André Filipe Domingos Gomes

Análise e Exploração de Dados em Cuidados de Saúde

Dezembro de 2021

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

IPV - ESTGV | 2021

Análise e Exploração de Dados em Cuidados de Saúde

André Gomes

Instituto Politécnico de Viseu

Escola Superior de Tecnologia e Gestão de Viseu

André Filipe Domingos Gomes

Análise e Exploração de Dados em Cuidados de Saúde

Tese de Mestrado

Sistemas e Tecnologias de Informação para as Organizações

Professora Doutora Ana Cristina Wanzeller Guedes de Lacerda

Professora Doutora Joana Rita da Silva Fialho

Dezembro de 2021

Dedico à minha família e amigos
por todo o suporte e compreensão

RESUMO

Em diversos setores, há uma grande quantidade de dados recolhidos e armazenados, mas que não são analisados. A área da saúde não é distinta e, pela importância que tem junto da população, esta situação não é desejável. Os dados podem fornecer informações históricas ou tendências que poderão ajudar a melhorar o desempenho das organizações no futuro.

Para contornar as dificuldades referidas, a mineração de processos (*Process Mining*) possibilita a extração de conhecimento a partir de dados gerados e armazenados nos sistemas de informação. Assim, este trabalho pretende mostrar os benefícios da utilização do *Process Mining* na melhoria de processos de saúde, neste caso aplicada a dados de um serviço de urgências.

O trabalho iniciou-se por uma fase de pesquisa e exploração, na qual os algoritmos e ferramentas de *Process Mining* foram investigados, analisados e comparados. Nesta fase, as ferramentas de *Process Mining*, *PM4Py*, *ProM* e *Disco*, foram testadas, bem como os algoritmos disponíveis *Alpha Miner*, *Directly-Follows Graphs*, *Fuzzy Miner*, *Heuristic Miner* e *Inductive Miner*, em cenários com dados que apresentavam desafios reais. Destes testes resultaram comparações e conclusões que foram apresentadas de forma compreensível e intuitiva. Esta fase foi de extrema importância, pois permitiu obter um conhecimento prático necessário para que a posterior aplicação das técnicas fosse o mais eficiente possível.

O caso de estudo final permitiu perceber o real potencial das contribuições que o *Process Mining* pode dar a uma área de tão relevante interesse geral. Com resultados muito positivos, chegou-se a um modelo que permitiu a análise do real funcionamento de um serviço de urgências. Este modelo foi examinado ao pormenor e, posteriormente, foi verificada a conformidade de acordo com o modelo previsto. Esta comparação possibilitou retirar conclusões em relação à gestão do processo, assim como dos seus recursos. Estas informações são extremamente úteis para que os profissionais de cada unidade possam agir sobre as ineficiências existentes e melhorar um serviço com tanta importância para a população.

ABSTRACT

In many industries, there is a large amount of data collected and stored but not analyzed. The health area is not distinct and, given its importance to the population, this situation is not desirable. Data can provide historical information or trends that can help improve organizations' performance in the future.

To overcome the aforementioned difficulties, Process Mining allows extracting knowledge from data generated and stored in information systems. Thus, this work aims to show the benefits of using Process Mining to improve health processes, in this case applied to data of an emergency service.

The work began with a research and exploration phase, where the Process Mining algorithms and tools were investigated, analyzed and compared. In this phase, the Process Mining tools, PM4Py, ProM and Disk, were tested, as well as the available algorithms, Alpha Miner, Directly-Follows Graphs, Fuzzy Miner, Heuristic Miner and Inductive Miner, in scenarios with data that represented real challenges. These tests resulted in comparisons and conclusions that were presented in an understandable and intuitive way. This phase was extremely important because it allowed to obtain the practical and necessary knowledge for the subsequent application.

The final case study allowed to realize the real potential of the contributions that Process Mining can make in an area of such relevant general interest. With very positive results, a model was generated in order to analyze the real functioning of an emergency service. This model was examined in detail and, subsequently, compliance was verified in accordance with the expected model. This comparison led to draw conclusions in relation to the management of the process, as well as its resources. This information is extremely useful for professionals in order to act on existing inefficiencies and improve a service that is so important to the population.

PALAVRAS CHAVE

Big Data de saúde
Process Mining
PM4Py
ProM
Disco
Alpha Miner
Directly-Follows Graph
Fuzzy Miner
Heuristic Miner
Inductive Miner

KEY WORDS

Big data in healthcare
Process Mining
PM4Py
ProM
Disco
Alpha Miner
Directly-Follows Graphs
Fuzzy Miner
Heuristic Miner
Inductive Miner

AGRADECIMENTOS

Gostaria de agradecer em primeiro lugar aos meus pais e à minha irmã, que estão comigo nos melhores e piores momentos, especialmente por confiarem em mim quando nos tempos do ensino secundário ninguém apoiou a minha continuação, talvez nem mesmo eu. De batalha em batalha chegamos aqui. Lembrando, claro, dos meus avós que me apoiaram e festejam com as minhas conquistas como se fossem as deles.

De ressaltar o apoio dado pelas orientadoras e pelo Instituto Politécnico de Viseu, o qual permitiu que todo o material de investigação desenvolvido pudesse ser apresentado em conferência e difundido para o público. Todas essas participações em conferência foram experiências importantes e gratificantes para mim. Em todo este tempo sempre senti um ambiente de comprometimento que me permitiu manter motivado e focado para dar resposta a todos os desafios propostos.

Quero deixar ainda um enorme agradecimento aos meus amigos e colegas de trabalho por toda a disponibilidade para dar uma opinião técnica de alguém que está de fora do trabalho.

Um enorme obrigado a todos!

ÍNDICE GERAL

ÍNDICE DE FIGURAS

ÍNDICE DE QUADROS

ABREVIATURAS E SIGLAS

BPM	*Business Process Management*
BPR	*Business Process Reengineering*
CID	Classificação Internacional de Doenças
CITI	*Collaborative Institutional Training Initiative*
CPT	Código de Terminologia Processual
CRISP-DM	*Cross Industry Standard Process for Data Mining*
CSV	*Comma-separated values*
DFG	*Directly-Follows Graph*
DRG	*Diagnosis Related Groups*
DWS	*Disjunctive Workflow Schema*
HIPAA	*Health Insurance Portability and Accountability Act*
IoT	*Internet of Things*
KDD	*Knowledge Discovery in Databases*
MIMIC-III	*Medical Information Mart for Intensive Care III*
MTS	*Manchester Triage System*
PM4Py	*Process Mining for Python*
SEMMA	*Sample, Explore, Modify, Model, Assess*
SNS	Sistema Nacional de Saúde
UTI	Unidade de Tratamento Intensivo
WFM	*Workflow Management*
XES	*eXtensible Event Stream*
XML	*eXtensible Markup Language*

1. Introdução

No contexto atual, as organizações dispõem de elevadas quantidades de dados (*Big Data*), mas o valor subjacente destes dados não está facilmente ao alcance de qualquer pessoa (L'Heureux et al., 2017). Uma solução possível para resolver este problema, que é também encarada como uma oportunidade de obter indicações para melhoramento do funcionamento das instituições, consiste em recorrer a processos de extração de conhecimento a partir de dados. Estes processos requerem, precisamente, quantidades elevadas de dados e permitem obter conhecimento útil para potenciar oportunidades (Mbassegue et al., 2016).

A área da saúde não é distinta e qualquer assunto relacionado com a saúde é sempre um tema sensível pela importância que tem junto da população. Está diretamente relacionado com o bem-estar das pessoas e, especialmente, com a sua sensação de segurança na obtenção dos cuidados básicos de saúde. Dados estatísticos mostram que a população está cada vez mais envelhecida, reforçando a importância da existência de um bom Sistema Nacional de Saúde (SNS) (Bárrios et al., 2020).

No entanto, a gestão e o planeamento do serviço referido são complexos, levando a que os hospitais deixem de conseguir dar resposta no tempo esperado, implicando a prestação de um serviço condicionado por recursos, que são, muitas vezes, escassos e dispendiosos. A comumente elevada quantidade de pacientes pode levar os profissionais de saúde a tomarem decisões sob pressão (Sakellarides, 2020).

Durante o atendimento de um paciente num dado hospital, os processos consistem em muitas atividades diferentes, incluindo tarefas administrativas, como admissão, alta ou transferência para uma enfermaria, e atividades clínicas, como triagem, testes, diagnóstico ou terapia. Essas atividades são desempenhadas por pessoas, com diferentes funções clínicas, sendo elas, médicos, enfermeiros, especialistas técnicos, entre outros, e variam de uma instituição de saúde para outra (Kurniati et al., 2018). Além disso, diferentes terapias podem ser seguidas para tratar pacientes com a mesma doença. O aparecimento contínuo de novos procedimentos, diretrizes e medicamentos de saúde também leva a modificações nos processos. Essa falta de

uniformidade produz um aumento de complexidade na condução e validação dos processos de saúde (Batista & Solanas, 2019).
Outro dos principais desafios é a heterogeneidade da informação proveniente de diferentes sistemas de informação, por vezes independentes e com lógicas diferentes nas suas estruturas (Batista & Solanas, 2019).
Conclui-se, naturalmente, que existe uma elevada complexidade nos processos da saúde, onde eventos imprevistos podem ocorrer (no decurso de uma doença ou durante o tratamento). Neste sentido, existe falta de tecnologias de *Business Process Management* (*BPM*), *Workflow Management* (*WFM*) e *Business Process Reengineering* (*BPR*) na área de saúde (Helm & Küng, 2016). Por conseguinte, torna-se importante analisar processos e encontrar abordagens que melhorem a otimização de recursos que são muito importantes e, por vezes, escassos (Almeida et al., 2020). Este trabalho pretende avaliar a exequibilidade da análise de processos na área da saúde e perceber de que forma essa análise poderá contribuir para otimizar recursos, resolver constrangimentos e ultrapassar obstáculos que estejam a impedir o desempenho esperado do processo.

1.1 Enquadramento

No setor de saúde, como em outros setores, há uma grande quantidade de dados recolhidos e armazenados, mas que não são analisados (Hendricks, 2019). Esta situação não é desejável, pois os dados podem fornecer informações históricas ou tendências que podem ajudar no desenvolvimento de métodos, estratégias e modelos de previsão atuais e futuros (Batista & Solanas, 2019).
É da possibilidade referida que nasceu a mineração de dados (*Data Mining*). Visa ajudar a encontrar tendências e padrões a partir dessas grandes quantidades de dados, além de fornecer maneiras de classificar ou identificar grupos ou entidades semelhantes (Sowmya & Suneetha, 2017). A mineração de dados é eficaz, mas a sua limitação pode estar no próprio processo ou nos conjuntos de dados com data e hora (Hendricks, 2019).
Ao analisar registos de dados com data e hora, as etapas são cronológicas e, portanto, dependentes de outras informações. Assim, os registos de eventos devem ser analisados como um todo e não em separado (Gatta et al., 2018). Correspondem a uma série de etapas que, separadas, não têm nenhum significado e, portanto, requerem análise como um todo, algo que a mineração de processos consegue lidar (Hendricks, 2019).
A mineração de processos é orientada para obter conhecimento sobre um determinado processo em execução e permite ter um modelo preciso do comportamento do processo, de forma a melhorar a implementação e avaliação deste, além de permitir configurar quaisquer requisitos adicionais não incluídos no sistema (Hendricks, 2019). Num cenário ideal, será possível descrever o que aconteceu, porque aconteceu, o que vai acontecer e o que pode melhorar no que vai acontecer (Kurniati et al., 2018).

Assim, a mineração de processos é uma abordagem de análise de dados para descobrir e analisar modelos de processos com base nas atividades reais capturadas em sistemas de informação (Kurniati et al., 2018). É uma disciplina de pesquisa emergente que combina inteligência computacional, mineração de dados, modelação de processos e abordagens de análise e que visa extrair conhecimento sobre processos, a partir dos dados armazenados em bases de dados de sistemas de informação corporativos (Rojas et al., 2019).
A abordagem referida ajuda a determinar quais as atividades, subprocessos, interações e características dos episódios que explicam porque o processo tem este ou aquele comportamento (Rojas et al., 2019). Em acréscimo, fornece informações adicionais que poderão ajudar a diminuir o tempo de execução, a reduzir possíveis congestionamentos e a incrementar a qualidade do processo (Kurniati & Hall, 2020).
Numa perspetiva de síntese, pode-se afirmar que a mineração de processos é um método orientado a processos e centrado em dados. Orientado para o processo, pois descreve os episódios como uma sequência de atividades que são executadas numa ordem específica. Centrado em dados porque se baseia em dados extraídos como *logs* de eventos, executados durante um período específico (Cho et al., 2020). Por conseguinte, defende-se a tese que a aplicação de mineração de processos a dados de processos de cuidados de saúde poderá ser uma contribuição para os possíveis problemas identificados. Constata-se ainda que existem várias ferramentas de mineração de processos, bem como distintos algoritmos para este efeito, não havendo estudos que apoiem a decisão na opção por uns em detrimento de outros.

1.2 Objetivos

Perante o enquadramento e desafios apresentados, os objetivos a que este projeto se propôs consistem em:

- Fundamentação teórica da aplicação de *Process Mining* na área da saúde, visando justificar a sua pertinência e contribuições;
- Análise e comparação de algoritmos e de ferramentas de *Process Mining*, com o intuito de entender as respetivas características, limitações e vantagens;
- Aplicação de *Process Mining* a um caso de estudo, de forma a demonstrar como se identificam eventuais problemas, gargalos ou constrangimentos nos processos de cuidados de saúde, analisando-os e compreendendo-os para que se encontrem soluções que melhorem o desempenho do processo.

1.3 Metodologia

A metodologia de investigação científica proposta para a concretização deste projeto baseia-se nas etapas descritas na metodologia (Oliveira, 2010) apresentada na Figura 1-1. Assim, este trabalho seguiu as seguintes etapas, durante a sua execução:

- Revisão da literatura:
  - Estudo do tema de *Process Mining* e dos principais algoritmos e ferramentas disponíveis neste âmbito;
  - Pesquisa de trabalhos e aplicações relacionadas com a utilização de *Process Mining* na área da saúde;
  - Estudo e aplicação da metodologia *CRISP-DM* (*CRoss Industry Standard Process for Data Mining*) (Martinez-Plumed et al., 2019), apresentada na Figura 1-2 e descrita mais abaixo, orientada para mineração de processos;
- Problemas e Hipóteses:

  Estudar a hipótese do *Process Mining* ser efetivamente um instrumento que ajude na gestão e melhoria de processos na área da saúde, nomeadamente hospitalares, partindo do conjunto de dados *MIMIC-III* (*Medical Information Mart for Intensive Care III*) (Kurniati et al., 2018);
- Experiências / Análises:
  - Estudo exploratório e comparativo de ferramentas de *Process Mining*, de forma a perceber as suas vantagens e desvantagens, nomeadamente do *ProM*, *Disco* e da *framework PM4Py*, em *Python*;
  - Estudo comparativo de algoritmos, identificando limitações e utilidade da sua aplicação, de forma a identificar o melhor tipo de abordagem e modelos;
- Caso de estudo:

  Análise e exploração de um processo específico de saúde, cujos dados se encontram em *MIMIC-III*;
- Teoria / Artigos:

  Escrita da dissertação e artigos relacionados com a aprendizagem e resultados obtidos com a realização deste projeto.

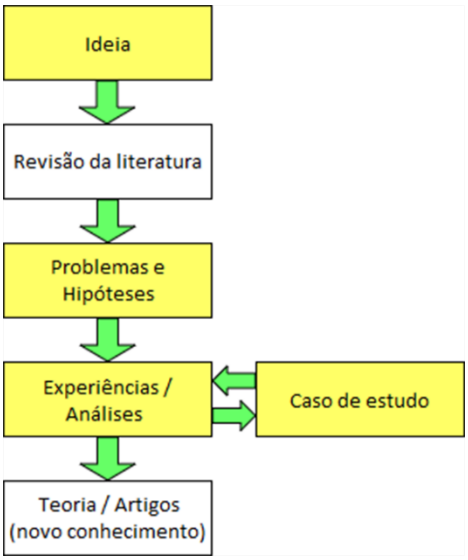


Figura 1-1: Metodologia de Investigação Científica (Adaptado de Oliveira, 2010).

A mineração de processos é a base deste trabalho, mas antes do processo, é necessário minerar os dados extraídos das etapas que compõem o processo.

O uso de metodologias de mineração de dados, como *CRISP-DM* (*Cross Industry Standard Process for Data Mining*), *KDD* (*Knowledge Discovery in Databases*) e *SEMMA* (*Sample, Explore, Modify, Model, Assess*) apresentados e comparados em (Daderman & Rosander, 2018), cresceu substancialmente na última década. No entanto, estas podem e devem ser adaptadas para fins específicos (Plotnikova et al., 2020). Apesar de todas as metodologias propostas para acomodar algumas das mudanças, o modelo *CRISP-DM* pode ser visto como a abordagem base a partir da qual a maioria das propostas subsequentes evoluíram, que se mantêm focadas no paradigma tradicional de uma lista sequencial de estágios, dos dados ao conhecimento (Martinez-Plumed et al., 2019).

O *CRISP-DM* tem cerca de duas décadas e, de acordo com muitas pesquisas de opinião, ainda é o padrão para o desenvolvimento de projetos de mineração de dados e descoberta de conhecimento (Martinez-Plumed et al., 2019). Ao introduzir a estrutura *CRISP-DM*, (Wirth, 2000) descreve a análise de negócios como um processo criativo que requer uma abordagem padrão para "ajudar a traduzir problemas de negócios em tarefas de dados, sugerir transformações de dados e técnicas de dados apropriadas e fornecer meios para avaliar a eficácia dos resultados e documentar a experiência ” (Jaggia et al., 2020).

Um dos motivos para que o *CRISP-DM* seja uma estrutura bem aceite é porque este se originou numa perspetiva de negócios, tornando-se um método iterativo e fácil de implementar em processos de desenvolvimento. É uma metodologia bem estruturada e com etapas bem definidas (Daderman & Rosander, 2018).

No entanto, existiram avanços consideráveis. Não apenas a natureza dos dados mudou, mas também os processos para extrair valor deles. Ainda assim, o *CRISP-DM* é considerado a metodologia de mineração de dados mais apta em termos de atendimento às necessidades de projetos industriais, tornando-se a metodologia mais utilizado para projetos de mineração de dados, de acordo com o *KDnuggets Polls* (https://www.kdnuggets.com) realizado em 2002, 2004, 2007 e 2014 (Martinez-Plumed et al., 2019).

Em suma, *CRISP-DM* é considerado o padrão para projetos de análise, mineração de dados e ciência de dados (Martinez-Plumed et al., 2019). Uma explicação disso pode ser a sua heterogeneidade, já que o *CRISP-DM* é um modelo de processo organizacional e não restrito a nenhuma tecnologia. Portanto, várias tecnologias podem-se apoiar nesse modelo (Schröer et al., 2021).

Para corroborar essa visão, pode-se encontrar um grande número de estudos convencionais onde foi aplicada ou adaptada a metodologia *CRISP-DM* para muitos domínios diferentes: cuidados de saúde (Niakšu, 2015), (Muriuki, 2015), (Azadeh-Fard et al., 2019), engenharia (Huber et al., 2019), educação (Castro R. et al., 2018), logística (Tumelaire, 2015), produção (Schäfer et al., 2020), sensores e aplicações da Internet das Coisas (*IoT*) (Nagashima & Kato, 2019), turismo (Gomez et al., 2019), desporto (Bunker & Thabtah, 2019) e direito (Barros et al., 2011).

A metodologia *CRISP* foi concebida para catalogar e orientar as etapas mais comuns em projetos de mineração de dados. Para tal, segue os seguintes passos (Schröer et al., 2021) (Jaggia et al., 2020):

- Entender o Negócio: o primeiro passo consiste em entender o objetivo do projeto, a partir de uma perspetiva de negócios, definindo um plano preliminar para atingir os objetivos;
- Entender os Dados: neste passo acontece a recolha de dados e início das atividades para familiarização com os dados, de forma a identificar problemas ou conjuntos de interesse;
- Preparação dos Dados: nesta etapa é feita a construção do conjunto de dados final;
- Modelação: consiste na aplicação de várias técnicas de modelação, onde os parâmetros são calibrados para otimização do processo. Durante essa fase é comum voltar à etapa anterior, de preparação dos dados;
- Avaliação: é nesta fase que se verifica se o modelo obtido vai ao encontro dos objetivos de negócio, tendo em conta as questões a que se pretendia dar resposta;
- Implementação: envolve organizar o conhecimento adquirido pelo modelo para ser apresentado ao cliente, de uma maneira que ele o consiga utilizar.

Percebe-se uma enorme vantagem desta metodologia que é o facto de ter uma estrutura que se assemelha ao desenvolvimento de software, como muitos outros problemas de engenharia (começando com as necessidades do negócio e terminando na implementação e análise dos resultados) (Martinez-Plumed et al., 2019). Mas é de salientar que, nos projetos atuais de ciência de dados, os modelos de processo não são fielmente seguidos, como mostra (Saltz et al., 2018), que refere que apenas 18% das equipas de ciência de dados seguem um modelo explícito (Schröer et al., 2021). Ao seguir uma estrutura rígida, como o *CRISP-DM*, pode-se perder alguma criatividade. Portanto, é preferível encontrar um equilíbrio entre garantir que os passos sejam seguidos e adaptar ao processo em estudo (Daderman & Rosander, 2018). O objetivo é que a infusão da estrutura *CRISP-DM* ao longo do projeto crie uma abordagem estruturada para um processo criativo de resolução de problemas (Jaggia et al., 2020).

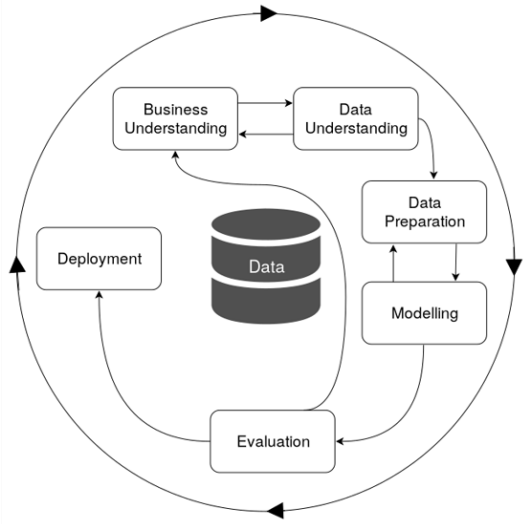


Figura 1-2: Metodologia *CRISP-DM*
(Martinez-Plumed et al., 2019).

1.4 Cronograma

No Quadro 1-1 é apresentado o cronograma do projeto, com as tarefas concretizadas assinaladas com X, nos meses em que se realizaram.

Quadro 1-1: Cronograma do projeto.

Programa de trabalho		**Meses**										
Fases	**Tarefas**	Fev	Mar	Abr	Maio	Jun	Jul	Ago	Set	Out	Nov	Dez
Revisão da literatura	Estudo do tema de *Process Mining* e dos principais algoritmos e ferramentas disponíveis neste âmbito.	x										
	Pesquisa de trabalhos e aplicações relacionadas com a utilização de *Process Mining* na área da saúde.	x	x									
	Estudo e aplicação da metodologia *CRISP-DM*, apresentada na Figura 1-2, orientada para mineração de processos.		x									
Problemas e Hipóteses	Estudar a hipótese de o *Process Mining* efetivamente ser um instrumento que ajude na gestão e melhoria de processos na área da saúde, nomeadamente, hospitalares, partindo do conjunto de dados *MIMIC-III*.			x	x							

Programa de trabalho		**Meses**										
Fases	**Tarefas**	Fev	Mar	Abr	Maio	Jun	Jul	Ago	Set	Out	Nov	Dez
Experiências / Análises	Estudo exploratório e comparativo de ferramentas de *Process Mining*, de forma a perceber as suas vantagens e desvantagens, particularmente *ProM*, *Disco* e o *framework PM4Py*, em *Python*.				x	x	x					
	Estudo comparativo de algoritmos, identificando limitações e utilidade da sua aplicação, de forma a identificar o melhor tipo de abordagem e modelos.				x	x	x					
Caso de estudo	Análise e exploração de um processo de saúde, cujos dados se encontram em *MIMIC-III*.						x	x	x	x	x	
Teoria / Artigo	Escrita da dissertação e artigos relacionados com a aprendizagem e resultados obtidos com a realização deste projeto.			x	x	x	x	x	x	x	x	x

1.5 Estrutura do documento

Este documento está organizado em mais cinco capítulos, para além desta introdução. Nesta é descrita a origem, o enquadramento e os objetivos que advêm do projeto proposto.
No segundo capítulo são abordados os principais temas abrangidos no âmbito deste projeto, *Process Mining* e as suas aplicações na área da saúde.
Segue-se o capítulo três, onde são apresentados os dois conjuntos de dados utilizados neste trabalho: os dados de teste e os dados para a aplicação no caso de estudo final.
No quarto capítulo, é apresentado o estudo exploratório e comparativo dos algoritmos e ferramentas de *Process Mining*, de forma a perceber as suas vantagens e desvantagens para o tipo de análise pretendida. A partir dos resultados dessas análises, são apresentadas as conclusões das comparações feitas aos algoritmos e ferramentas.
Com os dados recolhidos e o conhecimento adquirido na fase de testes, o capítulo cinco apresenta a aplicação real de *Process Mining* a dados de um serviço de urgências.
No sexto capítulo, os objetivos iniciais do trabalho são comparados com os resultados alcançados de forma a tirar conclusões acerca do trabalho realizado. Neste capítulo é feito um resumo do trabalho realizado e o enquadramento do trabalho na literatura atual.

2. Estado da Arte

Este capítulo apresenta o estado atual da arte respeitante a mineração de processos (*Process Mining*) e ainda como esta abordagem está a ser utilizada na área da saúde.

2.1 *Process Mining*

Na última década, a mineração de processos surgiu como um novo campo de pesquisa que se concentra na análise de processos, usando dados de eventos. As técnicas clássicas de mineração de dados, como classificação, clustering, regressão e aprendizagem de regras de associação não se concentram em modelos de processos de negócios e, geralmente, são usadas apenas para analisar uma etapa específica do processo geral (Van der Aalst, 2012). A mineração de processos concentra-se em processos, desde o início ao fim, e é possível devido à crescente disponibilidade de dados de eventos, às novas técnicas de descoberta de processos e à verificação de conformidade (Mans et al., 2015).
Os modelos de processo são usados para análise e execução por sistemas *BPM* / *WFM*, que são modelos de processo normalmente manuais, os quais não recorrem a dados de eventos. No entanto, as atividades executadas por pessoas, máquinas e *software* deixam rastos nos chamados registos de eventos (Van der Aalst, 2012). As técnicas de mineração de processos usam esses registos (*logs*) para descobrir, analisar e melhorar os processos de negócios (Batista & Solanas, 2019).
O *Process Mining* é implementado para atingir três objetivos (assinalados a vermelho na Figura 2-1): descoberta do processo, conformidade da implementação do processo e enriquecimento do processo (Mans et al., 2015).

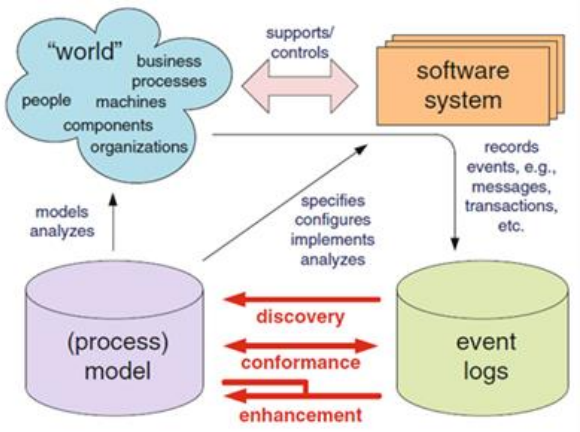


Figura 2-1: Modelo de *Process Mining* (Mans et al., 2015).

Como se pode verificar na Figura 2-2, a descoberta do processo revela os caminhos mais frequentes e as sequências incomuns, por meio da visualização do registo de eventos, recorrendo, por exemplo, a uma rede de *Petri*. Uma rede de *Petri* possui dois tipos de elementos, posições e transições. Uma posição pode conter um ou mais *tokens*. Uma transição é habilitada se todas as posições ligadas a si como entradas contiverem pelo menos um *token* (Desel & Juhás, 2001).

Na verificação da conformidade, o modelo de processo e os fluxos descobertos nos *logs* de eventos são analisados, e é verificado se o processo é realizado conforme identificado no seu modelo (Breitmayer, 2018). A verificação de conformidade mede, então, as diferenças entre o processo executado e a especificação do modelo, com o objetivo de identificar lacunas e/ou oportunidades de melhorar o modelo de processo, usando as informações obtidas no processo real (Batista & Solanas, 2019).

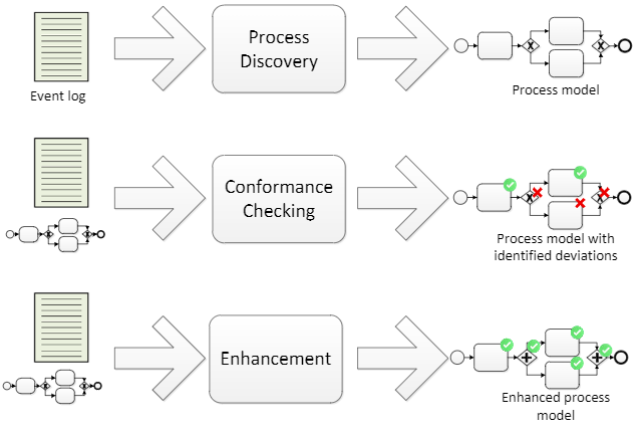


Figura 2-2: Tipos de *Process Mining* (Breitmayer, 2018).

Assim, a mineração de dados é usada para encontrar padrões e compreender as causas de certos comportamentos do processo, enquanto, por outro lado, ajuda a entender como estes estão a ser executados. Para o efeito, algoritmos de mineração especializados são aplicados para identificarem padrões a partir dos dados de evento que são registados no sistema de gestão de informações (Rojas et al., 2019).

2.2 Algoritmos de *Process Mining*

Existem vários algoritmos para mineração de processos. As relações locais internas entre os dados das atividades podem ser modeladas pelo algoritmo *Heuristics Miner* (Weijters et al., 2006). As relações globais ou externas entre as atividades podem ser modeladas pelos algoritmos *Genetic Miner* (Shinde & Rajeswari, 2018) e *Fuzzy Miner* (Wang et al., 2019). Este último é um algoritmo muito popular de descoberta, pois é capaz de gerar vários processos com diferentes graus de detalhe, sendo assim, comumente usado para compreender processos não estruturados, muito comuns também na área da saúde (Sundari & Nayak, 2020).
O algoritmo *Alpha* (Van der Aalst et al., 2004) examina o *log* de eventos à procura de padrões específicos. Este algoritmo trabalha, em simultâneo, um conjunto de sequências de eventos, que segue a relação de ordenação da atividade no *log* de eventos e mostra o resultado no diagrama de projetos da rede de *Petri*. Por exemplo, se a atividade X é seguida por Y, mas Y nunca é seguida por X, então assume-se que existe uma dependência causal entre X e Y. Todavia, o minerador *Alpha* não consegue destacar os gargalos do processo (Sundari & Nayak, 2020).
A maioria das ferramentas de mineração de processos comerciais usa *Directly-Follows Graphs* (*DFG*) como um primeiro meio para explorar os dados de eventos. Para lidar com a complexidade, os *DFG* são simplificados, removendo nós e ligações com base em limites de frequência. Esta simplicidade pode tornar esses *DFG* enganosos, pois podem ser interpretados erroneamente, levando a conclusões muito diferentes. Além disso, as informações de gargalo podem ser falsas, especialmente após a simplificação do modelo. Isto pode levar a vários tipos de problemas de interpretação, devido a "lacunas invisíveis" no modelo (Van der Aalst, 2019).
O resultado são gráficos onde os nós representam os eventos / atividades no registo e as ligações direcionadas estão presentes entre os nós se houver pelo menos um traço no registo onde o evento / atividade de origem é seguido pelo evento / atividade alvo. No topo dessas ligações direcionadas, é fácil representar métricas como frequência (contando o número de vezes que o evento / atividade de origem é seguido pelo evento / atividade de destino) e desempenho (a média de intervalo de tempo decorrido entre os dois eventos / atividades) (Van der Aalst, 2019).
Os algoritmos heurísticos contribuem com a ordem ou sequência das atividades e a ocorrência ou frequências dos eventos. Para tal, estes algoritmos encontram os caminhos frequentes e não frequentes no processo, sendo, desta forma, mais robustos em relação ao modelo apresentado (Weijters et al., 2006). Este é um dos algoritmos mais utilizados, principalmente devido à sua capacidade para lidar com dados ruidosos e incompletos, resultantes de problemas de qualidade de dados, como erros de registo, que se manifestam de forma infrequente no comportamento do processo (Conforti et al., 2015). O minerador heurístico usa conectores lógicos *XOR* e *AND* de relações de dependência, sendo o resultado deste minerador uma rede heurística que ajuda a visualizar o processo e a prever o fluxo (Sundari & Nayak, 2020).
O algoritmo genético usa um procedimento iterativo para interpretar o processo de evolução natural. As suas quatro etapas, inicialização, seleção, reprodução e término produzem um modelo de processo construído em matrizes casuais. Este lida com todos os tipos de dados, como atividades ruidosas, duplicadas, incompletas e ocultas (Sundari & Nayak, 2020).

O algoritmo *Fuzzy Miner* fornece a precisão máxima possível em medidas de desempenho. Este é um algoritmo complicado, dinâmico e complexo para resolver as atividades incomuns no *log* de eventos (Sundari & Nayak, 2020). O algoritmo *Fuzzy* é utilizado para desenvolver os modelos *Fuzzy* que determinam as métricas do *log* de eventos e agregam todo o processo. Portanto, pode produzir a visualização específica do processo e pode-se prever o risco no processo, sendo a matriz *Fuzzy* usada para encontrar a previsão no processo (Ma'arif, 2017).
O *Inductive Miner* é amplamente utilizado em diferentes áreas, com resultados muito promissores. Este algoritmo apresenta uma melhoria em relação aos mineradores *Alpha* e *Heuristics*, pois torna mais fácil explorar um *log* de eventos, sendo capaz de lidar com comportamentos pouco frequentes e grandes registos de eventos, enquanto garante solidez. O conceito básico do *Inductive Miner* é detetar um padrão nos *logs* e, em seguida, procurar esse padrão até que um caso base seja encontrado (Bogarín et al., 2018).
No Quadro 2-1 apresenta-se uma análise resumida dos principais algoritmos, de acordo com as suas características.

Quadro 2-1: Comparação entre algoritmos de *Process Mining* (Adaptado de Breitmayer, 2018).

	Alpha Miner	***Directly-Follows Graph***	***Heuristic Miner***	***Fuzzy Miner***	***Inductive Miner***
Descrição	Primeira abordagem de mineração que permite descobrir uma rede *Workflow* a partir de registos de eventos.	Usado como um primeiro meio para explorar os dados de eventos, mas podem ser enganosos.	Gera um modelo de processo com base em diferentes métricas de frequência.	O controle de significância permite ampliar e mostrar diferentes níveis de importância de etapas, do processo.	É uma melhoria em relação a *Alpha Miner* e *Heuristic Miner*.
Saída	Rede *Workflow*.	*Directly-Follows Graph* (*DFG*).	Rede heurística / casual.	Modelo *Fuzzy*.	Pode gerar vários tipos de modelos.
Desafios	Ruído; Dados incompletos; *Loop* envolvendo uma ou duas etapas; Escolha não livre.	As atividades que têm uma ordem flexível levam a *DFG* tipo *Spaghetti*[1] com *loops*, mesmo quando as atividades são executadas no máximo uma vez.	As regras de divisão e junção são consideradas apenas localmente, o que resulta em redes que não são sólidas.	O modelo *Fuzzy* não pode ser convertido; Sem diferenciação entre *XOR* e *AND*.	Gera um modelo sólido a partir de uma procura recursiva de padrões.
Resultado	As extensões são capazes de enfrentar alguns dos desafios.	As informações de desempenho podem ser enganosas; Tem problemas de interpretação devido a lacunas no modelo.	Pode minerar dependências longas com sucesso; Às vezes gera muitas dependências.	Pode lidar com ruído e dados incompletos; Pode minerar a maioria das dependências corretamente.	Gera um modelo que garante solidez.
Registos de evento	Não lida bem com dados incompletos ou com ruído.	Não lida bem com processo extensos, devido ao seu limite por frequência.	Pode, até certo ponto, lidar com dados incompletos.	Pode lidar com dados incompletos.	Lida com comportamentos infrequentes e grandes registos de eventos.

[1] Modelos altamente complexos que muitas vezes são difíceis de entender são chamados de modelos de espaguete, ou em inglês, spaghetti (Veiga & Ferreira, 2010).

Neste trabalho, optou-se por descrever o subconjunto de algoritmos mais usados e/ou disponíveis nas principais ferramentas exploradas. Além dos algoritmos descritos, encontram-se muitos outros na literatura, como sendo: *Genetic Algorithms* (Sundari & Nayak, 2020); *Trace Clustering* (Günther et al., 2008); *Region Miner* (Lang et al., 2008); *Palia algorithm*, orientado a atividades (Conca et al., 2018); *DWS* (*Day–Stout–Warren*) *algorithm* (Lang et al., 2008); *Multiphase Miner* (Lang et al., 2008); métodos de reconhecimento de padrões (Huang et al., 2013), e métodos probabilísticos usando modelos de *Markov* ocultos (Baker et al., 2017). Um modelo de *Markov* é usado para modelar sistemas que mudam aleatoriamente. Supõe-se que os estados futuros dependem apenas do estado atual, não dos eventos que ocorreram antes dele (Moerland, 1996).

2.3 Ferramentas de *Process Mining*

A nível de ferramentas para aplicação dos algoritmos, existem, atualmente, várias ferramentas de *software*, de código aberto e comerciais. O *software ProM* e *Apromore* são exemplos de *software* de código aberto. Os *software Disco*, *Celonis*, *ProcessGold* e *RapidMiner* são exemplos de *software* comerciais (Gatta et al., 2018). Na literatura, as ferramentas mais amplamente usadas são *ProM* (Van Dongen et al., 2005), *Disco* (Lohmann, 2012) e *Celonis* (Badakhshan et al., 2020).

ProM é uma *framework* extensível de código aberto que suporta uma ampla gama de algoritmos de mineração de processos como *plug-ins*, Figura 2-3. É claramente a ferramenta mais usada (Batista & Solanas, 2019). A *framework* é flexível, no que diz respeito ao formato de entrada e saída, pois suporta vários formatos, e.g. redes de *Petri*, redes sociais (Van der Aalst & Song, 2004), entre outras. Os *plug-ins* podem ser usados de várias formas e combinados para serem aplicados em situações da vida real (Van Dongen et al., 2005).

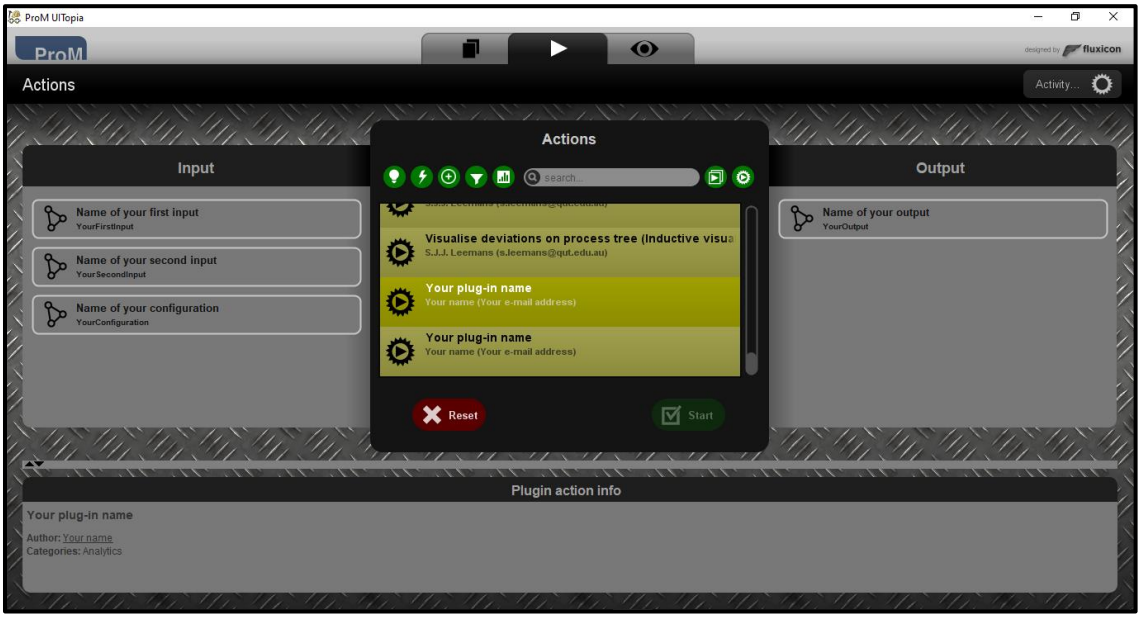


Figura 2-3: Interface *ProM* para utilização dos *plug-ins*.

Existem cinco tipos de *plug-ins* (Van Dongen et al., 2005):

- *Plug-ins* de mineração: implementam algum algoritmo de mineração, resultando num tipo de visualização, e.g., uma rede de *Petri*, uma rede social, entre outras;
- *Plug-ins* de exportação: implementam funcionalidades para guardar alguns objetos. Por exemplo, existem *plug-ins* para salvar redes de *Petri* como imagem, entre outros;
- *Plug-ins* de importação: são capazes de lidar com grandes conjuntos de dados e classificam os eventos por casos, pelos carimbos de data/hora, antes do início da mineração real. Implementam, ainda, funcionalidades para carregar uma grande variedade de modelos, desde uma rede de *Petri* a objetos que foram anteriormente exportados;
- *Plug-ins* de análise: geralmente implementam alguma análise que pode ser aplicada a um resultado de mineração. Por exemplo, existem *plug-ins* de análise para comparar um registo e um modelo, ou seja, verificação de conformidade. Outros, também de especial interesse, são os *plug-ins* de filtragem, para análises mais eficazes e customizadas;
- *Plug-ins* de conversão: implementam conversões entre formatos de dados diferentes, por exemplo, entre dois tipos de visualização de modelos.

O *software Disco* apresenta-se como uma solução proprietária e comercial com funcionalidades estendidas e melhoradas (Batista & Solanas, 2019). Esta ferramenta surge do facto dos analistas de processo precisarem de uma ferramenta que, acima de tudo, torne a mineração de processos fácil e rápida (Lohmann, 2012).
O *Disco* foi projetado para tornar a importação de dados realmente fácil, detetando automaticamente o tipo dos vários campos dos registos, de forma a carregar os conjuntos de dados com a máxima rapidez. Assim, como se depreende pela Figura 2-4, basta abrir um documento *Comma-separated values* (*CSV*) e, se necessário, configurar quais colunas contêm o *caseID*, *timestamps*, nomes de atividades e outros atributos que devem ser incluídos na análise, e a importação pode ser iniciada. De salientar que os conjuntos de dados são importados em modo somente de leitura. Portanto, os documentos originais não podem ser modificados, o que é importante, por exemplo, para auditorias.

