UHASSELT

KNOWLEDGE IN ACTION

Faculteit Bedrijfseconomische Wetenschappen

master handelsingenieur in de beleidsinformatica

Masterthesis

Een empirische analyse van de Inductive Miner

Cato Martens

Scriptie ingediend tot het behalen van de graad van master handelsingenieur in de beleidsinformatica

PROMOTOR :

Prof. dr. Benoit DEPAIRE

2023
2024

UHASSELT

KNOWLEDGE IN ACTION

www.uhasselt.be
Universiteit Hasselt
Campus Hasselt:
Martelarenlaan 42 | 3500 Hasselt
Campus Diepenbeek:
Agoralaan Gebouw D | 3590 Diepenbeek

UHASSELT

KNOWLEDGE IN ACTION

Faculteit Bedrijfseconomische Wetenschappen

master handelsingenieur in de beleidsinformatica

Masterthesis

Een empirische analyse van de Inductive Miner

Cato Martens

Scriptie ingediend tot het behalen van de graad van master handelsingenieur in de beleidsinformatica

PROMOTOR :

Prof. dr. Benoit DEPAIRE

Inductive Miner: Een empirische analyse

Cato Martens

Contributing authors: cato.martens@student.uhasselt.be;

Abstract

Het domein van Process Discovery heeft zich gericht op het ontwikkelen en evalueren van diverse algoritmes voor het ontdekken van processen. Dit resulteerde in een breed scala van Process Discovery algoritmes. Echter, de focus heeft een lange tijd gelegen op het competitief testen van deze algoritmes zonder diepgaand inzicht te verschaffen in de redenen achter hun prestaties en waarom de algoritmes goed werken. Deze paper tracht hierin een aanzet te geven door de invloed van Design Decisions op de prestaties van een Process Discovery algoritme te onderzoeken. Dit onderzoek tracht daardoor een meer diepgaande analyse uit te voeren en richt zich specifiek op het Inductive Miner algoritme. Een cruciale Design Decision binnen dit algoritme is de splitsingsvolgorde. Dit onderzoek analyseert empirisch hoe veranderingen in splitsingsvolgordes de kwaliteit van gegenereerde procesmodellen beïnvloeden aan de hand van vier belangrijke kwaliteitsmaatstaven: fitness, precision, generalisation en simplicity. De resultaten dragen bij tot een dieper inzicht in de impact van splitsingsvolgordes op het Inductive Miner algoritme en de body of knowledge van de Inductive Miner.

Keywords: Inductive Miner, Process Discovery, Empirical Analysis, Process Mining

1 Introduction

Informatiesystemen en grote hoeveelheden data zijn tegenwoordig niet meer weg te denken binnen een bedrijfscontext [1]. Bedrijven hebben daarnaast veel processen die beschrijven hoe bepaald werk uitgevoerd moet worden. In de informatiesystemen kunnen vervolgens activiteiten van een dergelijk proces geregistreerd worden [2–7]. Voor bedrijven en organisaties is het waardevol om de werkelijke prestaties en conformiteit van de processen te begrijpen en evalueren [2, 4–10]. Het domein Process Mining houdt zich hiermee bezig. Het toepassen van Process Mining stelt bedrijven en organisaties in staat om diepgaand inzicht te krijgen in hoe processen daadwerkelijk worden uitgevoerd en waar mogelijke verbeteringen zich kunnen voordoen. Kortom, Process Mining

wordt toegepast om processen uit event logs te ontdekken, monitoren en verbeteren met behulp van diverse algoritmes [1, 3, 7]. Deze event logs kunnen geëxtraheerd worden uit informatiesystemen van een bedrijf of organisatie en bevatten informatie over de geregistreerde activiteiten, zoals tijdstippen van uitvoering en de betrokkenen bij de uitvoering van de activiteit [7, 10, 11]. Het ontdekken van processen uit event logs fungeert vaak als basis voor verdere analyse, monitoring en verbetering van processen en is daarom een cruciaal aspect van Process Mining [7]. Het ontdekken van processen wordt binnen Process Mining, Process Discovery genoemd. Process Discovery neemt een event log en produceert een procesmodel dat het gedrag uit de event log zal representeren, zonder enige voorgaande kennis [1, 7, 11].

Er zijn al talrijke Process Discovery algoritmes ontwikkeld die succesvol processen kunnen ontdekken [8, 11]. Deze algoritmes worden in de literatuur beoordeeld, verbeterd en onderling vergeleken [8, 9, 12]. De focus van het Process Discovery domein heeft lang gelegen op het competitief testen van algoritmes [2, 13]. Hierbij is het voornaamste doel vaak om verbeteringen aan te brengen of aanwijzingen te verkrijgen over welk algoritme het best presteert, zonder diepgaand inzicht te verschaffen in de redenen achter die goede prestaties [2, 11, 13, 14]. Dit inzicht is echter essentieel om het onderzoek naar betere algoritmes voort te zetten en ervoor te zorgen dat deze algoritmes hun potentieel kunnen realiseren in reële bedrijfscases [2]. Dit benadrukt de noodzaak voor meer diepgaande onderzoeken naar Process Discovery algoritmes [2, 13].

In dit onderzoek zal de Inductive Miner onderzocht worden. Dankzij onderzoeken van Augusto, et al. (2019) en Peng, et al. (2021) blijkt dat de Inductive Miner een goed algoritme is binnen Process Discovery. De keuze voor de Inductive Miner werd verder gebaseerd op: (1) zijn vermogen om alle vier de semantische Process Tree operatoren te herkennen (AND, XOR, SEQUENCE, LOOP), (2) het genereren van een intuïtief procesmodel namelijk een Process Tree en (3) de beschikbaarheid van implementaties in PROM en PM4PY [6, 8, 9, 11, 15–17].

Het Inductive Miner algoritme volgt een Divide-en-Conquer werkwijze om processen te ontdekken uit event logs [8]. In het algoritme wordt een bepaalde volgorde van splitsingen gehanteerd om een event log op te splitsen (splitsingsvolgorde). Deze volgorde is een bepaalde Design Decision die gemaakt werd in het originele algoritme van Leemans, et al. (2013) [8]. Het effect van deze design keuze op de prestaties van de Inductive Miner zal empirisch onderzocht worden door veranderingen in deze splisingsvolgorde door te voeren. Dit onderzoek tracht eveneens de robuustheid van de Inductive Miner te onderzoeken door de relatie te analyseren tussen veranderingen in de splitsingsvolgorde binnen het algoritme en de Task Properties van het onderliggende proces dat als input voor het Inductive Miner algoritme wordt gebruikt. De twee centrale onderzoeksvragen van dit onderzoek luiden: "Wat is het effect van Design Decisions in het algoritme op de prestatie van de Inductive Miner?" en "Is er interactie tussen de Task Properties en de Design Decisions van de Inductive Miner?".

Het vervolg van de paper wordt gestructureerd als volgt: in sectie twee zullen relevante termen en concepten toegelicht worden die van belang zijn in dit onderzoek. In dezelfde sectie wordt het Inductive Miner algoritme besproken. De methodologie van dit onderzoek zal worden toegelicht in sectie drie, waarbij de aanpak voor het empirisch analyseren van de prestaties van de Inductive Miner wordt beschreven. De bevindingen

van het onderzoek worden gepresenteerd en besproken in sectie vier. Tot slot wordt in sectie vijf een conclusie gepresenteerd waarin de belangrijkste bevindingen worden samengevat en suggesties voor toekomstig onderzoek worden besproken.

2 Preliminary

Deze sectie biedt een uitleg van relevante termen en concepten en werpt verder ook een blik op de werking van de Inductive Miner.

2.1 Relevante termen en concepten

Het Inductive Miner algoritme heeft verschillende varianten. Deze paper focust zich op het basisalgoritme opgesteld door Leemans, et al. (2013) [8]. Dit algoritme neemt een event log als input en genereert een ontdekt proces als resultaat [3, 7, 10]. Dit resultaat wordt weergeven in de vorm van een Process Tree. Een Process Tree is een visuele manier om een proces weer te geven. Het is het een abstracte weergave van een blocked-structured workflow netwerk in de vorm van een boomstructuur waarin de knooppunten activiteiten of operatoren kunnen vertegenwoordigen [4, 6–9]. De activiteiten weerspiegelen de activiteiten die in het proces uitgevoerd kunnen worden. Een Process Tree beschijft dus stappen of activiteiten in een proces en hoe een uitvoering van het proces kan verlopen. Een voorbeeld van een Process Tree kan geobserveerd worden in figuur 1. Elke activiteit is een knooppunt in de Process Tree. Een Process Tree operator combineert deze knooppunten van het proces en geeft aan hoe de activiteiten van de subtakken worden samengevoegd tot een proces. De Process Tree operatoren worden in figuur 1 weergegeven in het rood. Er bestaan vier soorten operatoren die kunnen voorkomen in een proces: een sequentie operator (SEQUENCE), een keuze operator (CHOICE of XOR), een parallel operator (PARALLEL) en een herhaling (LOOP) [3, 4, 6, 8, 10, 14]. Specifiek voor een Process Tree worden deze operatoren weergegeven als volgt:

- $\rightarrow$: SEQUENCE
- $+$: PARALLEL
- $\times$: CHOICE of XOR
- $\star$: LOOP

Om de vier operatoren verder te verduidelijken wordt het proces van de Process Tree beschreven in figuur 1) als voorbeeld genomen. De notatie voor deze Process Tree is als volgt: $\rightarrow (\rightarrow (A, B), \rightarrow (\times (+ (C, D), E), \star(\rightarrow (F, G), H)))$. Het proces dat beschreven staat in de Process Tree start met activiteit A. Als deze activiteit voltooid wordt, volgt er activiteit B. Na activiteit B kan er gekozen worden om ofwel activiteit E uit te voeren ofwel activiteit C en D in parallel te gaan uitvoeren. Na activiteit E of de parallele uitvoering van activiteit C en D volgt activiteit F. Na activiteit F kan activiteit G uitgevoerd worden. Bij activiteit G wordt er gegaan naar activiteit H en deze weg kan vervolgens opnieuw over activiteit Fen G gaan. Op het punt van activiteit G kan er opnieuw beslist worden om voor de loop te kiezen of niet. Dit is een herhaling die zich steeds zal voordoen totdat gekozen wordt om het proces te beëindigen.

Een sequentie operator illustreert bijvoorbeeld dat activiteit B niet mag starten voordat een activiteit A voltooid is [8, 10]. In procesuitvoeringen kunnen er ook keuzes voorkomen. Er bestaan twee soorten keuzes: een OR-keuze en een XOR-keuze [8, 10]. In deze paper komt enkel de XOR-keuze aan bod. Een XOR-keuze houdt in dat slechts één van de alternatieven kan worden gekozen op een bepaald punt in een proces. Met andere woorden, als één van de alternatieven wordt gekozen, kunnen de andere alternatieven niet worden gevolgd. In het voorbeeld kan activiteit E niet uitgevoerd worden indien het pad van de parallele uitvoering van activiteit C en D gekozen wordt. Zoals in het geval van activiteit C en D, kunnen activiteiten ook in parallel uitgevoerd worden. [8, 10]. In het voorbeeld kan bijvoorbeeld activiteit C en activiteit D op hetzelfde moment of in een willekeurige volgorde uitgevoerd worden. De enige vereiste is dat beide activiteiten kunnen starten wanneer activiteit B wordt afgerond en beide activiteiten voltooid moeten zijn voordat activiteit F voorkomt. De laatste operator is de herhaling [8, 10]. In het lopende voorbeeld kunnen activiteiten F, G en H herhaaldelijk uitgevoerd worden totdat de beslissing wordt genomen om het proces te beëindigen.

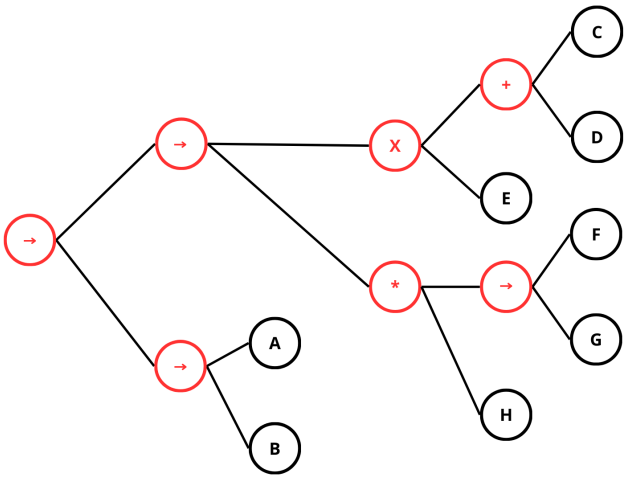


Fig. 1 Process Tree: illustratief voorbeeld

Indien het proces van een Process Tree uitgevoerd wordt, bekomt men een event log. Wanneer een proces uitgevoerd wordt in een bedrijf of organisatie worden deze uitvoeringen geregistreerd in informatiesystemen [10]. Uit die informatiesystemen kan een event log geëxtraheerd worden [2–7]. Een event log is een eindige collectie van procesuitvoeringen, waarbij elke procesuitvoering een opeenvolging van activiteiten van een procesinstantie weerspiegelt. Een voorbeeld van een

event log is: $L = \{(A, B, C, D, F, G)^8, (A, B, C, D, F, G, H, F, G)^3, (A, B, D, C, F, G)^4, (A, B, D, C, F, G, H, F, G), (A, B, E, F, G)^2, (A, B, E, F, G, H, F, G)^3\}$. Deze log wordt beschreven door het proces in de Process Tree van figuur 1 van het lopende voorbeeld. Een opeenvolging van activiteiten van een proces voor een bepaalde procesinstantie wordt een trace genoemd. In het lopende voorbeeld is (A, B, C, D, F, G) een bepaalde trace die acht keer voorkomt in event log L. Een trace bestaat op zijn beurt uit een opeenvolging van events [3, 3, 6–11, 17]. Bijvoorbeeld in trace (A, B, C, D, F, G) kunnen de events A, B, C, D, F en G teruggevonden worden. Events geven meestal het einde van een uitvoering van een activiteit van een bepaalde procesinstantie weer [3, 7, 10, 17]. In figuur 2 worden deze concepten gevisualiseerd.

