UNIVERSIDADE DE SANTIAGO DE COMPOSTELA

Departamento de Electrónica e Computación

PHD THESIS

TEMPORAL DATA MINING ALGORITHMS FOR METRIC TEMPORAL CONSTRAINT NETWORKS DISCOVERY

Author:

Miguel Rodríguez Álvarez

Advisors:

Dr. Paulo Félix Lamas

Dr. Purificación Cariñena Amigo

Santiago de Compostela, June 2013

Paulo Félix Lamas, Profesor Titular de Universidad del Área de Ciencias de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

María Purificación Cariñena Amigo, Profesora Contratada Doctora del Área de Ciencias de la Computación e Inteligencia Artificial de la Universidade de Santiago de Compostela

HACEN CONSTAR:

Que la memoria titulada **TEMPORAL DATA MINING ALGORITHMS FOR METRIC TEMPORAL CONSTRAINT NETWORKS DISCOVERY** ha sido realizada por D. **Miguel Rodríguez Álvarez** bajo nuestra dirección en el Centro de Investigación en Tecnologías de la Información de la Universidade de Santiago de Compostela (CITIUS), y constituye la Tesis que presenta para optar al grado de Doctor.

Santiago de Compostela, 13 de junio de 2013

Paulo Félix Lamas
Codirector de la tesis

María Purificación Cariñena Amigo
Codirectora de la tesis

Miguel Rodríguez Álvarez
Autor de la tesis

Acknowledgements

First I would like to thank my Ph.D. advisors, Paulo Félix and Purificación Cariñena, whose support and patience throughout this thesis have made the work presented in this document possible.

On a more personal level, this work would not have been the same without the support provided by my parents and siblings, as well as my friends and their patience during my absences. It is also necessary to thank my colleagues, specially Cristina, Óscar, José Manuel, José Carlos, María, Pablo, Juan, Fabián and Julián among many, many others. Also, although the time spent I spent there might have been short, I thank Antonio Gomariz, Christian Braune and Pascal Held for making me feel at home during my research stays.

I also want to acknowledge the following institutions: the Artificial Intelligence Knowledge Engineerging (AIKE) group of the University of Murcia, specially to Prof. Roque Marín for hosting me during my research visit in 2010; and the Institute for Knowledge and Language Engineering of the Otto-von-Guericke Univerity at Magdeburg, specially to Prof. Rudolf Kruse for his kind invitation that allowed my research stay there in 2012.

I also thank all the institutions that funded this work: the reasearch visit in 2012 to the Institute for Knowledge and Language Engineering of the Otto-von-Guericke-University was supported in part by the European Regional Development Fund (ERDF/FEDER) under the projects CN2012/151 and CN2011/058 of the Galician Ministry of Education; the Ministry of Education, Culture and Sport (FPU grant AP2008-02593), the Spanish Ministry of Science and Innovation (MICINN under the project TIN2009-14372-C03-03), the European Regional Development Fund of the European Commision and Spanish Ministry of Education and Science (FEDER and MEC under the project TIN2006-15460-C04-02), and the Xunta de Galicia (under the project 08SIN002206PR).

Santiago de Compostela, June 13, 2013

Resumen de la tesis

La representación y razonamiento temporales juegan un importante papel en múltiples áreas de la Inteligencia Artificial, tales como el procesado de lenguaje natural, la planificación y programación de tareas, o la realización de diagnósticos. Durante muchos años, el objetivo principal en la investigación en este campo ha sido la formalización de la representación de los distintos matices en el significado de los diversos conceptos temporales, así como el proporcionar los mecanismos de inferencia correspondientes. Estos esfuerzos han resultado en un conjunto de herramientas formales que permiten elicitar conocimiento para una eficaz resolución de problemas.

La creciente disponibilidad de datos temporales en las organizaciones como resultado de su actividad ha dado lugar a la aparición del campo de la Minería de Datos Temporales, que tiene por objetivo inducir conocimiento temporal útil a partir del análisis de estos datos. En un principio la comunidad científica se centró en el diseño de algoritmos eficientes para la minería de secuencias frecuentes. Los sucesivos algoritmos que se han propuesto desde entonces han tenido por objetivo inducir información más expresiva a partir de los datos, poniendo especial énfasis en el tratamiento de la incertidumbre, normalmente de forma cualitativa.

El trabajo aquí presentado plantea como hipótesis de partida que para que una técnica de minería de datos sea eficaz debe estar basada en algún formalismo de representación y razonamiento temporal; dicho formalismo debe permitir la integración de mecanismos de inducción y deducción, así como permitir al usuario interactuar con el proceso de minería mediante la incorporación de conocimiento previo acerca del tipo de patrones de interés, y la combinación de información tanto cuantitativa como cualitativa.

Bajo esta premisa, se elige el formalismo *Simple Temporal Problem* (STP) con el objetivo de representar los resultados del proceso de minería como un conjunto de patrones temporales frecuentes. El formalismo STP ha sido amplia y satisfactoriamente utilizado en

problemas de razonamiento temporal, así como en programación y planificación de tareas. El STP es un tipo de Problema de Satisfacción de Restricciones (PSR) o *Constraint Satisfaction Problem* (CSP) en inglés. Un CSP es un formalismo de representación y razonamiento sobre conocimiento que formula un problema como un conjunto de restricciones que se deben satisfacer para obtener una solución. Una de las ventajas de este tipo de formalismos reside en la posibilidad de modelar gráficamente un problema como una red de restricciones que puede ser representada mediante un grafo, lo que permite el tratamiento de tareas tales como la inferencia, análisis de consistencia, o búsqueda de escenarios, mediante la aplicación de técnicas de procesado de grafos. El formalismo STP representa un patrón temporal como un conjunto de variables temporales y un conjunto de restricciones temporales entre ellas. Cada variable temporal representa un instante, y cada restricción se representa mediante un intervalo que limita los valores admisibles para la distancia temporal entre cada par de instantes temporales. Adoptar un formalismo CSP permite al usuario representar conocimiento previo, hacer inferencia mediante un mecanismo de propagación de restricciones, verificar la consistencia de los patrones resultantes, y dar soporte a una fácil interpretación de los resultados del proceso de minería, como se pone de manifiesto en este trabajo. La elección del formalismo STP como marco de trabajo y de representación de conocimiento conlleva un importante paso adelante con respecto a trabajos previos en la bibliografía de minería de datos temporales a partir de secuencias de eventos, puesto que un STP puede representar restricciones tanto cualitativas como cuantitativas, subsumiendo por tanto a otros formalismos de representación cualitativos, como pueden ser el Álgebra de Puntos Convexa, o el Álgebra de Intervalos Simples Convexa. Por tanto, los algoritmos propuestos en esta tesis proporcionan unos resultados más expresivos que aquellos basados en órdenes frecuentes entre conjuntos de eventos en una colección. Por otra parte, el formalismo STP parece ser una buena elección para una propuesta de minería de datos temporal, puesto que representa un buen balance entre expresividad y complejidad, ya que permite la ejecución de tareas de procesado habituales en tiempo polinómico, en contraste con otros formalismos temporales en los que estas tareas de procesado son NP-completas.

El trabajo presentado desarrolla la hipótesis anteriormente citada y proporciona un conjunto de técnicas de minería de datos temporales agrupadas en dos algoritmos principales: *Apriori Simple Temporal Problem miner* (ASTPminer) y *Hierarchical Simple Temporal Problem miner* (HSTPminer). El trabajo recogido en este documento se resume de la siguiente forma:

– El capítulo 2 proporciona una revisión bibliográfica de formalismos de representación y razonamiento temporal, así como de métodos y técnicas de minería de datos temporales. La primera parte del capítulo se centra en la caracterización de la noción de tiempo y en las cuestiones que dicha caracterización plantea. La primera de estas cuestiones se refiere a la forma a adoptar para la representación del tiempo, puesto que éste puede representarse bien implícitamente mediante cambios en las entidades representativas a estudiar, o puede representarse explícitamente mediante una entidad independiente. Una segunda cuestión se refiere a la elección de las entidades temporales básicas, que en la bibliografía toman la forma de instantes, intervalos o duraciones. Una vez elegidas las entidades de representación temporal deben de discutirse las primitivas de representación de las relaciones temporales entre las entidades temporales, así como la estructura que adopta el tiempo en sí mismo. En esta parte del capítulo se revisan diversas propuestas de formalismos basados en restricciones para la representación y razonamiento temporal, divididos en formalismos cualitativos, cuantitativos y combinados.

  La segunda parte del capítulo se centra en los paradigmas de minería de datos temporales. El desarrollo tecnológico en las últimas décadas ha propiciado que el volumen y variedad de datos contenidos en las bases de datos haya incrementado dramáticamente. Aunque gran parte de estos datos se almacenan únicamente para servir de memoria histórica de las organizaciones, la información que contienen puede mostrarse útil para explicar el pasado y entender el presente, con el fin de predecir ocurrencias futuras. El método tradicional para la obtención de conocimiento a partir de los datos almacenados consiste en el análisis manual de los mismos por parte de un experto, a partir del cual el especialista plantea unas hipótesis o informes sobre las tendencias reflejadas por los datos. Sin embargo, a medida que el número de datos a analizar aumenta, este proceso se vuelve más y más inviable. La minería de datos agrupa técnicas provenientes de campos diversos que permiten automatizar el proceso de análisis, con el objetivo de encontrar relaciones entre los datos o resumirlos de forma comprensible y útil para el usuario. La minería de datos temporales es una extensión de la minería de datos en la que se incorpora una representación del tiempo en el proceso de minería con el objetivo de buscar patrones interesantes en grandes conjuntos de datos temporales. En este apartado nos centramos en aquella bibliografía donde se encuentran técnicas para obtener patrones temporales frecuentes a partir de secuencias de eventos o episodios.

– El capítulo 3 formaliza los principales elementos del problema a resolver. En resumen: dada una colección de secuencias de eventos y episodios temporalmente anotados, se quire encontrar un conjunto de patrones temporales frecuentes representados como un conjunto de STP. Se definen formalmente los conceptos principales que se van a utilizar en el diseño de un proceso de minería de datos temporal: en concreto, las conceptos de tipo de evento y tipo de episodio como entidades temporales fundamentales involucradas en el proceso de minería. Un concepto importante en esta tesis es el de distribución de distancias temporales, una distribución de frecuencias que cuenta el número de ocurrencias de cada distancia temporal diferente entre cada par de tipos de eventos en la secuencia. Este concepto nos permitirá agrupar aquellos patrones frecuentes que muestran disposiciones temporales similares de un conjunto de tipos eventos. Por otra parte, también permitirá propocionar un mecanismo para discriminar entre disposiciones temporales diferentes entre el mismo conjunto de tipos de eventos.

– El capítulo 4 describe en detalle el algoritmo ASTPminer: un procedimiento de minería de datos temporal basado en la estrategia Apriori que tiene por objetivo descubrir patrones temporales frecuentes representados mediante el formalismo STP. En su diseño existen tres elementos fundamentales que deben ser señalados: 1) ASTPminer aplica un criterio de similitud con el fin de distinguir disposiciones temporales diferentes entre el mismo conjunto de tipos de eventos. ASTPminer simplifica esta operación mediante una proyección del criterio de similitud entre cada par de tipos de eventos en el patrón. La aplicación de un procedimiento de agrupamiento sobre distribuciones de distancias temporales de ocurrencias de pares de tipos de eventos produce un conjunto de intervalos, de modo que cada uno de dichos intervalos representa una disposición temporal diferenciada del resto entre el par de tipos de eventos. El cálculo de consistencia de los patrones resultantes refuerza la consistencia de patrones de tamaño superior; 2) ASTPminer saca partido de aquellas secuencias de eventos en las cuales algunos eventos delimitan el comienzo y final de episodios, puesto que gestionándolos de modo que formen parte de una misma entidad, el episodio, es posible acelerar el proceso de minería; 3) ASTPminer permite a los usuarios participar en el proceso de minería mediante la introducción de conocimiento previo del dominio en forma de patrón semilla, representado también mediante el formalismo STP. El usuario proporciona este conocimiento en forma de un patrón semilla que involucra una serie de eventos o episodios de interés y algunas restricciones temporales entre ellos. Este conocimiento inicial se utiliza para

centrar el proceso de búsqueda en aquellos patrones que extienden a la semilla, bien mediante la incorporación de nuevos tipos de eventos, o bien mediante el refinado de las restricciones presentes en el patrón semilla. Como resultado de la introducción de patrones semilla se mejoran la eficacia y la eficiencia del algoritmo.
ASTPminer se valida contra un base de datos temporales de secuencias de eventos extraídas a partir de polisomnogramas de pacientes afectados del Síndrome de Apnea-Hipopnea del Sueño (SAHS). Esta validación tiene dos objetivos. Por una parte, intentamos poner de manifiesto la habilidad de ASTPminer de descubrir patrones frecuentes mediante la utilización de una base de datos temporal que contiene un patrón frecuente previamente descrito en la literatura médica. Por otra parte, evaluamos la eficiencia del proceso de minería con respecto a diferentes parámetros iniciales. Esta validación muestra la calidad de los resultados de la aplicación de ASTPminer en un escenario real y representativo. Sin embargo, también representa una aplicación particular en un tipo muy específico de problemas. Por este motivo, en el capítulo 4 se propone y diseña un generador de bases de datos temporales sintéticas, con el objetivo de evaluar ASTPminer contra un amplio abanico de bases de datos temporales diferentes, lo que nos permite valorar sus fortalezas y debilidades.

– El capítulo 5 describe en detalle el algoritmo HSTPminer: una evolución de ASTPminer que mejora su rendimiento mediante el diseño de dos mecanismos: 1) Una jerarquía de patrones que permite al proceso de búsqueda anotar las ocurrencias de patrones encontradas en cada iteración del procedimiento de búsqueda, con el objetivo de restringir el ámbito de búsqueda en la siguiente iteración del procedimiento de minería a aquellos patrones que extienden a los patrones anotados. Este proceso permite que el algoritmo evite buscar ocurrencias de patrones en aquellas subsecuencias en las que no se encontraron ocurrencias de los patrones que extiende, puesto que no puede haber una ocurrencia de una extensión si no existe ocurrencia del patrón original. Debido a la reducción del número de patrones que deben tenerse en consideración en cada paso del proceso de búsqueda, la eficiencia del proceso de minería se ve mejorada. 2) Los patrones temporales se reorganizan de acuerdo a una estructura de árbol de enumeración de subconjuntos. Esta estructura mejora el proceso de generación de candidatos permitiendo el acceso de forma eficiente a los patrones encontrados en iteraciones anteriores y que son necesarios para la generación de nuevos patrones.
El conjunto de patrones obtenido por el algoritmo HSTPminer sobre una base de datos

determinada es el mismo que el conjunto de patrones obtenido por el algoritmo ASTPminer. Se ha utilizado el mismo conjunto de bases de datos presentado en el capítulo 4, tanto la base de datos de SAHS como las bases de datos sintéticas generadas, con el fin de comparar el rendimiento de ambos algoritmos y analizar hasta qué punto las mejoras introducidas en el nuevo algoritmo afectan a la eficiencia del proceso de minería bajo diferentes condiciones de interés. Como resultado de esta comparación, llegamos a la conclusión de que HSTPminer presenta una mejora computacional con respecto a ASTPminer en ciertas situaciones. En concreto, en aquellas bases de datos en las que el porcentaje de entidades temporales en forma de intervalo es pequeño con respecto al número de entidades en forma de instante, HSTPminer muestra un mejor rendimiento, aunque este rendimiento se degrada a medida que el número de eventos por instante de tiempo aumenta. Por otra parte, en aquellas colecciones en las que se pueden encontrar patrones con gran número de eventos, HSTPminer presenta una mejor eficiencia debido a que permite reducir el número de patrones a tener en cuenta en cada paso, ya que el número total de patrones presentes en el proceso aumenta con el tamaño del patrón, y ASTPminer no tiene forma de discriminar entre aquéllos que pueden estar presentes en una determinada subsección. Sin embargo, en aquellas bases de datos en las que los patrones tienen un tamaño reducido, o que presentan un número de intervalos y de instantes similar, ASTPminer presenta un rendimiento comparable o incluso mejor, ya que el sobrecoste computacional introducido en HSTPminer para reducir el espacio de búsqueda en estas situaciones no se compensa con la reducción temporal obtenida.

– El capítulo 6 proporciona un resumen de los resultados y conclusiones clave que se pueden extraer de la investigación presentada, haciendo referencia a la hipótesis inicial al mismo tiempo que se proporcionan nuevas direcciones para la investigación posterior. El objetivo principal de esta tesis es el de proponer un método formal con el que extraer patrones temporales a partir de bases de datos temporales. Los formalismos de problemas de satisfacción de restricciones, entre los cuales se encuentra el formalismo STP, permiten representar de forma declarativa un conjunto de restricciones que cualquier solución del problema debe de satisfacer. El formalismo STP nos permite representar información temporal mediante una red de restricciones temporales métricas entre un conjunto de variables temporales, cada una de las cuales representa un instante temporal. Normalmente un STP se construye a partir de la descripción de conocimiento experto, y la obtención de un STP a partir de un proceso de minería aplicado sobre con-

junto de datos temporal representa un desafío innovador y relevante para el paradigma de descubrimiento de conocimiento.

La minería de datos temporales tanto en ASTPminer como en HSTPminer se basa en la aplicación de criterios de similitud a las diferentes disposiciones temporales entre los mismos eventos, lo que permite superar limitaciones de trabajos previos, en los que a lo sumo se puede extraer un orden u orden parcial para un conjunto de tipos de eventos dado. El agrupamiento de disposiciones temporales entre eventos es una tarea extremadamente compleja. Sin embargo, en ambos algoritmos esta operación se ve simplificada mediante la proyección de un criterio de similitud para cada par de tipos de eventos del patrón. La construcción de patrones de tamaño superior es posible mediante el análisis y la propagación de restricciones. Puesto que el proceso de agrupamiento únicamente tiene lugar en la búsqueda de patrones frecuentes de tamaño dos, la eficiencia computacioneal del procedimiento global no se ve comprometida.

Una de las principales contribuciones de ambos algoritmos consiste en permitir al usuario participar en el proceso de minería mediante la introducción de conocimiento previo del dominio en forma de patrones, utilizando para ello también el formalismo STP. Este conocimiento se proporciona en forma de patrón semilla consistente en un conjunto de eventos de interés y algunas restricciones entre ellos. Un patrón semilla supone un instrumento para podar el espacio de búsqueda de dos formas. En primer lugar, permite acotar e ignorar aquellas subsecuencias de la colección en las cuales no se puede encontrar una ocurrencia del patrón semilla, lo que permite reducir el número de eventos a considerar en etapas posteriores del proceso. En segundo lugar, las restricciones especificadas por el usuario limitan el número de patrones encontrados por los algoritmos a aquellos que son consistentes con la información incorporada, lo que reduce el número total de patrones presentes en el proceso a la vez que se incrementa su posible interés para el usuario. Ambos aspectos contribuyen a mejorar tanto la eficiencia como la eficacia del proceso.

Hay dos aspectos fundamentales en los que centrar las mejoras sobre ASTPminer y HSTPminer. En primer lugar, al tratarse ambos algoritmos de estrategias basadas en el paradigma Apriori, el número de patrones a tratar en cada una de las iteraciones del proceso de búsqueda sufre de problemas de explosión combinatoria, puesto que deben crearse y tratarse todos los posibles patrones candidatos a partir de los patrones frecuentes obtenidos en la anterior iteración. Este proceso no tiene solamente un alto coste en la generación de candidatos, sino también en el cálculo de frecuencia, por lo

que debe estudiarse el uso de otras estrategias. En segundo lugar, sería interesante incrementar la expresividad de los patrones obtenidos como resultado. Una primera vía de mejora de la expresividad radica en la posibilidad de incorporar el concepto de negación en el proceso de minería, entendido como la ausencia sistemática de algún tipo de evento en el contexto temporal de la ocurrencia de un patrón. Otra vía de mejora de expresividad se encuentra en explotar en mayor medida la información contenida en las distribuciones de distancias temporales. Esta información puede utilizarse para extender el formalismo STP incorporando al concepto de restricción los conceptos de preferencia, probabilidad o posibilidad, entre otros. Una tercera vía de mejora yace en el diseño e implementación de herramientas visuales que permitan una mejor usabilidad de ASTPminer y HSTPminer, lo que además permitiría a la comunidad científica compartir conjuntos de datos y resultados de referencia.

Contents

List of Figures **xix**

List of Tables **xxiii**

1 Introduction **1**

2 Literature review **5**

2.1 Constraint-based formalisms 6

2.1.1 Qualitative constraint formalisms 7

2.1.2 Quantitative constraint formalisms 11

2.1.3 Combined formalisms 12

2.2 Temporal data mining 13

2.3 Sequence data mining 15

2.4 Partial orders 24

2.5 Interval-based data mining 26

2.6 Time series data mining 28

2.7 Our approach 28

3 Definitions **31**

4 ASTPminer **39**

4.1 Basic algorithm 40

4.1.1 Candidate generation 41

4.1.2 Frequency calculation 44

4.1.3 Clustering 47

4.1.4 Validation of the ASTP_BASIC algorithm . . . 49
4.2 Time optimization: Event removal . . . 53
4.2.1 Event removal using window markings . . . 54
4.2.2 Event removal using interval markings . . . 55
4.2.3 Experimental results using the event removal approaches . . . 57
4.3 Providing seed knowledge . . . 58
4.3.1 Initialisation procedure . . . 62
4.3.2 Frequency calculation with a seed pattern . . . 63
4.3.3 Validation of `ASTP_SEED` . . . 66
4.4 ASTPminer . . . 73
4.4.1 Initialisation step in `ASTPminer` . . . 76
4.4.2 Frequent pattern calculation in ASTPminer . . . 77
4.4.3 Candidate generation in ASTPminer . . . 82
4.4.4 Correctness and completeness of ASTPminer . . . 83
4.4.5 Complexity analysis of ASTPminer . . . 85
4.4.6 ASTPminer validation with the SAHS database . . . 87
4.5 ASTPminer validation: synthetic databases . . . 91
4.5.1 Synthetic databases generation . . . 93
4.5.2 Experimental results with synthetic databases . . . 94

5 HSTPminer 103
5.1 Set enumeration tree . . . 104
5.2 Pattern hierarchy . . . 106
5.3 HSTPminer . . . 109
5.3.1 Initialisation . . . 111
5.3.2 Frequent pattern calculation . . . 112
5.3.3 Candidate generation . . . 121
5.3.4 Correctness and completeness of HSTPminer . . . 124
5.3.5 Complexity analysis of HSTPminer . . . 126
5.4 HSTPminer validation . . . 128
5.4.1 SAHS database . . . 128
5.4.2 Synthetic databases . . . 130

6 Main contributions and conclusions 137

Bibliography **143**

List of Figures

Fig. 2.1 IA-relations and corresponding PA-relations. 10
Fig. 2.2 Floyd-Warshall's `ALL-PAIRS-SHORTEST-PATHS` algorithm. 12
Fig. 2.3 Sequence database example. 16
Fig. 2.4 Example of a partial order and the sequences it summarises. 25

Fig. 3.1 Temporal association with event types A, B and C. 34
Fig. 3.2 An example of two frequent temporal patterns represented as STP, discovered in a collection of sequences, with $f_{min} = 2$. 38

Fig. 4.1 `ASTP_BASIC`: main algorithm. 41
Fig. 4.2 `CANDIDATE_GENERATION` algorithm. 42
Fig. 4.3 Pattern combination example. 43
Fig. 4.4 `FREQUENCY_CALCULATION` algortihm. 45
Fig. 4.5 Temporal distance distribution between two event types A and B with $\omega = 80$. 46
Fig. 4.6 Frequency calculation example. 47
Fig. 4.7 Example of clustering on a temporal distance distribution. 50
Fig. 4.8 Time required by `ASTP_BASIC` depending on the window size. 53
Fig. 4.9 Example of event removal using window marking. 55
Fig. 4.10 Frequency calculation for event removal using window marking: `FREQUENCY_CALCULATION_WM`. 56
Fig. 4.11 Example of event removal using interval marking. 56
Fig. 4.12 Frequency calculation for event removal using interval marking: `FREQUENCY_CALCULATION_IM`. 58

Fig. 4.13 Comparison of time required by the ASTP_BASIC algorithm without event removal, and with each one of the event removal strategies (WM and IM) during the frequency calculation stage. 59
Fig. 4.14 Example of a frequent pattern of size 8 obtained from the SAHS database with the ASTP_BASIC algorithm. 60
Fig. 4.15 Mining algorithm using a seed pattern: ASTP_SEED. 62
Fig. 4.16 Initialisation algorithm used in ASTP_SEED: INITIALISATION. 63
Fig. 4.17 Frequency calculation algorithm when using a seed pattern: FREQUENCY_CALCULATION_SEED. 65
Fig. 4.18 Example of frequency calculation when a seed pattern is provided. 66
Fig. 4.19 Seed pattern used in the validation experiments. 67
Fig. 4.20 Impact of using a seed pattern on a temporal distance distribution and its clustering. These temporal distance distributions correspond to event types *"begin airflow limitation"* and *"end airflow limitation"*. 68
Fig. 4.21 Second example on the impact of using a seed pattern on a temporal distance distribution and its clustering: number of patterns found may be higher when using a seed pattern. The temporal distance distributions correspond to event types *"begin oxygen desaturation"* and *"end oxygen desaturation"*. 70
Fig. 4.22 Third example on the impact of using a seed pattern on a temporal distance distribution and its clustering: patterns found are more specific, and more relevant to the final user. The temporal distance distributions correspond to event types *"begin thoracic limitation"* and *"end thoracic limitation"*. . . . 71
Fig. 4.23 Another example of the influence of the seed pattern in the clustering process. 72
Fig. 4.24 Comparison of time required by the ASTP_BASIC algorithm and two different implementations of the ASTP_SEED algorithm in the SAHS database. 72
Fig. 4.25 Comparison of two frequent patterns of size 8: (a) obtained by ASTP_BASIC without seed pattern, (b) obtained by ASTP_SEED when a seed pattern was provided. 74
Fig. 4.26 Main algorithm: ASTPminer. 75
Fig. 4.27 ASTPminer: Initialisation algorithm. 77
Fig. 4.28 Frequency calculation in ASTPminer: FREQUENCY_CALCULATION. . . 78
Fig. 4.29 Frequency calculation example in ASTPminer. 81

Fig. 4.30 Candidate generation algorithm in ASTPminer: `CANDIDATE_GENERATION`. 82
Fig. 4.31 Seed pattern used in the validation experiments. 88
Fig. 4.32 Time required by the `ASTPminer` algorithm with and without seed pattern. 89
Fig. 4.33 Patterns of size 8 found with and without introducing a seed pattern, with a window width of 80 seconds. 92
Fig. 4.34 Time required by the `ASTPminer` algorithm with and without seed pattern, in the SAHS database. max_P=8, N=120212, Δ=0.15, $|\mathcal{E}|$=0, $|\mathcal{G}|$=4. 95
Fig. 4.35 `ASTPminer` execution time in synthetic database SDB1: max_P=9, N=136000, Δ=0.07, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4 96
Fig. 4.36 `ASTPminer` execution time in synthetic database SDB2: max_P=11 , N=275000, Δ=0.15, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4 97
Fig. 4.37 `ASTPminer` execution time in synthetic database SDB3: max_P=9 , N=470000, Δ=0.25, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4 97
Fig. 4.38 `ASTPminer` execution time in synthetic database SDB4: max_P=8, N=705000, Δ=0.39, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4 98
Fig. 4.39 `ASTPminer` execution time in synthetic database SDB5: max_P=14, N=70000, Δ=0.22 , $|\mathcal{E}|$=12, $|\mathcal{G}|$=1 98
Fig. 4.40 `ASTPminer` execution time in synthetic database SDB6: max_P=19, N=76000, Δ=0.25, $|\mathcal{E}|$=13, $|\mathcal{G}|$=3 99
Fig. 4.41 `ASTPminer` execution time in synthetic database SDB7: max_P=13, N=65000, Δ=0.2, $|\mathcal{E}|$=13, $|\mathcal{G}|$=0 99
Fig. 4.42 `ASTPminer` execution time in synthetic database SDB8: max_P=10, N=50000, Δ=0.4, $|\mathcal{E}|$=10, $|\mathcal{G}|$=0 100
Fig. 4.43 `ASTPminer` execution time in synthetic database SDB9: max_P=13, N=95212, Δ=0.8, $|\mathcal{E}|$=13, $|\mathcal{G}|$=0 100
Fig. 4.44 `ASTPminer` execution time in synthetic database SDB10: max_P=12, N=123402, Δ=2.25, $|\mathcal{E}|$=12, $|\mathcal{G}|$=0 101

Fig. 5.1 Set enumeration tree example. 106
Fig. 5.2 Pattern hierarchy example. 108
Fig. 5.3 `ENDING_EVENTS` procedure. 108
Fig. 5.4 `HSTPminer`: Main algorithm. 110
Fig. 5.5 `HSTPminer`: Initialisation algorithm. 111

Fig. 5.6 HSTPminer: Frequency calculation. 113
Fig. 5.7 HSTPminer: Pattern verification. 115
Fig. 5.8 Pattern hierarchy frequency calculation example. 119
Fig. 5.9 Pattern hierarchy frequency calculation with episodes example. 121
Fig. 5.10 HSTPminer: Candidate generation. 122
Fig. 5.11 Example of candidate generation with set enumeration tree. 124
Fig. 5.12 Seed pattern used in the experiments. 129
Fig. 5.13 Time required by the ASTPminer algorithm with and without seed pattern, and the HSTPminer algorithm with and without seed pattern. 130
Fig. 5.14 SDB1: max_P=9, N=136000, Δ=0.07, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4 131
Fig. 5.15 SDB2: max_P=11 , N=275000, Δ=0.15, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4 131
Fig. 5.16 SDB3: max_P=9 , N=470000, Δ=0.25, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4 132
Fig. 5.17 SDB4: max_P=8, N=705000, Δ=0.39, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4 132
Fig. 5.18 SDB5: max_P=14, N=70000, Δ=0.22 , $|\mathcal{E}|$=12, $|\mathcal{G}|$=1 133
Fig. 5.19 SDB6: max_P=19, N=76000, Δ=0.25, $|\mathcal{E}|$=13, $|\mathcal{G}|$=3 133
Fig. 5.20 SDB7: max_P=13, N=65000, Δ=0.2, $|\mathcal{E}|$=13, $|\mathcal{G}|$=0 134
Fig. 5.21 SDB8: max_P=10, N=50000, Δ=0.4, $|\mathcal{E}|$=10, $|\mathcal{G}|$=0 134
Fig. 5.22 SDB9: max_P=13, N=95212, Δ=0.8, $|\mathcal{E}|$=13, $|\mathcal{G}|$=0 135
Fig. 5.23 SDB10: max_P=12, N=123402, Δ=2.25, $|\mathcal{E}|$=12, $|\mathcal{G}|$=0 135

List of Tables

Tabla 4.1 Number of possible candidates, candidates generated, combinations discarded and frequent patterns found in the SAHS database by `ASTP_BASIC`, with a window size ω=80 s. and a frequency threshold f_{min}=30 occurrences. 53

Tabla 4.2 Number of events removed from the collection in each iteration by both event removal strategies; window size 80 s. 59

Tabla 4.3 Number of possible candidates, candidates generated, combinations discarded and frequent patterns found in the database, with and without seed pattern (window size ω =80 s., frequency threshold f_{min} =30 occurrences). 69

Tabla 4.4 Number of possible candidates, candidates generated, combinations discarded and frequent patterns in the database, using the `ASTPminer` algorithm with and without seed pattern (using a window size ω=80 s., and a frequency threshold f_{min}=30 occurrences). 90

Tabla 4.5 Synthetic database generator parameters. 93

Tabla 4.6 Synthetic databases parameters. 95

CHAPTER 1

INTRODUCTION

Temporal representation and reasoning play an important role in multiple areas of Artificial Intelligence, such as natural language processing, planning, scheduling or diagnostics. For long years, the main point in the research activity has been to put on formal grounds the representation of the multiple nuances of meaning for the different temporal notions, and to provide the corresponding inference mechanisms. These efforts have resulted in a set of formal tools that enable to elicit effective problem solving knowledge.

The increasing availability of temporal data as a result of the activity of organizations has caused the emergence of the field of Temporal Data Mining, which aims to induce new and useful temporal knowledge from the computer data processing. Initially, the scientific community focused on designing efficient algorithms for mining frequent temporal patterns, where each temporal pattern emphasises a particular temporal order among a set of events. Since then, new algorithms have been proposed in order to induce more expressive information from data, specially focusing on coping with uncertainty, usually in a qualitative form.

The present work takes as a starting point the thesis that any effective temporal data mining technique should be based on some formalism for temporal reasoning and representation. This formalism should enable the integration of induction and deduction mechanisms, it should also allow the user to interact with the mining process by providing previous knowledge about the kind of patterns of interest, and it should combine qualitative and quantitative information.

Under this premise, the Simple Temporal Problem (STP) formalism has been selected to represent the results of the mining process as a set of frequent temporal patterns. The STP formalism has been widely and successfully applied on temporal reasoning, scheduling or planning tasks. The STP is a type of Constraint Satisfaction Problem (CSP). A CSP is

a formalism for knowledge representation and reasoning that formulates a problem as a set of constraints which must be satisfied in order to obtain a solution. The STP represents a temporal pattern as a set of temporal variables and a set of temporal constraints among them. Each temporal variable represents a time instant, and each constraint is represented by an interval limiting the permissible values for the distance between each pair of time instants. Adopting a CSP formalism allows the user to represent previous knowledge, to make inference by a simple mechanism of constraint propagation, to ensure the consistency of the resultant patterns, and to support an easy interpretability of the results of the mining process, as this work will highlight. On the other hand, the STP formalism seems to be a good choice for a temporal data mining proposal, since it represents a good balance between expressiveness and complexity, supporting the usual processing tasks in polynomial time, in contrast to other temporal formalisms where the usual processing tasks are NP-hard.

The present work develops the aforementioned thesis by providing a set of temporal data mining techniques grouped into two main algorithms: Apriori Simple Temporal Problem miner (`ASTPminer`) and Hierarchical Simple Temporal Problem miner (`HSTPminer`). The work presented in this document is summarised as follows:

– Chapter 2 offers a literature review on the formalisms used for temporal representation and reasoning, as well as on temporal data mining methods and techniques. This review is followed by a discussion on the suitability of the STP formalism for the representation of the temporal patterns resulting from a temporal data mining process.

– Chapter 3 formalises the main elements of the problem to be solved; in particular, the notions of event and episode types as the basic temporal entities involved in the mining procedures. In short: given a collection of sequences of time-stamped events and episodes, the aim is to find a set of frequent temporal patterns represented as a set of STP. An important notion for this thesis is that of temporal distance distribution, a frequency distribution counting the number of occurrences of every different temporal distance between every two event types in a sequence. This concept will allow us to group together those frequent patterns that show a similar temporal arrangement of a set of event types.

– Chapter 4 describes the `ASTPminer` algorithm in detail: an Apriori-based temporal data mining procedure for discovering frequent temporal patterns represented in the STP formalism. Three key elements must be highlighted in its design: 1) `ASTPminer`

applies similarity criteria in order to distinguish different temporal arrangements between the same event types; ASTPminer simplifies this operation by projecting the similarity criteria between each pair of event types in the pattern; a clustering procedure over temporal distance distributions of occurrences of pairs of event types produces a set of intervals, each one of them representing a distinguishable temporal arrangement between the pair of event types; then, consistency checking of the resulting patterns enforces the assembling of consistent patterns of bigger sizes; 2) ASTPminer takes advantage of those event sequences where some events delimit the beginning and the end of episodes, since managing them as taking part of the same entity, an episode, allows us to speed up the mining process; 3) ASTPminer allows users to participate in the mining process by providing previous domain knowledge, also represented by the STP formalism; user knowledge is provided as a seed pattern involving a number of events or episodes of interest and some temporal constraints between them; this initial knowledge then focuses the search process on those patterns that extend the seed, either by incorporating new event types or by refining the constraints in the seed pattern; as a result, seed patterns contribute to improve algorithm efficacy and efficiency.
ASTPminer is validated against a temporal database of sequences extracted from the polysomnograms of patients diagnosed with Sleep Apnea-Hypopnea Syndrome (SAHS). The objective is twofold. On the one hand, we try to bring out the ability of ASTPminer to discover frequent patterns by means of a temporal database containing a well-known frequent pattern previously described in the medical literature. On the other hand, we evaluate the efficiency of the mining process with respect to different initial parameters. This validation shows the quality of the results of applying ASTPminer in a highly representative real scenario. However, it just represents a particular application in a rather specific type of problems. For this reason, chapter 4 shows the proposal and design of a synthetic temporal database generator, with the aim of evaluating ASTPminer against a wide range of different temporal databases, and precisely assessing its strengths and weaknesses.

– Chapter 5 describes the HSTPminer algorithm in detail: an evolution of ASTPminer that improves its behaviour by designing two mechanisms: 1) A pattern hierarchy allows the search process to annotate occurrences of patterns found in each iteration of the mining procedure, with the aim of constraining the search scope in the next iteration to those patterns that extend the annotated ones. By reducing the number of patterns

that need to be taken into account in each step of the search process, the efficiency of the mining process is improved. 2) Temporal patterns are reorganised following a set enumeration tree structure. This structure improves the candidate generation process by allowing it to easily access patterns found in previous iterations that are needed for the generation of new patterns.
Patterns obtained using the `HSTPminer` algorithm are the same as the patterns obtained by the `ASTPminer` algorithm. The same set of databases used in chapter 4, both the SAHS database and the synthetic databases generated, are used to compare the performance of both algorithms and analyse to which extent the improvements introduced into the algorithm affect the efficiency of the mining process under several different conditions.

– The conclusion chapter provides a synthesis of the key results and findings drawn from the present research, making a stand regarding the thesis statement, and also provides new directions for future research.

CHAPTER 2

LITERATURE REVIEW

Time representation and reasoning plays a central role in Artificial Intelligence from its beginning, encouraging scientists to provide a formal inclusion of temporal information in problem solving. There is a number of questions to tackle in characterizing the notion of time itself. A first question regards the form adopted by the representation of time, since time can be implicitly represented in those changes in representative entities, or can be explicitly represented as an independent entity [Mccarthy and Hayes, 1969, Kowalski and Sergot, 1986]. A second one is about choosing the basic temporal entities. Here we can find three different possibilities in the bibliography: instants, intervals or durations [van Benthem, 1983, Allen, 1983, Navarrete et al., 2002]. There is a subsequent discussion regarding the primitive temporal relations among the basic temporal entities. A third question is about the time structure, as it can be bounded or unbounded, dense or discrete; and about the sort of order among temporal entities: total, partial, ramified,... [Vila, 1994].

Traditionally, there exist two distinctive approaches to representing and reasoning about time: temporal logics and constraint-based formalisms. Temporal logics is an approach to the semantics of expressions qualified in terms of time, emphasizing the expressiveness, but most often at the expense of efficiency, and sometimes, showing incompleteness in those expressible problems. Constraint-based formalisms can be seen as a sort of reified temporal logic, providing sound and easy methods for processing information, sometimes with reduced computational cost. A more in-depth review on representing and reasoning about time can be found in [Combi and Shahar, 1997, Chittaro and Montanari, 2000, Augusto, 2001, Pani and Bhattacharjee, 2001, Visser and Hübner, 2003, Adlassnig et al., 2006].

The present work deals with the problem of mining frequent temporal patterns in temporal datasets, a large scale intensive search-based problem. This sort of problem will require a rigorous but non-computationally expensive management of time, which leads us to review the literature on constraint-based formalisms for representing and reasoning about time.

2.1 Constraint-based formalisms

Formally, a Constraint Satisfaction Problem (CSP) is defined as a finite set of *variables* $X = \{x_1, x_2, ..., x_n\}$, with respective *domains* $D = \{D_1, ..., D_n\}$ which list the possible values for each variable $D_i = \{v_{i_1}, ..., v_{i_k}\}$, and a set of *constraints* $C = \{C_1, ..., C_t\}$ among the variables. Thus, a CSP can be viewed as a triple $R =< X, D, C >$ [Tsang, 1993, Dechter, 2003]. A constraint C_i is a relation R_i defined on a subset of variables $S_i \subseteq X$. If $S_i = \{x_{i_1}, ..., x_{i_r}\}$, then R_i is a subset of the Cartesian product $D_{i_1} \times \cdots \times D_{i_r}$. Thus, a constraint can also be viewed as a pair $C_i =< S_i, R_i >$. A constraint C_i is said to be *satisfied* by a tuple of assignments $(x_{i_1} = v_{i_1}, ..., x_{i_r} = v_{i_r})$ if $(v_{i_1}, ..., v_{i_r}) \in R_i$. A *solution* of a CSP is an assignment to all of its variables $(x_1 = v_1, ..., x_n = v_n)$ that satisfies all the constraints. A CSP is *consistent* if it has at least one solution, and it is inconsistent otherwise. The following tasks can be identified for any CSP:

1. To determine whether the CSP is consistent.
2. To find all of its solutions.
3. To find an optimal solution according to an objective function.
4. To find a partial solution, that is, an assignment to a subset of the variables.
5. To obtain the minimal CSP equivalent to the original one, that is, the CSP with the smallest domain size.

The graphical representation of a CSP was first studied for binary constraints [Freuder, 1982]. In a binary constraint network every constraint affects at most two variables. In this case, a constraint network can be assigned to a constraint graph, where each node represents a variable and each arc connects a pair of nodes related by a binary constraint. Representing a constraint network as a graph makes it possible to successfully apply graph-based techniques to task resolution in CSP.

Those techniques usually applied to CSP solving can be classified in three categories [Dechter et al., 1991]:

– Search-based techniques for systematic exploration of the solution space, usually by backtracking.

– Consistency checking techniques for obtaining a more explicit representation of the CSP, improving the ensuing systematic exploration. In general, a k-consistency algorithm removes all inconsistencies involving all subsets of size k of the n variables. For example, the node, arc and path consistency algorithms detect and eliminate inconsistencies involving k=1, 2 and 3 variables, respectively. A network of n variables is said to be globally consistent if it is k-consistent for $1 \leq k \leq n$ [Freuder, 1982, Koubarakis, 1997].

– Structure-based techniques for guiding systematic exploration by exploiting topological characteristics of the network.

A number of authors have approached the problem of representing and reasoning about time as a CSP, giving rise to different proposals where certain temporal relations (qualitative or quantitative) among different temporal variables (instants, intervals or durations) are a matter of choice. In the next section, we briefly review the most outstanding temporal CSP formalisms resulting from different combinations of choices.

2.1.1 Qualitative constraint formalisms

Multiple areas of Artificial Intelligence such as scheduling, planning, diagnostics or natural language processing pose the need for managing temporal information by means of qualitative relations among temporal entities. What really matters is the relative order among the temporal entities, more than their exact location in time.

The *Point Algebra* (PA) is an approach to representing temporal information in terms of qualitative relations between instants or temporal points [Vilain and Kautz, 1986]. Two instants p_i and p_j can be related by three qualitative basic relations: before (<), equals (=) and after (>). Sometimes, the information about two instants is incomplete, and can be expressed by a disjunction of the basic relations $Q_{pp} \equiv \{<,=,>\}$. Note that, for example, $p_i \leq p_j$ is an abbreviation of the disjunction $(p_i < p_j \wedge p_i = p_j)$. The set of possible relations in the Point Algebra is $2^{Q_{pp}} = \{\varnothing, <, >, =, \leq, \geq, ?\}$. Every PA-relation is a subset of basic relations. The

? relation is defined as the universal constraint $? \equiv \{<,=,>\}$ and represents the absence of information. The $\varnothing$ relation represents the unsatisfiability of any relation between two instants, and it appears in inconsistent networks.

The PA is formally defined as an structure $(2^{\mathcal{Q}_{pp}}, ^{-1}, \cap, \circ)$, where $2^{\mathcal{Q}_{pp}}$ represents the 8 subsets of PA-relations, $(^{-1})$ represents the inverse operator, such that $p_i\, R\, p_j \equiv p_j\, R^{-1}\, p_i$, $(\cap)$ represents the intersection, such that $p_i(R \cap S)p_j \equiv p_i\, R\, p_j \wedge p_i\, S\, p_j$, and $(\circ)$ the composition of relations: composition of relation R_{ij} between points p_i and p_j and relation R_{jk} between points p_j and p_k is a new relation $R_{ik} = R_{ij} \circ R_{jk}$ such that $p_i R_{ik} p_k \equiv p_i R_{ij} p_j \wedge p_j R_{jk} p_k$. If the $\neq$ relation is excluded from the subset of PA-relations we obtain the Convex Point Algebra (CPA) [Van Beek and Cohen, 1990]. Excluding a non convex relation as $\neq$ improves the computational cost of temporal reasoning.

PA-relations can be represented by a PA-network, where the nodes represent instants and the arcs represent PA-relations between them. A polynomial solution can be found to a number of temporal reasoning problems represented by means of a PA-network:

- *Consistency checking of a PA-network*. Consistency checking can be enforced in $O(n^3)$ for CPA-networks and in $O(n^4)$ for PA-networks, by using a path-consistency algorithm where n is the number of nodes in the network [Ladkin and Maddux, 1988].

- *To find a solution for a PA-network*. Van Beek proposes a more efficient algorithm, named *cspan*, for consistency checking and for finding a solution in $O(n^2)$. This algorithm is based on searching strongly connected components in the network for obtaining a reduced graph where arcs are labelled as $<$, $>$, and ? [van Beek, 1992].

- *To find the minimal PA-network*. Van Beek proves that the path consistency algorithm is not complete for PA-networks, only for CPA-networks. Thus, the minimal CPA-network can be obtained in $O(n^3)$. Obtaining the minimal PA-network can be accomplished in $O(n^4)$ by applying a 4-consistency algorithm [Van Beek and Cohen, 1990]. Later, the same author improves this result and presents an algorithm for obtaining the minimal PA-network in $O(max(n^3, mn^2))$, where m is the number of arcs labelled as $\neq$.

- *To decide global consistency*. Koubarakis has proved that 5-consistency is necessary and sufficient for achieving global consistency, and it can be enforced in $O(mn^4)$, where m is the number of arcs labelled as $\neq$ [Koubarakis, 1997].

