EFFICIENT SEQUENTIAL AND TEMPORAL PATTERN MINING

Natalia Mordvanyuk

Universitat de Girona

PhD Thesis

Efficient sequential and temporal pattern mining

Natalia Mordvanyuk

2021

Universitat de Girona

DOCTORAL THESIS

Efficient sequential and temporal pattern mining

Natalia Mordvanyuk

2021

DOCTORAL PROGRAM in TECHNOLOGY

Supervised by:
Dra. Beatriz López
Dr. Albert Bifet

[illegible]

Work submitted to the University of Girona in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

Agraïments

Aquesta tesi s'ha dut a terme gràcies a la paciència i els bons consells dels meus dos directors de tesis la Dra. Beatriz López i el Dr. Albert Bifet, als quals agraeixo de tot cor tota la dedicació i paciència que van tenir amb mi. Bea, tinc la sensació d'haver treballat i après moltissim al teu costat, i d'haver gaudit fent-ho. Un agraïment semblant podria fer al meu codirector, Albert Bifet, disposat a escoltar sempre qualsevol proposta, a resoldre qualsevol dubte, i a plantejar noves idees.

M'agradaria donar les gràcies a tots els membres i ex-membres del grup eXiT pel seu suport i bons moments durant l'estada en aquest grup. De fet, els debats sobre els diferents algoritmes entre l'Albert Plà i en Pablo Gay van ser els que em van inspirar a fer aquest doctorat, i que per tant sou els qui heu fet possible que hagi arribat fins on soc ara. A més a més, no se que faria sense les bromes i els consells gastronòmics d'en Quim Massana, els videos per concentrar-se i les plantes de'n Robert Rusek, i els moments d'esmorzar compartits amb tots vosaltres: Sara, Jaume, Alihuen i Òscar.

Voldría també donar-li les gràcies a tota la meva família per tot el seu suport. Especialment m'agradaria agraïrli al meu marit, Llorenç Burgas per tot el que ha fet i fa per mi. A la meva filla, Júlia Burgas per alegrar-me els dies. A la la Glòria Nadal i al Joan Burgas per encoratjar-me i fer costat. I, per descomptat, a la meva mare, Nina Maltseva. Мама! Спасибо тебе огромное за то, что дала жизнь, научила всему, всегда заботишься и поддерживаешь меня. Спасибо за всё. Я люблю тебя.

Finally, I would like to especially thank Albert Bifet and all the DIG (Data, Intelligence and Graphs) research group from the Télécom Paris Tech University for helping me during my time in France.

Acknowledgments

This work was developed with the support of the research group eXiT (Control Engineering and Intelligent Systems) of the IIiA (Institute of Informatics and Applications) of the Department of Electrical and Electronic Engineering and Automation of the University of Girona. This thesis has been funded through the competitive grant for doctoral education IFUdG2017 and a mobility grant MOB2019 (additional support for the mobility of UdG researchers) from the University of Girona granted to Natalia Mordvanyuk. Economic support was also received in the initial years from the PEPPER (Patient Empowerment through Predictive PERsonalised decision support) project, funded by the European Commission, and from the Generalitat de Catalunya 2017 SGR 1551. This thesis received joint funding from ERDF, the Spanish Ministry of the Economy, Industry and Competitiveness (MINECO) and the National Agency for Research, under grant no. RTC 2017-6071-1 (SERAS). This thesis received joint funding from the MoSHCA project, funded by the Ministerio de Economía y Competitividad of the Spanish Government (Ref. EUREKA ITEA 2 no. 11027 - IPT-2012-0943-300000), and has received support of the research group SITES (Generalitat de Catalunya SGR 2014–2016) and of the Universitat de Girona under grant MPCUdG2016. Also, the NVIDIA Corporation donated the Titan XP used in this research.

Publications

The scientific contributions derived from this thesis are listed below.

Journals

- Mordvanyuk, N., López, B. & Bifet, A. (2021). VEPRECO: Vertical databases with pre-pruning strategies and common candidate selection policies to fasten sequential pattern mining. Submitted to: Expert Systems with Applications.
  Quality index: [JCR IF (2019): 5.452, Q1]

- Mordvanyuk, N., López, B. & Bifet, A. (2021). TA4L: Efficient Temporal Abstraction of Multivariate Time Series. Submitted to: Knowledge-Based Systems.
  Quality index: [JCR IF (2019): 5.921, Q1]

- Mordvanyuk, N., López, B. & Bifet, A. (2021). vertTIRP: Robust and efficient vertical frequent time interval-related pattern mining. Expert Systems with Applications, 168, 114276.
  Quality index: [JCR IF (2019): 5.452, Q1]

Side publications developed during the PhD period which are not directly linked to the thesis:

Journals

- Pla, A., Mordvanyuk, N., López, B., Raaben, M., Blokhuis Taco, J. & Holstlag Herman, R. (2017). Bag-of-steps: Predicting lower-limb fracture rehabilitation length by weight loading analysis. Neurocomputing, 268, 109-115.
  Quality index: [JCR IF (2017): 3.241, Q1]

Conferences

- Mordvanyuk, N., Gauchola, J., & López, B. (2021, June). Understanding affective behaviour from physiological signals: Feature learning versus pattern mining. 34th IEEE CBMS International Symposium on Computer-Based Medical Systems, Online Event.

- Mordvanyuk, N., López, B., Reixach, M., Fabrellas, M., Planella, M., Cordon C., Simon, N., Duran, A., Perapoch, J., Bassols, J. & López-Bermejo, A. (2019, April). NOAH: Supporting Premature Babies Care With Mobile Phones. [Poster session]. MSF (Médecins Sans Frontières) Paediatric Days 2019, Stockholm, Sweden. https://doi.org/10.13140/RG.2.2.30306.99529

- Torrent-Fontbona, F., Mordvanyuk, N., & López, B. (2018, February). Prediction of hyperglycaemia and hypoglycaemia events using longitudinal data. [Poster session]. International Conference on Advanced Technologies & Treatments for Diabetes (ATTD), Vienna, Austria. Published in Diabetes Technology & Therapeutics, Vol 20, Issue S1, pp. A-80-A-81

- López, B., Mordvanyuk, N., Gay, P., Pla, A. (2018, October) Knowledge representation and machine learning on wearable sensor data: a study on gait monitoring. International Conference on Data Science, E-learning and Information Systems 2018 (Data'18), Madrid, Spain.

- Mordvanyuk, N., Torrent-Fontbona, F., & López, B. (2017, October). Prediction of hyperglycaemia and hypoglycaemia events using longitudinal data. Volume 300: Recent Advances in Artificial Intelligence Research and Development: Frontiers in Artificial Intelligence and Applications, 227–232. https://doi.org/10.3233/978-1-61499-806-8-227

- Dubosson, F., Mordvanyuk, N., López, B., & Schumacher, M. (2017). Negative results for the prediction of postprandial hypoglycemias from insulin intakes and carbohydrates: analysis and comparison with simulated data. In © Herrero, P., López, B., Martin, C.(eds).(2017). AID 2017: Proceedings of the 2nd International Workshop on Artificial Intelligence for Diabetes held in conjunction with the 16th Conference on Artificial Intelligence in Medicine (AIME): Vienna, Austria: 24th June 2017, p. 25-29. Artificial Intelligence for Diabetes (AID), Artificial Intelligence in Medicine (AIME).

- López, B., Soung, S., Mordvanyuk, N., Pla, A., Gay, P., & López-Bermejo, A. (2017). MATCHuP: An mHealth Tool for Children and Young People Health Promotion. Proceedings of the 10th International Joint Conference on

Biomedical Engineering Systems and Technologies, Porto, Portugal. https://doi.org/10.5220/0006143303130318

Side workshops developed during the PhD period include:

- López, B., Mordvanyuk, N., Massana, J., Torrent-Fontbona, F., Caceres, G., Pous, C. (2018, October) eXiT Research Group at the University of Girona: Artificial Intelligence and Machine Learning Applied to Medicine and Healthcare. I CAEPIA Workshop de Grupos de Investigación Españoles de IA en Biomedicina (IABiomed), Granada, Spain.

Acronyms

ARMADA	an algorithm for discovering richer relative temporal association rules from interval-based Data	6
ASL	American Sign Language Lexicon Video dataset	68
BFS	Breadth-First Search strategy	13
BIBLE	a conversion of the Bible into a sequence database dataset	37
bitSPADE	A Lattice-based Sequential Pattern Mining Algorithm Using Bitmap Representation	14
CI	Chronic illness dataset	71
CM-SPADE	Co-occurrence MAP in Sequential PAttern Discovery using Equivalence classes	14
CM-SPAM	Co-occurrence MAP in Sequential PAttern Mining	3
CUDA	Compute Unified Device Architecture (Nvidia)	117
DEAP	A Dataset for Emotion Analysis using EEG, Physiological and Video Signals	116

DFS Depth-First Search strategy 14

EWD Equal Width Discretization 20

FIFA a Clickstream data from the website of FIFA World Cup 98 37

FreeSpan Frequent pattern-projected sequential pattern mining 3

GPU a graphics processing unit 117

GSP Generalized Sequential Pattern 3

HAR Human Activity Recognition dataset dataset 61

KBTA Knowledge-Based Temporal Abstraction 20

LSTISs Lexicographical Symbolic Time Interval Sequences 2

MAV Mavlab dataset 73

MEMISP a MEMory Indexing approach for fast Sequential Pattern mining . 17

PrefixSpan Prefix-projected Sequential PAtterN mining 14

SAX Symbolic Aggregate approXimation 21

SDB Sequential DataBase 14

SPADE Sequential PAttern Discovery using Equivalence classes 3

SPAM Sequential PAttern Mining 3

SPM Sequential pattern mining 1

TA Temporal Abstraction 7

TA4L Temporal Abstraction for lexicographical symbolic time interval sequences 7

TD4C Temporal Discretization for Classification 20

TIRP Time interval-related pattern 1

VEPRECO VErtical databases with PRE-pruning strategies and COmmon candidate selection policies to fasten sequential pattern mining 4

vertTIRP Vertical Frequent Time Interval-Related Pattern Mining 9

Contents

List of figures xix

List of tables xxi

Abstract xxiii

Resum xxv

Resumen xxvii

1 Introduction 1

1.1 Motivation . . . 2

1.2 Sequential pattern mining . . . 3

1.3 TIRP mining . . . 4

1.4 Pre-processing for TIRP mining . . . 7

1.5 Objectives . . . 8

1.6 Contributions of this thesis . . . 8

1.7 Thesis outline . . . 10

2 State of the art 13

2.1 Sequential Pattern Mining . . . 13

2.1.1 Pruning in sequential pattern mining . . . 15

2.1.2 How the contributions of this thesis fit within the context of the existing SPM literature? . . . 15

2.2 TIRP Mining . . . 17

2.2.1 How the contributions of this thesis fit within the context of the existing TIRP mining literature? 18
2.3 Pre-processing 18
2.3.1 Temporal abstraction in the TIRPs mining field 20
2.3.2 How the contributions of this thesis fit within the context of the existing TA for TIRPs mining literature? 21

3 An improved sequential pattern mining: VEPRECO 23
3.1 Problem statement 23
3.2 The VEPRECO algorithm 24
3.2.1 DictMap 27
3.2.2 Pre-pruning strategies 29
3.2.3 A novel operation: c-extend 33
3.3 Experimental Evaluation methodology 35
3.3.1 Experimental platform for VEPRECO 35
3.3.2 Datasets 35
3.3.3 Experimental setup 36
3.4 Results 37
3.5 Discussion 42
3.6 Summary 45

4 Managing temporal information in sequential patterns: vertTIRP 47
4.1 Problem statement 47
4.1.1 Time interval sequence 47
4.1.2 Robust temporal relations 48
4.1.3 TIRP 53
4.2 The vertTIRP algorithm 55
4.2.1 Tabular LSTIS 56
4.2.2 Constraints on user control patterns 56
4.2.3 Pairing strategy 59
4.2.4 Vertical representation of TIRPs 61
4.2.5 Candidate generation 62

4.2.6 Transitivity for efficiency 64
4.3 Experimental Evaluation methodology 67
4.3.1 Experimental platform for vertTIRP 67
4.3.2 Datasets 67
4.3.3 Experimental setup 69
4.4 Results 71
4.5 Discussion 79
4.6 Summary 80

5 Data preparation as an external component of learning algorithms: TA4L 83
5.1 Problem statement 83
5.1.1 Input parameters 83
5.1.2 Output: LSTISs 84
5.2 The TA4L algorithm 86
5.2.1 Normalization 87
5.2.2 Framing 88
5.2.3 Discretization 88
5.2.4 Segmentation 90
5.2.5 Sequence generation 90
5.2.6 Data structures and parallel strategies 92
5.3 Experimental Evaluation methodology 94
5.3.1 Experimental platform for TA4L 95
5.3.2 Datasets 95
5.3.3 Experimental setup 97
5.4 Results 99
5.5 Discussion 112
5.6 Summary 113

6 Conclusions 115
6.1 Conclusions and summary of results 115
6.2 Future work 116

Bibliography 118

A Implementation of VEPRECO 129

B vertTIRP's appendix 131

B.1 Implementation of vertTIRP 131

B.2 Table of pairing strategies for epsilon=0 131

B.3 Intersection of temporal relations 132

B.4 Allen's transitivity table 132

B.5 Complete mining example 132

C TA4L's appendix 137

C.1 Implementation of TA4L 137

C.2 Auxiliar functions of TA4L 137

List of Figures

1.1 Temporal relations and their abbreviations (in green). 5

1.2 Example of new ambiguities arising from the use of an epsilon margin. 6

1.3 Example of the transitivity property between temporal events with and without the use of an epsilon margin. 6

2.1 Pre-processing step in the literature. 19

3.1 A sequence tree constructed from a sequential database. 26

3.2 DictMap 28

3.3 Tree's nodes of VEPRECO 31

3.4 SDB 1 31

3.5 SDB 2 31

3.6 SDB 3 31

3.7 SDB 4 31

3.8 c-extend function 33

3.9 c-extension 34

3.10 VEPRECO's and CM-SPAM's results for synthetic datasets. 38

3.11 VEPRECO's and CM-SPAM's results for real-world datasets. 39

3.12 Synthetic datasets: How tree traversing and pruning strategies affect the time and memory of the algorithm. 40

3.13 Real-world datasets: How tree traversing and pruning strategies affect time and memory of the algorithm 41

3.14 Synthetic datasets: How the c extension affect time and memory, where Mem. is memory usage, and Pat. is the number of patterns . . 43

3.15 Real-world datasets: How the c extension affect time and memory . . 44

4.1 Graphical representation of an example 49
4.2 Examples of a lack of robustness when epsilon margins are introduced. 50
4.3 Illustration of the need for the left contain relation 51
4.4 Illustration of how the problem of the lack of robustness is solved for overlaps, is finished by, meets, starts, equals, overlaps and contains . 52
4.5 The resulting TIRP tree. 58
4.6 Analysis of Allen's transitivity relation for $\epsilon = 0$ minutes, $r_1(A, B) = o$ and $r_2(B, C) = f$ 65
4.7 Analysis of Allen's transitivity relation for $\epsilon = 5$ minutes, $r_1(A, B) = o$ and $r_2(B, C) = f$ 65
4.8 Comparison with the state of the art methods for the MAV and ASL datasets 72
4.9 Efficiency by transitivity properties: transitivity vs. no transitivity (time in seconds; the scale of the y-axis is not uniform) 73
4.10 Efficiency with application of the min duration constraint. 74
4.11 Efficiency with application of the min gap constraint. 75
4.12 Efficiency with pairing of events: dummy approach vs. vertTIRP (time in seconds) 76
4.13 Epsilon sensitivity. 77
4.14 Complexity analysis of the vertTIRP algorithm in terms of time, memory and vocabulary size 78

5.1 Graphical representation of LSTISs 85
5.2 TA4L over multivariate time series of two variables. 87
5.3 SAX and TA4L over a signal. 89
5.4 The difference between the original binary insert and an adapted binary insert that remembers the last inserted position applied to LSTISs. 91
5.5 Concatenation example over a signal. 93
5.6 Parallelism strategies of TA4L 94
5.7 Synthetic datasets. How varying input parameters δ and $|\Sigma|$ affects the performance of the algorithms 101
5.8 Real datasets. How varying the δ affects the performance of the algorithms 102

5.9 Real datasets. How varying the $|\Sigma|$ affects the performance of the algorithms. 103
5.10 Synthetic datasets. How varying the PM and max_gap parameters affects the performance of the algorithms. 104
5.11 Synthetic datasets. Sensitivity to discretization methods while varying n. 105
5.12 Synthetic datasets. Sensitivity to discretization methods while varying q 105
5.13 Synthetic datasets. Sensitivity to discretization methods while varying n_{sid} 105
5.14 Real datasets. Sensitivity to discretization methods in terms of memory while varying q 106
5.15 Real datasets. Sensitivity to discretization methods in terms of memory while varying n 107
5.16 Real datasets. Sensitivity to discretization methods in terms of memory while varying n_{sid} 107
5.17 Real datasets. Sensitivity to discretization methods in terms of time while varying q 108
5.18 Real datasets. Sensitivity to discretization methods in terms of time while varying n 109
5.19 Real datasets. Sensitivity to discretization methods in terms of time while varying n_{sid} 110
5.20 How varying n and q affects the performance of the algorithms. . . . 111
5.21 How varying n_{sid} affects the performance of the algorithms. 111
5.22 How varying PM affects the performance of the algorithms, where |I| is the total number of time intervals. 112

B.1 Length-one S-TIRPs for the example in Table 4.1 134
B.2 Illustration of the extension $A - B$. 135
B.3 Illustration of the extension $AB - C$. 136

List of Tables

3.1 Synthetic datasets. 36
3.2 Public dataset 36

4.1 LSTIS for the example in Figure 4.1 57
4.2 Results of pairing strategies for each dataset, for epsilon>0 60
4.3 Common conditions among the temporal relations. 61
4.4 Horizontal database. 62
4.5 Vertical database. 62
4.6 Example of the vertical representation of patterns in verTIRP 63
4.7 Transitivity table for eps>0 66
4.8 Dataset description 70

5.1 Parallelism strategies of TA4L and corresponding data structures. . . 95
5.2 Dataset description. 97

B.1 Results of pairing strategies for each dataset for epsilon=0. 131
B.2 Intersections between temporal relations 132
B.3 Temporal relations from Allen's transitivity property 133

Abstract

The contributions of the present thesis are in the domain of Pattern Mining and Knowledge Discovery, being of particular relevance for the sequential pattern mining and time-interval related pattern mining fields.

Sequential pattern mining discovers, from event transactions recorded along time, patterns of events fulfilling a sequential order. This problem is an essential theme in data mining research and has broad applications in many different areas of behavioural analysis and prediction. However, it is a challenging problem for sequential pattern mining, as it involves examining a large number of intermediate subsequences, requiring significant memory and processing time. In this thesis, a new efficient sequential pattern mining algorithm called VEPRECO is introduced. VEPRECO has three main contributions that fasten the mining process: (i) a vertical representation of patterns; (ii) a pre-pruning strategies to avoid checking infrequent patterns; and (iii) a common candidate selection policies that reduce the number of iterations performed by the algorithm. An experimental evaluation has been carried out with synthetic and real data sets, and the results obtained have been better compared to the CM-SPAM algorithm, which has been taken as a state-of-the-art reference.

The inclusion of temporal relationships between the events ("A starts B", "A overlaps B") makes mining patterns even more expensive than mining sequential patterns. The inclusion of temporal relations in pattern mining is known as time-interval related pattern mining, that despite its high computationally cost, has gained much popularity in this last decade. The advantage of time-interval-related patterns is that they are more informative patterns than sequential ones that not only enrich the data analysis, but also give good results in behavior prediction problems. In this thesis, a new efficient algorithm for mining time-interval-related patterns is introduced, called vertTIRP. vertTIRP combines an efficient representation of these patterns, using their temporal transitivity properties to manage them, with a pairing strategy that sorts the temporal relations to be tested in order to speed up the mining process.

Moreover, this thesis presents a robust definition of the temporal relations that eliminate the ambiguities with other relations when considering the uncertainty in the start and end time of the events (epsilon-based approach). vertTIRP also includes a pair of constraints that allow the user to define better the time-interval-related patterns to be mined. An experimental evaluation of the method has been carried out with synthetic and real data sets, and the results show that vertTIRP requires significantly less computation time than other state-of-the-art algorithms and is an effective approach.

Finally, in this thesis, to open the door of time-interval-related pattern mining to multivariate time series (or time series), the TA4L algorithm is introduced. TA4L is an efficient algorithm to transform multivariate time series (or time series) into Lexicographically ordered Symbolic Time Interval Sequences, that is, sequences ready to feed time-interval-related pattern mining algorithms. The ultimate goal is to make the built-in ad-hoc preprocesses related to time-interval-related pattern mining algorithms explicit and at the same time offer an efficient solution for data preprocessing. TA4L divides the signals into segments based on time duration (instead of dividing them according to the number of samples, a method often used in the literature), allowing the construction of consistent time intervals. Concatenation of intervals is controlled by a maximum time gap constraint that reinforces the generated time intervals' consistency. Moreover, different ways to parallelise the algorithm are explored. TA4L has been experimentally evaluated with synthetic and real-world datasets, and the results show that TA4L requires significantly less computation time than other state-of-the-art approaches, revealing that it is an effective algorithm.

Resum

Les contribucions d'aquesta tesi estan en el domini de la mineria de patrons i el descobriment del coneixement, sent de particular rellevància pels camps de mineria de patrons seqüencials i els patrons d'intervals temporals.

La mineria de patrons seqüencials descobreix, a partir de transaccions d'esdeveniments registrades al llarg de el temps, patrons d'esdeveniments que compleixen un ordre seqüencial. Aquest problema és un tema essencial en la investigació de mineria de dades i té àmplies aplicacions en moltes àrees diferents d'anàlisi i predicció del comportament. No obstant això, és un problema que presenta reptes importats per a la mineria de patrons seqüencials, ja que implica examinar un gran nombre de subseqüències intermèdies, el que requereix una memòria i un temps de processament significatius. En aquesta tesi, s'ha presentat un nou algoritme eficient de mineria de patrons seqüencials anomenat VEPRECO. VEPRECO té tres contribucions principals que acceleren el procés de mineria: (i) una representació vertical dels patrons, (ii) unes estratègies de prepoda per evitar verificar patrons poc freqüents i (iii) unes polítiques de selecció de candidats comuns que redueixen el nombre d'iteracions realitzades per l'algoritme. S'ha realitzat una avaluació experimental amb conjunts de dades sintètiques i reals, i els resultats obtinguts han estat millors en comparació amb l'algorisme CM-SPAM, que s'ha pres com a referència de l'estat de l'art.

La inclusió de relacions temporals entre els esdeveniments ("A comença B" o "A es superposa a B") fan que la mineria de patrons sigui encara més costosa que la de patrons seqüencials. La inclusió de relacions temporals en la mineria de patrons es coneix com a mineria de patrons d'intervals temporals, que malgrat el seu elevat cost computacional, ha guanyat molta popularitat en aquesta última dècada. L'avantatge dels patrons d'intervals temporals és que són patrons més informatius que els seqüencials que no només enriqueixen l'anàlisi de dades, sinó que també donen bons resultats en els problemes de predicció del comportament. En aquesta tesi, s'ha presentat un nou algoritme eficient per minar patrons d'intervals temporals, denominat vertTIRP. vertTIRP combina una representació eficient d'aquests patrons, utilitzant les seves propietats de transitivitat temporal per gestionar-los, amb una estratègia d'aparellament que ordena les relacions temporals a provar per tal d'accelerar el procés de la mineria.

A més a més, aquesta tesi presenta una definició robusta de les relacions temporals que elimina les ambigüitats amb altres relacions al considerar la incertesa en els temps d'inici i final dels esdeveniments (enfocament basat en èpsilon). vertTIRP també inclou un parell de restriccions temporals que permeten a l'usuari definir millor els patrons d'intervals temporals a minar. S'ha realitzat una avaluació experimental del mètode amb conjunts de dades sintètiques i reals, i els resultats mostren que vertTIRP requereix un temps de càlcul significativament menor que altres algoritmes de literatura i que és un enfocament eficaç.

Finalment, en aquesta tesi, per obrir la porta de la mineria de patrons d'intervals temporals a sèries temporals multivariades (o sèries temporals), s'ha presentat l'algoritme TA4L. TA4L és un algoritme eficient per transformar sèries temporals multivariades (o sèries temporals) en seqüències d'intervals de temps simbòlics ordenats lexicogràficament, és a dir, seqüències aptes per alimentar algoritmes de mineria de patrons d'intervals temporals. L'objectiu final és fer explícits els preprocesos ad-hoc integrats relacionats amb els algoritmes de mineria de patrons d'intervals temporals, i al mateix temps oferir una solució eficient per al preprocessament de dades. TA4L divideix les senyals en segments segons la durada (en lloc de dividir-les segons el nombre de mostres, mètode que sovint s'utilitza en la literatura), el que permet la construcció d'intervals de temps consistents. La concatenació d'intervals es controla mitjançant una restricció d'interval de temps màxim que reforça la coherència dels intervals de temps generats. A més a més, s'exploren diferents formes de paral·lelitzar l'algoritme. TA4L s'ha avaluat experimentalment amb conjunts de dades sintètiques i reals, i els resultats mostren que és un algoritme eficaç.

Resumen

Las contribuciones de la presente tesis están en el dominio de la minería de patrones y el descubrimiento del conocimiento, siendo de particular relevancia para los campos de minería de patrones secuenciales y los patrones de intervalos temporales.

La minería de patrones secuenciales descubre, a partir de transacciones de eventos registradas a lo largo del tiempo, patrones de eventos que cumplen un orden secuencial. Este problema es un tema esencial en la investigación de minería de datos y tiene amplias aplicaciones en muchas áreas diferentes de análisis y predicción del comportamiento. Sin embargo, es un problema que presenta retos para la minería de patrones secuenciales, ya que implica examinar un gran número de subsecuencias intermedias, lo que requiere una memoria y un tiempo de procesamiento significativos. En esta tesis, se ha presentado un nuevo algoritmo eficiente de minería de patrones secuenciales llamado VEPRECO. VEPRECO tiene tres contribuciones principales que aceleran el proceso de minería: (i) una representación vertical de los patrones, (ii) unas estrategias de prepoda para evitar verificar patrones poco frecuentes y (iii) unas políticas de selección de candidatos comunes que reducen el número de iteraciones realizadas por el algoritmo. Se ha realizado una evaluación experimental con conjuntos de datos sintéticos y reales, y los resultados obtenidos han sido mejores en comparación con el algoritmo CM-SPAM, que se ha tomado como referencia del estado de arte.

La inclusión de las relaciones temporales entre los eventos ("A comienza B", "A se superpone a B") hacen que la minería de patrones sea aún más costosa que la de patrones secuenciales. La inclusión de relaciones temporales en la minería de patrones se conoce como minería de patrones de intervalos temporales, que a pesar de su elevado coste computacional, ha ganado mucha popularidad en esta última década. La ventaja de los patrones de intervalos temporales es que son patrones más informativos que los secuenciales que no solo enriquecen el análisis de datos, sinó que también dan buenos resultados en los problemas de predicción del comportamiento. En esta tesis, se ha presentado un nuevo algoritmo eficiente para minar los patrones de intervalos temporales, denominado vertTIRP. vertTIRP combina una representación eficiente de estos patrones, utilizando sus propiedades de transitividad temporal para gestionarlos, con una estrategia de emparejamiento que ordena las relaciones temporales a probar con el fin de acelerar el proceso de minería.

Además, esta tesis presenta una definición robusta de las relaciones temporales que elimina las ambigüedades con otras relaciones al considerar la incertidumbre en el tiempo de inicio y fin de los eventos (enfoque basado en épsilon). vertTIRP también incluye un par de restricciones temporales que permiten al usuario definir mejor los patrones de intervalos temporales a minar. Se ha realizado una evaluación experimental del método con conjuntos de datos sintéticos y reales, y los resultados muestran que vertTIRP requiere un tiempo de cálculo significativamente menor que otros algoritmos de literatura y es un enfoque eficaz.

Finalmente, en esta tesis, para abrir la puerta de la minería de patrones de intervalos temporales a series temporales multivariadas (o series temporales), se ha presentado el algoritmo TA4L. TA4L es un algoritmo eficiente para transformar series temporales multivariadas (o series temporales) en secuencias de intervalos de tiempo simbólicos ordenados lexicográficamente, es decir, secuencias listas para alimentar algoritmos de minería de patrones de intervalos temporales. El objetivo final es hacer explícitos los preprocesos ad-hoc integrados relacionados con los algoritmos de minería de patrones de intervalos temporales, y al mismo tiempo ofrecer una solución eficiente para el preprocesamiento de datos. TA4L divide las señales en segmentos según la duración (en lugar de dividirlas según el número de muestras, método que a menudo se utiliza en la literatura), lo que permite la construcción de intervalos de tiempo consistentes. La concatenación de intervalos se controla mediante una restricción de intervalo de tiempo máximo que refuerza la coherencia de los intervalos de tiempo generados. Además, se exploran diferentes formas de paralelizar el algoritmo. TA4L se ha evaluado experimentalmente con conjuntos de datos sintéticos y reales, y los resultados muestran que TA4L requiere un tiempo de cálculo significativamente menor que otros enfoques de la literatura, lo que revela que es un algoritmo eficaz.

Chapter 1

Introduction

This thesis focus on the efficiency of the algorithms of the two most important branches of pattern mining: (i) on the most popular and traditional branch, which is Sequential pattern mining (Sequential pattern mining (SPM)), and on the more recent and richer in knowledge branch, which is time-interval related pattern (Time interval-related pattern (TIRP)s) mining.

SPM deals with the problem of finding the most relevant frequent subsequences (or patterns) from collections of structured data that are represented in a sequential manner, with a frequency no less than a user-specified threshold [1]. This problem has been a focused theme in data mining research for over two decades and has broad applications in many different areas of behavioural analysis and prediction. However, it is also a challenging problem since the mining may have to generate or examine a combinatorially explosive number of intermediate subsequences, requiring significant memory and processing time.

The inclusion of the start and the end times as characteristics of the events in the sequences improves the data representation and enables finding richer patterns known as TIRP. A TIRP is a structured set of data representing the events, the order in which they occur, and the temporal relations between every two pair of events (i.e. event A happens before event B, event A meets event B, A overlaps B, and so on). The data mining field that deals with finding the most relevant TIRPs from collections of time interval events is called Time-interval-related pattern mining (or TIRPs mining). If the SPM problem was already computationally expensive, the relations between events make TIRPs mining even more costly than SPM. The advantage of TIRPs is that they are more informative patterns than sequential ones that enrich the data analysis and give good results in behavioural prediction problems [2, 3, 4].

Therefore, despite its high computationally cost, reasoning, analysing and mining interval-based events begins to have a lot of popularity in this last decade. Apart from its high cost, TIRPs mining has another big gap related to its information looseness derived from its ambiguous relation's definition. For example, with the state-of-the-art methods, Nausea ending with a Headache can be confused with a Headache that appears right after Nausea. For a doctor, Nausea ending with a Headache should not be the same as the Headache that occurs right after Nausea. The first case, for example, is two common symptoms of migraine, and in the second case, the two symptoms could be related to different health conditions.

Another disadvantage of the TIRPs mining algorithms is that they can not process multivariate time series (neither time series) directly. They usually require a pre-processing phase to obtain sorted sequences of time-interval data, called Lexicographical Symbolic Time Interval Sequences (Lexicographical Symbolic Time Interval Sequences (LSTISs)). The pre-processing stage consists of several steps and is complex. This pre-processing is usually overlooked in the TIRPs mining literature, which focuses on providing details about the pattern-finding algorithms. Besides, in TIRPs mining, as in SPM, the algorithms consider the user's restrictions (or constraints), such as the maximum gap (the time allowed between the two consecutive events) or the duration of the patterns. These restrictions should be considered from the early pre-processing stage and not just in the mining stage. Furthermore, in literature, there is no explicit unsupervised pre-processing method that transforms multivariate time series into LSTISs, which is both efficient and considers the temporal restrictions of the user.

1.1 Motivation

Given that current Pattern Mining algorithms are computationally expensive and that we live in a big data world, it is crucial to make advances towards improving their efficiency. Otherwise, it would soon become obsolete. In particular, this thesis has been carried out in the Medicine and Healthcare branch of the "eXiT" research group at the University of Girona, where several big data problems in these fields are subject of study and motivate the research. In that regard, most of the examples provided in this works are inspired by the medical field. Also, SPM and TIRPs mining are among the most important areas of knowledge discovery, and besides, they are multidisciplinary. Therefore improving the efficiency of these algorithms is crucial to make the most of their potential. Since TIRPs mining is a sibling field of SPM, the initial idea of this thesis was to develop an efficient SPM algorithm and then adapt it to mine TIRPs. Therefore, first, an efficient SPM algorithm was developed. However, while adjusting the SPM algorithm to mine TIRPs, it was observed that the difference between the two algorithms was still huge. For this reason, to make an efficient TIRPs mining algorithm, this thesis resorted to the best methods of TIRPs mining literature.

Furthermore, it would be interesting to open the door of TIRPs mining to other non-interval data types to make the most of its potential. For example, to develop a TIRPs mining's pre-processing algorithm for the most common data type in the literature, the multivariate time series or time-series data. And regarding the temporal constraints established by the user, it is crucial to consider the temporal constraints from the early pre-processing phase and not just from the mining phase to obtain correct TIRPs.

1.2 Sequential pattern mining

Sequence pattern mining (SPM) discovers frequent subsequences (or patterns) in a sequence database (i.e. collections of data represented sequentially) that allow finding associations between the different events for purposes such as data analysis, classification, prediction or planning. This field of research has emerged in the 1990s with the pioneering paper of Agrawal and Srikant [5], and which since that moment became a fundamental data mining field with broad applications, such as market basket analysis, text analysis, energy reduction in smart-homes, web-page click-stream analysis, e-learning, sentiment analysis, programmer's code recommendation and bioinformatics.

Discovering sequential patterns is a hard problem. A sequence containing q items in a sequence database can have up to $2^q - 1$ distinct subsequences. Thus, during the last three decades, there was much research aimed to reduce the runtime and memory usage of the sequential pattern mining process. Some works minimise the number of candidate sequences generated [6, 7, 8]; others pay attention to how sequences are generated and stored [9, 10]; others explore the way in which support is counted and how candidate sequences are tested for frequency [11, 12], and others improve the data structure used to store frequent sequences to save memory [13, 14].

The present thesis aims to go further on algorithm efficiency, taking as baseline the best sequential pattern mining algorithms for the moment, Co-occurrence MAP in Sequential PAttern Mining (CM-SPAM) [15]. CM-SPAM complements its predecessor Sequential PAttern Mining (SPAM) [16], with a table that helps to generate only useful candidate patterns, allowing to reduce an enormous amount of useless candidate checks. SPAM itself is a powerful approach that integrates the ideas of Generalized Sequential Pattern (GSP) [17], Sequential PAttern Discovery using Equivalence classes (SPADE) [18], and Frequent pattern-projected sequential pattern mining (FreeSpan) [19], and according to a recent survey about sequential pattern mining [20], CM-SPAM is one of the best sequential pattern mining algorithms. Because CM-SPAM is based on SPAM, there is at minimum one issue that remains to solve. SPAM is not a space-efficient algorithm. In view of this, the VErtical databases with PRE-pruning strategies and COmmon candidate selection policies to fasten sequential pattern mining (VEPRECO) algorithm is presented, which aims to manage memory in a more efficient approach. Moreover, other improvements are proposed regarding the procedure candidates are generated and tested.

1.3 TIRP mining

Time-interval-related pattern (TIRP) mining deals with the problem of finding the most relevant patterns from collections of sequences of time interval events. A TIRP is a structured set of data that represents the events, the order in which they occur, and the temporal relations between two pair of events (i.e. event A is before event B, event A meets event B, A overlaps B, and so on).

TIRPs mining applications include finding patterns in stock prices ("stock B rose during the rising duration of stock A" [21]), skating data ("gluteus is very high when leg is gliding" [22]), classification in the health field ("patients that have the flu starts with fever before cough", [4, 2]), analysis of medical treatments with Electronic Medical Record data ("patients that start treatment A and finish by treatment B survive"[23]), library lending [24], and analysis of the readings of sensors installed in a computerised apartment ("gas switch on during fridge door opening", [25]).

In the last two decades, several works have been dedicated to mining TIRPs. Kam el al. [26] put forward one of the first of these methods, which used Allen's temporal relationships [27], i.e. the before, after, during, contains, meets, is met by, overlaps, is overlapped by, starts, is started by, finishes, is finished by, and equal relations. The temporal relations, although equal, are dual (before/after, meets/is met-by, overlaps/is overlapped-by, during/contains, finishes/is finished-by, starts/is started-by; see Figure 1.1), and this fact allows these methods to be defined by a reduced set of temporal relations, assuming a complementary process for the dual ones.

Wu and Chen [21] found a looseness in temporal pattern expressions when more than two temporal intervals are considered. For example, in the pattern "(A overlaps C) overlaps B", the relation between A and B is not precisely known (A may overlap B, or A may happen before B). Hence, Wu and Chen proposed a new representation for temporal pattern expressions which unambiguously describes the temporal relationships between each pair of events.

Noise data can also be a source of ambiguity. For example, two events starting at 8:00 and 8:02 could be considered the same, due to issues of noise (clock synchro-

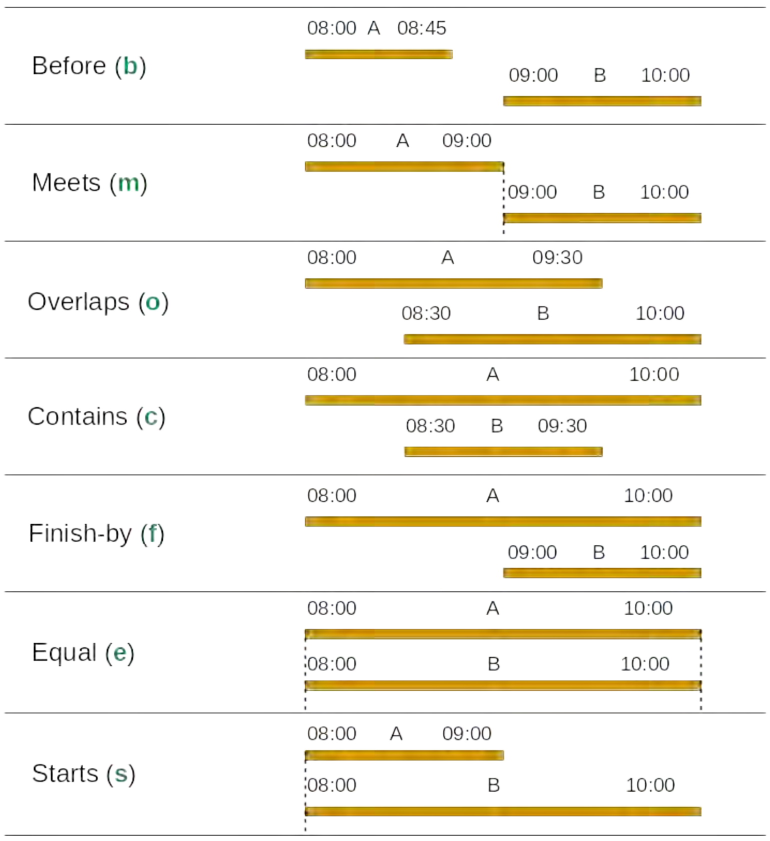


Figure 1.1: Temporal relations and their abbreviations (in green).

nisation). To deal with situations such as these, Papapetrou et al. [28] redefined the Allen relations with an epsilon value, to allow for a certain variability in the temporary boundaries of events caused by noisy data. However, the use of epsilon margins introduces ambiguities in temporal relations. For example, consider A and B in Figure 1.2. With no margin, A overlaps B; however, if an epsilon value of 10 minutes is considered, A may either meet B or be finished by B: If the epsilon is applied to the end time of A with the start time of B - the relation is meets, but if it is applied to the end time of A with the end time of B - the relation is finished by. Without a robust definition of the relations, the event in Figure 1.2 would be labelled with both relations, meets and finished by, that could not only create confusion, but would also slow down the mining process. To solve this problem, in this thesis, relationships have been defined in such a way that each pair of events can only be labelled with a single relation.

The epsilon margin was used in KarmaLego [4] when the transitivity property of Allen's temporal relationships (i.e. "A before B" and "B before C" implies "A before C") was exploited to speed up the mining process. However, the introduction

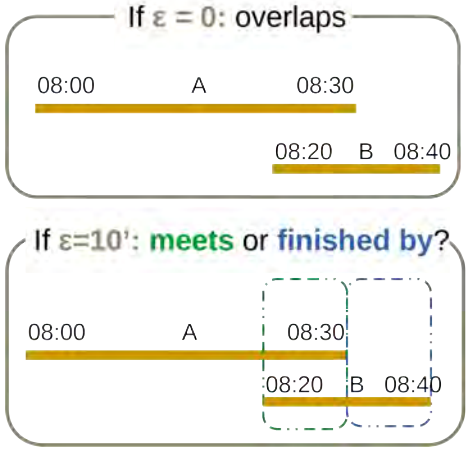


Figure 1.2: Example of new ambiguities arising from the use of an epsilon margin.

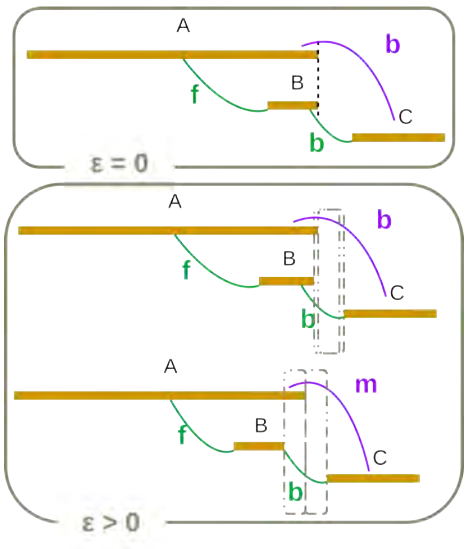


Figure 1.3: Example of the transitivity property between temporal events with and without the use of an epsilon margin.

of the epsilon margin raises several issues regarding the transitivity property of temporal reasoning. For example, consider the three events in Figure 1.3. With no margin, there are "A is finished by B", "B is before C", and from transitivity, "A occurs before C". However, if an epsilon margin is included, a new relation between A and C may hold (i.e. "A meets C"). The epsilon concept allows to move the events in the timeline, generating different relations that were not considered in the Allen's transitivity property. For example, consider C is Diarrhoea, in addition to A that is Nausea and B that is Headache. Nausea that ends with a Headache and after a while Diarrhoea arrives, could mark the separation between the symptoms of two different diseases, while Nausea that comes followed by Diarrhoea which ends with a Headache could be symptoms of the same disease. Both situations should be explored as possible patterns. For this reason, in the present thesis a new transitivity property adapted to the epsilon-based definitions was elaborated.

On the other hand, to improve the efficiency of the TIRP mining, Winarko and Roddick [29] proposed an algorithm for discovering richer relative temporal association rules from interval-based Data (ARMADA) to prune the search space based on the maximum gap constraint, which controls the separation between two events that may form a pattern. While this approach opens the way for the introduction of constraints to the TIRPs mining process, there is also much more scope for including additional constraints to enable the user to express the kind of patterns to be learnt. For example, in a clinical setting, low-level glucose events can only be considered hyperglycaemia if their duration is longer than a certain minimum time, and the user may therefore need to express a minimum time between events.

The way in which data are represented also plays a key role in the efficiency of a mining algorithm. The majority of TIRP methods apply a horizontal approach that focuses on accessing the source data (i.e. the sequence of time interval events), while vertical approaches, which have been successful in similar machine learning areas such as sequence learning, are seldom used in TIRPs mining (see [28] for a tentative approach).

The present thesis is concerned with solving the ambiguities arising from the use of the epsilon margin while taking advantage of and combining previous works regarding efficiency.

1.4 Pre-processing for TIRP mining

One of the major drawbacks of the TIRPs mining algorithms is that they can not process multivariate time series (neither time series) directly. They require a pre-processing phase to obtain sorted sequences of time-interval data. The pre-processing phase consists of several steps, usually carried out sequentially: 1-time series segmentation into intervals according to a given size, 2-discretization of numeric values, 3-sequence construction and 4-sequence sorting. The first two steps comprise the Temporal Abstraction (Temporal Abstraction (TA)) task, which involves the transformation of series of values indexed by time (i.e. $\langle 0.5, 8:00\rangle, \langle 0.7, 10:00\rangle, \langle 0.8, 12:00\rangle$, into time-intervals and the corresponding symbols that abstract the variable values along them (i.e. $\langle A, [8:00, 12:00]\rangle$). On the other hand, the third step concerns the concatenation of intervals with the same symbol (i.e. $\langle A, [8:00, 18:00]\rangle$ when $\langle A, [13:00, 18:00]\rangle$ follows $\langle A, [8:00, 12:00]\rangle$). Regarding the sequence sorting step, the sorting criteria from [4] proved to be necessary to achieve a robust and simple representation of TIRPs. That means that sequences are sorted according to the starting time of intervals, ending time, and the discretized values called Lexicographical Symbolic Time Interval Sequences (LSTISs). Nevertheless, this pre-processing is usually overlooked in the literature, which focuses on providing details about the pattern-finding algorithms.