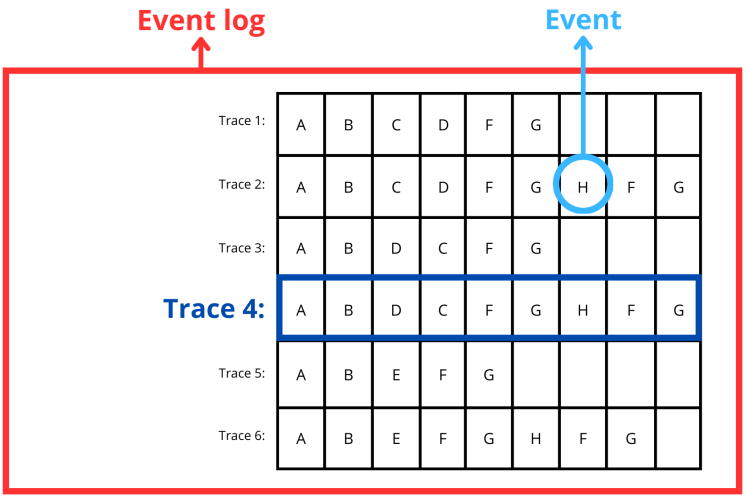


Fig. 2 Visualisatie definitie Event log, Traces en Events

2.2 Inductive Miner

In deze sectie wordt een basisidee van de Inductive Miner gepresenteerd om een globaal begrip van het algoritme te verschaffen, wat helpt bij de interpretatie van de rest van de paper. Voor een volledige beschrijving van het algoritme en de pseudo-code wordt er verwezen naar de originele publicatie van de Inductive Miner [8].

De Inductive Miner volgt een divide-and-conquer aanpak om een Process Tree te ontdekken uit een event log. In grote lijnen werkt deze methode door een initiële event log L recursief te splitsen in sublogs L_1 tot L_n [6–9]. Elke sublog wordt verder opgesplitst totdat er geen splitsingen meer mogelijk zijn of totdat de sublogs uit slechts één activiteit bestaan [6–9]. Op dat moment zijn de sublogs eenvoudig genoeg om

de knooppunten, die de events uit de sublogs representeren, samen te voegen in een Process Tree met de Process Tree operatoren tot een ontdekt model.

Het algoritme begint met het bepalen wanneer twee activiteiten elkaar direct opvolgen in de traces van de event log. Dit wordt voor alle activiteiten gedaan en kan grafisch worden weergegeven in een directly-follows graph (DFG) [8, 9]. Een DFG is een grafische representatie van de event log. In een DFG zijn de knooppunten de activiteiten die geregistreerd zijn geweest in de event log. Een verbinding tussen twee knooppunten komt enkel voor als $activiteit_1$ gevolgd wordt door $activiteit_2$ in de event log. Voor het lopende voorbeeld wordt de DFG voor de gegeven event log $L = \{(A, B, C, D, F, G)^8,\ (A, B, C, D, F, G, H, F, G)^3,\ (A, B, D, C, F, G)^4,\ (A, B, D, C, F, G, H, F, G),\ (A, B, E, F, G)^2,\ (A, B, E, F, G, H, F, G)^3\}$ weergeven in figuur 3. Event log L bevat de volgende geregistreerde traces:

- $(A, B, C, D, F, G)^8$
- $(A, B, C, D, F, G, H, F, G)^3$
- $(A, B, D, C, F, G)^4$
- $(A, B, D, C, F, G, H, F, G)$
- $(A, B, E, F, G)^2$
- $(A, B, E, F, G, H, F, G)^3$

In bovenstaande opsomming geven de exponenten bij de traces het aantal registraties in de event log van de desbetreffende trace weer.

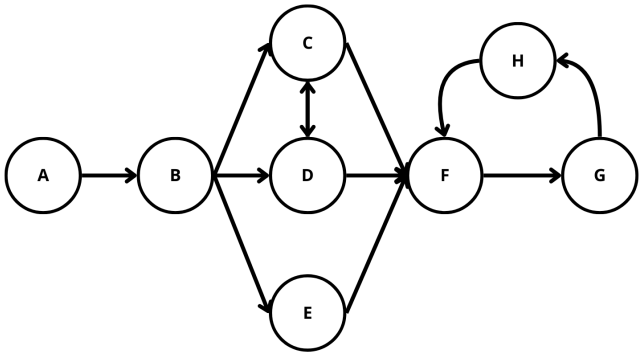


Fig. 3 De DFG voor event log L uit het illustratief voorbeeld

Het Inductive Miner algoritme zal in de DFG zoeken naar constructen die wijzen op een type van de splitsingen en zal vervolgens een van de Process Tree operatoren toewijzen. Het doel van een DFG is met andere woorden, het vinden van constructen die een van de Process Tree operatoren (SEQUENCE, CHOICE, PARALLEL of LOOP) aangeven. Het maken van splitsingen op basis van de DFG in het Inductive Miner algoritme gaat als volgt in zijn werk: voor een initiële log L zal het algoritme event log L splitsen in sublogs $L_1 ... L_n$. Vervolgens worden de splitsingen op een recursieve manier verder gezet tot, zoals hoger reeds vermeld, L niet verder gepslitst kan worden of tot er slechts één activiteit in een sublog zit. Indien er geen splitsingen gevonden worden, wordt er een model teruggegeven dat toelaat dat alle activiteiten na elkaar kunnen voorkomen. Dit valt te vergelijken met een flower model bij petri-netten [7–9].

Volgens Leemans, et al. (2013), zijn er vier soorten splitsingen die kunnen voorkomen in een vaste volgorde in het Inductive Miner algoritme [8]. Deze vaste volgorde waarin de vier splitsingen kunnen voorkomen, wordt vanaf heden aangeduid als de splitsingsvolgorde. De vier mogelijke splitsingen worden geïllustreerd door vier eenvoudige voorbeelden in figuur 4.

- CHOICE-splitsing: Een CHOICE-splitsing wordt gebruikt om beslissingen of keuzes in het proces weer te geven. Als er meerdere paden zijn die het proces kan volgen, afhankelijk van een beslissingspunt, toont de CHOICE-split aan dat er een splitsing aanwezig is waarbij één van de mogelijke paden wordt gekozen. In figuur 4 (a) wordt geobserveerd dat in event log $L = \{(A), (B)\}$, A en B niet samen voorkomen. Dit impliceert dat er wordt gekozen om ofwel A uit te voeren ofwel B, wat wijst op een CHOICE-splitsing en bijgevolg een CHOICE-operator. De CHOICE-splitsing zal event log L opsplitsen in sublogs $L_1 = \{(A)\}$ en $L_2 = \{(B)\}$. Door de events in de sublogs te combineren met de CHOICE-operator ($\times$), kan event log $L = \{\times(A, B)\}$ opnieuw worden gereconstrueerd. Deze splitsing komt overeen met een situatie waarbij de activiteiten boven de stippellijn niet voorkomen in de volgorde onderaan de stippellijn [8].
- SEQUENCE-splitsing: Een SEQUENCE-splitsing wordt gemaakt zoals in figuur 4 (b). Deze splitsing komt overeen met een reeks waarbij de activiteiten links van de stippellijn voorafgaan aan de activiteiten rechts van de stippellijn en gaat dus kijken naar activiteiten die andere activiteiten gaan opvolgen [8]. In de figuur kan geobserveerd worden dat in event log $L = \{(A, B)\}$, B na A volgt. Dit wijst op een SEQUENCE-splitsing en bijgevolg een SEQUENCE-operator ($\rightarrow$. De SEQUENCE-splitsing zal event log L opsplitsen in sublogs $L_1 = \{(A)\}$ en $L_2 = \{(B)\}$. Door de events in de sublogs te combineren met de SEQUENCE-operator ($\rightarrow$), kan event log $L = \{\rightarrow(A, B)\}$ opnieuw worden gereconstrueerd.
- PARALLEL-splitsing: In het eenvoudig voorbeeld wordt de PARALLEL-splitsing gevisualiseerd door figuur 4 (c). De PARALLEL-splitsing identificeert activiteiten die parallel kunnen plaatsvinden in om het even welke volgorde binnen de log en scheidt de event log in verschillende sublogs [8]. De PARALLEL-splitsing zal event log L opsplitsen in sublogs $L_1 = \{(A)\}$ en $L_2 = \{(B)\}$. Door de events in de sublogs te combineren met de PARALLEL-operator ($+$), kan event log $L = \{+(A, B)\}$ opnieuw worden gevormd.

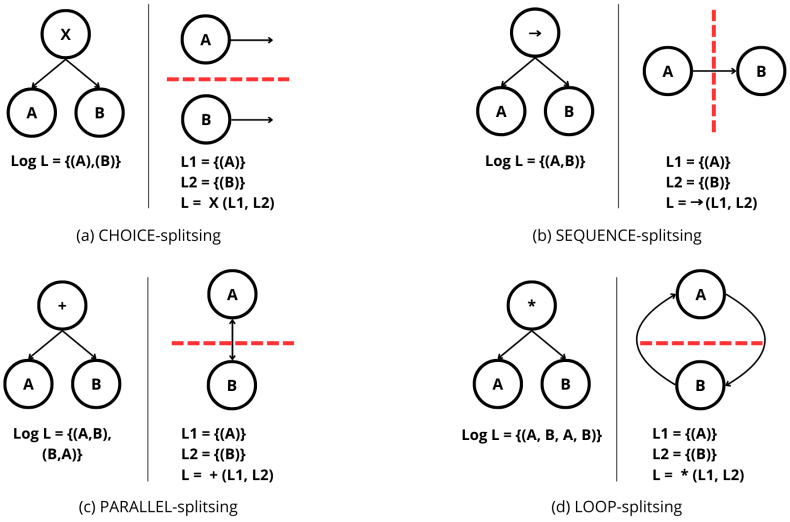


Fig. 4 Voorbeeld Splitsingen

- LOOP-split: Deze soort splitsing zoekt naar patronen waar dezelfde activiteit(en) meerdere keren kunnen voorkomen [8]. In het eenvoudig voorbeeld wordt deze splitsing weergeven in figuur 4 (d). Event log $L = L = \{(A, B, A, B)\}$ zal gesplitst worden in sublogs $L_1 = \{(A)\}$ en $L_2 = \{(B)\}$. Door de events in de sublogs te combineren met de LOOP-operator ($\star$), kan event log $L = \{\star(A, B)\}$ opnieuw worden gevormd.

In figuur 5 worden de vier soorten splitsingen gevisualiseerd voor het lopende voorbeeld waarbij log $L = \{(A, B, C, D, F, G)^8, (A, B, C, D, F, G, H, F, G)^3, (A, B, D, C, F, G)^4, (A, B, D, C, F, G, H, F, G), (A, B, E, F, G)^2, (A, B, E, F, G, H, F, G)^3\}$. In de DFG van het lopende voorbeeld kan er bijvoorbeeld een SEQUENCE-splitsing (figuur 5 (a)) gevonden worden die event log L splitst in sublogs $L_1 = \{(A, B)^{21}\}$ en $L_2 = \{(C, D, F, G)^8, (C, D, F, G, H, F, G)^3, (D, C, F, G)^4, (D, C, F, G, H, F, G), (E, F, G)^2, (E, F, G, H, F, G)^3\}$ Aangezien de Inductive Miner op een recursieve manier te werk gaat, zal het eerst verder gaan met log L_1. In sublog L_1 kan er vervolgens nog een SEQUENCE-splitsing gemaakt worden wat resulteert in sublogs $L_{11} = \{(A)^{21}\}$ en $L_{12} = \{(B)^{21}\}$. De Inductive Miner zal vervolgens verder zoeken naar mogelijke splitsingen in sublog L_2. Op die manier zullen de overige splitsingen in figuur 5 worden ontdekt. Deze splitsingen worden op dezelfde wijze uitgevoerd als het voorbeeld van de SEQUENCE-splitsing.

Indien het Inductive Miner algoritme volledig toegepast wordt op het voorbeeld event log L zal de Process Tree $\rightarrow (\rightarrow (A, B), \rightarrow (\times (+ (C, D), E), \star(\rightarrow (F, G), H)))$ uit figuur 1 ontdekt worden door alle geïdentificeerde sublogs te combineren met

de geïdentificeerde Process Tree operatoren. Omdat de Inductive Miner een gestructureerd model, namelijk een Process Tree, als uitvoer heeft, zal een gegenereerde proces altijd een sound model zijn [3, 7].

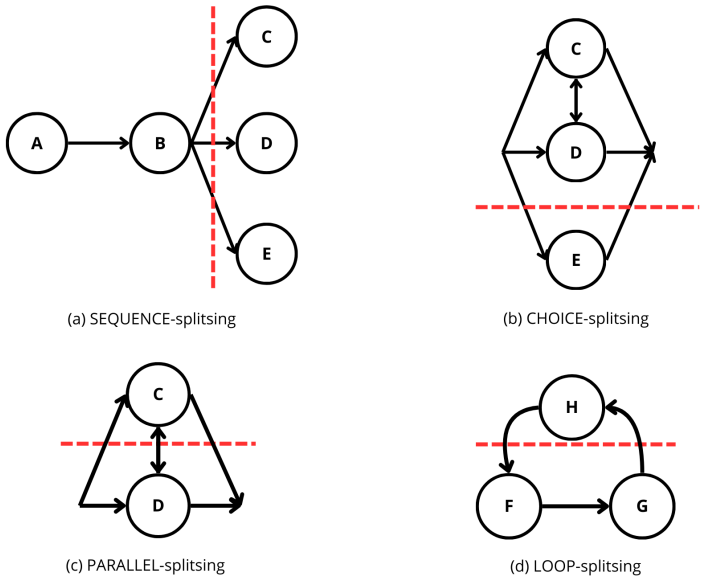


Fig. 5 De vier soorten splitsingen geïllustreerd voor event log L uit het illustratief voorbeeld

3 Methodologie

Deze sectie introduceert de methodologie die in dit onderzoek wordt toegepast, geïnspireerd op het onderzoek van Corstjens, et al. (2019)[13] en Peng, et al. (2021) [3]. Het doel van dit onderzoek is kennis toevoegen aan de body of knowledge van de Inductive Miner. De toegevoegde kennis, verworven door middel van een empirische analyse van de Inductive Miner, betreft Sensitivity Propositions [2]. Sensitivity Propositions doen uitspraken over hoe robuust de prestaties van een algoritme zijn indien er een bepaalde Design Decision aangepast wordt. Het hoofddoel is het analyseren van de impact van de splitsingsvolgorde (Design Decision) in het Inductive Miner algoritme en hoe deze Design Decision interageert met de kenmerken (Task Properties) van een event log waarop de Inductive Miner toegepast wordt. In deze sectie wordt de Design Decision keuze van de splitsingsvolgorde verder uitgediept en worden de Algorithm Tasks aangehaald. Als laatste, wordt het Experimental Design voor dit onderzoek toegelicht.