The *Interval Algebra* (IA) is an approach to representing temporal information in terms of qualitative relations between intervals [Allen, 1983]. Allen identifies 13 basic relations,

named IA-relations, in order to describe the relative position between a couple of intervals: before, meets, overlaps, starts, during, finishes, equals and their corresponding inverse relations. We shortly represent this set by $Q_{ii} = \{b, b^{-1}, m, m^{-1}, o, o^{-1}, s, s^{-1}, d, d^{-1}, f, f^{-1}, e\}$. These basic relations are mutually excluding, in the sense that two given intervals can satisfy only one of them.

However, as in the case of PA-relations, incomplete or indeterminate knowledge can also be expressed by disjunctive relations, and we get $2^{13} = 8192$ possible relations between intervals in the full algebra. For example, the disjunction $i_i\ b\ i_j \wedge i_i\ m\ i_j$, denoted by $i_i\{b,m\}i_j$, shows that i_i interval is before or meets i_j interval. Relations of special interest are the null relation $\varnothing$, representing the negation of all the basic relations, and the universal relation, representing the disjunction of all the basic relations.

The IA is formally defined as an structure $(2^{Q_{ii}}, ^{-1}, \cap, \circ)$, where $2^{Q_{ii}}$ represents all the possible subsets of IA-relations, $(^{-1})$ represents the inverse operator, $(\cap)$ represents the intersection and $(\circ)$ the composition of relations. Let $R = \{R_1, ..., R_m\}$ be a disjunction of IA-relations. Allen's method for inverse is to use the equivalence $R^{-1} \equiv \{R_1^{-1}, ..., R_m^{-1}\}$. Intersection of two disjunctive relations is the intersection of every constitutive basic relation. A 13×13 transitivity table for relation composition can be found in [Allen, 1983]; composition of two disjunctive IA-relations can be obtained as the composition of every two pairs of basic relations.

A basic IA-relation can be expressed as a conjunction of basic PA-relations between the start and end points of the intervals involved. Figure 2.1 shows the set of IA-relations and the corresponding PA-relations.

Freksa presents a generalization of Allen's interval-based approach [Freksa, 1992]. Relations between semi-intervals rather than intervals are used as the basic relations. Semi-intervals correspond to temporal beginnings or endings of intervals, that is, intervals where the beginning or ending, or both of them, are not defined. The author argues that semi-intervals are rather natural entities both from a cognitive and from a computational point of view. Recently, a fuzzy extension IA^{fuz} of IA has been proposed [Badaloni and Giacomin, 2006].

Every interesting reasoning problem for IA is NP-hard [Vilain et al., 1990]. In particular, consistency checking is NP-complete [Vilain and Kautz, 1986]. It is interesting to search for IA subclasses with good computational properties. Among them we mention:

IA-relation	Inverse	Relative position	PA-relation
X before (b) Y	b^{-1}	$\longleftarrow X \longrightarrow \quad \longleftarrow Y \longrightarrow$	$X^- < Y^-, X^- < Y^+$ $X^+ < Y^-, X^+ < Y^-$
X meets (m) Y	m^{-1}	$\longleftarrow X \longrightarrow \longleftarrow Y \longrightarrow$	$X^- < Y^-, X^- < Y^+$ $X^+ = Y^-, X^+ < Y^+$
X overlaps (o) Y	o^{-1}	$\longleftarrow X \longrightarrow$ $\longleftarrow Y \longrightarrow$	$X^- < Y^-, X^- < Y^+$ $X^+ > Y^-, X^+ < Y^+$
X during (d) Y	d^{-1}	$\leftarrow X \rightarrow$ $\longleftarrow Y \longrightarrow$	$X^- > Y^-, X^- < Y^+$ $X^+ > Y^-, X^+ < Y^+$
X starts (s) Y	s^{-1}	$\leftarrow X \rightarrow$ $\longleftarrow Y \longrightarrow$	$X^- = Y^-, X^- < Y^+$ $X^+ > Y^-, X^+ < Y^+$
X finishes (f) Y	f^{-1}	$\leftarrow X \rightarrow$ $\longleftarrow Y \longrightarrow$	$X^- > Y^-, X^- < Y^+$ $X^+ > Y^-, X^+ = Y^+$
X equals (e) Y	e	$\longleftarrow X \longrightarrow$ $\longleftarrow Y \longrightarrow$	$X^- = Y^-, X^- = Y^+$ $X^+ = Y^-, X^+ = Y^+$

Figure 2.1: IA-relations and corresponding PA-relations.

– *Pointisable subclass*, P-IA: a set of relations of IA that can be expressed as a conjunction of PA-relations involving the end points of each interval [Vilain et al., 1990, Van Beek and Cohen, 1990, van Beek, 1992].

– *Continuous endpoint subclass*, C-IA: a set of relations of IA that can be expressed as a conjunction of CPA-relations involving the end points of each interval [Vilain et al., 1990].

– *ORD-Horn subclass*, H-IA: a set of relations of IA that can be expressed as a conjunction of ORD-Horn constraints involving the end points of each interval [Nebel and Bürckert, 1995]. An ORD-Horn constraint is a disjunction of inequalities $a = b$ or $a \leq b$ and disequations $a \neq b$ where the number of inequalities does not exceed one. This is the unique maximal subclass containing all the basic interval relations, for which satisfiability can be solved using a polynomial-time algorithm [Ligozat, 1998].

Usual CSP tasks in P-IA and C-IA (consistency checking, finding a solution, obtaining a minimal network and deciding global consistency) can be solved by translating interval relations to a conjunction of point relations, and then applying PA and CPA algorithms.

2.1.2 Quantitative constraint formalisms

Quantitative temporal networks provide a convenient formalism to deal with metric information about temporal points and durations. Here we describe the *Temporal Constraint Satisfaction Problem* (TCSP), and a particular case, the *Simple Temporal Problem* (STP) [Dechter et al., 1991]. A TCSP involves a set of variables $\{x_1,...,x_n\}$ representing time points and a set of unary and binary constraints. Each constraint is represented by a set of intervals $\{I_1,...,I_k\} = \{[a_1,b_1],...,[a_k,b_k]\}$. A unary constraint L_i restricts the domain of variable x_i to the given set of intervals, that is, it represents the disjunction $(a_1 \leq x_i \leq b_1) \vee \cdots \vee (a_k \leq x_1 \leq b_k)$. A binary constraint L_{ij} restricts the permissible values for the difference $x_j - x_i$, that is, it represents the disjunction $(a_1 \leq x_j - x_i \leq b_1) \vee \cdots \vee (a_k \leq x_j - x_i \leq b_k)$. It is assumed that constraints are given in a canonical form in which all intervals are pair-wise disjoint.

A TCSP can be represented by a directed constraint graph, where nodes represent time points and each arc is labelled by a metric constraint, that is, the set of arcs is a set of intervals. A special time point, x_0, is introduced to represent "the beginning of the world". All times are relative to x_0; thus, we may treat each unary constraint L_i as an equivalent binary constraint L_{ij} where x_j is x_0. A tuple $x = (a_1,...,a_n)$ is called a *solution* if the assignment $(x_1 = a_1,...,x_n = a_n)$ does not violate any constraint.

With the aim of combining quantitative constraints, a metric algebra with set intersection $(\cap)$, composition $(\otimes)$ and inverse $(^{-1})$ operators is used. Given two constraints C_{ij} and C'_{ij} between the same time points, the composition $C_{ij} \otimes C'_{ij}$ is defined as:

$$C_{ij} \otimes C'_{ij} = \bigcup_{I_s \in C_{ij}, I_t \in C'_{ij}} I_s + I_t = [a_s,b_s] + [a_t,b_t] = [a_s + a_t, b_s + b_t]$$

The inverse constraint can be obtained as $C_{ij}^{-1} = \{I_s^{-1} \mid I_s \in C_{ij}\}$ where $[a,b]^{-1} = [-b,-a]$.

Unfortunately the main tasks of TCSP, deciding consistency and obtaining the minimal network, are NP-complete. A Simple Temporal Problem (STP) is a simplification of the TCSP where all constraints specify a single interval. In an STP, each arc is labelled by a single interval $[a_{ij},b_{ij}]$, that represents the constraint $a_{ij} \leq x_j - x_i \leq b_{ij}$. An STP can be associated with a directed edge-weighted graph G_d, called a distance graph. It has the same node set as the constraint graph, and each constraint $[a_{ij},b_{ij}]$ between nodes x_j and x_i is represented by two arcs: an arc labelled b_{ij} from x_i to x_j and an arc labelled $-a_{ij}$ from x_j to x_i, representing the pair of inequalities $x_j - x_i \leq b_{ij}$ and $x_i - x_j \leq -a_{ij}$. The well-known Floyd-Warshall's `ALL-PAIRS-SHORTEST-PATHS()` algorithm can be applied to the distance graph, in order to obtain the shortest path between each pair of nodes. This algorithm runs

```
   procedure ALL-PAIRS-SHORTEST-PATHS(V, E = {a_ij | i, j ∈ {1,...,n}})
 1 begin
 2   for i:= 1 to n do d_ii ← 0
 3   for i:= 1 to n do
 4     for j:= 1 to n do
 5       d_ij ← a_ij
 6   for k:= 1 to n do
 7     for i:= 1 to n do
 8       for j:= 1 to n do
 9         d_ij ← min(d_ij, d_ik + d_kj)
10   return (V, {d_ij | i, j ∈ [1..n]})
11 end;
```

Figure 2.2: Floyd-Warshall's ALL-PAIRS-SHORTEST-PATHS algorithm.

in time $O(n^3)$, and constitutes a polynomial time algorithm for deciding the consistency of an STP and for computing the minimal network. Figure 2.2 shows the pseudocode for this procedure, where given an STP consisting of a set of nodes V and a set of arcs E, the algorithm returns the minimal network for the same set of nodes, each arc d_{ij} representing the shortest distance between nodes i and j according to the values in E.

2.1.3 Combined formalisms

Some efforts have been made to integrate qualitative and quantitative temporal information on points and intervals. In this section we review some of these proposals.

In the *Qualitative Algebra* (QA) [Meiri, 1996], a qualitative constraint between two temporal entities O_i and O_j (each may be a point or an interval) is a disjunction of the form $(O_i\ r_1\ O_j) \wedge \cdots \wedge (O_i\ r_k\ O_j)$, where $\{r_1, ..., r_k\}$ is a set of basic qualitative relations that may exist between two entities. There are three types of basic qualitative relations: 1) point-point (PP) relations that can hold between a pair of points; 2) point-interval (PI) and interval-point (IP) relations that can hold between a point and an interval and vice versa; 3) interval-interval (II) relations that can hold between a pair of intervals. The set of PP relations is $(<, =, >)$ and II relations are Allen's relations shown in figure 2.1. The set of basic PI relations is $Q_{pi} = \{before, starts, during, finishes, after\}$. The set of basic IP relations is defined by the inverse operation $Q_{ip} = \{before, starts^{-1}, during^{-1}, finishes^{-1}, after\}$.

The set of QA-relations gathers together the subsets of basic relations of the same type: 2^{13} II-relations, 2^3 PP-relations, 2^5 PI-relations and 2^5 IP-relations. Meiri defines intersection and composition operations on disjunctions of relations. However, composition of IP-relations with PI-relations and vice versa is not defined. Thus, QA is not an algebra in a strict sense, since composition is not a closed operation.

Under this formalism, qualitative and quantitative constraints are integrated in an augmented CPA- or PA-network, with the addition of unary metric constraints to establish the position of a time point in the temporal axis, or binary metric constraints to establish the temporal distance between two time points.

In a subsequent work, computational complexity of relating time points with intervals is studied [Jonsson et al., 1999]. A complete classification of all subclasses of the point-interval algebra with respect to tractability is provided. The classification reveals that there exist five maximal tractable subclasses of the point-interval algebra, one of them being the only tractable subclass that contains all the basic relations.

Badaloni and Berati define the Interval Distance Sub Algebra (IDSA), where nodes are intervals [Badaloni and Berati, 1996]. These intervals are related by disjunctive 4-tuple-metric constraints between their ending time points $\{(I_i^-, I_j^-), (I_i^+, I_j^-), (I_i^-, I_j^+), (I_i^+, I_j^+)\}$. Staab and Hahn propose a model for reasoning on qualitative and metric boundaries of intervals [Staab and Hahn, 1998]. However, these models cannot handle constraints on interval durations, which were identified earlier by Allen, and which require a high-order expression [Dechter et al., 1991], or a duration primitive which should be integrated with interval and point constraints [Allen, 1983, Barber, 1993]. Particularly, Barber proposes two orthogonal networks to relate constraints on durations and time points. Later, Navarrete et al. extend the point algebra model to include additional variables that represent durations between time points, in the so-called *Point-Duration Network* [Navarrete et al., 2002]. Deciding consistency is NP-complete, but tractable special cases are identified, and efficient algorithms for checking consistency, finding a solution and obtaining the minimal network are provided.

2.2 Temporal data mining

The volume and variety of information contained in digital databases and other sources has dramatically increased in the last decades. While a great deal of this information is historical in nature, serving as a memory of the organisation that collects it, the information also may

be useful to explain the past and understand the present with the aim of predicting future occurrences [Hand et al., 2001].

The traditional method of extracting knowledge from data is performed by a domain expert, who manually analyses and interprets the data. From this analysis, the specialist may raise some hypothesis or reports reflecting the tendencies in the data. This kind of work is slow, expensive, subjective and virtually impracticable in domains where the volume of data grows rapidly. In addition, as the source of the data becomes more and more heterogeneus, the necessity of tools and techniques to analyse the data and obtain useful information grows more important [Fayyad et al., 1996].

Until recently, the analysis of data located in databases was performed by using general query languages as well as on-line transaction processing. However, this procedure only provided summarised information following previously established representations, which did not scale well under large volumes of data. Data warehouse technologies aim to ease the analysis of heterogeneous data sources by providing a unified schema and decision support tools [Han et al., 2005]. On-line analytical processing tools are also provided, allowing to aggregate information and view the information under different points of view. Yet these techniques only extract information that still needs to be analysed by a domain expert to be useful, and do not allow to generate rules, patterns or knowledge representations that may be applied to other datasets.

Data mining can be seen as the application of techniques arising from diverse disciplines, such as artificial intelligence, machine learning, data visualisation or statistics to the large amounts of data held in heterogeneus data sets. Data mining is defined as the analysis of large data sets to find unsuspected relations and to summarise the data in novel ways that are useful and understandable to the data owner [Hand et al., 2001].

The relations obtained from the data mining process are often referred to as *models* or *patterns*. However, it is important to point out that both terms refer to different types of entities. On the one hand, a model is a global summary of a data set. A model makes statements about any point in the whole space of measurements, even when the data point is missing some information. For instance, the function produced by a least squares regression method may predict values in sections of the data set where no data point was available.

On the other hand, patterns describe structures that refer to some sections of the space where the data may occur, that is, only part of the data set behaves as the patterns describes, as opposed to the rest of the data which behaves differently. Therefore, models and patterns

may be considered as opposing entities, where models describe the usual behaviour while patterns refer to those situations where an unusual behaviour is found [Hand et al., 2001].

Temporal data mining is an extension of data mining that can be defined as the search for interesting correlations of patterns in large sets of temporal data. Temporal data mining has the capability to discover patterns or rules which might be overlooked when the temporal component is ignored or treated as a simple numeric attribute [Roddick and Spiliopoulou, 2002]. A large volume of research has therefore been focused on temporal data mining to discover temporal rules such as sequential patterns, episodes, temporal association rules and inter-transaction association rules. The analysis of event sequences is a relevant issue within temporal data mining, since a sequence of events is a common representation of the temporal activity of multiple organisations. Thus, sequence data mining appears as a first effort to develop algorithms that seek frequent subsequences in event sequences, and where the order on the temporal axis of these events is taken into account. This type of analysis has originated an extense field of application known as sequence data mining. In the following section we present what we consider to be the main proposals in this field.

2.3 Sequence data mining

Sequence databases consist of sequences of ordered elements or events, where a concrete notion of time may or may not have been considered while the database was recorded. Sequence data can be found in multiple application domains, such as customer shopping sequences, alarm logs in telecommunication networks, biological sequences, manifestations in the course of a disease, and other natural and social domains. Sequential data mining discovers useful and interesting knowledge in the form of frequently occurring sets of ordered events, or subsequences, denominated patterns.

Sequence data mining techniques share a common vocabulary that allows us to simplify the discussion of the different proposals [Han et al., 2005]. An *item* is a basic entity of the domain of interest present in the data set. The set of all *items* is $\mathcal{I} = \{I_1, \ldots, I_n\}$. An *itemset* or *transaction* is a non-empty set of items. The size of an itemset corresponds to the number of items it contains, and an itemset of size i is known as an $i-itemset$. A *sequence* is an ordered list of *events*, represented as $S = \{e_1 e_2 \ldots e_n\}$. An event can also be called an *element* and usually corresponds to an itemset. The number of elements in a sequence is called the *length* of the sequence, and a sequence of length n is an *n-sequence*. A sequence $\beta = \{b_1 \ldots b_m\}$ is

the *supersequence* of another sequence $\alpha = \{a_1 \dots a_n\}$ denoted as *subsequence* if there exists a set of integers $1 \leq j_1 < \dots < j_n \leq m$ that satisfy that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$.

A *sequence database* $\mathcal{S}$ is a set of tuples (id, S), where id is an identifier and S is a sequence. A sequence α is *contained* in a tuple (id, S) if α is a subsequence of S. The *support* of a sequence in a sequence database is the number of tuples that contain the sequence. The *minimum support threshold* is a positive integer *min_sup* which represents the lowest support value that a sequence needs to hold to be considered *frequent*. Therefore, a sequence needs to be contained by at least *min_sup* tuples to be considered frequent. In this case, the sequence is called a *sequential pattern*. Given two frequent sequences α and β such that $\alpha \subseteq \beta$, then α is a *subpattern* of β and, conversely, β is a *superpattern* of α.

Figure 2.3 shows an example of a sequence database where the itemsets to be mined have a temporal value associated. Two equivalent representations can be seen. Each row in a table corresponds to a different itemset or transaction. The first representation is more similar to a traditional relational database, while the second emphasises the temporal dimension.

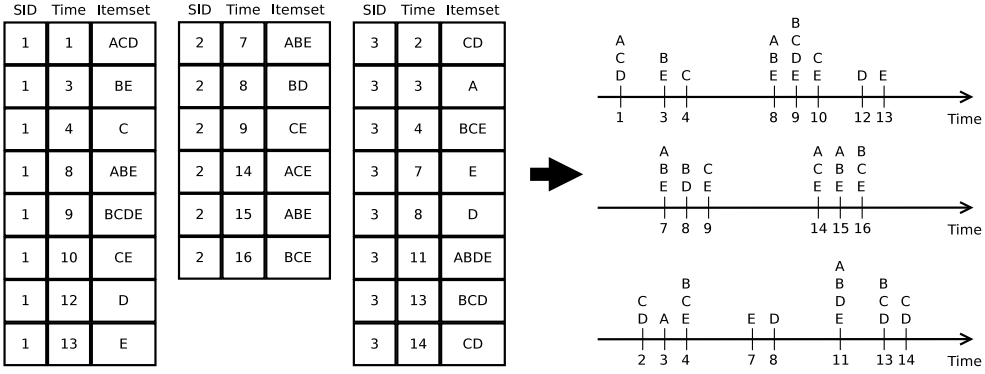


Figure 2.3: Sequence database example.

The aim of sequence data mining techniques is to induce a set of frequent sequential patterns, where each pattern is a partially ordered list of itemsets that may be commonly found in the dataset, and each itemset occurs before the next. Sequence data mining techniques face the problem of dealing with large amounts of sequential patterns, specially as the length of the patterns grows. The different proposals can be divided in two categories. The first category includes those proposals where the aim is to obtain the complete set of sequential patterns, whereas the second category refers to techniques that only contemplate the set of

closed sequential patterns. A pattern α is closed if there is no superpattern β with the same support value.

Techniques aimed to discover the complete set of sequential patterns rely, either directly or indirectly, in the *Apriori property*. This property states that every subpattern of a frequent pattern is also frequent. This property is antimononotic in nature, because if a sequence is not frequent, then necessarily all of its supersequences cannot be frequent. For this reason, the property is also known as *antimonotonicity property*, or *downward-closure property*.

The Apriori property receives its name from the *Apriori algorithm* originally proposed in [Agrawal and Srikant, 1994]. This algorithm uses prior knowledge about the properties of frequent sequences to prune the search space, while aiming to obtain frequent itemsets which are then used to generate sets of temporal association rules between the elements contained in the itemsets. These rules specify relations between items of the data sets, where each relation expresses some behaviour among the data based on the frequency of the simultaneous appearance of two or more items. Apriori defines an iterative search procedure where frequent $i-itemsets$ are used to explore $i+1-itemsets$, which results in a breadth-first or level-wise search, where each level must be fully explored before proceeding to the next, and level i represents all frequent patterns of size i.

Apriori divides the mining process in two steps. First, it obtains all frequent patterns, represented as frequent itemsets, in the data set. Then, it expresses the knowledge of the patterns in a set of association rules. A pattern is considered frequent if its support, that is, the number of itemsets in the sequence database where the pattern can be found, surpasses a user-established threshold designated as *minimum support*. Association rules are considered interesting if their confidence, that is, the ratio between the support of the antecedent of the rule and the support of the itemset containing every item in the rule surpasses another user-established parameter called *minimum confidence*.

Each iteration is divided into three steps: candidate generation, frequent sequence calculation, and removal of non-frequent candidates. In each iteration i, the algorithm uses the set of frequent sequences of size $i-1$, labelled as L_{i-1}, to generate the set of all possible candidates of size i, labelled as C_i, that satisfy the antimonotonicity property. Then, the frequency of all the candidates on the dataset is verified. Finally, those candidates that cannot be considered frequent are removed from the process. Therefore, the set of candidates C_i is a superset of the set of frequent patterns L_i.

The candidate generation procedure is further subdivided into two steps. In the first step, known as *join* step, the set L_{i-1} is joined with itself. Every two associations $p,q \in L_{i-1}$

such that both share all of their items except the last one produce a new association $p \cup q$ which is added to C_i. In the second step, labelled as *prune* step, all elements $r \in C_i$ such that there exists at least one subset $t \subset r$ of size $i-1$ where $t \notin L_{i-1}$ are removed from C_i, as the antimonotonicity property is not satisfied and therefore they cannot be frequent.

In the frequency calculation procedure, all transactions in the data sequences are analysed. If the presence of any candidate in C_i is detected in a transaction, then the support of the candidate is increased by one. The procedure finishes by removing from C_i all associations that do no satisfy the support threshold.

Due to its simplicity, and how the candidate generation procedure reduces the search scope of the mining process, the Apriori algorithm spawned a family of algorithms using the antimonotonicity property, often referred to as Apriori-like algorithms. These algorithms share the level-wise iterative approach, where the complete set of candidates of size i is generated, and then the sequence database is used to verify the support of each candidate and remove from the process those candidates below the support threshold.

In [Agrawal and Srikant, 1995] the authors propose an algorithm that tries to minimise the number of candidate sequences searched throughout the mining process. The process is divided into two phases, labelled as forward phase and backward phase. In the forward phase the algorithm performs the candidate generation step as the basic Apriori algorithm, but skips the frequency calculation in some iterations, depending on the ratio of frequent patterns found in the last frequency calculation performed. Then, in the backward phase, the algorithm calculates the frequency of those candidates generated whose frequency was not previously calculated, which are not subsequences of frequent patterns found in the forward phase. The same authors in [Srikant and Agrawal, 1996] introduce two types of temporal constraints during the frequency calculation procedure. The first type consists of two constraints, named *MinGap* and *MaxGap*, that respectively force a minimum and a maximum temporal separation between any two consecutive elements of a pattern, whereas the second type, called *sliding temporal window*, forces all elements of a pattern to fall within the same temporal proximity.

In [Lu et al., 2000] the authors propose the E-Apriori and EH-Apriori algorithms. Both algorithms extend the original Apriori algorithm to deal with several dimensions, instead of dealing only with time. Transactions are considered to be points in an n-dimensional space, and patterns consist of a sequence of normalised n-dimensional points. The concept of sliding temporal window is also extended to deal with n-dimensions and renamed as *maxspan*, forcing all points of the pattern to be under the n-dimensional area covered by the window.

[Ale and Rossi, 2000] modify the Apriori algorithm to take into account the lifespan of the items, that is, the time periods when the items really exist. The authors then consider that the support of a pattern can only be calculated by taking into account the subset of sequences where the item was active, instead of the complete set of sequences. For instance, if a product has been sold only for five months, the algorithm will only consider itemsets within that period, and ignore any other transaction that belongs to other time periods. This allows the algorithm to find patterns where items with a limited lifespan, such as new products, would be ignored due to the presence of items with a much greater lifespan. Therefore, the proposal needs that the explicit lifespan of the items being studied is provided.

The *SPADE* algorithm is proposed in [Zaki, 2001]. This algorithm subdivides the search space into smaller pieces, called sub-lattices, that can be analysed independently. By dividing the search space, and by binding each item to the list of sequences where it is found, it is possible to minimise the number of database passes to count the frequency of the patterns. In addition to the Apriori breadth-first search procedure, the author proposes a depth-first search algorithm, which recursively extends one pattern until no new items can be added, instead of searching for all patterns of size i before searching for patterns of size $i+1$.

Apriori-like pattern mining algorithms effectively reduce the search space by pruning by means of the antimonotonicity property. Still they bear nontrivial problems inherent to their breadth-first nature. The first problem lies in the number of passes over the database needed to find all frequent patterns. As pattern size grows by one at each iteration, and it is necessary to count the occurrences of all candidates throughout the database in each iteration of the mining process, the procedure will perform i passes over the full data set in order to find a pattern of size i. The second problem is the potentially large number of candidates generated through the process. In order to identify a frequent pattern of size i, an Apriori-like algorithm needs to generate and count the frequency of $2^i - 1$ patterns.

The authors in [Agarwal et al., 2000] propose a depth-first search by means of a lexicographic tree. Each node in the tree represents a frequent itemset, and its parent in the tree represents the same itemset but missing the last item. This representation allows the mining process to focus the counting of the itemset to those parts of the database where the parent itemset was found, reducing the search scope for each itemset. The same authors also add a breadth-first and a mixed approach in [Agarwal et al., 2001]. However, none of the three approaches is able to perform intratransactional mining, so the temporal aspect is left unexplored.

Another depth-first approach is proposed in [Han et al., 2000b]. The algorithm, called *FP-growth*, transforms the database into a tree structure labelled as FP-tree, which summarises the database by finding common prefixes in the sequences and reorganising them to reduce the space required. Each branch of the tree represents a sequence of items following a descending frequency order, and some part of the branches may be shared or connected to represent common sequences in the original database. Frequent patterns can then be obtained by traversing the different branches of the tree following a depth-first search, making it unnecessary to iteratively generate and test the whole set of candidates of a fixed size before searching for bigger patterns. Even though the approach outperforms Apriori-like approaches in mining frequent sequential patterns, items and sequences with different orderings imply that the tree will be huge, along with an increase of the memory requirements of the algorithm.

The same authors in [Han et al., 2000a] propose the FreeSpan algorithm, which uses frequent items to project transactional databases into smaller databases, where each projection may only contain ocurrences of a distinct subset of patterns. In this regard, a projection of a database with respecto to a pattern is the set of sequences where an occurrence of the pattern can be found. Therefore, the frequency calculation of each pattern can be confined to a subset of the whole database.

The *PrefixSpan* algorithm [Pei et al., 2001] is presented as an improvement over FreeSpan. The main difference lies in how the projection of the database for each pattern is calculated. Since the pattern being searched for in each iteration is obtained by adding one item at the end, it is enough to only search for the presence of the new item among the items left after the occurrence of the pattern being extended. Therefore, the projection of the database for one pattern consists of a set of subsequences of the original database, where each subsequence includes only those items of the original sequence that still need to be analysed. Due to how the projections are built, the search scope of the mining process is reduced as pattern size grows, since the frequency calculation process can be focused on those subsequences where the candidate pattern might be present. The MEMISP algorithm proposed in [Lin and Lee, 2005] has a similar behaviour to PrefixSpan. However, instead of building a new projection for each pattern analysed, the MEMISP algorithm builds a unique index for each frequent pattern of size one. Although both PrefixSpan and MEMISP perform similarly when the data sequences fit into memory, MEMISP performs better when this is not the case.

Similarly to Apriori-like algorithms, PrefixSpan has been used as a starting point for several data mining algorithms, all of them following a similar structure [Li and Wang, 2008, Chen et al., 2003, Hu et al., 2009, Wu and Chen, 2007]. The typical PrefixSpan algorithm can

be considered as a recursive method divided in three steps. The first step finds all frequent items in the current projection of the database. Then, each frequent item found is appended at the end of the current pattern, producing one new frequent pattern for each different item. Finally, the procedure makes a recursive call for each frequent pattern found in the previous step, where each call will search for extensions of only one pattern and also sees the database reduced to those subsets of the previous projection occurring after the item appended was found. Although PrefixSpan algorithms perform better than Apriori-like algorithms when searching for sequential patterns, it is not trivial to adapt them to find patterns not representing total orders.

The *MinGap* and *MaxGap* constraints proposed in Apriori are introduced into PrefixSpan in [Li and Wang, 2008]. In addition, a method is proposed to find only closed patterns, that is, patterns having a frequency value such that all of its extensions have a lower frequency value. Searching only for this kind of patterns allows the mining process to obtain a more compact set of patterns and improves the overall efficiency.

The concepts of *MinGap* and *MaxGap* are extended in [Chen et al., 2003]. Two algorithms are proposed, one based on Apriori and another based on PrefixSpan. Both algorithms assume a disjoint partition of the time axis, where each element of the partition represents an interval of possible distances between one element of the pattern and the next element. The algorithms obtain a set of frequent patterns, where the possible distance between one event and the next is limited by at least one of the intervals of the partition. It is possible to have different intervals in the same pattern, instead of forcing the same set of possible distances between two successive events as MinGap and MaxGap do. This approach is improved in [Hu et al., 2009] by introducing a constraint limiting the possible temporal distance between any two elements in the pattern, regardless of their occurrence order in the pattern. However, the partition of the time axis remains constant through the procedure, which means that even if some of the values in one interval associated to a pair of events are not possible due to other intervals in the same pattern, these values are not removed from the interval, forcing every interval to be the same as in the partition. This results in a loss of information in the process. In addition, due to the static nature of the intervals and the partition of the time axis, some temporal arrangements between events may arbitrarily fall under the same pattern when they should not, and the opposite may also be true, some temporal arrangements may fall into two different patterns when in fact they represent the same situation.

Instead of relying on a fixed partition to represent the possible distances between successive elements in a pattern, in [Giannotti et al., 2006] the authors propose to make use of

a clustering procedure to coalesce similar temporal distances between the same events in a single pattern. Patterns are described as a sequence of elements, where the temporal distance between one element and the next is calculated by taking into account the density of the distances observed throughout the database between the two events under consideration. Similarly, in [Nanni and Rigotti, 2007] the authors group similar occurrences of a sequential pattern and build a tree for each set of items. The arcs connecting two successive elements of a branch have an interval associated, which represents the minimum and maximum temporal distances between the events. Therefore, each branch of the tree represents a different sequential pattern.

Regardless of how complex is the model used to represent the patterns obtained, the computational workload remains similar: large databases, which are linked to a huge search space, which in turn results in a large solution set. Not only does the mining process become inefficient, but interesting knowledge becomes obfuscated among useless or spurious patterns. Therefore, a data mining process should be both interactive and iterative [Mitra et al., 2002]. Interaction may be present at various stages, and domain knowledge may be present either as a high-level specification of the model, or at a more detailed level. Usually, once all frequent patterns are obtained, the domain expert analyses, interprets and evaluates them, aiming to extract useful knowledge from them. However, the manual analysis of such a large amount of results is not a viable solution, and obtaining a new set of patterns that represent the temporal information contained in the database as a summarised representation should be the aim instead. This problem is known as *higher order knowledge discovery*, the mining of previously mined rules. In addition, the *constraint-based pattern mining* paradigm aims to allow the user to specify a set of constraints that are pushed into the mining process, where the constraints represent what situations are of interest to the user, focus the mining process on these situations, and remove early on uninteresting situations. The development of new, efficient algorithms that enable human experts to collaborate in the mining process more intuitively is recognised as an important challenge for future data mining proposals [Garofalakis et al., 1999, Bettini et al., 1998a, Pei et al., 2002, Sacchi et al., 2007b].

Conventional sequence and temporal data mining techniques only provided the minimum frequency threshold mechanism to specify the interestingness of the patterns, which is recognised as a limited mechanism.

[Hirate and Yamana, 2006] propose a mining technique that allows the user to specify several constraints over the mining procedure simultaneously. The first constraint consists of a temporal window, which specifies the maximum separation allowed between any two items

to be considered part of the same occurrence. The user may also specify the MinGap and MaxGap constraints, as well as the number of items that may be present between two successive items of a pattern so that a sequence can be considered an occurrence of the pattern. The SPIRIT algorithms, proposed in [Garofalakis et al., 1999], are designed to allow the user to provide a set of constraints, expressed as regular expressions, to the mining process with the aim of obtaining a set of frequent patterns that satisfy these constraints. The algorithms are based on Apriori, which allows patterns obtained to be extended with new events. Regular expressions are also used in [Trasarti et al., 2008] to prune the search space, but instead of forcing the patterns generated to adhere to the regular expression, they use an automata that accepts the regular expression introduced to obtain the patterns directly from the data sequences. A hierarchy of relaxations to regular expressions is proposed in [Antunes and Oliveira, 2005], allowing the user to provide guidelines on how to use the constraints introduced, and therefore allowing the user to obtain knowledge that could not be inferred from the knowledge introduced itself if it were applied in a more strict manner. In addition, the authors propose to use regular expressions as constraints to obtain patterns that do not satisfy them, which is a useful tool for fraud detection processes.

A technique that allows a domain expert to interact with the mining process to progressively refine the results is proposed in [Bettini et al., 1998a]. The user provides a skeletal definition of a pattern, represented as a finite automaton with only one root. Each node in the automaton represents an event type, the arcs connecting the nodes have some sort of temporal granularities associated, and each granularity limits the possible temporal distances between the instantiation of the event types connected. The user also provides the root event type, and may optionally specify some event types for the rest of the nodes of the automaton. The algorithm proposed aims to discover all possible frequent patterns consistent with the structure provided, where each pattern obtained represents a different instantiation of event types to the nodes left unspecified by the user, and the frequency is measured as the fraction of occurrences of the pattern with respect to the number of occurrences of the root event type. However, the algorithm cannot add new event types by itself, and requires the user to manually introduce them. Granularities are also used in [Li et al., 2003] to propose an Apriori-like technique that obtains a set of temporal association rules, where each rule specifies the values of each granularity provided in which each rule holds.

A formalisation of the constraint-based sequential pattern mining problem is presented in [Pei et al., 2002]. Constraints considered in previous approaches are formalised and separated into seven different categories, while their monotonicity and anti-monotonicity properties are

analysed, as well as their possible use in prefix-growth algorithms. The first category specifies whether a set of items should or should not be present in the patterns obtained. The second refers to the minimum or maximum length of the pattern. The third category of constraints forces the resulting patterns to be superpatterns of a set of patterns specified by the user. Aggregate functions, such as the sum or the average of the values of the items in the pattern constitute the fourth category, and regular expressions represent the fifth category. The sixth category forces patterns to have a minimum or maximum temporal duration, and the last category contains the MinGap and MaxGap constraints. The PrefixSpan algorithm is then adapted to allow the domain expert to specify constraints from these categories, forcing the algorithm to only search for frequent patterns that satisfy the constraints introduced. However, the authors identify some specific constraints within the category of aggregate functions called tough aggregates, for instance "the average cost of the items in the pattern", where the approach is suboptimal. In [Bonchi and Lucchese, 2007] an Apriori-like solution that solves this problem is proposed.

2.4 Partial orders

Frequent sequential patterns, whether closed or not, represent interesting total orders in terms of the support of the elements in the sequence database analysed. When the result set consists of closed patterns, the number of frequent patterns the user needs to review is reduced, which also reduces the importance of a careful choice of the minimum support threshold. However, total orders cannot represent all the particularities in sequential data. To solve this problem partial orders are proposed, which aim to summarise the information contained in different sequential patterns to improve their representation and comprehension. An example of a partial order and the sequences it summarises is shown in figure 2.4, where the three sequences on the left are grouped under a single partial order, each branch in the graph corresponding to a different sequence.

In [Mannila et al., 1997] an Apriori algorithm is proposed that makes use of the concept of sliding temporal window to force events of a pattern, called episodes, to occur within the same temporal context. Therefore, episodes are defined as a partial order of events occuring within the same temporal proximity. Two types of patterns can be found, depending on whether the order of the events in the window is important or not. In the first type, called *parallel episodes*, the order of the events is not important, as long as all the events fit in the same window. The second type, named *sequential episodes*, takes into account the order of the events in the

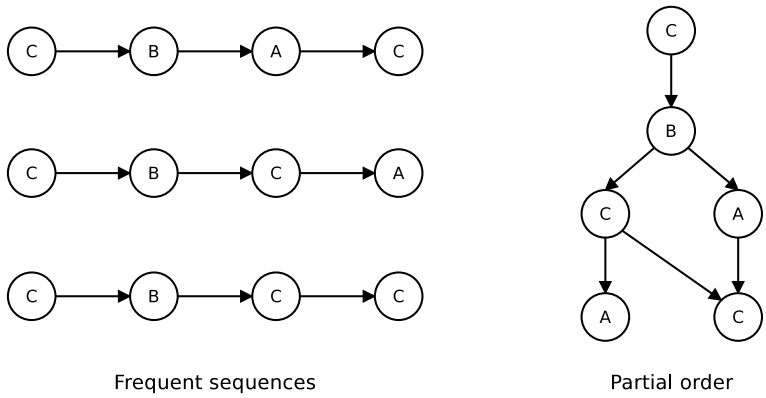


Figure 2.4: Example of a partial order and the sequences it summarises.

window, so that the same set of events represent an occurrence of different patterns depending on their order in the temporal window. However, even in the case of sequential episodes, the specific temporal arrangements between any two events of the pattern remain unknown. In [Harms et al., 2001] the authors propose an extension that searches for closed patterns, which reduces the number of patterns involved in each iteration of the process, and improves the efficiency of the mining process.

A method that allows the mining process to summarise the qualitative knowledge contained in the set of frequent sequential patterns obtained by other techniques, such as Apriori or PrefixSpan, and produces a reduced set of partial orders that represent that knowledge in a more compact manner is proposed in [Casas-Garriga, 2005]. The PrefixSpan presented in [Pei et al., 2004] is adapted in [Pei et al., 2006] to improve the process of finding the set of partial orders. Then, [Tatti and Cule, 2011] extend the problem to mining frequent closed episodes as defined by [Mannila et al., 1997] so that the patterns obtained may represent events occurring in no particular order, one event necessarily after the other, or both events simultaneously. The authors propose a depth-first approach but, instead of doing a database projection as in PrefixSpan, the method relies on annotating the windows where the pattern being analysed was found.

2.5 Interval-based data mining

Although most sequence data mining techniques assume items of interest to be point-based, there are domains where the observed phenomena is interval-based in nature, and previously mentioned proposals do not allow to perform an appropriate study.

In [Kam and Fu, 2000] the authors propose an Apriori-like algorithm that uses Allen's interval relations and a sliding temporal window to search for frequent patterns of interval-based events. In [Papapetrou et al., 2005] the authors distribute the patterns in a tree structure based on a set enumeration tree [Rymon, 1992], and then define both a breadth-first Apriori-like algorithm and a depth-first algorithm to construct the tree, improving the efficiency of the mining process. Once the set of frequent patterns has been found, temporal association rules may be extracted. In [Hoppner, 2001] another Apriori method that produces a set of temporal association rules extracted from the patterns obtained is proposed. A more efficient approach is proposed in [Winarko and Roddick, 2007], where the MEMISP algorithm is adapted, and the MaxGap constraint is added to the mining process, allowing the algorithm to reduce the search scope by removing uninteresting patterns. Although these approaches are able to deal with interval-based events, Allen's relations present problems, as they are both ambiguous and qualitative in nature, and therefore their expressiveness is limited.

[Wu and Chen, 2007] argue that using Allen's temporal relations results in a set of ambiguous patterns, as some temporal relations may be mapped to different patterns, and some relations between events may be lost. To solve this problem, the authors propose a PrefixSpan technique that represents patterns as a partial ordering of the ending points of the interval-based events. In [Patel et al., 2008] an additional Apriori-like method that provides some pruning strategies to reduce the search space is proposed. A proposal to deal with both point-based events and interval-based events simultaneously is presented in [Wu and Chen, 2009]. Patterns obtained are labelled as hybrid patterns. The authors argue that traditional sequence data mining techniques cannot deal with this problem satisfactorily, as they either transform intervals into its corresponding beginning and ending events and forego the pair-wise relation between both, or they simulate that point-based events are interval-based events with zero duration, which degrades performance. In their proposal the authors define a method that represents hybrid patterns as partial orders between point-based events while guaranteeing the integrity of the interval-based events. Another take on hybrid patterns is proposed in [Chen et al., 2011], where point-based events are elements present within interval-based events, and each interval-based event is represented by a sequence of point-based events.

Each interval-based event has its own domain of possible point-based events, and there may or may not be an occurrence of each point-based event within an occurrence of an interval-based event. Patterns obtained are represented by means of a partial order of the point-based events. The algorithm is divided into two steps. The first step obtains every different frequent partial order of each individual interval-based event. The second step combines the patterns obtained in the first step to obtain frequent patterns, represented as partial orders, that consider more than one interval-based event. These approaches handle the problem of the ambiguity of Allen's relations, yet the patterns extracted are still qualitative in nature, limited to giving partial orders between the events analysed, and do not provide more precise information about their temporal arrangements.

In [Moerchen and Ultsch, 2007] another improvement over Allen's interval relations is proposed. The authors identify that Allen's relations are ambiguous, and small changes in the boundaries of the intervals, for example under the effects of noise, may produce patterns with very different relations. For instance, a small change in the boundaries of the intervals shown in the *starts* in figure 2.1 relation may change the relation between the intervals to the *overlaps* relation or the *during* relation. Instead, the authors define a hierarchical language to represent interval-based events, and then propose a depth-first method that mines closed partial orders of intervals [Moerchen, 2006], while considering that small variations in the arrangements of the intervals represent the same situations.

Other proposals go beyond qualitatitive temporal relations between intervals, such as Allen's relations, and aim to obtain quantitative temporal relations as a result of the mining process. Similarly to [Giannotti et al., 2006] in point-based events domains, an Apriori-like technique is presented in [Guyet and Quiniou, 2008] to extract patterns where several different occurrences of sequences of interval-based events, all of them similar in terms of their temporal arrangement and duration of the events, are grouped with the representative of the pattern. The algorithm projects a hyper-cube around each possible representative and then estimates the density function of sequences within the hyper-cube. The authors then propose to use a density-based clustering algorithm, such as the Expectation-Maximization algorithm, to obtain the set of representatives. The same authors propose in [Guyet and Quiniou, 2011] a PrefixSpan approach to this problem. However, instead of projecting hyper-cubes, the clustering procedure is used to group sequences according to the similarity of their time intervals, both in terms of their instant of occurrence and their duration.

2.6 Time series data mining

In some application domains data cannot be found in the form of sequences, but they may come in the form of time series. For instance, electrocardiograms in the clinical domain. In order to use the techniques discussed in this chapter, these time series need to be preprocessed in some way to transform them into sequences of point-based events or interval-based events. In [Bellazzi et al., 2011] a review on techniques on the clinical field that need this sort of preprocessing can be found.

[Agrawal et al., 1995] use similarity measures in time series data with the aim of discovering all similar subsequences in a set of time series sequences, where the temporal axes of the time series do not need to be aligned. The similarity measure is able to deal with different scales in the vertical axis, as well as differences in the offset. Unsupervised neural networks to generate a description of the underlying time series are used in [Guimarães et al., 2001]. These descriptions are then used to represent the original database as a sequence, from which temporal rules are extracted. Similarly, the work in [Bellazzi et al., 2005] proposes a technique to detect trends in time series from the measurements of patients undergoing haemodialysis, and then produce a description of the precedence of the trend in the form of temporal association rules by using an Apriori strategy and the *precede* temporal relation, which includes the *overlaps, finished-by, meets, before, equals* and *starts* from Allen's temporal algebra. This work is improved in [Sacchi et al., 2007a] by allowing the domain expert to specify a set of trends of interest, which forces every element of the antecedents and consequents of the temporal rules to be part of the set of interest. The same authors in [Sacchi et al., 2007b] build upon the previous research with a new representation formalism by transforming the rules obtained into precedence temporal networks in the domain of gene expression data. The temporal association rule extraction is improved in [Concaro et al., 2009] by allowing the user to specify which event classes can appear in the antecedent or in the consequent of the rules, and then use a post-processing step that further prunes the number of rules produced by introducing a criterion based on the ratio of the confidence of a rule and the maximum confidence of all of its subrules.

2.7 Our approach

As we have previously stated, adopting a temporal formalism provides a set of formal tools to elicit and manage effective problem solving knowledge. The formalism of choice

needs to allow us to integrate mining models with inference models, as well as allow the user of the mining algorithms to interact with the process by providing patterns of interest to focus the induction process and constrain the search space.

Our choice of the STP formalism is not only based on the inference mechanisms seen in section 2.1.2, but also on the expressiveness it offers based on representing both precise and imprecise temporal information. Another important property of the STP formalism is that it can be easily understood by domain experts, allowing them to easily introduce knowledge into the mining process. The STP formalism and its extensions have been proposed to represent temporal knowledge in different problems, such as temporal diagnostic reasoning [Wainer and de Melo, 1997, Gamper and Nejdl, 1997, Palma et al., 2006], computerised representation of clinical guidelines [Duftschmid et al., 2002, Anselma et al., 2006], temporal abstraction [Campos et al., 2010], or query answering [Barro et al., 1994, Deshpande et al., 2003, Combi et al., 2009], among others.

There is a precedent of the use of the STP formalism in the temporal data mining field from sequence databases [Dousson and Duong, 1999]. The authors search for temporal patterns over sequences of events that are represented by the STP formalism. An Apriori-like procedure is proposed, where frequent patterns of size i are found before searching for patterns of size $i+1$. The candidate generation in this proposal constructs one, and only one, temporal constraint network for each different set of items. Each constraint in each one of these networks is obtained by means of the union of the constraints connecting the same nodes in every frequent pattern found in the previous iteration. Then, a Floyd-Warshall `ALL-PAIRS-SHORTEST-PATHS()` algorithm verifies that the network is consistent and produces the minimal network. The set of patterns of size two is extracted from the data collection by using a heuristic. The heuristic associates one interval $[a,b]$ to each pair of items. First, all possible constraints that satisfy the occurrences found in the data sequences are built. Then, to choose the appropriate constraint, the algorithm discards a certain percentage of the occurrences of the pair of items, which reduces the amount of noise. Finally, from the remaining constraints, the one that satisfies that both $b-a$ and $max(|a|,|b|)$ are the smallest possible values is chosen. If more than one constraint satisfy the restrictions, then the constraint resulting from the union of all of them is chosen.

