Behavioral Pattern Mining for Flexible Processes

Mehdi Acheli

PSL
UNIVERSITÉ PARIS

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PSL

Préparée à Université Paris Dauphine

Behavioral Pattern Mining for Flexible Processes

Soutenue par
Mehdi Acheli
Le **25/10/2021**

École doctorale n°543
ED de Dauphine

Spécialité
Informatique

Composition du jury :

Allel HADJALI Professeur, ISAE - ENSMA	*Président du jury*
Salima BENBERNOU Professeure, Université Paris Descartes	*Rapporteure*
Walid GAALOUL Professeur, Télécom SudParis (ex INT)	*Rapporteur*
Matthias WEIDLICH Professeur, Humboldt-Universität zu Berlin	*Examinateur*
Pavlos DELIAS Associate Professor, International Hellenic University	*Examinateur*
Daniela GRIGORI Professeur Université Paris-Dauphine-PSL	*Directrice*

Dauphine | PSL

À mon père, mon vieil ami et à ma mère...

Remerciements

Au tout commencement, j'aimerais remercier Mme. Daniela Grigori. Pour moi, l'arrivant en France sans attaches ni famille, elle a joué le rôle de figure maternelle. S'inquiétant pour moi et veillant toujours à mon bien être. Je ne saurais fidèlement décrire sa gentillesse, sa bienveillance et sa disponibilité. Week-ends ou vacances, matin ou soir, nous avons toujours été en contact pour notre travail de recherche. Ce travail dont elle m'a transmis les ficelles. Orienté détails comme je le suis, ce qui peut être handicapant, elle m'a appris à abstraire et hiérarchiser la complexité dans mes écrits. Elle m'a également appris comment me mettre à la place du lecteur pour garantir la compréhension de nos travaux. Je suis reconnaissant à Daniela pour ses remarques pertinentes qui, souvent, échappaient à celui qui s'est embourbé dans les technicités.

Au Pr. Matthias Weidlich, j'adresse mes remerciements les plus sincères. Ses capacités de synthèse et son style de rédaction ont sûrement pesé dans la balance quand nos articles ont été acceptés. Même si Matthias n'était pas mon encadrant officiellement, il s'est impliqué tout autant et a cosigné l'intégralité de nos travaux de thèse. J'ai eu beaucoup de chance d'avoir rencontré celui qui a précisé et défini un thème à ma thèse et avec lequel on se comprenait en l'espace de quelques secondes.

Ensuite, j'aimerais exprimer ma gratitude envers Pr. Walid Gaaloul et Pr. Salima Benbernou qui ont accepté d'être les rapporteurs de mon modeste travail et ont donc investi de leur temps et montré leur intérêt pour ma recherche. Particulièrement, je les remercie pour les questions pertinentes qu'ils m'ont posées, leur rigueur, leurs remarques et les différentes perspectives qu'ils ont soulevées. De même, mes remerciements vont à Dr. Pavlos Delias pour les différentes réflexions et les séances de brainstorming que nous avons eues durant ma thèse mais aussi pour avoir accepté d'être un membre de mon jury de thèse me faisant bénéficier de son expérience et expertise.

Un grand merci au président de mon jury de thèse: Pr. Allel Hadj Ali qui, pour l'anecdote, a été l'encadrant de mon stage de fin d'études et la personne qui m'a recommandé à Daniela. Ainsi, quatre années plus tard, le maître est revenu évaluer mon travail.

Cette thèse a été un combat que je n'ai pas mené seul. J'avais à mes côtés des camarades fidèles et tout aussi vaillants. Des camarades qui ont rejoint mon navire à différents moments de ma vie mais qui, chacun, ont su recoudre une voile, remplacer une planche ou pour certains, même me faire changer de cap. Je pense notamment à mes plus vieux amis, ma classe de lycée et que je connais maintenant depuis plus de dix ans. Je pense à Tarek, l'un de mes meilleurs amis, qui m'a appris le sens de la responsabilité et Dalil, le sens du calme. Je remercie également Abdelmalek pour son sourire et sa bonne humeur inextinguibles. Je pense à Chakib, l'artiste de notre groupe qui m'a toujours encouragé à libérer ma créativité et a apprécié, plus que nécessaire, mes maigres contributions. Je n'oublierai pas de citer Djamel, mon hébergeur officiel quand je suis arrivé en France qui m'a guidé dans cette nouvelle aventure avec ses conseils et recommandations. Il s'agit de la personne la plus courageuse et la plus battante que j'ai connue. Ma gratitude va aussi vers Salim, un autre ami de ce vieil âge qui m'a aidé à travers bien des péripéties. Il m'est également

impensable de ne pas remercier Tarek (Kassar) qui a été là pour moi à bien des occasions.

Ensuite viennent mes amis de mon école d'ingénieur, L'ESI. Ceux que j'ai côtoyés pendant cinq années. Que dire de la force des liens qui se sont tissés. Tout d'abord, je remercie ma bande joyeuse zaltonienne: Salah, Cherif et Amine complétée par le cinquième mousquetaire Ali. Nos retrouvailles qui ont ponctué mes quatre années de thèse ont été à chaque fois un tremplin qui me revigorait, effaçait ma fatigue et me propulsait vers les étapes supérieures. C'est avec ces personnes là et seulement avec elles que j'ai discuté des sujets les plus complexes et les plus profonds, de mes soucis et de mes questionnements. Je les remercie pour leur patience, leur absence totale de jugement et d'être mes chers amis avec qui je peux être pleinement moi-même et partage tellement de passions. J'ouvre une parenthèse pour dire à Ali que mes visites à Compiègne étaient un moment de paix dans une vie bien chargée. Tu ne peux pas savoir à quel point cela me faisait du bien de venir te voir et d'apprécier tes compétences culinaires. Pour ce qui est de mes autres camarades de L'ESI, séparés par notre nouvelle vie d'adultes et le train-train de la vie quotidienne, je suis content que nous nous soyons débrouillés pour organiser au moins une sortie annuelle. Vous retrouver ou vous parler, même par message, m'a toujours fait beaucoup de bien, Wissem, Billel, Islem, Rida, Abdelkader, Walid, Chahine, Amjed, Hani, Hadjer, Youcef, Nassim, Lotfi, Mehdi, Imene, Yacine, Dob, Aymen et bien d'autres. Je remercie particulièrement Sara qui a su m'écouter et que la distance ne nous a pas séparés. Merci pour ton écoute, ton support et tes conseils quand le morale était au plus bas. Enfin, la beauté de l'ESI fait qu'on ne se limite pas à sa génération mais qu'on côtoie les plus vieux et les moins vieux. C'est ainsi que j'ai retrouvé en Mourad, l'un de mes meilleurs amis et l'une des personnes qui font mon quotidien. Je ne saurais oublier Dahmane ou Amine (Remache) avec lesquels j'ai partagé tellement de bons moments. Parmi mes autres camarades, je peux citer Nassim (Ait Ali), Yasmine et Adnene. Tout ce lot joyeux a rendu possible ce travail de thèse.

Je voudrais également parler de ma dernière aventure au LAMSADE, le laboratoire de recherche qui m'a accueilli. Je peux dire sans exagération que ma thèse m'a apporté maintes fois plus sur le plan personnel que professionnel. Elle a été une aventure humaine extra-ordinaire. J'ai rencontré des personnes hors du commun qu'aucun autre endroit ne pourrait rassembler. Je remercie donc tous mes camarades lamsadiens pour nos débats, nos conversations mondaines ou plus sophistiquées, nos délires et nos rires, nos sorties, nos jeux et nos combats de lutte et bien d'autres choses. Je pense particulièrement à Mehdi, la gentillesse et l'innocence incarnée, Amine, le calme au milieu de la tempête, George, le tempétueux au milieu du calme, Axel, le droit, Hossein, le serviable, Ons, l'Hadja, Raja, la souriante, Beatrice, le rayon de soleil et la flamme du labo. Je remercie Nicolas et Felipe pour leur bonne humeur, Charles pour nos délires post-cinéma et enfin Pierre qui m'a accompagné dans un processus développemental compliqué qui ne s'est pas toujours fait en douceur. Merci également à Hiba en qui j'ai trouvé une amie et confidente ainsi que Diana qui m'a conseillé en sa qualité de prédécesseuse chez Daniela. Je ne me permets pas d'oublier Satya, Yassine, Justin et Nikos ainsi que toutes nos nouvelles recrues qui sont bien parties pour sauvegarder notre ambiance: Virginia, Ariane, Théo, Lucas, Alexandre, Marie, Louise et bien d'autres. Enfin, le labo ne pourrait exister sans le travail des équipes administratives, enseignantes et complémentaires. Je remercie tout ce grand monde qui m'a assisté et a assisté tout le monde durant notre passage. Je peux mentionner Marie, Eleni et

Tatiana pour n'en citer que quelques-uns. Enfin, je n'ai pas de mots pour remercier Céline pour toutes les raisons qu'elle saura reconnaître.

On est souvent tentés de prendre leur présence pour acquis mais ce n'est aucunement le cas. Je salue l'effort de ma famille qui a tout fait pour moi et a été là à chaque étape de ma vie, un allié dans l'ombre qui ne me quitte jamais et un lien infaillible et sacré. Particulièrement, je remercie ma tante, ma seconde mère qui a veillé à ce que je ne manque de rien. Ensuite, je remercie Amirouche, mon voisin, cousin et meilleur ami, la pilule dopante et l'injection de remontants. Je remercie mon frère Ali, ma petite soeur Hadjer, ma cousine Lylia et ma cousine adoptive Sabrina pour tous nos délires.

Enfin, à sa mémoire, je dédie ce mémoire. L'homme qui m'a tout appris, qui m'a inculqué ses valeurs et m'a sommé de ne jamais abandonner. Je lui avais promis de ne jamais arrêter d'apprendre et c'est ce que je compte faire. Pour ma mère, je ne trouve pas de mots. Par sa seule voix, elle m'a aidé durant les moments durs de cette thèse et félicité durant les épisodes de bonheur. Elle pleurait quand j'étais triste et riait aux éclats quand j'étais content. Ma force c'est d'elle que je la puisais et puiserai encore.

Je sais que j'ai beaucoup écrit, peut-être même un peu trop mais je tiens à retourner la faveur à ceux qui m'ont aidé, au moins par reconnaissance écrite. Néanmoins, n'étant qu'humain, peut-être ai-je omis des personnes. Donc, merci à ceux qui de près ou de loin m'ont accordé de leur temps ou égayé ma journée.

Résumé

Nous assistons aujourd'hui à une explosion de données manipulées par les systèmes informatiques. Des mesures de capteurs aux publications postées sur les réseaux sociaux, le flux de données est sans précédent. Un type particulier de ces données sont les journaux d'évènements que stockent les systèmes d'information dans le cadre d'exécution de différents processus. Le Process Mining est une discipline de recherche relativement jeune qui vise à extraire des connaissances à partir de ces journaux. Une des tâches les plus importantes est la construction de modèles décrivant le cheminement du processus. Ces modèles peuvent être structurés avec des chemins d'exécution bien clairs ou alors plus en "spaghetti" avec des branchements complexes et des structures de choix.

Différents algorithmes ont été proposés pour découvrir des modèles de processus de bout en bout mais peu réussissent à apprivoiser les processus non structurés. Pour ces cas particuliers, des techniques qui extraient des informations plus granulaires sont préconisées. Par exemple, à travers l'abstraction des évènements et de leurs relations ou alors par l'éclatement des journaux en clusters plus faciles à gérer.

C'est dans ce cadre que s'inscrit notre travail. Il tourne autour de la notion de patterns comportementaux. Il s'agit de modèles relativement petits qui décrivent des fragments importants dans l'exécution du processus; la mesure de l'importance étant la fréquence. Premièrement, nous proposons un algorithme rapide et robuste pour les extraire avec des stratégies d'élagage, plusieurs optimisations et garantissant certaines propriétés sur les patterns découverts. L'analyste peut alors avoir une vue d'ensemble sur le processus en étudiant les comportements fréquents qu'il contient.

En deuxième lieu, nous proposons un framework d'analyse des patterns découverts qui prenne en compte le contexte d'exécution. Il s'agit d'attributs accompagnant les journaux d'évènement et donnant des informations supplémentaires sur chaque exécution. Nous proposons une définition formelle des contextes, découvrons des patterns dans chaque contexte et définissons des types particuliers de pattern qui se chevauchent entre contextes. Cette approche permet une étude plus poussée des patterns découverts hors contexte mais aussi l'extraction de patterns complètement nouveaux nichés à l'intérieur des contextes. De plus, nous proposons des règles comportementales ainsi qu'une analyse de causalité entre attributs et occurrence des fragments. Une méthodologie qui sert de "guide d'utilisation" du framework est également fournie.

Finalement, nous apportons plusieurs améliorations sur l'algorithme initial de découverte classique ou hors contexte des patterns comportementaux. Elles viennent réduire encore plus les temps d'exécution et corriger le problème de difficulté d'analyse non contextuelle des fragments. Premièrement, nous avons proposé un algorithme incrémental qui permet d'évaluer la fréquence des patterns en profitant des informations déjà recueillies sur d'autres patterns augmentant ainsi nettement la vitesse d'exécution. De même, nous construisons une étape de post-traitement qui vient réduire le nombre de patterns découverts par l'élimination des redondances. Par la suite, ce dernier ensemble de patterns est affiché dans un graphe intéractif et intuitif. Il propose des relations intéressantes entre patterns et offre une vue globale dessus qui permet leur exploration.

L'algorithme initial, le framework et l'algorithme amélioré (ou avancé) ont été évalués sur des journaux réels et ont prouvé être efficients et efficaces. Dans de futurs travaux, nous prévoyons de chercher des patterns dans la dimension ressources plutôt que sur les structures de contrôles du processus. Nous voudrions aussi exploiter des framework de calcul distribué afin de gérer les journaux les plus volumineux. Finalement, nous comptons utiliser les patterns construits comme attributs dans des tâches de Machine Learning.

Résumé Étendu

Introduction

Contexte et Motivation

Nous assistons aujourd'hui à une explosion du nombre de données produites, stockées, manipulées et traitées dans les systèmes informatiques. Ce phénomène appelé "Big Data" se manifeste par cinq caractéristiques appelées les "5V" du Big Data [81]: le volume ou la taille des données, leur variété (structurées comme les bases de données relationnelles, non structurées comme du texte brut ou alors semi-structurées à mi-chemin entre les deux), la vélocité dans le cas de flux de données, la véracité ou l'incertitude sur les données dues aux facteurs incontrôlables et finalement la valeur de ces dernières et leurs capacité à dégager des informations utiles, pertinentes et effectivement exploitables. La composante valeur est donc l'ultime objectif et la raison du grand intérêt qu'on porte au Big Data.

La Science des Données, considérée le quatrième paradigme de la science, vient concrétiser cette valeur à travers des méthodes statistiques, de fouille et de visualisation permettant de réaliser des prédictions, classifications, catégorisations et d'extraire diverses formes de renseignements utiles.

D'un autre côté, dans le secteur commercial, les systèmes d'information sont étroitement liés aux processus métiers. Ces derniers sont un ensemble d'activités corrélées et exécutées dans le but d'atteindre un objectif métier tel qu'un produit ou un service. Une instance de processus ou un cas se définit comme une exécution unique du processus. Comme le rôle des systèmes d'information est de collecter, d'enregistrer et de traiter de l'information, chaque instance est représentée par une séquence d'évènements stockée et identifiée de manière unique dans un journal d'évènement. On appelle une telle séquence une trace. L'évènement renvoie à l'activité exécutée, à son horodatage, parfois à la ressource qui a entrepris la tâche mais aussi à des attributs donnant des informations sur l'environnement ou sur le contexte de l'exécution. Notez que ce dernier type d'attributs peut

également accompagner les traces [82].

Le Process Mining est une discipline relativement jeune qui extrait des informations utiles à partir des journaux d'exécution des processus s'intégrant ainsi dans le thème général de la Science des Données. Elle est divisée en quatre branches principales. La première (Process Discovery) vise à extraire des modèles décrivant l'exécution des processus, la deuxième (Conformance Checking) étudie la conformité du journal par rapport à un modèle pré-établi , la troisième (Process Enhancement) utilise les journaux pour procéder à des améliorations, modifications et extensions sur des modèles déjà construits et enfin la dernière (Predictive Process Monitoring) tente de délivrer des informations sur l'état futur des processus. La première branche est la plus étudiée. Divers algorithmes ont déjà été proposés afin d'extraire des modèles de bout-en-bout représentant l'entièreté du processus gouvernant le journal. On peut citer α-miner [83], Flexible Heuristics Miner [88] et Genetic Miner [16]. Même si ces algorithmes offrent un bon moyen d'analyse des processus structurés, à comprendre, ceux avec des chemins d'exécutions clairs et simples tel que celui présenté dans Fig. 1.1, ils n'arrivent pas à apprivoiser les modèles les plus complexes qu'on appelle "flexibles", "non structurés"ou "spaghetti" dont Fig. 1.2 est un exemple. En effet, ces derniers se présentent sous la forme de chemins d'exécutions enchevêtrés, difficile à appréhender et comportant de nombreuses structures de contrôle telles que les choix et les boucles. Pour ce type particulier, d'autres méthodes ont été proposées. Par exemple,le clustering des traces en groupes homogènes qui se traduisent chacun de manière isolée par des processus structurés [13, 15, 38, 73] ou l'application de différents niveaux d'abstraction pour diminuer l'effet "spaghetti" [39].

Un travail intéressant qui a constitué l'inspiration pour notre recherche est la fouille de patterns comportementaux ou la fouille de LPMs [80, 78]. Il s'agit de modèles de processus qui capturent des épisodes d'exécution relativement petits et fréquents dans le journal. L'idée de base est illustrée dans Fig. 1.3. Pour le journal d'exemple, décrivant les étapes de traitement d'un patient dans un hôpital, un algorithme de découverte traditionnel tel que le Flexible Heuristics Miner (FHM) [88] produit un modèle complexe. Cependant, on peut observer que les traces présentent un pattern comportemental spécifique : une exécution de l'activité *BT* suivie de *CO* et *RB* en parallèle. La détection d'un tel pattern permet de comprendre les régularités dans l'exécution des processus. Notez cependant qu'il ne peut pas être détecté en utilisant les techniques standard de fouille de patterns séquentiels, telles que PrefixSPAN [61] car celles-ci rateraient des dépendances comportementales complexes telles que la concurrence et les choix exclusifs.

Cependant, ces travaux pionniers sur l'extraction de patterns comportementaux ou LPMs présentent plusieurs failles qui ont un impact sur les temps d'exécution et sur la qualité des patterns dérivés. Dans cette thèse, nous améliorons leurs extractions tout en veillant à exploiter tous les renseignements qu'ils peuvent offrir. La section suivante décrit nos objectifs de recherche.

Objectifs de Recherche

Les algorithmes existants [80, 78] souffrent de l'imprécision et de la redondance des patterns extraits, ainsi que d'un effort de calcul comparativement élevé. En effet, même si certains comportements sont fréquents, les modèles peuvent capturer (i) seulement une partie du comportement fréquent

(c'est-à-dire qu'ils ne sont pas maximaux), ou (ii) une combinaison de comportements fréquents et de comportements non fréquents (on appelle de tels modèles non compacts). Par exemple, dans Fig. 1.3, le pattern *seq(BT, and(CO,RB))* est fréquent. On peut affirmer que la découverte d'autres patterns tels que *seq(BT,CO))* et *seq(BT, xor(CO,I))* n'apporterait aucune nouvelle information sur le processus. Il est donc suffisant de découvrir le premier. En même temps, les algorithmes existants souffrent d'un temps d'exécution élevé car les fréquences des patterns candidats sont évaluées sur le journal en entier. Le premier objectif de recherche que nous définissons est donc le suivant :

- **G1:** *Optimiser le temps d'exécution de l'extraction des patterns comportementaux tout en évitant les arbres inintéressants.*

Une fois les patterns comportementaux découverts, un deuxième objectif consiste à en tirer toutes les informations possibles.

- **G2:** *S'inspirer des domaines fondamentaux tels que la fouille de patterns fréquents et la fouille de patterns séquentiels pour découvrir d'autres utilisations des patterns comportementaux utiles à l'analyse des processus.*

Sur un sujet orthogonal, les algorithmes existants sont limités à une seule dimension d'étude, à savoir, le flux de contrôle, c'est-à-dire les interdépendances entre les activités. Cependant, la dimension des données ou le contexte d'exécution des activités et des instances est de la plus haute importance. D'où l'objectif de recherche suivant :

- **G3:** *Inclure la dimension données dans la procédure de fouille*.

Contributions

Pour atteindre l'objectif formulé dans **G1**, nous proposons COBPAM (COmbination Based PAttern Mining algorithm) [5], un nouvel algorithme basé sur une opération de combinaison pour extraire des patterns formalisés en tant qu'arbres de processus [46] (une notation de modélisation de processus) et ce à travers une approche de génération et de test. Elle identifie tous les arbres dont le comportement peut être retrouvé dans un certain nombre de traces du journal des événements, empruntant ainsi la notion bien établie de support de patterns dans les bases de données de séquences [36]. De plus, la métrique de précision connue du Conformance Checking [82] est utilisée pour évaluer le degré de matérialisation d'un arbre (si l'arbre apparaît sous les nombreuses formes permises par le modèle, alors il est fortement présent dans le journal). Par ailleurs, COBPAM définit un ordre partiel qui permet d'extraire uniquement des patterns maximaux et compacts. Il explore également l'espace de recherche en utilisant des stratégies d'élagage tout en évaluant la fréquence des patterns sur un nombre de traces limité. Tout cela le rend très efficient mais aussi efficace.

D'autre part, comme il est de pratique dans la fouille de patterns séquentiels, les patterns renvoyés par COBPAM pourraient servir dans une autre procédure d'analyse de données. A savoir, l'étude des associations entre les patterns eux-mêmes. Il s'agit d'une évolution naturelle pour COBPAM avec l'introduction de règles comportementales équivalentes aux règles séquentielles. Dans

l'exemple ci-dessus, la règle comportementale : $BT \rightarrow et(CO,RB)$ indiquerait que *BT* ne peut apparaître sans que *et(CO,RB)* ne le suive et *et(CO,RB)* ne peut pas apparaître sans que *BT* ne le précède. L'exploration de règles comportementales remplit l'objectif **G2**.

Dans une autre contribution, nous nous intéressons à l'objectif **G3** et étendons COBPAM avec une perspective de données. Considérons le journal d'exécution enrichi d'attributs dans Fig. 1.4. Il y a deux attributs représentant des informations contextuelles : le niveau de revenu du patient (faible ou élevé) et le groupe d'âge (<70 ou 70+). Dans un tel journal, une approche de découverte de patterns qui ne tient pas compte du contexte est grandement limitée en termes d'informations que l'on peut obtenir sur le processus. Les épisodes de comportement du processus qui sont communs pour un contexte d'exécution spécifique, mais qui ne sont pas fréquents dans tous les contextes, ne sont pas détectés. Par exemple, si l'on considère uniquement les patients à faibles revenus, on observe un modèle où *BT*, suivi de *SW*, est à nouveau suivi par la concurrence entre *CO* et *RB*. La détection de ce modèle contextuel fournit des renseignements pertinents. Ici on observe que les patients à faibles revenus ont tendance à discuter de la politique de remboursement avec un travailleur social. Cette information passerait inaperçue si l'on négligeait les attributs contextuels. Ainsi, les patterns contextuels permettent une analyse granulaire et fine des corrélations entre les facteurs contextuels et l'exécution du processus. De plus, on pourrait s'intéresser à répondre à des questions telles que : "Quels patterns comportementaux sont fréquents chez les patients à faible revenu exclusivement ?" ou "Quels sont les patterns comportementaux fréquents parmi les patients à hauts revenus, quel que soit leur âge ?". Pour ce faire, on peut définir différents types de fréquence. De fait, en répondant à ces questions, les données sont prises en compte de manière intrinsèque dans le processus d'extraction et les dépendances entre les données et les patterns sont découvertes sous forme de modèles contextuels.

Dans le cas où un pattern apparaît exclusivement dans une population, il est naturel de se demander si les attributs de cette population causent l'apparition de ce pattern. La corrélation et la causalité sont différentes. La première signifie que deux variables sont liées ou dépendantes l'une de l'autre, tandis que la causalité signifie qu'un changement dans la variable de cause entraîne un changement dans la variable de résultat. Nous proposons ainsi une étude de la causalité entre les attributs du journal et l'apparition d'un pattern, inspirée des études de cohortes rétrospectives.

En résumé, nous étendons COBPAM avec un framework d'analyse sensible aux données (DAF pour Data-aware Analysis Framework) [6] qui construit des dépendances, des causalités et des associations autour de patterns. Il inclut un algorithme CCOBPAM (Contextual COBPAM) qui découvre des patterns contextuels ; autrement dit, en relation avec des contextes. Il offre aussi la possibilité d'extraire des règles comportementales ou causales et définit une méthodologie complète sur la façon d'utiliser efficacement le framework pour transformer les résultats qu'il produit en informations utiles.

Enfin, après avoir proposé COBPAM qui a déjà réduit de manière significative les temps d'exécution, nous avons entrepris d'améliorer encore plus les performances, toujours dans l'esprit de satisfaire l'objectif **G1**. Ainsi, nous avons introduit ACOBPAM (Advanced COBPAM) qui utilise les alignements des arbres plus anciens pour évaluer les fréquences des arbres plus complexes. Notons que les alignements sont un outil qui permet d'affirmer qu'un pattern apparaît dans une trace. Par

exemple, ils permettent d'affirmer que le pattern dans Fig. 1.3c apparaît dans *trace 1* du journal dans Fig. 1.3a. De plus, le nouvel algorithme inclut une étape de post-traitement qui réduit le nombre d'arbres retournés par COBPAM en éliminant la redondance de manière encore plus restrictive que dans l'algorithme classique. Pour cela, de nouveaux concepts comme l'équivalence des arbres de processus ou la maximalité généralisée sont définis.

D'autre part, dans une dernière tentative pour satisfaire davantage **G2**, nous avons proposé une analyse approfondie des modèles retournés après le post-traitement. En effet, nous avons imaginé de nouvelles relations ou dépendances entre les patterns finaux. Le tout a été assemblé sous la forme d'une carte ou d'un graphe où les patterns sont des nœuds et les arêtes, des relations. Interactif et exhaustif, il permet de tirer le maximum de renseignements des patterns découverts en offrant une vue globale et navigable.

Nos contributions [5, 6] ont été publiées respectivement dans CAISE (International Conference on Advanced Information Systems Engineering) et IEEE TKDE (IEEE Transactions on Knowledge and Data Engineering) avec la collaboration du Pr. Matthias Weidlich. Le troisième travail concernant ACOBPAM est en cours de soumission à TKDE.

Définitions

Définition (Cas)**.** Soit C l'univers des cas (l'ensemble de tous les identifiants de cas possibles). Un journal d'événements est composé de cas. Chaque cas a des attributs. Pour un cas particulier $c \in C$ et $n \in ANC$ (ANC l'ensemble des noms d'attributs), un nom d'attribut, $\#_n(c)$ dénote la valeur de l'attribut n pour le cas c. Si le cas n'a pas de valeur pour un attribut n, on utilise la valeur nulle : $\#_n(c) = \perp$. Un attribut particulier obligatoire pour un cas est sa trace associée $\#_{trace}(c)$ et nous y référons par $\hat{c} = \#_{trace}(c)$.

Définition (trace, journal d'évènements)**.** Soit A un ensemble d'*activités* et A^* l'ensemble de toutes les séquences sur A. Un *journal d'évènements* L est un ensemble de cas c où l'attribut de trace est donné par $\hat{c} \in A^*$, une séquence d'activités. $|L|$ désigne la taille de L, c'est-à-dire le nombre de cas qu'il contient.

Définition (Arbres de Processus)**.** Un *arbre de processus* est un arbre ordonné où les feuilles représentent des activités et les nœuds internes des opérateurs. En considérant un ensemble d'activités A, un ensemble d'opérateurs binaires $\Omega = \{seq, and, loop, xor\}$ (séquence, concurrence, boucle et choix exclusif respectivement), un arbre de processus est défini de manière récursive comme :

- $a \in A$ est un arbre de processus.
- considérant un opérateur $x \in \Omega$ et deux arbres de processus P_1, P_2, $x(P_1, P_2)$ est un arbre de processus ayant x comme racine, P_1 comme fils gauche et P_2 comme fils droit.

La profondeur d'un nœud (activité ou opérateur) dans l'arbre est la longueur du chemin vers sa racine. La profondeur de l'arbre correspond à la profondeur maximale de n'importe lequel

de ses nœuds. $\Sigma(P)$ est le langage du modèle de processus; autrement dit, les traces qu'il peut générer. Par ailleurs, comme la taille du langage d'un arbre contenant une boucle est infini, nous considérons le n-langage où on se limite à traverser chaque boucle au plus une fois.

Fouille de Patterns Comportementaux

Opérations et Structures Algébriques sur des Arbres de Processus

Dans le but de construire des arbres de processus de manière incrémentale, nous proposons de combiner deux arbres composés de n activités pour obtenir un autre de $n + 1$ activités. Ces arbres doivent être similaires sauf au niveau d'une seule feuille. Nous imposons des conditions sur la position de ces feuilles.

Définition (Feuille de Combinaison Potentielle)**.** Soit un arbre de processus P de profondeur i. Une feuille a de profondeur d est appelée *feuille de combinaison potentielle* si $d \geq i - 1$ et il n'existe aucune autre feuille b de profondeur d' sur la gauche de a tel quel $d > d$

Définition (graines)**.** Deux arbres de processus P_1 et P_2 sont appelés *graines* si P_1 contient une feuille de combinaison potentielle a et P_2 en contient une autre a' tel que quand on remplace a' par a dans P_2, on obtient P_1. Seules des graines peuvent être combinées.

Définition (Opération de Combinaison)**.** Une *combinaison* de deux graines P_1 et P_2 *à travers* un opérateur x est une opération générant deux arbres de processus. À partir de P_1, la feuille de combinaison a est remplacée par l'opérateur x, dont les enfants sont fixés à a et a'. a devient l'enfant de gauche dans un des arbres résultants, et l'enfant de droite dans l'autre. a et a' sont appelés les feuilles de combinaison et x est appelé l'opérateur de combinaison.

Fig. 3.1 montre un exemple de cette opération. Par ailleurs, grâce aux conditions imposées sur les feuilles de combinaison potentielles, obtenons le théorème suivant :

Théorème. Pour un arbre de processus P de profondeur $i \geq 1$, il existe une paire unique de graines P_1 et P_2, dont la combinaison par un opérateur x donne P. P_1 et P_2 sont appelés 'les' graines de P et x est appelé l'opérateur de définition de P.

En appliquant le théorème en cascade à partir d'un arbre P, on obtient :

Définition (Arbre de construction)**.** Étant donné un arbre de processus P de profondeur $i \geq 1$, nous définissons son *arbre de construction*. Les nœuds de cet arbre sont des arbres de processus : La racine est P, les feuilles sont des arbres avec des nœuds d'activité unique ; les enfants d'un nœud interne sont ses graines.

Fig. 3.1b exemplifie l'arbre de construction de l'arbre de processus *seq(BT, et(CO, RB))*.

Définition (Graphe de construction)**.** Nous définissons le *graphe de construction* sur l'ensemble des activités A. Il s'agit d'un graphe acyclique dirigé. Son ensemble (infini) de nœuds est donné

par tous les arbres de processus possibles. Une arête est définie entre les nœuds n_1 et n_2, si n_1 est une graine de n_2. On dit que n_2 contient n_1 par l'opérateur de définition de n_2.

Pour identifier un arbre, COBPAM utilise le concept de mot représentatif.

Définition (Mot représentatif)**.** À chaque arbre de processus P est attribué un *mot représentatif* $RW(P)$. Il s'agit d'une séquence de caractères construite par impression des activités et opérateurs pendant une traversée pré-ordre de ses nœuds.

Par exemple, le mot représentatif de $seq(BT, et(CO, RB))$ est '(BT (CO RB et) seq)'.

Métriques de Qualité

Cette section définit les métriques qu'on utilise pour évaluer la qualité des patterns comportementaux extraits d'un journal d'évènements. Nous considérons qu'une trace $\hat{c}$ contient un arbre de processus P s'il existe un mot $\omega \in \Sigma(P)$ du langage de P tel que $\omega \leq \hat{c}$. Par exemple, Trace 1 affiche l'arbre de processus dans Fig. 1.4c. En appliquant une technique de Conformance Checking appelée alignement (un algorithme A^*), il est possible de définir les fonctions suivantes : $\epsilon(\hat{c}, P)$, une fonction booléenne, qui retourne vrai quand la trace $\hat{c}$ contient P et une fonction $\upsilon(\hat{c}, P)$ qui retourne le comportement exact de P présent dans la trace $\hat{c}$. À partir de là, nous définissons:

Définition (Projection)**.** Une *projection* est un sous-ensemble d'un journal d'événements L associée à un arbre de processus P qui contient les cas dont les traces peuvent être alignées avec P :

$$proj(P, L) = \{c \in L \mid \epsilon(\hat{c}, P) = 1\}.$$

Définition (Fréquence et support)**.** Étant donné un journal d'événements L, la *fréquence* d'un arbre de processus P est le nombre de cas qui présentent son comportement :

$$frequency(P, L) = \sum_{c \in L} \epsilon(\hat{c}, P) = |proj(P, L)|.$$

Son *support* est la fréquence sur la taille du log :

$$support(P, L) = \frac{frquence(P, L)}{|L|}.$$

Définition (Précision)**.** Étant donné un journal d'événements L, la *prcision* d'un arbre de processus P est le rapport entre le comportement vu dans le journal et tous les comportements autorisés par le modèle. Si P ne contient pas d'opérateurs de boucle, il est défini comme :

$$precision(P, L) = \frac{|\{\upsilon(\hat{c}, P) \mid c \in L \wedge \epsilon(\hat{c}, P) = 1\}|}{|\Sigma(P)|}.$$

Si P contient des opérateurs de boucle, son langage sera infini et sa précision tendra donc vers zéro. Dans ce cas, on utilise le n-langage de P :

$$precision(P, L) = \frac{|\{\upsilon(\hat{c}, P) \mid c \in L \wedge \epsilon(\hat{c}, P) = 1\}|}{|\Sigma_n(P)|}$$

Découverte de Patterns Comportementaux avec COBPAM

L'idée de l'algorithme est de parcourir le graphe de construction grâce à l'opération de combinaison. Chaque arbre est évalué contre un sous-ensemble du journal où le pattern pourrait exister. Nous employons des règles d'élagage et de projection pour optimiser les temps d'exécution.

Une Propriété de Monotonicité

L'opération de combinaison remplace une feuille de combinaison potentielle par un sous-arbre représentant une partie d'un comportement qui soit étend le comportement de l'arbre original lors de l'utilisation de l'opérateur de choix, soit le contraint lors de l'utilisation d'un opérateur de séquence, de boucle ou de concurrence qu'on appelle restrictif. Dans ce dernier cas, le comportement partagé entre un arbre de processus et ses graines représente un contexte auquel le comportement supplémentaire est joint. Par conséquent, si une trace ne présente pas le contexte, il n'est pas nécessaire d'évaluer le comportement supplémentaire.

De ce qui précède, il s'ensuit que, si l'une des graines n'est pas fréquente, il n'est pas nécessaire d'évaluer l'arbre, car il sera également non fréquent. C'est là, notre **première règle d'élagage**.

Arbres de Processus Compacts et Maximaux

Nous orientons notre recherche de patterns comportementaux vers les arbres de processus utiles du point de vue de l'analyse, à savoir, compacts et maximaux.

Définition (Arbre de processus compact)**.** Étant donné un journal d'événements L, un arbre de processus P est *compact*, s'il satisfait toutes les conditions suivantes :

1. L'arbre P ne présente pas l'opérateur de choix comme nœud racine. Si cette condition est invalide, l'arbre de processus serait l'union de comportements complètement séparés. Bien que cela puisse résulter en un arbre fréquent, il a sans doute peu d'intérêt.
2. P ne résulte pas d'une combinaison par un opérateur de choix, où l'une des graines est fréquente. En effet, si un arbre P_1 est fréquent, le combiner avec tout autre arbre P_2 à travers l'opérateur de choix résulte en un arbre fréquent. Cette addition n'a donc aucune valeur.
3. P ne contient pas d'opérateur de boucle $loop(P_1, P_2)$, de sorte que seul le comportement de P_1 apparaît dans L. Alors que d'un point de vue de langage, ce pattern existe en effet dans la trace, la notion de boucle perd son sens sans l'existence de P_2.

Notez que de la condition (2), nous dérivons immédiatement une **deuxième règle d'élagage** : lorsqu'on effectue une combinaison par l'opérateur de choix, les deux graines doivent être non fréquentes.

En plus de la compacité, les patterns doivent être maximaux. Cette propriété découle de la monotonicité. Si un pattern défini par un opérateur restrictif est fréquent alors ses graines le sont aussi. Il est donc suffisant de retourner le pattern complexe, dit maximal, car il contient toutes les informations de fréquence. D'où la définition suivante:

Définition (Arbre de processus maximal)**.** En considérant tous les patterns comportementaux de profondeur au plus égale à i, un pattern est *maximal*, s'il est fréquent et non contenu par un opérateur restrictif dans un autre arbre fréquent de profondeur inférieure ou égale à i.

Règles de Projection

La complexité d'exécution est régie par la taille du graphe de construction, qui est exponentielle par rapport au nombre d'activités, et par la taille du journal utilisé pour évaluer la qualité des arbres. Pour faire face à ce dernier point, nous présentons les règles de projections suivantes:

- Lors de l'exécution d'une combinaison par un opérateur restrictif, le comportement associé aux arbres résultants ne peut apparaître que dans l'intersection des projections des graines. Par conséquent, les métriques de qualité sont calculées uniquement sur la base de ladite intersection. De plus, la taille de l'intersection des projections des graines représente une limite supérieure de la fréquence des arbres résultants. Il en résulte une **troisième règle d'élagage** : Si la limite supérieure est inférieure au seuil de fréquence, la combinaison n'est plus considérée.
- Lorsqu'on effectue une combinaison par l'opérateur de choix, la projection associée aux arbres résultants est l'union des projections des graines.

L'algorithme COBPAM

L'idée de l'algorithme COBPAM est de construire de manière incrémentale des ensembles d'arbres de processus. Compte tenu des règles d'élagage, nous maintenons deux types d'ensembles, l'un contenant des arbres fréquents et l'autre non fréquents. Le premier type sert de base aux combinaisons effectuées par les opérateurs restrictifs, tandis que le second sert aux combinaisons basées sur le choix. Tous les arbres d'un ensemble sont identiques, à l'exception d'un seul nœud feuille, ce qui permet de les combiner.

L'algorithme s'articule autour de deux fonctions, *addFreq*, définie dans Alg. 1, qui ajoute l'arbre de processus P à un ensemble Γ contenant uniquement des arbres fréquents et *addInfreq*, définie dans Alg. 2, qui ajoute P à un ensemble γ contenant uniquement des arbres non fréquents. Par Θ, nous désignons en outre l'ensemble des arbres maximaux, compacts et fréquents, qui représente le résultat réel de notre algorithme. Notez que l'on peut utiliser une profondeur maximale de récursion d limitant ainsi la profondeur des arbres découverts et forçant la terminaison.

Framework d'Analyse Sensible aux Données (DAF)

Règles Comportementales

Une première perspective pour répondre à l'objectif **G2** est l'analyse de corrélations entre les patterns eux-même, ce qui permettra une compréhension plus profonde du cours du processus. En particulier, nous considérons un arbre fréquent dont la racine est une séquence. Pour cette

configuration, autrement dit, $P = seq(P_1, P_2)$ avec P_1, P_2, deux sous-arbres, nous étudions l'existence d'une règle comportementale $P_1 \rightarrow P_2$ semblable à une règle d'association. À travers la fréquence du pattern, nous déduisons que P_2 suit souvent P_1. Il est intéressant de se demander alors s'ils sont fortement associés, c'est à dire que l'un n'apparaît jamais sans l'autre validant ainsi la règle comportementale ou au cas contraire faiblement associés. On utilise la mesure du odds ratio pour évaluer le degré de satisfaction de la règle. Par ailleurs, si le nœud racine est une concurrence, la règle peut se traduire par une apparition significative des patterns parallèlement et jamais séparément.

COBPAM Contextuel

Les Contextes

Définition (Journal d'événements contextuel)**.** Considérons un journal d'événements simple L et une relation $\mathcal{R}(D_1, \ldots, D_n)$, telle que le i-ième attribut est un attribut de cas nommé $d_i \in ANC$ (l'ensemble des noms d'attributs), dont le domaine est D_i et $\forall c \in L$, $\#_{d_i}(c) \neq \perp$. Un journal d'événements contextuel est alors une paire (L, χ) où χ est une fonction qui associe à chaque cas c de L le tuple de $\mathcal{R}$ suivant : $\chi(c) = (v_1, \ldots, v_n)$ avec $v_i \in D_i$, $v_i = \#_{d_i}(c)$.

Pour définir la notion de contexte, nous introduisons D'_i comme une extension du domaine D_i en utilisant un symbole dédié et unique '$*$'. On établit ensuite un ordre d'inclusion $\subset_{D'_i} = \{(v_i, *) \mid v_i \in D_i\}$, avec $\subseteq_{D'_i} = \subset_{D'_i} \cup \{(v_i, v_i) \mid v_i \in D_i\}$ comme sa version réflexive. Le sens du symbole '$*$' est qu'il regroupe n'importe quelle valeur possible $v_i \in D_i$. La paire $(D'_i, \subset_{D'_i})$ définit une hiérarchie d'inclusion $\mathcal{H}(d_i)$ sur l'attribut de données d_i. Un exemple de hiérarchies est donné en Fig. 4.2. Par la suite, un contexte est défini comme un tuple $(v_1, \ldots, v_n)$ avec $v_i \in D'_i, \forall i \in \{1, 2, \ldots, n\}$. Les contextes suivent un ordre d'inclusion $\leq$, tel que deux contextes $C = (v_1, \ldots, v_n), C' = (v'_1, \ldots, v'_n)$ sont ordonnés, noté $C \leq C'$, si $v_i \subseteq_{D'_i} v'_i \ \forall i \in \{1, 2, \ldots, n\}$. Si $\exists 1 \leq i \leq n, v_i \subset_{D'_i} v'_i$, alors le contexte C' est dit plus général que C, tandis que C est dit plus spécifique et on écrit $C < C'$ (ordre d'inclusion strict). Un contexte $C = (v_1, \ldots, v_n)$ est atomique, si $v_i \in D_i, \forall i \in \{1, 2, \ldots, n\}$. Nous définissons également une décomposition d'un contexte C comme l'ensemble non vide des contextes atomiques qui sont plus spécifiques que C.

Patterns Comportementaux Contextuels

Définition (Journal d'événements associé)**.** Soit (L, χ) un journal d'événements contextuel et $C = (co_1, \ldots, co_n)$ un contexte. Le journal d'événements associé au contexte C est un journal d'événements contextuel (L', χ') avec $L' \subseteq L$ et tel que le cas $c \in L$ fait partie de L' si et seulement si $\chi(c) = (v_1, \ldots, v_n)$ et $\forall v_i, v_i \subseteq_{D'_i} co_i$. En outre, χ' est la restriction de χ à L' tandis que la taille du contexte C correspond à la taille de son journal associé.

Un arbre de processus est dit **C-fréquent** ou fréquent dans C, s'il est fréquent dans son journal associé. En pratique, nous distinguons deux types de patterns pour un contexte non atomique :

- **Pattern C-général**: Ils sont fréquents dans C et dans chaque descendant de C. Appliqués à notre exemple courant (voir Fig. 1.4), ils seraient des modèles répondant à des questions

telles que : quels sont les patterns fréquents dans la population à faibles revenus, quel que soit l'âge ?

- **Pattern C-exclusif** Les patterns sont C-fréquents uniquement dans C et ses descendants. Dans notre exemple, il s'agirait de modèles répondant à la question suivante : quels sont les patterns comportementaux exclusivement présents chez les patients de plus de 70 ans, quel que soit leur niveau de revenu ?

Si un pattern est C-fréquent dans au moins un contexte atomique et non C-général ou C-exclusif ailleurs, il est appelé **AC-frequent**. Si un pattern est C-fréquent dans le contexte le plus général (racine), c'est-à-dire que son journal associé est le journal des événements original, il est appelé **log-fréquent**.

En adoptant le raisonnement adopté dans la fouille de patterns séquentiels contextuels [63], nous observons ce qui suit : un arbre P est C-général si, et seulement si, il est fréquent dans les contextes atomiques de la décomposition de C. De manière analogue, un pattern P est C-exclusif si, et seulement si, il est fréquent dans les contextes atomiques de la décomposition de C et non fréquent ailleurs. Par conséquent, la découverte des deux types de patterns, C-général et C-exclusif dans tous les contextes doit commencer par la découverte de patterns dans les contextes atomiques. Il suffit ensuite de vérifier les conditions ci-dessus pour détecter quel type de fréquence s'applique.