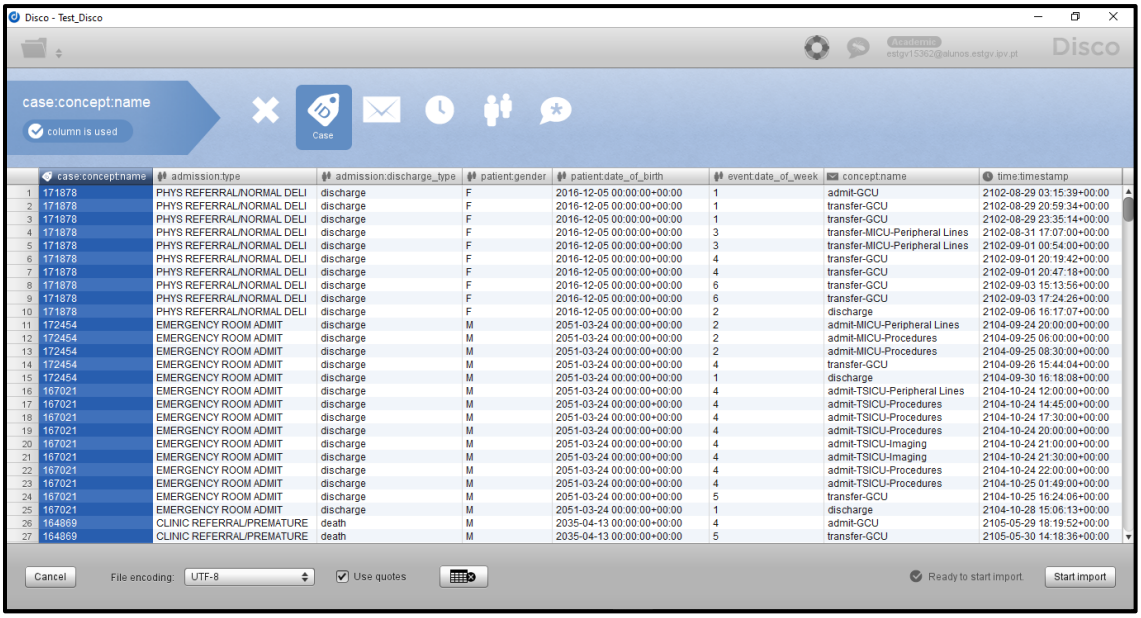


Figura 2-4: Interface do *Disco* para importação dos dados.

A principal funcionalidade da mineração de processos é a descoberta automatizada de mapas de processos. Depois da importação finalizada, aparece automaticamente o modelo do processo, onde se pode ver de forma rápida e objetiva como este foi realmente executado. Analisando a Figura 2-5, pode-se perceber que a espessura dos caminhos e a coloração das atividades mostram os principais caminhos dos fluxos de processo. Os *loops* desnecessários e casos pontuais são rapidamente descobertos.

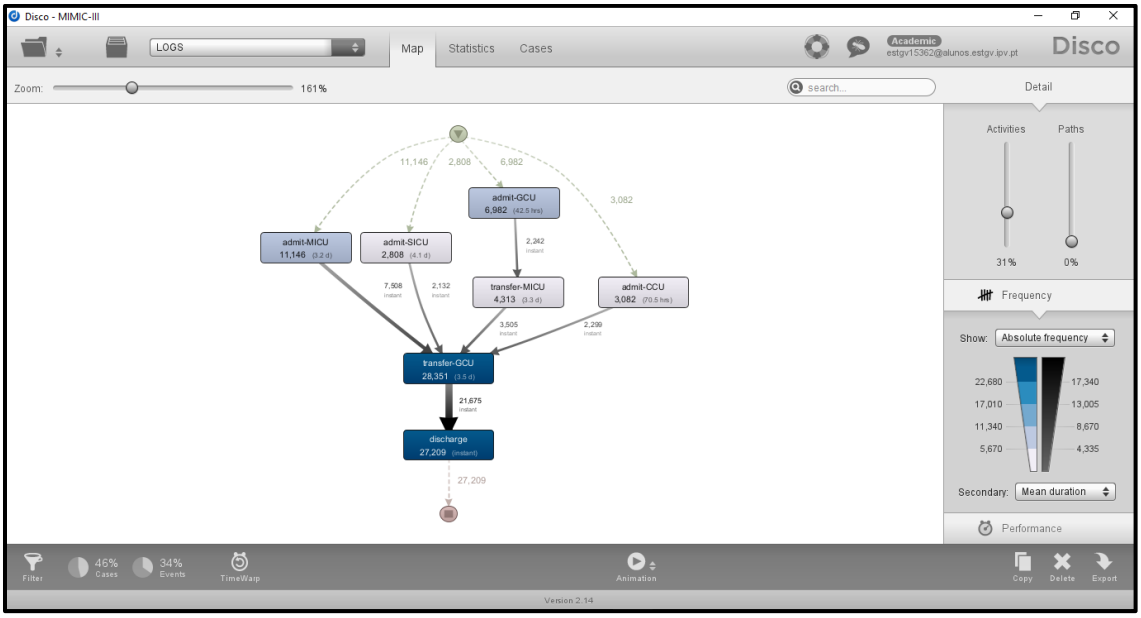


Figura 2-5: Interface do *Disco* para descoberta do processo.

Este *software* permite, ainda, obter informações gerais sobre o número de casos e eventos no conjunto de dados, o período em que os eventos ocorreram e gráficos de desempenho como, por exemplo, sobre a duração do caso, Figura 2-6. Outras visualizações de estatísticas fornecem

informações de frequência e desempenho para todas as atividades e recursos no processo. Além disso, existem estatísticas para qualquer coluna de atributo de dados adicional que seja incluída no conjunto de dados.

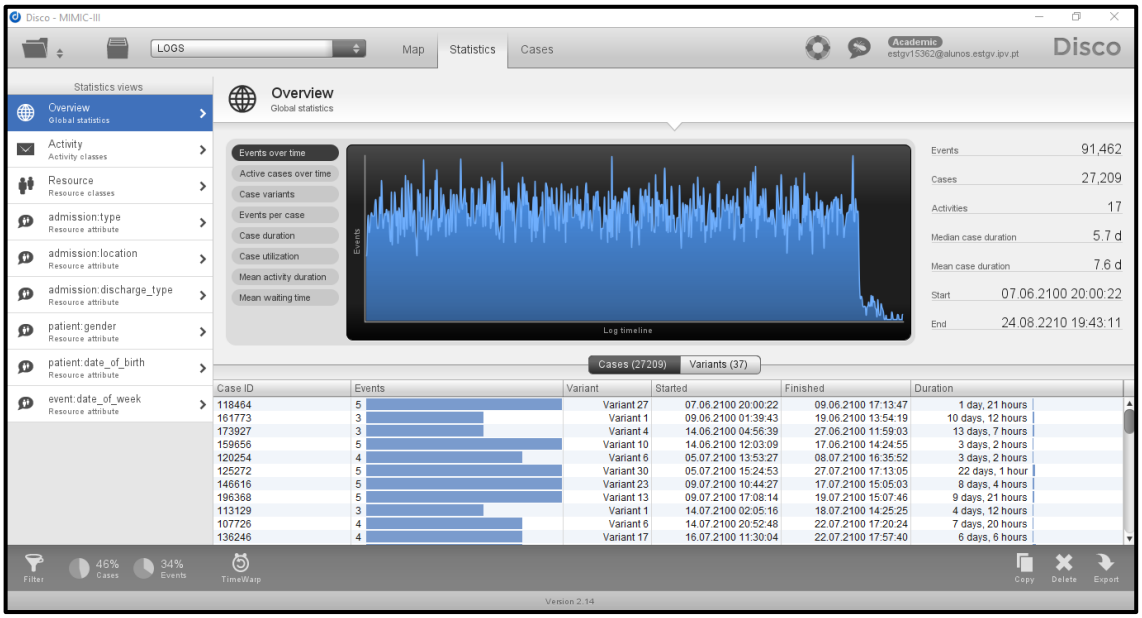


Figura 2-6: Interface do *Disco* para visualização das estatísticas dos dados.

Uma outra funcionalidade de elevada importância é obter acesso direto às variantes no processo, para além de uma lista completa de todos os casos no conjunto de dados. A Figura 2-7 mostra a interface do *Disco* que permite essa análise onde, para além da lista das variantes, apresenta estatísticas sobre cada variante. As variantes são parte integrante da análise do processo. Uma variante é uma sequência específica de atividades que pode ser vista como um caminho do início ao fim do processo. No mapa do processo, uma variante é então um caminho executado pelo processo do início ao fim. Normalmente, uma grande parte dos casos no conjunto de dados segue apenas algumas variantes, sendo útil saber quais são.

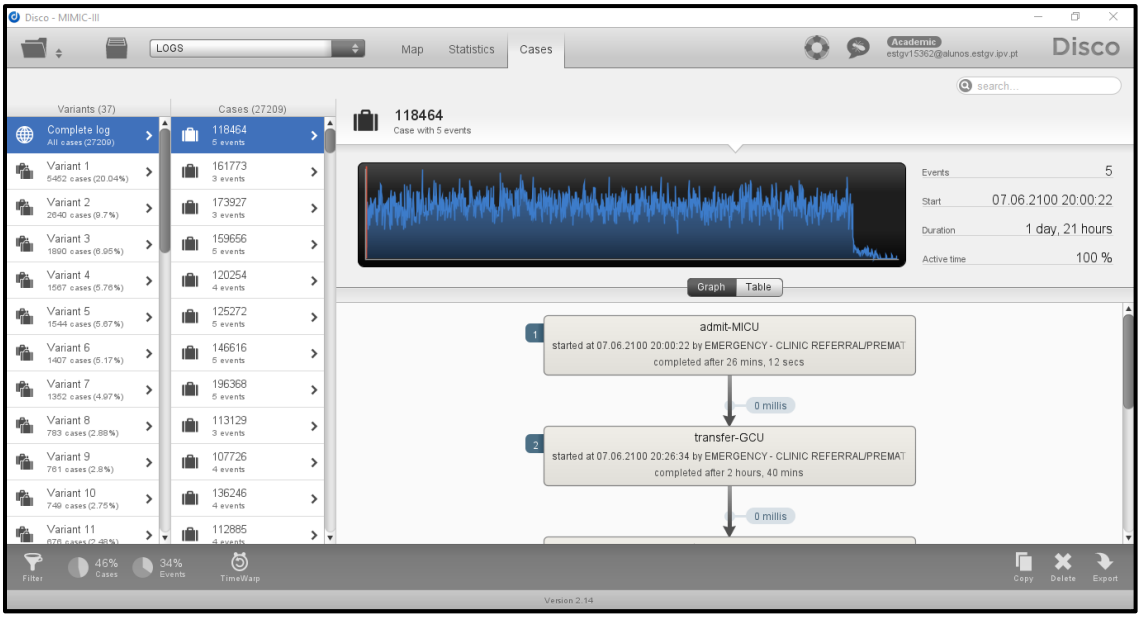


Figura 2-7: Interface do *Disco* para analise das variantes do processo.

O *Disco* oferece, ainda, recursos de filtragem. Esses filtros são rapidamente acessíveis de qualquer visualização e fáceis de configurar, Figura 2-8. Esses recursos de filtragem permitem explorar rápida e interactivamente várias direções e respondem a perguntas concretas sobre o processo.

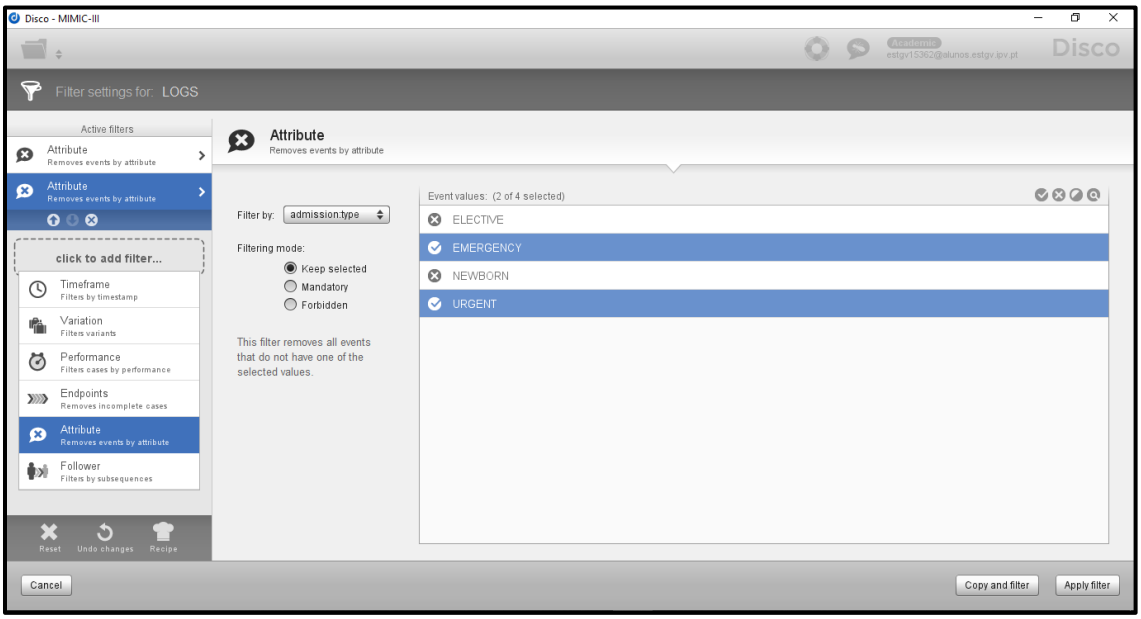


Figura 2-8: Interface do *Disco* para aplicação dos filtros.

Existem cinco tipos de filtros que podem ser combinados e acumulados:

- O filtro *Timeframe*: controlos de tempo intuitivos para selecionar casos e eventos com base numa janela de tempo;
- O filtro de variação: permite focar a análise no comportamento principal ou em casos excecionais, fazendo uso das variantes;

- O filtro de desempenho: foca-se em casos baseados numa variedade de diferentes métricas de desempenho como, por exemplo, a duração do caso;
- O filtro *Endpoints*: seleciona casos com base nas suas atividades de início e fim;
- O filtro Atributo: seleciona ou exclui certas atividades, recursos ou categorias de processo com base em atributos de dados.

O *Celonis* tem origem na necessidade de criar um ambiente integrado e centralizado para documentar e manter os processos de negócios, analisando o desempenho e a conformidade dos seus processos, bem como a possibilidade de integração com ferramentas de *BPM* (Badakhshan et al., 2020).
Já o *RapidProM* (Van der Aalst et al., 2017) permite a execução repetida de cenários de larga escala com algoritmos de mineração de processo no *RapidMiner*, que é uma plataforma de *software* de ciência de dados. No entanto, não fornece personalização algorítmica simples nem uma maneira fácil de integrar algoritmos personalizados desenvolvidos. Esta ferramenta é disponibilizada de forma paga, tendo uma edição grátis limitada a 10.000 linhas de dados.

Todas as ferramentas mencionadas falham no suporte de algoritmos personalizáveis de mineração de processos e na experimentação e análise em larga escala (Berti et al., 2019).
Numa perspetiva dum *software* de mineração de processo que seja facilmente extensível, que permita a personalização algorítmica e permita conduzir facilmente experiências em larga escala, existe a *framework* de *Process Mining for Python* (*PM4Py*). Proporciona uma perspetiva de integração em aplicações de grande escala, através de uma nova biblioteca de mineração de processos, com integração com bibliotecas de ciência de dados de última geração, como por exemplo, *pandas*, *numpy*, *scipy* e *scikit-learn* (Berti et al., 2019).
As principais vantagens da biblioteca *PM4Py* consistem em reduzir a barreira para o desenvolvimento algorítmico e personalização, ao realizar uma análise de mineração de processos, em comparação com ferramentas existentes, e em permitir a fácil integração de algoritmos de mineração de processos com algoritmos de outras áreas de ciência de dados, implementados em vários pacotes *Python* de última geração.
O *PM4Py* fornece suporte para diferentes tipos de estruturas de dados de eventos, designadamente, o registo de eventos. São ainda fornecidas funcionalidades de conversão para transformar objetos de dados de eventos de um formato para outro. Além disso, o *PM4Py* suporta o uso de *frames* de dados do *pandas*, que são eficientes no caso de usar dados de eventos maiores. Outros objetos atualmente suportados pelo *PM4Py* incluem: redes heurísticas, redes de *Petri*, árvores de processo e sistemas de transição. Uma árvore de processo é um diagrama esquemático das etapas pelas quais um produto passa durante o seu ciclo de vida. Essa ferramenta determina uma maneira de ordenar e priorizar processos por vários níveis que, juntos, criam uma hierarquia de processos com relacionamentos apropriados entre os componentes (Van Zelst & Leemans, 2020). Já um sistema de transição é usado para descrever o comportamento potencial de sistemas discretos. Consiste de estados e transições entre estados (Keller, 1976).

Em acréscimo, o *PM4Py* fornece várias técnicas principais de mineração de processo, incluindo:

- Descoberta de processo: a partir dos algoritmos *Alpha Miner* e *Inductive Miner*;
- Verificação de conformidade: a partir da reprodução e alinhamentos baseados em *token* (Adriansyah et al., 2011);
- Medição de adequação, precisão, generalização e simplicidade dos modelos de processo;
- Filtragem com base no intervalo de tempo, desempenho de caso, pontos finais de rastreamento, variantes de rastreamento, atributos e caminhos;
- Gestão de caso: estatísticas sobre variantes e casos;
- Gráficos: duração do caso, eventos por tempo, distribuição de valores de atributos numéricos;
- Análise de Redes Sociais (Van der Aalst & Song, 2004): *handover* de trabalho, trabalho em conjunto, subcontratação e redes de atividades afins.

O *PM4Py* inclui, também, bibliotecas de visualização como, *GraphViz*, para representação de gráficos de sequência direta, redes de *Petri*, sistemas de transição, árvores de processo, *NetworkX*, para representação estática de redes sociais e *Pyvis*, para representação dinâmica de rede social baseada na *web*. A Figura 2-9 mostra um exemplo de rede de *Petri* criada no *PM4Py*. A visualização do modelo gerado mostra várias informações visuais como, ponto de início (círculo verde) e fim (círculo a amarelo) do modelo. Pela coloração das atividades pode-se perceber que quanto mais escura, mais ocorrências tem e a existência de transições ocultas (caixas escuras) utilizadas por alguns algoritmos. Apresenta ainda, nas atividades e ligações, o número de ocorrências que tiveram.

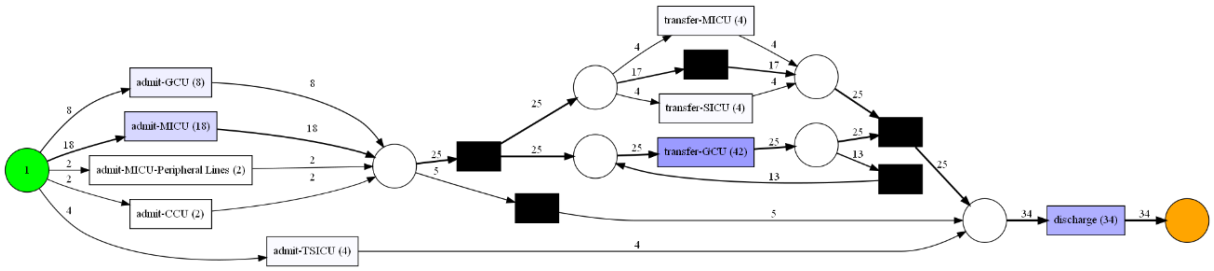


Figura 2-9: Rede de *Petri* resultante da descoberta do processo no *PM4Py*.

Antes de aplicar as técnicas e ferramentas de mineração de processos, dado que os *logs* de eventos são geralmente ruidosos e incompletos, há uma necessidade de técnicas de pré-processamento para alcançar resultados consistentes. Para tratar esse problema, é comum remover eventos imprecisos ou inconsistentes (Diba et al., 2020). Para além disso, a complexidade é outra razão para aplicar o pré-processamento, de forma a filtrar eventos cujas atividades são mais importantes ou para agrupar atividades semelhantes. Além disso, quando são usados *logs* de vários sistemas de informação, é comum a necessidade de fases relacionadas à integração e padronização das atividades (Batista & Solanas, 2019).

2.4 *Process Mining* na saúde

No que respeita à aplicação de *Process Mining* na área da saúde, na maioria das pesquisas existentes usaram-se dados recolhidos de hospitais para obter uma imagem mais ampla dos processos médicos e combinando registos de *logs* de várias fontes que permitem a comparação de resultados de várias instituições (Helm & Küng, 2016). Já para ambientes mais específicos, é relevante a quantidade de estudos realizados com dados de serviços de emergência e processos relacionados a emergências, dada a importância de gerir esses processos com a maior precisão possível (Rojas et al., 2019). Outros estudos concentraram-se em dados neonatais, unidades de terapia intensiva (Catley & James, 2011), lares de idosos (Fernández-Llatas et al., 2013), laboratórios dentários (Mans et al., 2012), entre outros. A Figura 2-10 apresenta as percentagens de aplicação de *Process Mining* em cada ambiente acima descrito (Batista & Solanas, 2019).

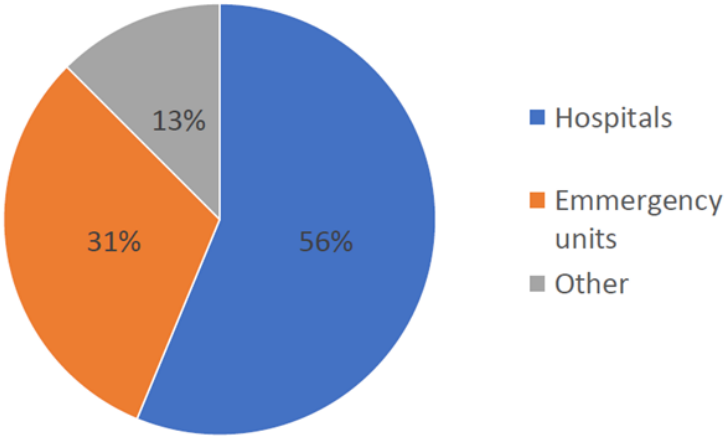


Figura 2-10: Tipos de registos usados em *Process Mining* (Batista & Solanas, 2019).

A heterogeneidade e complexidade do setor de saúde resulta numa grande variedade de dados médicos. O cenário mais comum é aquele em que os eventos estão associados às atividades realizadas durante o tratamento dos pacientes (Kurniati & Hall, 2020). Existe uma variedade de campos médicos onde é utilizado o *Process Mining*, sendo a mais encontrada a pesquisa relacionada com o cancro (Kurniati et al., 2016). Da mesma forma, a necessidade de respostas rápidas e procedimentos eficientes em serviços de emergência promove um uso significativo de *Process Mining*. Existem outros campos médicos onde é frequente a aplicação da mineração de processos como sendo a cirurgia, cardiologia e diabetes (Batista & Solanas, 2019). Segue-se a descrição sucinta de alguns estudos da área da saúde que usaram mineração de processos.
(Kurniati et al., 2018) e (Baker et al., 2017) focaram-se no tratamento de pacientes com cancro e na sequência de etapas administrativas nas vias de atendimento.
A metodologia seguida na pesquisa em (Kurniati et al., 2018), é o modelo de ciclo de vida *L**, sugerida para projetos típicos de mineração por processos. Esta metodologia começa com o planeamento que envolve a identificação de questões de pesquisa como um ponto de partida para a investigação. A fase posterior consiste na extração para construir o *log*, aplicando os critérios de seleção para recolher registos dos pacientes necessários. São, ainda, incluídas atividades de pré-processamento como criação de visualizações, agregação de eventos,

enriquecimento de *logs* e filtragem de *logs*. Seguidamente, os subconjuntos de dados extraídos são analisados usando várias abordagens de mineração de processos. Como resultados, a informação de tempo é utilizada para analisar os tempos de espera pelas vias de admissão de todos os pacientes com cancro.

Já em (Baker et al., 2017), de forma a analisar o impacto de um serviço de monitorização de hemograma de autoteste do paciente em pacientes em quimioterapia, o objetivo foi estabelecer métodos reproduzíveis de processamento de registos eletrónicos de saúde. Os resultados serviram para definir e quantificar as vias do paciente durante a quimioterapia e reunir dados robustos que são estruturados para serem capazes de informar um modelo de decisão de custo-efetividade de monitorização doméstico do estado de neutropenia[2] durante a quimioterapia. O resultado mostrou que a maioria dos pacientes não seguia o caminho idealizado no planeamento dos seus cuidados. Com estes dados, foi possível compreender melhor os caminhos clínicos na realidade, o que permitiu contribuir para a garantia da qualidade dos dados, identificar necessidades não atendidas e, principalmente, melhorar o atendimento ao paciente e os resultados dos tratamentos.

Num outro tipo de cenário, os artigos (Alvarez et al., 2018), (Rojas et al., 2017) e (Perimal-Lewis et al., 2016) mostram aplicações de *Process Mining* em ambientes de urgências ou emergências.

Assim, em (Alvarez et al., 2018), foi estudada a colaboração entre os diferentes profissionais de saúde nos processos de Serviços de Urgências, onde é fundamental o atendimento imediato dos pacientes que chegam ao hospital em estado delicado, exigindo atendimento rápido. Como tal, os objetivos deste estudo passaram por: descobrir modelos de interação de papéis em processos, usando técnicas de mineração de processos; compreender como os profissionais de saúde estavam a colaborar; fornecer conhecimento útil para ajudar a melhorar os processos de urgência. Os resultados mostram uma maneira útil de fornecer perceções relevantes sobre como os profissionais de saúde colaboram, descobrindo oportunidades de melhoria de processos.

No artigo (Rojas et al., 2017), os especialistas de emergência obtiveram uma melhor compreensão de como estavam a lidar com episódios relacionados com patologias específicas, gravidade da triagem e destinos de alta de pacientes. Para chegar a esse resultado foi utilizada a ferramenta *Disco* para gerar um modelo de processo, através da qual foi possível identificar tarefas e subprocessos que compõem os episódios. Devido à importância médica, a análise focou-se apenas no diagnóstico e no tratamento. Com base nos modelos obtidos com o *Disco*, acerca desses processos, os especialistas puderam proceder à sua avaliação.

Por fim, em (Perimal-Lewis et al., 2016), a metodologia e as ferramentas do *Process Mining* foram utilizadas para avaliar a qualidade dos dados do Departamento de Emergência com base no tempo, provenientes de registos eletrónicos de saúde. Esta pesquisa foi realizada trabalhando em estreita colaboração com os especialistas do domínio para validar os modelos de processo. O modelo do percurso do paciente do hospital foi usado para avaliar anormalidades de fluxo

[2] A neutropenia é uma disfunção do sangue caracterizada por uma diminuição no número de neutrófilos, a célula branca mais importante no sangue (Vaillant & Zito, 2021).

que resultaram de dados incorretos de carimbo de data e hora usados em métricas de desempenho baseadas em tempo.

Em resumo, a mineração de processos mostra-se uma metodologia viável para avaliar a qualidade dos dados de métricas de desempenho hospitalares, baseadas no tempo, e esse conhecimento permite que ações corretivas apropriadas sejam implementadas para resolver os problemas de qualidade de dados.

3. Cenário de Estudo

Nos últimos anos, tem havido um enorme aumento de sistemas de registo de saúde digital em hospitais. Apesar desse avanço, a interoperabilidade de sistemas digitais permanece um problema em aberto, levando a desafios na integração de dados. Como resultado, o potencial que os dados hospitalares oferecem em termos de compreensão e melhoria do atendimento ainda não é totalmente utilizado (Johnson et al., 2016).

Os dados usados neste trabalho são de uma base de dados, denominada *MIMIC-III* (*Medical Information Mart for Intensive Care III*). Esta é uma grande base de dados disponível gratuitamente que inclui dados, não identificados, relacionados à saúde e relativos a mais de quarenta mil pacientes que permaneceram em unidades de cuidados intensivos no *Beth Israel Deaconess Medical Center* entre 2001 e 2012. A natureza aberta dos dados permite que os estudos clínicos sejam reproduzidos e melhorados de uma maneira que, de outra forma, não seriam possíveis (Johnson et al., 2016).

Numa primeira fase deste estudo, foram utilizados os dados da versão demo do *MIMIC-III*, para cenários de testes. Já numa segunda fase e para a aplicação no caso de estudo final, já foram usados os dados da versão completa. Neste capítulo será, inicialmente, apresentada toda a pesquisa e estudo feito aos dados e, posteriormente, são apresentadas as situações utilizadas neste trabalho.

3.1 Estudo e acesso dos dados

Nesta secção, é apresentado o acesso aos dados *MIMIC-III*, de forma a conhecer os dados utilizados, desde a sua recolha e tratamento até à forma de acesso.

Quanto à recolha dos dados, o *MIMIC-III* oferece suporte a uma ampla gama de estudos analíticos que abrangem epidemiologia, melhoria de regras de decisão clínica e desenvolvimento de ferramentas eletrónicas. É notável por três fatores (Johnson et al., 2016):

- Está disponível gratuitamente para investigadores em todo o mundo;
- Abrange uma população diversificada e muito grande de pacientes de cuidados intensivos;
- Contém dados de alta resolução temporal, incluindo resultados de laboratório, documentação eletrónica, tendências e formas de onda de monitores de cabeceira.

A base de dados *MIMIC-III* foi preenchida com dados que foram adquiridos durante o atendimento hospitalar de rotina, de modo que não houve sobrecarga associada aos profissionais de saúde e nenhuma interferência no seu fluxo de trabalho. Os dados foram captados de várias fontes, incluindo (Johnson et al., 2016):

- Ficheiros de sistemas de informação de cuidados intensivos;
- Bases de dados de *software* eletrónico de hospitais;
- Ficheiro mestre de morte da administração da previdência social.

Dois sistemas diferentes de informações de cuidados intensivos estavam em vigor durante o período de recolha dos dados, e esses sistemas foram a fonte de dados clínicos, tais como (Johnson et al., 2016):

- Medições fisiológicas verificadas com carimbo de data/hora (por exemplo, documentação horária de frequência cardíaca, pressão arterial ou frequência respiratória);
- Notas de progresso documentadas pelos profissionais;
- Medicamentos de gotejamento intravenoso contínuo e equilíbrio de fluidos.

Foram recolhidas, ainda, informações adicionais de sistemas de registos de saúde de hospitais e laboratórios, incluindo (Johnson et al., 2016):

- Dados demográficos dos pacientes e mortalidade hospitalar;
- Resultados de testes de laboratório (por exemplo, resultados de hematologia, química e microbiologia);
- Resumos de alta e relatórios de estudos de eletrocardiograma e de imagem;
- Informações relacionadas com faturação, como a Classificação Internacional de Doenças, códigos da 9ª Edição (*CID-9*), códigos de Grupo Relacionado a Diagnósticos (*DRG*) e códigos de Terminologia Processual Atual (*CPT*).

Antes dos dados serem incorporados na base de dados *MIMIC-III*, eles foram primeiro desidentificados, de acordo com os padrões da Lei de Responsabilidade e Portabilidade de Seguro Saúde (*HIPAA*), usando limpeza de dados estruturados e mudança de data (Johnson et al., 2016).

Em particular, as datas foram alteradas para o futuro por um deslocamento aleatório para cada paciente individual de uma maneira consistente para preservar os intervalos, resultando em estadias que ocorrem em algum momento entre os anos 2100 e 2200. Também as datas de morte foram alteradas para uma data futura (Johnson et al., 2016).

A mudança de data preservou o seguinte (Johnson et al., 2016):

- Hora do dia - uma medição feita às 15:00:00 foi, na realidade, a essa hora;
- Dia da semana - uma medição feita num domingo aparecerá num domingo no futuro;
- Sazonalidade - uma medição feita durante os meses de inverno aparecerá durante um mês de inverno no futuro.

A mudança de data removeu o seguinte:

- Ano - o ano é distribuído aleatoriamente entre 2100 e 2200;
- Dia do mês - o dia absoluto do mês não é preservado;
- Informações interpacientes - dois pacientes nos cuidados intensivos em 2150-01-01 não estavam na unidade ao mesmo tempo.

As datas de nascimento não são datas verdadeiras, as que ocorrem antes do ano 1900 aparecem se o paciente tiver mais de 89 anos. Nesses casos, a idade do paciente na primeira admissão foi fixada em 300 (Johnson et al., 2016).

Para além disso, as informações protegidas de saúde foram removidas de campos de texto livre, como relatórios de diagnóstico e notas médicas, usando um sistema de desidentificarão avaliado rigorosamente com base em pesquisas extensas de dicionário e correspondência de padrões com expressões regulares (Johnson et al., 2016).

Analisando a estrutura da base de dados, o *MIMIC-III* é uma base de dados relacional que consiste em 26 tabelas. As tabelas são ligadas por identificadores que geralmente possuem o sufixo *'ID'*. Por exemplo, *SUBJECT_ID* refere-se a um único paciente, *HADM_ID* refere-se a uma única admissão no hospital e *ICUSTAY_ID* refere-se a uma única admissão numa unidade de terapia intensiva. Uma exceção é *ROW_ID*, que é simplesmente um identificador de linha exclusivo para uma tabela em questão (Johnson et al., 2016). Este esquema pode ser visto em detalhe no Anexo 1.

Eventos mapeados como notas, testes de laboratório e balanço de fluidos são armazenados numa série de tabelas de 'eventos'. Por exemplo, a tabela *OUTPUTEVENTS* contém todas as medidas relacionadas à saída de um determinado paciente, enquanto a tabela *LABEVENTS* contém os resultados dos testes laboratoriais de um paciente (Johnson et al., 2016).

As tabelas prefixadas com *'D_'* são tabelas de dicionário e fornecem definições para identificadores. Por exemplo, cada linha de *CHARTEVENTS* está associada a um único *ITEMID* que representa o parâmetro medido, mas não contém o nome real da medição. Ao juntar *CHARTEVENTS* e *D_ITEMS* pelo *ITEMID*, é possível identificar o parâmetro representado por um determinado *ITEMID* (Johnson et al., 2016).

O desenvolvimento do modelo de dados *MIMIC* envolveu o equilíbrio entre a simplicidade de interpretação e a proximidade com a verdade fundamental. Como tal, o modelo é um reflexo das fontes de dados subjacentes, modificadas em resposta ao *feedback* dos utilizadores. Houve cuidado para evitar fazer suposições sobre os dados subjacentes ao realizar as transformações, de modo que o *MIMIC-III* representa, o mais próximo possível, os dados brutos do hospital (Johnson et al., 2016).

As tabelas a seguir são usadas para definir e rastrear estadias de pacientes (Johnson et al., 2016):

- *ADMISSIONS*: cada hospitalização é única para cada paciente na base de dados (define *HADM_ID*);
- *CALLOUT*: fornece informações de quando um paciente estava pronto para a alta da unidade e quando o paciente realmente teve alta da unidade;
- *ICUSTAYS*: informações sobre uma única estadia numa unidade (define *ICUSTAY_ID*);
- *PATIENTS*: cada paciente único na base de dados (define *SUBJECT_ID*);

- *SERVICES*: lista dos serviços sob os quais um paciente passou;
- *TRANSFERS*: movimento do paciente de unidade em unidade dentro do hospital, incluindo admissão e alta de cada unidade.

Cada *ICUSTAY_ID* corresponde a um único *HADM_ID* e *SUBJECT_ID*. Cada *HADM_ID* corresponde a um único *SUBJECT_ID*. Um único *SUBJECT_ID* pode corresponder a vários *HADM_ID* (várias hospitalizações do mesmo paciente) e vários *ICUSTAY_ID* (várias unidades permanecem na mesma hospitalização ou em várias hospitalizações, ou ambos) (Johnson et al., 2016).

As tabelas a seguir contêm dados recolhidos na unidade de terapia intensiva (Johnson et al., 2016):

- *CAREGIVERS*: todos os profissionais que registaram dados na base de dados (define *CGID*);
- *CARTEVENTES*: contém todos os dados gráficos disponíveis para um paciente, durante a permanência numa unidade;
- *DATETIMEEVENTS*: todas as observações registadas que são datas, por exemplo, tempo de diálise;
- *INPUTEVENTS_CV*: dados de entrada dos pacientes usando o sistema *Philips CareVue*;
- *INPUTEVENTS_MV*: dados de entrada dos pacientes usando o sistema *iMDSoft Metavision*;
- *NOTEEVENTES*: notas desidentificadas, incluindo notas de enfermagem e médicos, relatórios de imagens e resumos de alta;
- *OUTPUTEVENTS*: informações de saída para os pacientes duma unidade;
- *PROCEDUREEVENTS_MV*: procedimentos do paciente que foram monitorizados na unidade usando o sistema *iMDSoft MetaVision*.

As tabelas a seguir contêm dados recolhidos no sistema de registo do hospital:

- *CPTEVENTES*: procedimentos registados como códigos de Terminologia Processual Atual (*CPT*);
- *DIAGNOSES_ICD*: diagnósticos atribuídos ao hospital, codificados usando o sistema de Classificação Estatística Internacional de Doenças e Problemas Relacionados à Saúde (*CID*);
- *DRGCODES*: grupos Relacionados a Diagnósticos (*DRG*), que são usados pelo hospital para fins de cobrança;
- *LABEVENTES*: medições laboratoriais para pacientes dentro do hospital e em clínicas externas;
- *MICROBIOLOGYEVENTS*: medidas microbiológicas e sensibilidades, da base de dados do hospital;
- *PRESCRIPTIONS*: medicamentos solicitados, não necessariamente administrados, para um determinado paciente;
- *PROCEDURES_ICD*: procedimentos do paciente, codificados usando o sistema de Classificação Estatística Internacional de Doenças e Problemas Relacionados à Saúde (*CID*).

As seguintes tabelas são dicionários:

- *D_CPT*: dicionário de alto nível de códigos de Terminologia Processual Atual (*CPT*);
- *D_ICD_DIAGNOSES*: códigos do Dicionário de Classificação Estatística Internacional de Doenças e Problemas Relacionados à Saúde (*CID*) relativos a diagnósticos;
- *D_ICD_PROCEDURES*: códigos do Dicionário de Classificação Estatística Internacional de Doenças e Problemas Relacionados à Saúde (*CID*) relativos a procedimentos;
- *D_ITEMS*: dicionário de *ITEMID´s* que constam na base de dados *MIMIC*, exceto aqueles que se relacionam com testes de laboratório;
- *D_LABITEMS*: dicionário de *ITEMID´s* na base de dados do laboratório que se relacionam com os testes de laboratório.

Finalmente, quanto ao acesso aos dados, como a base de dados contém informações detalhadas sobre o atendimento clínico dos pacientes, deve ser tratado com o devido cuidado e respeito. Assim, os investigadores são obrigados a solicitar formalmente o acesso por meio de um processo documentado no site do *MIMIC* e são obrigados a declarar não haver interesses financeiros concorrentes (Johnson et al., 2016).
Existem duas etapas principais que devem ser concluídas antes que o acesso seja concedido ao investigador (Johnson et al., 2016):

- Concluir o curso "Dados ou somente pesquisas de espécimes" fornecido pelo *CITI*, reconhecido em proteção de participantes de pesquisa com seres humanos que inclua os requisitos da Lei de Responsabilidade e Portabilidade de Seguro Saúde (*HIPAA*). No Anexo 2 e Anexo 3 podem ser vistos, respetivamente, o certificado e o relatório de conclusão deste curso;
- Assinar um acordo de uso de dados, que descreve o uso de dados e padrões de segurança apropriados, e proíbe esforços para identificar pacientes individuais.

A aprovação requer a submissão do formulário de conclusão do curso *CITI*. É necessário mencionar o porquê do interesse em aceder aos dados e adicionar um nome de referência, alguém que confirme a veracidade do pedido (Johnson et al., 2016).
Assim que o pedido for aprovado, o investigador recebe um e-mail com instruções de acesso à base de dados do *PhysioNetWorks*, um componente de acesso restrito do *PhysioNet* (Goldberger et al., 2000).

3.2 Testagem de ferramentas e algoritmos

Esta secção descreve a recolha e processamento feito aos dados para realizar o estudo exploratório e comparativo das ferramentas de *Process Mining*, nomeadamente *ProM*, *Disco* e o *framework PM4Py*. Além disso, e sendo a etapa de descoberta do processo um dos principais objetivos do *Process Mining*, foram testados e comparados todos os algoritmos disponíveis em cada ferramenta. Para tal, foram criados cenários de testes para analisar os desafios com que estes algoritmos conseguiram lidar.

3.2.1 Dados de teste

O tratamento dos dados para testar ferramentas e algoritmos consistiu em selecionar os dados de interesse, a partir da base de dados *MIMIC-III* (versão demo). Depois, a partir dessa recolha, os dados foram convertidos para o formato necessário para a aplicação dos algoritmos de descoberta de processos. Todo o código *Python* necessário para este processo pode ser seguido pelo Anexo 4.

Assim, na fase de tratamento de dados, procedeu-se à análise do esquema de tabelas da base de dados *MIMIC-III* (versão demo), de forma a encontrar a informação desejada para o *dataset* de testes. Selecionou-se um subconjunto de tabelas que satisfazem os requisitos propostos para o trabalho, Figura 3-1.

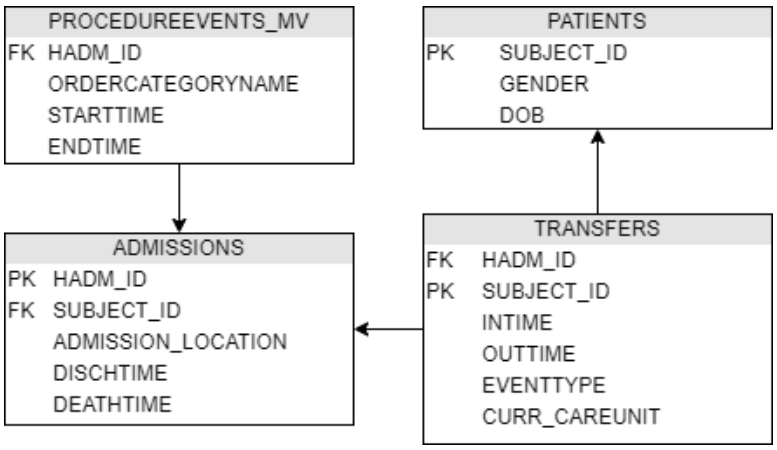


Figura 3-1: Esquema dos dados de teste.

Analisando o esquema, a tabela principal é *TRANSFERS*. Contém as localizações físicas dos pacientes durante o internamento. Os principais atributos desta tabela são: a unidade de cuidados (*CURR_CAREUNIT*) em que o paciente esteve, caso seja uma especialidade; a data e hora de entrada (*INTIME*) e saída (*OUTTIME*); o tipo de evento (*EVENTTYPE*), sendo *'admit'* para procedimentos feitos na fase de admissão / avaliação do paciente, *'transef'* para fases de transferência / estadia do paciente e *'discharge'* para etapas de alta do paciente.