Considering the computational complexity involved in the whole pre-processing step of the TIRPs mining algorithms, we believe that the present thesis sheds light on the most critical performance bottleneck. The Temporal Abstraction for lexicographical symbolic time interval sequences (TA4L) algorithm introduced in the present thesis aims to accelerate the pre-processing time, merging all these tasks to be executed together in a single, special purpose algorithm. As the algorithm presented in this thesis performs a temporal abstraction, and it returns LSTISs, it is decided to call it the name resulting from the union of the initials of the two concepts "TA4L".

In TA4L, intervals are constructed based on a certain duration. This fact plays an important role when the dataset has missing values and could eventually lead to losing some TIRPs. Conversely, TA4L split input data into segments of equal time duration that later are concatenated. Such pre-processing achieves a more reliable time representation of the variables inside intervals and provides a solid basis for forthcoming TIRPs mining.

1.5 Objectives

Based on the proposed motivations and the research's challenges, three main objectives for this thesis has been established.

1. The first objective focuses on developing a new efficient sequential pattern mining algorithm with several new techniques that fasten the mining process.
2. The second objective focuses on developing a new efficient TIRPs mining algorithm that uses an unambiguous and robust definition of temporal relations.
3. And the third objective focuses on opening the door of TIRPs mining algorithms to multivariate time series or time-series data. To that end, it is needed to make explicit the embedded, ad-hoc pre-processes related to TIRPs mining algorithms while offering an efficient solution for the required pre-processing. Furthermore, this pre-processing should consider temporal restrictions established by the user.

1.6 Contributions of this thesis

The contributions of the present thesis are in the domain of Pattern Mining and Knowledge Discovery, being of particular relevance for the SPM and TIRPs mining fields. Each of the contributions satisfies the objective corresponding to the numbering. This thesis introduces:

1. A new efficient sequential pattern mining algorithm, called VEPRECO. VEPRECO has three main contributions that fasten the mining process: a vertical representation of patterns, pre-pruning strategies to avoid checking candidate patterns with no possibility of support, and common candidate selection policies that reduce the number of iterations performed by the algorithm. The contributions of the VEPRECO are:

- Memory efficient approach. To store space, in this thesis, a novel data structure has been proposed, based on an abstract data type composed of a collection of (key, value) pairs, instead of memory-consuming bitmaps proposed in CM-SPAM.
- A novel procedure to generate pattern candidates, which combines in a single step previous approaches. All sequential pattern mining algorithms explore the search space of sequential patterns by performing two basic operations called s-extensions and i-extensions. In the present thesis, these two operations were simplified and carried out in a single method, called c-extension, when it concerns the common candidates in the two extensions.
- Two different pre-pruning strategies for pruning the search space of patterns. Both take advantage of the sequence tree that stores the frequent children and extensions, which are necessary to generate the frequent candidates. One of them prunes the candidates based on the last item of the pattern, and the other prunes the candidates based on the last two items. VEPRECO uses the combination of the two pruning methods depending on the tree level.

The implementation of VEPRECO is available from the Bitbucket repository (see A).

2. A new efficient algorithm for mining TIRPs, called Vertical Frequent Time Interval-Related Pattern Mining (vertTIRP). vertTIRP combines an efficient representation of time-interval-related patterns, the temporal transitivity properties and a pairing strategy to speed up the mining process. This algorithm utilises a robust definition of the temporal relations and includes two constraints that enable the user to better express the types of TIRPs to be learnt. The vertTIRP algorithm has the following new features:

   - A robust revision of the temporal relations in order to manage noise with epsilon margins;
   - A revision of the transitivity property of the Allen's temporal relations when epsilon margins are used;
   - Two new constraints that enable the user to express the kinds of patterns to be learnt (user-controlled patterns), namely min-duration and min-gap;
   - A combination of a vertical representation of patterns and the transitive properties of the temporal relationships, which reduces the response time of TIRPs mining;

  - Pairing strategies that are specifically used to improve the efficiency when mining temporal relations.

  The implementation of vertTIRP is available from the Bitbucket repository (see B.1).

3. A new efficient algorithm, called TA4L, to transform multivariate time series into Lexicographical Symbolic Time Interval Sequences (LSTISs), that is, sequences ready to feed time-interval related pattern (TIRP) mining algorithms. The ultimate goal is to make explicit the embedded, ad-hoc pre-processes related to TIRPs mining algorithms while offering an efficient solution for the required pre-processing. On the one hand, TA4L divides the signals into segments based on time duration (instead of the often-used practice based on the number of samples), which allows the construction of consistent time intervals. Concatenation of intervals is controlled by a maximum time gap constraint that reinforces the generated time intervals' consistency. Moreover, different ways to parallelize the algorithm are explored and accompanied by efficient data structures to speed up the pre-processing cost. The contributions of the TA4L are:

  - The pre-processing phase for TIRPs mining is explicitly described and formulated.
  - The TA4L algorithm construct intervals based on the time duration (instead of a fixed number of samples). In so doing, the limits of the time-intervals are defined over the real values in the interval (not over the missing values resulting from the fixed-length).
  - A maximum gap constraint is used to decide whether two consecutive points could be considered to be part of the same event (i.e. belonging to the same time-interval) or do not.
  - Use of special data structure that allows performing intervals insertions into LSTISs efficiently.
  - Different approaches to apply parallelism to the process.

  The code of the TA4L algorithm will be publicly available from the Bitbucket repository (see C.1).

1.7 Thesis outline

This document has been structured into seven chapters.

- Chapter 1: Introduction. This chapter offers an overview of this thesis, its motivations, a specific introduction to each field (SPM, TIRPs mining and pre-processing), its objectives, contributions, and the outline of the individual chapters.

- Chapter 2: State of the art. This chapter presents a literature review on SPM, TIRPs mining and on the pre-processing of the TIRPs mining fields needed to understand this thesis.

- Chapter 3: An improved sequential pattern mining: VEPRECO. This chapter presents the sequential pattern mining problem formulation, the novel VEPRECO algorithm, the setup used in our experiments with its corresponding hypothesis formulation, the results, the discussion and the conclusions of this chapter.

- Chapter 4: Managing temporal relations in sequential patterns: vertTIRP. This chapter presents the TIRP problem formulation, the novel vertTIRP algorithm, and the setup used in our experiments with its corresponding hypothesis formulation, the results, the discussion and the conclusions of this chapter.

- Chapter 5: Data preparation as an external component of learning algorithms: TA4L. This chapter presents the pre-processing problem formulation, the novel TA4L algorithm, and the setup used in our experiments with its corresponding hypothesis formulation, the results, the discussion and the conclusions of this chapter.

- Chapter 6: Conclusions. This chapter draws some conclusions from this thesis and future work.

Chapter 2

State of the art

In this chapter, firstly, a literature review on the SPM field is presented; secondly, the TIRPs mining state of the art is reviewed; and finally, related work on the pre-processing of the TIRPs mining field is provided.

2.1 Sequential Pattern Mining

The algorithms of SPM can be divided into three main categories according to the taxonomy analysis provided in [30]: (i) the Apriori-based, (ii) pattern-growth, and (iii) early-pruning algorithms, or depending on the database model employed during the mining process [20]: (i) horizontal, (ii) vertical, or (iii) projected. There exist also algorithms that combine several of these techniques.

Apriori-Based algorithms are categorised by breadth-first search (Breadth-First Search strategy (BFS), level-wise), generate-and-test of candidates, and multiple scans of the database. Examples of such algorithms include Apriori [31], AprioriAll [32], GSP [17], and Eclat [33] algorithms. All of them use a horizontal database approach; that is, databases are structured around sequences. Apriori-Based algorithms use the candidate generation-and-test paradigm introduced in [5], which entails generating (k+1)-candidates from frequent k-patterns using joins and then test if they are frequent against the transaction database. Although the Apriori-Based algorithms are of the 90s, it seems that they are coming back into fashion these last years due to their facility to became parallelized [34].

Pattern-growth algorithms are categorised by using a compressed representation of data to save memory (but usually in dispense of time) [35, 36, 37], candidate pruning [38, 39, 11, 15], search space partitioning [38, 18, 40], tree projection [35, 38, 36, 41, 42], depth-first search traversal [16, 18, 40], suffix growth

[35, 43] and prefix growth [38, 16, 41]. The Prefix-projected Sequential PAtterN mining (PrefixSpan) [38] is a representative algorithm for this category. In that case, the database is a projected model, meaning that the database is divided into smaller databases (projections) based on the prefixes. The prefix is then grown for each database partition, and the process is repeated recursively, following a depth-first search (Depth-First Search strategy (DFS)) strategy. The major advantage of pattern growth's methods is that it does not generate and test any candidate sequences that do not exist in a projected database; instead, they focus the search on a restricted portion (projection) of the initial database. The disadvantage is the cost of constructing projected databases, where suffixes are largely repeated.

Early-Pruning algorithms count the support without scanning the Sequential DataBase (SDB) iteratively, which is usually possible with vertical database representation, such as in the case of SPAM [16], SPADE[38], A Lattice-based Sequential Pattern Mining Algorithm Using Bitmap Representation (bitSPADE) [44] and their last improvements CM-SPAM and Co-occurrence MAP in Sequential PAttern Discovery using Equivalence classes (CM-SPADE) [15]. Vertical databases structure the data required for the data mining process around patterns. Early-Pruning algorithms alleviate the problem of the generate-and-test technique by utilising a sort of position induction to prune candidate sequences very early in the mining process to avoid support counting as much as possible.

Currently, many new pattern mining subfields are increasingly created. Recent interesting works include [45], [46]. In [45], Dong et al. present the Topk-NSP+ algorithm, which mines the top-k useful Negative Sequential Patterns (i.e. missing a medical treatment), and in [47], an algorithm is put forward to mine non-occurring repetition behaviours. Negative sequential patterns contain both occurring and non-occurring items, such as AB¬C. Song et al. in [48] address the problem of mining high utility itemsets in multi-relational databases and propose two algorithms for star schema-based data warehouses. Min et al. [46] propose an algorithm for a new type of pattern that divides the alphabet into strong, medium, and weak parts, with the aid of wildcard gaps (i.e. "don't care" characters).

In recent years, research on SPM is also aimed to make an incremental and/or parallel version [49, 41, 50, 51]. For example, Haoxing Wen et al. recently [52] proposed an incremental parallel Apriori-based algorithm, Shadi AlZu'bi et al. [53] proposed an Apriori-based recommender system for user requirements, and Sudhakar Singh et al. [54] propose several MapReduce based Apriori algorithms. To know more about the current status of parallel SPM, the reader should have a look at [34], a recent survey of Parallel Sequential Pattern Mining.

2.1.1 Pruning in sequential pattern mining

The basic mechanism for pruning the search space is based on the Apriori property [5, 31], also known as a downward-closure or anti-monotonicity property. This property states that "All nonempty subsets of a frequent itemset must also be frequent", which means that if a sequence can not pass the minimum support test, all of its super sequences will also fail the test. For example, in SPAM [16], the downward-closure property is part of the algorithm, which discards the candidates that resulted infrequent in the siblings and children nodes of the tree, and extends only frequent patterns.

In SPADE [18] pruning, it is not so feasible as in SPAM because it implies testing whether all the sub-sequences of a frequent sequence are frequent, which imposes significant memory and time overheads. However, when an adaption of pruning of SPADE was applied to the SPAM algorithm [55], it resulted useful in terms of time but with a little more memory overhead.

To reduce the number of join operations, the CM-SPAM, CM-SPADE, and the CM-SPADE based improvements [15, 40, 56] utilise a structure called the Co-occurrence Map (CMAP) structure that stores all frequent 2-sequences. If the two last items of a grown pattern do not appear in a CMAP structure, the pattern is discarded without performing the 'join' operation.

Constraint-based pruning lets users specify regular expressions on patterns to be found by converting constraints to an automaton for pruning patterns. Constraints can be applied on the pattern (e.g. size constraints or regular expression constraints [57, 58]), on the cover set (e.g. minimum andor maximum frequency or entropy [59, 60]) or over the inclusion relation (e.g. minimum andor maximum gap [61, 62]). Some studies focus on the generality of the constraint-based framework supporting various constraints, such as [63] and [10] studies. Despite their excellent declarative aspects, most of these algorithms have scaling problems due to the huge size of their constraint networks, which relies on reified constraints and additional variables.

Another way to reduce the search space while mining sequential patterns is with the use of maximality pruning. However, these methods often do not exhaustively return the full set of frequent patterns, but condensed representations such as maximal [64, 65] or closed [66, 67] patterns.

2.1.2 How the contributions of this thesis fit within the context of the existing SPM literature?

The starting point of this thesis is the CM-SPAM algorithm, as being one of the best SPM algorithms nowadays. CM-SPAM (based on SPAM) uses a vertical database, in which patterns are modelled with bitmaps representing the locations of different items in the sequences. A bitmap of item I is an array of zeros and ones, where ones indicate the presence of item I and zeros indicate its absence. For large sequences with many items, this array will have a significant number of zeros (sparse arrays), and it will waste much space. In addition, sparse arrays also affect the mining time since many unnecessary positions will have to be traversed. For this reason, a new, more compressed data structure called DictMap is proposed. Dictmap is based on a dictionary of all the items in the sequences and a map to localize them composed by (key, value) pairs. The key represents the sequence where the item appears, while the value represents the item's location in the sequence. For example, a-1:10,25,4:10 means that item "a" is in sequence 1 and 4. And particularly in positions 10 and 25 of sequence one and in position 10 of sequence 4. This data structure is similar to the 'idLists' of SPADE, which store a list of input-sequence and event identifier pairs for each event (i.e. a-(1,10),(1,25),(4,10)), but it does it more compactly. The proposed DictMap data structure also store the support count and the pattern length information to avoid recounting them in every iteration.

The DictMap data structure enables a new candidate generation approach called c-extension (common extension), in which pattern candidates can be extended with a new item simultaneously in two ways: at the end of the sequence (sequence extension o s-extension, e.g. *ab* is extended to *abb*), or as part of the last component of the sequence (itemset extension or i-extension, e.g. *ab* is extended to $a(bb)$). This c-extension is designed with the aim of run-time reduction.

Regarding the pruning heuristics, it must be said that this work relies on the idea of CM-SPAM's pruning, which consists of building for each pattern of length one a list of patterns (or candidates) of length 1 succeeding this pattern. Therefore, each 1-length pattern must have a list of candidates for making sequence extensions and another list for making itemset extensions. The first difference between CM-SPAM and the pruning proposed in this thesis is that CM-SPAM uses two tables, one for each type of extension (to save the lists of frequent candidates for each pattern), while this work takes advantage of the sequence tree itself to save the candidates.

The second difference between CM-SPAM and the pruning technique presented in this thesis is that CM-SPAM's pruning is based on the last item of the pattern to prune candidates, while the present work takes into account both the last item in the pattern and the last two items in the pattern to decide which candidates will not result in frequent extensions and should therefore be ruled out. For example, if the *abc* pattern is extended with *d*, the CM-SPAM will look if *d* exists among the list of frequent s-candidates of *c*, while this work will also look if *d* exists among the s-candidates of *b*, which will allow us to rule out more candidates.

2.2 TIRP Mining

The first TIRPs mining algorithms were introduced in [26], [21]. In Kam's [26], TIRP "(A overlaps C) overlaps B", the temporal relation between A and B is unknown. In addition, the same TIRP can be written in different ways (for example, "A overlaps (C during B)"), which reduces the pattern count. To overcome these problems, Wu and Chen [21] proposed a new canonical representation of TIRPs which describes the temporal relationships between each pair of events that can be written in a single way. The term 'canonical' means that the events are sorted in some way (i.e. based on the start and end times). Wu and Chen also proposed the TPrefixSpan algorithm (based on PrefixSpan [38]) for discovering TIRPs.

IEMiner [2] extended Kam's representation of a vector of temporal relation counts and also adopted a canonical representation. However, the ambiguity of temporal relations remained when the size of the TIRP was greater than three. To manage these ambiguities, R. Moskovitch and Y. Shahar [68] represented TIRPs using a matrix (as in the present work) that stored temporal relations between each pair of events, and these were then sorted based on their starting and end times. The algorithm presented in [68] was called KarmaLego. In one of the later works [69], to accelerate the TIRPs mining process, the authors used a hash-table to store all the instances of the 2-sized TIRPs in the DharmaIndex. This variation of KarmaLego algorithm that uses a DharmaIndex is labelled as DharmaLego in the experiments of this work.

In order to deal with ambiguities arising from noisy data, Papapetrou et al. [28] extended Allen's temporal relations with an epsilon margin (as in the present work), to mine patterns and rules (unlike in the present work), following the approach used by the PrefixSpan mechanism with a hybrid of BFS and DFS strategies, called H-DFS. They also used a vertical representation approach to speed up TIRPs mining, as in the present work.

To increase the performance of the algorithm, Winarko and Roddick [29] introduced ARMADA (based on a MEMory Indexing approach for fast Sequential Pattern mining (MEMISP) [70]) to mine association rules with time interval data that grow patterns by adding suffixes (as in the present work). They used a maximum gap constraint (i.e. a maximum time between two events below which they are considered as belonging to the same pattern).

All of the algorithms that have been developed recently take into account the duration of the events, as in the present work. Recently, Harel O. D. and Moskovitch R. [71] have introduced the TIRPClo algorithm for mining frequent closed TIRPs, which allows discarding non-closed frequent TIRPs, and potentially meaningless frequent TIRPs, which have very large time durations between their time intervals, with a maximal gap time constraint. For example, in [72] Chen and al. proposed the addition of an end-point and end-time in the TIRP representation, which allow three new types of TIRPs to be mined: temporal patterns, occurrence-probabilistic temporal patterns, and duration-probabilistic temporal patterns. Chen et al. proposed the TPMiner and P-TPMiner algorithms (inspired by PrefixSpan) to mine these patterns, and presented an incremental version of TPMiner in [24].

2.2.1 How the contributions of this thesis fit within the context of the existing TIRP mining literature?

The algorithm presented in this thesis combines the different improvements of the previous works. For example, the generation of candidates of this algorithm is based on the vertical approach of SPAM.

The TIRPs definitions, the transitivity properties and the definitions of temporal relations initially were based on the Papapetrou et al. [28] and the R. Moskovitch and Y. Shahar's approaches[68, 4].

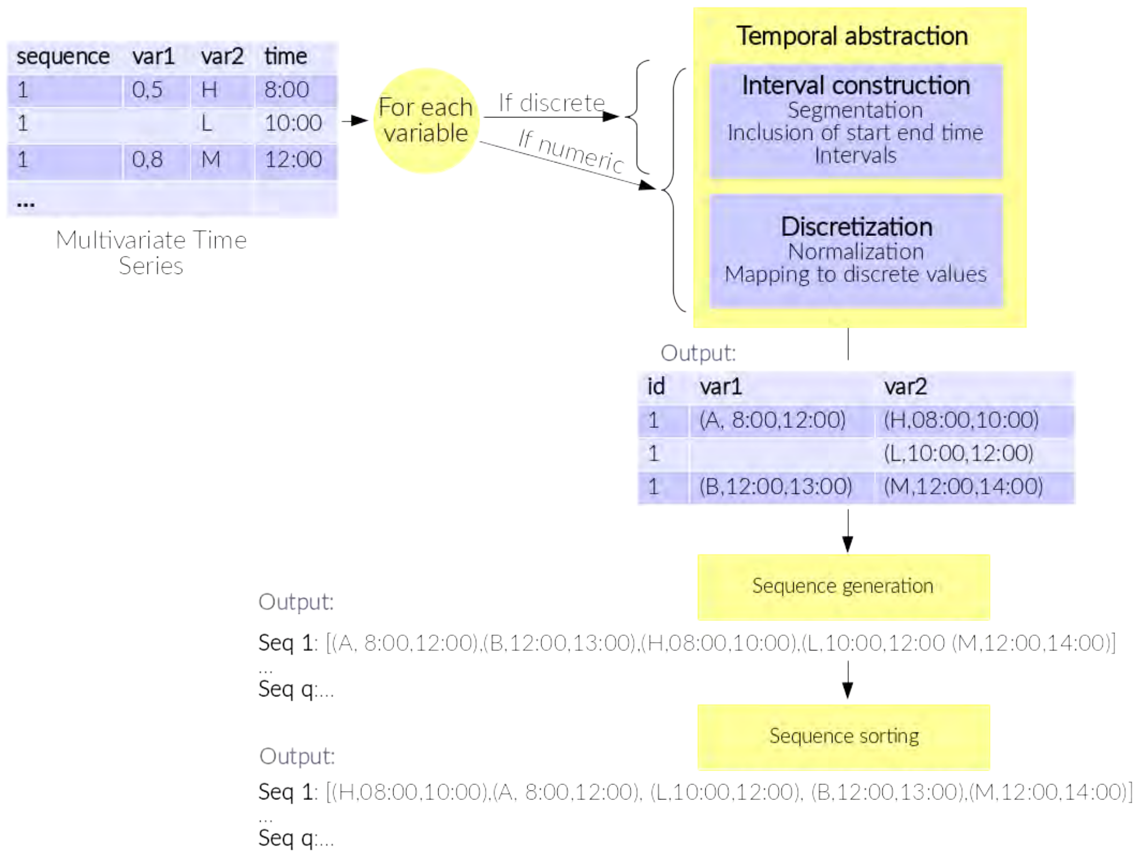
On the one hand, the work presented in this thesis is concerned with solving the ambiguities arising from the use of the epsilon's margins by presenting a robust revision of the temporal relations. This work also introduces a new table of transitivity for temporal relationships using epsilon and presents the pairing strategies between relationships for the first time.

Finally, regarding the time constraints in the TIRPs mining field, Moskovitch [68, 25, 4], Patel [2], and Papapetrou [28] used only two of them: the maximum gap C_{max_gap}, which determines the maximum distance between two events in a TIRP, and the maximum duration constraint $C_{max\_duration}$, which determines the maximum duration of each event interval. In the work presented in this thesis, two additional constraints for the TIRPs mining field are introduced, $C_{min\_duration}$ and C_{min_gap}.

2.3 Pre-processing

The literature on TIRPs mining points out that pre-processing is tightly related to the kind of data used to test the algorithms. When using synthetic datasets (as in [26, 29, 24]), data are usually generated with a synthetic data generator like [32], which have been modified to generate sorted symbolic time-intervals, and the pre-processing, in this case, is nonexistent.

On the other hand, when using real datasets as in [68, 25, 4, 73], or on both synthetic and real datasets (as in the present work) [21, 2, 72], different approaches to pre-processing to convert multivariate time series into LSTISs can be found: internal [74, 22, 28] or external [21, 2, 72] to the TIRPs mining algorithm. In general, the different steps followed in the literature are summarized in Figure 2.1.



sequence	var1	var2	time
1	0,5	H	8:00
1		L	10:00
1	0,8	M	12:00
...			

Output:

id	var1	var2
1	(A, 8:00,12:00)	(H,08:00,10:00)
1		(L,10:00,12:00)
1	(B,12:00,13:00)	(M,12:00,14:00)

Output:
Seq 1: [(A, 8:00,12:00),(B,12:00,13:00),(H,08:00,10:00),(L,10:00,12:00 (M,12:00,14:00)]
...
Seq q:...

Output:
Seq 1: [(H,08:00,10:00),(A, 8:00,12:00), (L,10:00,12:00), (B,12:00,13:00),(M,12:00,14:00)]
...
Seq q:...

Figure 2.1: Pre-processing step in the literature.

First, for each variable, a temporal abstraction is applied. Temporal abstraction is a conversion of a signal (e.g. blood pressure values) to an abstracted comprehensive to a human representation (e.g. "2 hours of high blood pressure") [75]. When the variable is numeric, temporal abstraction involves both interval construction and discretization steps. When the variable is already discrete, time-intervals are constructed.

Regarding the interval construction step, data is segmented into time intervals, where each interval should include its corresponding start and end times. Intervals are constructed usually performing bottom-up, top-down or sliding-window approaches [76]. In the sliding-window approach [77], a segment is grown until a specified error threshold is reached, where in each window, a linear approximation is performed. In the top-down approach, time series are repeatedly split according to the best splitting point from all considered points until the desired number of intervals is obtained [78]. The bottom-up approach [25] starts by segmenting series with small segments and then iteratively concatenating adjacent segments. In the present work, a bottom-up approach is used.

During the discretization step, first, the variable values are normalized. Afterwards, numeric values or the first derivative of values [79, 73] are converted into a discrete representation.

TA steps (interval construction and discretization) can be executed sequentially or at the same time with a mapping function [2, 21]. The output of the TA task is a set of symbolic time-intervals. Next, during the sequence construction step, all records belonging to the same sequence are grouped together. And, finally, during the sequence sorting step, sequences are ordered according to the TIRP's mining algorithm criteria: some TIRPs mining algorithms require time-intervals to be sorted by their end time [29]. Others, by their start and end times [2]. Others sort them by start and end times and lexicographically [68] (as done in the present work).

This is the process that, in general, is followed in the literature. But not always the starting point of the algorithm is the same. Examples of the full process, from multivariate time series to LSTISs, are [2, 21, 68, 25, 22]. Sometimes the starting point is an interval-based data [72], and in other cases, the starting point is already ordered sequences of time-intervals [26, 29].

2.3.1 Temporal abstraction in the TIRPs mining field

Knowledge-Based Temporal Abstraction (Knowledge-Based Temporal Abstraction (KBTA)) [75] method is the most widely used discretization method in the literature [2, 68, 25, 72]. BFS belongs to the group of supervised discretization methods, where data is discretized following the knowledge of an expert in the corresponding field. This fact makes the output of these methods significant that lead to the successful data interpretation (e.g. [21, 4, 73]). The problem is when the expert knowledge is lacking, or when the discretization is performed not for the interpretation of the time series, but rather for the performance of other tasks, possibly less intuitive for human experts, such as classification, clustering, and prediction. For example, these two works [4, 73] are focused on this problem. Temporal Discretization for Classification (Temporal Discretization for Classification (TD4C)) [73] is another supervised discretization method geared towards the enhancement of classification accuracy, which determines the cutoffs that will best discriminate among classes through the distribution of their states.

Alternative, non supervised methods could be used, such as Equal Width Discretization (Equal Width Discretization (EWD)) [80] or Symbolic Aggregate ApproXimation (Symbolic Aggregate approXimation (SAX))[70]. EWD involves sorting the observed values of a continuous feature and dividing the range of observed values for the variable into k equally sized bins, where k is a parameter supplied by the user. SAX divides the original time-series into equally sized frames, computes the mean for each frame, and assigns a symbol to each calculated mean, based on the statistical table (of a normal continuous random variable) and on the size of the vocabulary. The size of the vocabulary and the number of frames are the parameters supplied by the user.

In [4] the KBTA, Equal Width Discretization (EWD), and Symbolic Aggregate ApproXimation (SAX) methods have been compared, and it was found that discretization using SAX led to better accuracy on classification than using the EWD. While KBTA cut-off definitions, when available, were superior to both in terms of accuracy. TD4C was also compared to the EWD, KBTA, and SAX methods in [73]. In general, some configuration of the TD4C discretization method and the KBTA method outperformed the other methods, but SAX was always very close to the TD4C methods and sometimes even outperforms some configurations of the TD4C.

2.3.2 How the contributions of this thesis fit within the context of the existing TA for TIRPs mining literature?

Except for TD4C [73], in the literature on TIRPs mining, the pre-processing from multivariate time series to LSTISs is usually mentioned in a brief paragraph of the experimental setup or algorithm section of the article. Unlike TD4C, the algorithm proposed in this thesis is an unsupervised abstraction method that is transparent to discretization, which means that the function to convert a numeric value to a discrete one is a parameter of the algorithm.

The algorithm presented in this work is provided with a duration constraint, designed to do not miss any TIRP without neglecting efficiency while using the maximum gap constraint to decide whether two consecutive points or intervals belong to the same interval or not. The maximum gap constraint is not a novelty. It has been widely used in sequential pattern mining [81, 82] and also in TIRPs mining approaches [29, 83]. However, this is the first time it has been adapted in an algorithm to create LSTISs.

Chapter 3

An improved sequential pattern mining: VEPRECO

This chapter presents the work related to the fulfilment of the first objective of this thesis: a new algorithm for improving sequential pattern mining called VEPRECO.

3.1 Problem statement

Given a set of sequential records (called sequences) representing a sequential database (SDB) of events (or items) and a minimum support threshold min_sup, the problem of sequential pattern mining is to discover the set of all frequent sequences S in the given sequence database SDB at the given min_sup.

To formally state the problem and the algorithms that follow, some notation is introduced.

Definition 3.1.1. The set of n unique items or events is $I = \{i_1, i_2, ..., i_n\}$.

Definition 3.1.2. An itemset is a nonempty, unordered collection of items which are accessed at the same time, $IS = (i_1 i_2 ... i_m)$ (e.g., (bc)), where i_j is an item or event in I.

Definition 3.1.3. A sequence is an ordered list of itemsets, denoted as $S = \langle IS_1 IS_2 ... IS_k \rangle$. A sequence with k items is called a k-sequence.

For example, the sequence $S = \langle (bc)(de) \rangle$ is a 2-sequence composed of two itemsets, (bc) and (de). When the set has only one element, the parentheses are not necessary. Thus, the sequence $S = \langle a(bc)e(de) \rangle$ is a 4-sequence, in which the first and third elements are itemsets with a single item, a and e correspondingly.

Regarding the order, if an itemset IS_i occurs before an itemset IS_j, it is denoted as $IS_i \leq IS_j$. Analogously for items, $i_i \leq i_j$. In the case of items that occur at the same time (i.e. itemset), the lexicographical annotation is used (e.g. (abc)).

Definition 3.1.4. A sequence $S_a = \langle IS_1^a IS_2^a ... IS_{k_a}^a \rangle$ is said to be contained in another sequence $S_b = \langle IS_1^b IS_2^b ... IS_{k_b}^b \rangle$ (denoted as $S_a \sqsubset S_b$) if and only if there exist integers $1 \leq i_1 < i_2 < ... < i_{k_j} \leq k_b$ such that $IS_1^a \subseteq IS_{i_1}^b$, $IS_2^a \subseteq IS_{i_2}^b$, ..., $IS_n^a \subseteq IS_{i_{k_j}}^b$.

For example, the sequence $\langle a(bc) \rangle$ is contained in sequence $\langle a(bc)e(de) \rangle$, while the sequence $\langle abc \rangle$ it is not. If a sequence S_a is contained in a sequence S_b, S_a is said to be a subsequence of S_b.

Definition 3.1.5. A sequence $S_a = \langle IS_1^a IS_2^a ... IS_{k_a}^a \rangle$ is a prefix of a sequence $S_b = \langle IS_1^b IS_2^b ... IS_{k_b}^b \rangle$, $\forall k_a < k_b$, iff $IS_1^a = IS_1^b, IS_2^a = IS_2^b, ...IS_{k_a}^a = B_{k_a}$.

The frequency or support of a frequent sequence S (or pattern P) can be relative or absolute.

Definition 3.1.6. Relative support of S (denoted as $\sigma(S)$) is the total number of sequences of which S is a subsequence in the given database SDB divided by the total number of sequences in SDB.

Definition 3.1.7. Absolute support of S (denoted as support(S)) is the total number of sequences in SDB of which S is a subsequence.

3.2 The VEPRECO algorithm

The new VEPRECO algorithm is provided in Algorithm 1. The algorithm's inputs include the sequential database and the *min_support* value that determines which patterns considered frequent.

The first step of the algorithm consists of building a vertical representation (using DictMaps) of frequent patterns with length one (line 1). Then, the algorithm discovers all the frequent patterns of length two (lines 3-14). Therefore the algorithm performs a BFS (level headed) up to the third level of the tree where all frequent 2-length patterns are discovered. From the third level up to the rest of the levels, DFS (branch headed) have been applied (lines 15-17).

An example of the sequence tree T generated is provided in Figure 3.1. The root of the tree is labelled with "{}". The level of a node is defined by one + the number of connections between the node and the root. The root level is 1. The level is shown in Gray in Figure 3.1. Each sequence in T can be considered either a sequence-extended sequence or an itemset-extended sequence. A sequence-extended

```
Algorithm 1: VEPRECO
input  : SDB: sequential database
         min_sup: required for the discovered patterns
output:
1  $T2$ = vertical-lenght1(SDB)
2  $T3$ = {}
3  for each $j\_node \in T2$ do
4      for each $k\_node \in T2$ do
5          $j\_dictmap$ = getDictmap($j\_node$)
6          $k\_dictmap$ = getDictmap($k\_node$)
7          if $k > j$ then
8              $new\_s$, $new\_i$ = c-extend($j\_dictmap, k\_dictmap$)
9          else
10             $new\_s$ = s-extend($j\_dictmap, k\_dictmap$)
11         if $new\_s$.support $> min\_sup$ then
12             add_child($j\_node$, Node($new\_s$), $T3$)
13         if $new\_i$.support $> min\_sup$ then
14             add_child($j\_node$, Node($new\_i$), $T3$)
15 for each $N \in T3$ do
16     $s\_candidates$, $i\_candidates$ = getCandidates($T2$)
17     searchPatterns($N$, $s\_candidates$, $i\_candidates$, $min\_sup$)
```

sequence is a sequence generated by adding a single item to the end of its parent's sequence in T. This operation is called s-extension; the item to be added to the end is named s-candidate, and the new pattern s-extended pattern. An itemset-extended sequence is a sequence generated by adding an item to the last itemset in the parent's sequence, such that the item is greater than any item in that last itemset. This candidate generation operation is called i-extension, the item i-candidate, and the resulting pattern i-extended pattern.

Algorithm 1 tests all combinations of length 1 and 2 patterns if they are frequent. The extensions are done in order: aa, ab, (ab), ac, (ac), ba, bb, bc, (bc). Note that the nonsense extensions such as (aa), which is to check if a happens to itself at the same time, are not checked. Neither the algorithm checks the extension (ba), having checked the extension (ab). To avoid these cases, it is a question of looking at whether the symbol with which the algorithm extends is larger than the previous one or not. If it is larger, it is called the c-extend (line 8) with which it gets both the s-extension (e.g. ab) and the i-extension (e.g. (ab)), otherwise it only performs

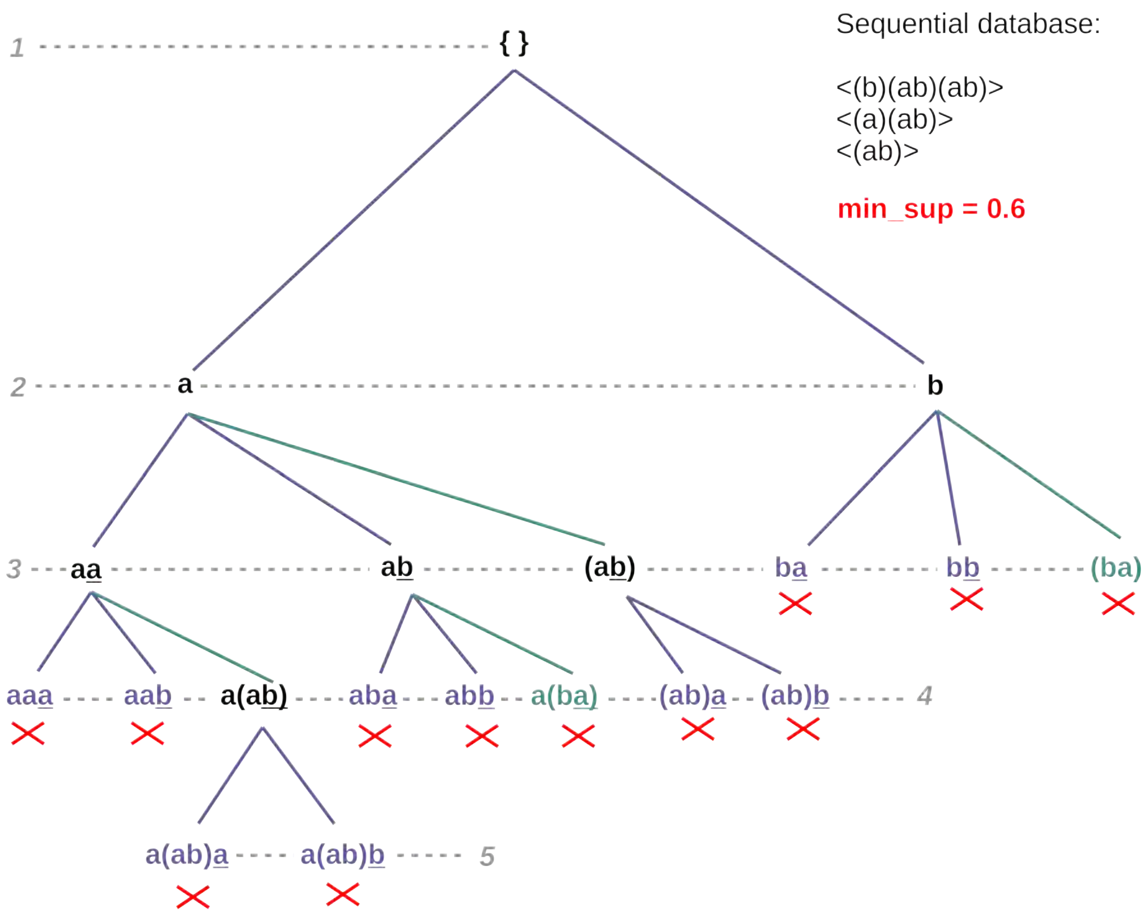


Figure 3.1: A sequence tree constructed from a sequential database using minimum relative support of 0.6. An item to be added to the end of the sequence is underlined. Itemset extensions are marked in turquoise, while sequencing extensions in violet and frequent patterns in black.

the s-extension (e.g. ba, line 10).

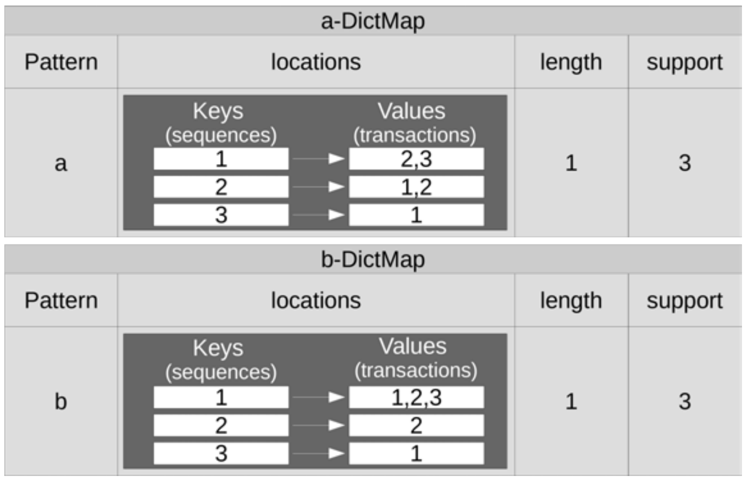
The recursive function *searchPatterns* (lines 14 and 17 of the Algorithm 1), which builds the tree from the third level down is explained in the Algorithm 2. The algorithm extends the DictMap P of the node N with the candidates (S and I), and saves the frequent candidates in S_{new} and I_{new}, and it's respective DictMaps in S_{pats} and I_{pats} (line 11). Patterns are extended with one function or another (c-extend, s-extend, i-extend) depending on the type of extension (c, s, or i). Finally, the algorithm prunes infrequent candidates of the S_{pats} and I_{pats} (lines 13 and 16), and recursively extends the new frequent extensions (lines 13 and 16).

```
Algorithm 2: searchPatterns
  input: N: a tree node with a bitmap to be extended
         S: a list of s-candidates
         I: a list of i-candidates
         min_sup: required for the discovered patterns
1 $S_{new}$ = []; $I_{new}$ = []; $S_{pats}$ = []; $I_{pats}$ = []
2 $P$ = getDictmap($N$)
3 Group $S$ and $I$ *candidates* by type of extension
4 for each $C$, *extension* $\in$ *candidates* do
5     if extension=="c" then
6         $P_s$, $P_i$ = c-extend(P, c)
7     else if extension=="s" then
8         $P_s$, $P_i$ = s-extend(P, c)
9     else
10        $P_s$, $P_i$ = i-extend(P, c)
11    If $P_s$'s (or $P_i$'s) support $>= min\_sup$, append it to $S_{pats}$ (or $I_{pats}$) and
      to the $N$'s childs. And append a candidate $C$ to $S_{new}$ (or to $I_{new}$).
12 for each $P_{new} \in S_{pats}$ do
13     $s$, $i$ = getCandidates($S_{new}$, $S_{new}$>$P_{new}$)
14     searchPatterns(Node($P_{new}$), $s$, $i$, min_support)
15 for each $P_{new} \in I_{pats}$ do
16     $s$, $i$ = getCandidates($S_{new}$, $I_{new}$>$P_{new}$)
17     searchPatterns(Node($P_{new}$), $s$, $i$, min_support)
```

3.2.1 DictMap

In Figure 3.2, a DictMap example is provided. It stores each sequence id as a key entry, and the value is a sorted set of all transaction ids where the item appears. For best efficiency, the list of transactions should have quick access to the first and last transactions. It also stores information regarding the support of the item and the sequence length, saving it from traversing the dictionary to do the support count and the sequence to discover the sequence length.

The DictMap model allow to redefine the s-extension and i-extension operations in a more efficient way that precedent SPM approaches. In the next subsections DictMap.pattern, DictMap.locations, DictMap.length and DictMap.support, are used when the pattern, locations, length and support of a DictMap are referred to. For clarity, the sequence is used to identify the DictMap. For example *a.locations* is used to refer to the locations of the $a - DictMap$.



a-DictMap			
Pattern	locations	length	support
a	Keys (sequences) → Values (transactions): 1 → 2,3; 2 → 1,2; 3 → 1	1	3

b-DictMap			
Pattern	locations	length	support
b	Keys (sequences) → Values (transactions): 1 → 1,2,3; 2 → 2; 3 → 1	1	3

Figure 3.2: DictMap

3.2.1.1 s-extension

The s-extensions are performed according to the Algorithm 3. The algorithm searches transactions in F that are equal to or greater than the first transaction in P of the corresponding sequence *sid*. Therefore the minimum transaction min_tid to search in F is established, that is, the first transaction in P plus one (See line 4). The DictMap model allows reducing the iterations by comparing the first occurrence of the pattern to be extended with the last occurrence of the pattern's extension. For example, to extend a Dictmap A with a Dictmap B, A has to happen before B. If A's first occurrence is in transaction 10, and the last occurrence of B is before or in transaction 10, for sure that there will be no pattern of AB, and the algorithm will not iterate through the transactions of this sequence. Line 9 of Algorithm 3 filter such cases. Another direct case is if the last occurrence of B is at transaction 11, where it is possible to deduce that there is only one matching of AB, reflected on lines 5-7 of Algorithm 3. The iterative case (lines 8-14) is when the last transaction in F is equal to or greater than min_tid, in which case the algorithm traverses the F.locations to find all transactions that are equal to or greater than min_tid.

3.2.1.2 i-extension

Algorithm 4 details how i-extensions are generated. For each sequence *sid* in the pattern P, the corresponding *sid* is searched in $F.locations$. If it does not exists, the algorithm does not search anything in this sequence (line 3). Otherwise, it performs an intersection of transactions to find patterns that happen in the same transaction. The fact that this intersection is tackled as an intersection of sorted arrays problem and that the DictMap model allows storing only the transactions

Algorithm 3: s-extend

```
input  : $P$: DictMap of length k
         $F$: DictMap of length one
output: $new\_P$: a sequence extension of $P$ with $F$
1  create a new DictMap, $new\_P$ of length (k+1), which sequence is a
     sequence extension of $P$ with $F$ (e.g. $ab$)
2  for all key-value pairs $(sid, tids_p) \in P.locations$ do
3      if $sid \in F.locations$ then
4          Let $min\_tid$ be the first transaction of $tids_p$ plus 1
5          if last transaction of $F.locations == min\_tid$ then
6              $new\_P.support = new\_P.support + 1$
7              Add $(sid, min\_tid)$ pair to $new\_P.locations$
8          else
9              if last transaction of $F.locations > min\_tid$ then
10                 for $tid_s \in F.locations$ of sequence sid do
11                     if $tid_s >= min\_tid$ then
12                         if not $sid \in new\_P.locations$ then
13                             $new\_P.support = new\_P.support + 1$
14                         Add $(sid, tid_s)$ to $new\_P.locations$
15 return $new\_P$
```

that contain a pattern, this version of the i-extend performs fewer iterations than intersecting arrays storing all the transactions like in the case of Bitmaps. A Bitmap store zero if the pattern does not appear in the transaction and one otherwise. An intersection of transactions allows finding patterns that happen together because they will have the same transaction.

3.2.2 Pre-pruning strategies

VEPRECO has two strategies to carry out the pre-pruning: one of them is related to consider only the last item of the k-length pattern to be extended to select the frequent extensions (e.g. if the pattern is ab, only b is considered), and the other one is particular to selecting the frequent extensions based on the last two items (e.g. if the pattern is ab, a and b are considered).