Découverte de Relations Causales entre Données et Patterns

Lorsqu'un pattern est C-exclusif dans un contexte C, on peut déduire qu'il y est fréquent de manière exclusive. Considérons un tel arbre où C inclut un seul attribut d fixé à une valeur précise v (contraint), les autres attributs étant génériques (affichant la valeur '$*$'). Cela veut dire que la présence du pattern est indépendante des attributs génériques et que son occurrence est corrélée à la valeur v. On est alors en droit de se demander si cette valeur cause l'apparition du pattern. La causalité est différente de la corrélation dans le sens où un changement de la valeur causante, ou l'exposition, entraîne un changement dans le phénomène causé ou la variable outcome.

Dans ce qui suit, nous développons une méthode pour vérifier une telle causalité en s'inspirant des études de cohorte rétrospectives [51]. L'exemple suivant illustre chaque étape. Nous supposons l'existence de trois attributs, d, a, b, dans le journal contextuel avec deux modalités pour d (v et v') et trois pour chacun des attributs a et b : $a_i, b_i, \forall 1 \leq i \leq 3$. Il s'ensuit les étapes suivantes :

(1) Nous transformons la relation $\mathcal{R}(D_1, D_2, D_3)$ qui capture les contextes possibles en termes de valeurs d'attributs en une relation booléennes $\mathcal{B}(B_1, \ldots, B_8)$. Ici, les modalités de chaque attribut d, a, b sont transformées en un ensemble de variables indicatrices booléennes. od mise à vrai (resp. od') si $d = v$ (resp. $d = v'$) et $\forall 1 \leq i \leq 3$ oa_i (resp. ob_i) mise à vrai si $a = a_i$ (resp. $b = b_i$).
(2) La variable booléenne od qui représente la valeur v de l'attribut d est définie comme la variable d'exposition.
(3) Nous définissons une variable booléenne d'outcome t par cas c et pattern P qui est égale à $\epsilon(c, P)$.

(4) Ensuite, nous identifions parmi les attributs présents dans le journal contextuel, ceux qui sont corrélés avec la variable outcome, t, et donc les facteurs causaux possibles. Les variables booléennes associées serviront de variables contrôlées. La raison est que nous voulons évaluer si la variable *od* étant mise à vrai provoque la véracité de la variable t parmi d'autres facteurs de causalité possibles. A cette fin, nous appliquons l'odds ratio comme mesure de corrélation. Pour notre exemple, nous supposons que seules les variables, oa_1, oa_2, oa_3, ob_1, ob_2, présentent des règles d'association avec *od*.
(5) Le journal des événements est divisé en un groupe d'exposition (cas où *od* est vrai) et un groupe de non-exposition (cas restants). Les groupes sont ensuite filtrés afin de s'assurer que les variables contrôlées sont distribuées de manière égale dans dans les deux groupes et ainsi atténuer leur effet.
(6) Pour chaque combinaison existante de valeurs des variables contrôlées, nous évaluons la valeur de la variable outcome. Les cas avec un résultat positif dans le groupe exposé et négatif dans le groupe non exposé fournissent des preuves d'une relation causale (soit n_1 leur nombre). Les cas avec un résultat négatif dans le groupe exposé et un résultat positif dans le groupe non-exposé, quant à eux, fournissent des preuves contre une relation causale (leur nombre est noté n_2). Si le rapport entre le nombre de cas fournissant des preuves pour et ceux fournissant des preuves contre une relation causale, $\frac{n_1}{n_2}$, est significativement supérieur à un, on conclut l'existence de la relation de cause à effet [51].

Méthodologie d'utilisation du DAF

Nous donnons dans ce qui suit les étapes d'utilisation de notre framework pour maximiser ses bénéfices.

(1) Afin d'obtenir une première vue sur l'ensemble des régularités comportementales, des patterns sont identifiés à l'aide de l'algorithme COBPAM (ou ACOBPAM) pour un seuil de support et de précision donné.
(2) Ensuite, les attributs contextuels sont sélectionnés en tenant compte de leur sémantique et de leurs modalités (continues ou discrètes). Si nécessaire, un prétraitement est appliqué pour adapter, normaliser ou discrétiser les valeurs des attributs.
(3) Une taille minimale de contexte est définie. Les contextes qui ne la respectent pas sont écartés car leurs traces associées ne sont pas considérées représentatives.
(4) Les modèles comportementaux contextuels sont extraits par l'algorithme CCOBPAM.
(5) Suivre les directives d'interprétation, détaillées ci-dessous, afin de tirer des des informations sur le processus.
(6) Si ces directives le suggèrent, adapter le seuil de support, répéter la découverte des patterns comportementaux, et noter les changements potentiels dans l'ensemble des patterns.
(7) Enfin, la présence de règles comportementales, puis de relations causales avec les données contextuelles pour les patterns C-exclusifs sont évaluées selon les procédures ci-dessus.

Les directives d'interprétations suivantes complémentent la méthodologie. Premièrement, CCOBPAM révèle des modèles par rapport à une hiérarchie de contextes et considère différents types de fréquence qui peuvent être interprétés comme suit :

- *AC-frequent* : Le pattern est fréquent dans au moins un contexte atomique et non C-général ou C-exclusif ailleurs.
- *C-exclusif* : Le pattern est exclusivement fréquent dans le contexte et tous ses descendants, ce qui signifie que son occurrence est indépendante des populations considérées à l'intérieur du contexte.
- *C-general* : La même interprétation que pour un pattern C-exclusif s'applique. Cependant, le pattern est également fréquent ailleurs dans la hiérarchie des contextes.

Deuxièmement, nous étudions les patterns retournés à la fois par COBPAM et CCOBPAM. Les différents cas sont donnés ci-après :

1. *Un pattern log-fréquent apparaît comme un pattern C-exclusif dans le contexte racine:* Le pattern est non seulement fréquent dans l'ensemble du journal, mais aussi dans chaque contexte. Cela signifie qu'il est fortement fréquent, indépendamment de la population considérée.
2. *Un pattern log-fréquent apparaît comme C-exclusif ou C-général dans de gros contextes ou C-fréquent dans de gros contextes atomiques:* Le pattern est en fait fréquent dans certaines parties du journal qui en représentent une partie significative, ce qui le rend log-fréquent. Pourtant, il est peu fréquent dans d'autres contextes. Ainsi, l'occurrence d'un pattern qui se produit très souvent (c.à.d. qu'il est fréquent dans le journal) peut être liée assez précisément à des contextes spécifiques.
3. *Un pattern log-fréquent apparaît comme C-exclusif ou C-général dans seulement quelques petits contextes ou comme C-fréquent dans quelques petits contextes atomiques :* Le pattern est fréquent dans certaines parties du journal et presque fréquent dans d'autres parties. Cela peut indiquer qu'il est nécessaire de revoir le seuil de support choisi.
4. *Un pattern n'est pas fréquent lorsque le contexte est négligé, mais l'est dans un certain contexte:* Le comportement identifié est fréquent, mais s'applique uniquement à des contextes spécifiques, ce qui démontre la pertinence d'une approche contextuelle.

Fouille de Patterns Comportementaux Avancée

Croissance d'Alignements

L'algorithme COBPAM parcourt l'espace de patterns de manière intelligente en utilisant les règles d'élagage et de projection. Des informations sur les fréquences et sur les traces pouvant contenir les patterns sont déduites à partir des informations disponibles sur les graines. Cependant, à chaque fois qu'un arbre est évalué en utilisant l'alignement, l'entièreté de l'arbre et de la trace sont considérées. En réponse à ce problème, ACOBPAM adopte le même fonctionnement décrit plus haut sur les alignements. Plus précisément, il exploite des informations sur les alignements des graines pour construire l'alignement de l'arbre candidat. Nous appelons l'opération "croissance d'alignements". Ainsi, étant donné un arbre P et une trace $\hat{c}$, l'approche accomplit deux tâches récursivement. La première est la détection des parties du comportement de l'arbre déjà présentes dans la trace qu'on appelle contexte validé. La deuxième est de réaligner si nécessaire les autres

parties de l'arbre. Puisque on aligne une partie de l'arbre candidat sur une trace plus petite et dénuée du contexte validé, les temps d'exécution sont substantiellement réduits. En effet, l'exponentialité de l'algorithme A^* de l'alignement est amoindrie.

Post-traitement

Le nombre de patterns retourné par COBPAM peut être handicapant et limiter sévèrement la capacité d'analyse. Pour régler ce problème, nous rajoutant une opération de post-traitement une fois l'extraction terminée. Elle repose sur deux concepts: la maximalité généralisée et l'équivalence:

Maximalité Généralisée

Graines Alternatives L'opération de combinaison s'effectue toujours sur la position d'une feuille de combinaison potentielle. Cela permet de définir l'ordre partiel et ainsi parcourir le graphe de construction sans revenir sur le même arbre. Cependant, la propriété de monotonicité reste valide si l'on applique la combinaison au niveau d'une autre position. Il suffit que l'opérateur soit restrictif et les mêmes arguments qui expliquent la monotonicité resteront valides. Pour un pattern P, nous pouvons donc le décomposer au niveau de n'importe quel opérateur restrictif en deux autres arbres pouvant être combinés dans le chemin inverse. Ces arbres sont appelés graines alternatives si différentes des graines, qu'on appelle régulières, présentées plus haut. En pratique, les deux types de graines sont éliminées dans l'ensemble final de patterns.

Dans un souci de formalisation, nous définissons la fonction suivante:

Définition. Étant donné un arbre de processus P, nous définissons une fonction, f, qui associe à P un ensemble d'arbres.

- Si $P = a$ avec $a \in A$, une activité, alors $f(P) = \{a\}$.
- Si $P = x(P_1, P_2)$ avec x un opérateur restrictif et P_1, P_2 des sous-arbres enfants, $f(P)$ est donné par l'union de :
  - $f(P_1)$.
  - $f(P_2)$.
  - l'ensemble : $\{x(P'_1, P'_2) \mid P'_1 \in f(P_1), P'_2 \in f(P_2)\}$.
- Si $P = xor(P_1, P_2)$ avec P_1, P_2 les sous-arbres fils de P, alors $f(P) = \{xor(P'_1, P'_2) \mid P'_1 \in f(P_1), P'_2 \in f(P_2)\}$.

Pour un arbre de processus de profondeur $n \in \mathbb{N}^*$, les éléments dans $f(P) \setminus \{P\}$ sont l'union de la paire des graines régulières et des graines alternatives. En particulier, les graines alternatives sont définies pour les arbres de profondeur $n \geq 2$. La condition de profondeur est justifiée par l'absence de graines pour la profondeur zéro et l'absence de graines alternatives (uniquement régulières) pour la profondeur une.

Graines de boucles Lorsqu'un arbre de processus contient un opérateur de boucle alors il contient deux comportements séquentiels aussi. En effet, l'apparition d'une boucle $loop(a, b)$ implique l'existence du comportement $\langle a, b, a \rangle$. Par conséquent les comportements $seq(a, b)$ et $seq(b, a)$ existent aussi. Les arbres où l'une de ces séquences remplace ladite boucle sont éliminés. Ils sont détectés syntaxiquement.

Équivalence

Non seulement nous supprimons tout type de graines dans l'étape de post-traitement mais nous gardons aussi une seule version des arbres équivalents. De fait, il existe deux types d'équivalence.

- **Équivalence syntaxique:** Elle est le résultat de l'existence des opérateurs symétriques. En effet, si l'on considère des arbres non sensible à l'ordre des opérateurs symétriques, ils se déclinent en plusieurs manières considérant l'interchangeabilité des fils de ce genre d'opérateurs.
- **Équivalence comportementale:** Deux arbres sont comportementalemnt équivalents ont le même langage.

L'équivalence syntaxique est gérée au niveau de l'algorithme COBPAM (ou ACOBPAM) lui-même. Durant la recherche, nous imposons un ordre sur les opérateurs symétriques pour que les arbres soient acceptés dans l'ensemble finale. Ensuite dans le post-traitement, nous gardons une seule version des arbres comportementalement équivalents.

Visualisation

Dans le souci de naviguer la multitude des arbres retournés par notre algorithme, nous concevons un graphe de visualisation qui offre non seulement une vue globale et simultanée mais aussi arbore des relations intéressantes entre les patterns eux-mêmes. Pour P_1 et P_2, deux patterns dans l'ensemble final d'arbres retournés Ω, nous définissons:

Définition (Relation Follows)**.** La *relation Follows* $\mathcal{F} \subset \Omega^2$ est définie à l'aide d'une métrique de support, $support_f(P_1, P_2)$:

$$support_f(P_1, P_2) = \frac{|\{c \in L \mid \epsilon(c, P_1) = 1 \ \wedge \ \epsilon(c, P_2) = 1 \ \wedge \ \beta_h(P_1) < \beta_l(P_2)\}|}{|L|}$$

Pour un seuil τ_f, on a, $(P_1, P_2) \in \mathcal{F}$ si et seulement si :

$$support_f(P_1, P_2) \geq \tau_f$$

Définition (Relation Inter-follows)**.** Si $(P_1, P_2) \notin \mathcal{F}$, la *relation Inter-follows* $\mathcal{F}' \subset \Omega^2$ est définie à l'aide d'une métrique de support, $support_{if}(P_1, P_2)$.

$$support_{if}(P_1, P_2) = \frac{|\{c \in L \mid \epsilon(c, P_1) = 1 \ \wedge \ \epsilon(c, P_2) = 1 \ \wedge \ \beta_h(P_1) < \beta_l(P_2)\}|}{|proj(P_1) \cap proj(P_2)|}$$

Pour un seuil τ_{if}, on a, $(P_1, P_2) \in \mathcal{F}'$ si et seulement si :

$$support_{if}(P_1, P_2) \geq \tau_{if}$$

Définition (Relation Spans)**.** La *relation Spans* $\mathcal{S} \subset \Omega^2$ est définie à l'aide d'une métrique de support, $support_s(P_1, P_2)$:

$$support_s(P_1, P_2) = \frac{|\{c \in L \mid \epsilon(c, P_1) = 1 \ \wedge \ \epsilon(c, P_2) = 1 \ \wedge \ \beta_h(P_1) < \beta_h(P_2) \ \wedge \ \beta_l(P_1) > \beta_l(P_2)\}|}{|L|}$$

Pour un seuil τ_s, on a, $(P_1, P_2) \in \mathcal{S}$ si et seulement si :

$$support_s(P_1, P_2) \geq \tau_s$$

Définition (Relation Inter-spans)**.** Si $(P_1, P_2) \notin \mathcal{S}$, la *relation Inter-spans* $\mathcal{S}' \subset \Omega^2$ est définie à l'aide d'une métrique de support, $support_{is}(P_1, P_2)$:

$$support_{is}(P_1, P_2) = \frac{|\{c \in L \mid \epsilon(c, P_1) = 1 \ \wedge \ \epsilon(c, P_2) = 1 \ \wedge \ \beta_h(P_1) < \beta_h(P_2) \ \wedge \ \beta_l(P_1) > \beta_l(P_2)\}|}{|proj(P_1) \cap proj(P_2)|}$$

Pour un seuil τ_{is}, on a, $(P_1, P_2) \in \mathcal{S}'$ si et seulement si :

$$support_{is}(P_1, P_2) \geq \tau_{is}$$

Définition (Graphe de visualisation)**.** Le graphe de visualisation $G = (\Omega, \mathcal{F}, \mathcal{S}, \mathcal{F}', \mathcal{S}')$ est un multigraphe orienté où Ω est l'ensemble final de patterns ou de nœuds retenus après l'étape de post-traitement, $\mathcal{F}$ est la *relation Follows*, $\mathcal{F}'$ est la *relation Inter-follows*, $\mathcal{S}$ est la *relation Spans* et $\mathcal{S}'$ est la *relation Inter-spans*. En considérant $P_1, P_2 \in \Omega$, un nouvel arc de P_1 à P_2 est créé chaque fois que (P_1, P_2) appartient à l'une des relations précédentes. Nous appliquons une réduction transitive sur les relations *Follows* et *Spans* car ce sont des relations transitives.

Évaluation Expérimentale

Cette section est dédiée aux expérimentations qu'on a réalisé afin de prouver la faisabilité, l'efficience et l'utilité ainsi que l'intérêt de nos algorithmes. Nous commençons par présenter l'environnement et les jeux de données utilisés.

Environnement et Données

Les seuils utilisés sont les suivants: 0.7 pour le support et la précision ainsi que pour les différentes relations dans le graphe de visualisation. 2 pour la profondeur maximale des arbres et 500 pour le nombre de LPMs à découvrir concernant le LPM Miner. Par ailleurs, la taille minimale des contextes a été fixée à 50. Les jeux de données suivants ont été utilisés:

- Sepsis : un journal pour le traitement du sepsis dans un hôpital (1050 cas, 15214 évènements et 16 activités). Les attributs contextuels sont "InfectionSuspected" (une infection est-elle suspectée) et "Infusion" (a-t-on administré une infusion).
- Traffic Fines : un journal d'un système d'information pour la gestion des amendes routières (150370 cas, 561470 évènements et 11 activités). Les attributs sont "amount", la valeur de l'amende, et "VehicleClass", la catégorie du véhicule.
- WABO : un journal lié aux demandes d'autorisation de construction aux Pays-bas (1434 cas, 8577 évènements et 27 activités). Les attributs sont : "department", le département gérant la procédure, et "channel", le canal de communication.
- BPI_2019S1 et BPI_2019S2 : respectivement 30% et 40% d'un journal lié à la procédure de traitement de commandes pour une multinationale (75519 cas, 479845 évènements et 41 activités et 105962 cas, 670583 évènements et 42 activités respectivement). Les attributs sont "Item Category", la catégorie de l'article, et "Company", la filiale concernée.

Évaluation de COBPAM

Efficience

La Table 6.1 montre les temps d'exécution pour COBPAM et LPM Miner. Nous remarquons que COBPAM a de meilleures performances avec des temps d'exécution un ordre de grandeur plus petits. Pour COBPAM, ces temps dépendent de la taille du journal, le nombre d'activités et d'évènements mais aussi la complexité des patterns présents.

Évaluation Quantitative

Maintenant, nous évaluons la qualité des patterns découverts par notre algorithme et par la découverte de LPMs. À la lumières des résultats présentés dans Table 6.5 nous remarquons que, pour le journal Sepsis, parmi les 500 patterns extraits par la découverte de LPMs, 336 patterns satisfont les seuils de support et de précision fixés par COBPAM. Parmi ceux-ci, 194 ne sont pas compacts et 17 ne sont pas maximaux. Ainsi, seulement 125 des patterns dérivés par la découverte de LPMs sont maximaux et compacts, alors que COBPAM a découvert 375 patterns de ce type. Des résultats similaires sont obtenus pour l'autre jeu de données. Nous concluons que les patterns dérivés par le LPM Mining contiennent beaucoup de redondances, alors que COBPAM produit des patterns beaucoup plus pertinents, à savoir, compacts et maximaux.

Évaluation Qualitative

La Fig. 6.1 montre les arbres retournés par LPM pour Sepsis tandis que Fig. 6.2 montre ceux retournés par COBPAM pour le même log. Nous remarquons la différence entre les deux ensembles d'arbres. Par exemple, l'arbre Fig. 6.1a n'a pas été retourné par COBPAM car il n'est pas maximal. Il est contenu dans l'arbre Fig. 6.2a. De même Fig. 6.1c, Fig. 6.1d et Fig. 6.1e ne sont pas retournés car non compacts. En effet, par exemple, l'arbre Fig. 6.1c est contenu dans Fig. 6.1a qui est déjà fréquent.

Évaluation du DAF

Efficience

Table 6.6 rapporte les temps d'exécution de CCOBPAM, en ignorant le contexte (équivalent à COBPAM), avec un attribut et ensuite avec les deux attributs susmentionnés. On remarque d'abord que l'introduction de la contextualité augmente les temps d'exécution. Ensuite, nous observons des irrégularités dans les temps d'exécution en augmentant le nombre d'attributs. Ceci peut être expliqué par deux forces opposées. La première résulte du fait qu'en augmentant les nombres d'attributs, les contextes deviennent plus petits et donc plus rapide à fouiller. La deuxième est que les nouveaux contextes créés par cette augmentation comportent des graphes de construction complètement nouveaux et différents et qui, parfois, contiennent des arbres fréquents plus complexes étendant ainsi les temps d'exécution.

Efficacité

Tout d'abord, en ce qui concerne la causalité, nous avons étudié le journal Sepsis et avons découverts 968 patterns dont 77 sont C-exclusifs. Parmi ces derniers, 35 affichent une relation causale. Plus précisément, ils sont tous causés par la variable "InfectionSuspected = 1". Un exemple de ces patterns est *seq(seq(ER Triage, Leucocytes), CRP)* et qui est donc causé par la suspicion d'une infection chez le patient. Par ailleurs, nous donnons dans Table 6.7 et Table 6.8 le nombre de règles comportementales découvertes dans chaque contexte. Comme exemple, pour WABO, l'arbre (1) dans Fig. 6.4 affiche une règle comportementale entre les deux fils de la racine (*seq(T02, T04)* et *seq(T06, T10)*). Notons que l'existence d'une règle dans un contexte n'implique pas son existence dans un autre. Enfin, dans la Fig. 6.4, nous illustrons les quatre cas donnés dans les directives d'interprétation discutées dans la méthodologie d'utilisation du DAF.

Évaluation de ACOBPAM

Efficience

Nous comparons d'abord les temps d'exécution de COBPAM et de ACOBPAM. Plus exactement, nous nous limitons au temps d'exécution de la découverte sans tenir compte du post-traitement et de la génération de la visualisation. Les résultats sont présentés dans Table 6.14 où deux profondeurs sont considérées: deux et trois. Nous pouvons observer une diminution de 60% à

66% dans le temps d'exécution prouvant ainsi l'efficience de notre algorithme. Ensuite, nous comparons dans Table 6.15 les temps d'exécution de notre dernière approche par rapport à la méthode originelle de découverte de LPM. Les résultats montrent que notre algorithme est des ordres de grandeur plus rapide. Ainsi le temps d'exécution pour Sepsis a été divisé par 100 et celui de WABO par 1000. Pour les autres journaux, la découverte de LPM n'a pas retourné de résultats. Enfin, nous reportons les temps d'exécution du post-traitement et de la visualisation pour les deux profondeurs considérées dans Table 6.16. Les valeurs augmentent avec le nombre d'arbres dans l'ensemble final car il est nécessaire de faire des comparaisons par paire pour vérifier les équivalences et pour détecter les nouveaux types de graines que nous avons introduits. De plus, ces temps sont exponentiels par rapport à la profondeur des arbres car plus celle-ci croît plus le nombre de graines et la taille du langage augmentent.

Efficacité

Les résultats dans Table 6.17 mettent en évidence une diminution du nombre d'arbres retournés de 35% à 73% de leur valeur initiale ce qui montre que le nombre de patterns redondants n'est pas négligeable et que l'opération de post-traitement est nécessaire et bienvenue. De plus, on peut également noter que le ratio d'arbres redondants augmente avec la profondeur. Cela est dû à la multiplication des graines et des équivalences par arbre. Par ailleurs, nous donnons dans Fig. 6.5 un ensemble de patterns découverts par COBPAM dans WABO. On peut voir un exemple de graines alternatives : Fig. 6.5a et Fig. 6.5c le sont par rapport à Fig. 6.5b et Fig. 6.5d respectivement. Ces graines ne sont pas conservés dans l'ensemble final. Enfin, nous joignons une capture d'écran de notre application de visualisation dans Fig. 6.6 et Fig. 6.7 ainsi que la légende des relations entre patterns dans Fig. 6.8.

Conclusions et Perspectives

La découverte de processus (Process Discovery), une branche du Process Mining, est un domaine dans lequel on s'efforce d'extraire des modèles et des informations sur les processus en cours d'exécution. Une caractéristique d'intérêt est le degré de structuration de ces processus. Lorsqu'il existe peu de comportements en commun entre les instances de processus, on fait face à des processus dits flexibles qui sont complexes et difficiles à analyser. Dans cette thèse, nous nous sommes intéressés à l'étude de ces processus en proposant un résumé simplifiant l'analyse sous forme de patterns comportementaux. Autrement dit, de petits modèles retraçant les régularités du processus. Nous avons commencé par concevoir un algorithme d'extraction efficient et efficace qui garantit certaines propriétés sur les patterns découverts, à savoir la maximalité et la compacité. Ensuite, nous avons construit un framework d'analyse qui introduit des dépendances entre patterns et données, entre les patterns eux-mêmes ainsi que des études de causalité; le tout agrémenté d'une méthodologie d'analyse. Enfin, nous avons fourni un algorithme d'extraction avancé qui améliore le premier avec une efficience poussée, une réduction maximale des arbres redondants et enfin une technique de visualisation. Toutes nos approches ont été évaluées sur des journaux d'évènements réels et testées pour leur efficience et efficacité quantitative et qualitative. Elles se sont avérées

supérieures à l'état de l'art. Par ailleurs, nos travaux laissent la porte ouverte à de nombreuses perspectives. Premièrement, l'utilisation de frameworks de calcul parallèle pour gérer les journaux les plus volumineux et complexes. Ensuite, il est possible d'impliquer la dimension données liées aux évènements ou même la dimension ressource pour découvrir des comportements entre agents exécuteur des activités. De manière orthogonale, nous prévoyons de filtrer et de classer les patterns comportementaux en utilisant des fonctions d'utilité et des contraintes ou conditions. Enfin, les patterns peuvent servir de variables dépendantes dans diverses tâches d'apprentissage automatique et de mining. Dans le même esprit, les règles comportementales peuvent s'insérer dans les systèmes de recommandation soit à un état statique ou de flux.

Abstract

In the last two decades, we assist to an explosion of readily available data; from sensors measurements to web queries going through commercial transactions. One type of these data is event logs recorded by information systems that capture process executions. Process Mining is a new research discipline that caters for the analysis of such event logs. One of its tasks is the discovery of process models which describe the process execution. Such models could appear structured with clear execution paths or be more "spaghetti" like in the sense of a great complexity and a big number of branchings and choice constructs.

Many algorithms were proposed to handle the extraction of end-to-end process models. Few were successful in handling unstructured processes. For these special cases, techniques that focus on more granular information about the process execution are preferred. Some leverage the abstraction of events and their relationship while others aim at subdividing the log in more manageable clusters.

Our work supplements these techniques with a new approach to extract insight from unstructured processes. The contributions revolve around the notion of behavioral patterns; relatively small process models that capture important fractions of the execution. The measure of importance is the frequency of the behaviors. We propose in a first work a robust and rapid algorithm to extract them leveraging pruning techniques, various optimizations and guaranteeing interesting properties on the extracted models. The analyst can then use them to grasp a global view of the unstructured process through the many frequent behaviors it exhibits.

In a second step, we propose a framework that analyzes dependencies, associations and causal relationships while taking into account contextual data. We propose a formal definition of contexts based on attributes introduced in the event log. We discover patterns in each context and provide a characterization of interesting patterns that overlap many contexts. This approach offers additional information on non context-aware behavioral patterns and actually reveals completely new patterns nested in the different contexts. Moreover, we infer behavioral rules from mined patterns and study causal relationships between data attributes and the occurrence of patterns.

Finally, we improve the initial context-agnostic algorithm in order to further decrease runtimes and to reduce the difficulty of analyzing a big number of patterns. We propose an incremental algorithm that takes advantage of previously assessed patterns to evaluate the frequency of the new ones; reducing thus greatly the execution times. Moreover, we put in place a post-processing step that reduces the number of discovered patterns by eliminating redundancy. The final set of patterns is then displayed in an interactive and intuitive map or graph that harbors interesting dependencies between the patterns and allows for an easy navigation.

The first algorithm, the framework and the revisited algorithm were assessed against real life logs and proved efficient and effective compared to the state of the art. In future works, we intend to explore behavioral patterns in resource utilization instead of the control flow dimension. We also plan to exploit parallel computing framework in order to handle the most challenging and biggest logs. Finally, we project to use behavioral patterns as features for various machine learning tasks.

Contents

List of Figures

List of Tables

1 Introduction

Contents

1.1 Context and Motivation

In one minute on the internet in 2020, Youtube users uploaded 500 hours of video, over 41 million of messages were sent on Whatsapp, Twitter gained 319 new users and the social media users were sending half a million tweets. In the same year, every person was generating 1.7 megabytes in one second [1, 2]. Moreover, in 2018, IDC (International Data Corporation) predicted that the world's digital sphere would grow from 33 zettabytes to 175 zettabytes by 2025. If we were able to stack this amount in DVDs, we would have enough to get to the moon 23 times [64].

This phenomenon, coined "Big Data", is defined as the exponential growth of data, their processing or in a more general way, any step involved in the extraction of interesting insight from the vast amount of raw data [81]. In 2011, the McKinsey Global Report [56] defined it as data whose scale, diversity and temporal distribution require new technical architectures and more in-depth analyzes in order to extract knowledge that represents a new source of entrepreneurial value. Although the definitions differ, they revolve around certain characteristics that the data share. It is originally the "3V" of Big Data: Volume, Velocity and Variety [68] then extended to cover "Veracity" and "Value" thus becoming the "5V" of Big Data [81].

Every day, more data is produced than all that is available in print media around the world [81]. The Volume is the most important feature of Big Data. Today, all fields produce hundreds of

terabytes. The phenomenon evolves all the more easily because of the low cost of data storage. Furthermore, the more the data, the more precise are analysis results which makes it a precious material. Yet, most systems have to adapt to the enormous size by ensuring scalability and parallel computing. The Velocity property is due to the high speed at which the data is generated, collected and ingested or analyzed [68]. The phenomenon also called Data Flow represents a second challenge for Big Data systems. Variety is another characteristic of Big Data. It refers to the heterogeneity of the data manipulated; either it is structured (relational databases tables, CSV format, etc), unstructured (images, audio, texts, etc) or semi structured (XML). As for Veracity, it refers to the uncertainty around the data collected as well as, consequently, the results of the analysis. The correctness, precision and quality of data becomes doubtful due to sometimes unreliable sources and too much variety. Finally, The Value of data represents what it can bring in gain, both to the scientific community and to the industry and business sectors through the effective study of the markets and the application of the fourth paradigm of science: Data Science [81].

Data Science is a new discipline that spans many fields. Most important are statistics, data mining, machine learning, visualization and visual analytics. As defined in [82], Data Science is an interdisciplinary field aiming to turn data into real value. Data may be structured or unstructured, big or small, static or streaming. Value may be provided in the form of predictions, automated decisions, models learned from data, or any type of data visualization delivering insights. Data science includes data extraction, data preparation, data exploration, data transformation, storage and retrieval, computing infrastructures, various types of mining and learning, presentation of explanations and predictions, and the exploitation of results taking into account ethical, social, legal, and business aspects.

On another hand, in companies, information systems are closely following business processes. A business process is a set of linked activities which, when executed, deliver a certain organizational goal, either a product or a service. A business activity or task is any action executed by a resource which participates in the unfolding of a process. A process instance or case represents a single execution of the process. These processes are intensively studied in Process Science. It is an interdisciplinary field utilizing information technology and management science to run, monitor, control, analyze, model and improve organizational processes [82]. Some branches are business process management, stochastics and operations management and research.

Since one of the roles of information systems is to collect, store and process information, the business activities executions are stored as events in event logs. Each process instance is represented by a series of events recorded and identified in an event log in the form of a trace. The main information provided by an event are the activity executed, the timestamp and sometimes the resource which enacted the event. There may be many timestamps recorded for each activity such as start time and completion time. Besides, other relevant attributes either associated to a trace or to the events indicating the execution context may appear in the event log.

Process Mining, one branch of Process Science, is the research discipline which studies event logs. It uses machine learning and data mining techniques to extract knowledge from execution data and in doing so links the two domains of Process Science and Data Science. While Process Science is model-driven and Data Science is process agnostic, Process Mining considers both

the process models and the evidence in the form of data behind their execution. There are four important research areas among the process mining community. Process Discovery which mines process models describing the execution of processes. Conformance Checking which checks for discrepancies and deviations between an existing model and an event log. Process Enhancement which updates a process model according to an event log either by modifying it to adjust to the event log or extending it and finally, Predictive Process Monitoring which provides the user with constant information about the future of a process execution [32].

The most explored branch of Process Mining is Process Discovery. It aims to reveal process models holding insight about the execution of the process. Some of the algorithms proposed extract end-to-end process models like the α-algorithm [83], Flexible Heuristics Miner [88] and Genetic Miner [16]. The goal is to describe the whole execution of the process from start to end. This method works well on structured processes like the one shown in *Figure 1.1*. One can note that the execution paths are few and clearly defined. However, they do not cater for what is called flexible processes.

A flexible process contains too many different execution paths and is highly unstructured. The execution behavior cannot be represented by a one-fit-all global model. In fact, the use of algorithms such as Flexible Heuristics Miner yields "spaghetti" models such as the one depicted in *Figure 1.2*. Other algorithms may also be prone to over-generalization [82]. Namely, they generate models that allow for more instances than those present in the log and permissible by the process.

With the aim of handling flexible processes, specific methods specializing in getting other forms of insight than a global model have emerged. Some propose to group homogeneous traces into clusters and extract structured models from each cluster [13, 15, 38, 73]. Others try to propose models with different level of abstraction to mitigate the spaghetti effect [39].

An interesting work is the mining of behavioral patterns [80, 78]. These patterns are formalized as process models, yet they capture only comparatively small episodes of a process' behavior that occur frequently. The basic idea is illustrated in Fig. 1.3. For the example log, depicting treatment steps a patient undergoes in a hospital, a traditional discovery algorithm such as the Flexible Heuristics Miner (FHM) [88] would yield a complex model. However, one may observe that the traces show a specific behavioral pattern: An execution of activity *BT* is followed by *CO* and *RB* in parallel. Detecting such a pattern provides a general understanding of the regularities in process execution. Note though, that such a pattern cannot be detected using standard techniques for sequential pattern mining, such as PrefixSPAN [61], as those would miss complex behavioral dependencies such as concurrency and exclusive choices.

However, these seminal works on behavioral pattern mining endure several drawbacks that impact the runtimes and the quality of derived patterns. In this thesis, we improve over the mining of such behavioral patterns while making sure to exploit all the data available. The next section describes the research goals we aim at.

1.2 Research Goals

Existing algorithms [80, 78] to mine behavioral patterns suffer from imprecision and redundancy of the mined patterns, and a comparatively high computational effort. That is, even though certain behavior is frequent, patterns may capture (i) only a part of the frequent behavior (i.e., they are not maximal), or (ii) a combination of frequent behavior with infrequent behavior (i.e., patterns are not compact). For instance, in Fig. 1.3, the pattern *seq(BT, and(CO,RB))* is frequent. Arguably, discovery of further patterns *seq(BT,CO))* and *seq(BT, xor(CO,I))* would not lead to any new insights on the process, so that it is sufficient to discover the former one. At the same time, existing algorithms suffer from high run-times since pattern candidates are evaluated based on the complete event log. The first research goal we address is:

- **G1:** *Optimize run-times of behavioral pattern mining while avoiding uninteresting trees.*

Once we discover behavioral patterns, a second objective is to get all available insights from them.

- **G2:** *Take inspiration from seminal fields such as Frequent Pattern Mining and Sequential Pattern Mining to uncover additional analysis-driven usage of behavioral patterns.*

On an orthogonal subject, while existing algorithms for behavioral patterns mining [80, 78] discover patterns that are frequent, they are constrained to one dimension of study, namely, the control flow perspective i.e. the inter-dependencies between activities. However, the data dimension, i.e., the context of execution of activities and instances, is of utmost interest and Process Mining involves considering all evidence available behind process execution. Hence, the next research goal:

- **G3:** *Include the data dimension in the mining procedure.*

1.3 Contributions

To meet the goal formulated in **G1**, we propose COBPAM (COmbination Based PAttern Mining algorithm) [5], a novel combination-based algorithm to mine behavioral patterns that are formalized as process trees [46] (a process modelling notation) through a generate and test approach. It identifies all trees of which the behavior can be found in a certain number of traces of the event log, which takes up the well-established notion of support for patterns in sequence databases [36]. Moreover, the metric precision as known from Conformance Checking [82] is used to assess how strongly a tree materializes (if the tree appears under the many forms allowed by the model, then it is strongly present in the log). Based thereon, the contributions of COBPAM are threefold:

1. It defines a partial order on pattern candidates to discover only those that are maximal and compact, thereby improving effectiveness of pattern mining. The partial order further allows to avoid multiple generation of the same patterns.
2. It efficiently explores the pattern search space by pruning strategies, exploiting that complex patterns are combinations of simpler patterns.

3. It further improves efficiency by considering only a subset of traces, when evaluating the support and precision of a pattern candidate.

On another hand, as in Sequential Pattern Mining, patterns returned by COBPAM could serve in another data analysis procedure. Namely, the study of associations between patterns themselves. This comes as a natural evolution for COBPAM with the introduction of behavioral rules as equivalents of sequential rules. In the example above, the behavioral rule *BT* → *and(CO,RB)* would indicate that *BT* cannot appear without *and(CO,RB)* following it and *and(CO,RB)* cannot appear without *BT* preceding it. Behavioral rule mining fulfills **G2**.

In another contribution, we tackle **G3** and extend COBPAM with a data perspective. Let's consider the data enhanced event log in Fig. 1.4. There are two attributes representing contextual information: The income level of the patient (low or high) and the age group (<70 or 70+). In such a log, a context-agnostic approach to pattern discovery constitutes a severe limitation in terms of the insights that may be gained about the process. Episodes of process behavior that are common for a specific execution context, but not frequent over all contexts remain undetected. For example, considering solely low income patients, there is a pattern of *BT*, followed by *SW*, which is again followed by the concurrent appearance of *CO* and *RB*. Detecting this contextual pattern provides insights (i.e., low income patients show a tendency to discuss the refund policy with a social worker) that would go unnoticed when neglecting context information. As such, contextual patterns enable fine-granular analysis of the correlations of contextual factors and process execution. Moreover, one could be interested in answering questions like: "What behavioral patterns are frequent among low income patients exclusively?" or "What behavioral patterns are frequent among high income patients whatever their age?". This can be accomplished by defining different types of frequentiality. By answering these questions, data is intrinsically considered in the mining process and dependencies between data and patterns are discovered in the form of contextual patterns.

In the case where a pattern appears exclusively in a population, it is wise to ask whether attributes of that population cause the appearance of the behavioral pattern. Correlation and causation are different. Correlation means that two variables are related or dependent on each other whereas causation means a change in the cause variable yields a change in the outcome one. In the spirit of attending to **G3**, we also propose a study of the causation between contextual event log attributes and the occurrence of a pattern inspired by retrospective cohort studies.

To sum up, we extend COBPAM with a data-aware analysis framework [6] that builds dependencies, causations and associations around patterns while considering the data dimension. The contributions are:

1. We introduce a model for contextual behavioral patterns. It includes different notions to link a behavioral pattern with contextual information of traces, such as contextual frequency, generality, and exclusiveness.
2. We present an algorithm CCOBPAM (Contextual COBPAM) to discover contextual behavioral patterns from an event log.
3. We discover behavioral rules akin to sequential rules in Sequential Pattern Mining.

4. We assess the existence of causal relationships between data and behavioral patterns occurrences.
5. We present a complete methodology on how to use effectively our framework and transform the obtained results into useful insights.

Finally, after proposing COBPAM which has already significantly reduced runtimes, we set out to improve the performances even more; always in the spirit of reaching **G1**. As such, we introduced ACOBPAM (Adanced COBPAM) that makes use of the alignments of older trees to assess the frquencies of the more complex ones. Note that alignments are a tool that allows to assert whether a pattern appears in a trace. For example, they allow to claim that the pattern in Fig. 1.3c appears in *trace 1* of the log in Fig. 1.3a. Moreover, the new algorithm includes a post-processing step that prunes the number of returned trees by COBPAM by eliminating redundancy further than what was detected in the classical algorithm. For that, new concepts like equivalency of process trees or generalized maximality are defined.

On another hand, in a last attempt to further satisfy **G2**, we proposed an in-depth analysis of the returned patterns after post-processing. In fact, we devised new relationships or dependencies between the final patterns. The whole was integrated in the form of a map or a graph where patterns are nodes and edges, relationships. Interactive and exhaustive, it helps getting the maximum of insights from the discovered patterns by offering a global navigable view. We recapitulate the major features of ACOBPAM:

1. ACOBPAM uses an incremental algorithm to compute alignments taking advantage of previously computed ones.
2. A pruning of the final returned trees in a post-processing step is realized. It caters for resolving the issues of high numbers of patterns initially returned by eliminating redundancy.
3. A visualization graph is proposed. It offers an interactive and navigable view on the patterns while harboring interesting dependencies. It comes as an attempt to facilitate analysis especially when there still is a high number of trees after the post-processing step.

Our contributions [5, 6] were published respectively in CAISE (International Conference on Advanced Information Systems Engineering) and IEEE TKDE (IEEE Transactions on Knowledge and Data Engineering) with the collaboration of Pr. Matthias Weidlich. The third work concerning ACOBPAM is under submission to TKDE.

1.4 Layout of the Thesis

This thesis is structured as follows. In Chapter 2, we present preliminary notions necessary to understand the following chapters and go over the state of the art in Process Discovery, Sequential Pattern Mining and Association Rule Mining, data-aware Process Discovery, Causality and finally Conformance Checking. Chapter 3 discusses behavioral pattern mining with COBPAM. It shows how the combination operation allows for adequate optimizations and the satisfaction of the target

properties while presenting the core of the algorithm. Next, Chapter 4 details the data-aware analysis framework. It starts by defining the behavioral rules. Then, it explicits the model we adopt for the definition of contexts and their interdependency with patterns. Afterwards, it presents an algorithm for the discovery of contextual patterns and finishes with a data-driven analysis of discovered patterns corroborated with a handy methodology enhanced with interpretation guidelines. The third contribution is presented in Chapter 5. The first section details the incremental algorithm which performs what we call the alignment growth. Afterwards, we proceed to specify the type of trees that we prune in post-processing and finish by presenting the visualization map along with its dependencies. Finally an experimental study that confirms the general feasibility of our algorithms and the effectiveness and efficiency of the different contributions is presented in Chapter 6 before we conclude and give perspectives for our work in Chapter 7.

