discovered learning path is considered complete when it can reproduce all the events contained in the log database. In order to guarantee a feasible and correct reconstruction of IMS LD UoLs, all the activities undertaken by the learners have to be included in the mined learning flow. Additionally, from the educational point of view, in order to guarantee feasible and correct evaluations of the learning paths, teachers need to access to all the activities performed by learners. Therefore, the fitness replay of discovered learning paths is a hard requirement.

- *Precision*, which measures if the discovered learning path allows an additional behavior that is not represented in the log, i.e., behaviour never undertaken by the students. Thus, a discovered learning path is considered as precise when it cannot reproduce events that are not available in the log database. From the point of view of the learners' evaluation, this kind of learning paths are desirable, but it is not a requirement as hard as the fitness replay: the additional learning paths are not needed for the reconstruction of the IMS LD control structure, since they did not happen. Our focus is on the exact behavior of the learners, i.e., *what they really did*, not in the prediction of their behavior; therefore we want to avoid overly-general models.

- *Simplicity*, which refers to discovered learning paths with the minimal structure that reflects the behavior contained in the log database. A desirable requirement for process discovery is to obtain simple learning flows, since the simpler is the discovered learning flow, the easier is to reconstruct the IMS LD control structure.



Step	Description
I	Build the initial population
II	Evaluate each solution
III	Stopping conditions fulfilled?
IV	Generate the new population

Figure 4.3: Simplification of the main steps of ProDiGen.

In general all process discovery algorithms make assumptions regarding the event log and, hence, the emphasis on these three different quality dimensions. For example, Heuristics Miner [153] usually retrieves models with high levels of precision, but lacking fitness replay and/or simplicity. Another example is the ILP miner [138] that guarantees a perfect replay fitness but resulting in very complex models. Inductive Miner [71], on the other hand, can guarantee a perfect replay fitness and high levels of simplicity, but the precision is low. Overall, a large amount of work has been done in this specific area by addressing different algorithms from different points of view. However, in this paper, and considering the three previous criteria and their importance when mining a learning process, we selected ProDiGen [147] as process mining algorithm.

ProDiGen is a genetic algorithm for process discovery that focuses its search towards solutions that replay all the behavior as possible, with *high levels* of precision and simplicity. In order to better understand how this algorithm works, Figure 4.3 shows a simplification of its mains steps: *i)* the initialization of the population; *ii)* the evaluation of the individuals; *iii)* the stopping criteria; and iv) the generation of the new population. The evolutionary cycle involves the II, III and IV steps, where the population evolves based on a fitness function. Finally, the algorithm stops whether it reaches a maximum number of generations, or it gets stuck on a local minimal solution for a certain number of times.

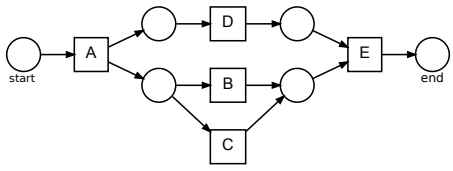
4.6.1 ProDiGen

The first step in any evolutionary algorithm is the initialization of the population. In this phase, a population is created with a group of individuals where each individual is a potential solution, e.g., a learning flow. In ProDiGen, each individual of the population codifies a learning path using a causal matrix representation, which can be easily translated into a Petri

net [30]. In terms of causality, both the Petri net and the causal matrix represent the same behavior —which learning activities enable the execution of other learning activities. Figure 4.4 shows an example of the mapping between a Petri net (Figure 4.4(a)) and its respective causal matrix (Figure 4.4(b)). This causal matrix has a row for each learning activity t in the log, and two columns corresponding to the inputs, $I(t)$, and outputs, $O(t)$, of the activity. The input and output sets are composed of several subsets, modelling the following relations:

- In the same subset:
  - Tasks in the same subset of the conditional function $O(t)$ have an OR-split relation.
  - Tasks in the same subset of the conditional function $I(t)$ have an OR-join relation.
- Between different subsets:
  - Tasks in different subsets of the conditional function $O(t)$ have an AND-split relation.
  - Tasks in different subsets of the conditional function $I(t)$ have an AND-join relation.

Once the initial population is created through heuristics based on the local information of the log, the individuals —causal matrices— of the population are modified through crossover and mutation operations to create new potential solutions, i.e., learning flows. These operators



(a) Example of a Petri net.

Task	I(Task)	O(Task)
A	{}	{{D},{C B}}
B	{{A}}	{{E}}
C	{{A}}	{{E}}
E	{{D},{B,C}}	{}
D	{{A}}	{{E}}

(b) Causal matrix of the Petri net.

Figure 4.4: Mapping of a Petri net into a causal matrix. This Petri net represents the learning flow of an UoL about polymorphism. The name of the activities are: Read about polymorphism (A), Exercise 1 (B), Exercise 2 (C), Answer test (D) and Exam (E).

add and/or remove[5] causal dependencies of the individual by modifying the input and output conditional functions of the tasks. Then, these new individuals are evaluated based on a fitness function.

In ProDiGen, the quality of an individual is measured taking into account fitness replay, precision, and simplicity, as ordering criteria, respectively. Hence, when contrasting two possible solutions, fitness replay is the primary ordering criteria. When two solutions have the same fitness replay, precision is used to decide the best solution. Finally, when two solutions have the same fitness replay and precision, simplicity is the decisive criterion.

Thus, when two solutions have to be compared, the individual with highest fitness replay will be the benefited one. If the values for the fitness replay are equal for both solutions, the second parameter to be checked is the precision, and, if the same as before occurs, the last one to be compared is the simplicity. Note that if we change the hierarchical order of the fitness measure, the algorithm may find a different solution, as fitness replay, precision and simplicity are three opposed objectives [147].

4.7 Mining adaptive rules from event logs

Using log files of VLEs can help to determine who has been active in the course, what they did, and when they did it. In this section we use these data to obtain the adaptive rules of the UoLs that determine the learning flow and the contents and services presented to students. Unfortunately, the log files provided by VLEs do not follow a standard and therefore a generic solution cannot be formulated for such purpose. In fact, log files are seldom used mainly because it is difficult to interpret and exploit them. In most of the cases, the data aggregated are incomplete or even not logged.

Taking this into account, in this section we describe our approach based on the extract of the log file represented in Figure 4.5. This log was recorded in OPENET4LD [151], and exemplifies the typical elements saved by VLEs. Of course, the records formats or even how they are stored may vary, e.g., Moodle saves this information in tables of a relational database—, but usually VLEs present very similar information. This particular example has two parts. On the one hand, the first four records represent the execution of activities and contain information about the UoL id, its name, the user that performed the activity, the execution time, and the specific name of the activity, among other information. On the other

[5]For example, the mutation operation can add a new subset into the output function of a task t, resulting in an AND-split.

(3, 'UOLID_201', 'GeoQuiz3', 'root', '2013-10-14 01:44:51', '13193839-3ddf-41e2-a4b8-7af8dc13d059'),
(4, 'UOLID_201', 'GeoQuiz3', 'root', '2013-10-14 01:44:52', 'e4901d0f-8232-4cce-a166-20e29133d279'),
(5, 'UOLID_201', 'GeoQuiz3', 'root', '2013-10-14 01:44:52', 'e4901d0f-8232-4cce-a166-20e29133d279'),
(6, 'UOLID_201', 'GeoQuiz3', 'd9', '2013-10-14 01:44:52', 'e4901d0f-8232-4cce-a166-20e29133d279'),
...
(289, 'OPENET_RMI_SOID_0', 'UOLID_201', 'setproperty', '2013-10-14 01:59:30', 'user_2', 'd2', 'Student', 'locpers_property_12', 'Answer1', 'locpers_property', 'string', 'Venezuela'),
(290, 'OPENET_RMI_SOID_0', 'UOLID_201', 'change_property_value_0', '2013-10-14 01:59:30', 'user_2', 'd2', 'Student', 'locpers_property_1', 'Value1', 'locpers_property', 'integer', '2'),
(291, 'OPENET_RMI_SOID_0', 'UOLID_201', 'change_visibility', '2013-10-14 01:59:30', 'user_2', 'd2', 'Student', 'learning_activity_5', 'flow3', 'locpers_property', 'boolean', 'false'),
(292, 'OPENET_RMI_SOID_0', 'UOLID_201', 'setproperty', '2013-10-14 01:59:33', 'user_2', 'd2', 'Student', 'locpers_property_13', 'Answer2', 'locpers_property', 'string', 'Siria'), ...

Figure 4.5: Example of a text-based log file.

hand, the last four records represent events on the properties used by the UoL. Specifically, each one of these records contains information about the UoL id, the operator that favor the change, the type of the property, and the new assigned value.

4.7.1 Identification of variables and activities

The identification of variables and activities is highly dependent on the type of events recorded in the log files. Since each VLE may record the events in a different format, the syntactic patterns used to identify these events are usually different. For example, the identified variables are highlighted in Figure 4.5. In this case, a simple regular expression with the keywords "setproperty" and "change_visibility" were used to identify the variables, but a different pattern should be used to identify these same variables in an XML-based log file. However, the main issue in the identification of variables is that they are not always recorded in the log files. When this situation happens, the mechanism for learning the adaptation rules presented in this section cannot be applied.

4.7.2 Determining the variable values for each activity

In the ideal situation, each time an event is produced, *i)* the state of the variables is saved in a log file. This means that, e.g., at the end of an activity the values of the variables of a UoL are stored in the corresponding log file. Moreover, *ii)* in this context a value change is also

considered an event and so is also recorded. However, reality is different and usually both mechanisms are not supported at the same time —e.g., in the log extract of Figure 4.5 only variable changes are included. The variables values are subsequently associated to an activity, so we can determine the state of the properties before and after the activity is performed. Therefore, each time a variable value changes we must determine when and by who it was modified. In this procedure, the time of the event is crucial since it will determine the initial value of variables in the next activity.

4.7.3 Learning rules with a decision tree

IMS LD rules use the following grammar to define the learning flow adaptation:

```
rule ::= IF <expression> THEN <action>
       | IF <expression> THEN <action> ELSE <action>
       | IF <expression> THEN <action> ELSE <rule>
```

In addition, IMS LD declares three types of actions:

```
action ::= SHOW <activity_id>
       | HIDE <activity_id>
       | <property_id> = <value>
```

The first and second actions are used to make visible or invisible a learning/support activity, activity structure, play, item, or environment. It's the main adaptation mechanism of IMS LD used to hide or show a part of the UoL based on some expression value. The last type of expression is a simple assignment.

IMS LD grammar for expressions include logical operators, *and*, *or*, and *not*, some comparative operators, $\leq, <, =, \neq . >$, and $\geq$, and some multiplicative and additive operators. Moreover, it also defines operators to check specifics of UoLs. For instance, to verify if a play, act, or activity has already finished, or, e.g., the specific role of users.