This proposal also allows a domain expert to directly introduce some already known patterns into the mining process. The patterns provided by the user are assumed to be frequent. Therefore, any of its subpatterns is frequent, which allows the algorithm to avoid generating

them and verifying their frequency. In turn, the overall number of patterns involved in the process is reduced.

We can identify the following limitations in this proposal. First, given a set of events of interest, the algorithm may extract at most one pattern, that is, at most one temporal constraint network, to represent temporal arrangements of the events in the sequence database. Therefore, there may be temporal arrangements between the same set of events, different than the ones represented by the pattern chosen yet frequent on their own, that are not taken into account by this algorithm, so some information would be lost. Moreover, each constraint in a pattern is obtained by the union of the constraint in every pattern found in the previous iteration, even if the intervals of the contraints are disjoint, which would allow the new pattern to accept temporal arrangements that were not allowed by the original constraints. In addition, the method proposed to obtain the first constraints is domain-specific. Finally, although the method allows the domain expert to provide patterns to the mining process, any pattern obtained will be subjected to the same previously stated restrictions of the algorithms. Therefore, even though the search space is reduced, and the patterns obtained will be consistent with the knowledge provided, the results could be incomplete if several distinct temporal arrangements are present in the data.

The work by Dousson and Duong can be considered a first and tentative approach to representing the result of a temporal data mining process by means of the STP formalism, but as it has been shown, they provide a naive answer to some problems. This analysis leads us to propose new algorithms to discover frequent temporal patterns represented as STP, providing a more satisfactory answer to those still pending challenges related to computational efficiency, expressiveness, domain knowledge inclusion, and interaction with the user.

CHAPTER 3

DEFINITIONS

In this chapter we introduce the main notions that will be used in the temporal data mining algorithms we have developed for the discovery of frequent temporal patterns expressed as a set of STP. Formal definitions are provided for the temporal primitives (events and episodes) and for the temporal patterns, as well as other concepts and operations that are needed in the implementation of the mining procedures.

We denote as $O = \{o_1, o_2, ..., o_n\}$ the finite set of *observables* in a given domain, that is, those entities in the domain for which an *observation procedure* is available. Every observation procedure identifies the presence of an observable in a temporal instant, and provides a measure for a set of its attributes. We will focus on a set of observables that are significant for analysis purposes; for instance, in the medical domain those with a pathophysiological meaning: an apnea, a tachycardia, or an arousal, and we will name *event* the result of an observation in a time instant. Formally:

Definition 1 *An **event** e is a tuple $(o, a = v, t)$, where $o \in O$ is an observable, a is an attribute of the observable with value $v \in V(a)$, $V(a)$ represents the set of possible values for the attribute a, and $t \in \mathbb{N}$ is a time instant.*

We assume that events are observed in an absolute dating system, possibly expressed with multiple granularities (year, month, day, etc.), and then every time instant is mapped in a fixed granularity representation. We assume that granularity to be second [Bettini et al., 1998b]. On the other hand, with this definition, an event has no duration. Those entities with duration could be represented by the notion of *episode*. Formally:

Definition 2 *An **episode** g is a tuple $(o, a = v, t_b, t_e)$, where $o \in O$ is an observable, a is an attribute belonging to the observable with value $v \in V(a)$, $V(a)$ represents the set of possible values for an attribute a, and $t_b, t_e \in \mathbb{N}$ are time instants representing the beginning and end of the episode, respectively.*

In practice, and for temporal data mining purposes, episodes $(o, a = v, t_b, t_e)$ will be represented by two events $(o_b, a = v, t_b)$ and $(o_e, a = v, t_e)$, marking the beginning and the end of the episode. We will assume in the following, in order to simplify notation, that all events and episodes are characterized by a single attribute, and thus only its value v will be represented.

From the set of different (o, v) pairs in the data, we can define an event type as follows:

Definition 3 *An **event type** E is a tuple (o, v, T), where T is a temporal variable representing an instant. We call $\mathcal{E} = \{E_1, \ldots, E_p\}$ to the set of different event types provided by the observation procedures in a particular domain.*

Intuitively, an event (o, v, t) corresponds to an instantiation or occurrence of an event type (o, v, T) in a particular instant $T = t$.

Definition 4 *An **episode type** G is a tuple (o, v, T_b, T_e), where T_b and T_e are temporal variables representing the beginning and the end of the episode. Two event types (o_b, v, T_b) and (o_e, v, T_e) describe the beginning and the end of every episode type $G = (o, v, T_b, T_e)$. We call $\mathcal{G} = \{G_1, \ldots, G_q\}$ to the set of different episode types provided by the observation procedures in a particular domain.*

We gather the set of event types $\mathcal{E}$ and the set of episode types $\mathcal{G}$ in a set of types of *facts* $\mathcal{F} = \mathcal{E} \cup \mathcal{G}$. The set of types of facts represents all entities of interest for the pattern discovery. As a result of the application of the observation procedures we can obtain a set $\mathbf{E}$ of events and a set $\mathbf{G}$ of episodes, gathered together in a set of facts $\mathbf{F} = \mathbf{E} \cup \mathbf{G}$, that will undergo the mining process. These facts will be represented as an event sequence by defining an order relation among events:

Definition 5 *Let $<$ be an **order relation** between two events $e_i = (o_i, v_i, t_i)$ and $e_j = (o_j, v_j, t_j)$ such that $(e_i < e_j) \Leftrightarrow (t_i < t_j) \vee ((t_i = t_j) \wedge (o_i < o_j))$, assuming a lexicographical order between observable names.*

This relation allows us to define the concept of event sequence.

Definition 6 *An **event sequence** is an ordered set of events $S = \{e_1, \ldots, e_m\}$ where for all $i < j$, $e_i < e_j$. The size of the sequence is $|S| = m$. Its beginning instant is $b_S = t_1$, its ending instant $e_S = t_m$ and its duration $d_S = t_m - t_1$. Every subset of a sequence is a subsequence.*

Since an event-based representation of an episode does not permit to distinguish two different episodes in a sequence, we will provide the mining process with both a sequence S and the set $\mathbf{G}$ of the episodes it contains. The set $\mathbf{G}$ connects the beginning and the end of each episode in the sequence.

A *recording* $(S, \mathbf{G})$ is defined as an enriched event sequence, involving both events and episodes, observed during a given time period. In practice, the set $\mathbf{G}$ is simply implemented as a set of links between the beginning and ending events of each episode in the sequence, so for the sake of simplicity we will denote a recording as a sequence S. An example of a recording is the set of events identified from a patient's polysomnography during the night, or the set of events identified from a patient's long term ECG monitoring, with the only constraint that it is impossible for two different events of the same observable to occur at the same instant, that is, no attribute may take different values simultaneously: $\forall e_i, e_j \in S, (o_i = o_j) \wedge (v_i \neq v_j) \Rightarrow t_i \neq t_j$; although it is possible to identify events corresponding to different observables at the same instant.

A *collection* is a set of recordings $\mathcal{S} = \{S_1, \ldots, S_n\}$, that will undergo a data mining process. As we are interested in mining relative temporal information, we subtract in every recording its onset time from the time-stamp of each event. As an example, a patient underwent a polysomnographic test beginning on 2010/04/12-23:42:16, and a central apnea episode was identified from 2010/04/13-00:02:26 until 2010/04/13-00:02:53. We can represent the notion of central apnea episode by an episode type (*apnea,central,*T_b, T_e), and by the corresponding event types (*apnea*$_b$*,central,*T_b) and (*apnea*$_e$*,central,*T_e). We can represent the observed central apnea episode by an episode (*apnea,central,1210,1237*) and by events (*apnea*$_b$*,central,1210*) and (*apnea*$_e$*,central,1237*).

The user of the mining algorithms may be interested in searching for short- mid- or long-term relations between events or episodes in a collection. His/her knowledge of the domain determines the scope of the search, by defining a temporal window of duration ω that scrolls through every recording $S \in \mathcal{S}$ in the search for frequent patterns. This way, the occurrence of a pattern can spread over at most ω time units, so that whichever two events of the occurrence are, at most, ω time units apart.

Definition 7 *A **temporal window** of width ω in a recording S is every subsequence $W = \{e_i,\ldots,e_k\}$ of S such that $t_k - t_i \leq \omega$, $e_i,\ldots,e_k \in S$ and for all $t_j \in [t_i,t_k]$, $e_j \in S \Rightarrow e_j \in W$.*

This window constrains the search by only considering those subsequences $W \subset \mathcal{S}$ where $d_W \leq \omega$, thus limiting the search space [Lu et al., 2000] and increasing the efficiency.

Another important element to be used in the mining algorithms is the concept of temporal association between event types. Temporal associations are used in the identification of sets of event types frequently found together. The different temporal arrangements within these associations will lead to different temporal patterns during the mining process. Formally:

Definition 8 *A **temporal association** of size n in a recording S is an ordered set of event types $A = \{E_1,\ldots,E_n\}$ with $E_i < E_{i+1}$ for all $i = 1,\ldots,n-1$, such that there exists a temporal window $W = \{e_1,\ldots,e_m\} \subseteq S$, with $m \geq n$, where every $E_i \in A$ has an event occurrence $e_j \in W$.*

By introducing the order relation $E_i < E_j \Leftrightarrow (o_i < o_j) \vee ((o_i = o_j) \wedge (v_i < v_j))$, we may represent event types E_i, E_j, E_k with capital letters: (A,T_A), (B,T_B), (C,T_C), maintaining their lexicographical order. When we refer to event types occurring in the same temporal window, we may also omit the temporal component in the notation, so we can identify event types simply by capital letters: A, B, C,... Fig. 3.1 shows an example of two occurrences of a temporal association with event types A, B, C.

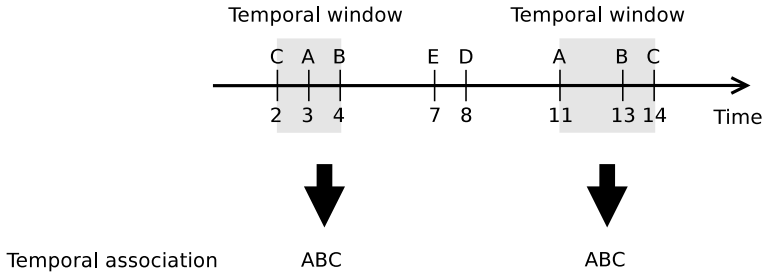


Figure 3.1: Temporal association with event types A, B and C.

The objective of the mining algorithms introduced in the next section is to obtain a set of temporal patterns that are frequently found in a collection $\mathcal{S}$, optionally starting from some previous knowledge provided by the user, that will constrain the search. Every pattern involved in the mining process is represented as a temporal constraint network between a set

of event types, according to the STP formalism [Dechter et al., 1991]. An STP defines a temporal constraint L_{ij} between two event types $E_i = (o_i, v_i, T_i)$ and $E_j = (o_j, v_j, T_j)$ as a closed interval $L_{ij} = [a_{ij}, b_{ij}]$, where a_{ij} and b_{ij} are integer numbers, restricting the possible values of the interval duration between both event types, so that $a_{ij} \leq T_j - T_i \leq b_{ij}$. Formally:

Definition 9 *A **temporal pattern** $P =< D, \mathcal{L} >$ of size n consists of a set of event types $D = \{E_1, \ldots, E_n\} \subseteq \mathcal{E}$, and a set of temporal constraints $\mathcal{L} = \{L_{ij};\ 1 \leq i, j \leq n\}$ between the event types in D.*

A temporal pattern can be represented as a directed graph, where each node is associated with an event type in D, and each arc from node E_i to node E_j is associated with the constraint L_{ij}. Given a constraint $L_{ij} = [a_{ij}, b_{ij}]$ its symmetrical constraint $L_{ji} = -L_{ij} = [-b_{ij}, -a_{ij}]$ contains the same information, so they are redundant. The absence of any explicit constraint between a pair of event types E_i and E_j is equivalent to considering a constraint $L_{ij} = L_U = (-\infty, \infty)$ defined as the *universal constraint*, that is, the constraint that does not restrict the possible values of the interval duration between both event types. Universal constraints are not usually represented in the graph. Given a temporal pattern P as an STP, there always exists a minimal pattern M equivalent to P, corresponding to the most explicit representation of P. Floyd-Warshall's `ALL-PAIRS-SHORTEST-PATHS()` algorithm is used to check the consistency of P and to obtain its minimal representation [Dechter et al., 1991].

Throughout the mining process the same set of events D can produce different temporal patterns. These patterns will correspond to the same temporal association, but with different temporal constraints between their event types.

Definition 10 *A **pattern occurrence** of a temporal pattern $P =< D, \mathcal{L} >$ is a subsequence $X = \{e_1, \ldots, e_n\}$ of some $S \in \mathcal{S}$ such that, for all $i = 1, \ldots, n$, every e_i is an occurrence of a different event type in D, satisfying all the temporal constraints in $\mathcal{L}$.*

Definition 11 *Given two temporal patterns $P =< D, \mathcal{L} >$ and $P' =< D', \mathcal{L}' >$, we say that P' is an **extension** of P, and we denote it by $P \preceq P'$, if $D \subseteq D'$ and, for all $E_i, E_j \in D$, $L'_{ij} \subseteq L_{ij}$ where $L_{ij} \in \mathcal{L}$ and $L'_{ij} \in \mathcal{L}'$.*

The temporal pattern P' usually contains an extended set of the events in P, but its temporal constraints are more restrictive [Dousson and Duong, 1999]. We can extract an occurrence of P from any occurrence of P'. Given an episode type $G = (o, v, T_b, T_e)$, the relation $G \preceq P$ holds if both event types $E_b = (o_b, v, T_b)$ and $E_e = (o_e, v, T_e)$ belong to D, assuming that an episode

type is formally equivalent to a temporal pattern $G =< D_G, \mathcal{L}_G >$ involving both ending event types $D_G = \{E_b, E_e\}$, and a temporal constraint $\mathcal{L}_G = \{L_{be} = [1, \infty)\}$ between them, forcing an episode to begin before it ends.

Definition 12 *Given a temporal pattern $P =< D, \mathcal{L} >$, its **extension over** the set D', being $D \subset D'$, is a new temporal pattern $P^{\uparrow D'} =< D', \mathcal{L}^{\uparrow D'} >$, where $\mathcal{L}^{\uparrow D'} = \{L_{ij}^{\uparrow D'};\ 1 \leq i, j \leq n\}$ such that $L_{ij}^{\uparrow D'} = L_{ij}$ if $E_i, E_j \in D$, and $L_{ij}^{\uparrow D'} = L_U$ otherwise.*

The searching procedures involve the combination of i frequent temporal patterns of size $i-1$ in order to build a candidate temporal pattern of size i. This is carried out by means of the following associative combination operation:

Definition 13 *Given two temporal patterns $P =< D_p, \mathcal{L}_p >$ and $Q =< D_q, \mathcal{L}_q >$, their **combination**, $P \bowtie Q$, is a new temporal pattern $R =< D_r, \mathcal{L}_r >$, where $D_r = D_p \cup D_q$ and $\mathcal{L}_r = \{L_{ij}^r = L_{ij}^{p\uparrow D_r} \cap L_{ij}^{q\uparrow D_r} | L_{ij}^{p\uparrow D_r} \in \mathcal{L}_p^{\uparrow D_r}, L_{ij}^{q\uparrow D_r} \in \mathcal{L}_q^{\uparrow D_r}\}$.*

The user of the mining algorithms is allowed to introduce some previous knowledge of the domain, describing the kind of patterns the mining process should be focused in, that is, what event types are considered interesting or relevant and which basic constraints must be satisfied between them.

Definition 14 *A **seed-pattern** $K \preceq P$ is a temporal pattern that is provided as starting knowledge for the search procedure for pattern P.*

The use of a seed pattern is not necessary to carry out the mining process, but we will show how an adequate seed can substantially improve the computational efficiency. Some of these initial constraints may be trivial; for instance, those defining a pair of event types E_b and E_e as the beginning and ending of an episode type G, so we can consider the set of episode types as a set of seed patterns. Seed patterns provided by the user undergo a consistency checking process in order to obtain the minimal equivalent representation. In case of any of the minimal constraints exceeding the window width, the user will be given the option to extend the window width or modify the seed pattern. It is important to note that any universal constraint in a seed-pattern is reduced to the interval $[-\omega, \omega]$ by the search procedures, being ω the window width. However, every constraint $L_{ij} = [-\omega, \omega]$ resulting from the mining process cannot be understood as a universal constraint.

The criterion used to consider whether a temporal pattern in a collection could possibly be interesting or not is its frequency. Different frequency measures for pattern occurrences can be found in the bibliography. [Mannila et al., 1997] define the frequency of a pattern as the ratio of windows among the total where the pattern can be found, so more compact patterns will have higher frequency values than those patterns whose events are spread over the window. [Bettini et al., 1998a] define the frequency of a pattern as the ratio of the number of occurences of the pattern with respect to the number of occurences of a reference event type, the root event of the pattern. [Dousson and Duong, 1999] consider the frequency of a pattern as the total number of ocurrences found in the data. In many references [Agrawal and Srikant, 1995, Chen et al., 2003, Giannotti et al., 2006, Pei et al., 2004, Chen et al., 2011] a pattern is considered frequent if it appears at least in a certain percentage of the sequences or transactions that constitute the data collection. [Moerchen and Ultsch, 2007] define the support of an interval-based pattern as the sum of the durations of the non-overlapping intervals that constitute the occurrences of the pattern. [Ale and Rossi, 2000] consider the frequency of a pattern as the fraction of transactions where it can be found with respect to its lifespan. Without loss of generality, we use as a definition of frequency the notion of support:

Definition 15 *The **frequency** of a temporal pattern P in $\mathcal{S}$ is defined as the number of occurrences of P in $\mathcal{S}$ and is denoted by $f(P)$.*

We could further specify the frequency calculation by taking into account the time fraction in which the pattern occurs, as in [Mannila et al., 1997]. Other definitions of frequency are also admissible. The domain usually establishes the most suitable design for frequency calculation, and also the minimum frequency that makes a pattern possibly interesting. Given a frequency threshold f_{min}, we say that a temporal pattern P is frequent if $f(P) \geq f_{min}$. Figure 3.2 shows an example of two different temporal patterns, both with the same set of events $D = \{A, B, C\}$ but with different constraints among them, obtained from a collection of event sequences when the frequency threshold is set to $f_{min} = 2$.

For every pair of event types involved in the mining process a frequency distribution of their temporal distances in the collection is calculated, in order to discriminate those similar temporal arrangements which are frequent.

Definition 16 *Given a window width ω and two different event types $E_i = (o_i, v_i, T_i)$ and $E_j = (o_j, v_j, T_j)$ in a recording, we define a **temporal distance distribution** $\{n_{ij}\}$ as a frequency distribution counting the number of occurrences of every different temporal distance $T_j - T_i$*

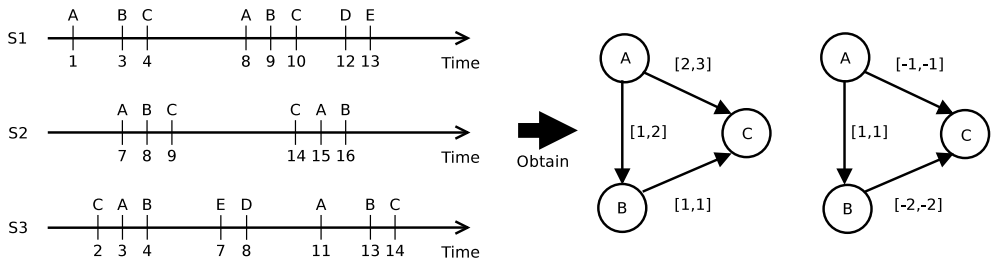


Figure 3.2: An example of two frequent temporal patterns represented as STP, discovered in a collection of sequences, with $f_{min} = 2$.

found in the collection, with a maximum separation of ω *units; that is, the temporal distance distributions are defined on the* $[-\omega, \omega]$ *axis.*

A clustering algorithm will be used to determine from each temporal distance distribution the frequent temporal arrangements according to a provided similarity criterion.

The next two chapters describe the temporal data mining algorithms that will allow us to discover a set of frequent temporal patterns expressed in the STP formalism from a collection of sequences of events and episodes.

CHAPTER 4

ASTPMINER

In this chapter we will introduce `ASTPminer`, Apriori Simple Temporal Problem miner, a temporal data mining algorithm that searches for frequent temporal patterns on databases consisting of a collection of sequences of both events and episodes. Temporal patterns will be represented as metric temporal constraint networks for a set of events using the STP formalism, where precise or imprecise information could be induced between each pair of events represented in the network.

Apriori-like algorithms are characterized by searching for frequent patterns of increasing size following an iterative procedure. Each iteration is divided in three different steps: candidate generation, frequency calculation and removal of those candidates whose frequency is below a minimum frequency threshold. Following this procedure, Apriori-like algorithms perform a breadth-first search, where all frequent patterns of size i need to be identified before searching for patterns of size $i+1$. Pattern size is determined by the number of events it represents.

The Apriori strategy is based on the premise that if a pattern is frequent, then all of its subpatterns are also frequent. Furthermore, if a certain pattern is infrequent, then it is not necessary to generate any of its extensions, because they cannot be frequent. This property is known as antimonotonicity property. For a pattern to be considered frequent, its frequency must be above a minimum threshold that users must establish according to their knowledge of the domain and the nature of the data.

In order to focus the mining process `ASTPminer` allows the user to introduce previous domain knowledge in the form of seed patterns that are also expressed as STP. As we will see, the use of seed patterns prunes the search space in two ways. On the one hand, seed

patterns are used to constrain the total number of distinct temporal arrangements of pairs of event types, because only those arrangements consistent with the seed patterns are considered. On the other hand, the procedure only generates candidates that are extensions of the seed patterns, so the total number of candidates is reduced, but the interest for the end user is increased.

4.1 Basic algorithm

Before introducing the full version of `ASTPminer`, we will describe in this section a basic version of the algorithm, that we labelled `ASTP_BASIC` [Álvarez et al., 2010], in order to introduce the main stages in the Apriori-like mining procedure: candidate generation and frequency calculation. In this `ASTP_BASIC` algorithm, previous knowledge of the domain is only used for setting two of the parameters of the algorithm before it is executed. First, the user must provide the maximum temporal extension allowed for a pattern to be counted: the width ω of the temporal window that defines how far apart two events can be to be considered part of the same pattern. The other parameter provided by the user is the frequency threshold f_{min}, i.e., how many times a pattern must occur in the collection to be considered frequent and, therefore, most likely interesting.

Given a collection of event sequences $\mathcal{S}$, a collection of event types $\mathcal{E}$, and the window width ω and frequency threshold f_{min} provided by the user, `ASTP_BASIC` iteratively searches in $\mathcal{S}$ for frequent temporal patterns of increasing size where events are, at most, ω time units apart. The result is a set of frequent temporal patterns P that represent common temporal arrangements between events found in the collection. No episodes are considered in this version of the algorithm, that is shown in figure 4.1.

`ASTP_BASIC` uses three lists to store the temporal patterns through their different states: list A^i holds frequent temporal associations comprised of i events, list C^i stores candidate patterns of i events, and list P^i represents frequent patterns of i events. The mining procedure is divided into several steps. The first step searches for event types in $\mathcal{E}$ that are frequent in $\mathcal{S}$ (line 1 in figure 4.1), which are stored in the list A^1, representing temporal associations of size 1, and in the list P^1 since they also represent frequent temporal patterns of size 1. Once frequent event types are found, an iterative process searches for frequent patterns of increasing size i until no new frequent patterns are found (line 5). Each iteration i follows two steps: candidate pattern generation (line 6) and frequent pattern calculation (lines 7-8). In each iteration i, a set of frequent temporal patterns of size i is found. The process ends when

```
    procedure ASTP_BASIC(S, E, ω, f_min)
1   begin
2     A^1 ← {E_j | E_j ∈ E ∧ f(E_j) ≥ f_min}
3     P^1 ← A^1
4     i ← 2
5     while (P^(i-1) ≠ ∅) do begin
6       C^i ← CANDIDATE_GENERATION(A^(i-1), P^(i-1))
7       P^i ← FREQUENCY_CALCULATION(C^i, S)
8       A^i ← {D_j | P^i_j=<D_j, L_j> ∈ P^i}
9       i ← i+1
10    end;
11  end;
```

Figure 4.1: ASTP_BASIC: main algorithm.

no new frequent patterns are found in a given iteration. The output of the mining process after this final iteration consists of a set of frequent temporal patterns of size i, stored in list P^i.

4.1.1 Candidate generation

Figure 4.2 presents the CANDIDATE_GENERATION() procedure for iteration i. This procedure receives the list of frequent temporal associations A^{i-1} and frequent patterns P^{i-1} found in the previous iteration, and produces the list of candidate patterns C^i for the iteration in course.

The first step in the generation of a candidate pattern of size i (line 4 in figure 4.2) consists of building a temporal association A^i between the event types in the pattern. Following an Apriori strategy, a temporal association of size i is built from the union of two frequent temporal associations of size $i-1$ that have all event types in common except the last one. Then, the procedure checks if all subsets of size $i-1$ of event types of the new temporal association, that were already calculated in the previous iteration of the main algorithm, are frequent. If A^i contains any non frequent temporal association, then no pattern with those event types can be frequent. For example, given the set of frequent temporal associations of size 3: $A^3 = \{ABC, ABD, ACD, BCD, BCE\}$, the temporal association of size 4 $\{ABCD\}$ could be frequent, whereas $\{BCDE\}$ must be infrequent because some of the associations of size 3 it contains (e.g. $\{CDE\}$) are not frequent.

```
     procedure CANDIDATE_GENERATION(A^{i-1}, P^{i-1})
1    begin
2      C^i ← ∅
3      h ← 1
4      for A_j^{i-1}, A_k^{i-1} ∈A^{i-1} ∧ E_{j1} = E_{k1}, ..., E_{j_{i-2}} = E_{k_{i-2}}, E_{j_{i-1}} < E_{k_{i-1}} do begin
5        A_h^i ← A_j^{i-1} ∪ {E_{k_{i-1}}}
6        h ← h+1
7        if all A_j^{i-1} ⊂A_h^i satisfies A_j^{i-1} ∈A^{i-1} then
8          for all combination of P_{h_k}^{i-1}, P_{h_k}^{i-1} ∈P^{i-1} ∧ D_{h_k}^{i-1} ⊂A_h^i do
9            if Q_l^i =ALL-PAIRS-SHORTEST-PATHS(⋈_h P_{h_k}^{i-1})
             is consistent then
10             C^i ← C^i ∪ {Q_l^i}
11     end;
12     return(C^i)
13   end;
```

Figure 4.2: CANDIDATE_GENERATION algorithm.

The second step in the procedure consists of the combination (line 7) of i frequent patterns $P_{h_k}^{i-1}$ of size $i-1$, representing subpatterns of a new candidate of size i, one for each frequent temporal association A_k^{i-1} verified in the previous step. According to definition 13, the result of combining two patterns is a new pattern whose constraints are obtained by the intersection of the constraints between common event types in both patterns, assuming the universal constraint when no constraint is specified. If any of the resulting constraints is given by the empty set, the pattern is inconsistent and any further combination involving it is bound to be inconsistent. If the resulting pattern contains no empty constraints then it is combined with the next pattern, until no further patterns $P_{h_k}^{i-1}$ with different event types $D_h^{i-1} \subset A_j^i$ can be combined.

Figure 4.3 shows an example of pattern combination, where a pattern of size 4 containing the event types A, B, C and D is obtained after the combination of four patterns of size 3, with event types {A,B,C}, {A,B,D}, {A,C,D} and {B,C,D}, respectively. The combination of the first two patterns (first row in figure 4.3) provides a pattern of size 4, whose constraints are the result of the intersection of the constraints in the combined patterns. Then, this size-4 pattern is combined with the next pattern of size 3 available (event types {A,C,D}). This combination does not add new event types to the pattern, but the resulting constraints are more specific:

constraint between event types C and D changes from the universal constraint in the original pattern (not depicted in the figure) to the interval [8,79] in the resulting pattern. Finally, the remaining pattern of size 3 (event types {B,C,D}) is combined with this pattern of size 4, producing the final pattern shown in the last row of the figure, where the constraint between B and D becomes more specific, since it changes from interval [-79,-12] to interval [-79,-33].

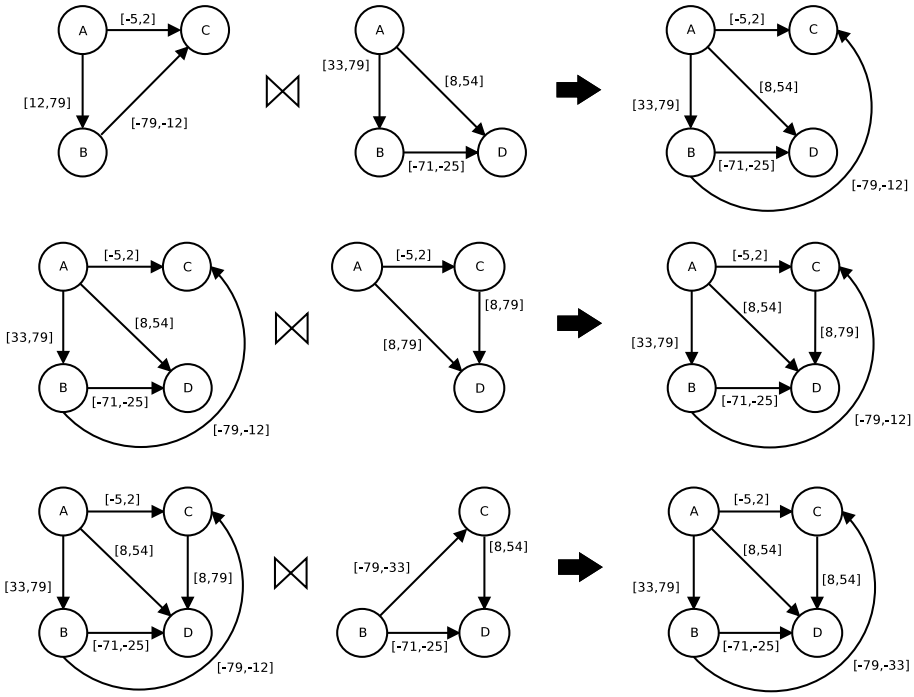


Figure 4.3: Pattern combination example.

After the combination step is finished, the consistency of the candidate pattern is checked by a Floyd-Warshall `ALL-PAIRS-SHORTEST-PATHS()` algorithm [Dechter et al., 1991]. This algorithm applies constraint propagation through the graph of the candidate pattern, obtaining minimal temporal constraints representing consistent temporal arrangements between the event types of the new pattern. If the pattern is consistent, then it is added to the list C^i of candidates that will be searched for in the collection during the frequency calculation step. If the network is not consistent, no occurrence of the pattern can be found and therefore the pattern is discarded.

Candidate generation during iteration $i = 2$ does not follow this schema. The combination step is not performed, as candidates consist of temporal associations of pairs of event types, and there are no known distinct temporal arrangements between event types at this point: constraints in patterns of size 2 are obtained during the frequency calculation step described in section 4.1.2. At this stage, the order of occurrence of the event types during the search is not relevant, since it is only required that the temporal distance between them is at most ω time units.

The number of frequent patterns for a given set of event types can be arbitrarily large, leading to a potentially large number of possible combinations during the candidate generation procedure. Nevertheless, if the result of the combination by the intersection set operation is the empty set, then the resulting pattern, and any other possible candidate pattern including those subpatterns, are inconsistent, and will be discarded in the mining process. Once all candidate patterns for the present iteration have been generated their frequency in the data collection is calculated by the `FREQUENCY_CALCULATION()` procedure.

4.1.2 Frequency calculation

The `FREQUENCY_CALCULATION()` procedure shown in figure 4.4 searches for occurrences of the candidates C^i in the data collection $\mathcal{S}$. Events are sequentially read from each sequence $S \in \mathcal{S}$ using a window W of width ω that moves along each sequence, completely processing one sequence before processing the next one.

With each new event e_j read from a sequence S, the temporal window W is updated. All events e_k in W that now satisfy $t_j - t_k > \omega$ lie outside the bounds of the current temporal window, so they must be removed from it (line 5 in figure 4.4). Then the new event e_j is added to the window (line 6): $W = \{e_{w_1}, ..., e_{w_n} = e_j\} \subseteq S$. Once the temporal window W is updated with event e_j, those candidate patterns that contain its corresponding event type E_j are checked for occurrences in W (line 7). For each candidate C_l^i, all different subsequences X of the temporal window W ending with e_j and containing an event occurrence of the rest of event types in D_l^i need to be checked against the constraints specified by the candidate (lines 8-9). Every subsequence that satisfies all the constraints represents a candidate pattern occurrence, and the temporal arrangement between each pair of events e_{k_p} and e_{k_q} in that occurrence is included in the temporal distance distribution between their event types $\{n_{k_p k_q}\}$ (line 10). The `UPDATE_DISTRIBUTIONS()` procedure increases by one the number of occurrences of the temporal distance between each pair of events e_{k_p} and e_{k_q} in the occurrence in the temporal

```
   procedure FREQUENCY_CALCULATION(C^i, S)
 1 begin
 2   for S ∈ S do begin
 3     W ← ∅
 4     for e_j ∈ S do begin
 5       W ← W − ({e_k | t_j − t_k > ω})
 6       W ← W ∪ {e_j}
 7       for Q_l^i =< D_l^i, L_l^i >∈ C^i do
 8         for X = {e_{k_1}, ..., e_{k_i} = e_j} ⊆ W ∧ {E_{k_1}, ..., E_{k_i} = E_j} = D_l^i do
 9           if (∀e_{k_p}, e_{k_q} ∈ X, t_{k_p} − t_{k_q} ∈ L_{k_p k_q} ∈ L_l^i) then begin
10             UPDATE_DISTRIBUTION(N = {n_{k_p k_q}}_{p,q}, X)
11             f(Q_l^i) ← f(Q_l^i)+1
12           end;
13     end;
14   end;
15   if i = 2 then P^i ← CLUSTERING(N = {n_{k_p k_q}}_{p,q}, f_min)
16   else P^i ← {Q_l^i | Q_l^i ∈Q^i ∧ f(Q_l^i) ≥ f_min}
17   return(P^i)
18 end;
```

Figure 4.4: FREQUENCY_CALCULATION algortihm.

distance distribution $n_{k_p k_q}$. These temporal distance distributions, one for each pair of events, are represented as a histogram, where the horizontal axis corresponds to the possible temporal distances between the pair of events occurrences, and is restricted to the interval $[-\omega, \omega]$, since events in a pattern can be at most ω time units apart. The sign represents the order in the occurrences: positive values mean that e_{k_p} occurs before e_{k_q}, and negative values indicate that e_{k_p} occurs after e_{k_q}.

Figure 4.5 shows an example of temporal distance distribution for a pair of event types A and B, where the horizontal axis represents the temporal distances between the event occurrences of A and B in the collection, and the vertical axis represents the number of times each temporal arrangement happens. The window size is 80 seconds.

After updating the temporal distance distributions, the frequency of the candidate is increased by one (line 11 in figure 4.4). Once all events of the collection are read, the algorithm filters out those candidates that do not satisfy the frequency threshold f_{min} and removes them

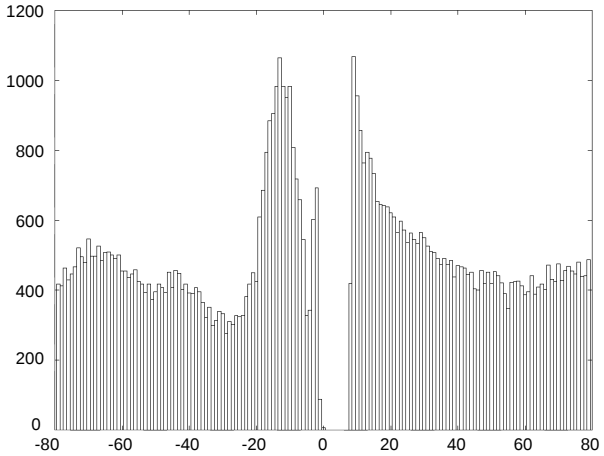


Figure 4.5: Temporal distance distribution between two event types A and B with $\omega = 80$.

from the mining procedure (line 16), returning only those candidates that are considered frequent.

`FREQUENCY_CALCULATION()` presents one difference when the candidates consist of a pair of event types ($i = 2$). In this case, the frequent patterns returned by the procedure are obtained by clustering all previously built temporal distance distributions $\mathcal{N} = \{n_{k_p k_q}\}_{p,q}$ (line 15 in figure 4.4). The clustering of a temporal distance distribution $\{n_{k_p k_q}\}$ produces a set of non-overlapping intervals $I_1, \ldots, I_m$. These intervals concentrate the most frequent temporal arrangements between events e_{k_p} and e_{k_q}, and therefore represent m different constraints between each pair of event types E_{k_p} and E_{k_q}, which results in m different frequent patterns of size two, with $D = \{E_{k_p}, E_{k_q}\}$. The clustering procedure itself is explained in section 4.1.3.

Figure 4.6 shows an example of the `FREQUENCY_CALCULATION()` procedure. Let the window width be $\omega = 8$, and the set of event types be $\mathcal{E} = \{E_1 = A, E_2 = B, E_3 = C, E_4 = D, E_5 = E\}$. Let $C =< D, \mathcal{L} >$ be a candidate pattern, where $D = \{A, B, C, D\}$ and $\mathcal{L} = \{L_{12} = [1,2], L_{13} = [2,3], L_{14} = [3,4], L_{23} = [1,1], L_{24} = [2,3], L_{34} = [1,2]\}$. The sequence where the candidate will be searched for is $S = \{(A,1), (B,2), (A,5), (B,6), (C,7), (D,9), (E,10)\}$. The procedure sequentially introduces the events into the temporal window, finding no occurrence of the candidate pattern being searched for until the event $(D,9)$ is read. This situation is labelled as "Temporal window 1" in the figure. After this event is read, the window contains at least one event of each event type in the candidate pattern, and it is necessary to check

all possible subsequences of the temporal window searching for occurrences of the pattern. According to line 8 of the procedure, only those subsequences where event $(D,9)$ is present need to be checked. The only subsequences satisfying this condition are $X_1 = \{(A,1), (B,2), (C,7), (D,9)\}$, $X_2 = \{(A,1), (B,6), (C,7), (D,9)\}$, $X_3 = \{(B,2), (A,5), (C,7), (D,9)\}$ and $X_4 = \{(A,5), (B,6), (C,7), (D,9)\}$. Sequence X_1 does not satisfy the constraints L_{13}, L_{14}, L_{23} and L_{24} of the pattern and therefore it does not represent an occurrence of this pattern. Sequence X_2 does not satisfy the constraints L_{12}, L_{13} and L_{14}. Sequence X_3 does not satisfy L_{12}, L_{23} and L_{24}. However, the events in sequence X_4 satisfy all the constraints, so this sequence represents a pattern occurrence, and therefore both the frequency of the candidate and the temporal distance distributions between the event types in the candidate need to be updated (lines 10-11 in figure 4.4). The procedure then reads the next event from the event sequence, $(E,10)$. Reading this event forces the temporal window to move to the position labelled as "Temporal window 2". In this position, the event $(A,1)$ is no longer present in the temporal window. Then, the event $(E,10)$ is introduced into the temporal window and, as the candidate pattern does not include the event type E it is not necessary to verify if there are any occurrences in this temporal window. As there are no events left to read in the sequence, the procedure ends.

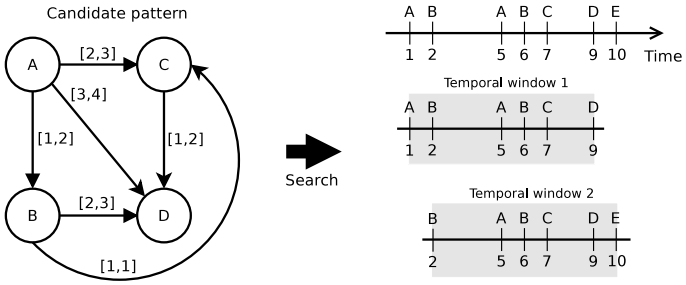


Figure 4.6: Frequency calculation example.

4.1.3 Clustering

A clustering procedure is used in the `FREQUENCY_CALCULATION()` algorithm to obtain a set of time intervals constraining the occurrences of pairs of event types. The clusters provided by the procedure represent similar temporal arrangements between pairs of event

types in the sequences. Each cluster corresponds to a temporal constraint between a pair of event types, thus producing a pattern of size 2.

Any clustering algorithm that can be applied to unidimensional domains could be used in this step. In our proposal, we have used an adaptation of the clustering method described in [Yager and Filev, 1994], due to its simplicity. In this clustering method, the object space is discretized by transforming the space into a grid and placing the objects in the intersection points of the grid. These intersection points are the possible prototypes of the clusters. The gridding establishes a compromise between the number of computations and the precision: the finer the grid the more precise the results at a higher computational cost. The clustering process is guided by a density function, defined on the nodes of the grid, that measures the density of objects near each intersection point. The greater the number of objects that can be found near an intersection, the higher the function value. Once the density function is calculated for all points in the grid, the first prototype is chosen at the point where the function takes its maximum value. Then, the density function is updated to subtract the influence of this prototype in the rest of the points in the grid. The clustering procedure continues to select prototypes and update the density function until the ratio between the function value in the first prototype and the function value in the last prototype falls below a certain threshold established by the user.

In our mining algorithms the clustering procedure is applied to the temporal distance distributions representing the number of occurrences of the different temporal arrangements between pairs of event types. Thus the object space corresponds to the set of possible temporal arrangements (represented by the horizontal axis in figure 4.5), that is limited by the size ω of the temporal window provided by the user. This space is already discrete due to the definition of the temporal domain. Since this is a unidimensional domain, a cluster consists of an interval of temporal arrangements. The starting value of the density function for each temporal arrangement is simplified to be the frequency of occurrence of that arrangement in the set of sequences. The reduction of the density function values after establishing a prototype in the original method is also simplified in our approach, since any temporal arrangement already associated with a cluster will have its density function value set to zero for the rest of the procedure.

The algorithm produces clusters iteratively, completely describing one cluster before proceeding to the next one. As usual in clustering procedures, the first step in the definition of a cluster consists of choosing its prototype. In our case, the prototype of the new cluster is the temporal arrangement with the maximum value of the density function. Once the prototype

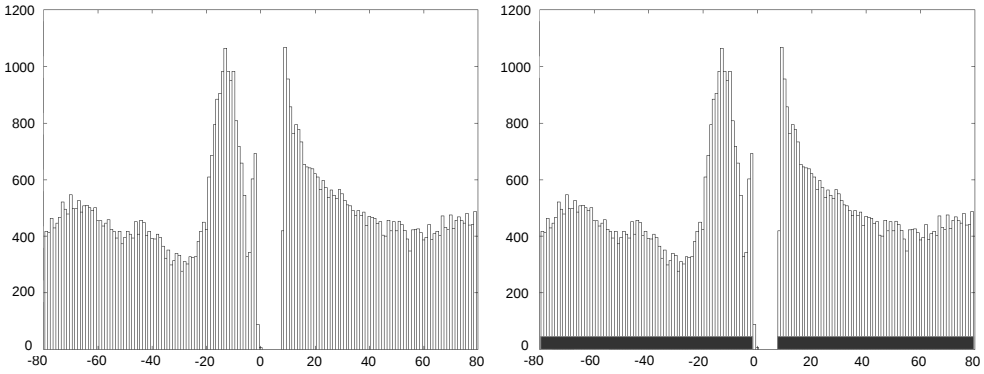
has been chosen, the bounds of the cluster (interval) are placed. Each bound of the interval starts in the same temporal arrangement as the prototype, and is moved to the next temporal arrangement until the density function value is less than a certain percentage of the value at the prototype. This percentage is a parameter of the clustering algorithm. The last step of the iteration consists of updating the density function values of the temporal arrangements that are included in the new cluster, which are set to zero. It is required that the sum of the frequencies of the temporal arrangements included in each cluster is greater than the frequency threshold f_{min}. If this condition does not hold for the current cluster, then the cluster is ignored. The procedure ends when the density function value of the prototype is lower than a percentage of the absolute maximum of the temporal distance distribution, which is provided as another parameter of the clustering algorithm. If the user so wishes, the results of the clustering procedure may be manually modified, either by removing some intervals, by adding new ones, or by changing the values of the extremes of an existing interval, in order to guide the process.

Figure 4.7a shows an example of temporal distance distribution between event types A and B. Figure 4.5b shows the result of the clustering algorithm over this temporal distance distribution. The clustering algorithm detects that there are two distinct intervals, [-80,7] and [12,80], present in the temporal distance distribution according to the similarity criteria defined in the procedure. Each interval represents the temporal constraint of a different pattern of size 2 with event types A and B. The first of the intervals produces a pattern between events A and B where B can either occur before A, at the same time, or up to 7 seconds after A; whereas the second interval produces a pattern where B can occur between 12 seconds and 80 seconds after A.

4.1.4 Validation of the ASTP_BASIC algorithm

A set of experiments was conducted to test the results of the mining process and the influence of the different parameters involved in our algorithms. We have selected a clinical domain, SAHS, where the medical community has described a well-known phenomenon from the repetitive observation of a temporal pattern in a data set. The experiments aimed to test the ability of our proposal to induce similar results from the same data. All experiments were conducted on a 2.83GHz Intel Core 2 Quad CPU with 8 GB of main memory running a 64-bit Ubuntu distribution.

SAHS is a very frequent sleep-breathing disorder characterised by the presence of apneas, which are interruptions of the respiratory airflow while the patient is sleeping. SAHS is espe-



(a) Temporal distance distribution. (b) Resulting clustering.

Figure 4.7: Example of clustering on a temporal distance distribution.

cially prevalent in adult males with obesity problems and is recognised as an important public health issue [Flemons, 2002].