VEPRECO applies pruning in two places, in line 16 of the VEPRECO algorithm (see Algorithm 1) to prune candidates of length 2, and in lines 13 and 16 of the

```
Algorithm 4: i-extend
input  : $P$: DictMap of length k
         $F$: DictMap of length one
output: $new\_P$: an itemset extension of $P$ with $F$
1  create a new DictMap, $new\_P$ of length (k+1), which sequence is an
     itemset extension of $P$ with $F$ (e.g. $(ab)$)
2  for all key-value pairs $(sid, tids_p) \in P.locations$ do
3      if $sid \in F.locations$ then
4          Let $tids_i$ be transactions of $sid$ in $F.locations$
5          $i_p = 0$ ;                                  // index to traverse $tids_p$
6          $i_i = 0$ ;                                  // index to traverse $tids_i$
7          while $i_p < len(tids_p)$ and $i_i < len(tids_i)$ do
8              if $tids_p[i_p] == tids_i[i_i]$ then
9                  if not $sid \in new\_P.locations$ then
10                     $new\_P.support = new\_P.support + 1$
11                 Add $(sid, tids_i[i_i])$ to $new\_P.locations$
12                 $i_p = i_p + 1$ $i_i = i_i + 1$
13             else if $tids_p[i_p] < tids_i[i_i]$ then
14                 $i_p = i_p + 1$
15             else
16                 $i_i = i_i + 1$
17 return $new\_P$
```

searchPatterns algorithm (see Algorithm 2) to prune candidates of length greater than 2. Any pruning strategy can be applied in both places. To prune candidates of length 2, VEPRECO applies pruning based on the last two items (section 3.2.2.2), while to prune candidates of length greater than 2, VEPRECO applies pruning based on the last item (section 3.2.2.2).

3.2.2.1 Pruning based on the last item

The pruning strategy is based on the $s-extensions$ and $i-extensions$. In the case of s-extending (the same for i-extending), a sequential pattern P of length k with a frequent pattern F of length 1, if the last item l in P does not have F in s-candidates (the same for i-candidates), that is, l does not have F as children in the search tree[1], then the pattern resulting from the extension of P with F will be

[1]Remember that one length patterns are generated at the beginning of Algorithm 1 (BFS strategy until level two)

infrequent, and an extension P with F is avoided. Instead of having a co-occurrence table for $s-extensions$ and $i-extensions$ (like in CM-SPAM) to prune candidates, nodes of the VEPRECO's tree store s and i candidates (see Figure 3.3).

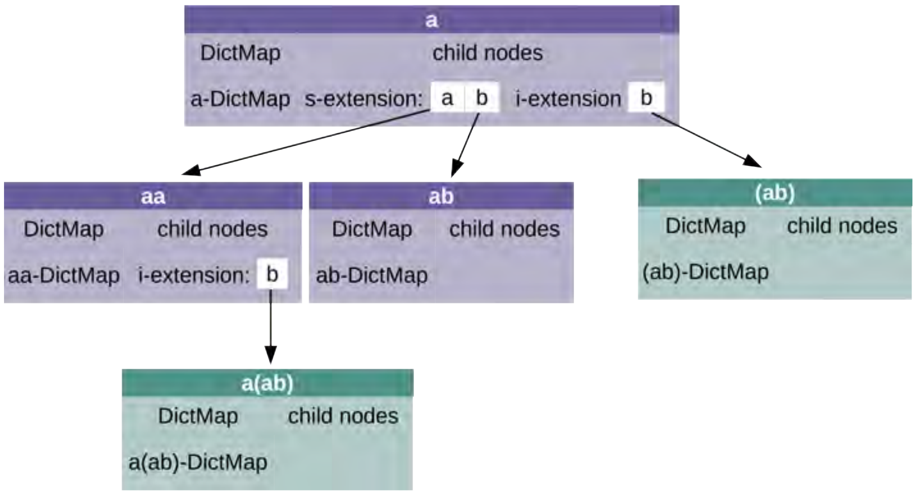
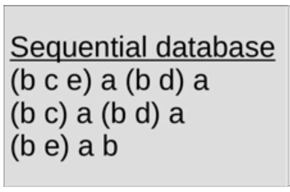


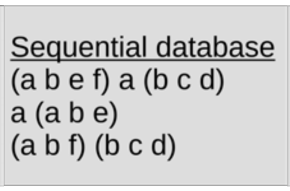
Figure 3.3: Tree's nodes of VEPRECO

3.2.2.2 Pruning based on the last two items.



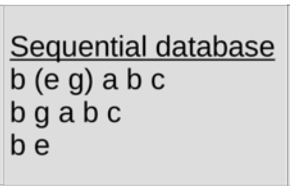
Sequential database
(b c e) a (b d) a
(b c) a (b d) a
(b e) a b

Figure 3.4: SDB 1



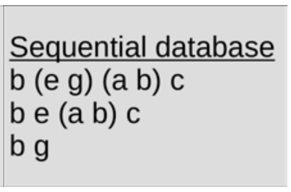
Sequential database
(a b e f) a (b c d)
a (a b e)
(a b f) (b c d)

Figure 3.5: SDB 2



Sequential database
b (e g) a b c
b g a b c
b e

Figure 3.6: SDB 3



Sequential database
b (e g) (a b) c
b e (a b) c
b g

Figure 3.7: SDB 4

This pruning strategy aims to select s-extension and i-extension candidates based on the frequent extensions of the last two items that form the pattern. In so doing, there are different combinations, depending on the kind of extension to be performed and the kind of extension performed previously to the pattern.

Selection of i-extension candidates for an s-extended pattern. Given an s-extended pattern $P_s = \langle ab \rangle$, a set of i-candidates for $\langle ab \rangle$ is the intersection of s-candidates of a with the i-candidates of b.

For example, consider a sequential database from Figure 3.4 and a $min\_support$ $= 0.6$. A set of s-candidates for an item a are items {a,b,d}, and a set of i-candidates of b are items {c,d,e}, an intersection of which is d. Therefore the i-extension of $\langle ab \rangle$ with d results in the frequent extension $\langle a(bd) \rangle$. Note that in the case of CM-SPAM, a set of i-candidates for $\langle ab \rangle$ are all the i-candidates of item b={c,e,d}. Therefore CM-SPAM would perform two extra extensions ($\langle a(bc) \rangle$ and $\langle a(be) \rangle$), which would result to be infrequent. As it is possible to observe, with this improvement, all the i-extensions of b that happened before a but not after b are eliminated.

Selection of i-candidats for an i-extended pattern. Given an i-extended pattern $P_i = \langle (ab) \rangle$, a set of i-candidates for $\langle (ab) \rangle$ is the intersection of i-candidates of a with the i-candidates of b.

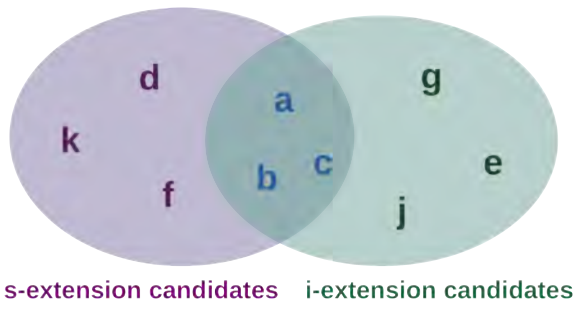
For example, consider a sequential database from Figure 3.5 and a *min_support* = 0.6. A set of i-candidates for an item a are items {b,e,f}, and a set of i-candidates of b are items {c,d,e,f}, an intersection of which is e,f. Therefore the i-extension of $\langle (ab) \rangle$ with e (equivalently with f) results in the frequent extension $\langle (abe) \rangle$ (equivalently $\langle (abf) \rangle$). Note that in the case of CM-SPAM, a set of i-candidates for $\langle (ab) \rangle$, are all the i-candidates of item b={c,d,e,f}. Therefore CM-SPAM would perform two extra extensions ($\langle (abc) \rangle$ and $\langle (abd) \rangle$), that would result to be infrequent. Therefore, VEPRECO eliminates all the i-extensions of b, which happened without a in the timeline with this improvement.

Selection of s-candidates. A set of s-candidates for the any-extended pattern (i.e. ab or (ab)) is the intersection of s-candidates of a with the s-candidates of b.

For example consider a sequential database from Figure 3.6, a *min_support* = 0.6 and an extension $P_s = \langle ab \rangle$. A set of s-candidates for an item a are items {b,c}, and a set of s-candidates of b are items {a,b,c,e,g}, an intersection of which is b,c. In this case, the s-extension of $\langle ab \rangle$ with b results in a not frequent extension, but the s-extension of $\langle ab \rangle$ with c is a frequent extension $\langle abc \rangle$. Note that in the case of CM-SPAM, a set of s-candidates for $\langle ab \rangle$ are all the s-candidates of item b={a,b,c,e,g}. Therefore CM-SPAM would performed four extra extensions ($\langle aba \rangle$, $\langle abb \rangle$, $\langle abg \rangle$ and $\langle abe \rangle$ extensions), that would result to be infrequent.

And finally consider a sequential database from Figure 3.7, a *min_support* = 0.6 and an extension $P_i = \langle (ab) \rangle$. A set of s-candidates for an item a are items {c}, and a set of s-candidates of b are items {a,b,c,e,g}, an intersection of which is c. In this case, the s-extension of $\langle (ab) \rangle$ with c is a frequent extension $\langle (ab)c \rangle$. Note that in the case of CM-SPAM, a set of s-candidates for $\langle (ab) \rangle$ are all the s-candidates of item b={a,b,c,e,g}. Therefore CM-SPAM would performed four extra extensions ($\langle (ab)a \rangle$, $\langle (ab)b \rangle$, $\langle (ab)e \rangle$ and $\langle (ab)g \rangle$ extensions), that would result to be infrequent.

As it can be observed, with this improvement, all the s-extensions that happened before a in the timeline are eliminated.



Previous approaches	
a,b,c,d,f,k	s-extend
a,b,c,e,g,j	i-extend

Using c-extend	
d,f,k	s-extend
e,g,j	i-extend
a,b,c	c-extend

Figure 3.8: c-extend function

3.2.3 A novel operation: c-extend

The main idea of common extension or c-extension is to take advantage of the loop for traversing the pattern's locations, constructing the s-extension and i-extension at the same time. Given a frequent pattern P of length k and a frequent pattern F of length 1, where $F \in i - candidates_P$ and $F \in s - candidates_P$, a c-extension of P with F is a pair of (P s-extended with F of length (k+1), P i-extended with F of length (k+1)). Therefore the c-extend is applied to the common candidates (the a, b, c candidates represented in blue in the Figure 3.8).

Figure 3.9 shows the result of applying c-extend to a and b [2]. In this figure, it is worthy to observe how the locations of the Dictmaps evolve. The change from the previous iteration has been highlighted in bold. Imagine a pointer traversing the locations of the a-DictMap and another one traversing the locations of the b-DictMap. The starting point is sequence 1. The two pointers point to the first position. Since 2 (first item in a-DictMap) is greater than 1 (first item in b-DictMap), the pointer of b is advanced. Now it points to 2. Since 2 is common in a-DictMap and b-DictMap, 2 is added to the (ab)-DictMap and the pointers of a and b are advanced. Since 3 is common in a-DictMap and b-DictMap, 3 is added to the (ab)-DictMap. Moreover, since 3 is greater than the first value in a-DictMap (2), the algorithm finishes constructing the ab-DictMap for sequence 1, copying all the values from the b-DictMap after 3. Then it advances the pointers of a and b and add 4 to the (ab)-DictMap because it is common. Next, the algorithm continues

[2]Note that the sequential database is distinct from the previous examples

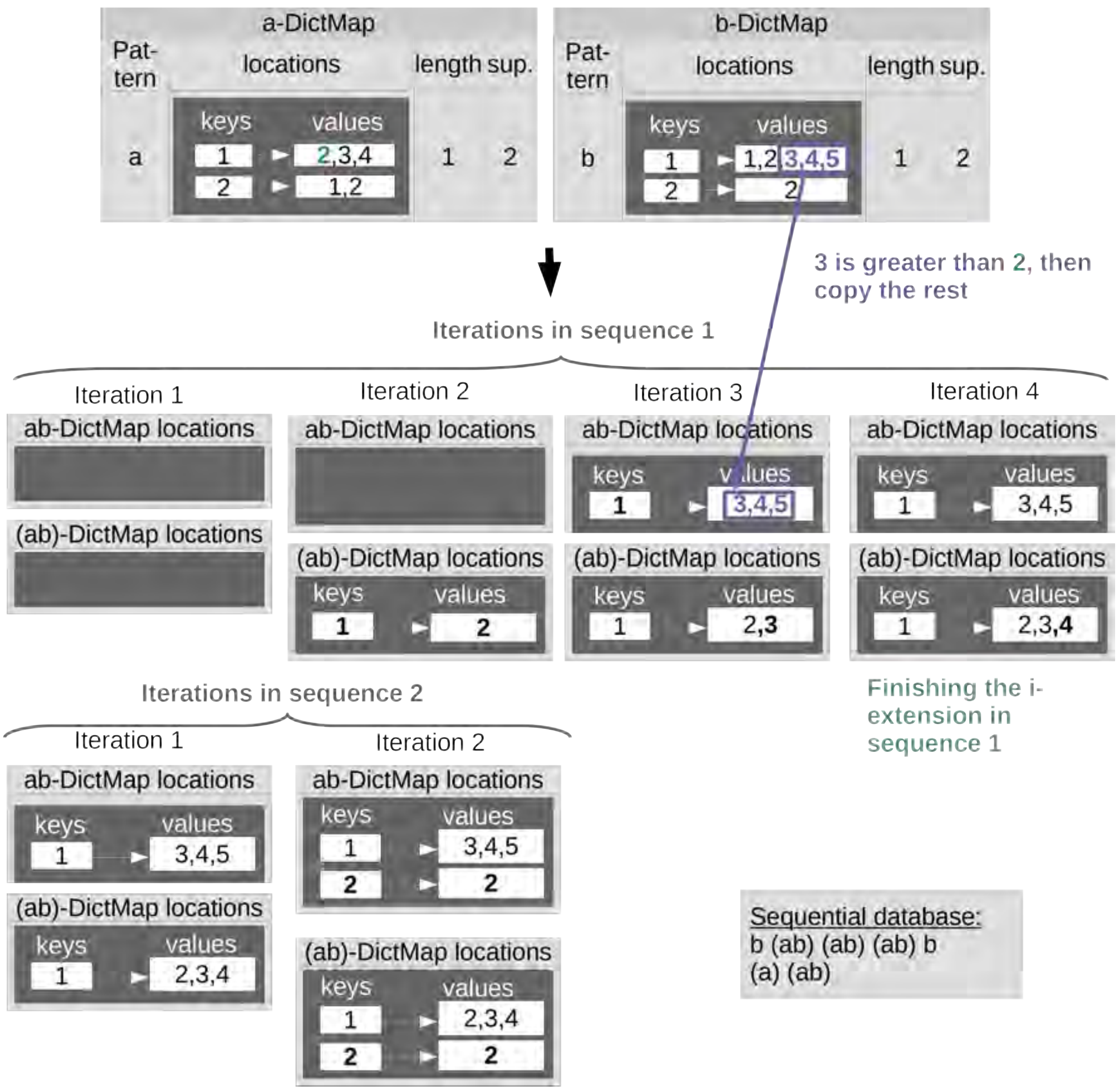


Figure 3.9: c-extension

with sequence 2. The two pointers point to the first position. Since the first item in a-DictMap (1) is smaller than the first item in b-DictMap (2), it advances the pointer of a. Since 2 is common in a-DictMap and b-DictMap, 2 is added to the (ab)-DictMap. Finally, since 2 is greater than the first item in a-DictMap, which is 1, the algorithm also adds 2 to the ab-DictMap.

Without using the c-extend function, the algorithm would have to perform six iterations to obtain the ab-DictMap using the s-extend function, and then six more iterations to obtain the (ab)-DictMap using the i-extend function. In total, it would have done 12 iterations instead of 6 it does with the c-extend function.

3.3 Experimental Evaluation methodology

In this section, we test VEPRECO with the aim to answer the following research question: How can we improve the efficiency in terms of time and memory of a sequential pattern mining algorithm with vertical pattern representation such as CM-SPAM?

Secondary research questions:

- What is the impact on time and memory if we perform the s-extensions and i-extensions in a single function?

- What impact does it have on time and memory if we use a dictionary-based structure to represent the data?

- What is the impact on time and memory if instead of looking at the last element of the pattern, we look at the last two elements of the pattern when pruning?

3.3.1 Experimental platform for VEPRECO

The VEPRECO algorithm presented in this thesis were implemented in Python 3.7.6. The experiments were carried out on a machine with Intel(R) Core(TM) i7-4790 CPU 3.6 GHz 1 physical processor, four cores, eight threads, 16GB of RAM and Titan Xp graphics card from Nvidia.

3.3.2 Datasets

The experiments were conducted with synthetic and real datasets.

3.3.2.1 Synthetic datasets for VEPRECO

To better explore the efficiency of the sequential pattern mining algorithm presented in this thesis, numerous synthetic datasets are generated (using an IBMGenerator [84]) in which the following factors were considered: (i) Number of sequences (NS); (ii) Sequences length (SL); (iii) Transaction length (TL); (iv) Number of items or vocabulary size (NI). Six synthetic datasets have been generated, the characteristics of which are summarized in Table 3.1.

3.3.2.2 Real datasets for VEPRECO

Five public datasets from the SPMF library [85] were used to test the Sequential Pattern Mining algorithms presented in this dissertation:

SDB name	NS	SL	TL	NI
data.NS_1000.SL_5.TL_5.NI_100	1000	5	5	100
data.NS_10000.SL_5.TL_5.NI_100	10000	5	5	100
data.NS_20000.SL_5.TL_5.NI_100	20000	5	5	100
data.NS_10000.SL_2.TL_2.NI_1000	10000	2	2	1000
data.NS_10000.SL_4.TL_4.NI_1000	10000	4	4	1000
data.NS_10000.SL_8.TL_8.NI_1000	10000	8	8	1000

Table 3.1: Synthetic datasets.

dataset	NS	SL	TL	NI
MSNBC	31790	3.82	2.48	17
BIBLE	36369	4.41	4.47	13905
FIFA	20450	5.70	5.71	2990
LEVIATHAN	5834	5.43	5.53	9025
SIGN	730	7.09	7.28	267

Table 3.2: Public dataset

- BIBLE This dataset is a conversion of the Bible into a sequence database (each word is an item).
- MSNBC a dataset of click-stream data from the MSNBC website, converted from original data from the UCI repository. The shortest sequences have been removed to keep only 31,790 sequences.
- FIFA Clickstream data from the website of FIFA World Cup 98.
- LEVIATHAN This dataset is a conversion of the novel Leviathan by Thomas Hobbes (1651) as a sequence database (each word is an item).
- SIGN a dataset of sign language utterance containing approximately 800 sequences.

In order to validate the correct operation of pruning strategies and the c-extend function, these datasets have been modified to have more than one item per transaction. These datasets are characterised in Table 3.2 with the same factors that the synthetic datasets.

3.3.3 Experimental setup

Several experimental scenarios were defined to test the memory and time improvements achieved by the VEPRECO algorithm:

- Comparison with the state of the art algorithm CM-SPAM. In this experiment, the state of the art approach, CM-SPAM, have been compared to VEPRECO. The hypothesis is: The runtime and memory usage of VEPRECO is lower than for CM-SPAM. This experiment has been conducted with the first 10000 sequences of the a Clickstream data from the website of FIFA World Cup 98 (FIFA) and a conversion of the Bible into a sequence database dataset (BIBLE) datasets in order to reduce the execution time and memory consumption of CM-SPAM. [3]

- Analysis of the tree traversing and pruning strategies In this experiment, VEP-RECO, with the full set of pruning strategies, is compared with VEPRECO_NO _P, that is, VEP-RECO without pruning strategies, and with ablations of VEPRECO in which only a single pruning strategy is considered: configuration VEPRECO _P1, for the pruning based on the last item, and VEPRECO_P2 for the pruning based on the last two items. In addition, it has been tried to prune based on the last item in the second level of the tree, and to prune based on the last two items in the rest of the levels labelled as VEPRECO_P1_P2. The hypothesis is: The runtime and memory usage of VEPRECO_NO_P is greater than for VEPRECO, or any other pruning variety of VEPRECO, especially in the case of datasets with large NS and similar SL and TL, since in this way, the intersections between the candidates carry more weight.

- Analysis of the c extension. In this experiment, VEPRECO is compared with an ablation of VEPRECO in which the c-extension has been removed and replaced by a sequential application of s-extension and i-extension configurations which have been labelled as VEPRECO_NO_C. The hypothesis is: The runtime of VEPRECO is lower than for VEPRECO_NO_C, in datasets with a large number of common candidates. This fact may occur in the case of datasets with large NS, large NI, and similar and large SL and TL.

3.4 Results

Regarding the comparison with the state-of-the-art algorithm, results are shown in Figure 3.10 and Figure 3.11 for the synthetic and real-world datasets correspondingly.

The results demonstrate that VEPRECO consumes less time and memory than CM-SPAM in all the configurations. In the case of memory usage, the difference between the two algorithms is more noticeable for large NS.For example, in synthetic

[3]When the experiment was run with all FIFA and BIBLE dataset's sequences, CM-SPAM did not finish or pop up for memory.

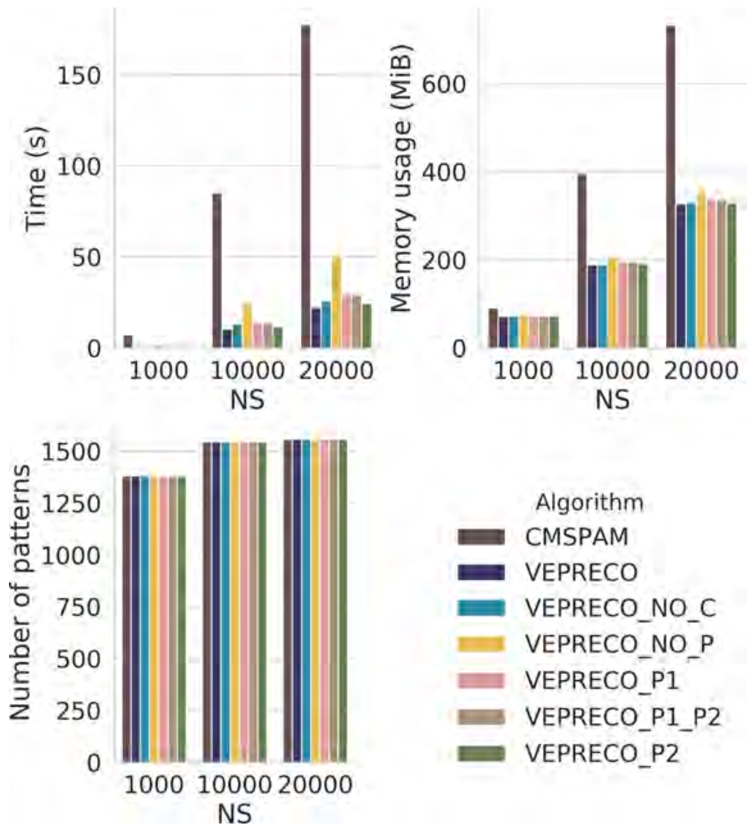


Figure 3.10: VEPRECO's and CM-SPAM's results for synthetic datasets.

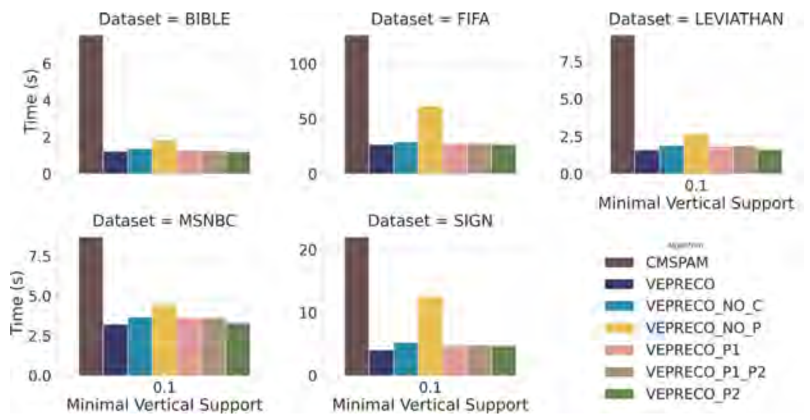
datasets, for NS 20000, CM-SPAM consumes 732.8819 MiB, while VEPRECO consumes 326.528 MiB, representing 56 per cent less than the baseline version. In real-world datasets happen the same; for instance, for the BIBLE dataset, CM-SPAM consumes around 1918.077 MiB, while VEPRECO consumes around 88.88 MiB.

In the case of runtime, the difference between the two algorithms is significant for all NS values. For example, in synthetic datasets, for NS 1000, the runtime of the CM-SPAM algorithm is 6.998 seconds, while the runtime of the VEPRECO algorithm is 0.78694 seconds. For the BIBLE dataset, the runtime of CM-SPAM is 7.5436 seconds, while VEPRECO is 1.227 seconds.

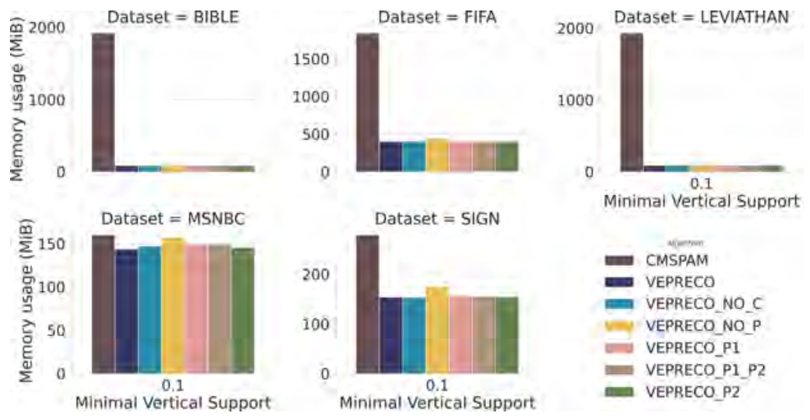
In the case of the discovered patterns, as expected, the results match, which means that VEPRECO and all its ablations work properly.

Regarding tree traversing and pruning strategies, results are shown in Figure 3.12 and Figure 3.13 for the synthetic and real-world datasets correspondingly.

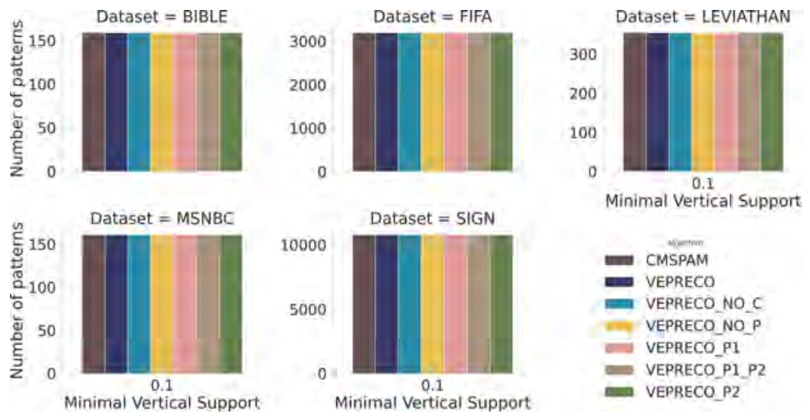
In the case of both synthetic and real-world datasets, not using any pruning strategy involves more time and memory consumption. The difference between the two versions is more relevant for large NS, SL and TL, especially in the time. For



(a) Time vs NS



(b) Memory vs NS



(c) Number of patterns vs NS

Figure 3.11: VEPRECO's and CM-SPAM's results for real-world datasets.

example, in synthetic datasets, for NS 20000, the runtime of the VEPRECO_NO_P is 52.883 seconds, and memory usage is 358.003 MiB, while VEPRECO's runtime is

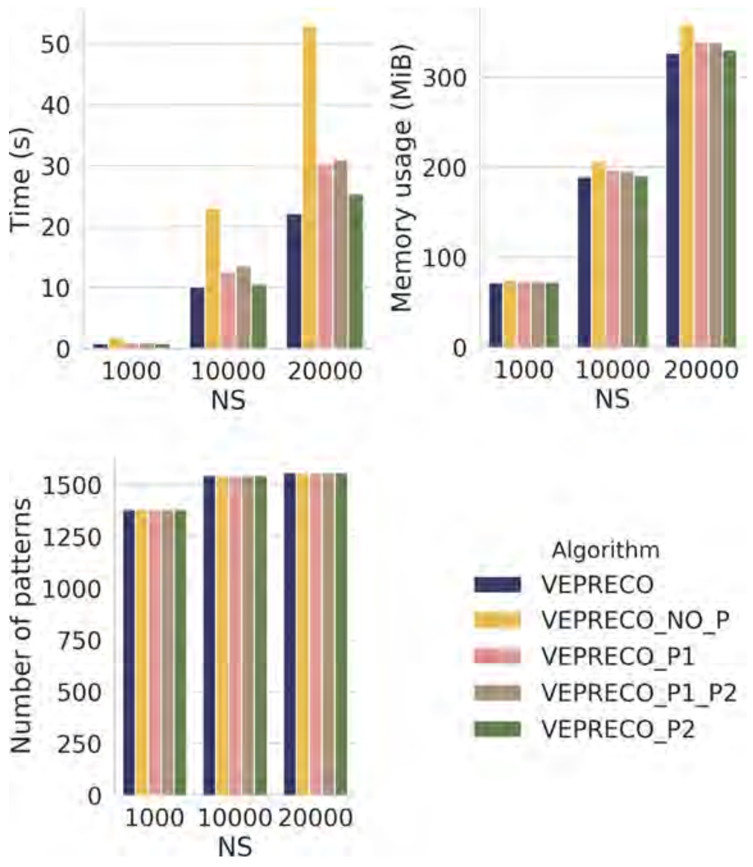
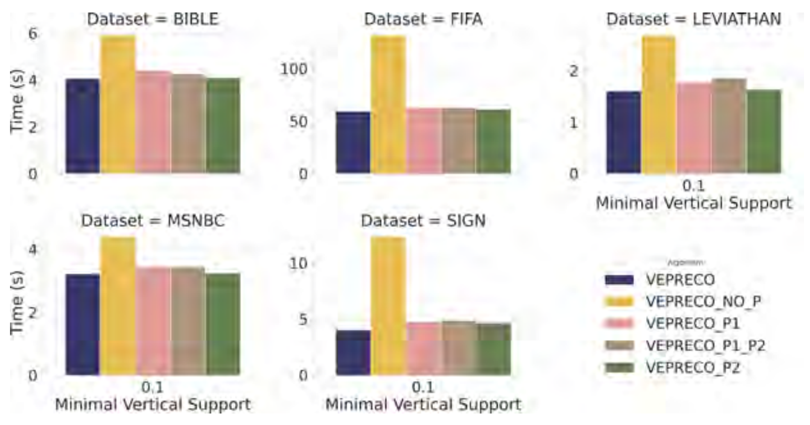


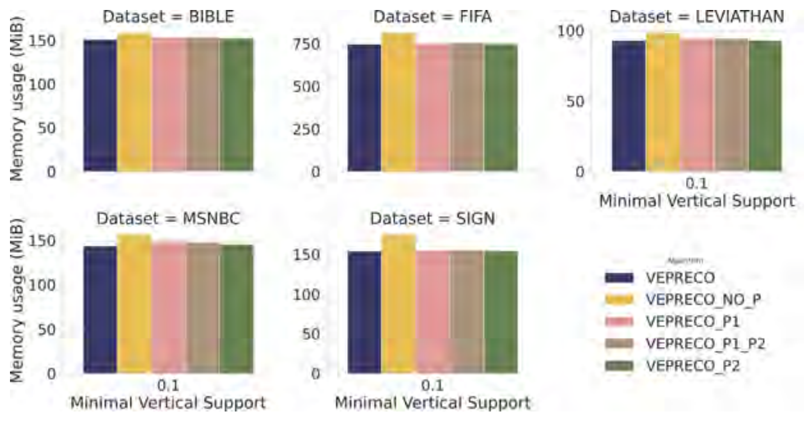
Figure 3.12: Synthetic datasets: How tree traversing and pruning strategies affect the time and memory of the algorithm.

22.1389 seconds, and it consumes 326.528 MiB. In real-world datasets happen the same. For example, for the FIFA dataset, VEPRECO_NO_P's runtime is 131.870 seconds and memory usage around 814 MiB, while VEPRECO's runtime is 59.312 seconds, which consumes around 746 MiB.

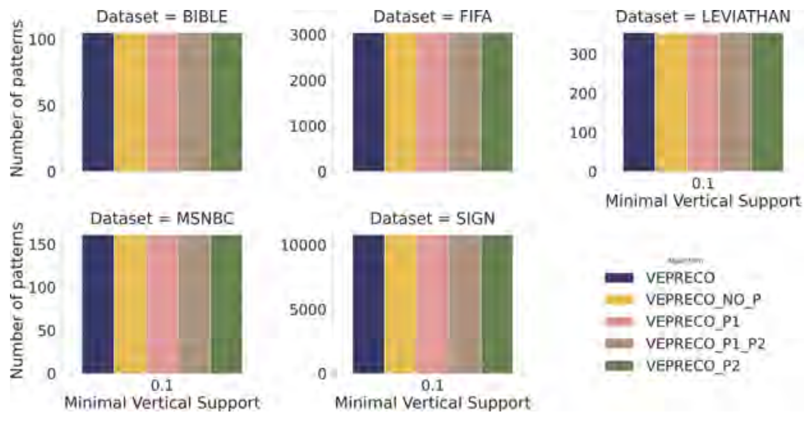
We can also observe that VEPRECO_P2 consumes less time and memory than VEPRECO_P1 and than VEPRECO_P1_P2. For example, in the case of synthetic datasets, for NS 20000, VEPRECO_P2's runtime is 25.332 seconds, and the memory usage is 329.961 MiB. VEPRECO_P1's runtime is 30.301 seconds, and memory usage is 339.109 MiB. And VEPRECO_P1_P2's runtime is 30.891 seconds, and memory usage is 338.0567 MiB. This slight difference between the two strategies also can be observed in the case of real datasets. For the FIFA dataset, the runtime of VEPRECO_P2 is 61.464 seconds, and memory usage is 748.193 MiB. The runtime of VEPRECO_P1 is 63.0247 seconds, and the memory usage is 750.0299 MiB. And the runtime of VEPRECO_P1_P2 is 62.869 seconds, and the memory usage is 755.282 MiB. Although VEPRECO and VEPRECO_P2 have very



(a) Time vs NS



(b) Memory vs NS



(c) Number of patterns vs NS

Figure 3.13: Real-world datasets: How tree traversing and pruning strategies affect time and memory of the algorithm

similar times and memories, VEPRECO is faster than VEPRECO_P2 for large NS, SL or TL. For example, in synthetic datasets, for NS 20000, VEPRECO_P2's runtime is 25.332 seconds, and the memory usage is 329.961 MiB, while VEPRECO's runtime is 22.1389 seconds and the memory usage is 326.5288 MiB. For the FIFA dataset, the runtime of VEPRECO_P2 is 61.464 seconds, and memory usage is 748.193 MiB, while the runtime of VEPRECO is 59.312 seconds and memory usage is 746.680 MiB.

Finally, regarding the discovered patterns, as expected, the results match for all versions. Therefore, it was possible to verify that all the pruning strategies prune the candidates properly. It does not discard frequent candidates; otherwise, the number of discovered patterns of that strategy would be smaller.

Regarding the analysis of the c extension, results are shown in Figure 3.14 and Figure 3.15 for the synthetic and real-world datasets correspondingly.

In synthetic and real-world datasets, the VEPRECO_NO_C version is slower than the VEPRECO version and consumes slightly more memory, especially for large SL and TL.

For example, in synthetic datasets, for SL and TL 8, VEPRECO_NO_C's runtime is 38.632 seconds, and the memory usage is 145.667 MiB, while VEPRECO's runtime is 30.907 seconds and the memory usage is 145.128 MiB.

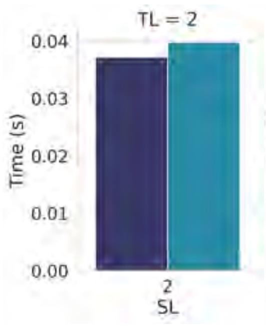
For the FIFA dataset, the runtime of VEPRECO_NO_C is 63.668 seconds, and memory usage is 749.581 MiB, while the runtime of VEPRECO is 59.312 seconds and memory usage is 746.680 MiB.

Finally, as expected, the results match for both versions regarding the discovered patterns, which confirms that the VEPRECO works properly.

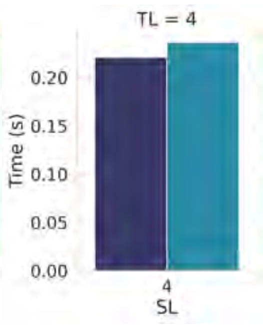
3.5 Discussion

VEPRECO consumes less time and memory than the CM-SPAM algorithm in all the configurations, and the difference is noticeable for both time and memory consumption. Therefore, the DictMap structure and the corresponding changes in the extensions have significantly reduced both runtime and memory usage.

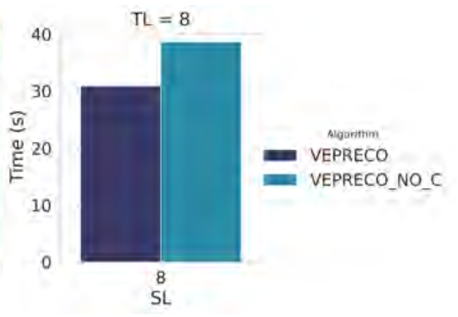
Regarding the pruning strategies, they provide savings in both time and memory. Pruning based on the last two items is more efficient than pruning based on the last item because the intersections discard more candidates than without them. Although combining the two pruning methods resulted in the best option, the type of pruning applied on the third tree's level (2-length patterns) also affects the algorithm's efficiency. If pruning based on the last item is applied to prune candidates of length 2, and pruning based on the last two items to prune the rest of the candidates



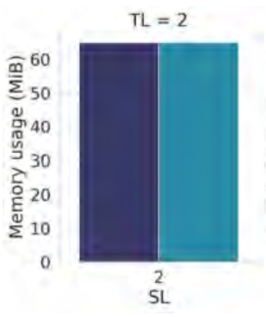
(a) Time vs SL=2



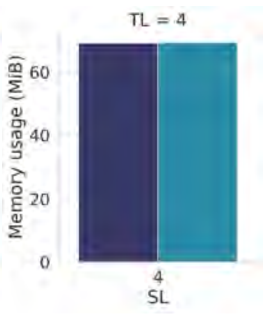
(b) Time vs SL=4



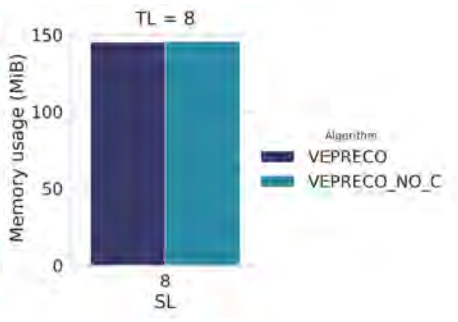
(c) Time vs SL=8



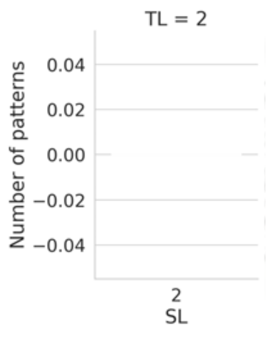
(d) Mem. vs SL=2



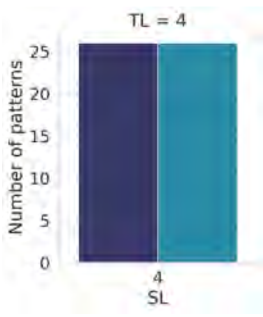
(e) Mem. vs SL=4



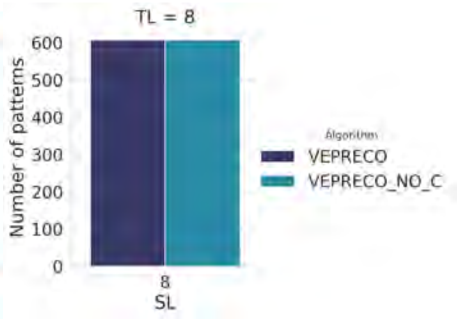
(f) Mem. vs SL=8



(g) Pat. vs SL=2



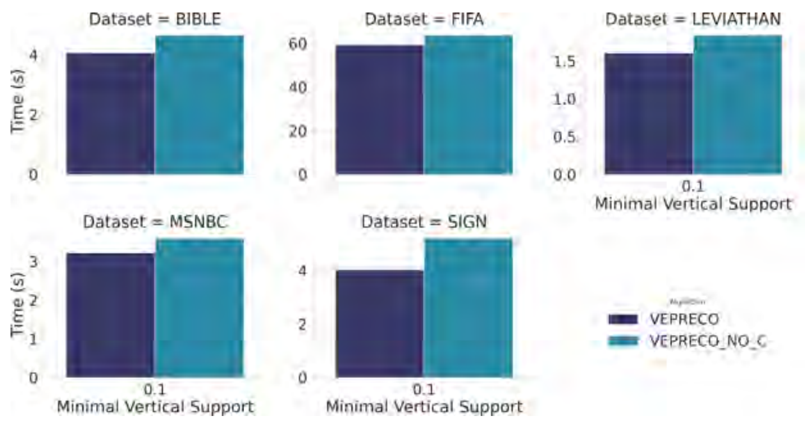
(h) Pat. vs SL=4



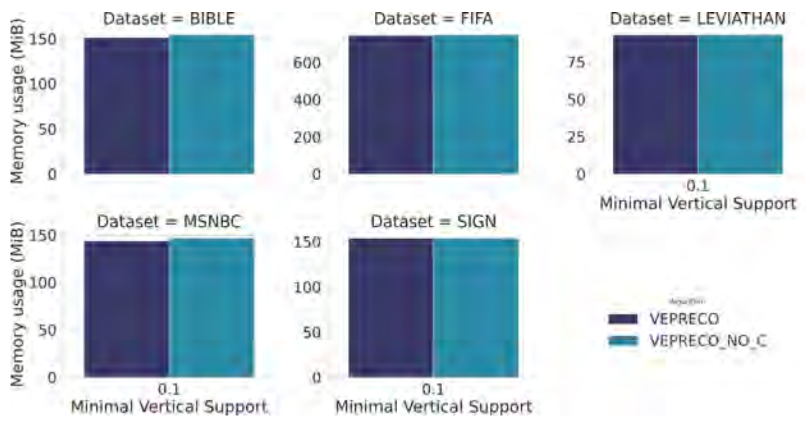
(i) Pat. vs SL=8

Figure 3.14: Synthetic datasets: How the c extension affect time and memory, where Mem. is memory usage, and Pat. is the number of patterns

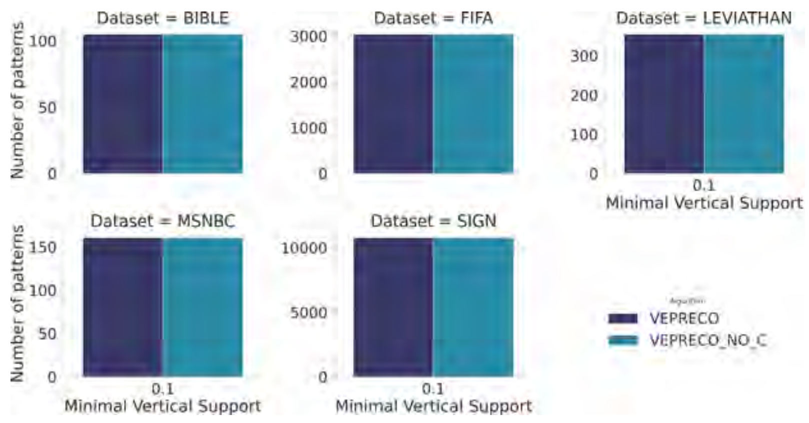
(i.e. VEPRECO_P1_P2's pruning strategy) is less efficient than doing it the other way around. That is, pruning based on the last two items to prune candidates of length 2, and pruning based on the last item to prune the rest of the candidates (i.e. VEPRECO's pruning strategy). VEPRECO's pruning strategy is more effective for large NS and NI, with similar SL and TL. The c extension is handy when the dataset has many common candidates (i.e. the same candidate exists in s and i-extension). This is more likely to happen when SL and TL have similar values. In



(a) Time vs SL



(b) Memory vs SL



(c) Number of patterns vs SL

Figure 3.15: Real-world datasets: How the c extension affect time and memory

these cases, VEPRECO is significantly faster than ablation of VEPRECO, labelled as VEPRECO_NO_C, in which the c-extension has been removed and replaced by a sequential application of s-extension and i-extension.

Regarding the discovered patterns, the results match in all versions, which means that all the VEPRECO's versions work correctly.