In this paper, the identification of the adaptive rules is performed by means of the J48 decision tree algorithm [95]. We selected this type of algorithm because of its simplicity, performance, and especially because adaptive rules can be transformed to a decision tree. As previously mentioned, IMS LD adaptive rules have an *if-then-else* structure very similar to the right part of Figure 4.6, which can be deduced from the graph structure returned by a decision tree algorithm. A decision tree is a graph-like structure in which each internal node

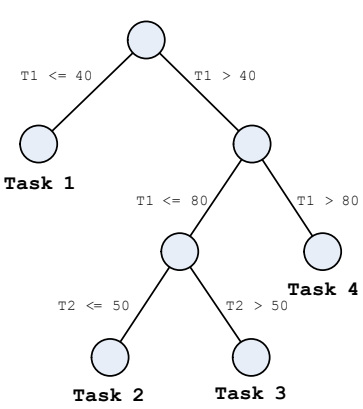
represents a test on an attribute, each branch represents the outcome of the test and each leaf node represents the class label —or decision taken after computing all attributes. A path from the root to the leaf represents classification rules, and in our case, an adaptive rule.

Transforming a decision tree to an adaptive rule is straightforward since these trees can easily be modelled as DNF (Disjunctive Normal Form) rules. For instance, the right part of Figure 4.6 shows the corresponding rules for this tree. Specifically, each branch is converted into a rule, where:

- The condition is set as the conjunction of the arc tests of a branch.
- The action of the rule is the leaf node of the branch —class/decision.
- The disjunction of all the rules has the semantics of the decision tree.

Notice that the rules extracted by the J48 support most of the grammar of IMS LD —logical and comparative operators—, although this is not the main objective here. In fact, we selected decision trees because they provide a simple but also generic grammar, that should be compatible with most of the adaptation mechanisms used in legacy systems. Therefore, it does not make sense to learn, e.g., IMS LD specific operators. There are however some limitations since we do not cover complex conditions that combine mathematical operators.

Tree complexity has its effect on the accuracy and is usually determined by the total number of nodes, total number of leaves, depth of tree, and number of attributes used in the tree




```
IF T1 <= 40
THEN visible(Task 1)

IF T1 > 40 AND T1 <= 80 AND T2 <= 50
THEN visible(Task 2)

IF T1 > 40 AND T1 <= 80 AND T2 > 50
THEN visible(Task 3)

IF T1 > 40 AND T1 > 80
THEN visible(Task 4)
```

Figure 4.6: Transformation of a decision tree to a DNF rule base.

construction. The number of variables is therefore a crucial factor since too many of them may reduce the accuracy of the tree. Moreover, the size of the data required to learn increases with the number of variables. However, adaptive rules are usually not based on many variables. Thus, our approach limits the variables considered during the learning of the decision tree, and specifically, only variables for which the value has changed in the execution of the activity are included.

4.8 IMS LD reengineering

IMS LD is a well-known EML for adaptive learning which is specified in three different levels of implementation —levels A, B, and C—, which determine the learning flow, the changes in the environment, and the notifications, respectively. In this section we describe an algorithm that compiles IMS LD levels A and B from the information retrieved by the data mining algorithms described in the former sections. On a first step, the proposed algorithm transforms the flat process structure retrieved by the process mining into an IMS LD-based structure. On the second step, the identified adaptation rules are associated to the learning flow.

4.8.1 Pairing the causal matrix with the IMS LD specification

IMS LD learning flow is described as a theatre metaphor where there is a number of plays that are *concurrently* performed, being independent of each other. Each of these plays is composed of a set of acts, which represent, for instance, the modules or chapters of a course. Acts are performed in *sequence* and define the activities that participants must do. This model also allows the assignation of roles to the participants and partitioning the activities of an act according to that roles. In this case, each one of the partitions can run *in parallel*. Finally, activities can be simple or complex, the latter may consist of a *sequence* or *selection* of activities (simple or complex).

Taking this structure into account, the reengineering process would consist in identifying the different IMS LD elements from the causal matrix (Petri net) returned by the process mining algorithm. However, this Petri net is a flatten process while the IMS LD is a tree-based structure in which each layer is composed by a different type of elements, i.e first plays, then acts, role-parts, and finally activities. Bearing in mind that the Petri net only identifies the atomic activities, we decided to structure the search space as a tree in which:

- Each node of the tree represents how learning activities are grouped. Suppose an ordered list of the n activities that must be performed. This list can be divided in $n-1$ parts —one for each consecutive activity. Taking this into account, each node is identified by $n-1$ digits, where a 1 in the position i indicates that the activities in the positions $i-1$ and i are in different groups.

- Each layer of the tree corresponds to a layer of IMS LD. Specifically, the first layer represents plays, the second acts, the third role-parts, and the remaining layers activities, simple or complex.

- An edge between a father node and a child node indicates that this child is grouped according to the configuration defined in the father node. Thus, the child node may add additional groups but they must respect the groups defined in the father node.

Figure 4.7 depicts the search space for a UoL composed of just tree simple activities, namely A, B, and C, ordered according to their position in the causal matrix. Let use the notation $[A\,B\,C]$ to facilitate the definition of the list and use the symbol $|$ to indicate a partition. Taking into account our previous definition of the search space, in this example, nodes can be identified by two digits where: 00 indicates that the three activities are in the same group —$[A\,B\,C]$; 01 that there are two groups, a first one with the activities A and B, and a second one with C —$[A\,B|C]$; 10 also represents a two groups be in this case the activities B and C compose the second group —$[A|B\,C]$; and finally, 11 implies that each activity is in a different group —$[A|B|C]$.

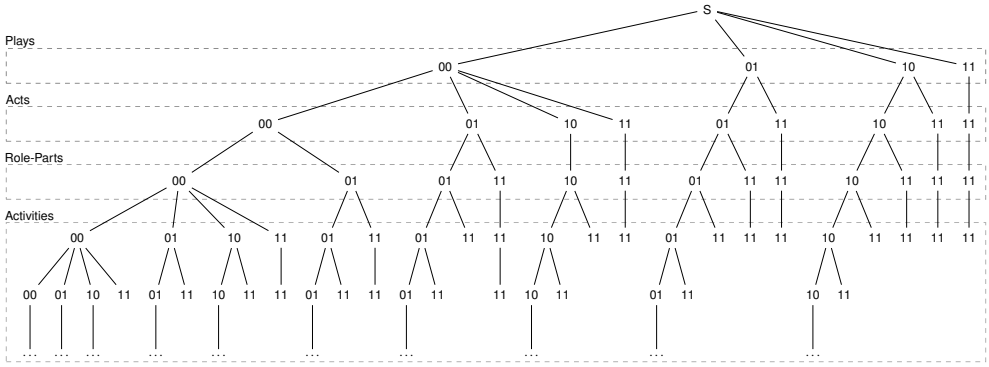


Figure 4.7: Example of the search space for a UoL composed of three activities.

Notice that the meaning of the coding is different for each level of the search tree. For instance, a node identified by 10 at the play level indicates that the activity A is in the first play, while the activities B and C are in the second play. Thus, the coding defines both the number of plays and how activities are grouped in the plays. If the node identified by 10 is at the act level, the node still identifies two groups of activities but in this case for the two acts defined in this coding.

In addition, child nodes must preserve the groups already defined in the father node. Let A and B be two nodes, where A is father node of B in the search tree, and $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_2^n$ denote the coding of size n of A and B, respectively. For each $a_i = 1 \in \mathbf{a}$ then $b_i = 1 \in \mathbf{b}$, $i = 1 \ldots n$, i.e each position i equals to 1 in the father node is also equal to 1 in the child node. This definition ensures the hierarchical disjointness of groups, that is, that two activities that are in different groups cannot be joined in the same group in a lower level of the tree. This constraint also reduces the search space since many father/child combinations are not allowed. For instance, a node 01 can only have two children, denoted as 01 and 11, since the father node already defines two groups —$[A\ B|C]$. Suppose that 10 is a valid child node of 01, this would imply that the activities B and C are in two disjoint groups in the father node but in the same group in the child node, which is clearly inconsistent with the hierarchical definition of the IMS LD specification.

Algorithm 4.1 details the main block of the depth-first search procedure. The algorithm receives two inputs: n_i which contains the node that is being evaluated, and the *verifiers* list which elements are the functions used to check the structural consistency of the node. Specifically, each function uses the causal matrix obtained during the process mining step to check if the activities are correctly grouped, that is, *i)* if they verify the dependencies of the causal matrix and *ii)* the structure is compliant with IMS LD. Since the structural requirements of each level of the tree are different, the *verifiers* list contains a specific verifier for plays, acts, role-parts, and activities. For the sake of simplicity, the position i of the *verifiers* list contains the function to check the level $i+1$ of the search tree. Since there are only 4 types of verifiers, nodes which level is greater than 3 will be checked by the last function of the list —i.e., as activities.

In the first line, the algorithm used the level of the node in the tree to select the corresponding verifier, and uses this function to check the node. This is the most complex part of th algorithm and consists in looking for specific structural patterns in the causal matrix. As we will detail in this section, each IMS LD control construct has structural constraints that are

Algorithm 4.1: explore.

Input: n_i is the node to explore, *verify* is a list of functions used to check the structure of the nodes.

Output: from n_p hangs a (sub)tree of possible UoL configurations that can be derived from the causal matrix.

```
1  verify ← function from verifiers used to check the nodes of the level in which (n_i) is
     situated
2  if verify(n_i) then
3      if level(n_i) < MAX and not solution(n_i) then
4          foreach j in expand(n_i) do
5              n_j ← new node which value is j
6              add n_j as children of n_i
7              explore(n_j, verify)
8  else
9      if n_i has a parent then
10         remove n_i from parent(n_i)
```

synthesized in one of these functions. When n_i is a solution (i.e., if it is a simple activity) or the *MAX* level limit has been reached, then the algorithm backtracks. Otherwise, the node is expanded (lines 4-7) and a new exploration starts for the children of n_i.

4.8.2 Consistency of plays

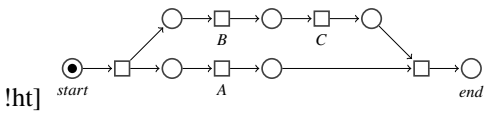
The verifier of the first level of the search tree checks if the plays are grouped in conformance with the IMS LD specification. Specifically, the objective of Algorithm 4.2 is to verify that the activities contained in each group are in parallel since each group identifies a play. Therefore, in lines 2-11, each activity a_j of the group g_i is compared with a different activity a_l of a different group g_k, until all the activities of the different groups have been compared. In order to check if two activities are in parallel we use the causal matrix M obtained during the process mining step. In this case, we just check *i)* that a_j is not connected to a_l, *ii)* that a_l is not connected to a_j, and *iii)* that $a_j \neq a_l$.

Let suppose the Petri net depicted in Figure 4.8 and its corresponding causal matrix, and the previously described search space depicted in Figure 4.7. A correct ordering of the activities of the causal matrix would return $[A\ B\ C]$. Taking this into account, Algorithm 4.2 would process the nodes of the first level and determine that nodes 01 and 11 are not in conformity

Algorithm 4.2: plays.