3.1 Algorithm Design Decisions

Zoals eerder vermeld, behandelt dit onderzoek de Design Decision van het Inductive Miner algoritme betreffende de volgorde van splitsingen in een event log. Zoals vermeld in sectie 2.2, kent het originele algoritme van de Inductive Miner een vaste volgorde voor de vier soorten splitsingen. In deze paper wordt deze volgorde aangeduid als de splitsingsvolgorde. In het originele algoritme van de Inductive Miner is die splitsingsvolgorde als volgt opgesteld: XOR-splitsing, SEQUENCE-splitsing, PARALLEL-splitsing en LOOP-splitsing. Om de impact van deze Design Decision te onderzoeken op de prestaties van de Inductive Miner, werden alle mogelijke combinaties van splitsingsvolgordes bekeken, wat resulteert in 23 varianten ten opzichte van het baseline algoritme. In totaal worden 24 algoritme varianten getest op een set van event logs. De 24 varianten op de splitsingsvolgorde en hun algoritmeID's zijn te vinden in tabel A1 in de appendices.

3.2 Algorithm Tasks Properties

Het startpunt van een Process Discovery algoritme is een event log. Een event log is gebaseerd op een onderliggend proces dat gepaard gaat met bepaalde karakteristieken. Die karakteristieken worden Task Properties genoemd. Task Properties zijn doorgaans niet controleerbaar. In werkelijkheid is er geen controle over welke informatie is opgeslagen in een geëxtraheerde event log. Dit benadrukt het belang van het onderzoek naar de robuustheid van de Inductive Miner en de splitsingsvolgorde. Om dit te onderzoeken, is er voldoende controle over de Task Properties nodig. Daarom is in dit onderzoek gewerkt met artificiële event logs. Door de event logs zelf te genereren, kan er op een bepaalde manier echter wel controle uitgeoefend worden op de kenmerken van de onderliggende proces populatie en op die manier indirect ook op de gegenereerde event logs. Hierdoor kon het effect van verschillende Task Properties van de onderliggende proces populatie op de prestaties van de Inductive Miner op gecontroleerde manier onderzocht worden. De artificiële event logs werden gegenereerd, gebruikmakend van de Process Tree Generator in PM4PY.

De Task Properties van de onderliggende proces populaties werden gedefinieerd door de kansen op het voorkomen van de Process Tree operatoren en het aantal activiteiten te laten variëren. Om het aantal activiteiten te specificeren, kunnen de variabelen MODE, MIN en MAX worden aangepast. MIN en MAX werden respectievelijk ingesteld op 10 en 30. Hierdoor zal het aantal activiteiten van een Process Tree variëren binnen een bereik van 10 tot 30 activiteiten. Het vastzetten van MIN en MAX is doorgevoerd om te voorkomen dat meerdere effecten tegelijkertijd optreden en om ervoor te zorgen dat eventuele causale verbanden kunnen worden toegeschreven aan een specifieke factor, namelijk het meest voorkomende aantal activiteiten (MODE) in de onderliggende populatie. De MODE geeft het aantal activiteiten aan dat gemiddeld in een proces populatie zal voorkomen. Voor deze variabelen zijn drie scenario's gespecificeerd: $15, 20$ en 25. Dit betekent dat een gemiddelde proces populatie naar verwachting ongeveer 15, 20 of 25 activiteiten zal bevatten.

Verder werden karakteristieken bepaald die betrekking hebben op de kans op de Process Tree operatoren zoals de kans op sequenties (SEQUENCE), herhalingen

(LOOPS), keuzes (CHOICE) en parallele activiteiten (PARALLEL). Er werden oorspronkelijke kansen toegekend aan de verschillende operatoren om hun relatieve belang aan te geven. De volgende oorspronkelijke kansen op het voorkomen van de operator (p_i waarbij i = [SEQUENCE, CHOICE, PARALLEL, LOOP]) werden toegekend:

- SEQUENCE (p_1): 9/20
- PARALLEL (p_2): 5/20
- CHOICE (p_3): 5/20
- LOOP (p_4): 1/20

Rekening houdend met bovenstaande oorspronkelijke kansen zou de kans op een CHOICE of PARALLEL operator bijvoorbeeld vijf keer groter zijn in een proces dan de kans op een loop operator. De SEQUENCE operator komt dan het meeste voor in de onderliggende processen ten opzichte van de andere operatoren.

Om een brede selectie van onderliggende processen te garanderen, worden de kansen op de Process Tree operatoren gevarieerd. Dit wordt bereikt door de kans van één Process Tree operator op een bepaalde manier aan te passen (oftewel te manipuleren), terwijl de verhoudingen van de andere drie operatoren constant blijven. Hierbij wordt rekening gehouden met de oorspronkelijk gespecificeerde kansen. De kans op de Process Tree operator die gemanipuleerd zal worden, wordt aangeduid met de letter p'_j waarbij j gelijk is aan de Process Tree operator die gemanipuleerd wordt. p'_j kan de volgende waarden aannemen $0.1, 0.15, 0.25, 0.35, 0.45, 0.55$ en 0.65. Deze waarden zijn gekozen om een breed bereik van lage tot hoge kansen te bieden. Voor de LOOP-operator werd echter besloten om het aantal variaties in waarden te verlagen, aangezien een log met oneindig veel loops niet haalbaar is voor deze analyse. De p'_j voor de LOOP operator bedraagt $0.02, 0.05$ en 0.1.

Voor de hierboven vernoemde waarden van p'_j wordt het resterende percentage $(100 - p'_j)\%$ verdeeld over de andere operatoren, volgens hun oorspronkelijke verhoudingen. Als een operator gemanipuleerd wordt op een vaste waarde p'_j, worden de kansen van de andere operatoren berekend met de volgende formule:

$$p'(i,j) = \frac{p_i}{\sum_{n=1, i \neq j}^{4} p_i} * (1 - p'_j)$$

Hierbij staat p_i voor de oorspronkelijke kans op operator i, p_j voor de oorspronkelijke kans op de gemanipuleerde operator en $p'(i,j)$ voor de nieuwe kans op operator i wanneer operator j gemanipuleerd is en vastgezet wordt op p'_j. Bijvoorbeeld, de formule voor $p'(PARALLEL, SEQUENCE)$ berekent de nieuwe kans op PARALLEL indien SEQUENCE gemanipuleerd wordt op p'_j. De formule zorgt ervoor dat de verhoudingen tussen de niet-gemanipuleerde operatoren gelijk blijven aan de oorspronkelijke kansen.

Ter illustratie wordt SEQUENCE ($= j$) vastgezet op $p'_j = 0.15$. Een waarde van $p'_j = 0.15$ geeft aan dat in het onderliggende proces er 15% kans is dat een SEQUENCE operator voorkomt. Voor de overige Process Tree operatoren blijft er dan nog een kans over van $1 - p'_j = 1 - 0.15 = 0.85$. Deze 85% zal verdeeld worden onder de operatoren

PARALLEL, CHOICE en LOOP naargelang hun onderliggende verhoudingen tussen de oorspronkelijke kansen (respectievelijk (p_2): 5/20, (p_3): 5/20 en (p_4): 1/20). Om te berekenen welke overeenkomstige percentages PARALLEL, CHOICE en LOOP vertegenwoordigen in het onderliggende proces waar SEQUENCE een kans van 15% heeft om voor te komen, worden de formules voor PARALLEL', CHOICE' en LOOP' gebruikt.

$$p'(PARALLEL, SEQUENCE) = \frac{5/20}{5/20 + 5/20 + 1/20} * (1 - 0.15) = 0.39$$

$$p'(CHOICE, SEQUENCE) = \frac{5/20}{5/20 + 5/20 + 1/20} * (1 - 0.15) = 0.39$$

$$p'(LOOP, SEQUENCE) = \frac{1/20}{5/20 + 5/20 + 1/20} * (1 - 0.15) = 0.08$$

Het onderliggende proces in het lopende voorbeeld zal 15% SEQUENCE, 39% PARALLEL en CHOICE en 8% LOOP bevatten. Indien dit mechanisme herhaald wordt voor elke p'_j en voor elke Process Tree operator, worden de combinaties in tabel 1, 2, 3 en 4 bekomen.

Daarnaast zijn de overige kenmerken van de Process Tree Generator van PM4PY constant gehouden aan de standaardwaarden van PM4PY. Dit vertaalt zich als volgt:

- DUPLICATES = 0
- LT_DEPENDENCY = 0
- INFREQUENT = 0.25
- NO_MODELS = 10
- UNFOLD = 0
- MAX_REPEAT = 0

Door p'_j voor elke Process Tree operator te manipuleren, zijn er dan zeven scenario's voor SEQUENCE, PARALLEL en CHOICE operatoren en drie scenario's voor de LOOP operator. Dit resulteert in een subtotaal van 24 onderliggende proces populaties. Daarnaast werd eveneens het scenario met de standaardwaarden van PM4PY opgenomen in de analyse. In dat scenario heeft elke Process Tree operator een gelijke kans (0.25). Het subtotaal bedraagt dan 25 onderliggende proces populaties.

Wanneer het aantal scenario's van de Process Tree operatoren (25) vermenigvuldigd wordt met het aantal scenario's van het aantal activiteiten, wordt er een totaal van 75 gegenereerde event logs bekomen. Dit aantal event logs vertegenwoordigt een brede basis aan processen. Dit is cruciaal omdat er weinig bekend is in de literatuur over echte bedrijfsprocessen, daarom is ervoor gekozen om verschillende ranges van onderliggende processen te bestuderen. Daarnaast is het genereren van een brede basis aan processen nodig om de interacties tussen de Task Properties en de Design Decisions te gaan onderzoeken. Een samenvatting van de gegenereerde event logs met bijhorende Task Properties wordt gepresenteerd in B2 in de appendices.

Sequence	Parallel	Choice	Loop
0.1	0.41	0.41	0.08
0.15	0.39	0.39	0.08
0.25	0.34	0.34	0.07
0.35	0.30	0.30	0.06
0.45	0.25	0.25	0.05
0.55	0.20	0.20	0.04
0.65	0.16	0.16	0.03

Table 1 De kans op een SEQUENCE-operator gemanipuleerd: PARALLEL, CHOICE en LOOP berekend aan de hand van de formules

Sequence	Parallel	Choice	Loop
0.54	**0.1**	0.30	0.06
0.51	**0.15**	0.28	0.06
0.45	**0.25**	0.25	0.05
0.39	**0.35**	0.22	0.04
0.36	**0.45**	0.20	0.04
0.30	**0.55**	0.17	0.03
0.21	**0.65**	0.12	0.02

Table 2 De kans op een PARALLEL-operator gemanipuleerd: SEQUENCE, CHOICE en LOOP berekend aan de hand van de formules

Sequence	Parallel	Choice	Loop
0.54	0.30	**0.1**	0.06
0.51	0.28	**0.15**	0.06
0.45	0.25	**0.25**	0.05
0.39	0.22	**0.35**	0.04
0.36	0.20	**0.45**	0.04
0.30	0.17	**0.55**	0.03
0.21	0.12	**0.65**	0.02

Table 3 De kans op een CHOICE-operator gemanipuleerd: SEQUENCE, PARALLEL en LOOP berekend aan de hand van de formules

3.3 Kwaliteitsmaatstaven

Na het ontdekken van een procesmodel binnen Process Mining kunnen kwaliteitsmaatstaven berekend worden door de overeenstemming tussen het ontdekte model en de werkelijke uitvoering van het proces (de event log) te evalueren. Er bestaan vier dimensies binnen de kwaliteitsmaatstaven: fitness, precision, generalisation en simplicity [3, 4, 7, 8, 10–12, 17]. De fitness dimensie meet hoe goed een model het gedrag in een event log verklaart, met een waarde tussen 0 en 1 [3, 4, 7, 8, 10–12, 17, 18].. Een perfecte fitness is 1, waarbij alle traces kunnen worden afgespeeld [3, 4, 7, 8, 10–12, 17]. Zoals hoger reeds aangehaald, garandeert de Inductive Miner een sound model

Sequence	Parallel	Choice	Loop
0.46	0.26	0.26	**0.02**
0.45	0.25	0.25	**0.05**
0.43	0.24	0.24	**0.1**

Table 4 De kans op een LOOP-operator gemanipuleerd: SEQUENCE, PARALLEL en CHOICE berekend aan de hand van de formules

dat de volledige event log kan afspelen op het ontdekte proces. Met andere woorden, de Inductive Miner garandeert een fitness die de waarde 1 benadert. [3, 7]. Er zijn twee methoden om de fitnessmaatstaf te berekenen: de token-based replay-methode [19] en de alignment-based methode. De alignment-based methode kost veel tijd waardoor het mogelijk is dat bij bepaalde observaties geen fitnessmaatstaf kan worden berekend, wat leidt tot gecensureerde data. Daarom wordt de voorkeur gegeven aan de token-based replay-methode, ondanks dat deze niet deterministisch is. In dit onderzoek worden volledige statistisch correcte data geprefereerd om juiste conclusies te kunnen trekken. De tegenhanger van fitness is precision. Precision meet of het gedrag van het model ook in de event log voorkomt, met een perfecte waarde van 1 als elke trace in het model in de log aanwezig is [3, 4, 7, 8, 10–12, 17, 18]. Deze maatstaf werd berekend gebruik makend van de ETConformance methode die eveneens gebruik maakt van de token-based replay [20]. Generalisation meet in welke mate een model traces bevat die niet in de event log voorkomen maar wel toegelaten zijn in het proces dat gegenereerd werd door het algoritme [3, 4, 7, 8, 10–12, 17]. Als laatste, is er de dimensie van simplicity. Simplicity stelt dat het eenvoudigste model dat het gedrag weerspiegelt, het beste is. Deze kwaliteitsmaatstaf staat eveneens bekend als Occam's Razor [7, 10]. PM4PY berekent generalisation en simplicity door de methode toe te passen die wordt beschreven in respectievelijk de volgende artikels: [21] en [22].

3.4 Experimental Design

Om een antwoord te bieden op de onderzoeksvragen wordt er gebruik gemaakt van een repeated measures design. Dit ontwerp houdt in dat alle varianten van het algoritme worden toegepast op elke event log, waarbij elke log werd gekenmerkt door specifieke Task Properties. Op deze manier hebben we herhaalde metingen verkregen van de prestaties van de algoritmes binnen dezelfde set event logs. De 24 varianten van het Inductive Miner algoritme uit sectie 3.1 zijn toegepast op de 75 gegenereerde event logs uit sectie 3.2, wat resulteerde in totaal 1800 observaties. Vervolgens worden de prestatieresultaten geanalyseerd door middel van de Friedman-test [23]. Op basis van die test kunnen we de relaties tussen de prestaties, Design Decision en Task Properties gaan onderzoeken en interpreteren. Voor de empirische analyse en het opzetten van de experimenten werd PM4PY gebruikt, een Python-pakket voor Process Mining, dat benaderingen biedt voor Process Discovery en Conformance Checking [15, 16]. PM4PY werd gebruikt voor zowel het genereren van event logs en het toepassen van het Inductive Miner algoritme als voor het berekenen van de kwaliteitsmaatstaven.