Note-se que, de forma a completar todas as unidades de cuidados, onde não existia especialidade, foi inserida a sigla GCU, traduzido para *Unidade de Cuidados Gerais*. A descrição das restantes siglas das unidades de cuidados especializado são apresentadas no Quadro 3-1.

Em seguida, recorrendo ao atributo *SUBJECT_ID*, foi feita a ligação à tabela *PATIENTS*, de forma a aceder a informações dos pacientes em questão, nomeadamente aos atributos género (*GENDER*) e data de nascimento (*DOB*).

Posteriormente, foi utilizado o atributo *HADM_ID* para aceder à tabela *ADMISSIONS* que contém informação acerca da admissão do paciente. Esta tabela permitiu recolher informações acerca do tipo / local de admissão (*ADMISSION_LOCATION*) e data de alta (*DISCHTIME*) ou óbito (*DEATHTIME*). Permitiu, ainda, aceder à tabela *PROCEDUREEVENTS_MV* para conseguir dados relativos aos eventos executados em cada admissão.

Quadro 3-1: Designação das unidades de cuidados
(Adaptado de https://mimic.mit.edu/docs/iii/tables/transfers).

Unidade de cuidados	Designação
CCU	Unidade Coronariana
CSRU	Unidade de recuperação de cirurgia cardíaca
MICU	Unidade de terapia intensiva médica
NICU	Unidade de terapia intensiva neonatal
NWARD	Enfermaria neonatal
SICU	Unidade de terapia intensiva cirúrgica
TSICU	Trauma / Unidade de terapia intensiva cirúrgica

A partir da tabela *PROCEDUREEVENTS_MV* foi possível recolher o nome do processo (*ORDERCATEGORYNAME*) e a data e hora de início (*STARTTIME*) e fim (*ENDTIME*) do processo. De salientar que, nesta etapa, foi feita a sincronização dos processos com as localizações físicas, pelas respetivas datas de início / entrada e fim / saída.
Finalmente, a partir da importação de dados em folha de cálculo, foi feito o respetivo tratamento para chegar ao formato de *dataset* necessário para a aplicação dos algoritmos de descoberta de processos do *PM4Py*.
O formato necessário consiste em 3 tipos de informação:

- *Case ID* - um identificador exclusivo de cada processo;
- *Event* - uma etapa do processo, qualquer atividade que faça parte do processo que se está a analisar;
- *Timestamp* – data e hora de um determinado evento.

O *HADM_ID*, sendo um identificador único de admissão, foi utilizado como identificador do caso. Para cada etapa do processo foi feita a agregação do tipo de evento (admissão, transferência ou alta), a unidade de cuidados (uma sigla identificadora) e o nome do processo executado, caso exista.
Para o *timestamp* da etapa foi utilizada a data de início do processo ou, em casos em que não foi identificado um processo, é utilizada a data de entrada na ala da unidade de cuidados.
Quanto aos parâmetros extra de pesquisa, foi utilizado o tipo de admissão, o tipo de saída, através do qual se verifica se o paciente faleceu ou teve alta, a data de nascimento do paciente, o seu género e o dia da semana em que o evento ocorreu.
No final foram removidas possíveis duplicações provenientes da sincronização dos processos com as localizações físicas. De realçar que, para viabilizar a utilização deste *dataset* pelos algoritmos, estes foram convertidos em formato de *logs*, ordenados por *timestamp*.

3.2.2 Cenários de teste para os algoritmos

Pela análise e comparação feitas aos algoritmos, na fase de pesquisa, percebeu-se que existem situações onde podem surgir desafios, como é o caso de *loops* entre etapas e duplicações. Assim, foram selecionadas admissões que permitissem testar isoladamente todos estes cenários.

Num cenário inicial, foram escolhidas admissões simples, onde não se verificavam nenhum dos casos anteriormente descritos. Este cenário, Figura 3-2, tem como principal objetivo uma primeira interação com o algoritmo e respetivo modelo.

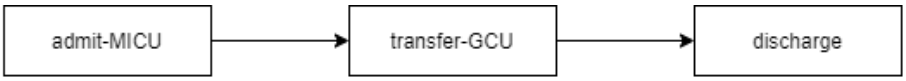


Figura 3-2: Cenário simples.

De seguida foi escolhido um cenário para testar o algoritmo em situações de etapas duplicadas, Figura 3-3. Como se pode verificar existe uma etapa que ocorre de forma repetida.

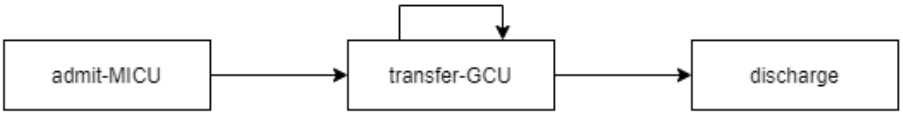


Figura 3-3: Cenário com etapas duplicadas.

No último cenário, foi testada a situação que expõe os algoritmos a *loops* entre etapas. Pela Figura 3-4 pode-se verificar uma ocorrência de *loop* entre 2 etapas.

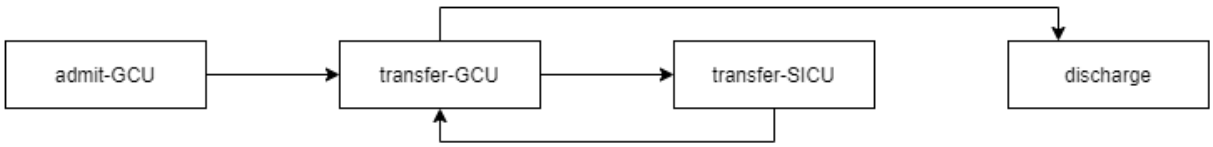


Figura 3-4: Cenário com *loops* entre etapas.

3.3 Dados dum serviço de urgências

Nesta secção, é apresentada a criação do *dataset* para a aplicação no caso de estudo final. Para tal, será apresentado e explicado todo o processo realizado desde a escolha do conjunto de dados a utilizar até ao *dataset* que será importado na ferramenta de *Process Mining*. De salientar que, pela experiência adquirida na fase de testes, a escolha dos dados e a estrutura do *dataset* final sofreram alterações para que a análise seja a mais eficaz possível.
Para cada etapa, será mostrado o bloco de código *Python* específico para realizar essa etapa, ainda assim, no Anexo 5 pode ser encontrado o código completo e comentado.
Para a criação do *dataset* começou-se por escolher quais as informações de interesse para a análise em questão. Sendo a análise focada nos percursos feitos pelos pacientes, a escolha da tabela *PATIENTS* foi uma opção fácil. A seguir está uma lista dos dados disponíveis na base de dados para um paciente (Johnson et al., 2016):

- *SUBJECT_ID*: identificador único que especifica um único paciente. Este campo é uma chave primária da tabela, portanto, é exclusiva para cada linha. As informações armazenadas nesta tabela são consistentes por toda a vida de um paciente;
- *GENDER*: sexo genótipo do paciente;

- *DOB*: data de nascimento de um determinado paciente;
- *DOD*: data da morte de um determinado paciente;
- *DOD_HOSP*: data da morte registada na base de dados do hospital;
- *DOD_SSN*: data da morte da base de dados da previdência social.

Tendo as informações dos pacientes, falta selecionar a informação relativa à passagem do paciente pelo hospital. As tabelas escolhidas foram *ADMISSIONS* e *TRANSFERS*, obtidas da base de dados do hospital. As tabelas *ICUSTAYS* e *SERVICES* também foram consideradas, mas entendeu-se que a tabela *TRANSFERS* continha a mesma informação, mas mais completa. A *ICUSTAYS* agrupa a tabela *TRANSFERS* com base nas unidades onde o paciente passou e exclui linhas onde nenhuma unidade foi identificada, mas sendo que a tabela *TRANSFERS* contém informações adicionais como o tipo de evento, isto é, se uma etapa pertence à fase de admissão, transferência ou alta. Decidiu-se utilizar esta tabela e fazer a remoção de linhas inválidas (sem unidade identificada) manualmente.
Já a tabela *SERVICES* descreve o serviço sob o qual um paciente foi admitido, mas embora um paciente possa estar fisicamente localizado num determinado tipo de unidade, ele não está necessariamente a ser atendido pela equipa que cuida dessa unidade. Isso pode acontecer por uma série de razões, incluindo falta de camas. Por esse motivo essa informação não se considerou relevante para a análise em questão.
A seguir apresenta-se uma lista de dados disponíveis na base de dados para uma única admissão hospitalar (Johnson et al., 2016):

- *SUBJECT_ID, HADM_ID*: cada linha desta tabela contém um único *HADM_ID*, que representa a admissão de um único paciente no hospital. É possível que esta tabela duplique um *SUBJECT_ID*, indicando que um único paciente teve múltiplas admissões no hospital. A tabela *ADMISSIONS* pode ser vinculada à tabela *PATIENTS* pelo *SUBJECT_ID*;
- *ADMITTIME*: hora da admissão do paciente no hospital;
- *DISCHTIME*: hora da alta hospitalar do paciente;
- *DEATHTIME*: data da morte do paciente, se ele morreu dentro do hospital;
- *ADMISSION_TYPE*: tipo de admissão: *'ELECTIVE'*, *'EMERGENCY'*, *'NEWBORN'* ou *'URGENT'*. *Emergency / Urgent* indicam cuidados médicos não planeados e, nos estudos, costumam ser agrupados numa única categoria. *Elective* indica uma internação hospitalar previamente planeada. *Newborn* indica que pertence ao nascimento do paciente;
- *ADMISSION_LOCATION*: fornece informações sobre a localização anterior do paciente antes de chegar ao hospital. Existem 9 valores possíveis. De salientar que o texto truncado ocorre nos dados brutos:
    - *EMERGENCY ROOM ADMIT*
    - *TRANSFER FROM HOSP/EXTRAM*
    - *TRANSFER FROM OTHER HEALT*
    - *CLINIC REFERRAL/PREMATURE*
    - *** INFO NOT AVAILABLE ***
    - *TRANSFER FROM SKILLED NUR*

  - *TRSF WITHIN THIS FACILITY*
  - *HMO REFERRAL/SICK*
  - *PHYS REFERRAL/NORMAL DELI*
- *DISCHARGE_LOCATION*: localização do paciente após a alta hospitalar;
- *INSURANCE*: tipo de seguro médico do paciente;
- *LANGUAGE*: idioma principal do paciente;
- *RELIGION*: religião declarada do paciente;
- *MARITAL_STATUS*: estado civil do paciente;
- *ETHNICITY*: etnia declarada do paciente;
- *DIAGNOSIS*: breve descrição do motivo da admissão do paciente.

Assim, chegou-se ao esquema de tabelas da Figura 3-5 que satisfazem os requisitos necessários.

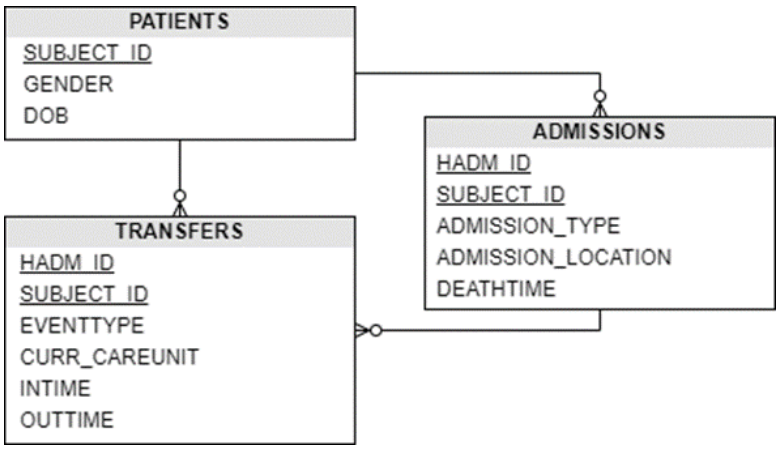


Figura 3-5: Modelo de dados utilizado.

Analisando o esquema, a tabela principal é *TRANSFERS*. Contém as localizações físicas dos pacientes durante o internamento. Os principais atributos desta tabela são: a unidade de cuidados (*CURR_CAREUNIT*) em que o paciente esteve, caso seja uma especialidade; a data e hora de entrada (*INTIME*) e saída (*OUTTIME*) da unidade; o tipo de evento (*EVENTTYPE*), admissão, transferência ou alta.
A descrição das siglas das unidades de cuidados especializado são apresentadas no Quadro 3-2.

Quadro 3-2: Descrição das unidades de cuidados
(Adaptado de https://mimic.mit.edu/docs/iii/tables/transfers).

Unidade de cuidados	Descrição
CCU (Unidade Coronariana)	É especializada no tratamento de pacientes adultos com problemas cardíacos que correm alto risco e necessitam de cuidados intensivos com monitoramento 24h.
CSRU (Unidade de recuperação de cirurgia cardíaca)	Consiste numa unidade de repouso, preferencialmente numa Unidade de Terapia Intensiva (UTI) nas primeiras 48 horas após o procedimento, onde existem todos os equipamentos que podem ser utilizados para monitorar o paciente nessa fase inicial, na qual existem maiores chances de ocorrer uma situação de emergência, podendo levar à morte.
MICU (Unidade de terapia intensiva médica)	Representa uma estrutura hospitalar que se caracteriza como "unidade complexa dotada de sistema de monitorização contínua que admite pacientes potencialmente graves ou com descompensação de um ou mais sistemas orgânicos e que com o suporte e tratamento intensivos tenham possibilidade de se recuperar".
NICU (Unidade de terapia intensiva neonatal)	Consiste num ambiente do hospital preparado para receber recém-nascidos que nasceram antes das 37 semanas de gestação, com baixo peso ou que possuem algum problema que possa interferir no seu desenvolvimento, como alterações cardíacas ou respiratórias.
NWARD (Enfermaria neonatal)	Envolve todo o universo dos recém-nascidos, o que requer cuidados especiais, sendo que eles são muito frágeis e estão num estágio muito vulnerável da vida.
SICU (Unidade de terapia intensiva cirúrgica)	Designa uma unidade de repouso para pacientes provenientes de cirurgias gerais, pacientes cirúrgicos de média e alta complexidade, onde passam os primeiros dias pós-cirurgia, os quais exigem maior cuidado e monitorização.
TSICU (Trauma / Unidade de terapia intensiva cirúrgica)	Destinada a pacientes clínicos e cirúrgicos em pós-operatório de cirurgia de urgência ou tardio, que necessitam de cuidados intensivos e pós-traumáticos. Os cuidados na manipulação de pacientes pós-trauma envolvem, além de uma detalhada avaliação, a prevenção do agravamento das lesões iniciais.

Em seguida, recorrendo ao atributo *SUBJECT_ID*, foi feita a ligação à tabela *PATIENTS*, de forma a aceder a informações dos pacientes em questão, nomeadamente aos atributos género (*GENDER*) e data de nascimento (*DOB*).
Posteriormente, foi utilizado o atributo *HADM_ID* e *SUBJECT_ID* para aceder à tabela *ADMISSIONS* que contém informação acerca da admissão do paciente. Esta tabela permitiu recolher informações acerca do tipo (*ADMISSION_TYPE*) e local de admissão (*ADMISSION_LOCATION*) e data de óbito (*DEATHTIME*).
A etapa de pré-processamento foi implementada em *Python*. Começou-se por importar os dados, a partir de ficheiros em formato *CSV*, utilizando a biblioteca *pandas*. Como se depreende pela Figura 3-6, os dados foram extraídos filtrando apenas as colunas que serão utilizadas. De salientar que, no caso da tabela *TRANSFERS*, foram logo removidas linhas inválidas, onde não existia o tipo de evento.

```
import pandas as pd

patients = pd.read_csv(tables + 'PATIENTS.csv')[['SUBJECT_ID', 'GENDER', 'DOB']]
admissions = pd.read_csv(tables + 'ADMISSIONS.csv')[['HADM_ID', 'SUBJECT_ID', 'ADMISSION_TYPE', 'ADMISSION_LOCATION', 'DEATHTIME']]
transfers = pd.read_csv(tables + 'TRANSFERS.csv')[['SUBJECT_ID', 'HADM_ID', 'EVENTTYPE',
                                                   'CURR_CAREUNIT', 'INTIME', 'OUTTIME']].dropna(subset=['EVENTTYPE'])
```

Figura 3-6: Importação dos dados no *Python*.

De seguida, Figura 3-7, as tabelas foram combinadas. Assim a tabela *TRANSFERS* foi intercalada com a tabela *PATIENTS*, a partir do *SUBJECT_ID*. O resultado dessa operação foi intercalado com a tabela *ADMISSIONS*, pelo *HADM_ID*.

```
# Join Transfers and Patient
data = pd.merge(left=transfers, right=patients, how='inner', left_on='SUBJECT_ID', right_on='SUBJECT_ID')
# Join Admissions
data = pd.merge(left=data, right=admissions, how='inner', left_on=['SUBJECT_ID', 'HADM_ID'], right_on=['SUBJECT_ID', 'HADM_ID'])
```

Figura 3-7: Junção de dados no *Python*.

Finalmente foi possível criar um primeiro *dataset*, onde foi feito algum processamento aos dados. Como se pode ver pela Figura 3-8, para a admissão foram definidos um identificador, o tipo e local de admissão e o tipo de alta. Para este último foi verificado se a admissão do paciente tinha data da morte, se sim, foi inserida a identificação *'death'*, senão adicionou-se para *'discharge'*.

Quanto ao paciente, foi definido o seu género e a data de nascimento, tendo-se removido a hora, pois não é relevante neste tipo de data. Para cada evento, foi guardado o tipo de evento e calculado o dia da semana em que o evento ocorreu, pela data de entrada na unidade.

Foi, ainda, guardada a unidade onde o paciente estava, com a data de entrada e saída. Note-se que, de forma a completar todas as unidades de cuidados, onde não existe especialidade, será inserida a sigla GCU, traduzido para Unidade de Cuidados Gerais, exceto se o tipo de evento seja a alta, onde não faz sentido relacionar a uma unidade.

```
dataset = pd.DataFrame(data =
    {
        'admission:id': data['HADM_ID'],
        'admission:type': data['ADMISSION_TYPE'],
        'admission:location': data['ADMISSION_LOCATION'],
        'admission:discharge_type': ['death' if str(deathtime) != 'nan' else 'discharge' for deathtime in data['DEATHTIME']],
        'patient:gender': data['GENDER'],
        'patient:date_of_birth': [date.split(' ')[0] for date in data['DOB']],
        'event:date_of_week': [datetime.datetime.strptime(str(inttime), '%Y-%m-%d %H:%M:%S').weekday() for inttime in data['INTIME']],

        'event:type': data['EVENTTYPE'],
        'event:care_unit_in': data['INTIME'],
        'event:care_unit_out': data['OUTTIME'],
        'event:care_unit': ['GCU' if str(careunit) == 'nan' and str(eventtype) != 'discharge' else careunit
                            for careunit, eventtype in zip(data['CURR_CAREUNIT'], data['EVENTTYPE'])]
    }
)
```

Figura 3-8: Modelo do *dataset* no *Python*.

4. Análise e Comparação de ferramentas e algoritmos em *Process Mining*

Nesta secção será descrito o estudo exploratório e comparativo efetuado, (Gomes et al., 2021b) e (Gomes et al., 2021c), às ferramentas e respetivos algoritmos de *Process Mining* disponíveis para este trabalho. Serão utilizadas as ferramentas *ProM*, *Disco* e o *framework PM4Py*, de forma a perceber as suas vantagens e desvantagens no tipo de análise pretendida.
Será testada e comparada a forma como as ferramentas fazem a importação do conjunto de *logs*, a descoberta de processos, a análise de variantes, a filtragem de *logs*, a verificação de conformidade e apresentação de estatísticas sobre os *logs*.
De salientar que, na etapa de descoberta do processo, foram testados e comparados todos os algoritmos disponíveis em cada ferramenta, de forma a encontrar o mais apto. No Anexo 6 encontra-se todo o código *Python* desenvolvido para testar o *framework PM4Py*.

4.1 Importação do conjunto de *logs*

A importação dos *logs* é o primeiro passo de qualquer cenário de implementação, pois o conjunto de *logs* é carregado para memória e, em seguida, tratado de forma a poder ser utilizado nos passos seguintes.
No *PM4Py*, esta fase foi bastante simples, pois o conjunto de *logs* já foi gerado com a estrutura padrão que esta ferramenta aceita. Assim, o conjunto de *logs* foi carregado a partir de um ficheiro *CSV*. Apenas foi necessário converter o carimbo de data/hora para *datetime*, porque no processo de importação, este campo ficou em formato de texto, bem como converter o objeto no tipo de *logs*, Figura 4-1.

```
import pandas as pd
from pm4py.objects.conversion.log import converter as log_converter

dataset = pd.read_csv(path + 'LOGS.csv')
dataset['time:timestamp'] = pd.to_datetime(dataset['time:timestamp'])
log = log_converter.apply(dataset)
```

Figura 4-1: Código *Python* para importar os *logs*.

Já para o *ProM*, o conjunto de *logs* foi importado pelo módulo *"CSV File (XES Conversion with Log package)"*, Figura 4-2.

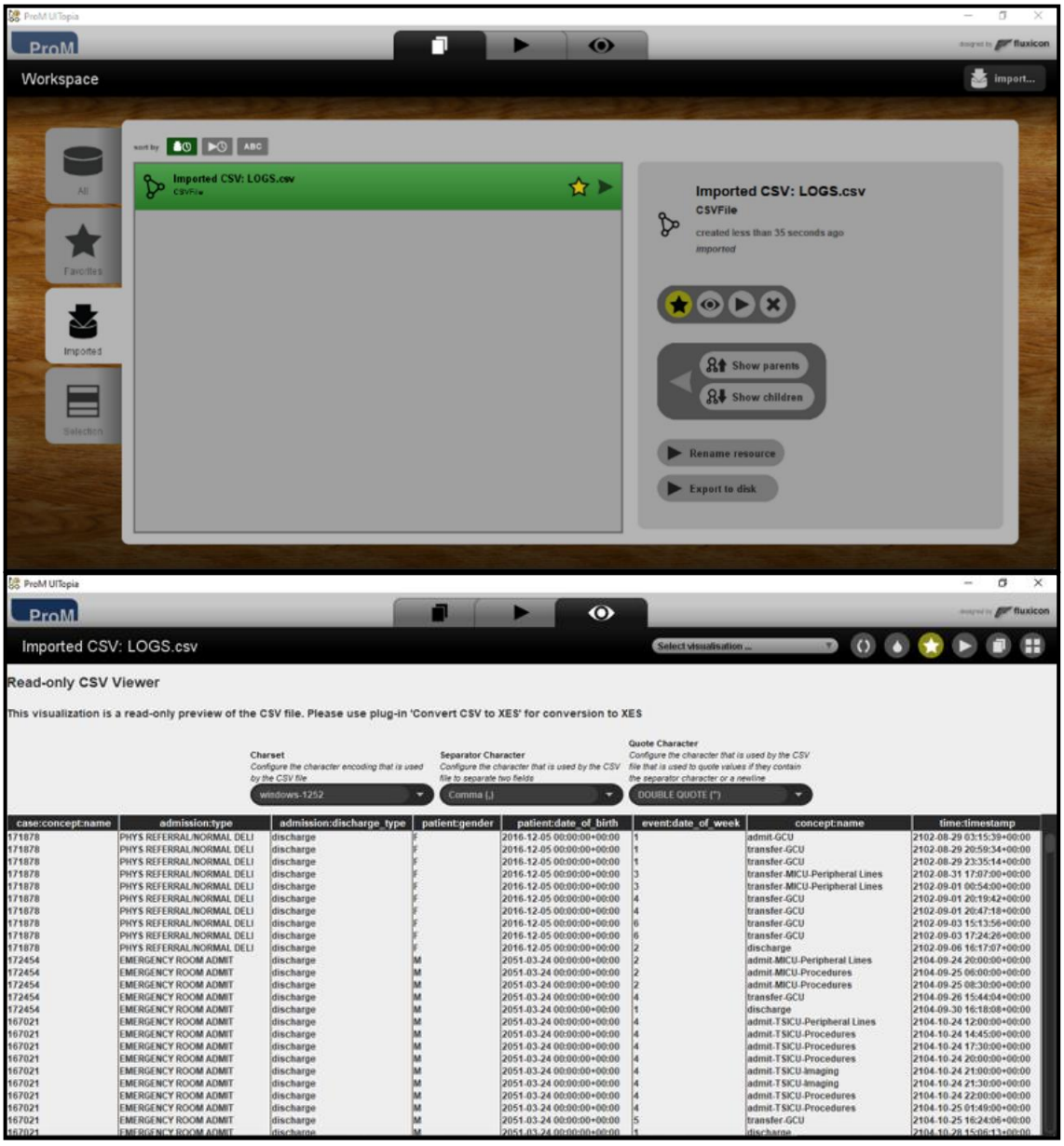


Figura 4-2: Etapas do *ProM* para importar *CSV*.

Porém, para utilizar os *logs* em módulos como os de descoberta de processos é necessário que estes estejam no formato *XES* (*eXtensible Event Stream*). Dessa forma, foi utilizado o módulo *"Convert CSV to XES"*, Figura 4-3. Foi necessário definir as colunas do *dataset* correspondentes ao identificador do caso, identificadores do evento, e o *timestamp* do evento,

neste caso o de início do evento. Esta operação acaba com o *dashboard* onde já é possível analisar algumas características e informações acerca do conjunto de *logs*.

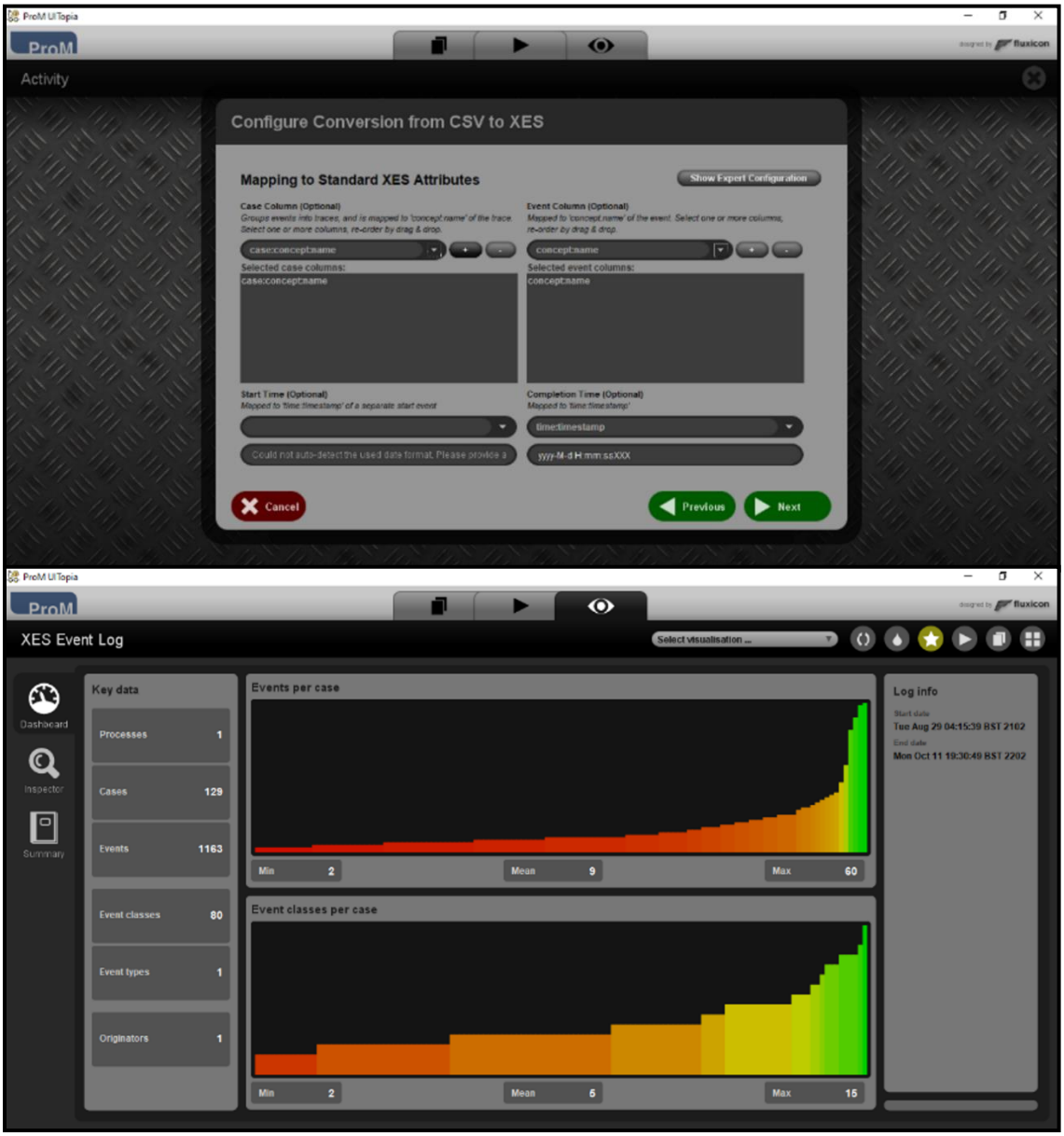


Figura 4-3: Etapas do *ProM* para converter *logs* para o formato *XES*.

Finalmente, o *Disco* torna a importação de dados simples. O *CSV* foi carregado e apenas foi preciso configurar quais colunas contêm o *ID* de caso, o carimbo de data/hora, o nome da etapa e outros atributos extra a incluir. De salientar que, como se verifica na Figura 4-4, foi necessário definir o formato da coluna de *timestamp* para que o *Disco* o pudesse interpretar.

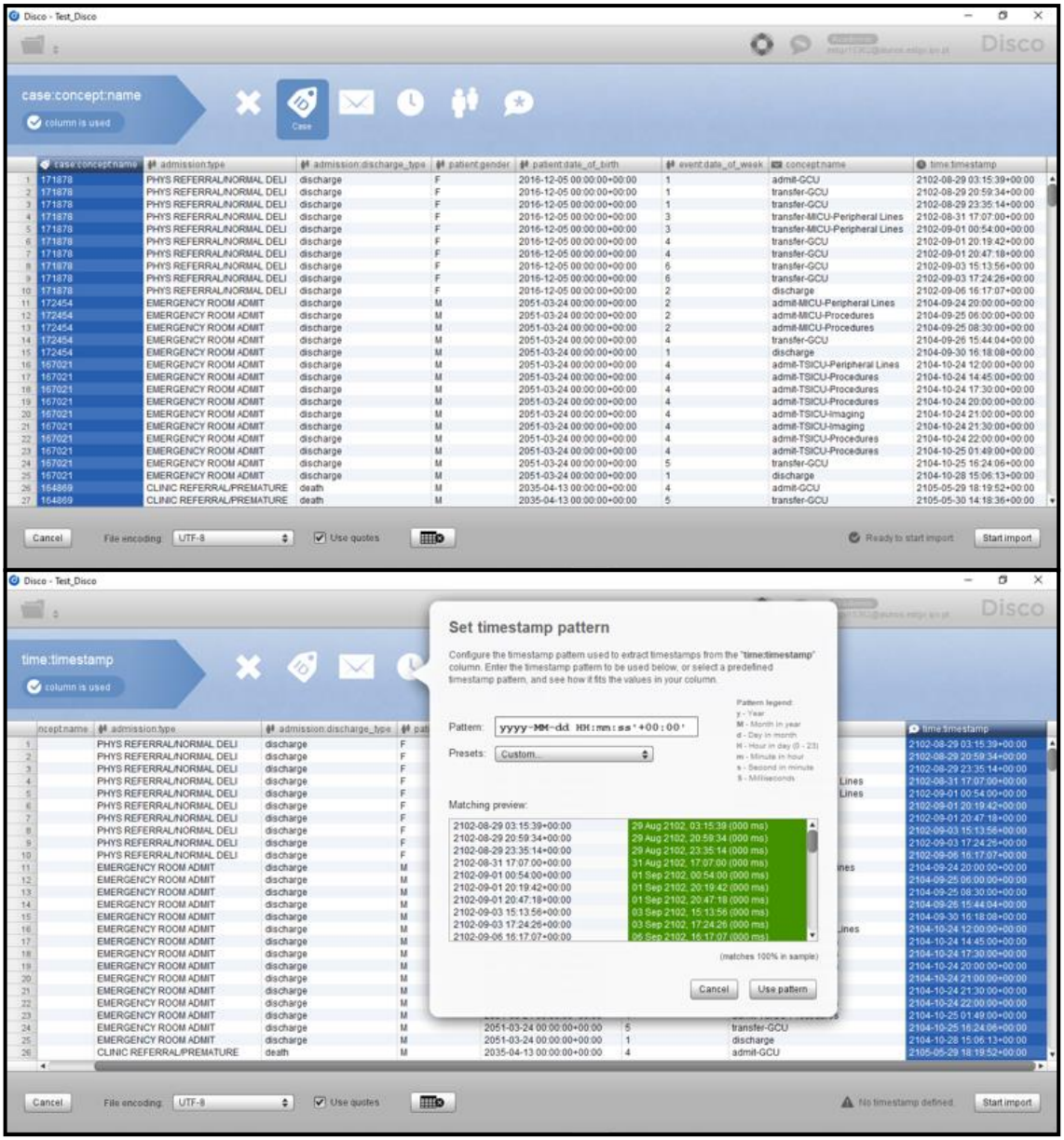


Figura 4-4: Etapas do *Disco* para importar os *logs*.

4.2 Descoberta de processos

Nesta secção vão ser apresentados os resultados obtidos no âmbito da descoberta de processos, (Gomes et al., 2021b) e (Gomes et al., 2021d). Assim, será apresentada e analisada a forma como as ferramentas fazem a descoberta de processos. Para cada ferramenta serão apresentados os resultados dos algoritmos disponíveis aos cenários anteriormente apresentados na secção 3.2.2, concluindo qual deles se apresentou mais apto.

4.2.1 *PM4Py*

No *PM4Py*, foram testados os algoritmos *Alpha Miner*, *Directly-Follows Graph*, *Heuristic Miner* e *Inductive Miner*. No Quadro 4-1 podem-se ver os modelos resultantes dos cenários de teste.

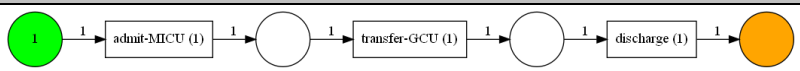
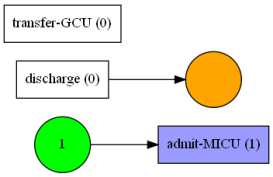
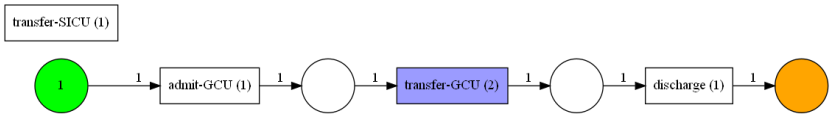
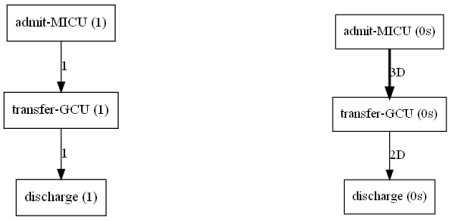
O *Alpha Miner* mostrou-se limitado para etapas duplicadas e *loops* entre duas etapas, pois estas ficam isoladas do modelo. Este resultado era previsível, pois este algoritmo, reconhecidamente, não suporta etapas duplicadas, nem lida com *loops* de comprimento um ou dois (Breitmayer, 2018).

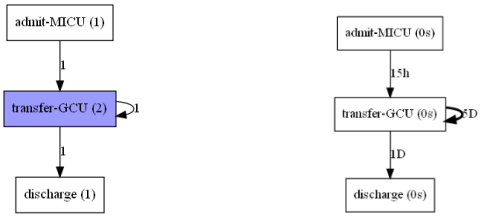
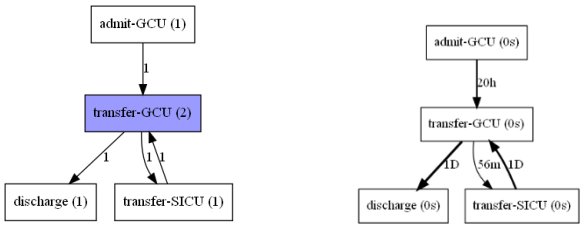
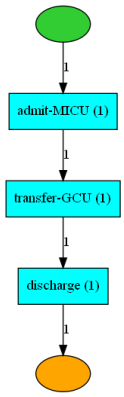
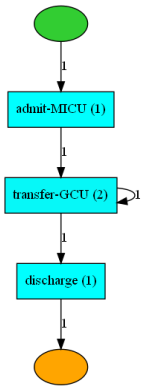
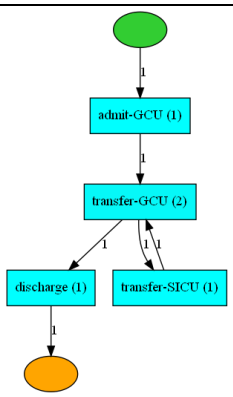
O *Directly-Follows Graph*, mostrou capacidade para lidar bem com os desafios de duplicações e *loops*, mas para todo o conjunto de *logs* o modelo gerado, tendo em conta a frequência, foi inválido. Uma característica do algoritmo que pode justificar este resultado é o facto dos *DFG* serem simplificados, removendo nós e ligações com base em limites de frequência (Van der Aalst, 2019).

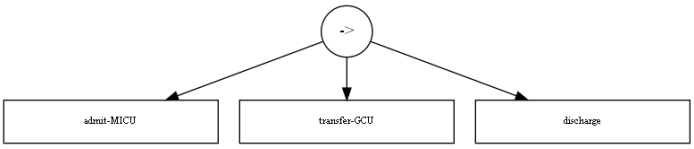
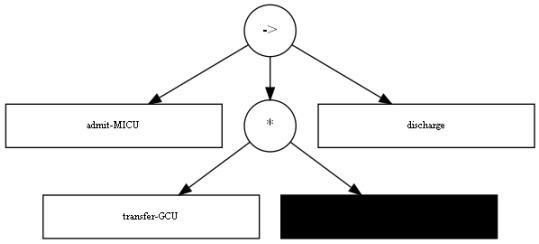
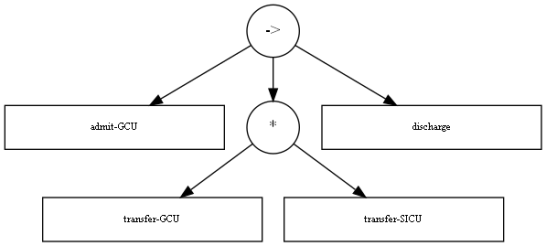
O *Heuristic Miner* mostrou que é compatível com etapas duplicadas e desafios de *loops*. Para um número maior de *logs*, o modelo resultante é difícil de analisar, pois criou modelos de *spaghetti*.

Para todos os *logs*, o *Inductive Miner*, apesar da grande quantidade de dados de *logs*, parece gerar um modelo mais pequeno, com menos etapas e ligações. Uma explicação deste resultado pode ser o melhoramento que este algoritmo tem na procura de divisões / padrões nos *logs* e ainda no excessivo uso de transições ocultas para ultrapassar *loops* em partes do modelo (Pohl, 2019).

Quadro 4-1: Modelos dos algoritmos disponíveis no *framework PM4Py*.

Algoritmo	Cenário	Resultado	Modelo
Alpha Miner	Simples	Valido	
	Com etapas duplicadas	Invalido	
	Com *loops* entre etapas	Invalido	
Directly-Follows Graph	Simples	Valido	

Algoritmo	Cenário	Resultado	Modelo
Directly-Follows Graph	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Valido	
	Para todos os *logs*	Invalido	[*Spaghetti*]
Heuristic Miner	Simples	Valido	
	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Valido	

Algoritmo	Cenário	Resultado	Modelo
Heuristic Miner	Para todos os *logs*	Valido	[*Spaghetti*]
Inductive Miner	Simples	Valido	
	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Valido	
	Para todos os *logs*	Valido	[*Spaghetti*]

O Quadro 4-2 mostra a média dos tempos de execução, em segundos, dos algoritmos *Heuristic Miner* e *Inductive Miner* para todo o conjunto de *logs*, sendo que estes algoritmos conseguiram apresentar um modelo válido. De salientar que cada algoritmo e tipo de modelo foram testados 5 vezes, nas mesmas condições, calculando-se a média, retirando o tempo máximo e mínimo. Estes tempos correspondem à execução do próprio algoritmo, pois o conjunto de *logs* estava já carregado em memória.

Quadro 4-2: Tempos de execução para todo o conjunto de *logs* no *framework PM4Py*.