Input: n is the node to evaluate
Data: M is the causal matrix obtained during the process mining procedure
Output: A boolean that indicates if the node verifies the constraints of IMS LD for plays.

```
1  g ← the groups defined in n
2  for i = 0 to length(g) − 1 do
3      g_i ← g[i]
4      for j = 0 to length(g_i) do
5          a_j ← g_i[j]
6          for k = k + 1 to length(g) do
7              g_k ← g[k]
8              for l = 0 to length(g_k) do
9                  a_l ← g_k[l]
10                 if a_j and a_l are not in parallel in M then
11                     return false
```



(a) Petri net.

Task	I(Task)	O(Task)
A	{}	{}
B	{}	{{C}}
C	{{B}}	{}

(b) Causal matrix.

Figure 4.8: Petri net with three activities and its causal matrix.

with the causal matrix. Specifically, in line 10 it would verify for both cases that task B is not in parallel with C.

4.8.3 Consistency of acts

Algorithm 4.3 checks the structural consistency at the level of acts. Specifically, this function is used to verify that the groups defined at this level are in conformity *i)* with the IMS LD specification and *ii)* with the causal matrix obtained from the process mining. In this case, IMS LD requires that the acts of a play must be in sequence. Therefore, the algorithm has an external loop that iterates the plays identified as consistent, and for each one of these plays it checks *i)* that the activities of an act are isolated from other acts (lines 6-8), *ii)* that last

Algorithm 4.3: acts.

Input: n is the node to evaluate
Data: M is the causal matrix obtained during the process mining procedure
Output: A boolean that indicates if the node verifies the constraints of IMS LD for acts.

```
1  foreach group k of the parent node of n do
2      g ← the groups defined in n that are contained in the father group k
3      for i = 0 to length(g) do
4          g_i ← g[i]
5          S_o ← set of final activities in g_i
6          foreach activity a in g_i \ S_o do
7              if a is connected to an activity not in g_i then
8                  return false
9          if i + 1 < length(g) then
10             g_j ← g[i + 1]
11             S_i ← set of initial activities in g_j
12             if there is a shared choice between the activities if S_o and S_i then
13                 return false
14             foreach activity a_o in S_o do
15                 foreach activity a_i in S_i do
16                     if a_i and a_o are not connected then
17                         return false
```

activities of an act and the next ones are not in conflict because of a choice node (lines 12-13), and *iii)* that the acts are connected (lines 14-15), i.e that the last activities of an act are connected to the first activities of the next act.

Continuing with the example depicted in Figure 4.8, let suppose that we want to evaluate the node 01 which father play node is also 01. In the two iterations of the first loop, $g_0 \leftarrow [A]$ and $g_1 \leftarrow [B\,C]$ will both have $S_o \leftarrow \{\}$ and $S_i \leftarrow \{\}$ and thus verify the conditions of the acts' level. However, let consider that the node to evaluate is 01 and its parent play node is 00. For the case $g_0 \leftarrow [A\,B]$ the algorithm will fail in line 7 since B is connected to an activity that is not included in g_0.

4.8.4 Consistency of role-parts

In order to check that role-parts are correctly defined we must verify that:

- All the activities included in the role-part have the same role.
- The groups identified in the role part are in parallel.

Since the conditions are similar to those imposed to plays, we will not detail the algorithm used to verify the consistency of role-parts. For instance, the node 10 pointed by the parent node 00 is a valid configuration since the activity *A* is in parallel with the activities *B* and *C*. As aforementioned, all configurations that are valid at the play level are also valid at this level, but, in addition, nodes must also be compatible with its parent node. Therefore, if the parent node is 11, the node 10 would no longer be a valid configuration.

4.8.5 Consistency of activities

In IMS LD, activities can be simple or structured as sequences or selection of activities. Taking this into account, Algorithm 4.4 checks that all the groups verify one of these categories. Specifically, the function *activity* only checks that the group has q unique element. The function *sequence* verifies that the elements of the group are in sequence, in the same way as Algorithm 4.3 checks that all the acts of a play are in sequence. Finally, the function *selection* checks the last activity structure of IMS LD. Notice that selections are a complex structure since they combine two patterns, such as a choice between several activities (simple or complex), and a loop, so students may select the same activity more than once. The main issue here is that process mining algorithms are not able to detect this pattern as it is modelled in

Algorithm 4.4: activities.

Input: n is the node to evaluate
Data: M is the causal matrix obtained during the process mining procedure
Output: A boolean that indicates if the node verifies the UoL constraints of IMS LD.

```
1 foreach group k of the parent node of n do
2     g ← the groups defined in n that are contained in the father group k
3     foreach group g_i in g do
4         if not activity(g_i) or not sequence(g_i) or not selection(g_i) then
5             return false
```

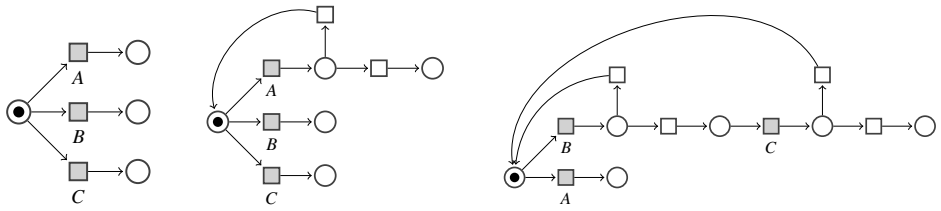
Algorithm 4.5: Selection.

Input: g is the group of activities to evaluate
Data: M is the causal matrix obtained during the process mining procedure
Output: A boolean that indicates if the group verifies the selection structure of IMS LD.

```
1 if length(g) < 2 then
2     return false
3 foreach activity a_i in g do
4     if a_i is not in a choice with another activity in g then
5         return false
6 if father node is a selection then
7     return false
```

IMS LD [151]. Instead they usually detect partial choices and combine them with multiple loops. In order to identify this pattern from the logs we just need to detect if the activities are in a choice with any other activity in the selection since the loops spread the choice dependencies, lines 3-5 of Algorithm 4.5. Finally, we do not allow the concatenation of selections (lines 6-7) since they do not change the behaviour of the net and just complicate the design: a subselection can be moved to a father selection without changing the behaviour of the control construct.

For instance, Figure 4.9 represents different ways in which process mining may retrieve



(a) Choice of activities.

(b) Selection where the activities are in choice and in a loop.

(c) Selection where one of the branches is a sequence of two activities with two loops.

Figure 4.9: Three different ways of representing a selection. Transitions colored in gray represent the activities of the UoL.

a selection of three activities. Figure 4.9(a) depicts the simplest selection pattern in which the three activities complete for the tokens of their input place and thus only one of them can be chosen. In Figure 4.9(b), the first branch also includes a loop, and thus students can select more than one activity. Finally, Figure 4.9(c) combines a choice, a loop, and a sequence structure. Notice, that our algorithm detects both the three cases since activities *A*, *B*, and *C* share the same input.

4.9 Results

The validation of the presented approach has been done with a set of UoLs with different degrees of complexity. In a first batch, four UoLs of the Degree of Computer Science, at the University of Santiago de Compostela, were performed in the real environment of OPENET4LD [151] by students. To complete the dataset, we selected another five UoLs, collected by the Open University in the Netherlands from different European projects[6], and simulated their behaviour in OPENET4LD environment but with virtual students.

All UoLs were generated in IMS LD, half of the units were performed in a real environment by students while the other half was simulated, and logs were recorded by OPENET4LD[7]. The objective of this experiment is to recompile the UoLs in IMS LD format and minimize information loss. Table 4.1 lists the nine UoLs tested, on the basis of their activities, structures of activity, acts, plays, and properties, ranging from UoLs with one only act to UoLs with several acts and different activity structures:

- *AutomatonClass* is an adaptative UoL about Automata Theory and Formal Languages.
- *TALF* is another UoL on the topic of Automata Theory and Formal Languages.
- *Boeing* is an UoL about the safety regulations when removing certain parts of a Boeing engine (simulated).
- *Cam* specifies the activities and exercises needed to complete a lecture (simulated).
- *Driving* specifies the process of a driving school (simulated).
- *Programming* is about learning imperative programming.

[6] `http://dspace.ou.nl/handle/1820/16/`
[7] Logs and Petri nets available at `http://tec.citius.usc.es/SoftLearn/Compiling.html`

Table 4.1: Structural features of the UoLs that have been used in the experiment.

							Activity structures			
UoL	**#LA**	**#ST**	**#AC**	**#PL**	**#RU**	**#PR**	**Sequence**	**Choice**	**Parallelism**	**Loop**
PeerReview	14	7	2	1	10	15	✓	✓	✓	
Cam	10	2	3	1	7	13	✓	✓	✓	
POO	9	3	6	1	6	12	✓	✓	✓	✓
Driving	7	3	5	1	3	5	✓	✓		
Boeing	10	5	4	1	9	12	✓	✓	✓	
AddWork	7	2	3	1	3	8	✓	✓		
Automaton	9	2	5	1	7	9	✓	✓	✓	
TALF	11	5	7	1	8	9	✓	✓	✓	
Programming	4	1	2	1	2	4	✓	✓	✓	

#LA, #ST, #AC, #PL, #RU, and #PR stands for the number of activities, structures of activity, acts, plays, rules and properties, respectively.

- *POO* is a UoL about learning object oriented programming.
- *PeerReview* explains the interaction in a peer review process (simulated).
- *AddWork* represents a quiz with different outcomes depending on the results (simulated).

Summarizing, four of the UoLs belong to courses of the Computer Science degree, while the remaining UoLs were taken from tutorials, seminars, and workshops previously recorded in OPENET4LD.

4.9.1 Process mining results

In order to check the efficiency of ProDiGen for learning flows discovery, we have conducted an experiment with the nine UoLs shown in Table 4.1 that have been undertaken in the OPENET4LD environment. OPENET4LD collects all the events generated by the learners when they perform the learning activities of an UoL, and then, with the OPENET4LD Adapter, we transform this behavior into a XES log. Although some of these UoLs are designed to be undertaken by several roles collaborating among them, ProDiGen discovers the learning flow related to each role. Table 4.2 shows the number of instances —traces— of each example. As we can see, only *Drive* and *Programming* UoL have less than 100 instances, i.e students that have participated in the courses. The remaining units are in between 120 and 190 instances,

Table 4.2: Number of instances of each UoL.

	PeerReview	Cam	POO	Driving	Boeing	AddWork	Automaton	TALF	Programming
# instances	186	186	190	97	134	120	120	144	42

Table 4.3: Performance of the genetic algorithm for learning flow discovery.

	UoLs								
	PeerReview	**Cam**	**POO**	**Driving**	**Boeing**	**AddWork**	**Automaton**	**TALF**	**Programming**
F_r	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
P	0.85	0.85	0.90	1.00	0.95	0.82	0.90	1.00	0.86
$(\#A,\#P,\#T)$	(15,7,6)	(38,13,13)	(36,12,18)	(26,10,11)	(24,10,10)	(36,15,16)	(34,12,17)	(34,12,17)	(37,13,16)

which is enough to perform with a high degree of confidence the mining process —for both the models and the rules (Section 4.9.2).