Voor de eerste onderzoeksvraag werden vier experimenten uitgevoerd (één voor elke kwaliteitsmaatstaf) om te bepalen of de splitsingsvolgorde de prestaties van de Inductive Miner beïnvloedt. De nulhypothese (H_0) voor experiment 1-4 is "De splitsingsvolgorde van de Inductive Miner heeft geen invloed op kwaliteitsmaatstaf j van het ontdekte model ten opzichte van de event log.". Deze H_0 wordt opgesteld voor j = [Fitness, Precision, Generalisation, Simplicity]. Als de nulhypothese niet wordt verworpen, impliceert dit dat er geen duidelijke rangorde is voor de kwaliteitsmaatstaven over alle event logs en dat er geen significant verschil is tussen de verschillende algoritme varianten wat betreft fitness, precision, generalisation en simplicity. Dit betekent echter niet dat er geen mogelijke verschillen zouden kunnen worden vastgesteld als de event logs zouden worden onderverdeeld op basis van specifieke kenmerken (Task Properties). Mogelijk zou er dan wel een effect kunnen worden waargenomen.

Om dit effect verder te onderzoeken, werden aanvullende experimenten uitgevoerd om de robuustheid van de Inductive Miner bloot te leggen en zijn met name bedoeld om de tweede onderzoeksvraag te beantwoorden, waarbij de nadruk ligt op het bestuderen van de robuustheid van de Inductive Miner ten opzichte van de splitsingsvolgorde en de kansverdeling van verschillende operatoren in het onderliggende proces (Task Properties). Er zijn in totaal 32 aanvullende experimenten uitgevoerd (twee testen voor elke Process Tree operator en voor elke kwaliteitsmaatstaf) om de kwaliteitsmaatstaven te vergelijken tussen datasets met verschillende hoeveelheden kans op een bepaalde Process Tree operator. Bij deze experimenten werd gekeken of de locatie van een bepaalde splitsing in de splitsingsvolgorde een impact heeft op een kwaliteitsmaatstaf van de ontdekte procesmodellen waarvan de event log afkomstig was van een proces met een groot of klein aandeel van de operator. De locatie van een bepaalde splitsing kon zich vooraan of achteraan in de splitsingsvolgorde bevinden. De experimenten werden uitgevoerd voor de vier kwaliteitsmaatstaven en voor zowel een groot als klein aandeel van de Process Tree operator in de event log. Bijvoorbeeld voor de SEQUENCE-splitsing werd getest of de locatie van de SEQUENCE-splitsing een impact heeft op de fitness van de ontdekte procesmodellen waarvoor de event log afkomstig was van een proces met een hoog aandeel op sequentiële relaties (en dus een grote kans op de SEQUENCE-operator). De hypothesen (H_0) voor experiment 5-36 luiden: "De locatie in de splitsingsvolgorde van de i-splitsing heeft geen impact op de kwaliteitsmaatstaf j voor de ontdekte procesmodellen van event logs die afkomstig zijn van processen met een hoge kans op operator i.". Deze H_0 wordt opgesteld voor i = [SEQENCE, CHOICE, PARALLEL, LOOP] en voor j = [Fitness, Precision, Generalisation, Simplicity]. Om te bepalen of een event log een hoge of lage kans heeft op de operator SEQUENCE, CHOICE, PARALLEL of LOOP, werden de volgende drempelwaarden gehanteerd: respectievelijk 0.45, 0.25, 0.25 en 0.05.

De statistische analyse van de resultaten voor ale experimenten (experiment 1-36) werd uitgevoerd met de Friedman-test en de post-hoc Nemenyi-test, gebruik makend van de aanpak uit het onderzoek van Demsar, et al. (2006) [23]. De Friedman-test is een niet-parametrische ANOVA-test. Deze test wordt gebruikt om verschillen tussen meerdere gerelateerde groepen te detecteren op basis van hun gemiddelde rangordes. De analyse werd uitgevoerd op een significantieniveau van 95%, wat overeenkomt met een α-waarde van 0.05. Als er een significant verschil werd waargenomen in de

kwaliteitsmaatstaven, werd de post-hoc Nemenyi-test uitgevoerd om te bepalen waar precies de verschillen liggen.

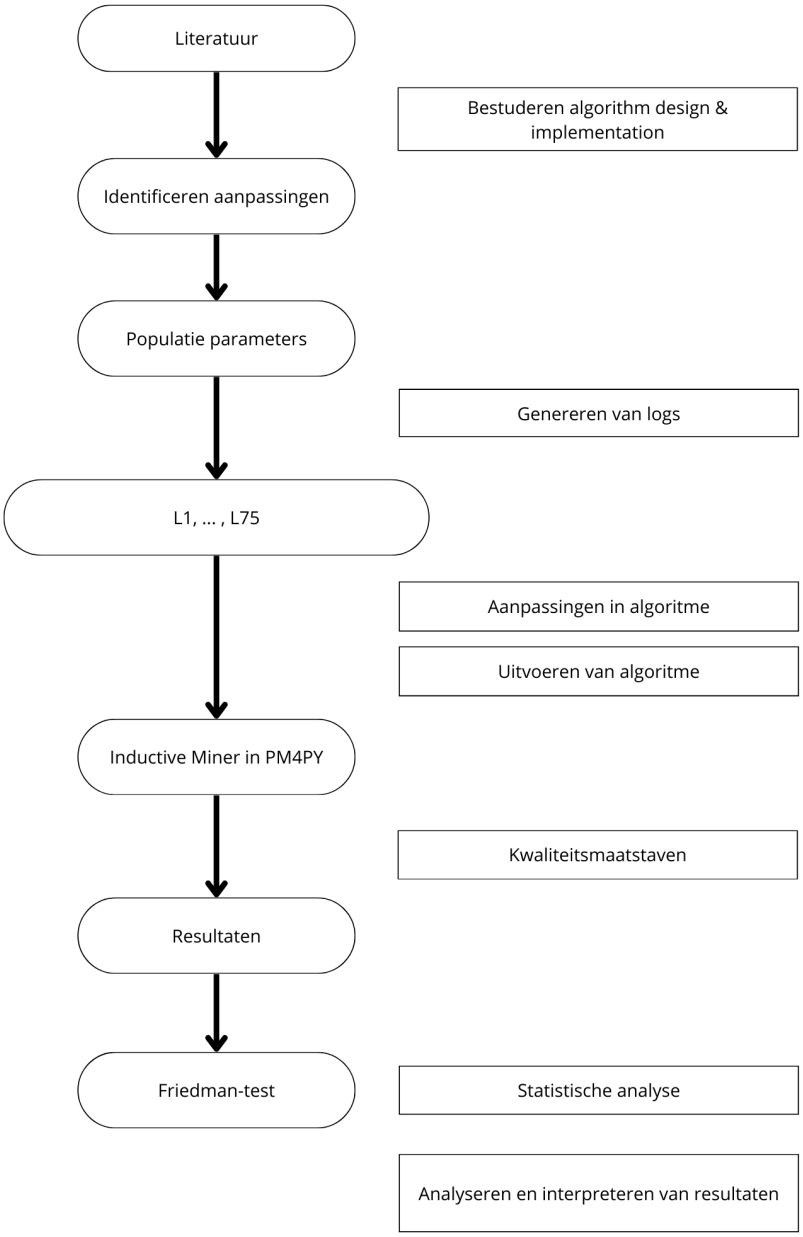


Fig. 6 Overzicht Methodologie

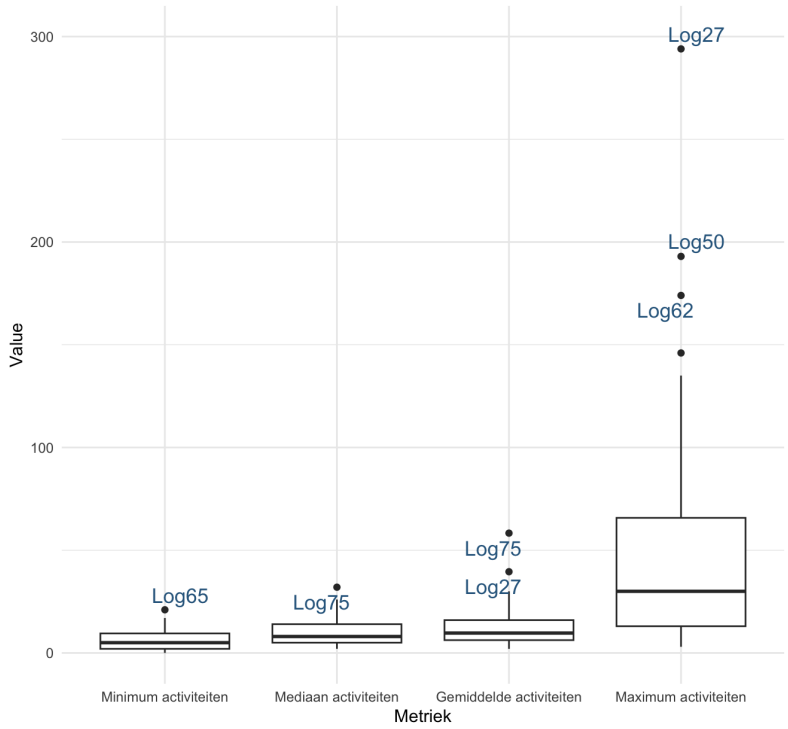


Fig. 7 Boxplot descriptieve metrieken event logs

4 Resultaten en discussie

Deze sectie bespreekt de resultaten van de experimenten. Allereerst wordt in sectie 5.1 een descriptieve analyse van de gegenereerde event logs besproken. Verder worden de behaalde resultaten met betrekking tot de kwaliteitsmaatstaven gepresenteerd in sectie 5.2. In sectie 5.3 en 5.4 worden de resultaten in verband met de experimenten toegelicht.

4.1 Event logs

De 75 gegenereede event logs vormen de basis voor de datasets die gebruikt werden in dit onderzoek. De event logs bevatten elk 1000 traces. Binnen die traces zitten er In tabel C3 kan er per gegenereerde event log het minimum en maximum aantal events in de traces van een event log geobserveerd worden. Eveneens wordt het gemiddelde en de mediaan in diezelfde tabel getoond. Over de event logs heen zitten er gemiddeld gezien 12.19 events. Het minimum en maximum bedraagt 0 en 741. De mediaan bedraagt 8 events over alle logs heen. De spreiding van de events in de traces over de logs heen kan geobserveerd worden in grafiek 7. Voor de maximum waarde viel er één waarde van de grafiek om de leesbaarheid te garanderen. Dit was voor log_75, hierbij bedroeg deze uitschieter een maximum van 741 activiteiten in een trace. De andere opvallende uitschieters worden getoond met een blauwe kleur in de grafiek.

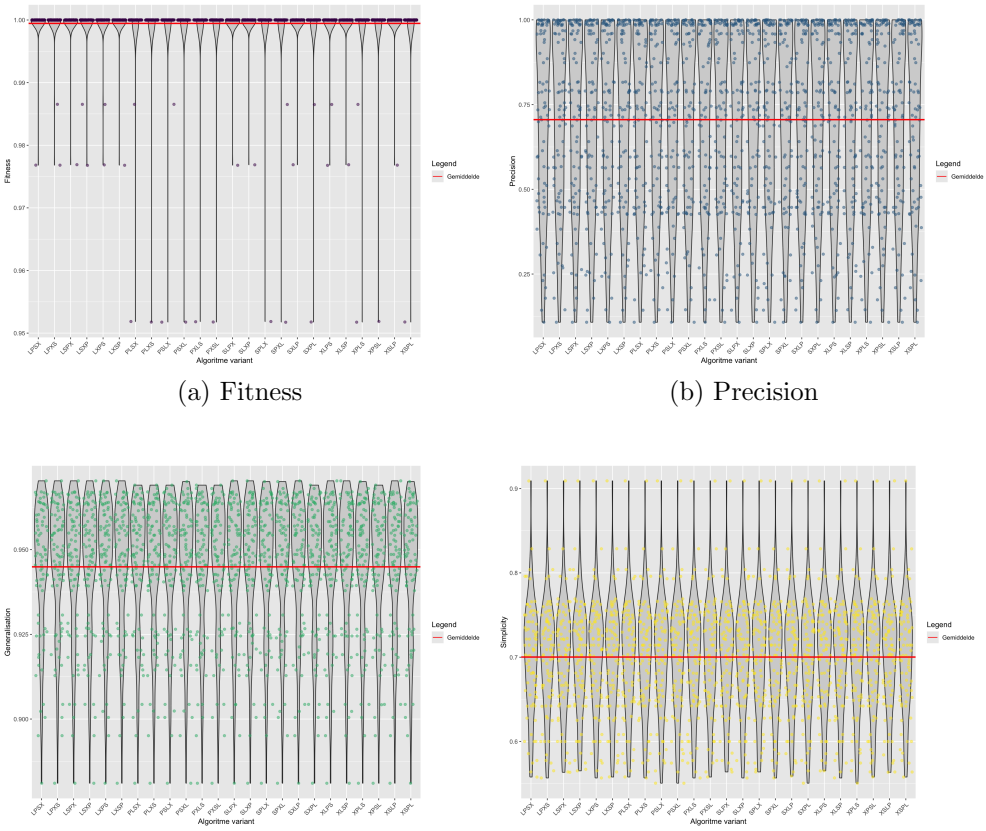
4.2 Kwaliteitsmaatstaven

Zoals beschreven in de literatuur, kan er gevonden worden dat de Inductive Miner in het basisscenario een fitness bereikt die 1 benadert. De resulaten tonen dat een fitness van 1 eveneens gegarandeerd kan worden indien de volgorde van de splitsingen veranderd wordt. Voor elk algoritme variant benadert of is de fitness maatstaf gelijk aan 1. Dit gedrag kan geobserveerd worden in tabel D4, tabel D8 en in figuur 8 (a). Enkel log_55 en log_56 naderen niet altijd een maximale fitness zoals eerder aangehaald in vorige sectie. Deze logs bedragen een gemiddelde fitness van respectievelijk 0.9949431 en 0.9643110. Niet alleen de fitness, maar ook de precision, generalisation en simplicity liggen sterk in dezelfde bereiken voor de algoritme varianten. In D9, D10 worden de gemiddelde, maximum- en minimumwaarden voor deze kwaliteitsmaatstaven weergegeven per algoritme variant. Binnen de bereiken van kwaliteitsmaatstaven is er echter een grote spreiding van precision, generalisation en simplicity. In figuur 8 kunnen de waardes voor respectievelijk fitness, precision, generalisation en simplicity eveneens waargenomen worden. Bij het bekijken van de minima en maxima van de kwaliteitsmaatstaven per algoritme variant kan worden geconcludeerd dat deze waarden ook sterk bij elkaar liggen. Voor de fitnesswaarden varieert het minimum tussen 0.9517577 en 0.9768732. Het maximum voor alle algoritme varianten voor de fitnessmaatstaf is 1. Voor de precision maatstaf wordt voor elk algoritme variant een minimum van 0.108 bereikt en een maximum van 1. Bij het bekijken van generalisation kan worden opgemerkt dat het minimum 0.8810528 bedraagt en het maximum varieert tussen 0.9690357 en 0.9702976. Voor simplicity varieert het minimum voor alle algoritme varianten tussen 0.5505618 en 0.5652174. Het maximum voor simplicity bedraagt voor alle algoritme varianten 0.9090909.