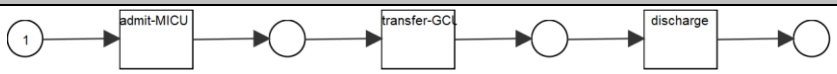
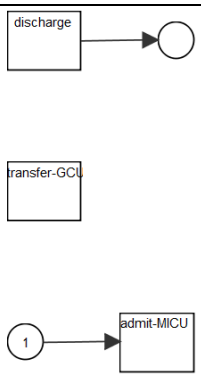
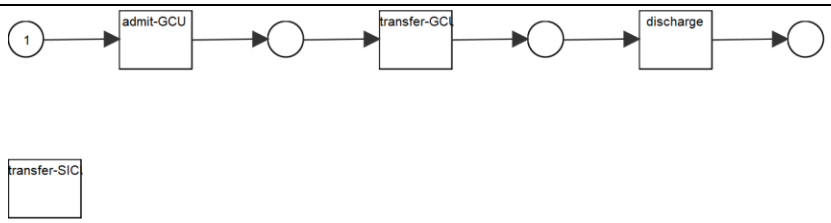
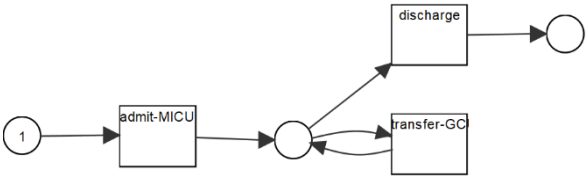
Heuristic Miner	***Heuristics Net***	***Petri Net***	***Inductive Miner***	***Process Tree***	***Petri Net***
	4,655	83,254		14,967	34,700

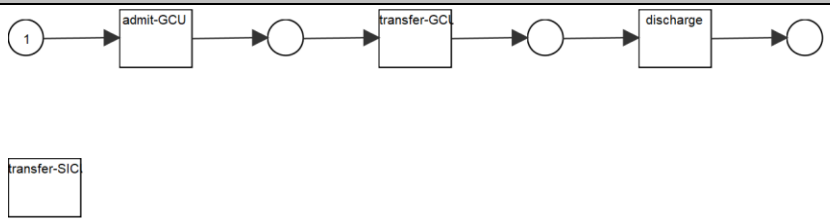
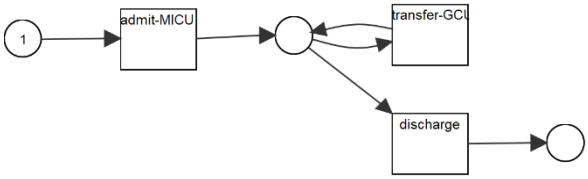
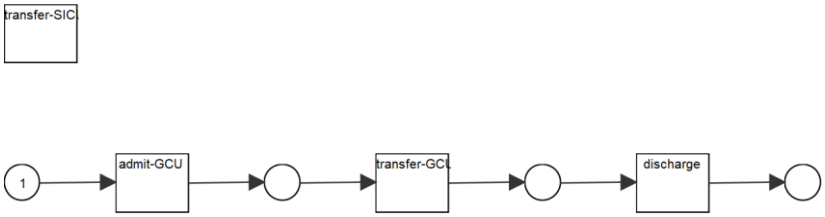
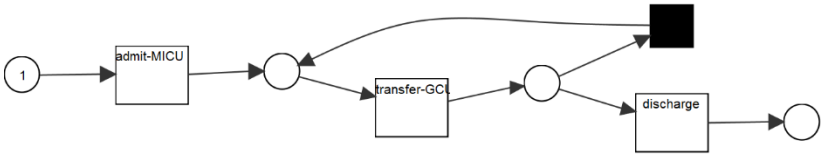
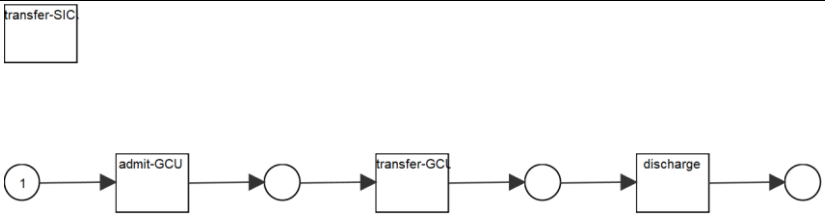
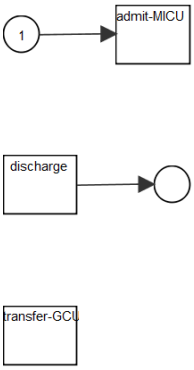
4.2.2 *ProM*

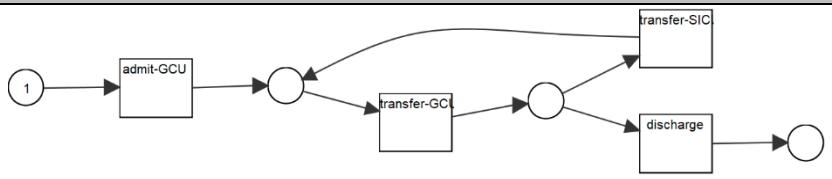
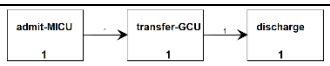
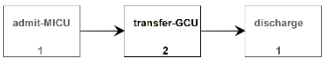
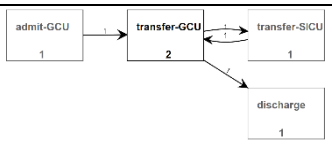
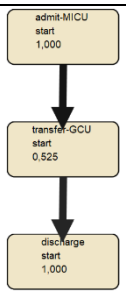
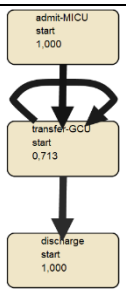
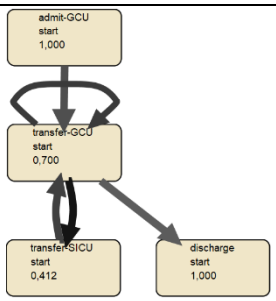
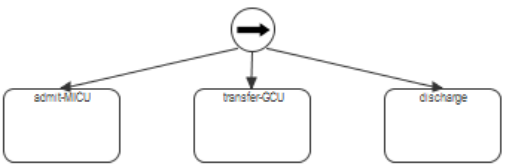
Os modelos resultantes do *software ProM* podem ser observados no Quadro 4-3. O algoritmo *Alpha Miner*, para *logs* com os conhecidos desafios de duplicações e *loops*, gerou um modelo inválido, confirmando as limitações, para estes cenários, reconhecidas na literatura.
O módulo do *ProM* utilizado para aplicar o algoritmo *Alpha Miner*, denominado pelo mesmo nome, permite utilizar outras versões deste algoritmo, como é o caso do *Alpha Miner +*, *Alpha Miner ++* e *Alpha Miner #*. Todas estas versões mostraram limitações em *logs* com *loops*, como a sua versão inicial. Uma outra versão disponível é a *Alpha Miner R* que foi a primeira a apresentar um modelo válido para cenários com *loops*, mas mostrou ter limitações em etapas duplicadas.

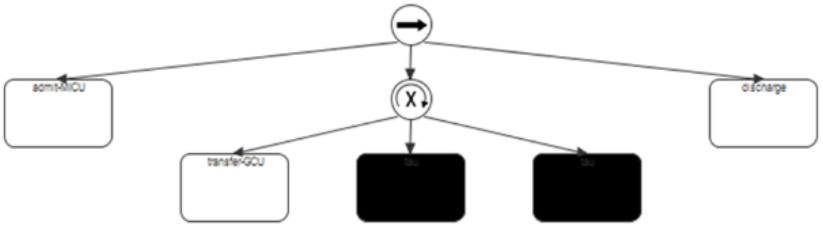
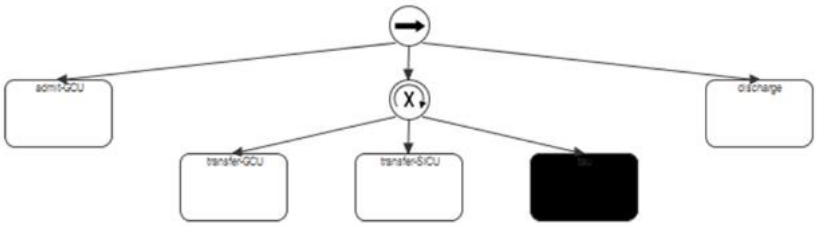
O módulo *"Mine for a Heuristics Net using Heuristics Miner"* do *ProM* permitiu a aplicação do *Heuristic Miner* no conjunto de *logs*. Este algoritmo mostrou-se compatível para todos os cenários testados, sendo que neste, as duplicações não são representadas por uma ligação.
Já o módulo *"Mine for a Fuzzy Model"* do *ProM* permitiu a implementação do algoritmo *Fuzzy Miner* para os cenários testados. Este algoritmo, nos cenários com *loops* entre 2 etapas, apresentou uma duplicação numa etapa que o *log* não tinha, mostrando assim uma inconsistência.
Um outro algoritmo permitido pelo *ProM* é o *Inductive Miner*, a partir do módulo *"Mine with Inductive visual Miner"*. Sendo um algoritmo melhorado, reconhecido na literatura, este exibiu modelos de *Process Tree* válidos para todos os cenários testados. Devido a estes resultados, este algoritmo foi ainda exposto a um conjunto maior de variantes, para perceber como lida com um volume maior de *logs*, onde apresentou modelos que aparentam ser válidos, mas, naturalmente, modelos de *spaghetti*. Pelos resultados obtidos, este foi o algoritmo utilizado para os restantes testes a esta ferramenta.

Quadro 4-3: Modelos resultantes dos algoritmos disponíveis no *ProM*.

Algoritmo	Cenário	Resultado	Modelo
Alpha Miner	Simples	Valido	
	Com etapas duplicadas	Invalido	
	Com *loops* entre etapas	Invalido	
Alpha Miner +	Simples	Valido	
	Com etapas duplicadas	Valido	

Algoritmo	Cenário	Resultado	Modelo
Alpha Miner +	Com *loops* entre etapas	Invalido	
Alpha Miner ++	Simples	Valido	
	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Invalido	
Alpha Miner #	Simples	Valido	
	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Valido	
Alpha Miner R	Simples	Valido	
	Com etapas duplicadas	Invalido	

Algoritmo	Cenário	Resultado	Modelo
Alpha Miner R	Com *loops* entre etapas	Valido	
Heuristic Miner	Simples	Valido	
	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Valido	
Fuzzy Miner	Simples	Valido	
	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Valido	
Inductive Miner	Simples	Valido	

Algoritmo	Cenário	Resultado	Modelo
Inductive Miner	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Valido	
	Para todos os *logs*	Valido	[*Spaghetti*]

4.2.3 *Disco*

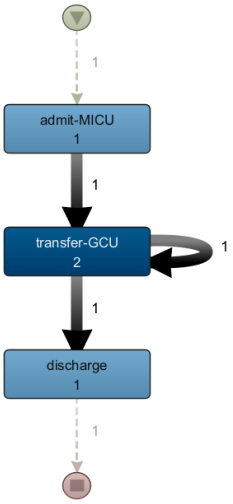
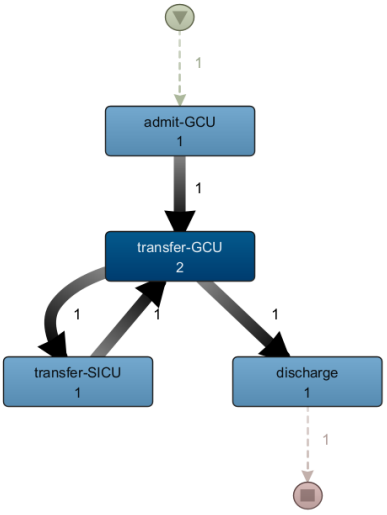
No *software Disco*, a funcionalidade principal da mineração de processos é a descoberta automatizada de mapas de processos. Assim que o processo de importação dos *logs* foi finalizado, aparece automaticamente a visualização do mapa, onde se pode visualizar de forma rápida e objetiva como o processo foi realmente executado.

Para tal, o *Disco* usa uma visualização de mapa de processo compreensível e aparentemente confiável. A espessura dos caminhos e as cores das atividades mostram os principais caminhos dos fluxos do processo.

O *Disco* é baseado no *Fuzzy Miner* de *Christian W. Günther*, já que este foi o primeiro algoritmo de mineração a introduzir a "metáfora do mapa" para mineração de processos, incluindo recursos avançados como simplificação contínua de processos e destaque de atividades e caminhos frequentes. Para o *Disco*, foi usada a abordagem cientificamente comprovada do *Fuzzy Miner*, combinada com uma vasta experiência de testes de utilizadores (Günther, 2012). O resultado é um algoritmo de mineração que, para além de fornecer resultados confiáveis para conjuntos de dados de complexidade alta, pode ser personalizado e compreendido de forma eficiente por especialistas da área do processo em análise, mas sem experiência anterior em mineração de processos (Günther, 2012).

Como se percebe pelo Quadro 4-4, os resultados dos modelos gerados no *Disco* mostram que o algoritmo usado suporta todos estes cenários. Perante estes resultados, o algoritmo e respetivo modelo foram testados para todo o conjunto de *logs*. O modelo de *spaghetti* era previsível, mas vai permitir mostrar as funcionalidades de personalização que esta ferramenta disponibiliza.

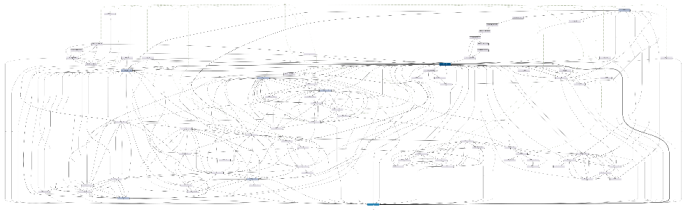
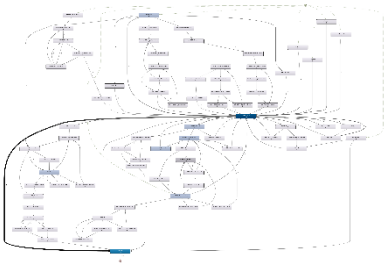
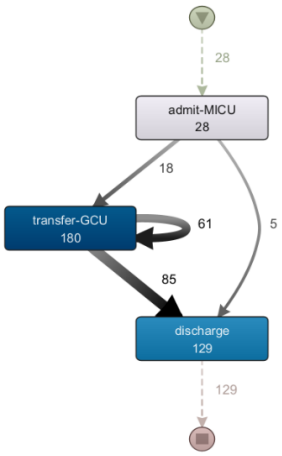
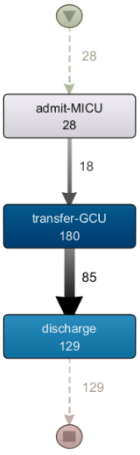
Quadro 4-4: Modelos resultantes dos algoritmos disponíveis no *Disco*.

Algoritmo	Cenário	Resultado	Modelo
***Fuzzy Miner* (*Christian W. Günther*)**	Simples	Valido	
	Com etapas duplicadas	Valido	
	Com *loops* entre etapas	Valido	
	Para todos os *logs*	Valido	[*Spaghetti*]

Quanto ao nível de detalhe, é possível ajustar as etapas, de acordo com a sua frequência e ligações, e de acordo com os fluxos de processo. Pelo Quadro 4-5 pode-se ver a diferença entre

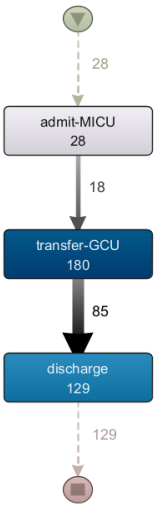
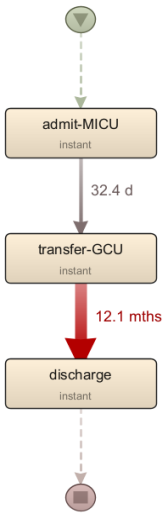
modelos com diferentes configurações, onde se verifica que a configuração com todas as etapas e ligações corresponde a um cenário de *spaghetti* e a configuração com as etapas e ligações mais frequentes corresponde ao fluxo com o conjunto de etapas e ligações que mais vezes ocorreu.

Quadro 4-5: Comparativo entre diferentes nível de detalhe no *Disco*.

100% das etapas **100% das ligações**	**100% das etapas** **Ligações do fluxo mais frequente**
	
Etapas do fluxo mais frequente **100% das ligações**	**Etapas do fluxo mais frequente** **Ligações do fluxo mais frequente**
	

Uma outra configuração disponível são as métricas de desempenho: análise por frequência e análise por desempenho. No Quadro 4-6 pode-se ver o resultado de cada uma delas para o mesmo modelo onde, para a frequência, se pode ver o número de vezes que cada etapa e ligação executa e, para o desempenho, o tempo decorrido entre etapas.

Quadro 4-6: Comparativo entre diferentes tipos de análise no *Disco*.

Análise por frequência	Análise por desempenho
	

4.2.4 Comparação dos algoritmos

A partir dos resultados dos testes feitos aos algoritmos disponíveis nas ferramentas *ProM*, *Disco* e no *framework PM4Py*, nesta secção, serão sintetizados esses resultados, de forma a ter uma classificação sistemática e comparativa dos algoritmos de *Process Mining* testados.

Pode-se concluir que nenhuma das versões do *Alpha Miner* se mostrou apta para lidar com etapas duplicadas e *loops* entre duas etapas. Trata-se de algo que o *Directly-Follows Graph* conseguiu, mas que por sua vez, para um conjunto de *logs* maior, o modelo gerado foi inválido, não conseguindo representar casos com mais de 5 etapas.

Quando aos restantes algoritmos, mostraram-se aptos para os desafios, com uma enorme desvantagem para o *Fuzzy Miner* implementado no *ProM*, que apresentou um modelo válido para *loops* entre 2 etapas, mas incoerente em relação ao *log* utilizado. Assim, o *Heuristic Miner*, o *Inductive Miner* e a versão *Fuzzy Miner* de *Christian W. Günther* foram os que se mostraram realmente aptos a lidar com desafios e maiores volumes de *logs*.

A nível de simplicidade do modelo, o *Heuristic Miner* apresentou um modelo de *spaghetti* mais complicado que o *Inductive Miner*, pois este faz uma simplificação do modelo pela pesquisa recursiva de padrões entre *logs*. Quando ao *Fuzzy Miner* de *Christian W. Günther* implementado no *Disco*, permite uma configuração bastante simples e intuitiva onde é possível apresentar o modelo com um número menor de etapas e ligações para que a análise seja mais eficaz.

Por fim, analisando os tipos de modelos gerados, o *Inductive Miner*, por padrão, apresenta modelos em árvore que tem um tipo de leitura muito pouco intuitiva. De salientar que o resultado deste algoritmo pode ser convertido para modelos de rede de *Petri* onde o resultado pode ser já mais intuitivo. No entanto, o modelo gerado pelo *Fuzzy Miner* de *Christian W. Günther*, implementado no *Disco*, tem a leitura mais intuitiva, apresentando destaque de atividades e caminhos frequentes, bem como análises por frequência e/ou desempenho.

O Quadro 4-7 mostra o resumo dos resultados conseguidos, onde os parâmetros de comparação são apresentados por ordem de prioridade, e se um algoritmo não mostra pontos positivos num parâmetro, já não é analisado nos próximos. Indica-se, assim, o algoritmo mais apto e intuitivo e que permite uma boa análise do processo.

Quadro 4-7: Resultados da comparação dos algoritmos.

Algoritmo	Limitações a desafios	Simplicidade do modelo	Tipo de modelo
Alpha Miner	↓	-	-
Directly-Follows Graph	↓	-	-
Fuzzy Miner	↓	-	-
Heuristic Miner	↑	↓	-
Inductive Miner	↑	↑	↓
Fuzzy Miner* de *Christian W. Günther	↑	↑	↑

4.3 Análise de variantes

A análise de variantes é de extrema importância, pois tem em consideração o número de ocorrências, a fim de remover do modelo as variantes menos relevantes, que ocorreram pontualmente. Uma variante é um conjunto de casos que compartilham a mesma perspetiva de fluxo de controle, portanto, um conjunto de casos que compartilham os mesmos eventos / atividades, na mesma ordem (Bolt et al., 2017).

No *PM4Py*, começou-se por identificar as variantes existentes e a respetiva frequência de ocorrência. Na Figura 4-5 pode-se analisar o código e as bibliotecas utilizadas, assim como o resultado das principais variantes.

```
import pandas as pd
from pm4py.statistics.traces.log import case_statistics
variants_count = case_statistics.get_variant_statistics(log)
variants_count = sorted(variants_count, key=lambda x: x['count'], reverse=True)
variants_view = pd.DataFrame(variants_count)
n_log = variants_view['count'].sum()
variants_view['percentage'] = variants_view.apply(lambda row: round((row['count'] / n_log) * 100, 2), axis=1)
```

Index	variant	count	percentage
0	admit-MICU,transfer-GCU,discharge	10	7.75
1	admit-MICU,discharge	5	3.88
2	admit-TSICU,discharge	4	3.1
3	admit-MICU,transfer-GCU,transfer-GCU,discharge	3	2.33
4	admit-MICU-Peripheral Lines,transfer-GCU,discharge	2	1.55

Figura 4-5: Código *Python* para descoberta das variantes do processo.

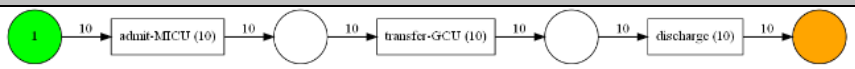
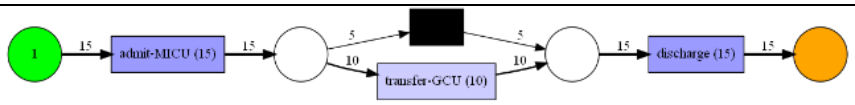
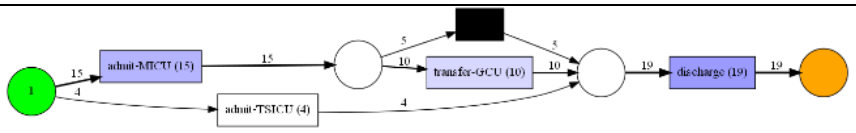
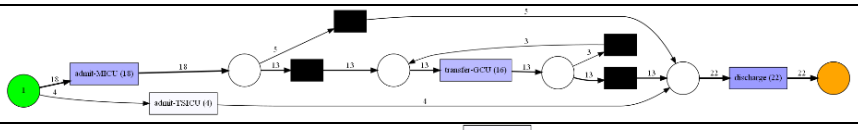
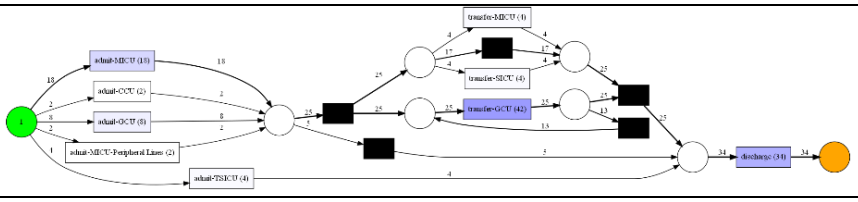
A Figura 4-6 apresenta o código necessário para a aplicação da filtragem por variantes, onde foi definida a percentagem a considerar. Quanto mais perto de 0 for definida a percentagem, mais recorrentes são as variantes consideradas.

```
from pm4py.algo.filtering.log.variants import variants_filter
log = variants_filter.filter_log_variants_percentage(log, percentage=0.01)
```

Figura 4-6: Código *Python* para aplicação da filtragem por variantes.

Já o Quadro 4-8 apresenta os modelos gerados para diferentes frequências de variantes. As restantes variantes têm um número inferior de ocorrências. Uma percentagem maior de variantes resultaria num modelo com mais *logs* incluídos, onde a análise de tornaria impossível e ineficiente.

Quadro 4-8: Modelos de diferentes frequências de variantes.

Nº Ocorrências	Nº Variantes	Modelo
10 ou mais	1	
5 ou mais	2	
4 ou mais	3	
3 ou mais	4	
2 ou mais	10	

Já no *ProM*, não foi encontrado nenhum módulo de descoberta de variantes, mas o módulo *"Filter Out Low-Occurrence Traces (Single Log)"* permitiu isolar os casos por número de ocorrência. Assim, foi possível isolar casos com diferentes números de ocorrências, com a desvantagem de não se saber à *priori* que variantes existem. Pela Figura 4-7 é possível analisar todos as etapas deste processo, assim como ver o resultado da análise das variantes com mais ocorrências.

Figura 4-7: Etapas do *ProM* para analise de variantes.

Por fim, no *Disco* a análise de variantes é extremamente simples. Após a importações dos *logs*, é apresentada uma lista completa de todas as variantes do processo, Figura 4-8, onde, para cada uma delas, se pode ver o número de ocorrências, a percentagem no conjunto geral e a lista de *logs* que a variante inclui.

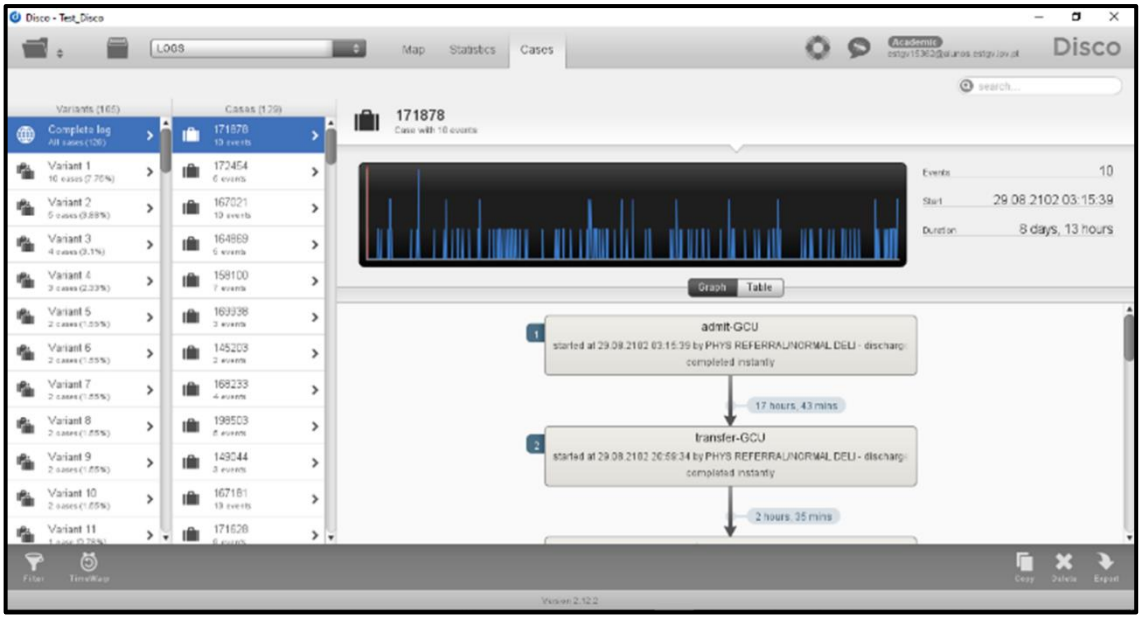


Figura 4-8: Análise de variantes no *Disco*.

Assim, é possível usar uma filtragem onde se pode escolher as variantes a considerar, como se percebe pela Figura 4-9. Selecionaram-se as duas variantes com mais frequência, o que permitiu um modelo mais fácil e eficaz de analisar.

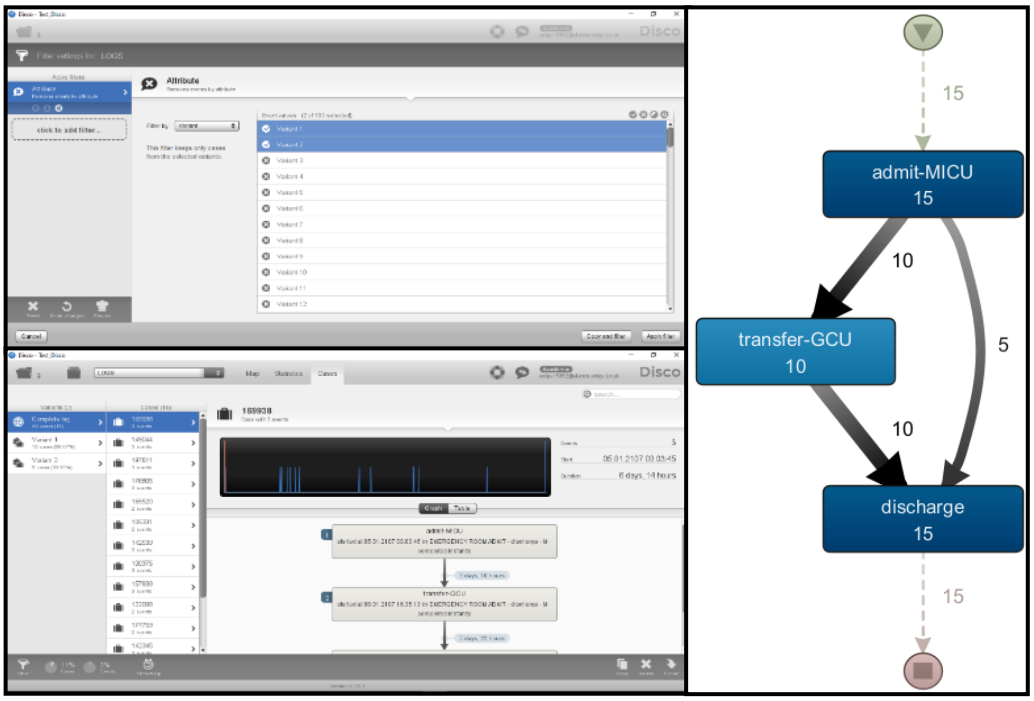
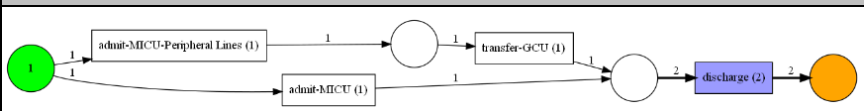
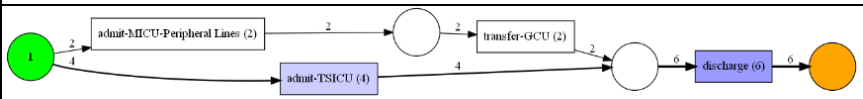
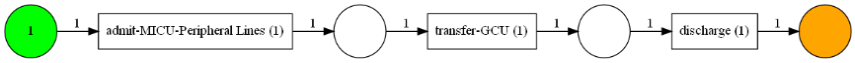


Figura 4-9: Etapas do *Disco* para filtragem por variantes.

4.4 Filtragem de *logs*

Nesta secção foram testadas filtragens no conjunto de *logs*, de forma a fazer análises mais especificas ao processo. No Quadro 4-9, podem-se observar os modelos resultantes para cada filtragem e respetivos valores testados.

Quadro 4-9: Modelos resultantes das diferentes filtragens e respetivos valores testados.

Filtragem	Valores	Modelo
Timeframe	"2161-09-19 17:54:42" "2163-11-21 19:01:00"	
Atividades iniciais	"admit-TSICU" "admit-MICU-Peripheral Lines"	
Valores de atributos	Tipo de admissão: "TRANSFER FROM SKILLED NUR"	

4.4.1 *Timeframe*

Testou-se a filtragem com intervalos de tempo, a partir do *timestamp* dos eventos, neste caso, entre "2161-09-19 17:54:42" e "2163-11-21 19:01:00". No *PM4Py* esta filtragem foi conseguida a partir de uma biblioteca de filtragem, Figura 4-10, onde são definidos o início e fim do intervalo a considerar. Nesta ferramenta, a filtragem por *timestamp* tem em conta todo o processo, isto é, apenas são consideradas admissões que ocorreram totalmente dentro do intervalo de tempo dado.

```
from pm4py.algo.filtering.log.timestamp import timestamp_filter
log = timestamp_filter.filter_traces_contained(log, "2161-09-19 17:54:42", "2163-11-21 19:01:00")
```

Figura 4-10: Código *Python* para aplicação da filtragem por *timeframe*.

No *ProM*, foi utilizado o módulo *"Filter Log on Event Attribute Values"* que permite selecionar os valores a considerar em cada parâmetro dos eventos. Assim, selecionou-se o parâmetro *timestamp* e escolheram-se os valores pretendidos, Figura 4-11. De salientar que a escolha do *timeframe* foi difícil, visto que este módulo não verifica se o caso é todo abrangido pela condição. Deste modo, poderão aparecer casos incompletos que não são interessantes para as análises.

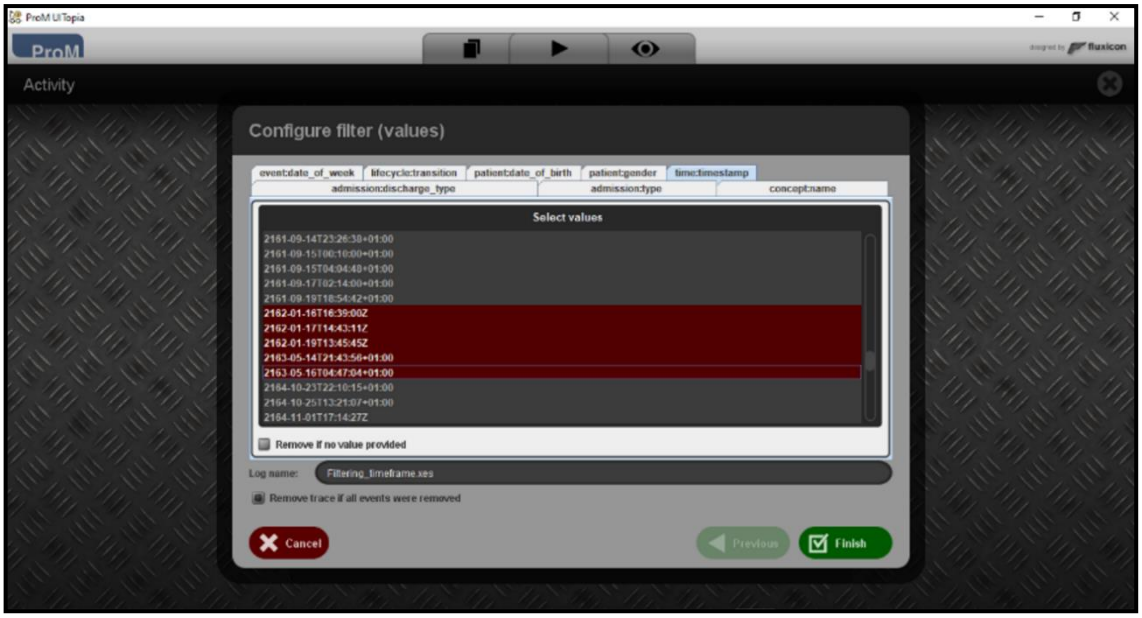


Figura 4-11: Etapa do *ProM* aplicação da filtragem por *timeframe*.

O *Disco* tem a filtragem referida. Consequentemente, selecionou-se o parâmetro *timestamp* e escolheram-se os valores pretendidos. Como se mostra na Figura 4-12, esta ferramenta permite ainda escolher a opção de manter apenas *logs* completos, o que é uma enorme vantagem desta ferramenta.

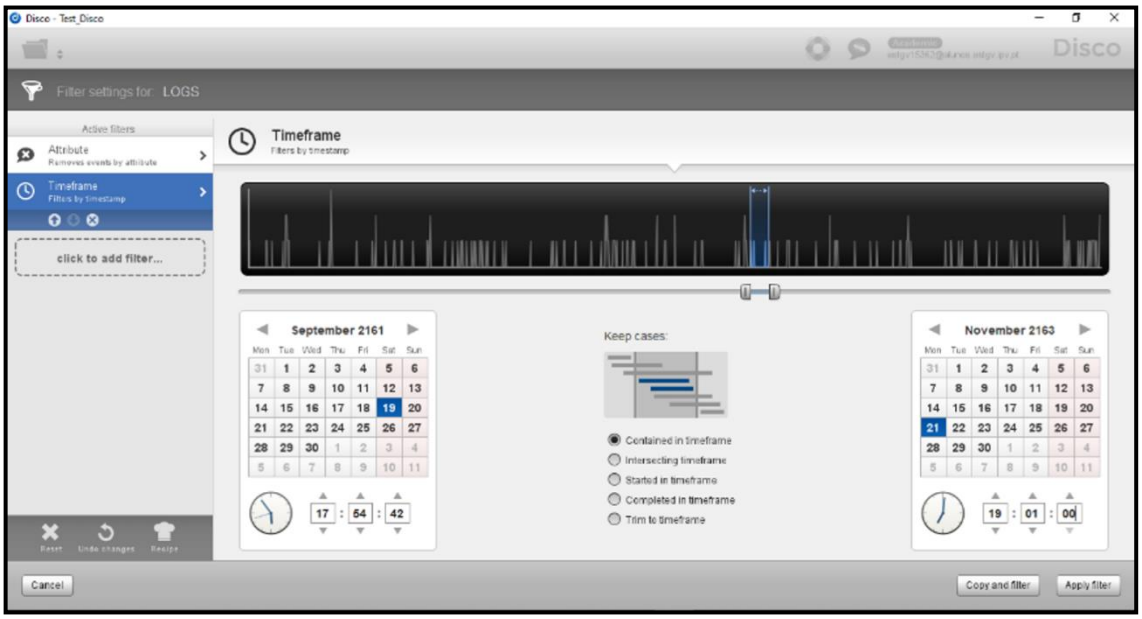


Figura 4-12: Etapa do *Disco* aplicação da filtragem por *timeframe*.

4.4.2 Atividades de início e fim

Num segundo teste, são filtrados os *logs*, cujas atividades inicial e final estão numa lista fornecida. No *PM4Py* existe uma biblioteca para esta filtragem, Figura 4-13, onde é passada a lista de atividades a considerar.

```
from pm4py.algo.filtering.log.start_activities import start_activities_filter
log = start_activities_filter.apply(log, ["admit-TSICU", "admit-MICU-Peripheral Lines"])


from pm4py.algo.filtering.log.end_activities import end_activities_filter
log = end_activities_filter.apply(log, ["discharge"])
```

Figura 4-13: Código *Python* para aplicação da filtragem por atividades de início e fim.

No *ProM*, para este tipo de filtragem foi utilizado um outro modulo, *"Filter Log using Simple Heuristics"*. Este permite escolher diretamente eventos de início, Figura 4-14.

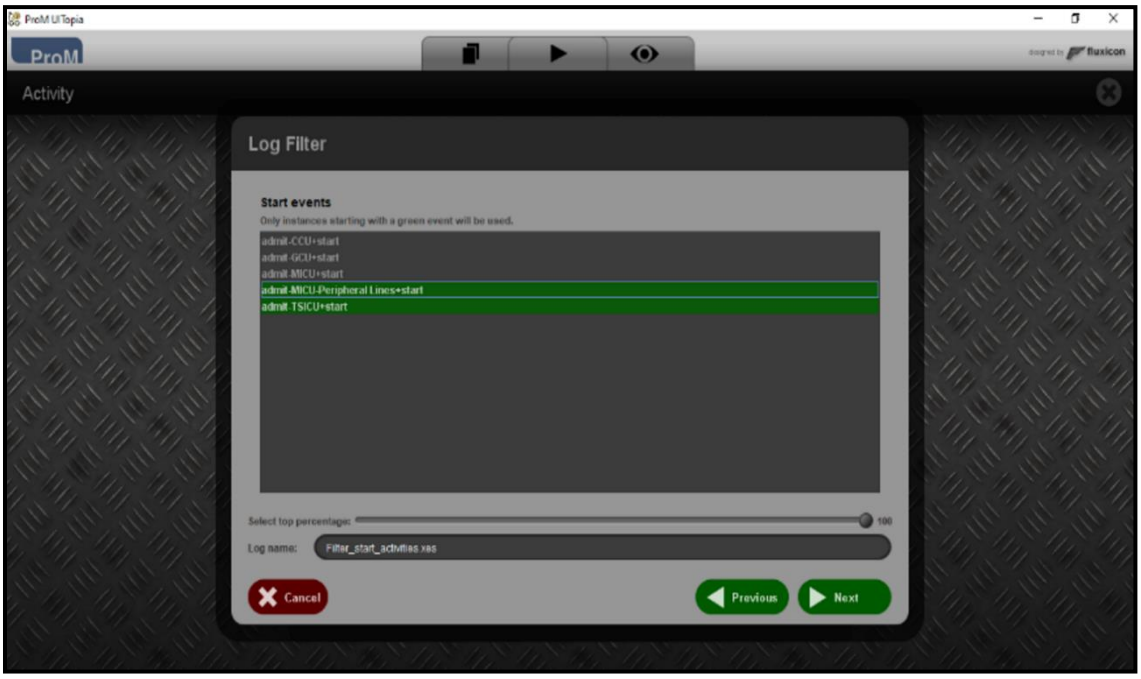


Figura 4-14: Etapa do *ProM* para aplicação da filtragem por atividades de início e fim.

No *Disco*, para este tipo de filtragem, existe um filtro de *Endpoints*, onde se pode definir as atividades de início e fim a considerar. Pela Figura 4-15 percebe-se que foram selecionadas duas etapas de início e uma de fim, no caso a única que existe.

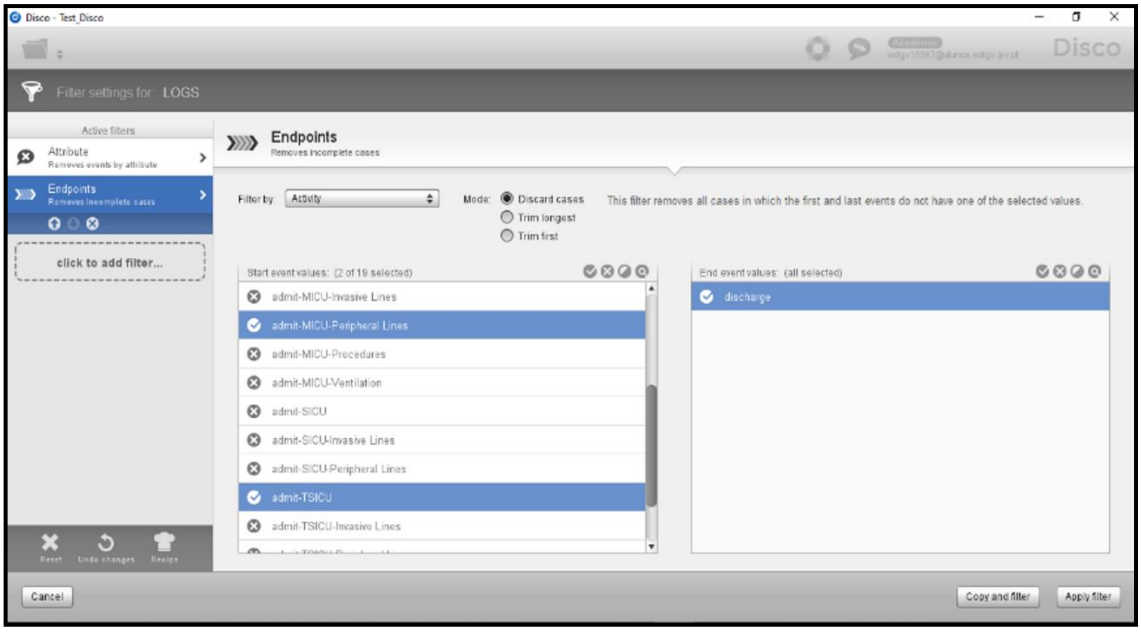


Figura 4-15: Etapa do *Disco* para aplicação da filtragem por atividades de início e fim.

4.4.3 Valores de atributos

Foi realizada uma última filtragem, com um enorme potencial de utilização, visto que utiliza um atributo do *log*, onde é definida uma lista de valores a considerar. A partir do *dataset* utilizado, este filtro permite filtragens por diversos parâmetros que os eventos têm, sendo exemplos o género do paciente, a data de nascimento do paciente, o dia da semana em que os eventos ocorreram, o tipo de admissão, o local de admissão e o tipo de saída (alta ou óbito). Neste exemplo, foram filtrados casos em que os pacientes foram transferidos de enfermarias especializadas, "TRANSFER FROM SKILLED NUR".
O *PM4Py* tem uma biblioteca que permite definir o atributo, a lista de valores a considerar e o carácter da condição, Figura 4-16.

```
from pm4py.algo.filtering.log.attributes import attributes_filter
log = attributes_filter.apply(log, ["TRANSFER FROM SKILLED NUR"],
                              parameters={attributes_filter.Parameters.ATTRIBUTE_KEY: "admission:type",
                                          attributes_filter.Parameters.POSITIVE: True})
```

Figura 4-16: Código *Python* para aplicação da filtragem por valores de atributos.

No *ProM*, o módulo anteriormente utilizado permite outras filtragens, como é o caso da filtragem pelos atributos extra. Na Figura 4-17 é possível ver a aplicação e o resultado de filtragens por tipo de admissão, com o valor "TRANSFER FROM SKILLED NUR".