3.6 Summary

Regarding a new efficient SPM algorithm, in this thesis, a new VEPRECO algorithm is presented, which is based on the CM-SPAM. The algorithm has three main contributions. The first is related to how patterns are represented in the database, based on (key, value) pairs with fast access to the first and last values for localising items in sequences instead of sparse bitmaps of items. The second contribution is related to pruning strategies when generating candidates in the search tree. Two strategies are provided: One of them prunes the candidates based on the last item of the pattern, and the other prunes the candidates based on the last two items. VEPRECO uses the combination of the two pruning methods depending on the tree level. Both strategies take advantage of the sequence tree to store the frequent children and extensions, which are necessary to generate the frequent candidates. The third contribution is related to the novel c-extension, which considers sequential and itemset extensions at once. VEPRECO has been evaluated using both synthetic and real-world datasets in terms of runtime and memory usage. Experiments carried out demonstrated VEPRECO be more effective than the state of the art algorithm CM-SPAM in terms of time and memory, and than all the ablations of VEPRECO, in all the configurations. Pruning enhancements help save time on large datasets with the similar transaction and sequence lengths. The c-extension improvement is helpful when there are many common candidates, which usually happens when the length of sequences and the length of transactions is similar, and the number of different itemsets is low.

Chapter 4

Managing temporal information in sequential patterns: vertTIRP

This chapter addresses the second objective of this thesis, which entails the inclusion of starting and ending times as characteristics of events in sequences that enable finding new patterns that deal with temporal relations with a new algorithm called vertTIRP.

4.1 Problem statement

Given a dataset of sequences of time interval events, each annotated with a symbol value A,B,C,D..., start and end time points, and a set of user constraints $\mathcal{C}$, the problem consists of discovering frequent TIRPs, i.e. TIRPs with a frequency equal to or higher than a given threshold $min\_support$, which fulfil the constraints $\mathcal{C}$. To formalise this problem, several forms of notation are introduced in the following.

4.1.1 Time interval sequence

Definition 4.1.1. A symbolic time interval sequence, $IS = \langle I^1, I^2, ..., I^n \rangle$ represents a sequence of symbolic time intervals I^i.

Definition 4.1.2. Symbolic time interval. A symbolic time interval I is a triple $I =< s, e, sym >$, composed of a start time (s), an end time (e), and a symbol (sym).

Time scales for the start and end times are application-dependent. The symbol is assumed to belong to a given alphabet Σ, which is also application-dependent.

In the definitions below, I.s, I.e, and I.sym are used when the start time, end time and symbol of an interval are referred to. For clarity, the symbol is used to identify the symbolic time interval. For example, a symbolic time interval A is represented as <_,_,A>.

The vertTIRP algorithm requires some kind of sorting for time intervals, based on relational operators. Rather than comparing two time intervals with an exact operator, this thesis looks for an approximate (epsilon) approach (following [4]). To achieve this, some additional definitions are required.

Definition 4.1.3. Quasi-equal "$=^{\epsilon}$". Two time-points t^i and t^j are quasi-equal, $t^i =^{\epsilon} t^j$, if $|t^i - t^j| \leq \epsilon$.

Definition 4.1.4. Precedes "$<^{\epsilon}$". Time-point t^i precedes time-point t^j , $t^i <^{\epsilon} t^j$, if $t^j - t^i > \epsilon$.

The symbols $=^{\epsilon}$ and $<^{\epsilon}$ refer to imprecision derived from noise data and enable sorting time interval events into a sequence.

Definition 4.1.5. A lexicographical symbolic time interval sequence $LSTIS$ is a symbolic time interval sequence, IS=$\langle I^1, I^2, ..., I^n\rangle$, which is sorted in order of the start and end times using the relations $<^{\epsilon}$, $=^{\epsilon}$ and the symbols with a lexicographical order, that is: $\forall I^i, I^j \in IS(i < j)$: $((I^i_{.s} <^{\epsilon} I^j_{.s}) \vee (I^i_{.s} =^{\epsilon} I^j_{.s} \wedge I^i_{.e} <^{\epsilon} I^j_{.e}) \vee (I^i_{.s} =^{\epsilon} I^j_{.s} \wedge I^i_{.e} =^{\epsilon} I^j_{.e} \wedge I^i_{.sym} < I^j_{.sym}))$.

For example, $\langle\langle< 8:00, 10:00, A >, < 8:00, 12:00, B >, < 11:00, 13:00, C >\rangle, \langle< 14:00, 17:00, C >, < 16:00, 17:00, A >, < 16:00.18:00, B >\rangle, \langle< 10:00, 13:00, C >, < 11:00, 14:00, B >, < 12:00, 13:00, A >, < 14;00, 15:00, C >\rangle\rangle$ is a LSTISs that contains three sequences, the first of which $\langle< 8:00, 10:00, A >, < 8:00, 12:00, B >, < 11:00, 13:00, C >\rangle$ is composed of three symbolic time intervals. Figure 4.1 shows a graphical representation of this example. Note that Figure 4.1 is a database representation and it is an input of the TIRPs mining algorithms. When vertTIRP algorithm is applied to this database, the output of the algorithm is a set of patterns that fulfils the user constraints.

4.1.2 Robust temporal relations

The temporal relations used in vertTIRP overcome the limitations of previous works in terms of the ambiguities arising from the use of the epsilon approach. They can also be customised according to user constraints.

The temporal relations considered in this work are the 13 temporal relations introduced by Allen [27] and revised by KarmaLego [4] based on the $=^{\epsilon}$ and $<^{\epsilon}$

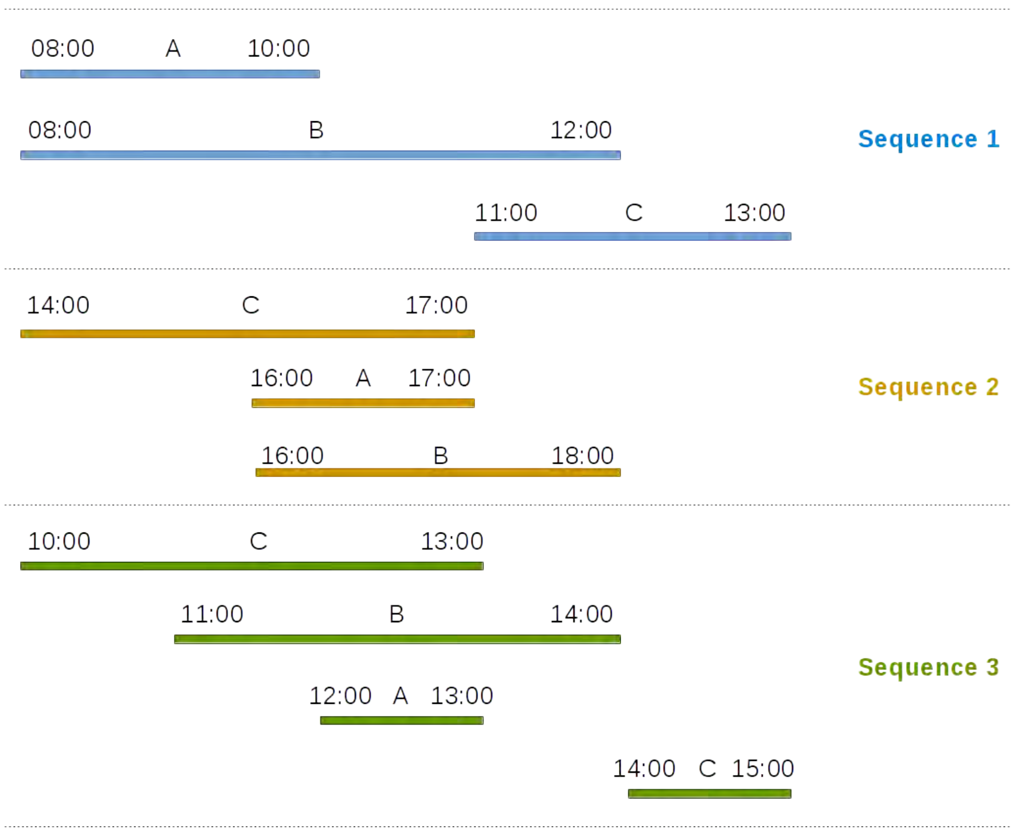


Figure 4.1: Graphical representation of an example

relations and given an *epsilon* value. However, in this approach, two ambiguities remain: overlaps is confused with finished by (i.e. when the end time of A is very similar to the end time of B), and contains (i.e. when the end time of A is greater than the end time of B), while meets is confused with starts, finished by and equals in case where the duration of A and B events is greater than ϵ and there exists an overlapping area that corresponds to the ϵ value. (see Figure 4.2). For example, in the Figure 4.2 if A is "a sore throat" and B is a "Fever" event, for a doctor it should not be the same that the fever begins after the sore throat has passed, that the fever ends with the sore throat. Perhaps in the first case the fever is the indicator of a symptom of a new disease, while in the second case, it could be considered that the fever is related to the sore throat. It is important to have robust definitions of relations to avoid confusions.

In [68, 25, 4], when $\epsilon > 0$, the left contains relation also becomes necessary, since some temporal relations cannot be identified otherwise (see the example in Figure 4.3).

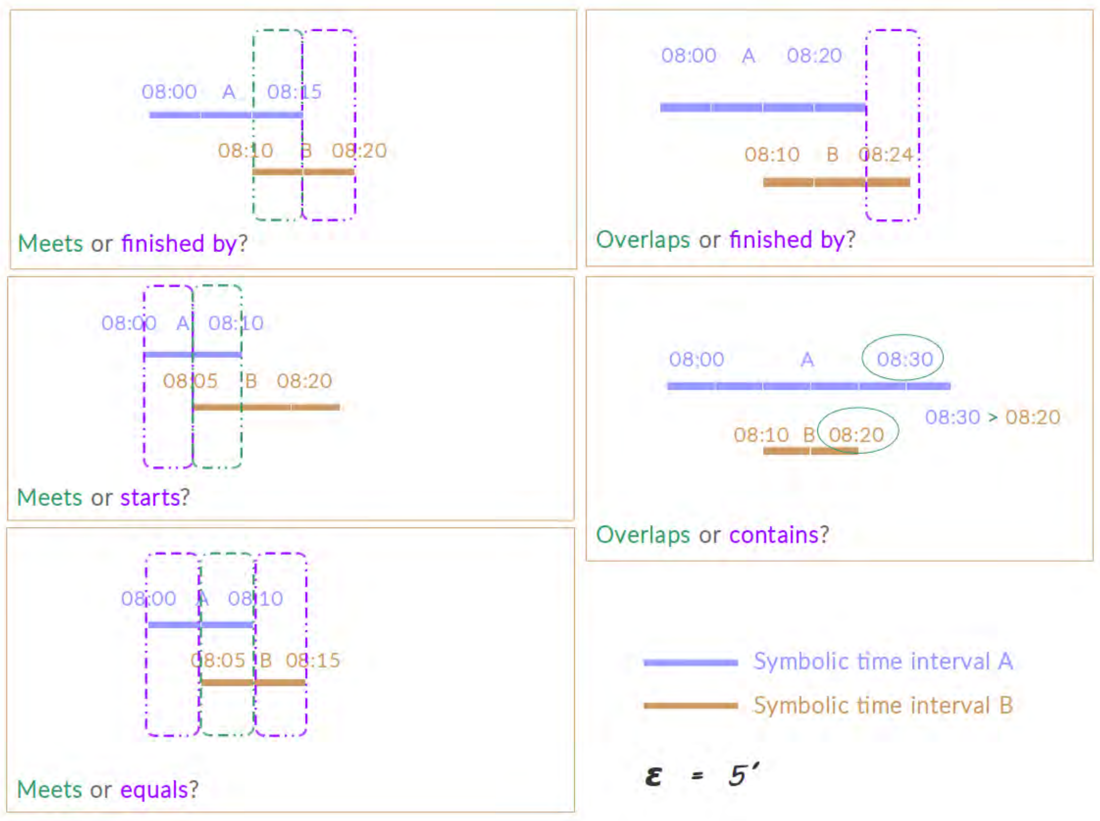


Figure 4.2: Examples of a lack of robustness when epsilon margins are introduced.

Therefore, in this thesis robust temporal relations are defined based on the epsilon approach to solve ambiguities and lack of temporal identification issues, that are as follows.

Definition 4.1.6. Temporal relations. Given two symbolic time intervals, I^A and I^B

r(I^A,I^B)=b: A is before B if:

$(B.s - A.e) > \epsilon$ and

$(B.s - A.e) < C_{max\_gap}$ and $\boldsymbol{(B.s - A.e) > C_{min\_gap}}$

r(I^A,I^B)=m: A meets B if:

$|B.s - A.e| \leq \epsilon$ and

$\boldsymbol{(B.s - A.s) > \epsilon}$ and $\boldsymbol{(B.e - A.e) > \epsilon}$

r(I^A,I^B)=o: A overlaps B if:

$(B.s - A.s) > \epsilon$ and $(A.e - B.s) > \epsilon$ and

$\boldsymbol{(B.e - A.e) > \epsilon}$

r(I^A,I^B)=l: A left contains B if:

$\boldsymbol{\epsilon > 0}$ and $|B.s - A.s| \leq \epsilon$ and $(A.e - B.e) > \epsilon$

r(I^A,I^B)=c: A contains B if:

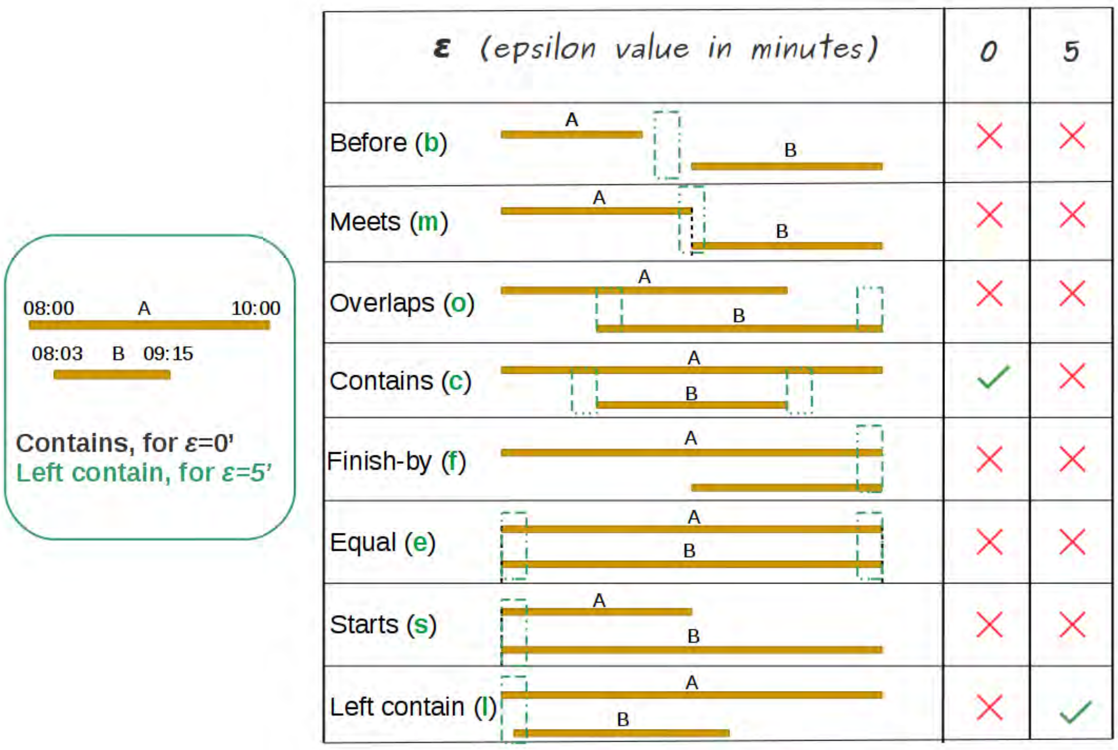


Figure 4.3: Illustration of the need for the left contain relation

$(B.s - A.s) > \epsilon$ and $(A.e - B.e) > \epsilon$

r(I^A,I^B)=f: A is finished by B if:

$(B.s - A.s) > \epsilon$ and $|B.e - A.e| \leq \epsilon$

r(I^A,I^B)=e. A and B are equal if:

$|B.s - A.s| \leq \epsilon$ and $|B.e - A.e| \leq \epsilon$

r(I^A,I^B)=s: A starts B if:

$|B.s - A.s| \leq \epsilon$ and $(B.e - A.e) > \epsilon$

In the definitions, conditions in boldface have been modified in this work. Firstly, the before temporal relation is modified to include the C_{min_gap} constraint. Although this constraint is used in SPM, previous work on TIRPs mining ignored it. Secondly, to avoid the ambiguity between the meets and starts, meets and finished by, and meets and equal relations, two new conditions are included in the definition of meets, $(B.s - A.s) > \epsilon$ and $(B.e - A.e) > \epsilon$. These conditions restrict the start time of B to be a minimum of ϵ times greater than the start time of A, and the end time of B to be a minimum of ϵ times greater than the end time of A. Thirdly, overlaps is modified in order to disambiguate contains and finished by, by adding a condition on the separation of the end points of the two related event intervals, $|B.e - A.e| > \epsilon$. Finally, with respect to the left contains temporal relation, a condition in which epsilon must be greater than zero ($\epsilon > 0$) is added to avoid ambiguity with the contains relation that arises when epsilon is zero. It is important in the following to avoid confusing starts with left contains; the difference between these two temporal relations is that in the starts relation, B.end is greater than A.end, while in left contains, B.end is smaller than A.end. Figure 4.4 shows how these ambiguities are solved with the robust definition given above.

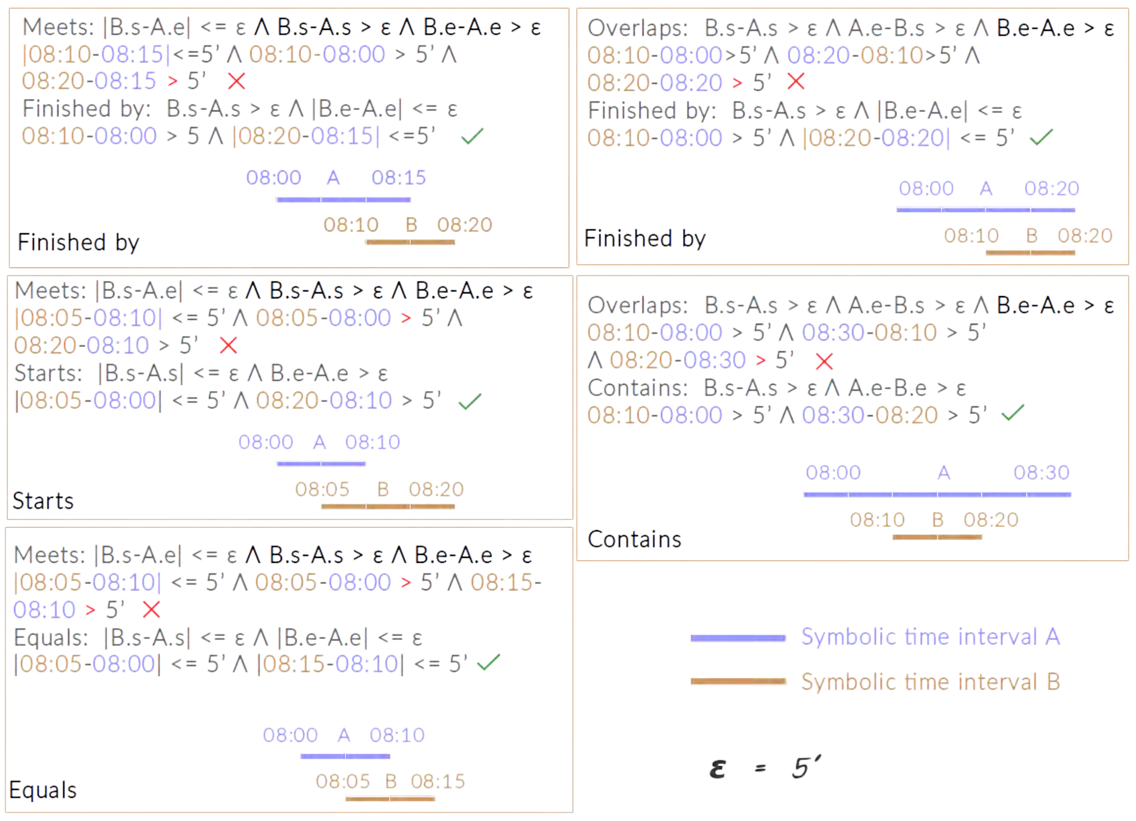


Figure 4.4: Illustration of how the problem of the lack of robustness is solved for overlaps, is finished by, meets, starts, equals, overlaps and contains

The temporal relations $r(I^A, I^B)$ are denoted as $r_{A,B}$ for simplicity. On the other hand, Σ_T denotes the set of all possible values of the temporal relations, i.e. $\Sigma_T = \{b, m, o, l, c, f, e, s\}$.

4.1.3 TIRP

The output of vertTIRP is a set of frequent TIRPs based on the definition given below.

Definition 4.1.7. Time-Interval-Related Pattern (TIRP). A non-ambiguous TIRP is defined as the pair $\langle SeqSym, \alpha \rangle$, where $SeqSym$ is the string of symbols ($SeqSym \in \Sigma^*$) and $\alpha \in \Sigma_T^*$ is a string that defines all the temporal relations R among each of the $(k^2 - k)/2$ pairs of symbolic time intervals supporting $SeqSym$ in the database $R = \{r(I^1, I^2), ..., r(I^1, I^k), r(I^2, I^3), ..., r(I^{k-1}, I^k)\}$

Note that a TIRP having a single symbolic time interval can exist as well, in which case, SeqSym will be of length one and R will be empty. From the example above, $< AB, s >$ ("A starts B") is a TIRP, supported by $< 8 : 00, 10 : 00, A >$ $, < 8 : 00, 12 : 00, B >$ of the first sequence, and by $< 16 : 00, 17 : 00, A >, < 16 : 00, 18 : 00, B >$ of the second sequence. Moreover, $< ABC, sbo >$ ("A starts B", "A before C", and "B overlaps C") is a TIRP supported by $< 8 : 00, 10 : 00, A >, < 8 : 00, 12 : 00, B >, < 11; 00, 13 : 00, C >$.

It is should be noted that there may be several TIRPs with the same $SeqSym$ but different temporal relations.

Definition 4.1.8. S-TIRP. A S-TIRP $\widehat{SeqSym}$ is the set of all TIRPs with the same SeqSym.

For example, $\widehat{BC} = \{< BC, o >, < BC, m >\}$ is an S-TIRP from the example. Hence, although an SPM algorithm would find the single pattern "BC", a TIRPs mining algorithm such as vertTIRP can identify several different patterns according to their temporal relations.

TIRPs can be characterised by a set of indicators including vertical support, horizontal support, mean horizontal support, and mean duration. The first two indicators refer to how patterns match sequences in the source data.

Definition 4.1.9. Pattern matching. A TIRP $\langle SeqSym, \alpha \rangle$ matches a sequence IS if it is contained within the sequence, i.e. $\forall s_i, s_j \in SeqSym, \exists I^{\sigma(i)}, I^{\sigma(j)} \in IS$ such that $I^{\sigma(i)}.sym = s_i, I^{\sigma(j)}.sym = s_j$, and $r(I^{\sigma(i)}, I^{\sigma(j)}) = r_{i,j}$.

Definition 4.1.10. Vertical support for a TIRP P. $vs(P)$ measures how many sequences from the source data match the pattern: $vs(P) = |S^P|/|S|$, where $|S|$ is the overall number of data sequences and $|S^P|$ is the number of data sequences that match P.

For instance, the pattern <AB,s> in the example of Figure 4.1 has a vertical support value of 2/3, since there are two sequences in the data source that contains it: $\langle <8:00, 10:00, A>, <8:00, 12:00, B> \rangle$, and $\langle <16:00, 17:00, A>, <16:00, 18:00, B> \rangle$.

Definition 4.1.11. Horizontal support for a TIRP P. $hs(P, IS^i)$ measures the number of times the pattern matches the sequence IS^i: $hs(P, IS^i) = |E^P_{IS_i}|/|E_{IS_i}|$, where E_{IS_i} is the overall number of time interval events in the sequence IS^i, and $E^P_{IS_i}$ is the number of time interval events in the sequence IS^i that matches P.

In the example of Figure 4.1, pattern <C,_> in the third sequence has a value of 2/4 for horizontal support: $hs(<C, \_>, \langle <10:00, 13:00, C>, <11:00, 14:00, B>, <12:00, 13:00, A>, <14;00, 15:00, C> \rangle) = 2/4$.

Definition 4.1.12. Mean horizontal support (mhs) of a TIRP P. This is the average of all the horizontal supports of P found in the data set: $mhs(P, S^P) = \frac{\sum_{i=1}^{|S^P|} hs(P, IS_i)}{|S^P|}$

In the example of Figure 4.1, <C,_> appears once in the first and second time interval sequences, and twice in $s3$; the mean horizontal support of the pattern <C,_> is therefore mhs(<C,_>)=(1/3+1/3+2/4)/3 =0.38889.

Definition 4.1.13. Mean duration (md) of a TIRP P. This is the average duration of the time intervals of the n sequences that provide horizontal support to P. The duration of a TIRP P is calculated based on the earliest and latest end times of the time intervals from which the TIRP P receives support.

$md(P) = \frac{\sum_{i=1}^{n} (Max_{j=1}^{k} I^{i,j}_{.e} - I^{i,1}_{.s})}{|S^P|}$

where $I^{i,j}$ is the j-th time interval in the i-th sequence where P matches, and the Max operator selects the time interval with the latest end time from the k times P matches sequence i. Note that the earliest time interval in a sequence is always the first in $(I^{i,1}_{.s})$.

In the example of Figure 4.1, TIRP <AC,b> is supported by the first and third time interval sequences (see Figure 4.1). In the first sequence, <AC,b> is supported by <8:00, 10:00, A> and <11:00, 13:00, C> with a start time at 08:00 and end time at 13:00, meaning that the duration of <AC,b> is five hours in the first sequence. In an analogous way for the third sequence, <AC,b> is supported by <12:00, 13:00, A> and <14:00, 15:00, C> with a duration of three hours. This means that <AC,b> has a mean duration of $md(<AC, b>) = 4$ h ((5h+3h)/2).

Finally, it is important to note that the vertical support is used to retain only those TIRPs that have a value higher than a given threshold $min\_support$, i.e. frequent TIRPs. The reason for choosing vertical support to reduce the number of candidates is because it is the most well-known anti-monotonic constraint [86]. Other measures such as horizontal support or mean duration are used for descriptive purposes, or to obtain patterns in order to make classifications or predictions.

4.2 The vertTIRP algorithm

To find frequent TIRPs from a dataset of sequences of time interval events, vertTIRP follows Algorithm 5. The inputs of the algorithm include the dataset or collection of time interval sequences, a set of constraints C, and the *min_support* value that determines which patterns considered frequent. The dataset is expected to fulfil the definition of an LSTISs (i.e. the sequences are sorted). The first step of the algorithm consists of customising the temporal relations according to the customer constraints C={min_gap, max_gap, min_duration, max_duration}. In the second step, all time interval events in the dataset with a duration shorter than the one set by the user, (*min_duration*), are filtered out. Next, a pairing strategy is configured based on the current dataset, which is essential in speeding up the pattern discovery process in the following steps. In the fourth step, vertTIRP builds a vertical representation of frequent TIRPs with length one, and then generates a tree using a DFS strategy that recursively explores candidates.

As shown in Algorithm 6, candidates are generated by appending length-one frequent patterns (or length-one suffixes) to the end of the actual length-n frequent pattern, by means of a join operation (step 6). At each node, the list of candidates and a list of length-one suffixes are minimised using the Apriori principle, in which only frequent children are kept. Backtracking is applied when no more candidates can be generated. In this way, vertTIRP starts to generate a branch of the first frequent event and all the possible temporal relations, i.e. A. Next, branches AB, ABC, and AC are generated. Following this, vertTIRP generates the branch of the next frequent event B, i.e. B, BA, BAC, BC. The algorithm stops when all branches of frequent events have been generated.

Figure 4.5 shows an example of a search tree generated by vertTIRP. It is a typical sequence tree (also referred to as a Lexicographical Tree [30]) of a generic SPM adapted to TIRPs. Each node of the tree represents an S-TIRP, with the corresponding frequent TIRPs and their indices (vertical support, horizontal support, and mean duration). Frequent TIRPs of length one are at the top, while longer ones are positioned at the leaves. Some of the patterns contain a single temporal relation (e.g. <BC,o> -B overlaps C- and <BC,m> -B meets C-), while others have more than one (e.g. <ABC,sbo> has three: A starts B, A before C, B overlaps C). It can be seen that shorter candidates have higher support than longer ones.

vertTIRP is then based completely on the DFS approach. It uses several efficiency strategies, including a pairing strategy, the transitivity properties of the temporal relations, and the vertical data representation of TIRPs. The details are described in the following subsections, starting with a description of the inputs to the algorithms (LSTIS and constraints). B.5 provides a complete mining example.

```
Algorithm 5: vertTIRP
  input  :  LSTIS: a collection of symbolic time interval sequences sorted by
            lexicographical order
            C = {min_gap,max_gap,min_duration,max_duration}:
  user-controlled constraints
            min_support: required for the discovered patterns
  output: frequentPatterns
1 setTemporalRelations(min_gap, max_gap)
2 customizedLSTIS = removeI(min_duration)
3 PS = pairingStrategy(customizedLSTIS)
4 F = vertical-lenght1(customizedLSTIS)
5 frequentPatterns = []
6 for each pattern $\overbrace{P} \in F$ do
7     discoveredPatterns = searchPatterns($\overbrace{P}$, F, PS, min_support,
        max_duration)
8     append(frequentPatterns, discoveredPatterns)
9 return frequentPatterns
```

4.2.1 Tabular LSTIS

As input, vertTIRP takes an LSTISs containing the source time interval sequences. To enable efficient data management, these can be stored in a table, as shown in Table 4.1 for the example in Figure 4.1. Each time interval sequence has a sequence identifier sid (which is unique to each dataset) and contains one or more time interval events, which are sorted based on the start time. Each time interval event in a sequence has an event identifier eid that is unique within the sequence. These identifiers are used internally by vertTIRP for data management. For example, the first event of sequence $s2$, $eid = 1$, occurs between 14:00 and 17:00, with value C.

4.2.2 Constraints on user control patterns

The second input for vertTIRP is a set of user constraints that allow the user to express particular requirements for the patterns to be found. For TIRPs, Moskovitch [68, 25, 4], Patel [2], and Papapetrou [28] used two constraints: the maximum gap C_{max_gap} which determines the maximum distance between two events in a TIRP, and the maximum duration constraint $C_{max\_duration}$, which determines the maximum duration of each event interval.

vertTIRP introduces two additional constraints, $C_{min\_duration}$ and C_{min_gap}.

```
Algorithm 6: searchPatterns
input : $\overbrace{P}$: S-TIRP of length k containing frequent patterns
        $L$: List of length-one S-TIRPs of frequent patterns
        $PS$: pairing strategy
        $min\_support$: required for the discovered patterns
        $max\_duration$: required for the discovered patterns
output: discoveredPatterns
1  discoveredPatterns = []
2  append(discoveredPatterns, $\overbrace{P}$)
3  newPats = []
4  newL = []
5  for each S-TIRP $\overbrace{F} \in L$ do
6      newSPattern = join($\overbrace{P}$, $\overbrace{F}$, PS, max_duration)
7      if support(newSPattern) > min_support then
8          append(newPats,newSPattern)
9          append(newL,$\overbrace{F}$)
10 for each $\overbrace{new\_p} \in newPats$ do
11     new_discovered_patterns = searchPatterns($\overbrace{new\_p}$, newL, PS,
         min_support, max_duration)
12     append(discoveredPatterns, new_discovered_patterns)
13 return discoveredPatterns
```

sid	eid	start time	end time	value
s1	1	8:00	10:00	A
s1	2	8:00	12:00	B
s1	3	11:00	13:00	C
s2	1	14:00	17:00	C
s2	2	16:00	17:00	A
s2	3	16:00	18:00	B
s3	1	10:00	13:00	C
s3	2	11:00	14:00	B
s3	3	12:00	13:00	A
s3	4	14:00	15:00	C

Table 4.1: LSTIS for the example in Figure 4.1

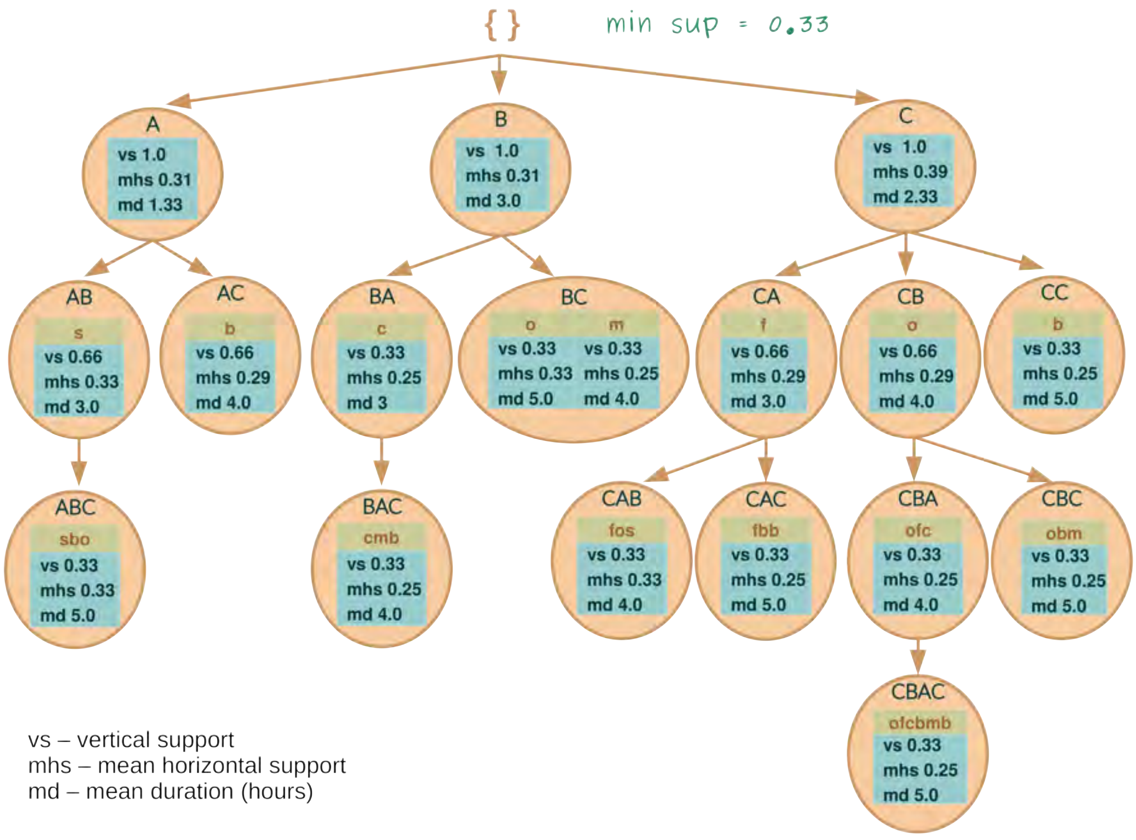

{ }
min sup = 0.33
A vs 1.0 mhs 0.31 md 1.33
B vs 1.0 mhs 0.31 md 3.0
C vs 1.0 mhs 0.39 md 2.33
AB s vs 0.66 mhs 0.33 md 3.0
AC b vs 0.66 mhs 0.29 md 4.0
BA c vs 0.33 mhs 0.25 md 3
BC o m vs 0.33 vs 0.33 mhs 0.33 mhs 0.25 md 5.0 md 4.0
CA f vs 0.66 mhs 0.29 md 3.0
CB o vs 0.66 mhs 0.29 md 4.0
CC b vs 0.33 mhs 0.25 md 5.0
ABC sbo vs 0.33 mhs 0.33 md 5.0
BAC cmb vs 0.33 mhs 0.25 md 4.0
CAB fos vs 0.33 mhs 0.33 md 4.0
CAC fbb vs 0.33 mhs 0.25 md 5.0
CBA ofc vs 0.33 mhs 0.25 md 4.0
CBC obm vs 0.33 mhs 0.25 md 5.0
CBAC ofcbmb vs 0.33 mhs 0.25 md 5.0
vs – vertical support
mhs – mean horizontal support
md – mean duration (hours)

Figure 4.5: The resulting TIRP tree (the abbreviations used are as follows (see Figure 1.1): s/starts, b/before, o/overlaps, m/meets, c/contains, f/finished by).

Definition 4.2.1. Minimum duration constraint $C_{min\_duration}$: Each TIRP should have a duration of at least $C_{min\_duration}$ time units; if not, it is discarded.

Definition 4.2.2. Minimum gap constraint C_{min_gap}: Two time interval events that have a before relation should be separated by at least C_{min_gap} time units.

For example, in the case of Type 1 diabetes mellitus, if the continuous glucose monitoring readings are above 180 mg/dl for at least 60 minutes between two and six hours after bolus administration [87, 88], the event is considered a hyperglycaemic event; otherwise it is labelled as a non-hyperglycaemic event. In this case, events with a duration of less than 60 minutes are not relevant($C_{min\_duration} = 01:00$). In terms of the minimum gap, events that occur sooner than two hours after bolus administration are also irrelevant and can be discarded by setting the minimum gap to two hours ($C_{min\_gap} = 02:00$).

In total, four constraints are managed in vertTIRP (C={ $C_{min\_duration}$, C_{min_gap}, C_{max_gap}, $C_{max\_duration}$}), of which two are new and two are adopted from previous works. Constraints play several different roles in Algorithm 5. First, in step 1 of the algorithm, C_{min_gap} and C_{max_gap} are used to customise the temporal relations (see definition 4.1.6). In step 2, $C_{min\_duration}$ is used to remove noise from the source data. Finally, $C_{max\_duration}$ is used in step 7 for pattern searching.

4.2.3 Pairing strategy

Step 3 of Algorithm 5 involves defining a strategy to test the temporal relationships between two time interval events. To calculate the temporal relation between two time intervals, one could use a naive approach by analysing all the relations in no particular order, and without taking into account their probabilities. vertTIRP uses a more efficient way to compute these temporal relations by including two paring strategies: (i) checking the temporal relations in order of frequency (from more frequent to less frequent), and (ii) grouping the temporal relations by common conditions to avoid unnecessary checks. Both of these strategies can be combined.

4.2.3.1 Sorting temporal relations by frequency

There are several algorithms from related fields to TIRPs mining, such as in the frequent itemset mining and the utility mining fields, that sort patterns by their support [33, 18, 89] or by utility decreasingly [90, 91] to speed up the mining process. This idea is transferred to temporal relations, ordering the relations by their frequency and checking the most frequent relations first.

First, an initial analysis, which involves ordering the relations from more to less frequent, is carried out. Table 4.2 shows the resulting process for the datasets used in the experimental section.

4.2.3.2 Grouping the temporal relations by common conditions

Looking at the definitions of the temporal relations (Definition 4.1.6), it is possible to observe that the c, f, o, m relations have a common condition, $(B.s - A.s) > \epsilon$. The same applies to the s, l and e relations, the common condition of which is $(B.s - A.s) <= \epsilon$. This means that if the common condition is not met, all of the temporal relations involved in this condition can be ruled out.

In order to allow discarding temporal relations based on these common conditions, an intersection, group and subgroup of relations are defined as follows.

Definition 4.2.3. Intersection of temporal relations. Given two temporal relations $\alpha, \beta \in \Sigma_T$, and the conditions that define them $\lambda(\alpha), \lambda(\beta)$, the intersection of α

dataset	epsilon	sorted relations	common conditions
MAV	10	bfseclmo	b fcmo sel
MAV	30	bfsemlco	b fmoc sel
MAV	50	bfesmlco	b fmoc esl
ASL	10	bemcfosl	b esl mocf
ASL	30	bemcfosl	b esl mocf
ASL	50	bemcfosl	b esl mocf
HAR	10	bmfseclo	b mofc sel
HAR	30	bmfselco	b mofc sel
HAR	50	bemfslco	b esl mofc
CBLS	10	mbefscol	mofc b esl
CBLS	30	mebfsclo	mofc esl b
CBLS	50	emfsbcol	esl mofc b
SI	10	fsbeclmo	fcmo sel b
SI	30	fsbeclmo	fcmo sel b
SI	50	fsbelmco	fmoc sel b
CI	10	ebmcfosl	esl b mocf
CI	30	ebmcfosl	esl b mocf
CI	50	ebmcfosl	esl b mocf

Table 4.2: Results of pairing strategies for each dataset, for epsilon>0

and β , denoted by $\lambda(\alpha) \cap \lambda(\beta)$, is the set containing all logical conditions of $\lambda(\alpha)$ that also belongs to $\lambda(\beta)$.

Definition 4.2.4. Group of temporal relations. Two temporal relations α and β are in the same group if they share a common logical condition.

Definition 4.2.5. Subgroup of temporal relations. Two temporal relations α and β belongs to the same subgroup if they share at least two common logical conditions c_1 and c_2 in their definition, and there exists a third temporal relation χ that shares with α and β either the logical condition c_1 or c_2.

Following Definition 4.2.3, a table of common conditions (Table 4.3) among the temporal relations was defined (see B.3 for details on intersection of conditions). In Table 4.2, the relations that can be ruled out based on the same condition (subgroup of temporal relations) are underlined.

4.2.3.3 Sequence of strategies

Both of these strategies can be applied in sequence. Once it is known the frequency of the temporal relations, it is possible to apply the check for a common condition.

relation / condition	b	m	o	l	c	f	e	s
$B.s - A.s > \epsilon$		🟩	🟩		🟩	🟩		
$B.e - A.e > \epsilon$		▨	▨					🟦
$A.e - B.e > \epsilon$				🟦	🟦			
$\|B.s - A.s\| \leq \epsilon$				🟧			🟧	🟧
$\|B.e - A.e\| \leq \epsilon$						🟦	🟦	

Table 4.3: Common conditions among the temporal relations (rows show common conditions, while columns show temporal relations)

Since the first strategy is dataset-dependent, the resulting common conditions are also dataset-dependent. For example, for the Human Activity Recognition dataset dataset (HAR) dataset, the temporal relations are checked in the following order: b, m, o, c, f, s, e. First, the b relation is checked, and if the condition is not met, then the m,o,c,f common condition is checked. If this is not met, then the common condition $|B.s - A.s| \leq \epsilon$ for the s,e relations is checked. In the case where the $|B.s - A.s| \leq \epsilon$ condition is met, the algorithm continues with checking while preserving the order: first it checks the s relation, and if it is not met, it then checks the e relation.

4.2.4 Vertical representation of TIRPs

Step 4 of Algorithm 5 deals with the internal representation of patterns. TIRP algorithms have mainly used two kinds of data structures to represent TIRPs that support the generation of candidate patterns: horizontal and vertical. In a horizontal representation, each row contains a sid and the corresponding symbols present in the sequence, in the same order as in the sequence. For example, Table 4.4 shows the horizontal data structure for the example used in this paper (Table 4.1). All of the existing TIRPs mining algorithms in the literature [26, 21, 2, 68, 25, 4, 72, 24, 74, 22, 29] (except for the hybrid DFS approach in [28]), use a horizontal-like representation.

In contrast, vertTIRP utilises a vertical database representation of patterns, inspired by previous works on sequence mining (SPADE[18] and SPAM[16]). Vertical representations have been proven to be more efficient in the mining process. In a

sid	symbol sequence
1	A,B,C
2	C,A,B
3	C,B,A,C

Table 4.4: Horizontal database.

pattern	sid list
A	1,2,3
B	1,2,3
C	1,2,3

Table 4.5: Vertical database.

vertical database representation, each row contains a pattern item and the correspondent sid lists, i.e. a list of sequence ids in which the corresponding pattern appears. Table 4.5 shows a vertical representation of the example provided in Table 4.1.

vertTIRP adopts and extends the vertical representation, as it allows for the efficient management of TIRP discovery (as explained in the candidate generation step in Section 4.2.5). First, patterns are grouped by S-TIRPS to form the first dimension of the table. For the sake of simplicity, a TIRP within an S-TIRP is identified by the temporal relation (defined in Σ_T, see Definition 4.1.7). Following this, the sid list is extended with the following information: the event id (or the position at which the pattern starts in the sid sequence), the start and end time of the corresponding pattern (which means choosing the minimum of the start times and the maximum end time), the source time intervals, and the different pattern indices vs, mhs, and md. All of these elements are kept sorted, in order to make the mining process faster. Table 4.6 shows an example of the vertical representation for the dataset used in this paper (Table 4.1) with three patterns: a length-one pattern, <A,_>, and two length-two patterns, <AB,o> and <AB,m>, which are grouped under the S-TIRP $\widehat{BC}$.

Although H-DFS [28] uses the vertical representation, it ignores the event id, using only the sid and the list of intervals. In contrast, the eid is included since this it is computationally more efficient to compare two integer numbers than two date-times. Another limitation of the structure of H-DFS (which is discussed in the paper under future work) is that it assumes that each sequence can support a pattern at most once (i.e. A in the sequence ABA is counted only once), while the approach presented in this thesis captures multiple occurrences of each pattern in a sequence, and uses a special indicator for this (i.e. the horizontal support).