Voor de kwaliteitsmaatstaven per log kunnen tabel D4, D5, D6 en D7 geraadpleegd worden. Voor de fitness maatstaf had log_56 voor alle varianten de laagste fitness waarde. Log_55 had eveneens de laagste fitnesswaarde voor negen varianten (SPXL, SXPL, XLPS, XPLS, LSXP, LPXS, LXPS, PSLX en PLSX). Alle andere event logs hadden een waarde van 1 voor de fitness maatstaf. Voor de precision waarden hadden log_21, log_32, log_42 en log_44 zowel de laagste als de hoogste fitness waardes voor alle varianten. De andere event logs schommelden tussen deze waardes. De hoogste generalisation waarden werden bereikt door log_75 voor twaalf varianten (SLPX, SLXP, SXLP, XSLP, XLPS, XLSP, LSPX, LSXP, LPSX, LPXS, LXPS, LXSP). Log_46 realiseerde voor elke variant de laagste generalisation waarde. In termen van simplicity behaalde log_32 de hoogste simplicity en log_66 voor de varianten SXPL, XPLS, PSXL, PSLX.

4.3 Verandering in splitsingsvolgorde

Deze reeks experimenten werd uitgevoerd om te bepalen of er een significant verschil bestaat tussen de algoritme varianten waarbij de splitsingvolgorde is gewijzigd, voor vier verschillende kwaliteitsmaatstaven. Hetzelfde experiment werd dus herhaald voor elk van de vier maatstaven. Om eventuele significante verschillen in de waarden van de kwaliteitsmaatstaven voor alle algoritme varianten vast te stellen, werd de Friedman-test uitgevoerd. Dit onderzoek hanteert een significantieniveau van 95%, wat



(a) Fitness (b) Precision

(c) Generalisation (d) Simplicity

Fig. 8 Spreiding van de kwaliteitsmaatstaven voor de algoritme varianten

overeenkomt met een α-waarde van 0.05. Voor de Friedman-testen werden de Friedman Statistic (χ_F^2), de Iman and Davenport Statistic (FF), F-waarde (F-stat) en het kritische verschil (CD) berekend met de formules uit [23]. Deze waarden kunnen geobserveerd worden voor elke test in tabel 6. De CD voor een significantieniveau van 95% bedroeg 6.30 voor elke test. De F-waarde bedroeg 1.54 voor een signficantieniveau van 95% voor 23 en 1702 vrijheidsgraden. Indien de waarde voor de Iman en Davenport statistic groter is dan de kritische F-waarde (FF $>$ F-stat), wordt de nulhypothese verworpen op een significantieniveau van 95%. Dit zou betekenen dat er een significant verschil is tussen de verschillende algoritme varianten wat betreft kwaliteitsmaatstaven en dus prestaties.

De gemiddelde rang van de algoritme varianten per Friedman-test kunnen geobserveerd worden in figuur 9, 10, 11 en 12. Uit de resultaten van de Friedman-tests (in tabel 6) blijkt dat de verkregen kritische F-waarde groter is dan de Iman en Davenport-statistiek voor alle kwaliteitsmaatstaven. Dit suggereert dat er onvoldoende bewijs is om te concluderen dat er significante verschillen zijn tussen de algoritme varianten

wat betreft de kwaliteitsmaatstaven op een significantieniveau van 95%. Hierdoor kan de nulhypothese voor het experiment van fitness, precision, generalisation en simplicity niet worden verworpen. Deze nulhypothese stelde voor elk experiment dat er geen significante verschillen zijn tussen de verschillende algoritme varianten in termen van prestaties, gebaseerd op de verzamelde en geanalyseerde data. Dit impliceert dat er geen duidelijke rangorde is voor de kwaliteitsmaatstaven over alle event logs en dat er geen significant verschil is tussen de verschillende algoritme varianten wat betreft fitness, precision, generalisation en simplicity.

Friedman-test	χ_F^2	**FF**	**F-stat**	**F-stat<FF**
Experiment 1: Log fitness	−1886.03	−38.65	1.54	×
Experiment 2: Precision	−1872.81	−38.52	1.54	×
Experiment 3: Generalisation	−1964.81	−39.40	1.54	×
Experiment 4: Simplicity	−1921.20	−38.99	1.54	×

Table 5 Overzicht: output Friedman-testen (experiment 1-4)

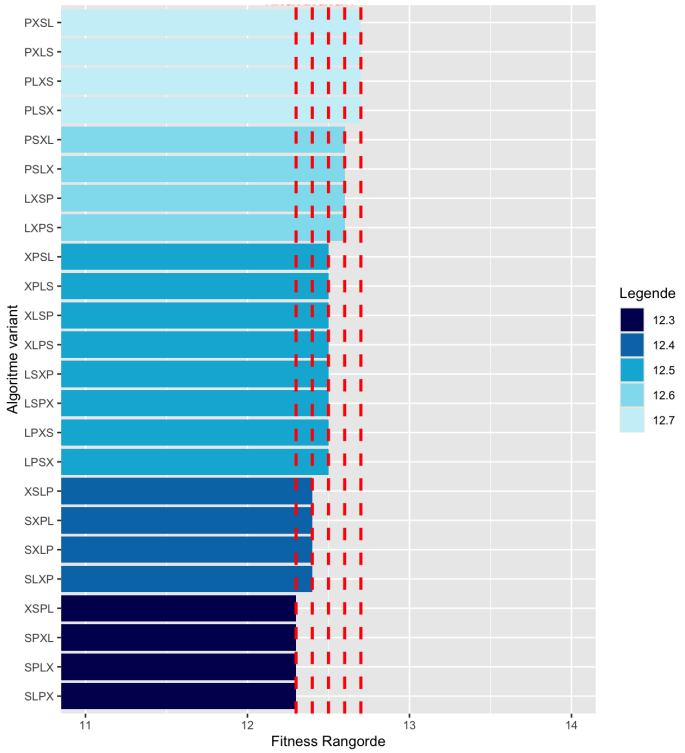


Fig. 9 Fitness Friedman-test: Gemiddelde rangordes van algoritme varianten

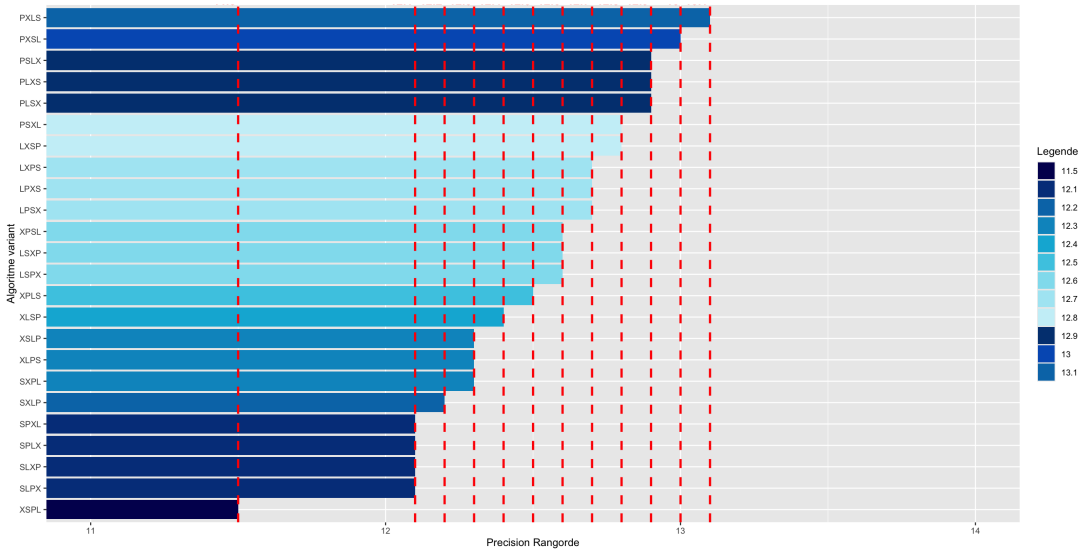


Fig. 10 Precision Friedman-test: Gemiddelde rangordes van algoritme varianten

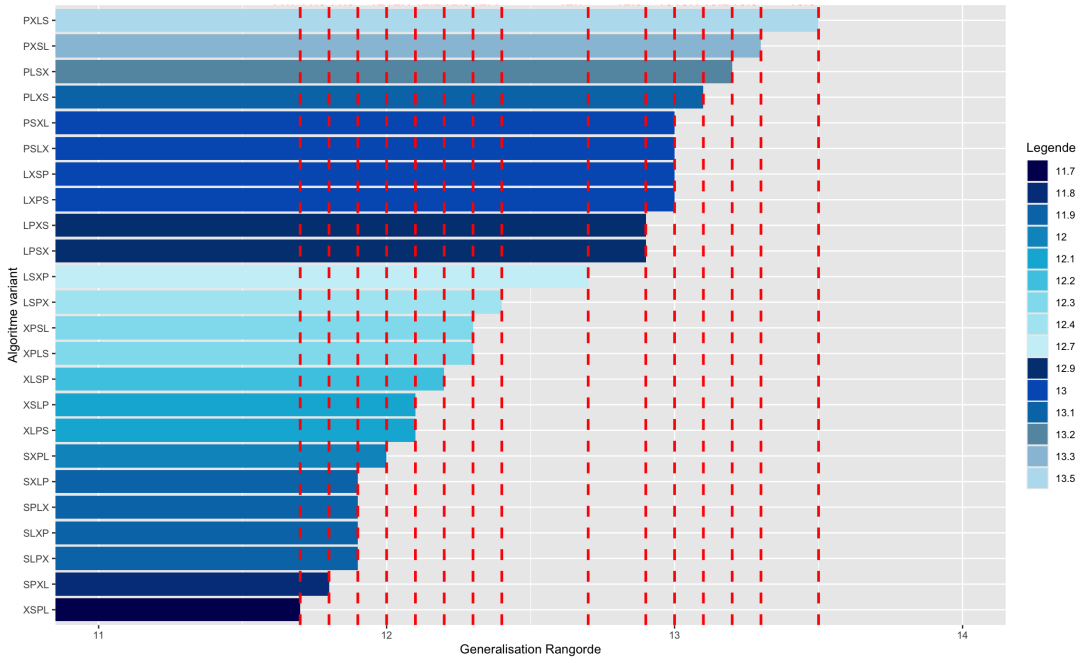


Fig. 11 Generalisation Friedman-test: Gemiddelde rangordes van algoritme varianten

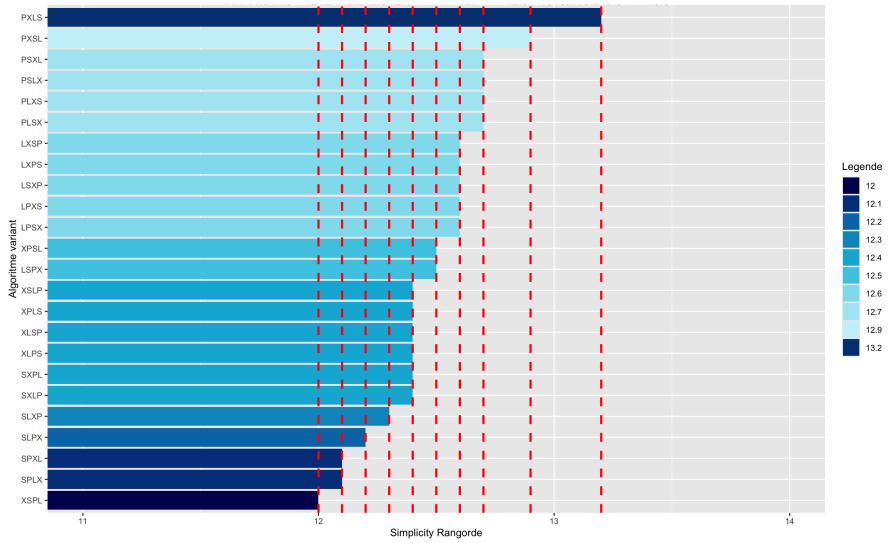


Fig. 12 Simplicity Friedman-test: Gemiddelde rangordes van algoritme varianten

4.3.1 Robuustheid: impact van verandering splitsingsvolgorde en Task Properties

De resultaten van de 32 experimenten, die gericht waren op het onderzoeken van de effecten van zowel veranderingen in splitsingsvolgorde als Task Properties op de kwaliteitsmaatstaven van de ontdekte modellen, werden eveneens geanalyseerd met behulp van de Friedman-test. Dit onderzoek hanteerde een significantieniveau van 95%, met een α-waarde van 0.05.

De berekende Friedman Statistic (χ_F^2), Iman and Davenport Statistic (FF), F-waarde (F-stat), en het kritische verschil (CD) werden afgeleid van de formules zoals beschreven in de literatuur [23]. Deze waarden zijn voor elk experiment samengevat in tabel 6. Daarin worden tevens het aantal logs (n) en algoritme varianten (p) per experiment weergegeven.

De resultaten van alle experimenten leidden tot het verwerpen van de nulhypothese H_0 op een significantieniveau van 95% (FF $>$ F-stat). Dit impliceert dat, ongeacht de specifieke kenmerken van de onderliggende processen, de kwaliteit van de ontdekte processen niet significant varieert. Dit wijst op de robuustheid van de Inductive Miner, die consistent presteert onder diverse omstandigheden. Deze bevinding is van groot belang voor het gebruik van de Inductive Miner in praktijksituaties, waarbij er geen controle is over de kenmerken van de event log en het onderliggende proces. Het suggereert dat de Inductive Miner kan omgaan met verschillende contexten zonder dat de prestaties of kwaliteit van de ontdekte processen daaronder lijden. Hierdoor kan de Inductive Miner worden ingezet voor het ontdekken van uiteenlopende types onderliggende processen en event logs.