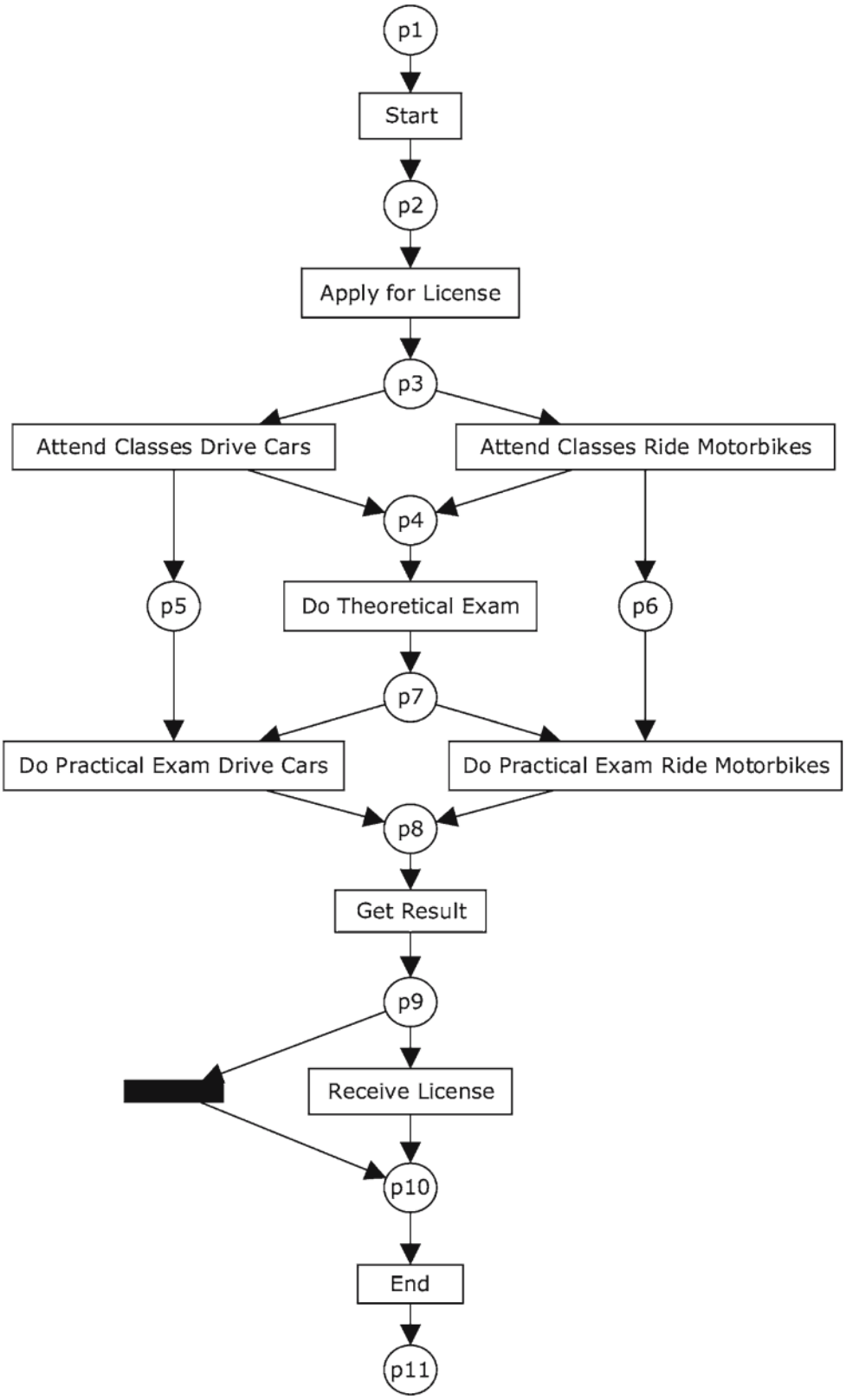


Figure 1.1: Structured process for driving licence exam

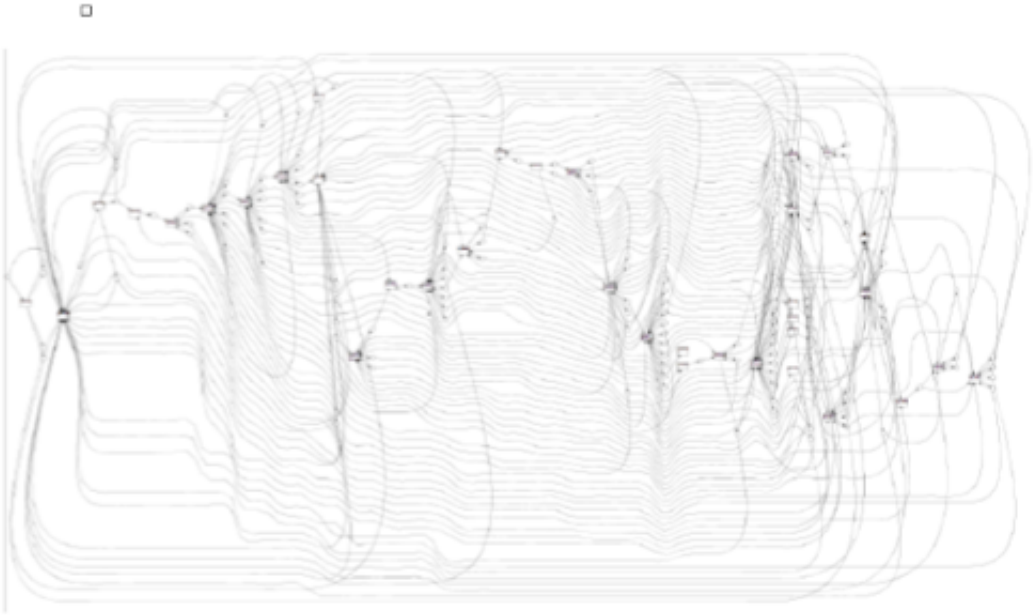


Figure 1.2: Spaghetti model mined with Flexible Heuristics Miner

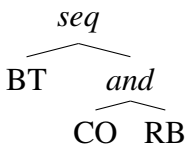
Trace ID	Event Sequence
1	EI ET PS ED BT GP TD SW CO RB
2	ET EI CV XS BT SW CS D RB CO
3	CI PS CV I BT XS SW E CO RB I
4	CI CV PS XS BT D SW CS GP RB CO
5	CI PS EI ED I XS GP TD CV
6	EI ET PS ED BT GP TD CO RB
7	ET EI CV XS BT CS D CO RB
8	CI PS CV I BT XS E CO RB I
9	CI CV PS XS BT D CS GP CO RB
10	CI PS EI ED I XS GP TD CV
11	ET PS ED BT GP TD SW CO RB
12	CI PS EI ED I XS GP TD CV

SW: Meet with social worker, CO: Checkout, BT: Blood test, CI: Check-In, GP: Give prescription, CS: Recheck sec. number, EI: Emergency intubation, ET: Emergency transfusion, PS: Process sec. number, CV: Check Vitals, XS: X-ray Scan, TD: Temp. Diagnosis, D: Diagnosis, ED: Emergency defibril., I: Infusion, E: Echography, RB: Retrieve belongings

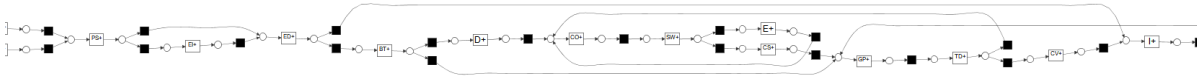
(a)

Sequential Patterns
BT RB
BT CO

(b)



(c)



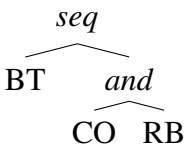
(d)

Figure 1.3: (a) Event log; (b) sequential patterns discovered with PrefixSPAN [61]; (c) behavioral pattern; (d) end-to-end model mined by FHM [88]. Patterns have a support > 0.7 (i.e., occur in more than eight out of 12 traces).

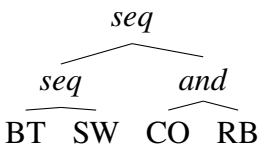
Trace ID	Context [Income, age]	Event Sequence
1	[low, <70]	EI ET PS ED BT GP TD SW CO RB
2	[low, <70]	ET EI CV XS BT SW CS D RB CO
3	[low, <70]	CI PS CV I BT XS SW E CO RB I
4	[low, <70]	CI CV PS XS BT D SW CS GP RB CO
5	[low, <70]	CI PS EI ED I XS GP TD CV
6	[high, <70]	EI ET PS ED BT GP TD CO RB
7	[high, <70]	ET EI CV XS BT CS D CO RB
8	[high, <70]	CI PS CV I BT XS E CO RB I
9	[high, <70]	CI CV PS XS BT D CS GP CO RB
10	[high, <70]	CI PS EI ED I XS GP TD CV
11	[low, 70+]	ET PS ED BT GP TD SW CO RB
12	[high, 70+]	CI PS EI ED I XS GP TD CV

SW: Meet with social worker, CO: Checkout, BT: Blood test, CI: Check-In, GP: Give prescription, CS: Recheck sec. number, EI: Emergency intubation, ET: Emergency transfusion, PS: Process sec. number, CV: Check Vitals, XS: X-ray Scan, TD: Temp. Diagnosis, D: Diagnosis, ED: Emergency defibril., I: Infusion, E: Echography, RB: Retrieve belongings

(a)



(b)



(c)

Figure 1.4: (a) Data enhanced event log; (b) behavioral pattern for the whole log; (c) behavioral pattern for the context [low, *]. Patterns have a support > 0.7.

2 Preliminaries and Related Work

Contents

Even though the Process Mining field is relatively young, it has gained a big interest within the research community. Many methods spanning different issues were proposed. This chapter is dedicated to basic notions about Process Mining needed to understand the breakthroughs mentioned later in the works related to our thesis. It is divided in Section 2.1 which presents the preliminaries and Section 2.2 which discusses the state of the art relevant to our work.

2.1 Preliminaries

This section covers some definitions necessary to understand the next material. We start by the origin of it all, the evidence behind process executions: the event log [7, 82]. We move then to talk about process models and notations in general.

Table 2.1: Example of event log

Case ID	Event ID	Properties				
		Timestamp	Activity	Resource	Transaction type	...
1	10005	08-04-2020 17:32	Apply for Licence	Mehdi	Complete	...
1	10006	11-04-2020 07:15	Attend Classes Ride Motorbikes	Mehdi	Complete	...
1	10007	15-04-2020 08:00	Do Theoretical Exam	Mehdi	Complete	...
1	10008	27-04-2020 08:15	Do Practical Exam Ride Motorbikes	Mehdi	Complete	...
1	10009	01-05-2020 10:23	Get Result	Mehdi	Complete	...
2	9008	13-04-2020 11:45	Apply for Licence	Ali	Complete	...
2	9009	16-04-2020 07;30	Attend Classes Drive Cars	Ali	Complete	...
2	9010	17-04-2020 08:00	Do Theoretical Exam	Ali	Complete	...
2	9011	18-04-2020 08:15	Do Practical Exam Drive Cars	Ali	Complete	...
2	9012	01-05-2020 09:27	Get Result	Ali	Complete	...
2	9013	01-05-2020 10:30	Receive License	Ali	Complete	...
...	...	...	...	...	...	...

2.1.1 Event Log

An event log records information about the execution of an organizational process. We suppose that each log records information about one distinct process. An example of an event log is given in Table 2.1. The events related to a single case of the process are grouped together in what is called a trace. Inside traces, events are ordered. To each event, a set of attributes is associated. The bare minimum in process mining are the activity represented by the event and its case identifier. In the example event log of Table 2.1, activities can be : *Apply for License*, *Get Result*, etc. However, other attributes can exist in the event log. They include:

- **Timestamp:** the time at which the event was enacted.
- **Transaction type:** the execution of activities spans a period of time. The transaction type indicates the current step in the life cycle of the activity. It can be for example: "start" or "complete".
- **Resource:** the organizational entity executing the activity. It can be a physical entity, e.g., a person or an abstract one, e.g., a program.

It is to be noted that any other property on the event can constitute an attribute. Besides, Not all events need to have the same attributes, although the events associated to the same activity tend to have the same.

In the following, we formalize the various notions around event logs [7, 82].

Definition 2.1 (Complex event, attribute)**.** *Let E be the universe of events (the set of all possible event identifiers) and AN the set of the attributes names. The value of the attribute $n \in AN$ for the event $e \in E$ is given by $\#_n(e)$. If the event isn't associated to an attribute n, we use the null value $\#_n(e) = \perp$. In particular, the standard attributes mentioned above are given by :*

- $\#_{activity}(e)$ *is the activity associated to the event e.*
- $\#_{time}(e)$ *is the time of execution or timestamp.*

- $\#_{resource}(e)$ *is the resource that executed the event.*
- $\#_{trans}(e)$ *is the transaction type of e, e.g., "complete".*

The activity and timestamp attributes yield a control-flow perspective on the event log; meaning, the different structures of execution of the underlying business process. The resource attribute yields an organizational perspective while other contextual attributes represent a data perspective.

An event log represents a set of traces where each trace represents a unique case. Generally, the execution of a case doesn't directly influence other cases. For example, a loan application is handled for each person independently of the other applications. As such, we consider the events in each case in isolation during analysis.

Remark 2.2 (Sequence)**.** In order to represent traces, we use a convenient way in the form of sequences [7, 82]. This notation will also be useful in modeling behaviors when defining semantics in Petri nets. A^* is the set of finite sequences over a set A. A finite sequence over A of length n is a mapping $\sigma \in \{1, 2, \ldots, n\} \rightarrow A$ and we write $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$ where $a_i = \sigma(i)$. $|\sigma| = n$ denotes the length of σ. $\sigma \oplus a$ denotes the sequence where a is appended to the end of σ. By extension, σ_2 appended to σ_1 is written $\sigma_1.\sigma_2$.

Considering a sequence $\sigma = \langle a_1, a_2, \ldots, a_n \rangle$, a subsequence of σ is a sequence $\gamma = \langle a_{\omega(1)}, a_{\omega(2)}, \ldots, a_{\omega(m)} \rangle$ where $m \leq n$ and $\omega : \mathbb{N} \rightarrow \mathbb{N}$ an increasing function. We say that σ contains γ and we write $\gamma \leq \sigma$.

$hd^k(\sigma) = \langle a_1, a_2, \ldots, a_{min(k,n)} \rangle$ denotes the prefix or the head of size k of σ. The set of prefixes of σ is given by $pref(\sigma) = \{hd^k(\sigma) \mid 0 \leq k \leq n\}$.

$tl^k(\sigma) = \langle a_{min(k,n)}, a_{min(k,n)+1}, \ldots, a_n \rangle$ is the suffix or tail of σ starting from $\sigma(k)$, inclusive.

$In_k^l(\sigma)$ is the subsequence of σ starting from $\sigma(k)$ to $\sigma(l)$ (inclusive).

$\sigma \uparrow X$ is the projection of σ over a set $X \subset A$. For example, $\langle a, a, b, a, b, c, a \rangle \uparrow \{b, c\} = \langle b, b, c \rangle$.

$\delta_{set}(\sigma)$ represents the set of the elements present in σ, e.g., $\delta_{set}(\langle a, a, b, a, b, c, a \rangle) = \{a, b, c\}$. $\delta_{multiset}(\sigma)$ or the Parikh vector represents the multiset containing the elements of σ, e.g., $\delta_{multiset}(\langle a, a, b, a, b, c, a \rangle) = [a^4, b^2, c]$.

Remark 2.3 (Multiset)**.** A multiset [7, 82] or bag is similar to a set but can contain multiple occurrences of one element. $[a^2, b, c^3]$ is a multiset that contains two elements of a, one of b and three of c. Formally, the set of bags over a set P is given as $\mathbb{B}(P) = P \rightarrow \mathbb{N}$. So if $M \in \mathbb{B}(\mathbb{P})$ then $M(p)$ gives the number of elements p in the bag.

Operations on multisets are defined in a straightforward manner : the addition, e.g., $[a^2, b, c^3] \uplus [a, b] = [a^3, b^2, c^3]$; the difference, e.g., $[a^2, b, c^3] \setminus [a, b] = [a, c^3]$; the presence of an element in the multiset, e.g., $a \in [a, b]$ and the notion of subset, e.g., $[a, b] \leq [a^2, b]$. These operations are directly applicable on sets by considering a set as a multiset where each element occurs only once.

Definition 2.4 (Case, complex event log)**.** *Let C be the universe of cases (the set of all possible case identifiers). An event log is composed of cases. Each case has attributes as for events. For a*

particular case $c \in C$ and $n \in ANC$, an attribute name, $\#_n(c)$ denotes the value of the attribute n for the case c. If the case doesn't have a value for an attribute n, we use the null value: $\#_n(c) = \perp$. A particular mandatory attribute for a case is its associated trace $\#_{trace}(c) \in E^*$ and we refer to it by $\hat{c} = \#_{trace}(c)$.

A trace is a finite sequence of events $\hat{c} \in E^*$ such that :

- *There is no two identical events in the trace. For $1 \leq i < j \leq |\hat{c}|,\ \hat{c}(i) \neq \hat{c}(j)$.*
- *Events in the trace $\hat{c}$ are ordered according to their timestamp. For $1 \leq i < j \leq |\hat{c}|,\ \#_{time}(\hat{c}(i)) < \#_{time}(\hat{c}(j))$*

A complex event log $L \subset C$ is a set of cases. There are no two identical events in the event log, meaning, for any $c_1, c_2 \in L, \delta_{set}(\hat{c}_1) \cap \delta_{set}(\hat{c}_2) = \emptyset$.

An example of a trace from the event log of Table 2.1 could be: ⟨ *10005*, *10006*, *10007*, *100058*, *10009* ⟩

On another hand, Process Mining is about the analysis of event logs. Discovery algorithms start from such logs and yield models such as the one depicted in Fig. 1.1. In order to do that, events must be transformed into activities in the modelling language. Since events in our formalism are identifiers associated with attributes, they need to be converted into names used by a process model. For that, we propose the notion of classifiers.

Definition 2.5 (Classifier)**.** *A classifier is a function that maps each event $e \in E$ to a representative name denoted $\underline{e}$ which will be used in analysis.*

For example, the representative name could be the activity attribute: $\underline{e} = (\#_{activity}(e))$ or a combination of two attributes $\underline{e} = (\#_{activity}(e), \#_{trans}(e))$. In the latter case, we could see in the model depicted in Fig. 1.1 new activities like *Do theoritical Exam_Start*. We consider the former classifier the default one and we will use it unless specified otherwise.

In the following, we propose a simpler definition of event logs. It is the one we use in our work.

Definition 2.6 (Simple event log)**.** *Let A be a set of* activity identifiers *(*activities*), and A^* the set of all sequences over A. An* event log *L is a set of cases c where the trace attribute is given by $\hat{c} \in A^*$, a sequence of activities. $|L|$ denotes the size of L i.e., the number of cases it contains.*

It is straightforward to convert a complex event log to a simpler one using classifiers.

Definition 2.7 (Transform a complex event log into simple event log)**.** *Let $L \subset C$ be a complex event log over a complex event universe E. We convert each trace $\hat{c} = \langle e_1, e_2, \ldots, e_{|\hat{c}|} \rangle$ from L using a classifier to $\underline{\hat{c}} = \langle \underline{e_1}, \underline{e_2}, \ldots, \underline{e_{|\hat{c}|}} \rangle$. The simple event log is then given as $SL = \{c \mid c \in L \wedge \#_{trace}(c) = \underline{\hat{c}}\}$.*

For example, for the case one ($c = 1$) in Table 2.1, its associated trace would be: $\hat{c}$ = [⟨ *Apply for License*, *Attend Classes Ride Motorbikes*, *Do Theoretical Exam*, *Do Practical Exam Ride Motorbikes*, *Get Result* ⟩, ⟨ *Apply for License* . . . ⟩, . . .].

2.1.2 Process Models

First of all, we want to stress some important notions we stumbled upon in our work. That is, the types of process models. From our search in the literature, we distinguish three types:

- **Procedural process models:** Those are models that define exactly and explicitly the behavior to execute at any moment during the process evolution. They give a *procedure* to follow or orders. That's why they are also called imperative. The next steps in the execution of the process are defined through control flow structures: sequence, loops, concurrence, etc [82].

- **Declarative process models:** A model is declarative if it specifies a set of rules or constraints that must be satisfied during the execution. Any behavior that is not forbidden by these rules is allowed. As such, this kind of process models don't specify *a procedure* to follow. At any moment during the execution, we have no explicit order to execute, just conditions not to transgress [82].

- **Descriptive process models:** Another kind of proces models is descriptive [43]. Process models that fall under this definition cannot be called declarative nor procedural. For instance, behavioral patterns do not prescribe a procedure to follow nor do they prohibit certain behavior. They *just describe* a process model in the sense where they give information about its execution. In our case, these information are frequent patterns. Even though the discovery of all these types of models share similarities like the use of frequency to assert significance, the underlying reasoning is not the same. For declarative and procedural model discovery, if a behavior is frequent, it still means that it is mandatory. Indeed the absence of the behavior is considered noise. Conversely, in behavioral pattern discovery, the absence of behavior is not considered noise but just what it is. We can only say that the behavior is frequent or not from what we are observing, we do not infer on the requirements imposed by the underlying process.

Second, it is important to know that many of the branches related to Process Science rely heavily on process modelling. A formal process representation allows to pursue in-depth analysis on the execution of processes. In Operations Management, such process models are used to realize simulations, predict bottlenecks and evaluate effectiveness of a process. They are mainly used to mitigate the complexity of business processes that grew out of the evolution of information technology allowing cross-organizational processes and complicated event chains. Moreover, a process notation represents a common ground on which stakeholders can converge and can be used either for first time design or in a renewal procedure.

Understanding how important a process model is, it is not surprising to observe the plethora of notations that exist. In the following, we only present notations that we use and mention in this thesis.

Petri Nets

Petri nets are a well investigated model among the oldest allowing concurrency. It is executable and offers a large set of analysis tools [65]. Petri nets use simple and intuitive notations as shown in Fig. 2.1. They are a bipartite graph composed of places and transitions. Transitions can be labelled with activities. A Marked Petri net adds dynamism through tokens assigned to the places. The state of the Petri net is given by the distribution of such tokens. In Fig. 2.1, there is only one token in the start place. Formally, a Petri net is defined as [82, 7] :

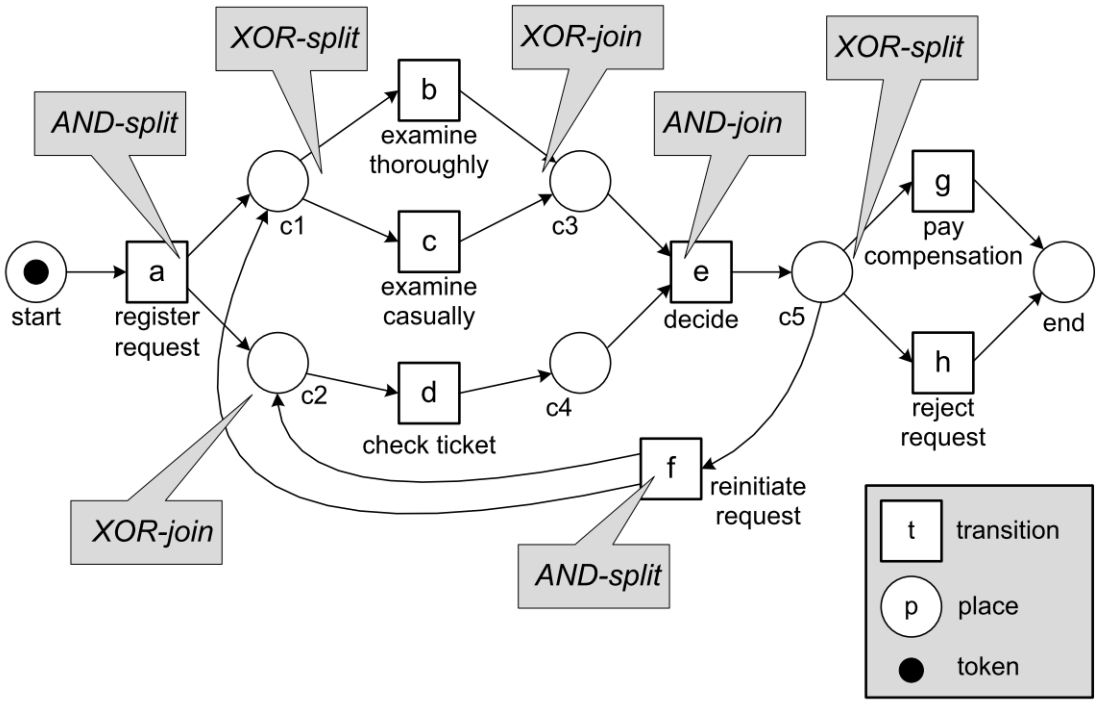


Figure 2.1: Marked Petri net example with its different structures [82]

Definition 2.8 (Petri net, Marked Petri net, Accepting Petri net)**.** *Let $A \subset \mathcal{A}$ ($\mathcal{A}$ is the universe of activities) and A^τ the set of activities to which τ is added. τ represents an empty label or a silent transition. A Petri net N over A is a tuple $(P, T, F, \alpha, m_i, m_f)$ where P is a finite set of places and T a finite set of transitions. $F \subset (P \times T) \cup (T \times P)$ is called the flow relation. $\alpha : T \to A^\tau$ is a mapping function from transitions to labels. Finally, m_i and m_f represent the initial and final marking respectively.*

A marked Petri net is a pair (N, M) where N is a Petri net and $M \in \mathbb{B}(P)$ the marking defined as a multiset of places.

An accepting Petri net is a Petri net $(P, T, F, \alpha, m_i, M_f)$ with a set of possible final markings $M_f \subset \mathcal{B}(\mathcal{P})$ instead of just one. It is to be noted that $\forall m_1, m_2 \in M_f, m_1 \not\leq m_2$

To be dynamic, Petri nets follow a *firing rule*. A transition is enabled at a marking M denoted $(N, M)[t\rangle$ if there is at least a token for each of its input places. Upon firing, the transition consumes

one token from each place in the input and produces one token in each output place.

In order to formalize this, we call elements of $P \cup T$ nodes. A node x is an input to node y if $(x, y) \in F$. $\bullet y$ is then defined as $\bullet y = \{x \mid (x, y) \in F\}$. Similarly, a node x is an output to node y if $(y, x) \in F$ and $y\bullet = \{x \mid (y, x) \in F\}$. In the example of Fig. 2.1, $\bullet e = \{c3, c4\}$ and $e\bullet = \{c5\}$.

We have that $(N, M)[t\rangle$ iff $\bullet t \leq M$. $M \xrightarrow{t}_N M'$ denotes the firing of transition t from the old marking M to the new marking M' and we have $M' = M \uplus \bullet t \setminus t\bullet$. N can be ommitted if the context is clear.

A sequence $\sigma \in T^*$ such that $\sigma = \langle \sigma_1, \sigma_2, ..., \sigma_n \rangle$ is called a firing sequence from M to M' iff there exists $M_1, M_2, \ldots, M_{n-1}$ such that $M \xrightarrow{\sigma_1}_N M_1 \xrightarrow{\sigma_2}_N M_2 \ldots \xrightarrow{\sigma_n}_N M'$ abbreviated $M \xrightarrow{\sigma}_N M'$. We also overload the notation $(N, M)[\sigma\rangle$ to signify that σ is indeed a firing sequence and enabled at marking M. σ is a complete firing sequence if $m_i \xrightarrow{\sigma}_N m_f$.

Example 2.9. Let N be the Petri net shown in Fig. 2.1. At the initial marking $m_i = [start]$, the empty firing sequence $\epsilon = \langle\rangle$ is enabled and we have $m_i \xrightarrow{\epsilon} m_i$. Also, upon firing a then b from the initial marking, we move to a new marking $M = [c3, c2]$ and we write $m_i \xrightarrow{\langle a,b \rangle} M$.

A marking M' is *reachable* from a marking M if and only if there exists a firing sequence σ from M to M'. The set of reachable markings from M is denoted $(N, M\rangle$. In the Petri net of Fig. 2.1, there are 7 reachable markings from the initial marking [82, 7].

Workflow Nets

Worfklow nets (WF-nets) [83] are a subtype of Petri nets satisfying certain properties that make them suitable to model business processes. Indeed, this type of processes has a clear begining and ending. The process is executed many times through instanciation and for each instance a case with a start and an end is generated. Take the process of handling some type of applications. Many candidates hand their files and the process is executed for each one. It follows certain activities until reaching the end point, i.e., the acceptation or rejection of the application. The Petri net in Fig. 2.1 is a WF-net.

Definition 2.10 (WF-nets)**.** *Let $A \subset \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a Petri net over A. N is a WF-net if and only if [82, 7]:*

- *There is a single source place $p_i \in P$, i.e., $\{p \in P \mid \bullet p = \emptyset\} = \{p_i\}$.*
- *There is a single sink place $p_s \in P$, i.e., $\{p \in P \mid p\bullet = \emptyset\} = \{p_s\}$.*
- *The source and sink place are the initial and final marking respectively, i.e. $m_i = [p_i], m_f = [p_s]$.*
- *Every node is traversed by a path from p_i to p_s.*

There are many properties that are interesting in the analysis and modelling of processes through WF-nets. A correctness property is that of soundness.

Definition 2.11 (Soundness)**.** *Let $A \subset \mathcal{A}$ be a set of activities. Let $N = (P, T, F, \alpha, m_i, m_f)$ be a WF-net over A. N is sound if and only if [82, 7]:*

- *(safeness) places cannot hold multiple tokens at the same time.*
- *(proper completion) for any marking $M \in (N, m_i\rangle$, $m_f \in M$ implies $M = [m_f]$*
- *(option to complete) for any marking $M \in (N, m_i\rangle$, $[m_f] \in (N, M\rangle$*
- *(absence of dead parts) there is no dead transitions in $(N, [m_i])$. Meaning, for any transition $t \in T$, there is a firing sequence enabling t.*

Soundness allows for a proper and simple execution of business processes facilitating modelling and analysis.

Process Trees

Process trees are a block structured process models known for their soundness. They are composed of activities in the leaf nodes and operators in the non-leaf nodes. Fig. 2.2 gives an example of a process tree with the different possible operators: choice or exclusive choice (*xor*), parallelism or concurrence (*and*), sequence (*seq*) and finally loop (*loop*). The silent activity τ cannot be observed. It can also indicate that no activity is executed. The set of traces generated by this tree, or its *language*, is equivalent to that of the WF-net in Fig. 2.1. Process trees can be represented textually by performing an inorder traversal of the tree. Example seq(a, and(b,c)). We propose in the following a formal definition of process trees [82, 7].

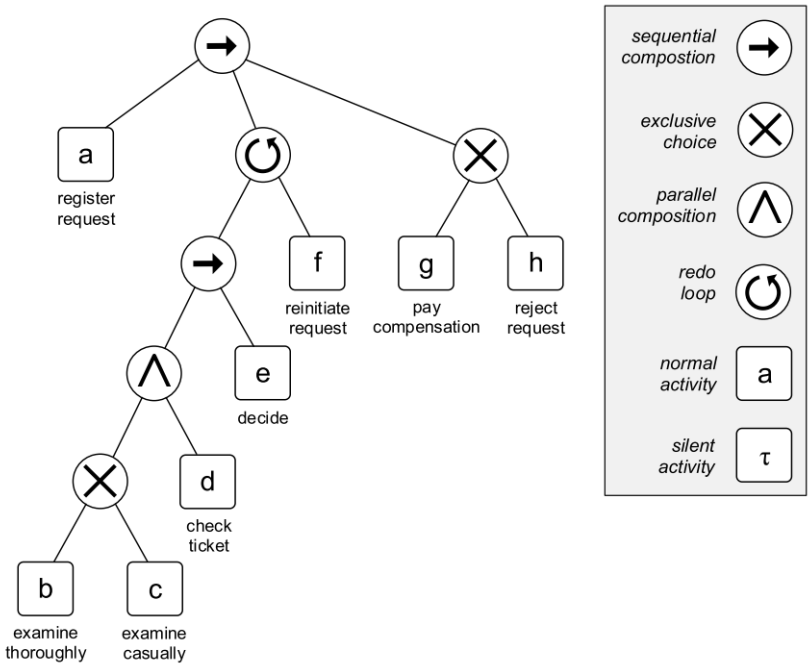


Figure 2.2: Example of a process tree with its different structures [82]

Definition 2.12 (Process tree)**.** *Let $\mathcal{A} \subset \mathcal{A}$ be a set of activities with $\tau \notin A$. $\{seq, and, xor, loop\}$ are the process tree operators.*

- *if $a \in A \cup \{\tau\}$ then $Q = a$ is a process tree.*
- *if $n \geq 1$ $Q_1, Q_2, \ldots, Q_n$ are process trees and $\oplus \in \{seq, and, xor\}$ then $\oplus(Q_1, Q_2, \ldots, Q_n)$ is a process tree.*
- *if $n \geq 2$ $Q_1, Q_2, \ldots, Q_n$ are process trees then $loop(Q_1, Q_2, \ldots, Q_n)$ is a process tree.*

The loop operator is composed of at least two children, the do part and the redo part. The process starts by executing the do part then looping over any of the children of the redo part. Take the tree *loop(a, b, c)*. The possible traces generated or the language of the tree is $\{\langle a\rangle, \langle a, b, a\rangle, \langle a, c, a\rangle, \langle a, b, a, b, a\rangle,$
$\langle a, b, a, c, a\rangle \ldots\}$. Activity a is always executed. It is followed by either b or c in any cycle of the loop. As a side note, the loop operator is the same as the *redo_loop* operator shown in Fig. 2.2 and Fig. 2.3.

Process trees can easily be converted to WF-nets as shown in Fig. 2.3. The transformations can be applied recursively in order to convert a more complex process tree. Thanks to the structuredness of process trees, they can also be straightforwardly converted to other modelling languages like BPMN, YAWL, EPCs, UML activity diagrams, etc.

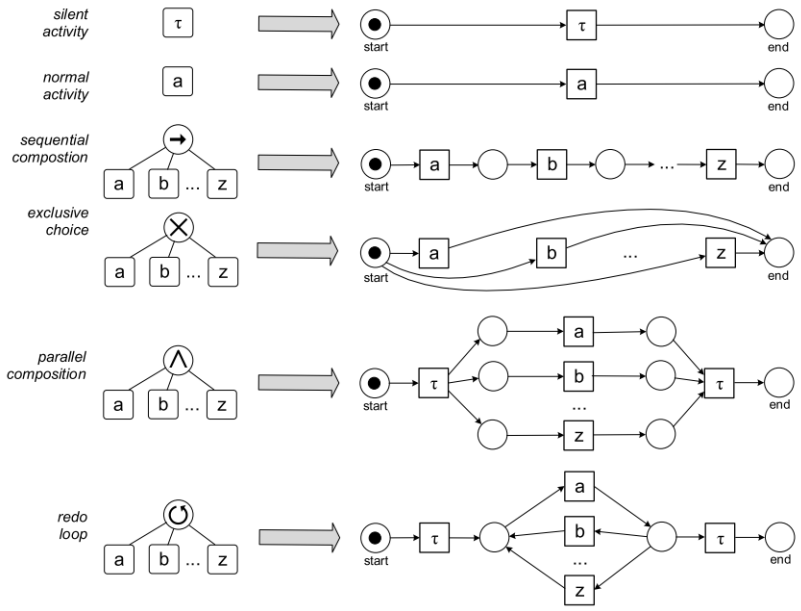


Figure 2.3: Converting process trees to WF-nets [82]

Next, we formalize the semantics of the operators and how they generate the language of the tree.

First, we define the concatenation (.) operator. Let $\sigma_1, \sigma_2 \in A^*$ be two sequences. $\sigma_1.\sigma_2$ represents the concatenation of the two sequences, eg, $\langle a, b\rangle.\langle a\rangle = \langle a, b, a\rangle$. Applied to sets,

the operator is written and defined as : if $S_1, S_2, \ldots, S_n \subset A^*$ then $\odot_{1\leq i\leq n} S_i = S_1.S_2 \ldots .S_n = \{\sigma_1.\sigma_2 \ldots .\sigma_n \mid \sigma_1 \in S_1, \sigma_2 \in S_2, \ldots, \sigma_n \in S_n\}$, e.g, if $S_1 = \{\langle a, b\rangle\}, S_2 = \{\langle c, d\rangle, \langle e, f\rangle\}$ then $\odot_{1\leq i\leq 2} S_i = \{\langle a, b, c, d\rangle, \langle a, b, e, f\rangle\}$.

The shuffle operator ($\diamond$) generates the set of interleaved sequences, e.g., $\langle a, b\rangle \diamond \langle c, d\rangle = \{\langle a, b, c, d\rangle, \langle c, d, a, b\rangle, \langle a, c, b, d\rangle, \langle a, c, d, b\rangle, \langle c, a, d, b\rangle, \langle c, a, b, d\rangle, \}$. The order between the elements of the original sequences is preserved. When applied to sets, the operator is defined as : if $S_1, S_2 \subset A^*$ then $S_1 \diamond S_2 = \{\sigma \in \sigma_1 \diamond \sigma_2 \mid \sigma_1 \in S_1, \sigma_2 \in S_2\}$, e.g., if $S_1 = \{\langle a\rangle\}, S_2 = \{\langle b, c\rangle, \langle d, e\rangle\}$ then $S_1 \diamond S_2 = \{\langle a, b, c\rangle, \langle b, a, c\rangle, \langle b, c, a\rangle, \langle a, d, e\rangle, \langle d, a, e\rangle, \langle d, e, a\rangle\}$. Since the operator is associative and commutative, we generalize it to a set of sets : if $S_1, S_2, \ldots, S_n \subset A^*$ then $\diamond_{1\leq i\leq n} S_i = S_1 \diamond S_2 \cdots \diamond S_n$.

Definition 2.13 (Semantics of operators)**.** *Let $A \subset \mathcal{A}$ be a set of activities. Let Q be a process tree. The language of Q, $\Sigma(Q)$ is given recursively by :*

- *if $Q = \tau$ then $\Sigma(Q) = \{\langle\ \rangle\}$*
- *if $Q = a$ then $\Sigma(Q) = \{\langle a\rangle\}$*
- *if $Q_1, Q_2, \ldots, Q_n$ are process trees and $Q = seq(Q_1, Q_2, \ldots, Q_n)$ then $\Sigma(Q) = \odot_{1\leq i\leq n}\Sigma(Q_i)$*
- *if $Q_1, Q_2, \ldots, Q_n$ are process trees and $Q = xor(Q_1, Q_2, \ldots, Q_n)$ then $\Sigma(Q) = \cup_{1\leq i\leq n}\Sigma(Q_i)$*
- *if $Q_1, Q_2, \ldots, Q_n$ are process trees and $Q = and(Q_1, Q_2, \ldots, Q_n)$ then $\Sigma(Q) = \diamond_{1\leq i\leq n}\Sigma(Q_i)$*
- *if $Q_1, Q_2, \ldots, Q_n$ are process trees and $Q = loop(Q_1, Q_2, \ldots, Q_n)$ then*

  $\Sigma(Q) = \{\sigma_1.\sigma'_1.\sigma_2.\sigma'_2.\ldots.\sigma_m \mid m \geq 1, \forall_{1\leq j\leq m}\sigma_j \in \Sigma(Q_1), \forall_{1\leq j\leq m}\sigma'_j \in \cup_{2\leq i\leq n}\Sigma(Q_i)\}$

  Example 2.14.

- $\Sigma(seq(a, and(b, c))) = \{\langle a, b, c\rangle, \langle a, c, b\rangle\}$
- $\Sigma(xor(a, and(b, c))) = \{\langle a\rangle, \langle b, c\rangle, \langle c, b\rangle\}$

In the remainder of the thesis, we consider trees with binary operators solely where each activity appears only once. Any process tree could be converted to another with binary operators in a straightforward manner. Also, we use the term *word* to designate each potential trace generated by the model, meaning the elements of its language.

2.1.3 Alignments

Conformance Checking is the branch of Process Mining that caters for the discovery of deviations between a certain process model and an event log. It could be used in the context where an organization has a standard business process to follow and would like to check if it is correctly applied in reality. Another use is to check how well does a discovered process model represent the traces in the event log.

Alignments [7] are a tool that uncovers such incompatibility between the log and the process model. An alignment replays a model and finds a matching between a modeled behavior (a word)

and an observed behavior (a trace). Let's take an example to explain this notion. We consider the trace $\sigma = \langle a, d, b, e, h \rangle$ and the four WF-nets in Fig. 2.4. We notice that σ is one of the words generated by N_1 and N_4 but isn't one of those generated by N_2 and N_3. We say that σ perfectly fits N_1 and N_4.

σ	a	d	b	e	h
Complete firing sequence of N_1	a	d	b	e	h

Table 2.2: Alignment γ_1

An optimal alignment is the best matching between a trace and a word of the language of the model. The optimal alignment between σ and N_1 is depicted in Table 2.2. The top row corresponds to σ and the bottom row corresponds to a complete firing sequence of N_1.

As for σ and N_2, there exists three possible optimal alignments given in Fig. 2.5. When replaying the model, the symbol "≫" means matching couldn't be performed between the model and the trace. In alignment γ_{2a}, activity a was observed both in the model and in the trace. Then, activity d of the trace couldn't be executed in the model, so b is executed and we insert a move on model; meaning a move in the model execution that can't be matched to an event in the trace. Subsequently, d is executed synchronously in the model and in the log. Afterwards, b was observed but has no matched executed activity in the model. Thus, we perform a move on log. Finally e and h are executed synchronously.

a	≫	d	b	e	h
a	b	d	≫	e	h

(a)

a	≫	d	b	e	h
a	c	d	≫	e	h

(b)

a	d	b	≫	e	h
a	≫	b	d	e	h

(c)

Figure 2.5: (a) Alignment γ_{2a}; (b) alignment γ_{2b}; (c) alignment γ_{2c}

Ignoring the symbol "≫", the top row always represents the trace σ and the bottom one always represents a complete firing sequence. When the symbol appears on top, it means a move on model; otherwise, it is a move on log.

When replaying a model on a trace, a move on model leads to a firing of a transition thus bringing the model to a new marking, a move on log advances the position of the next event to match and a synchronous move does both.

Alignments allow us to closely analyze the differences between an instance of the model and a trace. In fact, a move on model points to an activity that is supported by the model and should be present in the trace but is skipped while a move on log indicates a superfluous event that shouldn't exist. Both moves on log and on model symbolize deviations except for a move on a silent transition in the model which doesn't impact the fitness of the trace.

Definition 2.15 (move, alignment)**.** *A move* $(x, (y, t))$ *is a couple where x represents an event, t, a transition and y, its label. In case of no duplicate transitions, the move is shortened to* (x, y)*.*

A move is legal:

- *if* $x = y$ *and y is a label of a visible transition t (synchronous move)*
- *if* $x =\gg$ *and y is a label of a visible transition t (visible move on model)*
- *if* $x =\gg$ *and* $y = \tau$ *and t is a silent transition (invisible move on model)*
- *if* $x \neq\gg$ *and* $(y, t) =\gg$ *(move on log)*

An alignment is a sequence of legal transitions such that by removing all "≫" symbols, the top row yields the trace to be aligned and the bottom row yields a complete firing sequence of the model [82, 7].

For a certain trace and a model, there can be multiple (if not an infinite number) alignments. Take σ and N_2. We could consider the alignment $\gamma 2d$ in Table 2.3. Obviously, alignments γ_{2a}, γ_{2b} and γ_{2c} do a better job at bringing together the model and the trace since they use less "≫" symbols. In order to rank alignments, costs on moves can be introduced so that undesirable moves are avoided. A standard cost function δ could be [82, 7]:

- $\delta((x, (y, t))) = 0$ if $(x, (y, t))$ is synchronous. Indeed, this is a perfect matching move and their number should be maximized.
- $\delta((x, (y, t))) = 0$ if $(x, (y, t))$ is an invisible move on model. This is because as stated earlier, a move on a silent transition in the model doesn't impact the fitness of a trace.
- $\delta((x, (y, t))) > 0$ if $(x, (y, t))$ is a visible move on model or a move on log. These behaviors are undesirable and should be penalized. The cost can be different for each of these moves and for each considered activity. For example, skipping the activity *"decide"* is a more serious deviation then superfluous executions of *"check casually"*.

a	d	b	≫	≫	e	h
a	≫	≫	c	d	e	h

Table 2.3: Alignment γ_{2d}

The cost of the alignment is the sum of the costs of its moves. An optimal alignment is one that minimizes the cost function. There can be multiple optimal alignments for a given trace and model. For example, between σ and N_2 there is three optimal alignments (see Fig. 2.5). An algorithm to compute an optimal alignment is presented in [7]. It uses an A^* algorithm that explores a graph where arcs represent all possible alignment moves weighted with their cost and nodes represent a pair of a model marking and a position in the event trace resulting from those moves. The goal is to find a shortest path to a final state which represents a valid alignment (a pair of a final model marking and the end position of the trace) uncovering thus an optimal alignment. The time

complexity of the algorithm is exponential with respect to the complexity of the Petri net and the size of the trace.

Log Move Only Alignment

In the case where the purpose of alignments is not to find a best matching between a trace and a word of the language of the model but to check whether a trace *contains* (in the sense of subsequence containment defined above) such a word, we use *log move only alignments [80]*.

Since we search for a complete modelled behavior in the trace, we forbid moves on a visible transition in the model. In the case where no word is a subsequence of the trace, no valid alignment exists since the final marking is never reached in the model (no complete firing sequence is formed in the bottom row). Take the trace $\omega = \langle a, e, d, f, b, e, e, h \rangle$. A log move only alignment between ω and N_1 could be the one depicted in Table 2.4. Such alignment allows us to know which behavior among all the words of the language of N_1 is present in the trace. In our thesis, we leverage the log move only alignment algorithm presented in [80].

a	e	d	f	b	e	e	h
a	≫	d	≫	b	e	≫	h

Table 2.4: Alignment γ_3

2.2 Related Works

In this section, we go over the state of the art pertaining to our work on behavioral patterns [78, 80]. We start by a quick overview over the first works in Process Discovery in order to show the limitations of these in catering for flexible processes.

2.2.1 Discovery of Structured Processes

α-algorithm

A seminal work in Process Discovery was the α-algorithm [83]. Considering an event log L over a set of activities A, the algorithm searches for the existence of particular relationships between pairs of activities a and b. There exists four ordering relations possible defined in what follows.

- $a >_L b$ if the log contains a trace $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$ and $1 \leq i < n$ such that $\sigma_i = a$ and $\sigma_{i+1} = b$ (Directly follows relation).
- $a \rightarrow_L b$ if and only if $a >_L b$ and $b \not>_L a$ (causality relation).
- $a \#_L b$ if and only if $a \not>_L b$ and $b \not>_L a$.
- $a \|_L b$ if and only if $a >_L b$ and $b >_L a$.

The α-algorithm uses these ordering relations to extract control flow patterns and transform them into a WF-net. Fig. 2.6 shows basic ideas behind these transformations.

The α-algorithm discovers a large class of WF-nets assuming the completeness of the log with respect to directly follows relation. This is a strong condition that is not always satisfied in today's event logs. Besides, the basic algorithm is unable to discover short-loops (of length one or two) which was corrected in the α^+-algorithm [27] and non-local dependencies which was partially addressed in [90]. Moroever, the discovery of WF-nets containing duplicate and silent transitions is not supported. Finally, the algorithm is very sensitive to noise since it has no consideration for frequency.

Heuristics Miner

The Heuristics miner [89] comes as a consistent improvement to the α-algorithm where it handles noisy event logs. It doesn't provide a formal characterization of the types of models it can discover but its extensions are the most widely used in the construction of end-to-end process models. As for α-algorithm, Heuristics Miner defines some ordering relations. Considering two activities $a, b \in A$ and a log L over A:

- $a >_L b$ if the log contains a trace $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$ and $1 \leq i < n$ such that $\sigma_i = a$ and $\sigma_{i+1} = b$.
- $a \gg_L b$ if and only if the log contains a trace $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$ and $1 \leq i < n - 1$ such that $\sigma_i = a$, $\sigma_{i+1} = b$ and $\sigma_{i+2} = a$.
- $a \ggg_L b$ if and only if the log contains a trace $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$ and $1 \leq i < j < n$ such that $\sigma_i = a$ and $\sigma_j = b$.