4.2.5 Candidate generation

Step 7 of Algorithm 5 forms the core of the candidate generation process and is set out in Algorithm 6. The algorithm is based on the extension of each available S-TIRP with a length-one pattern.

S-TIRP	TIRP (Σ_T)	sid	eid	starts	ends	source intervals	vs	mhs	md
$\widehat{A}$	-	1	1	8:00	10:00	[8:00,10:00]	3	1	1.3
		2	2	16:00	17:00	[16:00,17:00]			
		3	3	12:00	13:00	[12:00,13:00]			
$\widehat{BC}$	$\{o\}$	1	3	8:00	13:00	[8:00,12:00], [11:00,13:00]	1	1	5
	$\{m\}$	3	4	11:00	15:00	[11:00,14:00], [14:00,15:00]	1	1	4

Table 4.6: Example of the vertical representation of patterns in verTIRP

First, in the joining operation in step 6, the benefits of the vertical representation of patterns are exploited in order to generate and obtain the support of new candidates, as set out in Algorithm 7. The first step of the joining operation consists of the creation of a new S-TIRP, $\widehat{new\_P}$, by concatenating the symbol of a length-one pattern F at the end of P $(P + F)$. Next, for each time interval sequence s_i in $\widehat{P}$ that is also in $\widehat{F}$ (i.e. that gives support to both patterns), and for each event eid_i and eid_j for $\widehat{P}$ and $\widehat{F}$, respectively, the algorithm searches for an event id that is greater than or equal to the event eid_i. In the case where the event corresponding to $\widehat{F}$ is greater than eid_i (i.e. eid_i occurs before eid_j), the algorithm computes the corresponding temporal relation between the events and create the new TIRP. Additional relations can be generated by using the transitivity property of the temporal relations. Finally, the corresponding support indices are computed for the patterns based on the last additions.

Special attention should be paid to the equal relation. Note that A = B is the same as B = A, so if the algorithm have mined a pattern with an extension containing the symbol "A", there is no need to mine the extension with the symbol "B". The condition shown in line 6 filters these situations.

Lines 7 to 9 of Algorithm 6 then aim to minimise the size of candidates using the Apriori prunning principle [31], which states that if there is a certain sequence of symbols (ignoring time) that is infrequent, all its supersets will also be infrequent

```
Algorithm 7: join
input : $\overbrace{P}$ : S-TIRP of length k containing frequent patterns,
        $\overbrace{F}$ : S-TIRP of length one containing a frequent pattern,
        $PS$: pairing strategy,
        $max\_duration$: requirement for the discovered patterns
output: $\overbrace{P+F}$, a new S-TIRP of length (k+1) containing frequent patterns
1  create a new S-TIRP pattern, $\overbrace{new\_P}$, which is a concatenation of the
     symbol $F$ at the end of the string $P$ $(P+F)$
2  for each $eid_i, eid_j$ so that $sid \in \overbrace{P} \cap \overbrace{F}$ and $eid_i \in events(\overbrace{P}, sid)$ and
     $eid_j \in events(\overbrace{F}, sid)$ do
3      if $eid_i \leq eid_j$ then
4          Find the temporal relation r between $eid_i$ and $eid_j$ following PS
5          if $duration(r, eid_i, eid_j) \leq max\_duration$ then
6              if (r = 'e' and $last(P) < F$) or (r $\neq$ 'e') then
7                  Create a new TIRP with r and add it to $\overbrace{new\_P}$
8                  Compute the transitivity relations tr derived from r
9                  Add tr to $\overbrace{new\_P}$
10                 updateSupport($\overbrace{new\_P}$)
11 return $new\_p$
```

(i.e. if pattern AA is infrequent, then all other extensions such as ABA, ACA, ADA will be infrequent). This principle is applied to keep frequent patterns of length one within the neighbourhood search space.

In the second part of Algorithm 6 (lines 10 - 12) a recursive call to the algorithm is performed.

4.2.6 Transitivity for efficiency

Allen introduces a transitivity table to allow for reasoning about temporal purposes [27]. For instance, given the knowledge that event A happened before event B, and B happened before C, then it can be inferred that A happened before C. This is represented in the transitivity table as T(b,b)=b (see the full table in B.4).

From the relations $r_1(I^A, I^B) = o$ (A overlaps B) and $r_2(I^B, I^C) = f$ (B is finished by C), it could be found that the only possibilities for the temporal relation between

A and C are the b,m,o relations (see Figure 4.6). However, when the temporal relation definitions are based on $\epsilon > 0$, the possibilities for the temporal relation between A and C became multiple, i.e. the b,m,o and f relations (see Figure 4.7).

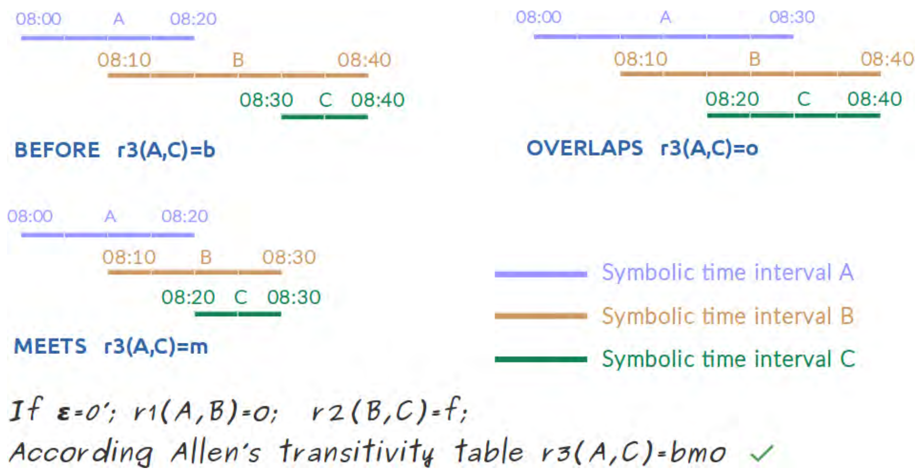


Figure 4.6: Analysis of Allen's transitivity relation for $\epsilon = 0$ minutes, $r_1(A, B) = o$ and $r_2(B, C) = f$

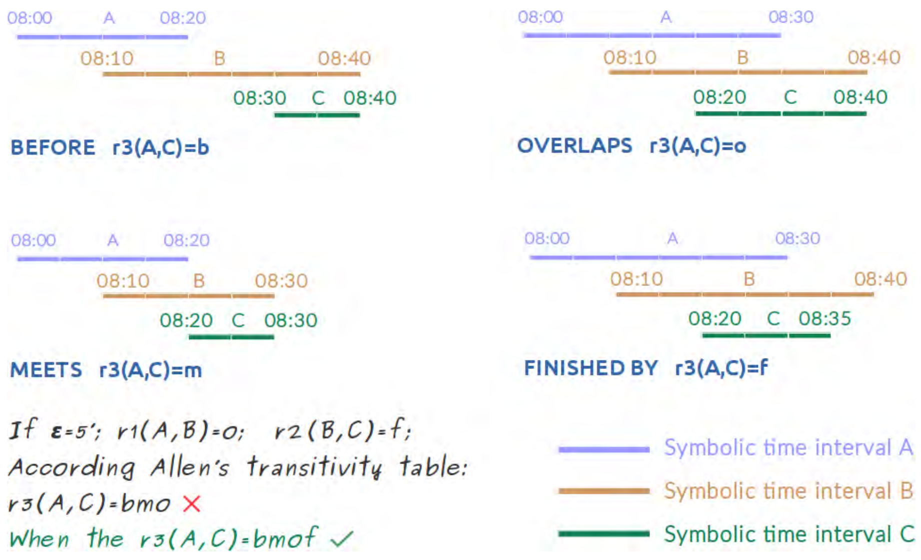


Figure 4.7: Analysis of Allen's transitivity relation for $\epsilon = 5$ minutes, $r_1(A, B) = o$ and $r_2(B, C) = f$

Allen's transitivity table is therefore not valid for a value of epsilon greater than zero. In the present work, following Definition 4.1.6, a new transitivity table for epsilon greater than zero is defined (see Table 4.7). It should be observed that the transitivity table reduces the number of possible temporal relations between two events to a smaller degree, although the actual relation between two events is a single temporal relation when applied to a given set of time interval events.

$r_2(I^B, I^C)$ / $r_1(I^A, I^B)$	b	c	o	m	s	f	e	l
b "before"	b	b	b	b	b	b	b	b
c "contains"	b c f m o	c	c f o	c f o	c f o	c f	c f	c
o "overlaps"	b	b c f m o	b m o	b m	m o	b f m o	m o	c f o
m "meets"	b	b m	b m	b m	b m	b m	b m	b m
s "starts"	b	b c f m o	b m o	b m	m s	b m o	e m o s	c f l m o
f "finished-by"	b m	c f	f m o	b m o	f m o	c f m o	c f l m o	c f
e "equal"	b m	c f	f m o	b e m o	e o s	c f m	c e f o	c f l
l "left contain"	b c f m o	c	c f o	c m o	c e l o	c f	c e f l	c

Table 4.7: Transitivity table for eps>0

KarmaLego [4] was the first TIRPs mining algorithm to use Allen's transitivity table to improve mining efficiency. However, vertTIRP is the first algorithm that applies the transitivity property to a vertical representation of TIRPs (since KarmaLego uses a horizontal representation of the data). The difference is that in the horizontal representation, transitivity is used to generate candidates, while in the vertical representation, it is used to assign the temporal relation. Using the vertTIRP representation, it is not possible to generate candidates for temporal relations in the same way as with symbols (i.e. AB may have several relations). Hence, the temporal relation is calculated during the joining process (see step 8 of Algorithm 7), and only the relations given by the transitivity table are checked.

4.3 Experimental Evaluation methodology

In this section, we test vertTIRP with the aim to answer the following research question: How can we develop a TIRP mining algorithm that is efficient in terms of time and memory and which discovers robust TIRPs?

Secondary research questions:

- What impact does it have on time and memory if we use pairing strategies that are specifically used to improve the efficiency when mining temporal relations.

- What impact does it have on time and memory if we use a combination of a vertical representation of patterns and the transitive properties of the temporal relationships?

- What impact does it have on time and memory if we use two new constraints that enable the user to express the kind of patterns to be learnt (user-controlled patterns), namely min-duration and min-gap?

- What impact does it have on time and memory if we use two new constraints that enable the user to express the kind of patterns to be learnt (user-controlled patterns), namely min-duration and min-gap?

- What robustness problems are there in the temporal relations and in the transitivity property with epsilon and how could they be solved?

4.3.1 Experimental platform for vertTIRP

The vertTIRP presented in this thesis were implemented in Python 3.5.2. The experiments were carried out on two virtual machines: (i) Lubuntu18.4, 32bits (one CPU, one thread, 3 GB of RAM), and (ii) Lubuntu18.4, 64bits (one CPU, one thread, 12 GB of RAM) created with the intention of exploring larger datasets.

4.3.2 Datasets

The experiments were conducted with synthetic and real datasets.

4.3.2.1 Synthetic datasets for vertTIRP

In order to better investigate the efficiency of the TIRP mining algorithm introduced in this dissertation, numerous synthetic datasets are generated in which, the following factors were considered: (i) number of time interval sequences(NS); (ii) sequence average length (SAL); and (iii) vocabulary size (V). With values for NS of 3, 1000, 5000, 10000 and 20000; values for SAL of 3, 10, 100, 200 and 300; and values for V of 3, 10, 100, 1000, 100000 and 1000000 is experimented.

4.3.2.2 Real datasets for vertTIRP

The following public datasets were used for the experiments of TIRP mining algorithms (see Table 4.8):

- Mavlab (MAV) dataset [92], which contains activities related to daily living, collected using the MavLab testbed during March and April of 2003. This dataset captured an inhabitant's interactions with an intelligent home, through sensors placed in different rooms. This dataset allows us to compare vertTIRP with a state-of-the art algorithm.
- American Sign Language Lexicon Video Dataset (ASL) [93], which consists of videos of American Sign Language Lexicon Video dataset (ASL) signs in citation form, each produced by native ASL signers. Linguistic annotations include gloss labels, start and end time codes for signs, start and end handshape labels for both hands, and morphological and articulatory classifications of the type of sign. This dataset allows to compare the work presented in this thesis with a state-of-the art algorithm.
- Human Activity Recognition (HAR) dataset [94] built from the recordings of 30 subjects performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors. This dataset can be found in the UCI Machine Learning Repository [95].
- Suicides in India (SI) [96] dataset, which contains annual suicide records of all states of India with various parameters from between 2001 and 2012.
- Childhood Blood Lead Surveillance (CBLS) [97] dataset, which contains data on blood levels of lead in children from between 1995 and 2015, from several U S states and local health departments.
- Chronic illness: symptoms, treatments and triggers (CI) [98] dataset, which was collected with a Flaredown app that can help patients with chronic autoimmune and invisible illnesses to improve their symptoms by avoiding triggers and evaluating their treatments. Every day, patients tracked the severity of their symptoms, treatments and doses, and any potential environmental triggers (foods, stress, allergens, etc) they encountered.

The aforementioned datasets are time series that need to be converted into event time intervals. They are characterised by timestamps followed by values of different attributes, which are pre-processed by converting them into time interval events. If the attribute had discrete values, the start time of the time interval was set according to the first time the discrete value appeared, while the end time was set with the start time of the first sample where the discrete value is different. For example, in the case of sensor values, for the following sequence of readings: 10:00-ON, 11:00-ON, 12:00-ON, 13:-OFF, 14:00-ON, 15:00-ON, the time intervals would be: (ON,10:00,13:00), (OFF,13:00,14:00), (ON,14:00,15:00).

Otherwise, if the attribute had numerical values, the attribute was discretised with the SAX algorithm [99], with a vocabulary size of three (i.e. $|\Sigma| = 3$).

4.3.3 Experimental setup

Several experimental configurations were defined to test the properties of the vertTIRP algorithm.

- Vertical representation of TIRPs. The first experiment included a comparison of the vertTIRP algorithm with the state-of-the-art algorithm KarmaLego and its efficient extension DharmaLego, [68] that has been found to be faster than H-DFS [28], ARMADA [29] and iMiner[2] for the MAV_p and ASL_p datasets.
- Efficiency by transitivity. In this case, verTIRP was analysed including the transitivity relation. Two configurations of verTIRP were compared, i.e. with and without the transitivity property. In the latter case, temporal relations were computed using the pairing strategy presented in this thesis.
- User controlled patterns. While user constraints allow the user to express how patterns can be found, they affect the number of patterns found and the expected computation time. Different *min_gap* and *min_duration* constraint values were tested. In particular, since the datasets have different time scales, *min_gap* and the *min_duration* were set at 0%, 1%, 5%, 10%, 30%, 50%, 70%, 90% and 99% with respect to the maximum time interval. This experiment was run with a fixed support of 0.01.
- Pairing strategies. Two configurations were defined: vertTIRP with the pairing strategy and a verTIRP with a dummy strategy (in which conditions were tested in a random order and without taking into consideration the common condition). The hypothesis is that if the pairing strategies were included, the computation time would be shorter than if not using them.
- Scalability. The last experiment included a complexity analysis of the algorithm presented in this thesis in terms of time and memory, which was

dataset	attribute sequence (NS)	attributes	TI(s)	num. TI	SAL
MAV	zone (11)	device, value	1 second	1472	133,81
ASL	Session, Scene (946)	Main_New_Gloss, D_Start_HS, D_End_HS, Passive_Arm	1 second	61370	64,87
MAV_p	integer numbers (1000)	integer numbers	unknown	61127	61,127
ASL_p	integer numbers (65)	integer numbers	unknown	2037	31,338
HAR	subject (30)	BodyAcc, GravityAcc, BodyGyro, Activity	5 minutes	1938	64,60
SI	State (38)	Age group, Gender, number of suicides	5 years	7349	193,39
CBLS	State (44)	popless72months, confirmedBLL	5 years	215	4,88
CI	user id (20843)	age, sex, country, trackable name	1 year	3488553	167,37

Table 4.8: Dataset description: The attribute sequence field is the name of the attribute used to generate the time interval sequences, where NS is the number of sequences in the dataset; attributes are the name of the attributes used to construct the sequences; TI(s) is the minimum time interval duration in seconds; num. TI is the total number of time intervals in the dataset; and SAL is the sequence average length. MAV_p and ASL_p are preprocessed MAV and ASL datasets, used in state of the art works, where the attributes were encoded with integer numbers.

conducted on synthetic datasets.

C_{min_gap} and $C_{min\_duration}$ constraints were set to zero unless otherwise specified.

4.4 Results

The results were evaluated in terms of the computational runtime, the memory usage, the number of TIRPs discovered and their support.

Concerning the comparison with the state of the art, the Figure 4.8 shows the results from the vertTIRP, KarmaLego [68] and DharmaLego [69] algorithms for the ASL_p and MAV_p datasets. In each plot, the x-axis represents the different values of *min_support* required for a pattern to be frequent, while the y-axis shows the runtime in seconds. As expected, the execution time for the MAV_b dataset has been longer than for the ASL_p dataset, since the number of intervals in the MAV_b dataset is 61127, while that of ASL_p dataset is 2037.

In the case of the ASL_p dataset for support 5, the execution time of KarmaLego is 2,45 s, of DharmaLego is 1,80 s, while that for vertTIRP is 1,38 s. Moreover, for the supports between 10 and 50, the execution time of DharmaLego was slightly less than that of vertTIRP. In that regard, it should be noted that the programming languages were Python for vertTIRP and $C\sharp$ for KarmaLego and DharmaLego. In [100] Python has been found up to 25 times slower than $C\sharp$. However, in the present work, the results are analysed as if all the algorithms are executed with the same language.

In the case of the MAV_b dataset for support 10, the execution time for DharmaLego was more than 187 s, while for vertTIRP it was only 8,39 s. KarmaLego could not be run with the MAV_b dataset, as it gave a memory error. This represents a huge reduction in computation time.

Concerning the transitivity properties in vertical representations, the hypothesis was that the transitivity property would improve the computation time for the vertical representation. Figure 4.9 shows the results obtained in terms of time for each dataset in both scenarios; i.e. vertTIRP with and without the transitivity property. As it could be seen from Figure 4.9, the improvement in the calculation time with transitivity is noticeable for the HAR dataset, while for the remainder of the datasets, the improvement is small to be considered significant. There are two possible reasons for these results: firstly, the dataset may be too small for the difference to be noticeable, and secondly, the more frequent temporal relations of these datasets return many possibilities from the transitivity table (i.e. datasets in which the contain or overlap relations predominate). In the HAR dataset, the most frequent temporal relations are the before and meet relations, while for the Chronic illness dataset (CI) dataset, the two most frequent relations are the contain and start relations (see Table 4.2). From the transitivity table (Table 4.7), it can be observed that the equal, before, and meet relations almost always return only one possible temporal relation. This fact can reduce the number of checks and hence

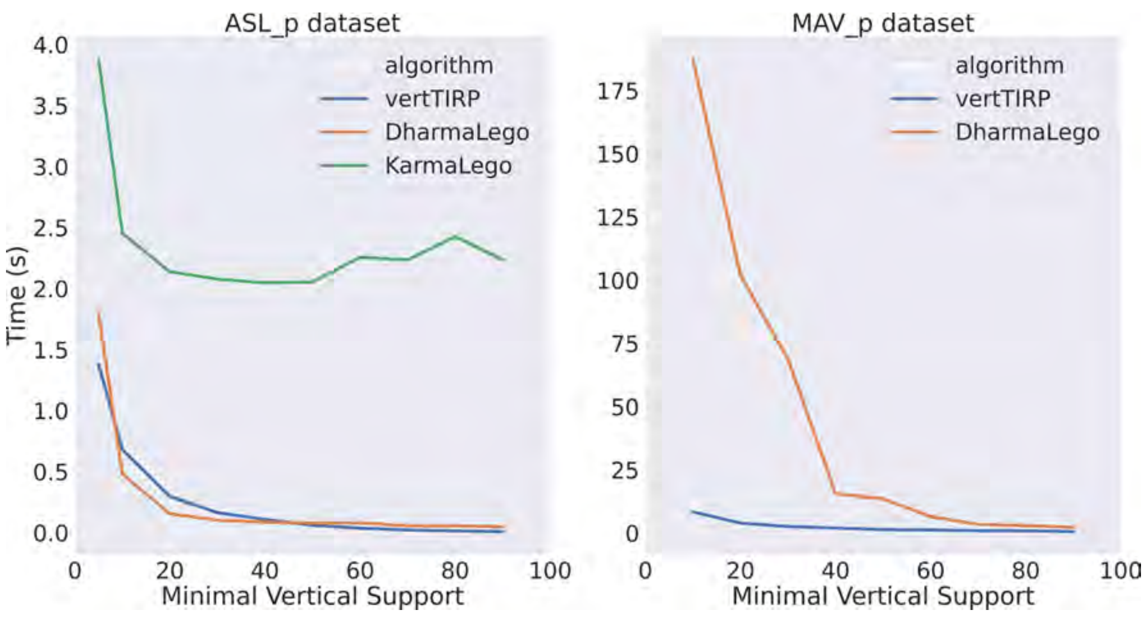


Figure 4.8: Comparison with the state of the art methods for the MAV and ASL datasets

the computation time.

Regarding applying the minimum gap and the minimum duration, the multiple y-axes of Figures 4.10 and 4.11 show the computation time, the memory usage and the number of patterns found, based on the inclusion of the user constraints. As expected for the two constraints, the computation time is greatly reduced. The reduction in computation time is directly related to the decrease in the number of TIRPs mined and the memory consumed. However, the number of TIRPs discovered in the HAR dataset was greater than for the CI dataset, while the computation time was lower for the HAR dataset. This is because the number of time interval sequences, the average number of time interval events per sequence, and the number of time intervals in the HAR dataset were lower than in the CI dataset.

Regarding the efficiency due to pairing of events, as it can be observed from Figure 4.12, the computation time required to find the temporal relations with pairing strategies is lower than when not these are not used (i.e. the dummy approach). Note that the y-axes of the graphs use different time scales in order to highlight the difference between the two algorithms. In general, the pairing strategy reduced the computation time by about 10%.

Regarding the epsilon sensitivity, the percentage uncertainty in the start and end times of the intervals (epsilon value) affects the number of the TIRPs that are discovered, the computation time and the memory usage. Figure 4.13 shows the results of these measures for each of the datasets for varying values of epsilon

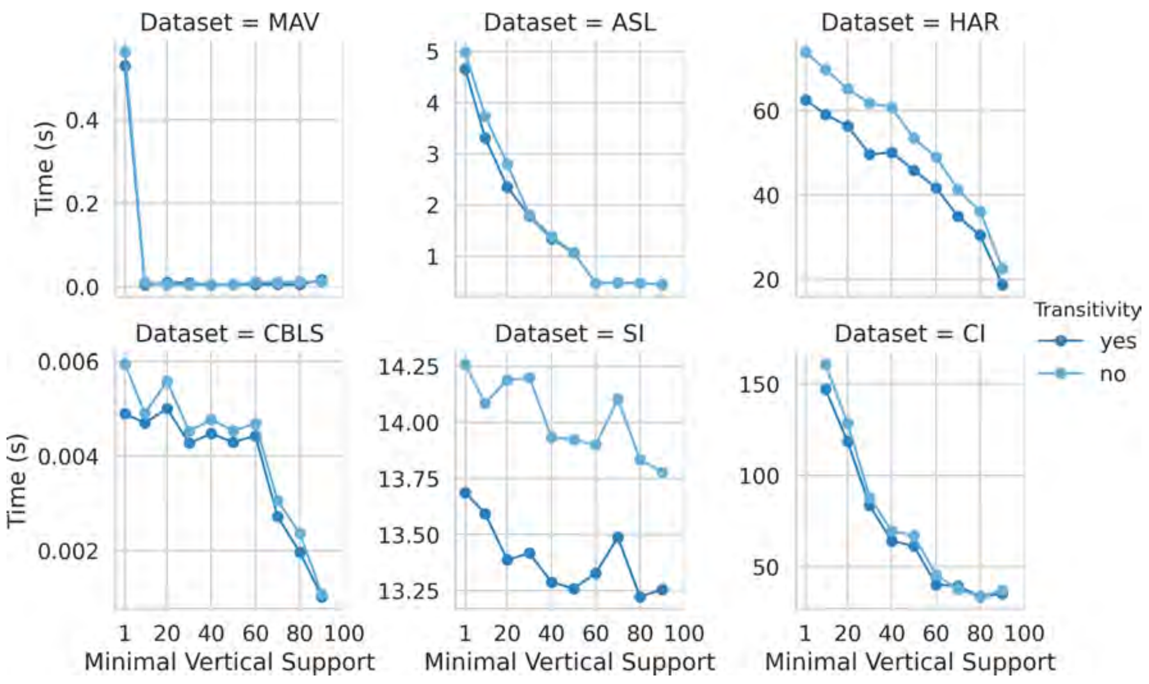
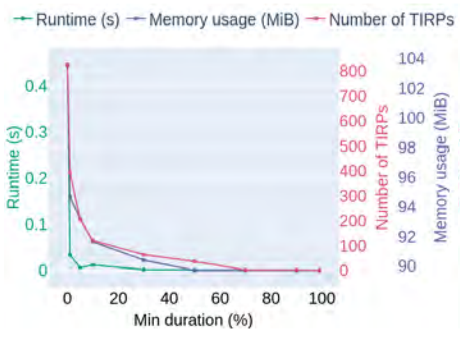


Figure 4.9: Efficiency by transitivity properties: transitivity vs. no transitivity (time in seconds; the scale of the y-axis is not uniform)

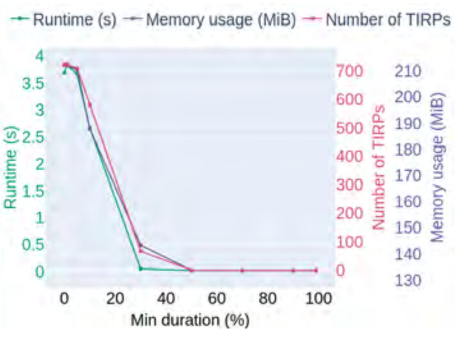
(shown on the x-axis).

As it can be observed, epsilon sensitivity depends on the dataset. For datasets where, the information was collected via sensors (i.e. the Mavlab dataset (MAV) and HAR datasets), the epsilon curve was smoother than for datasets where the information was collected in other ways. ASL and CI show significant increases in time, numbers of patterns found and memory usage. This is not only because these are the datasets with the highest number of time interval sequences (61370 and 3488553, respectively, see Table 5.2), but also because of the characteristics of the data. In the case of the ASL dataset, the data have no missing values, and there is also a marked temporal separation between samples. This means that for epsilon values smaller than the temporal separation between the samples, only the before and equal relations are found. If epsilon is increased, the algorithm finds more relations, and therefore more TIRPs, and this requires more time and memory. In contrast, for the CI dataset, varying the epsilon does not affect the number of patterns found, and in fact, the number of patterns fluctuates between 28 and 39. This is because many time interval events have a short duration, and increasing the epsilon affects little as much to the number of new TIRPs that it finds, as to the time, or to the memory.

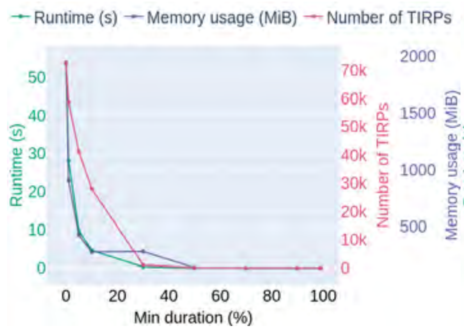
There also exists a value of epsilon that maximises or minimises the number of



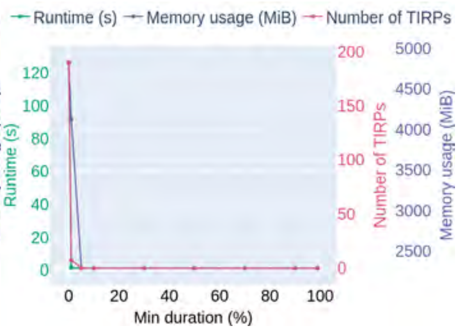
(a) MAV



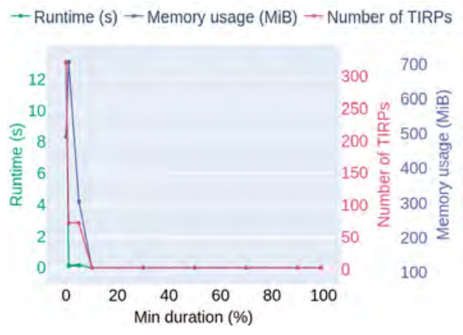
(b) ASL



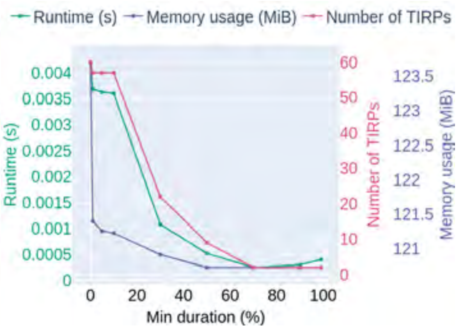
(c) HAR



(d) CI

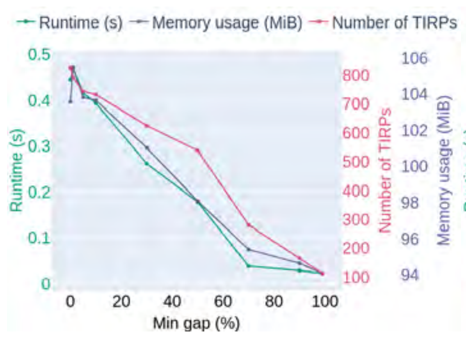


(e) SI

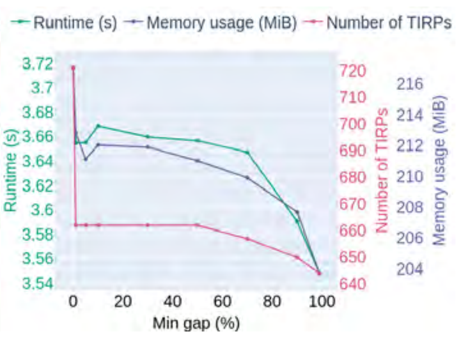


(f) CBLS

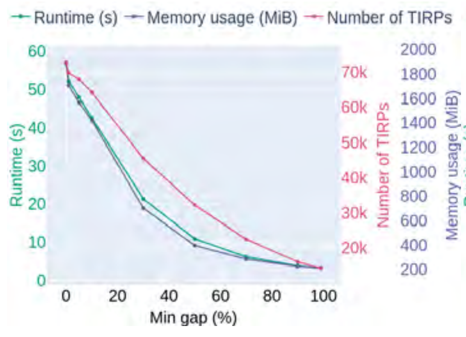
Figure 4.10: Efficiency with application of the min duration constraint. Note that multiple y-axes have been used to represent time, memory and the number of frequent TIRPs in the same graph. The x-axis represents the different values of the minimum duration as a percentage (0%,1%,5%,10%,30%, 50%,70% , 90% and 99%) of the maximum time interval in the dataset



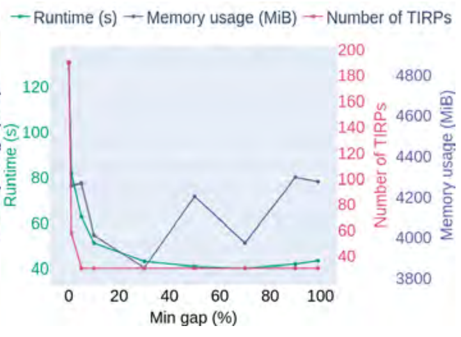
(a) MAV



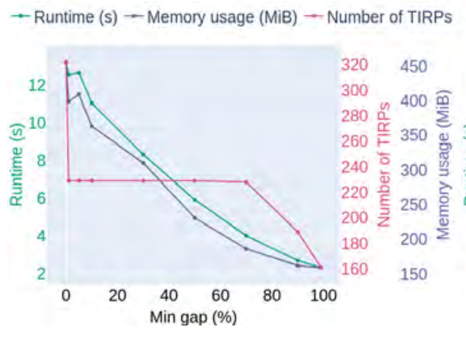
(b) ASL



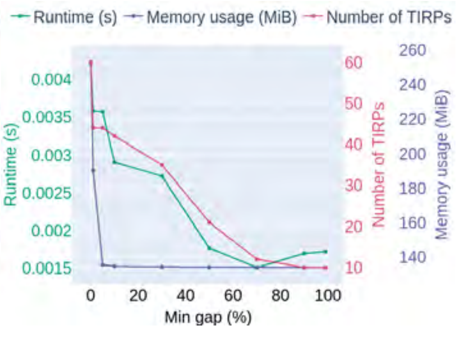
(c) HAR



(d) CI



(e) SI



(f) CBLS

Figure 4.11: Efficiency with application of the min gap constraint. Note that multiple y-axes have been used to represent time, memory and the number of frequent TIRPs in the same graph. The x-axis represents the different minimum gap values as a percentage (0%,1%,5%,10%,30%, 50%,70% , 90% and 99%) of the maximum time interval in the dataset

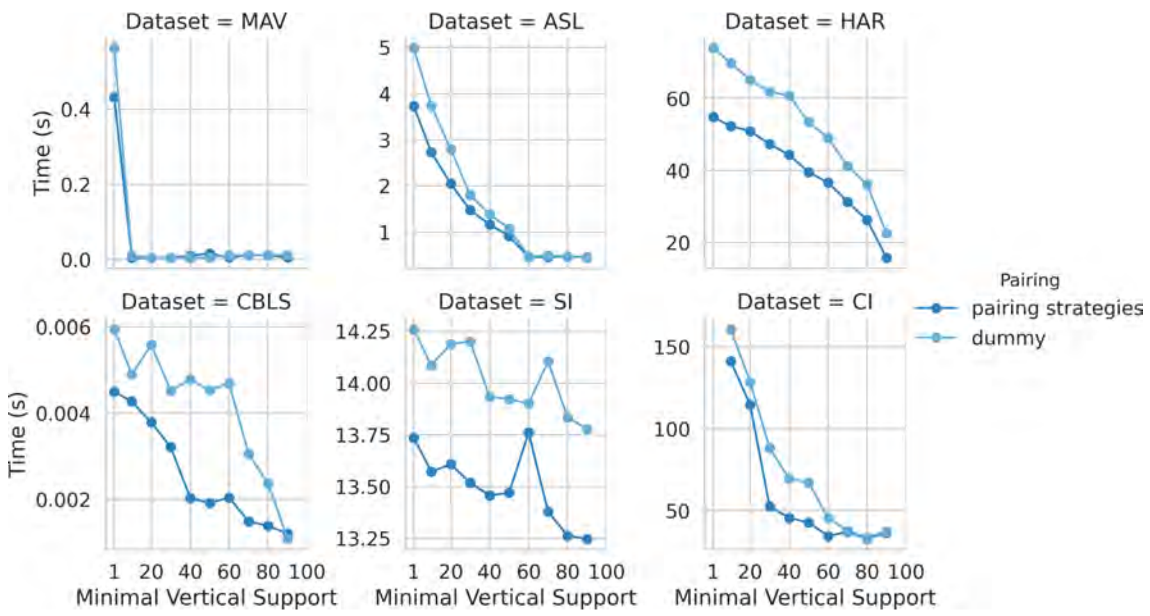


Figure 4.12: Efficiency with pairing of events: dummy approach vs. vertTIRP (time in seconds)

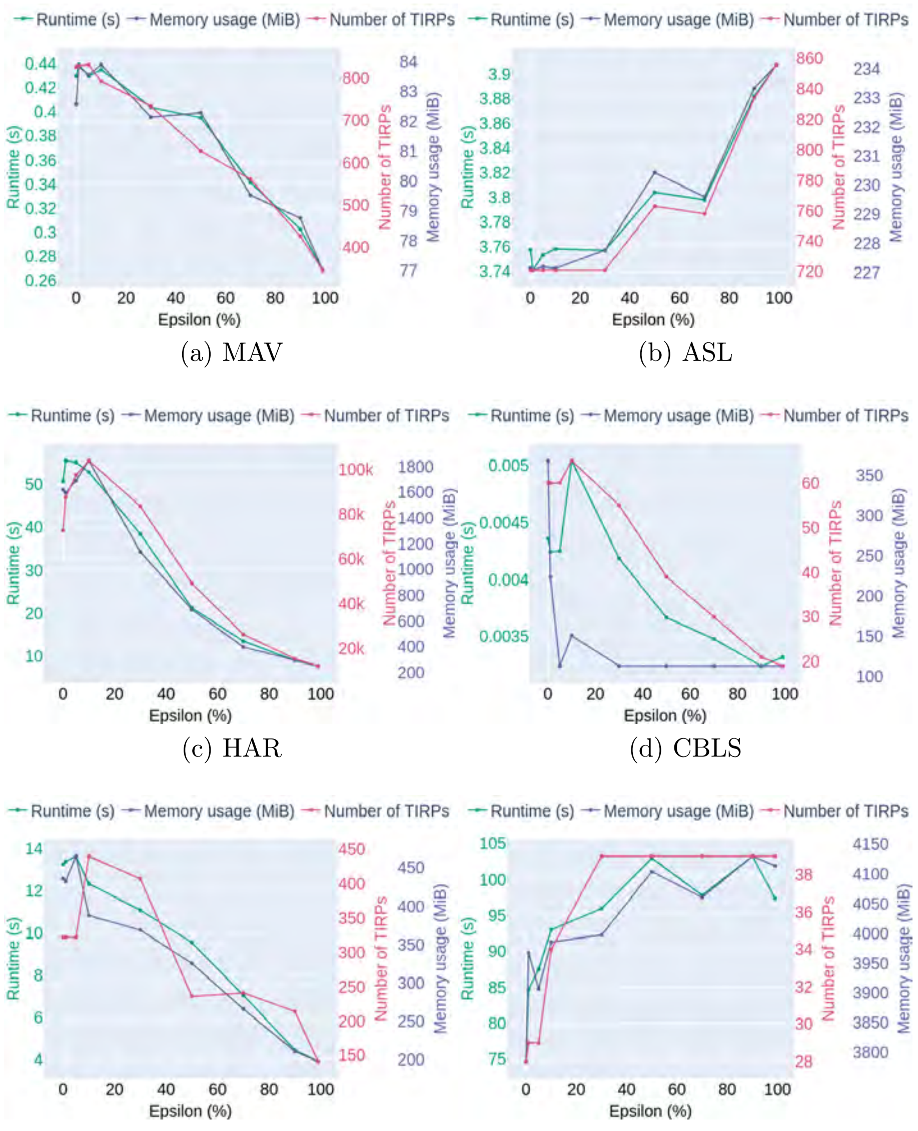
TIRPs discovered for each dataset. In four of the six datasets, the epsilon value that maximises the number of discovered TIRPs is around 10. Finally, it can be observed that the number of TIRPs and the time and memory required are linked, as expected.

Finally, regarding the complexity analysis, since the exact cost of the vertTIRP algorithm is complex to calculate (as for any TIRPs mining algorithm), its cost is estimated both analytically and experimentally in this section.

The complexity of the algorithm depends on the following factors: (i) the size of the TIRPs (s); (ii) the vocabulary size ($V = \Sigma$); (iii) the number of sequences (NS); (iv) the average number of time interval events per sequence (TS); and (v) the number of infrequent patterns (IP) for a given TIRP size, to quantify the reduction of candidates.

As the vertTIRP algorithm generates candidates and applies the join operation to each of them, first the cost of the join operation is calculated, then the cost of generating the candidates is calculated, and finally, all these costs are summed out together.

The join algorithm (Algorithm 7) contains a nested loop for time interval events that is inside the loop of sequences. The cost of the tour around the time interval events is $TS-1+TS-2+...+TS-(TS-1)$, which when simplified upwards becomes $(TS-1)*TS$, and hence $O(TS^2)$. The assignation of the temporal relation and



(a) MAV (b) ASL

(c) HAR (d) CBLS

(e) SI (f) CI

Figure 4.13: Epsilon sensitivity. Note that multiple y-axes have been to represent time (seconds), memory (MiB) and the number of frequent TIRPs in the same graph. The x-axis represents the different epsilon values as a percentage (0%,1%,5%,10%,30%, 50%,70% , 90% and 99%) of the maximum time interval in the dataset

the increment in the support operations are O(1). Finally, since the time interval event loop is executed the same number of times as the number of sequences in the dataset, the time complexity of the join operation is $O(NS * TS^2)$.

The time complexity of candidate generation is $\sum_{i=1}^{s} V^i$. When vertTIRP reduces the number of candidates with infrequent patterns IP, it actually reduces V in IP units, resulting in a cost of $\sum_{i=1}^{s}(V - IP)^i$. This reduction in candidates spreads to all children in the branch. The time complexity of each loop can be considered as $O(LogLogn)$ if the loop variables are reduced/increased exponentially by a constant amount. Hence, the time complexity of candidate generation is $O(loglog(V^s))$.

If combining the two costs, it can be found that the approximate theoretical cost of the vertTIRP algorithm is $O(loglog(V^s) * NS * TS^2)$.

The analytical cost of the algorithm is compared with an empirical result by using synthetic datasets. Figure 4.14 shows how time and memory behave while increasing the number of sequences, the number of time interval events per sequence, or the vocabulary size. It can be observed that the theoretical calculation is confirmed: time grows as a linear function of the number of time interval sequences, as an exponential function of the number of time interval events per sequence, and logarithmically depending on the size of the vocabulary. Memory behaves in the same way as time.

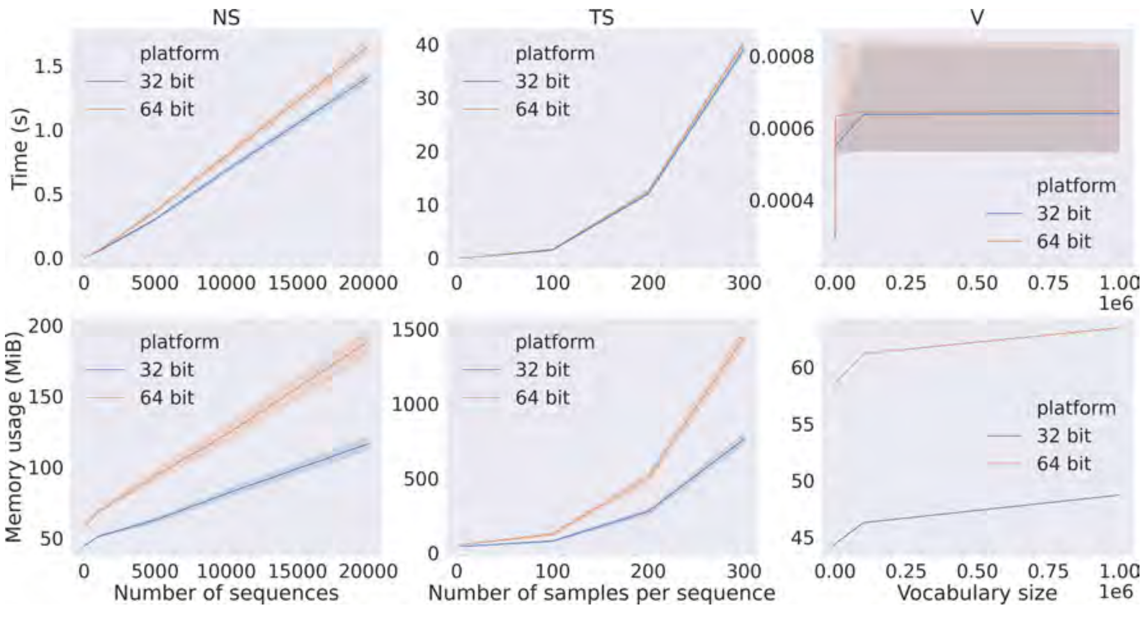


Figure 4.14: Complexity analysis of the vertTIRP algorithm in terms of time, memory and vocabulary size

4.5 Discussion

As far as efficiency is concerned, the use of a vertical representation of TIRPs (giving a reduction in cost of a factor of up to 20 compared to state-of-the-art alternatives) with transitivity among temporal relations and pairing strategies (reating a time reduction of up to 10%) gives a reduction in the computational cost that opens the way for addressing big data problems. Although the transitivity property contributes to this time reduction, the difference is less significant than that due to the incorporation of pairing strategies, which reduce the computation time by about 10%. The transitivity is particularly useful when the more frequent temporal relations in the dataset are those that return fewer possible relations from the transitivity table, i.e. e,b, and m. As more temporal relations offers a cell of the transitivity table, more time is required to check them, although this result can be modified by sorting the temporal relations based on their frequency and which depend of the dataset. In this regard, more noticeable was the improvement in time by transitivity with the sensor-based datasets, due to the presence of frequent b,m relations. The pairing strategy, which was simple, easy to implement, and tailored to the datasets, seemed to provide a powerful mechanism for reducing the search space.