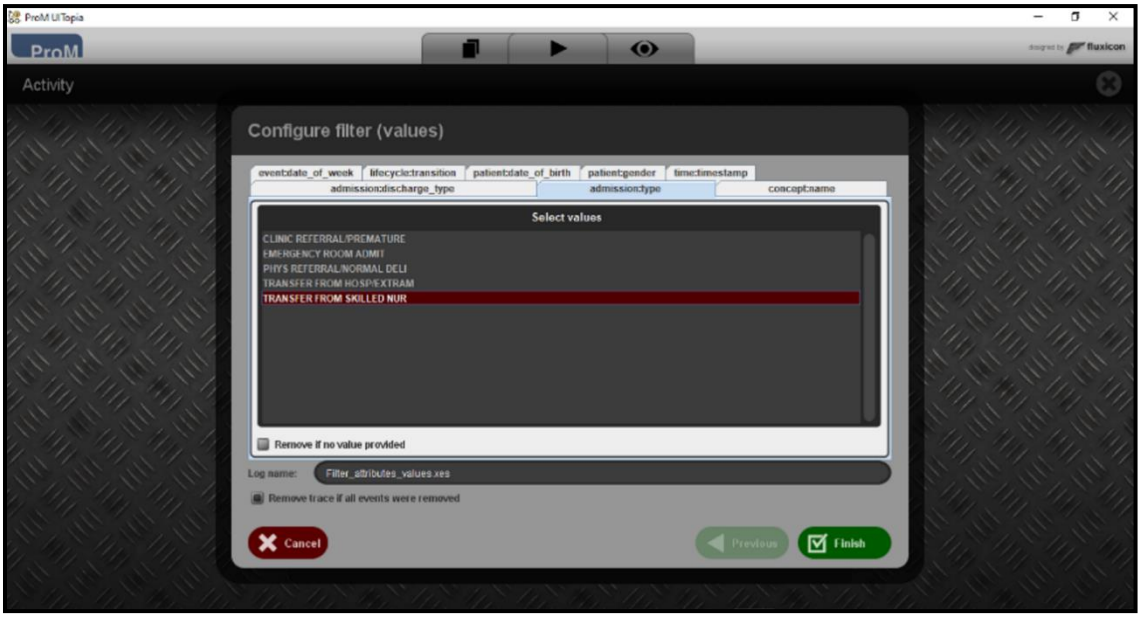


Figura 4-17: Etapa do *ProM* para aplicação da filtragem por valores de atributos.

Finalmente, no *Disco*, foi utilizado o filtro por atributos, onde se seleciona o atributo e os valores a filtrar. No caso apresentado na Figura 4-18, foi selecionado o atributo "admission:type" que contém informação do tipo de entrada do paciente, e foi selecionado o valor "TRANSFER FROM SKILLED NUR" que significa que o paciente foi transferidos de uma enfermaria especializada.

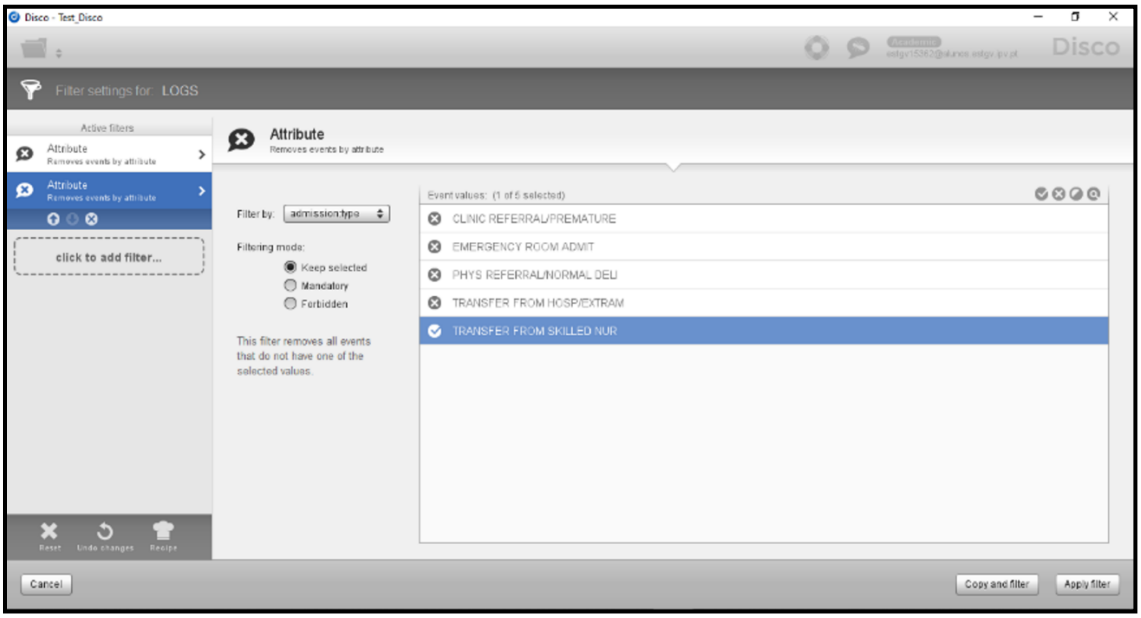


Figura 4-18: Etapa do *Disco* para aplicação da filtragem por valores de atributos.

4.5 Estatísticas sobre os *logs*

No *PM4Py* é possível calcular diferentes estatísticas sobre os *logs* de eventos. Na Figura 4-19 pode-se analisar duas estatísticas ao *dataset*, duração média dos casos e razão de dispersão do caso, que é a distância média entre a conclusão de dois casos consecutivos no *log*.

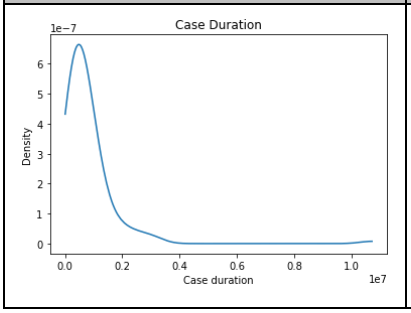
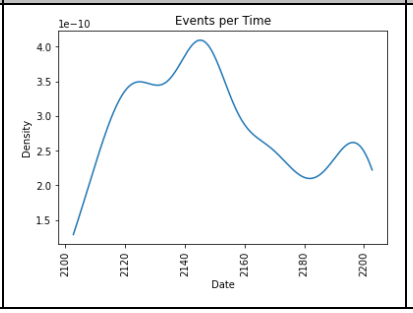
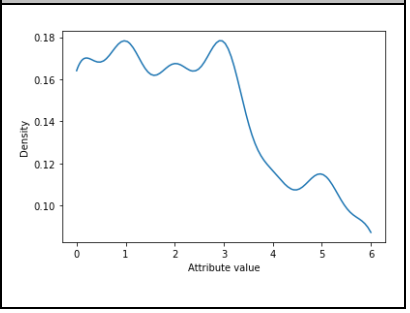
```python
# Median case duration
from pm4py.statistics.traces.log import case_statistics
median_case_duration = case_statistics.get_median_caseduration(log, parameters={
    case_statistics.Parameters.TIMESTAMP_KEY: "time:timestamp"
})

# Case dispersion ratio

from pm4py.statistics.traces.log import case_arrival
case_dispersion_ratio = case_arrival.get_case_dispersion_avg(log, parameters={
    case_arrival.Parameters.TIMESTAMP_KEY: "time:timestamp"})
```

Statistics	
Median Case Duration	Average Distance
571283.0	93077625.03030303

Figura 4-19: Código *Python* e respetivo resultado das estatísticas ao *dataset*.

É, ainda, possível a criação de gráficos, Quadro 4-10, que permitem entender vários aspetos do *dataset* utilizado no modelo, como por exemplo, a distribuição de um atributo numérico, a distribuição da duração do caso, ou a distribuição dos eventos ao longo do tempo.

Quadro 4-10: Gráficos de distribuição dos eventos no *Python*.

Distribuição da duração do caso	Distribuição dos eventos ao longo do tempo	Distribuição dos eventos por dia da semana
		

O código para gerar os gráficos antes apresentados pode ser analisado na Figura 4-20.

```python
# Distribution of case duration
x, y = case_statistics.get_kde_caseduration(log, parameters={constants.PARAMETER_CONSTANT_TIMESTAMP_KEY: "time:timestamp"})
gviz = graphs_visualizer.apply_plot(x, y, variant=graphs_visualizer.Variants.CASES)
graphs_visualizer.save(gviz, "4-Statistics/Distribution of case duration.png")
# Distribution of events over time
x, y = attributes_filter.get_kde_date_attribute(log, attribute="time:timestamp")
gviz = graphs_visualizer.apply_plot(x, y, variant=graphs_visualizer.Variants.DATES)
graphs_visualizer.save(gviz, "4-Statistics/Distribution of events over time.png")
# Distribution of a numeric attribute
attribute = "event:date_of_week"
x, y = attributes_filter.get_kde_numeric_attribute(log, attribute)
gviz = graphs_visualizer.apply_plot(x, y, variant=graphs_visualizer.Variants.ATTRIBUTES)
graphs_visualizer.save(gviz, "4-Statistics/Distribution of "+attribute.split(':')[1]+".png")
```

Figura 4-20: Código *Python* para gerar os gráficos de distribuição dos eventos.

No *ProM* e no *Disco* as estatísticas podem ser analisadas diretamente nos *dashboards* das ferramentas, sem nenhum processo explícito. No caso do *ProM*, apenas é possível ver as estatísticas básicas, Figura 4-21, como o número de casos, número de eventos e gráfico de eventos por caso.

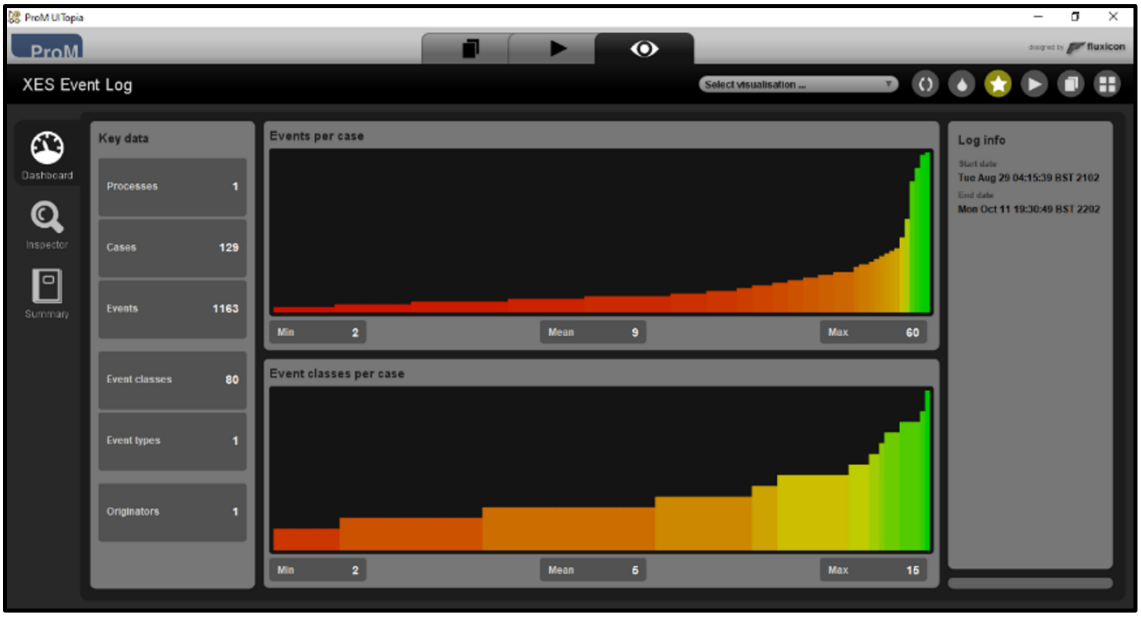


Figura 4-21: Estatísticas no *ProM*.

Já no *Disco* é possível verificar várias estatísticas em relação ao conjunto de dados, assim como para casos específicos. Na Figura 4-22 podem-se ver as estatísticas gerais, onde, ao centro é possível analisar gráficos de número de eventos por tempo, casos por tempo, número de casos por variante, número de eventos por caso e número de casos por duração. Já à direita é possível ver informação acerca de número total de eventos e casos, média e mediana da duração dos casos e data de início e fim do *dataset* geral. É, ainda, apresentada em baixo a tabela com informações acerca dos casos.

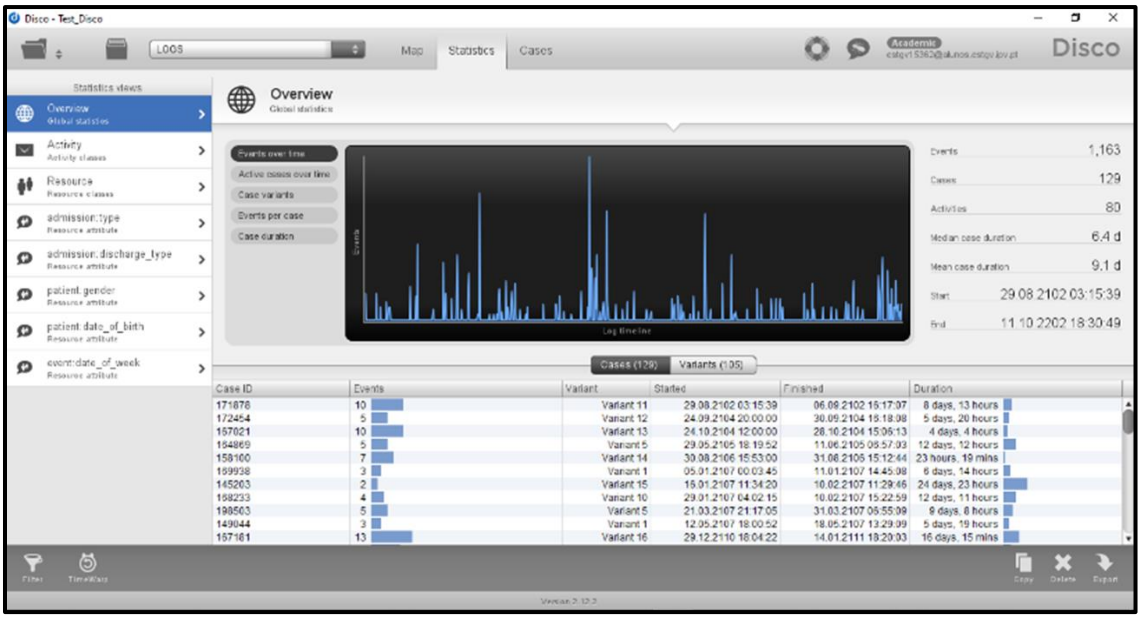


Figura 4-22: Estatísticas gerais no *Disco*.

Na Figura 4-23, é possível ver informação acerca das etapas / atividades do *dataset*, como o número total existente e estatísticas de frequência de ocorrência de uma etapa. Abaixo mostra-se uma tabela com informações de cada etapa diferente.

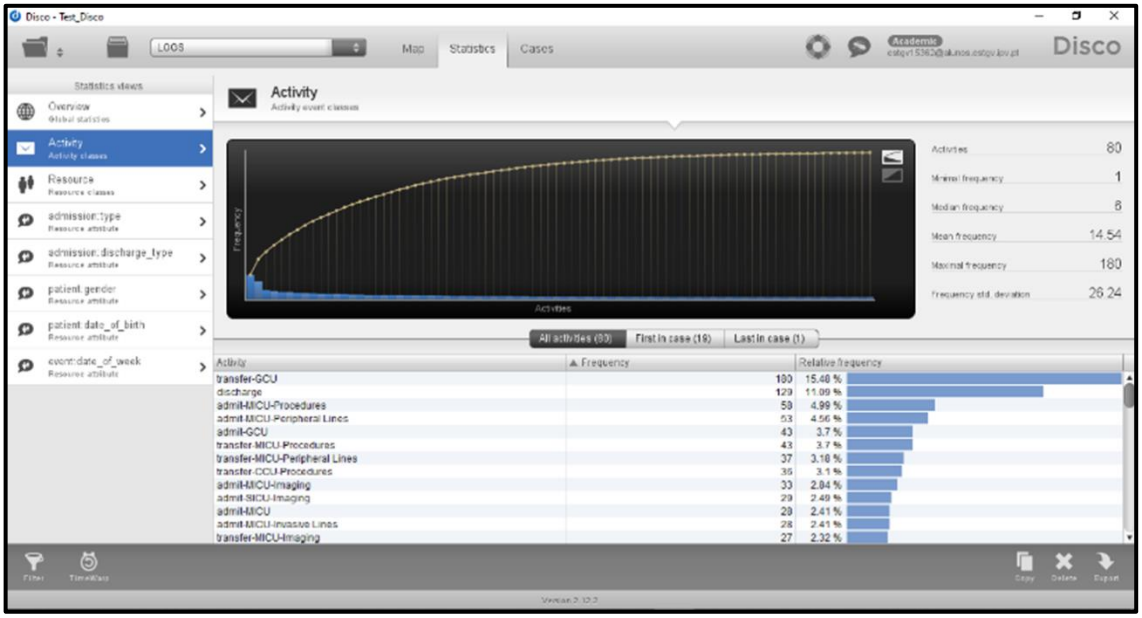


Figura 4-23: Estatísticas das etapas / atividades no *Disco*.

Por fim, na Figura 4-24, é possível verificar um exemplo de *dashboard* sobre as diferentes informações do tipo de admissão. Ao centro aparece a frequência dos valores de cada parâmetro, à direita as suas estatísticas e abaixo uma tabela com informações de cada valor, destacando-se a frequência com que ocorrem.

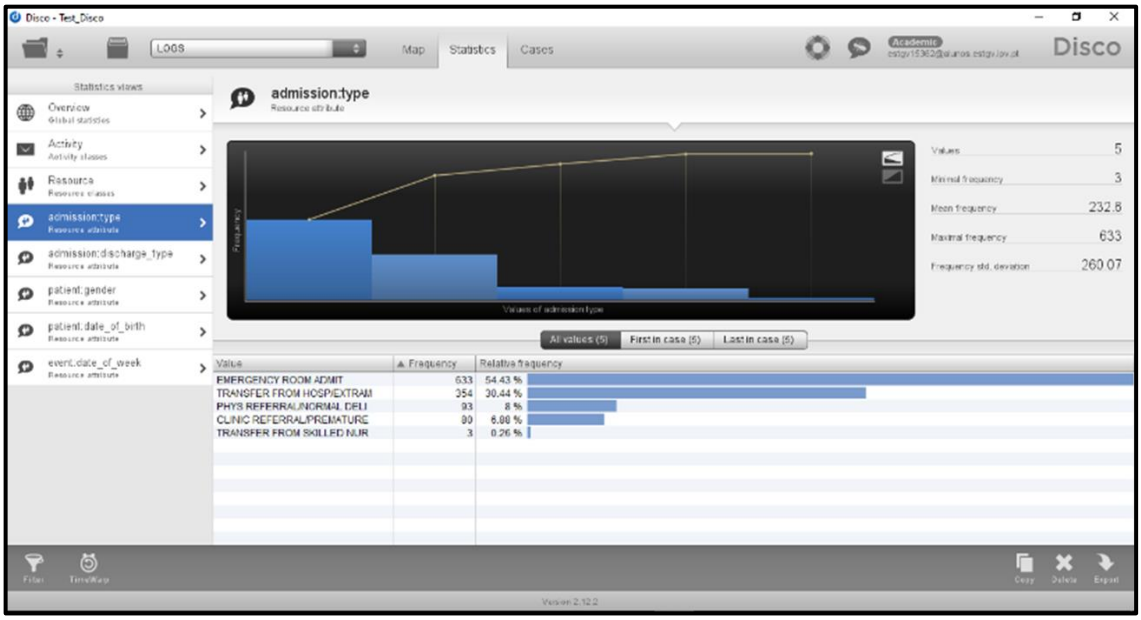


Figura 4-24: Estatísticas dos atributos extra das etapas no *Disco*.

4.6 Verificação de conformidade

A verificação de conformidade é uma técnica para comparar o modelo previsto do processo com um conjunto de *logs* de eventos reais do mesmo processo. O objetivo é verificar se os *logs* estão de acordo com o modelo e vice-versa (Munoz-Gama & Carmona, 2011). De salientar que o *Disco* não suporta esta funcionalidade. O mais próximo é um filtro onde podemos selecionar a atividade de referência e a próxima para verificar se existem *logs* onde isso acontece. Trata-se de algo que não é viável para *logs* longos, pois eram precisos vários filtros sobre filtros. Outro problema é que apenas teríamos informação se o caminho existe ou não no modelo, enquanto o objetivo é saber o quão perto o caminho está do modelo.

No *PM4Py*, podem ser implementadas duas técnicas fundamentais para verificar a conformidade: reprodução baseada em *token* e reprodução baseada em alinhamentos (Pohl, 2019). A reprodução baseada em *token* corresponde a um modelo de rastreamento da rede de *Petri*, partindo do local inicial, para descobrir quais as transições que são executadas e em quais locais se têm *tokens* restantes ou ausentes para a instância do *log* testado. Um *log* está de acordo com o modelo se, durante a sua execução, as transições puderem ser disparadas sem a necessidade de inserir nenhum *token* ausente (Berti & Van der Aalst, 2020).

Para o modelo e o conjunto de *logs* testado, o resultado está representado na Figura 4-25. No primeiro *log* percebe-se que o modelo não o conseguiu satisfazer, *trace_is_fit* é *False*, pois como se pode verificar no *transitions_with_problems* houve uma transição em que o percurso não conseguiu seguir. Daí terem sido consumidos os 9 *tokens* produzidos, mas faltou 1 *token*. Uma vez que conseguiu satisfazer grande parte do percurso, o *trace_fitness* acaba por estar perto do 1, sendo aproximadamente 0.889.

Já para o segundo *log*, vendo o *trace_is_fit* a *True*, percebe-se que o modelo satisfez todas as transições do *log*, tendo consumido todas as *tokens* produzidas, 10, e sem *tokens* restantes ou em falta.

```
from pm4py.algo.conformance.tokenreplay import algorithm as token_replay
replayed_traces = token_replay.apply(log_test, net, initial_marking, final_marking)
```

trace_is_fit	trace_fitness	activated_transitions	reached_marking	enabled_transitions_in_marking	transitions_with_problems	missing_tokens	consumed_tokens	remaining_tokens	produced_tokens
False	0.8888888888888888	[admit-GCU, tauSplit_2, transfer-MICU, transfer-MICU, transfer-GCU, tauJoin_3, discharge]	['p_11:1', 'sink:1']	set()	[transfer-MICU]	1	9	1	9
True	1.0	[admit-MICU, tauSplit_2, transfer-GCU, transfer-MICU, skip_6, transfer-GCU, tauJoin_3, discharge]	['sink:1']	set()	[]	0	10	0	10

Figura 4-25: Resultados da repetição baseada em *token* no *Python*.

A reprodução baseada em alinhamentos visa encontrar um dos melhores alinhamentos entre o *log* e o modelo. Para cada *log*, a saída de um alinhamento é uma lista de pares onde o primeiro elemento é um evento do *log* e o segundo elemento é uma transição do modelo. Para cada par, a seguinte classificação pode ser fornecida (Bloemen et al., 2019):

- Movimento de sincronização: a classificação do evento corresponde ao nome da transição; neste caso, o *log* e o modelo avançam da mesma maneira durante o *replay*;
- Mover no registo: pares onde o segundo elemento é » correspondem a um movimento de repetição no *log* que não é semelhante no modelo. Esse tipo de movimento é impróprio e sinaliza um desvio entre o *log* e o modelo;
- Mover no modelo: pares onde o primeiro elemento é » correspondem a um movimento de repetição no modelo que não é semelhante no *log*. Para movimentos no modelo, pode-se fazer a seguinte distinção:
    - Movimentos no modelo envolvendo transições ocultas: neste caso, mesmo que não seja um movimento sincronizado, o movimento é adequado;
    - Movimentos no modelo que não envolvem transições ocultas: neste caso, o movimento é impróprio e sinaliza um desvio entre o *log* e o modelo.

A cada verificação de conformidade do *log*, é associado um dicionário contendo, entre os demais, as seguintes informações:

- *alignment*: contém o alinhamento (movimentos de sincronização, movimentos no registo, movimentos no modelo);
- *cost*: contém o custo do alinhamento de acordo com a função de custo fornecida, que pode ser personalizada;
- *fitness*: é igual a 1 se o *log* for perfeitamente adequado.

Para o modelo e o conjunto de *logs* testado, pode-se analisar que o primeiro *log* teve um *fitness*, isto é, uma adequação ao modelo perto de 1, o que indica que não conseguiu concluir todo o percurso do processo a partir do modelo utilizado, ao contrário do segundo, onde o processo é capaz de finalizar o percurso do *log*, Figura 4-26.

```
from pm4py.algo.conformance.alignments import algorithm as alignments
alignments = alignments.apply_log(log_test, net, initial_marking, final_marking)
```

alignment	cost	queued_states	visited_states	closed_set_length	num_visited_markings	exact_heu_calculations	fitness
[('admit-GCU', 'admit-GCU'), ('transfer-MICU', '>>'), ('>>', None), ('transfer-MICU', 'transfer-MICU'), ('transfer-GCU', 'transfer-GCU'), ('>>', None), ('discharge', 'discharge')]	10000	20	10	7	8	3	0.8571428571428572
[('admit-MICU', 'admit-MICU'), ('>>', None), ('transfer-GCU', 'transfer-GCU'), ('transfer-MICU', 'transfer-MICU'), ('>>', None), ('transfer-GCU', 'transfer-GCU'), ('>>', None), ('discharge', 'discharge')]	0	21	10	8	8	1	1.0

Figura 4-26: Resultados da reprodução baseada em alinhamento no *Python*.

No *ProM*, a verificação de conformidade pode ser implementada a partir do modelo *"Replay a Log on Petri Net for Conformance Analysis"*. Nesse módulo do *ProM*, é possível medir o *fitness* e é, ainda, possível medir o alinhamento entre os *logs* e o modelo.

Como se pode verificar pela Figura 4-27, nos resultados do alinhamento, o *fitness* do primeiro *log* foi 1, sem nenhum custo de alinhamento, pois este caso pertence a uma variante do modelo, fazendo com que o processo real esteja perfeitamente alinhado com o modelo previsto. Já para o segundo *log*, o *fitness* é de 0.83, mostrando que o processo real não está plenamente em conformidade com o modelo previsto. Assim, como se pode ver no alinhamento, foi necessário inserir um novo evento, que estava em falta no modelo, para suportar o caso. Perante isto pode-se ver que a aptidão (*fitness*) do modelo para suportar os dois *logs* é de 0.92.

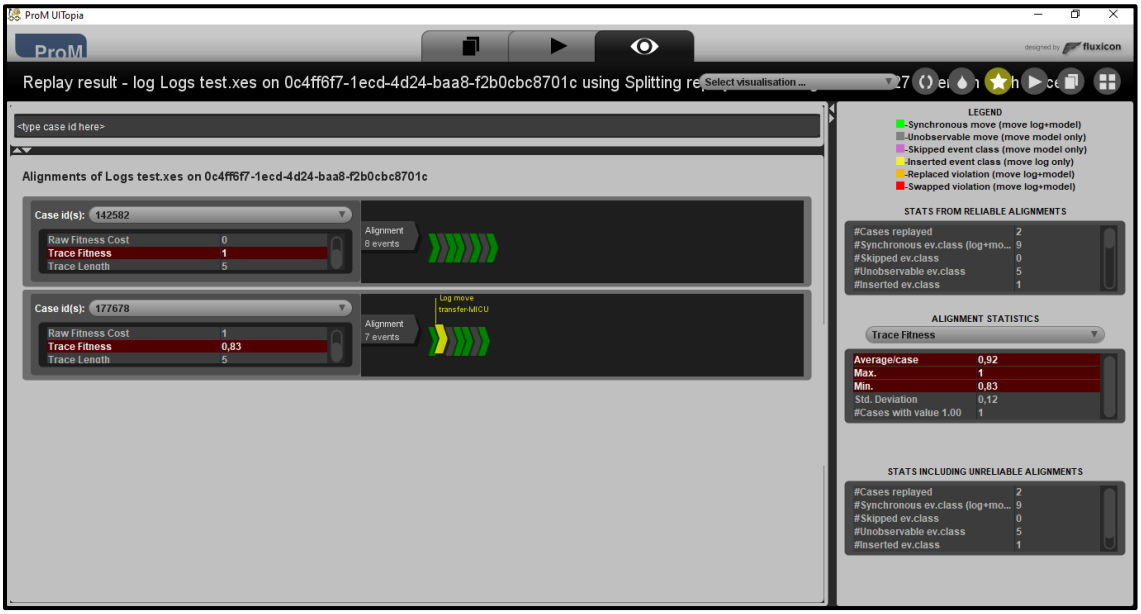


Figura 4-27: Resultados da verificação de conformidade do modulo do *ProM*.

Estes resultados também podem ser analisados de forma gráfica, Figura 4-28. As transições com contorno a verde mostram onde o modelo estava de acordo com o *log* e as transições com a barra no fundo em lilás e contorno vermelho indicam que o modelo não estava em conformidade com o *log*. De salientar que é, ainda, possível perceber que a intensidade da cor de fundo das transições mostra a sua frequência de ocorrência (quanto mais escuras mais ocorrências). Quanto às transições ocultas, quando apresentam a barra no fundo em lilás significa que algum *log* passou por elas, mas não foram necessárias. Quando apresentam a marcação a cinza é porque nenhum *log* passou por ela.
Os círculos brancos indicam os trajetos que foram seguidos em conformidade com o modelo, enquanto os círculos amarelos indicam movimentos fora do modelo, ou seja, não conformes. Os círculos amarelos maiores assinalam as movimentações mais frequentes.

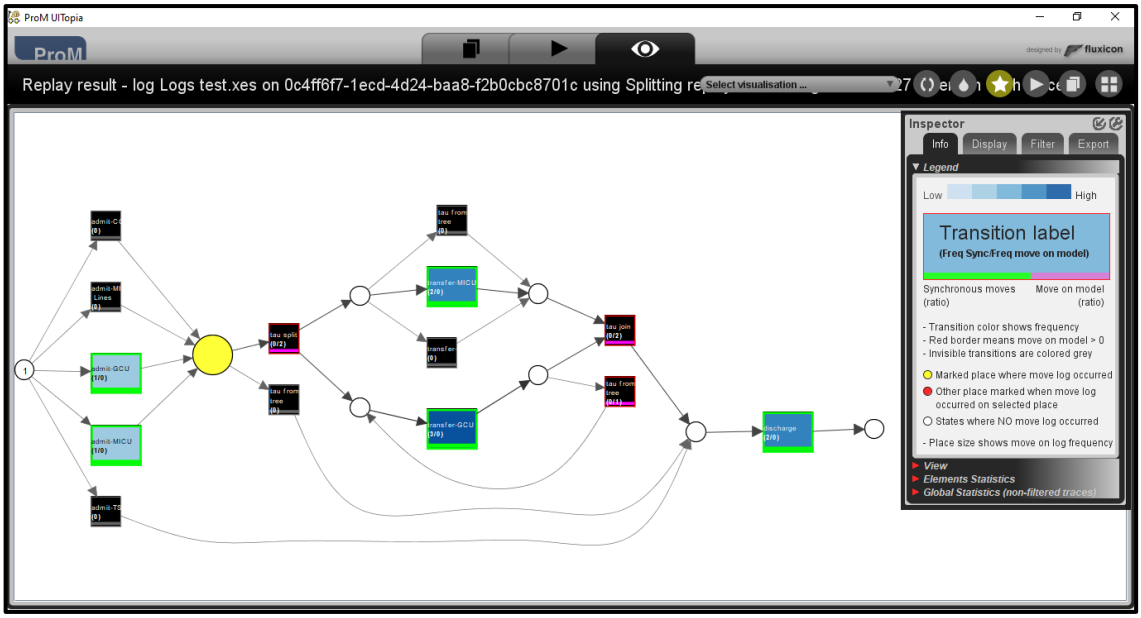


Figura 4-28: Modelo dos resultados da verificação de conformidade do modulo do *ProM*.

4.7 Análise dos resultados

De acordo com as análises feitas anteriormente, nesta secção serão sintetizados os resultados de forma a ter uma classificação sistemática e comparativa das ferramentas de *Process Mining* utilizadas. O objetivo desta secção é perceber qual ou quais as melhores ferramentas a utilizar em *Process Mining*, de forma a que a análise seja a melhor possível.

Com base nos testes efetuados às ferramentas de descoberta de processos, pode-se concluir que o *Disco* é uma ferramenta extremamente simples de utilizar para as funcionalidades que disponibiliza. Apenas tem o aspeto negativo de não incluir funcionalidades de verificação de conformidade, onde o *ProM* se mostrou muito intuitivo e eficaz.

É importante salientar que ambas as ferramentas, *Disco* e *ProM*, mostraram muitas limitações a nível de personalização de mais baixo nível, algo a que o *PM4Py* se propõe. Esta *framework* mostrou-se bastante completa e com uma possibilidade grande de personalização, permitindo a utilização de todas as funcionalidades apresentadas, sem necessidade de muito código ou aprendizagem.

Pelo exposto, se o profissional não tiver muito conhecimento em *Process Mining* ou quiser (e confiar) uma implementação mais automática, a melhor opção será a utilização do *Disco*, até obter o modelo do processo, para depois fazer a verificação de conformidade no *ProM*. Mas caso o profissional tenha conhecimento em *Process Mining* e pretenda uma implementação mais personalizada, o *PM4Py* poderá ser melhor opção.

Tendo em conta apenas fermentas gratuitas, a melhor opção seria a utilização do *PM4Py* para a descoberta dos processos e a utilização do *ProM* para a verificação de conformidade, pois foi a ferramenta que mais se destacou neste tópico.

O Quadro 4-11 contém o resumo das conclusões deste estudo, apresentando a melhor ferramenta para cada tópico, tendo em conta as características custo, nível de personalização e o nível de conhecimento necessário.

Quadro 4-11: Resultados da comparação das ferramentas.

Tópico	Menor Custo	Maior Personalização	Menor Conhecimento
Importação do conjunto de *logs*	*PM4Py*	*PM4Py*	*Disco*
Descoberta de processos	*PM4Py*	*PM4Py*	*Disco*
Análise de variantes	*PM4Py*	*PM4Py*	*Disco*
Filtragem de *logs*	*PM4Py*	*PM4Py*	*Disco*
Verificação de conformidade	*ProM*	*PM4Py*	*ProM*
Estatísticas sobre os *logs*	*PM4Py*	*PM4Py*	*Disco*

5. Aplicação de *Process Mining* a um serviço de urgências

Neste capítulo pretende-se mostrar os benefícios da utilização do *Process Mining* na melhoria de processos de saúde, neste caso num serviço de urgências (Gomes et al., 2021a).
Para tal, será feita a análise a um serviço de urgências, visando perceber de que forma o *Process Mining* pode ajudar a fornecer informações que permitam agir sobre as ineficiências existentes, e projetar intervenções que podem diminuir os tempos de espera, reduzir o congestionamento do paciente, aumentar a qualidade do atendimento e, eventualmente, reduzir os custos.
Como foi concluído na fase experimental, o *Disco* é a ferramenta mais simples, intuitiva e que produz melhores resultados para a descoberta do processo. Como tal, primeiramente serão analisadas as configurações do *dataset* e da ferramenta *Disco*. Posteriormente serão apresentadas estatísticas do conjunto final de dados utilizado, de forma a conhecê-los melhor.
De seguida, será feita uma apresentação do que são os serviços de urgências, assim como do que é aceite, na literatura, como o modelo previsto para este serviço. De salientar que o ideal seria ter um modelo previsto referente ao conjunto de dados em análise, mas este não foi disponibilizado pelo *MIMIC-III*.
Finalmente, será apresentado o modelo real do processo, resultante da descoberta do processo no *Disco*. Para a fase de verificação de conformidade, isto é, para a comparação entre os modelos previsto e *logs* reais do processo, será utilizada a ferramenta *ProM*.

5.1 Preparação da análise

Para a análise do processo, começou-se por examinar os tipos de configurações e análises que o *dataset* e a ferramenta *Disco* permitem. Assim, a nível de configurações considerou-se importante fazer uma filtragem pelas variantes mais comuns do processo, de forma a eliminar

casos inválidos ou pontuais que existam nos dados. Neste tipo de aplicação as dificuldades estão em perceber quais variantes são ruído, sob pena de eliminar casos de interesse para a análise.
Em relação ao modelo gerado, as configurações disponíveis estão relacionadas com as métricas a considerar na análise e com a percentagem de etapas e ligações a mostrar. No primeiro, a ferramenta permite definir uma das duas métricas: frequência, que corresponde ao número de ocorrência das etapas e ligações; desempenho, medido pelo tempo de duração das etapas e ligações. Assim, considerando os objetivos deste projeto, a análise por frequência é a mais indicada. Como a ferramenta permite a configuração de uma segunda métrica a ser apresentada, considerou-se de interesse apresentar uma métrica de desempenho, que será a duração média das etapas e ligações.
Como se pretende uma análise aos serviços de urgências, a filtragem por tipo de admissão faz todo o sentido, neste caso o filtro será por 'Emergência' e 'Urgência', Figura 5-1. Estes dois valores são comumente utilizados em conjunto neste tipo de trabalhos (Johnson et al., 2016).
A principal diferença entre esses dois estados é que uma emergência apresenta ameaça imediata para o bem-estar, enquanto a urgência é uma ameaça num futuro próximo, que pode vir a tornar-se uma emergência se não for solucionada. Na emergência, o aparecimento é súbito e imprevisto, exigindo solução imediata, e na urgência não. Porém a solução deve ser no curto prazo (Romani et al., 2009).

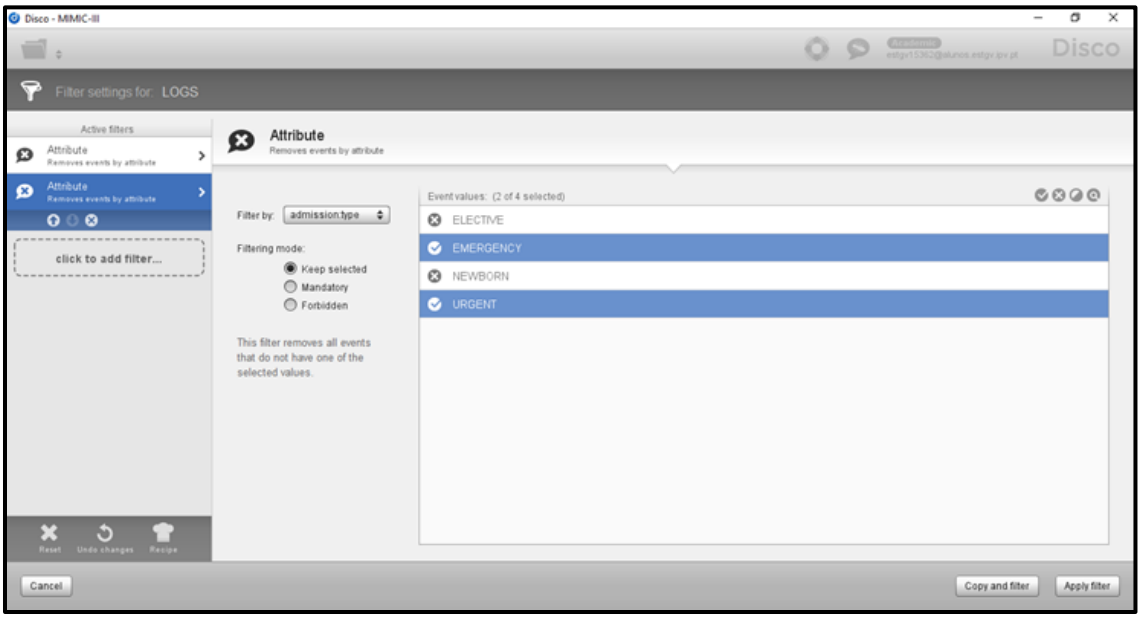


Figura 5-1: Filtragem por tipo de admissão no *Disco*.

Quanto à filtragem por variantes, a percentagem de etapas e ligações a considerar, a partir do número de ocorrências que elas apresentam, é algo que se deve utilizar com extremo cuidado. Apesar de se querer um modelo simples que permita uma análise otimizada, numa primeira análise não se devem excluir muitos casos ou variantes, sob pena de se perder informação relevante.
Com as configurações a utilizar definidas, começou-se por analisar as variantes existentes, de modo a eliminar o ruído e manter apenas as mais relevantes para o modelo do processo. Como

se pode verificar pelo Quadro 5-1, as variantes foram agrupadas por percentagem para uma observação mais fácil, visto que são 3936 variantes ao todo. Pela coloração, percebe-se que foram consideradas variantes acima dos 0.5%. Esta decisão foi tomada tendo em conta as variantes que incluam todas as unidades de cuidados, de forma ter uma análise de todo o serviço.

Quadro 5-1: Lista das variantes do processo.

Variantes	Percentagens	Nº de casos
1	9.26%	5458
2	4.5%	2651
3 - 4	3.54% - 3.28%	2087 - 1931
5 - 10	2.98% - 2.29%	1754 - 1352
11 - 20	1.74% - 1.04%	1025 - 613
21 - 38	0.97% - 0.5%	572 - 293
39 - 119	0.48% - 0.11%	282 - 64
120 - 3936	0.09% - 0%	56 - 1

5.2 Estatísticas dos dados

Após a aplicação das configurações e filtragens anteriormente descritas, verifica-se que, a partir do *dataset* carregado, serão utilizados 46% dos casos, correspondentes a 37 variantes. Na Figura 5-2 podem-se observar estatísticas e informações gerais, como o número de eventos, número de casos e número de etapas que formam o conjunto de dados, a mediana e média da duração dos casos, e o intervalo de datas em que os casos ocorreram.

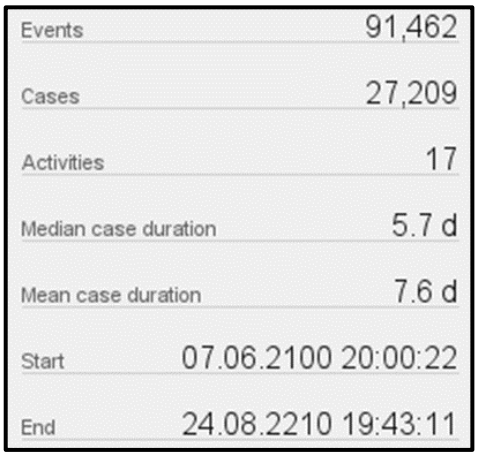


Figura 5-2: Estatísticas e informações gerais dos dados.

Na Figura 5-3, é possível perceber a dispersão dos casos pelo intervalo de tempo.