Heuristics Miner uses these relations to build a dependency graph where each node is an activity and the existence of an arc means the existence of a dependency relationship between two activities. The dependency relationship is subjected to the computation of the following metric where $|a >_L b|$ represents the number of ordering relations of type $a >_L b$:

$$a \Rightarrow_L b = \frac{|a >_L b| - |b >_L a|}{|a >_L b| + |b >_L a| + 1} \tag{2.1}$$

A high value in absolute denotes the presence of a dependency relationship between a and b. The algorithm uses heuristics and thresholds to define what is a high value.

Similarly, Heuristics Miner uses two additional metrics to uncover short loops (length one or two). They are given in the following.

$$a \Rightarrow_L a = \frac{|a >_L a|}{|a >_L as| + 1} \tag{2.2}$$

$$a \Rightarrow_{2L} b = \frac{|a \gg_L b| + |b \gg_L a|}{|a \gg_L b| + |b \gg_L a| + 1} \tag{2.3}$$

After mining the dependency graph, the algorithm uses the following metric to check whether two activities b and c preceded by a form an *AND* construct or a *XOR* one.

$$a \Rightarrow_L b \wedge c = \frac{|b >_L c| + |c >_L b|}{|a >_L b| + |a >_L c| + 1} \tag{2.4}$$

Using this metric, the algorithm creates a causality matrix [89] which is converted to a Petri net in a straightforward manner. Heuristics Miner also mines long-distance relationships (non free choices or non local dependencies). This type of dependency is depicted in Fig. 2.7. After the execution of D, there is a choice between E and F which is conditioned however by the earlier execution of B or C. So this is not a free choice. This kind of structure can't be mined by considering local dependencies between an activity and the next one solely.

Heuristics Miner was followed by other algorithms such as Heuristics Miner++ [18] which takes into consideration non punctual events that span a certain interval, [19] which is adapted to handle streaming event data, Flexible Heuristics Miner [88] which mines a Causal net - an improved representational notation and finally Fodina [86] which improves on robustness against noise and flexibility of configuration.

Other Algorithms

The Inductive Miner [46] is a discovery algorithm that mines block-structured process models (process trees) in a recursive manner. It ensures correctness and soundness of the extracted models which makes it a well used algorithm for process discovery. The algorithm works by subdividing the log into different parts, discovering the process model in each sublog and then recursively joining the discovered process models to construct a tree. The Inductive Miner was extended to navigate through infrequent behavior [47] and incomplete event logs [48]. The IMd frameword [49] works on the directly follows graph (based on the ordering relation proposed in the α-algorithm) and can handle scalability issues.

Genetic Process Miners [28, 16] are algorithms based on the evolutionary natural process. They involve many steps where an initial population of process models is constructed. The initial population then undergoes operations like crossover or mutation to generate children. The fittest children are selected as the next generation and the process reiterates from this new population. The algorithm stops when the individuals (process models) are fit enough, meaning, correctly represent the event log. This process is not deterministic and is largely based on randomness. The method is flexible and robust. It handles noise and incompleteness and allows a great deal of customization. It is however inadapted for large logs because of the big number of models handled. It is to be noted that genetic mining can be coupled with heuristics mining in order to reduce runtimes and to deliver better results.

The α-algorithm, Genetic Miners and the Heuristics Miner algorithms family do not do well with unstructured processes. That is, processes that have too much variability in their behavior. The result of executing the FHM algorithm on a log pertainig to an unstructured process is given in

Fig. 1.2. One can easily see how its complexity prevents all kinds of analysis. The Inductive Miner family has problems with under-fitting and can produce flower models such as the one depicted in Fig. 2.8. In other terms, the model discovered allows far more behavior than what is allowed by the business process model. It is said to be more generalizing.

In the next section, we introduce the field of pattern mining and particularly sequential pattern mining before talking about approaches dedicated to the discovery of insights in flexible processes.

2.2.2 Pattern Mining

Frequent itemset mining and association rule mining are two research areas introduced by the seminal work of [8]. Frequent itemset mining consists in finding patterns in transactional databases such as a retail store database. One could find that the itemset *{milk, cookies}* is frequent meaning these two items are often bought together. A support metric for the presence of the pattern is used with a threshold to attest for its frequency. After the A-priori algorithm [8], many other methods were proposed to accelerate runtimes, handle scalability and optimize memory usage [37, 40, 91].

Association rules are derived from these itemsets. For example, the association rule $milk \rightarrow cookies$ indicates that if one bought milk, there is a strong chance that they also bought cookies. These insights are used for decision making and marketing strategies like co-promoting products or offering discounts. There are many metrics used to assess the validity of a rule. Confidence and odds ratio are two examples but many other measures exist [77].

Sequential pattern mining was introduced by [75] and is the problem of finding frequent subsequences with respect to a defined support in a database of sequences. Database sequences could be the set of sentences in a text or of words in sentences, sequences of items purchased by customers or of web-clicks of users while visiting a website. As for frequent itemset mining, various algorithms were proposed to cater for finding these frequent subsequences. It is to be noted that their output is the same and the only difference is the strategy and data structures used for the discovery. One of this algorithms is GSP [75]. It follows a generate and test approach where the search space of candidate sequences is explored in a breadth-first fashion. It finds all k-sequential patterns (patterns of k element) and uses them to construct candidate $k+1$-sequences by combining those that have $k-1$ common items. In its workings, the algorithm benefits from the monotonicity property stating that all subsequences of a frequent pattern are frequent as well. As it will be discussed later, we adopt the combination principle for COBPAM when generating behavioral patterns. Moreover, we adopt the maximality principle as discussed for sequential patterns in [36].

The GSP algorithm generates candidate patterns that may not be present in the database. In order to avoid that, PrefixSpan [61] uses a pattern-growth approach where it recursively scans the database while growing a pattern. It introduces the concept of projected database which is a subset of the dataset where a pattern appears. Except for the particular property that the projected database is a shifted or suffix version of the original database. The algorithm uses a depth-first search where the projected database of a k-sequential pattern is scanned to find frequent items. These items are then appended to the k-sequential patterns to form $k+1$-sequential patterns. The $k+1$-sequential

pattern's projected database is a suffix version starting right after the item just appended and is then used to form larger patterns and so on. With respect to our works, in a first time, in COBPAM we borrow the concept of a projected database in the form of log projections to evaluate the pattern candidates on the minimal number of traces possible. Then again, in ACOBPAM, we use the idea of suffixes to further shorten the traces that will be used for evaluation.

Lastly, approaches to incorporate contextual data in procedures for sequential pattern mining have been proposed, which include a formal definition of contexts, a characterization of different notions of frequentiality with respect to context hierarchies (e.g., a proposed notion of generality and exclusiveness) and an algorithm to mine the sequential patterns [63]. We later argue that the results exploited by this algorithm can be adapted to the discovery of behavioral patterns.

2.2.3 Discovery of Insights in Flexible Processes

To cope with event logs that show a large degree of variability in the behavior, it was suggested to first employ trace clustering before process discovery. Trace clustering is an active research area concerned with inferring insights from logs of flexible processes [13, 15, 38, 73, 76]. These techniques group traces into homogeneous clusters such that process discovery techniques applied on each cluster yield comparatively structured models. Such techniques do not cater for cases where subdivision is impossible.

Let's take the example of [76]. The authors propose a clustering approach that discovers clusters while reducing the complexity of the process models in each one and later on improves their accuracy. They use a top-down greedy method where the log is recursively divided into two sublogs with optimal complexity. The sublog with the highest complexity is then chosen to continue the division. The algorithm uses the notion of trace behaviors which simply represents a sequential pattern. The log is divided into the two sublogs according to the presence or absence of the trace behavior that optimizes complexity. In the second step, the algorithm HIF [4] is used to improve the fitness of the generated sub-processes. It searches for behavioral patterns in the event log that can't be expressed in the model and converts them into more expressible behavioral structures.

The Fuzzy miner [39] targets the domain of flexible processes and generates simplified and abstracted process models describing only the most significant behavior. It employs two metrics in doing so: significance which indicates the importance given to activities or to a relationship between a pair of them. For example, the importance can be based on a frequency metric, meaning, the more frequent is an event, the more interesting it is. The second metric is correlation which measures how closely related are two events. It can be expressed in terms of similarity of data attributes, of event names, etc. In the simplification step, highly significant behavior is kept as it is. Less significant but highly correlated behavior is aggregated in clusters. Less significant and poorly correlated behavior is abstracted, meaning removed from the final model. The simplified model is enhanced with an emphasis concept where the most significant parts are highlighted. Nevertheless, the algorithm fails to mine certain behavioral structures, such as concurrence and choices.

The Episode Miner [45] is an algorithm that discovers frequent patterns with partial ordering constructs. An episode consists of a set of activities linked by order relationships. The algorithm is akin to Apriori with a two stage approach. At first, episodes with activity nodes only are generated and in the second phase, the partial orders are generated. Episodes of overlapping $k-1$ activities are combined to construct episodes of $k+1$ activities and episodes of overlapping $k-1$ partial orders are combined to get episodes of $k+1$ partial orders. A monotonicity property is applied for pruning. Once, the episodes are discovered, episode rules can be extracted. They are assessed using the confidence metric and a magnitude metric which denotes how much complexity does the left part of the rule adds to the right part. This method, however, does not support loops and choices constructs.

Declare [54] discovers a set of rules that are satisfied by a certain share of traces. These rules adhere to templates represented in Linear Temporal Logic formulas and capture presence, absence, succession and exclusiveness between two activities. The process model is thus specified as a compact set of rules rather than in procedural conventional notations. Any behavior that doesn't violate the rules is allowed which offers more flexibility (see Section 2.1.2). The LTL templates are also more expressive than the partial orders in episode mining. The algorithm generates all possible constraints and checks their validity in the log while handling noise. In order to reduce runtimes, a percentage of the most frequent events can be specified for rules generation. A thorough comparison between behavioral patterns mining and Declare mining is available in Section 6.2.3.

Following Declare, an extension, TBDeclare [31] was proposed. Instead of having one activity only in the target part of the declarative constraints (consequent of the constraint), it offers the possibility to mine disjunctions of target activities. It follows that many event classes are branched out of the activation activity (antecedent of the constraint). The algorithm relies on two metrics for the constraints: support and confidence as they are defined in Association Rule Mining. It also exploits two key properties: set-dominance, in the sense where if a rule states that b or c follow a and another states that b or c or d follow a, then the former rule is discarded because the target set is contained in the latter rule. The second property is subsumption hierarchy where certain templates of rules imply others. So the discovery of one rule may imply the discovery of the second one. The former one is then kept and the second one discarded. On another note, even though the number of activities is no longer constrained within the consequent of a constraint, the operator on that side is limited to an exclusive choice while the activation side still holds a unique activity. In COBPAM, however, there is no bounds to the number of activities.

The notion of behavioral patterns used in our work has first been proposed in [80] under the term Local Process Models (LPMs). The authors propose a generate and test algorithm where process trees are grown by replacing an activity with a small behavior in the form of a subtree. They propose different metrics to evaluate the interestingness of a process tree, namely, support, confidence, converage, determinism and language fit. They also use a pruning strategy to reduce the number of trees evaluated. During this evaluation they use log move only alignments. This first discovery algorithm was not grounded in the traditional definition of support, as known from sequential pattern mining. In addition, by following its specific generate-and-test approach, the

respective algorithm incurred the overhead of redundant testing of pattern candidates, and suffers from runtime issues due to the computation of alignments on the entire log. Compared to the mining of LPMs, our COBPAM algorithm adopts a well-established definition of support for behavioral patterns. It further provides several innovations. Mined patterns are guaranteed to show desirable properties (maximality and compactness), while the discovery algorithm also leverages pruning strategies and explores pattern candidates solely on a subset of the traces of a log. Some patterns that were missed by LPM disocvery like ones that are an exclusive choice between two infrequent behaviors are also uncovered. In a second step, ACOBPAM further improves the runtimes and the relevancy of the discovered patterns. It also provides a global visualization of the discovered trees leveraging different dependencies between them. To the best of our knowledge, such visualization of behavioral patterns was never proposed in the litterature.

The initial approach to discover LPMs has been extended to include goal-driven strategies to mine patterns based on their utility and constraints satisfaction [78]. An initiative which is orthogonal to our work. The discovery is oriented towards answering business questions through the definition of constraint and utility functions. A constraint function defines a requirement that either holds or not. It is a strong preference. The utility function however is a soft measure of how interesting a pattern is in answering the business question at hand. Both types of functions can be defined on four scopes : trace-level, event-level, activity-level and model-level and can be combined to indicate the total utility of an LPM. In the following, we present the characteristics of each scope:

- The trace-level scope deals with the parts of the trace that fit the LPM. More exactly, the events, the event properties in these parts and the case properties where they appear. For example, a utility function on this scope could be used to discover LPMs whose events hold the lion's share of the financial costs in the trace.
- The event-level scope is related to utility and constraint functions that involve event properties without needing case information. For example, a constraint function in this scope could be the discovery of LPMs executed solely by a certain set of resources.
- Activity-level utility and constraint functions are defined with respect to the frequency of activities in the log and the parts of the log that fit the LPMs.
- Model-level utility and constraint functions define structural properties of the LPM and are independent of the log.

There is another mention of behavioral patterns in the literature defined in another way than in our work. In [33], the authors propose an algorithm to mine behavioral patterns describing resource-annotated activities (an activity a executed by resource R_1 becomes a new activity [a, R_1]). The method needs a general describing process consisting of resource-agnostic activities. So, it uses Process Discovery techniques such as Inductive Miner to construct one. From there, it generates instance graphs based on traces including resource data. It is to be noted that this generation uses information from the mined end-to-end model. If there is discrepancies, the instance graphs are repaired. The particular resource-annotated traces yield an unstructured process model from which behavioral patterns are extracted in the form of frequent sub-graphs among the set of instance graphs. Two measures of interest are defined for the sub-graphs. The

occurrence frequency and the sub-graph complexity. The more complex, a behavioral pattern is, the more insight it holds. In order to discover the sub-graphs, the algorithm uses hierarchical clustering where the sub-graphs are organized in a lattice such that lower-level sub-graph contain top-level ones as nodes. Each discovered sub-graph is the representative of the cluster of instance graphs containing the sub-graph. Here, the behavioral patterns are frequent subgraphs of a global process model that needs to be discovered whereas in our case, no such model is needed as only the frequency of generated patterns is important. Also, the method [33] involves the resource perspective in the discovery contrary to our work.

2.2.4 Data-aware Discovery of Insights in Flexible Processes

In the continuity of the aforementioned algorithms, there are a few that take into account contextual data in the discovery process.

[53] propose the first data-aware Declare Miner. Declarative rules are enhanced by data attributes using a First-Order Linear Temporal Logic notation. An example of a data enhanced rule would be, if activity A appears and condition x holds then B must follow. The approach exploits the concept of constraint activation which indicates the occurrence of the activation activity (antecedent of the constraint) in the trace. Then, if the target activity appears, the constraint is *fulfilled*. If not, it is *violated*. The method first mines the set of declarative rules from the log then performs a replay on it to detect the fulfilled and the violated rules in each trace. Afterwards, it uses a classification technique to discriminate between fulfillments and violations using the data attributes. The method presented in [23] is used to discover conjunctive expressions allowing the discrimination while decision trees are used to discover disjunctive ones.

In [69], the authors extend the work of [53] with other types of conditions. They incorporate the notion of payload which is the set of pairs attribute/value associated to an activity and use it to define the following:

- activation condition: is a statement that must hold when the activation occurs. For example, a constraint could be: whenever A is executed and condition x holds, then B follows. This type of conditions can be discriminative allowing to differentiate the fulfillments and violations of constraints or descriptive in the sense that they describe the values of the attributes connected to fullfilments regardless of the violations.
- target condition: a condition is imposed on the payload of the target activity when the constraint is activated. For example: if A is executed then B follows with condition x.
- correlation condition: is a statement that must be valid in the target activity involving the payloads of both the target and the activation.
- temporal conditions: specifies a minimal and maximal time distance between the activation and target activity's timestamps.

The authors use two metrics to validate rules : support and confidence. Moreover, they propose a highly customizable SQL based discovery of all the types of conditions. The queries must however

be manually specified.

[50] tackle a related problem of finding correlations between activation and target conditions. An example of a such enhanced rule would be, if A occurs with condition x then B will eventually follow with condition y. The authors propose two approaches. Both rely on already discovered data-agnostic constraints.

The first one constructs fulfillment and violation feature vectors out of the target and activation payloads. Then, the target payloads of the fulfillment feature vectors are clustered. A rule-based classification algorithm called RIPPER is used to extract target payloads representative of each cluster. The fulfillment feature vectors are subsequently labeled with the representatives and projected on activation payloads. The projected labeled fulfillment feature vectors and the violation feature vectors are used by RIPPER again to find correlations between activation and target conditions.

The second approach relies on a method from unsupervised descriptive knowledge discovery called redescription mining. It takes as input the set of fulfillment feature vectors and outputs correlations between attributes of the activation payloads and attributes of the target payloads.

Another method [14] uses event correlations to assess the relevance of a constraint. It defines events correlations as relationships between event attributes. These relationships can help:

- prune uninteresting constraints. The authors suppose that events in significant rules share attributes.
- disambiguate events. For example, in a trace $\langle A, A, B, C \rangle$, the rule "if A is executed then B must follow" is ambiguous. Which activity A in the trace is concerned by the rule ?
- extract insights about the process. For example, if some event correlations hold for most of the traces where the constraint is fulfilled, the remaining ones could be considered outliers.

On another note, the authors propose a categorization of correlations:

- Property-based correlations: events are grouped together based on a function on their attributes.
- Reference-based correlations: two events are correlated if the attribute of one is connected through a function to an attribute of the other.
- Moving time-window correlations: two events are correlated if they occur within a specific duration from one another.

The algorithm starts by generating all possible correlations for a given constraint. Then, for each constraint, the ambiguous rules are discarded and the support of a correlation is computed as the number of non-ambiguous constraints where the correlation is satisfied over the total number of non-ambiguous rules. A correlation is significant if its support exceeds a certain threshold. The significant correlations are then used to disambiguate ambiguous activations or as features in machine learning approaches in order to reach the above goals.

Notice that the majority of the previous data-aware declarative methods utilize event data not trace attributes which makes them orthogonal to our work. Besides, our focus is not on an enhancement of discovered rules with data conditions, but on the discovery of novel patterns that would be missed by a non-data-aware discovery approach. We need also to consider the intrinsic differences between the Declare constraints mining and behavioral pattern mining (see Section 2.2.3).

Another interesting work that could be leveraged for dealing with contextual data in flexible processes is [11]. The authors present a formalization of a framework revolving around a process cube representation of event collections similar to what is done in OLAP. It exploits several notions:

- An event base is a set of independent events characterized by properties.
- A process cube structure contains many data attributes organized in dimensions and hierarchies.
- A mapper function creates a relation between the event base properties and the process cube structure attributes.
- A process cube view represents the visible part or the perspective of the process cube after applying typical OLAP operations such as slice, dice, roll up, etc.
- A cell set is the set of visible cells of a process cube and represents pairs attribute/values that satisfy the realized operations.
- A materialized process cube view places events satisfying cell sets' attribute/value pairs inside process cube cells.

The method allows to manipulate the process cube using OLAP operations to observe the events under different perspectives in order to get different insights and enable comparisons. Particular process mining algorithms can be executed on each perspective or view. For instance, COBPAM could be executed on each view to extract data-dependent behavioral patterns. Our work, however, links the attributes and the hierarchies in the view to the behavioral patterns by defining many types of frequencies discovering correlations and dependencies between behavioral patterns and context data.

Other algorithms that consider data attributes are geared towards delivering end-to-end process models and are hence not suitable for flexible processes. They include [52] which mines finite state machines annotated with data. Yet, the approach lacks concurrency support which is a main operator in business processes. Decision Miner [67] is an algorithm that annotates choice constructs with data conditions on a pre-built process model. Finally, Context-aware Inductive Miner [71] integrates contextual data while discovering the model. It delivers process trees enhanced with a data operator.

2.2.5 Discovery of Causality Relationships

Causal relationships discovery is a well explored research area. One main branch is probabilistic causality which makes use of Bayesian networks [59] and other similar probabilistic models. Though many works used Bayesian networks for the discovery of causal structures [3, 42], they are computationally heavy. Some methods [22, 72] can be more efficient by constraining the search to some local structures instead of discovering the whole network but in doing so are limited.

Since our proposal is inspired from medical studies, namely, cohort studies, we present in the following the different methods for assessing causal relationships in the medical field [41]. They are ordered from the ones providing the weakest to the ones providing the strongest evidence of causality as presented in the Evidence-Based Medicine pyramid [66]. The firsts belong to the observational studies category. Unlike the experimental studies, they simply observe phenomena without acting on their course. We can mention case reports and case series where data from medical records is collected and the evolution of the patients is analyzed, for instance after the administration of treatment, without the presence of a control group where such treatment was not given. They are mostly used to establish hypothesises rather than to confirm causality. Indeed, in the case where an outcome was observed after treating the patients, it cannot be associated with certainty with this treatment since other factors could have played a role. In the absence of a control group, we cannot speak of causality.

Cohort studies are another type of observational studies. They usually monitor two groups of patients where the exposure factor/treatment appears in one and not in the other. This allows for a comparison of their effects on a specified outcome. Although, researchers try to keep the characteristics of the two groups identical, there may be hidden confounding variables. Those are variables that cause the outcome while being correlated to the exposure (treatment). For example, a strong correlation can be observed between high coffee consumption and lung cancer. The reason is the presence of a causal relationship between smoking and higher coffee consumption [10] as well as between smoking and the incidence of lung cancer [62]. A cohort study that misses the hidden variable, smoking, could conclude on a causal link between high coffee consumption and lung cancer which is evidently false. Note that there are two types of cohort studies, prospective where patients are followed in real time and assessed for the appearance of the outcome or retrospective where data is retroactively picked and analyzed.

Case-control studies analyze medical records and pick two groups of patients, one with the presence of the outcome and another without. The data about the exposure/treatment factor is then retrospectively analyzed. This type of study is commonly used to assess treatment failures.

Finally, the experimental study, randomized controlled trials, is one of the strongest causal study types and lies at the top of the Evidence-Based Medicine pyramid. It consists in constructing two groups where one is specifically administered a treatment as part of the study protocol. The patients are randomly affected to their groups in order to mitigate the confounding variables. An aspect that represents the strength of this method. The appearance of the outcome is then monitored.

On a final note, since in our work, we deal with observational correlated data, causal association rules discovery [51] caught our attention for its simplicity, intuitivity and efficiency. It uses retrospective cohort studies that are one of the most trustworthy observational studies that look into readily available data. Again, one should keep in mind that the methods presented here are tailored for causality studies. By adopting them, our approach finds the cause of patterns rather than predict or explain behavior based on case features as proposed in other works [30, 29].

2.2.6 Alignment and Conformance Checking

As stated earlier, Conformance Checking is the sub-field of process mining concerned about the studies of deviations between models and event logs. Given those two, conformance checking algorithms give a measure of the discrepancies encountered. Many metrics were proposed; the first and most important to us being the fitness. It indicates how well a log trace can be affiliated to a model. In other terms, how sure are we that this specific trace was generated by the handled model. The precision is a second metric that we use in our thesis. A low value indicates that the discovered model allows far more behavior than what is actually authorized by the underlying process model. In turn, the metric generalization describes how well does the discovered model allow for behaviors that are part of the underlying process model although nonexistent in the event log. The goal is thus to find a balance between the two metrics. The discovered model must be precise enough as to not allow behavior nonexistent in the underlying process (avoid under-fitting) and general enough as to allow for additional behavior not present in the log but still backed-up by the underlying process (avoid over-fitting). The last metric is simplicity which applies the Ockham's Razor stating that the simpler the model discovered the better. Indeed, simplicity allows for ease of analysis and quality insights [82]. Along with frequency, the precision metric is used to evaluate our behavioral patterns.

Coming back to the fitness metric, traditionally, two main categories of methods are employed to compute it. The first makes use of alignments as defined in Section 2.1.3. The second is coined *Log Replay*. In one method of this category, called token-based log replay, the trace is replayed by the model such that additional tokens may be inserted to reach the final markings when the trace doesn't allow the completion. In turn, there may still stay leftover tokens in the model that were not consumed. Both types of tokens represent unwanted deviations [34]. A third category that made its appearance is constraint-based conformance checking where instead of an imperative model, a set of constraints or rules, such as in a Declare model, are tested for satisfaction by the log traces [12, 20]. Some of these works incorporate multi-perspective conformance checking that takes into account various dimensions like data or organization [12].

Alignment-based conformance checking represents the de facto standard in the field. Contrary to the log replay techniques, they provide more precise and valuable results. Most commonly, the techniques are designed for procedural or imperative models. Besides, since the alignment operation uses the A^* algorithm, works propose different forms of cost functions. We can mention the ones based on: manually defined costs for specific process deviations [25], probabilities [44], region theory and state similarity [17], etc.

In the following, we will mention some contributions that leverage alignments in different settings. By detecting the spurious activities or missing links in the model, [35] are able to repair process models so that they become more in line with reality. Moreover, alignments can be leveraged to adapt to multi-dimensional settings. In [55] authors use advanced alignments to include the resource and data dimensions. Cost functions may also be extended to take into account data and agents information, such us [25] where the optimization problem is solved through integer linear programming. Another work uses data-enhanced causal nets for a multi-perspective alignment of BPMN process models [26]. On another hand, there is a handful of methods that use alignments on declarative models such as [24, 21].

In order to cater for large scale event logs and/or models, other approaches use a divide-and-conquer strategy where they aim to align parts of the model instead of the whole one [58, 74, 87]. Since the parts or submodels are aligned individually, parts only of the traces are used. However, the results are only an appoximation of the complete traditional alignment. These techniques can still be used to locate fitness problems in a large model and focus on the most interesting parts for the stakeholders. It is important to precise that ACOBPAM uses somehow the divide-and-conquer approach by decomposing the trees and trying to separate the already aligned behaviors from the ones that need realignment.

A particular work of interest for us is [85] where authors propose an incremental framework for aligning process models in the context of online conformance checking. Instead of a complete event log recording past behavior of a process until its termination, the study of deviations is performed on event streams, meaning, conformance is realized in real-time concurrently with the process execution. In this setting and given a process model, a prefix-alignment is computed upon receiving each event. This particular alignment takes into account the fact that subsequent events may still bring synchronous moves and append them into the alignment reducing potentially its cost. As such, a notable property is satisfied by the technique, prefix-alignments always underestimate the real cost of the complete alignment between the presently received trace and the model. If the stream for a certain case comes to an end, the prefix-alignment already calculated is not guaranteed to be optimal although it represents a lower bound for the actual optimal alignment. The degree of deviation can hence be bounded. In some cases, the new prefix-alignment uses the last step of the A^* algorithm of the previous prefix-alignment; hence the incremental denomination.

Compared with the last contribution, our ACOBPAM algorithm has some key differences. First we have at disposal a complete log containing information about already terminated cases. So, in our case, each alignment computed is guaranteed to be optimal. Next, there is not only one process model to align but several process trees. In fact, instead of using an *incomplete* alignment (prefix-alignment) and adapt it to the *same* process model, we use *complete* alignments to construct those of *more complex* trees. In fact, the new alignments in our method are not created by resuming from a certain step in the A* algorithm from the alignment before. A complete new algorithm is enacted which may or may not re-execute a complete A* star algorithm. Besides, instead of the incrementality based on continuing the A* algorithm, the incrementality in ACOBPAM uses the "results" of a previous independent alignment. These results include a specification of the already aligned behaviors along with their occurrences in the traces. If these behaviors are also part of the

tree to evaluate, we call them validated context because they don't need to be replayed for the new model. Moreover, our alignment uses log moves only and searches for a complete occurrence of a word of the language of the model in the trace.

2.3 Conclusion

In this chapter, we started by laying the ground for the presentation of different algorithms in Process Mining through preliminary definitions; mainly, event logs, Petri nets, process trees and alignments. Afterwards, we gave a quick tour of the related work. Particularly, we presented the classical methods for Pattern Discovery in structured processes as well as the new methods more geared towards flexible processes either with or without the data dimension. We saw that the latter are quite diverse with little to no similarity. Nevertheless, for each approach, we discussed the limits and how our contributions differentiate from them. We also discussed subsidiary fields like Pattern Mining, alignments and causality study, the last one being an additional and interesting link between data and pattern. The next chapter introduces our first approach COBPAM for extracting behavioral patterns out of flexible processes.

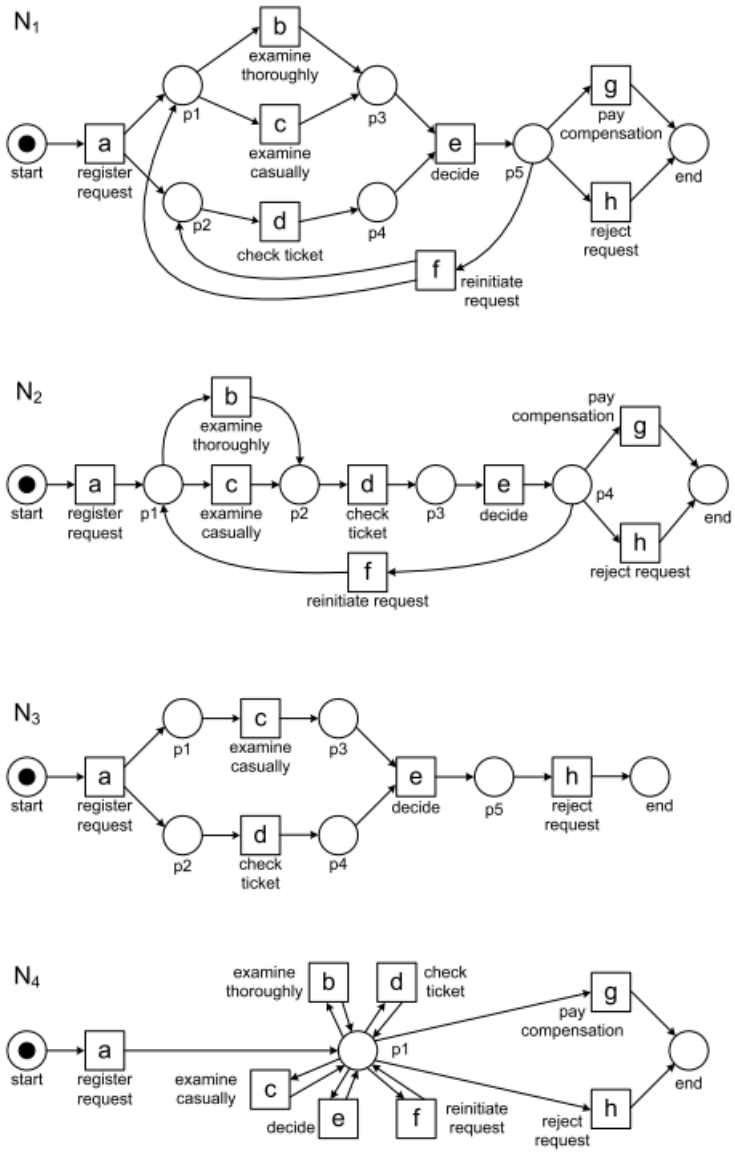


Figure 2.4: Example of four WF-nets [82]

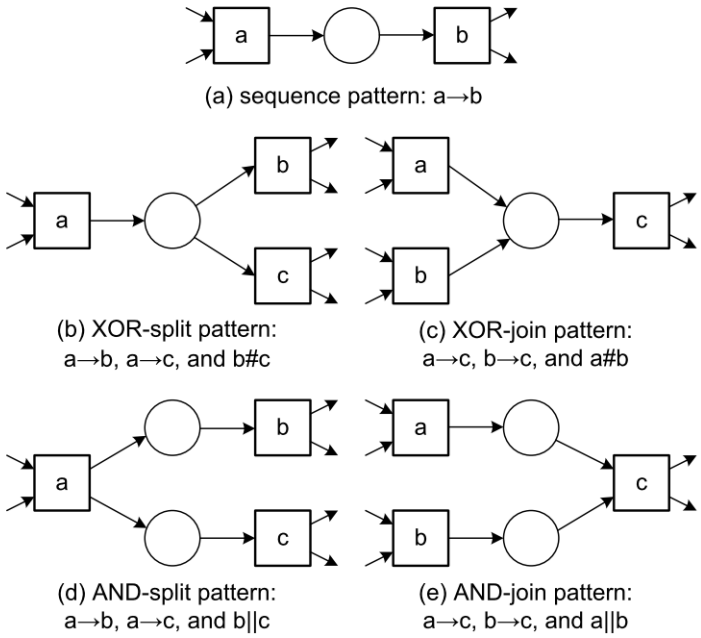


Figure 2.6: Typical control flow structures and the ordering relations they leave in the event log [82]

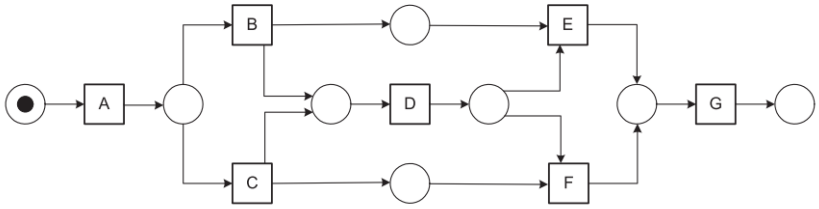


Figure 2.7: A process model with a non-free choice construct [89]

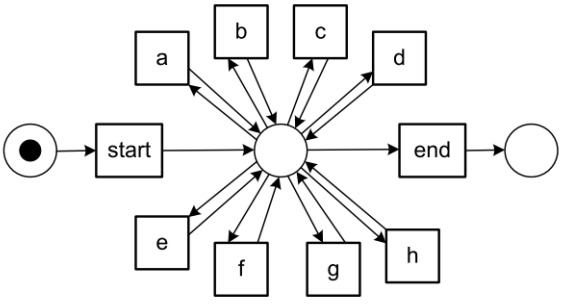


Figure 2.8: A flower model allowing any sequence of activities [89]

3 Behavioral Pattern Mining with COBPAM

Contents

We recall that we introduced COBPAM as a response to the research goal **G1** which is to improve over the state-of-the-art algorithm LPM Discovery [80]. Inspired from the Sequential Pattern Mining algorithm, GSP [75], this first algorithm uses a combination based approach to construct behavioral pattern, i.e., process trees present in most of the traces in the event log. In the following, we present in *Section 3.1* the said combination operation and its supporting structures. Then, we list the quality metrics we use to evaluate insightful process trees in *Section 3.2*. Finally, *Section 3.3* discusses the core of the algorithm, its target properties, optimizations and pseudocode.

3.1 Algebraic Operations and Structures on Process Trees

In this section, we devise a method for constructing process trees incrementally. We propose to combine two process trees composed of n activities to derive process trees of $n + 1$ activities. The process trees combined must be identical except for a single leaf node. We further impose conditions on these leaves, as follows:

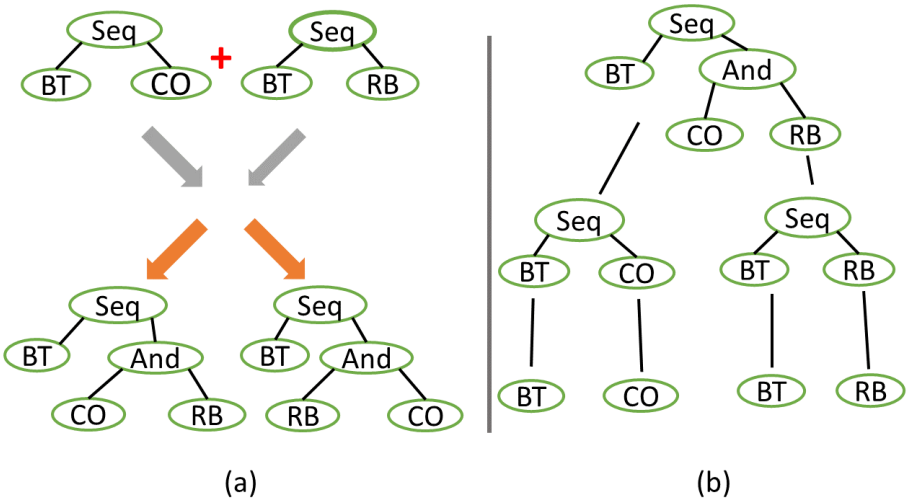


Figure 3.1: (a) a combination operation of two process trees (b) the construction tree of the process tree *seq(BT, and(CO, RB))*

Definition 3.1 (Potential Combination Leaf)**.** *Given a process tree P of depth i, a leaf node a of depth d is called* potential combination leaf*, if $d \geq i - 1$ and there is no leaf b of depth d' on the left of a such that $d' > d$.*

Two process trees that can be combined are called seeds.

Definition 3.2 (Seeds)**.** *Process trees P_1 and P_2 are called* seeds*, if they contain two potential combination leaves, a in P_1 and a' in P_2, such that by replacing a in P_1 with a', we obtain P_2.*

The above notion requires that both process trees are identical except at the level of the leaves a and a'. For instance, from our running example in Fig. 1.3, *seq(BT, CO)* and *seq(BT, RB)* are seeds. Next, we formally define the algebraic operation of combination.

Definition 3.3 (Combination operation)**.** *A* combination *of two seeds P_1 and P_2 through an operator x is an operation generating two process trees. Starting from P_1, the combination leaf a is replaced by the operator x, whose children are set to a and a'. a becomes the left child in one resulting tree, and the right child in another one. a and a' are called the combination leaves and x is called the combination operator.*

Fig. 3.1a shows an example of a combination of two process trees *seq(BT, CO)* and *seq(BT, RB)* through the concurrency operator, which results in two trees: *seq(BT, and(CO, RB))* and *seq(BT, and(RB, CO))*.

Thanks to the conditions characterizing the potential combination leaves, the following theorem holds true: t

Theorem 3.4. *For a process tree P of depth $i \geq 1$, there is a unique pair of seeds P_1 and P_2, whose combination through an operator x results in P. P_1 and P_2 are called 'the' seeds of P and x is called the defining operator of P.*

Proof. Let's consider a process tree P of depth i. It will contain at least two leaves of depth i since all operators have two children. Let $2n$ ($n \geq 1$) be the number of leaves of depth i in P. We will consider two cases:

$n = 1$**:** P has only two leaves a_j and a_k of depth i that are children nodes of some operator x. In parallel, let us consider two process trees P_1 and P_2 which are the same as P except that the node x is replaced with the activity a_j in P_1 and with a_k in P_2 leaving thus P_1, P_2, a_j and a_k with depth $i-1$. Now, since there are no deeper leaves on the left of a_j (resp. a_k) in P_1 (resp. P_2), P can be obtained by applying the combination operation on P_1 and P_2 through the operator x.

Moreover, there is only a unique couple of seeds whose combination results in P. To prove that, we suppose that there exists another couple P'_1, P'_2 that can produce P when combined. This means that P'_1, P'_2 are similar except for a single leaf (a'_j in P'_1 and a'_k in P'_2). The two leaves appear in P under a certain operator x' which corresponds to the combination operator. Besides, considering how the combination is realized, any other branch in P appears both in P'_1 and P'_2 which means the depth of these two processes is i since the branches in P leading to a_j and a_k are of depth i.

However, the combination being performed means that, in P'_1 and P'_2, a'_j and a'_k are of at least depth $i-1$. This leaves two cases:

- a'_j and a'_k are of depth $i-1$ in P'_1 and P'_2 meaning their depth in P after the combination is i which contradicts the hypothesis that there exists only two leaves a_j and a_k of depth i in P.
- a'_j and a'_k are of depth i in P'_1 and P'_2 meaning their depth in P after the combination is $i+1$ which contradicts the hypothesis that P is of depth i.

We conclude that P'_1 and P'_2 cannot exist and that P_1 and P_2 are the unique seeds that produce P.

$n > 1$**:** In this case, P has at least four leaves of depth i. Let a_j be the leftmost deepest leaf, x its parent operator and a_k the other child of x. We then consider two process trees P_1 and P_2 that are the same as P except that x is replaced with a_j in P_1 and with a_k in P_2. These process models are of depth i and contain $2(n-1)$ leaves of the same depth. Moreover, by construction, there are no deeper leaves on the left of a_j (resp. a_k) in P_1 (resp. P_2) and a_j (resp. a_k) is of depth $i-1$ in P_1 (resp. P_2). That means P_1 and P_2 can be combined resulting in P.

In addition, these two processes are the unique pair whose combination outputs P. Again, to prove this, we will consider another pair P'_1 and P'_2 which, when combined results in P. This means, the only difference between them is a single leaf (a'_j in P'_1 and a'_k in P_2). These two leaves appear in P under a certain operator x' which corresponds to the combination operator between P'_1 and P'_2. Besides, any branch leading to a leaf different than a'_j and a'_k present in P also appears in P'_1 and P'_2. Hence, the depth of the two fragments is also i and the leaves a'_j and a'_k are of at least depth $i-1$ in P'_1 and P'_2. We will separate the cases :

- a'_j and a'_k are of depth $i-1$ in P'_1 and P'_2 meaning their depth in P is i and since a_j is the leftmost deeper leaf, then it's on the left of a'_j and a'_k in P. Thus, there exists a deeper leaf (of depth i) on the left of a'_j (resp. a'_k) in P'_1 (resp P'_2) which means P'_1 and P'_2 are not seeds and that contradicts the combination conditions.

- a'_j and a'_k are of depth i in P'_1 and P'_2 meaning their depth in P is $i + 1$ which contradicts the hypothesis that P is of depth i.

We conclude that P'_1 and P'_2 can't exist and that P_1 and P_2 are the unique couple of process trees that produce P. In conclusion, any process tree of depth $i \geq 1$ is the result of a combination of two unique seeds. □

Given that every process tree of depth larger than zero results from the combination of two unique seeds, we introduce additional structures.

Definition 3.5 (Construction tree)**.** *Given a process tree P of depth $i \geq 1$, we define its* construction tree*. The nodes of this tree are process trees: The root is P, the leaves are trees with single activity nodes; the children of a non-leaf node are its seeds.*

Fig. 3.1b exemplifies the construction tree of the process tree *seq(BT, and(CO, RB))*.

Definition 3.6 (Construction graph)**.** *We define the* construction graph *over the set of activities A. It is a directed acyclic graph. Its (infinite) set of nodes is given by all possible process trees. An edge is defined between nodes n_1 and n_2, if n_1 is a seed of n_2. We say that n_2 contains n_1 through the defining operator of n_2.*

To identify a tree, COBPAM uses the concept of representative word.

Definition 3.7 (Representative word)**.** *Each process tree P is assigned a* representative word *$RW(P)$, a sequence of characters. It is constructed by pre-order traversal of its nodes, outputting activities and operators.*

For example, the representative word of $seq(BT, and(CO, RB))$ is '(BT (CO RB and) seq)'.

3.2 Quality Metrics

This section defines metrics to evaluate the quality of a behavioral pattern with respect to a log in the form of a set of cases (we adopt the definition of simple event log). Following the reasoning given in [80], we consider quality metrics that are based on a notion of behavioral containment. That is, given a case c and its associated trace $\hat{c}$, the behavior of a process tree P is exhibited by the trace, if there exists a word $w \in \Sigma(P)$ of the language of P, such that w is a subsequence of $\hat{c}$. For example, in Fig. 1.4, the process tre defines the language $\{\langle BT, SW, CO, RB\rangle, \langle BT, SW, RB, CO\rangle\}$. Trace 1 of the event log in Fig. 1.4a exhibits this behavior, since $\langle BT, SW, CO, RB\rangle$, is a subsequence of Trace 1. Trace 8, in turn, is a counter-example. It does not exhibit the behavior since neither of the two words are subsequences of it.

Using the log move only alignments defined in Section 2.1.3, we define two functions: a boolean one $\epsilon(\hat{c}, P)$ that returns one, if $\hat{c}$ exhibits P, i.e., trace $\hat{c}$ fits the process tree P; otherwise, it returns zero. A second one, $\upsilon(\hat{c}, P)$, returns the exact behavior exhibited by the trace $\hat{c}$ among all the words generated by the process tree P .

We employ these functions to define the concept of projection and several quality metrics that provide the foundation for the COBPAM algorithm.