The minimum gap and the minimum duration user constraints are useful when there is an interest in discarding events that last less than X time units or looking for patterns with a minimum interval of Y time units. For instance, in the example involving diabetes, there is an interest in looking for events that lasted less than one hour and where there was a minimum interval of two hours from the administration of the bolus. These also contribute to increasing the performance of the algorithm, but the number of patterns may be affected.

The value of epsilon allows for variability between events. While the user constraints limit the search space, the value of epsilon is responsible for two events being paired via one temporal relation or another. Based on the results, it can be observed that for each dataset there is a value of epsilon that maximises or minimises the number of the TIRPs discovered and hence the computation time and memory usage. The value of epsilon depends on the impression or the time delay of the sensor reading. In practice, most of the time this value is unknown. In this case, it could be found empirically. For example, suppose the final objective of TIRPs mining was for classification purposes. In that case, it should be kept the value of the epsilon that gives greater precision or ROC_AUC value in classification. On the other hand, if the final objective is the analysis of the found TIRPs, perhaps some utility measures should be implemented to know the usefulness of the found patterns, and keep the epsilon value that generates the greatest number of useful patterns.

Regarding the patterns discovered, on the one hand, it could be verified that the left contain relation is a discriminant relationship between the "SITTING" activity and all the other activities in the HAR dataset. For example, the pattern <"BodyAcc_b" "GravityAcc_a", l> with vertical support 20 only appears in the "SITTING" activity, where "BodyAcc_b" represents the acceleration of the person's moderate movement and "GravityAcc_a" represents the force of gravity mild. And on the other hand, it is worthy of commenting on some of the patterns extracted from the Chronic Illness dataset that were found to be quite logical and interesting. For example, an overlap of stressed and walking events <"stressed" "walked", o > may mean that walking is conducive to de-stressing. Or a pattern such as < "Fibromyalgia" "Joint_pain" "Naproxen", eoo >, which means that the person was registering joint pain and fibromyalgia for several days, which overlapped with the event of taking the Naproxen. In this case, it would be logical to think that Naproxen helped eliminate joint pain and fibromyalgia. The pattern < "Joint_pain" "Migraine" "tired", eff >, which suggests that joint pain in conjunction with migraine makes you feel tired. Or the pattern < "precip_intensity" "good sleep", c > might mean that heavy rainfall makes us sleep well.

4.6 Summary

Regarding a new efficient TIRPs mining algorithm, two important aspects have been under investigation over the last two decades in TIRPs mining: the first is efficiency, and the second is ambiguity in the discovered patterns. The present thesis is concerned with both of these issues. A proposal for a robust definition of temporal relations for TIRPs without ambiguities together with a vertical representation of TIRPs that exploits the transitivity property between temporal relations, pairing strategies and user-controlled constraints to increase the efficiency. Regarding the robustness of the discovered patterns, the meet, finished by, starts, equals, contains and overlap relations are disambiguated by redefining the meet and overlap relations. These definitions are based on the use of an epsilon value that also enables the management of noisy data. In addition, the left contains relation from [28] was adapted, which proved to be necessary when $\epsilon > 0$. A revised transitivity table for temporal relations was proved to be valid when using epsilon approaches in the definition of temporal relations; however, the epsilon value impacts the number of patterns found and memory usage. Given this, the epsilon approach should be regarded as a hyperparameter that can be adjusted for each application field, with certain consequences in memory usage and the number of patterns found.

Concerning efficiency, vertTIRP outperforms other state-of-the-art algorithms in terms of time. It scales well for small supports and large datasets. A vertical representation of patterns in conjunction with the transitivity property and a pairing strategy provided a significant reduction in time and improved the efficiency of

TIRPs mining. Pairing strategies reduced the computation time by about 10%.

Chapter 5

Data preparation as an external component of learning algorithms: TA4L

This chapter presents the algorithm TA4L, which copes with the third objective of this thesis, related to raising awareness of pre-processing tasks required for TIRP mining.

5.1 Problem statement

Given multivariate time series with missing values, where samples gather information about n variables in q situations or sequences, a minimum duration of an interval δ (e.g. in seconds), an alphabet Σ, and a maximum gap constraint max_gap, the aim is to provide LSTISs, one per sequence $LSTIS_1$, ..., $LSTIS_q$, where variable values have been abstracted to symbols in the alphabet, annotated with time intervals bounded in $[s, e]$ (with $\delta \leq e - s \leq max\_gap$), and sorted according to time-interval boundaries and alphabet symbols.

5.1.1 Input parameters

Each sequence sid $\in [1, q]$ of the multivariate time series, has a total of n_{sid} samples. Each sample $Y(sid, tid)$ comprises the values of the n variables for a given sequence sid and time tid, that is, $Y(sid, tid) = \langle y_1(sid, tid), \ldots, y_n(sid, tid)\rangle$, where $y_i(sid, tid)$ is the value of the i variable of the sample. Moreover, for the purposes of this work, $y_v(sid)$ denotes all of the values corresponding to variable v in a given sequence sid (i.e. signal).

The minimum duration of an interval can be expressed in any time scale (e.g.

seconds, hours, days). The max_gap constraint should be consistent with the max_gap scale.

The alphabet Σ by default is defined in $[A, Z]$, but any other alternative or vocabulary is also possible. An important issue regarding the complexity of the problem to be tackled is its size $|\Sigma|$.

5.1.2 Output: LSTISs

Several notations related to LSTISs should be considered first.

Definition 5.1.1. Symbolic time-interval. A symbolic time-interval I is a triple $I =< s, e, v.sym >$, composed of a start time (s), an end time (e), and a symbol v.sym where v is a variable name and $sym \in \Sigma$.

Time scales for the start and end times are application-dependent, as stated above. The same alphabet is used for all of the variables. Therefore the prefix v $(v.sym_j)$ is used with different symbols corresponding to the different variables. Eventually, different alphabets Σ_v could also be used, and thus $\Sigma = \bigcup_v \Sigma_v)$.

In the definitions below, I.s, I.e, and I.sym are used when the start time, end time and symbol of an interval are referred to.

Definition 5.1.2. A symbolic time-interval sequence, $IS = \langle I^1, I^2, ..., I^k \rangle$ represents a sequence of symbolic time-intervals I^i.

Given that there are n variables and n_{sid} samples per variable, a multivariate time series can be represented by a IS of length $k = (n * n_{sid})$.

To sort time intervals, usually, relational operators are used. In the present work, rather than comparing two time-intervals with an exact operator, this thesis looks for an approximate (epsilon) approach (following [4]). The epsilon approach allows for a certain variability in the temporary boundaries of events caused by noisy data [28]. The epsilon approach is implemented by means of the $=^\epsilon$ and $<^\epsilon$ operators.

Definition 5.1.3. Quasi-equal "$=^\epsilon$". Two time-points t^i and t^j are quasi-equal, $t^i =^\epsilon t^j$, if $|t^i - t^j| \leq \epsilon$.

Definition 5.1.4. Precedes "$<^\epsilon$". Time-point t^i precedes time-point t^j , $t^i <^\epsilon t^j$, if $t^j - t^i > \epsilon$.

Definition 5.1.5. Follows "$<^\epsilon$". Time-point t^i follows time-point t^j, $t^i >^\epsilon t^j$, if $t^i - t^j > \epsilon$.

The relational operators $=^\epsilon$ and $<^\epsilon$ are used to sort time-interval events into a sequence.

Definition 5.1.6. A lexicographical symbolic time-interval sequence $LSTIS$ is a symbolic time-interval sequence, IS=$\langle I^1, I^2, ..., I^{(n*n_{sid})}\rangle$, in which the elements are sorted in order of the start and end times using the relations $<^\epsilon$, $=^\epsilon$ and the symbols with a lexicographical order, that is: $\forall I^i, I^j \in IS(i<j)$:
$((I^i_{.s} <^\epsilon I^j_{.s}) \vee$
$(I^i_{.s} =^\epsilon I^j_{.s} \wedge I^i_{.e} <^\epsilon I^j_{.e}) \vee$
$(I^i_{.s} =^\epsilon I^j_{.s} \wedge I^i_{.e} =^\epsilon I^j_{.e} \wedge I^i_{.sym} < I^j_{.sym}) \vee$
$(I^i_{.s} =^\epsilon I^j_{.s} \wedge I^i_{.e} =^\epsilon I^j_{.e} \wedge I^i_{.sym} = I^j_{.sym} \wedge |I^i_{.e} - I^i_{.s}| = max\_gap)$.

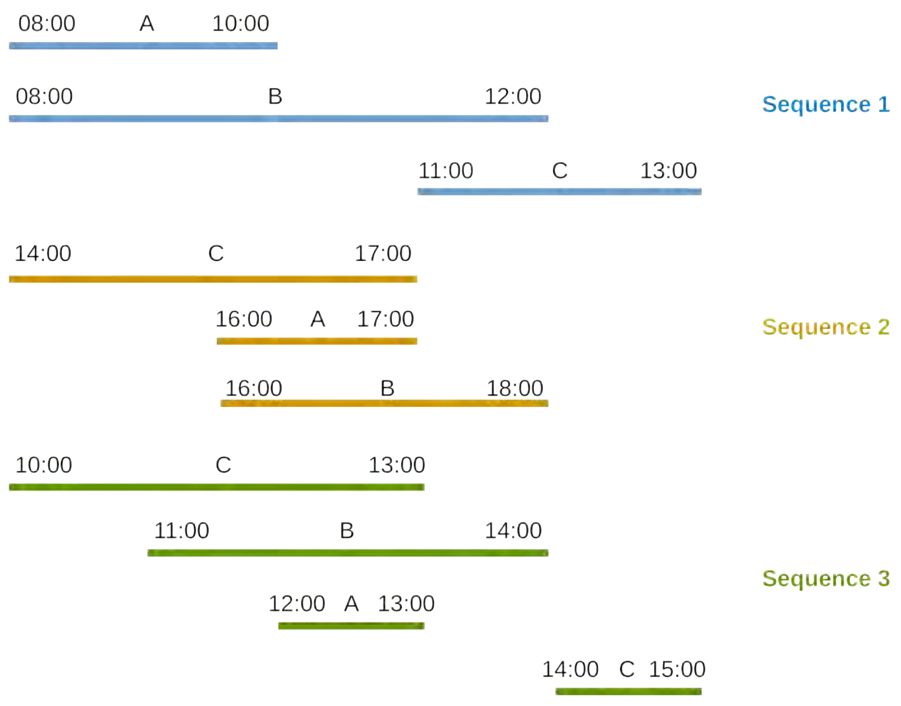


Figure 5.1: Graphical representation of LSTISs

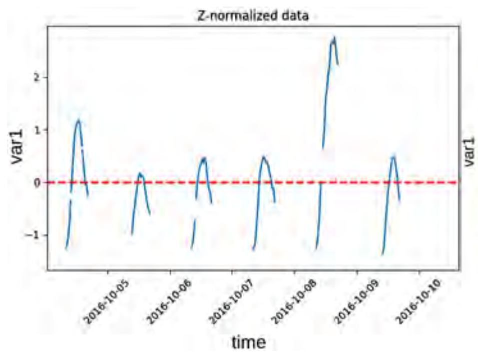
For example, ⟨⟨ <8:00, 10:00, var1.A>, <8:00, 12:00, var1.B>, <11:00, 13:00, C>⟩, ⟨ <14:00, 17:00, var1.C>, <16:00, 17:00, var1.A>, <16:00. 18:00, var1.B>⟩, ⟨ <10:00, 13:00, var1.C>, <11:00, 14:00, var1.B>, <12:00, 13:00, var1.A>, <14;00, 15:00, var1.C>⟩⟩ is a LSTISs that contains three sequences, all of them from the same variable var1. The first sequence is composed of three symbolic time-intervals, ⟨ <8:00, 10:00, var1.A>, <8:00, 12:00, var1.B>, <11:00, 13:00, var1.C>⟩. Figure 5.1 shows a graphical representation of this example.

5.2 The TA4L algorithm

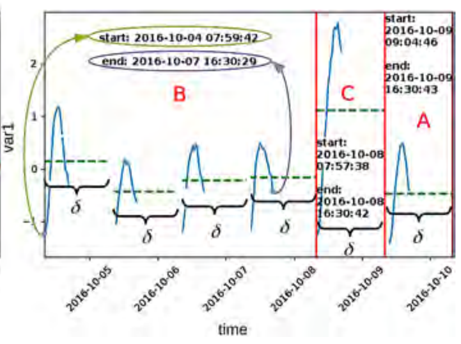
The inputs of the TA4L algorithm include multivariate time series MTS, the maximum gap max_gap allowed between two consecutive time intervals, and the minimum duration of time intervals dur. The algorithm returns a list of $LSTISs$, where each position will be an LSTIS corresponding to a certain sequence.

The first step of the algorithm is to initialize the list of LSTISs. Then for each sequence, the algorithm constructs an $LSTIS_{seq}$ and adds it to LSTISs. If the variable is numeric first, a Z-normalization of this variable is performed. Then each frame is discretized and concatenated with the next frame, based on the maximum gap and the symbol. If the variable is discrete, segmentation and concatenation are performed, also depending on the maximum gap and the symbol. And finally, in the case we decide not to use a sorted insertion to $LSTIS_{seq}$, the $LSTIS_{seq}$ must be sorted. The FDCS and SCS functions are detailed in Appendix C.2. The fundamentals of the proposal are technically in-deep introduced in the remaining of this section.

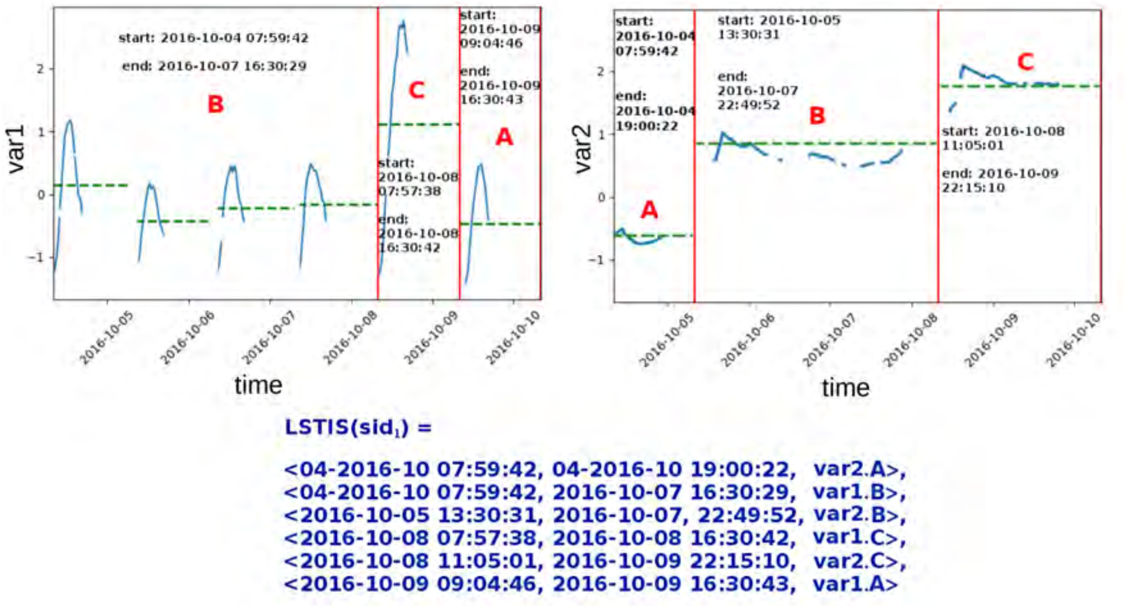
```
Algorithm 8: TA4L
   input: MTS: multivariate time series
          max_gap: the maximum gap constraint
          dur: duration of the interval
 1 output: LSTISs: a list of LSTISs for each sequence
 2 $LSTISs = \{\}$
 3 for each $MTS_{seq} \in MTS$ do
 4     $LSTIS_{seq} = \{\}$
 5     for each $var \in MTS_{seq}$ do
           /* Temporal abstraction                                    */
 6         if $MTS_{seq}(var)$ is numeric then
 7             $Z_{seq}(var)$ = Z-norm($MTS_{seq}(var)$)
               /* Framing, Discretization, Concatenation, and Sequence
                  Generation (FDCS)                                    */
 8             $LSTIS_{seq}$ = FDCS($Z_{seq}(var)$,max_gap,dur,$LSTIS_{seq}$)
 9         else
               /* Segmentation, Concatenation and Sequence Generation (SCS)
                  */
10             $LSTIS_{seq}$ = SCS($MTS_{seq}(var)$,max_gap,$LSTIS_{seq}$)
11         $LSTIS_{seq}$ = sort($LSTIS_{seq}$) ;        // if not using a sorted insertion
12     $LSTISs$ = append($LSTIS_{seq}$,$LSTISs$)
```



(a) Z-normalized signal



(b) Temporal abstraction



LSTIS(sid_1) =

```
<04-2016-10 07:59:42, 04-2016-10 19:00:22,  var2.A>,
<04-2016-10 07:59:42, 2016-10-07 16:30:29,  var1.B>,
<2016-10-05 13:30:31, 2016-10-07, 22:49:52, var2.B>,
<2016-10-08 07:57:38, 2016-10-08 16:30:42,  var1.C>,
<2016-10-08 11:05:01, 2016-10-09 22:15:10,  var2.C>,
<2016-10-09 09:04:46, 2016-10-09 16:30:43,  var1.A>
```

(c) The resulting LSTISs

Figure 5.2: TA4L over multivariate time series of two variables. The algorithm parameters used in this example have been: Σ=3, δ=24h, and discretization=SAX.

An example of TA4L is summarised on the Figure 5.2 for two variables, var1 and var2. This example is used along the section to illustrate the details of the algorithm.

5.2.1 Normalization

Given a sequence sid and the values of a variable v, $y_v(sid) = \langle y_v(sid, 1), \ldots, y_v(sid, n_{sid})\rangle$, the normalization steps applies the normalization method using the

Z-method, obtaining $y_v^Z(sid) = \langle y_v^Z(sid, 1), \ldots, y_v^Z(sid, n_{sid})\rangle$. Figure 5.2a shows the normalisation of the variable var1. In the figure, it is possible to observe that there is several missing information for the variable.

5.2.2 Framing

Instead of reducing the dimensionality of the input data based on the number of samples w as is done in SAX, TA4L reduces it based on a duration δ, with time interval boundaries defined in known data (i.e. avoiding defining the boundaries in time points corresponding to missing data).

Given a normalized variable v in the scope of a sequence sid, $y_v^Z(sid) = \langle y_v^Z(sid, 1), \ldots, y_v^Z(sid, n_{sid})\rangle$, data is divided into time intervals of duration δ in the framing step, where the start time $I.s$ and end time $I.e$ is marked by the first and last point known inside the δ frame.

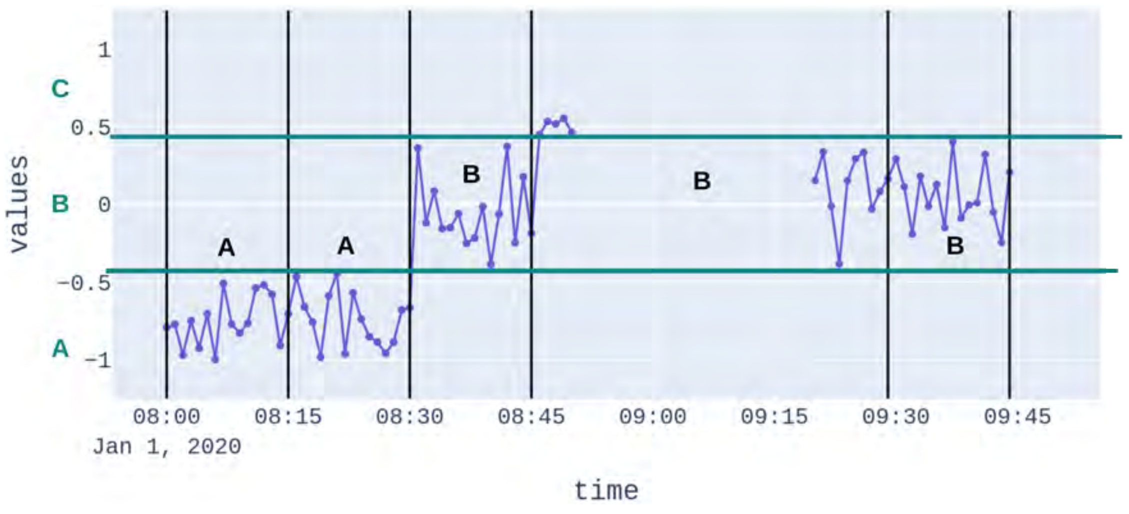
For example, in Figure 5.2b, the start time of the real first time-interval is 2016-10-04 07:59:42, and the real end time is 2016-10-04 19:00:22 (not the 2016-10-05 07:59:42 marked by δ). Values from 2016-10-04 19:00:22 to 2016-10-05 07:59:41 are missing. There is no symbol assigned to the time interval at this stage but the mean values along with δ. Therefore, the output of the fragmentation stage is $y_v^f(sid) = \langle y_v^f(sid, 1), \ldots, y_v^f(sid, m_{sid})\rangle$, where $y_v^f(sid, j) = \langle s_j, e_j, mean_j\rangle$, $|s_j - e_j| \leq \delta$, and $s_j < s_{j+1} \forall j$. Observe that $n_{sid} - m_{sid}$ is the data reduction achieved in this stage.

The fact that TA4L defines the time interval boundaries with known values greatly impacts discovering TIRP patterns in the future and presents some advantages from previous approaches as SAX[70]. For example, Figure 5.3 illustrates the difference between applying SAX and TA4L to a signal. SAX splits the original signal into five segments with the same amount of samples; its fourth segment contains unknown values, representing a 45' time interval. In contrast to SAX, TA4L will characterize the fourth time interval with a 5' duration. Consequently, a TIRPs mining algorithm could eventually find the pattern "A before B before C" thanks to TA4L, but that will not be the case when using SAX.

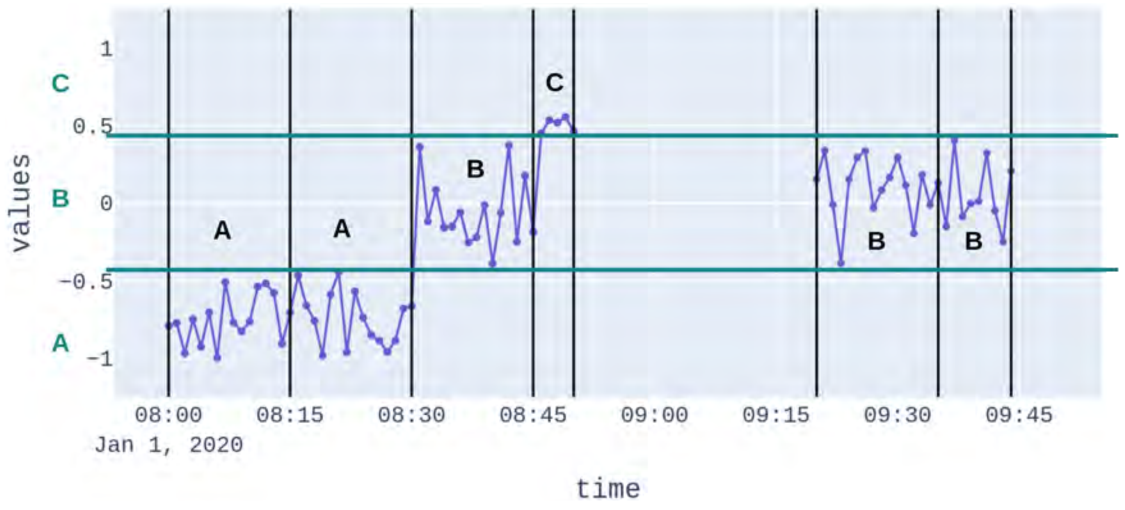
5.2.3 Discretization

Discretization is the process of replacing the $mean_j$ value of the time intervals with $sym_j \in \Sigma$. To that end, TA4L uses by default the SAX approach [70], but other methods are also available, such as EWD [80] and KBTA [75]. Therefore, the output of the fragmentation stage is $y_v^d(sid) = \langle y_v^d(sid, 1), \ldots, y_v^d(sid, m_{sid})\rangle$, where $y_v^d(sid, j) = \langle s_j, e_j, v.sym_j\rangle$, $|s_j - e_j| \leq \delta$, $s_j < s_{j+1} \forall j$, and $sym_j \in \Sigma$.

If discretization is performed following the SAX approach, it must first determine



(a) SAX



(b) TA4L

Figure 5.3: SAX and TA4L over a signal. The algorithm parameters used in this example have been: Σ=3, δ=15 minutes.

the breakpoints. Breakpoints are a sorted list of numbers $B = \beta_1, ..\beta_{\Sigma-1}$ such that the area under a N(0,1) Gaussian curve from β_i to $\beta_{i+1} = \frac{1}{|\Sigma|}$ (β_0 and β_Σ are defined as $-\infty$ and $+\infty$, respectively). These breakpoints are determined by looking them up in a statistical table computed from the data. The statistical table is used here to produce symbols with equiprobability. Secondly, the mapping from $mean_i$ to α_j, the j_{th} element of the alphabet (i.e., $\alpha_1 = A$ and $\alpha_2 = B$), is obtained as follows: $sym_i = \alpha_j$, iff $\beta_{j-1} \leq mean_i < \beta_j$. That means that all means that are below the smallest breakpoint are mapped to the symbol "A", all means greater than or equal to the smallest breakpoint and less than the second smallest breakpoint are mapped to the symbol "B" and so on.

If the EWD discretization approach is used, the range of variables is divided into Σ breakpoints of equal size. Finally, when the KBTA approach is used, the breakpoints are acquired from a domain expert.

5.2.4 Segmentation

If the type of a v variable is discrete, temporal abstraction consists of a segmentation process: symbolic time intervals are generated with the start time of the interval corresponding to the time a discrete value (symbol) appears, and the end time when there is a symbol change, or when there is a maximum distance with the start time of δ unit times. The output of the segmentation stage is equivalent to the discretization step of the numerical variables, $y_v^d(sid) = \langle y_v^d(sid, 1), \ldots, y_v^d(sid, m_{sid})\rangle$, where $y_v^d(sid, j) = \langle s_j, e_j, v.sym_j\rangle$, $|s_j - e_j| \leq \delta$, $s_j < s_{j+1} \forall j$, and $sym_j \in \Sigma$. Although, in this case, the alphabet Σ is induced from the data, the Σ notation for the sake of simplicity is kept.

5.2.5 Sequence generation

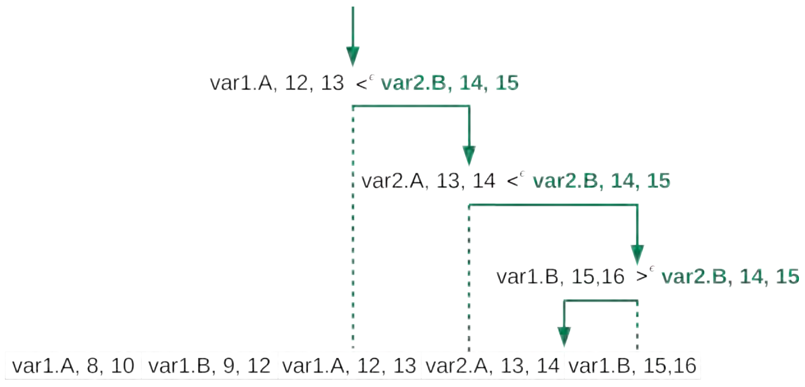
Sequence generation is based on a data structure, named sorted list, with two main components: an LSTISs and the pointer to the last inserted element into the list. The pointer is initialised to 1 when the first interval of each variable is inserted. The remainder intervals are inserted following the binary search method [101] adapted for LSTISs. Moreover, at the moment of the time-interval insertion into the LSTISs a concatenation process is applied.

5.2.5.1 Binary search for LSTISs

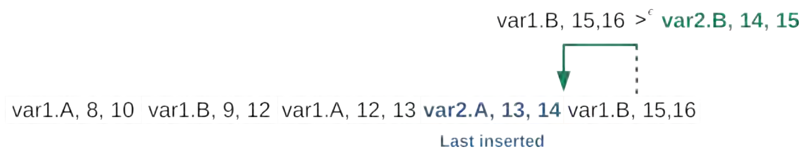
Binary search [101] works on sorted arrays, and it is theoretically the optimal and the fastest in practice search algorithm, which order is $O(log2N)$. The algorithm compares an element in the middle of the array with the value to be inserted. If the value to be inserted matches the element, its position in the array is returned; if it is less than the element, the search continues in the lower half of the array; if it is greater than the element, the search continues in the upper half of the array. By doing this, each iteration of the binary search narrows the search interval by half of the search interval of its previous iteration.

In the present work, an adapted version of the binary search was employed to search the position into LSTISs to perform the insertion of time intervals. The difference is that (i) it is applied to an LSTISs; and (ii) it takes advantage of the fact that data arrive in sorted order by time, and (iii) it uses the $=^{\epsilon}$ and $<^{\epsilon}$ relational operators.

First, a binary search is applied to each LSTISs instead of to an array of integers. Secondly, each variable time-interval is processed according to time. Therefore, the insertion takes advantage of that to start the search from the last inserted position as shown in the Figure 5.4b, instead since the position 0 as shown in Figure 5.4a. Finally, time intervals are compared with $>^{\epsilon}$ and $<^{\epsilon}$ epsilon-relational operators instead of $>$ and $<$ relational operators.


var1.A, 12, 13 <ᵋ var2.B, 14, 15
var2.A, 13, 14 <ᵋ var2.B, 14, 15
var1.B, 15,16 >ᵋ var2.B, 14, 15
var1.A, 8, 10 var1.B, 9, 12 var1.A, 12, 13 var2.A, 13, 14 var1.B, 15,16

(a) Insert with an original binary search adapted to LSTISs


var1.B, 15,16 >ᵋ var2.B, 14, 15
var1.A, 8, 10 var1.B, 9, 12 var1.A, 12, 13 var2.A, 13, 14 var1.B, 15,16
Last inserted

(b) Insert with an improved binary search adapted to LSTISs

Figure 5.4: The difference between the original binary insert and an adapted binary insert that remembers the last inserted position applied to LSTISs.

5.2.5.2 Concatenation: Intervals of maximum duration

If there are two or more consecutive intervals with the same symbol, these intervals are concatenated into a single interval, where the start time is the time corresponding to the start time of the first interval and the end time is the time corresponding to the end time of the last interval. The advantage of that is not only storing the memory but also providing a short, reliable and meaningful summary of information to the final user. For example, in the var1 of Figure 5.2b, the first four intervals correspond to symbol B. Therefore, all of them are combined in a new time-interval where the start time is the start time of the first time-interval (2016-10-04 07:59:42), and the end time is the end time of the last time-interval (2016-10-07 16:30:29).

On the other hand, concatenation is controlled by the parameter max_duration, which aims to deal with particular situations of application domains. For example, for the following up of a person who suffers diabetes, data analysed to evaluate her state should be within one day ($max\_gap = 24h$).

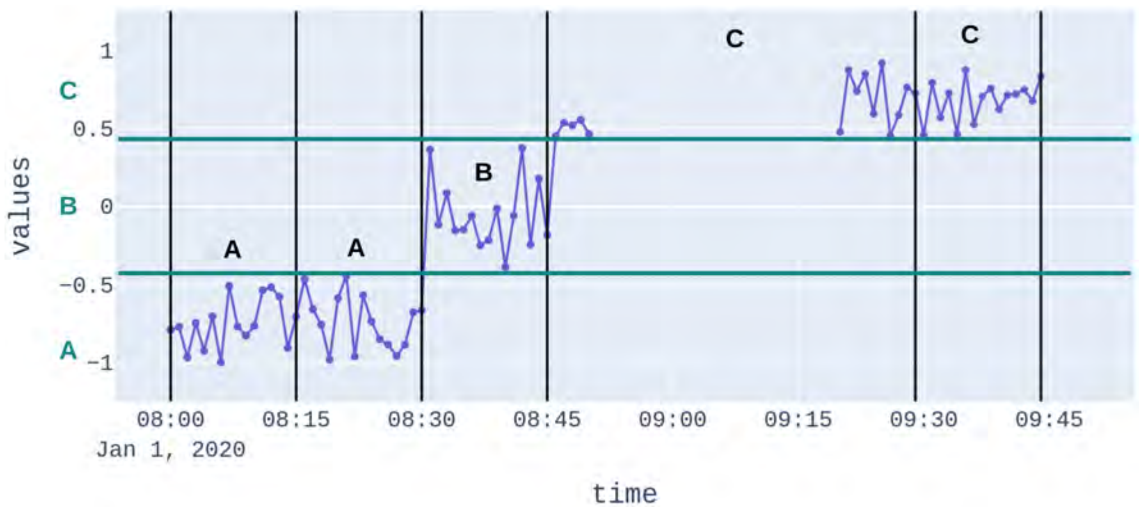
Therefore, two consecutive time-intervals of the same sequence and symbol sorted by start time $I_i(sid) =< s_i, e_i, v.sym >$ and $I_j(sid) =< s_j, e_j, v.sym >$, are transformed into a single interval $I_k(sid) =< s_i, max(e_i, e_j), sym >$, iff $(I_j.e - I_i.s) < max\_gap$ and $s_i <= s_j$.

This has some advantages when discovering TIRPs. For example, in Figure 5.5b, it can be observed that from 8.50 to 9:20, there is no data. If $max\_gap = 30'$, both would not be concatenated into a single interval, and as a consequence, a TIRPs mining algorithm could eventually find the pattern "a before b before c" thanks to the maximum gap constraint in the TA4L (see Figure 5.5b), but that will not be the case when using SAX (see Figure 5.5a).

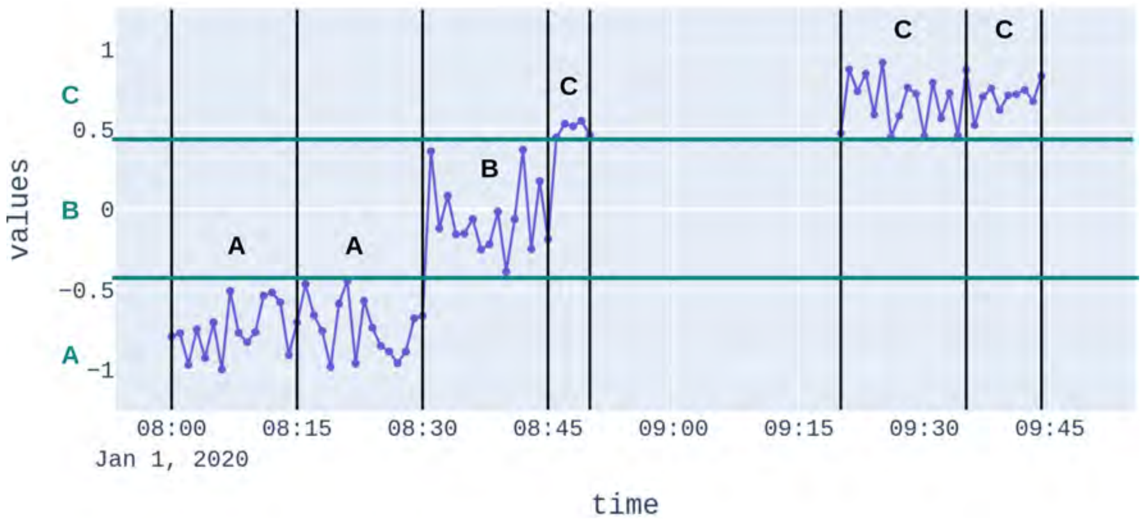
5.2.6 Data structures and parallel strategies

The basic procedure of TA4L presented in Figure 5.2 considers the sorted list data structure where time intervals are inserted in a sorted manner with an adapted version of binary search as soon as they are obtained. With this data structure, parallelism can be applied to sequences, i.e. the procedure of transforming time series to LSTISs is executed as an independent asynchronous thread for each sequence of the TA4L algorithm, as shown in the first image of Figure 5.6.

On the other hand, an alternative simple list data structure could be considered, and applying a sorting procedure in the end, opening the opportunity to other parallel strategies: parallelism can be applied apart from sequences (see the second image of the Figure 5.6) to the variables (see the third image of the Figure 5.6), i.e. the procedure of transforming a signal into LSTISs is executed as an independent



(a) SAX



(b) TA4L

Figure 5.5: Concatenation example over a signal. The algorithm parameters used in this example have been: Σ=3, δ=15 minutes, and $max\_gap = 30'$.

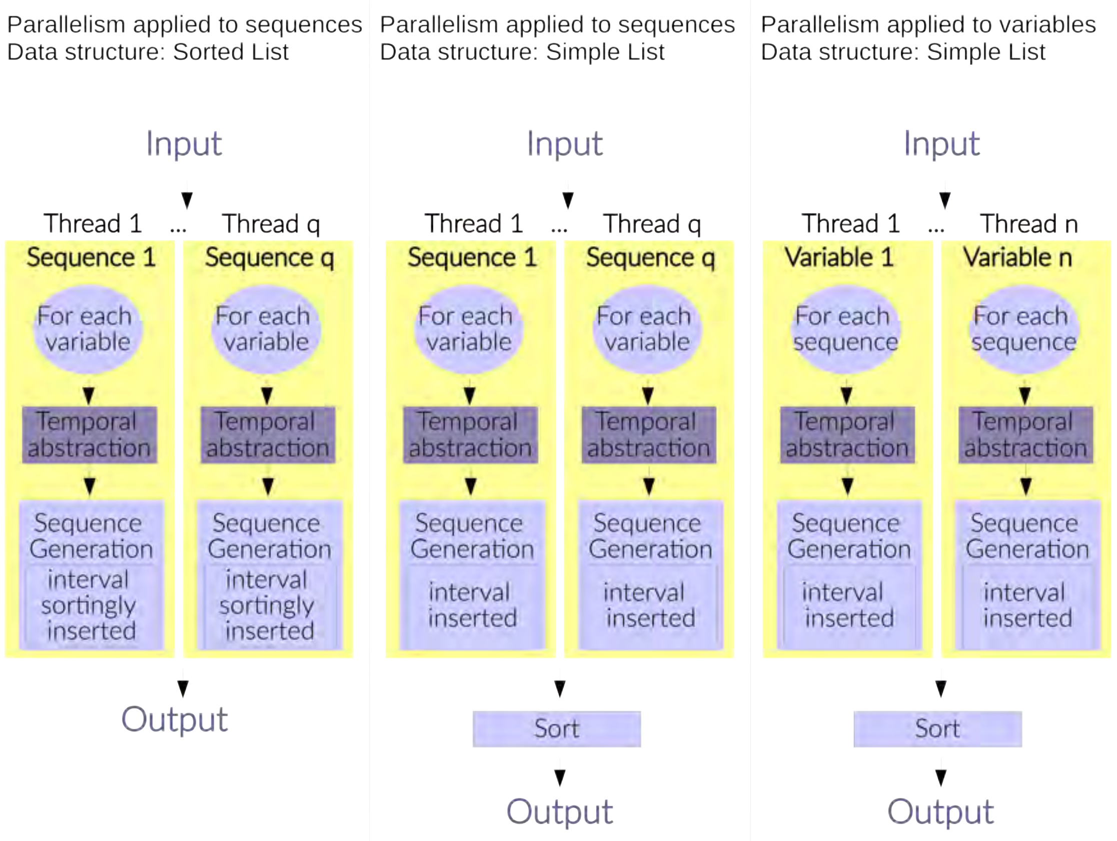


Figure 5.6: Parallelism strategies of TA4L

asynchronous thread for each variable in a sequence. Table 5.1 summarises all the parallelism strategies.

5.3 Experimental Evaluation methodology

In this section, we test TA4L with the aim to answer the following research question: How can we make explicit the embedded, ad-hoc pre-processes related to TIRPs mining algorithms when the input is multivariate time series (or time series) while offering an efficient solution for the required pre-processing and considering temporal restrictions established by the user?

Secondary research questions:

-How can we decide whether two consecutive points could be considered to be part of the same event (i.e. belonging to the same time-interval) or do not?

-What will be the difference between constructing time intervals based on the

Name	Data structure	Parallelism strategy
TA4L_sl	sorted list	binary insert that remembers the last inserted element (no parallelism)
TA4L_sl_paral_seq	sorted list	parallelism applied to sequences (first image of the Figure 5.6)
TA4L_se	simple list	sorts at the end (no paralellism).
TA4L_se_paral_seq	simple list	sorts at the end and parallelism applied to sequences (second image of the Figure 5.6)
TA4L_se_paral_var	simple list	sorts at the end and parallelism applied to variables (third image of the Figure 5.6)

Table 5.1: Parallelism strategies of TA4L and corresponding data structures.

time duration constraint instead of using a fixed number of samples from the dataset.

- Can an adapted version of binary search be useful when performing ordered insertions of TI according to the criteria of the LSTISs?

- How could a multivariate time series pre-processing algorithm be more efficiently parallelised to LSTISs?

5.3.1 Experimental platform for TA4L

The TA4L algorithm presented in this thesis were implemented in Python 3.7.6. The experiments were carried out on a virtual machine, Lubuntu 19.10, 64 bits, eight CPUs, and 12 GB of RAM.

5.3.2 Datasets

The experiments were conducted with synthetic and real datasets.

5.3.2.1 Synthetic datasets for the TA4L

To better explore the efficiency of the pre-processing algorithm presented in this thesis, numerous synthetic datasets are generated in which the following factors were considered: (i) Percentage of missing values (PM); (ii) Vocabulary size ($|\Sigma|$); (iii) Interval size (δ); (iv) Number of variables (n); (v) Number of sequences (q); (vi) Number of samples per sequence (n_{sid}). With values for PM of 1%, 25%, 50%, 75% and 99%; values for $|\Sigma|$ of 3, 10 and 20 (i.e. $\Sigma \in [A, C]; \Sigma \in [A, J]; and \Sigma \in [A, T]$); values for δ of 1%, 25% and 50% (this percentage is in respect to the total time); values for n of 3, 167, 334 and 501; values for q of 3, 167, 334 and 501; and values for n_{sid} of 3, 167, 334 and 501 is experimented.

To analyse the complexity of the worst case, datasets with numeric variables are generated.

5.3.2.2 Real datasets for the TA4L

The following public datasets were used to test the pre-processing algorithms for TIRP mining presented in this dissertation:

- Childhood Blood Lead Surveillance (CBLS) [97] dataset, which contains data on blood levels of lead in children from between 1995 and 2015, from several U.S. states and local health departments.
- COVID-19 in Spain ($COVID_{esp}$) [102] dataset, which contains data on daily increments split by age and gender of the COVID-19 pandemic in Spain.
- Suicides in India (SI) [96] dataset, which contains annual suicide records of all states of India with various parameters from between 2001 and 2012.
- Mavlab (MAV) dataset [92], which contains activities related to daily living, collected using the MavLab testbed during March and April of 2003. This dataset captured an inhabitant's interactions with an intelligent home through sensors placed in different rooms.
- COVID-19 by country - Daily update ($COVID_{all}$) [103] dataset, which contains data on daily increments of the COVID-19 pandemy at the country level.
- Human Activity Recognition (HAR) dataset [94] built from the recordings of 30 subjects performing activities of daily living while carrying a waist-mounted smartphone with embedded inertial sensors. This dataset can be found in the UCI Machine Learning Repository [95].

dataset	q	n	n_{sid}	$\|\Sigma\|$	PM	MTPS (s)
CBLS	44	2	18.136	-	2.82	540742254.545
$COVID_{esp}$	11	8	51.727	-	0.0	4697018.182
SI	38	3	134.526	4.0	0.0	347068800.0
MAV	11	2	116.818	30.5	0.0	4444193.636
$COVID_{all}$	188	14	332.761	188.0	0.918	25056000.0
HAR	30	562	343.3	6.0	0.0	876.288

Table 5.2: Dataset description. If a dataset does not have discrete variables, $|\Sigma|$ is marked with "-"; otherwise, $|\Sigma|$ is the mean vocabulary size of the discrete variables.

The datasets mentioned above are characterised on Table 5.2 in terms of the factors considered for the synthetic datasets, plus the mean total time per sequence (MTPS, in seconds).

5.3.3 Experimental setup

Different implementations of the TA4L have been implemented to test the different methods proposed in this thesis, as shown in Table 5.1. Moreover, the state-of-the-art approach (Figure 2.1) has been implemented according to the following configuration:

- Z-normalization
- Framing using a given amount of samples w
- Discretization using the SAX approach
- Sequence generation with concatenation, without any gap constraint (i.e. $max\_gap = \infty$)

The state of the art approach is labelled as the Baseline in the experiments. Is it possible to observe that these configurations are close to the SAX approach; however, in a sensitivity analysis, the discretization approaches of EWD and KBTA are also tested, therefore covering a broad representation of the currently available best approaches.

Three type of experimental configurations were defined:

1. Performance in the function of the algorithm parameters. This experimental configuration is directed to test the performance of the algorithm based on its parameters, such as $|\Sigma|$, δ and max_gap, and compare the results with the

state of the art approaches (i.e. Baseline). In so doing, the efficiency of the data structures proposed in TA4L is tested under the different parallelism strategies. This experiment has been applied to both synthetic and real datasets.