The origin of an apnea can be central, which is caused by abnormal operation of the central motoneural system, or obstructive, which is caused by local obstruction or structural deformation of the upper airways. Each episode of apnea causes a decrease in the blood oxyhaemoglobin saturation. Episodes of central apnea and obstructive apnea can be distinguished by analysing the respiration movements at the thorax and at the abdomen. In the case of a central apnea, the movements cease or have only very low amplitudes relative to normal breathing. During obstructive apnea, the obstruction of airways leads to faster respiratory movements that try to overwhelm the obstruction. Additionally, an episode of apnea may cause arousals. The result is a disruption of the normal sleep architecture, which reduces its refreshing effects. Consequently, patients usually suffer from day-time drowsiness and cognitive deficits, which increase the risk of accidents in the workplace and when driving vehicles [American Academy of Sleep Medicine Task Force, 1999].

The gold standard test for the diagnosis of SAHS is polysomnography, which is performed in a hospital sleep unit and consists of recording a wide range of physiological parameters while the patient is asleep. The recording is then visually inspected by a physician. In accordance with the criteria of the American Academy of Sleep Medicine, an apnea is understood as being a respiratory pause lasting at least 10 s. A patient suffering from severe apnea may

have up to 500 apneas during the night, where each apnea has an average duration of a little over half a minute [American Academy of Sleep Medicine Task Force, 1999].

The database used for our experiments consists of a collection of sequences, where each sequence gathers the events found in a polysomnography test of a patient diagnosed with SAHS[1]. The collection consists of 50 recordings obtained from 50 patients who underwent a sleep study in the sleep unit of the Complexo Hospitalario Universitario da Universidade de Santiago de Compostela. No distinction between obstructive and central apnea patients was made. The average age of the patients was 53.2 $\pm$ 12.7 (mean $\pm$ standard deviation) years, with a minimum age of 23 and a maximum of 88. The average weight was 89.9 $\pm$ 16.4 kg., with an average body mass index (BMI) of 32.4 $\pm$ 6.1 kg/m^2. The recordings were automatically annotated and then the annotations were reviewed by a pulmonologist [Otero et al., 2011]. The whole collection totals over 120,000 events and 280 hours of sleep. Annotations reviewed by the physician on apnea patterns allow us to validate the results of the mining procedures.

The set of annotated event types in the SAHS collection is $\mathcal{E} = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8\}$:

E_1 : "*start airflow limitation*",

E_2 : "*end airflow limitation*",

E_3 : "*start abdominal movement limitation*",

E_4 : "*end abdominal movement limitation*",

E_5 : "*start thoracic movement limitation*",

E_6 : "*end thoracic movement limitation*",

E_7 : "*start oxygen desaturation*",

E_8 : "*end oxygen desaturation*".

On the one hand, the objective of the validation experiments is to induce a pattern including these event types that represents the accepted knowledge of the apnea pattern. On the other hand, the time required to carry out each step of the algorithms must be minimized. Both the time required by each step of the algorithms, and the time required by the algorithms

[1] http://www.gsi.dec.usc.es/datacollections - Section: Apnea (Accessed: 31 May 2013)

as a whole are measured. The individual step measurements are: frequency calculation, removal of non-frequent candidates, and candidate generation. Candidate generation is further divided into three additional measurements: subpattern selection for the combination step, the combination step itself, and consistency checking using the Floyd-Warshall algorithm. These measurements will be used to decide where to focus later optimization strategies.

Due to the definition of the combination operation, a combinatory explosion is expected during the candidate generation step, making it specially costly. In order to quantify this problem, the following measurements are taken in every iteration:

1. **Combinations**: the number of possible *combinations* of patterns according to the number of frequent patterns found in the previous iteration.

2. **Candidates**: the number of consistent combinations, representing the candidate patterns.

3. **Discarded**: the number of combinations discarded by the Floyd-Warshall calculation.

4. **Frequent**: the number of frequent combinations in the data collection, which correspond to the frequent patterns.

Table 4.1 shows the result of these measurements for the `ASTP_BASIC` algorithm, using a window size of 80 seconds and a frequency threshold of 30 occurrences. For patterns of size 2, there is no combination process in the candidate generation procedure, since these patterns are obtained by the clustering algorithm, so both measurements (combinations and candidates) are not available at iteration 2.

Figure 4.8 shows the time required by `ASTP_BASIC` for different window sizes. As the window size grows, the amount of events included in the window also grows, therefore increasing the amount of possible combinations of events in the window that must be tested in order to find every occurrence of every pattern. For every window size shown in the figure, most of the time required by the algorithm belongs to the frequency calculation step, being the time required by candidate generation and non-frequent pattern removal negligible in comparison. For example, in the case ω =80 s., the candidate generation step takes 40 ms., whereas the frequency calculation step requires 83518 ms.

The `ASTP_BASIC` algorithm would improve its efficiency if those events that do not belong to occurrences of any pattern could be removed from the analysis, thus reducing the number of combinations and candidate patterns generated, that need then to be checked in

Size	Combinations	Candidates	Discarded	Frequent
1	8	8	0	8
2	NA	NA	NA	46
3	239	174	65	165
4	5237	306	1885	303
5	94184	298	5136	297
6	7.9×10^5	162	4218	160
7	1.7×10^6	47	1439	46
8	1.1×10^6	6	198	6

Table 4.1: Number of possible candidates, candidates generated, combinations discarded and frequent patterns found in the SAHS database by `ASTP_BASIC`, with a window size ω=80 s. and a frequency threshold f_{min}=30 occurrences.

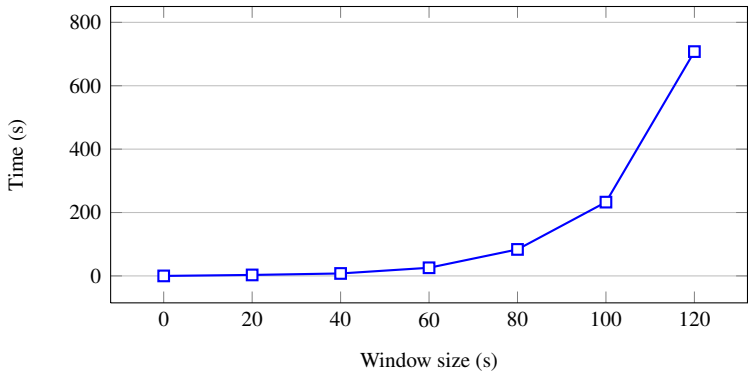


Figure 4.8: Time required by `ASTP_BASIC` depending on the window size.

the frequency calculation step. A modification of this basic algorithm that contemplates this event removal is described in the next section.

4.2 Time optimization: Event removal

As we have mentioned, the `ASTP_BASIC` algorithm would improve its efficiency if we introduce the possibility of detecting those events of the data collection that do not belong to any occurrence of any pattern in a given iteration. These events may be removed from the mining process, as they cannot belong to any pattern occurrence in a later iteration, without changing the results of the mining procedure but increasing its performance. By removing

events from the collection, the number of combinations of events that need to be checked during the frequency calculation step is decreased. Two different approaches are developed to deal with event removal: window-marking and interval-marking.

4.2.1 Event removal using window markings

The first approach for event removal shifts the temporal window through an event sequence in the same manner as `ASTP_BASIC`. The difference appears when reading an event from the collection results in finding an occurrence of a pattern. In this case, every event in the window is marked as useful, independently of whether they belong to the pattern occurrence or not. All events leaving the window without this mark are removed from the collection.

This method is simple and adds few checks to the algorithm, so the overload added to the process is small. However, this approach does not remove from the collection some events that could have been removed: for example, events located at the beginning of the window that do not take part in occurrences of any pattern.

Figure 4.9 shows an example of this event removal procedure, using a window of size 4. We assume, for the sake of simplicity, that the only candidate pattern being searched for is a pattern with event types B and C, that we will call BC pattern. Given the event sequence in the example, the first occurrence of the BC pattern is found when the event (B,4) is read. At this point, all three events C, A, B, in the window are marked as useful. Assuming that no pattern occurrence is found when reading events (E,7) and (D,8) then both events are not marked, and they are both removed from the collection when they leave the temporal window. The second occurrence of the BC pattern is found when reading event (C,14), again marking all three events A, B, C in the window labelled as "Occurrence BC 2" in the figure. Note that in this example, both events (A,3) and (A,11) do not belong to any occurrence of the pattern being searched for, yet they belong to the same temporal window where a pattern occurence is found, and therefore they are both marked and will not be removed from the procedure even when they are of no interest.

Figure 4.10 shows the `FREQUENCY_CALCULATION_WM()` procedure performing the event removal using window marks. This procedure makes use of a list R of the intervals where an occurrence of any pattern has been found. Every time an occurrence of a pattern is identified, the interval $[t_j - \omega, t_j]$ (line 15) is introduced into the list R. This interval covers all the temporal window ending with the event e_j just read, and marks all its events as useful. With every shift of the temporal window to introduce a new event, all events exiting the

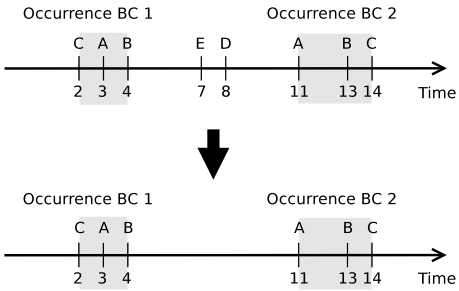


Figure 4.9: Example of event removal using window marking.

window that do not belong to any interval in R are also removed from S (line 6 in figure 4.10), because they do not belong to any pattern. In addition, those intervals that no longer overlap with the current temporal window are removed from R (line 7).

4.2.2 Event removal using interval markings

The second approach for event removal restricts the event marking to a subinterval within the temporal window. For each pattern occurrence found, every event in the window located between the initial event of the occurence and the event read, which will be the ending event of the occurrence, are marked as useful. Those events that leave the temporal window without being marked are removed from the process.

This method adds more comparisons to the process than the first approach, but allows the algorithm to remove more events from the collection. In particular, events located at the beginning of the temporal window not belonging to any pattern occurrence may be removed, whereas the previous approach would mark them as useful. On the other hand, it is possible that some of the events in the marked interval do not belong to any pattern occurence, yet they are still marked and kept in the data collection.

Figure 4.11 shows an example of this event removal procedure, where the pattern being searched for includes again the event types B and C anywhere in the window, using a window of width 4. We assume, for the sake of simplicity, that this is the only candidate pattern. In this example, the first occurrence of the BC pattern is found when the event (B,4) is read. This occurrence includes events (C,2) and (B,4), and all three events in the window are marked as useful, as event (A,3) lies in the time interval between the beginning and ending events of the pattern occurence. The second occurrence of the BC pattern is found when reading event

```
   procedure FREQUENCY_CALCULATION_WM(C^i, S)
 1 begin
 2   for S ∈ S do begin
 3     W ← ∅
 4     R ← ∅
 5     for e_j ∈ S do begin
 6       S ← S − {e_k | t_j − t_k > ω ∧ ∀[t_a,t_b] ∈ R, t_k ∉ [t_a,t_b]}
 7       R ← R − {[t_a,t_b] ∈ R | t_j − t_b > ω}
 8       W ← W − {e_k | t_j − t_k > ω}
 9       W ← W ∪ {e_j}
10       for Q_l^i =< D_l^i, L_l^i >∈ C^i do
11         for X = {e_{k_1},...,e_{k_i} = e_j} ⊆ W ∧ {E_{k_1},...,E_{k_i} = E_j} = D_l^i do
12           if (∀e_{k_p}, e_{k_q} ∈ X, t_{k_p} − t_{k_q} ∈ L_{k_p k_q} ∈ L_l^i) then begin
13             UPDATE_DISTRIBUTION(N = {n_{k_p k_q}}_{p,q}, X)
14             f(Q_l^i) ← f(Q_l^i)+1
15             R ← R ∪ {[t_j − ω, t_j]}
16           end;
17     end;
18   end;
19   if i = 2 then P^i ← CLUSTERING(N = {n_{k_p k_q}}_{p,q}, f_min)
20   else P^i ← {Q_l^i | Q_l^i ∈ C^i ∧ f(Q_l^i) ≥ f_min}
21   return(P^i)
22 end;
```

Figure 4.10: Frequency calculation for event removal using window marking: `FREQUENCY_CALCULATION_WM`.

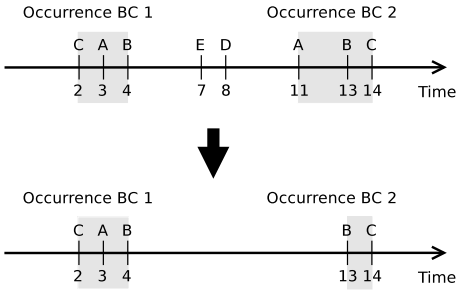


Figure 4.11: Example of event removal using interval marking.

(C,14), including the event (B,13). In this case, the event (A,11) lies outside the interval and therefore it is not marked, and will be removed when it leaves the temporal window. As in the example in figure 4.9, events (E,7) and (D,8) will not be marked, and they are both removed from the collection. For this example, this approach removes one event more than the previous one, yet still cannot remove event (A,3) from the collection. In the case there were more than one candidate pattern to be searched for, events cannot be removed from the collection until all candidate patterns have been checked.

Figure 4.12 shows the frequency calculation procedure modified to perform the event removal using interval marks: `FREQUENCY_CALCULATION_IM()`. It also makes use of a list R of intervals where an occurrence of any pattern has been found. Every time an occurrence of a pattern is identified when reading an event e_j, the interval $[t_{k_1}, t_j]$ (line 15) is introduced into the list R. This interval covers all the events in the temporal window between the earliest event in the candidate pattern to the last event e_j read, and all its events are marked as useful. With every shift of the temporal window to introduce a new event, all events exiting the window that do not belong to any interval in R are also removed from S (line 6). In addition, those intervals that no longer overlap with the current temporal window are removed from R (line 7).

4.2.3 Experimental results using the event removal approaches

The results of applying both event removal strategies were checked against the SAHS database described in section 4.1.4. The `ASTP_BASIC` algorithm (see figure 4.1) was executed for a window size of 80 seconds, but the `FREQUENCY_CALCULATION()` procedure was replaced with the modified procedures `FREQUENCY_CALCULATION_WM()` and `FREQUENCY_CALCULATION_IM()`, respectively. Table 4.2 shows the number of events removed in each iteration by both of the approaches. As expected, the algorithm using `FREQUENCY_CALCULATION_IM()` is more exhaustive regarding the events marked, and removes more events throughout the mining process. In spite of the small difference between the total number of removed events between both approaches for this particular database, `FREQUENCY_CALCULATION_IM()` removes more events in earlier iterations, resulting in a lesser number of combinations of events to check in later iterations. This might reduce the time required by a noticeable amount.

Figure 4.13 displays the time required for the mining procedure on the SAHS database when applying the original `ASTP_BASIC` algorithm, and when we introduce each one of

```
   procedure FREQUENCY_CALCULATION_IM(C^i, S)
 1 begin
 2   for S ∈ S do begin
 3     W ← ∅
 4     R ← ∅
 5     for e_j ∈ S do begin
 6       S ← S − {e_k | t_j − t_k > ω ∧ ∀[t_a,t_b] ∈ R, t_k ∉ [t_a,t_b]}
 7       R ← R − {[t_a,t_b] ∈ R | t_j − t_b > ω}
 8       W ← W − {e_k | t_j − t_k > ω}
 9       W ← W ∪ {e_j}
10       for Q_l^i =< D_l^i, L_l^i >∈ C^i do
11         for X = {e_{k_1},...,e_{k_i} = e_j} ⊆ W ∧ {E_{k_1},...,E_{k_i} = E_j} = D_l^i do
12           if (∀e_{k_p}, e_{k_q} ∈ X, t_{k_p} − t_{k_q} ∈ L_{k_p k_q} ∈ L_l^i) then begin
13             UPDATE_DISTRIBUTION(N = {n_{k_p k_q}}_{p,q}, X)
14             f(Q_l^i) ← f(Q_l^i) +1
15             R ← R ∪ {[t_{k_1}, t_j]}
16           end;
17     end;
18   end;
19   if i = 2 then P^i ← CLUSTERING(N = {n_{k_p k_q}}_{p,q}, f_min)
20   else P^i ← {Q_l^i | Q_l^i ∈ Q^i ∧ f(Q_l^i) ≥ f_min}
21   return(P^i)
22 end;
```

Figure 4.12: Frequency calculation for event removal using interval marking: FREQUENCY_CALCULATION_IM.

the two event removal approaches in the frequency calculation stage. The use of the interval marking strategy produces the best results in this database, as expected.

4.3 Providing seed knowledge

The number of patterns found in every step of the procedure and their frequency when using an event removal algorithm remains unchanged with respect to the initial approach ASTP_BASIC. Both optimization strategies focus on removing from the collection those subsequences no longer relevant for the analysis, as no pattern occurrence can be found in them, thus reducing execution time while not affecting the resulting patterns. But the user of the mining algorithm may not only be interested in its time efficiency. The output of

Pattern size	Window marking	Interval marking
2	28	29
3	4265	4426
4	5173	5227
5	9178	9201
6	12042	12119
7	22842	22853
8	30203	29922
Total	83731	83777

Table 4.2: Number of events removed from the collection in each iteration by both event removal strategies; window size 80 s.

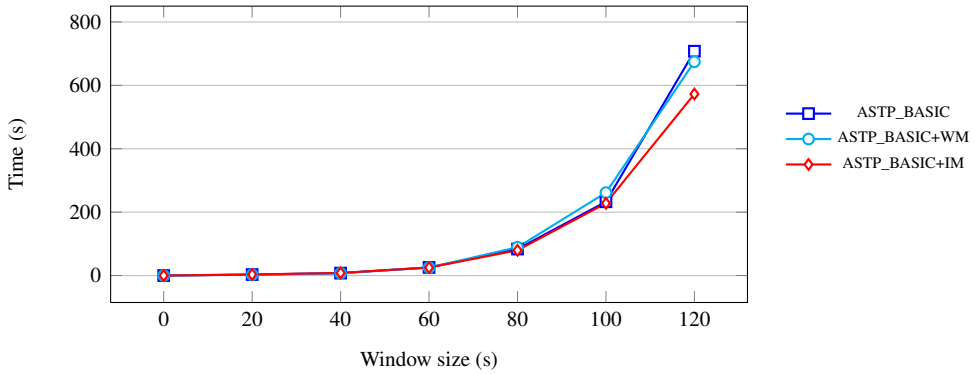


Figure 4.13: Comparison of time required by the `ASTP_BASIC` algorithm without event removal, and with each one of the event removal strategies (WM and IM) during the frequency calculation stage.

the mining process is a set of temporal patterns that were found to be frequent, and therefore possibly interesting for the user. Nevertheless, the degree of interest of a pattern relies not only in its frequency: the pattern should be compatible with the domain knowledge and represent useful and correct information.

Figure 4.14 shows one of the frequent patterns of size 8 found in the SAHS database when we used `ASTP_BASIC` (without any event removal strategy and a window size of 80 seconds). The patterns obtained from this database represent temporal information about patients suffering from SAHS, that is, patients that may suffer several apnea occurrences during their sleep. A central apnea occurrence starts with a limitation in abdominal and thoracic movements that produce an airflow limitation, leading in a few seconds to a subsequent de-

crease in oxygen saturation in blood. When the patient resumes breathing, airflow is restored and oxygen saturation recovers its original level. We can observe in the figure that the pattern found consists of a general description of an apnea pattern, where airflow limitation events, and thoracic and abdominal limitation events occur roughly at the same time, and at least 10 seconds before the oxygen saturation events. However, this pattern presents three flaws that need to be addressed. The first one refers to the temporal imprecision of the airflow, abdominal and thoracic limitations: the ending event of each limitation is simply located after the starting event in the same window. The second flaw is found in the constraint between the "start oxygen desaturation" and "end oxygen desaturation" events, where the interval associated to the constraint accepts that oxygen saturation in blood may recover its normal levels before it falls. The third flaw lies in constraints associating the end of the oxygen desaturation with the rest of the ending events of the limitations, representing situations where the oxygen saturation recovers before the patient resumes breathing.

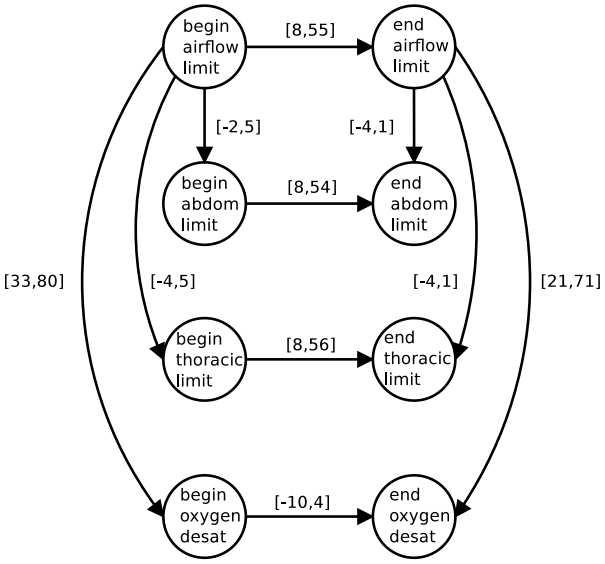


Figure 4.14: Example of a frequent pattern of size 8 obtained from the SAHS database with the `ASTP_BASIC` algorithm.

All these flaws can be explained considering the nature of the apnea pattern. It is common that different apnea pattern occurrences overlap during the night, resulting in the detection of events related to oxygen saturation of a previous episode of airflow limitation while a new

limitation is on course. The algorithm associates these events with both previous and ongoing apneas, resulting in a pattern where the ending of the oxygen desaturation is allowed to happen before normal breathing is resumed (third flaw). This overlapping of occurrences in the temporal window also induces the second flaw, as the algorithm consistently detects that the ending of oxygen desaturation event occurs before the starting event, producing patterns that allow these kind of relations. This situation is also responsible for the first flaw. As previously described, the frequency calculation procedure tries to associate all events present in the same temporal window. Following this procedure, it is possible to associate, for example, the beginning of an airflow limitation located at the beginning of the temporal window, with the ending of a later airflow limitation located at the end of the temporal window, an uninteresting association in this domain. The presence of this kind of association in a temporal window extends the duration of the intervals produced in the clustering procedure, and therefore produces abnormally long constraints.

These problems can be dealt with by introducing the available domain knowledge into the mining process. Using this knowledge it would be possible to reduce the search scope by removing uninteresting temporal arrangements from the process, and focusing on those situations of interest to the domain expert. The user would provide a set of event types and a set of constraints between those event types, and the mining process would only extract frequent patterns consistent with the knowledge provided. This approach is described in detail in this section and deals with the second and third flaws. The first flaw is due to the inability of the frequency calculation procedure to deal with the interval based events that compose an apnea pattern. An approach to deal with this issue is described in section 4.4.

All previous algorithms (`ASTP_BASIC` and both event removal strategies) search for all possible temporal arrangements between events in order to produce frequent patterns between them. However, a domain expert may know that some temporal arrangements are of no interest, or represent noise in the mining procedure. From iteration 2 in the algorithm, candidate patterns of size 2 are generated and then used to produce new candidate patterns that need to be searched for in the data collection. If these candidate patterns are uninteresting or even wrong according to domain knowledge, since this process is repeated in every iteration of the mining process, unnecessary workload is added to the process. Moreover, some uninteresting frequent patterns will be included in the resulting set of patterns.

The algorithm described in this section, `ASTP_SEED`, discovers frequent patterns consistent with some initial knowledge expressed as STP seed patterns, which are extended and/or refined during the course of the algorithm [Álvarez et al., 2011]. The domain expert describes

a set of event types of interest and some constraints between them, forcing the temporal arrangements between the events to satisfy the available domain knowledge. The mining procedure may add new event types and constraints to the seed pattern, or just make the constraints more specific. In the description of this algorithm we will assume, for the sake of simplicity and without losing generality, that the user specifies only one seed pattern.

The structure of ASTP_SEED is shown in figure 4.15, where K represents the seed pattern provided. An INITIALISATION() algorithm is introduced in the procedure: lines 4-5 now obtain the set of frequent patterns of size 2 consistent with the knowledge represented by the seed pattern. The iterative procedure begins after this initialisation step, searching for frequent patterns of size $i > 2$ that are generated by combination of the frequent patterns of size 2 obtained from the initialisation. Both the candidate generation and frequency calculation procedures are modified to take into account the knowledge provided by the seed pattern.

```
    procedure ASTP_SEED(S, E, K, ω, f_min)
1   begin
2     A^1 ← {E_j | E_j ∈ E ∧ f(E_j) ≥ f_min}
3     P^1 ← A^1
4     P^2 ← INITIALISATION(S, E, K, ω, f_min)
5     A^2 ← {D_j | P^2_j=<D_j, L_j>∈P^2}
6     i ← 3
7     while (P^(i-1) ≠ ∅) do begin
8       C^i ← CANDIDATE_GENERATION_SEED(A^(i-1), P^(i-1))
9       P^i ← FREQUENCY_CALCULATION_SEED(S, C^i, ω, f_min)
10      A^i ← {D_j | P^i_j=<D_j, L_j>∈P^i}
11      i ← i+1
12    end;
13  end;
```

Figure 4.15: Mining algorithm using a seed pattern: ASTP_SEED.

4.3.1 Initialisation procedure

Figure 4.16 shows the initialisation procedure that produces frequent patterns of size 2 consistent with the knowledge provided by the user. The first step in the procedure consists of searching for patterns where both event types belong to the seed pattern. The frequency of the

seed pattern in the data collection is checked, and frequent patterns found where both event types are present in the seed pattern are included in set K^2 (line 2). In addition, subsequences of the data collection where no occurrence of the seed pattern can be found are removed from the collection, because those events cannot be used to extend any occurrence of the seed pattern. This event removal can be done with any of the techniques described in section 4.2. The second step obtains frequent patterns of size 2 where at least one of the event types is not present in the seed pattern (lines 3-4).

```
    procedure INITIALISATION(S,E,K,ω,f_min)
1   begin
2     K^2 ← FREQUENCY_CALCULATION_SEED({K},S,ω,f_min)
3     C^2 ← CANDIDATE_GENERATION_SEED(A^1,C^1)
4     P^2 ← K^2 ∪ FREQUENCY_CALCULATION_SEED(C^2,S,ω,f_min)
5     return P^2
6   end;
```

Figure 4.16: Initialisation algorithm used in ASTP_SEED: INITIALISATION.

4.3.2 Frequency calculation with a seed pattern

Figure 4.17 describes the procedure FREQUENCY_CALCULATION_SEED for searching for occurrences of either seed or candidate patterns. The first time the algorithm is called, Q^i represents the set containing the seed pattern $K =< D_K, \mathcal{L}_K >$, where $i = |D_K|$. In succesive steps, it represents the set of candidate patterns C^i. The algorithm sequentially introduces the events of every event sequence S into the temporal window W (line 11). For every event introduced it is necessary to search for occurrences of every pattern Q_l^i in Q^i (line 12) in all possible subsequences X of the temporal window (line 13). Each subsequence satisfying every constraint of the pattern Q_l^i represents an occurrence of the pattern, so its frequency is increased (line 16) and the temporal distance distributions of every pair of event types present in the pattern are updated.

Using a window marking event removal approach, if the occurrence found belongs to the seed pattern, the interval $[t_j - \omega, t_j]$ corresponding to the current window is introduced into the list R (line 17). This interval indicates that an occurrence of the seed pattern was found in

this window, and no event should be removed from it, because any event in the window could be part of an occurrence of an extension of the seed pattern.

Whenever an event leaves the temporal window, it is necessary to check if it can be used in later iterations, removing it from S in case it does not belong to any interval in R (line 7). It is also necessary to remove from R any interval that no longer overlaps with the temporal window, as no further event can belong to them (line 8). After every subsequence in every sequence of the collection has been checked, the procedure uses a clustering procedure over the temporal distance distributions, producing a set of frequent patterns of size 2, when $i = 2$ or $Q^i = \{K\}$ (line 21). In the general case, it removes those candidate patterns not satisfying the minimum frequency f_{min} criteria (line 22) and returns the remaining patterns.

Event removal in the `FREQUENCY_CALCULATION_SEED()` algorithm requires some additional considerations. In this algorithm, whenever an occurrence of the seed pattern is found the whole temporal window is considered to be useful (line 17), as it could be possible to find an occurrence of an extension of the seed pattern in it, using the window marking event removal procedure described in section 4.2.1. However, the domain expert may be only interested in those extensions of the seed pattern where any new event can be found within the interval delimited by the events of the previous occurrences. In that case it would be possible to use the interval marking event removal approach described in section 4.2.2, marking only the actual time interval where the events of the seed pattern occurrences are found. In addition, the event removal function can be used in all iterations of the algorithm if the user so desires.

Figure 4.18 shows an example of the frequency calculation procedure when a seed pattern is defined, using the interval marking event removal strategy. Let K=$< D, \mathcal{L} >$ be the seed pattern specified by the user, where $D = \{A, B, C, D\}$ and $\mathcal{L} = \{L_{12} = [1,2], L_{13} = [2,3], L_{14} = [3,4], L_{23} = [1,1], L_{24} = [2,3], L_{34} = [1,2]\}$. Let the set of event types be $\mathcal{E} = \{E_1 = A, E_2 = B, E_3 = C, E_4 = D, E_5 = E\}$, and the temporal window width $\omega = 8$. The sequence where the seed pattern will be searched for is shown in figure 4.18 a): $S = \{(A,1), (B,2), (A,5), (B,6), (C,7), (D,9), (E,10)\}$. Events are sequentially introduced into the temporal window until event $(D,9)$ is read, labelled as "Temporal window 1" in figure 4.18 b). With this event in the window, it is possible to find an occurrence of the seed pattern in the subsequence $X = \{(A,5), (B,6), (C,7), (D,9)\}$, as these events satisfy every constraint in the pattern. The frequency of the pattern is then increased and the temporal distance distributions for each pair of event types in the pattern are updated. In addition, the interval $[5,9]$ is inserted in R. Now the temporal window moves, and event $(E,10)$ is introduced, forcing event $(A,1)$ to leave the window, as the temporal distance between them is higher than ω (figure 4.18 c)). In

```
     procedure FREQUENCY_CALCULATION_SEED(Q^i, S, ω, f_min)
 1   begin
 2     for S ∈ S do begin
 3       R ← ∅
 4       W ← ∅
 5       for e_j ∈ S do begin
 6         if Q^i = {K} then begin
 7           S ← S − {e_k | t_j − t_k > ω ∧ ∀[t_a,t_b] ∈ R, t_k ∉ [t_a,t_b]}
 8           R ← R − {[t_a,t_b] ∈ R | t_j − t_b > ω}
 9         end
10         W ← W − {e_k | t_j − t_k > ω}
11         W ← W ∪ {e_j}
12         for Q_l^i =< D_l^i, L_l^i > ∈ Q^i do
13           for X = {e_k1, ..., e_ki = e_j} ⊆ W ∧ {E_k1, ..., E_ki = E_j} = D_l^i do
14             if (∀e_kp, e_kq ∈ X, t_kq − t_kp ∈ L_kpkq ∈ L_l^i) then begin
15               UPDATE_DISTRIBUTION(N = {n_kpkq}_p,q, X)
16               f(Q_l^i) ← f(Q_l^i)
17               if Q^i = {K} then R ← R ∪ {[t_j − ω, t_j]}
18             end;
19       end;
20     end;
21     if i=2 ∨ Q^i = {K} then P^i ← CLUSTERING(N = {n_kpkq}_p,q, f_min)
22     else P^i ← {Q_l^i | Q_l^i ∈ Q^i ∧ f(Q_l^i) ≥ f_min}
23     return(P^i)
24   end;
```

Figure 4.17: Frequency calculation algorithm when using a seed pattern: FREQUENCY_CALCULATION_SEED.

addition, since $(A,1)$ does not belong to the time interval $[5,9]$ in R, it is also removed from the sequence S. In the next movement of the temporal window through the sequence, event $(B,2)$ leaves the window without belonging to any interval in R, so it is also removed from the sequence S. The final sequence after this removal is shown in figure 4.18 d). It is worth mentioning that both events would be kept in S if the window marking event removal scheme were used instead of the interval marking scheme.

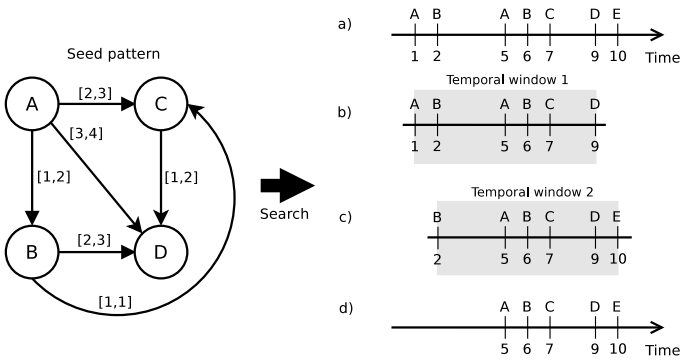


Figure 4.18: Example of frequency calculation when a seed pattern is provided.

4.3.3 Validation of `ASTP_SEED`

The definition of a seed pattern is expected to impact the results of the algorithm in several ways. Providing a seed pattern forces the mining process to focus on those temporal arrangements between the events present in the data collection that satisfy each and every one of the constraints, ignoring all others. This would modify the shape of the temporal distance distributions, possibly changing the results of the clustering procedure, both in terms of the number of frequent patterns found and in terms of the constraints they represent. These changes would result in a better performance of the mining process, as the time required by the frequency calculation procedure would be decreased due to the reduced number of patterns that need to be verified in each iteration.

Figure 4.19 represents the starting knowledge provided as a seed pattern to the mining process during validation with the SAHS database. This pattern represents simple knowledge available in the sleep apnea domain. In particular, it is well-known that some events occur before others, for example, the patient must stop breathing before it can resume normal breathing, and therefore the only interesting ocurrences of the *"end airflow limitation"* event are found after a *"begin airflow limitation"* event. This situation can be represented using the $L_{12} = [1, \omega]$ constraint, where $E_1 =$ *"begin airflow limitation"* and $E_2 =$ *"end airflow limitation"*. The same criteria can be applied to the events related to the thoracic and abdominal movement limitations, as well as the oxygen desaturation events. Moreover, it is expected that an airflow limitation beginning and end occur roughly at the same time as both abdominal and thoracic movement limitations. In addition, oxygen saturation in blood falls after the patient

has stopped breathing. The resulting pattern including this knowledge can be formally represented as $K =< D_K, \mathcal{L}_K >$, where $D = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8\}$ and the user-defined constraints are $\mathcal{L} = \{L_{12} = [1, \omega], L_{34} = [1, \omega], L_{56} = [1, \omega], L_{78} = [1, \omega], L_{13} = [-5, 5], L_{15} = [-5, 5], L_{17} = [0, \omega], L_{24} = [-5, 5], L_{26} = [-5, 5], L_{ij} = [-\omega, \omega]$ in other case$\}$.

Given that the seed pattern in this case includes all event types in the data collection, it is possible to remove events from the sequences using the interval marking scheme, because no new events can be added to the seed pattern and therefore it is not possible to eliminate events from any sequence that might be used by an occurrence of any pattern throughout the mining process. Even if the seed pattern did not include every event type in the collection, the user might be interested in using this removal scheme if he or she were sure that any event extending the seed pattern would be located after the events belonging to any of its occurrences.

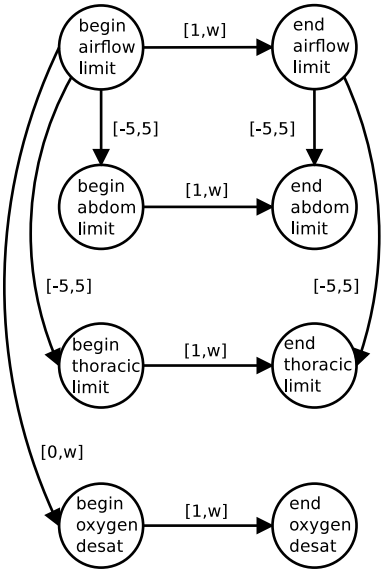
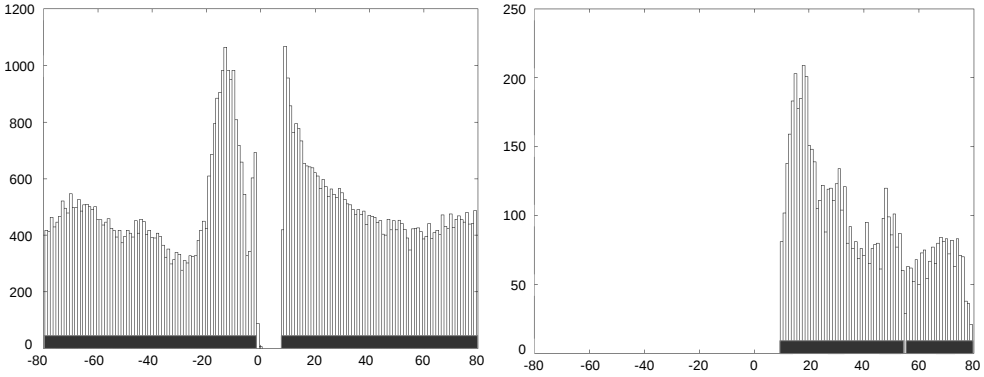


Figure 4.19: Seed pattern used in the validation experiments.

An example of the impact a seed pattern has in temporal distance distributions and clustering can be seen in figure 4.20. Both subfigures represent the temporal distance distribution between event types *"begin airflow limitation"* and *"end airflow limitation"*: positive temporal arrangements represent situations where the ending event is found after the beginning

event, and negative arrangements correspond to those situations where the beginning event is found after the ending event. In both cases the clustering procedure finds that there are two distinct intervals of temporal arrangements shaping up two different frequent patterns of size 2. However, while in ASTP_BASIC the patterns cover the whole temporal axis, in ASTP_SEED both patterns are located in the positive axis, representing a partition of the original pattern found when no seed knowledge is provided. In this case, the use of a seed pattern does not have any effect in the number of patterns found, but their meaning is changed, as it is now possible to differentiate situations that were otherwise considered the same. The patterns found are more specific, and also more relevant to the final user, since those patterns that are not compatible with the seed are discarded by the procedure.



(a) Without seed pattern (ASTP_BASIC). (b) With seed pattern (ASTP_SEED).

Figure 4.20: Impact of using a seed pattern on a temporal distance distribution and its clustering. These temporal distance distributions correspond to event types *"begin airflow limitation"* and *"end airflow limitation"*.

Table 4.3 shows, for each pattern size, the number of possible combinations, candidates generated and frequent patterns found with the algorithms ASTP_BASIC and ASTP_SEED. The 'Combinations' columns contain the total number of combinations of frequent patterns found in the previous iteration. The number of consistent networks from all possible combinations is shown in the columns labelled as 'Candidates'. The number of candidates with a frequency value higher than the threshold f_{min} is shown under 'Frequent'. In the case of patterns of size 2, the frequent patterns are obtained through the use of a clustering procedure, instead of the combination procedure used in later iterations, so the information for

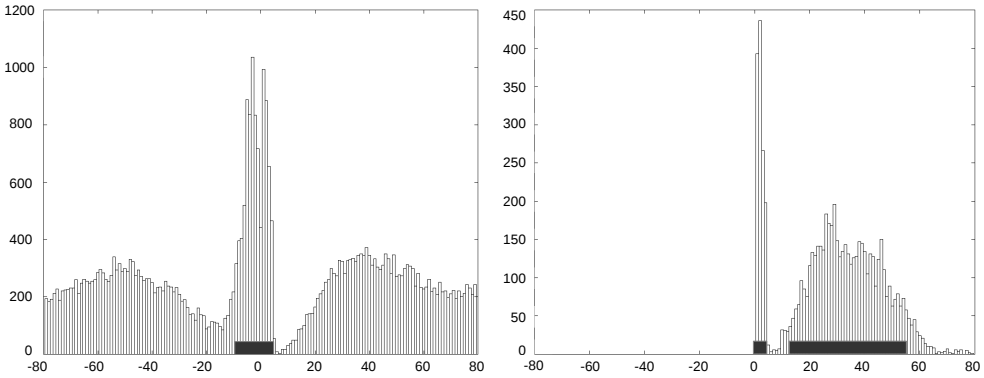
the 'Combinations', 'Candidates' and 'Discarded' columns in the second row is not available (NA).

Size	Combinations		Candidates		Discarded		Frequent	
	seed	**w/o seed**	**seed**	**w/o seed**	**seed**	**w/o seed**	**seed**	**w/o seed**
1	8	8	8	8	0	0	8	8
2	NA	NA	NA	NA	NA	NA	38	46
3	142	239	111	174	31	65	102	165
4	1019	5237	175	306	541	1885	174	303
5	11179	94184	196	298	1633	5136	192	297
6	1.2×10^5	7.9×10^5	131	162	2226	4218	128	160
7	5.5×10^5	1.7×10^6	49	47	1242	1439	48	46
8	1.1×10^6	1.1×10^6	7	6	276	198	7	6
Total	1.8×10^6	3.6×10^6	669	1001	5949	12491	689	1031

Table 4.3: Number of possible candidates, candidates generated, combinations discarded and frequent patterns found in the database, with and without seed pattern (window size ω =80 s., frequency threshold f_{min} =30 occurrences).

It is worth mentioning that the number of candidate and frequent patterns involved in the mining process is slightly reduced when a seed pattern is provided, although this reduction is not as significant as initially expected. By removing some temporal arrangements between events, the mining process can focus on the allowed arrangements, making it possible for the clustering procedure to discriminate between situations that were previously considered to be the same, thus creating more (and more specific) patterns. In this case, the number of patterns found will be similar, but their meaning will be different. Figure 4.21 shows a comparison of temporal distance distributions where the clustering procedure provides a different number of frequent patterns depending on whether a seed pattern is provided or not. Both temporal distance distributions represent the number of occurrences of the different temporal arrangements between the *"begin oxygen desaturation"* and *"end oxygen desaturation"* event types, where the positive axis represents those situations where the ending event is found after the beginning event. When no seed pattern is provided only one frequent pattern is found, while introducing knowledge into the procedure results in two different frequent patterns, one of them a subpattern of the one previously found and the other one representing an interval of temporal arrangements previously overshadowed by the former. In this case, removing uninteresting temporal arrangements from the procedure results in an increment in the number of

frequent patterns found, contradicting the intuitive idea that restricting the search scope would result in a reduced number of patterns involved in the mining process.

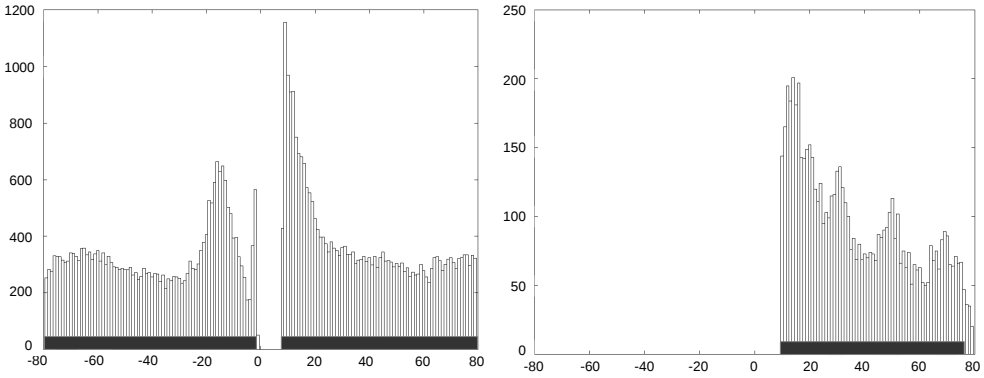


(a) Without seed pattern. (b) With seed pattern.

Figure 4.21: Second example on the impact of using a seed pattern on a temporal distance distribution and its clustering: number of patterns found may be higher when using a seed pattern. The temporal distance distributions correspond to event types *"begin oxygen desaturation"* and *"end oxygen desaturation"*.

Figure 4.22 shows another comparison between temporal distance distributions obtained with and without seed patterns where this situation can be observed. The temporal distance distributions represent the number of different temporal arrangements between the event types *"begin thoracic limitation"* and *"end thoracic limitation"*, where the positive axis represents those situations where the ending event is found after the beginning event. In this example, the use of a seed pattern in the mining process removes the negative axis from the temporal distance distribution: one of the size two frequent patterns produced by the clustering procedure when using `ASTP_BASIC` did not satisfy the constraints specified in the seed pattern. If this situation is repeated in each temporal distance distribution, the number of candidates resulting from the candidate generation procedure would be reduced, but previous examples have already shown that this situation is not necessarily true for every temporal distance distribution.

Figure 4.23 shows another example of application of the clustering procedure to the temporal distance distribution of a temporal association between two event types, the start of an abdominal movement limitation and the start of oxygen desaturation events, in two different situations. Case (a) represents the temporal distance distribution of the temporal arrangements within a temporal window of 80 seconds, together with the clustering results for the proce-

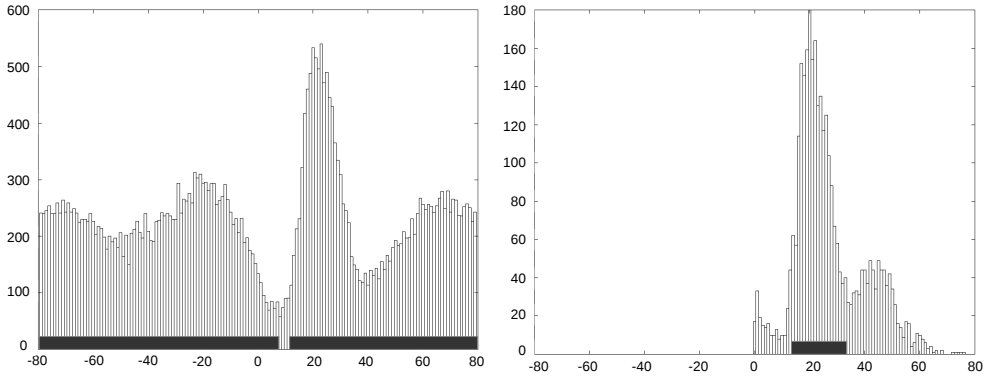


(a) Without seed pattern. (b) With seed pattern.