The performance of ProDiGen over the UoLs has been measured taking into account completeness —fitness replay—, precision and simplicity with two metrics of the state of the art. For completeness, we use the proper completion measure [105], which is the fraction of properly completed process instances. Proper completion (C) takes a value of 1 if the mined model can process all the traces without having missing tokens or tokens left behind. On the other hand, to measure the simplicity, we used the alignment precision (P) defined in [124]. For the simplicity, we show the number of arcs (#A), places (#P) and transitions (#T) of the retrieved Petri net.

Table 4.3 shows the results of ProDiGen over the 8 UoLs. As can be seen, ProDiGen is able to retrieve, for all the cases, a model that perfectly fits the behavior of the recorded event logs, i.e the fitness replay is equal to one in all the UoLs. Moreover the precision of the mined models is high, which indicates that the models do not underfit the log, i.e., they do not show much more behavior than the actually observed in the logs. Related to these precision levels, in none of the cases the retrieved model is the overfitted *trace model*, a solution that creates a path for each trace of the log, which can be seen with the simplicity measure —the number of actual transitions and arcs.

4.9.2 Rules mining results

A ten fold cross-validation has been performed for each one of the rules of each UoL detailed in Table 4.1. Table 4.4 details the results of the adaptive rules mining. We can see that, in

Table 4.4: Performance of the adaptive rules mining, where the percentage shows the instances that have been correctly classified by the rule learned by the decision tree.

Rule id	Correctly
PeerReview	
1	100%
2	86.02%
3	100%
4	98.39%
5	88.07%
6	95.70%
7	100%
8	53.22%
9	100%
10	95.70%
Driving	
1	97.93%
2	90.72%
3	86.60%

Rule id	Correctly
Cam	
1	100%
2	100%
3	98.82%
4	97.93%
POO	
1	100%
2	99.47%
3	100%
4	100%
5	100%
Programming	
1	100%
2	100%

Rule id	Correctly
Boeing	
1	89.55%
2	74.62%
3	95.52%
4	79.10%
5	72.39%
6	88.06%
TALF	
1	100%
2	100%
3	100%
4	100%
5	100%
6	100%
7	86.81%

Rule id	Correctly
AddWork	
1	100%
2	100%
Automaton	
1	99.17%
2	100%
3	100%
4	100%
5	100%
6	100%
7	100%

54.54% of the cases, the rule did classify correctly the 100% of the instances. In fact, the results are quite good considering that in 77.27% of the cases at least 90% of instances were correctly categorized, percentage that even grows to 90.91% if we consider the cut level of 80%. Only four cases obtain less positive results. Specifically, three of them are in the *Boeing* UoL and one in the *PeerReview* UoL. For instance, one of the rules of the *Boeing* UoL in this situation is defined as follows:

```
visible_Test_components = false: false
visible_Test_components = true
|   visible_Test_hazard = false: true
|   visible_Test_hazard = true
|   |   Lessons_components_counter <= 1
|   |   |   visible_Extra_Lessons_hazards = false
|   |   |   visible_Extra_Lessons_hazards = true
|   |   Lessons_components_counter > 1: false
```

As we can see, this rule uses four different variables to correctly classify 72.39% of the instances. In this and the other cases in which the learning process obtained weaker results, the number of examples were clearly insufficient to get the convergence of the algorithm. How-

Table 4.5: Results of the reengineering process.

	PeerReview	Cam	POO	Driving	Boeing	AddWork	Automaton	TALF	Programming
# solutions	2	12	9	90	2	394	44	55	2
Time (ms)	184531	10960	29476	603	1162	2273	7003	65335	144

ever, it would be misleading to point out that the number of variables is the unique factor, since most of the other units have rules with a similar number of variables. In these cases, we concluded that the complexity of the rule has more influence and would require a greater number of examples to obtain better results.

Finally, it should be mentioned that learning these rules is usually a difficult task, since the criteria used for the adaptation are not always clear. For instance, in many cases the instructor does manually the adaptation and not always with a uniform criteria. However, in these cases rules mining may achieve a secondary objective since it will precisely clarify these criteria from the event logs.

4.9.3 Reenginering results

Table 4.5 shows the results of the reengineering part of our proposal. As we can see, the processes were recompiled successfully for all UoLs. It should be noted the huge number of solutions that can be derived from the causal matrix in some of the cases. For instance, there are 394 different possibilities to structure a valid UoL from *AddWork* logs. There is a simple explanation: IMS LD is mainly structured with parallels and sequences. Plays and role-parts are parallel structures, while acts and sequences of activities are an ordered list of elements. Therefore, the number of possible combinations grows as the number of parallels and sequences increases. Notice that there would be even more solutions if we did not limit the concatenation of selections between a father and a child node.

The time required to obtain all the results depends on two factors. On the one hand, the net complexity, i.e the greater is the number of structures, the more combinations. On the other hand, the structures of activities play an important role in computational time since they increase the depth of the search tree. Therefore, it is not surprising that *PeerReview* and *TALF* UoLs need more time since the have more structural elements and activities.

It should be noted that, from a practical viewpoint, it does not make sense to show all the

reengineering results to an instructor. For instance, the selection of the most suitable structure from the 394 solutions returned for the AddWork UoL would be unmanageable for an instructor without a proper way to filter the results. For this reason, solutions are ordered according to criteria identified in [37]. In this study, participants were asked to transform a given textual design description into an IMS LD UoL. The analyses identified a number of conceptual structures which presented challenges to teachers' understanding, e.g., the management of role-parts. Specifically, in our proposal we use the following ordering criterion:

- *Simplicity*. UoLs with fewer structural elements take precedence.
- *Plays precedence*. Plays and role-parts are used to define parallel structures within a UoL, but plays have precedence over role-parts.
- *Acts precedence*. Acts and activity-structures are used to define sequences of activities within a UoL, but acts have precedence over activity-structures.

4.10 Conclusions and Future Work

In this paper we have proposed a global approach that facilitates the reuse of UoLs defined in legacy systems or VLEs. Our solution has been implemented to support the mining of event log files (i) to obtain the learning flow, formalized as a Petri net, and performed by students and instructors, and (ii) to identify the adaptation rules, represented as decision trees, used to adapt the learning flow to each student. An important feature of the described approach is its independence from any target EML, although in this paper the reengineering part of the framework is tied to IMS LD. However, it is sufficient to define the translation, from the Petri net models to the specific process representation, and from the decision tree to the rules grammar used by the EML, to have a compete specification of the UoL. In fact, as future work we plan to use this framework to recompile the same event logs to a different EML, such as ADL SCORM.

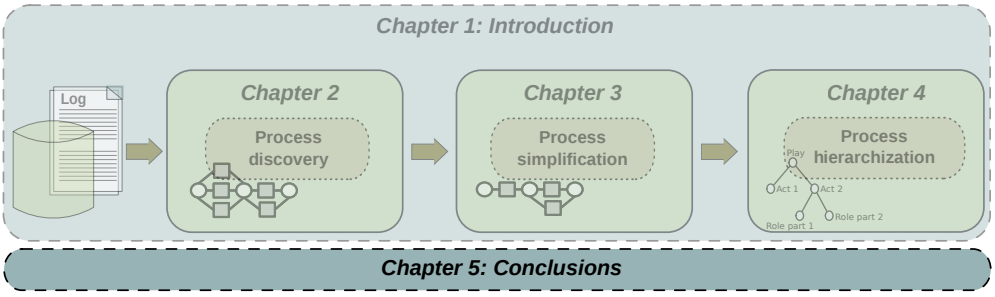
To validate this approach, we tested our framework with 9 UoLs with different degrees of complexity. Specifically, the three parts of the reengineering approach have been analyzed separately. Firstly, we showed that process mining is a good solution to retrieve a process structure from a set of event log files. Specifically, ProDiGen had a precision greater than 90% in most of the cases, and where the worst precision was of 82%. These high precision values minimize the number of UoLs obtained by the reengineering process. Secondly, the

identification of the adaptive rules also obtained very good results. In fact, the exact rule was extracted in most of the analyzed cases and at least 80% of the instances correctly classified in nearly all the remaining cases. Finally, we must mention that the correct IMS LD structure was retrieved for all the analyzed UoLs.

It should be remarked that a secondary objective of this paper is to facilitate the reuse of UoLs but from the perspective of instructors. In this sense, our system only requires the participation of instructors to select the most suitable process structure from the set of solutions recompiled by our framework. Notice that this is still a difficult task, since instructors do not always have a deep knowledge of how IMS LD structures a UoL. This decision is more complex as the number of recompiled structures, that match the logs, grows. Therefore, as future work we plan to define some criteria, in addition to those identified in [37], to order the recompiled UoLs, making this selection easier. So far our experience makes us think that the complexity is in the activities part of the UoL, since plays, acts, and role-parts are usually not considered when designing a UoL in non-IMS LD environments. However, this point must still be verified.

CHAPTER 5

CONCLUSIONS



In this PhD dissertation we have addressed the problem of automatic discovery of process models. Particularly, we have focused on obtaining models, in general domains, with high levels of replay fitness, precision and simplicity, while paying special attention in retrieving highly interpretable process models. For this purpose, we first developed ProDiGen, a genetic algorithm tailored towards the search of process models with high levels of replay fitness, precision, and simplicity. Using the results retrieved by ProDiGen, we then focused on retrieving high interpretable process models. To this end, we proposed two different approaches. On the one hand, we developed SLAD, a local search algorithm to improve the readability of an already discovered process model through the inclusion of duplicate labels. On the other hand, we developed a framework for the *hierarchization* of process models, providing a well known structure to a process model. More specifically, regarding the latter, the discovery of hierarchical process models is enabled through the inclusion of domain knowledge.

The main conclusions can be summarized as follows:

First, in Chapter 2 a novel genetic process algorithm, ProDiGen, has been described. With this algorithm we proved that following a hierarchical search towards replay fitness, precision, and simplicity, is a good criteria to retrieve rich(er) process models in a general manner. The algorithm uses a hierarchical fitness function that takes into account completeness, precision and simplicity (with new definitions for both precision and simplicity) and uses heuristics to optimize the genetic operators: *i)* a crossover operator that selects the crossover point from a Probability Density Function (PDF) generated from the errors of the mined model, and *ii)* a mutation operator guided by the causal dependencies of the log. ProDiGen was validated with 39 process models and several noise levels, giving a total of 111 different logs. Results conclude that using a hierarchical fitness based on completeness, precision and simplicity shows a great performance when retrieving the original model. Moreover, ProDiGen was able to retrieve the original model in the 84% of the tested logs. Furthermore, we have compared our approach with four state of the art algorithms; non-parametric statistical tests show that our algorithm outperforms the other approaches, and that the difference is statistically significant.