- To analyze the sensitivity to the size of δ, $|\Sigma|$ was set to the maximum value 20, and PM was set to 1. The hypothesis is: as the smaller is the δ, the larger the time and memory consumption.
- To analyse the sensitivity to the size of $|\Sigma|$, PM and δ was set to 1. The hypothesis is: as larger is $|\Sigma|$, more time and memory consumption will be required.
- To analyse the sensitivity to the length of *max_gap*, $|\Sigma|$ was set to 3. Since the effect of the *max_gap* depends on the number of nulls in the database, PM has been tested with 1%, 25%, 50%, 75% and 99%. The hypothesis is: as smaller is the *max_gap*, the larger the number of time intervals found, and consequently larger the memory and time consumption.

2. Performance-based on the discretization method. The symbol assignation from SAX, EWD and KBTA methods was compared in the Baseline and in the TA4L (TA4L_sl strategy) algorithms in the function of q, n_{sid} and $= |\Sigma|$ parameters to analyze the sensitivity of the discretization method. Note that when this thesis refers to applying SAX to TA4L, it refers only to the task of assigning the symbol (as explained in Section 5.2.3). The hypothesis is: there should not be a significant difference between the runtimes of the discretization methods (since all methods assign the symbol based on the range where the value falls), but regarding the memory consumption, SAX will be the method that consumes more memory, due to the storage of statistical tables, and the breakpoints determination.

3. Performance in the function of the dataset characteristics. This other configuration aims to test the algorithm's performance based on the dataset characteristics, such as PM, q, n and n_{sid}. This experiment has been applied to synthetic datasets. In particular, the PM analysis aims to validate the approach followed in TA4L in which time-intervals are fragmented according to known data.

   - To analyze the <u>PM sensitivity</u>, the n_{sid} and n was set to 501; $|\Sigma|$ to 20; q to 10; and δ to 1. The discretization method was SAX. The hypothesis is: since the algorithm constructs intervals of equal time size, while tackles not missing values inside them, as smaller is the PM values, more time and memory will need the algorithm because there will be more values to tackle inside each frame, showing an improvement over the Baseline, state-of-the-art, configuration.

- To analyze the n sensitivity, $|\Sigma|$ was set to 20; q, PM and δ to 1. The discretization method was SAX. The hypothesis is that: as more n, more time and memory will need the algorithm.
- To analyze the n_{sid} sensitivity, $|\Sigma|$ to 20; q, PM and δ to 1. The discretization method was SAX. The hypothesis is: as more n_{sid}, more time and memory will need the algorithm.
- To analyze the q sensitivity, $|\Sigma|$ was set to 20, δ and PM to 1. The discretization method was SAX. The hypothesis is: as larger is the q, more time and memory will need the algorithm.

In order to be compared to the baseline approach, the epsilon parameter was set to 0.

5.4 Results

In this section, the cost of the TA4L algorithm is estimated, both analytically and experimentally.

The complexity of analysis has been carried out according to the different component of the algorithm. Fist, Z normalization cost for one variable of a sequence is $n_{sid} - n_{sid} * PM$, which when simplified upwards (when $PM = 0$) becomes $O(n_{sid})$.

The cost of fragmentation and symbol assignation (which is only applicable to numeric variables) is complex. Here although the algorithm loops through the values only once $O(n_{sid})$, every $100/\delta$ times it averages the accumulated values and assigns the symbol value, where δ is represented as the percentage of time concerning 100 per cent of the total time. The symbol assignation cost is $O(|\Sigma|)$. Therefore the cost of fragmentation and symbol assignation is $O(n_{sid} + (100/\delta) * |\Sigma|)$. If the symbol of the previous I is the same as the symbol of the actual I, the end of the previous I will be updated with the actual end. This assignation has constant time complexity $O(1)$.

Z normalization $O(n_{sid})$, fragmentation with symbol assignation $O(n_{sid}+(100/\delta)* |\Sigma|)$, and symbol insertion into the LSTISs with its' corresponding start and end times are repeated for every variable n and sequence q. Therefore, the cost of the algorithm for numeric variables is $q * (n * (n_{sid} + (100/\delta) * |\Sigma| + insertionCost) + sortingCostIfAny)$, and for discrete variables is $q*(n*(insertionCost)+sortingCost IfAny)$.

The final cost depends on the symbol insertion and sorting methods:

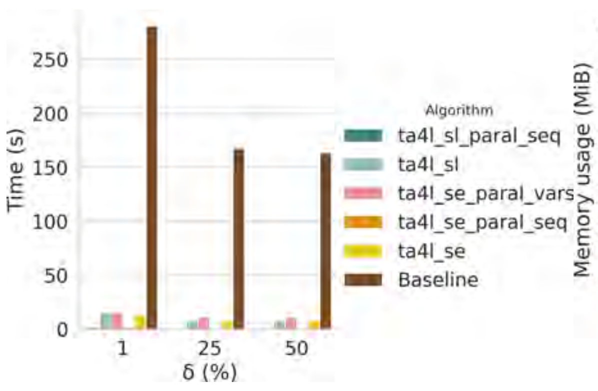
- Sorted insert. It was opted for an adapted version of binary search, that instead of searching from the beginning, searches from the last inserted position to find the position to insert the I. It is not a big complexity reduction since only the value of $n * n_{sid}$ is reduced, which is the total number of elements into a sequence of the LSTISs to explore. Therefore let us consider the cost of an adapted version of binary search to be $O(log(n * n_{sid}))$, such as in a typical binary search. Here and throughout, $log(n * n_{sid}) = log_2(n * n_{sid})$ denotes the binary logarithm of $(n * n_{sid})$. Then, every time the algorithm inserts a value into position x, all the r remaining elements from the position x to the end of the list have to be moved 1 position. Therefore the cost of the sorted insert of one element is $O(r * log(n * n_{sid}))$. In the worst case, there will be $(n * n_{sid})$ insertions (i.e. $O((n * n_{sid}) * (rlog(n * n_{sid}))))$, and r will be $(n * n_{sid})$ (i.e. $O((n * n_{sid}) * (n * n_{sid})log(n * n_{sid}))$ or simplified $O((n * n_{sid})^2 log(n * n_{sid})))$. So the cost of the algorithm in the case where all the variables are discrete is $O(q * n * ((n * n_{sid})^2 * log(n * n_{sid})))$. And the cost of the algorithm in the worst case, when all the variables are numeric (which corresponds to the final cost of the algorithm) is $O(q * (n * (n_{sid} + (100/\delta) * |\Sigma| + (n * n_{sid})^2 * log(n * n_{sid}))))$.

- Append elements normally and sort the list at the end. Append element at the end of the list, has constant time complexity i.e., $O(1)$. Since this operation will be executed n_{sid} times for each sequence and variable, the $insertionCost$ in this case is $O(n_{sid})$. The complexity of sorting the list with mergesort at the end is $O((n * n_{sid})log(n * n_{sid}))$, where $(n * n_{sid})$ is the total number of elements into a sequence of the LSTISs. So the cost of the algorithm in the case all the variables are discrete is $O(q * ((n * n_{sid}) + (n * n_{sid}) * log(n * n_{sid})))$. And the cost of the algorithm in the worst case, when all the variables are numeric is $O(q * (n * (n_{sid} + (100/\delta) * |\Sigma|) + ((n * n_{sid})log(n * n_{sid}))))$.

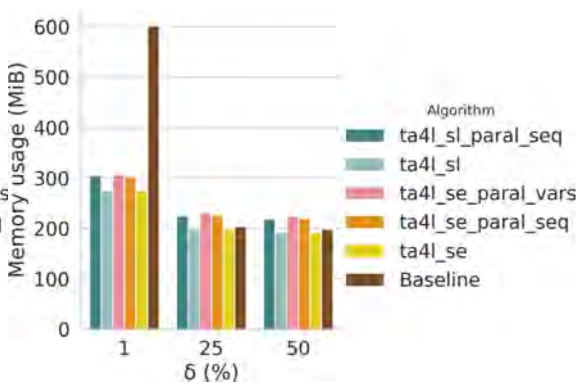
Regarding the algorithm behaviour, this thesis analysed experimentally the behaviour of algorithms based on input parameters, such as Σ, δ and max_gap.Figure 5.7 shows the results obtained with the synthetic datasets, and Figures 5.8 and 5.9 for the real ones.

It can be observed that in the case of both synthetic and real datasets, time and memory increase with increasing vocabulary size and with decreasing interval size. In real datasets (Figures 5.8 and 5.9) the increase in time and memory is most noticeable with large datasets, such as HAR and $COVID\_all$ datasets.

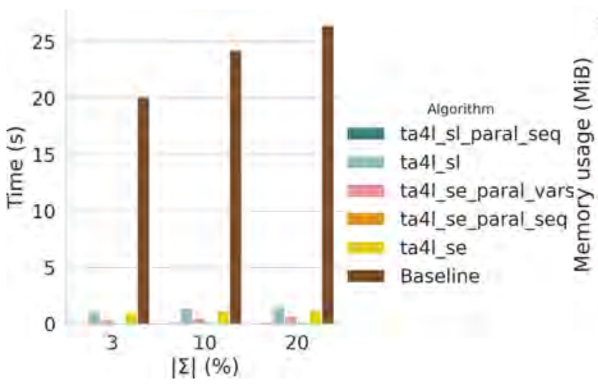
Multithread version by sequence (TA4L_sl_paral_seq and TA4L_se_paral_seq) consumes less time for both parameters (δ and $|\Sigma|$). For example in the case of synthetic datasets for interval size 1 (Figure 5.7a) the runtime is around 1.56 seconds, while for the rest of versions is from 12.74 up to 281.24 seconds. For $|\Sigma|$ 20 (Figure 5.7c) the runtime is around 0.14 seconds, while for the rest of versions is from 0.66



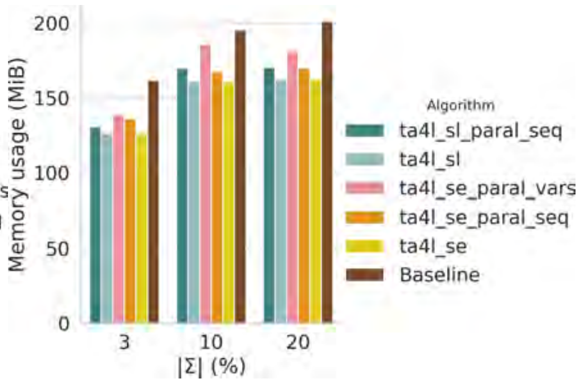
(a) Time vs δ



(b) Memory vs δ



(c) Time vs $|\Sigma|$

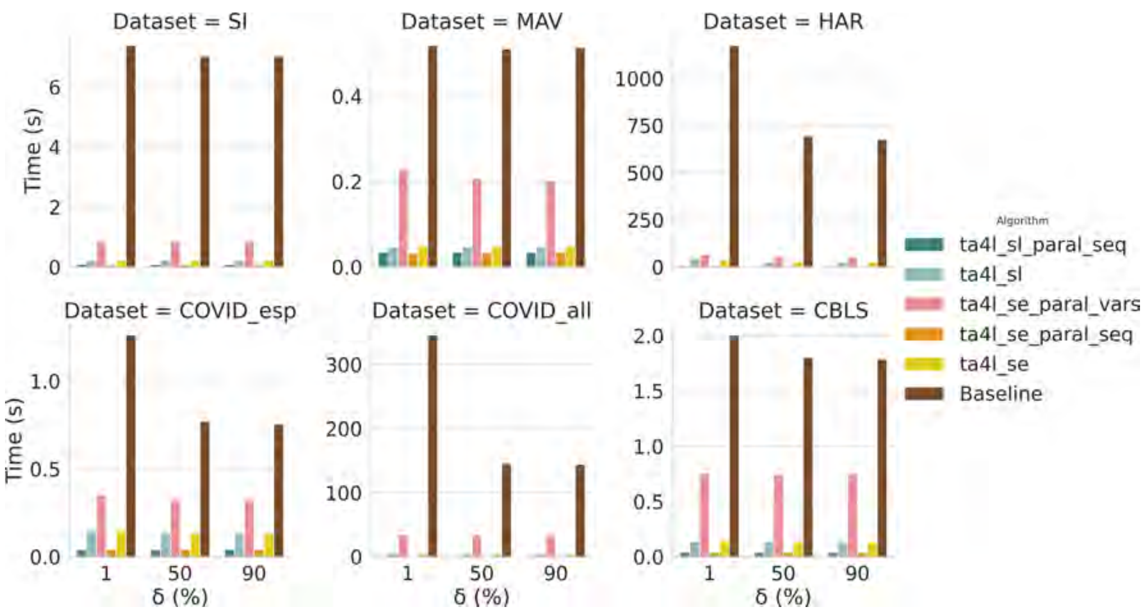


(d) Memory vs $|\Sigma|$

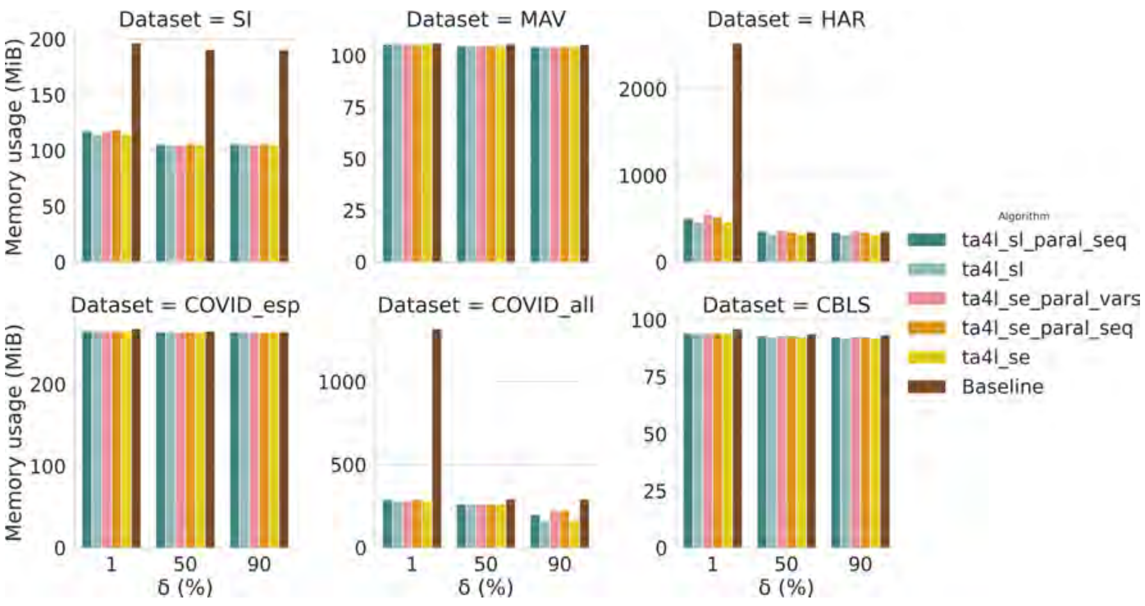
Figure 5.7: Synthetic datasets. How varying input parameters δ and $|\Sigma|$ affects the performance of the algorithms

up to 26.42 seconds. In the case of real datasets, it is worthy of analysing the largest dataset, which is the HAR dataset. In the case of HAR dataset with interval size one (Figure 5.8a) and for $|\Sigma|$ 20 (Figure 5.9a) the runtime is around 5.54 seconds, while for the rest of versions is from 38.69 up to 1175.67 seconds. For a single thread version and δ one and $|\Sigma|$ 20, the data structure that sorts at the end is slightly faster rather than a sorted list version, which runtime is 1.22 vs 1.51 seconds in the case of $|\Sigma|$, and 12.75 vs 15.37 seconds in the case of δ for synthetic datasets, and 38.69 vs 43.74 seconds for the HAR dataset.

The parallel version by variables is the structure that consumes slightly more memory than the rest of the TA4L versions. For example, for $|\Sigma|$ 20, in synthetic datasets, the memory usage of TA4L_se_paral_var version is 181.14 MiB, while the rest of the TA4L versions consumes from 162.076 to 170.23 MiB. In the HAR dataset for δ one and $|\Sigma|$ 20, TA4L_se_paral_var consumes more memory (550 MiB) than the rest of the TA4L versions (that consume from 460MiB up to 527MiB). However,



(a) Time vs δ

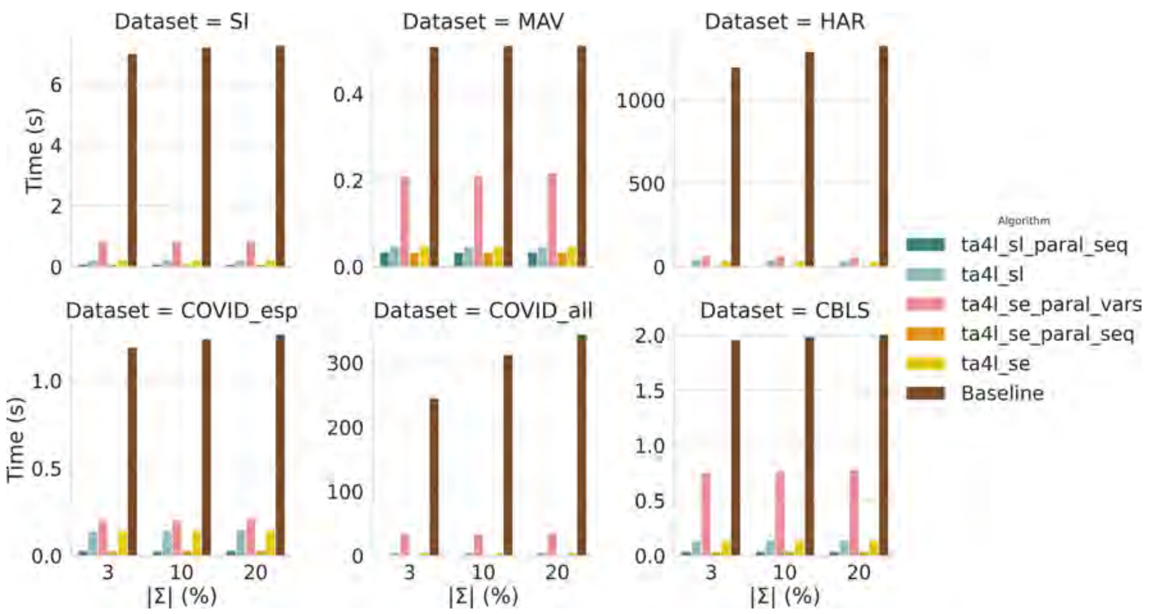


(b) Memory vs δ

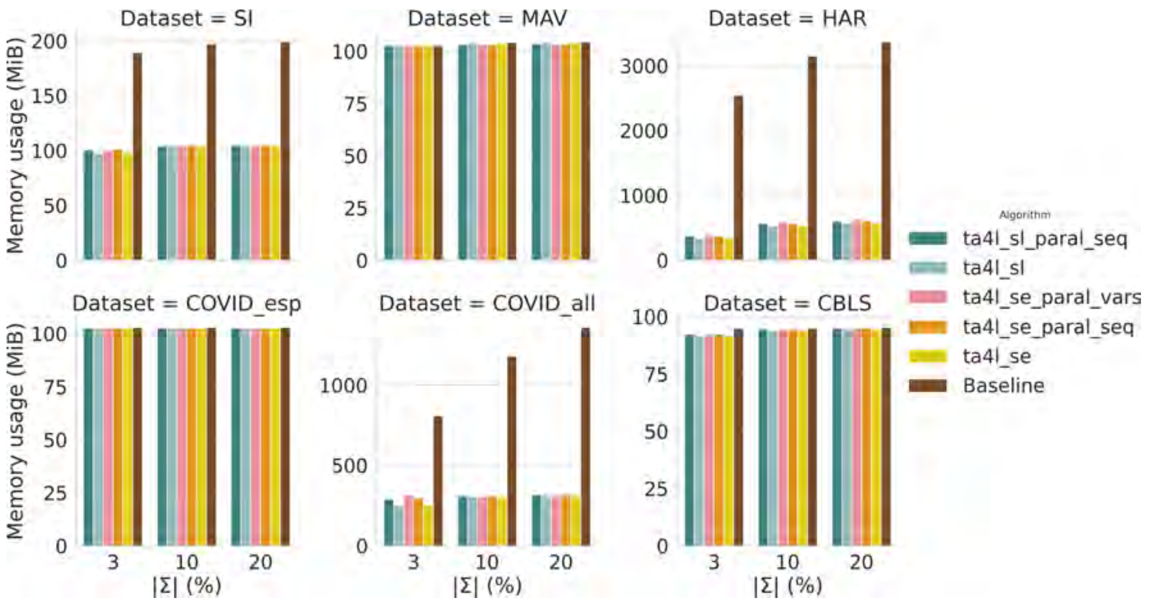
Figure 5.8: Real datasets. How varying the δ affects the performance of the algorithms

the memory consumption of the Baseline algorithm shoots up to 2532MiB.

In general, in all real datasets, TA4L is significantly faster than Baseline for all configurations, and in terms of memory consumption, TA4L is more efficient than



(a) Time vs $|\Sigma|$

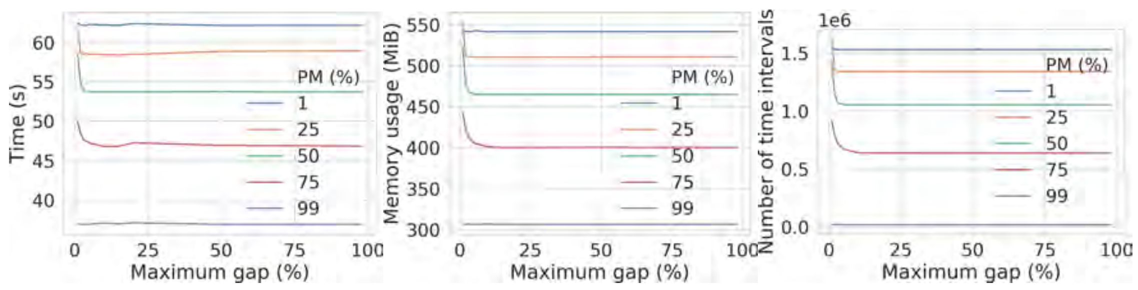


(b) Memory vs $|\Sigma|$

Figure 5.9: Real datasets. How varying the $|\Sigma|$ affects the performance of the algorithms.

Baseline for relatively large datasets, such as HAR, $COVID\_all$ and IS datasets.

Regarding the max_gap constraint (see Figure 5.10), on the one hand, it can be appreciated that time, and memory consumption is directly proportional to the



(a) Time vs *max_gap* (b) Memory vs *max_gap* (c) |I| vs *max_gap*

Figure 5.10: Synthetic datasets. How varying the *PM* and *max_gap* parameters affects the performance of the algorithms, where |I| is the total number of time intervals.

number of the constructed time intervals (|I|). The more |I|, the greater time and memory consumption. For small values of *max_gap*, there are more |I| (1624261 |I| for *max_gap* 1 and missing 25%), and for big values of *max_gap* there are less |I| (1341094 |I| for *max_gap* 10 and missing 25%). For example, in the case of 25% of missing values, for *max_gap*=1% the |I| is 1624261, the runtime is 61.7 seconds, and the memory usage is 552.7 MiB, while from the *max_gap*=10% on the |I| is 1341094, the runtime is 58.5 seconds, and the memory usage is 510.3 MiB. On the other hand, this effect is less pronounced in cases of extreme missing values, that is, if the percentage of missing data is too small (1 %) or too large (99 %). For example, in the case of 1% of missing values, for *max_gap*=1% the |I| is 1546397, the runtime is 62.5 seconds, and the memory usage is 543.5 MiB, while for the *max_gap*=10% the |I| is 1531279, the runtime is 62.2 seconds, and the memory usage is 541.42 MiB.

Regarding the performance according to the discretization method, this thesis presents the results of applying EWD, KBTA and SAX discretization methods to the TA4L and Baseline approaches, based on the dataset characteristics such as n, q and n_{sid}.

In the case of synthetic datasets, in general for small values of q, n, and n_{sid} SAX and KBTA consumes significantly less time and memory than the EWD approach, while for large values of q, n, and n_{sid} EWD consumes less memory and runtime than SAX (see Figures 5.11, 5.12 and 5.13). The baseline is significantly slower than TA4L (observe that time scales of plots in the figures are different in TA4L from Baseline, due to the big difference in the results achieved), and it also consumes more memory than the TA4L approach.

In the case of real datasets, TA4L generally consumes less time and memory than Baseline (observe the differences on the time scales of plots in the figures as happened with the synthetic datasets). Nevertheless, in TA4L implementations, the

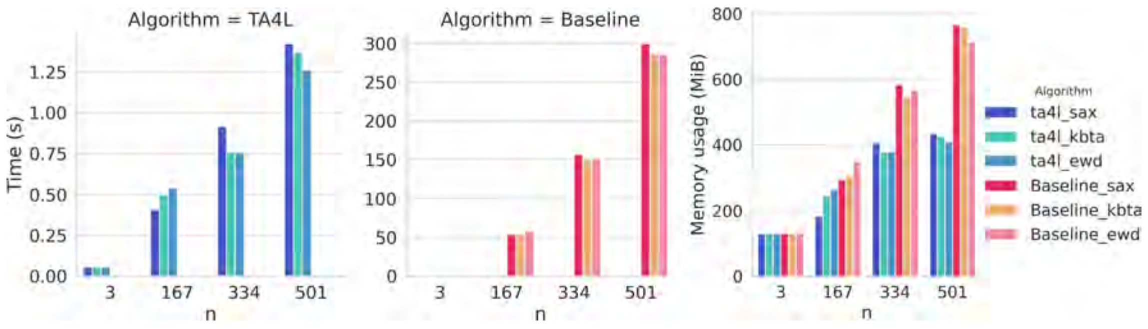


Figure 5.11: Synthetic datasets. Sensitivity to discretization methods while varying n.

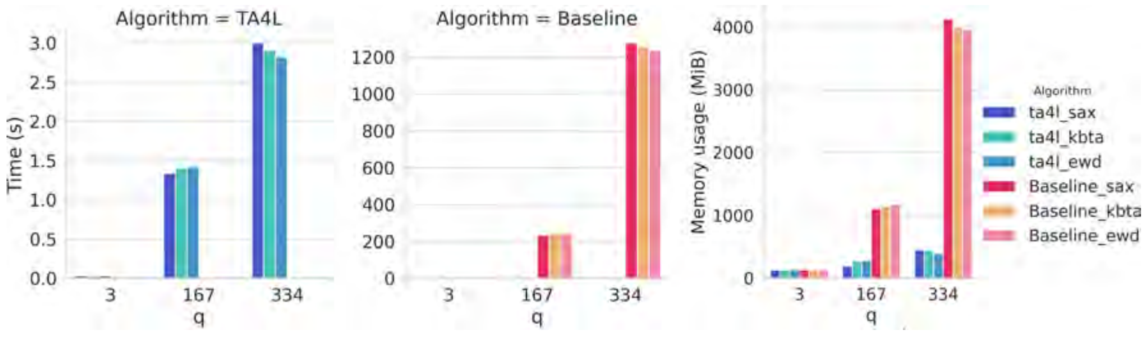


Figure 5.12: Synthetic datasets. Sensitivity to discretization methods while varying q

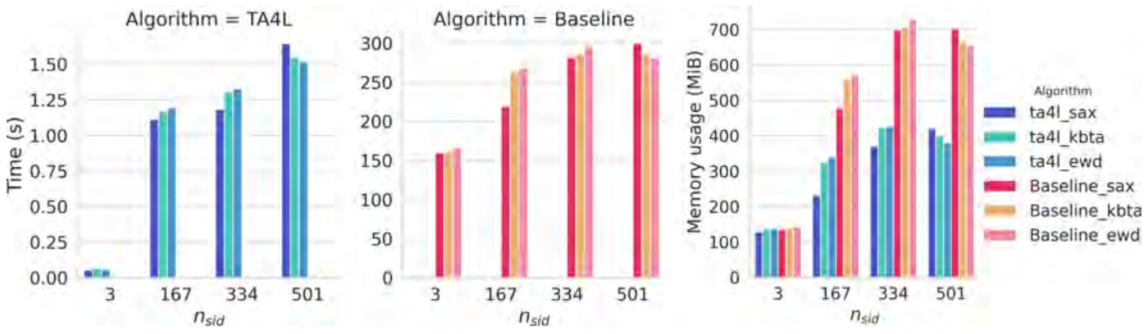


Figure 5.13: Synthetic datasets. Sensitivity to discretization methods while varying n_{sid}

behaviour of the discretization approach requires a deepening analysis.

For example, with respect to memory consumption, in the case of the HAR dataset, for large q, n, and n_{sid} values (Figures 5.14, 5.15 and 5.16), Baseline_KBTA is the one that consumes significantly more memory than Baseline_EWD. Whereas in the case of the COVID_all dataset, for large values of q, n and n_{sid}, Baseline_SAX consumes significantly more memory than Baseline_EWD. In the case of large datasets (such as HAR and COVID_all datasets), for small values of q, n and n_{sid}, TA4L_SAX is the least memory consuming than TA4L_EWD and TA4L_KBTA methods.

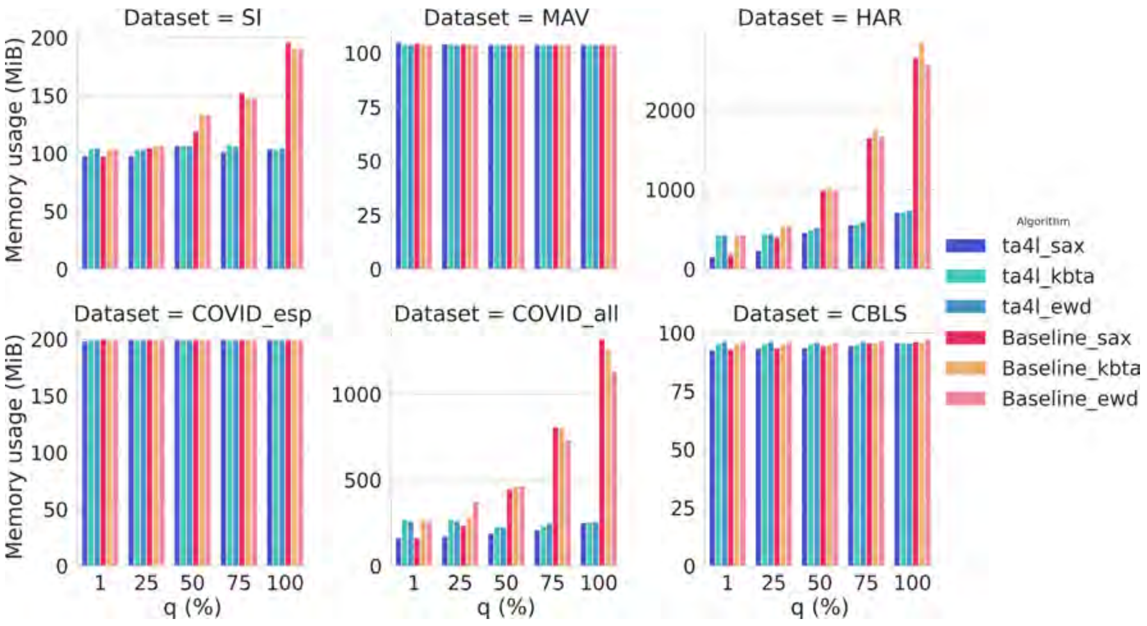


Figure 5.14: Real datasets. Sensitivity to discretization methods in terms of memory while varying q

Regarding the time consumption (Figures 5.17, 5.18 and 5.19), in the case of the COVID_all dataset, Baseline_EWD is significantly slower than Baseline_KBTA especially for n_{sid} experiments. Regarding the sensitivity to discretization in TA4L, in the case of the HAR dataset, in both n and n_{sid} experiments, SAX is significantly slower than the EWD and KBTA methods.

Finally, concerning the performance according to the dataset characteristics, this thesis analysed experimentally the cost of algorithms based on dataset characteristics, such as q, n, n_{sid} and PM.

In the case of q (Figures 5.20d and 5.20c), multithread version by sequence consumes less time for both data structures, which is around 3.8 seconds for q 501 (worst case). The state of the art approach is that consumes more time (732.19 seconds for q 501). However, the multithread version of TA4L by variables also presents a great runtime concerning other versions of TA4L, which is 35.22 seconds for q 500. Therefore, if there are many sequences and few variables, it is not useful to parallelize by variables. In the case of n (Figures 5.20b and 5.20a), n_{sid} (Figures 5.21b and 5.21a), and PM (Figure 5.22), the two multithread versions by sequence also present the shortest runtime (0.134 seconds for n 501 and 1.39 seconds for PM

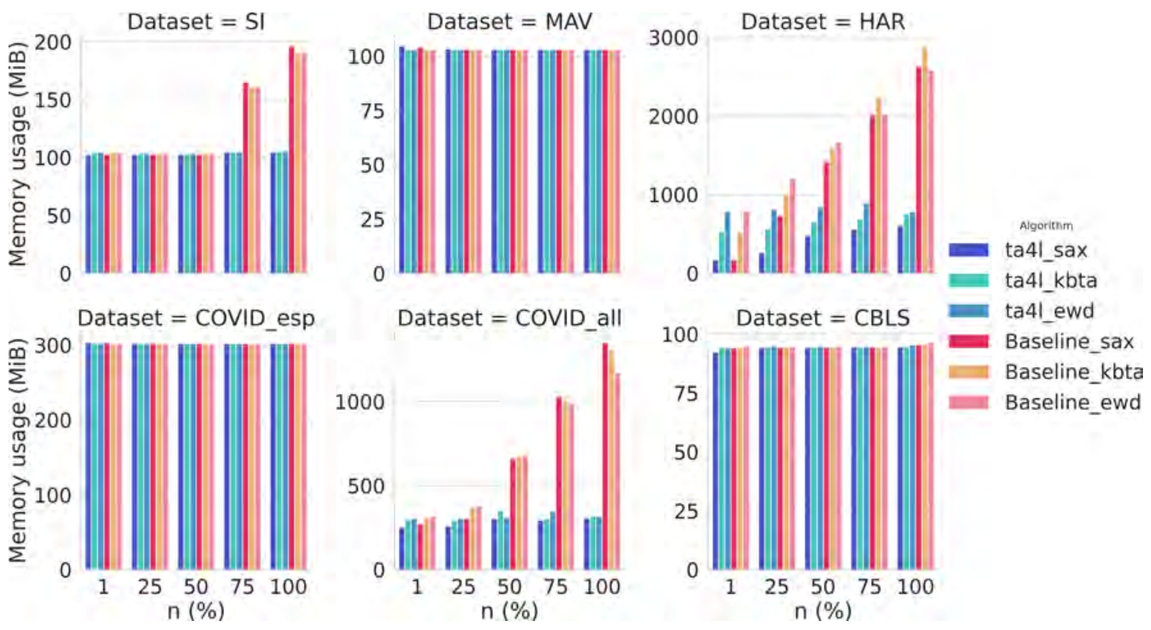


Figure 5.15: Real datasets. Sensitivity to discretization methods in terms of memory while varying n

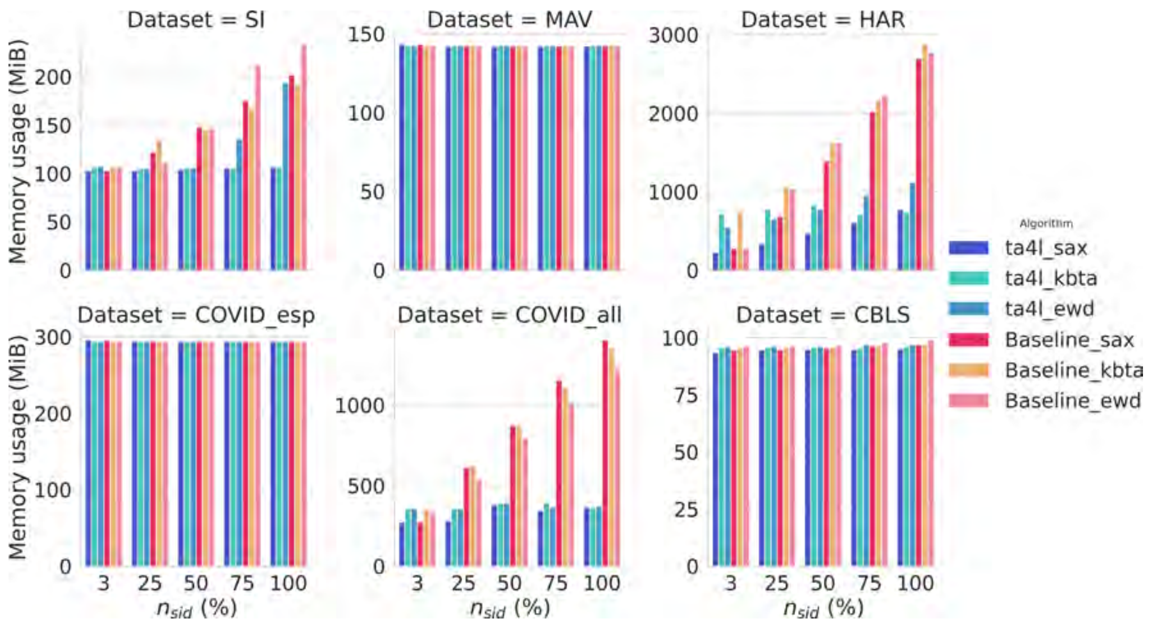
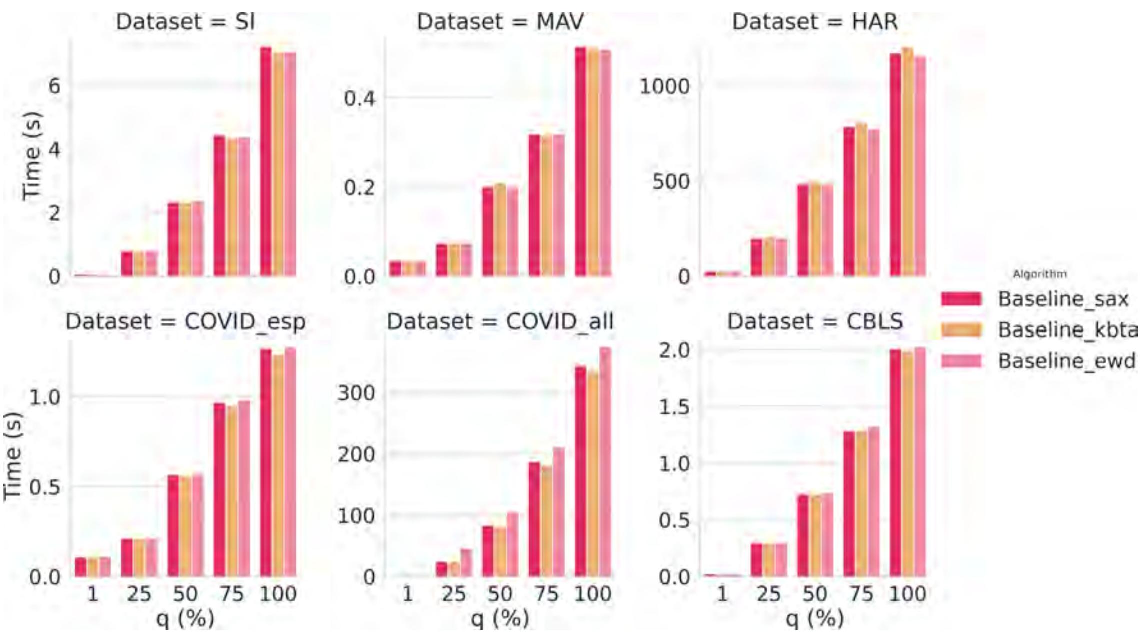
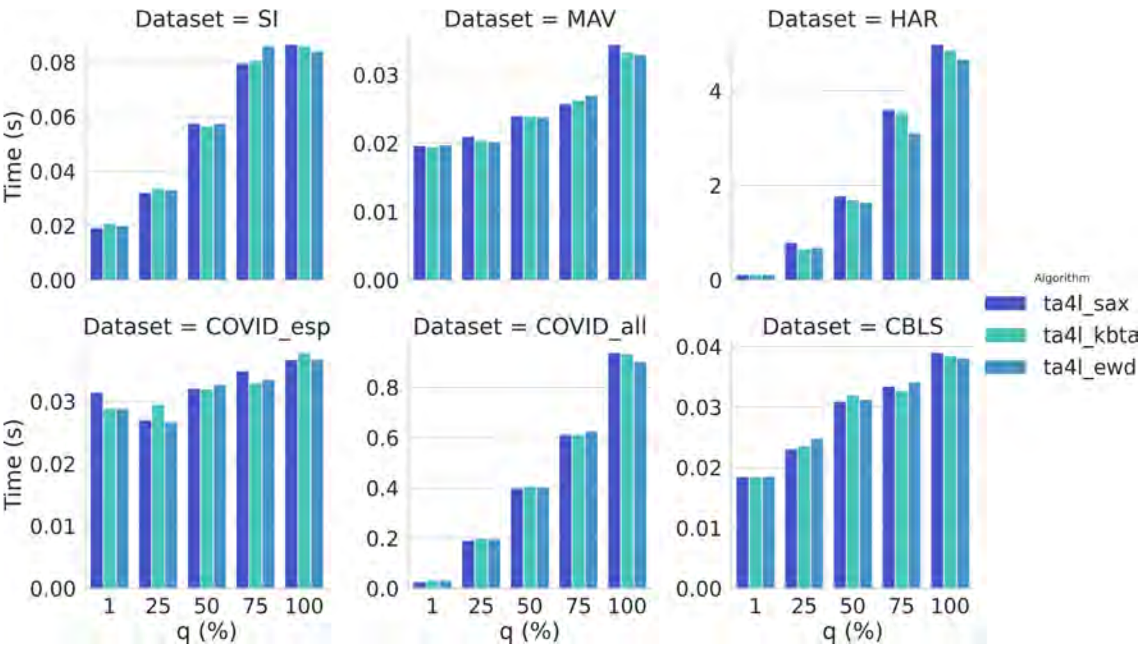


Figure 5.16: Real datasets. Sensitivity to discretization methods in terms of memory while varying n_{sid}

1), while the state of the art is the longest one (23.75 seconds for n 501 and 321.33 for PM 1). In the case of PM values, on the one hand, it can be appreciated that



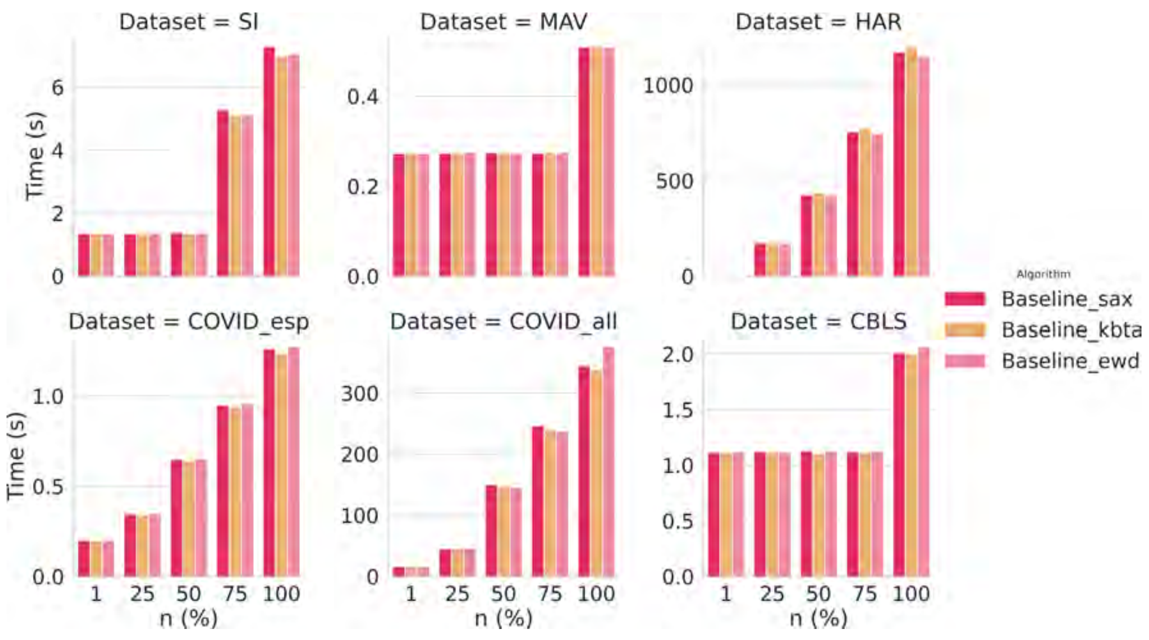
(a) Baseline: Time vs q



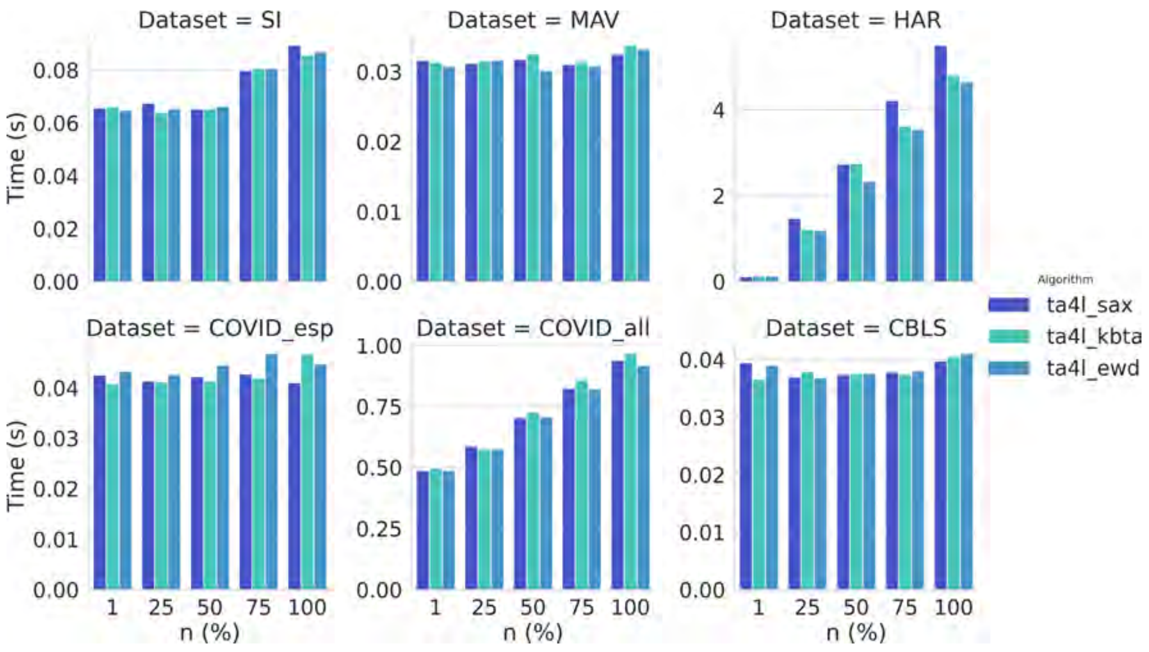
(b) TA4L: Time vs q

Figure 5.17: Real datasets. Sensitivity to discretization methods in terms of time while varying q