Friedman-test	**n**	**p**	**CD**	χ_F^2	**FF**	**F-stat**	**F-stat<FF**
Experiment 5: Veel SEQUENCE Fitness	33	12	5.44	0.69	0.06	1.82	×
Experiment 6: Veel SEQUENCE Precision	33	12	5.44	1.797	0.159	1.815	×
Experiment 7: Veel SEQUENCE Generalisation	33	12	5.44	1.797	0.159	1.815	×
Experiment 8: Veel SEQUENCE Simplicity	33	12	5.44	1.082	0.096	1.815	×
Experiment 9: Weinig SEQUENCE Fitness	42	12	5.338	0	0	1.809	×
Experiment 10: Weinig SEQUENCE Precision	42	12	5.338	1.456	0.130	1.809	×
Experiment 11: Weinig SEQUENCE Generalisation	42	12	5.338	5.073	0.455	1.809	×
Experiment 12: Weinig SEQUENCE Simplicity	42	12	5.338	1.267	0.113	1.809	×
Experiment 13: Veel CHOICE Fitness	48	12	5.288	0.154	0.014	1.807	×
Experiment 14 Veel CHOICE Precision	48	12	5.288	1.054	0.094	1.807	×
Experiment 15: Veel CHOICE Generalisation	48	12	5.288	4,508	0.405	1.807	×
Experiment 16: Veel CHOICE Simplicity	48	12	5.288	3.958	0.355	1.807	×
Experiment 17: Weinig CHOICE Fitness	27	12	5.533	0.380	0.033	1.822	×
Experiment 18: Weinig CHOICE Precision	27	12	5.533	0.380	0.0333	1.822	×
Experiment 19: Weinig CHOICE Generalisation	27	12	5.533	2.594	0.229	1.822	×
Experiment 20: Weinig CHOICE Simplicity	27	12	5.533	1.254	0.110	1.822	×
Experiment 21: Veel PARALLEL Fitness	48	12	5.288	0,130	0.012	1.807	×
Experiment 22: Veel PARALLEL Precision	48	12	5.288	2.594	0.232	1.807	×
Experiment 23: Veel PARALLEL Generalisation	48	12	5.288	7,498	0.677	1.807	×
Experiment 24: Veel PARALLEL Simplicity	48	12	5.288	2.697	0.241	1.807	×
Experiment 25: Weinig PARALLEL Fitness	27	12	5.533	0,380	0.0333	1.822	×
Experiment 26: Weinig PARALLEL Precision	27	12	5.533	2.625	0.232	1.822	×
Experiment 27: Weinig PARALLEL Generalisation	27	12	5.533	1.070	0.094	1.822	×
Experiment 28: Weinig PARALLEL Simplicity	27	12	5.533	2.650	0.200	1.822	×
Experiment 29: Veel LOOP Fitness	42	12	5.339	0.192	0.017	1.810	×
Experiment 30: Veel LOOP Precision	42	12	5.339	2.418	0.216	1.810	×
Experiment 31: Veel LOOP Generalisation	42	12	5.339	11.349	1.033	1.810	×
Experiment 32: Veel LOOP Simplicity	42	12	5.339	5.451	0.489	1.810	×
Experiment 33: Weinig LOOP Fitness	33	12	5.440	0.297	0.026	1.816	×
Experiment 34: Weinig LOOP Precision	33	12	5.440	1.108	0.098	1.816	×
Experiment 35: Weinig LOOP Generalisation	33	12	5.440	0.550	0.049	1.816	×
Experiment 36: Weinig LOOP Simplicity	33	12	5.440	2,068	0.186	1.816	×

Table 6 Overzicht: output Friedman-testen (experiment 5-32)

5 Conclusie

Op basis van de bestaande literatuur werd een belangrijke research gap geïdentificeerd: het is niet alleen cruciaal om te weten of een algoritme goed werkt, maar ook waarom het goed werkt en welke factoren hiervoor verantwoordelijk zijn. Competitive testing biedt waardevolle inzichten door aan te tonen welke algoritmes superieur zijn ten opzichte van andere. Uit dergelijk onderzoek blijkt bijvoorbeeld dat de Inductive Miner een effectief algoritme is voor Process Discovery, zoals aangetoond door Augusto et al. (2019) [11]. Dit onderzoek richtte zich echter op een diepgaandere analyse van wat de Inductive Miner tot een goed presterend algoritme maakt en waarom dat zo is.

Een specifiek aspect dat werd onderzocht, was de volgorde waarin splitsingen in het Inductive Miner algoritme worden gemaakt. De bevindingen suggereren met 95% zekerheid dat deze volgorde geen significant verschil maakt voor de kwaliteitsmaatstaven fitness, precision, generalisation en simplicity.

Verder wijzen de resultaten erop dat de Inductive Miner een robuust algoritme is dat consistent presteert onder verschillende contexten (Task Properties), zonder dat dit ten koste gaat van de kwaliteit van de ontdekte processen. Dit is van cruciaal belang voor het praktische gebruik van de Inductive Miner, vooral omdat in reële situaties de kenmerken van de event log en het onderliggende proces niet onder controle zijn. Hierdoor kan de Inductive Miner effectief worden ingezet voor het ontdekken van diverse typen onderliggende processen en event logs.

Daarnaast benadrukt dit onderzoek de noodzaak van toekomstig onderzoek naar andere ontwerpkeuzes van de Inductive Miner. Dit moet niet alleen beperkt blijven tot de Inductive Miner, maar moet ook worden uitgebreid naar andere Process Discovery algoritmes. Deze benadering zal niet alleen het huidige begrip van Process Discovery-algoritmes verdiepen, maar zal ook de ontwikkeling van geavanceerde en efficiënte Process Discovery-algoritmes bevorderen. Bovendien zal het inzicht bieden in hoe deze algoritmes presteren in praktijktoepassingen.

References

[1] J. C. A. M. Buijs, B. F. van Dongen, W. M. P. van der Aalst: A genetic algorithm for discovering process trees. In: 2012 IEEE Congress on Evolutionary Computation, pp. 1–8 (2012). https://doi.org/10.1109/CEC.2012.6256458 . Journal Abbreviation: 2012 IEEE Congress on Evolutionary Computation

[2] Mendling, J., Depaire, B., Leopold, H.: Theory and Practice of Algorithm Engineering (2021) https://doi.org/10.48550/arXiv.2107.10675 . arXiv:2107.10675 [cs]. Accessed 2023-10-10

[3] Peng, W., Zhang, Z., Hildebrant, R., Ren, S.: Empirical Studies of Three Commonly Used Process Mining Algorithms. 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2492–2499 (2021) https://doi.org/10.1109/SMC52423.2021.9658861

[4] Buijs, J., Dongen, B., Aalst, W.: On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery vol. 7565, (2012). https://doi.org/10.1007/978-3-642-33606-5_19 . Pages: 322

[5] De Weerdt, J., De Backer, M., Vanthienen, J., Baesens, B.: A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. Information Systems **37**(7), 654–676 (2012) https://doi.org/10.1016/j.is.2012.02.004

[6] Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Bruno, G.: Automated discovery of structured process models from event logs: The discover-and-structure approach. Data & Knowledge Engineering **117** (2018) https://doi.org/10.1016/j.datak.2018.04.007

[7] Van Der Aalst, W.M.P.: Process Mining: Data Science In Action, Second edition edn., 2016 (2016)

[8] Leemans, S., Fahland, D., Aalst, W.: Discovering Block-Structured Process Models from Event Logs - A Constructive Approach, (2013). https://doi.org/10.1007/978-3-642-38697-8_17 . Pages: 329

[9] Leemans, S., Fahland, D., Aalst, W.: Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour vol. 171, (2014). https://doi.org/10.1007/978-3-319-06257-0_6 . Pages: 78

[10] Carmona, J., Van Dongen, B., Solti, A., Weidlich, M.: Conformance Checking: Relating Processes and Models, (2018)

[11] A. Augusto, R. Conforti, M. Dumas, M. L. Rosa, F. M. Maggi, A. Marrella, M. Mecella, A. Soo: Automated Discovery of Process Models from Event Logs: Review and Benchmark. IEEE Transactions on Knowledge and Data Engineering

31(4), 686–705 (2019) https://doi.org/10.1109/TKDE.2018.2841877

[12] Janssenswillen, G., Donders, N., Jouck, T., Depaire, B.: A comparative study of existing quality measures for process discovery. Information Systems **71**, 1–15 (2017) https://doi.org/10.1016/j.is.2017.06.002

[13] Corstjens, J., Depaire, B., Caris, A., Sörensen, K.: A multilevel evaluation method for heuristics with an application to the VRPTW - Corstjens - 2020 - International Transactions in Operational Research - Wiley Online Library (2019). https://onlinelibrary.wiley.com/doi/abs/10.1111/itor.12631 Accessed 2024-02-17

[14] Jouck, T., Depaire, B.: Generating Artificial Data for Empirical Analysis of Control-flow Discovery Algorithms. Business & Information Systems Engineering **61**(6), 695–712 (2019) https://doi.org/10.1007/s12599-018-0541-5 . Accessed 2024-02-16

[15] Berti, A., Zelst, S., Schuster, D.: PM4Py: A process mining library for Python - ScienceDirect (2023). https://www.sciencedirect.com/science/article/pii/S2665963823000933 Accessed 2023-10-24

[16] Berti, A., Zelst, S., Van Der Aalst, W.: Process Mining for Python (PM4Py): Bridging the Gap Between Process- and Data Science (2019). https://www.researchgate.net/publication/333130393_Process_Mining_for_Python_PM4Py_Bridging_the_Gap_Between_Process-_and_Data_Science Accessed 2023-10-24

[17] Augusto, A., Conforti, R., Dumas, M., La Rosa, M., Bruno, G.: Automated Discovery of Structured Process Models: Discover Structured Vs. Discover and Structure, (2016). https://doi.org/10.1007/978-3-319-46397-1_25 . Pages: 329

[18] Van Houdt, G., Leoni, M., Martin, N., Depaire, B.: An empirical evaluation of unsupervised event log abstraction techniques in process mining. Information Systems **121**, 102320 (2024) https://doi.org/10.1016/j.is.2023.102320 . Accessed 2024-04-27

[19] Berti, A., Aalst, W.M.P.v.d.: Reviving Token-based Replay: Increasing Speed While Improving Diagnostics (2019). Accessed 2024-06-02

[20] Munoz-Gama, J., Carmona, J.: A Fresh Look at Precision in Process Conformance **6336**, 211–226 (2010) https://doi.org/10.1007/978-3-642-15618-2_16

[21] Buijs, J.C.A.M., Van Dongen, B.F., Van Der Aalst, W.M.P.: Quality Dimensions in Process Discovery: The Importance of Fitness, Precision, Generalization and Simplicity. International Journal of Cooperative Information Systems **23**(01), 1440001 (2014) https://doi.org/10.1142/S0218843014400012 . Accessed 2024-06-02

[22] Blum, F.: Metrics in process discovery (2015). Accessed 2024-06-02

[23] Demsar, J.: Statistical Comparisons of Classifiers over Multiple Data Sets. Journal of Machine Learning Research **7**, 1–30 (2006)

Appendix A Legende AlgoritmeIDs

AlgoritmeID	Algoritme	Variant splitsingsvolgorde
XSPL	Baseline	CHOICE-SEQUENCE-PARALLEL-LOOP
SPXL	Algoritme variant 1	SEQUENCE-PARALLEL-CHOICE-LOOP
SPLX	Algoritme variant 2	SEQUENCE-PARALLEL-LOOP-CHOICE
SLPX	Algoritme variant 3	SEQUENCE-LOOP-PARALLEL-CHOICE
SLXP	Algoritme variant 4	SEQUENCE-LOOP-CHOICE-PARALLEL
SXLP	Algoritme variant 5	SEQUENCE-CHOICE-LOOP-PARALLEL
SXPL	Algoritme variant 6	SEQUENCE-CHOICE-PARALLEL-LOOP
XSLP	Algoritme variant 7	CHOICE-SEQUENCE-LOOP-PARALLEL
XLPS	Algoritme variant 8	CHOICE-LOOP-PARALLEL-SEQUENCE
XLSP	Algoritme variant 9	CHOICE-LOOP-SEQUENCE-PARALLEL
XPLS	Algoritme variant 10	CHOICE-PARALLEL-LOOP-SEQUENCE
XPSL	Algoritme variant 11	CHOICE-PARALLEL-SEQUENCE-LOOP
LSPX	Algoritme variant 12	LOOP-SEQUENCE-PARALLEL-CHOICE
LSXP	Algoritme variant 13	LOOP-SEQUENCE-CHOICE-PARALLEL
LPSX	Algoritme variant 14	LOOP-PARALLEL-SEQUENCE-CHOICE
LPXS	Algoritme variant 15	LOOP-PARALLEL-CHOICE-SEQUENCE
LXPS	Algoritme variant 16	LOOP-CHOICE-PARALLEL-SEQUENCE
LXSP	Algoritme variant 17	LOOP-CHOICE-SEQUENCE-PARALLEL
PSXL	Algoritme variant 18	PARALLEL-SEQUENCE-CHOICE-LOOP
PSLX	Algoritme variant 19	PARALLEL-SEQUENCE-LOOP-CHOICE
PLXS	Algoritme variant 20	PARALLEL-LOOP-CHOICE-SEQUENCE
PLSX	Algoritme variant 21	PARALLEL-LOOP-SEQUENCE-CHOICE
PXSL	Algoritme variant 22	PARALLEL-CHOICE-SEQUENCE-LOOP
PXLS	Algoritme variant 23	PARALLEL-CHOICE-LOOP-SEQUENCE

Table A1 Splitsingsvolgordes met overeenkomstige Algoritme-IDs

Appendix B Overzicht onderliggende processen

Table B2: Overzicht onderliggende Process Tree en bijhorende event log

Log	Mode	Min activities	Max activities	Sequence	Parallel	Choice	Loop
log_1	15	10	30	0.10	0.41	0.41	0.08
log_2	15	10	30	0.15	0.39	0.39	0.08
log_3	15	10	30	0.25	0.34	0.34	0.07
log_4	15	10	30	0.35	0.30	0.30	0.06
log_5	15	10	30	0.45	0.25	0.25	0.05
log_6	15	10	30	0.55	0.20	0.20	0.04
log_7	15	10	30	0.65	0.16	0.16	0.03
log_8	15	10	30	0.54	0.10	0.30	0.06
log_9	15	10	30	0.51	0.15	0.28	0.06
log_10	15	10	30	0.45	0.25	0.25	0.05
log_11	15	10	30	0.39	0.35	0.22	0.04
log_12	15	10	30	0.36	0.40	0.20	0.04
log_13	15	10	30	0.30	0.50	0.17	0.03
log_14	15	10	30	0.21	0.65	0.12	0.02
log_15	15	10	30	0.54	0.30	0.10	0.06
log_16	15	10	30	0.51	0.28	0.15	0.06
log_17	15	10	30	0.45	0.25	0.25	0.05
log_18	15	10	30	0.39	0.22	0.35	0.04
log_19	15	10	30	0.36	0.20	0.40	0.04
log_20	15	10	30	0.30	0.17	0.50	0.03
log_21	15	10	30	0.21	0.12	0.65	0.02
log_22	15	10	30	0.46	0.26	0.26	0.02
log_23	15	10	30	0.45	0.25	0.25	0.05
log_24	15	10	30	0.43	0.24	0.24	0.10
log_25	15	10	30	0.25	0.25	0.25	0.25
log_26	20	10	30	0.10	0.41	0.41	0.08
log_27	20	10	30	0.15	0.39	0.39	0.08
log_28	20	10	30	0.25	0.34	0.34	0.07
log_29	20	10	30	0.35	0.30	0.30	0.06
log_30	20	10	30	0.45	0.25	0.25	0.05
log_31	20	10	30	0.55	0.20	0.20	0.04
log_32	20	10	30	0.65	0.16	0.16	0.03
log_33	20	10	30	0.54	0.10	0.30	0.06
log_34	20	10	30	0.51	0.15	0.28	0.06
log_35	20	10	30	0.45	0.25	0.25	0.05