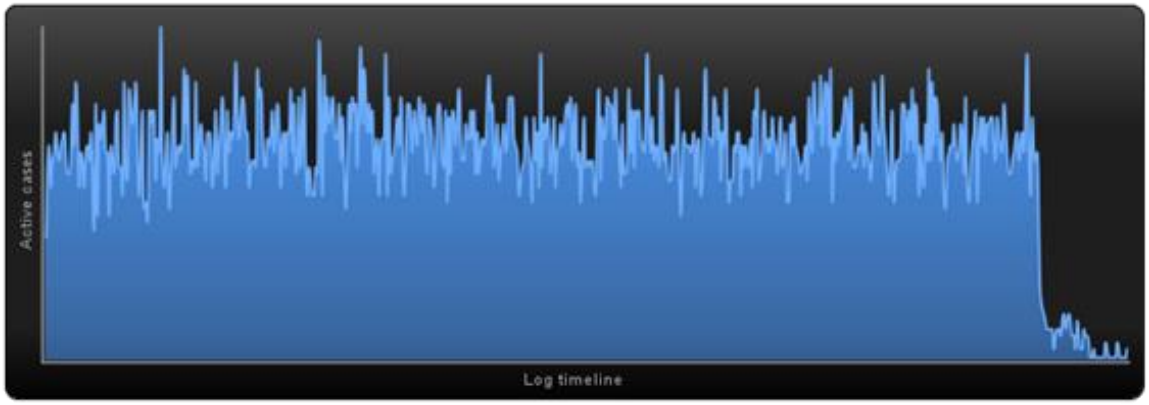


Figura 5-3: Dispersão dos casos pelo intervalo de tempo.

Outro gráfico de interesse disponível na ferramenta é o apresentado na Figura 5-4 que mostra o número de casos por variante, onde se percebe que existe uma variante mais relevante no processo.

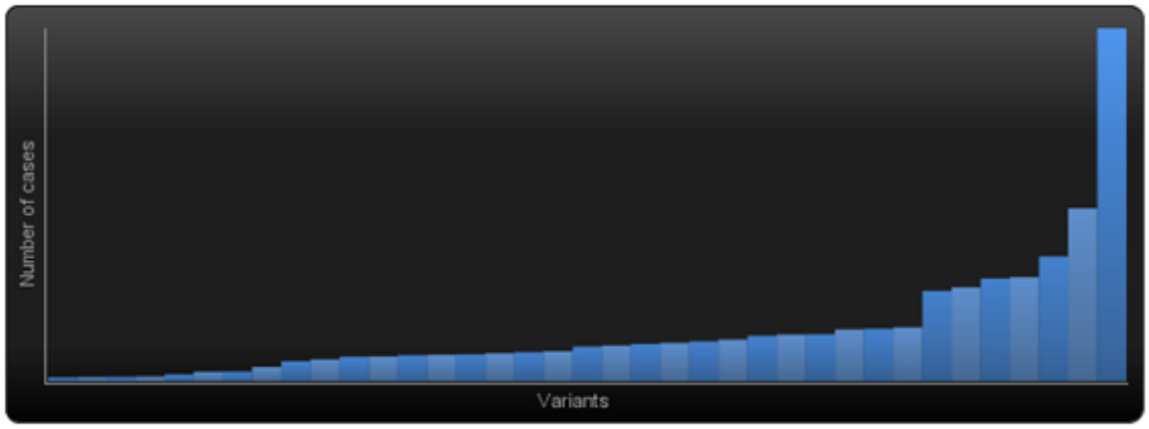


Figura 5-4: Número de casos por variante do processo.

Na Figura 5-5 apresenta-se o número de eventos por caso, sendo 3 os eventos mais comuns.

Figura 5-5: Número de eventos por caso.

A duração dos casos pode ser visualizada na Figura 5-6, onde o caso com mais ocorrência durou 8 dias e 18 horas e o segundo 17 dias e 12 horas.

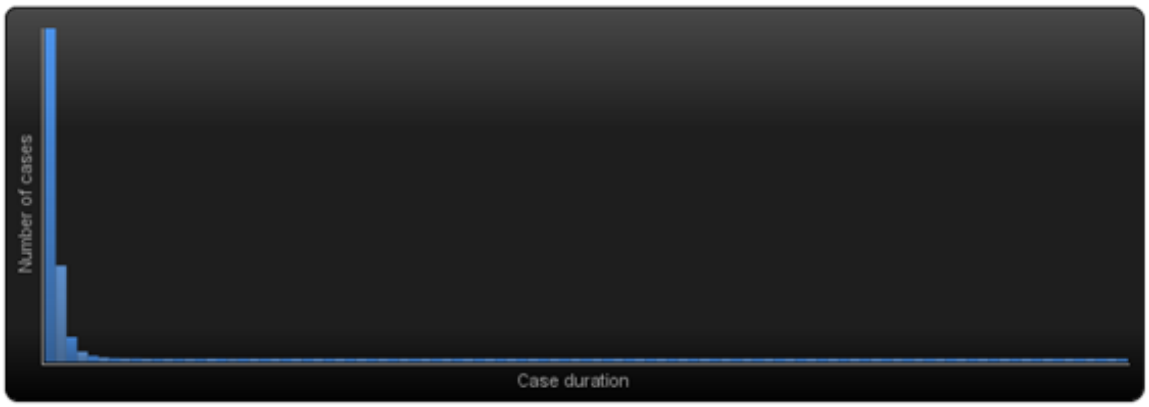


Figura 5-6: Duração dos casos.

Quanto às etapas / atividades, as estatísticas são apresentadas na Figura 5-7, onde se mostra que o conjunto de dados contém 17 etapas diferentes e apresenta uma série de métricas de frequência dessas etapas.

Activities	17
Minimal frequency	5
Median frequency	1,829
Mean frequency	5,380.12
Maximal frequency	28,351
Frequency std. deviation	8,922.66

Figura 5-7: Estatísticas e informações das atividades.

Pela Figura 5-8 pode-se ver a frequência das etapas, sendo as mais comuns “transfer-GCU” e “discharge”, transferência para a unidade de cuidados gerais e a alta do paciente, respetivamente.

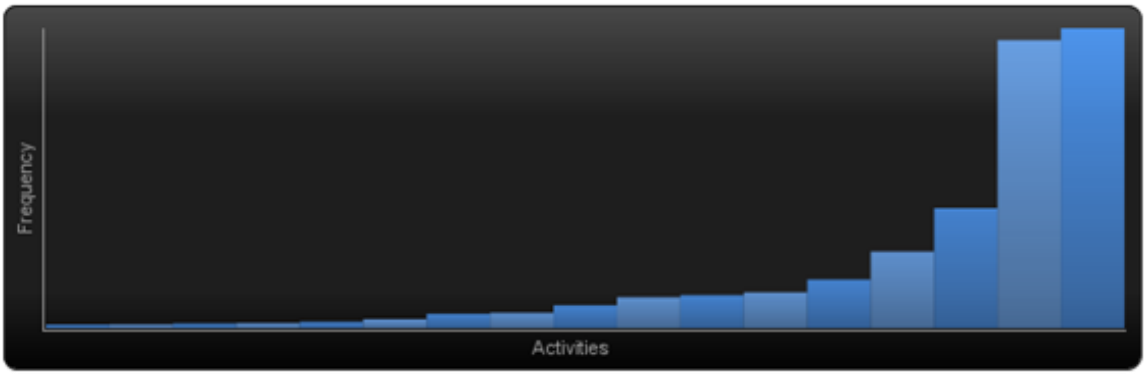


Figura 5-8: Frequência das etapas.

Na Figura 5-9 mostra-se a duração média das etapas. As duas mais comuns duram, em média, 8 dias e 17 horas, e 7 dias e 56 minutos, ambas relativas à unidade de cuidados intensivos neonatal.

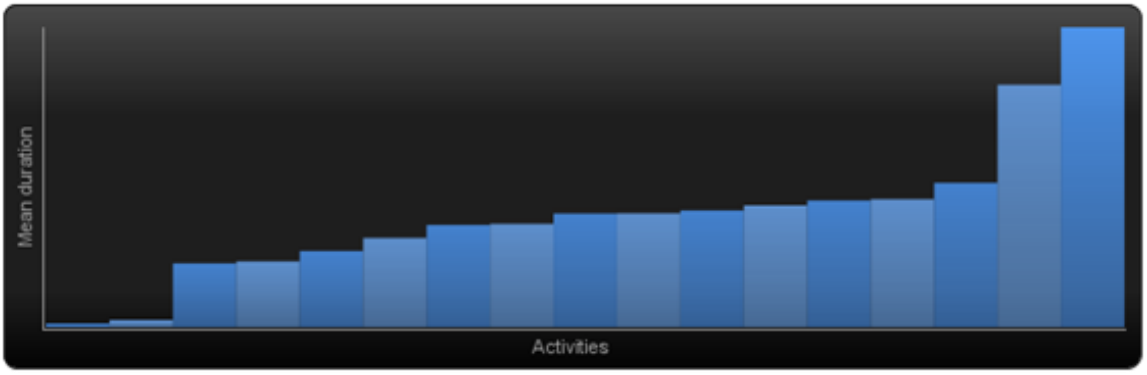


Figura 5-9: Duração média das etapas.

Relativamente às informações extra, pode-se ver, na Figura 5-10, a percentagem de casos de emergência e urgência, que foram os tipos de admissão considerados para a análise do processo. Os casos de emergência são maioritários.

Figura 5-10: Percentagem de casos por tipo de admissão.

Já a nível de locais de admissão, pela Figura 5-11, compreende-se que os mais comuns são a sala de admissão de emergência, encaminhamento clínico e transferência de um hospital, sendo o primeiro o mais expressivo com mais de metade dos casos.

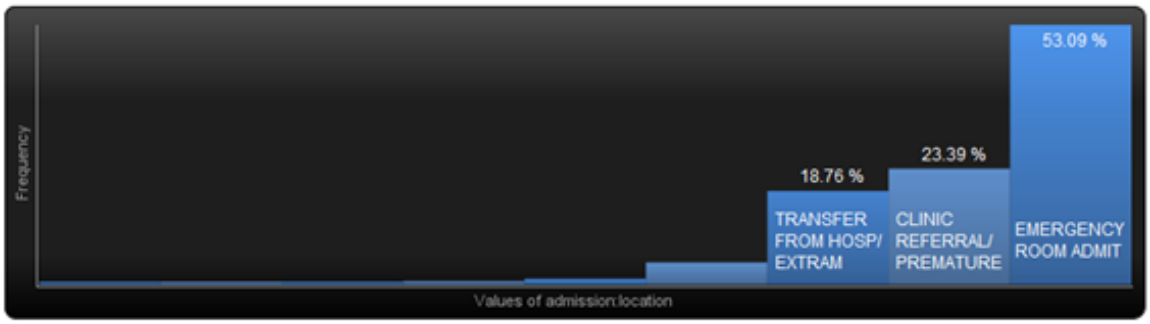


Figura 5-11: Percentagem de casos por local de admissão.

Outra informação de interesse é quanto ao tipo de alta, morte ou alta hospitalar, onde se percebe que a mortalidade ocorre numa minoria dos casos, menos de 10%, Figura 5-12.

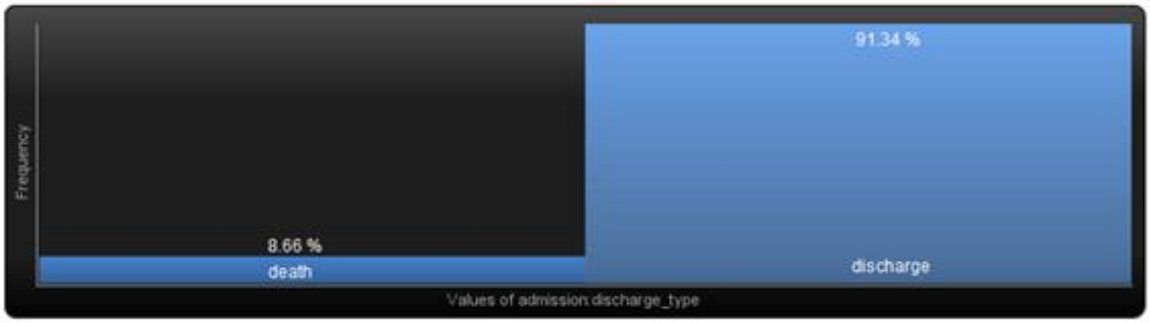


Figura 5-12: Percentagem de casos por tipo de alta.

Analisando os dados respeitantes ao género dos pacientes, Figura 5-13, percebe-se que é algo relativamente equilibrado, sendo que o género masculino tem uma diferença de cerca de 10% em relação ao feminino.

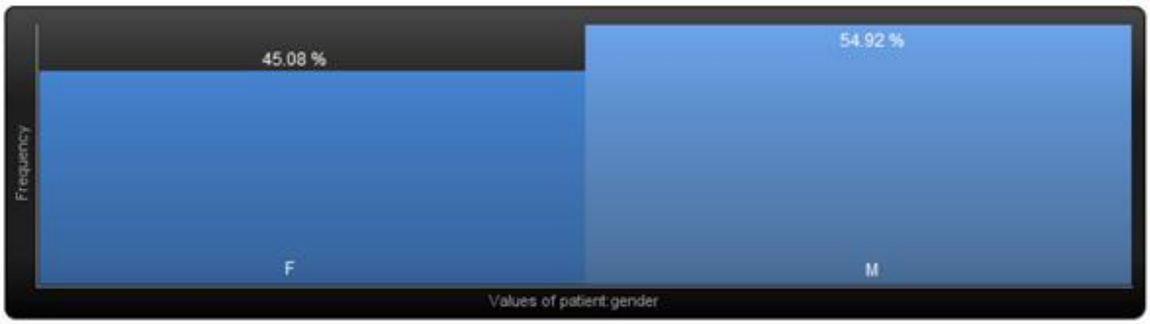


Figura 5-13: Percentagem de casos por género dos pacientes.

Uma análise de elevado interesse para esta área é a dispersão de casos em relação aos dias da semana. Segunda-feira é representada por 0 e o domingo por 6. A Figura 5-14 mostra que

domingo é o dia com menos casos e sexta-feira é o dia mais concorrido. No entanto, a diferença entre estes dois dias é de apenas 5%.

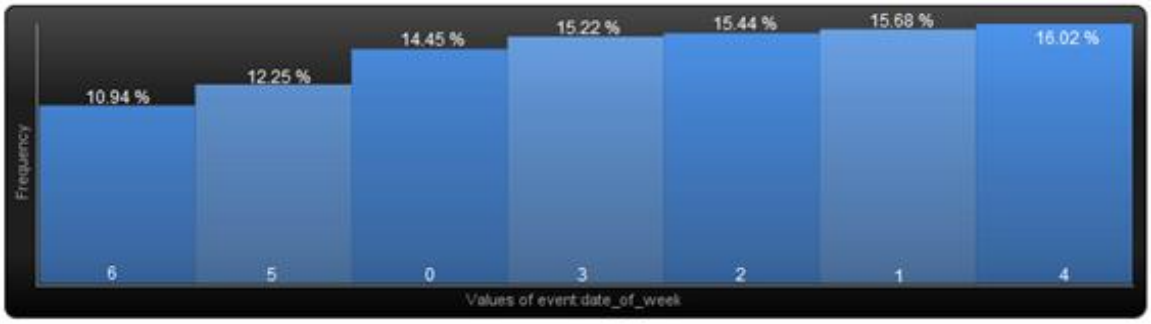


Figura 5-14: Percentagem de casos por dia da semana em que ocorreram.

Por fim, na Figura 5-15, exibe-se a dispersão de datas de nascimento dos pacientes. Este tipo de informação revela-se de enorme importância, pois permite uma análise por idades, mas sendo datas alteradas devido à anonimização dos dados, esta análise não seria realista.

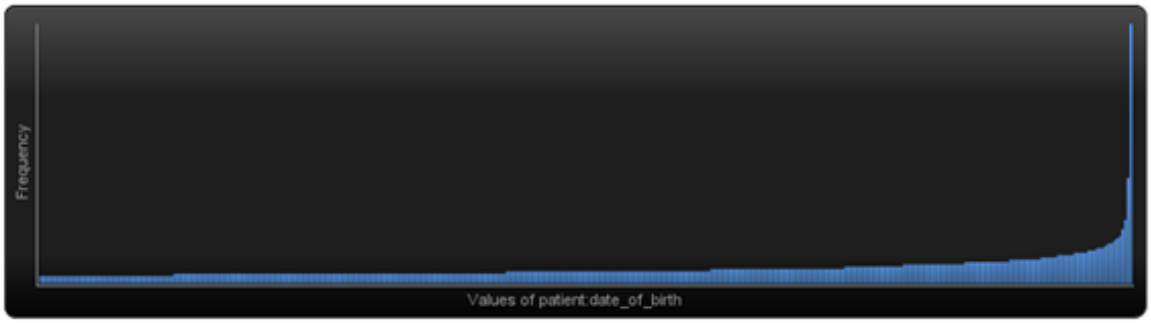


Figura 5-15: Percentagem de casos por data de nascimento dos pacientes.

5.3 Modelo previsto

O serviço de urgências funciona em estreita articulação com a Unidade de Cuidados Intensivos e restantes especialidades, nomeadamente as que suportam a realização dos diversos meios complementares de diagnóstico (Yousefi & Yousefi, 2020).
O fluxo dos pacientes num serviço de urgência, isto é, o modelo previsto para este serviço, com base na literatura, pode ser visualizado na Figura 5-16. Os pacientes chegam aos serviços de urgências e, em seguida, aguardam o atendimento na admissão para fazerem o respetivo registo. Depois, os pacientes aguardam a disponibilidade da sala de triagem para atendimento. Na sala de triagem, a gravidade de cada paciente é verificada com base no *Manchester Triage System* (*MTS*) (Azeredo et al., 2015) que categoriza os pacientes em cinco grupos prioritários: Imediato (Vermelho), Muito Urgente (Laranja), Urgência (Amarelo), Padrão (Verde) e Não Urgente (Azul).

Em seguida, cada paciente aguarda o atendimento. Quando é atendido, pode passar por diferentes unidades de cuidados, recebendo o tratamento necessário. Finalmente, passa para observação, caso precise desse cuidado antes de sair, ou tem alta imediatamente após tratamento (Yousefi & Yousefi, 2020).

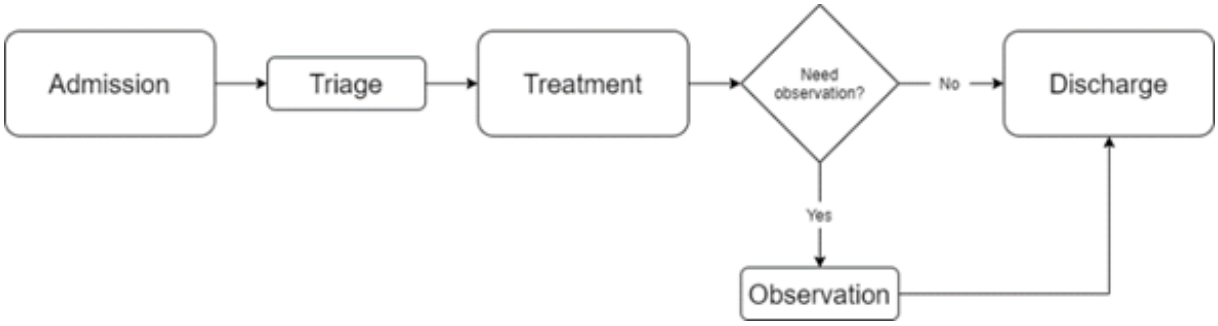


Figura 5-16: Fluxo dos pacientes no serviço de urgências
(Adaptado de Yousefi & Yousefi, 2020).

Devido à indisponibilidade de obter o modelo previsto do processo em análise, este será o modelo utilizado na verificação da conformidade.

5.4 Descoberta do Processo

Nesta secção vão ser apresentados os resultados obtidos na descoberta do processo. A Figura 5-17 representa o modelo gerado pelo *Disco*, com as configurações anteriormente explicadas. Foi dado como prioritário englobar todas as unidades de cuidados e o modelo não foi simplificado para não se perderem etapas e ligações que podem ser de interesse. O resultado é um modelo grande e difícil de analisar como um todo. O modelo real do processo pode ser analisado em maior escala no Anexo 7.

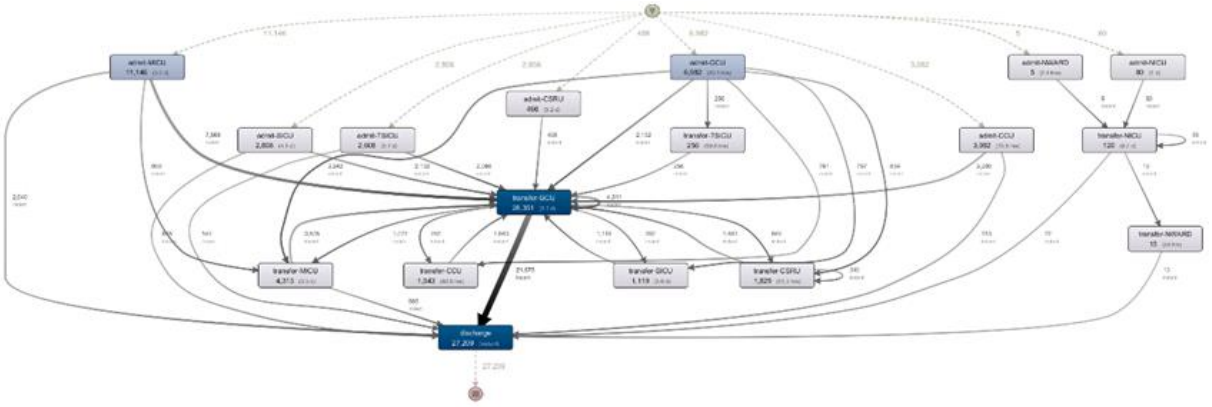


Figura 5-17: Modelo real do processo.

Assim, para uma análise mais fácil e eficaz, o modelo foi partido por unidade de cuidados, a partir da filtragem pelas diferentes admissões existentes. Tendo em conta que existem áreas

muito diferentes de atuação num hospital, foram construídos 5 modelos, um modelo referente às unidades que envolvem problemas cardíacos, um modelo relativo às unidades de cirurgia, um modelo respeitante às unidades neonatal, um modelo da unidade intensiva geral e um último que envolve as admissões que aconteceram na unidade geral.

A Figura 5-18 apresenta o modelo referente às unidades que envolvem problemas cardíacos. São consideradas 2 unidades, CCU que é a unidade de cuidados intensivos para problemas cardíacos e CSRU que é a unidade de recuperação de cirurgia cardíaca. Numa primeira análise visual, pode-se perceber que a unidade CCU segue o modelo dos serviços de urgências anteriormente descrito (Figura 5-16). Já no caso do CSRU parece haver uma dependência anormal da unidade de cuidados geral, 'transfer-GCU'.

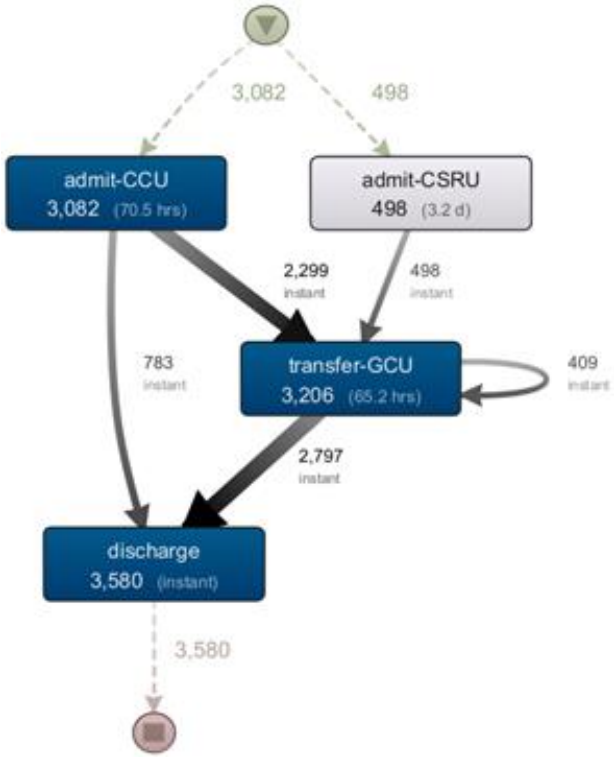


Figura 5-18: Modelo real das unidades CCU e CSRU.

O modelo representado na Figura 5-19 apresenta os fluxos das unidades relativas a cirurgias, sendo SICU uma unidade de terapia intensiva cirúrgica e TSICU uma mesma unidade, mas para pacientes pós-traumáticos. Aparentemente ambas seguem o modelo do serviço de urgências (Figura 5-16).

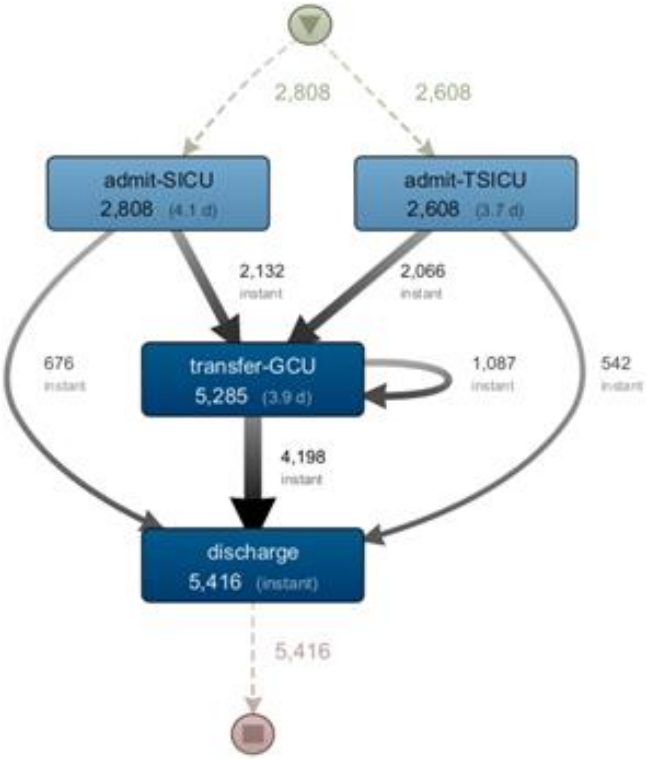


Figura 5-19: Modelo real das unidades SICU e TSICU.

Uma outra área de grande importância nos serviços de saúde é a respeitante aos recém-nascidos, um universo que requer cuidados especiais, devido à grande fragilidade e vulnerabilidade. Pela Figura 5-20 pode-se analisar o funcionamento e relacionamento entre 2 unidades, NICU que é a unidade de terapia intensiva neonatal, e a NWARD que consiste numa enfermaria neonatal.
Uma primeira análise interessante que se pode fazer do modelo é que esta é uma área muito específica e encontra-se à parte de todas as outras unidades. A unidade NWARD poderá corresponder à unidade de repouso geral para adultos, GCU. Já em comparação ao modelo do serviço de urgências (Figura 5-16) percebe-se que visualmente não existe uma semelhança direta.

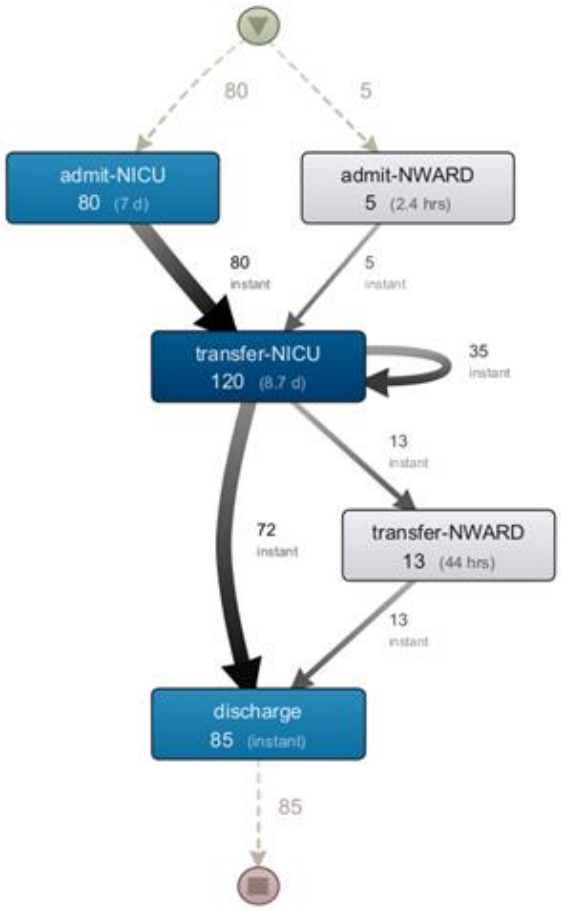


Figura 5-20: Modelo real das unidades NICU e NWARD.

Relativamente a casos gerais e sem uma especialidade relacionada, a Figura 5-21 apresenta o modelo da unidade de terapia intensiva médica, MICU. Esta é uma unidade complexa, dotada dos melhores sistemas de monitorização contínua, que admite pacientes potencialmente graves ou com descompensação de um ou mais sistemas orgânicos. Visualmente, este modelo também segue o previsto da Figura 5-16. Mas numa análise mais detalhada percebe-se um elevado número de ocorrências de um *loop* na etapa GCU, o que mostra um elevado número de trocas dentro da própria unidade. Esta situação, também visível em modelos anteriores, denota trocas de cama na unidade. Este acontecimento pode estar a expor um mau planeamento de camas, algo a ser revisto com os profissionais desta área.

Percebem-se ainda vários *loops* entre a unidade geral e a unidade intensiva, talvez pela sensibilidade que os casos deste tipo requerem, estando sujeitos a avanços e recuos na saúde do paciente. De salientar ainda que esta foi a unidade que registou um maior número de admissões.

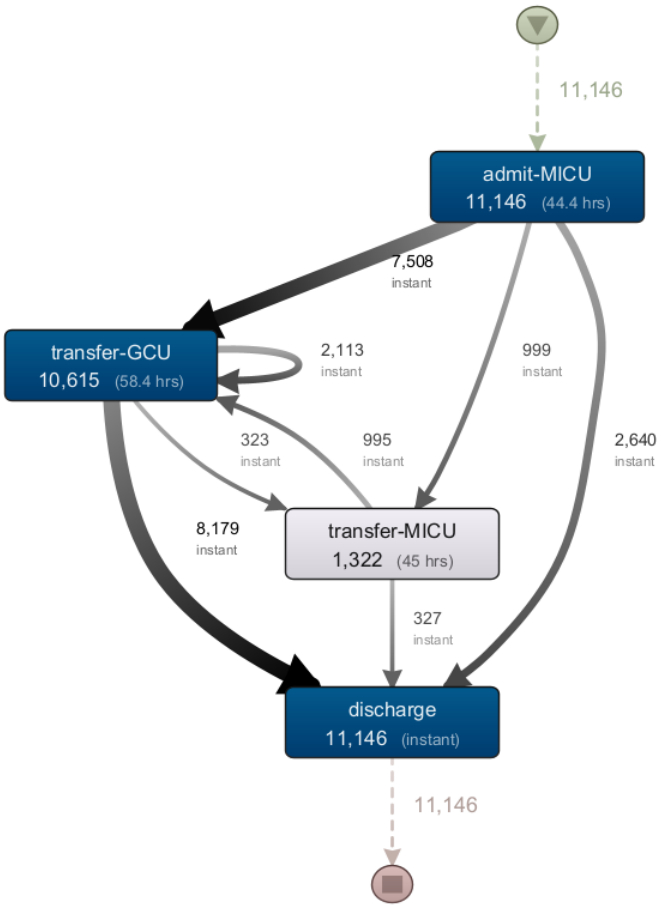


Figura 5-21: Modelo real da unidade MICU.

Finalmente, quanto a admissões na unidade geral, pelo modelo mostrado na Figura 5-22, percebe-se que os pacientes que são admitidos pela unidade geral são, na sua maioria, reencaminhados para uma especialidade. As exceções são casos, possivelmente mais complexos, que são reencaminhados para a unidade de terapia intensiva, MICU, e casos menos graves que, depois de passarem pela unidade geral, 'transfer-GCU', têm alta.

Semelhante à análise feita à unidade de cuidados intensivos, neste modelo percebem-se situações onde o paciente passa para a unidade geral, mas que depois tem de voltar à especialidade. Ainda, de notar que a unidade de recuperação de cirurgia cardíaca, CSRU, apresenta algumas trocas de cama na própria unidade. Numa analise mais cuidadosa percebe-se ainda que nenhum dos casos admitidos na unidade geral e que passaram para especialidade tiveram alta direta, passando sempre pela unidade de cuidados gerais.

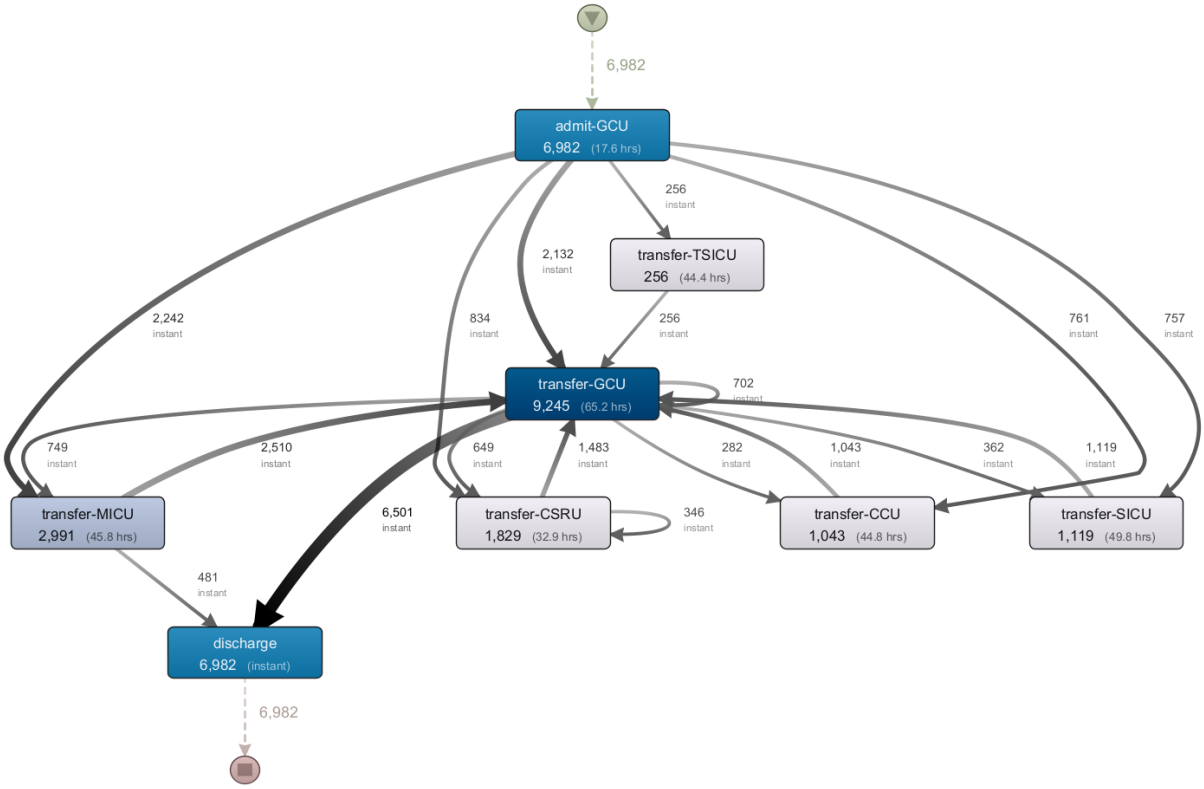


Figura 5-22: Modelo real da unidade GCU.

Ainda relativamente ao modelo anterior, existe uma análise de elevada importância que deve ser feita e que está diretamente relacionada com a qualidade do atendimento. Deve-se calcular a percentagem de admissões feitas pela unidade geral que são, posteriormente, encaminhadas para especialidade e a percentagem de admissões feitas diretamente pela especialidade. Este resultado pode ser analisado no Quadro 5-2. Salienta-se a unidade CSRU, onde a maioria dos casos entra pela unidade geral. Este facto pode estar a evidenciar uma má gestão de admissões nesta unidade.

Quadro 5-2: Análise das admissões pela unidade geral.

Unidade de cuidados	Nº admissões diretamente pela especialidade	Nº admissões pela unidade geral	% de casos admitidos pela unidade geral
CCU (Unidade Coronariana)	3.082	761	20%
CSRU (Unidade de recuperação de cirurgia cardíaca)	498	834	63%
MICU (Unidade de terapia intensiva médica)	11.146	2.242	17%
SICU (Unidade de terapia intensiva cirúrgica)	2.808	757	21%
TSICU (Trauma / Unidade de terapia intensiva cirúrgica)	2.608	256	9%

5.5 Verificação de Conformidade

Nesta secção vão ser apresentados os resultados obtidos na verificação de conformidade, utilizando o módulo *"Replay a Log on Petri Net for Conformance Analysis"* da ferramenta *ProM*. Este módulo recebe o modelo previsto do processo, numa rede de *Petri*, e o ficheiro dos *logs* reais onde, por meio de alinhamentos, apresenta as dispersões entre ambos.
Nesta etapa achou-se mais benéfico separar cada unidade de cuidados para que a analise seja a melhor possível, visto que por vezes as admissões da mesma área partilham partes dos fluxos. É uma prática comum, quando o modelo a analisar é demasiado grande, que este seja dividido em partes e verificado em separado.
A partir do que foi apresentado na secção do modelo previsto (Figura 5-16), identificaram-se os fluxos esperados para cada unidade. Para as admissões pela unidade geral, visto que na maioria das vezes o paciente acaba por ser reencaminhado para uma especialidade, não se achou benéfico fazer a verificação de conformidade para esses casos.
Para a criação dos modelos previstos começou-se por criar um ficheiro com os *logs* previstos. A criação do ficheiro com os *logs* previstos foi feita no *Python*, onde foi criado um *script* que recebe uma lista de objetos compostos pelo identificador do caso e a lista das etapas do caso. O código *Python* necessário para este procedimento pode ser analisado no Anexo 8.
O *Python* utiliza estruturas de árvore para lidar com dados *XML* (*eXtensible Markup Language*). *XML* é um formato de dados hierárquico e a maneira mais natural de o representar é recorrendo a uma árvore. No *Python*, o modulo *"xml.etree.ElementTree"* tem duas classes para este propósito: *ElementTree*, que representa todo o documento *XML* como uma árvore, e *Element*, que representa um único nó nesta árvore. As interações com todo o documento (leitura e gravação de/para arquivos) são, geralmente, feitas no *ElementTree*. As interações com um único elemento *XML* e seus subelementos são feitas no *Element*.
Assim, começou-se por criar a estrutura para receber os *logs*, em formato *XES*. *XES* é um formato baseado em *XML*, utilizado na mineração de processos para padronizar o formato dos dados para *logs* de eventos. A Figura 5-23 descreve a etapa onde é criado o documento *XML*. Nesta etapa, para além da definição da versão do *XES* e nome do ficheiro, no objeto *extension*, é possível perceber que evento terá o campo *'concept:name'* que corresponde ao nome do evento.

```
# Creates a tree struct with a log format
import xml.etree.ElementTree as ET
log = ET.Element('log')
log.set('xes.version', '1.0')
extension = ET.SubElement(log, 'extension')
extension.set('name', 'Concept')
extension.set('prefix', 'concept')
extension.set('uri', 'http://www.xes-standard.org/concept.xesext')
classifier = ET.SubElement(log, 'classifier')
classifier.set('name', 'Event Name')
classifier.set('keys', 'concept:name')
log_string = ET.SubElement(log, 'string')
log_string.set('key', 'concept:name')
log_string.set('value', logs_name+'.xes')
```

Figura 5-23: Criação da estrutura em árvore para suportar os *logs*.

Seguidamente, a lista de *logs* fornecida é percorrida e cada *log* é convertido em elementos traço da estrutura em árvore. No nome do traço foi atribuído o identificador do caso e a lista de eventos desse caso foi convertida em elementos do tipo evento, Figura 5-24.

```
logs = [('1', ['admit-MICU', 'discharge']), ('2', ['admit-MICU', 'transfer-NICU', 'discharge'])]
# Convert logs to xes format
for trace_name, events in logs:
    trace = ET.SubElement(log, 'trace')
    trace_string = ET.SubElement(trace, 'string')
    trace_string.set('key', 'concept:name')
    trace_string.set('value', trace_name)
    for event_name in events:
        event = ET.SubElement(trace, 'event')
        event = ET.SubElement(event, 'string')
        event.set('key', 'concept:name')
        event.set('value', event_name)
```

Figura 5-24: Conversão dos *logs* fornecidos em elementos traço da estrutura em árvore.

Por fim, a estrutura em árvore é guardada num ficheiro com extensão *XES*, Figura 5-25. Um exemplo de resultado deste procedimento pode ser visto no Anexo 9.

```
# Saves a xes structure in a file with a .xes extension
from xml.dom import minidom
tree_out = minidom.parseString(ET.tostring(log)).toprettyxml(indent="\t", encoding='utf-8')
with open(path_file+file_name, 'wb') as f:
    f.write(tree_out)
```

Figura 5-25: – Exportação da estrutura em árvore para um ficheiro com extensão *XES*.

Este ficheiro foi importado no *ProM* e convertido para rede de *Petri* a partir do algoritmo de descoberta *Inductive Miner*. De seguida, os *logs* reais do processo foram exportados do *Disco* e importados também no *ProM*. Como o *Disco* apresenta o ciclo de vida de cada etapa duplicando-a no estado *'start'* e *'complete'*, o ficheiro de *logs* reais foi filtrado no módulo "*Filter events"* do *ProM* de forma a remover os estados *'start'* e assim remover essas duplicações. Estes *logs* reais correspondem aos *logs* utilizados para a descoberta do modelo real feita na secção anterior.

Com o modelo previsto e o ficheiro de *logs* reais, executou-se a verificação de conformidade onde o resultado dos alinhamentos é disponibilizado de várias formas. A partir desse resultado percebem-se os locais ou etapas do modelo previsto que estão, ou não, de acordo com os *logs* reais do processo. De salientar que, para cada alinhamento a seguir apresentado, será identificado o *log* real em questão.

A Figura 5-26 apresenta os modelos resultantes da verificação de conformidade para as unidades que envolvem problemas cardíacos. Analisando os modelos verifica-se que todas as transições estão a verde, o que mostra que os *logs* reais utilizaram todas as transições esperadas. As transições ocultas aparecem na barra no fundo a lilás, o que significa que alguns *logs* passaram por ela, mas não a utilizaram. Nas admissões CSRU a barra a cinza mostra que nenhum *log* passou por ela. Tal como tinha sido visualizado no modelo da descoberta, os pacientes desta unidade passam sempre pela unidade geral. Já no modelo da unidade CCU, o círculo amarelo mostra uma dispersão que se deve ter em atenção.