Figure 4.22: Third example on the impact of using a seed pattern on a temporal distance distribution and its clustering: patterns found are more specific, and more relevant to the final user. The temporal distance distributions correspond to event types *"begin thoracic limitation"* and *"end thoracic limitation"*.

dure when no seed pattern is used. In this case, the clustering provides two intervals as a result, $[-80,7]$ and $[12,80]$, each one representing a temporal pattern of size 2. In particular, the interval $[-80,7]$ allows the oxygen saturation in blood to decrease before the abdominal movement limitation, associating the abdominal movement limitation to the oxygen desaturation decrease of a previous apnea pattern, resulting in a non interesting pattern. Case (b) represents the result when a seed pattern specifies that the abdominal movement limitation event must occur before the starting oxygen desaturation event. One of the patterns in case (a) is discarded by the definition of the seed pattern, whereas the remaining pattern is shortened into a more specific pattern in case (b) producing a frequent pattern of size 2 covering the interval $[14,32]$. As can be seen, the definition of seed patterns alters the shape of temporal distance distributions, and thus modifies the result of the clustering.

Figure 4.24 presents the time required by `ASTP_BASIC` and several implementations of `ASTP_SEED` in the SAHS database. 'SEED+IM' and 'SEED+WM' represent versions of the `ASTP_SEED` algorithm where the seed pattern shown in figure 4.19 was provided, along with the interval and window marking event removal schemes, respectively. In both cases, event removal was performed only once, during the frequency calculation of the seed pattern in the initialisation step. Both implementations improve on the performance of the basic algorithm, achieving the best performance when the interval marking system is used. These results show



(a) Without seed pattern. (b) Using a seed pattern.

Figure 4.23: Another example of the influence of the seed pattern in the clustering process.

how providing knowledge to the procedure effectively reduces the search scope of the mining process and improves its performance.

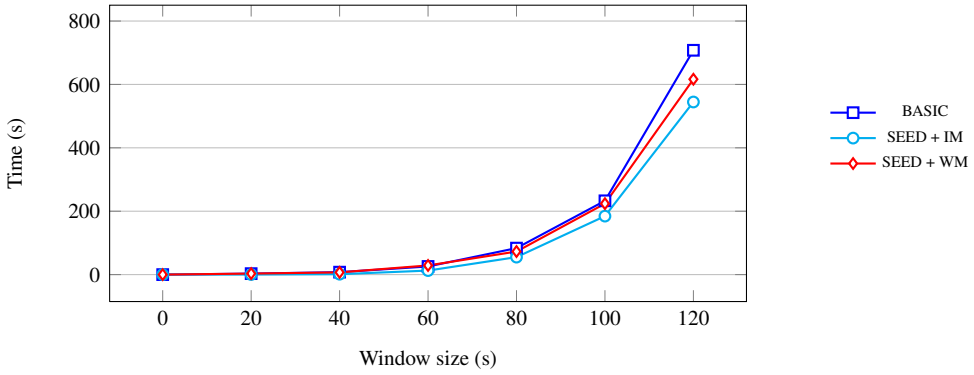


Figure 4.24: Comparison of time required by the `ASTP_BASIC` algorithm and two different implementations of the `ASTP_SEED` algorithm in the SAHS database.

Two of the frequent patterns of size 8 found by the algorithms in the validation experiments are shown in figure 4.25. Some of the constraints have been removed for better visualisation. Figure 4.25(a) shows one of the six frequent patterns found by `ASTP_BASIC`, when no seed pattern is provided. This pattern coarsely corresponds to the representation of a central apnea episode. Figure 4.25(b) shows the closest frequent pattern to the previous one

among the seven frequent patterns found by ASTP_SEED, when seed knowledge is provided. Even though the resulting pattern in this case does not add any new event type to the seed pattern, the constraints obtained when the seed pattern is provided are more precise, providing a more accurate representation of the knowledge. In addition, the constraint between the event types *"begin oxygen desaturation"* and *"end oxygen desaturation"* no longer includes negative values, as the definition of the seed pattern precluded them from being accepted, so the constraint now reflects only situations where the ending event occurs after the beginning event.

4.4 ASTPminer

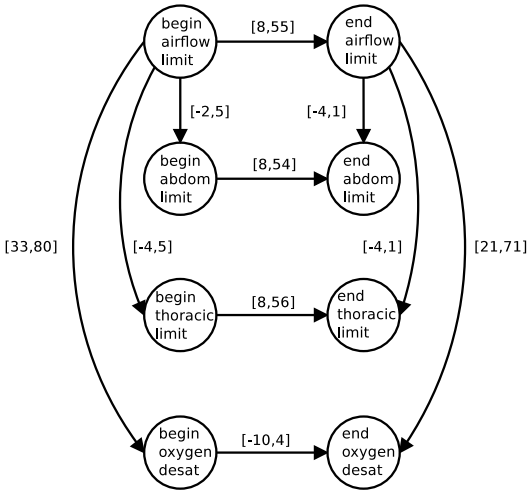
In this section we will present the full version of the ASTPminer algorithm introduced in [Álvarez et al., 2013] for discovering frequent temporal patterns from a set of time-stamped event sequences. ASTPminer contemplates the inclusion of both events and episodes as temporal entities to be mined as part of a time-stamped event sequence. An episode is represented in an event sequence as a pair of events but is considered to be a single entity in the mining process, which precludes the generation of spurious associations and saves computational time.

Figure 4.26 presents the structure of ASTPminer. The initial parameters for this algorithm are given by the tuple $(\mathcal{S},\mathcal{F},\mathrm{K},\omega,\mathrm{f}_{min})$:

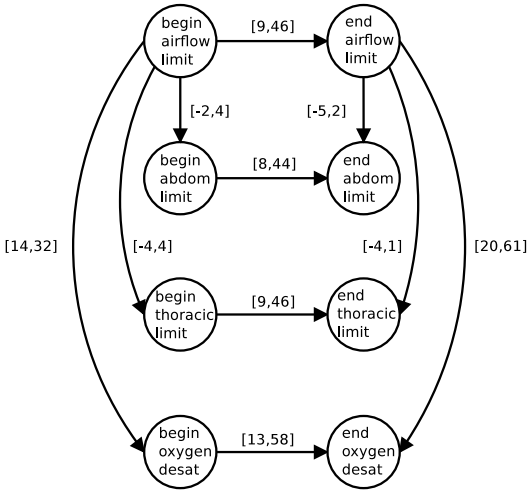
- a collection of recordings $\mathcal{S}$,
- a set of types of facts (events and episodes) $\mathcal{F}=\mathcal{E}\cup\mathcal{G}$ in $\mathcal{S}$,
- an initial seed pattern K,
- a window size ω,
- a frequency threshold f_{min}.

For simplicity in the description of the algorithms, we will assume that only one seed pattern is specified by the user.

The objective of ASTPminer is to iteratively search in $\mathcal{S}$ for those frequent temporal patterns P extending K, that is $\mathrm{K}\preceq P$, where events are, at most, ω time units apart. These temporal patterns P will represent common temporal arrangements between events and/or episodes



(a) Without seed pattern.



(b) With seed pattern.

Figure 4.25: Comparison of two frequent patterns of size 8: (a) obtained by `ASTP_BASIC` without seed pattern, (b) obtained by `ASTP_SEED` when a seed pattern was provided.

found in the collection. The seed pattern K is the minimal network consistent with the knowledge provided by the user, obtained using the ALL-PAIRS-SHORTEST-PATHS() algorithm before the seed pattern is provided to ASTPminer. The collection of recordings $\mathcal{S}$ can be pruned during execution by removing those subsequences where no frequent patterns can be found in further iterations. The other four parameters remain constant throughout the procedure.

```
    procedure ASTPminer(S, F, K, ω, f_min)
1   begin
2     A^1 ← {E_j | E_j ∈ E ∧ f(E_j) ≥ f_min}
3     P^1 ← A^1
4     P^2 ← INITIALISATION(S, A^1, P^1, F, K, ω, f_min)
5     A^2 ← {D_j | P^2_j=<D_j, L_j>∈P^2}
6     i ← 3
7     while (P^{i-1} ≠ ∅ ∨ C_p^{i-1} ≠ ∅) do begin
8       C^i ← CANDIDATE_GENERATION(A^{i-1}, P^{i-1} ∪ C_p^{i-1})
9       if C_c^i ≠ ∅ then
          P^i ← FREQUENCY_CALCULATION(C_c^i, S, F, ω, f_min)
10      A^i ← {D_j | P^i_j=<D_j, L_j>∈P^i}
11      i ← i+1
12    end;
13  end;
```

Figure 4.26: Main algorithm: ASTPminer.

In each iteration i, a set of frequent temporal patterns of size i is found. The process ends when no new frequent patterns are found in a given iteration. The output of the mining process after this final iteration consists of a set of frequent temporal patterns that are consistent with the knowledge provided by the user as seed pattern. As both algorithms previously described, ASTPminer uses three lists that are updated in every iteration i: the list A^i holds frequent temporal associations comprised of i events; the list C^i stores candidate patterns with i events, and the list $P^i \subseteq C^i$ contains those candidate patterns with i events that are confirmed to be frequent.

In the case of ASTPminer, each list C^i is subdivided into two additional lists, $C^i = C^i_c \cup C^i_p$. List C^i_c contains those complete candidate patterns whose frequency will be checked during the frequency calculation step, whereas list C^i_p contains partial patterns that are necessary for

candidate generation in subsequent iterations, but not to be searched for in the present iteration: those patterns that contain either the beginning or the ending event type of an episode, but not both, belong to C_p^i. Section 4.4.3 shows how both lists are obtained, and further explains the criteria for introducing a particular pattern in one of these lists.

`ASTPminer` begins with a search for frequent event types in the collection $\mathcal{S}$, which are stored in list A^1, representing temporal associations of size 1. These frequent event types are also stored in list P^1 since they also represent frequent temporal patterns of size 1. Then, the algorithm uses the `INITIALISATION()` procedure (Figure 4.26, line 4) to obtain frequent temporal patterns of size 2. This procedure finds the set of frequent patterns of size 2 that are consistent with the information supplied by the user in the seed pattern, discarding infrequent event types and also filtering those fragments of the initial collection $\mathcal{S}$ where no occurrence of the seed pattern can be found. At the end of the `INITIALISATION()` procedure all the temporal associations of size 2 are stored in the list A^2; these temporal associations correspond to the different pairs of event types that appear frequently together, and are extracted from the set of frequent temporal patterns of size 2.

After the initialisation step, the search continues for more complex patterns in two iterative steps (Figure 4.26, line 7), involving candidate generation and frequency calculation. The algorithm finishes when one iteration results in both no new frequent patterns found and an empty C_p^i list, meaning that no additional patterns may be built. Notice that it is possible that the result of the candidate generation step is an empty C_c^i list in some iteration while producing a non-empty C_p^i list. In this case, it is not necessary to perform the frequency calculation step, as there are no complete patterns to search for, so execution proceeds to the next iteration.

4.4.1 Initialisation step in `ASTPminer`

Figure 4.27 shows the `INITIALISATION()` procedure for obtaining all frequent patterns of size 2, which are stored in list P^2. This list contains those pairs of events whose temporal arrangements are consistent with the event types and constraints in the seed pattern K, and whose frequency is higher than the threshold.

The procedure begins by obtaining the frequency of the seed pattern K$= < D_K, \mathcal{L}_K >$ (Figure 4.27, line 2), using the `FREQUENCY_CALCULATION()` algorithm described in section 4.4.2, that checks all temporal windows of width ω throughout the collection $\mathcal{S}$ for occurrences of the seed pattern. If the seed pattern turns out to be not frequent after this counting,

```
    procedure INITIALISATION(S, A^1, P^1, F, K, ω, f_min)
1   begin
2     K^2 ← FREQUENCY_CALCULATION({K}, S, F, ω, f_min)
3     if (K^2 = ∅) then stop
4     C^2 ← CANDIDATE_GENERATION(A^1, P^1)
5     P^2 ← K^2 ∪ FREQUENCY_CALCULATION(C^2, S, F, ω, f_min)
6     return P^2
7   end;
```

Figure 4.27: `ASTPminer`: Initialisation algorithm.

the algorithm stops (line 3). The result of this step is the set K^2 of frequent patterns of size 2 that involve pairs of event types from the seed pattern, and are consistent with it.

The next step during initialisation consists of building the set C^2 of those candidate patterns of size 2 including other frequent event types in P^1 that have not been extracted from K. This is carried out by the procedure `CANDIDATE_GENERATION()` described in section 4.4.3. `FREQUENCY_CALCULATION()` obtains from these candidate patterns an additional set of frequent patterns of size 2, also consistent with the seed pattern. The list P^2 is finally updated by gathering together both kinds of frequent patterns of size 2: those including only event types from the seed pattern, and those with some additional event types (line 4), to be added as possible extensions of the seed pattern.

4.4.2 Frequent pattern calculation in ASTPminer

Figure 4.28 shows the procedure `FREQUENCY_CALCULATION()` used to determine the frequency of the candidate patterns in the data collection.

This algorithm searches for and counts the occurrences in $\mathcal{S}$ of the candidate patterns Q^i. Events are sequentially read from each recording $S \in \mathcal{S}$ by means of a window W of width ω that moves along each recording of the collection, completely processing every recording before processing the next one.

With each new event $e_j = (o_k, v_j, t_j)$ from S, the temporal window is updated, so $W = \{e_{w_1}, ..., e_{w_n} = e_j\} \subseteq S$. All events e_k in W now satisfying $t_j - t_k > \omega$ lie outside the bounds of the current temporal window, and they are therefore removed from it (Figure 4.28, line 10). In addition, those events $e_h = (o_e, v, t_h) \in W$ that represent the end of an episode $g = (o, v, t_k, t_h)$

```
   procedure FREQUENCY_CALCULATION(Q^i, S, F, ω, f_min)
 1 begin
 2   for S ∈ S do begin
 3     R ← ∅
 4     W ← ∅
 5     for e_j ∈ S do begin
 6       if Q^i = {K} then begin
 7         S ← S − {e_k | t_j − t_k > ω ∧ ∀[t_a,t_b] ∈ R, t_k ∉ [t_a,t_b]}
 8         R ← R − {[t_a,t_b] ∈ R | t_j − t_b > ω}
 9       end
10       W ← W − ({e_k | t_j − t_k > ω} ∪
           {e_h = (o_e,v,t_h) ∈ W | g = (o,v,t_k,t_h) ∧ t_j − t_k > ω})
11       W ← W ∪ {e_j}
12       for Q^i_l =< D^i_l, L^i_l > ∈ Q^i do
13         for X = {e_{k_1},...,e_{k_i} = e_j} ⊆ W ∧ {E_{k_1},...,E_{k_i} = E_j} = D^i_l do
14           if (∀e_{k_p}, e_{k_q} ∈ X, t_{k_q} − t_{k_p} ∈ L_{k_p k_q} ∈ L^i_l) ∧
             (∀G = (o,v,T_{k_p},T_{k_q}) ⪯ Q^i_l ⇒
             ∃g = (o,v,t_{k_p},t_{k_q}) | (o_b,v,t_{k_p}),(o_e,v,t_{k_q}) ∈ X) then begin
15             UPDATE_DISTRIBUTION(N = {n_{k_p k_q}}_{p,q}, X)
16             f(Q^i_l) ← f(Q^i_l)
17             if Q^i = {K} then R ← R ∪ {[t_{k_1}, t_{k_i} = t_j]}
18           end;
19     end;
20   end;
21   if i = 2 ∨ Q^i = {K} then
       P^i ← CLUSTERING(N = {n_{k_p k_q}}_{p,q}, f_min)
22   else P^i ← {Q^i_l | Q^i_l ∈ Q^i ∧ f(Q^i_l) ≥ f_min}
23   return(P^i)
24 end;
```

Figure 4.28: Frequency calculation in ASTPminer: FREQUENCY_CALCULATION.

with its beginning $e_k = (o_b, v, t_k)$ lying outside the window are also removed, since both ends of an episode must occur within the temporal window for an occurrence of the episode to be counted.

Once the temporal window W is updated with the new event e_j, those candidate patterns that contain its corresponding event type E_j are checked for occurrences in W. For each of these candidates Q_l^i, every different subsequence X of the temporal window W ending with e_j and containing the event occurrences $e_{k_1}, \ldots, e_{k_{i-1}}$ of the rest of event types $E_{k_1}, \ldots, E_{k_{i-1}}$ in the candidate is checked for consistency with the temporal constraints of this candidate (Figure 4.28, lines 13-14). An additional restriction for X to be counted as a possible occurrence of a candidate pattern is that if this candidate involves the event types $E_{k_p} = (o_b, v, T_{k_p})$ and $E_{k_q} = (o_e, v, T_{k_q})$ representing the beginning and the end of an episode $G = (o, v, T_{k_p}, T_{k_q})$, then X must include a pair of events $e_{k_p} = (o_b, v, t_{k_p})$ and $e_{k_q} = (o_e, v, t_{k_q})$ corresponding to the same episode $g = (o, v, t_{k_p}, t_{k_q})$. If a subsequence X satisfies all constraints in a candidate then it is a possible occurrence of this candidate in W, so the temporal distance distributions are updated (Figure 4.28, line 15). This updating typically further restricts the temporal constraints of each candidate pattern, as some temporal arrangements that appear in patterns of a given size disappear in extensions with more event types. Clustering is no longer necessary if temporal distance distributions remain convex. Then, the frequency of the candidate is increased (Figure 4.28, line 16).

`FREQUENCY_CALCULATION()` presents one difference when the candidates consist of a pair of event types ($i = 2$), or the candidates correspond to the seed patterns. In the case where $i = 2$, when an occurrence is found, in addition to increasing the frequency of the candidate, the temporal arrangement between each pair of events e_{k_p} and e_{k_q} in the occurrence is included in the temporal distance distribution between their event types $\{n_{k_p k_q}\}$, one distribution for each pair, represented as a histogram. The frequent patterns returned by the procedure are obtained by clustering all temporal distance distributions $\mathcal{N} = \{n_{k_p k_q}\}_{p,q}$ that were previously built (line 21). From the clustering of every temporal distance distribution $\{n_{k_p k_q}\}$ a set of non-overlapping intervals $I_1, \ldots, I_m$ is obtained where the occurrences of the candidates are concentrated. These intervals represent different constraints between each pair of event types E_{k_p} and E_{k_q} and results in m different frequent patterns of size 2.

When the set of candidates Q^i is the set containing the seed pattern K, if an occurrence is found the procedure updates every temporal distance distribution between each pair of event types present in the seed pattern. Each temporal distance distribution is subjected to the clustering procedure, which produces a set of frequent patterns of size 2 for each temporal

distance distribution (line 21). Therefore, the result of the procedure in this case is also a set of frequent patterns of size 2, but each pattern represents temporal arrangements between events of the seed pattern. Additionally, the procedure also removes any subsequence of the data collection where no occurrence of the seed pattern can be found. If no occurrence of the seed pattern can be found in a subsequence, then no occurrence of any extension can be found in later iterations, so the subsequence is not interesting for the mining process. To this end, whenever the procedure detects an occurrence of a seed pattern, it inserts into the list R the instants $[t_{k_1}, t_{k_i} = t_j]$ limiting the temporal intervals where an occurrence of K can be found (line 17). Those events that leave the temporal window and do not belong to any interval in R can be removed from the collection (line 7), as they cannot belong to an occurrence of a seed pattern extension, which reduces the search scope in later iterations. Those intervals of R that lie outside the temporal window can be removed from R, as they are no longer useful for removing events (line 8).

In all the iterations, once all events of the collection are read, the algorithm filters out those candidates that do not satisfy the frequency threshold f_{min} and removes them from the mining procedure (line 22), returning only those candidates that are considered frequent.

Figure 4.29 shows an example of the `FREQUENCY_CALCULATION()` procedure. Let the window width be $\omega = 8$, the set of event types be $\mathcal{E} = \{E_1 = A, E_2 = B, E_3 = C, E_4 = D, E_5 = E\}$, and the set of episode types be $\mathcal{G} = \{G_1 = (o_1, v_1, T_b = T_A, T_e = T_B), G_2 = (o_2, v_2, T_b = T_C, T_e = T_D)\}$. Let $K =< D, \mathcal{L} >$ be a seed pattern, where $D = \{A, B, C, D\}$ and $\mathcal{L} = \{L_{12} = [1,2], L_{13} = [2,3], L_{14} = [3,4], L_{23} = [1,1], L_{24} = [2,3], L_{34} = [1,2]\}$. The sequence where the seed pattern will be searched for is $S = \{(A,1), (B,2), (A,5), (B,6), (C,7), (D,9), (E,10)\}$, where there are two episodes $g_1 = (o_1, v_1, 1, 2)$ and $g_2 = (o_1, v_1, 5, 6)$ of the episode type G_1, represented in the sequence by the events $(A,1)$ and $(B,2)$ and events $(A,5)$ and $(B,6)$ respectively, while the episode $g_3 = (o_2, v_2, 7, 9)$ corresponds to the episode type G_2 and is represented by the events $(C,7)$ and $(D,9)$ in the sequence.

The procedure sequentially introduces the events from the sequence into the temporal window, finding no occurrence of the seed pattern until the event $(D,9)$ is read. This situation is labelled as "Temporal window 1". After this event is read, the window contains at least one event of each event type represented by the seed pattern, and it is necessary to check all possible subsequences of the temporal window searching for occurrences of the seed pattern. According to line 13 of the `FREQUENCY_CALCULATION()` procedure, only those subsequences where event $(D,9)$ is present need to be checked. In addition, as shown in the second part of line 14, both events of an episode must be present in the sequence at the same

time. According to this condition, the sequence $\{(A,1), (B,6), (C,7), (D,9)\}$ is not valid, as the events $(A,1)$ and $(B,6)$ do not belong to the same episode: the first one corresponds to episode g_1, whereas the second one corresponds to episode g_2. The only subsequences satisfying both conditions are $X_1 = \{(A,1), (B,2), (C,7), (D,9)\}$ and $X_2 = \{(A,5), (B,6), (C,7), (D,9)\}$. Sequence X_1 does not satisfy the constraints L_{13}, L_{14}, L_{23} and L_{24} of the seed pattern and therefore it is not an occurrence. On the other hand, the events in sequence X_2 satisfy all the constraints, and therefore the frequency of the seed pattern and the temporal distance distributions between the event types of the seed pattern need to be updated (lines 15-16). The interval $[5,9]$ indicating where the events of the occurrence can be found is introduced into the list R (line 17). The procedure then reads the event $(E,10)$ from the event sequence. Reading this event forces the temporal window to move to the position labelled as "Temporal window 2". In this position, the event $(A,1)$ is no longer present in the temporal window. At the same time, it is necessary to check if the event $(A,1)$ belongs to any interval present in R (line 7). In this case, only the interval $[5,9]$ is present in R, and the event $(A,1)$ does not belong to it, resulting in the removal of the event from the event sequence. In addition, event $(B,2)$ is also removed from the window, because it belongs to the same episode as the event $(A,1)$ (line 10). Then, the event $(E,10)$ is introduced into the temporal window and, as the seed pattern does not include the event type E it is not necessary to verify if there are any occurrences of the seed pattern in the temporal window. As there are no events left to read in the sequence, the procedure ends. Finally, events $(B,2)$ and $(E,10)$ can be removed from the sequence, as they do not belong to any interval in R.

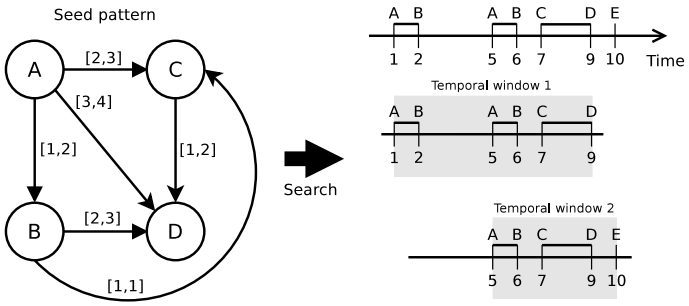


Figure 4.29: Frequency calculation example in `ASTPminer`.

```
     procedure CANDIDATE_GENERATION(A^{i-1}, P^{i-1})
1    begin
2      C^i ← ∅
3      h ← 1
4      for A_j^{i-1}, A_k^{i-1} ∈A^{i-1} ∧ E_{j1} = E_{k1}, ..., E_{j_{i-2}} = E_{k_{i-2}}, E_{j_{i-1}} < E_{k_{i-1}} do begin
5        A_h^i ← A_j^{i-1} ∪ {E_{k_{i-1}}}
6        h ← h + 1
7        if all A_j^{i-1} ⊂A_h^i satisfies A_j^{i-1} ∈A^{i-1} then
8          for all combination of P_{h_k}^{i-1}, P_{h_k}^{i-1} ∈P^{i-1} ∧ D_{h_k}^{i-1} ⊂A_h^i do
9            if Q_l^i =ALL-PAIRS-SHORTEST-PATHS(⋈_h P_{h_k}^{i-1})
             is consistent then
10             if ∀E_j in Q_l^i, E_j in G_k ∈ G ⇒G_k ⪯ Q_l^i then
11               C_c^i ← C_c^i ∪ {Q_l^i}
12             else
13               C_p^i ← C_p^i ∪ {Q_l^i}
14     end;
15     return(C^i = C_c^i ∪ C_p^i)
16   end;
```

Figure 4.30: Candidate generation algorithm in ASTPminer: CANDIDATE_GENERATION.

4.4.3 Candidate generation in ASTPminer

The CANDIDATE_GENERATION() procedure, shown in figure 4.30, receives as arguments the frequent temporal associations and frequent patterns found in the previous iteration, and produces the candidate patterns for the iteration in course, whose frequency must be checked by the FREQUENCY_CALCULATION() procedure.

The first step in the generation of a candidate pattern of size i (line 4 in figure 4.30) consists of building a temporal association A^i between the event types in the pattern. Following an Apriori strategy, a temporal association of size i is built from the union of two frequent temporal associations of size $i-1$ that have all event types in common except the last one. Then, the procedure checks if all subsets of size $i-1$ of event types of the new temporal association, which were already calculated in the previous iteration of the main algorithm, are frequent.

The second step in the procedure consists of the combination (line 7) of i frequent patterns $P_{h_k}^{i-1}$ of size $i-1$, representing subpatterns of a new candidate of size i, one for each frequent

temporal association A_k^{i-1} verified in the previous step. After the combination step is performed, the consistency of the candidate is checked by `ALL-PAIRS-SHORTEST-PATHS()` [Dechter et al., 1991] (line 9). This algorithm applies constraint propagation through the graph of the candidate pattern, obtaining minimal temporal constraints representing consistent temporal arrangements between the event types of the new pattern. If the pattern is consistent, then it is added to one of two different lists (line 11): the first one, C_c^i, contains those complete candidates that will be searched for during the frequency calculation step; the second one, denoted as C_p^i, contains those partial candidate patterns that should be extended before being searched for in further iterations. In order to select the appropriate list we check if every event type involved in the new pattern, and also corresponding to the beginning or the end of an episode type $G_k \in \mathcal{G}$, entails the presence of the other ending event type of the same episode type in the new pattern. In the affirmative case, the new pattern is a complete candidate, and is added to C_c^i; otherwise, some episode type involved in the new pattern is not yet completed: one of its ending event types is not included, so the pattern is added to C_p^i. By taking apart the partial candidate patterns we avoid performing the frequency calculation step, the most time consuming, for example in odd iterations for those patterns which only involve episode types.

As previously mentioned, candidate generation during the initialisation stage is carried out in a different manner. On one hand, the combination step is not performed, as candidates consist of temporal associations of pairs of event types, and there are no known distinct temporal arrangements between event types at this point: constraints in patterns of size 2 are obtained using the clustering procedure in the frequency calculation step. On the other hand, the only candidates generated are those pairs of event types not already extracted from the seed pattern K, but nevertheless consistent with it.

The `CANDIDATE_GENERATION()` procedure generates candidate patterns that are extensions of previously found frequent patterns. Considering that frequent patterns of size 2 extracted in the initialisation step are necessarily consistent with the seed pattern K, the procedure ensures that candidate patterns generated in later iterations will also be consistent with the information provided in K.

4.4.4 Correctness and completeness of ASTPminer

In this section we examine both the correctness and completeness of the `ASTPminer` algorithm.

Lemma 1: (Correctness) Temporal patterns obtained by the ASTPminer algorithm are frequent.

Rationale: The FREQUENCY_CALCULATION() procedure searches for occurrences of all candidate patterns in all possible temporal windows of every recording in the data collection. During the calculation of P^i, i.e., the set of frequent patterns of size i, for every event e_j introduced into the temporal window, all subsequences X of size i of the temporal window that include the event e_j are checked against all patterns containing the event type E_j of the event e_j (lines 12-13). For each candidate pattern Q_l^i and each subsequence X in the window, the frequency of Q_l^i is updated if, and only if, the temporal arrangements between the events satisfy every constraint in Q_l^i (line 14). This procedure ensures that every occurrence of every pattern present in the data collection is accounted for, whereas no combination of events in the collection is considered to be an occurrence of any pattern unless all constraints are satisfied. The algorithm discards any candidate pattern Q_l^i having a frequency value $f(Q_l^i)$ less than the user-defined threshold f_{min} (line 22), which ensures that resulting patterns are frequent.

Lemma 2 (Completeness) Candidate generation in ASTPminer is exhaustive.

Rationale: The Apriori strategy summarised in lines 4-7 ensures that all different temporal associations of size i are obtained from the set A^{i-1} of frequent temporal associations of size $i-1$. Additionally, the procedure builds the set C^i of all the candidate patterns of size i from the set P^{i-1} of frequent patterns of size $i-1$, which are different temporal arrangements of the aforementioned temporal associations (line 8).

P^1 is constructed by removing non-frequent event types from $\mathcal{E}$. Given P^{i-1}, i.e., the set of frequent patterns of size $i-1$, the CANDIDATE_GENERATION() procedure produces all possible combinations $\bowtie_h P_{h_k}^{i-1}$ of i patterns in P^{i-1}, where the set of event types of every pattern $P_{h_k}^{i-1}$ represents a different temporal association in A^{i-1}. Only inconsistent patterns are discarded during the CANDIDATE_GENERATION() procedure (line 9). The set of temporal patterns is closed under the combination operation, so CANDIDATE_GENERATION() produces temporal patterns from temporal patterns. The consistency of the pattern is checked using the ALL-PAIR-SHORTEST-PATHS() algorithm, which results in a minimal network.

The reasoning above does not take into account the use of clustering to obtain P^2. Clustering procedures are summarisation procedures that aim to extract from the dataset a reduced, but significant, number of groups. The criterion used to select these groups is the similarity of a set of temporal arrangements that, when gathered together, exceed the frequency threshold f_{min} established by the user. Therefore, ASTPminer satisfies only a partial completeness.

The CANDIDATE_GENERATION() procedure is exhaustive but ASTPminer is not exhaustive because of the clustering procedure.

4.4.5 Complexity analysis of ASTPminer

We assume that $|\mathcal{E}|$ contains both limiting event types of each episode type in $\mathcal{G}$. The computational complexity of the ASTPminer algorithm in every iteration i can be evaluated as follows:

- In the *temporal association generation* step, for a temporal association A_j^i to be frequent, all of the i temporal associations $\mathrm{A}_k^{i-1} \subset \mathrm{A}_j^i$ must be frequent. The maximum number of frequent temporal associations of size $i-1$ is $|\mathrm{A}^{i-1}| = \binom{|\mathcal{E}|}{i-1}$, therefore a binary search among these frequent associations has a complexity of $O(log|\mathrm{A}^{i-1}|)$. In addition, each temporal association of size $i-1$ can be added $|\mathcal{E}|-i$ different event types to create a temporal association of size i. Taking into account the combination step, which compares the event types of two temporal associations of size $i-1$ to ensure that they share the first $i-2$ event types, the complexity of generating all the candidate temporal associations is $O(|\mathrm{A}^{i-1}||\mathcal{E}|i^2 log|\mathrm{A}^{i-1}|)$.

- In the *candidate generation* step, each of the $i(i-1)/2$ constraints of a temporal pattern $\mathrm{P}_{j_k}^i$ is obtained by combining those patterns $\mathrm{P}_{h_k}^{i-1}$ where $D_h^{i-1} \subset \mathrm{A}_j^i$. The combination has a complexity of $O(i^2)$, with a maximum number of $\prod_h |\mathrm{P}_h^{i-1}|$ possible combinations for each temporal association of size i. In terms of the initial parameters, the maximum number of temporal patterns of size two between any two event types corresponds to the situation where every temporal arrangement presents a frequency value greater than the frequency threshold f_{min}, but their frequency values are so different that the clustering procedure produces one pattern for each temporal arrangement. In this situation, the clustering procedure produces 2ω patterns for every temporal association of size two. Under this assumption, to build a pattern of size i it is sufficient to establish the $i-1$ constraints between one event type and the rest of the event types, as the remaining constraints can be completely specified during the ALL-PAIRS-SHORTEST-PATHS() step. Additionally, the number of all possible combinations of frequent patterns to explore to build all candidate patterns for any temporal association of size i is $\omega^{i(i-1)}$. Every combination is subject to a consistency

checking process by the `ALL-PAIRS-SHORTEST-PATHS()` algorithm, which has a complexity of $O(i^3)$, which results in a complexity of $O(|\mathrm{A}^i|i^3\omega^{i(i-1)})$.

In the worst case, this result is worse than the result in [Dousson and Duong, 1999], as our approach is able to find more than just one temporal arrangement among a given set of event types. However, when we restrict our search to provide only one temporal pattern for each set of event types, our approach is more efficient.

– In the *frequency calculation* step, the worst case scenario corresponds to a situation where every time unit may present an occurrence of all $|\mathcal{E}|$ event types with a total of $n = |\mathcal{E}|d_{\mathcal{S}}$ events in $\mathcal{S}$ (where n represents the length of $\mathcal{S}$ and $d_{\mathcal{S}}$ represents the sum of the durations of the sequences in the collection). Every time the window is updated, $|\mathcal{E}|$ events enter the window and $|\mathcal{E}|$ leave. Under this assumption, for every pattern of size i, up to $i\omega^{i-1}$ new occurrences may be found in every window update for every pattern. Checking whether an occurrence fulfils all of the constraints in a pattern has a complexity of $O(i^2)$. Considering that the maximum number of candidate patterns is $|\mathrm{A}^i|\omega^{i-1}$, the overall complexity for this step is $O(n\omega^{2i-2}i^3|\mathrm{A}^i|)$.

Given the iterative nature of the mining process, and considering that the maximum pattern size is the number of different event types $|\mathcal{E}|$, the overall complexity of the algorithm is dominated by the candidate generation step, which results in a complexity of $O(|\mathcal{E}||\mathrm{A}^i|i^3\omega^{i(i-1)})$. However, in practical terms, candidate generation is no the most time consuming step in our experiments. It is worth mentioning that this complexity corresponds to a very improbable situation and, as the experimental results show, in practical situations this complexity is dramatically decreased.

The use of seed patterns is expected to have an impact in reducing the number of patterns $|\mathrm{P}^i|$ generated in each iteration, and in the overall number of events n of the collection $\mathcal{S}$. Those segments of the collection where no occurrences are found are discarded for further iterations, which reduces the value of n. These changes contribute to improve the total efficiency of the algorithm. As seen in section 4.3, it is possible that $|\mathrm{P}^2|$ is greater when seed patterns are used, as the shape of the temporal distance distributions may vary and their clustering may result in more numerous yet more specific patterns. However, if that were the case, the number of possible combinations in later iterations would be expected to be lower, as more specific patterns would have fewer possible consistent combinations.

4.4.6 ASTPminer validation with the SAHS database

Recalling the set of annotated event types in the SAHS database: $\mathcal{E} = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8\}$, where E_1 : "*start airflow limitation*", E_2 : "*end airflow limitation*", E_3 : "*start abdominal movement limitation*", E_4 : "*end abdominal movement limitation*", E_5 : "*start thoracic movement limitation*", E_6 : "*end thoracic movement limitation*", E_7 : "*start oxygen desaturation*" and E_8 : "*end oxygen desaturation*", in this domain, the most basic knowledge that we can introduce in the mining algorithms lies in the notion of episode: airflow limitations as well as thoracic and abdominal movement limitations must start before they can end, and oxygen saturation must fall before it can rise. The introduction of episodes in the mining process allows us to remove from the search procedure those occurrences that are not reasonable in the domain yet can be found in the records, for example, due to successive occurrences of some patterns in the same temporal window. For instance, a temporal association could be created by combining the beginning event of an airflow limitation with the beginning event of a previous decrease in oxygen saturation. The pulmonologist knows that those types of associations are not interesting and should be avoided.

Therefore, in the SAHS database, common knowledge allows us to define some episode types between event types that necessarily appear together, and in a particular order, in the database. The episode types involved are:

G_1 : "*airflow limitation*"

G_2 : "*abdominal movement limitation*"

G_3 : "*thoracic movement limitation*"

G_4 : "*oxygen desaturation*"

We have conducted our validation experiments using the seed pattern shown in figure 4.31. This pattern coarsely approximates a representation of a central apnea, and consists of the eight event types present in the records, with four of the constraints corresponding to the four episodes ($G_1 - G_4$) previously described; another two constraints force the events representing the beginning of the apnea, abdominal limitation and thoracic limitation episodes to occur roughly at the same time; two additional constraints force the ending events of these same episodes to occur at the same time; one constraint limits the oxygen desaturation to begin after the beginning of an apnea; and an implicit constraint is required since all eight event types must occur within the same temporal window. The rest of the constraints between

the event types, which are not represented in the figure for better visualisation, correspond to the universal constraint $(-\infty,\infty)$ fitted to the temporal window $[-\omega,\omega]$, which indicates that any temporal arrangement within the same temporal window is allowed. The formal definition of the seed pattern is K=$< D,\mathcal{L} >$, where

$$D = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8\}$$

and the user-defined constraints are

$$\mathcal{L} = \{L_{12} = [1,\omega], L_{34} = [1,\omega], L_{56} = [1,\omega], L_{78} = [1,\omega], L_{13} = [-5,5], L_{15} = [-5,5],$$
$$L_{17} = [0,\omega], L_{24} = [-5,5], L_{26} = [-5,5], L_{ij} = [-\omega,\omega]\ otherwise\}$$

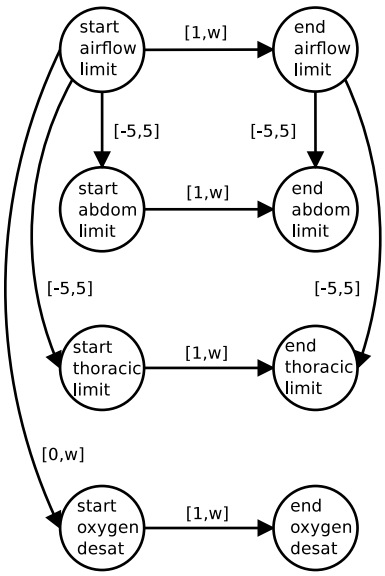


Figure 4.31: Seed pattern used in the validation experiments.

Figure 4.32 shows the execution time for two versions of the algorithm `ASTPminer`, with and without using a seed pattern, and several window sizes (up to 120 seconds). The version using the seed pattern requires consistently less time to finish the search than the version performing a full search, regardless of window size. This supports the hypothesis that the use of a seed pattern effectively helps to reduce the search space: the number of patterns generated is reduced because they need to be consistent with the provided information about the particular

domain. It is worth mentioning that, in both versions of the algorithms, candidates in those iterations where i is an odd number contain at least one event type belonging to an episode type where the other event type of the same episode type is not present. Therefore, the frequency calculation step is omitted in odd numbered iterations.

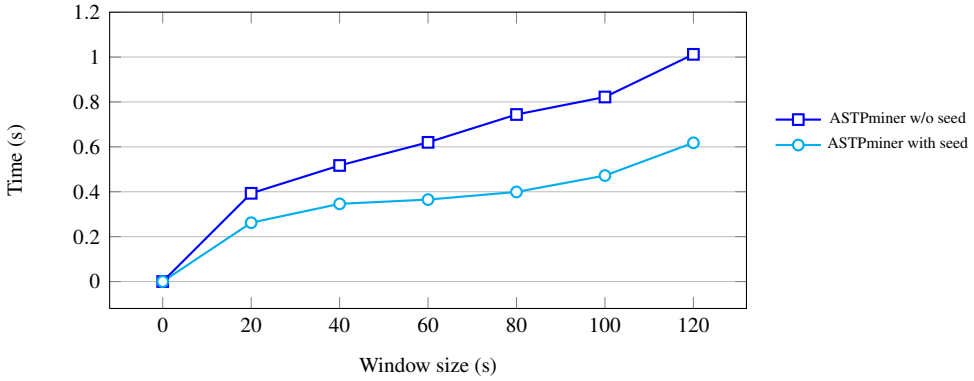


Figure 4.32: Time required by the `ASTPminer` algorithm with and without seed pattern.

As the window size increases, so do the number of events in the window and the number of overlapping pattern occurrences in the collection, resulting in a rapid increase in the time required for the search procedures. Some apnea episodes can have a duration of 60 seconds, specially in late hours of the night or in long-time SAHS patients, when oxygen saturation takes longer to recover, so it is possible that the same temporal window holds two or more occurrences of the apnea episode, increasing the cost of pattern search. By introducing constraints in the temporal arrangements between events, seed patterns reduce the number of possible occurrences to those satisfying the constraints, allowing the algorithm to discard some overlapping occurrences and therefore reducing the cost of pattern recognition and execution time, while providing more relevant and interesting knowledge.

Table 4.4 represents the number of candidate and frequent patterns found by `ASTPminer` using a window size of 80 seconds and a frequency threshold of 30 occurrences when the four episodes are defined, with and without introducing the seed pattern displayed in figure 4.31. The 'Combinations' columns represent the number of all possible combinations of frequent patterns found in the previous iteration. 'Candidates' displays the number of consistent candidates generated from all 'Combinations' in both cases. 'Frequent' shows the number of frequent patterns found among 'Candidates' in both situations. These results show that the

number of patterns involved in any stage of the process is reduced when the seed pattern is provided. It is worth noting the reduction of possible combinations for every pattern size when the seed pattern is provided. In addition, the number of frequent patterns found throughout the process is also reduced, reaching the point where only one frequent pattern of size 8 is obtained. Again, as pointed out in the discussion of the validation of the ASTP_SEED algorithm (section 4.3), the number of combinations and candidate patterns of size 2 is not available (NA), since frequent patterns of size 2 are obtained through the use of a clustering procedure, instead of the combination procedure. These frequent patterns contribute to the final number of patterns displayed in the last row of table 4.4, making that the total number of frequent patterns found is greater than the total number of generated candidates.

Size	Combinations		Candidates		Discarded		Frequent	
	seed	w/o seed	seed	w/o seed	seed	w/o seed	seed	w/o seed
1	8	8	8	8	0	0	8	8
2	NA	NA	NA	NA	NA	NA	32	44
3	86	215	71	157	15	58	71	157
4	215	6754	101	330	108	2084	99	328
5	422	4.9×10^5	82	363	187	8256	82	363
6	385	5.0×10^6	38	227	120	8813	38	209
7	101	1.4×10^7	9	54	26	3454	9	54
8	2	2.1×10^6	1	7	1	286	1	4
Total	1219	2.1×10^7	310	1146	457	22951	340	1167

Table 4.4: Number of possible candidates, candidates generated, combinations discarded and frequent patterns in the database, using the ASTPminer algorithm with and without seed pattern (using a window size ω=80 s., and a frequency threshold f_{min}=30 occurrences).

Figure 4.33(a) shows the pattern of size 8 found by the algorithm when using the seed pattern from figure 4.31 and a window width of 80 seconds. Some of the constraints have been removed for better visualization. The pattern represented corresponds to a central apnea episode. It is worth noting that extending the seed pattern in this case does not add new event types to the seed pattern, since it already included all event types in the collection, but just refines the initial constraints provided by the user, discovering more specific and probably interesting knowledge.

Figure 4.33(b) shows one of the patterns of size 8 found by the algorithm using a window width of 80 seconds when no seed pattern is provided by the user. Some of the constraints

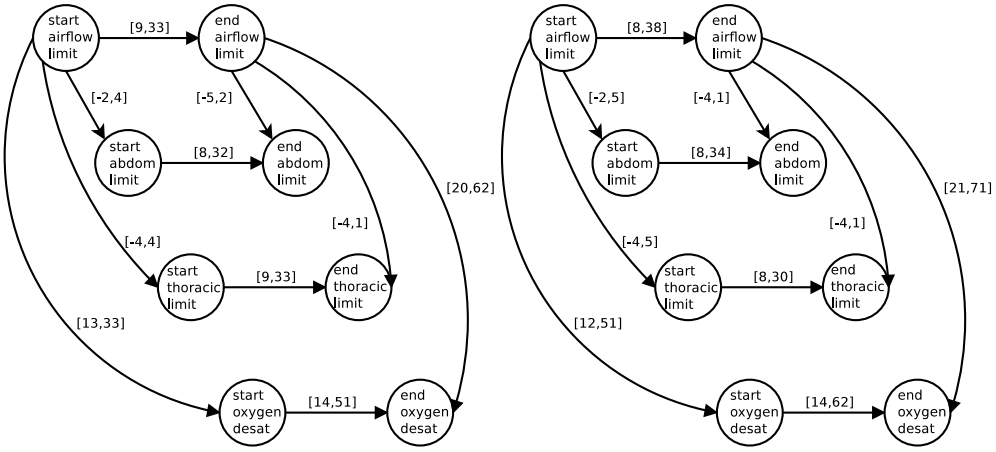
have been removed for better visualization. This pattern corresponds to a less specific central apnea episode than the one represented in figure 4.33(a). As can be seen, both versions of the algorithms obtain very similar patterns, although they have been obtained through different processes.

Introducing a seed pattern involves two fundamental differences. The first difference lies in the improvement of the performance of the mining process, as can be seen in figure 4.32 and table 4.4. By removing those patterns not consistent with the knowledge introduced, the search scope is reduced and therefore the time required by the mining process is improved. Table 4.4 shows how the number of possible patterns is reduced from 2.1×10^7 to 1219 when the seed pattern is provided, therefore reducing the cost of the candidate generation procedure. The number of frequent patterns falls from 1167 to 340, thus reducing the time required by the frequency calculation procedure. The second difference lies in the constraints of the resulting pattern shown in figure 4.33. Introducing a seed pattern allows the mining process to focus on those temporal arrangements between events that are consistent with the knowledge provided, and remove those that do not satisfy any of the constraints specified by the seed knowledge. The removal of temporal arrangements from the process modifies the temporal distance distributions, thus possibly modifying the clustering results, leading to different patterns in later iterations. All frequent patterns obtained when a seed pattern is provided are consistent with the knowledge introduced, and therefore the user should obtain more interesting results.