Definition 3.8 (Projection)**.** *A* projection *is a subset of an event log L associated with a process tree P that contains the cases of which the traces can be aligned with P:*

$$proj(P, L) = \{c \in L \mid \epsilon(\hat{c}, P) = 1\}.$$

Definition 3.9 (Frequency and Support)**.** *Given an event log L, the* frequency *of a process tree P is the number of cases that exhibit its behavior:*

$$frequency(P, L) = \sum_{c \in L} \epsilon(\hat{c}, P) = |proj(P, L)|.$$

Its support *is the frequency over the size of the log:*

$$support(P, L) = \frac{frequency(P, L)}{|L|}.$$

Definition 3.10 (Precision)**.** *Given an event log L, the precision of a process tree P is the ratio of the behavior seen in the log and all the behavior allowed for by the model. If P does not contain loop operators, it is defined as:*

$$precision(P, L) = \frac{|\{\upsilon(\hat{c}, P) \mid c \in L \wedge \epsilon(\hat{c}, P) = 1\}|}{|\Sigma(P)|}.$$

If P contains loop operators, its language will be infinite, so that its precision will tend to zero. In this case, we use the n-language of P:

$$precision(P, L) = \frac{|\{\upsilon(\hat{c}, P) \mid c \in L \wedge \epsilon(\hat{c}, P) = 1\}|}{|\Sigma_n(P)|}$$

Here, a low n value allows for loops being 'precise' with little repetition, whereas a higher value imposes more repetitions before a model is considered to represent the log behavior well. This decision has to be taken manually by an analyst and offers control over the number of repetitions observed in the log that shall be taken as evidence for the presence of an actual loop in the model. The question is how many forms does the loop need to appear in before it becomes an actual loop model; not just ad-hoc repetitions observed and not backed-up by the underlying process model. In other words, at which point, do we consider that at some moment a decision is made to repeat a set of activities again? Indeed, a loop model gives the possibility to execute even more repetitions when some conditions are met. So, incorporating the right loops in the discovered models will ensure a higher generalization when considering new traces.

3.3 Behavioral Pattern Discovery with COBPAM

This section presents a new algorithm to discover process trees that represent frequent behavior in a log. Our idea is to explore a construction graph, starting from process trees of single activities. Each process tree is evaluated against a part of the log that may exhibit its behavior to calculate the aforementioned quality metrics. We also introduce a projection based optimization and pruning rules to limit the number of process trees to evaluate and the number of traces used for evaluation.

In Section 3.3.1, we discuss a monotonicity property that is later exploited in our pattern search. We also define what we consider compact and maximal process trees in Section 3.3.2, and introduce optimization based on projections in Section 3.3.3. Finally, in Section 3.3.4, a detailed view of the algorithm is given.

3.3.1 A Monotonicity Property

The combination operation introduced in Section 3.1 replaces a potential combination leaf with a sub-tree representing a portion of a behavior that either extends the behavior of the original tree (when using the choice operator) or constrains it (when using a sequence, loop, or concurrency operator). When evaluating a process tree whose defining operator is a constraining operator (sequence, loop, concurrency), we essentially want that the trace exhibits all the behavior of its seeds except at the position of the combination leaf. At this position, additional behavior shall replace the appearance of an activity in the trace. The shared behavior between a process tree and its seeds represents a context to which the additional behavior is joined. Hence, if a trace does not exhibit the context, there is no need to evaluate the added behavior.

From the above, it follows that, if one of the seeds is not frequent, there is no need to evaluate the tree, as it will be infrequent too. This is a monotonicity property of the support metric. Based thereon, we specify a **first pruning rule**: If a seed is infrequent, it should not be combined using a constraining operator.

3.3.2 Compact and Maximal Process Trees

We further direct our search for behavioral patterns towards process trees that are useful from an analysis point of view. We therefore define compactness of process trees, as follows:

Definition 3.11 (Compact process tree)**.** *Given an event log L, a process tree P is* compact*, if it satisfies all of the following conditions:*

1. *P does not exhibit the choice operator as a root node. If this condition is violated, the process tree would be the union of completely separate behaviors. While this may result in a frequent tree, the tree is arguably of little interest.*
2. *P does not result from a combination through a choice operator, where, given L, one of the seeds is frequent. This is motivated as follows: If a tree P_1 is frequent, combining it with any*

other tree P_2 through the choice operator results in a frequent tree. Yet, P_2 adds complexity by means of behavior that may not even appear in the log.

3. *P does not contain a loop operator $loop(P_1, P_2)$, such that only the behavior of P_1 appears in L. While having only the behavior of P_1 yields a valid trace of the respective process tree, the derivation of an operator $loop(P_1, P_2)$ is not meaningful, if L does not contain the behavior of P_2.*

Note that from condition (2), we immediately derive a **second pruning rule** for the exploration of candidate patterns: When performing a combination through the choice operator, both seeds must be infrequent.

In addition to compactness, there is a second property that is desired for behavioral patterns. It is motivated by the monotonicity property. The latter states that a frequent process tree P whose defining operator is a constraining operator must have two frequent seeds. Hence, we shall return solely P, as the seeds can simply be derived from P and are known to be frequent. In other words, we consider P to be the representative of its seeds. Furthermore, by transitivity, P is a representative of the paths in the construction tree composed solely of trees defined by constraining operators. As a consequence, discovery shall be limited to the largest representatives, which we call maximal behavioral patterns.

Definition 3.12 (Maximal process tree)**.** *Considering all behavioral patterns of at most depth i, a behavioral pattern is* maximal*, if it is frequent and not contained through a constraining operator in another frequent process tree of depth smaller or equal to i.*

In the example of Fig. 1.3, the trees *seq(BT, CO)* and *seq(BT, RB)* are frequent, but not maximal, since they are contained in *seq(BT,and(CO, RB))*. When *seq(BT,and(CO, RB))* is discovered, all the frequent trees it represents, such as *seq(BT, RB)*, can be deduced.

3.3.3 Optimization Based on Projections

We recall that we aim at the discovery of frequent process trees. The runtime complexity of a method to solve this problem is governed, among other factors, by the size of the construction graph, which increases exponentially when the number of activities increases, and by the size of the log used to evaluate the quality of the trees. To cope with the latter, we present an optimization that complements the two pruning rules introduced in Section 3.3.1 and Section 3.3.2. Our optimization uses projections to assess the frequency of a tree based on a small number of traces:

- When performing a combination through a constraining operator, the behavior associated with the resulting trees may only appear in the intersection of the projections of the seeds. As a result, quality metrics are calculated solely based on the said intersection. Moreover, the size of the intersection of the seeds projections represents an upper bound for the frequency of the resulting trees. This yields a **third pruning rule**: If the upper bound is less than the frequency threshold, the combination is not considered further.
- When performing a combination using the choice operator, the projection associated with the resulting trees is the union of the projections of the seeds. Moreover, the frequency

Algorithm 1: COBPAM: Function *addFreq*

input : P, a process tree; Γ, set of frequent process trees;
Θ, a set of frequent compact maximal process trees;
τ_S, a support threshold; τ_L, a precision threshold.

```
1  Γ' ← ∅;
2  for P' ∈ Γ do
3      Γ' ← combine P, P' through operators {and,loop,seq} with pruning rules;
4  for R ∈ Γ' do
5      if τ_S < support(R, L) (Definition 3.9) then
6          if τ_L < precision(R, L) (Definition 3.10) then
7              Θ ← Θ ∪ R;
8              Θ' ← trees on constraining operat. paths in construction tree of R;
9              Θ ← Θ \ Θ' ;
10         for each potential combination leaf a in R do
11             RW ← create representative word, replace a with '_' in RW(R);
12             Γ_RW ← set containing frequent trees identified by RW;
13             addFreq(R, Γ_RW, Θ, τ_S, τ_L);
14     else
15         for each potential combination leaf a in R do
16             RW ← create representative word, replace a with '_' in RW(R);
17             γ_RW ← set containing infrequent trees identified by RW;
18             addInfreq(R, γ_RW, Θ, τ_S, τ_L);
19 Γ ← Γ ∪ P;
```

of the resulting trees can be precisely derived and corresponds to the size of the union of the seeds projections. On another hand, the language of the new trees is the union of the languages of the seeds.

3.3.4 The COBPAM Algorithm

Now we are ready to present COBPAM, an algorithm that strives for efficient discovery of behavioral patterns that are frequent, compact and maximal. In order to achieve high efficiency, it largely neglects infrequent activities. More precisely, it discovers process trees that are built from frequent activities as well as frequent combinations of two infrequent activities through the choice operator. Here, a frequent combination of two infrequent activities is considered as a single activity in the remainder of the algorithm.

Note that pruning of infrequent trees, in general, implies a certain loss of patterns. Due to the choice operator, trees that are infrequent at some point can be combined to get frequent ones at a later stage. Hence, pruning infrequent trees potentially leads to missing some frequent patterns that comprise a choice operator. Despite this, COBPAM applies the respective pruning, since without it, a large number of infrequent process trees would need to be evaluated. Moreover, the loss of frequent behavioral patterns in the discovery process is limited to process trees that comprise the choice operator. Completeness of the discovery result for trees built of constraining operators is not affected.

Algorithm 2: COBPAM: Function *addInfreq*

input : P, a process tree; γ, set of infrequent process trees;
Θ, a set of frequent compact maximal process trees;
τ_S, a support threshold; τ_L, a precision threshold.

```
1  γ' ← ∅;
2  for P' ∈ γ do
3      γ' ← combine P and P' through choice operator;
4  for R ∈ γ' do
5      if τ_S < support(R, L) (Definition 3.9) then
6          if τ_L < precision(R, L) (Definition 3.10) then
7              Θ ← Θ ∪ R;
8          for each potential combination leaf a in R do
9              RW ← create representative word, replace a with '_' in RW(R);
10             Γ_RW ← set containing frequent trees identified by RW;
11             addFreq(R, Γ_RW, Θ, τ_S, τ_L);
12     else
13         for each potential combination leaf a in R do
14             RW ← create representative word, replace a with '_' in RW(R);
15             γ_RW ← set containing infrequent trees identified by RW;
16             addInfreq(R, γ_RW, Θ, τ_S, τ_L);
17 γ ← γ ∪ P;
```

The idea of the COBPAM algorithm is to incrementally build up sets of process trees. In the light of the pruning rules, we maintain two kinds of sets, containing only frequent and infrequent trees, respectively. The former kind serves as the basis for combinations through the constraining operators, whereas the latter serves for choice-based combinations. All trees inside a set are identical except for a single leaf node. In fact, any two trees in a set are seeds and can be combined. Moreover, since the difference between two seeds is a single leaf, we associate each set with an identifier in the form of a representative word that applies to any of the representative words of the contained trees. Take, for example, the tree *seq(BT, CO)*. Its representative word is *(BT CO seq)*. The process tree can be added to the set defined by *(BT _ seq)*, where the underscore is a placeholder for any activity. So, any other tree, e.g., *seq(BT, RB)*, can be added to the set by replacing the placeholder with an activity. The placeholder is always at the position of a potential combination leaf.

The algorithm revolves around two functions, *addFreq*, defined in Alg. 1, that adds the process tree P to a set Γ containing only frequent trees; and *addInfreq*, defined in Alg. 2, that adds P to a set γ containing only infrequent trees. By Θ, we further denote the set of frequent compact maximal trees, which represents the actual result of our algorithm. As such, the respective trees must satisfy a given precision threshold. Moreover, since the result shall contain only maximal trees, each time a frequent tree defined by a constraining operator is added to it, parts of its construction tree are deleted.

The COBPAM algorithm starts by creating the set of frequent process trees identified by the word '_', i.e., any frequent tree with a single activity is added to it. We apply function *addFreq* on

this set for each frequent activity. The algorithm then proceeds recursively, switching between *addFreq* and *addInfreq*. Les deux fonctions réalisent constamment des opérations de combinaison. Note that one may use a maximum recursion depth d, which then also limits the maximum depth for the discovered trees to force termination.

3.4 Conclusion

In this chapter, we presented COBPAM, a context-agnostic approach for the discovery of particular behavioral patterns that are meaningful and non redundant. We first introduced important notions revolving around the combination operation, a central element to our approach. Then, we discussed the metrics we use to evaluate the relevance of our patterns before turning to the algorithm itself. We also covered its different optimizations through the pruning of the search space and the use of projection rules. We finally discussed the wanted properties of maximality and compactness. In the following chapter, we include the data dimension to enrich our study of flexible processes.

4 Data-aware Analysis Framework

Contents

Our second contribution is a complete analysis framework that takes into account the data attributes in the event log. It extends COBPAM with the study of different types of dependencies in order to validate research goals **G2** and **G3**. In the first section of this chapter, Section 4.1, we discuss correlations between behavioral patterns themselves through the discovery of behavioral rules that express strong links between patterns. We then present in Section 4.2, CCOBPAM, a data-aware process discovery method that builds over COBPAM in order to discover patterns with relation to contexts. Finally, we turn to a study of causality between data and the occurrence of behavioral patterns in Section 4.3 before providing a general methodology for the usage of our framework in Section 4.4.

4.1 Behavioral Rules

A first perspective for the analysis of discovered patterns are correlations between the patterns themselves. Understanding the interplay between patterns, again, provides deeper insights into the conduct of the considered process.

In particular, we explore the interplay of behavioral patterns that are part of a frequent behavioral pattern, which includes a sequence or concurrence operator as the root node. For such a setting, we investigate the presence of correlations between child patterns.

First, consider a discovered pattern $P = seq(P_1, P_2)$ with P_1, P_2 being subtrees. For such a pattern, we investigate whether there is evidence for an association rule $P_1 \rightarrow P_2$. From the existence of P, we know that the behavior of P_2 appears after the behavior of $P1$ during process execution. Two cases are possible; either P_2 is strongly associated with P_1, meaning P_2 is observed always and only when P_1 is observed; or it is loosely associated, if that is not the case. We compute the odds ratio to evaluate the potential correlation which should be greater than one. For an association rule R $\rightarrow$ Q, the odds ratio is given as $(support(R \wedge Q) * support(\neg R \wedge \neg Q))/(support(\neg R \wedge Q) * support(R \wedge \neg Q))$ while the confidence interval's lower bound at 95% writes as in (4.1). If this latter value exceeds one then the association rule holds.

$$exp(log(oddsRatio) - 1.96 * \sqrt{\frac{1}{supp(R \wedge Q)} + \frac{1}{supp(\neg R \wedge \neg Q)} + \frac{1}{supp(\neg R \wedge Q)} + \frac{1}{supp(R \wedge \neg Q)}}) \tag{4.1}$$

Fig. 4.1 gives an example of a process configuration where there is no interplay (odds ratio is below one). It is to be noted that the computation of the odds ratio is done efficiently based on the results of the actual pattern discovery. Since the root of P is a sequence operator and due to monotonicity, $P1$ and $P2$ are also frequent and the set of traces in which they appear are already known.

The same procedure is applied to behavioral patterns that have the concurrency operator as a root node, $P = and(P_1, P_2)$. The sole difference is that the semantics of the concurrency operator have to be incorporated: positive evidence for a correlation is provided by traces, in which the behavior of P_1 appears in parallel with the one of P_2, and never without it. Also, if the behavior of P_1 is not observed in a trace, neither is the behavior of P_2.

4.2 Contextual COBPAM

This section presents our approach for taking into account contexts when discovering behavioral patterns. In order to obtain fine-granular insights into the behavior of a process, behavioral patterns can be associated to different contexts specified through conditions based on attributes that are attached to traces. This way, it is revealed whether a pattern is specific to a context, or materializes independently of any execution context.

4.2.1 Contexts

As a first step, we clarify the notion of a contextual event log. It includes attribute values that represent the context in which a case was recorded. It is to be noted that such attributes could be

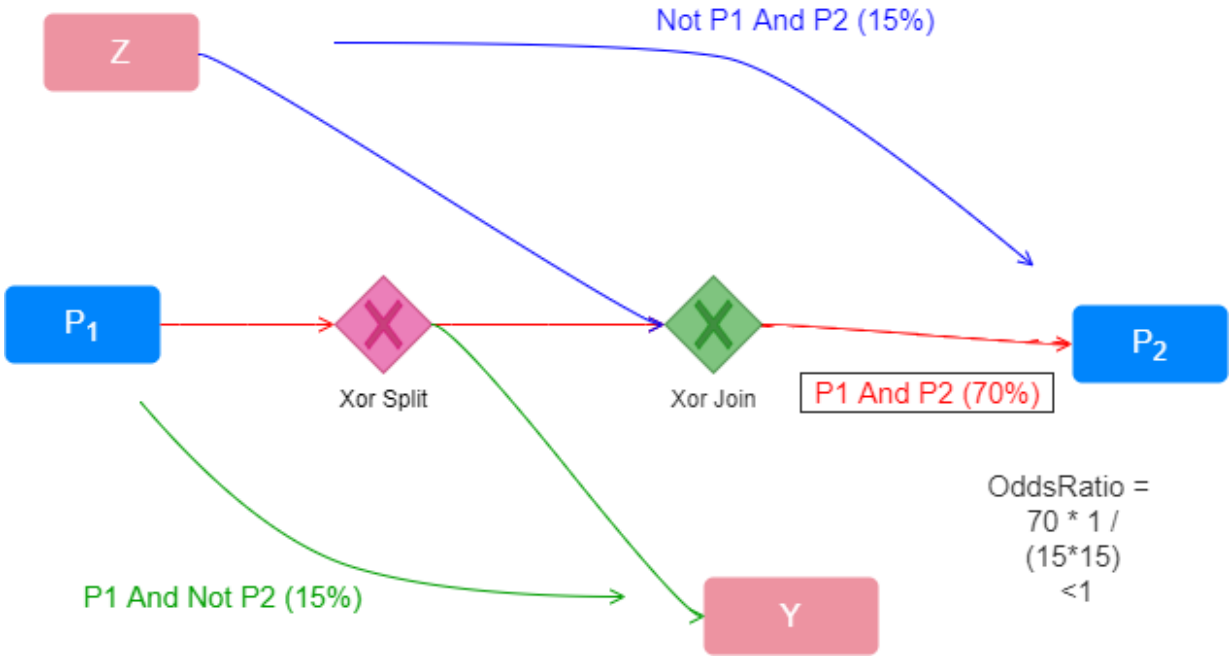


Figure 4.1: Example of a non-valid behavioral rule.

related to the intrinsic features concerning a case, e.g., the age group of the patient undergoing treatment, or could concern properties of the execution environment and the interdependencies between cases, e.g., how many patients checked in within the same time interval as the patient under consideration [70]. In our case, the possible attribute values are defined by a relation $\mathcal{R}(D_1, \ldots, D_n)$ with D_i being the domain of the i-th attribute. In the remainder, we write d_i for the name of the i-th attribute. A tuple of this relation is assigned to a case of the event log, as follows:

Definition 4.1 (Contextual event log)**.** *Given a simple event log L and a relation $\mathcal{R}(D_1, \ldots, D_n)$, such that the i-th attribute is a case attribute named $d_i \in ANC$ (the set of attribute names) and $\forall c \in L$, $\#_{d_i}(c) \neq \perp$. A contextual event log is then a pair (L, χ) where χ is a function that maps each case c of L to $\chi(c) = (v_1, \ldots, v_n)$, with $v_i \in D_i$, $v_i = \#_{d_i}(c)$, i.e, a tuple of $\mathcal{R}$.*

To define the notion of a context, we introduce D_i' as an extension of the domain D_i with a dedicated, unique symbol '$*$', which represents a wildcard. We capture this semantics by an inclusion order $\subset_{D_i'} = \{(v_i, *) \mid v_i \in D_i\}$, with $\subseteq_{D_i'} = \subset_{D_i'} \cup \{(v_i, v_i) \mid v_i \in D_i\}$ as its reflexive version, so that the wildcard symbol includes any value $v_i \in D_i$. The pair $(D_i', \subset_{D_i'})$ defines an inclusion hierarchy $\mathcal{H}(d_i)$ on data attribute d_i. An example for two such hierarchies of our initial example is given in Fig. 4.2.

A context is defined as a tuple $(v_1, \ldots, v_n)$ with $v_i \in D_i', \forall i \in \{1, 2, \ldots, n\}$. Contexts are organized through an inclusion order $\leq$, such that two contexts $C = (v_1, \ldots, v_n), C' = (v_1', \ldots, v_n')$ are ordered, denoted as $C \leq C'$, if $v_i \subseteq_{D_i'} v_i' \; \forall i \in \{1, 2, \ldots, n\}$. If $\exists 1 \leq i \leq n, v_i \subset_{D_i'} v_i'$, then context C' is said to be more general than C, while C is referred to as being more specific and we write $C < C'$ (strict inclusion order). A context $C = (v_1, \ldots, v_n)$ is atomic, if $v_i \in D_i, \forall i \in \{1, 2, \ldots, n\}$. We also

designate a decomposition of a context C as the non-empty set of atomic contexts that are more specific than C.

For illustration purposes, consider the example of Fig. 4.2. The context (*,<70) is more general than (low, <70). As stated earlier, '$*$' is a wildcard symbol that represents any value v_i; meaning the context (*, <70) represents all the population with the age being smaller than 70, independent of the income. Contexts are derived directly from the data attribute values assigned to the traces in a log, and their hierarchy is well-defined due to the inclusion order over these values.

Note that the granularity of the contexts definition is controlled by the size of the domains of the data attributes and their number. In particular, for continuous data attributes, discretization may be employed, which divides the domain into several intervals, as to avoid generating a big number of contexts. The *age* attribute of our example illustrates such a discretization by considering solely two age groups (<70 and 70+) instead of the actual age values.

4.2.2 Contextual Behavioral Patterns

Now that we introduced contexts, we link a context to a contextual event log. That is, we consider the set of traces for which the contextual information is contained in the context.

Definition 4.2 (Associated event log)**.** *Let (L,χ) be a contextual event log and $C = (co_1,\ldots,co_n)$ a context. The associated event log of context C, is a contextual event log (L',χ') with $L' \subseteq L$, such that case $c \in L$ is part of L' if $\chi(c) = (v_1,\ldots,v_n)$ and for all v_i it holds that $v_i \subseteq_{D_i'} co_i$; and χ' is the restriction of χ to L'.*

Moreover, we define the size of the context C with respect to (L,χ) as the size of the associated event log (L',χ').

A process tree is said to be **C-frequent** or frequent in C, if it is frequent in its associated log. In fact, we distinguish two types of behavioral patterns for a non-atomic context:

- **C-general behavioral patterns:** The patterns are frequent in C and in every descendant of C. Applied to our running example (see Fig. 1.4), these would be patterns answering questions such as what patterns are frequent in the low income population whatever their age?

- **C-exclusive behavioral patterns:** The patterns are C-frequent only in C and its descendants. In our example, these would be patterns answering the question what behavioral patterns are exclusively present for patients that are older than 70, whatever their level of income?

If a pattern is C-frequent in at least an atomic context and not C-General or C-exclusive in any context, it is called **AC-frequent**. If a pattern is C-frequent in the most general context, i.e., the associated event log is the original event log, it is called **log-frequent**.

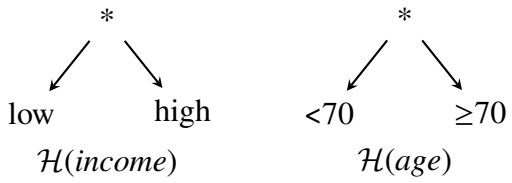


Figure 4.2: Hierarchies on data attributes *income* and *age*

4.2.3 Contextual Pattern Discovery Approach

Based on the notions of context and contextual behavioral patterns, we are ready to define the problem of discovering these patterns from a given event log. We first formulate the problem explicitly, before we discuss how to tackle it.

Problem 4.3 (Contextual Behavioral Pattern Discovery)**.** Given a contextual event log, (L, χ), the problem of *contextual behavioral pattern discovery* is to find the set Φ of C-general and C-exclusive behavioral patterns in each context of the hierarchy induced by the mapping function χ. Φ contains the C-frequent patterns in the atomic contexts as well.

The objective here is to exploit the data dimensions present in the event log, which have been ignored by previous methods for behavioral pattern discovery. To address the above problem, we introduce Contextual COBPAM (short CCOBPAM), as an adaptation of the COBPAM algorithm. Before defining the algorithm, we motivate the underlying design choices. Adopting the reasoning presented for sequential pattern mining with contexts [63], we observe the following: A behavioral pattern P is C-general, if and only if, it is frequent in the atomic contexts in the decomposition of C. In the same vein, a pattern P is C-exclusive, if and only if, it is frequent in the atomic contexts in the decomposition of C and not frequent in any other atomic context. Consequently, discovery of the two types of patterns, C-general and C-exclusive, in all contexts shall start with the discovery of behavioral patterns in atomic contexts.

The CCOBPAM algorithm takes as input the data attributes d_i that shall be considered for the definition of contexts, and a contextual event log. It returns a set of discovered behavioral patterns Φ, and three functions *atomLoc*, *genLoc*, and *excLoc*. These functions map each behavioral pattern P to, respectively, the atomic contexts, in which it is C-frequent; the contexts, in which it is C-general; and the contexts, in which it is C-exclusive.

CCOBPAM is based on two functions executed sequentially. The first one, *atomMine*, defined in Alg. 3, extracts the behavioral patterns from the atomic contexts by applying COBPAM. It returns the set of contextual behavioral patterns and, for each contextual pattern, the set of atomic contexts in which it is frequent (through *atomLoc*). Note that ACOBPAM could be applied here. In case the precision of the contextual behavioral patterns is important, COBPAM is recommended because as we will see in Chapter 5, the precision metric has been deleted in ACOBPAM.

A contextual behavioral pattern could be the one in Fig. 1.4c, noted *Pex*. It is discovered in both contexts [low, <70] and [low, 70+] resulting in *atomLoc*(*Pex*) = {[low, <70], [low, 70+]}

Algorithm 3: CCOBPAM: Function *atomMine*

input : AC, the set of atomic contexts;
τ_S, a support threshold; τ_L, a precision threshold;
τ_D, a depth threshold.
output Φ, set of behavioral patterns;
:
atomLoc.

```
1 for C ∈ AC do
2     Γ ← COBPAM(C, τ_S, τ_L, τ_D), derive maximal compact C-frequent behavioral patterns;
3     for P ∈ Γ do
4         add P to Φ;
5         add C to atomLoc(P);
```

Algorithm 4: CCOBPAM: Function *nonAtomMine*

input : NC, the set of non atomic contexts;
Φ, set of behavioral patterns.
output *genLoc*; *excLoc*.
:

```
1 for P ∈ Φ do
2     for C ∈ NC do
3         if the decomposition of C is contained strictly in atomLoc(P) then
4             add C to genLoc(P)
5         else if the decomposition of C equals atomLoc(P) then
6             set excLoc(P) to C
```

A second function, *nonAtomMine*, defined in Alg. 4, iterates over the non-atomic contexts to discover the C-general and C-exclusive patterns thus returning *genLoc* and *excLoc*. We recall that a pattern P is C-general in a context C if P is C-frequent in each context of the decomposition of C and is C-exclusive if it is only frequent in that decomposition. Since *atomLoc(P)* points to the atomic contexts where it is frequent, then a C-general pattern in C is one such that the decomposition of C is part of *atomLoc*(P) and a C-exclusive pattern in C is one such that the decomposition of C is equal to *atomLoc(P)*. Coming back to our running example, when executing *nonAtomMine*, for *Pex*, we iterate over the non-atomic contexts. Encountering [low, *], we realize that *atomLoc*(*Pex*) is equal to the decomposition of [low, *]. As such, *Pex* is considered C-exclusive in [low, *] and *excLoc* will be set to [low, *].

The support and precision of any pattern on a non atomic context NC is directly inferred from the aggregation of the frequencies and language seen in the decomposition of NC. Particularly, the support of a pattern in context NC is the sum of its frequencies in the atomic contexts from the decomposition of NC over the size of NC. The precision of a pattern in NC is the size of the union of the words seen in the atomic contexts of the decomposition of NC over the size of its language.

4.3 Causal Relationship between Data and Occurrences of Behavioral Patterns

C-exclusive patterns are frequent solely in some context C and its descendants. This suggests that the frequency of the patterns is independent of the unconstrained attributes (i.e., set to '$*$') and is, in fact, correlated to the constrained ones. Knowing this, in order to interpret a pattern, it is useful to assess whether the traces actually suggest a causal relation between the context C and the occurrence of the pattern. Below, we limit ourselves to such causal relations for contexts that define a specific value for one of the attributes, while leaving the other attributes unconstrained. This restriction is motivated by the potential existence of causal relations between multiple constrained attributes, which would compromise the analysis. Specifically, if context C assigns a specific value v to some attribute d, the question is whether the occurrence of pattern P that is frequent in just context C and its descendants, is caused by the value v. The derivation of such a causal relation helps to interpret the discovered pattern and provides further insights into how the context influences the execution of the process.

We approach the analysis of causal relations between the context and a behavioral pattern by adopting the idea of a cohort study, as known in the medical domain. We recall that it aims to assess the impact of a risk factor, called the exposure variable, on an outcome variable. The procedure follows two groups with common characteristics apart from the risk factor. The group with the risk factor is the exposure group, the other one the non-exposure group. The common characteristics must be evenly distributed among the two groups and serve as controlled variables. A study may be a perspective study, if the groups are followed until the outcome appears, or a retrospective study, if it is conducted after the outcome has been observed as in our case.

In our context, a C-exclusive pattern P induces a certain correlation. With two variables for each trace, one indicating whether attribute d is set to value v and one indicating whether the behavior of P is exhibited, we may formalize the dependency as an association rule linking these variables. As a next step, we are interested in the presence of a causal association rule between the variables, which we assess following common procedures for cohort studies [51]. Causality is a stronger notion than correlation which states that a change in the exposure variable provokes a change in the outcome. However, for any point in time, concerning an individual in a population, we cannot observe the outcome in the presence *and* in the absence of the risk factor. The observed event is called factual and the hidden one is called counterfactual. To prove causality, the outcome should appear in the presence of the exposure and disappear otherwise (either if the presence of the exposure is the factual or the counterfactual). A cohort study works with observational data. For each data point in the exposure group, meaning an individual with some characteristics (controlled variables) where the risk factor is present, we simulate the counterfactual by choosing another data point with exactly the same characteristics in the non-exposure group. This concordance of the characteristics is, of course, an assumption of the method as some variables can be unrecorded. On another hand, if the outcome is clearly independent of some variable, there is no use to it as its value has no impact on the observed outcome. We can, in fact, use an individual with a different value for that variable as counterfactual because the outcome will not change.

In the following, we develop the method based on the above principles. A running example will illustrate each step. We suppose the existence of three attributes, d, a, b, in the contextual event log with two modalities for d (v and v') and three for each of a and b, $a_i, b_i, \forall 1 \leq i \leq 3$. The following steps ensue:

(1) We transform the relation $\mathcal{R}(D_1, D_2, D_3)$ that captures possible contexts in terms of attribute value combinations into a relation of Boolean variables $\mathcal{B}(B_1, \ldots, B_8)$. Here, the modalities of each attribute d, a, b are transformed into a set of Boolean indicator variables, od set to true (resp. od') if $d = v$ (resp. $d = v'$) and oa_i (resp. ob_i) set to true $\forall 1 \leq i \leq 3$ if $a = a_i$ (resp. $b = b_i$).
(2) The Boolean variable od that represents value v of attribute d is defined as the exposure variable.
(3) We define a Boolean outcome variable t per case c and pattern P that is equal to $\epsilon(c, P)$, i.e., it is true if the pattern is part of the trace, and false otherwise.
(4) Next, we identify among the attributes present in the contextual log, the ones that are correlated with the outcome variable, t and thus possible causal factors. These variables will serve as controlled variables. The reason is that we want to assess if, variable od being set to true, causes variable t being set to true among other possible causal factors. To this end, we apply the odds ratio defined previously as a measure of correlation. For our example, we suppose that only oa_1, oa_2, oa_3, ob_1, ob_2 hold association rules with od.
(5) The event log is divided into an exposure group (cases where od is true) and non-exposure group (remaining cases). The groups are then filtered to ensure that the controlled variables are evenly distributed in both groups, in order to mitigate their effect. As it can be seen in Fig. 4.3, cases 6 and 11 were filtered out because both of their controlled variables could not be found in the opposite groups. By eliminating them, we ensure an equal distribution of the controlled variables values between the groups.
(6) For each existing combination of values of the controlled variables, we assess the value of the outcome variable. Cases with a positive outcome in the exposure group and a negative one in the non-exposure group provide evidence for a causal relation (let their number be n_1). Cases with a negative outcome in the exposure group and a positive one in the non-exposure group, in turn, provide evidence against a causal relation (their number is noted n_2). If the ratio of the number of cases providing evidence for and those providing evidence against a causal relation, $\frac{n_1}{n_2}$, is larger than one, we conclude on the existence of the causal relation. The lower bound of the confidence interval of the latter ratio is given by $exp(log(\frac{n_1}{n_2}) - 1.96 * \sqrt{\frac{1}{n_1} + \frac{1}{n_2}})$. A value higher than 1 confirms the causal relationship [51].

Following this procedure, we are able to identify whether the context information of attribute d (or od, respectively) can indeed be seen as the cause of the occurrence of pattern P.

Exposure group

Case	od	oa_1	oa_2	oa_3	ob_1	ob_2	t
1	1	1	0	0	0	0	1
3	1	0	0	1	0	0	1
4	1	0	1	0	0	0	0
12	1	0	1	0	1	0	0
6	1	0	0	1	1	0	0

Non-exposure group

Case	od	oa_1	oa_2	oa_3	ob_1	ob_2	t
9	0	1	0	0	0	0	0
2	0	0	0	1	0	0	1
8	0	0	1	0	0	0	1
5	0	1	1	0	1	0	0
11	0	0	1	0	0	1	0

Figure 4.3: Exposure and non exposure groups construction.

4.4 Methodology for Using the Data-aware Framework

Finally, we integrate the presented approach for the discovery of behavioral patterns with the procedures for the analysis of discovered patterns in a general methodology, as follows:

(1) To get a first overview of behavioral regularities, context-agnostic behavioral patterns are identified with the COBPAM algorithm (or ACOBPAM), given a configuration of thresholds for the support and precision.
(2) Next, contextual attributes are selected, paying attention to their semantics and modalities (continuous or discrete). If needed, pre-processing is applied to adapt, normalize, or discretize the attribute values.
(3) A minimal context size is defined, in relation to the size of the log and, potentially, knowledge about the process under investigation. Contexts that don't meet the minimal size are discarded from the analysis and their associated traces deemed not representative enough of the context.
(4) Contextual behavioral patterns are mined with the CCOBPAM algorithm.
(5) Follow the interpretation guidelines, detailed below, to derive insights on the process.
(6) If suggested by these guidelines, adapt the support threshold, repeat the discovery of behavioral patterns, and note potential changes in the set of patterns.
(7) Finally, the presence of behavioral rules, then of causal relations with context data for C-exclusive patterns are evaluated according to the aforementioned procedures.

To complete the above methodology, we provide guidelines for the interpretation of discovered contextual behavioral patterns. CCOBPAM reveals patterns with respect to a hierarchy of contexts and considers different types of frequency that can be interpreted as follows:

- *AC-frequent*: The pattern is frequent in at least an atomic context and not C-General or C-exclusive somewhere else.
- *C-exclusive*: The pattern is exclusively frequent in the context and all its descendants, meaning that its occurrence is independent of the populations considered inside the context.
- *C-general*: The same interpretation as for a C-exclusive pattern applies. Yet, the pattern is also frequent somewhere else in the context hierarchy.

Moreover, patterns that are discovered by context-agnostic discovery with COBPAM *and* context-aware discovery with CCOBPAM are particularly interesting to derive insights on the process. In the following, we characterize the possible cases and provide an interpretation:

- *A log-frequent pattern appears as a C-exclusive pattern in the root context:* The pattern is not only frequent in the whole log, but also frequent in every context. This means that the pattern is strongly frequent, independent of the considered population.
- *A log-frequent pattern appears as C-exclusive or C-general in large contexts or C-frequent in large atomic contexts:* The pattern is actually frequent in some parts of the log that represent a significant portion of the log, which renders the pattern log-frequent. Yet, it is infrequent in other contexts. As such, the occurrence of a pattern that occurs very often (i.e., it is log-frequent) can be linked rather accurately to specific contexts.
- *A log-frequent pattern appears as C-exclusive or C-General in only some small contexts or as C-frequent in some small atomic contexts:* The pattern is frequent in some parts of the log and close to frequent in other parts. This may point to a need to revisit the chosen support threshold.
- *A pattern is infrequent when the context is neglected, but is frequent under some context:* The identified behavior is frequent, but applies solely to specific contexts, showing the relevance of a contextual analysis.

4.5 Conclusion

This chapter was dedicated to the Data-aware Analysis Framework. We started by discussing the behavioral rules which were inspired by Sequential Rule Mining. We then presented our model for contexts definition and the different types of frequency we considered such as C-generality and C-exclusivity. We used interesting theorems to devise the search algorithm and continued with a causality study. Finally, the exhaustive methodology for the analysis is detailed. In the next chapter, we went back to the context-agnostic setting and set out to further improve execution times and to maximize the insights extracted.

5 Advanced Behavioral Pattern Mining

Contents

In this chapter, we look into Advanced COBPAM (ACOBPAM), an upgrade of the initial COBPAM algorithm that aims at bringing an improvement on three aspects: the runtime, the number of returned patterns and finally the analysis difficulty, thereby further satisfying the research goal **G1**. It also provides additional dependencies between patterns helping in realizing research goal **G2**. One section is dedicated to each facet of the improvements brought by ACOBPAM. Section 5.1 presents the alignment growth procedure which we applied to significantly reduce runtimes. It relies on using past established alignments in computing new ones. We recall that alignments allow to compute the quality metrics. Then in Section 5.2, we introduce a post-processing step to reduce the number of returned trees further eliminating irrelevancy. Finally, in Section 5.3, we devise a graph that allows for an interactive and intuitive visualization with the most relevant patterns discovered.

5.1 Alignment Growth

The COBPAM algorithm browses the construction graph while outputting frequent trees. It chooses paths in an intelligent manner by applying the pruning and projection rules. Also, seeds are combined to get more complex trees. The information on the frequency of the complex trees are aggregated from the frequencies of those seeds. The same with the set of traces that are susceptible to welcome the trees resulting from the combinations. This particular set is determined from the seeds' projections. However, each time a tree is evaluated and its frequency assessed, the whole tree is aligned using the A* algorithm with the complete trace. Advanced COBPAM applies the same reasoning on frequencies and projections mentioned above on alignments. Practically, we will use the already computed seeds' alignments to extract the one of the tree at hand. We call this procedure the alignment growth.

Given a process tree P, a case c and its associated trace, $\hat{c}$, the alignment growth recursively accomplishes two tasks. First, it detects which part of the behavior of the tree is already present in the trace, named validated context. This is the behavior previously aligned in the children of P. The second task is to decide, if needed, to re-align other parts of the tree. The re-alignment is applied on a version of the trace stripped of the validated context. In the end, we are aligning smaller trees on smaller traces taking advantage of the older alignments. The execution times are thus substantially reduced.

In the following, we will introduce necessary definitions, an interesting property on which the alignment growth is built and then detail the growth procedure. Finally, we dedicate a section to the concessions we made and the limits of our new alignment algorithm while divulging the intuitions behind it.

5.1.1 Definitions

Definition 5.1 (Shadow Map)**.** *Given a sequence of events $\sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_n \rangle$, a case c and its associated trace $\hat{c}$ where $\sigma \leq \hat{c}$, we call the shadow map of σ on $\hat{c}$ the increasing mapping function $s : \{1, 2, \ldots, n\} \to \{1, 2, \ldots, |\hat{c}|\}$ where $\forall i, j \in \mathbb{N}, 1 \leq i \leq n, 1 \leq j \leq |\hat{c}|,\ s(i) = j$ if and only if $\sigma_i = \hat{c}(j)$. We may also write $s(\sigma_i)$ to refer to $s(i)$ when σ_i is unique in the trace. Moreover, we call $s(i)$ the shadow of σ_i.*

During the execution of Advanced COBPAM, a shadow map of each discovered behavior $\upsilon(\hat{c}, P)$ for each case in $proj(P)$ is stored. This shadow map is also called for simplification the shadow map of P on $\hat{c}$. Note that when we mention $\upsilon(\hat{c}, P)$, we automatically imply that the pattern appears in a trace. In other words, $\epsilon(\hat{c}, P) = 1$.

Definition 5.2 (Behavioral Context)**.** *A behavioral context of a pattern P is a subsequence of a word of the language of P. Meaning, σ is a behavioral context of P if and only if $\sigma \leq w$ where $w \in \Sigma(P)$. This subsequence is named "context" because it could be interleaved or concatenated with another sequence to generate the word w. In fact, σ serves as a context in which new activities are incorporated.*

Definition 5.3 (Validated Context)**.** *Given a process tree P, a case c and its associated trace $\hat{c}$, a validated context $V(\hat{c}, P)$ is a behavioral context of P whose existence in $\hat{c}$ can be asserted without recourse to the classical alignment of P. We can give the validated context by its shadow map on $\hat{c}$. Besides, if the context is clear, we can write $V(P)$.*

Definition 5.4 (Boundary of a Pattern)**.** *The boundary of a pattern P contained in a trace $\hat{c}$ associated to a case c, symbolized by $\beta(\hat{c}, P)$ is a couple $(s(1), s(|\upsilon(\hat{c}, P)|))$ where s is the shadow map of P on $\hat{c}$. The lowest index $\beta_l(\hat{c}, P)$ is called lower boundary and the highest $\beta_h(\hat{c}, P)$, the highest boundary. If the context is clear, we may write $\beta_l(P)$, $\beta_h(P)$ and $\beta(P)$.*

Definition 5.5 (The Loop Block)**.** *For a given leaf* a *for a process tree P, the loop block of* a *in P, called $L(a, P)$, is a subtree of P whose root, noted* r*, is an ancestor operator of* a*.* r *is a loop operator and no ancestor of* r *is a loop operator. When no ancestor of* a *is a loop operator or when P is a leaf, $L(a, P) = a$.*

5.1.2 Leftmost Occurrence First (LOF Property)

Essentially, the job of the alignment algorithm is to find a word of the language of the process tree with which it is possible to construct a shadow map for the current trace. While doing so, the alignment growth algorithm guarantees a certain property. If several shadow maps are applicable for the same word of the language of the process tree, it chooses the one with the lowest indexes for each activity. We can formally describe it as:

Property 5.6. *Let a process tree P, a case* c *and its associated trace $\hat{c}$. We suppose the process tree P appears in $\hat{c}$ in the form of the word $w = \upsilon(P, \hat{c})$. The shadow map* s *returned by the algorithm satisfies:if $s(i-1) = j'$, $s(i) = j$, $\forall 2 \leq i \leq |w|$ then $\nexists j' < k < j$ such that $w(i) = \hat{c}(k)$ and $\nexists 1 \leq k < s(1)$ such that $w(1) = \hat{c}(k)$.*

This property, coined, the LOF property, is verified thanks to the workings of the classical alignment and is preserved throughout the alignment growth. During the classical alignment, when reading the trace from left to right, if an activity is susceptible to contribute to the construction of $\upsilon(P, \hat{c})$, it is aggregated right away and the reading continues. The property can be formulated differently: the classical alignment algorithm and a fortiori the new one find the first occurrence of $\upsilon(P, \hat{c})$ in the trace. For example, in the trace $\langle a, b, c, b, a\rangle$, the occurrence discovered for $seq(a, and(b, c))$ is $\langle \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{c}, b, a\rangle$ even though $\langle \boldsymbol{a}, b, \boldsymbol{c}, \boldsymbol{b}, a\rangle$ is a valid occurrence too. Thus, when aligning a sequence behavior, if the precedent: a is already aligned, we can search for the consequent directly after it since that's the only way for the behavior to be validated.

5.1.3 The Growth Procedure

First we need to precise that the trees containing a choice operator are aligned classically. The reason will be specified later. An exception is the trees combined through the *xor* operator whose frequency can be directly inferred (see Section 3.3.3). For the other operators, having at disposal a new pattern P constructed out of the combination of two frequent seeds P_1 and P_2 (according to the monotonicity property), the objective is to determine for each trace, the validated context and

the behaviors to realign. Of course, in the case where P is a leaf, a simple classical alignment is applied (in reality, it is a simple reading of the trace from left to right until the leaf is found). If not, we set a as the combination leaf in P_1 and b, the combination leaf in P_2.

Let c a case and $\hat{c}$ its associated trace and s_1 (resp. s_2) the shadow map of P_1 on $\hat{c}$ (resp. P_2). We execute the following algorithm initialized with $P' = P$ and $\hat{c}' = \hat{c}$. We bring attention on the fact that the alignment issued from this algorithm respects the LOF property which is proved recursively. Moreover, any alignment of P' is executed on $\hat{c}' \uparrow \mathcal{A}(P')$ since the other activities can't be in the alignment result anyway. For simplification, this is implied when not mentioned.