Vervolg op de volgende pagina

Table B2 – Vervolg van vorige pagina

Log	Mode	Min activities	Max activities	Sequence	Parallel	Choice	Loop
log_36	20	10	30	0.39	0.35	0.22	0.04
log_37	20	10	30	0.36	0.40	0.20	0.04
log_38	20	10	30	0.30	0.50	0.17	0.03
log_39	20	10	30	0.21	0.65	0.12	0.02
log_40	20	10	30	0.54	0.30	0.10	0.06
log_41	20	10	30	0.51	0.28	0.15	0.06
log_42	20	10	30	0.45	0.25	0.25	0.05
log_43	20	10	30	0.39	0.22	0.35	0.04
log_44	20	10	30	0.36	0.20	0.40	0.04
log_45	20	10	30	0.30	0.17	0.50	0.03
log_46	20	10	30	0.21	0.12	0.65	0.02
log_47	20	10	30	0.46	0.26	0.26	0.02
log_48	20	10	30	0.45	0.25	0.25	0.05
log_49	20	10	30	0.43	0.24	0.24	0.10
log_50	20	10	30	0.25	0.25	0.25	0.25
log_51	25	10	30	0.10	0.41	0.41	0.08
log_52	25	10	30	0.15	0.39	0.39	0.08
log_53	25	10	30	0.25	0.34	0.34	0.07
log_54	25	10	30	0.35	0.30	0.30	0.06
log_55	25	10	30	0.45	0.25	0.25	0.05
log_56	25	10	30	0.55	0.20	0.20	0.04
log_57	25	10	30	0.65	0.16	0.16	0.03
log_58	25	10	30	0.54	0.10	0.30	0.06
log_59	25	10	30	0.51	0.15	0.28	0.06
log_60	25	10	30	0.45	0.25	0.25	0.05
log_61	25	10	30	0.39	0.35	0.22	0.04
log_62	25	10	30	0.36	0.40	0.20	0.04
log_63	25	10	30	0.30	0.50	0.17	0.03
log_64	25	10	30	0.21	0.65	0.12	0.02
log_65	25	10	30	0.54	0.30	0.10	0.06
log_66	25	10	30	0.51	0.28	0.15	0.06
log_67	25	10	30	0.45	0.25	0.25	0.05
log_68	25	10	30	0.39	0.22	0.35	0.04
log_69	25	10	30	0.36	0.20	0.40	0.04
log_70	25	10	30	0.30	0.17	0.50	0.03
log_71	25	10	30	0.21	0.12	0.65	0.02
log_72	25	10	30	0.46	0.26	0.26	0.02
log_73	25	10	30	0.45	0.25	0.25	0.05
log_74	25	10	30	0.43	0.24	0.24	0.10
log_75	25	10	30	0.25	0.25	0.25	0.25

Appendix C Descriptieve analyse: Event logs

Table C3: Descriptieve waarden voor het aantal activiteiten binnen de traces in een event log

LogID	Minimum	Maximum	Gemiddelde	Mediaan
log_1	0	6	2.828	2.0
log_2	5	23	5.942	5.0
log_3	2	7	4.319	3.0
log_4	14	36	17.156	17.0
log_5	2	9	5.227	2.0
log_6	10	92	22.467	18.0
log_7	11	128	25.912	25.0
log_8	3	14	6.441	5.5
log_9	2	146	14.080	8.0
log_10	13	105	22.857	20.0
log_11	5	13	8.262	8.0
log_12	5	28	9.619	9.0
log_13	10	46	14.192	11.0
log_14	10	10	10.000	10.0
log_15	8	36	10.042	8.0
log_16	10	116	22.898	17.0
log_17	4	70	11.309	5.0
log_18	3	9	5.610	7.0
log_19	6	44	12.355	11.0
log_20	1	5	2.194	2.0
log_21	1	3	1.984	2.0
log_22	7	11	8.531	8.0
log_23	8	24	9.857	9.0
log_24	5	27	6.427	5.0
log_25	2	135	19.947	14.0
log_26	0	32	3.316	2.0
log_27	1	294	39.537	26.0
log_28	4	9	6.463	6.0
log_29	4	21	5.058	4.0
log_30	6	18	11.088	6.0
log_31	5	5	5.000	5.0
log_32	15	16	15.462	15.0
log_33	9	40	11.416	10.0
log_34	2	79	11.188	8.0

Vervolg op de volgende pagina

Table C3 – Vervolg van vorige pagina

LogID	Minimum	Maximum	Gemiddelde	Mediaan
log_35	3	25	6.051	6.0
log_36	9	127	23.053	19.0
log_37	8	16	12.184	14.0
log_38	1	49	5.856	5.0
log_39	16	17	16.483	16.0
log_40	13	76	20.390	18.0
log_41	2	85	8.636	3.0
log_42	1	10	3.953	2.0
log_43	7	53	9.574	9.0
log_44	1	6	2.910	3.0
log_45	2	6	4.439	5.0
log_46	1	10	2.824	3.0
log_47	3	42	7.013	4.0
log_48	7	132	18.624	7.0
log_49	9	35	13.378	13.0
log_50	2	193	29.988	19.0
log_51	6	8	7.439	7.0
log_52	10	45	13.006	12.0
log_53	5	10	6.771	7.0
log_54	3	72	6.716	4.0
log_55	6	9	7.464	7.0
log_56	17	86	25.352	23.0
log_57	15	20	17.714	17.0
log_58	11	41	14.274	13.0
log_59	1	43	3.947	4.0
log_60	4	13	8.330	8.0
log_61	5	41	11.500	11.0
log_62	4	174	21.581	16.0
log_63	5	38	9.677	6.0
log_64	4	13	7.939	9.0
log_65	21	56	25.737	25.0
log_66	1	116	9.008	2.0
log_67	10	47	13.676	12.0
log_68	1	8	3.671	2.0
log_69	4	17	7.296	7.0
log_70	7	13	8.990	9.0
log_71	1	17	2.540	2.0
log_72	11	14	12.443	11.0
log_73	11	69	19.563	17.0
log_74	11	43	17.354	16.0
log_75	2	741	58.309	32.0

Appendix D Descriptieve analyse: Kwaliteitsmaatstaven

Table D4: Fitness waarden voor de event logs

LogID	Gemiddelde fitness	Minimum fitness	Maximum fitness
log_1	1.0000000	1.0000000	1.0000000
log_2	1.0000000	1.0000000	1.0000000
log_3	1.0000000	1.0000000	1.0000000
log_4	1.0000000	1.0000000	1.0000000
log_5	1.0000000	1.0000000	1.0000000
log_6	1.0000000	1.0000000	1.0000000
log_7	1.0000000	1.0000000	1.0000000
log_8	1.0000000	1.0000000	1.0000000
log_9	1.0000000	1.0000000	1.0000000
log_10	1.0000000	1.0000000	1.0000000
log_11	1.0000000	1.0000000	1.0000000
log_12	1.0000000	1.0000000	1.0000000
log_13	1.0000000	1.0000000	1.0000000
log_14	1.0000000	1.0000000	1.0000000
log_15	1.0000000	1.0000000	1.0000000
log_16	1.0000000	1.0000000	1.0000000
log_17	1.0000000	1.0000000	1.0000000
log_18	1.0000000	1.0000000	1.0000000
log_19	1.0000000	1.0000000	1.0000000
log_20	1.0000000	1.0000000	1.0000000
log_21	1.0000000	1.0000000	1.0000000
log_22	1.0000000	1.0000000	1.0000000
log_23	1.0000000	1.0000000	1.0000000
log_24	1.0000000	1.0000000	1.0000000
log_25	1.0000000	1.0000000	1.0000000
log_26	1.0000000	1.0000000	1.0000000
log_27	1.0000000	1.0000000	1.0000000
log_28	1.0000000	1.0000000	1.0000000
log_29	1.0000000	1.0000000	1.0000000
log_30	1.0000000	1.0000000	1.0000000
log_31	1.0000000	1.0000000	1.0000000
log_32	1.0000000	1.0000000	1.0000000
log_33	1.0000000	1.0000000	1.0000000
log_34	1.0000000	1.0000000	1.0000000
log_35	1.0000000	1.0000000	1.0000000

Vervolg op de volgende pagina

Table D4 – Vervolg van vorige pagina

LogID	Gemiddelde fitness	Minimum fitness	Maximum fitness
log_36	1.0000000	1.0000000	1.0000000
log_37	1.0000000	1.0000000	1.0000000
log_38	1.0000000	1.0000000	1.0000000
log_39	1.0000000	1.0000000	1.0000000
log_40	1.0000000	1.0000000	1.0000000
log_41	1.0000000	1.0000000	1.0000000
log_42	1.0000000	1.0000000	1.0000000
log_43	1.0000000	1.0000000	1.0000000
log_44	1.0000000	1.0000000	1.0000000
log_45	1.0000000	1.0000000	1.0000000
log_46	1.0000000	1.0000000	1.0000000
log_47	1.0000000	1.0000000	1.0000000
log_48	1.0000000	1.0000000	1.0000000
log_49	1.0000000	1.0000000	1.0000000
log_50	1.0000000	1.0000000	1.0000000
log_51	1.0000000	1.0000000	1.0000000
log_52	1.0000000	1.0000000	1.0000000
log_53	1.0000000	1.0000000	1.0000000
log_54	1.0000000	1.0000000	1.0000000
log_55	0.9949431	0.9865149	1.0000000
log_56	0.9643110	0.9517577	0.9768732
log_57	1.0000000	1.0000000	1.0000000
log_58	1.0000000	1.0000000	1.0000000
log_59	1.0000000	1.0000000	1.0000000
log_60	1.0000000	1.0000000	1.0000000
log_61	1.0000000	1.0000000	1.0000000
log_62	1.0000000	1.0000000	1.0000000
log_63	1.0000000	1.0000000	1.0000000
log_64	1.0000000	1.0000000	1.0000000
log_65	1.0000000	1.0000000	1.0000000
log_66	1.0000000	1.0000000	1.0000000
log_67	1.0000000	1.0000000	1.0000000
log_68	1.0000000	1.0000000	1.0000000
log_69	1.0000000	1.0000000	1.0000000
log_70	1.0000000	1.0000000	1.0000000
log_71	1.0000000	1.0000000	1.0000000
log_72	1.0000000	1.0000000	1.0000000
log_73	1.0000000	1.0000000	1.0000000
log_74	1.0000000	1.0000000	1.0000000
log_75	1.0000000	1.0000000	1.0000000

Table D5: Precision waarden voor de event logs

LogID	Gemiddelde precision	Minimum precision	Maximum precision
log_1	0.8854062	0.8854062	0.8854062
log_2	0.7301677	0.6895395	0.7373272
log_3	0.9675058	0.9675058	0.9675058
log_4	0.1077705	0.1077705	0.1077705
log_5	0.9600661	0.9600661	0.9600661
log_6	0.7342975	0.7342975	0.7342975
log_7	0.7900664	0.7900664	0.7900664
log_8	0.9960912	0.9960912	0.9960912
log_9	0.7042949	0.7042949	0.7042949
log_10	0.4258538	0.4258538	0.4258538
log_11	0.9870318	0.9870318	0.9870318
log_12	0.5664591	0.5176920	0.6012927
log_13	0.5951470	0.5951470	0.5951470
log_14	0.9334902	0.9334902	0.9334902
log_15	0.9837387	0.9837387	0.9837387
log_16	0.4756879	0.4511434	0.4874234
log_17	0.6138977	0.6138977	0.6138977
log_18	0.9587429	0.9587429	0.9587429
log_19	0.4958402	0.4958402	0.4958402
log_20	0.9875992	0.9875992	0.9875992
log_21	1.0000000	1.0000000	1.0000000
log_22	0.9984742	0.9984742	0.9984742
log_23	0.6137090	0.5645244	0.7050710
log_24	0.8898925	0.8168258	0.9226844
log_25	0.4466731	0.4466731	0.4466731
log_26	0.4280884	0.4280884	0.4280884
log_27	0.3075866	0.3062816	0.3086440
log_28	0.8106342	0.8106342	0.8106342
log_29	0.9473735	0.8617904	0.9910076
log_30	0.8736118	0.8736118	0.8736118
log_31	0.9969657	0.9969657	0.9969657
log_32	1.0000000	1.0000000	1.0000000
log_33	0.2553633	0.2433311	0.2744220
log_34	0.6869145	0.6869145	0.6869145
log_35	0.8726551	0.8170901	0.9282201
log_36	0.3765021	0.3400712	0.4573427
log_37	0.4623553	0.4623553	0.4623553
log_38	0.7184444	0.7184444	0.7184444
log_39	0.4324529	0.4324529	0.4324529

Vervolg op de volgende pagina

Table D5 – Vervolg van vorige pagina

LogID	Gemiddelde precision	Minimum precision	Maximum precision
log_40	0.2919152	0.2412685	0.3287988
log_41	0.7871290	0.7871290	0.7871290
log_42	1.0000000	1.0000000	1.0000000
log_43	0.9884695	0.9884695	0.9884695
log_44	1.0000000	1.0000000	1.0000000
log_45	0.9416320	0.9416320	0.9416320
log_46	0.9981133	0.9981133	0.9981133
log_47	0.9933029	0.9933029	0.9933029
log_48	0.5976371	0.5976371	0.5976371
log_49	0.9842486	0.9842486	0.9842486
log_50	0.4513094	0.4304278	0.4660364
log_51	0.7852484	0.7852484	0.7852484
log_52	0.4470023	0.4470023	0.4470023
log_53	0.7130119	0.7130119	0.7130119
log_54	0.7553060	0.7553060	0.7553060
log_55	0.2261508	0.2096505	0.2536512
log_56	0.1763469	0.1739463	0.1787474
log_57	0.2559324	0.2217801	0.2987220
log_58	0.9580199	0.9580199	0.9580199
log_59	0.9889992	0.9889992	0.9889992
log_60	0.5631015	0.5631015	0.5631015
log_61	0.5111479	0.5111479	0.5111479
log_62	0.4765581	0.4765581	0.4765581
log_63	0.5668476	0.5668476	0.5668476
log_64	0.6407680	0.6407680	0.6407680
log_65	0.5309614	0.5309614	0.5309614
log_66	0.1439019	0.1425544	0.1447119
log_67	0.9841448	0.9841448	0.9841448
log_68	0.9986500	0.9986500	0.9986500
log_69	0.6813926	0.6542809	0.7410343
log_70	0.8156927	0.8156927	0.8156927
log_71	0.9916356	0.9916356	0.9916356
log_72	0.7923648	0.7923648	0.7923648
log_73	0.7455657	0.7455657	0.7455657
log_74	0.7410727	0.7410727	0.7410727
log_75	0.3755047	0.3553498	0.3869768