4.5 ASTPminer validation: synthetic databases

The characteristics of the SAHS database are very specific of this particular domain, so the database does not permit to fully evaluate the performance of the developed algorithms. In order to test the validity of our algorithms with a more thorough set of databases covering a wider range of characteristics, a synthetic database generator was developed, in collaboration with the AIKE (Artificial Intelligence Knowledge Engineering) research group of the University of Murcia. This section details the rationale behind the data collections generated, the parameters used by the database generator and how databases are generated from a given set of parameters.

The database generator we have developed is based on the IBM Quest synthetic data generator described in [Agrawal and Srikant, 1994, Agrawal and Srikant, 1995, Srikant, 1996], and may generate two different types of databases. The first type is that of a transactional database, where each transaction has a number of items associated, called itemset, and each



(a) Using a seed pattern.

(b) Without seed pattern.

Figure 4.33: Patterns of size 8 found with and without introducing a seed pattern, with a window width of 80 seconds.

item represents the occurrence of an event. This type of database is used to evaluate algorithms intended to find sequential patterns which represent sequences of itemsets where itemsets follow a total order. In this type of problem, the temporal distance between itemsets is either fixed or not important. Therefore, we will not describe how these databases are generated, nor will they be used for the validation of the algorithms developed.

The second type of database is that of a collection of recordings, that is, event sequences involving episodes, where each sequence consists of an ordered set of time-stamped events. These databases are intended to be used to induce a set of frequent temporal patterns that represent common temporal arrangements between the events and/or episodes. The database generator creates a number of temporal patterns according to definition 9 and then proceeds to introduce instances of them in the collection. The parameters shown in table 4.5 allow the database generator to produce a wide array of situations to validate the algorithms.

The user selects the number of different event types $|\mathcal{E}|$ and episode types $|\mathcal{G}|$, and the average duration for the episodes, $|\mu_G|$. These event and episode types will be used to generate a number of different patterns $|P|$ involving them, that will then be introduced in a collection of $|\mathcal{S}|$ sequences. The user also selects the maximum pattern size max_P and the average size μ_P. The parameter Ω determines the maximum temporal distance between any two events in a pattern. This value is related to the size of the temporal window ω to be used in the mining

process: if the user of the mining algorithm provides a value $\omega < \Omega$, patterns generated with this Ω value will not be found. The maximum time duration of the sequences t_S, the maximum number of events in a sequence $|S|$ and the maximum number of events in a temporal instant $|T|$ are used for defining the density of events in the collection of sequences Δ. Higher density values are expected to have a greater impact in the algorithm performance, since there will be more events in every temporal window.

$\|\mathcal{E}\|$	Number of event types.
$\|\mathcal{G}\|$	Number of episode types.
$\|\mu_G\|$	Average episode duration.
$\|P\|$	Number of patterns to generate.
$\|\mathcal{S}\|$	Number of sequences.
f_P	Number of occurrences of each pattern.
μ_P	Average pattern size.
max_P	Maximum pattern size
Ω	Maximum temporal distance between any two events in a pattern.
t_S	Maximum time duration of a sequence.
$\|S\|$	Maximum number of events in a sequence.
$\|T\|$	Maximum number of events in a temporal instant.
Δ	Density of events in the collection.
N	Total number of events in the collection.

Table 4.5: Synthetic database generator parameters.

4.5.1 Synthetic databases generation

For our validation experiments we have generated a set of different synthetic databases trying to model a broad range of real situations that may be found in different domains. The parameters defining these synthetic databases are shown in table 4.6. The $|P|$ column represents the number of different patterns that we aim to introduce in the database. Each pattern P is generated in two steps. First, the number of event types in the pattern (the pattern size) is randomly determined by a Poisson distribution of average μ_P, generating a new number if the result is higher than the maximum size allowed for a pattern, max_P. Event types are randomly chosen among all $|\mathcal{E}|$ event types, where each event type can be chosen at most once. In terms of pattern size, an episode type is considered as an event type. Then, the possible temporal constraints between events are generated. For each pair of event types E_i and

E_{i+1} in D an interval $[min, max] \subseteq [-\omega, \omega]$, where $max - min \leq \Omega$, is randomly calculated following an uniform distribution in the interval $[-\Omega, \Omega]$. Therefore, given a pattern of size i, the procedure randomly generates $i-1$ constraints, initialises the rest of the constraints to the interval $[-\Omega, \Omega]$, and then uses a Floyd-Warshall `ALL_SHORTEST_PATHS()` algorithm to propagate the constraints. In the case that episode types are present in the pattern, constraints where an episode type is involved are assumed to represent temporal distances with respect to the beginning of the episode.

Once all patterns have been generated, f_P instances of every pattern are created and inserted into the sequences. Each occurrence of a pattern P is generated by choosing a random value for each constraint $L_{i,i+1} = [a, b]$, using a uniform distribution in the interval $[a, b]$ to obtain the temporal distance between the events e_i and e_{i+1}. A random sequence in the collection is chosen to insert the pattern occurrence, and then a random initial instant in the interval $[0, t_S]$ is calculated to insert the first event of the occurrence. The rest of events and episodes in the pattern are then inserted according to the temporal distances previously calculated. If inserting the pattern occurrence takes the number of events above the $|S|$ value a new sequence is chosen. If inserting the occurrence at the alloted time would result in a temporal instant having more than $|T|$ events, or would make two occurrences of the same episode type overlap, a different time instant within the same sequence is chosen. If there are no available time instants in the sequence, a new sequence is chosen. If there are no available sequences to introduce the occurrence, an error is produced.

The user can choose whether to allow the database generator to make use of already placed events in a sequence to insert new occurrences, or force it to calculate a new beginning instant when it detects it needs to insert an event in a temporal instant where the same event is already present. It is not possible for two occurrences of the same episode type to overlap. In addition, when an episode is inserted, it is necessary to verify that no temporal instant between its beginning and its ending will contain more than $|T|$ events.

Once the database is generated, we can compute the total number of events N involved.

4.5.2 Experimental results with synthetic databases

For comparison purposes, we summarize now the corresponding parameter values in the case of the SAHS database. The number of event types $|\mathcal{E}|$ is 0, and the number of episode types $|\mathcal{G}|$ is 4 (implicitly defining a set of 8 event types). The number of sequences is 50, and the total number of events in the collection is N=120212. Each sequence has an average

DB	$\|P\|$	μ_P	max_P	Ω	f_P	Δ	t_S	$\|T\|$	$\|\mathcal{E}\|$	$\|\mathcal{G}\|$	$\|\mu_G\|$
SDB1	10	5	8	60	2000	0.07	8000	10	6	4	10
SDB2	10	5	11	60	5000	0.15	20000	10	6	4	10
SDB3	10	5	9	60	10000	0.25	20000	10	6	4	10
SDB4	10	5	8	60	15000	0.39	8000	10	6	4	10
SDB5	1	14	14	80	5000	0.22	8000	4	12	1	10
SDB6	1	19	19	80	4000	0.25	8000	4	13	3	10
SDB7	1	13	13	80	5000	0.2	8000	4	13	0	-
SDB8	1	10	10	40	5000	0.4	4000	4	10	0	-
SDB9	1	13	13	20	10000	0.8	2000	4	13	0	-
SDB10	1	10	10	10	7000	2.25	2000	4	12	0	-

Table 4.6: Synthetic databases parameters.

duration of 8 hours, meaning that t_S=28800 s. The density of events in the collection Δ is approximately 0.15. As we can see, the main features of this database are the existence of episodes, and no independent event types, and a low density, that means a reduced number of events in a temporal window.

In order to compare the results of `ASTPminer` in the SAHS database to the results of the validation experiments using the set of synthetic databases, figure 4.34 shows the time required by the `ASTPminer` algorithm with and without seed pattern, in the SAHS database for different window sizes ω, and a frequency threshold of 30 occurrences.

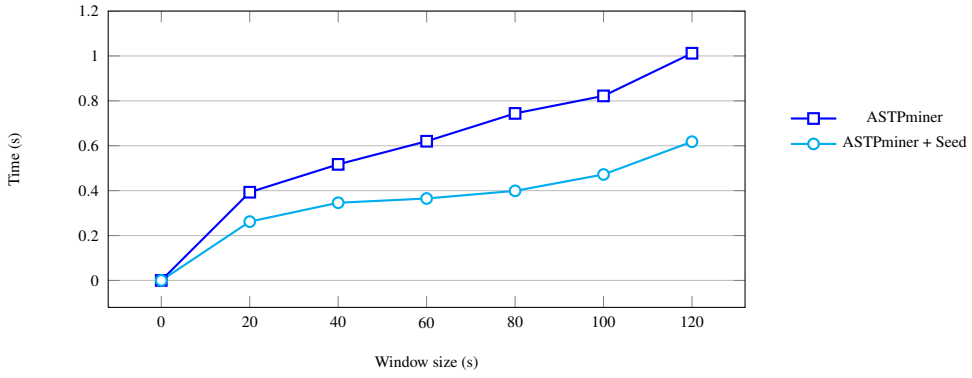


Figure 4.34: Time required by the `ASTPminer` algorithm with and without seed pattern, in the SAHS database. max_P=8, N=120212, Δ=0.15, $|\mathcal{E}|$=0, $|\mathcal{G}|$=4.

The first synthetic databases in table 4.6 (SDB1 to SDB6) correspond to databases with both event and episode types, whereas the last ones (SDB7 to SDB10) are databases that do not contain any episode type, only event types are included in the collections.

The first set of experiments were carried out using databases SDB1 to SDB6, and the aim was to analyse the impact of pattern size (values between 5 and 19), density of events (between 0.15 and 0.39) and ratio between the number of event and episode types. It was expected that the algorithm behaved better for databases with a higher number of episode types, as is the case of the SAHS database. Figures 4.35, 4.36, 4.37 and 4.38 show the execution time required by `ASTPminer` in databases SDB1 to SDB4, whose main difference is an increasing density value (more events to analyse in each temporal window). As we can observe in the figures, execution time increases notably with the density.

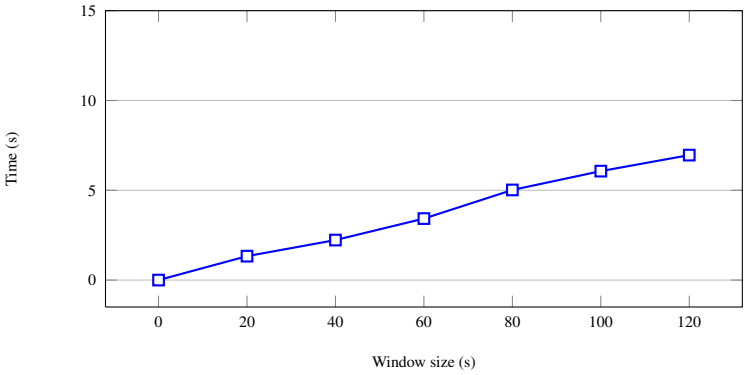


Figure 4.35: `ASTPminer` execution time in synthetic database SDB1: max_P=9, N=136000, Δ=0.07, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4

Figures 4.39 and 4.40 show the execution time for `ASTPminer` in databases SDB5 and SDB6, corresponding to databases with a low ratio of episode types and patterns of greater sizes (14 and 19 respectively).

The last set of experiments correspond to databases without episode types in the collection. We can observe in figure 4.41 that, for a database with a similar density to the SAHS database, which is the case of SDB7, the execution time greatly increases (from 450 to 4000 seconds). This higher value is also due to the higher number of event types and higher pattern size, but confirms that `ASTPminer` has better performance in databases incorporating episodes in the collection.

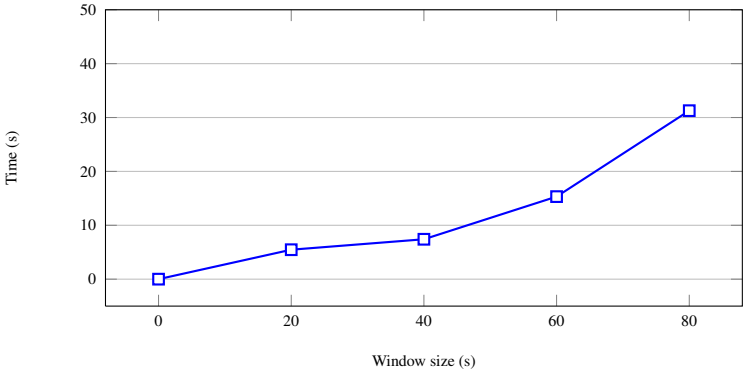


Figure 4.36: ASTPminer execution time in synthetic database SDB2: max_P=11 , N=275000, Δ=0.15, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4

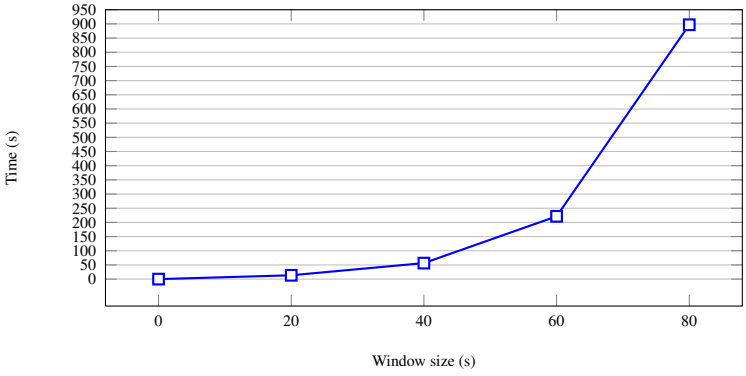


Figure 4.37: ASTPminer execution time in synthetic database SDB3: max_P=9 , N=470000, Δ=0.25, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4

Figures 4.42, 4.43 and 4.44 provide execution times for databases without episode types and an increasing density value. As expected, execution time increases since more events have to be checked in each temporal window.

These sets of experiments corroborate the efficiency of ASTPminer in the discovery of frequent temporal patterns in databases with a low density of events in the collection, and a high ratio of episode types versus event types. For different types of databases, new strategies must be designed to improve execution time of the mining procedure.

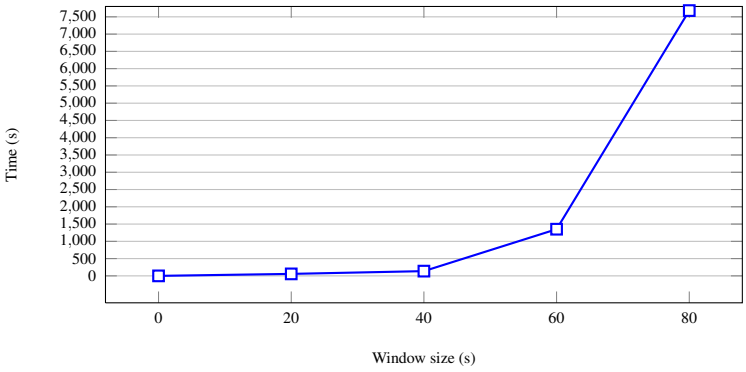


Figure 4.38: `ASTPminer` execution time in synthetic database SDB4: max_P=8, N=705000, Δ=0.39, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4

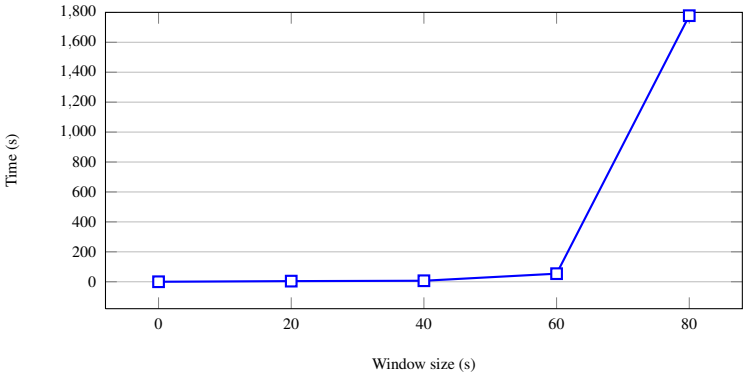


Figure 4.39: `ASTPminer` execution time in synthetic database SDB5: max_P=14, N=70000, Δ=0.22 , $|\mathcal{E}|$=12, $|\mathcal{G}|$=1

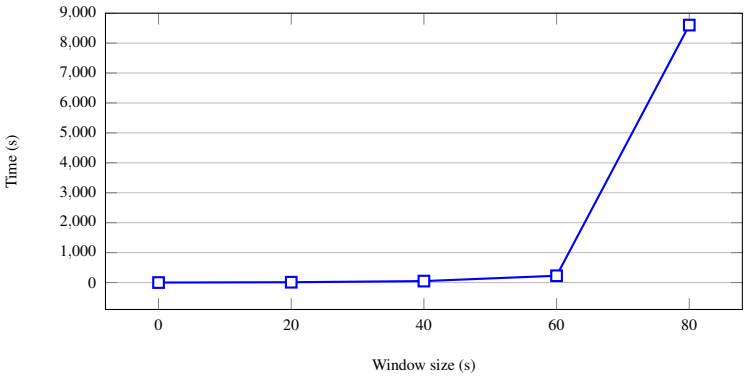


Figure 4.40: `ASTPminer` execution time in synthetic database SDB6: max_P=19, N=76000, Δ=0.25, $|\mathcal{E}|$=13, $|\mathcal{G}|$=3

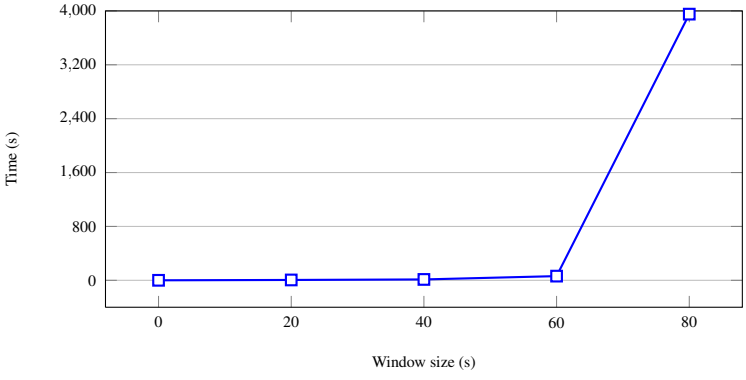


Figure 4.41: `ASTPminer` execution time in synthetic database SDB7: max_P=13, N=65000, Δ=0.2, $|\mathcal{E}|$=13, $|\mathcal{G}|$=0

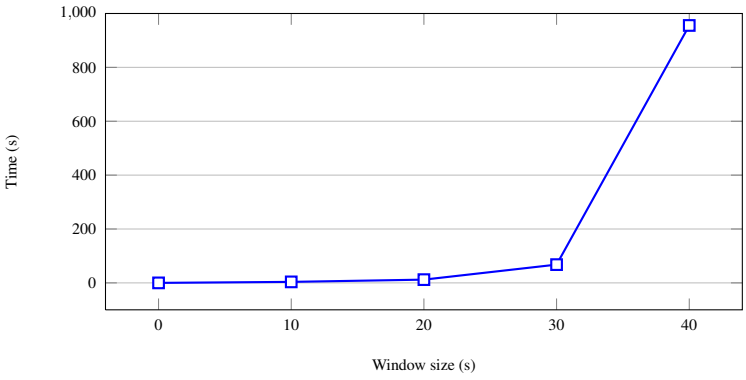


Figure 4.42: `ASTPminer` execution time in synthetic database SDB8: max_P=10, N=50000, Δ=0.4, $|\mathcal{E}|$=10, $|\mathcal{G}|$=0

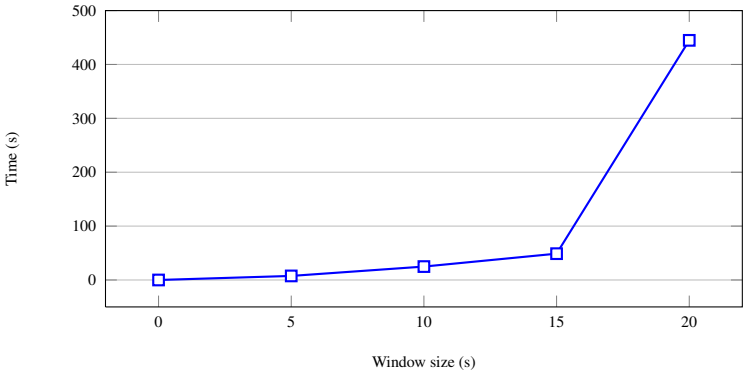


Figure 4.43: `ASTPminer` execution time in synthetic database SDB9: max_P=13, N=95212, Δ=0.8, $|\mathcal{E}|$=13, $|\mathcal{G}|$=0

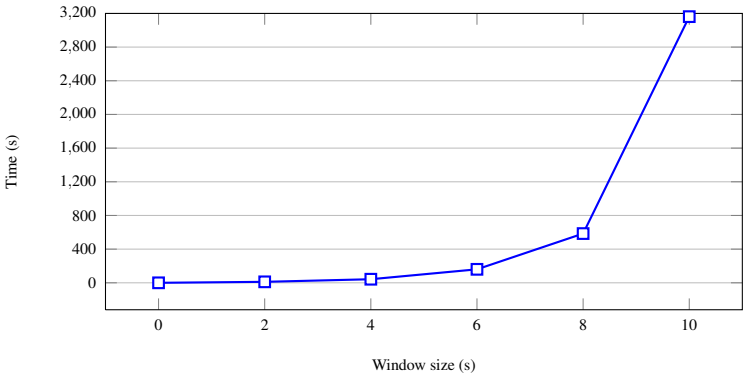


Figure 4.44: `ASTPminer` execution time in synthetic database SDB10: max_P=12, N=123402, Δ=2.25, $|\mathcal{E}|$=12, $|\mathcal{G}|$=0

CHAPTER 5

HSTPMINER

Chapter 4 presented the ASPTminer algorithm that searches for frequent patterns represented as a STP in a collection of sequences, where events are associated with instantaneous occurrences of observables while episodes are associated with interval-based observables. Experiments with real-life data, in the form of annotated sequences obtained from polysomnographies of SAHS patients, showed that ASTPminer is able to infer useful knowledge. In addition, ASTPminer allows a domain expert to introduce knowledge into the mining process. Experimental results with the SAHS database showed how introducing knowledge effectively improves the performance of the algorithms, as it allows the mining process to focus on temporal arrangements of interest to the user, therefore reducing both the search scope and the time required by the algorithms to perform the search procedure. It was also shown that ASTPminer dramatically improves its performance when the database mostly consists of episodes.

The SAHS database, however, represents a very specific problem. Experimental results with synthetic databases showed that there is room for improvement in databases with different characteristics. This chapter presents some improvements to the ASTPminer algorithm that aim to increase the performance in these databases. The resulting algorithm is named Hierarchical Simple Temporal Problem miner, HSTPminer, because it builds and maintains a pattern hierarchy throughout the mining process, with the aim of improving the performance of the FREQUENCY_CALCULATION procedure. The intent of this hierarchy is to allow the algorithm to use information regarding which patterns where found in each window in previous iterations of the procedure, with the aim of reducing the number of candidate patterns that need to be checked in each temporal window as much as possible, regardless of whether

the user introduces knowledge into the process or not. While the ASTPminer needed to verify every candidate pattern that contained the last event introduced into the window, the HSTPminer algorithm will only verify those candidates that also extend frequent patterns found in the same window in the previous iteration. This will improve the performance of the algorithm in databases where the number of potential candidates is large, yet the number of actual occurrences remains low, such as databases with long patterns but a somewhat reduced number of events per time unit. Before going into the details of the HSTPminer algorithm, we will introduce the data structure that will be used to maintain the pattern hierarchy, a set enumeration tree.

5.1 Set enumeration tree

The data structure chosen to maintain the pattern hierarchy is a set enumeration tree [Rymon, 1992]. This structure consists of a set of supernodes, where each supernode may be either the empty set or the root of several disjoint trees. A supernode in the level i of the tree contains all patterns that share the first $i-1$ event types, and groups together patterns into nodes according to event type i. Formally:

Definition 17 *A **node** in the level i of the tree is a pair* $N_k^i = (A_k^i, \{P_l^i =< D_l^i = A_k^i, \mathcal{L}_l^i >\})$, *where the first element of the pair is a temporal association of size i and the second element is the set of temporal patterns whose set of event types is the same as the temporal association of the first element. We can represent the node that contains the temporal association* A_k^i *as* N_k^i.

We can also define an order relation among nodes:

Definition 18 *Let* $<$ *be an **order relation** between two nodes* N_j^i *and* N_k^i *such that* $N_j^i < N_k^i$ *if, and only if,* $A_j^i < A_k^i$.

By extending the lexicographical order among event types defined in chapter 3, nodes are also lexicographically ordered. Therefore, nodes within a supernode are organised following the lexicographical order of the $i-$th event type of their corresponding temporal associations, as the rest of the $i-1$ event types are shared between all nodes of the supernode. Therefore, a supernode can be formally defined as:

Definition 19 *A **supernode** is an ordered set $SN = \{N_1, \ldots, N_n\}$ of n nodes, where $N_i < N_{i+1}, \forall i, 1, \ldots, n-1$. The temporal associations of all nodes included in a supernode share all of their event types except the last.*

By extending the lexicographical order between nodes to a lexicographical order between supernodes, we denote the $j-$th supernode in the level i of the tree as SN_j^i, and we denote the set of all supernodes in level i as SN^i.

Each node in a supernode can be the parent of a different supernode. This supernode will contain every pattern that can be obtained by adding one event type at the end of the temporal association of the parent node, while maintaining the lexicographical order. For simplicity in the notation, we can omit the set of temporal patterns in a node, unless otherwise required in the definition or procedure, and therefore we represent a node by its corresponding temporal association.

Definition 20 *Let N_j^i be a node in the set enumeration tree. N_j^i can serve as the **parent** for the root supernode SN of a subtree, denoted as $N_j^i = PARENT(SN)$. Conversely, the subtree spawned by the node can be denoted as $SN = SUBTREE(N)$.*

Figure 5.1 shows an example of a set enumeration tree for the set of event types $\mathcal{E} = \{A,B,C,D\}$. For ease of visualisation, only the temporal association of each node is represented. The root of the tree is the supernode labelled as '1', which contains the nodes for the temporal associations of size one that correspond to the event types *A*, *B*, *C*, and *D*. The node *A* is the parent for the tree that begins with the supernode labelled as '2', which contains temporal associations *AB*, *AC* and *AD*, where the first event type is always *A*, the temporal association of the parent node. The node *AB* then serves as the parent for the subtree with supernode '5' at its root. This supernode contains nodes *ABC* and *ABD*, which are all temporal associations that begin with the event types *A* and *B*, the ones in the parent node. The leaf supernode in this branch of the tree is supernode '8', which only contains the node *ABCD* and has node *ABC* as its parent. The rest of the branches in the tree are constructed similarly.

The set enumeration tree has been used in the bibliography to efficiently mine associations [Bayardo, 1998, Coenen et al., 2004], closed partial orders [Pei et al., 2006] and temporal patterns [Guil et al., 2004]. In `HSTPminer` we will use the set enumeration tree to store the patterns involved in the mining process, with the aim of making them more readily accesible and improving the efficiency of the process.

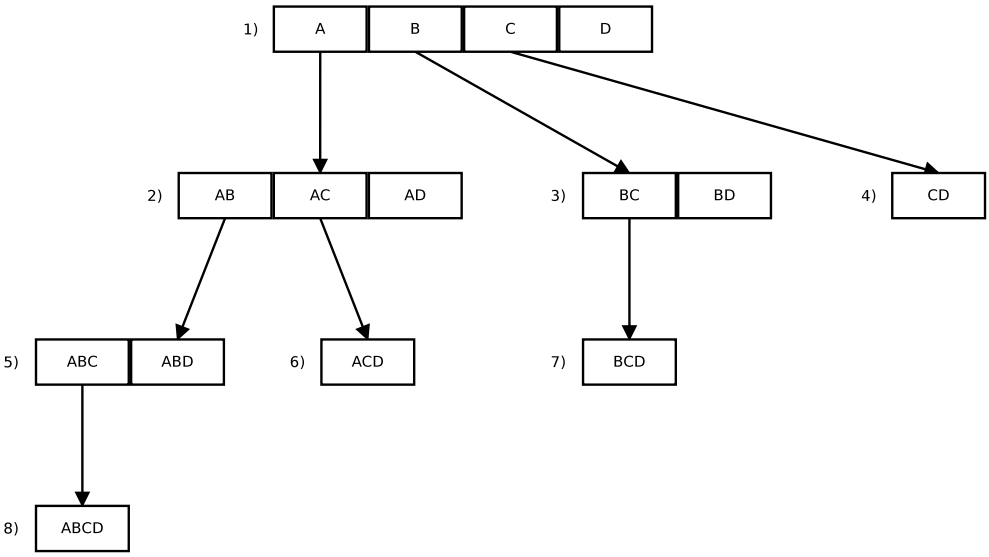


Figure 5.1: Set enumeration tree example.

5.2 Pattern hierarchy

If a pattern of size i can be found in a temporal window, then there is necessarily at least one occurrence of all of its subpatterns within the same window. Following the reverse reasoning, if there is no occurrence of a candidate pattern in a temporal window in one iteration, then no occurrence of any extension of that candidate can be found in later iterations. Therefore, given a temporal window in the collection, the mining process should focus on those candidate patterns that extend frequent patterns found in the same temporal window in the previous iteration.

Previous structures need to be adapted to make use of this concept. Event sequences are modified to allow every event e_k in a sequence to be associated with a set of patterns. The patterns in the annotation set are then used to reduce the number of candidate patterns that need to be tested in the window $W = \{e_i, \ldots, e_k\}$. Formally:

Definition 21 *An **annotated event sequence** is an ordered set of pairs* $S = \{(e_1, p_1), \ldots, (e_m, p_m)\}$ *where for all* $i < j$, $e_i < e_j$, *and* $p_i = \{P_1, \ldots, P_n\}$ *is a set of temporal patterns.*

As it will be thoroughly explained in the next section, the sets associated to the events in the collection are first built during the search for frequent patterns of size three. In the following iterations, the sets are updated by replacing the patterns each set contains with annotations of those candidates where at least one occurrence was found within the same temporal window.

In order to efficiently find the extensions of the patterns present in the annotation set it is necessary introduce a new structure. The structure used is a hierarchy, which associates every pattern with each one of its extensions by means of the event type present in the extension but not present in the pattern. This hierarchy was implicit in `ASTPminer`, as candidate patterns were built from frequent patterns found in the previous iteration. However, `HSTPminer` is based on making an efficient use of it.

There are two kinds of relations between patterns that need to be made explicit to make use of the pattern hierarchy. The first one links a pattern C_k^i with the set containing all of its subpatterns P_l^{i-1}, which represents those patterns that were combined to produce C_k^i. Formally:

Definition 22 *Let C_k^i be a candidate temporal pattern of size i. We define the set of all of its **subpatterns** as SUBPATTERNS(C_k^i) = $\{P_l^{i-1} =< D_l^{i-1}, \mathcal{L}_l^{i-1} > | P_l^{i-1} \in P^{i-1} \wedge P_l^{i-1} \preceq C_k^i\}$).*

The second relation of interest links a pattern P_l^{i-1} and an event type E to all of its extensions C_k^i that add the event type E.

Definition 23 *Let $P_l^{i-1} =< D_l^{i-1}, \mathcal{L}_l^{i-1} >$ be a frequent temporal pattern of size $i-1$ and $E \in \mathcal{E}$ an event type. We define the set of all of its **superpatterns** C_k^i extended with the event type E as SUPERPATTERNS(P_l^{i-1},E) = $\{C_k^i \,|\, D_k^i = D_l^{i-1} \cup \{E\} \wedge P_l^{i-1} \preceq C_k^i\}$.*

Figure 5.2 shows an example of a pattern hierarchy, where pattern ABC_1 is the result of the combination of AB_1, AC_1 and BC_1, whereas pattern ABC_2 is the result of combining AB_2, AC_2 and BC_1. Therefore, SUBPATTERNS(ABC_1) = $\{AB_1, AC_1, BC_1\}$ and SUBPATTERNS(ABC_2) = $\{AB_2, AC_2, BC_1\}$. In addition, SUPERPATTERNS(AB_1, C) = SUPERPATTERNS(AC_1,B) = $\{ABC_1\}$,SUPERPATTERNS(AB_2,C) = SUPERPATTERNS(AC_2,B) = $\{ABC_2\}$ and $\{ABC_1$, $ABC_2\}$ = SUPERPATTERNS(BC_1, A). If, for example, the event $e_j = (A, t_j)$ introduced into the temporal window has the set $p_j = \{AB_2, AC_2\}$ associated, it is only necessary to check if there are occurrences of the candidate pattern ABC_2. There can be no occurrence of the pattern ABC_1, because no occurence of any of its subpatterns was found in the previous iteration.

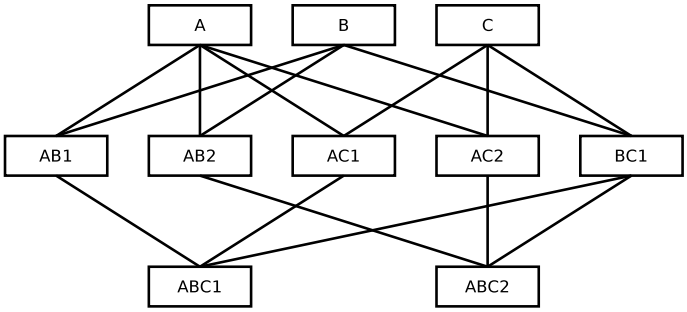


Figure 5.2: Pattern hierarchy example.

With the aim of managing the pattern hierarchy structure we use an auxiliary procedure labelled as `ENDING_EVENTS()`, which analyses the constraints of a pattern and calculates the set of possible event types that may finalise an occurrence of the pattern. Figure 5.3 shows the `ENDING_EVENTS()` procedure. Given a pattern C_l^i, the procedure calculates which event types in D_l^i can be found at the end of an occurrence of the pattern, according to the constraints found in $\mathcal{L}_k^i$. The procedure starts by assuming that all event types can be found at the end of an occurrence (line 2), an then analyses each constraint between the event types E_j and E_k to deduce whether one of the event types involved necessarily occurs before the other. If the minimum temporal distance between the events is strictly positive ($a > 0$), then E_j will always occur before E_k, so E_j is discarded (line 4). If, on the other hand, the maximum temporal distance between the events is strictly negative ($b < 0$), then the occurrence of E_k will always be present before the occurrence of E_j, so E_k is discarded (line 5). Once all constraints have been processed, the procedure returns the resulting set (line 6).

```
    procedure ENDING_EVENTS(C_l^i =< D_l^i, L_l^i >)
1   begin
2     EV ← D_l^i
3     for L_jk = [a,b] ∈ L_l^i do
4       if (a > 0) then EV_l^i ← EV_l^i − {E_j}
5       else if (b < 0) then EV_l^i ← EV_l^i − {E_k}
6     return(EV)
7   end;
```

Figure 5.3: `ENDING_EVENTS` procedure.

5.3 HSTPminer

The HSTPminer algorithm infers knowledge from temporal databases made up of sequences of time-stamped events and episodes. The result of the algorithm is a set of frequent temporal patterns represented by means of the STP formalism. These patterns are stored in the set enumeration tree structure described in section 5.1, as well as in the pattern hierarchy shown in section 5.2. These structures are exploited to improve the performance of the mining process. As its predecessor, HSTPminer allows the introduction of previous domain knowledge in the form of seed patterns also expressed using the STP formalism to focus the mining process and prune the search space. For the sake of simplicity in the description of the algorithms, we will assume that only one seed pattern is specified by the user.

HSTPminer uses a tuple of parameters $(\mathcal{S},\mathcal{F},\mathrm{K},\omega,\mathrm{f}_{min})$, where $\mathcal{S}$ represents the collection of event sequences, $\mathcal{F}=\mathcal{E}\cup\mathcal{G}$ is the set of types of facts in $\mathcal{S}$, K represents an initial seed pattern, ω the window width, and f_{min} the minimum frequency threshold. The HSTPminer algorithm (Figure 5.4) iteratively searches in $\mathcal{S}$ for frequent temporal patterns P, extension of K, whose occurrences are spread over ω time units, at most. Each frequent pattern P represents a set of similar temporal arrangements between the events and episodes present in the collection $\mathcal{S}$. Those subsequences of the collection where no occurrence of the seed pattern is present can be discarded from the rest of the procedure, as no occurrence of any extension of the seed pattern can be found in them. The rest of the parameters remain constant throughout the procedure.

The algorithm begins by searching for frequent event types (line 2), which represent frequent patterns of size one (line 3). Then, the INITIALISATION() procedure searches for all frequent patterns of size two consistent with the seed pattern K (line 4). Subsequent patterns are obtained through an iterative procedure, where each iteration i results in the set of frequent temporal patterns of size i. This process ends when no new frequent patterns are found in a given iteration. The output of the HSTPminer procedure is a set of frequent temporal patterns, all of them consistent with the knowledge represented by the seed pattern. As in ASTPminer, the HSTPminer algorithm uses three lists in every iteration i: the list A^i holds frequent temporal associations of i event types; the list C^i stores candidate patterns of size i , and the list $\mathrm{P}^i \subseteq \mathrm{C}^i$ contains those candidate patterns with i events whose frequencies were found to be higher than the frequency threshold f_{min}. These lists are divided among the different levels and nodes present in the set enumeration tree.

```
     procedure HSTPminer(S, F, K, ω, f_min)
1    begin
2      A^1 ← {E_j | E_j ∈ E ∧ f(E_j) ≥ f_min}
3      P^1 ← A^1
4      P^2 ← INITIALISATION(S, A^1, P^1, F, K, ω, f_min)
5      A^2 ← {D_j | P^2_j=<D_j, L_j> ∈ P^2}
6      i ← 3
7      while (P^{i-1} ≠ ∅ ∨ C^{i-1}_p ≠ ∅) do begin
8        C^i ← CANDIDATE_GENERATION(A^{i-1}, P^{i-1} ∪ C^{i-1}_p)
9        if C^i_c ≠ ∅ then
           P^i ← FREQUENCY_CALCULATION(C^i_c, S, F, ω, f_min)
10       A^i ← {D_j | P^i_j=<D_j, L_j> ∈ P^i}
11       i ← i+1
12     end;
13   end;
```

Figure 5.4: HSTPminer: Main algorithm.

List C^i is subdivided into two additional lists $C^i = C^i_c \cup C^i_p$. List C^i_c contains only those candidate patterns where, for every episode type, either both of its event types or none of them are part of the pattern. The candidates in this list will be the ones whose frequency will be verified by the FREQUENCY_CALCULATION() procedure. List C^i_p contains the rest of the patterns, where only one event type of at least one episode type is present. These patterns are necessary for the CANDIDATE_GENERATION() procedure in later iterations, but they are not to be searched for in the iteration in course. Section 5.3.3 details how both lists are constructed, and the criterion used to introduce one candidate pattern to each list. Frequent and candidate patterns are stored in the set enumeration tree structure described in section 5.1, which improves the efficiency of the CANDIDATE_GENERATION() procedure.

The algorithm finishes when one iteration results in both no new frequent patterns found and an empty C^i_p list, meaning that no additional patterns may be built. Notice that it is possible that the result of the candidate generation step in some iterations is an empty C^i_c list and a non-empty C^i_p list. In this case, the FREQUENCY_CALCULATION() procedure is not necessary, and the algorithm may proceed to the next iteration.

5.3.1 Initialisation

The INITIALISATION procedure shown in figure 5.5 obtains all frequent patterns of size 2 and stores them in the list P^2. Every pattern obtained in this procedure represents a set of temporal arrangements between two event types, where every temporal arrangement is consistent with the constraints present in the seed pattern K, and its frequency satisfies the minimum frequency threshold.

```
    procedure INITIALISATION(S, A^1, P^1, F, K, ω, f_min)
1   begin
2     K^2 ← FREQUENCY_CALCULATION({K}, S, F, ω, f_min)
3     if (K^2 = ∅) then stop
4     C^2 ← CANDIDATE_GENERATION(A^1, P^1)
5     P^2 ← K^2 ∪ FREQUENCY_CALCULATION(C^2, S, F, ω, f_min)
6     return P^2
7   end;
```

Figure 5.5: HSTPminer: Initialisation algorithm.

The procedure begins by obtaining the frequency of the seed pattern K$= < D_K, \mathcal{L}_K >$ (line 2), using the FREQUENCY_CALCULATION() algorithm described in section 5.3.2. The algorithm verifies all temporal windows of width ω throughout the collection $\mathcal{S}$, searching for occurrences of the seed pattern. If the result is that the seed pattern is not frequent, the algorithm stops (line 3). The result of this step is the set K^2 of frequent patterns of size 2 consistent with the seed pattern that involve only pairs of event types from the seed pattern, and there is at least one pattern for each different pair of event types.

Afterwards, the procedure builds the set C^2 of candidate patterns of size 2 where at least one event type was not included in the seed pattern, using the set of frequent event types in P^1 (line 4). These candidates are obtained by the CANDIDATE_GENERATION() procedure described in section 5.3.3.

Once the set C^2 has been generated, the FREQUENCY_CALCULATION() procedure verifies the frequency of the candidates. The list P^2 then includes patterns extracted from the seed pattern K^2, and frequent patterns extracted from C^2 (line 5).

As will be detailed in section 5.3.2, the pattern hierarchy will not be used in this step, and the set of patterns attached to every event in the collection will remain empty throughout the procedure.

5.3.2 Frequent pattern calculation

Figure 4.28 shows the `FREQUENCY_CALCULATION()` procedure used to verify the frequency of the candidate patterns in the data collection. The procedure incorporates the use of the pattern hierarchy to the search procedure. Occurrences of frequent patterns found in previous iterations are used to narrow down the number of candidates Q_l^i that need to be verified in each temporal window W. When an occurrence of a candidate pattern is found, the candidate is annotated in the set attached to the last event introduced into the window, as this event constitutes the ending event of the occurrence.

Every sequence S in the collection is analysed in search for occurrences of the candidate patterns, sequentially introducing events in the temporal window following an increasing temporal order. When an event e_j of the sequence is inserted in the window (line 10), all events e_k such that their temporal distance $t_j - t_k$ with e_j is greater than ω are removed from the window (line 11). In addition, any event $e_h = (o_e, v, t_h)$ representing the ending event of an episode $g = (o, v, t_k, t_h)$ where the beginning event e_k is no longer present in W is also removed from the window, even if $t_j - t_h < \omega$. Once the temporal window has been updated, the procedure needs to verify if there exist any occurrences of candidate patterns in the temporal window. Depending on the type of candidate patterns to be searched for, three different situations may arise.

The first case is found when the set of candidate patterns consists of the seed pattern (line 6). In this case, the procedure first tests if any of the events removed from the window belonged to any interval in R, a set of intervals where each intervals represents the beginning and ending instants of an occurrence of the seed pattern. Any event that does not belong to any of these intervals is not part in an occurrence of, nor is it found in the temporal proximity of an occurrence of the seed pattern. Therefore, it cannot belong to any extension of the seed pattern and can be safely removed from the mining process (line 7). Then, any interval that does no longer overlap with the temporal window W is removed from R, as no further event in the sequence may be present in them (line 8). The procedure then searches for occurrences of the seed pattern in the temporal window (line 14), considering that there are still no annotations

```
   procedure FREQUENCY_CALCULATION(Q^i, S, F, ω, f_min)
 1 begin
 2   for S ∈ S do begin
 3     R ← ∅
 4     W ← ∅
 5     for (e_j, p_j) ∈ S do begin
 6       if Q^i = {K} then begin
 7         S ← S − {e_k | t_j − t_k > ω ∧ ∀[t_a,t_b] ∈ R, t_k ∉ [t_a,t_b]}
 8         R ← R − {[t_a,t_b] ∈ R | t_j − t_b > ω}
 9       end;
10       W ← W ∪ {e_j}
11       W ← W − ({e_k | t_j − t_k > ω} ∪
           {e_h = (o_e, v, t_h) ∈ W | g = (o, v, t_k, t_h) ∧ t_j − t_k > ω})
12       np ← ∅
13       if Q^i = {K} or i = 3 or i = 4 then
14         VERIFY_PATTERNS(Q^i, W, F, R, np, E_j)
15       else begin
16         for P∈ p_j do
17           if P = P_l^{i−1} =< D_l^{i−1}, L_l^{i−1} > then
18             for E_k ∈ E do
19               V^i ← SUPERPATTERNS(P_l^{i−1}, E_k)
20               VERIFY_PATTERNS(V^i, W, F, R, np, e_j, p_j)
21             end;
22             if ∃Q_m^i ∈C_p^i, P_l^{i−1} ⪯Q_m^i then np ← np ∪ {P_l^{i−1}}
23           end;
24           if P = P_l^{i−2} =< D_l^{i−2}, L_l^{i−2} >
25             for G = (o, v, T_k, T_m) ∈ G do
26               for P_h^{i−1} =< D^{i−1}, L_h^{i−1} >∈SUPERPATTERNS(P_l^{i−2}, E_k) do
27                 V^i ← SUPERPATTERNS(P_h^{i−1}, E_m)
28                 VERIFY_PATTERNS(V^i, W, F, R, np, e_j, p_j)
29         end;
30         p_j ← np
31   end;
32   if i = 2 ∨ Q^i = {K} then P^i ← CLUSTERING(N = {n_{k_p k_q}}_{p,q}, f_min)
33   else P^i ← {Q_l^i | Q_l^i ∈Q^i ∧ f(Q_l^i) ≥ f_min}
34   return(P^i)
35 end;
```

Figure 5.6: HSTPminer: Frequency calculation.

in p_j, and the VERIFY_PATTERNS() procedure will not put any annotations of the seed pattern in the annotation set.

Similarly, the second case appears when the set of candidate patterns consist of temporal patterns of size three (line 13). In this case, the procedure does not remove any event from the collection, but there are still no annotations present in p_j. Therefore, the procedure needs to verify if there are occurrences of any of the candidates generated (line 14). The difference with respect to the seed pattern case is that as a result of the VERIFY_PATTERNS() procedure, the set np will contain references to the patterns that presented at least one occurrence in the window. The case $i = 4$ also falls in this case. Candidate patterns in C_p^3 could not be searched for in the previous iteration, and therefore there are no annotations of their extensions in C_c^4. These extensions need to be verified as in the case $i = 3$.