In Chapter 3 we presented SLAD, a novel algorithm for mining duplicate activities on an already discovery process model. Through this algorithm we show how it is possible to improve the structural clarity of an already process model by extending it with duplicate labels. Specifically, the novelties of this approach are: *i)* the discovering of the duplicate activities is performed *after* the discovery process, in order to *unfold* the overly connected nodes than may introduce extra behavior not recorded in the log; *ii)* new heuristics to focus the search of the duplicated tasks on those activities that better improve the model; and *iii)* new heuristics to detect potential duplicate activities involved in loops. SLAD has been validated with 18 different logs and 54 different initial solutions from four different process mining algorithms. Results show that the algorithm was able to enhance the initial solutions in 45 of the 54 tested scenarios. Furthermore, in *none of the cases* SLAD retrieved the trace-model after the label splitting process, avoiding the overfitted models with one path per trace. Also, none of the resulting models after applying SLAD were worse than its respective initial solution in none of the four quality dimensions. Moreover, we have compared SLAD with the state of the art process discovery algorithms capable to mine duplicate tasks. Statistical test have shown that the combination between ProDiGen and SLAD outperforms the rest of the algorithms, and that the differences are statistically significant.

Finally, in Chapter 4, we presented a novel framework for the hierarchization of process models. More specifically, we show how it is possible to *automatically translate* an already

discovered process model using domain knowledge and, thus, retrieve a more interpretable process model. The main components of this framework are as follows: *i)* a new framework to hierarchize process models using domain knowledge regardless the target language; *ii)* the automatic identification of the adaptive rules from event logs; *iii)* a new framework to make process models more interpretable based on domain knowledge. We have implemented and validated this framework within the educational domain, enabling the translation of a discovered process model into a standardized learning process model more suitable for teachers, i.e., the standard IMS LD. Hence, the different components of the framework have been analyzed using a set of nine real courses with different degrees of complexity. Firstly, experimental results showed that process mining is a good solution to retrieve a process structure from a set of event log files. Specifically, ProDiGen had a precision greater than 90% in most of the cases, and where the worst precision was of 82%. These high precision values minimize the number of courses obtained by the reengineering process. Secondly, the identification of the adaptive rules also obtained very good results. In fact, the exact rule was extracted in most of the analyzed cases and at least 80% of the instances correctly classified in nearly all the remaining cases. Finally, we must mention that the correct IMS LD structure was retrieved for all the analyzed courses.

The research accomplished in this thesis leads to a number of interesting applications in different fields and new developments that could be taken into consideration to continue as a future work:

- *Improvement of the replay fitness metric in the evolutionary algorithm*. The runtimes of ProDiGen are currently mostly limited by the fitness calculation. For evolutionary algorithms, fitness estimations might be sufficient during successive iterations.

- *Better genetic operators*. Although the current genetic operators of ProDiGen have already been largely optimized with different heuristics, the specific operators might be further improved. Currently, the repairing step after changing an individual is performed in a random way. This could be greatly improved by introducing heuristics to decide which kind of change should be performed, and what kind of operator should be applied. Additionally, through experimentation, the performance of the crossover operator is adequate in the first generations of the evolutionary cycle. Thereby, it would be interesting to introduce a trade-off between mutation and crossover depending on the iteration of the algorithm.

- *Improve the search space of SLAD*. Currently, SLAD only takes into account the already mined relations in a model. A extension of this algorithm would be the possibility to not only redistribute the already mined dependencies of the model, but to introduce new relations based on the combinations of the potential duplicate activities, in order to enhance those models with a lower fitness.

- *Extension of the semantics in the automatic reconstruction of courses*. The automatic reconstruction of courses does not support all the operators defined in IMS LD. In order to give support to the complete grammar of IMS LD, the number of operators used by the decision tree algorithm might be extended.

- *Hierarchization of process models in other domains*. An important feature of the described approach for hierarchization of process models is its independence from any target modelling language, although in this PhD Thesis the reengineering part of the framework is tied to IMS LD. However, it is sufficient to define the translation, from the Petri net models to the specific process representation, and from the decision tree to the rules grammar used by the target modelling language, to automate the hierarchization of a process model. Hence, as future work it might be possible to use this framework to recompile the same event logs to, for instance, a different educational modelling language, such as ADL SCORM.

Bibliography

[1] M. Abdous. Towards a framework for business process reengineering in higher education. *Journal of Higher Education Policy and Management*, 33(4):427–433, 2011.

[2] M. Abdous and W. He. A framework for process reengineering in higher education: A case study of distance learning exam scheduling and distribution. *The International Review of Research in Open and Distributed Learning*, 9(3):1–12, 2008.

[3] R. Accorsi and T. Stocker. On the exploitation of process mining for security audits: the conformance checking case. In S.Y. Shin and J.C. Maldonado, editors, *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC*, pages 1709–1716, Coimbra, Portugal, 2013. ACM.

[4] R. Accorsi, T. Stocker, and G. Müller. On the exploitation of process mining for security audits: the process discovery case. In S.Y. Shin and J.C. Maldonado, editors, *Proceedings of the 28th Annual ACM Symposium on Applied Computing, SAC*, pages 1462–1468, Coimbra, Portugal, 2013. ACM.

[5] A. Adriansyah. *Aligning observed and modeled behavior*. PhD thesis, Technische Universiteit Eindhoven, 2014.

[6] A. Adriansyah, J. Munoz-Gama, J. Carmona, B.F. van Dongen, and W.M.P. van der Aalst. Alignment based precision checking. In M. La Rosa and P. Soffer, editors, *Proceedings of the 10th International Workshops on Business Process Management, BPM*, volume 132 of *Lecture Notes in Business Information Processing*, pages 137–149, Tallinn, Estonia, 2012. Springer.

[7] R. Agrawal, D. Gunopulos, and F. Leymann. Mining process models from workflow logs. In H. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Proceedings of the in 6th*

International Conference on Extending Database Technology, EDBT, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483, Valencia, Spain, 1998. Springer.

[8] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford university press, 1st edition, 1996.

[9] E. Badouel and P. Darondeau. Theory of regions. In W. Reisig and G. Rozenberg, editors, *Lectures on Petri Nets I: Basic Models*, volume 1491 of *Lecture Notes in Computer Science*, pages 529–586, Dagstuhl, 1998. Springer.

[10] T. Baier, J. Mendling, and M. Weske. Bridging abstraction layers in process mining. *Information Systems*, 46:123–139, 2014.

[11] R. Bergenthum, J. Desel, A. Harrer, and S. Mauser. Modeling and mining of learnflows. *Trans. Petri Nets and Other Models of Concurrency*, 5:22–50, 2012.

[12] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process mining based on regions of languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Proceedings of the 5th International Conference on Business Process Management, BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 375–383, Brisbane, Australia, 2007. Springer.

[13] R.P.J.C. Bose and W.M.P. van der Aalst. Abstractions in process mining: A taxonomy of patterns. In U. Dayal, J. Eder, J. Koehler, and H.A. Reijers, editors, *Proceedings of the 7th International Conference on Business Process Management, BPM*, volume 5701 of *Lecture Notes in Computer Science*, pages 159–175, Ulm, Germany, 2009. Springer.

[14] R.P.J.C. Bose, H.M.W. Verbeek, and W.M.P. van der Aalst. Discovering Hierarchical Process Models Using ProM. In S. Nurcan, editor, *Proceedings of the CAiSE Forum 2011*, volume 107 of *Lecture Notes in Business Information Processing*, pages 33–40, London, UK, 2012. Springer.

[15] S.K.L.M. vanden Broucke. *Advances in Process Mining: Artificial Negative Events and Other Techniques*. PhD thesis, Katholieke Universiteit Leuven, 2014.

[16] S.K.L.M. vanden Broucke, J. Vanthienen, and B. Baesens. Declarative process discovery with evolutionary computing. In *Proceedings of the IEEE Congress on Evolutionary Computation, CEC*, pages 2412–2419, Beijing, China, 2014. IEEE.

[17] S.K.L.M. vanden Broucke, J. De Weerdt, J. Vanthienen, and B. Baesens. A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM*, pages 254–261, Singapore, Singapore, 2013. IEEE.

[18] J.C.A.M. Buijs. *Flexible Evolutionary Algorithms for Mining Structured Process Models*. PhD thesis, Technische Universiteit Eindhoven, 2014.

[19] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In R. Meersman, H. Panetto, T.S. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I.F. Cruz, editors, *OTM Federated Conferences, Proceedings of the 20th International Conference on Cooperative Information systems, CoopIS*, volume 7565 of *Lecture Notes in Computer Science*, pages 305–322, Rome, Italy, 2012. Springer.

[20] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems*, 23(1):1–39, 2014.

[21] A. Burattin and A. Sperduti. Heuristics Miner for Time Intervals. In *Proceedings of the 18th European Symposium on Artificial Neural Networks, ESANN*, Bruges, Belgium, 2010.

[22] A. Burattin, A. Sperduti, and W.M.P. van der Aalst. Heuristics miners for streaming event data. *CoRR*, abs/1212.6383, 2012.

[23] J. Carmona. The label splitting problem. *Transactions on Petri Nets and Other Models of Concurrency*, 6:1–23, 2012.

[24] J. Carmona, J. Cortadella, and M. Kishinevsky. A region-based algorithm for discovering petri nets from event logs. In M. Dumas, M. Reichert, and M.C. Shan, editors, *Proceedings of the 6th International Conference on Business Process Management, BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 358–373, Milan, Italy, 2008. Springer.

[25] J. Carmona, J. Cortadella, and M. Kishinevsky. New region-based algorithms for deriving bounded petri nets. *IEEE Transactions on Computers*, 59(3):371–384, 2010.

[26] R. Conforti, M. Dumas, L. García-Bañuelos, and M. La Rosa. BPMN miner: Automated discovery of BPMN process models with hierarchical structure. *Information Systems*, 56:284–303, 2016.

[27] J.E. Cook and A.L. Wolf. Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology*, 7(3):215–249, 1998.

[28] J. Dalziel. Implementing learning design. the learning activity management system (LAMS). In G. Crisp, D. Thiele, I. Scholten, S. Barker, and J. Baron, editors, *Proceedings of the 20th Annual Conference of the Australasian Society for Computers in Learning in Tertiary Education, ASCILITE*, pages 51–58, Adelaide, Australia, 2003. IADIS Press.

[29] M. de Leoni, W.M.P. van der Aalst, and M. Dees. A general framework for correlating business process characteristics. In S.W. Sadiq, P. Soffer, and H. Völzer, editors, *Proceedings of the 12th International Conference on Business Process Management, BPM*, volume 8659 of *Lecture Notes in Computer Science*, pages 250–266, Eindhoven, The Netherlands, 2014. Springer.

[30] A.K.A. de Medeiros. *Genetic Process Mining*. PhD thesis, Technische Universiteit Eindhoven, 2006.