The objective is to align a tree P' using the growth method on a trace tail $\hat{c}'$. In other terms, we are searching for a validated context while replaying some parts of P'. We suppose P' is constructed out of the combination of two seeds P'_1 and P'_2 where a and b are the respective combination leaves. P'_1 and P'_2 have thus already been aligned. We face the following cases:

Case 1: if P'_1 is not a leaf and P' writes as $seq(Q', Q'')$ with Q', Q'' two subtrees. Then, we will enumerate two cases:

Case 1.1: if $L(a, P'_1)$ is contained in Q'', then according to the workings of the combination operation P'_1 writes as, $P'_1 = seq(Q', Q''_1)$ and $P'_2 = seq(Q', Q''_2)$ with Q''_1, Q''_2, the two seeds of Q''. As such Q' was already aligned on $\hat{c}'$ while respecting the LOF property: the occurrence computed is the leftmost one. This implies that Q' has the same shadow map in both seeds. Consequently, all we need to do is to align Q'' using the alignment growth algorithm after Q', meaning on $tl^{\beta_h(Q')+1}(\hat{c})$. Indeed, since the occurrence of Q' considered is the leftmost one, any occurrence of Q'' that validates $seq(Q', Q'')$ comes after Q'. Moreover, since both alignments respect the LOF property then the whole alignment of P' respects it too. It is to be noted that the shadow map of Q' on $\hat{c}'$ is part of the validated context of P' and that the seeds of Q'', Q''_1 and Q''_2, were aligned on $tl^{\beta_h(Q')+1}(\hat{c})$.

Finally, We set P' to Q'' and $\hat{c}'$ to $tl^{\beta_h(Q')+1}(\hat{c})$ and repeat the alignment growth algorithm.

Case 1.2: if $L(a, P'_1)$ is contained in Q', then, according to the combination operation, $P'_1 = seq(Q'_1, Q'')$ and $P'_2 = seq(Q'_2, Q'')$ with Q'_1, Q'_2 the seeds of Q'. This means that Q'' was aligned in both seeds. However, they don't necessarily share the same shadow map, as in P'_1, Q'' was aligned after Q'_1 and in P'_2 after Q'_2. We will note β^1 the boundary of the appearance of Q'' during the alignment of P'_1 and β^2 that of its appearance during the alignment of P'_2. On the same subject, while it is true that Q'' is aligned in $\hat{c}'$, we have no guarantee that Q' exists in $\hat{c}'$ before the already calculated appearances of Q'' (either before β^1_l or β^2_l). So we have to align Q' using the growth algorithm on $\hat{c}'$ (we set P' to Q' and repeat the growth procedure. $\hat{c}'$ stays unchanged). Once that done, we check if $\beta_h(Q') < min(\beta^1_l, \beta^2_l)$ and if not, we check $\beta_h(Q') < max(\beta^1_l, \beta^2_l)$. If the first condition is met, then the leftmost occurrence of Q'' calculated in the seeds is the one aggregated into the validated context. Else, it is the second leftmost occurrence calculated in the seeds. That is because in those cases, Q' still precedes Q'' in $\hat{c}'$. If neither conditions are met, we realign classically Q'' on the trace $tl^{\beta_h(Q')+1}(\hat{c})$. On another note, since both the alignment of Q' and Q'' adhere to the LOF property, the whole alignment does.

Case 2: if P'_1 is not a leaf and P' writes as $P' = and(Q', Q'')$ with Q', Q'' two subtrees, we test if Q'' contains $L(a, P'_1)$. In that case, we swap Q' and Q'' since the operator is symmetric and that has neither an impact on the language of the tree nor on the alignment. As a result, we have $P'_1 = and(Q'_1, Q'')$ and $P'_2 = and(Q'_2, Q'')$ with Q'_1, Q'_2, the two seeds of Q'. Since the seeds were already aligned then Q'' was too. Besides, when aligning P', there is no order constraint between Q'' and Q'. As such, the already calculated alignment of Q'' serves as a validated context for P'. The next step is to align Q' using the growth algorithm on $\hat{c}'$. The trace we align on doesn't change because of the absence of an order enforcement. So we set only P' to Q'. Finally, the alignment of P' respects the LOF property because both the alignments of Q' and Q'' do.

Case 3: P'_1 is not a leaf and P' is $L(a, P'_1)$. In this case, a classical alignment of P' is required. There is no validated context $V(P)$ and the LOF property is respected thanks to the classical algorithm.

Case 4: $P' = seq(a, b)$. We know that a and b have already been aligned. So, we test if $\beta_l(b) > \beta_h(a)$. In that case, no classical alignment is needed and the shadow maps of a and b are part of the validated context. Else, b is realigned classically after a. In other words, b is re-aligned on $tl^{\beta_h(a)+1}(\hat{c})$. In both cases, the LOF property is respected because we use the classical procedure right after a leftmost first occurrence and/or use already calculated alignments.

Case 5: $P' = and(a, b)$. We know that a and b are aligned. Since there is no order constraint between them then P' is aligned too. No classical alignment is needed and the shadow maps of both a and b are included in $V(P)$. In this particular case, the whole alignment of P is directly constructed and $\upsilon(\hat{c}, P)$ is a validated context in its whole. It can be proven straightforwardly by backtracking. Here too, the LOF property is satisfied.

Case 6: $P' = loop(a, b)$. There is no validated context and P' is aligned classically satisfying thus the LOF property.

A hidden feature that is not directly visible is that when excluding the validated context from the alignment, we end up excluding its alphabet from the trace. As we mentioned earlier, the realignment of any behavior is realized on a version of the trace containing only the relevant activities. So, in addition to truncating the trace and using only its tail, the number of activities to align on is reduced. The classical alignment on the other hand proceeds using the alphabet of the entire tree which is less than optimal.

5.1.4 Changes, Limits and Intuitions

Compactness

Our first modification was to further precise the compactness property. A fourth condition was added:

- For a tree P constructed out of the combination of two seeds P_1, P_2, with the respective combination leaves being a and b, if the defining operator is a concurrence operator, then both the behaviors $\langle a, b\rangle$ and $\langle b, a\rangle$ have to appear in the log. Indeed, in the previous version

of COBPAM, there were cases where only one of the orderings appeared, say $\langle a, b\rangle$, and the tree with the concurrence was still considered frequent alongside the tree with the sequence, $seq(a, b)$. Yet, the *and* operator indicates the absence of an order constraint between the two activities and that can only be deduced if both orderings appear in the log.

Loop Language

In this section, we give more information about the loop alignment. In fact, we changed the way we search for loops in the log. Actually, if multiple iterations of a loop exist in a trace $\hat{c}$ associated to a case c, the behavior discovered $\upsilon(\hat{c}, P)$ is shortened to one iteration. For example, for the pattern *loop(a,b)* and the trace $\sigma = \langle a, b, e, a, d, b, a, c, d, a, f, g, h, e, k\rangle$, the behavior discovered is $\langle \mathbf{a}, \mathbf{b}, e, \mathbf{a}, d, b, a, c, d, a, f, g, h, e, k\rangle$ whereas COBPAM discovers $\langle \mathbf{a}, \mathbf{b}, e, \mathbf{a}, d, \mathbf{b}, \mathbf{a}, c, d, a, f, g, h, e, k\rangle$. While this has no impact on the frequency of the patterns and the ϵ function is still equal to 1, it influences the accuracy of the discovery. Indeed, using the alignment growth algorithm, we cannot detect the exact behavior of the pattern that materializes in the trace.

Considering this change, the metric *precision* loses its meaning. Since we do not precisely detect the word of the language of the pattern that appears in the trace, we cannot assert how strongly a pattern is contained in the log. Consequently, we no longer compute this metric nor do we use a threshold on it when returning trees. Moreover, as we described in Chapter 4, the n parameter when defining the *n_language* of a tree allowed us to indicate at which moment a loop in the traces is considered a loop in the model. For example, if we chose $n = 5$, then the loops in the traces should appear in a range of one to five iterations to validate the precision metric. Meaning, the returned trees incorporate a model loop only if a maximum of different numbers of iterations from one to five is observed. The idea is, if different numbers of iterations are observed, then there really is a task that can be repeated x times according to some decision; the higher n is, the higher the probability of such looping. Now, the change we made in ACOBPAM removes this degree of control on the semantics of the loops. All loops appear in the same way as one iteration. In other words, if $\langle a, b\rangle$ appeared in the log and in a consistent manner, a followed as in $\langle a, b, a\rangle$, it will be considered a loop even though in the model there is no looping; meaning, there is no possibility to add new iterations. It is just a simple and systematic re-execution of a after $\langle a, b\rangle$. The risk of returning patterns that are loosely represented in the log , due to the absence of the second metric, is the only limit to the new algorithm and we made this concession because, often times, it is more important to find frequent patterns than to find out how they appear exactly. Just as it is more interesting to have false positives than to miss out on some patterns or not being able to discovering them at all. ACOBPAM allows us to make the exploration possible thanks to feasible and, order of magnitude reduced, runtimes.

Let's now justify this one iteration limitation. We suppose we didn't apply it and consider the aforementioned trace σ. We introduce the following seeds: *seq(loop(a, b), c)*, *seq(loop(a,b), d)* which have already been aligned on σ. Following the alignment growth procedure, P_1, P_2 appear respectively as: $\langle \mathbf{a}, \mathbf{b}, e, \mathbf{a}, d, \mathbf{b}, \mathbf{a}, \mathbf{c}, d, a, f, g, h, e, k\rangle$ and $\langle \mathbf{a}, \mathbf{b}, e, \mathbf{a}, d, \mathbf{b}, \mathbf{a}, c, \mathbf{d}, a, f, g, h, e, k\rangle$. Indeed, the loop in these two trees were aligned classically and in its working, the alignment

finds the longest word of the language of the tree that exists in the trace, i.e. the appearance that yields the less deviations with respect to the model, see optimal alignment in Section 2.1.3. Next, we combine P_1 and P_2 to generate the following pattern: $P = seq(loop(a, b), seq(d, c))$. When attempting to align P, the subtree $loop(a, b)$ will be considered a validated context since it was already aligned (**Case 1.2**). d is also a validated context (**Case 4**), so the algorithm searches for c right after d: $\langle \mathbf{a}, \mathbf{b}, e, \mathbf{a}, d, \mathbf{b}, \mathbf{a}, c, \mathbf{d}, \overrightarrow{a, f, g, h, e, k} \rangle$. As there is no occurrence of c after d, then $\epsilon(\sigma, P) = 0$ and the pattern is deemed non existent in the trace, which is false. Indeed, if we loop only once, we get: $\langle \mathbf{a}, \mathbf{b}, e, \mathbf{a}, \mathbf{d}, b, a, \mathbf{c}, d, a, f, g, h, e, k \rangle$. The problem is, when the loop appears before the new behavior that needs to be classically aligned, it is considered a validated context. In fact, the loop itself is a validated context, but the form it takes is not. The number of iterations may change to adapt to the new behavior added since we are searching for an appearance of the process and that its appearance has many variants. In a sense, here, the alignment of the loop doesn't respect the spirit of the LOF property. We recall that the property allows us to search for a consequent without looking back again at the precedent.

In order to cater for this problem, two solutions are possible: the first is to realign the precedent if it contains a loop when encountering **Case 1.2**. The other is to make sure the validated context is the shortest possible, so that the consequent has all the space to appear after it. Hence, the choice to restrict the loops to one iteration only since it is the minimum for a loop behavior and we also don't have to recompute the actual iterations that can make space for the consequent behavior. We adopted the latter option because the main goal of ACOBPAM was to reduce runtimes and the validated contexts are the way to do that, especially considering how time consuming it is to align loops (see Section 5.1.4). Also, we think the drawback is minor since as we mentioned above, the change has no impact on the frequency of the patterns.

The Xor Case

The *xor* operator puts us in a position similar to that of the previous section. Indeed, let $P_1 = seq(xor(a, seq(b, c)), d)$, $P_2 = seq(xor(a, seq(b, c)), e)$ and the trace $\sigma = \langle a, d, b, c, e, a, d, f \rangle$. P_1 and P_2 appear respectively in σ as: $\langle a, d, \mathbf{b}, \mathbf{c}, e, a, \mathbf{d}, f \rangle$ and $\langle a, d, \mathbf{b}, \mathbf{c}, \mathbf{e}, a, d, f \rangle$ since this is the alignment that occupies most of the trace and is a leftmost occurrence (leftmost optimal alignment). However, for $P = seq(xor(a, seq(b, c)), seq(d, e))$, if we're tempted to apply the growth procedure, we find $L(d, P) = d$ and the appearance of the *xor* subtree as well as the leaf d is considered a validated context. As such the realignment would take the form: $\langle a, d, \mathbf{b}, \mathbf{c}, e, a, \mathbf{d}, \overrightarrow{f} \rangle$. Of course, it would be an unfructuous one. Yet, the trace does contain the pattern P, albeit on a different *xor* path. The shadow map can be given as: $\langle \mathbf{a}, \mathbf{d}, b, c, \mathbf{e}, a, d, f \rangle$. As in the loop case, the *xor* subtree is indeed a validated context but it doesn't appear necessarily the same way.

Another issue with the *xor* operator can be observed in the following case. Let $P'_1 = seq(xor(a, b), d)$, $P'_2 = seq(xor(a, c), d)$ and a trace $\sigma' = \langle c, b, a, d \rangle$. P'_1 and P'_2 appear respectively as: $\langle c, \mathbf{b}, a, \mathbf{d} \rangle$ and $\langle \mathbf{c}, b, a, \mathbf{d} \rangle$. Our objective is to align $P' = seq(xor(a, seq(b, c)), d)$. On another hand, in the growth algorithm, we tend to realign the changes, while mostly leaving the leftmost validated context untouched. However, aligning the changes below the choice operator is

not the right thing to do. Indeed, the realignment would take the form $\langle c, \mathbf{b}, \underrightarrow{a, d} \rangle$ and wouldn't yield a result. Conversely, adopting an other path in the *xor* subtree allows us to determine an appearance of $P : \langle c, b, \mathbf{a}, \mathbf{d} \rangle$. This path will bypass the new changes that are non existent in the trace. It is important to know that the a path was not chosen initially because the other path was encountered earlier in the trace.

In conclusion, if we would want to apply the growth procedure, the alignment of the choice operator must be the one that leaves the most space on the right for new behaviors in the tail of the trace. Only in that case, can it serve as an appropriate validated context. Also, the *xor* subtree needs to be reevaluated when its offsprings are modified. Contrary to the loop case presented earlier, we cannot guarantee the suitable alignment for the growth method by limiting a static parameter like the number of iterations. The path to choose is really case-dependent and a unique algorithm cannot cover them all without adding too much complexity that may be detrimental in the long term. Facing this, we decided to align classically trees containing *xor* operators and guarantee this way the detection of the pattern in the trace.

The Loop Block

This section is dedicated to the intuition behind the loop block. First, we consider the pattern P in Fig. 5.1 and its two seeds P_1 and P_2. Let the following trace $\sigma = \langle a, c, a, b, c, b, d, e \rangle$ where both P_1 and P_2 appear in the respective forms: $\sigma = \langle \mathbf{a}, \mathbf{c}, \mathbf{a}, b, c, b, \mathbf{d}, \mathbf{e} \rangle$ and $\sigma = \langle a, c, a, \mathbf{b}, \mathbf{c}, \mathbf{b}, \mathbf{d}, \mathbf{e} \rangle$.

Now, if we want to align P, the left child of the loop becomes *seq(b, a)*; meaning, we have to find the word $\langle b, a \rangle$, followed by c and then a repetition of $\langle b, a \rangle$. So, with respect to the seeds, the combination operation brought two changes in what we are searching for, one before the right child of the loop c and another after; whereas the sequence and the concurrence operators bring only one change. Both these alterations need to be aligned and the validated context is minimal: b. Indeed, we search for $\langle a, c, b, a \rangle$ in $\sigma = \langle a, c, a, \mathbf{b}, \underrightarrow{c, b, \mathbf{d}}, \mathbf{e} \rangle$ which doesn't exist in this case. In the end, most of the behavior we are searching for needs a complete alignment and detecting a validated context comes down to assessing a lot of individual cases with a low probability of seeing actual gains. An extreme example is having a cascade of loops like : $P' = loop(loop(loop(seq(a, b), c), d), e)$. When we align this tree, the change introduced is only at the level of $seq(a, b)$ but it has a repercussion on all the higher loops. Each one of them has to modify the first occurrence of the left child as well as its repetition. A complete realignment of the highest loop containing the change is necessary; which is exactly the definition of the loop block.

An alignment unwrapping

We present in the following an instance of an alignment using the growth procedure. The tree to align is R depicted in Fig. 5.2 along with its seeds R_1 and R_2. We suppose the trace to align on is: $\sigma = \langle a, e, f, c, b, c, a, b, c, d, f, e \rangle$. The seed R_1 is contained as: $\langle \mathbf{a}, e, f, \mathbf{c}, b, c, a, b, c, \mathbf{d}, \mathbf{f}, \mathbf{e} \rangle$ and R_2 as: $\langle a, e, f, c, \mathbf{b}, \mathbf{c}, a, b, c, \mathbf{d}, \mathbf{f}, \mathbf{e} \rangle$. Here's the alignment's recursive evolution. We first set $P' = R$ and $\hat{c}' = \sigma$.

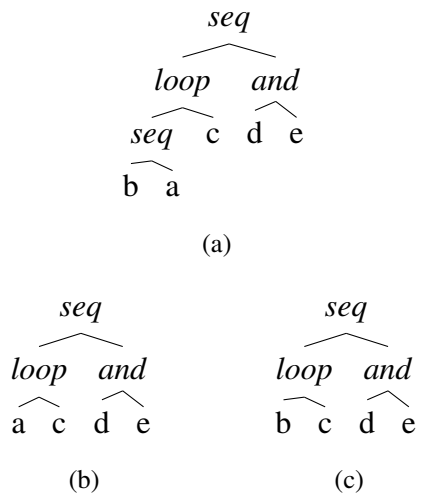


Figure 5.1: A tree P (a) and its seeds: P_1 (b) and P_2 (c).

1. **(Case 1.2.)** We have $P'_1 = R_1$ is not a leaf and $P' = R$ writes as a sequence. As $L(a, R_1) = a$ is contained in the first child, we realign *and(seq(seq(a, b), c), d)* on σ. We set $P' = and(seq(seq(a, b), c), d)$, $P'_1 = and(seq(a, c), d)$ and $P'_2 = and(seq(b, c), d)$ and $\hat{c}' = \sigma$.

   (a) **(Case 2.)** We have P'_1 is not a leaf and the root of P' is a concurrence operator. As such, d is a validated context. The shadow map of P' and a fortiori of R contains $\langle a, e, f, c, b, c, a, b, c, \mathbf{d}, f, e\rangle$. We align $P' = seq(seq(b, a), c)$ with $P'_1 = seq(a, c)$, $P'_2 = seq(b, c)$ and $\hat{c}' = \sigma$.

      i. **(Case 1.2.)** We have $L(a, P'_1) = a$ contained in the left child of P'. So we have to align the first child $P' = seq(b, a)$ on $\hat{c}' = \sigma$. In this case, $P'_1 = a$ and $P'_2 = b$.

         A. **(Case 4.)** We have P'_1 is a leaf and $P' = seq(b, a)$. b and a have already been aligned in the forms: $\langle \mathbf{a}, e, f, c, b, c, a, b, c, d, f, e\rangle$ and $\langle a, e, f, c, \mathbf{b}, c, a, b, c, d, f, e\rangle$ respectively. However, we notice that in this alignment a is not after b. So we have to align a after b: $\langle a, e, f, c, \mathbf{b}, c, a, b, c, d, f, e\rangle$; (the actual trace aligned is the projection on the considered alphabet: $\langle a, e, f, c, \mathbf{b}, \underrightarrow{a}\rangle$) which results in $\langle a, e, f, c, \mathbf{b}, c, \mathbf{a}, b, c, d, f, e\rangle$. That is the alignment of *seq(b, a)*.

      ii. (Follow-up of (i), **Case 1.2.**) The left child *seq(a, b)* of P' has just been aligned. We have that $P'_1 = seq(a, c)$ and $P'_2 = seq(b, c)$ have already been aligned in the seeds in the respective forms: $\langle \mathbf{a}, e, f, \mathbf{c}, b, c, a, b, c, d, f, e\rangle$ and $\langle a, e, f, c, \mathbf{b}, \mathbf{c}, a, b, c, d, f, e\rangle$. Both the occurrences of c in these two forms are before the occurrence of $seq(b, a)$. So we have to realign c after the latter occurrence: $\langle a, e, f, c, \mathbf{b}, c, \mathbf{a}, \underrightarrow{b, c, d, f, e}\rangle$ (here too, the actual trace realigned: $\langle a, e, f, c, \mathbf{b}, c, \mathbf{a}, \underrightarrow{c}\rangle$) which results in $\langle a, e, f, c, \mathbf{b}, c, \mathbf{a}, b, \mathbf{c}, d, f, e\rangle$. This is the alignment of $P' = seq(seq(b, a), c)$.

(b) (Follow-up of (a), **Case 2.**). The only thing to do here is to incorporate the validated context d and construct the alignment of P': $\langle a, e, f, c, \mathbf{b}, c, \mathbf{a}, b, \mathbf{c}, \mathbf{d}, f, e\rangle$

2. (Follow-up of (1), **Case 1.2.**) The first child of R, $and(seq(seq(a, b), c), d)$ has been aligned. Now we test if the already computed occurrence of the right child *and(f,e)* exists after the first child. That is the case and the occurrence is considered a validated context. Finally, the complete alignment of R is $\langle a, e, f, c, \mathbf{b}, c, \mathbf{a}, b, \mathbf{c}, \mathbf{d}, \mathbf{f}, \mathbf{e}\rangle$.

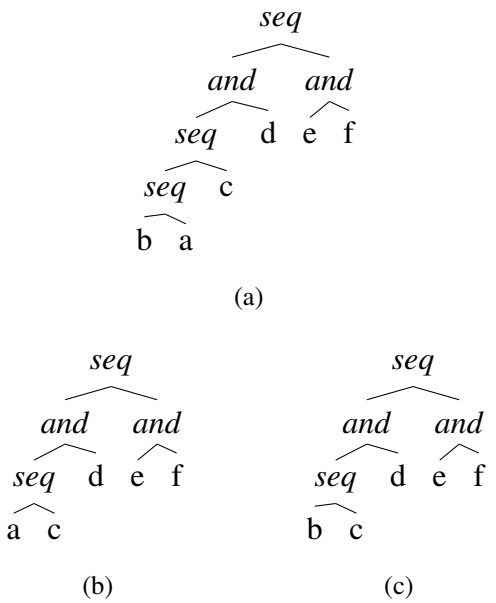


Figure 5.2: A tree R (a) and its seeds: R_1 (b) and R_2 (c).

In conclusion, we took advantage of the validated contexts and aligned just two times a single leaf on a single leaf trace. We reduced an exponentially hard alignment problem to two *if conditions*. In fact, the more the validated context alignment is time consuming, the higher the gains.

The Redefinition of the Potential Combination Leaves

We revised the definition of the potential combination leaves to take full advantage of the alignment growth procedure. We present in the following the new definition before elaborating on the justification. It is to be noted that Theorem 3.4 is still valid and can be demonstrated following an analogous proof to the one in Section 3.1.

Definition 5.7 (Potential Combination Leaves)**.** *Given a process tree* P *of depth* i*, a leaf node* a *of depth* d *is called potential combination leaf, if* $d \geq i - 1$ *and there is no leaf* b *of depth* d' *on the* ***right of*** a *(*instead of left*) such that* $d' > d$.

First of all, it is important to note that, apart from **Case 5.**, where the alignment of the pattern doesn't involve classical alignment at all, the most beneficial case we can encounter during the alignment growth is **Case 1.1.** Indeed, it allows us to detect a structural validated context, meaning a context that doesn't depend on the trace plus the realignment being executed on a partial trace. There are two other cases that show a gain in the form of validated contexts. **Case 1.2** which is

not as interesting because the existence of the validated context depends on the trace and **Case 2** which is structural in nature too but doesn't shorten the realignment trace.

Knowing this, let's analyze Fig. 5.3 and Fig. 5.4. They represent a sequence of templates. *X* represents leaves while *Y* represents operators. The leaves don't necessarily share the same values; the same for operators. The two symbols are more of a placeholder. Actually, the trees are templates of structures and each tree (for example (d)) is the template for the seeds of the next tree (for example (e)). The red *X* indicates the position for the next combination operation and the blue *Y* indicates the defining operator of the current tree. Starting from the tree (h), and going back, each tree represents the structure shared by all the seeds in the corresponding level of the construction tree of (h). The tree (h) for the level 0, the tree (g) for the level 1, ... and the tree (a) for level 7.

The only difference between the two figures is that the first used the old definition of potential combination leaves and the second the new one. Since the rules are no longer the same, the unique path that must be followed to construct a certain pattern (the construction tree) is not the same which is clearly visible in the two figures. If we look closely, we notice that in the first, patterns are constructed from right to left; meaning, the structures on the right of the trees remain static while new behavior is added progressively on the left. During the alignment process, it means that we have to align the new behavior which is on the left and align after it the static behavior on the right (of course, it depends on the actual operators in the tree but we are more concerned here with the general tendency). This generally falls under **Case 1.2** which is an unwanted behavior as it is rare to detect a validated context and a classical realignment is often required after the occurrence of the changed behavior. To sum up, using Definition 3.1, the right portion of the tree complexifies while new behavior appears on the left side. The order of the changes in the tree doesn't follow the order of the exploration of the traces.

On the opposite, by adopting Definition 5.7, the complexification process follows the order of reading of the trace: from left to right. Indeed, the left parts remain static while new behaviors are added on the right. This allows to use previous alignments through **Case 1.1.** when the operators permit it. As stated earlier, this the most favorable case. In the remainder of the chapter, we use Definition 5.7.

5.2 Post-processing

The number of behavioral patterns returned by COBPAM may be overwhelming. In order to cater for this issue, we include further processing steps at the end of the execution. We operate on two aspects, Maximality and Equivalence. In Section 5.2.1, we define extended types of maximality that we coin generalized. In all these types as in the original one, the idea is to avoid outputting trees that can be deduced from others. In Section 5.2.2, we present definitions of equivalent trees. Of course, equivalent trees should be outputted only once.

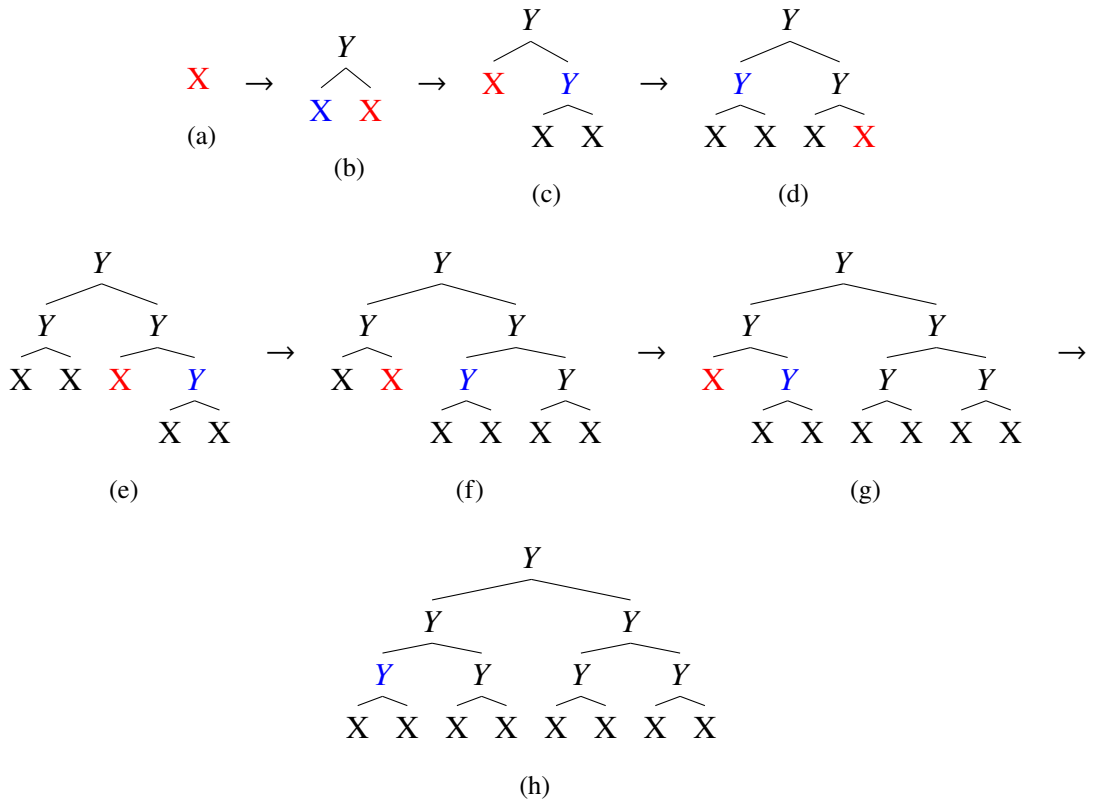


Figure 5.3: Construction tree of the template (h) according to the first definition of potential combination leaves. (Definition 3.1).

5.2.1 Generalized Maximality

In the following, we present two new types of maximality. A maximal tree is one that can replace its seed/seeds in the output of COBPAM/ACOBPAM. It serves as a representative. Therefore, we define the generalized maximality through the different kinds of seeds.

Alternative Seeds

In Section 3.3.2, we introduced maximal process trees as ones that are the largest representatives of a group of trees of smaller size. As such, a maximal tree aggregates all the information about that group in one big pattern. Maximal trees are defined with respect to their construction trees and in a small measure to their seeds. However, the monotonicity property on which the maximality property is built can be extended. Indeed, in the case where a maximal tree is generated from a pair of seeds, the latter serve as a context that will be constrained. The seeds of a pattern in our method are well defined and unique according to Theorem 3.4 and allow for a smooth traversal of the construction graph without redundant construction. But when we consider an isolated frequent tree P, many of what we call alternative seeds can be combined to construct it and they also serve as a context to P. Alternative seeds differ in only one leaf just like the regular seeds albeit that leaf is not at a position of a potential combination leaf. We give in Fig. 5.5 an example of a tree T with

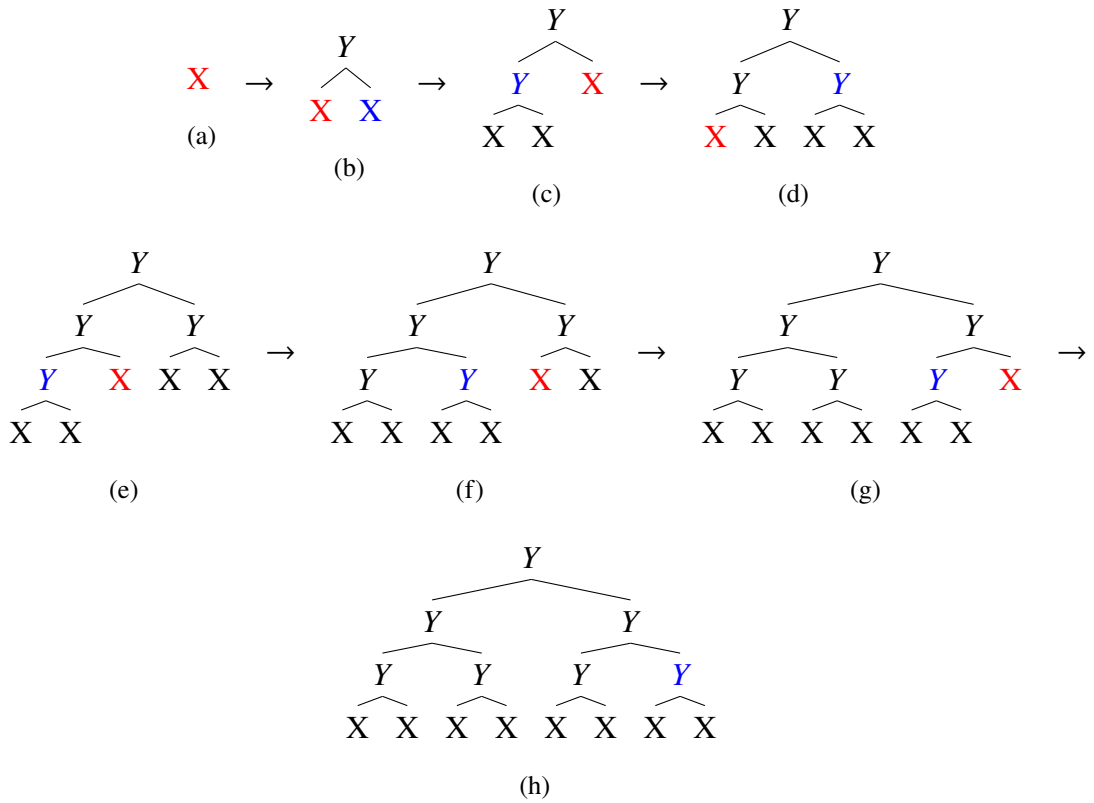


Figure 5.4: Construction tree of the template (h) according to the second definition of potential combination leaves. (Definition 5.7).

its two seeds T_1 and T_2. However, we can also see two alternative seeds that can be combined to get T: T_3 and T_4. The alternative seeds of a tree P are not combined following a potential combination leaf.

In Alg. 1, we can see that when adding a pattern P in the final frequent patterns set, its offspring seeds are deleted. However, the alternative seeds are not accessible since they follow another construction path that respects Theorem 3.4. They are actually on another portion of the construction graph browsed by the algorithm and there is no connection between them and P because the edges symbolize a regular seed relationship. However, If the ancestor trees that are on their path, and thus constructed out of them, are frequent, then they are deleted by the same mechanism. In the other case, they stay in the final set even though they are perfectly deductible from other trees. So the idea is to delete them once the discovery finished in a post-processing step.

In the following, we first give a formalization of the alternative seeds, then prove their monotonicity property.

Definition 5.8. *Given a process tree P, we define a function, f, that maps P to a set of trees.*

- *if $P = a$ with $a \in A$, an activity, then $f(P) = \{a\}$.*

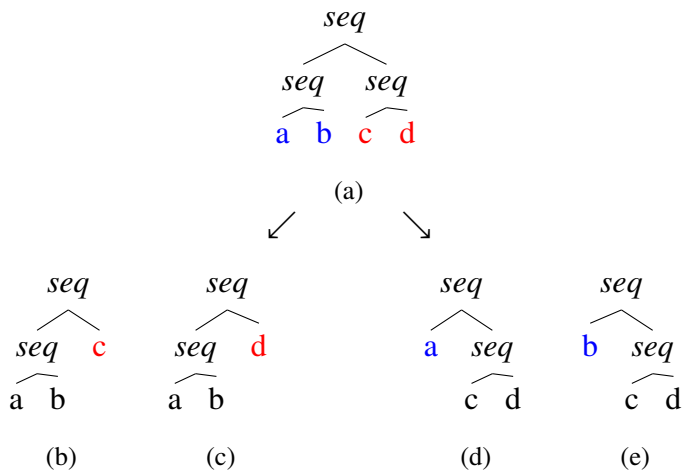


Figure 5.5: A tree T (a), its seeds: T_1 (b) and T_2 (c) and two alternative seeds: T_3 (d) and T_4 (e)

- *if $P = x(P_1, P_2)$ where x is a constraining operator and P_1, P_2 children subtrees, $f(P)$ is given as the union of:*
  - $f(P_1)$.
  - $f(P_2)$.
  - *the set:* $\{x(P'_1, P'_2) \mid P'_1 \in f(P_1), P'_2 \in f(P_2)\}$.
- *if $P = xor(P_1, P_2)$ with P_1, P_2 children subtrees, then $f(P) = \{xor(P'_1, P'_2) \mid P'_1 \in f(P_1), P'_2 \in f(P_2)\}$.*

For a process tree of depth $n \in \mathbb{N}^*$, the elements in $f(P) \setminus \{P\}$ are the union of the pair of "the" seeds of P with its alternative seeds. In particular, alternative seeds are defined for trees of at least depth two. If we consider such a tree P then its alternative seeds are the elements of $f(P) \setminus \{P\}$ minus the regular seeds defined through the combination operation (see Theorem 3.4). The depth condition is justified by the absence of seeds for depth zero and the absence of alternative seeds (only regular) for depth one.

A generalized monotonicity property states that if a process tree P is frequent then all its alternative seeds are frequent too. As such, in the output of ACOBPAM, we make sure that none of the patterns discovered are alternative seeds for others. We prove the generalized monotonicity as well as the regular one through recursion.

Monotonicity. We will prove by induction that $\forall n \in \mathbb{N}^*$ the following statement is true:
S(n): "if a process tree P of depth n is frequent then any process tree P' in $f(P)$, the set of regular and alternative seeds, is frequent too. More precisely, considering a case c and its trace $\hat{c}$, if $\epsilon(\hat{c}, P) = 1$ then $\epsilon(\hat{c}, P') = 1$"

Base case: When $n = 1$: P writes as $P = x(a, b)$ with $a, b \in A$ two activities. If x is:

- **a constraining operator:** Whatever the constraining operator x is (*seq*, *and* or *loop*) we have, if $\epsilon(\hat{c}, P) = 1$ then $\upsilon(\hat{c}, P) \in \{\langle ab\rangle, \langle ba\rangle, \langle aba\rangle, \langle ababa\rangle, \langle abababa\rangle \dots\}$. If the trace exhibits any of the words in the previous set, then it surely exhibits:
  - $\langle a\rangle$. So, for $P' = a \in f(P)$, $\epsilon(\hat{c}, P') = 1$.
  - $\langle b\rangle$. The same reasoning as above is followed.

  Finally, for $P' = x(a, b) \in f(P)$, we have $P' = P$ and $\epsilon(\hat{c}, P') = \epsilon(\hat{c}, P) = 1$ □
- **the xor operator:** For $P' = xor(a, b) \in f(P)$, we have $P' = P$ and $\epsilon(\hat{c}, P') = \epsilon(\hat{c}, P) = 1$ □

We conclude that *S(1)* is true. □

Induction step Let $n \in \mathbb{N}^*$ and we suppose $S(l)$ is true $\forall l \leq n$. We aim to prove $S(n+1)$. Let $P = x(P_1, P_2)$, a tree of depth $n+1$. P_1, P_2 are two subtrees with the respective depths d and d'. We suppose $\epsilon(\hat{c}, P) = 1$. According to x, we have:

x **is a constraining operator:** This means that P_1, P_2 are present in $\hat{c}$. We set $w_1 = \upsilon(\hat{c}, P_1)$ and $w_2 = \upsilon(\hat{c}, P_2)$. For $P' \in f(P)$, we separate three cases:

- if $P' \in f(P_1)$, then according to $S(d)$ and since P_1 is present in the trace, P' is frequent too and we have $\epsilon(\hat{c}, P') = 1$ □
- if $P' \in f(P_2)$, we follow the same reasoning as above since P_2 is interchangeable with P_1 □
- $P' = x(P'_1, P'_2)$ where $P'_1 \in f(P_1)$ and $P'_2 \in f(P_2)$. Depending on x, we separate the following cases:
  - ***x = seq***. We have that $\upsilon(\hat{c}, P) = w_1.w_2$. Meaning, P_1 is present in $\sigma_1 = hd^{\beta_h(\hat{c},P_1)}(\hat{c})$. By applying $S(d)$ on P_1 and the trace head σ_1, we conclude that P'_1 is present in σ_1 and we set $w'_1 = \upsilon(\sigma_1, P'_1)$. Similarly, we know that P_2 is present in $\sigma_2 = tl^{\beta_l(\hat{c},P_2)}(\hat{c})$. We apply $S(d')$ on P_2 and σ_2 and conclude: $\epsilon(\sigma_2, P'_2) = 1$. We also set $w'_2 = \upsilon(\sigma_2, P'_2)$. Finally, we can deduce that $\epsilon(\hat{c}, w'_1.w'_2) = 1$ and $\epsilon(\hat{c}, P') = 1$ □
  - ***x = loop***. Using the definition of the *loop* operator, we have $\upsilon(\hat{c}, P) = w_1.(w_2.w_1^2).(w_2^2.w_1^3).(w_2^3.w_1^4)\dots$ where $w_1, w_1^2, w_1^3, w_1^4, \dots \in \Sigma(P_1)$ and $w_2, w_2^2, w_2^3, \dots \in \Sigma(P_2)$. We can assert that :
    - P_1 is present in the trace $\sigma_1 = hd^{\beta_h(\hat{c},P_1)}(\hat{c})$ and by applying $S(d)$ on P_1 and the trace head σ_1, we conclude that P'_1 is present in σ_1 and we set $w'_1 = \upsilon(\sigma_1, P'_1)$.
    - P_1 is present in the trace $\sigma_3 = tl^{\beta_h(\hat{c},P_2)+1}(\hat{c})$ and by applying $S(d)$ on P_1 and the trace tail σ_3, we conclude that P'_1 is present in σ_3 and we set $w'^2_1 = \upsilon(\sigma_3, P'_1)$
    - P_2 is present in the trace $\sigma_2 = In^{\beta_h(\hat{c},P_2)}_{\beta_l(\hat{c},P_2)}(\hat{c})$ and by applying $S(d')$ on P_2 and the trace σ_2, we conclude that P'_2 is present in σ_2 and we set $w'_2 = \upsilon(\sigma_2, P'_2)$

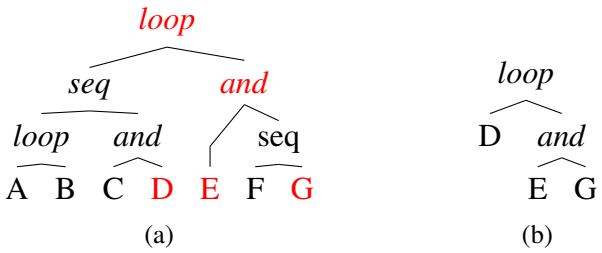


Figure 5.6: A tree P (a) and one of its alternative seeds (b).

Thanks to the constructions of the traces σ_1, σ_2 and σ_3, we can state that $w' = w'_1.w'_2.w'^2_1$ exists in $\hat{c}$. Since $w' \in \Sigma(loop(P'_1, P'_2))$, we conclude that $\epsilon(\hat{c}, P') = 1$ □

– ***x* = *and***. Using the definition of the concurrence operator, we have $\upsilon(\hat{c}, P) \in w_1 \diamond w_2$. On another hand, if we apply $S(d)$ (resp. $S(d')$) on P_1 (resp. P_2) and $\hat{c}$, we get $\epsilon(\hat{c}, P'_1) = 1$ (resp. $\epsilon(\hat{c}, P'_2) = 1$). We set then $w'_1 = \upsilon(\hat{c}, P'_1)$ and $w'_2 = \upsilon(\hat{c}, P'_2)$. Since both P'_1 and P'_2 are present in $\hat{c}$, then an interleaving w' of w'_1 and w'_2 exists in the trace. In other words, $w' \in w'_1 \diamond w'_2$ Which implies: $\epsilon(\hat{c}, and(P'_1, P'_2)) = \epsilon(\hat{c}, P') = 1$ □

x **is a choice operator:** Meaning $P = xor(P_1, P_2)$. We know that P' writes as: $P' = xor(P'_1, P'_2)$ where $P'_1 \in f(P_1)$ and $P'_2 \in f(P_2)$. On another hand, Since P is present in the trace, either P_1 or P_2 is present in the trace (or both). If P_1 is present, in other terms, $\epsilon(\hat{c}, P_1) = 1$, by applying $S(d)$ on P_1, we have $\epsilon(\hat{c}, P'_1) = 1$. This means that one path in P' is satisfied which translates ultimately to: $\epsilon(\hat{c}, P') = 1$ □

In the case where P_2 is present without P_1, we apply an analogous reasoning.

We have proven that for P of depth $n+1$, $P' \in f(P)$ and for a trace $\hat{c}$, if $\epsilon(\hat{c}, P) = 1$ then, $\epsilon(\hat{c}, P') = 1$. We conclude that *S(n+1)* is true. □

Finally, by the principle of induction, *S(n)* is true $\forall n \in \mathbb{N}^*$. □

Concerning the post-processing step, in order to discard alternative seeds from the output set, we compare the patterns pairwise. For two process trees, P' the less deep one and P, we test if P' belongs to $f(P)$. We remove it if that is the case. This is realized through the function *isSeed(P', P)* in *Alg*. 5.

Loop Seeds

The loop operator exhibits the particular property that its behavior cannot appear without involving that of another operator: the sequence. Indeed, when respecting the compactness property and the changes we introduced in Section 5.1.4, the behavior of $loop(a, b)$ is $\Sigma(loop(a, b)) = \{\langle a, b, a\rangle\}$. If a trace contains this behavior (the next statement is also valid in the case of several iterations as in the classical COBPAM algorithm), it automatically contains $\Sigma(seq(a, b)) = \{\langle a, b\rangle\}$ and $\Sigma(seq(b, a)) = \{\langle b, a\rangle\}$. Therefore, outputting the three trees $seq(a, b)$, $seq(b, a)$ and $loop(a, b)$ is one of the redundancies we strive to avoid.

Algorithm 5: Function *isSeed*

input : P, a process tree ;
P', a process tree.
output : res, a boolean.

```
1  if depth(P') ≤ depth(P) then
2      if P is a leaf then
3          res ← (P' = P);
4      else if P = x(P_1, P_2) with P_1, P_2, two subtrees then
5          if x is a constraining operator AND (isSeed(P', P_1) OR isSeed(P', P_2)) then
6              res ← true;
7          else if P' = x(P'_1, P'_2) where P'_1, P'_2 two subtrees AND isSeed(P'_1, P_1) AND isSeed(P'_2, P_2) then
8              res ← true;
9  else
10     res ← false;
```

In fact, if in the final set, a sequence operation in a tree P' is already accounted for in another pattern P through a loop, then we remove P'. That is because P' with its sequence operators can serve as a context for looping. We call P' a loop seed. Of course, the difference between P and P' is at the level of the loop operator with potentially inverted children.