In the third and general case $i > 4$ (line 15), the annotation sets have already been initialised and can be used to constrain the number of patterns from Q^i that need to be verified in the temporal window W. If p_j contains no annotations, then no pattern needs to be verified and the procedure may skip to the next event in the collection (line 5). Otherwise, the procedure sequentially reads patterns from the annotated set (line 16), where two different types of annotations can be found. The first type corresponds to annotations made in the previous iteration, corresponding to patterns of the form P_l^{i-1} (line 17), whereas the second type corresponds to annotations made two iterations before the current one, where patterns annotated are of the form P_l^{i-2} (line 24). For each pattern P_l^{i-1}, the procedure iteratively uses all event types (line 18) to extend the pattern in the annotation set (line 19) and then verifies each set of extensions separately (line 20). In addition, if there is an extension of P_l^{i-1} that belongs to the set C_p^i, then that extension cannot be searched for in the iteration in course, so the pattern will not be attached to any annotation set, so none of its extensions in the set C_c^{i+1} will be verified in the next iteration. Therefore, in order to avoid this situation, either the extensions in C_p^i need to be added to the set np or the pattern P_l^{i-1} itself is added. With the aim of minimising the number of annotations in each set, P_l^{i-1} is added (line 22).

In the next iteration of the mining process, these annotations will take the form P_l^{i-2} (line 24). In this case, instead of iterating over the set of event types, the procedure needs to test extensions where complete episodes $G = (o, v, T_k, T_m)$ have been added (line 25). This is done in two steps. In the first step, the beginning event type $E_k = (o_b, v, T_k)$ of the episode is used to extract the extensions of P_l^{i-2} (line 26), and in the second step the ending event type of the episode $E_m = (o_e, v, T_m)$ is added to each of these extensions (line 27). The resulting patterns

are then verified in the temporal window (line 27) and their occurrences annotated in the set np.

Once all annotations in p_j have been processed, the set p_j is replaced with the set np of all annotations made for the current event (line 30). This process is repeated with every event of the collection. Then, the procedure discards any pattern not satisfying the minimum frequency threshold and puts all patterns that can be considered frequent in the list P^i (line 33). This last step is not performed when the set of candidate patterns corresponds either with the seed pattern or with the set of patterns of size two. In both cases, the result of the procedure is obtained through the use of a `CLUSTERING()` procedure (line 32) over the temporal distance distributions built within `VERIFY_PATTERNS()`.

```
   procedure VERIFY_PATTERNS(V^i, W, F, R, np, e_j, p_j)
 1 begin
 2   for V_r^i =< D_r^i, L_r^i > ∈ C_c^i,  E_j ∈ ENDING_EVENTS(V_r^i)  ∧
     (SUBPATTERNS(V_r^i) −
     ({P_m^{i-1} =< D_m^{i-1}, L_m^{i-1} > | E_j ∉ D_m^{i-1}}  ∪  {P_m^{i-1} | P_m^{i-1} ∈ C_p^{i-1}}) ) ⊆ p_j  do
 3     for X = {e_{k_1}, ..., e_{k_i} = e_j} ⊆ W  ∧  {E_{k_1}, ..., E_{k_i} = E_j} = D_r^i  do
 4       if  (∀e_{k_p}, e_{k_q} ∈ X, t_{k_q} − t_{k_p} ∈ L_{k_p k_q} ∈ L_r^i)  ∧
         (∀G = (o, v, T_{k_p}, T_{k_q}) ⪯ V_r^i  ⇒  ∃g = (o, v, t_{k_p}, t_{k_q}) | (o_b, v, t_{k_p}), (o_e, v, t_{k_q}) ∈ X)
         then begin
 5         UPDATE_DISTRIBUTION(N = {n_{k_p k_q}}_{p,q}, X)
 6         f(V_r^i)  ←  f(V_r^i) + 1
 7         if  V^i = {K}  then  R ← R  ∪  {[t_{k_1}, t_{k_i} = t_j]}
 8         if  i ≥ 3  then  np ← np  ∪  {V_r^i}
 9       end;
10   end;
11 end;
```

Figure 5.7: `HSTPminer`: Pattern verification.

The `VERIFY_PATTERNS()` procedure is shown in figure 5.7. The parameter V^i contains the set of candidate patterns that need to be searched for in the temporal window W. However, there are three conditions that any candidate V_r^i needs to satisfy in order to be tested in the current window (line 2). First, the pattern must belong to the set C_c^i. Second, the event type E_j corresponding to the event e_j currently under analysis must belong to the set of possible `ENDING_EVENTS`(V_r^i). And third, the set of annotations p_j must contain all subpatterns of

the candidate with the exception of those that do not contain E_j, and those that could not be annotated in the previous iteration of the mining process because they belonged to C_p^{i-1}. If the three conditions are met, then the procedure uses all subsequences X of the temporal window that satisfy some conditions to test if there are occurrences of the candidate. The conditions are that the subsequence must end with e_j, it must contain one event for each event type present in the candidate, and it must also contain the beginning and ending events of the same episode occurrence (line 3). For each one of these subsequences that satisfies the constraints of the candidate pattern V_r^i (line 4) the frequency of the candidate is increased (line 6) and the appropriate temporal distance distributions $\mathcal{N} = \{n_{k_p k_q}\}$ are updated (line 5). In addition, if the candidate is the seed pattern K, it is necessary to introduce the interval $[t_{k_1}, t_j]$ in R, where the beginning instant of the interval is the time of the first event in the subsequence e_{k_1} and the ending of the interval is marked by e_j (line 7). Otherwise, if $i \geq 3$ (line 8) then the candidate is added to the set of patterns found in the current window np, which will replace the set p_j once all annotations currently in p_j have been examined. It is worth mentioning that there may be non-frequent patterns present in some of the annotation sets of the collection. These annotations are not removed until the next iteration of the process, where the `FREQUENCY_CALCULATION()` procedure will ignore them.

No annotations of neither the seed pattern nor any pattern of size two are made. In both cases, frequent patterns are obtained by clustering the temporal distance distributions, and they are not available until the end of the `FREQUENCY_CALCULATION()` procedure. Since it is not possible to know which frequent pattern will be consistent with the occurrence, or if the occurrence will be consistent with any pattern at all, it would be necessary to annotate the temporal association, forcing the procedure to test all candidates in the next iteration of the mining process. This would be the same situation as if no annotation were made. In the case of the seed pattern, in addition to the previous comment, it is also necessary to point out that it would not be enough to make the annotations in the event e_j, and the pattern would need to be added to the annotation sets of every event in the occurrence. In this case the procedure will end by obtaining all frequent patterns of size two consistent with the seed pattern and having both event types within the seed pattern. Since not all of these patterns will contain the event type E_j, there would not be any annotation to indicate where to search for candidates that would not contain E_j.

An example of the `FREQUENCY_CALCULATION()` procedure can be found in figure 5.8. Let the window width be $\omega = 8$, the set of event types be $\mathcal{E} = \{E_1 = A,\ E_2 = B,\ E_3 = C,\ E_4 = D,\ E_5 = E\}$ and the set of epysode types be the empty set $\mathcal{G} = \varnothing$. Let

ABCD$_1 = < D_1, \mathcal{L}_1 >$ and ABCD$_2 =< D_2, \mathcal{L}_2 >$ be candidate patterns, where the first candidate consists of $D_1 = \{A, B, C, D\}$, $\mathcal{L}_1 = \{L_{12} = [1,2], L_{13} = [2,3], L_{14} = [3,4], L_{23} = [1,1], L_{24} = [2,3], L_{34} = [1,2]\}$, whereas the components of the second candidate are $D_2 = \{A, B, C, D\}$, $\mathcal{L}_2 = \{L_{12} = [-1,-1], L_{13} = [1,1], L_{14} = [0,1], L_{23} = [-2,-2], L_{24} = [-1,0], L_{34} = [1,2]\}$. Given both sets of constraints, the possible ending events for the patterns are ENDING_EVENTS(ABCD$_1$)= $\{D\}$ and ENDING_EVENTS(ABCD$_2$)= $\{B, D\}$. The sequence where the candidates will be searched for is $S = \{((A,1),\varnothing), ((B,2),\varnothing), ((A,5),\varnothing), ((B,6),\varnothing), ((C,7),\{ABC_1\}), ((D,9),\{ABD_1, ACD_1, BCD_1\}), ((E,10),\varnothing)\}$. Consider the following pattern hierarchy:

- SUPERPATTERNS(ABC_1,D)= $\{$ABCD$_1\}$
- SUPERPATTERNS(ABD_1,C)= $\{$ABCD$_1\}$
- SUPERPATTERNS(ACD_1,B)= $\{$ABCD$_1\}$
- SUPERPATTERNS(BCD_1,A)= $\{$ABCD$_1\}$
- SUPERPATTERNS(ABC_2,D)= $\{$ABCD$_2\}$
- SUPERPATTERNS(ABD_2,C)= $\{$ABCD$_2\}$
- SUPERPATTERNS(ACD_2,B)= $\{$ABCD$_2\}$
- SUPERPATTERNS(BCD_2,A)= $\{$ABCD$_2\}$
- SUBPATTERNS(ABCD$_1$)= $\{ABC_1, ABD_1, ACD_1, BCD_1\}$
- SUBPATTERNS(ABCD$_2$)= $\{ABC_2, ABD_2, ACD_2, BCD_2\}$

The procedure sequentially introduces the events into the temporal window, while finding no occurrence of any candidate as there are not enough different event types and no annotations of occurrences found in the previous iteration. When the event $(C,7)$ enters the window, its annotation set contains the pattern ABC_1, meaning that it could be possible to find occurrences of any of its extensions. However, there are only occurrences of three different event types (A, B and C) while four would be needed, since candidate patterns are of size four. Therefore no occurrence can be found and the new annotation set associated with the event will be empty. When event $(D,9)$ is introduced, the process reaches the situation depicted in "Temporal window 1". In this case, the annotation set contains three patterns suggesting

which candidates of size four might be present in the window, those extending any of the three. The first annotation that needs to be analysed corresponds to the pattern ABD_1. The procedure tries to extend de pattern in the annotation set with all event types in $\mathcal{E}$, one at a time. The only possible extension, which corresponds with the candidate $ABCD_1$, is found when the event type C is added. As D belongs to the set ENDING_EVENTS($ABCD_1$), the procedure verifies that all patterns in SUBPATTERNS($ABCD_1$) are in the annotation set, with the only exception of ABC_1, for it does not contain the event type D. Since the three subpatterns are present in the annotation set, the procedure searches for occurrences of $ABCD_1$ in all the subsequences of the temporal window that satisfy that there is one event for each event type in $ABCD_1$. The only subsequences satisfying this condition are $X_1 = \{(A,1), (B,2), (C,7), (D,9)\}$, $X_2 = \{(A,1), (B,6), (C,7), (D,9)\}$, $X_3 = \{(B,2), (A,5), (C,7), (D,9)\}$ and $X_4 = \{(A,5), (B,6), (C,7), (D,9)\}$. Sequence X_1 does not satisfy the constraints L_{13}, L_{14}, L_{23} and L_{24} of the pattern and therefore it is not an occurrence. Sequence X_2 does not satisfy L_{12}, L_{13} and L_{14}. Sequence X_3 does not satisfy L_{12}, L_{23} and L_{24}. However, the events in sequence X_4 satisfy all the constraints, and therefore both the frequency of the candidate and the temporal distance distributions between the event types need to be updated (lines 13-14 in figure 5.6). In addition, the candidate $ABCD_1$ is added to the set np of patterns found (line 15). Extensions to patterns ACD_1 and BCD_1 are then explored, but the only candidate pointed by them is $ABCD_1$ which was already used to search for occurrences and cannot be used again. As candidate $ABCD_2$ cannot be accessed from any of the annotations made during the previous iteration of the algorithm, it is not necessary to search for occurrences in the window, for none could be found. Therefore the procedure has fully explored the current temporal window, substitutes the set of annotations for the new one (line 17), and then reads the event $(E,10)$ from the event sequence. Reading this event moves the temporal window to the situation labelled as "Temporal window 2", where event $(A,1)$ is removed from the window. Again, there are no annotations associated with the event, so it is not necessary to search for occurrences of the candidates. The procedure ends by removing those candidates with a frequency value below the frequency threshold from the process (line 21), returning the remaining ones, which are considered frequent. In this step, the procedure does not remove non-frequent patterns from the annotation sets, so it does not need to access the data collection again.

The previous example assumed that there were no episodes in the collection. An example of the `FREQUENCY_CALCULATION()` procedure with episodes can be found in figure 5.9. Let the window width be $\omega = 8$, the set of event types be $\mathcal{E} = \{E_1 = A, E_2 = B,$

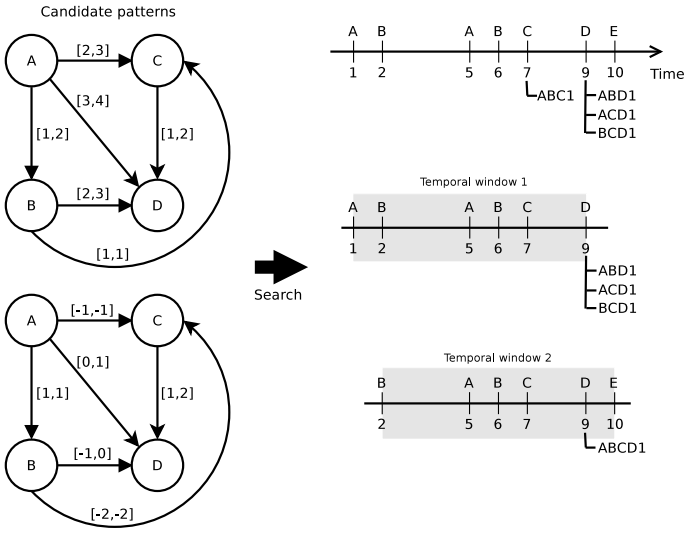


Figure 5.8: Pattern hierarchy frequency calculation example.

$E_3 = C$, $E_4 = D$, $E_5 = E\}$ and the set of epysode types be $\mathcal{G} = \{G_1 = (o_1, v_1, T_A, T_B)$, $G_2 = (o_2, v_2, T_C, T_D)\}$. Let ABCD$_1$ $=< D_1, \mathcal{L}_1 >$ be a candidate pattern, where $D_1 = \{A, B, C, D\}$, and $\mathcal{L}_1 = \{L_{12} = [1,2], L_{13} = [2,3], L_{14} = [3,4], L_{23} = [1,1], L_{24} = [2,3], L_{34} = [1,2]\}$. Given the set of constraints, ENDING_EVENTS(ABCD$_1$)$= \{D\}$. The sequence where the candidates will be searched for is $S = \{((A,1), \varnothing)$, $((B,2), \{AB_1\})$, $((A,5), \varnothing)$, $((B,6), \{AB_1\})$, $((C,7), \varnothing)$, $((D,9), \{CD_1\})$, $((E,10), \varnothing)\}$. There are two episodes $g_1 = (o_1, v_1, 1, 2)$ and $g_2 = (o_1, v_1, 5, 6)$ of the episode type G_1 represented by the events $(A,1)$ and $(B,2)$ as well as $(A,5)$ and $(B,6)$ in the sequence. There is also one episode $g_3 = (o_2, v_2, 7, 9)$ of the episode type G_2 represented by the events $(C,7)$ and $(D,9)$. Consider the following pattern hierarchy:

- SUPERPATTERNS(AB_1,C)$= \{$ABC$_1\}$
- SUPERPATTERNS(AB_1,D)$= \{$ABD$_1\}$
- SUPERPATTERNS(CD_1,A)$= \{$ACD$_1\}$
- SUPERPATTERNS(CD_1,B)$= \{$BCD$_1\}$
- SUPERPATTERNS(ABC_1,D)$= \{$ABCD$_1\}$
- SUPERPATTERNS(ABD_1,C)$= \{$ABCD$_1\}$

- SUPERPATTERNS(ACD_1,B)= $\{\text{ABCD}_1\}$
- SUPERPATTERNS(BCD_1,A)= $\{\text{ABCD}_1\}$
- SUBPATTERNS(ABCD_1)= $\{ABC_1, ABD_1, ACD_1, BCD_1\}$

The procedure sequentially introduces the events into the temporal window. When the event $(B,2)$ enters the window, its annotation set contains the pattern AB_1, meaning that it could be possible to find occurrences of any of its extensions. However, there are only occurrences of two different event types (A and B) while four would be needed, since candidate patterns are of size four. Therefore no occurrence can be found and the new annotation set associated with the event will be empty. The same reasoning applies to the event $(B,4)$. When event $(D,9)$ is introduced, the process reaches the situation in "Temporal window 1". In this situation, the annotation set contains one pattern CD_1 of size two. Since the size of the candidates in the current iteration is four, this annotation represents an occurrence of a pattern that was extended with an episode type, meaning that an episode type must be used to calculate the extensions instead of a single event type as in the previous example. In this case, the episode type G_1 is used. The procedure first obtains the set of all superpatterns of CD_1 with the beginning event type A of the episode type G_1, which results in the set $\{ACD_1\}$. Then, the procedure would use the ending event type D of G_1 to obtain the extensions of size four of each pattern in the set. In this example, this process results in the set $\{\text{ABCD}_1\}$. Since D belongs to the set ENDING_EVENTS(ABCD_1), the procedure would need to verify that all patterns in SUBPATTERNS(ABCD_1) are in the annotation set, with the only exception of ABC_1. However, every subpattern of ABCD_1 belongs to the set C_p^3, meaning that these patterns cannot be present in the annotation set, so the procedure needs to verify if there are occurrences of the ABCD_1 in any subsequence X of the window that satisfies that it contains the event $(D,9)$, one event for each event type in ABCD_1, and both the beginning and ending events of the same episode. The only sequences that satify the three constraints are $X_1 = \{(A,1), (B,2), (C,7), (D,9)\}$ and $X_2 = \{(A,5), (B,6), (C,7), (D,9)\}$. Since X_2 corresponds to an occurrence of ABCD_1, the procedure adds the pattern to the annotation set. The procedure then would need to use the episode type G_2 to explore the extensions of the candidate pattern, but there are none. Therefore, the procedure has already fully explored the current temporal window, so it substitutes the annotation set and introduces the next event of the sequence $(E,10)$ into the temporal window, reaching the situation seen in "Temporal window 2". As there are no

patterns in the annotation set, the procedure ends by removing non-frequent candidates. In this step the procedure does not remove non-frequent patterns from the annotation sets.

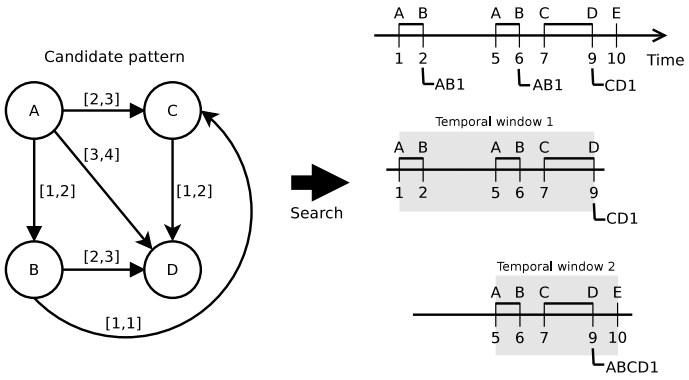


Figure 5.9: Pattern hierarchy frequency calculation with episodes example.

5.3.3 Candidate generation

Figure 5.10 shows the `CANDIDATE_GENERATION()` procedure that builds the candidate patterns whose frequency will be verified in the current iteration of the mining process.

This procedure iterates over the set of supernodes in the $i-1$ level of the set enumeration tree (line 4). All nodes within the same supernode represent temporal associations that share all event types except the last one. Each node N_k^{i-1} (line 5) will be the root of a new supernode in the level i of the tree (line 6). All nodes in the new supernode will share all event types with node N_k^{i-1} and each new node N_h^i will add a new event type provided by a node $N_l^{i-1} > N_k^{i-1}$ within the same supernode (lines 7-8). The procedure then needs to verify whether the temporal association A_h^i that represents the node N_h^i may be frequent by checking if all of the temporal associations A_j^{i-1} it contains are frequent (line 11). In the negative case, then no pattern with those event types can be frequent, and it is not necessary to generate any candidate pattern with this set of event types. This checking operation needs to access other supernodes on the same level of the set enumeration tree. However, the set enumeration tree allows the procedure to quickly find the node where the temporal association and the patterns can be, instead of having to search in the complete set A^{i-1}.

Then, the `CANDIDATE_GENERATION()` procedure generates the temporal constraint networks of the candidate patterns. Every candidate of size i is built from i networks of

```
    procedure CANDIDATE_GENERATION(A^{i-1}, P^{i-1})
1   begin
2     C^i ← ∅
3     h ← 1
4     for SN_j^{i-1} ∈ SN^{i-1} do begin
5       for N_k^{i-1} ∈ SN_j^{i-1} do begin
6         SN_m^i ← ∅
7         for N_l^{i-1} ∈ SN_j^{i-1}, N_k^{i-1} < N_l^{i-1} do begin
8           A_h^i ← A_k^{i-1} ∪ A_l^{i-1}
9           h ← h + 1
10          V ← ∅
11          if all A_j^{i-1} ⊂ A_h^i satisfies A_j^{i-1} ∈ A^{i-1} then
12            for all combination of P_{h_k}^{i-1} ∈ P^{i-1} ∧ D_{h_k}^{i-1} ⊂ A_h^i  do
13              if Q_l^i = ALL-PAIRS-SHORTEST-PATHS(⋈_h P_{h_k}^{i-1})
                is consistent then begin
14                if ∀E_j in Q_l^i, E_j in G_k ∈ G ⇒ G_k ⪯ Q_l^i then
15                  C_c^i ← C_c^i ∪ {Q_l^i}
16                else
17                  C_p^i ← C_p^i ∪ {Q_l^i}
18                V ← V ∪ {Q_l^i}
19              end;
20          if V ≠ ∅ then SN_m^i ← SN_m^i ∪ {N_h^i = (A_h^i, V)}
21        end;
22        SUBTREE(N_k^{i-1}) ← SN_m^i
23        SN^i ← SN_m^i
24    end;
25    return(C^i)
26  end;
```

Figure 5.10: HSTPminer: Candidate generation.

size $i-1$, one frequent pattern for each different temporal association of size $i-1$ found in the previous step. The result of the combination of these patterns is a new pattern of size i, which is then subjected to the ALL-PAIRS-SHORTEST-PATHS() Floyd-Warshall algorithm to propagate the constraints and ensure that it is consistent (lines 12-13). If the pattern is consistent, the candidate is added to one of the two lists of candidate patterns. If for every event type E_j limiting an episode type G_k the pattern also contains the other event type of

G_k, then the candidate belongs to the list C_c^i (line 15), else it is inserted in the list C_p^i (line 17). Candidates in C_p^i are not used during the `FREQUENCY_CALCULATION()` procedure, but they will be necessary in further iterations of the `CANDIDATE_GENERATION()` procedure. When no additional patterns for the same set of event types can be generated, the procedure introduces the node N_h^i that contains both the temporal association and the temporal patterns in the supernode SN_m^i (line 20). Once all possible nodes for SN_m^i have been generated, the procedure links SN_m^i with the root node N_k^{i-1} (line 22).

The `CANDIDATE_GENERATION()` procedure behaves differently when called during the `INITIALISATION()` step. In this case, the candidates generated consist of temporal associations of size two, where no frequent temporal arrangements between the event types are yet known. Therefore, it is not possible to perform the combination step. In addition, every candidate is inserted into the list C_c^i regardless of whether both event types in the association are part of the same episode type or not. This exception allows the process to obtain common temporal arrangements between event types that belong to different episode types, or where one of them does not belong to an episode type. It is also worth mentioning that any candidate pattern of size two where both event types belong to K were already obtained from the seed pattern, and are not generated at this point.

Figure 5.11 shows an example of how the set enumeration tree can aid in the candidate generation step. The objective of this example is to build the third level of the set enumeration tree, consisting of the supernodes labelled as '5', '6' and '7' given the second level of the tree made up from the supernodes labelled as '2', '3' and '4'. The procedure begins by selecting the supernode '2'. Then the procedure chooses the node $N_1^2 = AB$ (line 5). The first node that may be combined with the node AB to generate new candidates is the node AC (line 7). From the temporal associations of both nodes, the new temporal association generated is ABC (line 8). Before generating the candidate patterns, the procedure needs to verify that BC is a frequent temporal association (line 11). To evaluate this, the procedure starts in supernode '1', where it accesses the node B. There, SUBTREE(B) takes the procedure to supernode '3', where it finds node BC, meaning that the temporal association is frequent. Therefore, the procedure combines the frequent patterns found in nodes AB, AC, and BC, which results in candidate patterns $ABC1$ and $ABC2$. The node $N_1^3 = (ABC, \{ABC1, ABC2\})$ is added to supernode '5' (line 20). The procedure then chooses the node AD to be combined with AB, which results in the temporal association ABD. Then a similar procedure is followed to verify that the temporal association BD is frequent, and generate the candidate pattern $ABD1$. As there are no more nodes in supernode '2', supernode '5' is finished and linked to its PARENT,

the node *AB* (line 22). However, there are still nodes within supernode '2' that may act as root of another subtree. In particular, node *AC* may be combined with node *AD* to generate the supernode '6' by following the same process. When supernode '2' has been fully explored, the procedure jumps to supernode '3', where the process is repeated to generate supernode '7'. Finally, since there is only one node in supernode '4', no subtree may be generated from node *CD*.

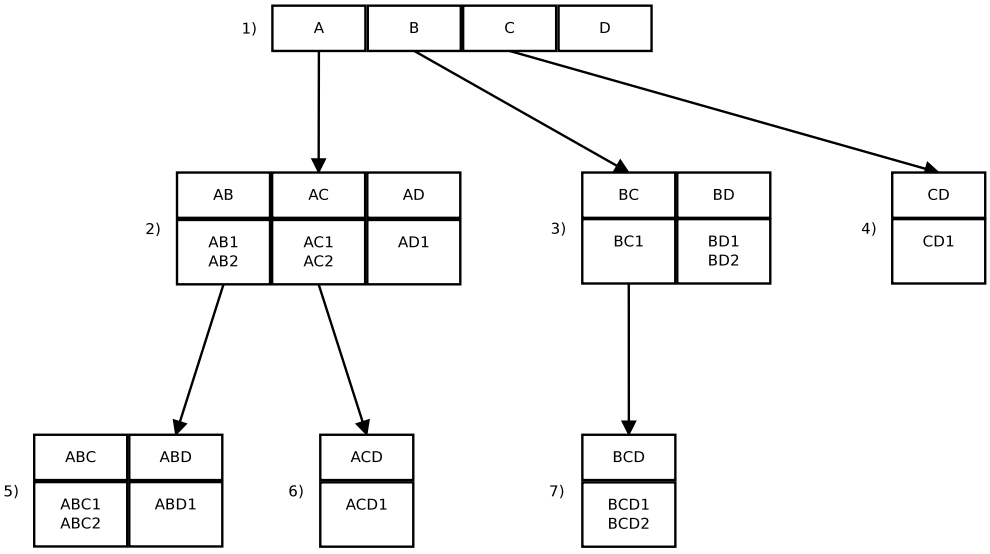


Figure 5.11: Example of candidate generation with set enumeration tree.

5.3.4 Correctness and completeness of HSTPminer

In this section we examine both the correctness and completeness of the `HSTPminer` algorithm.

Lemma 1: (Correctness) Temporal patterns obtained by the `HSTPminer` algorithm are frequent.

Rationale: The `FREQUENCY_CALCULATION()` procedure, along with the auxiliary procedure `VERIFY_PATTERNS()`, search for occurrences of all candidate patterns in all possible temporal windows of every recording in the data collection. During the calculation of P^i, i.e., the set of frequent patterns of size i, for every event e_j introduced into the temporal window, all subsequences X of size i of the temporal window that include the

event e_j are checked against those patterns that contain the event type E_j of the event e_j (VERIFY_PATTERNS(), figure 5.7 line 3). Only those patterns that are extensions of patterns found in the same temporal window need to be verified, and only those extensions such that there are annotations of all of its subpatterns except those without the event type E_j (VERIFY_PATTERNS(), figure 5.7 line 3). These conditions reduce the number of patterns that need to be checked in each window, but do not leave any patterns unchecked if there can be an occurrence of them. If there is an occurrence of a pattern of size i, then necessarily there would be annotations of its subpatterns of size $i-1$ or $i-2$. For each candidate pattern V_r^i and each subsequence X in the window, the frequency of V_r^i is updated if, and only if, the temporal arrangements between the events satisfy every constraint in Q_r^i (VERIFY_PATTERNS(), figure 5.7 line 4). This procedure ensures that every occurrence of every pattern present in the data collection is accounted for, whereas no combination of events in the collection is considered to be an occurrence of any pattern unless all constraints are satisfied. The algorithm discards any candidate pattern Q_l^i having a frequency value $f(Q_l^i)$ less than the user-defined threshold f_{min} (FREQUENCY_CALCULATION(), figure 5.6 line 32), which ensures that resulting patterns are frequent.

Lemma 2 (Completeness) Candidate generation in HSTPminer is exhaustive.

Rationale: The set enumeration tree does not change the candidates generated, it only assigns temporal associations and patterns a fixed position and allows the procedure to easily find them in subsequent calls to the procedure. The Apriori strategy summarised in lines 5-11 (figure 5.10) ensures that all different temporal associations of size i are obtained from the set A^{i-1} of frequent temporal associations of size $i-1$. Additionally, the procedure builds the set C^i of all the candidate patterns of size i from the set P^{i-1} of frequent patterns of size $i-1$, which are different temporal arrangements of the aforementioned temporal associations (figure 5.10 line 12).

P^1 is constructed by removing non-frequent event types from $\mathcal{E}$. Given P^{i-1}, i.e., the set of frequent patterns of size $i-1$, the CANDIDATE_GENERATION() procedure produces all possible combinations $\bowtie_h P_{h_k}^{i-1}$ of i patterns in P^{i-1}, where the set of event types of every pattern $P_{h_k}^{i-1}$ represents a different temporal association in A^{i-1}. Only inconsistent patterns are discarded during the CANDIDATE_GENERATION() procedure (figure 5.10 line 13). The set of temporal patterns is closed under the combination operation, therefore the CANDIDATE_GENERATION() procedure produces temporal patterns from temporal patterns. The consistency of the pattern is checked using the ALL-PAIR-SHORTEST-PATHS() algorithm, which results in a minimal network.

The reasoning above does not take into account the use of a clustering procedure to obtain P^2. Clustering procedures are summarisation procedures that aim to extract from the dataset a reduced, but significant, number of groups. The criterion used to select these groups is the similarity of a set of temporal arrangements that, when gathered together, exceed the frequency threshold f_{min} established by the user. Therefore, `HSTPminer` satisfies only a partial completeness: the `CANDIDATE_GENERATION()` procedure is exhaustive but HSTPminer is not exhaustive because of the clustering procedure.

5.3.5 Complexity analysis of HSTPminer

In the following analysis of the computational complexity of these algorithms we assume that $|\mathcal{E}|$ contains both limiting event types of each episode type in $\mathcal{G}$. Then, the complexity of each procedure in an iteration i can be evaluated as follows:

- In the *temporal association generation* step, for a temporal association A^i_j to be frequent, all of the i temporal associations $A^{i-1}_k \subset A^i_j$ must be frequent. The maximum number of frequent temporal associations of size $i-1$ is $|A^{i-1}| = \binom{|\mathcal{E}|}{i-1}$, and each temporal association of size $i-1$ can be added $|\mathcal{E}|-i$ different event types to create a temporal association of size i. Finding a temporal association of size $i-1$ in the set enumeration tree requires accessing $i-1$ supernodes, starting from the root supernode of the tree and goind down one level in each step. Therefore, verifying that all temporal associations $|A^{i-1}_k|$ are frequent has a complexity of $O(i^2)$. The combination step is no longer necessary, because all temporal associations within the same supernode share all event types except the last one. The complexity of generating all the candidate temporal associations is $O(|A^{i-1}||\mathcal{E}|i^2)$.

- The *candidate generation* step has not changed with respect to `ASTPminer`. Each of the $i(i-1)/2$ constraints of a temporal pattern $P^i_{j_k}$ is obtained through the combination of those patterns $P^{i-1}_{h_k}$ where $D^{i-1}_h \subset A^i_j$. The combination has a complexity of $O(i^2)$, and the maximum number of possible combinations of patterns of size i is $\prod_h |P^{i-1}_h|$. In terms of the initial parameters, the maximum number of temporal patterns of size two between any two event types corresponds to the situation where every temporal arrangement presents a frequency value greater than the frequency threshold f_{min}, but their frequency values are so different that the clustering procedure produces one pattern for each temporal arrangement. In this situation, the clustering procedure produces 2ω

patterns for every temporal association of size two. Under this assumption, to build a pattern of size i it is sufficient to establish the $i-1$ constraints between one event type and the rest of the event types, as the remaining constraints can be completely specified during the `ALL-PAIRS-SHORTEST-PATHS()` step. Additionally, the number of all possible combinations of frequent patterns to explore to build all candidate patterns for any temporal association of size i is $\omega^{i(i-1)}$. Every combination is subjected to a consistency checking process by the `ALL-PAIRS-SHORTEST-PATHS()` algorithm, which has a complexity of $O(i^3)$. Therefore, the overall complexity of this step is $O(|\mathrm{A}^i| i^3 \omega^{i(i-1)})$.

– In the *frequency calculation* step, the worst case scenario corresponds to a situation where every time unit may present an occurrence of all $|\mathcal{E}|$ event types with a total of $n = |\mathcal{E}| d_{\mathcal{S}}$ events in $\mathcal{S}$ (where n is the length of $\mathcal{S}$ and $d_{\mathcal{S}}$ is its duration, the sum of the durations of each sequence in the collection). Every time the window is updated, $|\mathcal{E}|$ events enter the window and $|\mathcal{E}|$ leave. Under this assumption, for every pattern of size i, up to $i\omega^{i-1}$ new occurrences may be found in every window update for every pattern. Checking whether an occurrence fulfils all of the constraints in a pattern has a complexity of $O(i^2)$. Assuming that in the previous iteration an occurrence of each candidate was found in the window, the procedure extends every pattern found in the annotation set attached to each event with all $|\mathcal{E}|$ event types, and then for each extension the procedure needs to verify that there are $i-1$ of its subpatterns annotated. Considering that the maximum number of candidate patterns in the i iteration is $|\mathrm{A}^i|\omega^{i-1}$, the number of patterns found in the annotation set is $|\mathrm{A}^{i-1}|\omega^{i-2}$. Therefore, for each annotation there are ω^{i-1} extensions of each frequent pattern with one event type, and the overall complexity for this step is $O(n|\mathcal{E}|\omega^{3i-4} i^4 |\mathrm{A}^{i-1}|)$. This complexity is higher than the complexity of the `ASTPminer` algorithm. Since in the worst case scenario there are occurrences of all candidate patterns in every temporal window, the processing of the annotation set by `HSTPminer` turns out that all candidates need to be verified in each temporal window, introducing an overhead into the `FREQUENCY_CALCULATION()` procedure without any benefit. However, as will be seen in the validation process in section 5.4, in the average case the cost of processing the annotation sets would be less than the cost of verifying all subsequences of the temporal window for all candidate patterns.

Given the iterative nature of the mining process, and considering that the maximum pattern size is the number of different event types $|\mathcal{E}|$, the overall complexity of the algorithm is dominated by the candidate generation step, which results in a complexity of $O(|\mathcal{E}||\mathrm{A}^i|i^3\omega^{i(i-1)})$.

The use of seed patterns is expected to have an impact in reducing the number of patterns $|\mathrm{P}^i|$ generated in each iteration, and in the overall number of events n of the collection $\mathcal{S}$. Those segments of the collection where no occurrences are found are discarded for further iterations, which reduces the value of n. These changes contribute to improve the total efficiency of the algorithm. As seen in section 4.3, it is possible that $|\mathrm{P}^2|$ is greater when seed patterns are used, as the shape of the temporal distance distributions may vary and their clustering may result in more numerous yet more specific patterns. However, if that were the case, the number of possible combinations in later iterations would be expected to be lower, as more specific patterns would have fewer possible consistent combinations.

5.4 HSTPminer validation

In this section we compare the performance of the `HSTPminer` and the `ASTPminer` algorithms in both the SAHS database and the synthetic databases introduced in section 4.5.

5.4.1 SAHS database

As in the `ASTPminer` algorithm, a seed pattern is provided to the `HSTPminer` algorithm. This pattern is shown in figure 5.12, and it describes an approximate representation of a central apnea pattern, where the beginning of an airflow limitation episode occurs roughly at the same time as the beginning of the abdominal and thoracic movement limitation episodes, the ending of an airflow limitation episode also occurs roughly at the same time as the ending of the thoracic and abdominal movement episodes, and the blood oxygen saturation falls after the beginning of the airflow limitation. The formal definition of this seed pattern is $K =< D, \mathcal{L} >$, where $D = \{E_1, E_2, E_3, E_4, E_5, E_6, E_7, E_8\}$ and the user-defined constraints are $\mathcal{L} = \{L_{12} = [1, \omega], L_{34} = [1, \omega], L_{56} = [1, \omega], L_{78} = [1, \omega], L_{13} = [-5, 5], L_{15} = [-5, 5], L_{17} = [0, \omega], L_{24} = [-5, 5], L_{26} = [-5, 5], L_{ij} = [-\omega, \omega]$ otherwise$\}$.

Figure 5.13 shows the time required by the `ASTPminer` and `HSTPminer` algorithms, with and without using the seed pattern shown in figure 5.12, using different values for the window size and a frequency threshold of 30 occurrences. These results show that, for both algorithms, providing a seed pattern helps to reduce the search scope and improve the perfor-

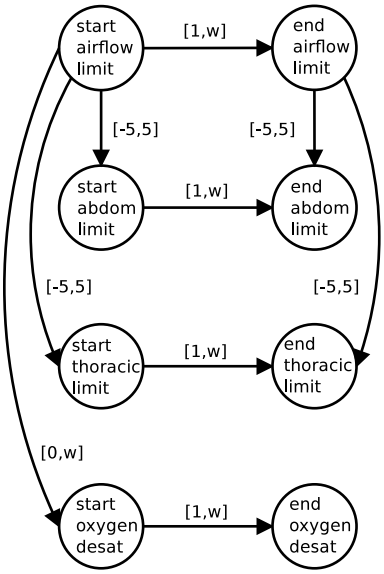


Figure 5.12: Seed pattern used in the experiments.

mance. However, the inclusion of the pattern hierarchy seems to have a detrimental effect on the performance. This would imply that the cost of analysing the SAHS database is too small when compared to the overhead introduced by the pattern hierarchy. It is worth mentioning that both algorithms skip the `FREQUENCY_CALCULATION()` procedure for odd values of i. In these cases, since the database consists only of episode types, for every candidate there will be at least one episode type where one of the limiting event types will be in the candidate but the other will not. Therefore, in odd numbered iterations $C_c^i = \varnothing$, so the algorithm skips the `FREQUENCY_CALCULATION()` procedure.

The seed pattern allows the mining process to ignore some temporal arrangements between events that may be detected due to the overlapping nature of the apnea pattern. That is, it is common for SAHS patients that several apnea patterns occur in rapid sucession. As the window width grows, those occurrences may be found in the same window, increasing the number of subsequences that need to be verified for each candidate pattern. Introducing a seed pattern allows to ignore those subsequences where the temporal arrangements between the events included are of no interest, reducing the cost of the analysis. Moreover, the adequate use of episodes allows to further reduce the number of subsequences of each temporal

window that need to be analysed, as a beginning event of an episode forces its corresponding ending event to be present in the same subsequence, making it possible to ignore those subsequences where this is not satisfied. The pattern hierarchy allows to reduce the number of candidate patterns that the `FREQUENCY_CALCULATION()` procedure needs to verify in each remaining subsequence of the temporal window to only those candidates that extend a frequent pattern found either in the previous iteration, or the iteration before the previous one, as explained in section 5.3.2. Nevertheless, navigating through this hierarchy seems to introduce a cost that the reduction of the number of patterns verified cannot compensate in this database.

The `HSTPminer` algorithm focuses on detecting which candidate patterns need not be tested in a temporal window. When compared to the `ASTPminer` algorithm, there is no difference in neither the number nor their frequency, nor in the contraints of the patterns themselves. Therefore, the number of possible, candidate and frequent patterns in each iteration remains the same as in table 4.4.

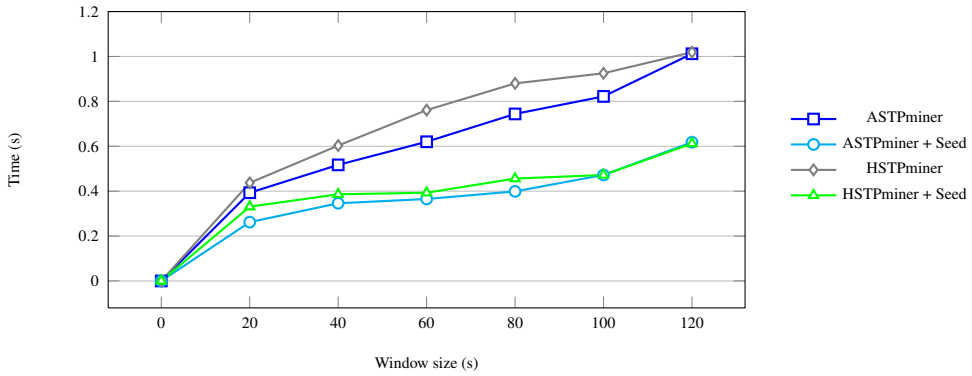


Figure 5.13: Time required by the `ASTPminer` algorithm with and without seed pattern, and the `HSTPminer` algorithm with and without seed pattern.

5.4.2 Synthetic databases

The first set of experiments involves databases where both event types and episode types are present in the collection. With these datasets, we analyse the impact of pattern size, event density, as well as the ratio between the number of episode types and event types (SDB1 to

SDB6 in table 4.6). The second set of experiments compares both algorithms in databases where no episode type is present (SDB7 to SDB10).

Figures 5.14, 5.15, 5.16 and 5.17 show similar databases, where the number of events and their density in the collection are gradually increased. In the smaller databases `ASTPminer` obtains better results, but in the bigger databases both algorithms seem to behave similarly. In these four cases, candidate generation times are negligible.

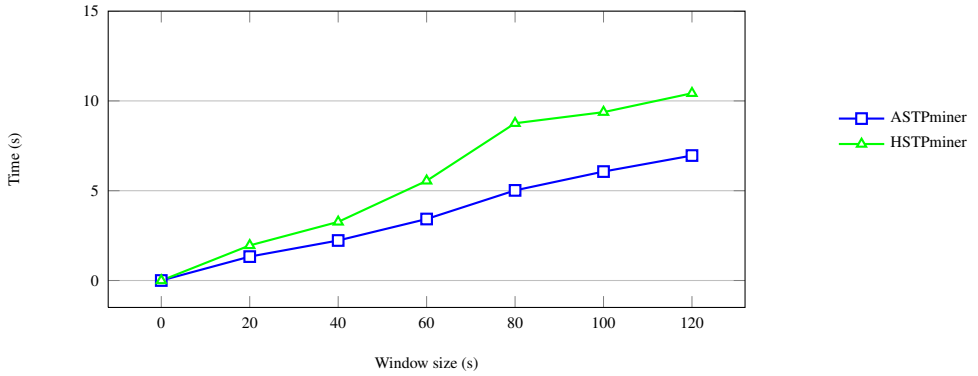


Figure 5.14: SDB1: max_P=9, N=136000, Δ=0.07, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4

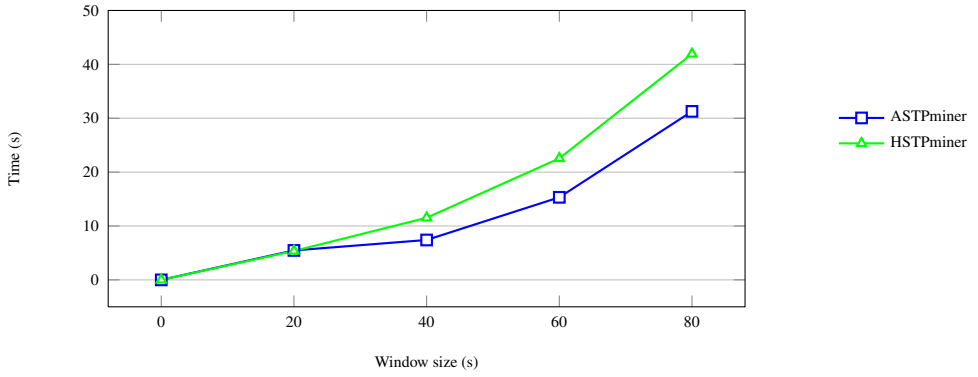


Figure 5.15: SDB2: max_P=11 , N=275000, Δ=0.15, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4

In the databases corresponding to figures 5.18 and 5.19 larger patterns are included and the number of episodes is reduced. In both cases, `HSTPminer` proves to be more efficient, reducing the time required by the mining process considerably when compared to the `ASTPminer`

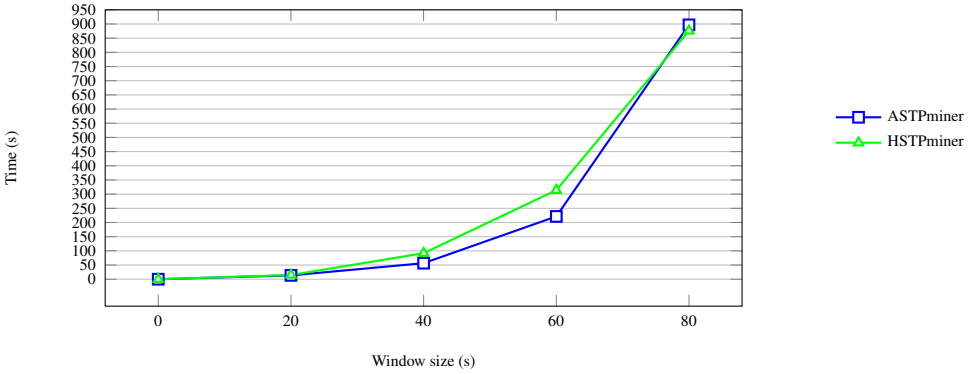


Figure 5.16: SDB3: max_P=9 , N=470000, Δ=0.25, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4

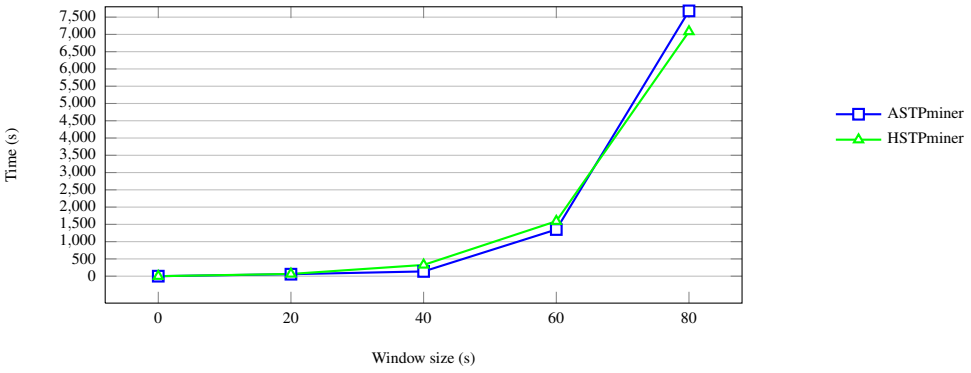


Figure 5.17: SDB4: max_P=8, N=705000, Δ=0.39, $|\mathcal{E}|$=6, $|\mathcal{G}|$=4

algorithm. As pattern size grows, the number of temporal patterns involved in every iteration of the mining process also grows. While `ASTPminer` needs to test whether each of the candidate patterns can be found in every temporal window, the `HSTPminer` algorithm is able to reduce the number of patterns verified in each window, which results in an improvement of the performance.