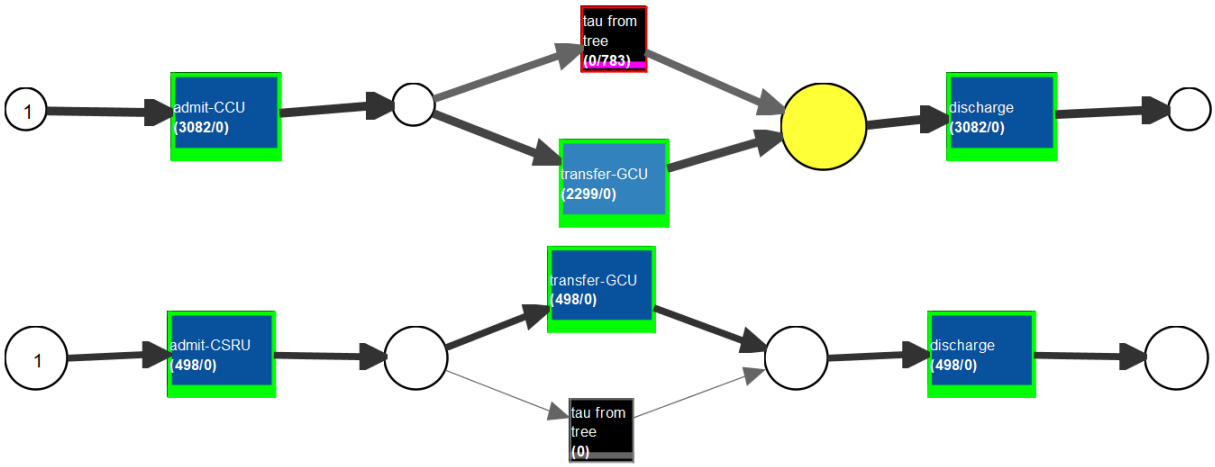


Figura 5-26: Modelos da verificação de conformidade para as unidades CCU e CSRU.

Para perceber efetivamente o que falhou, recorreu-se aos resultados dos alinhamentos, Figura 5-27, onde se percebe que um *log* com admissão na CCU apresenta uma sinalização a amarelo. Esta sinalização indica que a etapa da unidade de cuidados gerais estava duplicada, o que não é esperado. Esta duplicação mostra que o paciente mudou de cama dentro da própria unidade. Em resumo, pela análise de conformidade pode-se verificar que, em geral, a unidade CCU segue o modelo dos serviços de urgências anteriormente descrito (Figura 5-16). Apesar disso, a troca de cama na unidade geral poderá estar a expor uma má gestão deste recurso nessa unidade. No caso do CSRU, os pacientes acabam por passar sempre pela unidade de cuidados geral. Esta questão é algo a ser analisado com os profissionais, mas poderá ser explicada pelo facto de esta ser uma unidade de recuperação onde, nas primeiras 48 horas após o procedimento, o paciente deve ser monitorizado com maior cuidado, já que existe maior probabilidade de ocorrer uma situação de emergência. Depois deste tempo, o paciente passa ainda para a unidade geral para um recobro com menor risco, antes de ter alta.

Figura 5-27: Alinhamentos para as unidades CCU e CSRU.

Seguindo para a próxima análise, os modelos da verificação de conformidade representados na Figura 5-28 apresentam os fluxos das unidades relativas a cirurgias. Nestas unidades, a análise é simples pois todas as etapas estão a verde e as transições ocultas com a barra no fundo a lilás, mostram que o modelo real está em total conformidade com o previsto. Apenas é necessário verificar o que aconteceu nas situações onde os modelos apresentam círculos a amarelo.

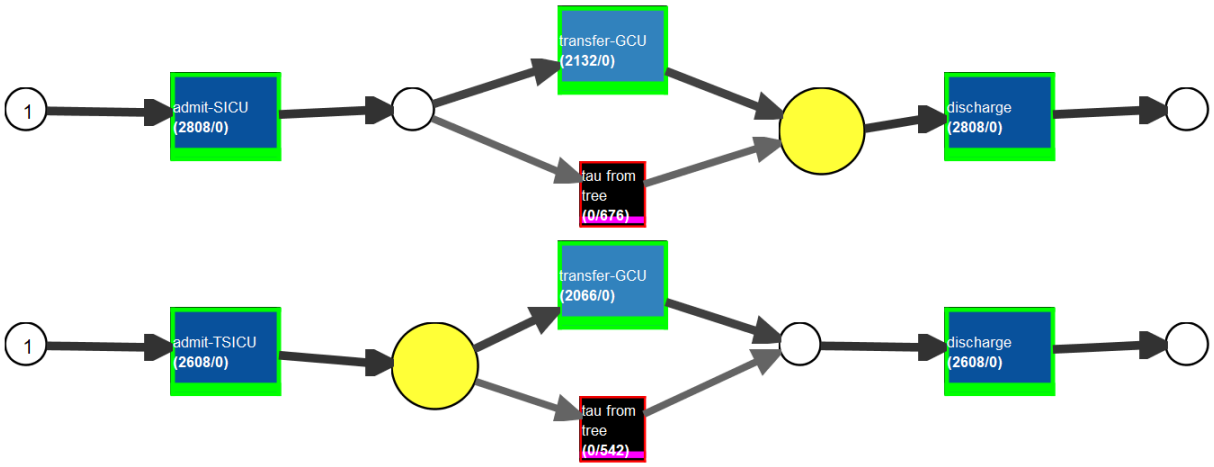


Figura 5-28: Modelos da verificação de conformidade para as unidades SICU e TSICU.

Nos alinhamentos da Figura 5-29, verifica-se que todas as etapas estão a verde representam a sincronia entre os *logs* reais e o modelo previsto. A exceção são as etapas sinalizadas a amarelo que mostram trocas de cama dentro da unidade geral, 'transfer-GCU'. Trata-se de algo que já tinha sido verificado noutros modelos.

Figura 5-29: Alinhamentos para as unidades SICU e TSICU.

O próximo modelo da verificação de conformidade analisado é respeitante aos recém-nascidos, Figura 5-30. Tal como nas unidades para adultos, é expectável que os recém-nascidos passem pela enfermaria geral antes de terem alta, se precisarem de ficar em observação. Pelos modelos, podem-se observar os círculos a amarelo que mostram que alguns casos apresentam dispersões logo após a fase de admissão. Quanto às admissões NWARD, pela barra cinza na transição oculta, verifica-se que estas passam sempre pela enfermaria geral, 'transfer-NWARD'.

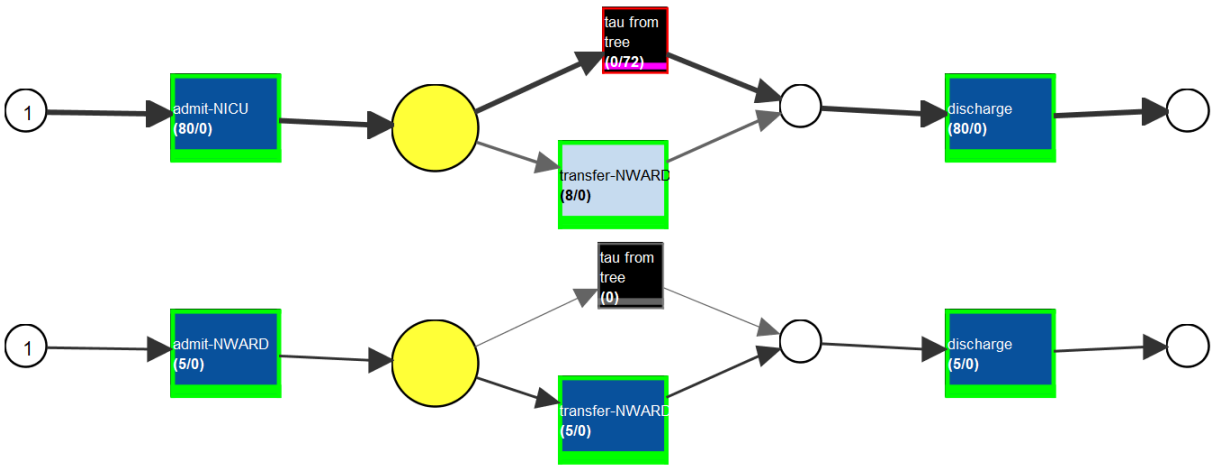


Figura 5-30: Modelos da verificação de conformidade para as unidades NICU e NWARD.

Analisando em detalhe as situações de divergência anteriormente identificadas, pelos alinhamentos, Figura 5-31, pode-se chegar a uma conclusão importante. Pelas etapas marcadas a amarelo para a etapa 'transfer-NICU', percebe-se que os recém-nascidos, independentemente da unidade em que são admitidos, passam sempre pela unidade de terapia intensiva. Esta situação é algo que deve ser analisado junto dos profissionais de saúde, pois gera um aumento de ocupação nessa unidade, agravado pelo facto de existirem, ainda, várias trocas de cama nessa unidade. Uma explicação simples desta dependência da unidade de terapia intensiva é o facto de serem pacientes que requerem uma atenção acima do normal, pois estão ainda numa fase muito precoce e frágil.

Figura 5-31: Alinhamentos para as unidades NICU e NWARD.

Por fim, relativamente a casos gerais e sem uma especialidade relacionada, a Figura 5-32 apresenta o modelo da verificação de conformidade para a unidade de terapia intensiva médica. Como esperado, a unidade de cuidados gerais é utilizada em alguns casos, mas ainda assim pode-se perceber dois locais de falha, nos círculos a amarelo.

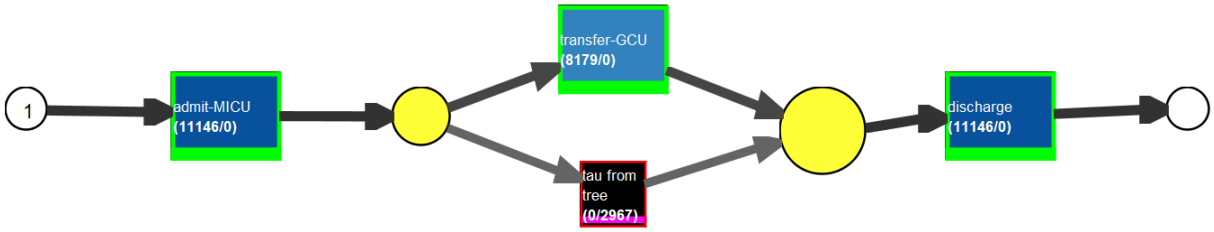


Figura 5-32: Modelo da verificação de conformidade para a unidade MICU.

Assim, analisando as etapas a amarelo realçadas na Figura 5-33, tem-se as falhas que merecem maior atenção. Algumas delas são relativas às duplicações já analisadas anteriormente, nomeadamente na unidade geral, pelas trocas de cama. As restantes falhas mostram algo semelhante: em alguns casos, o paciente apresenta transferências dentro da unidade de cuidados intensivos, 'transfer-MICU'. Esta situação deve ser analisada com os profissionais das unidades para perceber a que se devem estas transferências. Se também forem trocas de cama é necessário avaliar se esta situação é necessária ou de que forma pode ser evitada.

Figura 5-33: Alinhamentos para a unidade MICU.

5.6 Conclusões da análise do processo

Os resultados alcançados, recorrendo às ferramentas *Disco* e *ProM*, e às configurações escolhidas mostraram-se muito eficientes. Foi possível alcançar um modelo que permitiu analisar o funcionamento real de cada unidade de cuidados. É, assim, possível tirar conclusões em relação à sua gestão, assim como dos seus recursos:

- A maioria das unidades de cuidados segue o padrão do serviço de urgências, o que é um bom indicador de gestão e planeamento;
- A unidade que regista mais entradas é a unidade de cuidados intensivos gerais. Nesta unidade registam-se possíveis trocas de cama, o que pode sinalizar um mau planeamento deste recurso. O mesmo acontece na unidade de cuidados gerais e na unidade de terapia intensiva neonatal;
- Os recém-nascidos, independentemente da unidade em que são admitidos, passam sempre pela unidade de terapia intensiva, algo a ser analisado com os profissionais desta área, pois cria um natural aumento de camas utilizadas;
- Os pacientes admitidos pela unidade geral e reencaminhados para uma especialidade acabam sempre por voltar à unidade geral antes de receberem alta. Tal situação pode estar a causar um aumento de ocupação de camas também nesta unidade. Seria de interesse rever com os profissionais da unidade os critérios de alta direta ou observação antes da alta. A mesma situação é encontrada para todos os casos com admissão na unidade de recuperação de cirurgia cardíaca;
- Na unidade de recuperação de cirurgia cardíaca registaram-se mais casos que foram admitidos pela unidade geral do que diretamente pela especialidade, podendo assim haver uma má gestão de admissões nesta unidade.

6. Conclusões

Os processos de saúde são complexos e envolvem etapas executadas por pessoas de várias disciplinas e subáreas. Essa complexidade torna esta área interessante, mas difícil de analisar e compreender. Estes processos fazem uso de sistemas de informação que registam grandes volumes de dados e são difíceis de explorar.

Como resposta às dificuldades referidas, a mineração de processos possibilita a extração de conhecimento a partir de dados gerados e armazenados nos sistemas de informação. Para tal, são utilizados os registos de eventos que fornecem carimbos de data e hora das etapas, a designação das próprias etapas, bem como informações adicionais dos recursos utilizados.

Este trabalho pretendeu mostrar a viabilidade e os benefícios da utilização do *Process Mining* na análise, gestão e possível melhoria de processos de saúde, neste caso num serviço de urgências. Como tal, este trabalho assumiu um caracter exploratório na área de mineração de dados, mais precisamente na mineração de processos.

Numa primeira fase, foi estudado o tema *Process Mining* e as suas aplicações na área da saúde, de forma a ter uma fundamentação teórica da aplicação de *Process Mining* na área da saúde. Desta revisão da literatura percebeu-se que, a partir dos resultados de estudos recentes, a mineração de processos se mostrava como uma metodologia viável para avaliar processos no âmbito hospitalar, permitindo chegar a conclusões que se transformavam em ações corretivas para resolver os problemas encontrados.

Quanto ao *Process Mining*, a descoberta do processo permite descobrir o modelo real do processo a partir dos registos de eventos. E, pela verificação da conformidade, é possível perceber de que forma o modelo descoberto se aproxima do modelo previsto. Foram, ainda, estudados os algoritmos e ferramentas de *Process Mining*. Devido ao carácter académico deste trabalho, escolheu-se testar o *framework PM4Py* e as ferramentas *ProM* e *Disco*, bem como os algoritmos disponíveis. Estas ferramentas disponibilizam licenças gratuitas ou académicas.

De seguida, foi feita uma análise exploratória dos dados que se definiram tratar neste trabalho, com o intuito de os conhecer melhor e de os preparar para as etapas seguintes. A base de dados

utilizada foi a *MIMIC-III*. Esta é uma grande base de dados disponível gratuitamente que compreende dados não identificados relativos a saúde, associados a mais de quarenta mil pacientes que permaneceram em unidades de cuidados intensivos.

Para a realização do estudo exploratório e comparativo das ferramentas de *Process Mining* foram utilizados os dados da versão demo do MIMIC-III. Nessa fase, foram criados cenários de testes com desafios de dados reais para analisar de que forma os algoritmos disponíveis lidavam com eles. Num cenário inicial, foram escolhidas admissões simples, com o objetivo de estabelecer uma primeira interação com o algoritmo e respetivo modelo. De seguida foi escolhido um cenário com etapas duplicadas, onde uma etapa ocorre de forma repetida. E foi testada a situação que expõe os algoritmos a *loops* entre etapas.

O estudo exploratório e comparativo de algoritmos e de ferramentas de *Process Mining* teve o propósito de entender as suas características, limitações e vantagens. As ferramentas selecionadas na fase de pesquisa, *ProM*, *Disco* e o *framework PM4Py*, foram, ainda, testadas e comparadas para as etapas de importação do conjunto de *logs*, descoberta de processos, análise de variantes, filtragem de *logs*, verificação de conformidade e disponibilização de estatísticas sobre os *logs*. Note-se que, na etapa de descoberta do processo, foram testados e comparados todos os algoritmos disponíveis em cada ferramenta.

A partir dos resultados dos testes feitos aos algoritmos disponíveis nas ferramentas, sintetizaram-se conclusões de forma a ter uma classificação sistemática e comparativa. Em resumo, concluiu-se que o *Alpha Miner*, em todas as versões disponíveis, é o único algoritmo que não se mostrou apto para lidar com etapas duplicadas e *loops* entre duas etapas. O *Directly-Follows Graph*, por sua vez, para um conjunto de *logs* maior, gerou um modelo inválido, não conseguindo representar casos com mais de 5 etapas. Quanto aos restantes algoritmos, mostraram-se aptos para os desafios, com uma enorme desvantagem para o *Fuzzy Miner* implementado no *ProM*, que apresentou um modelo incoerente para *loops* entre 2 etapas.

Os algoritmos *Heuristic Miner*, *Inductive Miner* e a versão do *Fuzzy Miner* de *Christian W. Günther* foram os que se mostraram aptos a lidar com mais desafios e maiores volumes de *logs*. Salienta-se que, a nível de simplicidade do modelo, o *Inductive Miner* apresentou modelos mais simples de analisar que o *Heuristic Miner*, isto porque este faz uma simplificação do modelo a partir duma pesquisa recursiva de padrões entre *logs*. O *Fuzzy Miner* de *Christian W. Günther* implementado no *Disco*, permite uma descoberta e configuração do modelo bastante simples, tirando partido duma ferramenta muito intuitiva, que apresenta um modelo fácil de perceber, destacando atividades e caminhos frequentes, bem como análises por frequência e/ou desempenho.

A nível das ferramentas, pode-se concluir que o *Disco* é uma ferramenta extremamente simples de utilizar para as funcionalidades que disponibiliza. Apenas tem o aspeto negativo de não incluir funcionalidades explicitas para verificação de conformidade, uma funcionalidade em que o *ProM* se mostrou muito eficaz. Destaca-se o facto de ambas as ferramentas, *Disco* e *ProM*, se mostrarem limitadas a nível de personalização. No sentido contrário, o *PM4Py* mostrou-se bastante completo, com uma possibilidade grande de personalização, permitindo a utilização de todas as funcionalidades apresentadas, sem necessidade de muito código ou conhecimento.

Pelo exposto, o *Disco* é a ferramenta mais simples, intuitiva e que produz melhores resultados para a descoberta do processo. Já para a verificação de conformidade, o módulo utilizado no *ProM* disponibiliza os resultados de diferentes formas, fáceis de compreender e explicar. Assim, o cenário que se mostrou mais eficaz para utilizar no caso de estudo final foi a utilização do *Disco* para a descoberta do processo e do *ProM* para a verificação de conformidade.

Para o caso de estudo final, foi feita a análise e exploração de um processo de saúde a partir dos dados da versão 3 completa do *MIMIC*. Apesar de serem dados da mesma fonte, pela experiência adquirida na fase de testes, a escolha dos dados finais sofreu alterações para que a análise fosse a mais eficaz possível.

A aplicação de *Process Mining* a um caso de estudo tinha como objetivo demonstrar como esta técnica permite a identificação de eventuais problemas, gargalos ou constrangimentos nos processos de cuidados de saúde. A informação recolhida nessa análise permite fornecer informações de modo a agir sobre as ineficiências existentes, e projetar intervenções como diminuir os tempos de espera, reduzir o congestionamento do paciente, aumentar a qualidade do atendimento e, eventualmente, reduzir custos.

Uma vez que foi decidido englobar todas as unidades de cuidados e o modelo não foi simplificado para não se perderem etapas e ligações que poderiam ser de interesse, o resultado foi um modelo grande e difícil de analisar como um todo. Assim, para uma melhor análise, o modelo foi separado pelas admissões às diferentes unidades de cuidados. Foram construídos 5 modelos, um modelo referente às unidades que envolvem problemas cardíacos, um modelo relativo às unidades de cirurgia, um modelo respeitante às unidades neonatal, um modelo da unidade intensiva geral e um último que envolve as admissões que aconteceram na unidade geral. Com os modelos das diversas áreas de atuação separados, os resultados foram muito mais simples de analisar.

Utilizando o módulo *"Replay a Log on Petri Net for Conformance Analysis"* da ferramenta *ProM* para verificação de conformidade foi possível identificar de forma clara os pontos de divergências entre o modelo previsto e os registos de eventos reais. Este módulo recebe o modelo previsto do processo, numa rede de *Petri*, e o ficheiro dos *logs* reais do processo. O resultado é apresentado na forma de gráfico ou de modelo com as respetivas identificações dos locais onde o modelo está ou não de acordo com os *logs* reais do processo.

Os resultados obtidos permitiram tirar conclusões em relação à gestão das diferentes unidades de cuidados, assim como dos seus recursos. Percebeu-se que a maioria das unidades de cuidados segue o padrão do serviço de urgências. Mas, na unidade de cuidados intensivos gerais registou-se um elevado número de trocas de cama, o que pode sinalizar um mau planeamento deste recurso nessa unidade. O mesmo acontece na unidade de cuidados gerais e na unidade de terapia intensiva neonatal. Já na unidade de recuperação de cirurgia cardíaca registaram-se mais casos que foram admitidos pela unidade geral do que diretamente pela especialidade, podendo assim haver uma má gestão de admissões nessa unidade.

Existem outras situações em que não se pode concluir que existam problemas claros, mas que devem ser analisadas junto dos profissionais das respetivas áreas. Nas unidades dos recém-nascidos, independentemente da unidade em que são admitidos, os recém-nascidos passam sempre pela unidade de terapia intensiva, podendo criar um natural aumento de camas utilizadas

dessa unidade. A mesma situação é encontrada para todos os casos com admissão na unidade de recuperação de cirurgia cardíaca. Outra situação diz respeito à unidade geral, onde os pacientes admitidos nesta e reencaminhados para uma especialidade acabam sempre por voltar à unidade antes de receberem alta. Essa situação pode estar a causar um aumento de ocupação de camas nessa unidade, onde se poderia sugerir aos responsáveis da unidade a revisão dos critérios de alta direta ou observação antes da alta. A mesma situação é encontrada para todos os casos com admissão na unidade de recuperação de cirurgia cardíaca.

Neste trabalho foi possível perceber a dificuldade de trabalhar com dados reais relacionados com seres humanos, pela segurança e proteção de dados requeridos pelas respetivas fontes. No entanto, percebe-se que os dados dos sistemas de informação podem ser seguramente utilizados nas técnicas aqui propostas e retirar inúmeras vantagens na gestão de processos de saúde / hospitalares. Assim, para trabalho futuro seria importante validar as técnicas utilizadas em ambientes em tempo real.

Todo o trabalho realizado resultou na escrita de diversos artigos relacionados com a aprendizagem e resultados obtidos. Portanto, este estudo apresenta-se como uma demonstração do potencial da utilização do *Process Mining* na melhoria de processos de saúde, de forma a estimular futuros trabalhos desta área, com a aplicação destas práticas em dados reais.

REFERÊNCIAS

Adriansyah, A., Sidorova, N., & Van Dongen, B. F. (2011). Cost-based fitness in conformance checking. *Proceedings - International Conference on Application of Concurrency to System Design, ACSD, January 2014*, 57–66. https://doi.org/10.1109/ACSD.2011.19

Almeida, S. V. de, Costa, E., Lopes, F. V., Santos, J. V., & Barros, P. P. (2020). Concerns and adjustments: How the Portuguese population met COVID-19. *PLoS ONE*, *15*(10 October), 1–17. https://doi.org/10.1371/journal.pone.0240500

Alvarez, C., Rojas, E., Arias, M., Munoz-Gama, J., Sepúlveda, M., Herskovic, V., & Capurro, D. (2018). Discovering role interaction models in the Emergency Room using Process Mining. *Journal of Biomedical Informatics*, *78*, 60–77. https://doi.org/10.1016/j.jbi.2017.12.015

Azadeh-Fard, N., Megahed, F. M., & Pakdil, F. (2019). Variations of length of stay: A case study using control charts in the CRISP-DM framework. *International Journal of Six Sigma and Competitive Advantage*, *11*(2–3), 204–225. https://doi.org/10.1504/IJSSCA.2019.101418

Azeredo, T. R. M., Guedes, H. M., Rebelo de Almeida, R. A., Chianca, T. C. M., & Martins, J. C. A. (2015). Efficacy of the manchester triage system: A systematic review. *International Emergency Nursing*, *23*(2), 47–52. https://doi.org/10.1016/j.ienj.2014.06.001

Badakhshan, P., Geyer-Klingeberg, J., El-Halaby, M., Lutzeyer, T., & Affonseca, G. V. L. (2020). Celonis process repository: A bridge between business process management and process mining. *CEUR Workshop Proceedings*, *2673*, 67–71.

Baker, K., Dunwoodie, E., Jones, R. G., Newsham, A., Johnson, O., Price, C. P., Wolstenholme, J., Leal, J., McGinley, P., Twelves, C., & Hall, G. (2017). Process mining routinely collected electronic health records to define real-life clinical pathways during chemotherapy. *International Journal of Medical Informatics*, *103*, 32–41. https://doi.org/10.1016/j.ijmedinf.2017.03.011

Bárrios, M. J., Marques, R., & Fernandes, A. A. (2020). Aging with health: aging in place strategies of a Portuguese population aged 65 years or older. *Revista de Saude Publica*, *54*, 129. https://doi.org/10.11606/s1518-8787.2020054001942

Barros, R., Peres, A., Lorenzi, F., Wives, L. K., & Jaccottet, E. H. da S. (2011). Case law analysis with machine learning in brazilian court. *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, December 2019*. https://doi.org/10.1007/978-3-642-03503-6

Batista, E., & Solanas, A. (2019). Process mining in healthcare: A systematic review. *2018 9th International Conference on Information, Intelligence, Systems and Applications, IISA 2018*, 1–6. https://doi.org/10.1109/IISA.2018.8633608

Berti, A., & Van der Aalst, W. (2020). A novel token-based replay technique to speed up conformance checking and process enhancement. *ArXiv*. https://doi.org/10.1007/978-3-662-63079-2_1

Berti, A., Van Zelst, S. J., Van der Aalst, W., & Gesellschaf, F. (2019). Process mining for python (PM4py): Bridging the gap between process- And data science. *CEUR Workshop Proceedings*, *2374*, 13–16.

Bloemen, V., Zelst, S. Van, & Van der Aalst, W. (2019). *Aligning Observed and Modelled Behaviour by Maximizing Synchronous Moves and Using Milestones*.

Bogarín, A., Cerezo, R., & Romero, C. (2018). Discovering learning processes using inductive miner: A case study with learning management systems (LMSs). *Psicothema*, *30*(3), 322–329. https://doi.org/10.7334/psicothema2018.116

Bolt, A., Aalst, W. M. P. Van Der, Leoni, M. De, Van der Aalst, W., & de Leoni, M. (2017). Finding process variants in event Logs. *On the Move to Meaningful Internet Systems. OTM 2017 Conferences*, 45–52.

Breitmayer, M. (2018). *Applying Process Mining Algorithms in the Context of Data Collection Scenarios*.

Bunker, R. P., & Thabtah, F. (2019). A machine learning framework for sport result prediction. *Applied Computing and Informatics*, *15*(1), 27–33. https://doi.org/10.1016/j.aci.2017.09.005

Castro R., L. F., Espitia P., E., & Montilla, A. F. (2018). Applying CRISP-DM in a KDD process for the analysis of student attrition. *Communications in Computer and Information Science*, *885*, 386–401. https://doi.org/10.1007/978-3-319-98998-3_30

Catley, C., & James, A. (2011). *A Process Mining Driven Framework for Clinical Guideline Improvement in Critical Care*.

Cho, M., Song, M., Park, J., Yeom, S. R., Wang, I. J., & Choi, B. K. (2020). Process mining-supported emergency room process performance indicators. *International Journal of Environmental Research and Public Health*, *17*(17), 1–20. https://doi.org/10.3390/ijerph17176290

Conca, T., Saint-Pierre, C., Herskovic, V., Sepúlveda, M., Capurro, D., Prieto, F., & Fernandez-Llatas, C. (2018). Multidisciplinary collaboration in the treatment of patients with type 2 diabetes in primary care: Analysis using process mining. *Journal of Medical Internet Research*, *20*(4). https://doi.org/10.2196/jmir.8884

Conforti, R., La Rosa, M., & ter Hofstede, A. H. M. (2015). Noise Filtering of Process Execution Logs based on Outliers Detection. *Institute for Future Environments; School of Information Systems; Science & Engineering Faculty*, 1–16.

Daderman, A., & Rosander, S. (2018). Evaluating Frameworks for Implementing Machine Learning in Signal Processing. *Examensarbete Inom Teknik*, 1–36.

Desel, J., & Juhás, G. (2001). "What Is a Petri Net?" Informal Answers for the Informed Reader. *Lecture Notes in Computer Science*, *2128*, 1–25.

Diba, K., Batoulis, K., Weidlich, M., & Weske, M. (2020). Extraction, correlation, and abstraction of event data for process mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, *10*(3), 1–24. https://doi.org/10.1002/widm.1346

Fernández-Llatas, C., Benedi, J. M., García-Gómez, J. M., & Traver, V. (2013). Process mining for individualized behavior modeling using wireless tracking in nursing homes. *Sensors (Switzerland)*, *13*(11), 15434–15451. https://doi.org/10.3390/s131115434

Gatta, R., Vallati, M., Lenkowicz, J., Casa, C., Cellini, F., Damiani, A., & Valentini, V. (2018). A framework for event log generation and knowledge representation for process mining in healthcare. *Proceedings - International Conference on Tools with Artificial Intelligence, ICTAI*, *2018-Novem*, 647–654. https://doi.org/10.1109/ICTAI.2018.00103

Goldberger, A. L., Amaral, L. A., Glass, L., Hausdorff, J. M., Ivanov, P. C., Mark, R. G., Mietus, J. E., Moody, G. B., Peng, C. K., & Stanley, H. E. (2000). PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals. *Circulation*, *101*(23). https://doi.org/10.1161/01.cir.101.23.e215

Gomes, A., Lacerda, A., & Fialho, J. (2021a). Application of Process Mining in an Emergency Service. *Accepted for Publishing in IBIMA*.

Gomes, A., Lacerda, A., & Fialho, J. (2021b). Comparative Analysis of Process Mining Algorithms in Python. *Accepted for Publishing in EAI GoodTechs*.

Gomes, A., Lacerda, A., & Fialho, J. (2021c). Comparative Analysis of Process Mining Tools. *Accepted for Publishing in CAPSI*.

Gomes, A., Lacerda, A., & Fialho, J. (2021d). Comparative Analysis of Process Mining Algorithms in Process Discover. *Accepted for Publishing in DiTTEt*.

Gomez, S. B., Gomez, M. C., & Quintero, J. B. (2019). *Business Intelligence Applied to Ecotourism in Colombia*. *June*, 1–6. https://doi.org/10.23919/cisti.2019.8760802

Günther, C. (2012). *Disco Discover Your Processes*.

Günther, C., Rozinat, a, van der Aalst, W., & van Uden, K. (2008). Monitoring deployed application usage with process mining. *BPM Center Report*, 1–8. http://tmpmining.win.tue.nl/_media/publications/bpm-08-11.pdf

Helm, E., & Küng, J. (2016). Process mining: Towards comparability of healthcare processes. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9832 LNCS*, 249–252. https://doi.org/10.1007/978-3-319-43949-5_20

Hendricks, R. (2019). Process Mining of Incoming Patients with Sepsis. *Online Journal of Public Health Informatics*, *11*(2). https://doi.org/10.5210/ojphi.v11i2.10151

Huang, Z., Lu, X., Duan, H., & Fan, W. (2013). Summarizing clinical pathways from event logs. *Journal of Biomedical Informatics*, *46*(1), 111–127. https://doi.org/10.1016/j.jbi.2012.10.001

Huber, S., Wiemer, H., Schneider, D., & Ihlenfeldt, S. (2019). DMME: Data mining methodology for engineering applications - A holistic extension to the CRISP-DM model. *Procedia CIRP*, *79*, 403–408. https://doi.org/10.1016/j.procir.2019.02.106

Jaggia, S., Kelly, A., Lertwachara, K., & Chen, L. (2020). Applying the CRISP-DM Framework for Teaching Business Analytics. *Decision Sciences Journal of Innovative Education*, *18*(4), 612–634. https://doi.org/10.1111/dsji.12222

Johnson, A. E. W., Pollard, T. J., Shen, L., Lehman, L. W. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., & Mark, R. G. (2016). MIMIC-III, a freely accessible critical care database. *Scientific Data*, *3*, 1–9. https://doi.org/10.1038/sdata.2016.35

Keller, R. M. (1976). Formal Verification of Parallel Programs. *Communications of the ACM*, *19*(7), 371–384. https://doi.org/10.1145/360248.360251

Kurniati, A. P., & Hall, G. (2020). *Process mining on the extended event log to analyse the system usage during healthcare processes (Case study : the GP Tab usage during chemotherapy treatments) Patient Pathways Manager (PPM) EHR System*. 1–12.

Kurniati, A. P., Hall, G., Hogg, D., & Johnson, O. (2018). Process mining in oncology using the MIMIC-III dataset. *Journal of Physics: Conference Series*, *971*(1). https://doi.org/10.1088/1742-6596/971/1/012008

Kurniati, A. P., Johnson, O., Hogg, D., & Hall, G. (2016). Process mining in oncology: A literature review. *Proceedings of the 6th International Conference on Information Communication and Management, ICICM 2016*, *i*, 291–297. https://doi.org/10.1109/INFOCOMAN.2016.7784260

L'Heureux, A., Grolinger, K., Elyamany, H. F., & Capretz, M. A. M. (2017). Machine Learning with Big Data: Challenges and Approaches. *IEEE Access*, *5*, 7776–7797. https://doi.org/10.1109/ACCESS.2017.2696365

Lang, M., Bürkle, T., Laumann, S., & Prokosch, H. U. (2008). Process mining for clinical workflows: Challenges and current limitations. *Studies in Health Technology and Informatics*, *136*, 229–234.

Lohmann, N. M. (2012). Disco Discover Your Processes. *Prceedings*, *September*.

Ma'arif, M. R. (2017). Revealing daily human activity pattern using process mining approach. *International Conference on Electrical Engineering, Computer Science and Informatics (EECSI), 2017-Decem*(September), 19–21. https://doi.org/10.1109/EECSI.2017.8239160

Mans, R. S., Reijers, H. A., Van Genuchten, M., & Wismeijer, D. (2012). Mining processes in dentistry. *IHI'12 - Proceedings of the 2nd ACM SIGHIT International Health Informatics Symposium*, 379–388. https://doi.org/10.1145/2110363.2110407

Mans, R. S., Van der Aalst, W., & Vanwersch, R. J. B. (2015). *Process Mining in the Healthcare*.

Martinez-Plumed, F., Contreras-Ochando, L., Ferri, C., Hernandez Orallo, J., Kull, M., Lachiche, N., Ramirez Quintana, M. J., & Flach, P. A. (2019). CRISP-DM Twenty Years Later: From Data Mining Processes to Data Science Trajectories. *IEEE Transactions on Knowledge and Data Engineering*, *4347*(c), 1–1. https://doi.org/10.1109/tkde.2019.2962680

Mbassegue, P., Escandon-Quintanilla, M. L., & Gardoni, M. (2016). Knowledge management and big data: Opportunities and challenges for small and medium enterprises (SME). *IFIP Advances in Information and Communication Technology*, *492*, 22–31. https://doi.org/10.1007/978-3-319-54660-5_3

Moerland, P. D. (1996). A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Neurocomputing* (Vol. 12, Issue 4, pp. 371–373). https://doi.org/10.1016/0925-2312(96)00016-1

Munoz-Gama, J., & Carmona, J. (2011). Enhancing precision in Process Conformance: Stability, confidence and severity. *IEEE SSCI 2011: Symposium Series on Computational Intelligence - CIDM 2011: 2011 IEEE Symposium on Computational Intelligence and Data Mining*, 184–191. https://doi.org/10.1109/CIDM.2011.5949451

Muriuki, N. (2015). *Clustering and visualizing the status of child health in Kenya: a data mining approach*. *December*.

Nagashima, H., & Kato, Y. (2019). APREP-DM: A Framework for Automating the Pre-Processing of a Sensor Data Analysis based on CRISP-DM. *2019 IEEE International Conference on Pervasive Computing and Communications Workshops, PerCom Workshops 2019*, 555–560. https://doi.org/10.1109/PERCOMW.2019.8730785

Niakšu, O. (2015). CRISP Data Mining Methodology Extension for Medical Domain. *Baltic J. Modern Computing*, *3*(2), 92–109.

Oliveira, E. (2010). MIC- Metodologias de Investigação Científica : Uma Visão sobre Teorias acerca do desenvolvimento e da caracterização da Investigação Científica. *Feup*, 1–43. http://paginas.fe.up.pt/~eol/PRODEI/mic1011.htm%0A1.

Perimal-Lewis, L., Teubner, D., Hakendorf, P., & Horwood, C. (2016). Application of process mining to assess the data quality of routinely collected time-based performance data sourced from electronic health records by validating process conformance. *Health Informatics Journal*, *22*(4), 1017–1029. https://doi.org/10.1177/1460458215604348

Plotnikova, V., Dumas, M., & Milani, F. (2020). Adaptations of data mining methodologies: A systematic literature review. *PeerJ Computer Science*, *6*, 1–43. https://doi.org/10.7717/PEERJ-CS.267

Pohl, T. (2019). *An Inductive Miner Implementation for the PM4PY Framework*. 1–66.

Rojas, E., Cifuentes, A., Burattin, A., Munoz-Gama, J., Sepúlveda, M., & Capurro, D. (2019). Performance analysis of emergency room episodes through process mining. *International Journal of Environmental Research and Public Health*, *16*(7). https://doi.org/10.3390/ijerph16071274

Rojas, E., Sepúlveda, M., Munoz-Gama, J., Capurro, D., Traver, V., & Fernandez-Llatas, C. (2017). Question-driven methodology for analyzing emergency room processes using process mining. *Applied Sciences (Switzerland)*, *7*(3). https://doi.org/10.3390/APP7030302

Romani, H. M., Sperandio, J. A., Sperandio, J. L., Diniz, M. N., & Inácio, M. A. (2009). Uma visão assistencial da urgência e emergência no sistema de saúde. *Revista Bioética*, *17*(1), 41–53.

Sakellarides, C. (2020). National health service: Responding to current challenges through necessary transformations. *Acta Medica Portuguesa*, *33*(2), 133–142. https://doi.org/10.20344/amp.12626

Saltz, J., Hotz, N., Wild, D., & Stirling, K. (2018). Exploring project management methodologies used within data science teams. In *Americas Conference on Information Systems 2018: Digital Disruption, AMCIS 2018*.

Schäfer, F., Zeiselmair, C., & Becker, J. (2020). Synthesizing CRISP-DM and quality management: A data mining approach for production processes. *2020 IEEE International Conference on Technology Management, Operations and Decisions, ICTMOD 2020*, 190–195.

Schröer, C., Kruse, F., & Gómez, J. M. (2021). A systematic literature review on applying CRISP-DM process model. *Procedia Computer Science*, *181*(2019), 526–534. https://doi.org/10.1016/j.procs.2021.01.199

Shinde, S. A., & Rajeswari, P. R. (2018). Intelligent health risk prediction systems using machine learning: A review. *International Journal of Engineering and Technology(UAE)*, *7*(3), 1019–1023. https://doi.org/10.14419/ijet.v7i3.12654

Sowmya, R., & Suneetha, K. (2017). Data Mining with Big Data. *Proceedings of 2017 11th International Conference on Intelligent Systems and Control, ISCO 2017*, 246–250.

Sundari, M. S., & Nayak, R. K. (2020). Process mining in healthcare systems: A critical review and its future. *International Journal of Emerging Trends in Engineering Research*, *8*(9), 5197–5208. https://doi.org/10.30534/ijeter/2020/50892020

Tumelaire, V. V. (2015). *Development of a repair cost calculation model for DAF Trucks N . V . using the CRISP- DM framework*. *May*.

Vaillant, A. A. J., & Zito, P. M. (2021). Neutropenia. *National Center for Biotechnology, National Library of Medicine*, 1–5.

Van der Aalst, W. (2012). Process mining: Overview and opportunities. *ACM Transactions on Management Information Systems*, *3*(2), 1–17. https://doi.org/10.1145/2229156.2229157

Van der Aalst, W. (2019). A practitioner's guide to process mining: Limitations of the directly-follows graph. *Procedia Computer Science*, *164*, 321–328. https://doi.org/10.1016/j.procs.2019.12.189

Van der Aalst, W., Bolt, A., & van Zelst, S. J. (2017). RapidProm: Mine your processes and not just your data. *ArXiv*, 1–25.

Van der Aalst, W., & Song, M. (2004). Mining Social Networks: Uncovering Interaction Patterns in Business Processes. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *3080*, 244–260. https://doi.org/10.1007/978-3-540-25970-1_16

Van der Aalst, W., Weijters, T., & Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, *16*(9), 1128–1142. https://doi.org/10.1109/TKDE.2004.47

Van Dongen, B. F., De Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., & Van Der Aalst, W. M. P. (2005). The ProM framework: A new era in process mining tool support. *Lecture Notes in Computer Science*, *3536*(i), 444–454. https://doi.org/10.1007/11494744_25

Van Zelst, S. J., & Leemans, S. J. J. (2020). Translating workflow nets to process trees: An algorithmic approach. *Algorithms*, *13*(11), 1–27. https://doi.org/10.3390/a13110279

Veiga, G. M., & Ferreira, D. R. (2010). Understanding spaghetti models with sequence clustering for ProM. *Lecture Notes in Business Information Processing*, *43 LNBIP*, 92–103. https://doi.org/10.1007/978-3-642-12186-9_10

Wang, L., Du, Y., & Qi, L. (2019). Efficient deviation detection between a process model and event logs. *IEEE/CAA Journal of Automatica Sinica*, *6*(6), 1352–1364. https://doi.org/10.1109/JAS.2019.1911750

Weijters, A. J. M. M., van der Aalst, W. M. P., & de Medeiros;, A. K. A. (2006). Process Mining with the HeuristicsMiner Algorithm. *Beta Working Papers*.

Wirth, R. (2000). CRISP-DM : Towards a Standard Process Model for Data Mining. *Proceedings of the Fourth International Conference on the Practical Application of Knowledge Discovery and Data Mining*, *24959*, 29–39.