Detecting a loop seed is performed syntactically. If P' is a loop seed of a tree P, then the difference between their representative words must be a loop operator in P instead of a sequence operator as in P' with a potential inversion of leaves.

5.2.2 Trees Equivalency

Not only do we eliminate non maximal trees in the post-processing operation but we also detect equivalent trees. We distinguish two notions of equivalency between behavioral patterns:

- **Syntactical equivalency:** This equivalence appears as a direct result to the existence of symmetrical operators in the process trees. Indeed, the same order-insensitive process tree can be depicted in more than one way considering the interchangeability of the children of every symmetrical operator.
- **Behavioral equivalency:** two behavioral patterns are behaviorally equivalent if their languages are equal. In other words, the set of traces they generate are the same.

In order to achieve the objective behind our post-processing, we eliminate redundant equivalent trees. It is to be noted that syntactically equivalent process trees were already taken care of in the main algorithm of COBPAM. This was handled through the definition of a unified representative word that is unique to all syntactically equivalent trees. It works by forcing an order on the children of each symmetrical operator. We adopt the lexicographical order of the leaves so that the left child must be inferior to the right child. In the case where either or both children are subtrees, the comparison is performed according to what we call representative leaves. The representative leaf

of a subtree is the lowest leaf in the lexicographical order among all the leaves it contains. The unified representative word is then generated by the pre-order traversal of the tree while respecting the previous order. For example, the unified representative word of the tree *seq(and(d, c), a)* is '(c d and a seq)'. Going back to the COBPAM algorithm, each time we add a new tree to Θ, we make sure that it doesn't contain a tree with the same unified representative word as the new tree.

Post-processing, in turn, makes sure to avoid outputting trees exhibiting a behavioral equivalency. Accordingly, all trees outputted are behaviorally unique. Moreover, if an alternative or loop seed is removed, all its equivalent trees are removed too. Behaviorally equivalent trees are detected by comparing their languages.

5.3 Visualization

In order to navigate the behavioral patterns uncovered by our method, we conceive a graph-based visualization. Not only does it ensure a global and simultaneous view of all patterns but also harbors interesting relationships between them.

In the visualization graph, nodes represent patterns in the final set of retained trees after the post-processing step. The boundaries of patterns will be utilized in the definition of the relationships binding them. We consider four types of relationships:

For Ω, the final set of behavioral patterns and P_1, P_2 two behavioral patterns in Ω:

Definition 5.9 (Follows relationship)**.** *The* Follows relationship $\mathcal{F} \subset \Omega^2$ *is defined using a support metric,* $support_f(P_1, P_2)$:

$$support_f(P_1, P_2) = \frac{|\{c \in L \mid \epsilon(c, P_1) = 1 \ \wedge \ \epsilon(c, P_2) = 1 \ \wedge \ \beta_h(P_1) < \beta_l(P_2)\}|}{|L|}$$

For a threshold τ_f*, we have,* $(P_1, P_2) \in \mathcal{F}$ *if and only if:*

$$support_f(P_1, P_2) \geq \tau_f$$

In other words, if the relative frequency of the traces inside the log where the appearance of P_2 *is after the appearance of* P_1 *is greater than the threshold* τ_f *then,* $(P_1, P_2) \in \mathcal{F}$ *and we say that* P_2 *follows* P_1*.*

This relationship can be translated by: P_1 followed by P_2 is an order frequently seen in the log.

Definition 5.10 (Inter-follows relationship)**.** *If* $(P_1, P_2) \notin \mathcal{F}$*, the* Inter-follows relationship $\mathcal{F}' \subset \Omega^2$ *is defined using a support metric,* $support_{if}(P_1, P_2)$*.*

$$support_{if}(P_1, P_2) = \frac{|\{c \in L \mid \epsilon(c, P_1) = 1 \ \wedge \ \epsilon(c, P_2) = 1 \ \wedge \ \beta_h(P_1) < \beta_l(P_2)\}|}{|proj(P_1) \cap proj(P_2)|}$$

For a threshold τ_{if}*, we have,* $(P_1, P_2) \in \mathcal{F}'$ *if and only if:*

$$support_{if}(P_1, P_2) \geq \tau_{if}$$

In other words, if the relative number of traces inside the shared traces between P_1 and P_2 where the appearance of P_2 is after the appearance of P_1 is greater than the threshold τ_{if} then, $(P_1, P_2) \in \mathcal{F}'$ and we say that P_2 inter-follows P_1.

In the case where there is no *Follows relationship* between P_1 and P_2, we can conclude that P_1, P_2 don't appear in this order frequently. However, this can be due to the fact that they do not frequently appear together at all. In this case, if the *Inter-follows relationship* is validated, that means that when they *do* occur together, they always appear in that order. This is an interesting insight too. We can give the following real life example to illustrate our point. Imagine a senior consultant who always intervenes with his assistant and a junior consultant who intervenes alone. We suppose both senior and junior consultant are called up for missions frequently. In a hypothetical log, the relation of hierarchy (analogous to the order in the traces) between the senior consultant and his assistant is frequent in the whole log. However, since the senior and junior consultants rarely work on the same mission, the hierarchy relationship between them cannot be seen at the level of the log. Yet, if we study the rare occurrences where they *did* work together, we can see a frequent hierarchical relationship. Arguably, this is an important information. That is what the *Inter-follows* relationships captures.

Definition 5.11 (Spans relationship)**.** *The* Spans relationship $\mathcal{S} \subset \Omega^2$ *is defined using a support metric, $support_s(P_1, P_2)$:*

$$support_s(P_1, P_2) = \frac{|\{c \in L \mid \epsilon(c, P_1) = 1 \ \wedge \ \epsilon(c, P_2) = 1 \ \wedge \ \beta_h(P_1) < \beta_h(P_2) \ \wedge \ \beta_l(P_1) > \beta_l(P_2)\}|}{|L|}$$

For a threshold τ_s, we have, $(P_1, P_2) \in \mathcal{S}$ if and only if:

$$support_s(P_1, P_2) \geq \tau_s$$

In other words, if the relative number of traces inside the log where the appearance of P_1 fits inside the interval defined by the boundaries of P_2 is greater than the threshold τ_s then, $(P_1, P_2) \in \mathcal{S}$ and we say that P_2 spans P_1.

The *Spans relationship* can be translated by: It is frequently observed in the log that the occurrence of P_1 spans the occurrence of P_2.

Definition 5.12 (Inter-spans relationship)**.** *If $(P_1, P_2) \notin \mathcal{S}$, the* Inter-spans relationship $\mathcal{S}' \subset \Omega^2$ *is defined using a support metric, $support_{is}(P_1, P_2)$:*

$$support_{is}(P_1, P_2) = \frac{|\{c \in L \mid \epsilon(c, P_1) = 1 \ \wedge \ \epsilon(c, P_2) = 1 \ \wedge \ \beta_h(P_1) < \beta_h(P_2) \ \wedge \ \beta_l(P_1) > \beta_l(P_2)\}|}{|proj(P_1) \cap proj(P_2)|}$$

For a threshold τ_{is}, we have, $(P_1, P_2) \in \mathcal{S}'$ if and only if:

$$support_{is}(P_1, P_2) \geq \tau_{is}$$

In other words, if the relative number of traces inside the shared traces of P_1 and P_2 where the appearance of P_1 fits inside the interval defined by the boundaries of P_2 is greater than the threshold τ_{is} then, $(P_1, P_2) \in \mathcal{S}'$ and we say that P_2 inter-spans P_1.

The intuition behind the *Inter-spans relationship* is the same as for the *Inter-follows relationship*.

Finally, we define the visualization graph.

Definition 5.13 (Visualization graph)**.** *The visualization graph $G = (\Omega, \mathcal{F}, \mathcal{S}, \mathcal{F}', \mathcal{S}')$ is a directed multigraph where Ω is the final set of retained patterns or nodes after the post-processing step, $\mathcal{F}$ is the* Follows *relationship, $\mathcal{F}'$ is the* Inter-follows *relationship, $\mathcal{S}$ is the* Spans *relationship and $\mathcal{S}'$ is the* Inter-spans *relationship. Considering $P_1, P_2 \in \Omega$, a new arc from P_1 to P_2 is created, each time (P_1, P_2) belongs to one of the previous relationships. We apply a transitive reduction on the relationships* Follows *and* Spans *as they are transitive relationships.*

The resulting graph can be seen as a descriptive, non imperative, hierarchical and simplified process model of L where instead of activities, behavioral patterns are used as nodes. The hierarchy is induced by the *Spans* and *Inter-spans* relationships. Moreover, the relationships in this graph operate at two levels of granularity. One between patterns as discussed in the definitions above and the other between activities themselves inside the patterns. This offers even more analytical information. Indeed, in the case of "spaghetti" processes, such a structure is highly useful in reducing the difficulty of the analysis while fostering interesting insights. We recall that this isn't an abstraction from low-level events to high-level activities which is a separate subject of research [9, 57, 79]. Our visualization technique can be adapted to any event log unlike in this research branch where the orderings of fine-granular events must have an atomic and clear semantic which is translated afterwards to activities. For example, the sequence *"open dishwasher", "put plates", "close dishwasher"* can be transformed to a high-level activity: *"use dishwasher"*. This transformation is not always possible in our case as can be seen in Fig. 1.3d; a pattern that can't be labelled by such a self-explanatory title. Furthermore, we insist on the fact that our goal is not to discover an end-to-end process model with abstraction properties but just to offer a global view of the discovered patterns.

5.4 Conclusion

In this chapter, we detailed our last contribution, named ACOBPAM. First, we introduced the alignment growth procedure where we defined important notions, presented the algorithm and then gave some of our intuitions along with the changes we had to incorporate. Particularly, we defined new types of seeds that respect also the monotonicity property and tried to detect all cases of process equivalency. Second, we defined the generalized maximality and equivalency notions for our post-processing purposes. Finally, we devised four relationships between patterns and

integrated them in an interactive visualization graph. The following chapter validates our methods through a thorough investigation and experimentation.

6 Experimental Evaluation

Contents

This chapter presents the evaluation of our algorithms both in terms of efficiency and effectiveness on real life event logs. We start by presenting our setup along with the logs used in Section 6.1. Then, we experiment on each contribution: COBPAM in Section 6.2, the Data-aware Analysis Framework in Section 6.3 and finally, ACOBPAM in Section 6.4. We also discuss the limitations of the experiments in Section 6.5.

6.1 Setup and Datasets

All our algorithms were implemented as a plugin in the ProM framework [84] as the package *BehavioralPatternMining* except for the visualization graph generation. Due to the limited capabilities of the ProM framework in terms of graph drawing, we extract a JSON model of the visualization graph (containing behavioral patterns and their interdependencies) and use it to generate the graph interface itself through the library GraphViz and Pydot. The Python script is publicly available[1]. Note that we ran the experimental evaluation on a PC with an i7-2.2Ghz processor, 16GB RAM and Windows 10.

Our experiments used the following real-world event logs. They cover different domains and are publicly available.[2] Moreover, they are related to flexible processes, are of reasonable size to be explored and include data attributes with clear semantics.

- Sepsis: A log of a treatment process for Sepsis cases in a hospital. It contains 1050 traces with 15214 events that have been recorded for 16 activities.
- Traffic Fines: A log of an information system managing road traffic fines, containing 150370 traces, 561470 events, and 11 activities.
- WABO: A log of a building permit application process in the Netherlands. It contains 1434 traces with 8577 events, recorded for 27 activities.
- BPI_2019S1: A 30% sample of the BPI Challenge 2019 log. The log belongs to a multinational company working in the area of coatings and paints and records the purchase order handling process. The sample regroups 479845 events distributed over 75519 traces with 41 event classes.
- BPI_2019S2: A 40% sample of the previously mentioned event log, BPI Challenge 2019. The sample regroups 670583 events distributed over 105962 traces with 42 event classes.

For the above event logs, we derived possible contexts based on the recorded attributes. Moreover, we considered a minimal size for each possible context, set to 50. Since in real-world data, the associated log of a certain context may not be representative of all possible behavior of the context population, such a minimal size helps to avoid the discovery of non-relevant behavioral patterns. The contexts considered for the event logs are summarized as follows.

- Sepsis: Two attributes were considered: 'InfectionSuspected', which is a Boolean variable stating if an infection is suspected, and 'Infusion', a Boolean variable stating if an infusion has been administered.
- Traffic Fines: Contexts were constructed from two attributes, 'amount', the amount of the fine, and 'VehicleClass', the type of the vehicle.

[1] https://github.com/Alchimehd/ACOBPAM-Vis-

[2] https://data.4tu.nl/search?q=:keyword:"real%20life%20event%20logs"

- WABO: From all data attributes available, we chose ‘department’ and ‘channel’ to construct contexts. The former represents the department working on the procedure. The latter refers to the channel of communication with the applicants.
- BPI_2019S1 and BPI_2019S2: Each trace refers to a line item of a certain purchase order. The first attribute considered was the item category, ’Item Category’, which specifies the method of invoice handling. The four categories are: ‘3-way matching, invoice after goods receipt’, ‘3-way matching, invoice before goods receipt’, ‘2-way matching (no goods receipt needed)’, and ‘Consignment’. The second attribute is the company concerned, ’Company’. We observed two values: ‘companyID_0000’ and ‘companyID_0003’.

The discovery algorithms were configured with a threshold of 0.7 for the support and precision (precision in COBPAM and CCOBPAM) and two for the maximal depth of the patterns, unless stated otherwise.

The implementation of LPM discovery in ProM has a single parameter of interest, i.e., the bound for the number of LPMs to discover. We set this bound to 500, the maximal possible value.

For the Episode Miner, for which we presented a qualitative comparison on WABO, the frequency threshold was set to 70% and the 70% most frequent activities were used to construct the episodes. Moreover, as the Episode Miner uses a maximum distance between two events for them to be partially ordered, we set this parameter to 100. Since the size of traces in WABO is below this number, the partial orders become equivalent to the sequence operator in COBPAM.

Concerning the visualization graph, we chose the following thresholds: $\tau_s = \tau_{is} = \tau_f = \tau_{if} = 0.7$.

6.2 COBPAM Evaluation

In this section, we evaluate the efficiency and effectiveness (quantitative and qualitative) of COBPAM by comparing it to the previous, state-of-the-art, behavioral pattern discovery algorithm (LPM discovery) [80]. We also include a qualitative comparison with Declare and Episode Miners.

6.2.1 Efficiency

Running both algorithms for behavioral pattern discovery, we observed the execution times reported in Table 6.1. They depend on the size of the log, the number of activities and events, and the complexity of the behavioral patterns in the log. COBPAM generally turns out to be more efficient. Its runtimes are an order of magnitude smaller than those of LPM discovery, which fails to complete for one of the logs due to memory issues. COBPAM fails to complete for BPI_2019S2 for the same reason. That is because in this first version of our behavioral pattern discovery algorithm, the languages and exact behaviors seen in the log are stored for each evaluated pattern in order to be used later for the computation of the precision metric in the different contexts under CCOBPAM. For some logs, due to the complexity of their construction graph, there is simply not enough memory.

Table 6.1: Execution times of COBPAM and LPM Discovery

	Sepsis	Traffic Fines	WABO	BPI_2019S1	BPI_2019S2
COBPAM	49s	3m	8s	9.1mn	Insuff.Mem
LPM discovery	13mn	Insuff.Mem	19mn	>24h	>24h

Table 6.2: Scalability analysis on Sepsis

Sample	nbAct	nbEvents	Q1	Q2	Q3	nbTrees	nbEva	nbAlig	nbChoice	runtime
10	15	1434	9	13	16	382	17916	9780	5830	9260
20	15	2892	8	13	16	287	8501	5218	0	14
30	16	4416	8	13	17	476	38151	12679	8972	8289
40	16	5685	9	13	17	370	8692	5476	5	27
50	16	7045	9	13	17	356	8727	5499	5	31
60	16	8904	10	13	17	484	13645	8387	1642	85
70	16	10144	9	13	16	344	8624	5428	0	40
80	16	11632	8	13	16	352	8022	4914	0	37
90	16	12693	9	13	16	377	8093	4992	6	41
100	16	15214	9	13	16	375	8029	4967	0	49

Scalability

Next, we aim to study the impact of the size of the log on runtimes and how COBPAM handles scaling up. For that, we sampled two logs, Sepsis and Traffic Fines, to get 9 other logs for each (from 10% to 90% of the size of the whole log). In Table 6.2 and Table 6.3, we report the execution time for each log with other information:

- **nbAct:** the number of activities in the log.
- **nbEvents:** total number of events.
- **Q1, Q2, Q3:** First, second and third quartile on the size of the trace series.
- **nbTrees:** number of trees returned.
- **nbEva:** number of trees evaluated.
- **nbAlig:** number of alignments performed.
- **nbChoice:** number of choice alignments (alignments performed on trees containing a choice operator).

Table 6.2 shows the results on Sepsis. We notice that the execution times are very random in comparison with the size of the log. This is because the size of the log isn't the only factor influencing the execution times. The specific frequent patterns present in the log along with their complexity are another factor. The number of trees kept in the search process which had to be evaluated plays a role too. In fact, the alignments represent the bottleneck of the algorithm (the derivation of the optimal alignment is computationally hard with respect to the size of the trace

Table 6.3: Scalability analysis on Traffic Fines

Sample	nbAct	nbEvents	Q1	Q2	Q3	nbTrees	nbEva	nbAlig	nbChoice	runtime
10	11	55913	2	5	5	4	290	149	149	9
20	11	112247	2	5	5	4	290	145	145	20
30	11	168709	2	5	5	4	290	146	146	34
40	11	224244	2	5	5	4	290	142	142	42
50	11	280529	2	5	5	4	290	142	142	52
60	11	337111	2	5	5	4	290	145	145	76
70	11	393139	2	5	5	4	290	150	150	90
80	11	449320	2	5	5	4	290	139	139	105
90	11	504971	2	5	5	4	290	148	148	141
100	11	561470	2	5	5	4	290	141	141	164

and the complexity of the tree). The more trees are evaluated, the more alignments are performed and the more time is consumed as can be seen in lines 1, 3 and 6 which represent the highest runtimes. To be more precise, the number of alignments that contain choice operators is the factor that increases runtimes the most as can be seen in line 1 and 3 where the ratio of the number of choice alignments to the number of alignments is high. Also, the difference between lines 1,3 and 6 is that in line 6, there was mostly trees containing only one choice operator which decreased runtime. The choice operator reveals to be the most expensive in runtimes; not only because of the alignment but also because of the computation of its language during the assessment of the precision metric. In terms of expensiveness, comes next the loop operator whose impact is evaluated in the next section.

The previous findings are confirmed in Table 6.3. Since the number of evaluated trees stays the same (which means the same trees were explored), then the same frequent trees are present in each portion of the log. The number of activities is also constant. In this case, the only impacting factor is the size of the log and the size of the traces. The size of the traces follows the same distribution as shown by the quartiles. Consequently, the runtime increases with the size of the log.

Impact of repetitive events

We call repetitive events those that occur more than one time in a trace and may create a loop pattern. Table 6.4 gives the runtimes of COBPAM for the feasible logs while leaving only the first occurrence of a repetitive event in the trace. For each log we give **R1:** the ratio of the number of removed events to the global number of events in the log. We also give **R2:** the mean ratio of the number of removed events to the size of the trace.

We notice that there is a reduction in runtime when the loop triggering events are removed. This is due to the overhead of computing an alignment with a loop involving backtracking and to the call to the function that computes the language of a process tree which is particularly consuming in the case of a tree containing a loop operator.

Table 6.4: Execution times of COBPAM on logs where repetitive events were removed

	Sepsis	Traffic Fines	WABO	BPI_2019S1
Runtime	14s	3m	6s	6.3mn
R1	37%	1%	3%	18%
R2	52%	1%	2%	28%
Relative runtime	28%	100%	75%	69%

Table 6.5: Pattern statistics of LPM Discovery and COBPAM

	COBPAM	**LPM Discovery**		
		Relevant	Non-Maximal	Non-Compact
Sepsis	375	125	17	194
WABO	33	21	5	235

6.2.2 Quantitative Effectiveness

Now, we assess the relevance of patterns discovered by our algorithm and LPM discovery. While COBPAM guarantees that discovered patterns are compact and maximal, we check how many patterns derived by LPM discovery also satisfy these properties. Given the above execution times (Table 6.4), we focus on the two feasible logs: Sepsis and WABO.

The results are summarized in Table 6.5. For instance, for the Sepsis log, among the 500 patterns mined by the LPM discovery, 336 patterns satisfy the support and precision thresholds set by COBPAM. Among these, 194 are not compact and 17 are not maximal. So, only 125 of the patterns derived by LPM discovery are maximal and compact, whereas COBPAM discovered 375 such patterns. Similar results are obtained for the other dataset. We conclude that the patterns derived by LPM discovery contain much redundant information, whereas COBPAM yields many more relevant patterns.

6.2.3 Qualitative Effectiveness

Next, we conducted a qualitative analysis on the patterns derived by COBPAM and LPM discovery as well as on the episodes extracted by ProM's Episode Miner and the rules discovered by Declare Miner. All algorithms were executed on Sepsis. Episode and Declare Miner were chosen because they represent alternative methods for dealing with flexible processes (see Section 2.2).

Sepsis is well-suited for our analysis as its end-to-end process model obtained with the FHM algorithm shows that the process is highly unstructured with intertwined execution paths, a high number of choice and loop constructs, and many edges.

COBPAM versus LPM Discovery

In Fig. 6.1, we show some patterns derived by LPM discovery that satisfy the thresholds set to COBPAM. Patterns found by COBPAM are shown in Fig. 6.2. We notice the difference in the trees derived by the two algorithms. For instance, the tree Fig. 6.1a was not extracted by COBPAM, because it is not maximal. In fact, it is contained in tree Fig. 6.2a. Knowing that Fig. 6.2a is frequent, one knows that *"ER Registration"* followed by *"CRP"* followed by *"Leucocytes"* is frequent. Hence, it follows that *"CRP"* followed by *"Leucocytes"*, as in tree Fig. 6.1a, is frequent too.

The trees Fig. 6.1c, Fig. 6.1d, and Fig. 6.1e are not discovered by COBPAM either, as they are not compact. Trees Fig. 6.1c and Fig. 6.1d are obtained from Fig. 6.1a by replacing activity *"CRP"* by a choice that includes activity *"CRP"* and some other behavior. Then, knowing that tree Fig. 6.1a is frequent, one concludes that the trees Fig. 6.1c and Fig. 6.1d are also frequent. Hence, they do not give new information. In fact, we do not know if the path *seq(ER Registration, Leucocytes)* is frequent in Fig. 6.1c or even if it exists in the log. Similarly, tree Fig. 6.1e can be constructed from tree Fig. 6.1b. The construction and evaluation (support, language fitness, coverage, determinism and confidence) of such trees lead to execution time being wasted without gaining further information about the process.

COBPAM versus Episode Miner

Now that we have compared our method to the closest concurrent in flexible processes oriented process discovery, we turn to less similar methods. First, we'll take a look at Episode Miner.

Using Episode Miner, we obtained the partially ordered sets of activities shown in Fig. 6.3. The episodes Fig. 6.3b and Fig. 6.3c can be explained by the episode Fig. 6.3d where the activity *"ER Sepsis Triage"* follows *"ER Triage"* which follows *"ER Registration"*. Indeed, the two episodes contain partial orders that can be obtained by transitivity from the fourth episode. Moreover, with respect to Fig. 6.3b which suggests a parallel execution (or absence of order) after *ER Registration*, the parallelism cannot be confirmed as it could be just a result of Fig. 6.3d. No other episode discovered shows the second ordering *ER Triage* follows *ER Sepsis Triage*. That does not mean it is nonexistent, it just means it is not frequent. COBPAM, on the other hand, is able to confirm the parallelism as demonstrated in Fig. 6.2c thanks to the precision metric which is equal to 1.

Moreover, Fig. 6.3a can be interpreted as an execution of *ER Registration* followed by the concurrence of *CRP* and *Leucocytes*. Neither of the orderings of the parallel activities appear in other episodes. Meaning, neither of them are frequent. From the existence of Fig. 6.3a, we conclude that both orderings appear in the log without being frequent. Here, the parallelism can be deduced from the Episode Miner.

Finally we recall that more complex structures like loops and exclusive choices cannot be discovered by Episode Miner.

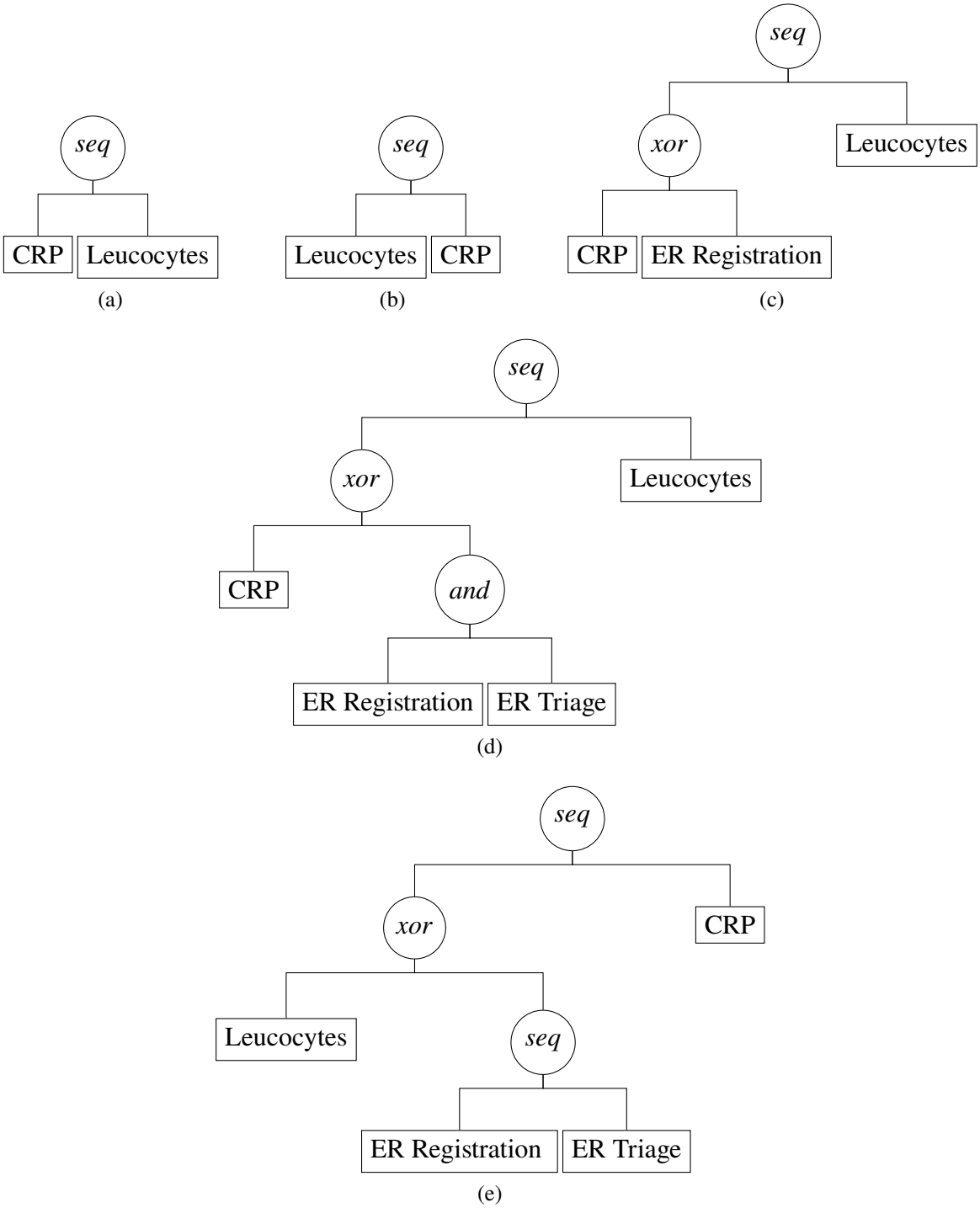


Figure 6.1: Behavioral Patterns mined by LPM discovery.

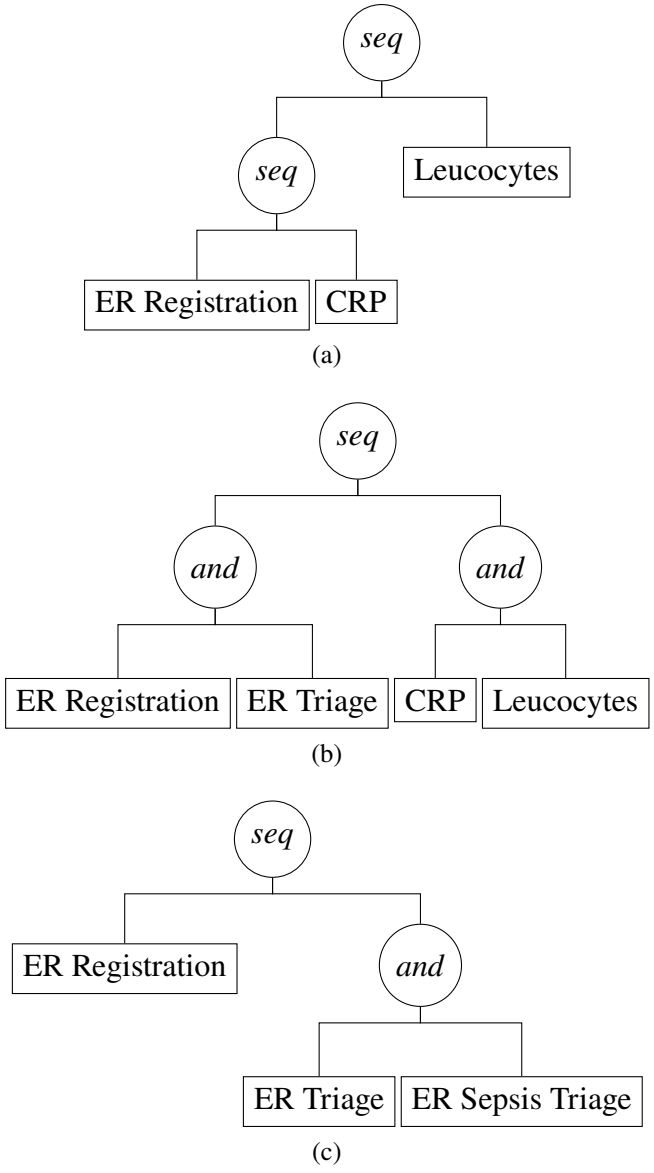


Figure 6.2: Behavioral Patterns mined with COBPAM.

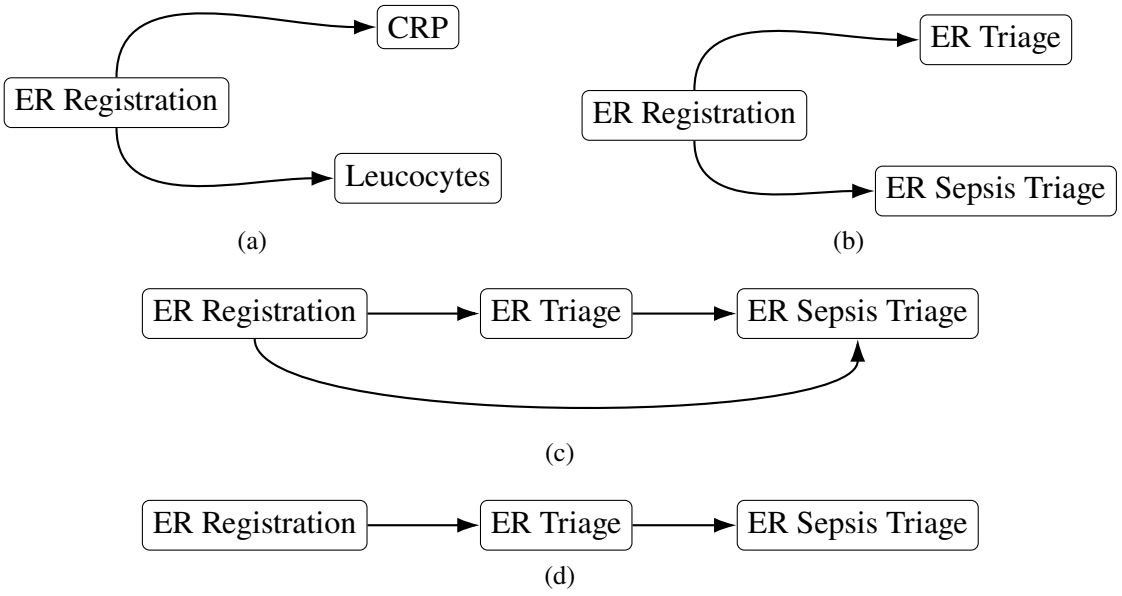


Figure 6.3: Episodes mined with PROM's Episode Miner

COBPAM versus Declare Miner

Contrary to behavioral trees and episodes, the Declare Miner generates a set of rules. These rules constitute a declarative process model representing the log in the sense that any behavior that is not explicitly prohibited by them can be part of the underlying process model. It is process *defining* where as the behavioral patterns and episodes are process *descriptive*. Indeed, the latter does not aim at defining the process model governing the log, whether it is by imperatively specifying the behavior to execute or by specifying a set of rules to not transgress.

In order to explain further our point, let's take the example where Declare Miner generated a rule *response(ER Registration, ER Triage)* using a support threshold of 0.7 while leaving the other parameters untouched. This means that the Declare Miner proposes this rule as one governing the execution of the process. Every trace should comply to this constraint. The remaining 30% of the traces are considered noise by the algorithm.

Now, if we consider a tree *seq(ER Registration, ER Triage)* returned by COBPAM using the same support threshold, the behavior is considered frequent if it exceeds a 70% occurrence. However, the remaining traces are valid altogether. They just don't contain the behavior. The tree *is not* a *rule* to blindly follow. There is a semantics difference.

Keeping this essential difference in mind, we will compare some of the rules generated with the COBPAM patterns. We use the same parameters as above.

The previous rule, *response(ER Registration, ER Triage)* translates as: if *ER Registration* appears in the trace, then *ER Triage* will eventually follow. As it can be noted, the rules are formulated in Linear Temporal Logic; meaning, the previous rule holds true when *ER Registration* is absent in

the trace. This is the first difference with behavioral pattern discovery. The frequency of the rule does not imply anything on the frequency of the activities mentioned. Hence, the monotonocity property is not valid. Consequently, if the rule is discovered, we cannot conclude on the existence of the pattern $P = seq(ER\ Registration,\ ER\ Triage)$. For example, by considering the same threshold for COBPAM, if the tree P has a threshold of 60% and the *ER Registration* does not appear in at least 10% of the traces, then the pattern is deemed non frequent while the previous rule is declared valid.

Another obvious difference is the number of activities present in each rule which is limited to two. Behavioral pattern mining however does not impose such a condition although a low number is recommended for facilitating analysis and ensuring feasible runtimes. On that matter, the low number of activities in the rules allows the algorithm to generate and then test negative constraints. For example, the *not co-existence(A, B)* means that A and B never appear together in a trace. It comes as a counterpart to the rule *co-existence(A,B)* which means if either A or B appear in the trace, then the other activity must appear too. Considering the complexity of behavioral trees, the generation of negative patterns is generally unfeasible. Note though that TB-Declare [31] extends the number of activities in the rules but there is still key differences with our behavioral patterns as detailed in Section 2.2.

6.3 Data-aware Analysis Framework Evaluation

This section presents an evaluation of our techniques for the discovery and analysis of contextual patterns as well as for the discovery of behavioral rules.

6.3.1 Efficiency

We start by evaluating the influence of the introduction of contexts on the baseline algorithm COBPAM. Note that the Data-aware Analysis Framework can use either COBPAM or ACOBPAM as its building block to uncover behavioral patterns in the atomic contexts. The results presented in this section are valid for either algorithms.

Table 6.6 reports a break-down of the runtimes of the discovery of contextual patterns with CCOBPAM . For each log, we first list the runtime without incorporating context (0-attribute, corresponds to COBPAM on the whole log). Then, a 1-attribute experiment considers solely the first attribute mentioned for each event log in Section 6.1 while the 1-attributeS experiment considers the second attribute mentioned. The 2-attribute experiments incorporated two attributes. The table also includes information on the number of atomic context, given in brackets after the runtime. This number of atomic contexts depends on the number of attributes considered and their domain.

First, we can observe that extending COBPAM to take into account contexts increases execution time. Second, the change in runtime introduced by adding a new attribute can be explained by two opposite forces. Contexts are smaller when increasing the number of attributes, which, in general,

Table 6.6: Runtimes of CCOBPAM for different context definitions, including the numbers of atomic contexts

	Sepsis	Traffic Fines	WABO	BPI_2019S1
0-attribute	49s (0)	3m (0)	8s (0)	3.4mn (0)
1-attribute	93s (2)	4.4h (49)	10s (1)	57mn (4)
1-attributeS	94s (2)	6.5mn (3)	12s (3)	5.3mn (2)
2-attribute	2m (3)	3.5h (66)	13s (3)	57mn (4)

Table 6.7: Contexts statistics for WABO

	Size	**C-exclusive**	**AC-frequent**	**Rules**
[General, Internet]	1211	/	8	9
[General, Post]	53	/	2	19
[General, desk]	105	/	2	9
[General, *]	1369	30	/	19
[*, *]	1369	30	/	19

reduces the required runtime. On the other hand, when increasing the number of attributes, i.e., when increasing the number of atomic contexts, new construction graphs, along with the specific set of frequent trees they contain, are observed in each additional context. In some cases, these construction graphs are bigger or contain more frequent and/or complex trees and need to be explored, which requires additional runtime. As can be seen in Table 6.6, both of these effects have varying impact, depending on the considered event log. The scale of the change in runtimes depends on the number of atomic contexts added.

The previous statements are corroborated by the scalability study in Section 6.2.1. Indeed, since the contexts are subsets of the contextual event log, they can be assimilated to the samples we defined in Table 6.2 and Table 6.3. Our analysis showed great variability in execution times with respect to the size of the samples/potential atomic contexts. The factors are discussed in the same section.

6.3.2 Effectiveness of Pattern Discovery

Next, we conducted a quantitative analysis on the patterns returned by CCOBPAM for the WABO and BPI_2019S1 logs. For the other logs, without further domain knowledge, the number of possible contexts and patterns turned out to be overwhelming. Our observations are summarized in Table 6.7 and Table 6.8. First, we analyze the contexts constructed with respect to the minimal context size in WABO. There are three atomic contexts, as shown in the hierarchy in Table 6.7. Overall, 30 patterns are C-exclusive in the root context. Some patterns are neither C-exclusive nor C-general. They are AC-frequent and their number is indicated in the corresponding column. Concerning BPI_2019S1 log, there are 4 atomic contexts and only one pattern is C-exclusive. All the other 319 patterns discovered are AC-frequent.

Table 6.8: Contexts statistics for BPI_2019S1

	Size	C-exclusive	AC-frequent	Rules
[Consignment, comp.ID_0000]	4331	/	0	0
[inv. before GR, comp.ID_0000]	66318	/	41	20
[inv. after GR, comp.ID_0000]	4565	/	279	10
[2-way match, comp.ID_0003]	304	/	2	1
[Consignment, *]	4331	/	/	0
[invoice before GR, *]	66318	/	/	0
[*, companyID_0003]	304	/	/	0
[2-way match, *]	1369	/	/	0
[*, companyID_0000]	75214	1	/	0
[invoice after GR, *]	4565	/	/	0
[*, *]	75518	/	/	0

6.3.3 Patterns Analysis

Causal Relations between Data and Patterns

For the WABO event log, causal relations between context data and patterns, see Section 4.3, could not be found due to the peculiar hierarchy of contexts (the counterfactual of the constrained attribute in the unique context that can hold C-exclusive patterns is only present in discarded small-sized contexts). We, therefore, considered the Sepsis event log and explored causality of context data and patterns. Based on the two Boolean attributes 'InfectionSuspected' and 'Infusion', three contexts were constructed: *[true, true]*, *[true, false]* and *[false, false]*. The total number of patterns discovered was 968 and 77 of those were C-exclusive. We found causal relations for 35 of the C-exclusive patterns. Specifically, they were caused by the exposure variable *InfectionSuspected = true*, highlighting that a suspected infection can be seen as the root cause of various behavioral regularities like *seq(seq(ER Triage, Leucocytes), CRP)*. As for BPI_2019S1, the only C-exclusive pattern discovered did not show any causal dependencies while no C-exclusive patterns were accounted for in Traffic Fines.

Interplay of Behavioral Patterns (Rules)

The column *Rules* in Table 6.7 and Table 6.8 indicates how many association rules that describe the interplay of the discovered patterns were found in each context in WABO log and BPI_2019S1 respectively, see Section 4.1. Specifically, the number represents the number of patterns that hold a rule. An example of those patterns for WABO is Tree (1) in Fig. 6.4 which holds a rule in all contexts of the log. In fact, there is a strong dependency between *seq(T02, T04)* and *seq(T06, T10)* (child subtrees of the root) in all contexts, in which the tree is frequent. Note, however, that the presence of a rule in a context does not mean that the same rule is present also in a more general context. The opposite does not hold either. The existence of a rule is independent between contexts. This is known as the Simpson's Paradox [60]. We give the example of a process $P = seq(P_1, P_2)$, with P_1, P_2, two subtrees which is C-General in [low, *] while holding a rule in that context (as shown in Table 6.9). The pattern, however, does not satisfy a rule in the more specific contexts [low, <70] (see Table 6.10) and [low, 70+] (see Table 6.11). Discovering the occurrences of the Simpson's Paradox is another upside of the contextual technique. Uncovering statistical association

Table 6.9: Presence of P_1 and P_2 in the traces of [low, *], the odds ratio being 7.

	P_2	$\overline{P_2}$
P_1	70	10
$\overline{P_1}$	10	10

Table 6.10: Presence of P_1 and P_2 in the traces of [low, <70], the odds ratio being 0.93.

	P_2	$\overline{P_2}$
P_1	28	6
$\overline{P_1}$	5	1

that is not verified when stratifying the data prevents the analyst from considering misleading interplays. Indeed, the strong association between P_1 and P_2 in [low, *] is spurious as neither the "<70" nor the "70+" population confirms it. It is to be noted that when variables are missing, there is a risk of taking fallacious interplays for real ones. This happens when stratifying the data on the missing variables brings opposite results on the behavioral rules.

Methodology in practice: Interpretation of Patterns

In this section, we apply our methodology to the patterns discovered. As the procedure needs to determine which patterns were discovered both by CCOBPAM (on the two attributes) and COBPAM (on the whole log) and in which contexts, manually, we could only apply it on WABO and BPI_2019S1 due to the limited number of patterns to inspect. For WABO, Table 6.12 shows that CCOBPAM discovered 40 contextual patterns, whereas COBPAM discovered 33 context-agnostic patterns. All of these were also discovered by CCOBPAM, so that they are listed in the *Common* column. Of the common patterns, 30 are C-exclusive and three AC-frequent. So, these three patterns are frequent in the log. Yet, they are frequent solely in some contexts, not in all of them. Seven patterns were revealed only by our novel CCOBPAM algorithm and are AC-frequent: They are frequent in some atomic contexts, but not in the whole log. Hence, they are missed by context-agnostic discovery. Table 6.13 shows on its turn that the only C-exclusive pattern discovered by CCOBPAM for BPI_2019S1 is not log-frequent. Moreover, among the 319 AC-frequent patterns returned, 28 were also log-frequent.

For a more concrete example on the contextual patterns analysis methodology, we illustrate in Fig. 6.4, a discovered pattern for each of the cases of pattern frequentiality presented in the end

Table 6.11: Presence of P_1 and P_2 in the traces of [low, 70+], the odds ratio being 0.6.

	P_2	$\overline{P_2}$
P_1	42	10
$\overline{P_1}$	7	1

Table 6.12: Pattern statistics of CCOBPAM and COBPAM for WABO

	CCOBPAM	**Common**
C-exclusive	/	30
AC-frequent	7	3

Table 6.13: Pattern statistics of CCOBPAM and COBPAM for BPI_2019S1

	CCOBPAM	**Common**
C-exclusive	1	0
AC-frequent	291	28

of Section 4.4. It is important to note that any tree discovered in WABO or BPI_2019S1 falls under one of the four categories we are going to exemplify. The trees we present here were selected randomly. Tree (1) is observed in all contexts. Since the tree is C-exclusive in the whole population, it is frequent regardless of the department and communication channel used in the building permit application process, thereby representing generic, recurring behavior. Tree (2), in turn, exemplifies a log-frequent behavior that occurs solely in two small atomic contexts, specific to applications handled by desks and through the post office. The total size of these contexts is only 158 traces, while the whole log contains 1434 traces. This indicates that the pattern is close to frequent in the other contexts suggesting a potential need to re-calibrating the support threshold.