The databases SDB7 to SDB10 (Table 4.6) used in the next set of experiments do not contain any episode type and therefore consist only of event types. The results of the experiments are shown in figures 5.20, 5.21, 5.22 and 5.23. In all of them, the `HSTPminer` algorithm requires consistently less time to produce the results. The experiments with these databases

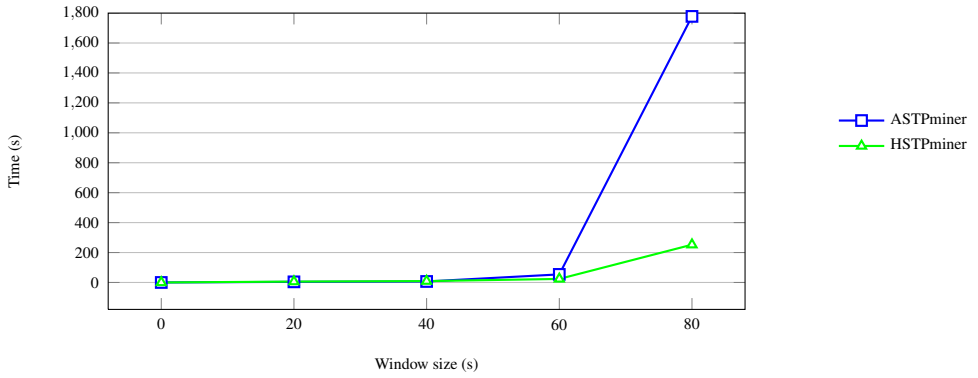


Figure 5.18: SDB5: max_P=14, N=70000, Δ=0.22 , $|\mathcal{E}|$=12, $|\mathcal{G}|$=1

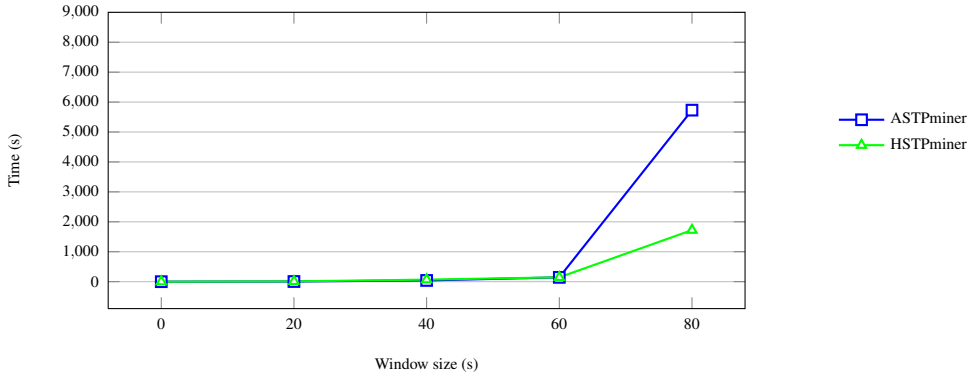


Figure 5.19: SDB6: max_P=19, N=76000, Δ=0.25, $|\mathcal{E}|$=13, $|\mathcal{G}|$=3

allow us compare the efficiency of the algorithms for varying degrees of density of events in the collection, while maintaining a similar number of events in the same temporal window. The `HSTPminer` algorithm outperforms the `ASTPminer` algorithm in all the databases, yet the improvement provided by the algorithm decays as the number of events per time unit grows. As the event density increases, if the number of event types remains constant, the frequency of all possible temporal arrangements between any pair of event types becomes more and more similar. The `CLUSTERING()` procedure will tend to produce less patterns, but will also include more temporal arrangements in each pattern. Therefore the overall number of patterns is reduced, and the impact of discarding patterns becomes less important, perhaps

being even counter-productive due to the cost associated to analysing and maintaining the annotations.

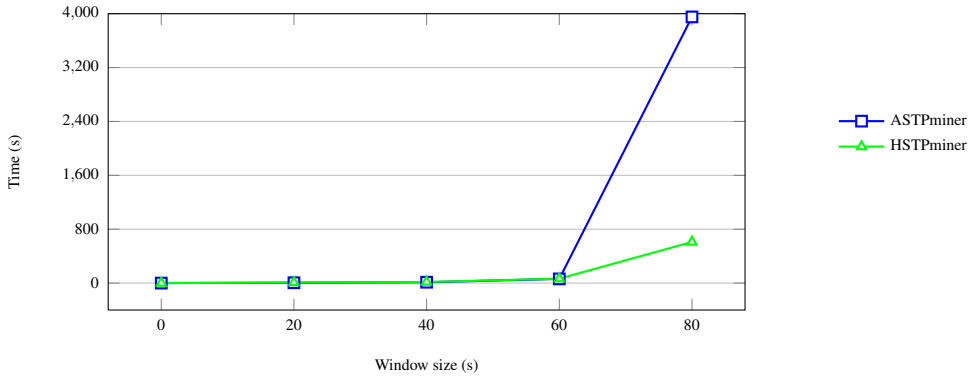


Figure 5.20: SDB7: max_P=13, N=65000, Δ=0.2, $|\mathcal{E}|$=13, $|\mathcal{G}|$=0

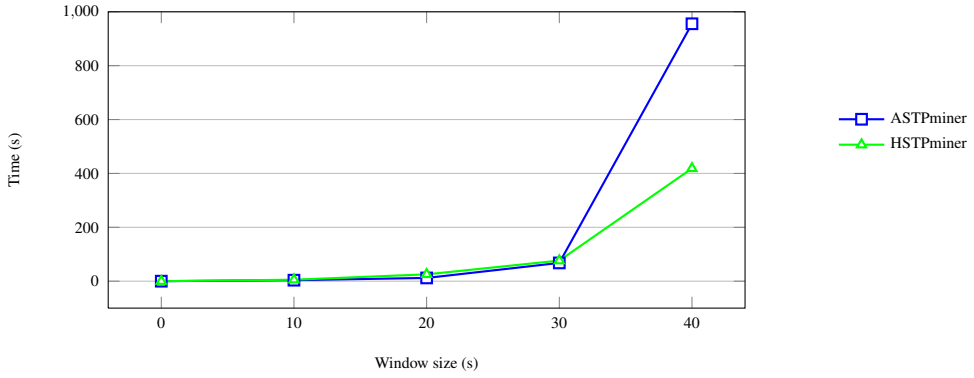


Figure 5.21: SDB8: max_P=10, N=50000, Δ=0.4, $|\mathcal{E}|$=10, $|\mathcal{G}|$=0

The experiments described in this section show that the `HSTPminer` algorithm outperforms the `ASTPminer` algorithm in databases with a low ratio of episode types versus event types, as well as in databases where long patterns are present. This is usually the case of activity logs data sets, such as the ones result of the monitoring of Information Systems. The results also show that as the event density of the database grows the improvement of the

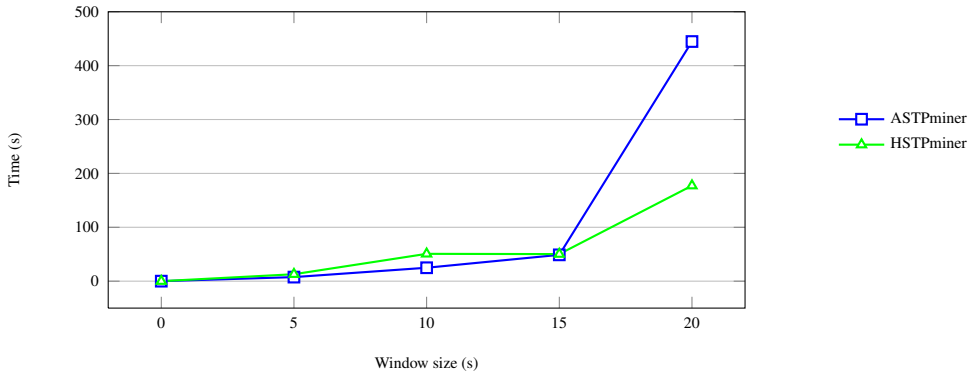


Figure 5.22: SDB9: max_P=13, N=95212, Δ=0.8, $|\mathcal{E}|$=13, $|\mathcal{G}|$=0

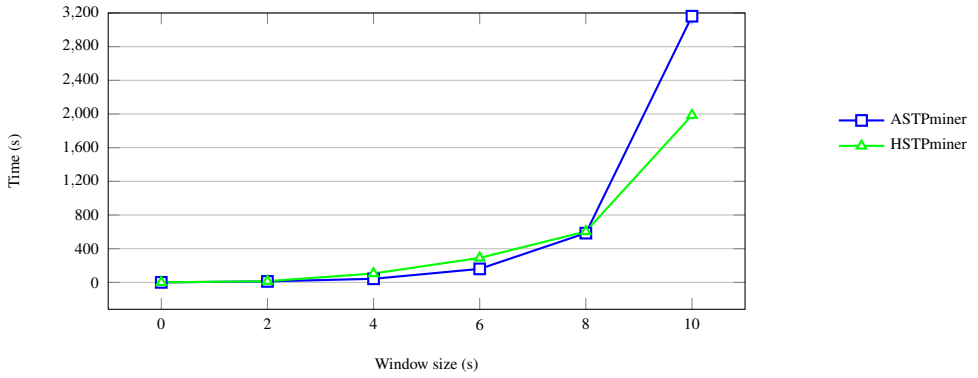


Figure 5.23: SDB10: max_P=12, N=123402, Δ=2.25, $|\mathcal{E}|$=12, $|\mathcal{G}|$=0

`HSTPminer` over the `ASTPminer` algorithm is reduced. Finally, in databases where the ratio of episode types vs event types is high both algorithms behave similarly.

CHAPTER 6

MAIN CONTRIBUTIONS AND CONCLUSIONS

The main purpose of this thesis work has been to propose a formal solution for mining temporal patterns from temporal databases. The Simple Temporal Problem (STP) has been chosen as the formal framework for representing the results of the mining process, as it has been widely and successfully applied on temporal reasoning, scheduling or planning tasks [Dechter, 2003]. The STP belongs to the set of Constraint Satisfaction Problems (CSP), a formalism that represents in a declarative manner a number of constraints that must be satisfied in order to obtain a solution to a given problem. One of the advantages of CSP is the possibility of modeling a problem graphically, as a constraint network that can be represented as a constraint graph, enabling the treatment of tasks like inference, consistency checking, or scenario finding by using graph processing techniques. The STP allows us to represent temporal information as a network of metric temporal constraints between a set of temporal variables, each one representing a time instant. A particular STP is usually set up from a description of expert knowledge, and obtaining an STP from a temporal dataset, after a data mining process, represents an innovative and relevant challenge for the knowledge discovery paradigm.

The present thesis proposes a number of techniques, grouped under the `ASTPminer` and `HSTPminer` names, for discovering frequent temporal patterns, each one represented as an STP, from a collection of event sequences. From the very beginning, choosing the STP as a representation framework entails an important step forward from previous works in the literature of temporal data mining from event sequences, since the STP can represent both quantitative and qualitative constraints, subsuming other qualitative constraint formalisms, like the

Convex Point Algebra [Van Beek and Cohen, 1990] or the Convex Simple Interval Algebra [Gerevini et al., 1996]. Thus, the `ASTPminer` and `HSTPminer` algorithms provide a more expressive result than those based on frequent orders between a set of events in a collection. Furthermore, the STP provides a good balance between expressiveness and complexity, since the usual information processing tasks supported by the STP are computed in polynomial time, whereas most of the temporal formalisms entail NP-hard tasks.

Temporal data mining in both `ASTPminer` and `HSTPminer` is based on the application of similarity criteria to different temporal arrangements between the same event types. In this sense, an important limitation in a previous work [Dousson and Duong, 1999] is overcome, where at most one temporal pattern is mined for a fixed set of event types. A temporal clustering involving temporal arrangements between events appears as an extremely complex task. However, both of the techniques mentioned above simplify this operation by projecting the similarity criteria in each pair of event types in the pattern. A clustering procedure over temporal distance distributions of occurrences of pairs of event types produces a set of intervals. Each one of these intervals represents a distinguishable temporal arrangement between the pair of event types, and thus originates a pattern of size two with this pair of event types. A consistency checking of the resulting candidate patterns enforces the assembling of consistent patterns of any size. Because clustering is only conducted when searching for patterns of size two, this process does not compromise the efficiency of the global procedure.

A relevant contribution of both techniques is that they allow users to participate in the mining process by introducing previous domain knowledge. The patterns provided by the user are also represented by means of the STP formalism. User knowledge is provided as a seed pattern involving a number of events and episodes of interest and some temporal constraints between them. This initial knowledge then focuses the search process on those patterns that extend the seed, either by incorporating new event types or by refining the constraints in the seed pattern. A seed pattern provides a strong instrument to prune the search space in two ways. First, it allows subsequences where no occurrence of the seed pattern is found to be ignored, which therefore reduces the number of events to be considered. Second, the constraints specified by the user limit the patterns found to those consistent with the information provided, which reduces the number of patterns while increasing their interest for the end user. Both aspects contribute to the improvement of algorithm efficacy and efficiency.

`ASTPminer` provides the user with a temporal data mining tool to discover frequent temporal patterns by exploiting the Apriori strategy. `ASTPminer` has been designed to take advantage of the possible occurrence of frequent episodes in a sequence. In fact, a suitable

treatment of episodes in the mining process allows us to skip some of the iterations in the frequency calculation stage showing an important reduction in the overall computational cost. Correctness of ASTPminer has been proved in its exhaustive search on a given temporal dataset, and partial completeness has been pointed out after the summarizing operation of temporal clustering.

To validate ASTPminer, several experiments were conducted over a medical database of time-stamped sequences, obtained from polysomnography tests in patients with Sleep Apnea-Hypopnea Syndrome. We have selected this clinical domain since the medical community has described a well-known phenomenon from the repetitive observation of a temporal pattern in a data set. The experiments aimed to test the ability of ASTPminer to induce similar results from the same data, producing successful results both from the point of view of reproducing the pattern discovery results of medical experts, and from the point of view of computational efficiency. Nevertheless, the SAHS database has very particular properties, so further tests have been necessary on a different set of databases of time-stamped sequences. Thus, a temporal dataset generator was designed and implemented, with the aim of completely characterizing the behaviour of temporal data mining algorithms in a wide variety of simulated situations, where the quantitative information about the temporal arrangement of events is relevant.

HSTPminer is an evolution of the ASTPminer algorithm, focused on improving the performance of both the frequency calculation and the candidate generation stages. In the HSTPminer algorithm every sequence in any iteration is annotated with those patterns that were found in every temporal window, with the aim of reducing the number of candidate patterns that need to be checked in the next iteration of the frequency calculation procedure. Thus the mining process focus on those patterns that extend previous frequent patterns found in the same temporal window. HSTPminer builds and maintains a pattern hierarchy throughout the mining process, by means of a set enumeration tree. This pattern hierarchy associates every pattern with each one of its extensions by means of the event type that the extended pattern adds to the former. This structure improves the candidate generation process by allowing it to easily access patterns found in previous iterations that are needed for generating new patterns.

Tests show that HSTPminer outperforms ASTPminer under certain conditions. In databases where the number of episode types is small when compared to the number of event types HSTPminer shows a better performance, yet this improvement decays as the number of events per time unit in the data collection increases. On the other hand, in those databases where large patterns can be found, HSTPminer is able to improve the efficiency of the pro-

cess by reducing the number of patterns that need to be verified in each temporal window. However, in the presence of smaller patterns and a similar number of episode types and event types, the overhead introduced by the pattern hierarchy seems to bring the performance of both algorithms to a similar level. In summary, `ASTPminer` is shown to be suitable for mining temporal databases with a sparse occurrence of events and episodes, and relatively small frequent temporal patterns. This is usually the case of annotated physiological recordings, as it has been proven with a typical SAHS database. `HSTPminer` seems appropriate for mining temporal databases with a more dense occurrence of time-stamped events, and larger frequent temporal patterns. This is usually the case of activity log data sets, commonly resulting from the monitoring of Information Systems.

There is a number of future research directions for the present work that can be summarized as follows:

1. Other search strategies should be explored, beyond the Apriori paradigm. The main aim should be to reduce the cost of the frequency calculation procedure. In the short term, this could be done by omitting the generation of candidates in a combinatorial fashion, by computing just those plausible candidates attached to each temporal window. The main difficulty lies in designing a procedure not too memory consuming for reducing the number of possible occurrences to check in each temporal window.

2. Including the negation in the mining procedures appears as a challenging task. A first form of negation can be expressed as the systematic absence of some event type during the occurrence of a frequent temporal pattern. This absence can be modelled by including negated event types in the assembly of temporal associations. These negated event types cannot appear in the resulting STP, so the final frequent temporal pattern should be read as a frequent temporal association enhanced by a temporal arrangement between the positive event types.

3. Both `ASTPminer` and `HSTPminer` provide the user with a temporal distance distribution for each pair of event types of a frequent temporal pattern. Applying clustering proves to be a simple and effective tool for identifying those similar patterns in a dataset, with good results. However, a temporal distance distribution also provides a valuable support for the application of a wide range of tools from statistics, in order to achieve a more in-depth analysis of those frequent temporal patterns. This sort of analysis de-

pends largely on the nature of the domain and the processes where the datasets come from.

4. Building a temporal distance distribution for each pair of event types makes it possible to obtain, from a mining process, a family of constraint solving schemes based on a semiring structure [Bistarelli et al., 1997], extending the STP framework in the direction of providing more flexible semantics to the very notion of constraint: those of preference, probability or possibility, amongst others. These new constraint solving schemes specify, in addition to each temporal constraint $L_{ij} = [a_{ij}, b_{ij}]$, a function $f_{ij} : [a_{ij}, b_{ij}] \rightarrow A$, where A is a set of admissible values, and the tuple $\langle A, +, \times, 0, 1 \rangle$ is a semiring such that: A is a set and $0, 1 \in A$; $+$ is commutative, associative and 0 is its unit element; $\times$ is associative, distributes over $+$, 1 is its unit element, and 0 is its absorbing element. In general, solving these new constraint solving schemes is NP-complete, but three practical assumptions make them solvable in polynomial time: (1) the functions f_{ij} are semiconvex; (2) the $\times$ operator is idempotent; (3) the values of the semiring are totally ordered. `ASTPminer` and `HSTPminer` can be applied to obtain frequent temporal patterns corresponding to three such new constraint solving schemes:

   – *Simple Temporal Problem with Preferences* [Khatib et al., 2001]. Planning and scheduling tasks involve not only quantitative temporal constraints, but also soft temporal preferences among different possible choices, as solutions of the corresponding STP. In a Simple Temporal Problem with Preferences, a temporal constraint is represented as $L_{ij} =< [a_{ij}, b_{ij}], \phi_{ij} >$, where $\phi_{ij} : [a_{ij}, b_{ij}] \rightarrow [0, 1]$ represents a preference function. The construction of a temporal distance distribution from a temporal dataset for each pair of event types allows us to model a corresponding preference function, assuming that a higher frequency in a specific duration between a pair of event types agrees with a higher preference exhibited by the decision maker in the temporal dataset.

   – *Simple Temporal Problem with Probabilities* [Morris et al., 2001]. The interval-based representation for the temporal uncertainty provided by the STP framework can be enhanced by a probabilistic representation, assuming that a temporal constraint depicts a duration between events as a continuous random variable. In a Simple Temporal Problem with Probabilities, a temporal constraint is represented as $L_{ij} =< [a_{ij}, b_{ij}], p_{ij} >$, where $p_{ij} : [a_{ij}, b_{ij}] \rightarrow [0, 1]$ represents a probability density function. If the dataset is very large, the temporal distance distribution

for each pair of event types of a frequent temporal pattern can be considered as a good approximation to the underlying probability density function.

- *Fuzzy Temporal Constraint Networks* [Barro et al., 1994, Marín et al., 1994]. A possibilistic representation of uncertainty is based on an ordering structure rather than the additive one of probability, explicitly handling incomplete knowledge, and naturally supporting the combination of subjective, linguistic-like information. In a Fuzzy Temporal Constraint Network, a temporal constraint is represented as $L_{ij} =< [a_{ij}, b_{ij}], \pi_{ij} >$, where $\pi_{ij} : [a_{ij}, b_{ij}] \rightarrow [0, 1]$ represents a possibility distribution. If the dataset is very large, a probability-possibility transformation can be applied in order to obtain a possibility distribution from a temporal distance distribution [Dubois et al., 2004]. Fuzzy Temporal Constraint Networks suitably support the processing of vague or uncertain temporal information, bringing the temporal data mining closer to the natural language.

5. The design and implementation of a web-based visual tool will make it easier to access and use `ASTPminer` and `HSTPminer`. This tool will also allow the scientific community to share benchmark datasets, and it will facilitate the dissemination of temporal data mining beyond the academic context.

Bibliography

[Adlassnig et al., 2006] Adlassnig, K.-P., Combi, C., Das, A. K., Keravnou, E. T., and Pozzi, G. (2006). Temporal representation and reasoning in medicine: Research directions and challenges. *Artificial Intelligence in Medicine*, 38(2):101 – 113.

[Agarwal et al., 2000] Agarwal, R. C., Aggarwal, C. C., and Prasad, V. V. V. (2000). Depth first generation of long patterns. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, pages 108–118, New York, NY, USA. ACM.

[Agarwal et al., 2001] Agarwal, R. C., Aggarwal, C. C., and Prasad, V. V. V. (2001). A tree projection algorithm for generation of frequent item sets. *Journal of Parallel and Distributed Computing*, 61(3):350 – 371.

[Agrawal et al., 1995] Agrawal, R., Lin, K.-I., Sawhney, H. S., and Shim, K. (1995). Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the 21th International Conference on Very Large Data Bases*, VLDB'95, pages 490–501, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Agrawal and Srikant, 1994] Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In Bocca, J. B., Jarke, M., and Zaniolo, C., editors, *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB'94, pages 487–499. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Agrawal and Srikant, 1995] Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In Yu, P. S. and Chen, A. L. P., editors, *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, pages 3–14. IEEE Computer Society, Washington, DC, USA.

[Ale and Rossi, 2000] Ale, J. and Rossi, G. (2000). An approach to discovering temporal association rules. In *Proceedings of the ACM Symposium on Applied Computing*, pages 294–300.

[Allen, 1983] Allen, J. (1983). Maintaining knowledge about temporal intervals. *Communications of the ACM Transaction on Information Systems*, 26(11):832–843.

[Álvarez et al., 2011] Álvarez, M. R., Félix, P., and Cariñena, P. (2011). Mining temporal constraint networks by seed knowledge extension. In Peleg, M., Lavrac, N., and Combi, C., editors, *Artificial Intelligence in Medicine*, volume 6747 of *Lecture Notes in Computer Science*, pages 250–254. Springer-Verlag, Berlin/Heidelberg.

[Álvarez et al., 2013] Álvarez, M. R., Félix, P., and Cariñena, P. (2013). Discovering metric temporal constraint networks on temporal databases. *Artificial Intelligence in Medicine*. In press.

[Álvarez et al., 2010] Álvarez, M. R., Félix, P., Cariñena, P., and Otero, A. (2010). A data mining algorithm for inducing temporal constraint networks. In Hüllermeier, E., Kruse, R., and Hoffmann, F., editors, *Computational Intelligence for Knowledge-Based Systems Design*, volume 6178 of *Lecture Notes in Computer Science*, pages 300–309. Springer-Verlag, Berlin/Heidelberg.

[American Academy of Sleep Medicine Task Force, 1999] American Academy of Sleep Medicine Task Force (1999). Sleep-related breathing disorders in adults: recommendations for syndrome definition and measurement techniques in clinical research. *Sleep*, 22:667–689.

[Anselma et al., 2006] Anselma, L., Terenziani, P., Montani, S., and Bottrighi, A. (2006). Towards a comprehensive treatment of repetitions, periodicity and temporal constraints in clinical guidelines. *Artificial Intelligence in Medicine*, 38(2):171–195.

[Antunes and Oliveira, 2005] Antunes, C. and Oliveira, A. (2005). *Constraint Relaxations for Discovering Unknown Sequential Patterns*, pages 11–32. Springer.

[Augusto, 2001] Augusto, J. C. (2001). The logical approach to temporal reasoning. *Artificial Intelligence Review*, 16(4):301–333.

[Badaloni and Berati, 1996] Badaloni, S. and Berati, M. (1996). Hybrid temporal reasoning for planning and scheduling. In *Proceedings of the Third International Workshop on Temporal Representation and Reasoning*, TIME '96, pages 39–44.

[Badaloni and Giacomin, 2006] Badaloni, S. and Giacomin, M. (2006). The algebra IA^{fuz}: a framework for qualitative fuzzy temporal reasoning. *Artificial Intelligence*, 170(10):872–908.

[Barber, 1993] Barber, F. A. (1993). A metric time-point and duration-based temporal model. *SIGART Bulletin*, 4(3):30–49.

[Barro et al., 1994] Barro, S., Marín, R., Mira, J., and Patón, A. (1994). A model and a language for the fuzzy representation and handling of time. *Fuzzy Sets and Systems*, 61:153–175.

[Bayardo, 1998] Bayardo, Jr., R. J. (1998). Efficiently mining long patterns from databases. In *Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, SIGMOD '98, pages 85–93, New York, NY, USA. ACM.

[Bellazzi et al., 2011] Bellazzi, R., Ferrazzi, F., and Sacchi, L. (2011). Predictive data mining in clinical medicine: a focus on selected methods and applications. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 1(5):416–430.

[Bellazzi et al., 2005] Bellazzi, R., Larizza, C., Magni, P., and Bellazzi, R. (2005). Temporal data mining for the quality assessment of hemodialysis services. *Artificial Intelligence in Medicine*, 34(1):25–39.

[Bettini et al., 1998a] Bettini, C., Sean Wang, X., Jajodia, S., and Lin, J.-L. (1998a). Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):222–237.

[Bettini et al., 1998b] Bettini, C., Wang, X., and Jajodia, S. (1998b). A general framework for time granularity and its application to temporal reasoning. *Annals of Mathematics and Artificial Intelligence*, 10(1-2):29–58.

[Bistarelli et al., 1997] Bistarelli, S., Montanari, U., and Rossi, F. (1997). Semiring-based constraint satisfaction and optimization. *Journal of the ACM*, 44(2):201–236.

[Bonchi and Lucchese, 2007] Bonchi, F. and Lucchese, C. (2007). Extending the state-of-the-art of constraint-based pattern discovery. *Data & Knowledge Engineering*, 60(2):377 – 399.

[Campos et al., 2010] Campos, M., Juárez, J., Palma, J., and Marín, R. (2010). Using temporal constraints for temporal abstraction. *Journal of Intelligent Information Systems*, 34(1):57–92.

[Casas-Garriga, 2005] Casas-Garriga, G. (2005). Summarizing sequential data with closed partial orders. In *SIAM International Conference on Data Mining (SDM-05)*, pages 380–391.

[Chen et al., 2011] Chen, Y., Wu, S., and Wang, Y. (2011). Discovering multi-label temporal patterns in sequences databases. *Information Sciences*, 181:398–418.

[Chen et al., 2003] Chen, Y.-L., Chiang, M.-C., and Ko, M.-T. (2003). Discovering time-interval sequential patterns in sequence databases. *Expert Systems with Applications*, 25(3):343–354.

[Chittaro and Montanari, 2000] Chittaro, L. and Montanari, A. (2000). Temporal representation and reasoning in artificial intelligence: Issues and approaches. *Annals of Mathematics and Artificial Intelligence*, 28(1-4):47–106.

[Coenen et al., 2004] Coenen, F., Goulbourne, G., and Leng, P. (2004). Tree structures for mining association rules. *Data Mining and Knowledge Discovery*, 8(1):25–51.

[Combi et al., 2009] Combi, C., Gozzi, M., Oliboni, B., Juárez, J., and Marín, R. (2009). Temporal similarity measures for querying clinical workflows. *Artificial Intelligence in Medicine*, 46:37–54.

[Combi and Shahar, 1997] Combi, C. and Shahar, Y. (1997). Temporal reasoning and temporal data maintenance in medicine: Issues and challenges. *Computers in Biology and Medicine*, 27(5):353 – 368.

[Concaro et al., 2009] Concaro, S., Sacchi, L., Cerra, C., Fratino, P., and Bellazzi, R. (2009). Mining healthcare data with temporal association rules: improvements and assessment for a practical use. In Combi, C., Shahar, Y., and Abu-Hanna, A., editors, *10th Conference on Artificial Intelligence in Medicine, AIME 2009*, volume 5651 of *Lecture Notes in Artificial Intelligence*, pages 16–25. Springer-Verlag, Berlin/Heidelberg.

[Dechter, 2003] Dechter, R. (2003). *Constraint Processing*. Morgan-Kaufmann, San Francisco, CA, USA.

[Dechter et al., 1991] Dechter, R., Meiri, I., and Pearl, J. (1991). Temporal constraint networks. *Artificial Intelligence*, 49:61–95.

[Deshpande et al., 2003] Deshpande, A., Brandt, C., and Nadkarni, P. (2003). Temporal query of attribute-value patient data: utilizing the constraints of clinical studies. *International Journal of Medical Informatics*, 70(1):49–67.

[Dousson and Duong, 1999] Dousson, C. and Duong, T. V. (1999). Discovering chronicles with numerical time constraints from alarm logs for monitoring dynamic systems. In Dean, T., editor, *Proceedings of the 16th international joint conference on Artifical intelligence - Volume 1*, IJCAI'99, pages 620–626. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Dubois et al., 2004] Dubois, D., Foulloy, L., Mauris, G., and Prade, H. (2004). Probability-possibility transformations, triangular fuzzy sets, and probabilistic inequalities. *Reliable Computing*, 10(4):273–297.

[Duftschmid et al., 2002] Duftschmid, G., Miksch, S., and Gall, W. (2002). Verification of temporal scheduling constraints in clinical practice guidelines. *Artificial Intelligence in Medicine*, 25(2):93–121.

[Fayyad et al., 1996] Fayyad, U., Piatetsky-shapiro, G., and Smyth, P. (1996). From data mining to knowledge discovery in databases. *AI Magazine*, 17:37–54.

[Flemons, 2002] Flemons, W. (2002). Obstructive sleep apnea. *New England Journal of Medicine*, 347(7):498–504.

[Freksa, 1992] Freksa, C. (1992). Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1-2):199–227.

[Freuder, 1982] Freuder, E. C. (1982). A sufficient condition for backtrack-free search. *Journal of the ACM*, 29(1):24–32.

[Gamper and Nejdl, 1997] Gamper, J. and Nejdl, W. (1997). Abstract temporal diagnosis in medical domains. *Artificial Intelligence in Medicine*, 10(3):209–234.

[Garofalakis et al., 1999] Garofalakis, M. N., Rastogi, R., and Shim, K. (1999). Spirit: Sequential pattern mining with regular expression constraints. In Atkinson, M. P., Orlowska, M. E., Valduriey, P., Zdonik, S. B., and Brodie, M. L., editors, *Proceedings of the 25th International Conference on Very Large Data Bases*, VLDB'99, pages 223–234. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Gerevini et al., 1996] Gerevini, A., Perini, A., and Ricci, F. (1996). Incremental algorithms for managing temporal constraints. In *Proceedings of the Eight IEEE International Conference on Tools with Artificial Intelligence*, pages 360–363.

[Giannotti et al., 2006] Giannotti, F., Nanni, M., and Pedreschi, D. (2006). Efficient mining of temporally annotated sequences. pages 346–357, Philadelphia, PA, USA. Society for Industrial and Applied Mathematics.

[Guil et al., 2004] Guil, F., Bosch, A., and Marín, R. (2004). TSET^{MAX}: An algorithm for mining frequent maximal temporal patterns. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04). Workshop on Data Mining, 7 pages*.

[Guimarães et al., 2001] Guimarães, G., Peter, J.-H., Penzel, T., and Ultsch, A. (2001). A method for automated temporal knowledge acquisition applied to sleep-related breathing disorders. *Artificial Intelligence in Medicine*, 23(3):211 – 237.

[Guyet and Quiniou, 2008] Guyet, T. and Quiniou, R. (2008). Mining temporal patterns with quantitative intervals. In *Data Mining Workshops, 2008. ICDMW'08. IEEE International Conference on*, pages 218–227.

[Guyet and Quiniou, 2011] Guyet, T. and Quiniou, R. (2011). Extracting temporal patterns from interval-based sequences. In *Proceedings of the Twenty-Second international joint conference on Artificial Intelligence - Volume Two*, IJCAI'11, pages 1306–1311. AAAI Press.

[Han et al., 2005] Han, J., Kamber, M., and Pei, J. (2005). *Data Mining: Concepts and Techniques*. Morgan-Kaufmann, San Francisco, CA, USA.

[Han et al., 2000a] Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M.-C. (2000a). Freespan: frequent pattern-projected sequential pattern mining. In *Proceedings*

of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining, KDD'00, pages 355–359, New York, NY, USA. ACM.

[Han et al., 2000b] Han, J., Pei, J., and Yin, Y. (2000b). Mining frequent patterns without candidate generation. *SIGMOD Rec.*, 29(2):1–12.

[Hand et al., 2001] Hand, D., Mannila, H., and Smyth, P. (2001). *Principles of Data Mining*. MIT.

[Harms et al., 2001] Harms, S., Deogun, J., Saquer, J., and Tadesse, T. (2001). Discovering representative episodal association rules from event sequences using frequent closed episode sets and event constraints. In *Data Mining, 2001. ICDM 2001, Proceedings IEEE International Conference on*, pages 603–606.

[Hirate and Yamana, 2006] Hirate, Y. and Yamana, H. (2006). Generalized sequential pattern mining with item intervals. *Journal of Computers*, 1(3):51–60.

[Hoppner, 2001] Hoppner, F. (2001). Learning temporal rules from state sequences. In *IJCAI'01 Workshop on Learning from Temporal and Spatial Data*, pages 25–31, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Hu et al., 2009] Hu, Y.-H., Huang, T. C.-K., Yang, H.-R., and Chen, Y.-L. (2009). On mining multi-time-interval sequential patterns. *Data & Knowledge Engineering*, 68(10):1112–1127.

[Jonsson et al., 1999] Jonsson, P., Drakengren, T., and Bäckström, C. (1999). Computational complexity of relating time points with intervals. *Artificial Intelligence*, 109(1–2):273 – 295.

[Kam and Fu, 2000] Kam, P.-s. and Fu, A. W.-C. (2000). Discovering temporal patterns for interval-based events. In Kambayashi, Y., Mohania, M., and Tjoa, A., editors, *Data Warehousing and Knowledge Discovery*, volume 1874 of *Lecture Notes in Computer Science*, pages 317–326. Springer Berlin Heidelberg.

[Khatib et al., 2001] Khatib, L., Morris, P., Morris, R., and Rossi, F. (2001). Temporal constraint reasoning with preferences. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 1*, IJCAI'01, pages 322–327, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Koubarakis, 1997] Koubarakis, M. (1997). From local to global consistency in temporal constraint networks. *Theoretical Computer Science*, 173(1):89 – 112.

[Kowalski and Sergot, 1986] Kowalski, R. and Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, 4(1):67–95.

[Ladkin and Maddux, 1988] Ladkin, P. and Maddux, R. (1988). The algebra of constraint satisfaction problems and temporal reasoning. In *Technical report, Krestel Institute*.

[Li and Wang, 2008] Li, C. and Wang, J. (2008). Efficiently mining closed subsequences with gap constraints. In *Proceedings of the SIAM International Conference on Data Mining, SDM 2008, April 24-26, 2008, Atlanta, Georgia, USA*, pages 313–322. SIAM.

[Li et al., 2003] Li, Y., Ning, P., Wang, X. S., and Jajodia, S. (2003). Discovering calendar-based temporal association rules. *Data & Knowledge Engineering*, 44(2):193–218.

[Ligozat, 1998] Ligozat, G. (1998). Corner relations in Allen's algebra. *Constraints*, 3(2/3):165–177.

[Lin and Lee, 2005] Lin, M.-Y. and Lee, S.-Y. (2005). Fast discovery of sequential patterns through memory indexing and database partitioning. *Journal of Information Science and Engineering*, 21:109–128.

[Lu et al., 2000] Lu, H., Feng, L., and Han, J. (2000). Beyond intratransaction association analysis: mining multidimensional intertransaction association rules. *ACM Transaction on Information Systems*, 18(4):423–454.

[Mannila et al., 1997] Mannila, H., Toivonen, H., and Verkamo, A. (1997). Discovery of frequent episodes in event sequences. *Data Mining and Knowledge Discovery*, 1(3):259–289.

[Marín et al., 1994] Marín, R., Barro, S., Bosch, A., and Mira, J. (1994). Modeling time representation from a fuzzy perspective. *Cybernetics and Systems*, 25(2):201–215.

[Mccarthy and Hayes, 1969] Mccarthy, J. and Hayes, P. J. (1969). Some philosophical problems from the standpoint of artificial intelligence. In *Machine Intelligence*, pages 463–502. Edinburgh University Press.

[Meiri, 1996] Meiri, I. (1996). Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87(1–2):343 – 385.

[Mitra et al., 2002] Mitra, S., Pal, S., and Mitra, P. (2002). Data mining in soft computing framework: a survey. *IEEE Transactions on Neural Networks*, 13(1):3–14.

[Moerchen, 2006] Moerchen, F. (2006). Algorithms for time series knowledge mining. In *KDD'06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 668–673, New York, NY, USA. ACM.

[Moerchen and Ultsch, 2007] Moerchen, F. and Ultsch, A. (2007). Efficient mining of understandable patterns from multivariate interval time series. *Data Mining Knowledge Discovery*, 15:181–215.

[Morris et al., 2001] Morris, P., Muscettola, N., and Vidal, T. (2001). Dynamic control of plans with temporal uncertainty. In *Proceedings of the 17th international joint conference on Artificial intelligence - Volume 1*, IJCAI'01, pages 494–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Nanni and Rigotti, 2007] Nanni, M. and Rigotti, C. (2007). Extracting trees of quantitative serial episodes. In Džeroski, S. and Struyf, J., editors, *Knowledge Discovery in Inductive Databases*, volume 4747 of *Lecture Notes in Computer Science*, pages 170–188. Springer Berlin / Heidelberg. 10.1007/978-3-540-75549-4_11.

[Navarrete et al., 2002] Navarrete, I., Sattar, A., Wetprasit, R., and Marin, R. (2002). On point-duration networks for temporal reasoning. *Artificial Intelligence*, 140(1–2):39 – 70.

[Nebel and Bürckert, 1995] Nebel, B. and Bürckert, H.-J. (1995). Reasoning about temporal relations: a maximal tractable subclass of Allen's interval algebra. *Journal of the ACM*, 42(1):43–66.

[Otero et al., 2011] Otero, A., Félix, P., and Álvarez, M. (2011). Algorithms for the analysis of polysomnographic recordings with customizable criteria. *Expert Systems with Applications*, 38:10133–10146.

[Palma et al., 2006] Palma, J., Juárez, J., Campos, M., and Marín, R. (2006). Fuzzy theory approach for temporal model-based diagnosis: an application to medical domains. *Artificial Intelligence in Medicine*, 38:197–218.

[Pani and Bhattacharjee, 2001] Pani, A. and Bhattacharjee, G. (2001). Temporal representation and reasoning in artificial intelligence: A review. *Mathematical and Computer Modelling*, 34(1–2):55–80.

[Papapetrou et al., 2005] Papapetrou, P., Kollios, G., Sclaroff, S., and Gunopulos, D. (2005). Discovering frequent arrangements of temporal intervals. In *Proceedings of the Fifth IEEE International Conference on Data Mining*, ICDM'05, pages 354–361, Washington, DC, USA. IEEE Computer Society.

[Patel et al., 2008] Patel, D., Hsu, W., and Lee, M. L. (2008). Mining relationships among interval-based events for classification. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, SIGMOD'08, pages 393–404, New York, NY, USA. ACM.

[Pei et al., 2001] Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2001). PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 17th International Conference on Data Engineering*, ICDE'01, pages 215–224, Washington, DC, USA. IEEE Computer Society.

[Pei et al., 2004] Pei, J., Han, J., Mortazavi-Asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., and Hsu, M. (2004). Mining sequential patterns by pattern-growth: the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1424–1440.

[Pei et al., 2002] Pei, J., Han, J., and Wang, W. (2002). Mining sequential patterns with constraints in large databases. In *Proceedings of the eleventh international conference on Information and knowledge management*, CIKM'02, pages 18–25, New York, NY, USA. ACM.

[Pei et al., 2006] Pei, J., Wang, H., Liu, J., Wang, K., Wang, J., and Yu, P. S. (2006). Discovering frequent closed partial orders from strings. *IEEE Transactions on Knowledge and Data Engineering*, 18(11):1467–1481.

[Roddick and Spiliopoulou, 2002] Roddick, J. and Spiliopoulou, M. (2002). A survey of temporal knowledge discovery paradigms and methods. *IEEE Transactions on Knowledge and Data Engineering*, 14(4):750–767.

[Rymon, 1992] Rymon, R. (1992). Search through systematic set enumeration. In *Third Internation Conference on Principles of Knowledge Representation and Reasoning (KR'92)*, pages 539–550.

[Sacchi et al., 2007a] Sacchi, L., Larizza, C., Combi, C., and Bellazzi, R. (2007). Data mining with temporal abstractions: learning rules from time series. *Data Mining and Knowledge Discovery*, 15(2):217–247.

[Sacchi et al., 2007b] Sacchi, L., Larizza, C., Magni, P., and Bellazzi, R. (2007). Precedence temporal networks to represent temporal relationships in gene expression data. *Journal of Biomedical Informatics*, 40(6):761–774.

[Srikant, 1996] Srikant, R. (1996). *Fast algorithms for mining association rules and sequential patterns*. PhD thesis. University of Wisconsin - Madison.

[Srikant and Agrawal, 1996] Srikant, R. and Agrawal, R. (1996). Mining sequential patterns: Generalizations and performance improvements. In Apers, P. M. G., Bouzeghoub, M., and Gardarin, G., editors, *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT'96, pages 3–17. Springer-Verlag, London, UK.

[Staab and Hahn, 1998] Staab, S. and Hahn, U. (1998). Distance constraint arrays: A model for reasoning on intervals with qualitative and quantitative distances. In Mercer, R. E. and Neufeld, E., editors, *Advances in Artificial Intelligence*, volume 1418 of *Lecture Notes in Computer Science*, pages 334–348. Springer Berlin Heidelberg.

[Tatti and Cule, 2011] Tatti, N. and Cule, B. (2011). Mining closed episodes with simultaneous events. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD'11, pages 1172–1180, New York, NY, USA. ACM.

[Trasarti et al., 2008] Trasarti, R., Bonchi, F., and Goethals, B. (2008). Sequence mining automata: A new technique for mining frequent sequences under regular expressions. pages 1061–1066.

[Tsang, 1993] Tsang, E. (1993). *Foundations of constraint satisfaction*. Academic Press, London, UK.

[van Beek, 1992] van Beek, P. (1992). Reasoning about qualitative temporal information. *Artificial Intelligence*, 58(1-3):297–326.

[Van Beek and Cohen, 1990] Van Beek, P. and Cohen, R. (1990). Exact and approximate reasoning about temporal relations1. *Computational Intelligence*, 6(3):132–144.

[van Benthem, 1983] van Benthem, J. (1983). *The Logic of Time*. D. Reidel.

[Vila, 1994] Vila, L. (1994). A survey on temporal reasoning in Artificial Intelligence. *AI Communications*, 7:4–28.

[Vilain et al., 1990] Vilain, M., Kautz, H., and van Beek, P. (1990). Constraint propagation algorithms for temporal reasoning: a revised report. In Weld, D. S. and Kleer, J. d., editors, *Readings in qualitative reasoning about physical systems*, pages 373–381. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Vilain and Kautz, 1986] Vilain, M. B. and Kautz, H. A. (1986). Constraint Propagation Algorithms for Temporal Reasoning. In *5th National Conference on Artificial Intelligence*, AIII, pages 377–382. Morgan Kaufmann.

[Visser and Hübner, 2003] Visser, U. and Hübner, S. (2003). Temporal representation and reasoning for the semantic web. In *Technical report, Center for Computing Technologies. Technical TZI-Bericht Br. 28*.

[Wainer and de Melo, 1997] Wainer, J. and de Melo, A. (1997). A temporal extension to the parsimonious covering theory. *Artificial Intelligence in Medicine*, 10(3):235–255.

[Winarko and Roddick, 2007] Winarko, E. and Roddick, J. F. (2007). Armada - an algorithm for discovering richer relative temporal association rules from interval-based data. *Data & Knowledge Engineering*, 63(1):76–90.

[Wu and Chen, 2007] Wu, S.-Y. and Chen, Y.-L. (2007). Mining nonambiguous temporal patterns for interval-based events. *IEEE Transactions on Knowledge and Data Engineering*, 19(6):742–758.

[Wu and Chen, 2009] Wu, S.-Y. and Chen, Y.-L. (2009). Discovering hybrid temporal patterns from sequences consisting of point- and interval-based events. *Data & Knowledge Engineering*, 68(11):1309–1330.

[Yager and Filev, 1994] Yager, R. and Filev, D. (1994). Approximate clustering via the mountain method. *IEEE Transactions on Systems, Man and Cybernetics*, 24(4):1279–1284.

[Zaki, 2001] Zaki, M. J. (2001). SPADE: An efficient algorithm for mining frequent sequences. *Machine Learning*, 42:31–60.