Yousefi, M., & Yousefi, M. (2020). Human resource allocation in an emergency department: A metamodel-based simulation optimization. *Kybernetes*, *49*(3), 779–796. https://doi.org/10.1108/K-12-2018-0675

ANEXO 1 - ESQUEMA COMPLETO DO *MIMIC-III*

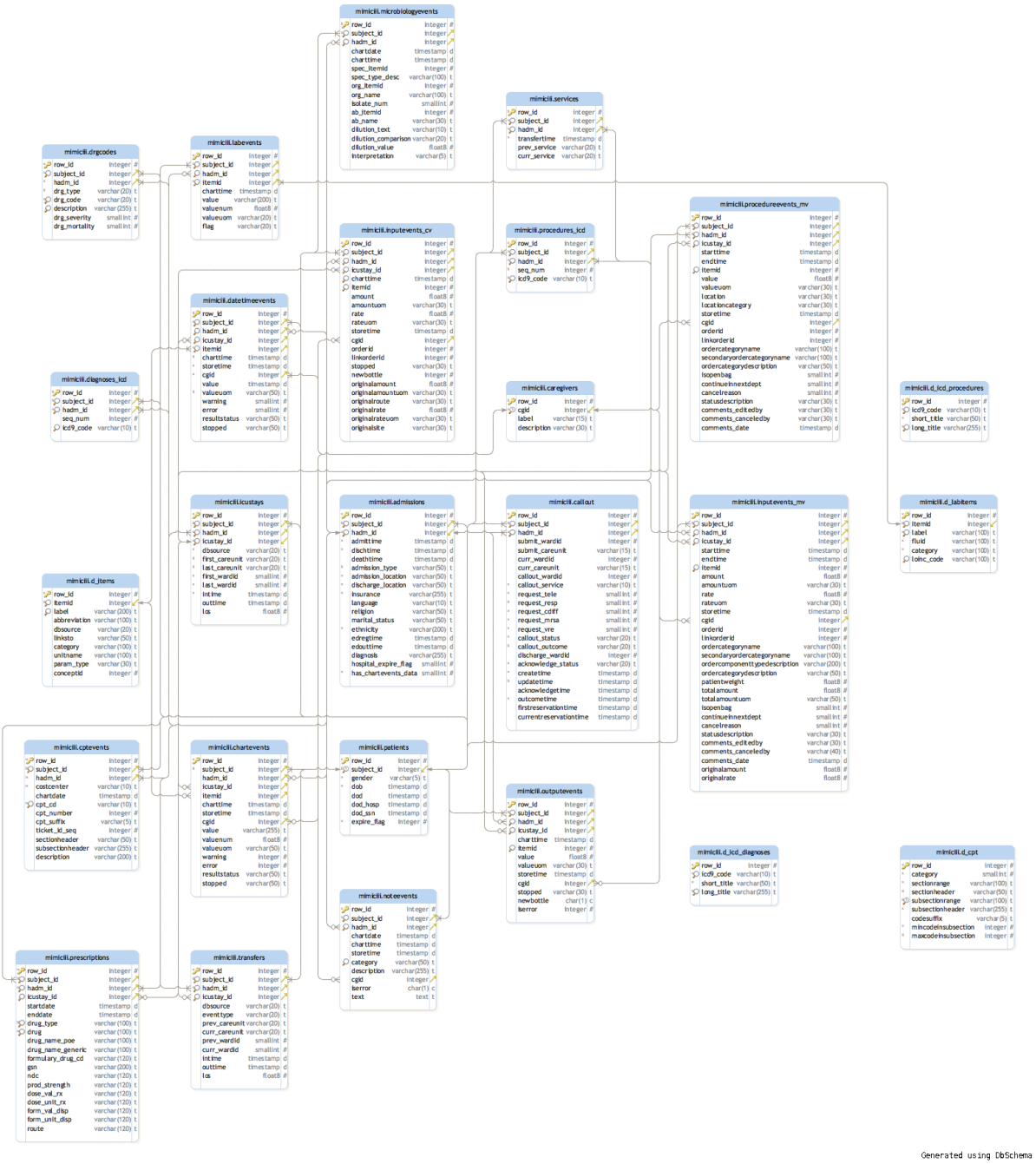


Diagrama extraído de https://mit-lcp.github.io/mimic-schema-spy/relationships.html.

ANEXO 2 – CERTIFICADO DE CONCLUSÃO DO *CITI PROGRAM COURSE*

CITI PROGRAM

Completion Date 24-May-2021
Expiration Date 23-May-2024
Record ID 42677266

This is to certify that:

André Filipe Gomes

Has completed the following CITI Program course:

Not valid for renewal of certification through CME.

Human Research
(Curriculum Group)
Data or Specimens Only Research
(Course Learner Group)
1 - Basic Course
(Stage)

Under requirements set by:

Massachusetts Institute of Technology Affiliates

CITI
Collaborative Institutional Training Initiative

Verify at www.citiprogram.org/verify/?w6bda2e8c-d4aa-43de-8ed0-f09a226bdd5c-42677266

ANEXO 3 – RELATÓRIO DE CONCLUSÃO DO *CITI PROGRAM COURSE*

COLLABORATIVE INSTITUTIONAL TRAINING INITIATIVE (CITI PROGRAM)

COMPLETION REPORT - PART 1 OF 2
COURSEWORK REQUIREMENTS*

* NOTE: Scores on this Requirements Report reflect quiz completions at the time all requirements for the course were met. See list below for details. See separate Transcript Report for more recent quiz scores, including those on optional (supplemental) course elements.

- **Name:** André Filipe Gomes (ID: 10153409)
- **Institution Affiliation:** Massachusetts Institute of Technology Affiliates (ID: 1912)
- **Institution Email:** estgv15362@alunos.estgv.ipv.pt
- **Institution Unit:** Computer Science

- **Curriculum Group:** Human Research
- **Course Learner Group:** Data or Specimens Only Research
- **Stage:** Stage 1 - Basic Course

- **Record ID:** 42677266
- **Completion Date:** 24-May-2021
- **Expiration Date:** 23-May-2024
- **Minimum Passing:** 90
- **Reported Score*:** 100

REQUIRED AND ELECTIVE MODULES ONLY	DATE COMPLETED	SCORE
Belmont Report and Its Principles (ID: 1127)	24-May-2021	3/3 (100%)
History and Ethics of Human Subjects Research (ID: 498)	24-May-2021	5/5 (100%)
Basic Institutional Review Board (IRB) Regulations and Review Process (ID: 2)	24-May-2021	5/5 (100%)
Records-Based Research (ID: 5)	24-May-2021	3/3 (100%)
Genetic Research in Human Populations (ID: 6)	24-May-2021	5/5 (100%)
Populations in Research Requiring Additional Considerations and/or Protections (ID: 16680)	24-May-2021	5/5 (100%)
Research and HIPAA Privacy Protections (ID: 14)	24-May-2021	5/5 (100%)
Conflicts of Interest in Human Subjects Research (ID: 17464)	24-May-2021	5/5 (100%)
Massachusetts Institute of Technology (ID: 1290)	24-May-2021	No Quiz

For this Report to be valid, the learner identified above must have had a valid affiliation with the CITI Program subscribing institution identified above or have been a paid Independent Learner.

Verify at: www.citiprogram.org/verify/?k046eabfc-a1c7-4415-be69-9f8262ac5446-42677266

Collaborative Institutional Training Initiative (CITI Program)
Email: support@citiprogram.org
Phone: 888-529-5929
Web: https://www.citiprogram.org

COLLABORATIVE INSTITUTIONAL TRAINING INITIATIVE (CITI PROGRAM)

COMPLETION REPORT - PART 2 OF 2
COURSEWORK TRANSCRIPT**

** NOTE: Scores on this Transcript Report reflect the most current quiz completions, including quizzes on optional (supplemental) elements of the course. See list below for details. See separate Requirements Report for the reported scores at the time all requirements for the course were met.

- **Name:** André Filipe Gomes (ID: 10153409)
- **Institution Affiliation:** Massachusetts Institute of Technology Affiliates (ID: 1912)
- **Institution Email:** estgv15362@alunos.estgv.ipv.pt
- **Institution Unit:** Computer Science

- **Curriculum Group:** Human Research
- **Course Learner Group:** Data or Specimens Only Research
- **Stage:** Stage 1 - Basic Course

- **Record ID:** 42677266
- **Report Date:** 24-May-2021
- **Current Score**:** 100

REQUIRED, ELECTIVE, AND SUPPLEMENTAL MODULES	MOST RECENT	SCORE
Basic Institutional Review Board (IRB) Regulations and Review Process (ID: 2)	24-May-2021	5/5 (100%)
Belmont Report and Its Principles (ID: 1127)	24-May-2021	3/3 (100%)
Records-Based Research (ID: 5)	24-May-2021	3/3 (100%)
Genetic Research in Human Populations (ID: 6)	24-May-2021	5/5 (100%)
Research and HIPAA Privacy Protections (ID: 14)	24-May-2021	5/5 (100%)
History and Ethics of Human Subjects Research (ID: 498)	24-May-2021	5/5 (100%)
Populations in Research Requiring Additional Considerations and/or Protections (ID: 16680)	24-May-2021	5/5 (100%)
Conflicts of Interest in Human Subjects Research (ID: 17464)	24-May-2021	5/5 (100%)
Massachusetts Institute of Technology (ID: 1290)	24-May-2021	No Quiz

For this Report to be valid, the learner identified above must have had a valid affiliation with the CITI Program subscribing institution identified above or have been a paid Independent Learner.

Verify at: www.citiprogram.org/verify/?k046eabfc-a1c7-4415-be69-9f8262ac5446-42677266

Collaborative Institutional Training Initiative (CITI Program)
Email: support@citiprogram.org
Phone: 888-529-5929
Web: https://www.citiprogram.org

ANEXO 4 – CÓDIGO *PYTHON* DE PRÉ-PROCESSAMENTO DOS DADOS DE TESTE

```python
import pandas as pd
import datetime
from pm4py.objects.log.util import dataframe_utils

path = '*****'

# ------------------------------------ #
# Read data and select important columns #
# ------------------------------------ #
patients = pd.read_csv(path + '2-Dados\\PATIENTS.csv')[['subject_id', 'gender', 'dob']]
admissions = pd.read_csv(path + '2-Dados\\ADMISSIONS.csv')[['hadm_id', 'admission_location', 'deathtime']]
transfers = pd.read_csv(path + '2-Dados\\TRANSFERS.csv')[['subject_id', 'hadm_id', 'eventtype', 'curr_careunit', 'intime', 'outtime']]
procedures = pd.read_csv(path + '2-Dados\\PROCEDUREEVENTS_MV.csv')[['hadm_id', 'ordercategoryname', 'starttime', 'endtime']]

# ------------------------------------ #
# Join data                            #
# ------------------------------------ #
# Get Patient info in Transfers
dataset = pd.merge(left=transfers, right=patients, how='inner', left_on='subject_id', right_on='subject_id')
# Admissions
dataset = pd.merge(left=dataset, right=admissions, how='inner', left_on='hadm_id', right_on='hadm_id')
# Procedures
order_proc = pd.merge(left=dataset[['hadm_id', 'intime', 'outtime']], right=procedures, how='left', left_on='hadm_id', right_on='hadm_id')
order_proc = order_proc.loc[((order_proc['starttime'] >= order_proc['intime']) &
                             (order_proc['starttime'] < order_proc['outtime'])) |
                            ((order_proc['endtime'] > order_proc['intime']) &
                             (order_proc['endtime'] <= order_proc['outtime']))]
dataset = pd.merge(left=dataset, right=order_proc, how='left', left_on=['hadm_id', 'intime', 'outtime'], right_on=['hadm_id', 'intime', 'outtime'])
# ------------------------------------ #
# Prepare final dataset                #
# ------------------------------------ #
dataset = pd.DataFrame(
    data = {'admission:id': dataset['hadm_id'],

            'admission:type': dataset['admission_location'],
            'admission:discharge_type': ['death' if str(deathtime) != 'nan' else 'discharge' for deathtime in dataset['deathtime']],
            'patient:gender': dataset['gender'],
            'patient:date_of_birth': [date.split(' ')[0] for date in dataset['dob']],
            'event:date_of_week': [datetime.datetime.strptime(str(inttime), '%Y-%m-%d %H:%M:%S').weekday() for inttime in dataset['intime']],

            'event:type': dataset['eventtype'],
            'event:care_unit_in': dataset['intime'],
            'event:care_unit_out': dataset['outtime'],
            'event:care_unit': ['GCU' if str(careunit) == 'nan' and str(eventtype) != 'discharge' else careunit
                                for careunit, eventtype in zip(dataset['curr_careunit'], dataset['eventtype'])],

            'event:procedure': dataset['ordercategoryname'],
            'event:procedure_start': dataset['starttime'],
            'event:procedure_end': dataset['endtime']
    }
)
dataset.to_csv (path + 'DATASET.csv', index = False, header=True)
# ------------------------------------ #
# Convert to log format                #
# ------------------------------------ #
dataset = pd.DataFrame(
    data = {'case:concept:name': dataset['admission:id'],

            'admission:type': dataset['admission:type'],
            'admission:discharge_type': dataset['admission:discharge_type'],
            'patient:gender': dataset['patient:gender'],
            'patient:date_of_birth': dataset['patient:date_of_birth'],
            'event:date_of_week': dataset['event:date_of_week'],

            'concept:name': [ '-'.join([item for item in [event, unit, procedure] if str(item) != 'nan'])
                              for event, unit, procedure in zip(dataset['event:type'], dataset['event:care_unit'], dataset['event:procedure'])],

            'time:timestamp': [ start if str(start) != 'nan' else intime
                                for start, intime in zip(dataset['event:procedure_start'], dataset['event:care_unit_in'])]
    }
)
dataset = dataset.drop_duplicates()
dataset = dataframe_utils.convert_timestamp_columns_in_df(dataset)
dataset = dataset.sort_values('time:timestamp')
dataset.to_csv (path + 'LOGS.csv', index = False, header=True)
```

ANEXO 5 – CÓDIGO *PYTHON* DE PRÉ-PROCESSAMENTO DOS DADOS FINAIS

```python
import pandas as pd
import datetime

root = '*****'
tables = root + '2-Dados\\'
path = root + '3-Dataset\\'

# ---------------------------------------- #
# Read data and select important columns #
# ---------------------------------------- #
patients = pd.read_csv(tables + 'PATIENTS.csv')[['SUBJECT_ID', 'GENDER', 'DOB']]
admissions = pd.read_csv(tables + 'ADMISSIONS.csv')[['HADM_ID', 'SUBJECT_ID', 'ADMISSION_TYPE', 'ADMISSION_LOCATION', 'DEATHTIME']]
transfers = pd.read_csv(tables + 'TRANSFERS.csv')[['SUBJECT_ID', 'HADM_ID', 'EVENTTYPE', 'CURR_CAREUNIT', 'INTIME', 'OUTTIME']].dropna(subset=['EVENTTYPE'])

# ---------------------------------------- #
# Join data                                #
# ---------------------------------------- #
# Join Transfers and Patient
data = pd.merge(left=transfers, right=patients, how='inner', left_on='SUBJECT_ID', right_on='SUBJECT_ID')
# Join Admissions
data = pd.merge(left=data, right=admissions, how='inner', left_on=['SUBJECT_ID', 'HADM_ID'], right_on=['SUBJECT_ID', 'HADM_ID'])

# ---------------------------------------- #
# Prepare final dataset                    #
# ---------------------------------------- #
dataset = pd.DataFrame(data =
    {
        'admission:id': data['HADM_ID'],
        'admission:type': data['ADMISSION_TYPE'],
        'admission:location': data['ADMISSION_LOCATION'],
        'admission:discharge_type': ['death' if str(deathtime) != 'nan' else 'discharge' for deathtime in data['DEATHTIME']],
        'patient:gender': data['GENDER'],
        'patient:date_of_birth': [date.split(' ')[0] for date in data['DOB']],
        'event:date_of_week': [datetime.datetime.strptime(str(inttime), '%Y-%m-%d %H:%M:%S').weekday() for inttime in data['INTIME']],

        'event:type': data['EVENTTYPE'],
        'event:care_unit_in': data['INTIME'],
        'event:care_unit_out': data['OUTTIME'],
        'event:care_unit': ['GCU' if str(careunit) == 'nan' and str(eventtype) != 'discharge' else careunit
                            for careunit, eventtype in zip(data['CURR_CAREUNIT'], data['EVENTTYPE'])]
    }
)
dataset.to_csv (path + 'DATASET.csv', index = False, header=True)
# ---------------------------------------- #
# Convert to log format                    #
# ---------------------------------------- #
logs = pd.DataFrame(data =
    {
        'case:concept:name': dataset['admission:id'],

        'admission:type': dataset['admission:type'],
        'admission:location': dataset['admission:location'],
        'admission:discharge_type': dataset['admission:discharge_type'],
        'patient:gender': dataset['patient:gender'],
        'patient:date_of_birth': dataset['patient:date_of_birth'],
        'event:date_of_week': dataset['event:date_of_week'],

        'concept:name': ['-'.join([item for item in [event, unit] if str(item) != 'nan'])
                         for event, unit in zip(dataset['event:type'], dataset['event:care_unit'])],

        'time:timestamp_in': dataset['event:care_unit_in'],
        'time:timestamp_out': dataset['event:care_unit_out']
    }
)
logs = logs.drop_duplicates()
logs = logs.sort_values('time:timestamp_in')
logs.to_csv (path + 'LOGS.csv', index = False, header=True)
```

ANEXO 6 – CÓDIGO *PYTHON* PARA IMPLEMENTAR AS FUNCIONALIDADES DO *PM4PY*

```
import time
# ------------------------------- #
# Set log level:                  #
#  * Easy logs          - 0       #
#  * Duplicated logs    - 1       #
#  * Loop logs          - 2       #
#  * All logs           - 3       #
# Save logs/logs_test/variants:   #
#  * Disable            - 0       #
#  * Enable             - 1       #
# Variants:                       #
#  * Disable            - 0       #
#  * Enable             - 1       #
# Filtering Event Data:           #
#  * None               - 0       #
#  * Timeframe          - 1       #
#  * Start activity     - 2       #
#  * End activity       - 3       #
#  * Attributes values  - 4       #
# Process Discovery:              #
#  * Disable algorithm  - 0       #
#  * Enable algorithm   - 1       #
# Conformance Checking:           #
#  * Disable technique  - 0       #
#  * Enable technique   - 1       #
# Statistics:                     #
#  * Disable            - 0       #
#  * Enable             - 1       #
# Evaluation:                     #
#  * Disable            - 0       #
#  * Enable             - 1       #
# ------------------------------- #
# Set log level
level = 3
# Save logs / variants
save_logs = 0
save_logs_test = 0
save_variants = 0
# Variantes
data_variants = 0
# Filtering Event Data
data_filter = 0
# Process Discovery
alpha = 0
dfg_f = 0
dfg_p = 0
heuristics = 0
heuristics_pn = 0
inductive = 0
inductive_pn = 0
# Conformance Checking
token_replay_pn = 0
alignments_pn = 0
# Statistics
statistics = 0
# Evaluation
evaluation = 0
# ------------------------------- #
# Load all the required libraries #
# ------------------------------- #
# Data
import pandas as pd
from pm4py.objects.conversion.log import converter as log_converter
from pm4py.objects.log.exporter.xes import exporter as xes_exporter
import matplotlib.pyplot as plt
# Filtering Event Data
from pm4py.algo.filtering.log.timestamp import timestamp_filter
from pm4py.algo.filtering.log.start_activities import start_activities_filter
from pm4py.algo.filtering.log.end_activities import end_activities_filter
from pm4py.algo.filtering.log.attributes import attributes_filter
from pm4py.algo.filtering.log.variants import variants_filter
from pm4py.statistics.traces.log import case_statistics
# Process mining
from pm4py.algo.discovery.alpha import algorithm as alpha_miner
from pm4py.algo.discovery.inductive import algorithm as inductive_miner
from pm4py.algo.discovery.heuristics import algorithm as heuristics_miner
from pm4py.algo.discovery.dfg import algorithm as dfg_discovery
# Visualization
from pm4py.visualization.petrinet import visualizer as pn_visualizer
from pm4py.visualization.process_tree import visualizer as pt_visualizer
from pm4py.visualization.heuristics_net import visualizer as hn_visualizer
from pm4py.visualization.dfg import visualizer as dfg_visualization
from pm4py.objects.conversion.process_tree import converter as pt_converter
# Conformance Checking
from pm4py.algo.conformance.tokenreplay import algorithm as token_replay
from pm4py.algo.conformance.alignments import algorithm as alignments
# Statistics
from pm4py.statistics.traces.log import case_arrival
from pm4py.util import constants
from pm4py.visualization.graphs import visualizer as graphs_visualizer
# Evaluation
from pm4py.evaluation.replay_fitness import evaluator as replay_fitness_evaluator
from pm4py.evaluation.precision import evaluator as precision_evaluator
from pm4py.evaluation.generalization import evaluator as generalization_evaluator
from pm4py.evaluation.simplicity import evaluator as simplicity_evaluator
# ------------------------------- #
# Import log data                 #
# ------------------------------- #
root = '*****'
path = root + '3-PM4Py'
dataset = pd.read_csv(root + '1-Dataset\\LOGS.csv')
dataset['time:timestamp'] = pd.to_datetime(dataset['time:timestamp'])
```

ANEXO 6 – Código *Python* para implementar as funcionalidades do *PM4Py*

```
# ------------------------------- #
# Convert to log format           #
# ------------------------------- #
dataset_test = dataset[187:192].append(dataset[651:656])
log_test = log_converter.apply(dataset_test)
if level == 0:
    dataset = dataset[37:40]
if level == 1:
    dataset = dataset[808:812]
if level == 2:
    dataset = dataset[324:329]
log = log_converter.apply(dataset)
# Save logs
if save_logs == 1 and level == 3:
    xes_exporter.apply(log, '1-Logs/Logs.xes')
# Save logs test
if save_logs_test == 1 and level == 3:
    xes_exporter.apply(log_test, '1-Logs/Logs test.xes')
    data_logs = dataset_test['case:concept:name'].unique()
    logs_size = len(data_logs)
    fig = plt.figure(figsize=(20,3))
    ax=plt.subplot(111)
    ax.axis('off')
    headers = plt.table(cellText=[['Logs']],loc='bottom',cellLoc='center',bbox=[0, 0, 0.1, 0.3*logs_size])
    headers[0,0].set_facecolor("lightgray")
    for i in range(logs_size):
        plt.table(cellText=[dataset_test.loc[dataset_test['case:concept:name'] == data_logs[i]]['concept:name'].values],loc='bottom',cellLoc='center',
                  bbox=[0.1, 0.3*(logs_size-i-1), 1, 0.3])
    plt.savefig('1-Logs/Logs test.png')
# Save variants
variants = variants_filter.get_variants(log)
if save_variants == 1 and level == 3:
    variants_count = case_statistics.get_variant_statistics(log)
    variants_count = sorted(variants_count, key=lambda x: x['count'], reverse=True)
    variants_view = pd.DataFrame(variants_count)
    n_log = variants_view['count'].sum()
    variants_view['percentage'] = variants_view.apply(lambda row: round((row['count'] / n_log) * 100, 2), axis=1)
    variants_view.to_csv('1-Logs/Variants.csv', sep=';')
# ------------------------------------------------------------------------ #
# Variants                                                                 #
# ------------------------------------------------------------------------ #
variant_per = ''
if data_variants == 1:
    # Alter variable per(0.00, 0.01, 0.08, 0.12, 0.15)
    per = 0.15
    log = variants_filter.filter_log_variants_percentage(log, percentage=per)
    variant_per = '_variants(' + str(per) + ')'
# ------------------------------------------------------------------------ #
# Filtering Event Data                                                     #
# ------------------------------------------------------------------------ #
filter_name = ''
if data_filter == 1:
    # Alter start and end timestamp
    log = timestamp_filter.filter_traces_contained(log, "2161-09-19 17:54:42", "2163-11-21 19:01:00")
    filter_name = '_timeframe'
if data_filter == 2:
    # Alter array start activities
    log = start_activities_filter.apply(log, ["admit-TSICU", "admit-MICU-Peripheral Lines"])
    filter_name = '_start_activity'
if data_filter == 3:
    # Alter array end activity
    log = end_activities_filter.apply(log, ["discharge"])
    filter_name = '_end_activity'
if data_filter == 4:
    # Alter attribute array values, name and POSITIVE(manter ou remover)
    log = attributes_filter.apply(log, ["TRANSFER FROM SKILLED NUR"], parameters={attributes_filter.Parameters.ATTRIBUTE_KEY: "admission:type",
                                                                                  attributes_filter.Parameters.POSITIVE: True})
    filter_name = '_attributes_values'
# ------------------------------------------------------------------------ #
# Process Discovery                                                        #
# ------------------------------------------------------------------------ #
# ------------------------------- #
# Alpha Miner                     #
# ------------------------------- #
if alpha == 1:
    start = time.time()
    # Petri net
    net, initial_marking, final_marking = alpha_miner.apply(log)
    parameters = {pn_visualizer.Variants.FREQUENCY.value.Parameters.FORMAT: "png"}
    gviz = pn_visualizer.apply(net, initial_marking, final_marking,
                               parameters=parameters,
                               variant=pn_visualizer.Variants.FREQUENCY,
                               log=log)
    if level == 0:
        pn_visualizer.save(gviz, "2-Process Discovery/Alpha Miner/alpha_miner_petri_net"+variant_per+filter_name+"(easy_logs).png")
    if level == 1:
        pn_visualizer.save(gviz, "2-Process Discovery/Alpha Miner/alpha_miner_petri_net"+variant_per+filter_name+"(duplicated_logs).png")
    if level == 2:
        pn_visualizer.save(gviz, "2-Process Discovery/Alpha Miner/alpha_miner_petri_net"+variant_per+filter_name+"(loop_logs).png")
    if level == 3:
        pn_visualizer.save(gviz, "2-Process Discovery/Alpha Miner/alpha_miner_petri_net"+variant_per+filter_name+"(all_logs).png")
    end = time.time()
    print ("Alpha Miner: " + str(end - start))
# ------------------------------------------------------------------------ #
# ------------------------------- #
# Directly-Follows Graph          #
# ------------------------------- #
if dfg_f == 1:
    # Directly-Follows Graph with frequency
    start = time.time()
    dfg = dfg_discovery.apply(log)
    gviz = dfg_visualization.apply(dfg, log=log, variant=dfg_visualization.Variants.FREQUENCY)
    if level == 0:
        dfg_visualization.save(gviz, "2-Process Discovery/Directly-Follows Graph/directly_follows_frequency"+variant_per+filter_name+"(easy_logs).png")
    if level == 1:
        dfg_visualization.save(gviz, "2-Process Discovery/Directly-Follows Graph/directly_follows_frequency"+variant_per+filter_name+"(duplicated_logs).png")
    if level == 2:
        dfg_visualization.save(gviz, "2-Process Discovery/Directly-Follows Graph/directly_follows_frequency"+variant_per+filter_name+"(loop_logs).png")
    if level == 3:
        dfg_visualization.save(gviz, "2-Process Discovery/Directly-Follows Graph/directly_follows_frequency"+variant_per+filter_name+"(all_logs).png")
    end = time.time()
    print ("DFG with frequency: " + str(end - start))
if dfg_p == 1:
    # Directly-Follows Graph with performance
    start = time.time()
    dfg = dfg_discovery.apply(log, variant=dfg_discovery.Variants.PERFORMANCE)
    gviz = dfg_visualization.apply(dfg, log=log, variant=dfg_visualization.Variants.PERFORMANCE)
    if level == 0:
        dfg_visualization.save(gviz, "2-Process Discovery/Directly-Follows Graph/directly_follows_performance"+variant_per+filter_name+"(easy_logs).png")
    if level == 1:
        dfg_visualization.save(gviz, "2-Process Discovery/Directly-Follows Graph/directly_follows_performance"+variant_per+filter_name+"(duplicated_logs).png")
    if level == 2:
        dfg_visualization.save(gviz, "2-Process Discovery/Directly-Follows Graph/directly_follows_performance"+variant_per+filter_name+"(loop_logs).png")
    if level == 3:
        dfg_visualization.save(gviz, "2-Process Discovery/Directly-Follows Graph/directly_follows_performance"+variant_per+filter_name+"(all_logs).png")
    end = time.time()
    print ("DFG with performance: " + str(end - start))
```

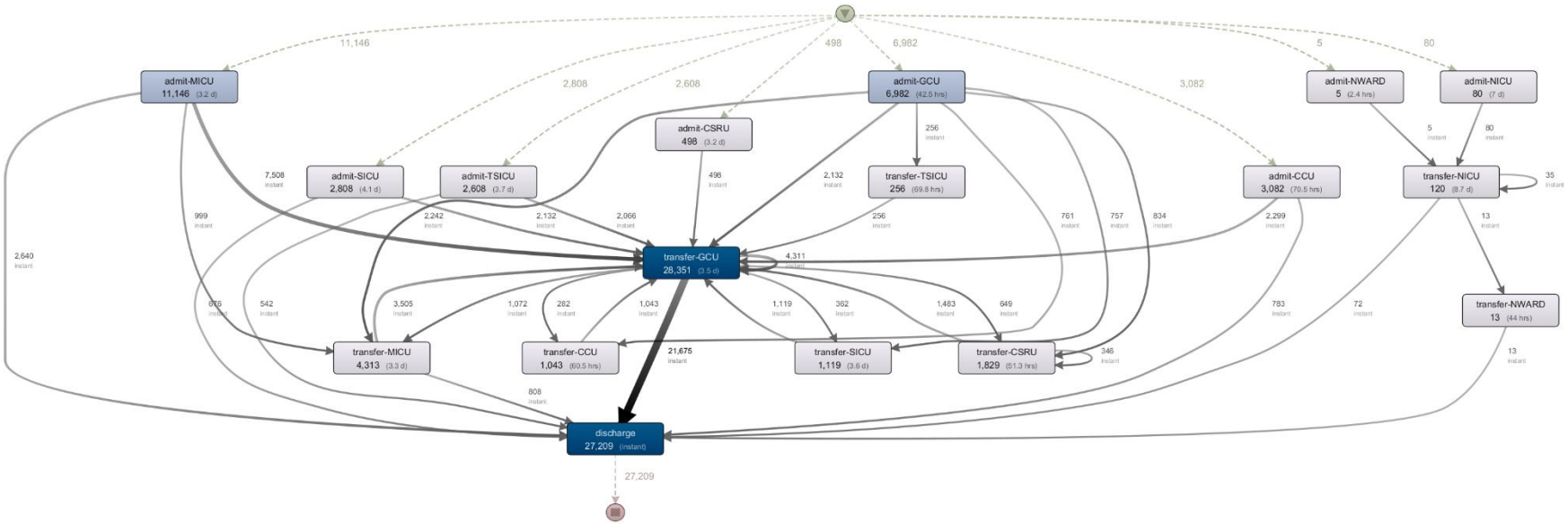
ANEXO 6 – Código *Python* para implementar as funcionalidades do *PM4Py*

```
# ------------------------------------------------------------------------------ #
# ------------------------------- #
# Heuristic Miner                 #
# ------------------------------- #
if heuristics == 1:
    # Heuristics net
    start = time.time()
    heu_net = heuristics_miner.apply_heu(log)
    gviz = hn_visualizer.apply(heu_net)
    if level == 0:
        hn_visualizer.save(gviz, "2-Process Discovery/Heuristic Miner/heuristic_miner"+variant_per+filter_name+"(easy_logs).png")
    if level == 1:
        hn_visualizer.save(gviz, "2-Process Discovery/Heuristic Miner/heuristic_miner"+variant_per+filter_name+"(duplicated_logs).png")
    if level == 2:
        hn_visualizer.save(gviz, "2-Process Discovery/Heuristic Miner/heuristic_miner"+variant_per+filter_name+"(loop_logs).png")
    if level == 3:
        hn_visualizer.save(gviz, "2-Process Discovery/Heuristic Miner/heuristic_miner"+variant_per+filter_name+"(all_logs).png")
    end = time.time()
    print ("Heuristic Miner with heuristics net: " + str(end - start))
if heuristics_pn == 1:
    # Petri net
    start = time.time()
    net, initial_marking, final_marking = heuristics_miner.apply(log)
    parameters = {pn_visualizer.Variants.FREQUENCY.value.Parameters.FORMAT: "png"}
    gviz = pn_visualizer.apply(net, initial_marking, final_marking,
                               parameters=parameters,
                               variant=pn_visualizer.Variants.FREQUENCY,
                               log=log)
    if level == 0:
        pn_visualizer.save(gviz, "2-Process Discovery/Heuristic Miner/heuristic_miner_petri_net"+variant_per+filter_name+"(easy_logs).png")
    if level == 1:
        pn_visualizer.save(gviz, "2-Process Discovery/Heuristic Miner/heuristic_miner_petri_net"+variant_per+filter_name+"(duplicated_logs).png")
    if level == 2:
        pn_visualizer.save(gviz, "2-Process Discovery/Heuristic Miner/heuristic_miner_petri_net"+variant_per+filter_name+"(loop_logs).png")
    if level == 3:
        pn_visualizer.save(gviz, "2-Process Discovery/Heuristic Miner/heuristic_miner_petri_net"+variant_per+filter_name+"(all_logs).png")
    end = time.time()
    print ("Heuristic Miner with petri net: " + str(end - start))
# ------------------------------------------------------------------------------ #
# ------------------------------- #
# Inductive Miner                 #
# ------------------------------- #
if inductive == 1:
    # Process tree
    start = time.time()
    tree = inductive_miner.apply_tree(log)
    gviz = pt_visualizer.apply(tree)
    if level == 0:
        pt_visualizer.save(gviz, "2-Process Discovery/Inductive Miner/inductive_miner"+variant_per+filter_name+"(easy_logs).png")
    if level == 1:
        pt_visualizer.save(gviz, "2-Process Discovery/Inductive Miner/inductive_miner"+variant_per+filter_name+"(duplicated_logs).png")
    if level == 2:
        pt_visualizer.save(gviz, "2-Process Discovery/Inductive Miner/inductive_miner"+variant_per+filter_name+"(loop_logs).png")
    if level == 3:
        pt_visualizer.save(gviz, "2-Process Discovery/Inductive Miner/inductive_miner"+variant_per+filter_name+"(all_logs).png")
    end = time.time()
    print ("Inductive Miner with process tree: " + str(end - start))
if inductive_pn == 1:
    # Petri net
    start = time.time()
    tree = inductive_miner.apply_tree(log)
    net, initial_marking, final_marking = pt_converter.apply(tree)
    parameters = {pn_visualizer.Variants.FREQUENCY.value.Parameters.FORMAT: "png"}
    gviz = pn_visualizer.apply(net, initial_marking, final_marking,
                               parameters=parameters,
                               variant=pn_visualizer.Variants.FREQUENCY,
                               log=log)
    if level == 0:
        pn_visualizer.save(gviz, "2-Process Discovery/Inductive Miner/inductive_miner_petri_net"+variant_per+filter_name+"(easy_logs).png")
    if level == 1:
        pn_visualizer.save(gviz, "2-Process Discovery/Inductive Miner/inductive_miner_petri_net"+variant_per+filter_name+"(duplicated_logs).png")
    if level == 2:
        pn_visualizer.save(gviz, "2-Process Discovery/Inductive Miner/inductive_miner_petri_net"+variant_per+filter_name+"(loop_logs).png")
    if level == 3:
        pn_visualizer.save(gviz, "2-Process Discovery/Inductive Miner/inductive_miner_petri_net"+variant_per+filter_name+"(all_logs).png")
    end = time.time()
    print ("Inductive Miner with petri net: " + str(end - start))

# ------------------------------------------------------------------------------ #
# Conformance Checking                                                           #
# ------------------------------------------------------------------------------ #
# ------------------------------- #
# Token-based replay              #
# ------------------------------- #
if (heuristics_pn == 1 or inductive_pn == 1) and level == 3 and token_replay_pn == 1:
    replayed_traces = token_replay.apply(log_test, net, initial_marking, final_marking)
    replayed_traces_view = pd.DataFrame(replayed_traces)
    replayed_traces_view.to_csv('3-Conformance Checking/Token-based replay.csv', sep=';')
# ------------------------------------------------------------------------------ #
# ------------------------------- #
# Alignments                      #
# ------------------------------- #
if (heuristics_pn == 1 or inductive_pn == 1) and level == 3 and alignments_pn == 1:
    alignments = alignments.apply_log(log_test, net, initial_marking, final_marking)
    no_fitness = 0
    for alignment in alignments:
        if alignment['fitness'] != 1:
            no_fitness +=1
    print ("Alignments:")
    print("\t- no_fitness: " + str(no_fitness))
    alignments_view = pd.DataFrame(alignments)
    alignments_view.to_csv('3-Conformance Checking/Alignments.csv', sep=';')
# ------------------------------------------------------------------------------ #
# Statistics                                                                     #
# ------------------------------------------------------------------------------ #
if statistics == 1:
    # Median case duration
    median_case_duration = case_statistics.get_median_caseduration(log, parameters={
        case_statistics.Parameters.TIMESTAMP_KEY: "time:timestamp"
    })
    # Case dispersion ratio
    case_dispersion_ratio = case_arrival.get_case_dispersion_avg(log, parameters={
        case_arrival.Parameters.TIMESTAMP_KEY: "time:timestamp"})

    fig = plt.figure(figsize=(5,3))
    ax=plt.subplot(111)
    ax.axis('off')
    headers = plt.table(cellText=[['Statistics']],loc='bottom',cellLoc='center',bbox=[0, 0.45, 1, 0.15])
    headers[0,0].set_facecolor("lightgray")
    headers = plt.table(cellText=[['Median Case Duration', 'Average Distance']],loc='bottom',cellLoc='center',bbox=[0, 0.3, 1, 0.15])
    headers[0,0].set_facecolor("lightgray")
    headers[0,1].set_facecolor("lightgray")
    plt.table(cellText=[[str(median_case_duration), str(case_dispersion_ratio)]],loc='bottom',cellLoc='center',bbox=[0, 0, 1, 0.3])
    plt.savefig('4-Statistics/Statistics.png')
```

ANEXO 6 – Código *Python* para implementar as funcionalidades do *PM4Py*

```
    # Distribution of case duration
    x, y = case_statistics.get_kde_caseduration(log, parameters={constants.PARAMETER_CONSTANT_TIMESTAMP_KEY: "time:timestamp"})
    gviz = graphs_visualizer.apply_plot(x, y, variant=graphs_visualizer.Variants.CASES)
    graphs_visualizer.save(gviz, "4-Statistics/Distribution of case duration.png")
    # Distribution of events over time
    x, y = attributes_filter.get_kde_date_attribute(log, attribute="time:timestamp")
    gviz = graphs_visualizer.apply_plot(x, y, variant=graphs_visualizer.Variants.DATES)
    graphs_visualizer.save(gviz, "4-Statistics/Distribution of events over time.png")
    # Distribution of a numeric attribute
    attribute = "event:date_of_week"
    x, y = attributes_filter.get_kde_numeric_attribute(log, attribute)
    gviz = graphs_visualizer.apply_plot(x, y, variant=graphs_visualizer.Variants.ATTRIBUTES)
    graphs_visualizer.save(gviz, "4-Statistics/Distribution of "+attribute.split(':')[1]+".png")
# ------------------------------------------------------------------------ #
# Evaluation                                                               #
# ------------------------------------------------------------------------ #
if (heuristics_pn == 1 or inductive_pn == 1) and level == 3 and evaluation == 1:
    # Replay Fitness
    fitness_token = replay_fitness_evaluator.apply(log_test, net, initial_marking, final_marking, variant=replay_fitness_evaluator.Variants.TOKEN_BASED)
    fitness_align = replay_fitness_evaluator.apply(log_test, net, initial_marking, final_marking, variant=replay_fitness_evaluator.Variants.ALIGNMENT_BASED)
    # Precision
    precision_token = precision_evaluator.apply(log_test, net, initial_marking, final_marking, variant=precision_evaluator.Variants.ETCONFORMANCE_TOKEN)
    precision_align = precision_evaluator.apply(log_test, net, initial_marking, final_marking, variant=precision_evaluator.Variants.ALIGN_ETCONFORMANCE)
    # Generalization
    generalization = generalization_evaluator.apply(log_test, net, initial_marking, final_marking)
    # Simplicity
    simplicity = simplicity_evaluator.apply(net)

    fig = plt.figure(figsize=(20,3))
    ax=plt.subplot(111)
    ax.axis('off')
    headers = plt.table(cellText=[['Generalization', 'Simplicity']],loc='bottom',cellLoc='center',bbox=[0.5, 0.3, 0.5, 0.3])
    headers[0,0].set_facecolor("lightgray")
    headers[0,1].set_facecolor("lightgray")
    plt.table(cellText=[[str(generalization), str(simplicity)]],loc='bottom',cellLoc='center',bbox=[0.5, 0, 0.5, 0.3])
    headers = plt.table(cellText=[['Replay Fitness', 'Precision']],loc='bottom',cellLoc='center',bbox=[0, 0.45, 0.5, 0.15])
    headers[0,0].set_facecolor("lightgray")
    headers[0,1].set_facecolor("lightgray")
    headers = plt.table(cellText=[['Token-based Replay', 'Alignments', 'Token-based Replay', 'Alignments']],loc='bottom',cellLoc='center',bbox=[0, 0.3, 0.5,
    headers[0,0].set_facecolor("lightgray")
    headers[0,1].set_facecolor("lightgray")
    headers[0,2].set_facecolor("lightgray")
    headers[0,3].set_facecolor("lightgray")
    plt.table(cellText=[[str(fitness_token["average_trace_fitness"]), str(fitness_align["averageFitness"]), str(precision_token), str(precision_align)]],
              loc='bottom',cellLoc='center',bbox=[0, 0, 0.5, 0.3])
    plt.savefig('5-Evaluation/Evaluation.png')
```

ANEXO 7 – MODELO REAL DO PROCESSO



ANEXO 8 – CÓDIGO *PYTHON* PARA CRIAÇÃO DOS *LOGS* PREVISTOS

```python
# ---------------------------------- #
# Load all the required libraries    #
# ---------------------------------- #
import xml.etree.ElementTree as ET
from xml.dom import minidom

# ---------------------------------- #
# Setup parameters                   #
# ---------------------------------- #
path_file = '*****'
file_name = 'logs_MICU.xes'
logs_name = 'Real previstos'
logs = [('1', ['admit-MICU', 'discharge']), ('2', ['admit-MICU', 'transfer-NICU', 'discharge'])]

# ---------------------------------- #
# Create a log                       #
# ---------------------------------- #

# Creates a tree struct with a log format
log = ET.Element('log')
log.set('xes.version', '1.0')
extension = ET.SubElement(log, 'extension')
extension.set('name', 'Concept')
extension.set('prefix', 'concept')
extension.set('uri', 'http://www.xes-standard.org/concept.xesext')
classifier = ET.SubElement(log, 'classifier')
classifier.set('name', 'Event Name')
classifier.set('keys', 'concept:name')
log_string = ET.SubElement(log, 'string')
log_string.set('key', 'concept:name')
log_string.set('value', logs_name+'.xes')
# Convert logs to xes format
for trace_name, events in logs:
    trace = ET.SubElement(log, 'trace')
    trace_string = ET.SubElement(trace, 'string')
    trace_string.set('key', 'concept:name')
    trace_string.set('value', trace_name)
    for event_name in events:
        event = ET.SubElement(trace, 'event')
        event = ET.SubElement(event, 'string')
        event.set('key', 'concept:name')
        event.set('value', event_name)

# Saves a xes structure in a file with a .xes extension
tree_out = minidom.parseString(ET.tostring(log)).toprettyxml(indent="\t", encoding='utf-8')
with open(path_file+file_name, 'wb') as f:
    f.write(tree_out)
```

ANEXO 9 – EXEMPLO DE FICHEIRO DE *LOGS* PREVISTOS

```
<?xml version="1.0" encoding="utf-8"?>
<log xes.version="1.0">
    <extension name="Concept" prefix="concept" uri="http://www.xes-standard.org/concept.xesext"/>
    <classifier keys="concept:name" name="Event Name"/>
    <string key="concept:name" value="Logs previstos.xes"/>
    <trace>
        <string key="concept:name" value="1.1"/>
        <event>
            <string key="concept:name" value="admit-MICU"/>
        </event>
        <event>
            <string key="concept:name" value="discharge"/>
        </event>
    </trace>
    <trace>
        <string key="concept:name" value="1.2"/>
        <event>
            <string key="concept:name" value="admit-MICU"/>
        </event>
        <event>
            <string key="concept:name" value="transfer-GCU"/>
        </event>
        <event>
            <string key="concept:name" value="discharge"/>
        </event>
    </trace>
</log>
```