[31] A.K.A. de Medeiros, A. Guzzo, G. Greco, W.M.P. van der Aalst, A.J.M.M. Weijters, B.F. van Dongen, and D. Saccà. Process mining based on clustering: A quest for precision. In A.H.M. Ter Hofstede, B. Benatallah, and H.Y. Paik, editors, *Proceedings of the 5th International Workshops on Business Process Management, BPM*, volume 4928 of *Lecture Notes in Computer Science*, pages 17–29, Brisbane, Australia, 2007. Springer.

[32] A.K.A. de Medeiros, B.F. van Dongen, W.M.P. van der Aalst, and A.J.M.M. Weijters. Process mining: Extending the α-algorithm to mine short loops. BETA Working Paper Series WP 113, Eindhoven University of Technology, 2004.

[33] A.K.A. de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007.

[34] J. de San Pedro, J. Carmona, and J. Cortadella. Log-Based Simplification of Process Models. In H.R. Motahari-Nezhad, J. Recker, and M. Weidlich, editors, *Proceedings of the 13th International Conference on Business Process Management, BPM*, volume 9253 of *Lecture Notes in Computer Science*, pages 457–474, Innsbruck, Austria, 2015. Springer.

[35] K. Deb, A. Pratap, S. Agarwal, and T.A.M.T. Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[36] P. Delias, M. Doumpos, E. Grigoroudis, P. Manolitzas, and N.F. Matsatsinis. Supporting healthcare management decisions via robust clustering of event logs. *Knowledge-Based Systems*, 84:203–213, 2015.

[37] M. Derntl, S. Neumann, D. Griffiths, and P. Oberhuemer. The conceptual structure of IMS learning design does not impede its use for authoring. *IEEE Transactions on Learning Technologies*, 5(1):74–86, 2012.

[38] P. Dixit, J.C.A.M. Buijs, W.M.P. van der Aalst, B. Hompes, and H. Buurman. Enhancing process mining results using domain knowledge. In P. Ceravolo and S. Rinderle-Ma, editors, *Proceedings of the 5th International Symposium on Data-driven Process Discovery and Analysis, SIMPDA*, volume 1527 of *CEUR Workshop Proceedings*, pages 79–94, Vienna, Austria, 2015. CEUR-WS.org.

[39] J.M. Dodero, Á.M. del Val, and J. Torres. An extensible approach to visually editing adaptive learning activities and designs based on services. *Journal of Visual Languages & Computing*, 21(6):332–346, 2010.

[40] M. Dumas, M. La Rosa, J. Mendling, and H.A. Reijers. *Fundamentals of Business Process Management*. Springer, 2013 edition, 2013.

[41] M. Dumas, W.M.P. van der Aalst, and A. Ter Hofstede. *Process-aware information systems: bridging people and software through process technology*. Wiley-Interscience, 1st edition, 2007.

[42] A. Ehrenfeucht and G. Rozenberg. Partial (Set) 2-Structures. Part II: State Spaces of Concurrent Systems. *Acta Informatica*, 27(4):343–368, 1990.

[43] A.E. Eiben and J.E. Smith. *Introduction to evolutionary computing*. Springer, 2nd edition, 2010.

[44] D. Fahland and W.M.P. van der Aalst. Simplifying discovered process models in a controlled manner. *Information Systems*, 38(4):585–605, 2013.

[45] M. Friedman. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200):675–701, 1937.

[46] IMS Global Learning Consortium. *IMS Learning Design Information Model*, 2003. Version 1.0 Final Specification.

[47] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust process discovery with artificial negative events. *The Journal of Machine Learning Research*, 10:1305–1340, 2009.

[48] G. Greco, A. Guzzo, F. Lupia, and L. Pontieri. Process discovery under precedence constraints. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 9(4):32, 2015.

[49] G. Greco, A. Guzzo, L. Ponieri, and D. Sacca. Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027, 2006.

[50] G. Greco, A. Guzzo, and L. Pontieri. Mining hierarchies of models: From abstract views to concrete specifications. In W.M.P. van der Aalst, B. Benatallah, F. Casati, and F. Curbera, editors, *Proceedings of the 3rd International Conference on Business Process Management, BPM*, volume 3649 of *Lecture Notes in Computer Science*, pages 32–47, Nancy, France, 2005.

[51] G. Greco, A. Guzzo, and L. Pontieri. Mining taxonomies of process models. *Data & Knowledge Engineering*, 67(1):74–102, 2008.

[52] G. Greco, A. Guzzo, L. Pontieri, and D. Sacca. Mining expressive process models by clustering workflow traces. In H. Dai, R. Srikant, and C. Zhang, editors, *Proceedings of the 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD*, volume 3056 of *Lecture Notes in Computer Science*, pages 52–62, Sydney, Australia, 2004. Springer.

[53] D. Griffiths, P. Beauvoir, and P. Sharples. Advances in editors for IMS LD in the tencompetence project. In *Proceedings of the 8th IEEE International Conference on Advanced Learning Technologies, ICALT*, pages 1045–1047, Santander, Cantabria, Spain, 2008. IEEE Computer Society.

[54] C.W. Günther. *Process mining in flexible environments*. PhD thesis, Technische Universiteit Eindhoven, 2009.

[55] C.W. Günther and W.M.P. van der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *Proceedings of the in 5th International Conference on Business Process Management, BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343, Brisbane, Australia, 2007. Springer.

[56] Q. Guo, L. Wen, J. Wang, Z. Yan, and S.Y. Philip. Mining invisible tasks in non-free-choice constructs. In H.R. Motahari-Nezhad, J. Recker, and M. Weidlich, editors, *Proceedings of the 13th International Conference on Business Process Management, BPM*, volume 9253 of *Lecture Notes in Computer Science*, pages 109–125, Innsbruck, Austria, 2015. Springer.

[57] J. Herbst. A machine learning approach to workflow management. In R. López de Mántaras and E. Plaza, editors, *Proceedings of the 11th European Conference on Machine Learning, ECML*, volume 1810 of *Lecture Notes in Computer Science*, pages 183–194, Barcelona, Spain, 2000. Springer.

[58] J. Herbst and D. Karagiannis. Workflow mining with InWoLvE. *Computers in Industry*, 53(3):245–264, 2004.

[59] D. Hernández-Leo, E.D. Villasclaras-Fernández, J.I. Asensio-Pérez, Y. Dimitriadis, I.M. Jorrín-Abellán, I. Ruiz-Requies, and B. Rubia-Avi. COLLAGE: A collaborative Learning Design editor based on patterns. *Educational Technology & Society*, 1(9):58–71, 2006.

[60] S. Heyer, P. Oberhuemer, S. Zander, and P. Prenner. Making sense of IMS learning design level B: from specification to intuitive modeling software. In E. Duval, R. Klamma, and M. Wolpers, editors, *Proceedings of the 2nd European Conference*

on Technology Enhanced Learning, (EC-TEL), volume 4753 of *Lecture Notes in Computer Science*, pages 86–100, Crete, Greece, 2007. Springer.

[61] J.L. Hodges and E.L. Lehmann. Rank methods for combination of independent experiments in analysis of variance. *The Annals of Mathematical Statistics*, 33(2):482–497, 1962.

[62] S. Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70, 1979.

[63] L. Howard, J. Johnson, and C. Neitzel. Examining learner control in a structured inquiry cycle using process mining. In R.S.J. de Baker, A. Merceron, and P.I. Pavlik Jr., editors, *Proceedings of the 3rd International Conference on Educational Data Mining, (EDM)*, pages 71–80, Pittsburgh, PA, USA, 2010. www.educationaldatamining.org.

[64] A. Kalenkova, M. de Leoni, and W.M.P. van der Aalst. Discovering, Analyzing and Enhancing BPMN Models Using ProM. In Lior Limonad and Barbara Weber, editors, *Proceedings of the BPM Demo Sessions co-located with the 12th International Conference on Business Process Management BPM*, volume 1295 of *CEUR Workshop Proceedings*, page 36, Eindhoven, The Netherlands, 2014. CEUR-WS.org.

[65] Y. Karagiorgi and L. Symeou. Translating constructivism into instructional design: Potential and limitations. *Educational Technology & Society*, 8(1):17–27, 2005.

[66] P. Karampiperis and D. Sampson. A flexible authoring tool supporting adaptive learning activities. In Kinshuk, D.G. Sampson, and P. T. Isaías, editors, *Proceedings of the IADIS International Conference on Cognition and Exploratory Learning in Digital Age, CELDA*, pages 51–58, Lisbon, Portugal, 2004. IADIS Press.

[67] M.H. Karray, B. Chebel-Morello, and N. Zerhouni. PETRA: process evolution using a trace-based system on a maintenance platform. *Knowledge-Based Systems*, 68:21–39, 2014.

[68] R. Kimball and M. Ross. *The data warehouse toolkit: the complete guide to dimensional modeling*. John Wiley & Sons, 3rd edition, 2011.

[69] M. Leemans and W.M.P. van der Aalst. Discovery of frequent episodes in event logs. In R. Accorsi, P. Ceravolo, and B. Russo, editors, *Proceedings of the 4th International*

Symposium on Data-driven Process Discovery and Analysis, SIMPDA, volume 1293 of *CEUR Workshop Proceedings*, pages 31–45, Milan, Italy, 2014. CEUR-WS.org.

[70] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering block-structured process models from event logs-a constructive approach. In J.M. Colom and J. Desel, editors, *Proceedings of the 34th International Conference on Application and Theory of Petri Nets and Concurrency PETRI NETS*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329, Milan, Italy, 2013. Springer.

[71] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In N. Lohmann, M. Song, and P. Wohed, editors, *Proceedings of the 12th International Workshops on Business Process Management, BPM*, volume 171 of *Lecture Notes in Business Information Processing*, pages 66–78, Beijing, China, 2013. Springer.

[72] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Scalable process discovery with guarantees. In K. Gaaloul, R. Schmidt, S. Nurcan, S. Guerreiro, and Q. Ma, editors, *Proceedings of the 16th International Conference on Enterprise, Business-Process and Information Systems Modeling, BPMDS*, volume 214 of *Lecture Notes in Business Information Processing*, pages 85–101, Stockholm, Sweden, 2015. Springer.

[73] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Using life cycle information in process discovery. In H.R. Motahari-Nezhad, J. Recker, and M. Weidlich, editors, *Proceedings of the 13th International Workshops on Business Process Management, BPM*, Lecture Notes in Computer Science, Innsbruck, Austria, 2015. Springer.

[74] M. de Leoni, J. Munoz-Gama, J. Carmona, and W.M.P. van der Aalst. Decomposing Alignment-Based Conformance Checking of Data-Aware Process Models. In R. Meersman, H. Panetto, T.S. Dillon, M. Missikoff, L. Liu, O. Pastor, A. Cuzzocrea, and T.K. Sellis, editors, *OTM Federated Conferences, Proceedings of the 2014 International Conference on Cooperative Information systems, CoopIS and On the Move to Meaningful Internet Systems, ODBASE*, volume 8841 of *Lecture Notes in Computer Science*, Amantea, Italy, 2014. Springer.