Tree (3) is log-frequent and frequent in the largest atomic context concerning communication channel (i.e., the internet channel). This is another insight regarding the specific set of traces in which a pattern is frequent. Another interesting tree is (4), which is not discovered when neglecting contextual information, because it is only frequent in cases handled via the internet channel. As such, it exemplifies that context-aware discovery reveals patterns that are otherwise missed since they relate to a small set of traces.

6.4 Advanced COBPAM Evaluation

Our last experiments pertain to the evaluation of ACOBPAM. As the algorithm brings improvements on three aspects (runtimes, number of returned trees and difficulty of analysis), we evaluate each change separately. We start with the execution times.

6.4.1 Efficiency

Our aim is to evaluate the alignment growth method and assess how much it shortens runtimes. However, some of the modifications we introduced in Section 5.1 impacted the construction graph of each log. Indeed, some portions of the construction graph were no longer explored due to the update in the compactness property. Trees that were kept in the search before were deemed non compact with respect to the fourth compactness condition (see Section 5.1.4) and were

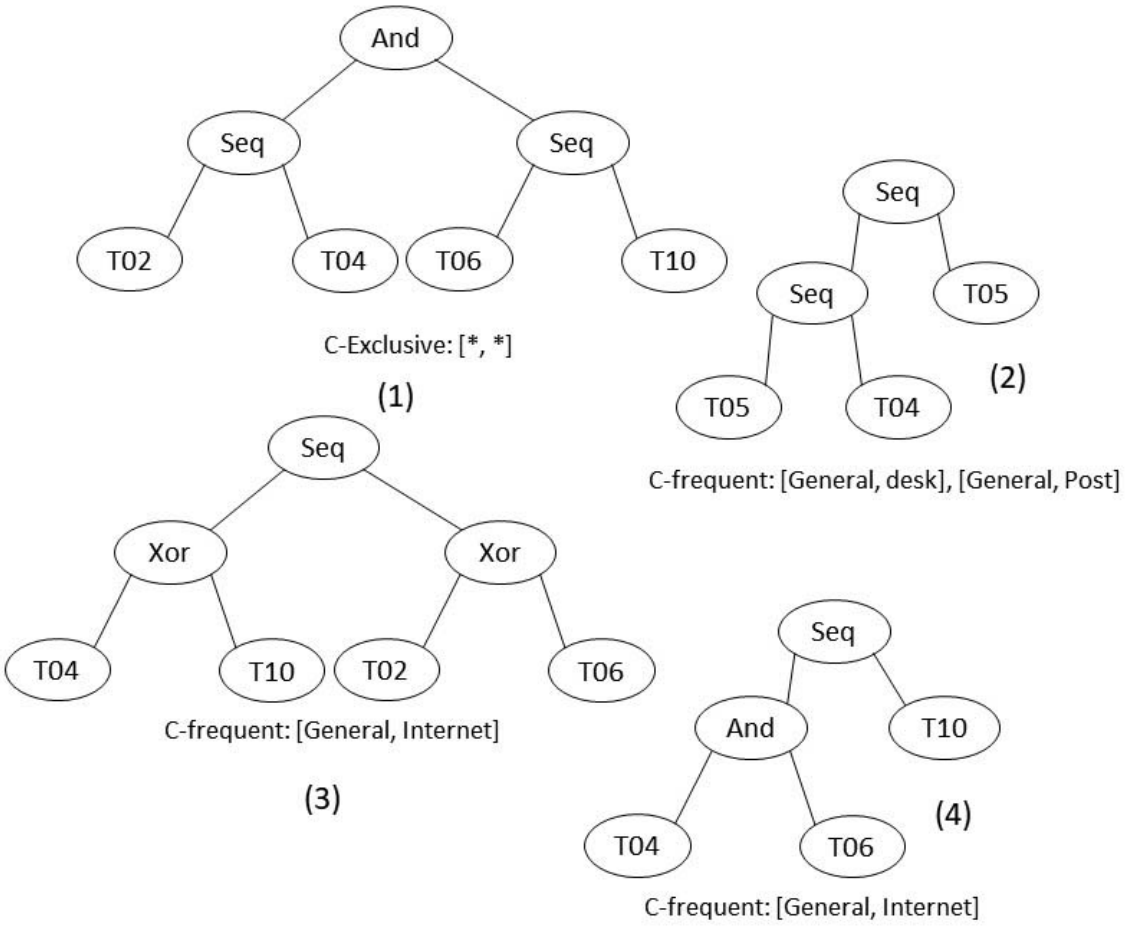


Figure 6.4: Illustrative examples of behavioral patterns mined with CCOBPAM

discarded. Moreover, other trees that were removed from the search in the previous version for not respecting the precision threshold are now preserved as we deleted this metric. Consequently, the number of trees evaluated and a fortiori returned changed. In these conditions, we cannot compare the classical alignment with the growth algorithm in isolation. For this reason, we echoed the changes we made in ACOBPAM into COBPAM. COBPAM now finds trees that are compact according to the latest definition and is not bound by a precision threshold. (In the remainder of the section, by COBPAM, we refer to the version incorporating the changes.) As a result, the same trees are evaluated and explored in both algorithms. The number of returned trees is equal too. Consequently, the only difference between the two tested algorithms is the method of alignment which makes it possible to directly compare them in a fair configuration.

Of course, in ACOBPAM only the time needed to return the initial set of trees is considered since that is the end point of COBPAM as well. The post-processing and visualization are orthogonal

and depend only on the returned set. The comparison results are presented in Table 6.14. For each algorithm and each log, there were two executions with a depth of two for the discovered trees at first and then with a depth of three. The gain in execution time with respect to the initial runtime is also given. For Traffic Fines, there was no reduction in runtimes since all the trees evaluated contained the *xor* operator which is treated classically. Yet, for the most time consuming logs, there was a highly substantial decrease, between 60% and 66%, in execution times, proving the efficiency of the alignment growth algorithm. However, the extraction of behavioral patterns for Sepsis under depth 3 was not possible.

Furthermore, for the most demanding logs, the gap in runtimes between COBPAM and ACOBPAM widens when we increase the depth. That is only logical because the new trees constructed of depth 3 are way more numerous than those of depth 2. The entire structure of each tree needs to be evaluated on each complete trace in COBPAM. There is an exponentiallity property on two aspects when moving from depth two to three in COBPAM: the number of trees to evaluate increases exponentially, but also the runtime of the alignment operation on the new trees (of depth three) increases exponentially. That is because the classical alignment algorithm is hard with respect to the complexity/size of the tree. On the contrary, the exponentiallity in ACOBPAM still applies on the number of new trees to evaluate but is less strong with respect to the alignment as, mostly, the alignments are grown incrementally.

Yet, for BPI_2019S2, the gap didn't widen but shortened on the contrary. We thoroughly investigated this behavior and we observed that this particular log consumed a lot of memory. The garbage collector which is a java program that frees unused space to increase the memory available was called a lot. As such, more time was dedicated to freeing memory than to executing the algorithm. Finally, the supposed gain in runtime thanks to the alignment growth method was spoiled by the garbage collector interventions.

Table 6.14: Execution times of COBPAM and ACOBPAM (for a depth parameter of two and three)

		Sepsis	Traffic Fines	WABO	BPI_2019S1	BPI_2019S2
Depth 2	COBPAM	34s	13s	2s	2.45mn	5.8mn
	ACOBPAM	12s	13s	1s	59s	2mn
	Runtime decrease	65%	0%	50%	60%	66%
Depth 3	COBPAM	>24h	13s	3s	5.81mn	14.9mn
	ACOBPAM	>24h	13s	2s	98s	6.9mn
	Runtime decrease	N/A	0%	33%	72%	49%

To sum up, we give in Table 6.15 the decrease in runtime observed with respect to the original algorithm of LPM Discovery. We used the results for ACOBPAM with depth 2 because the trees with this parameter contain at most four activities which corresponds to the number of activities composing the LPMs. We can see that the execution times of our algorithm are order of magnitude lower than the state of the art. Indeed, we divided the runtime of Sepsis by 100 and that of WABO

by 1000. Moreover, LPM was not able to return any results for three out of the five evaluated logs (Traffic Fines, BPI_2019S1 and BPI_2019S2).

Table 6.15: Runtime decrease with respect to the state-of-the-art algorithm for mining behavioral patterns (LPM)

	Sepsis	Traffic Fines	WABO	BPI_2019S1	BPI_2019S2
Runtime decrease	99%	N/A	99.9%	N/A	N/A

Next, we report in Table 6.16 the runtimes of the post-processing operation and visualization graph construction for both considered depths. The most time consuming part is the post-processing operation. Its runtime increases with respect to the initial number of trees returned. Indeed, there is a need of pairwise comparison to check equality of languages and existence of generalized maximality seeds. This can be verified by analyzing Table 6.17. On another hand, the computation of the language, the detection of the loop seeds and the detection of the alternative seeds are all computationally hard with respect to the depth. That is because the tree gets exponentially more complex and the number of its subtrees (which are used in the definition of alternative seeds) grows exponentially too when it gets deeper. This last statement is confirmed in the depth 3 runtime of BPI_2019S2 seen in Table 6.16. It is the time needed for the pairwise comparison of the 79 trees returned. However, its value: 1.8mn is almost equivalent to the time needed to compare 597 trees for Sepsis under depth 2 (1.95mn).

Table 6.16: Runtimes of the post-processing operation and visualization graph generation

	Sepsis	Traffic Fines	WABO	BPI_2019S1	BPI_2019S2
Depth 2	1.95mn	0s	3s	14s	19s
Depth 3	N/A	0s	1s	14s	1.8mn

6.4.2 Effectiveness

Quantitative Effectiveness

We give in Table 6.17 the number of trees returned before and after the application of post-processing for each considered depth along with the ratio between the initial number and the final one. Except for Traffic Fines which returned initially only one tree, the post-processing operation managed to reduce the number of trees returned by ACOBPAM and appearing during the visualization to between 35% to 73% their initial number. This demonstrates that the number of redundant trees in the first set is non negligible and that the generalized maximality with the equivalency study are indeed welcome and necessary. An other interesting observation is that the gain is higher in depth 3; that is because, each tree of such depth has a higher number of alternative/loop seeds and equivalent trees which are undoubtedly initially returned and consequently eliminated.

Table 6.17: Number of returned trees w/o Post-processing

		Sepsis	Traffic Fines	WABO	BPI_2019S1	BPI_2019S2
Depth 2	Without	597	1	32	36	37
	With	439	1	16	26	27
	Ratio	73%	100%	50%	72%	73%
Depth 3	Without	N/A	1	26	47	79
	With	N/A	1	9	26	31
	Ratio	N/A	100%	35%	55%	39%

Qualitative Effectiveness

We show in Fig. 6.5, a subset of the trees returned by COBPAM for WABO. It corresponds to the trees returned before the post-processing operation. Arguably, since trees Fig. 6.5a and Fig. 6.5c represent the alternative seeds of Fig. 6.5b and Fig. 6.5d respectively, only Fig. 6.5b and Fig. 6.5d were kept in the final set returned and visualized in ACOBPAM.

6.4.3 Visualisation

We give in Fig. 6.6 a global view of the generated visualization graph for WABO (support threshold of 0.7 and depth of 2) as well as a zoomed-in version in Fig. 6.7. The behavioral patterns are represented in their tree form inside the nodes of the graph. The node is circular and its size is proportional to the support of the pattern. The support can be displayed by hovering over it. There are four edge shapes and colors to depict each of the four relationships: *Follows, Inter-follows, Spans and Inter-spans relationships*. The legend, which also appears in the interface, is given in Fig. 6.8. Note that the edges are weighted with the supports of the relationships. Obviously, this graph contains only *Spans relationships*.

Moreover, the activities inside the trees are replaced by numbers to avoid cluttering and complexity which is the opposite of what we expect from the visualization graph. The legend for these numbers is also given in the interface.

6.5 Discussion

The experimental study we conducted has several limitations. We enumerate them in the following:

- **Unavailability of contextual logs:** The scarcity of real life logs containing contextual data prevented us from doing more extensive experiments to study the influence of contexts. Even when data attributes are present in some logs, they are not documented and their meaning stays obscure. An industrial use case would allow us to do a more in-depth analysis.
- **Missing values for context attributes:** In our experiments, traces whose context attributes have no values, have not been included in the discovery. In a real application, the missing values could be handled using methods known from databases in order to infer them.

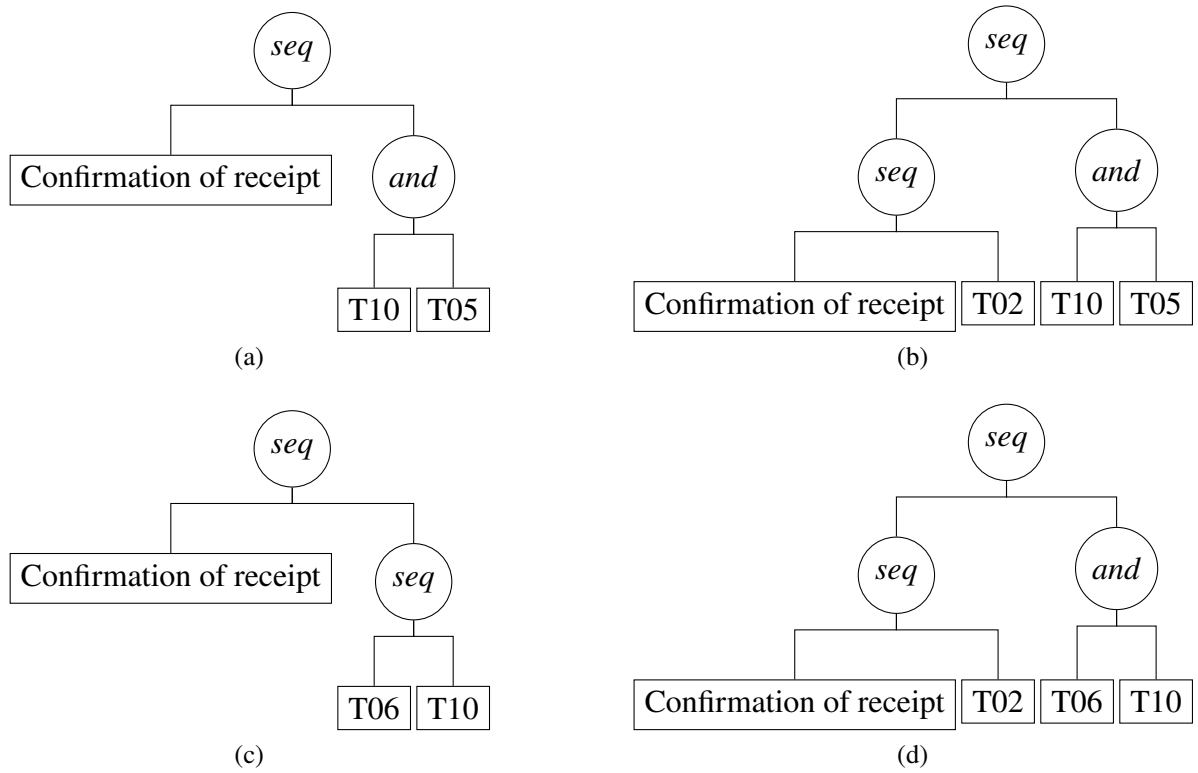


Figure 6.5: Behavioral Patterns mined by COBPAM for WABO

Alternatively, a Null value context could be defined and the discovered patterns in it could help in identifying causes of missing values.

- **User validation:** While the effectiveness evaluations showed that our methods give a compact, concise and relevant view of the behavior of a flexible process, a user-assisted evaluation should be conducted to conclude that business analysts find indeed a significant aid using our approaches.

6.6 Conclusion

As does any scientific work, our contributions needed validation, which was the objective of this chapter. We proved the feasibility of our algorithms through an extensive efficiency evaluation; not only in comparison to the state of the art but also with respect to certain parameters such as the size of the logs through a scalability study, the repetition ratio in the log, the number of considered attributes and the depth threshold. We also gave statistics about the results of our algorithms, making sure to highlight the relevance and effectiveness of our methods and the different insights they provide. Additionally, we made sure to give concrete and immediate examples of our results and methodologies and how they advance the understanding of flexible processes compared to the existing work. Lastly we discussed the limits of our experimentations. In the following and final chapter, we conclude our thesis and list future directions for research.

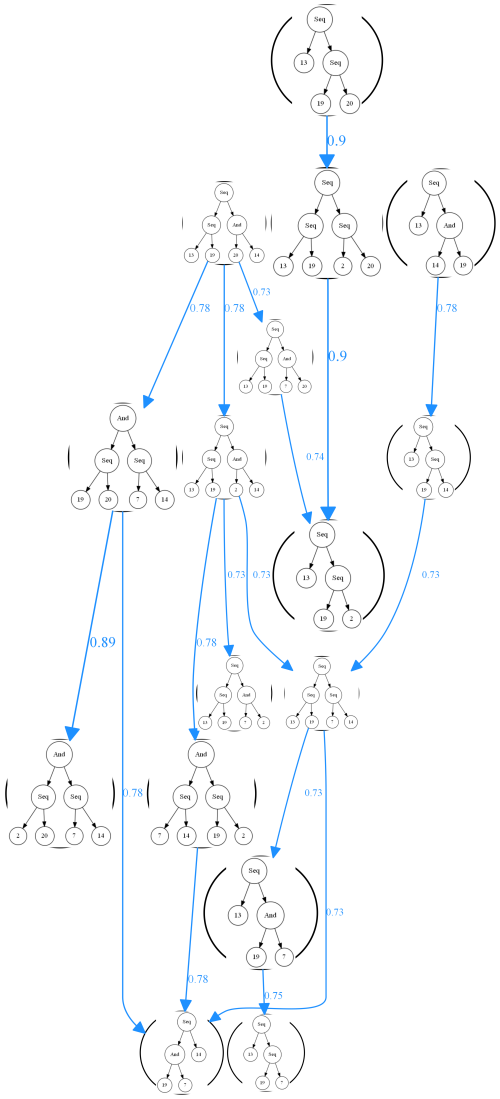


Figure 6.6: Visualisation graph of the patterns returned by WABO (support threshold of 0.7, depth of 2)[Global view]

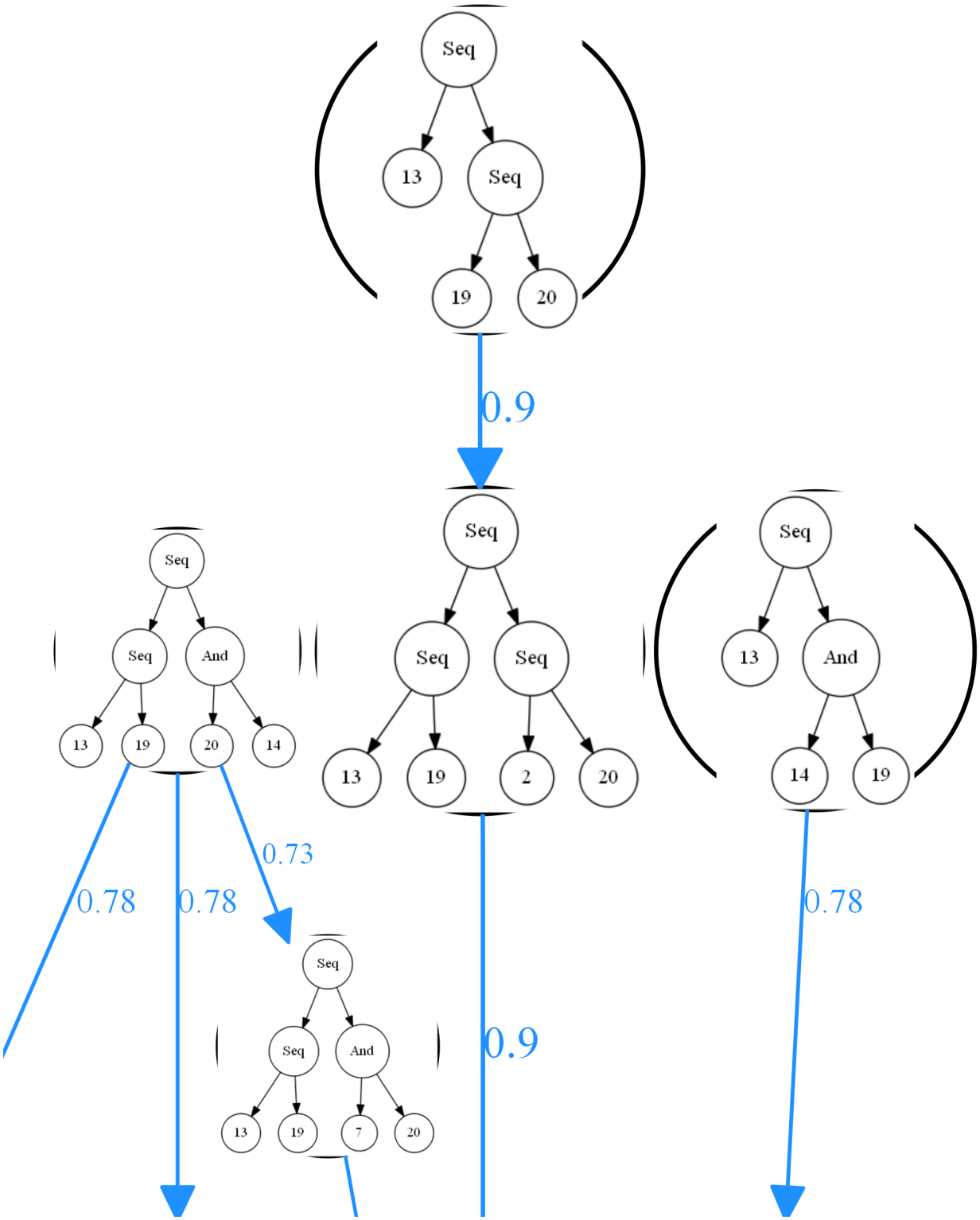


Figure 6.7: Visualisation graph of the patterns returned by WABO (support threshold of 0.7, depth of 2)[Zoomed-in view]

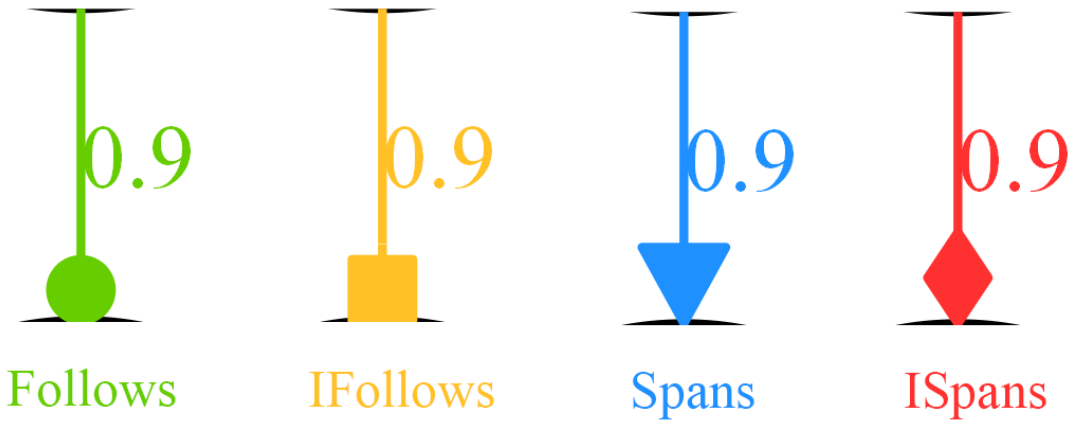


Figure 6.8: Legend for the relationships edges

7

Conclusion and Future Works

Process mining is about setting a bridge between process-agnostic data mining and evidence-lacking process management. Process discovery is a field where one strives to extract models and insights into the processes under execution. These processes can be found in healthcare, education, logistics, etc. One particular characteristic of interest is how structured they are; meaning, how much of a common general layout, do the process instances follow. In the absence of such commonalities between the instances, we are essentially dealing with what is coined spaghetti processes due to their multiple interleaving paths of execution. Out of their complexity, this type of processes poses a challenge for analysts as to grasp any useful information out of them.

In this thesis, we are interested in such flexible processes and we attempted to summarize them so that they become understandable and clear for analysis. In order to do that, we revisited the notion of behavioral patterns, small models representing recurrent commonalities between instances. We proposed an extraction algorithm, an extensive analysis framework which builds upon it and an advanced algorithm which improves on the first one. The contributions were as follows :

- A novel approach, named COBPAM, for discovering behavioral patterns yielding different techniques for optimization and guaranteeing interesting properties on the discovered patterns.
- A framework including a data-aware extraction of behavioral patterns nested in contexts. We characterized different properties with respect to contexts such as frequency, generality and exclusiveness. Finally, we studied correlations between patterns themselves and causal relations between the data and the occurrence of patterns. A methodology for using the framework along with interpretation guidelines are given.
- An advanced incremental algorithm, ACOBPAM, that enhances COBPAM in three aspects: it further reduces runtimes, decreases the number of returned trees and helps analyze the patterns through a visualization feature.

All our methods were evaluated through an extensive study of efficiency and effectiveness, both quantitative and qualitative, on real life event logs and proved superior to the state of the art in

terms of runtime and the pertinence of the discovered insights as well as in terms of ease of analysis.

Our work opens up many research avenues on the long term. A first perspective is the adaptation of the algorithms to handle large event logs; further than what was already accomplished with ACOBPAM. Indeed, right now, ACOBPAM is still unable to to guarantee scalability and fails to mine logs with a big number of events due to a high runtime and logs with a large number of activities since it runs on memory issues. CCOBPAM encounters the same difficulties and in particular memory insufficiency because all memory structures need to be continuously stored for each context. In order to solve these problems, distributed processing frameworks should be investigated. Note that this would also allow to discover deeper behavioral patterns.

In addition, there is opportunity to propose a tool for managing context discretization. For instance, considering an age attribute that we want to split into two intervals, an automatic procedure to choose the split boundary would be interesting. The goal being to maximize patterns relevancy in the resulting contexts. On the same subject, CCOBPAM uses only the contextual information associated to cases. The data dimension associated to events is also rich in insights and should be included in future extensions. Moreover, speaking of dimensions, we plan to extract patterns from the resource perspective. Patterns would hence describe frequent behavior between the agents enacting the events themselves.

Orthogonally, in the spirit of further facilitating the exploration of the discovered patterns, filtering and ranking features should be introduced using constraints and utility measures.

Finally, since behavioral patterns are in a way a summary of the underlying process, one could use them as features in various mining and machine learning tasks. Particularly, they can be leveraged for trace clustering, quality of service and performance prediction, outcome prediction, etc. Moreover, the behavioral rules discovered could be used in recommendation systems either in a static or streaming setting.

Bibliography

[1] 25+ Impressive Big Data Statistics for 2021. https://techjury.net/stats-about/big-data-statistics/[Accessed: 2021-07-15].

[2] Here's What Happens Every Minute on the Internet in 2020.

[3] Bayesian network approach to making inferences in causal maps. *European Journal of Operational Research*, 128(3):479–498, feb 2001.

[4] A novel heuristic method for improving the fitness of mined business process models. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9936 LNCS, pages 537–546. Springer Verlag, oct 2016.

[5] M. Acheli, D. Grigori, and M. Weidlich. Efficient discovery of compact maximal behavioral patterns from event logs. In *International Conference on Advanced Information Systems Engineering*, pages 579–594. Springer, 2019.

[6] M. Acheli, D. Grigori, and M. Weidlich. Discovering and analyzing contextual behavioral patterns from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 2021.

[7] A. Adriansyah. *Aligning Observed and Modeled Behavior*. PhD thesis, 2014.

[8] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *Proc. of 20th International Conference on Very Large Data Bases, {VLDB'94}*, 1994.

[9] T. Baier and J. Mendling. Bridging abstraction layers in process mining by automated matching of events and activities. In *Business process management*, pages 17–32. Springer, 2013.

[10] J. H. Bjørngaard, A. T. Nordestgaard, A. E. Taylor, J. L. Treur, M. E. Gabrielsen, M. R. Munafò, B. G. Nordestgaard, B. O. Åsvold, P. Romundstad, and G. Davey Smith. Heavier

smoking increases coffee consumption: findings from a Mendelian randomization analysis. *International Journal of Epidemiology*, 46(6):1958–1967, 08 2017.

[11] A. Bolt and W. M. Van Der Aalst. Multidimensional process mining using process cubes. In *Lecture Notes in Business Information Processing*, volume 214, pages 102–116. Springer Verlag, 2015.

[12] D. Borrego and I. Barba. Conformance checking and diagnosis for declarative business process models in data-aware scenarios. *Expert Systems with Applications*, 41(11):5340–5352, 2014.

[13] R. J. C. Bose and W. M. Van der Aalst. Context aware trace clustering: Towards improving process mining results. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 401–412. SIAM, 2009.

[14] R. P. C. Bose, F. M. Maggi, and W. M. Van Der Aalst. Enhancing declare maps based on event correlations. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8094 LNCS, pages 97–112. Springer, Berlin, Heidelberg, 2013.

[15] R. P. C. Bose and W. M. Van Der Aalst. Trace clustering based on conserved patterns: Towards achieving better process models. In *LNBIP*, 2010.

[16] J. C. Buijs, B. F. Van Dongen, and W. M. Van Der Aalst. A genetic algorithm for discovering process trees. In *CEC 2012*, pages 1–8. IEEE, 6 2012.

[17] A. Burattin and J. Carmona. A framework for online conformance checking. In *International Conference on Business Process Management*, pages 165–177. Springer, 2017.

[18] A. Burattin and A. Sperduti. Heuristics miner for time intervals. In *Proceedings of the 18th European Symposium on Artificial Neural Networks - Computational Intelligence and Machine Learning, ESANN 2010*, 2010.

[19] A. Burattin, A. Sperduti, and W. M. van der Aalst. Heuristics miners for streaming event data. *arXiv preprint arXiv:1212.6383*, 2012.

[20] A. Burattin, S. J. van Zelst, A. Armas-Cervantes, B. F. van Dongen, and J. Carmona. Online conformance checking using behavioural patterns. In *International Conference on Business Process Management*, pages 250–267. Springer, 2018.

[21] J. Carmona, B. van Dongen, A. Solti, and M. Weidlich. *Conformance checking*. Springer, 2018.

[22] G. F. Cooper. A simple constraint-based algorithm for efficiently mining observational databases for causal relationships. *Data Mining and Knowledge Discovery*, 1(2):203–224, 1997.

[23] M. De Leoni, M. Dumas, and L. García-Bañuelos. Discovering branching conditions from business process execution logs. In *Lecture Notes in Computer Science (including subseries*

Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), volume 7793 LNCS, pages 114–129. Springer, Berlin, Heidelberg, 2013.

[24] M. De Leoni, F. M. Maggi, and W. M. van der Aalst. Aligning event logs and declarative process models for conformance checking. In *International Conference on Business Process Management*, pages 82–97. Springer, 2012.

[25] M. De Leoni and W. M. Van Der Aalst. Aligning event logs and process models for multi-perspective conformance checking: An approach based on integer linear programming. In *Business Process Management*, pages 113–129. Springer, 2013.

[26] M. De Leoni, W. M. Van Der Aalst, and B. F. Van Dongen. Data-and resource-aware conformance checking of business processes. In *International Conference on Business Information Systems*, pages 48–59. Springer, 2012.

[27] A. K. De Medeiros, W. M. Van Der Aalst, and A. J. Weijters. Workflow Mining: Current Status and Future Directions. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2888:389–406, 2003.

[28] A. K. A. de Medeiros, A. J. M. M. Weijters, and W. M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 4 2007.

[29] P. Delias, D. Grigori, M. L. Mouhoub, and A. Tsoukias. Discovering characteristics that affect process control flow. In *Decision Support Systems IV-Information and Knowledge Management in Decision Processes*, pages 51–63. Springer, 2014.

[30] P. Delias, A. Lagopoulos, G. Tsoumakas, and D. Grigori. Using multi-target feature evaluation to discover factors that affect business process behavior. *Computers in Industry*, 99:253–261, 2018.

[31] C. Di Ciccio, F. M. Maggi, and J. Mendling. Efficient discovery of Target-Branched Declare constraints. *Information Systems*, 56:258–283, mar 2016.

[32] C. Di Francescomarino, C. Ghidini, F. M. Maggi, and F. Milani. Predictive Process Monitoring Methods: Which One Suits Me Best? In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11080 LNCS, pages 462–479. Springer Verlag, sep 2018.

[33] C. Diamantini, L. Genga, and D. Potena. Behavioral process mining for unstructured processes. *Journal of Intelligent Information Systems*, 47(1):5–32, 8 2016.

[34] S. Dunzer, M. Stierle, M. Matzner, and S. Baier. Conformance checking: a state-of-the-art literature review. In *Proceedings of the 11th international conference on subject-oriented business process management*, pages 1–10, 2019.

[35] D. Fahland and W. M. van Der Aalst. Model repair—aligning process models to reality. *Information Systems*, 47:220–243, 2015.

[36] P. Fournier-Viger, J. Chun, W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A Survey of Sequential Pattern Mining. *Ubiquitous International*, 1(1):54–77, 2017.

[37] P. Fournier-Viger, J. C.-W. Lin, B. Vo, T. T. Chi, J. Zhang, and H. B. Le. A survey of itemset mining. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 7(4):e1207, jul 2017.

[38] G. Greco, A. Guzzo, L. Pontieri, and D. Saccà. Discovering expressive process models by clustering log traces. *IEEE TKDE*, 2006.

[39] C. W. Günther and W. M. Van Der Aalst. Fuzzy mining–adaptive process simplification based on multi-perspective metrics. In *International conference on business process management*, pages 328–343. Springer, 2007.

[40] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8(1):53–87, jan 2004.

[41] B. Hanson and B. Kopjar. Clinical studies in spinal surgery. *European Spine Journal*, 14(8):721–725, 2005.

[42] D. Heckerman. A bayesian approach to learning causal networks. *arXiv preprint arXiv:1302.4958*, 2013.

[43] K. Johnson. *A descriptive process model for open-source software development*. Graduate Studies, 2001.

[44] M. Koorneef, A. Solti, H. Leopold, and H. A. Reijers. Automatic root cause identification using most probable alignments. In *International Conference on Business Process Management*, pages 204–215. Springer, 2017.

[45] M. Leemans and W. M. van der Aalst. Discovery of frequent episodes in event logs. In *Lecture Notes in Business Information Processing*, volume 237, pages 1–31. Springer, Cham, 11 2015.

[46] S. J. Leemans, D. Fahland, and W. M. van der Aalst. Discovering block-structured process models from event logs-a constructive approach. In *International conference on applications and theory of Petri nets and concurrency*, pages 311–329. Springer, 2013.

[47] S. J. Leemans, D. Fahland, and W. M. van der Aalst. Discovering block-structured process models from event logs containing infrequent behaviour. In *International conference on business process management*, pages 66–78. Springer, 2013.

[48] S. J. Leemans, D. Fahland, and W. M. Van Der Aalst. Discovering block-structured process models from incomplete event logs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8489 LNCS, pages 91–110. Springer Verlag, 2014.

[49] S. J. Leemans, D. Fahland, and W. M. van der Aalst. Scalable process discovery and conformance checking. *Software and Systems Modeling*, 17(2):599–631, may 2018.

[50] V. Leno, M. Dumas, F. M. Maggi, M. La Rosa, and A. Polyvyanyy. Automated discovery of declarative process models with correlated data conditions. *Information Systems*, 89:101482, mar 2020.

[51] J. Li, T. D. Le, L. Liu, J. Liu, Z. Jin, and B. Sun. Mining causal association rules. In *Proceedings - IEEE 13th International Conference on Data Mining Workshops, ICDMW 2013*, pages 114–123. IEEE, dec 2013.

[52] D. Lorenzoli, L. Mariani, and M. Pezzè. Automatic generation of software behavioral models. In *Proceedings - International Conference on Software Engineering*, pages 501–510, New York, New York, USA, 2008. ACM Press.

[53] F. M. Maggi, M. Dumas, L. García-Bañuelos, and M. Montali. Discovering data-aware declarative process models from event logs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 8094 LNCS, pages 81–96. Springer, Berlin, Heidelberg, 2013.

[54] F. M. Maggi, A. J. Mooij, and W. M. Van Der Aalst. User-guided discovery of declarative process models. In *CIDM 2011*, pages 192–199. IEEE, 4 2011.

[55] F. Mannhardt, M. De Leoni, H. A. Reijers, and W. M. Van Der Aalst. Balanced multi-perspective checking of process conformance. *Computing*, 98(4):407–437, 2016.

[56] F. Mannhardt, M. De Leoni, H. A. Reijers, and W. M. Van Der Aalst. Data-driven process discovery - Revealing conditional infrequent behavior from event logs. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 10253 LNCS, pages 545–560, 2017.

[57] F. Mannhardt, M. De Leoni, H. A. Reijers, W. M. Van Der Aalst, and P. J. Toussaint. From low-level events to activities-a pattern-based approach. In *International conference on business process management*, pages 125–141. Springer, 2016.

[58] J. Munoz-Gama, J. Carmona, and W. M. Van Der Aalst. Single-entry single-exit decomposed conformance checking. *Information Systems*, 46:102–122, 2014.

[59] L. G. Neuberg. CAUSALITY: MODELS, REASONING, AND INFERENCE, by Judea Pearl, Cambridge University Press, 2000. *Econometric Theory*, 19(04):675–685, aug 2003.

[60] J. Pearl et al. Models, reasoning and inference. *Cambridge, UK: CambridgeUniversityPress*, 2000.

[61] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M. C. Hsu. Mining sequential patterns by pattern-growth: The prefixspan approach. *IEEE Transactions on Knowledge and Data Engineering*, 2004.

[62] R. N. Proctor. The history of the discovery of the cigarette–lung cancer link: evidentiary traditions, corporate denial, global toll. *Tobacco control*, 21(2):87–91, 2012.

[63] J. Rabatel, S. Bringay, and P. Poncelet. Mining sequential patterns: a context-aware approach. In *Advances in Knowledge Discovery and Management*, pages 23–41. Springer, 2013.

[64] D. Reinsel, J. Gantz, and J. Rydning. The Digitization of the World From Edge to Core. Technical report, 2018.

[65] W. Reisig and Wolfgang. *Petri nets : an introduction*. Springer Berlin Heidelberg, 1985.

[66] A. L. Rosner. Evidence-based medicine: revisiting the pyramid of priorities. *Journal of Bodywork and Movement Therapies*, 16(1):42–49, 2012.

[67] A. Rozinat and W. M. Van Der Aalst. Decision mining in ProM. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 4102 LNCS, pages 420–425. Springer Verlag, 2006.

[68] S. Sakr. *Big Data 2.0 Processing Systems*. SpringerBriefs in Computer Science. Springer International Publishing, Cham, 2016.

[69] S. Schönig, C. Di Ciccio, F. M. Maggi, and J. Mendling. Discovery of multi-perspective declarative process models. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9936 LNCS, pages 87–103. Springer Verlag, oct 2016.

[70] A. Senderovich, C. Di Francescomarino, C. Ghidini, K. Jorbina, and F. M. Maggi. Intra and inter-case features in predictive process monitoring: A tale of two dimensions. In *International Conference on Business Process Management*, pages 306–323. Springer, 2017.

[71] R. Shraga, A. Gal, D. Schumacher, A. Senderovich, and M. Weidlich. Inductive context-aware process discovery. In *Proceedings - 2019 International Conference on Process Mining, ICPM 2019*, pages 33–40. Institute of Electrical and Electronics Engineers Inc., jun 2019.

[72] C. Silverstein, S. Brin, R. Motwani, and J. Ullman. Scalable techniques for mining causal structures. *Data Mining and Knowledge Discovery*, 4(2-3):163–192, 2000.

[73] M. Song, C. W. Günther, and W. M. Van Der Aalst. Trace clustering in process mining. In *Lecture Notes in Business Information Processing*, 2009.

[74] W. Song, X. Xia, H.-A. Jacobsen, P. Zhang, and H. Hu. Efficient alignment between event logs and process models. *IEEE Transactions on Services Computing*, 10(1):136–149, 2016.

[75] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *International Conference on Extending Database Technology*, pages 1–17. Springer, 1996.

[76] Y. Sun, B. Bauer, and M. Weidlich. Compound trace clustering to generate accurate and simple sub-process models. In E. M. Maximilien, A. Vallecillo, J. Wang, and M. Oriol, editors, *Service-Oriented Computing - 15th International Conference, ICSOC 2017, Malaga,*

Spain, November 13-16, 2017, Proceedings, volume 10601 of *Lecture Notes in Computer Science*, pages 175–190. Springer, 2017.

[77] P. N. Tan, V. Kumar, and J. Srivastava. Selecting the right objective measure for association analysis. In *Information Systems*, volume 29, pages 293–313. Pergamon, jun 2004.

[78] N. Tax, B. Dalmas, N. Sidorova, W. M. van der Aalst, and S. Norre. Interest-driven discovery of local process models. *Information Systems*, 77:105–117, 9 2018.

[79] N. Tax, N. Sidorova, R. Haakma, and W. M. van der Aalst. Event abstraction for process mining using supervised learning techniques. In *Proceedings of SAI Intelligent Systems Conference*, pages 251–269. Springer, 2016.

[80] N. Tax, N. Sidorova, R. Haakma, and W. M. van der Aalst. Mining local process models. *Journal of Innovation in Digital Ecosystems*, 3(2):183–196, 2016.

[81] R. Tudoran. *High-Performance Big Data Management Across Cloud Data Centers*. PhD thesis, 2014.

[82] W. Van der Aalst. *Process mining: Data science in action*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016.

[83] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 9 2004.

[84] B. F. Van Dongen, A. K. A. de Medeiros, H. Verbeek, A. Weijters, and W. M. van Der Aalst. The prom framework: A new era in process mining tool support. In *International conference on application and theory of petri nets*, pages 444–454. Springer, 2005.

[85] S. J. van Zelst, A. Bolt, M. Hassani, B. F. van Dongen, and W. M. van der Aalst. Online conformance checking: relating event streams to process models using prefix-alignments. *International Journal of Data Science and Analytics*, 8(3):269–284, 2019.

[86] S. K. vanden Broucke and J. De Weerdt. Fodina: A robust and flexible heuristic process discovery technique. *Decision Support Systems*, 100:109–118, 8 2017.

[87] H. Verbeek and W. M. van der Aalst. Decomposed process mining: The ilp case. In *International conference on business process management*, pages 264–276. Springer, 2014.

[88] A. J. M. M. Weijters and J. T. S. Ribeiro. Flexible heuristics miner (FHM). In *CIDM 2011*, pages 310–317, 2011.

[89] a. J. M. M. Weijters, W. M. P. van der Aalst, and A. K. A. de Medeiros. Process mining with the HeuristicsMiner algorithm. *Cirp Annals-manufacturing Technology*, 166:1–34, 2006.

[90] L. Wen, W. M. Van Der Aalst, J. Wang, and J. Sun. Mining process models with non-free-choice constructs. *Data Mining and Knowledge Discovery*, 15(2):145–180, oct 2007.

[91] M. J. Zaki. Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390, 2000.

RÉSUMÉ

Les journaux d'évènements stockent la trace de l'exécution des processus. Le Process Mining est la discipline de recherche qui vise à analyser ce genre de données et à construire des modèles d'exécution décrivant le déroulement des processus. Plusieurs algorithmes ont été proposés pour extraire de tels modèles mais ils se heurtent aux cas où le modèle contient trop d'irrégularités parmi les différentes instances d'exécution.

Dans cette thèse, nous nous attelons à étudier ces cas où le processus est non structuré et nous nous concentrons sur une méthode particulière d'analyse qui est l'extraction de modèles comportementaux. Nous proposons un nouvel algorithme plus efficace pour ce faire qui garantit certaines propriétés sur les fragments trouvés. Nous proposons également un framework pour analyser et récupérer ces patterns dans une approche orientée données contextuelles alliant corrélation et causalité. Enfin, nous mettons au point un nouvel algorithme de découverte des patterns encore plus rapide avec des résultats visualisables plus concis et plus pertinents.

MOTS CLÉS

Process Mining, Journaux D'évènements, Processus Flexibles, Patterns Comportementaux, Données Contextuelles, Corrélation, Causalité

ABSTRACT

Event logs contain recorded data about business processes execution. Process Mining is the research discipline that analyzes such event logs and aims to discover models describing the unfolding of the process. Many algorithms were proposed but most of them don't take into account cases of high irregularities between execution instances.

In this thesis, we focus on such cases where the processes are unstructured and more exactly on a particular method to get insight from them. Namely, the mining of behavioral patterns. We propose a novel and more efficient algorithm that guarantees certain properties on the extracted patterns. We also propose a framework to analyze and retrieve such patterns in a contextual data-aware fashion manipulating correlation and causation. Lastly, we devise an advanced algorithm for the pattern discovery that is further optimized. It yields more concise and relevant results while offering a visualization interface for easy and interactive analysis.

KEYWORDS

Process Mining, Event logs, Flexible processes, Behavioral Patterns, Contextual data, Correlation, Causality