Table D6: Generalisation waarden voor de event logs

LogID	Gemiddelde generalisation	Minimum generalisation	Maximum generalisation
log_1	0.9248651	0.9248651	0.9248651
log_2	0.9257154	0.9195224	0.9266764
log_3	0.9394133	0.9394133	0.9394133
log_4	0.9190127	0.9190127	0.9190127
log_5	0.9454478	0.9454478	0.9454478
log_6	0.9595438	0.9595438	0.9595438
log_7	0.9670430	0.9670430	0.9670430
log_8	0.9427034	0.9427034	0.9427034
log_9	0.9572105	0.9572105	0.9572105
log_10	0.9651444	0.9651444	0.9651444
log_11	0.9485705	0.9485705	0.9485705
log_12	0.9582922	0.9582702	0.9583079
log_13	0.9667676	0.9667676	0.9667676
log_14	0.9618220	0.9618220	0.9618220
log_15	0.9632256	0.9632256	0.9632256
log_16	0.9650630	0.9644463	0.9654947
log_17	0.9500510	0.9500510	0.9500510
log_18	0.9481242	0.9481242	0.9481242
log_19	0.9515285	0.9515285	0.9515285
log_20	0.9043207	0.9043207	0.9043207
log_21	0.9182021	0.9182021	0.9182021
log_22	0.9508041	0.9508041	0.9508041
log_23	0.9556313	0.9556313	0.9556313
log_24	0.9266410	0.9239361	0.9282765
log_25	0.9663463	0.9663463	0.9663463
log_26	0.9128162	0.9128162	0.9128162
log_27	0.9559303	0.9542627	0.9582649
log_28	0.9489684	0.9489684	0.9489684
log_29	0.9223676	0.9201192	0.9245262
log_30	0.9498486	0.9498486	0.9498486
log_31	0.9392834	0.9392834	0.9392834
log_32	0.9662039	0.9662039	0.9662039
log_33	0.9544212	0.9524654	0.9563171
log_34	0.9558452	0.9558452	0.9558452
log_35	0.9408182	0.9404722	0.9411642
log_36	0.9606019	0.9594906	0.9636609
log_37	0.9569661	0.9569661	0.9569661
log_38	0.9426262	0.9426262	0.9426262
log_39	0.9638071	0.9638071	0.9638071

Vervolg op de volgende pagina

Table D6 – Vervolg van vorige pagina

LogID	Gemiddelde generalisation	Minimum generalisation	Maximum generalisation
log_40	0.9641994	0.9640080	0.9643093
log_41	0.9444546	0.9444546	0.9444546
log_42	0.9158531	0.9158531	0.9158531
log_43	0.9483919	0.9479395	0.9485782
log_44	0.9245216	0.9245216	0.9245216
log_45	0.9257570	0.9257570	0.9257570
log_46	0.8810528	0.8810528	0.8810528
log_47	0.9454248	0.9454248	0.9454248
log_48	0.9548177	0.9548177	0.9548177
log_49	0.9601989	0.9601989	0.9601989
log_50	0.9661398	0.9652262	0.9669017
log_51	0.9408740	0.9408740	0.9408740
log_52	0.9568130	0.9568130	0.9568130
log_53	0.9412733	0.9412733	0.9412733
log_54	0.9466034	0.9466034	0.9466034
log_55	0.9428547	0.9379004	0.9511120
log_56	0.9652424	0.9624253	0.9680537
log_57	0.9588636	0.9588482	0.9590326
log_58	0.9464644	0.9462046	0.9468975
log_59	0.8950921	0.8950921	0.8950921
log_60	0.9306967	0.9306967	0.9306967
log_61	0.9548896	0.9548896	0.9548896
log_62	0.9635540	0.9635540	0.9635540
log_63	0.9542255	0.9542255	0.9542255
log_64	0.9478200	0.9478200	0.9478200
log_65	0.9634973	0.9634973	0.9634973
log_66	0.9218039	0.9130151	0.9268546
log_67	0.9439225	0.9439225	0.9439225
log_68	0.9144969	0.9144969	0.9144969
log_69	0.9061124	0.9022551	0.9206551
log_70	0.9424840	0.9424840	0.9424840
log_71	0.9004573	0.9004573	0.9004573
log_72	0.9614831	0.9614831	0.9614831
log_73	0.9639476	0.9639476	0.9639476
log_74	0.9528654	0.9528654	0.9528654
log_75	0.9698844	0.9690357	0.9702976

Table D7: Simplicity waarden voor de event logs

LogID	Gemiddelde simplicity	Minimum simplicity	Maximum simplicity
log_1	0.6521739	0.6521739	0.6521739
log_2	0.5953651	0.5903614	0.6000000
log_3	0.7391304	0.7391304	0.7391304
log_4	0.5744681	0.5744681	0.5744681
log_5	0.8039216	0.8039216	0.8039216
log_6	0.7142857	0.7142857	0.7142857
log_7	0.8285714	0.8285714	0.8285714
log_8	0.7959184	0.7959184	0.7959184
log_9	0.7142857	0.7142857	0.7142857
log_10	0.7551020	0.7551020	0.7551020
log_11	0.7419355	0.7419355	0.7419355
log_12	0.6661813	0.6585366	0.6716418
log_13	0.7419355	0.7419355	0.7419355
log_14	0.7419355	0.7419355	0.7419355
log_15	0.7297297	0.7297297	0.7297297
log_16	0.7120958	0.6956522	0.7254902
log_17	0.7272727	0.7272727	0.7272727
log_18	0.7627119	0.7627119	0.7627119
log_19	0.6666667	0.6666667	0.6666667
log_20	0.6533333	0.6533333	0.6533333
log_21	0.6000000	0.6000000	0.6000000
log_22	0.7333333	0.7333333	0.7333333
log_23	0.7648839	0.7454545	0.7959184
log_24	0.7178419	0.7000000	0.7230769
log_25	0.6923077	0.6923077	0.6923077
log_26	0.6277056	0.6277056	0.6277056
log_27	0.6661536	0.6637168	0.6695652
log_28	0.7011494	0.7011494	0.7011494
log_29	0.6601924	0.6470588	0.6666667
log_30	0.7662338	0.7662338	0.7662338
log_31	0.6727273	0.6727273	0.6727273
log_32	0.9090909	0.9090909	0.9090909
log_33	0.6360609	0.6279070	0.6444444
log_34	0.6811594	0.6811594	0.6811594
log_35	0.7339989	0.7215190	0.7464789
log_36	0.7177796	0.7099237	0.7500000
log_37	0.6969697	0.6969697	0.6969697
log_38	0.7600000	0.7600000	0.7600000
log_39	0.7021277	0.7021277	0.7021277

Vervolg op de volgende pagina

Table D7 – Vervolg van vorige pagina

LogID	Gemiddelde simplicity	Minimum simplicity	Maximum simplicity
log_40	0.6704057	0.6595745	0.6818182
log_41	0.7215190	0.7215190	0.7215190
log_42	0.7534247	0.7534247	0.7534247
log_43	0.7481303	0.7435897	0.7500000
log_44	0.6000000	0.6000000	0.6000000
log_45	0.6551724	0.6551724	0.6551724
log_46	0.6082474	0.6082474	0.6082474
log_47	0.7500000	0.7500000	0.7500000
log_48	0.7319588	0.7319588	0.7319588
log_49	0.7627119	0.7627119	0.7627119
log_50	0.6626881	0.6470588	0.6774194
log_51	0.6250000	0.6250000	0.6250000
log_52	0.6421053	0.6421053	0.6421053
log_53	0.6842105	0.6842105	0.6842105
log_54	0.7142857	0.7142857	0.7142857
log_55	0.5644001	0.5639098	0.5652174
log_56	0.6453230	0.6417910	0.6488550
log_57	0.6998413	0.6842105	0.7319588
log_58	0.7490835	0.7464789	0.7534247
log_59	0.7209302	0.7209302	0.7209302
log_60	0.6806723	0.6806723	0.6806723
log_61	0.6603774	0.6603774	0.6603774
log_62	0.7090909	0.7090909	0.7090909
log_63	0.7078652	0.7078652	0.7078652
log_64	0.6956522	0.6956522	0.6956522
log_65	0.7657658	0.7657658	0.7657658
log_66	0.5582933	0.5505618	0.5670103
log_67	0.7349398	0.7349398	0.7349398
log_68	0.7500000	0.7500000	0.7500000
log_69	0.5855722	0.5833333	0.5862069
log_70	0.6421053	0.6421053	0.6421053
log_71	0.5774648	0.5774648	0.5774648
log_72	0.7931034	0.7931034	0.7931034
log_73	0.7551020	0.7551020	0.7551020
log_74	0.7307692	0.7307692	0.7307692
log_75	0.6995510	0.6923077	0.7027027

Algoritme variant	Gemiddelde fitness	Gemiddelde precision	Gemiddelde generalisation	Gemiddelde simplicity
XSPL (Baseline)	0.9993568	0.7055422	0.9447549	0.6997209
SPXL	0.9991770	0.7069308	0.9450019	0.6999656
SPLX	0.9993578	0.7050640	0.9447585	0.7001845
SLPX	0.9996908	0.7029017	0.9449406	0.6999336
SLXP	0.9996916	0.7063440	0.9450568	0.7009188
SXLP	0.9996916	0.7063020	0.9448406	0.7006538
SXPL	0.9991770	0.7044747	0.9449009	0.6992757
XSLP	0.9996908	0.7053832	0.9449854	0.7005271
XLPS	0.9995110	0.7045329	0.9450050	0.7006040
XLSP	0.9996916	0.7055115	0.9448945	0.7006018
XPLS	0.9991770	0.7072262	0.9450800	0.7001783
XPSL	0.9993578	0.7046850	0.9447917	0.6996814
LSPX	0.9996916	0.7034267	0.9447679	0.7007221
LSXP	0.9995110	0.7048071	0.9450660	0.7005156
LPSX	0.9996908	0.7058964	0.9448438	0.7010075
LPXS	0.9995110	0.7059220	0.9452775	0.7001142
LXPS	0.9995118	0.7054453	0.9450444	0.7000995
LXSP	0.9996908	0.7052018	0.9451147	0.7004247
PSXL	0.9993568	0.7094283	0.9447944	0.7001744
PSLX	0.9991770	0.7067456	0.9449647	0.6992602
PLXS	0.9993568	0.7065583	0.9446990	0.7003069
PLSX	0.9991780	0.7049088	0.9449531	0.6993055
PXSL	0.9993568	0.7050925	0.9448350	0.6999567
PXLS	0.9993578	0.7047687	0.9447173	0.6998669

Table D8 Gemiddelde log fitness, precision, generalisation en simplicity per algoritme variant

Algoritme variant	Maximum log fitness	Maximum precision	Maximum generalisation	Maximum simplicity
XSPL (Baseline)	1	1	0.9700808	0.9090909
SPXL	1	1	0.9700808	0.9090909
SPLX	1	1	0.9700808	0.9090909
SLXP	1	1	0.9702976	0.9090909
SLPX	1	1	0.9702976	0.9090909
SXLP	1	1	0.9702976	0.9090909
SXPL	1	1	0.9690357	0.9090909
XSLP	1	1	0.9702976	0.9090909
XLPS	1	1	0.9702976	0.9090909
XLSP	1	1	0.9702976	0.9090909
XPLS	1	1	0.9700808	0.9090909
XPSL	1	1	0.9690357	0.9090909
LSPX	1	1	0.9702976	0.9090909
LSXP	1	1	0.9702976	0.9090909
LPSX	1	1	0.9702976	0.9090909
LPXS	1	1	0.9702976	0.9090909
LXPS	1	1	0.9702976	0.9090909
LXSP	1	1	0.9702976	0.9090909
PSXL	1	1	0.9700808	0.9090909
PSLX	1	1	0.9690357	0.9090909
PLXS	1	1	0.9690357	0.9090909
PLSX	1	1	0.9690357	0.9090909
PXSL	1	1	0.9690357	0.9090909
PXLS	1	1	0.9690357	0.9090909

Table D9 Maximum fitness, precision, generalisation en simplicity per algoritme variant

Algoritme variant	Minimum log fitness	Minimum precision	Minimum generalisation	Minimum simplicity
XSPL (Baseline)	0.9517577	0.1077705	0.8810528	0.5578947
SPXL	0.9517577	0.1077705	0.8810528	0.5567568
SPLX	0.9518371	0.1077705	0.8810528	0.5639098
SLPX	0.9768125	0.1077705	0.8810528	0.5639098
SLXP	0.9768732	0.1077705	0.8810528	0.5567568
SXLP	0.9768732	0.1077705	0.8810528	0.5589744
SXPL	0.9517577	0.1077705	0.8810528	0.5505618
XSLP	0.9768125	0.1077705	0.8810528	0.5634518
XLPS	0.9768125	0.1077705	0.8810528	0.5652174
XLSP	0.9768732	0.1077705	0.8810528	0.5567568
XPLS	0.9517577	0.1077705	0.8810528	0.5505618
XPSL	0.9518371	0.1077705	0.8810528	0.5578947
LSPX	0.9768732	0.1077705	0.8810528	0.5634518
LSXP	0.9768125	0.1077705	0.8810528	0.5652174
LPSX	0.9768125	0.1077705	0.8810528	0.5578947
LPXS	0.9768125	0.1077705	0.8810528	0.5567568
LXPS	0.9768732	0.1077705	0.8810528	0.5567568
LXSP	0.9768125	0.1077705	0.8810528	0.5578947
PSXL	0.9517577	0.1077705	0.8810528	0.5505618
PSLX	0.9517577	0.1077705	0.8810528	0.5505618
PLXS	0.9517577	0.1077705	0.8810528	0.5567568
PLSX	0.9518371	0.1077705	0.8810528	0.5578947
PXSL	0.9517577	0.1077705	0.8810528	0.5578947
PXLS	0.9518371	0.1077705	0.8810528	0.5567568

Table D10 Minimum log fitness, precision, generalisation en simplicity per algoritme variant