[75] J. Li, D. Liu, and B. Yang. Process mining: Extending α-algorithm to mine duplicate tasks in process logs. In K.C.C. Chang, W. Wang, L. Chen, C.A. Ellis, C.H. Hsu,

A. Chung Tsoi, and H. Wang, editors, *Advances in Web and Network Technologies, and Information Management*, volume 4537 of *Lecture Notes in Computer Science*, pages 396–407, Huang Shan, China, 2007. Springer.

[76] V. Liesaputra, S. Yongchareon, and S. Chaisiri. Efficient process model discovery using maximal pattern mining. In H.R. Motahari-Nezhad, J. Recker, and M. Weidlich, editors, *Proceedings of the 13th International Conference on Business Process Management, BPM*, volume 9253 of *Lecture Notes in Computer Science*, pages 441–456. Springer, Innsbruck, Austria, 2015.

[77] T. Liu, Y. Cheng, and Z. Ni. Mining event logs to support workflow resource allocation. *Knowledge-Based Systems*, 35:320–331, 2012.

[78] R. Lorenz, S. Mauser, and G. Juhás. How to synthesize nets from languages: a survey. In S.G. Henderson, B. Biller, M.H. Hsieh, J. Shortle, J.D. Tew, and R.R. Barton, editors, *Proceedings of the 39th Conference on Winter Simulation, WSC*, pages 637–647, Washington DC, USA, 2007. IEEE Press, WSC.

[79] F.M. Maggi, T. Slaats, and H.A. Reijers. The automated discovery of hybrid processes. In S.W. Sadiq, P. Soffer, and H. Völzer, editors, *Proceedings of the 12th International Conference on Business Process Management, BPM*, volume 8659 of *Lecture Notes in Computer Science*, pages 392–399, Eindhoven, The Netherlands, 2014. Springer.

[80] R.S. Mans, W.M.P. van der Aalst, and R.J.B. Vanwersch. *Process Mining in Healthcare: Evaluating and Exploiting Operational Healthcare Processes*. Springer Briefs in Business Process Management. Springer, 2015 edition, 2015.

[81] H. Martens and H. Vogten. CopperCore 3.3, 2009.

[82] I. Martinez-Ortiz, J.L. Sierra, and B. Fernandez-Manjon. Authoring and reengineering of ims learning design units of learning. *IEEE Transactions on Learning Technologies*, 2(3):189–202, 2009.

[83] L. Maruster. *A machine learning approach to understand business processes*. PhD thesis, Technische Universiteit Eindhoven, 2003.

[84] S. Mertens, F. Gailly, and G. Poels. Enhancing declarative process models with DMN decision logic. In K. Gaaloul, R. Schmidt, S. Nurcan, S. Guerreiro, and Q. Ma, editors,

Enterprise, Business-Process and Information Systems Modeling, volume 214 of *Lecture Notes in Business Information Processing*, pages 151–165. Springer, Stockholm, Sweden, 2015.

[85] T. Molka, D. Redlich, W. Gilani, X.J. Zeng, and M. Drobek. Evolutionary computation based discovery of hierarchical business process models. In W. Abramowicz, editor, *Proceedings of the 18th International Conference on Business Information Systems, BIS*, volume 208 of *Lecture Notes in Business Information Processing*, pages 191–204, Poznań, Poland, 2015. Springer.

[86] J. Munoz-Gama. *Conformance checking and diagnosis in process mining*. PhD thesis, Universitat Politècnica de Catalunya, 2014.

[87] J. Munoz-Gama, J. Carmona, and W.M.P. van der Aalst. Single-Entry Single-Exit Decomposed Conformance Checking. *Information Systems*, 46:102–122, 2014.

[88] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[89] G. Paquette and M. Léonard. The educational modeling of a collaborative game using MOT+LD. In *Proceedings of the 6th IEEE International Conference on Advanced Learning Technologies, ICALT*, pages 1156–1157, Kerkrade, The Netherlands, 2006. IEEE Computer Society.

[90] M. Pechenizkiy, N. Trcka, E. Vasilyeva, W.M.P. van der Aalst, and P. De Bra. Process mining online assessment data. In T. Barnes, M.C. Desmarais, C. Romero, and S. Ventura, editors, *Proceedings of the 2nd International Conference on Educational Data Mining, EDM*, pages 279–288, Cordoba, Spain, 2009. www.educationaldatamining.org.

[91] H. Ponce-de León, C. Rodríguez, J. Carmona, K. Heljanko, and S. Haar. Unfolding-based process discovery. In B. Finkbeiner, G. Pu, and L. Zhang, editors, *Proceedings of the - 13th International Symposium on Technology for Verification and Analysis, ATVA*, volume 9364, pages 31–47, Shanghai, China, 2015. Springer.

[92] W. Poncin, A. Serebrenik, and M. van den Brand. Mining student capstone projects with FRASR and ProM. In C.V. Lopes and K. Fisher, editors, *Proceedings of the*

26th Annual international conference companion on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA, pages 87–96, Portland, OR, USA, 2011. ACM.

[93] W. Poncin, A. Serebrenik, and M. van den Brand. Process mining software repositories. In T. Mens, Y. Kanellopoulos, and A. Winter, editors, *Proceedings of the 15th European Conference on Software Maintenance and Reengineering, CSMR*, pages 5–14, Oldenburg, Germany, 2011. IEEE Computer Society.

[94] I.R. Pulshashi, H. Bae, R.A. Sutrisnowati, B. Nugroho Y., and S. Park. Slice and connect: Tri-dimensional process discovery with case study of port logistics process. *Procedia Computer Science*, 72:461–468, 2015.

[95] J.R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 1993.

[96] D. Redlich, T. Molka, W. Gilani, G.S. Blair, and A. Rashid. Constructs competition miner: Process control-flow discovery of bp-domain constructs. In S.W. Sadiq, P. Soffer, and H. Völzer, editors, *Proceedings of the 12th International Conference on Business Process Management, BPM*, volume 8659 of *Lecture Notes in Computer Science*, pages 134–150, Eindhoven, The Netherlands, 2014. Springer.

[97] A.J. Rembert, A. Omokpo, P. Mazzoleni, and R. Goodwin. Process discovery using prior knowledge. In S. Basu, C. Pautasso, L. Zhang, and X. Fu, editors, *Proceedings of the 11th International Conference on Service-Oriented Computing , ICSOC*, volume 8274 of *Lecture Notes in Computer Science*, pages 328–342, Berlin, Germany, 2013. Springer.

[98] M.C. Rodríguez, M. Derntl, and L. Botturi. Visual instructional design languages. *Journal of Visual Languages and Computing*, 21(6):311–312, 2010.

[99] I. Rodríguez-Fdez, A. Canosa, M. Mucientes, and A. Bugarín. STAC: A web platform for the comparison of algorithms using statistical tests. In A. Yazici, N.R. Pal, U. Kaymak, T. Martin, H. Ishibuchi, C.T. Lin, J.M.C. Sousa, and B. Tütmez, editors, *Proceedings of the IEEE International Conference on Fuzzy Systems, FUZZ-IEEE*, pages 1–8, Istanbul, Turkey, 2015. IEEE.

[100] A. Rodríguez Groba, B. Vázquez-Barreiros, M. Lama, A. Gewerc, and M. Mucientes. Using a learning analytics tool for evaluation in self-regulated learning. In E. Tovar M. Castro, editor, *Proceedings of the 44th International Conference on Frontiers in Education, FIE*, pages 2484–2491, Madrid, Spain, 2014. IEEE.

[101] C. Romero, S. Ventura, and E. García. Data mining in course management systems: Moodle case study and tutorial. *Computers & Education*, 51(1):368–384, 2008.

[102] A. Rozinat, A.K.A. de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst. Towards an evaluation framework for process mining algorithms. BETA Working Paper Series WP 224, Eindhoven University of Technology, 2007.

[103] A. Rozinat, A.K.A de Medeiros, C.W. Günther, A.J.M.M. Weijters, and W.M.P. van der Aalst. The need for a process mining evaluation framework in research and practice. In A.H.M. Ter Hofstede, B. Benatallah, and H.Y. Paik, editors, *Proceedings of the 5th International Conference on Business Process Management, BPM*, volume 4928 of *Lecture Notes in Computer Science*, pages 84–89, Brisbane, Australia, 2008. Springer.

[104] A. Rozinat and W.M.P. van der Aalst. Decision mining in ProM. In S. Dustdar, J.L. Fiadeiro, and A.P. Sheth, editors, *Proceedings of the 4th International Conference on Business Process Management, BPM*, volume 4102 of *Lecture Notes in Computer Science*, pages 420–425, Vienna, Austria, 2006. Springer.

[105] A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

[106] N. Russell, A.H.M. Ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow resource patterns. BETA Working Paper Series WP 127, Eindhoven University of Technology, 2004.

[107] N. Russell, A.H.M. Ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Workflow data patterns. Technical Report FIT-TR-2004-01, Queensland University of Technology, 2004.

[108] N. Russell, A.H.M. Ter Hofstede, and N. Mulyar. Workflow control-flow patterns: A revised view. Technical Report BPM-06-22, BPMcenter.org, 2006.

[109] N. Russell and W.M.P. van der Aalst. Exception handling patterns in process-aware information systems. Technical Report BPM-06-04, BPMcenter.org, 2006.

[110] N. Russell, W.M.P. van der Aalst, and A.H.M. Ter Hofstede. *Workflow Patterns: The Definitive Guide*. MIT Press, 2016.

[111] L. Sánchez-González, F. García, J. Mendling, F. Ruiz, and M. Piattini. Prediction of business process model quality based on structural metrics. In J. Parsons, M. Saeki, P. Shoval, C.C. Woo, and Y. Wand, editors, *Proceedings of the 29th International Conference on Conceptual Modeling, (ER)*, volume 6412 of *Lecture Notes in Computer Science*, pages 458–463, Vancouver, BC, Canada, 2010. Springer.

[112] R. Sarno, W.A. Wibowo, and A. Solichah. Time based discovery of parallel business processes. In E. Kurniawan, editor, *Proceedings of the 2015 International Conference on Computer, Control, Informatics and its Applications, IC3INA*, pages 28–33, Bandung, Indonesia, 2015. IEEE.

[113] M. Solé and J. Carmona. Process mining from a basis of state regions. In J. Lilius and W. Penczek, editors, *Proceedings of the 31st International Conference on Applications and Theory of Petri Nets, PETRI NETS*, volume 6128 of *Lecture Notes in Computer Science*, pages 226–245, Braga, Portugal, 2010. Springer.

[114] M. Solé and J. Carmona. An smt-based discovery algorithm for c-nets. In S. Haddad and L. Pomello, editors, *Proceedings of the 33rd International Conference on Application and Theory of Petri Nets, PETRI NETS*, volume 7347 of *Lecture Notes in Computer Science*, pages 51–71, Hamburg, Germany, 2012. Springer.

[115] G. O. Spagnolo, E. Marchetti, A. Coco, P. Scarpellini, A. Querci, F. Fabbrini, and S. Gnesi. An experience on applying process mining techniques to the tuscan port community system. In D. Winkler, S. Biffl, and J. Bergsmann, editors, *Proceedings of the 8th International Conference on Software Quality. The Future of Systems-and Software Development, SWQD*, volume 238 of *LNBIP*, pages 49–60, Vienna, Austria, 2016. Springer.

[116] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In P.M.G. Apers, M. Bouzeghoub, and G. Gardarin, editors,

Proceedings of the 5th International Conference on Advances in Database Technology, EDBT, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17, Avignon, France, 1996. Springer.

[117] T. Sztyler, J. Völker, J. Carmona, O. Meier, and H. Stuckenschmidt. Discovery of personal processes from labeled sensor data–an application of process mining to personalized health care. In W.M.P. van der Aalst, R. Bergenthum, and J. Carmona, editors, *Proceedings of the International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED*, volume 1371 of *CEUR Workshop Proceedings*, pages 22–23, Brussels, Belgium, 2015. CEUR-WS.org.

[118] N. Tax, N. Sidorova, R. Haakma, and W.M.P. van der Aalst. Log-based evaluation of label splits for process models. *Procedia Computer Science*, 96:63–72, 2016.

[119] W.M.P. van der Aalst. Business process simulation revisited. In J. Barjis, editor, *Proceedings of the 6th International Workshop on Enterprise and Organizational Modeling and Simulation, EOMAS*, volume 63 of *Lecture Notes in Business Information Processing*, pages 1–14, Hammamet, Tunisia, 2010. Springer.

[120] W.M.P. van der Aalst. On the representational bias in process mining. In S. Reddy and S. Tata, editors, *Proceedings of the 20th International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE*, pages 2–7, Paris, France, 2011. IEEE Computer Society.

[121] W.M.P. van der Aalst. Decomposing Petri Nets for Process Mining: A Generic Approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.

[122] W.M.P. van der Aalst. *Process Mining: Data Science in Action*. Springer Publishing Company, Incorporated, 2nd edition, 2016.

[123] W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Causal nets: a modeling language tailored towards process discovery. In J.P. Katoen and B. König, editors, *Proceedings of the 22nd International Conference on Concurrency Theory CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 28–42, Aachen, Germany, 2011. Springer.

[124] W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[125] W.M.P. van der Aalst and C.W. Günther. Finding structure in unstructured processes: The case for process mining. In T. Basten, G. Juhás, and S.K. Shukla, editors, *Proceedings of the 7th International Conference on Application of Concurrency to System Design, ACSD*, pages 3–12, Bratislava, Slovak Republic, 2007. IEEE Computer Society.

[126] W.M.P. van der Aalst, A. Kalenkova, V. Rubin, and H.M.W. Verbeek. Process discovery using localized events. In R.R. Devillers and A. Valmari, editors, *Proceedings of the 36th International Conference on Application and Theory of Petri Nets and Concurrency, PETRI NETS*, volume 9115 of *Lecture Notes in Computer Science*, pages 287–308. Springer, Brussels, Belgium, 2015.

[127] W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.

[128] W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering social networks from event logs. *Computer Supported Cooperative Work (CSCW)*, 14(6):549–593, 2005.

[129] W.M.P. van der Aalst, V. Rubin, B.F. van Dongen, E. Kindler, and C.W. Günther. Process mining: A two-step approach using transition systems and regions. Technical Report BPM-06-30, BPMcenter.org, 2006.

[130] W.M.P. van der Aalst, M.H. Schonenberg, and M. Song. Time prediction based on process mining. *Information Systems*, 36(2):450–475, 2011.

[131] W.M.P. van der Aalst and M. Song. Mining social networks: Uncovering interaction patterns in business processes. In J. Desel, B. Pernici, and M. Weske, editors, *Proceedings of the 2th International Conference on Business Process Management, BPM*, Lecture Notes in Computer Science, pages 244–260, Potsdam, Germany, 2004. Springer.

[132] W.M.P. van der Aalst, A.H.M. Ter Hofstede, B. Kiepuszewski, and A.P. Barros. Workflow patterns. *Distributed and parallel databases*, 14(1):5–51, 2003.

[133] W.M.P. van der Aalst and B.F. van Dongen. Discovering workflow performance models from timed logs. In Y. Han, S. Tai, and D. Wikarski, editors, *Proceedings of the 1st International Conference onEngineering and Deployment of Cooperative Information Systems, EDCIS*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63, Beijing, China, 2002. Springer.

[134] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: a survey of issues and approaches. *Data & Knowledge Engineering*, 47(2):237–267, 2003.

[135] W.M.P. van der Aalst and A.J.M.M. Weijters. Process mining: a research agenda. *Computers in Industry*, 53(3):231–244, 2004.

[136] W.M.P. van der Aalst and A.J.M.M. Weijters. Process mining: a research agenda. *Computers in Industry*, 53(3):231–244, 2004.

[137] W.M.P. van der Aalst, A.J.M.M. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004.

[138] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. *Fundamenta Informaticae*, 94(3-4):387–412, 2009.

[139] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM Framework: A New Era in Process Mining Tool Support. In G. Ciardo and P. Darondeau, editors, *Proceedings of the 26th International Conference Applications and Theory of Petri Nets, ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454, Miami, USA, 2005. Springer.

[140] B.F. van Dongen, A.K.A. de Medeiros, and L. Wen. Process mining: Overview and outlook of petri net discovery algorithms. *Transactions on Petri Nets and Other Models of Concurrency II*, 5460:225–242, 2009.

[141] M.L. van Eck, J.C.A.M Buijs, and B.F. van Dongen. Genetic Process Mining: Alignment-Based Process Model Mutation. In F. Fournier and J. Mendling, editors, *Proceedings of the 12th International Workshop on Business Process Management,*

BPM, volume 202 of *Lecture Notes in Business Information Processing*, pages 291–303, Eindhoven, The Netherlands, 2014. Springer.

[142] M.L. van Eck, X. Lu, S.J.J. Leemans, and W.M.P. van der Aalst. PM^2: A process mining project methodology. In J. Zdravkovic, M. Kirikova, and P. Johannesson, editors, *Proceedings of the 27th International Conference on Advanced Information Systems Engineering, CAiSE*, volume 9097 of *Lecture Notes in Computer Science*, pages 297–313, Stockholm, Sweden, 2015. Springer.

[143] S.J. van Zelst, A. Burattin, B.F. van Dongen, and H.M.W. Verbeek. Data Streams in ProM 6: A Single-node Architecture. In L. Limonad and B. Weber, editors, *Proceedings of the BPM Demo Sessions co-located with the 12th International Conference on Business Process Management BPM*, volume 1295 of *CEUR Workshop Proceedings*, page 81, Eindhoven, The Netherlands, 2014. CEUR-WS.org.

[144] S.J. van Zelst, B.F. van Dongen, and W.M.P. van der Aalst. Avoiding Over-Fitting in ILP-Based Process Discovery. In H.R. Motahari-Nezhad, J.R., and M. Weidlich, editors, *Proceedings of the 13th International Conference on Business Process Management, BPM*, volume 9253 of *Lecture Notes in Computer Science*, pages 163–171, Innsbruck, Austria, 2015. Springer.

[145] B. Vázquez-Barreiros, D. Chapela, M. Mucientes, and M. Lama. Process Mining in IT Service Management: A Case Study. In W.M.P. van der Aalst, R. Bergenthum, and J. Carmona, editors, *Proceedings of the 2016 International Workshop on Algorithms & Theories for the Analysis of Event Data, ATAED*, CEUR Workshop Proceedings, pages –, Toruń, Poland, 2016. CEUR-WS.org.

[146] B. Vázquez Barreiros, M. Lama, M. Mucientes, and J.C. Vidal. Softlearn: A process mining platform for the discovery of learning paths. In D.G. Sampson, J.M. Spector, N.S. Chen, R. Huang, and Kinshuk, editors, *Proceedings of the 14th International Conference on Learning Technologies ICALT*, pages 373–375, Athens, Greece, 2014. IEEE Computer Society.

[147] B. Vázquez-Barreiros, M. Mucientes, and M. Lama. ProDiGen: Mining complete, precise and minimal structure process models with a genetic algorithm. *Information Sciences*, 294:315–333, 2015.

[148] B. Vázquez-Barreiros, A. Ramos-Soto, M. Lama, M. Mucientes, A. Bugarín, and S. Barro. Soft computing for learner's assessment in softlearn. In *Proceedings of the 17th International Conference on Artificial Intelligence in Education, AIED*, pages 925–926, Madrid, Spain, 2015. Springer.

[149] B. Vázquez-Barreiros, S.J. van Zelst, J.C.A.M. Buijs, M. Lama, and M. Mucientes. Repairing alignments: Striking the right nerve. In *Proceedings of the 17th International Conference on Business Process Modeling, Development, and Support BPMDS*, volume 248 of *Lecture Notes in Business Information Processing*, pages 266–281, Toruń, Poland, 2016.

[150] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van Der Aalst. XES, XESame, and ProM 6. In P. Soffer and E. Proper, editors, *Information Systems Evolution*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75, Hammamet, Tunisia, 2011. Springer.

[151] J.C. Vidal, M. Lama, and A. Bugarín. Petri net-based engine for adaptive learning. *Expert Systems With Applications*, 39(17):12799–12813, 2012.

[152] H. Vogten, H. Martens, R. Nadolski, C. Tattersall, P. van Rosmalen, and R. Koper. Coppercore service integration. *Interactive Learning Environments*, 15(2):171–180, 2007.

[153] A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible heuristics miner (FHM). BETA Working Paper Series WP 334, Eindhoven University of Technology, 2010.

[154] A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

[155] A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K.A de Medeiros. Process mining with the heuristics miner-algorithm. BETA Working Paper Series WP 166, Eindhoven University of Technology, 2006.

[156] L. Wen, W.M.P. van der Aalst, J. Wang, and J. Sun. Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, 2007.

[157] L. Wen, J. Wang, and J. Sun. Mining invisible tasks from event logs. In G. Dong, X. Lin, W. Wang, Y. Yang, and J.X. Yu, editors, *Proceedings of the Joint Conference of the 9th Asia-Pacific Web Conference, APWeb, and the 8th International Conference on Web-Age Information Management, WAIM*, volume 4505 of *Lecture Notes in Computer Science*, pages 358–365, Huang Shan, China, 2007. Springer.

[158] L. Wen, J. Wang, W.M.P van der Aalst, B. Huang, and J. Sun. Mining process models with prime invisible tasks. *Data & Knowledge Engineering*, 69(10):999–1021, 2010.

[159] M. Westergaard, C. Stahl, and H.A. Reijers. Unconstrainedminer: efficient discovery of generalized declarative process models. Technical Report BPM-13-28, BPMcenter.org, 2013.

[160] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin 1*, pages 80–83, 1945.

List of Figures

List of Tables

List of Algorithms