Baseline discovers fewer time intervals than the TA4L approach (see Figure 5.22), due to the effect of missing values. And on the other hand, there is a decrease in time,



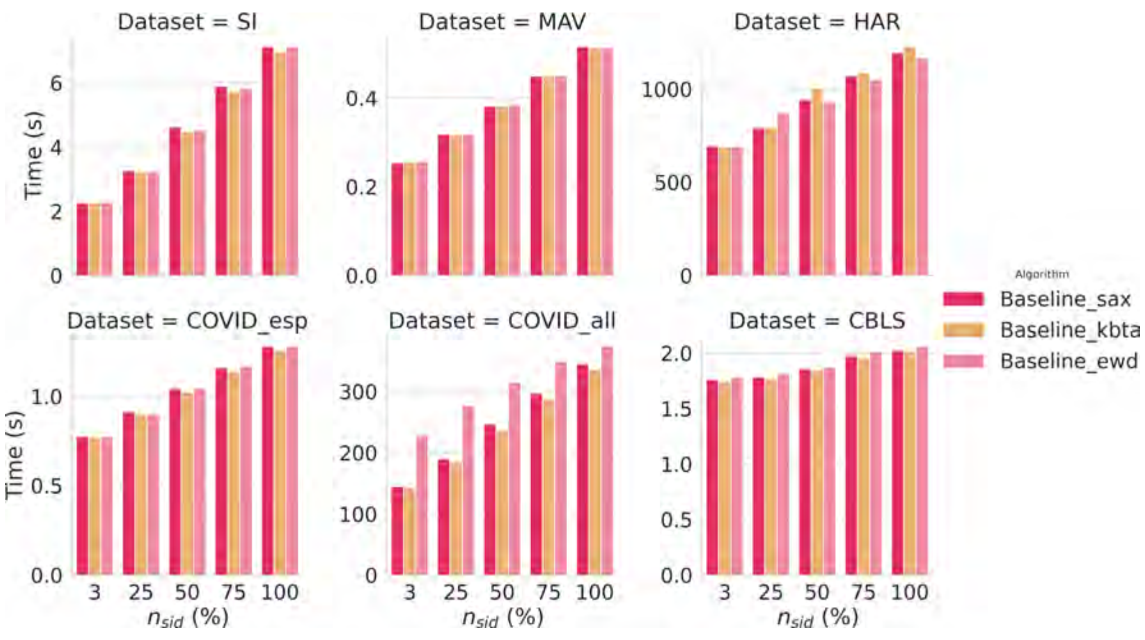
(a) Baseline: Time vs n



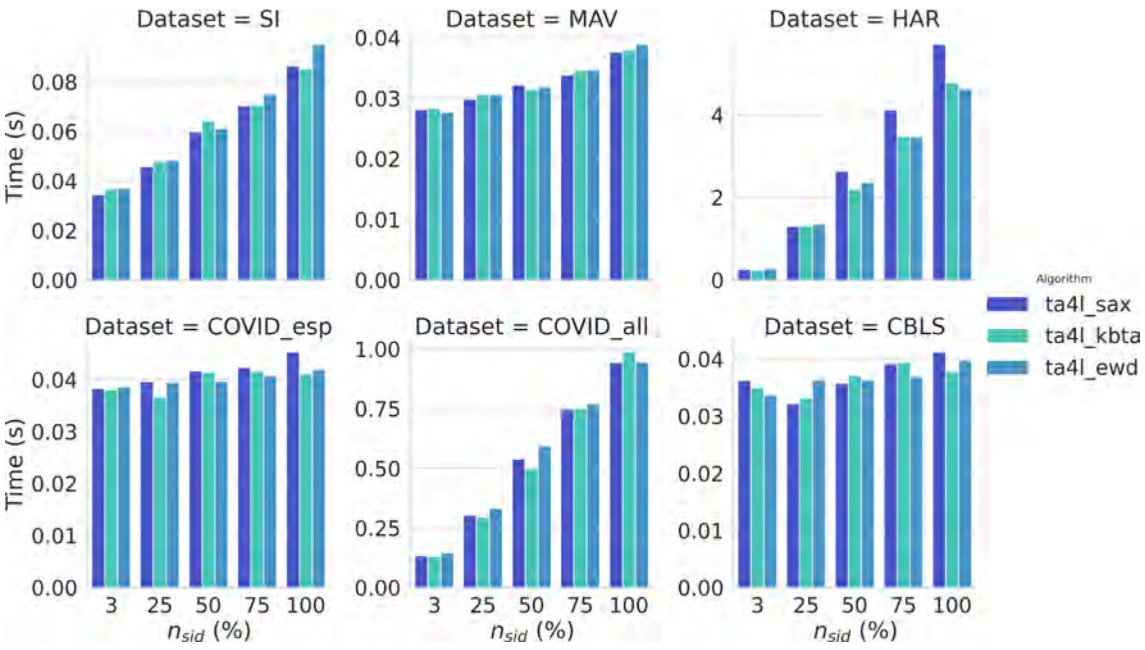
(b) TA4L: Time vs n

Figure 5.18: Real datasets. Sensitivity to discretization methods in terms of time while varying n

memory and number of time intervals with an increase of PM values, confirming the hypothesis. And as for the number of Time intervals, we can check that when there



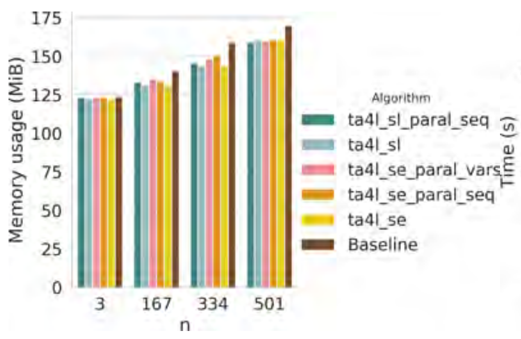
(a) Baseline: Time vs n_{sid}



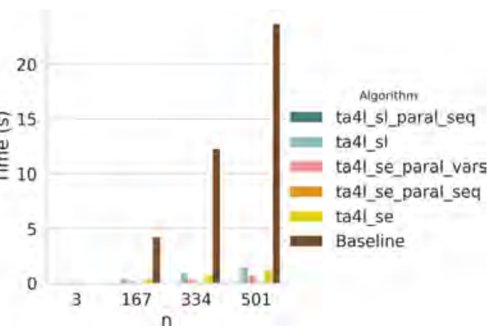
(b) TA4L: Time vs n_{sid}

Figure 5.19: Real datasets. Sensitivity to discretization methods in terms of time while varying n_{sid}

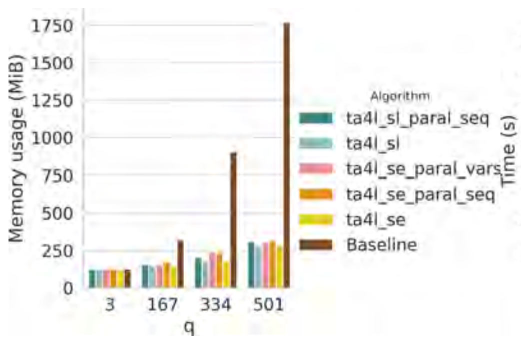
are no missing values, the number of time intervals is the same for all versions, which indicates that the segmentation is done correctly. And regarding memory



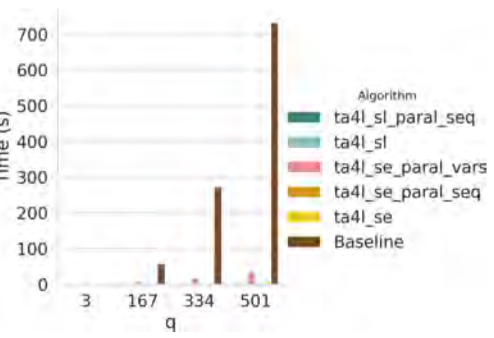
(a) Memory vs n



(b) Time vs n

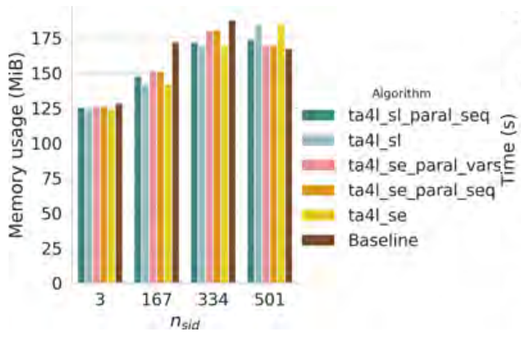


(c) Memory vs q

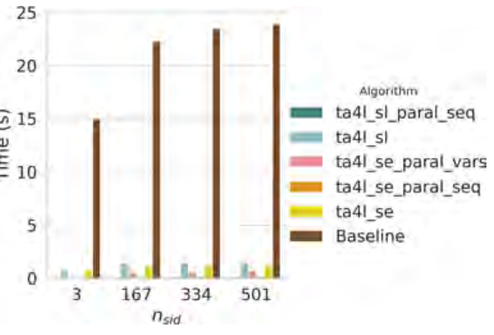


(d) Time vs q

Figure 5.20: How varying n and q affects the performance of the algorithms.



(a) Memory vs n_{sid}



(b) Time vs n_{sid}

Figure 5.21: How varying n_{sid} affects the performance of the algorithms.

consumption, the state of the art approach is that consumes more memory. In the TA4L versions, all algorithms consume approximately the same memory, except the parallel versions that consume more, especially the parallel version of TA4L by variables.

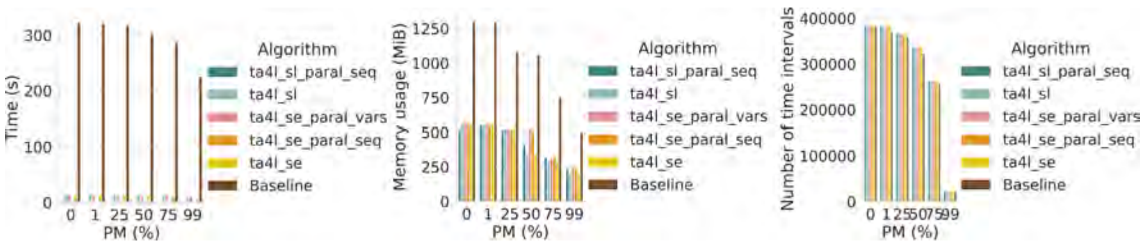


Figure 5.22: How varying PM affects the performance of the algorithms, where |I| is the total number of time intervals.

5.5 Discussion

One of the first insights from the experimental analysis regarding the dataset characteristics is that it is observed that there is a decrease in time, memory and number of time intervals constructed with an increase in PM values, since as larger is PM values, fewer data per interval, and therefore, fewer time intervals constructed, memory usage and runtime. This also applies to the state-of-the-art algorithm, in which, first, each value is discretized and then concatenated. In addition, due to the fact that the TA4L algorithm constructs intervals based on the frame duration (instead of a fixed number of samples), the number of time intervals provided by the TA4L versions is greater than the proposed by the state of the art approach.

Regarding the algorithm parameters, as smaller is the δ, more intervals are generated, and therefore the algorithm performs more mean computations, symbol assignations and insertions, and therefore more time and memory consumption. And as larger is the $|\Sigma|$, more iterations performs the algorithm in the symbol assignation stage, and therefore more time and memory consumption. This behaviour is reflected in all versions of datasets and algorithms. Concerning max_gap, as smaller is the max_gap parameter, the algorithm performs fewer concatenations between the adjacent intervals, and therefore more intervals are generated, and consequently, more time and memory consumption is required. As expected, this effect is less noticeable if the dataset has a very small percentage of missing values (there are no gaps) or a very large number of missing values (low number of time intervals).

In terms of discretization, in the case of synthetic datasets, for large q, n, and n_{sid} values EWD is the more memory efficient method, while for small values of q, n and n_{sid}, SAX is that consumes less time nor memory. In the case of real datasets, it is possible to verify that, in general, there is no significant difference between the methods in terms of time and memory consumption. However, concerning memory consumption, in the case of large datasets (such as HAR or COVID_all datasets), for large q, n, and n_{sid} values, EWD is the more memory efficient method, while for small values of q, n and n_{sid}, SAX is the least memory consuming method.

The cost is greater for numeric variables and a sorted-list like structure. Sorting all at once has a lot more design freedom than maintaining the ordering incrementally since an incremental update has to maintain a complete order at all times, which represents more time and memory consumption. And regarding the multi-threading, if there are a sudden processor and RAM, the best option is to parallelize the TA4L algorithm per sequence instead of per variables.

5.6 Summary

Finally, regarding the pre-processing for TIRPs mining algorithms, this thesis presented the TA4L algorithm that converts multivariate time series into sequences known as LSTISs (symbolic time-intervals sorted by time and symbols). In the literature, it is not easy to find a specific pre-processing algorithm. Usually, pre-processing is explained in a subsection of a TIRPs mining algorithm description. On the one hand, the TA4L algorithm aims to make the process explicitly and provide a solution based on a temporal abstraction and a sequence generation method to accelerate the pre-processing time, merging all the pre-processing tasks to be executed together in a single algorithm. On the other hand, in TA4L, instead of reducing the dimensionality (from time points to time-intervals) based on the number of samples (as is done in the state-of-the-art algorithms), it is reduced based on the time duration adjusted to the real known values. This guarantees a reliable posterior TIRPs mining application with the obtained LSTISs.

Moreover, the size of the time intervals is managed by a given max_gap constraint that enables controlling the maximum separation of two points to be considered part of the same event (i.e. symbol), a property of particular interest in different application domains. Finally, TA4L considers a sorting list data structure and several parallelism strategies (per variable and sequence multi-threads) that make the whole pre-processing efficient. The experimentation carried out in different synthetic and real datasets show that the TA4L cost varies in function of the data structure used and the type of the variables in the dataset (numeric or discrete). In general, TA4L provides a noticeable reduction in time and memory than the state of the art algorithm. Multithread version by variables is useless in the case the dataset has few variables and many sequences. On the contrary, the multi-thread version by sequence consumes less time for all the experiments.

Chapter 6

Conclusions

This final chapter draws some conclusions and provides possible future research directions derived from this thesis. This thesis aimed to provide new efficient SPM and TIRPs mining algorithms and an efficient pre-processing algorithm that transforms multivariate time series (or time series) into sequences ready to feed TIRPs mining algorithms.

6.1 Conclusions and summary of results

The methods presented in this thesis have been evaluated according to the methodology proposed in the "Experimental evaluation methodology" section of chapters 3, 4 and 5. With each proposed algorithm, different experiments have been carried out. The first experiment was to compare the new algorithm with the best state-of-the-art algorithm. All the algorithms developed in this thesis have passed the test, proving to be better than the baseline approach.

This thesis focused the rest of the experiments on testing the efficiency of specific parts of the algorithm or analysing the algorithm's behaviour based on its parameters. These experiments allowed the study and understanding of the algorithms' behaviour in various situations and also to know its strengths and weaknesses, which are necessary when selecting an algorithm to solve some problem.

In the case of VEPRECO, the new vertical representation of patterns structure and the corresponding changes in the extensions have significantly reduced both runtime and memory usage in respect to the state-of-the-art approach (CM-SPAM). VEPRECO's pruning strategy and the common candidate selection policies are handy for large NS and NI, with similar SL and TL.

In the case of vertTIRP, the use of a vertical representation of TIRPs with transitivity among temporal relations and pairing strategies played an important role in improving the algorithm's efficiency. The minimum gap and the minimum duration user constraints are helpful when there is an interest in discarding events that last less than X time units or in looking for patterns with a minimum interval of Y time units. While the user constraints limit the search space, the epsilon value is responsible for two events being paired via one temporal relation or another. The value of epsilon allows for variability between events. Based on the results, it is possible to observe that there is a value of epsilon that maximises or minimises the number of the TIRPs discovered and hence the computation time and memory usage for each dataset.

Finally, in the case of TA4L, there is a decrease in time, memory, and time intervals constructed with an increase in the percentage of missing values. As more minor is the interval's size or as smaller is the maximum gap allowed between two consecutive intervals, the algorithm generates more intervals and consumes more time and memory. The more significant the vocabulary size, the more time and memory consumption will be required. In terms of discretisation, for large datasets, EWD is the more memory efficient method, while for small o medium datasets, SAX consumes less time and memory. The cost is more significant for datasets with many numeric variables and a sorted-list-like structure than the sort-in-the-end approach. Finally, the best option is to parallelise the TA4L algorithm per sequence instead of per variables.

6.2 Future work

An efficient algorithm is a good starting point regardless of the direction of future work. The most direct way of future work of this thesis is in the field of application, i.e. to test the algorithms developed in this thesis with data with which it had not yet been tested before. For example, the vertTIRP algorithm has been already applied to the A Dataset for Emotion Analysis using EEG, Physiological and Video Signals (DEAP) dataset in [104], where a comparative analysis was performed regarding the use of unsupervised techniques for mining patterns from physiological signals against deep learning approaches for feature learning.

In the case of VEPRECO, there are a couple of clear options for future work. One option would be to continue the line of efficiency research of the discovered patterns, either by creating a version of VEPRECO that uses parallelism or by creating an incremental version of VEPRECO that will process the patterns through windows, opening the doors to big data analysis. The other option would be to create new SPM subfields to improve the quality of the discovered patterns, either by researching different new metrics to filter out redundant patterns or by defining a new type of pattern to look for that might be useful.

In the case of TIRPs mining, being a more recent field, the ideas of future work for TIRPs mining have more degree of freedom, as most can be taken directly from the SPM field. For instance, to adapt vertTIRP to mine the Top-K high-utility, closed or negative sequential patterns. Another inspiring idea, this time regarding the application of the vertTIRP, would be to use it to predict the side effects of the COVID-19 vaccines in a given patient. Concerning the efficiency of the algorithm, there are a couple of exciting ideas for future work. One of them is to parallelize or create an incremental version of the vertTIRP in a graphics processing unit (GPU) by using the Nvidia Compute Unified Device Architecture (Nvidia) (CUDA) platform. The other one is related to investigating ways to generate candidates with machine learning models.

Finally, in the TA4L algorithm, an imminent future work would be to adapt it to the data, which by the same variable and the same timestamp record many different states. In addition, another imminent future work includes classifying the TIRPs discovered from the intervals obtained with TA4L and those obtained from the Baseline and comparing the results. This experiment would allow us to power the results regarding the reliability of the intervals obtained from TA4L. Other future work options for TA4L are in the direction of making an incremental version to be able to process big data or improve the efficiency of concatenation and the quality of adjacent intervals using clustering methods.

Bibliography

[1] Wei Shen, Jianyong Wang, and Jiawei Han. Sequential Pattern Mining, pages 261–282. Springer International Publishing, Cham, 2014.

[2] Dhaval Patel, Wynne Hsu, and Mong Li Lee. 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, 2008. Association for Computing Machinery.

[3] Iyad Batal, Hamed Valizadegan, Gregory F. Cooper, and Milos Hauskrecht. A temporal pattern mining approach for classifying electronic health record data. ACM Transactions on Intelligent Systems and Technology (TIST), 4(4):1–22, 2013.

[4] Robert Moskovitch and Yuval Shahar. Classification of multivariate time series via temporal abstraction and time intervals mining. Knowledge and Information Systems, 45(1):35–74, 2015.

[5] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining Association Rules between Sets of Items in Large Databases. In Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, SIGMOD '93, pages 207–216, New York, NY, USA, 1993. Association for Computing Machinery.

[6] Unil Yun, Heungmo Ryang, and Keun Ho Ryu. High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates. Expert Systems with Applications, 41(8):3861–3878, 2014.

[7] Bac Le, Ut Huynh, and Duy-Tai Dinh. A pure array structure and parallel strategy for high-utility sequential pattern mining. Expert Systems with Applications, 104:107–120, 2018.

[8] Yoonji Baek, Unil Yun, Heonho Kim, Hyoju Nam, Gangin Lee, Eunchul Yoon, Bay Vo, and Jerry Chun-Wei Lin. Erasable pattern mining based on tree

structures with damped window over data streams. Engineering Applications of Artificial Intelligence, 94:103735, 2020.

[9] Jian Pei, Jiawei Han, B. Mortazavi-Asl, Jianyong Wang, H. Pinto, Qiming Chen, U. Dayal, and Mei-Chun Hsu. Mining sequential patterns by pattern-growth: the PrefixSpan approach. IEEE Transactions on Knowledge and Data Engineering, 16(11):1424–1440, 2004.

[10] Amina Kemmar, Yahia Lebbah, Samir Loudni, Patrice Boizumault, and Thierry Charnois. Prefix-projection global constraint and top-k approach for sequential pattern mining. Constraints, 22(2):265–306, 2017.

[11] Zhenglu Yang and Masaru Kitsuregawa. LAPIN-SPAM: An Improved Algorithm for Mining Sequential Pattern. In Proceedings of the 21st International Conference on Data Engineering Workshops, ICDEW '05, page 1222, USA, 2005. IEEE Computer Society.

[12] M. K. Sohrabi and V. Ghods. CUSE: A novel cube-based approach for sequential pattern mining. In 2016 4th International Symposium on Computational and Business Intelligence (ISCBI), pages 186–190, 2016.

[13] Morteza Zihayat, Yan Chen, and Aijun An. Memory-adaptive high utility sequential pattern mining over data streams. Machine Learning, 106(6):799–836, 2017.

[14] W. Gan, J. C. Lin, J. Zhang, H. Chao, H. Fujita, and P. S. Yu. ProUM: High Utility Sequential Pattern Mining. In 2019 IEEE International Conference on Systems, Man and Cybernetics (SMC), pages 767–773, 2019.

[15] Philippe Fournier Viger, Antonio Gomariz, Manuel Campos, and Rincy Thomas. Fast Vertical Mining of Sequential Patterns Using Co-occurrence Information. In Pacific-Asia Conference on Knowledge Discovery and Data Mining, pages 40–52, May 2014.

[16] Jay Ayres, Jason Flannick, Johannes Gehrke, and Tomi Yiu. Sequential pattern mining using A bitmap representation. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 429–435, 2002.

[17] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In International conference on extending database technology, pages 1–17. Springer, 1996.

[18] Mohammed J. Zaki. SPADE: An Efficient Algorithm for Mining Frequent Sequences. Machine Learning, 42:31–60, 2001.

[19] Jiawei Han, Jian Pei, Behzad Mortazavi-Asl, Qiming Chen, Umeshwar Dayal, and Mei-Chun Hsu. 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, 2000. Association for Computing Machinery.

[20] Philippe Fournier-Viger, Jerry Chun-Wei Lin School, Rage Uday Kiran University, Yun Sing Koh, and Rincy Thomas. A Survey of Sequential Pattern Mining. In Data Science and Pattern Recognition, volume 1, pages 54–77, 2017.

[21] Shin Yi Wu and Yen Liang Chen. Mining nonambiguous temporal patterns for interval-based events. IEEE Transactions on Knowledge and Data Engineering, 19(6):742–758, 2007.

[22] Jen-Wei Huang, Bijay Prasad Jaysawal, Kuan-Ying Chen, and Yong-Bin Wu. Mining frequent and top-k high utility time interval-based events with duration patterns. Knowledge and Information Systems, 61(3):1331–1359, 2019.

[23] Hieu Hanh Le, Muneo Kushima, Kenji Araki, and Haruo Yokota. Differentially private sequential pattern mining considering time interval for electronic medical record systems. In Proceedings of the 23rd International Database Applications & Engineering Symposium, pages 1–9, 2019.

[24] Lin Hui, Yi Cheng Chen, Julia Tzu Ya Weng, and Suh Yin Lee. Incremental mining of temporal patterns in interval-based database. Knowl Inf Syst, 46(2):423–448, 2016.

[25] Robert Moskovitch and Yuval Shahar. Fast time intervals mining using the transitivity of temporal relations. Knowledge and Information Systems, 42(1):21–48, 2013.

[26] Po-shan Kam and Ada Wai-chee Fu. Discovering Temporal Patterns for Interval-based Events. In Yahiko Kambayashi, Mukesh Mohania, and A. Min Tjoa, editors, Data Warehousing and Knowledge Discovery, pages 317–326, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[27] James F. Allen. Maintaining Knowledge about Temporal Intervals. Artificial Intelligence and Language Processing, 26(11):832–843, 1983.

[28] Panagiotis Papapetrou, George Kollios, Stan Sclaroff, and Dimitrios Gunopulos. Mining frequent arrangements of temporal intervals. Knowledge and Information Systems, 21(2):133–171, 2009.

[29] Edi Winarko and John F. Roddick. ARMADA - An algorithm for discovering richer relative temporal association rules from interval-based data. Data and Knowledge Engineering, 63(1):76–90, 2007.

[30] Nizar R. Mabroukeh and C. I. Ezeife. A taxonomy of sequential pattern mining algorithms. ACM Computing Surveys, 43(1), 2010.

[31] Rakesh Agrawal, Ramakrishnan Srikant, and Others. Fast algorithms for mining association rules. In Proc. 20th int. conf. very large data bases, VLDB, volume 1215, pages 487–499, 1994.

[32] R. Agrawal and R. Srikant. Mining sequential patterns. In Proceedings of the Eleventh International Conference on Data Engineering, pages 3–14, 1995.

[33] Mohammed J. Zaki. Scalable algorithms for association mining. Knowledge and Data Engineering, IEEE Transactions on, 12:372–390, 2000.

[34] Wensheng Gan, Jerry Chun-Wei Lin, Philippe Fournier-Viger, Han-Chieh Chao, and Philip S. Yu. A Survey of Parallel Sequential Pattern Mining. ACM Trans. Knowl. Discov. Data, 13(3), 2019.

[35] Jian Pei, Jiawei Han, Behzad Mortazavi-asl, and Hua Zhu. Mining Access Patterns Efficiently from Web Logs. In Takao Terano, Huan Liu, and Arbee L. P. Chen, editors, Knowledge Discovery and Data Mining. Current Issues and New Applications, pages 396–407, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[36] Maged El-Sayed, Carolina Ruiz, and Elke A. Rundensteiner. FS-Miner: Efficient and incremental mining of frequent sequence patterns in Web logs. Proceedings of the Interntational Workshop on Web Information and Data Management, pages 128–135, 2004.

[37] Jen-Wei Huang, Chi-Yao Tseng, Jian-Chih Ou, and Ming-Syan Chen. On progressive sequential pattern mining. In Proceedings of the 15th ACM international conference on Information and knowledge management, pages 850–851, 2006.

[38] Jian Pei, Jiawei Han, B. Mortazavi-Asl, H. Pinto, Qiming Chen, U. Dayal, and Mei-Chun Hsu. PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth. In Proceedings 17th International Conference on Data Engineering, pages 215–224, 2001.

[39] Christie I. Ezeife and Yi Lu. Mining web log sequential patterns with position coded pre-order linked wap-tree. Data Mining and Knowledge Discovery, 10(1):5–38, 2005.

[40] Huy M. Huynh, Loan T. Nguyen, Bay Vo, Anh Nguyen, and Vincent S. Tseng. Efficient methods for mining weighted clickstream patterns. Expert Systems with Applications, 142:112993, 2020.

[41] Xin Lyu and Hongxu Ma. An Efficient Incremental Mining Algorithm for Discovering Sequential Pattern in Wireless Sensor Network Environments. Sensors, 19:29, 2018.

[42] Wensheng Gan, Jerry Chun-wei Lin, Jiexiong Zhang, Han-chieh Chao, Hamido Fujita, and Philip S. Yu. ProUM : Projection-based utility mining on sequence data. Information Sciences, 513:222–240, 2020.

[43] Rina Singh, Jeffrey A. Graves, Douglas A. Talbert, and William Eberle. Prefix and Suffix Sequential Pattern Mining. In Petra Perner, editor, Advances in Data Mining. Applications and Theoretical Aspects, pages 309–324, Cham, 2018. Springer International Publishing.

[44] Sujeevan Aseervatham, Aomar Osmani, and Emmanuel Viennet. bitSPADE: A lattice-based sequential pattern mining algorithm using bitmap representation. In Sixth International Conference on Data Mining (ICDM'06), pages 792–797. IEEE, 2006.

[45] Xiangjun Dong, Ping Qiu, Jinhu Lu, Longbing Cao, and Tiantian Xu. Mining Top- k Useful Negative Sequential Patterns via Learning. IEEE transactions on neural networks and learning systems, 30(9):2764–2778, 2019.

[46] Fan Min, Zhi Heng Zhang, Wen Jie Zhai, and Rong Ping Shen. Frequent pattern discovery with tri-partition alphabets. Information Sciences, 507:715–732, 2020.

[47] Xiangjun Dong, Yongshun Gong, and Longbing Cao. E-RNSP: An Efficient Method for Mining Repetition Negative Sequential Patterns. IEEE Transactions on Cybernetics, 50(5):2084–2096, 2020.

[48] Wei Song, Beisi Jiang, and Yangyang Qiao. Mining multi-relational high utility itemsets from star schemas. Intelligent Data Analysis, 22(1):143–165, 2018.

[49] Omer Adam, Zailani Abdullah, Amir Ngah, Kasypi Mokhtar, Wan Muhamad Amir Wan Ahmad, Tutut Herawan, Noraziah Ahmad, Mustafa Mat Deris, Abdul Razak Hamdan, and Jemal H. Abawajy. IncSPADE: An Incremental Sequential Pattern Mining Algorithm Based on SPADE Property, chapter 8, pages 81–92. Springer International Publishing, Cham, 2016.

[50] Sumalatha Saleti and R. B. V. Subramanyam. A novel mapreduce algorithm for distributed mining of sequential patterns using co-occurrence information. Applied Intelligence, 49(1):150–171, 2019.

[51] Saleti Sumalatha and R. B. V. Subramanyam. Distributed mining of high utility time interval sequential patterns using mapreduce approach. Expert Systems With Applications, 141:112967, 2020.

[52] Haoxing Wen, Mingdong Kou, Hengyi He, Xiaoguang Li, Huaixiao Tou, and Yulu Yang. A Spark-Based Incremental Algorithm for Frequent Itemset Mining. In Proceedings of the 2018 2nd International Conference on Big Data and Internet of Things, BDIOT 2018, pages 53–58, New York, NY, USA, 2018. Association for Computing Machinery.

[53] S. AlZu'bi, B. Hawashin, M. EIBes, and M. Al-Ayyoub. A Novel Recommender System Based on Apriori Algorithm for Requirements Engineering. In 2018 Fifth International Conference on Social Networks Analysis, Management and Security (SNAMS), pages 323–327, 2018.

[54] Sudhakar Singh, Rakhi Garg, and P. K. Mishra. Performance optimization of MapReduce-based Apriori algorithm on Hadoop cluster. Computers & Electrical Engineering, 67:348–364, 2018.

[55] Xu Yusheng, Ma Zhixin, Li Lian, and Tharam S. Dillon. Effective pruning strategies for sequential pattern mining. In First International Workshop on Knowledge Discovery and Data Mining (WKDD 2008), pages 21–24. IEEE, 2008.

[56] Huy Minh Huynh, Nam Ngoc Pham, Zuzana Komínková Oplatková, Loan Thi Thuy Nguyen, and Bay Vo. Sequential Pattern Mining Using IDLists. In Ngoc Thanh Nguyen, Bao Hung Hoang, Cong Phap Huynh, Dosam Hwang, Bogdan Trawiński, and Gottfried Vossen, editors, Computational Collective Intelligence, pages 341–353, Cham, 2020. Springer International Publishing.

[57] M. Garofalakis, R. Rastogi, and K. Shim. Mining sequential patterns with regular expression constraints. IEEE Transactions on Knowledge and Data Engineering, 14(3):530–552, 2002.

[58] R. Trasarti, F. Bonchi, and B. Goethals. Sequence Mining Automata: A New Technique for Mining Frequent Sequences under Regular Expressions. In 2008 Eighth IEEE International Conference on Data Mining, pages 1061–1066, 2008.

[59] Siegfried Nijssen, Tias Guns, and Luc De Raedt. Correlated itemset mining in ROC space: a constraint programming approach. In Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 647–656, 2009.

[60] Q. Wang, V. S. Sheng, and C. Hu. Keyphrase Extraction Using Sequential Pattern Mining and Entropy. In 2017 IEEE International Conference on Big Knowledge (ICBK), pages 88–95, 2017.

[61] Mohammed J. Zaki. Sequence mining in categorical domains: incorporating constraints. In Proceedings of the ninth international conference on Information and knowledge management, pages 422–429, 2000.

[62] "P. Gay, B. López, and J. Meléndez". Learning Complex Events from Sequences with Informed Gaps. In 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA), pages 1089–1094, 2015.

[63] Benjamin Negrevergne and Tias Guns. Constraint-based sequence mining using constraint programming. In International Conference on AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems, pages 288–305. Springer, 2015.

[64] René Arnulfo García-Hernández, José Francisco Martínez-Trinidad, and Jesús Ariel Carrasco-Ochoa. A new algorithm for fast discovery of maximal sequential patterns in a document collection. In International Conference on Intelligent Text Processing and Computational Linguistics, pages 514–523. Springer, 2006.

[65] Philippe Fournier-Viger, Cheng-Wei Wu, Antonio Gomariz, and Vincent S. Tseng. VMSP: Efficient vertical mining of maximal sequential patterns. In Canadian conference on artificial intelligence, pages 83–94. Springer, 2014.

[66] Fabio Fumarola, Pasqua Fabiana Lanotte, Michelangelo Ceci, and Donato Malerba. CloFAST: closed sequential pattern mining using sparse and vertical id-lists. Knowledge and Information Systems, 48(2):429–463, 2016.

[67] Thi-Thiet Pham, Tung Do, Anh Nguyen, Bay Vo, and Tzung-Pei Hong. An efficient method for mining top-K closed sequential patterns. IEEE Access, 8:118156–118163, 2020.

[68] Robert Moskovitch and Yuval Shahar. Medical temporal-knowledge discovery via temporal abstraction. In AMIA 2009 Symposium Proceedings, pages 452–456, 2009.

[69] Robert Moskovitch, Colin Walsh, Fei Wang, George Hripcsak, and Nicholas Tatonetti. Outcomes Prediction via Time Intervals Related Patterns. In 2015 IEEE International Conference on Data Mining, pages 919–924, November 2015.

[70] Ming-Yen Lin and Suh-Yin Lee. Fast Discovery of Sequential Patterns by Memory Indexing. In Yahiko Kambayashi, Werner Winiwarter, and Masatoshi Arikawa, editors, International Conference on Data Warehousing and Knowledge Discovery, pages 150–160, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[71] Omer David Harel and Robert Moskovitch. Complete closed time intervals-related patterns mining. Proceedings of the AAAI Conference on Artificial Intelligence, 35(5):4098–4105, May 2021.

[72] Yi Cheng Chen, Wen Chih Peng, and Suh Yin Lee. Mining temporal patterns in interval-based data. In 2016 IEEE 32nd International Conference on Data Engineering, ICDE 2016, volume 27, pages 1506–1507. IEEE, 2016.

[73] Robert Moskovitch and Yuval Shahar. Classification-driven temporal discretization of multivariate time series. Data Mining and Knowledge Discovery, 29:871–913, 2015.

[74] Cheng Wei Yang, Bijay Prasad Jaysawal, and Jen Wei Huang. Subsequence search considering duration and relations of events in time interval-based events sequences. In Proceedings - 2017 International Conference on Data Science and Advanced Analytics, DSAA 2017, volume 2018-Janua, pages 293–302, 2018.

[75] Yuval Shahar. A framework for knowledge-based temporal abstraction. Artificial Intelligence, 90(1):79–133, 1997.

[76] Revital Azulay, Robert Moskovitch, Dima Stopel, Marion Verduijn, Evert de Jonge, and Yuval Shahar. Temporal discretization of medical time series-a comparative study. In Proceedings of IDAMAP2007: Intelligent Data Analysis in Biomedicine and Pharmacology, volume 43, pages 48–58, 2007.

[77] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. Segmenting time series: A survey and novel approach. In Data mining in time series databases, pages 1–21. World Scientific, 2004.

[78] Fabian Mörchen and Alfred Ultsch. Optimizing time series discretization for knowledge discovery. In Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 660–665, 2005.

[79] Sofya S. Titarenko, Valeriy N. Titarenko, Georgios Aivaliotis, and Jan Palczewski. Fast implementation of pattern mining algorithms with timestamp uncertainties and temporal constraints. Journal of Big Data, 6(1):37, 2019.

[80] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and Unsupervised Discretization of Continuous Features. In Armand Prieditis and Stuart Russell, editors, Machine Learning Proceedings 1995, pages 194–202. Morgan Kaufmann, San Francisco (CA), 1995.

[81] Cláudia Antunes and Arlindo L. Oliveira. Generalization of pattern-growth methods for sequential pattern mining with gap constraints. In International Workshop on Machine Learning and Data Mining in Pattern Recognition, pages 239–251. Springer, 2003.

[82] Chun Li and Jianyong Wang. Efficiently mining closed subsequences with gap constraints. In proceedings of the 2008 SIAM International Conference on Data Mining, pages 313–322. SIAM, 2008.

[83] Natalia Mordvanyuk, Beatriz López, and Albert Bifet. vertTIRP: Robust and efficient vertical frequent time interval-related pattern mining. Expert Systems with Applications, page 114276, 2020.

[84] Mohammed J. Zaki. IBMGenerator. https://github.com/zakimjz/IBMGenerator, 2016. Accessed: 10-04-2021.

[85] P. Fournier-Viger, C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam. The SPMF Open-Source Data Mining Library Version 2. In Proc. 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD 2016) Part III, pages 36–40. Springer LNCS 9853, 2016.

[86] Siegfried Nijssen and Albrecht Zimmermann. Constraint-based pattern mining. In Frequent pattern mining, pages 147–163. Springer, 2014.

[87] Natalia Mordvanyuk, Ferran Torrent-Fontbona, and Beatriz López. Prediction of Glucose Level Conditions from Sequential Data. In I. O. S. Press, editor, Volume 300: Recent Advances in Artificial Intelligence Research and Development, pages 227–232, 2017.

[88] Chiara Zecchin. Online Glucose Prediction in Type 1 Diabetes by Neural Network Models. PhD dissertation, Università degli Studi di Padova, 2014.

[89] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. ACM sigmod record, 29(2):1–12, 2000.

[90] Vincent S. Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip S. Yu. UP-Growth: An Efficient Algorithm for High Utility Itemset Mining. In Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '10, pages 253–262, New York, NY, USA, 2010. Association for Computing Machinery.

[91] V. S. Tseng, B. Shie, C. Wu, and P. S. Yu. Efficient Algorithms for Mining High Utility Itemsets from Transactional Databases. IEEE Transactions on Knowledge and Data Engineering, 25(8):1772–1786, 2013.

[92] Joseph Bugeja. Smart home datasets and a realtime Internet-connected home, 2019. Accessed: 10-10-2020.

[93] Carol Neidle. American Sign Language Linguistic Research Project, 2017.

[94] Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra, and Jorge Luis Reyes-Ortiz. A public domain dataset for human activity recognition using smartphones. In Esann, pages 437–442, 2013.

[95] Dheeru Dua and Casey Graff. UCI Machine Learning Repository, 2017. Accessed: 10-10-2020.

[96] Rajanand Ilangovan. Suicides in India, 2012. Accessed: 10-10-2020.

[97] Centers for Disease Control and Prevention. Childhood Blood Lead Surveillance, 2017. Accessed: 10-10-2020.

[98] Flaredown. Chronic illness: symptoms, treatments and triggers, 2018.

[99] Jessica Lin, Eamonn Keogh, Stefano Lonardi, and Bill Chiu. A symbolic representation of time series, with implications for streaming algorithms. In Proceedings of the 8th ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery, DMKD '03, pages 2–11, 2003.

[100] Mathieu Fourment and Michael R. Gillings. A comparison of common programming languages used in bioinformatics. BMC Bioinformatics, 9:1–9, 2008.

[101] Paul-Virak Khuong and Pat Morin. Array layouts for comparison-based searching. Journal of Experimental Algorithmics, 22:1–39, 2017.

[102] Daniel García. Kaggle, 2020. Accessed: 10-10-2020.

[103] Juan Carlos Santiago Culebras. Kaggle, 2020. Accessed: 10-10-2020.

[104] N. Mordvanyuk, J. Gauchola, and B. López. Understanding affective behaviour from physiological signals: Feature learning versus pattern mining. In 34th IEEE CBMS International Symposium on Computer-Based Medical Systems, pages 438–443, jun 2021.

Appendix A

Implementation of VEPRECO

The implementation of VEPRECO is available from the Bitbucket repository at "git clone https://InvitedToVEPRECO@bitbucket.org/natalia_mordvanyuk/vepreco.git".

The following username and password can be used to log in:

username: invited3bynatalia

email: invited3bynatalia@gmail.com

password: 2wGW4hxgMVUtGkF..,

Appendix B

vertTIRP's appendix

B.1 Implementation of vertTIRP

The implementation of vertTIRP is available from the Bitbucket repository at "https://bitbucket.org/natalia_mordvanyuk/verttirp/src/master/". The following username and password can be used to log in:

username: invitedbymordvanyuk@gmail.com

password: YgGhjhGllgFFdlkbvFM..,

B.2 Table of pairing strategies for epsilon=0

Table B.1 shows the relations after sorting and grouping based on the pairing strategies proposed in this thesis for epsilon=0.

dataset	sorted relations	common conditions
MAV	bcmsfoe	$\underline{\text{b}}\ \underline{\text{c}\underline{\text{mo}}\text{f}}\ \underline{\text{se}}$
ASL	bemcfos	$\underline{\text{b}}\ \underline{\text{es}}\ \underline{\underline{\text{mo}}\text{cf}}$
HAR	bmcfsoe	$\underline{\text{b}}\ \underline{\underline{\text{mo}}\text{cf}}\ \underline{\text{se}}$
CBLS	besfcom	$\underline{\text{b}}\ \underline{\text{es}}\ \underline{\text{fc}\underline{\text{om}}}$
SI	fsbecmo	$\underline{\text{fc}\underline{\text{mo}}}\ \underline{\text{se}}\ \underline{\text{b}}$
CI	csfebmo	$\underline{\text{cf}\underline{\text{mo}}}\ \underline{\text{se}}\ \underline{\text{b}}$

Table B.1: Results of pairing strategies for each dataset for epsilon=0. Note that for epsilon=0, the left contain relation l does not exist.

B.3 Intersection of temporal relations

See Table B.2 for details on intersections between temporal relations.

R	b	m	o	l	c	f	e	s
b	▨	▨	▨	▨	▨	▨	▨	▨
m	-	▨	▨	▨	▨	▨	▨	▨
o	-	$B.s - A.s > \epsilon \land B.e - A.e > \epsilon$	▨	▨	▨	▨	▨	▨
l	-	-	-	▨	▨	▨	▨	▨
c	-	$B.s - A.s > \epsilon$	$B.s - A.s > \epsilon$	$A.e - B.e > \epsilon$	▨	▨	▨	▨
f	-	$B.s - A.s > \epsilon$	$B.s - A.s > \epsilon$	-	$B.s - A.s > \epsilon$	▨	▨	▨
e	-	-	-	$\|B.s - A.s\| \leq \epsilon$	-	$\|B.e - A.e\| \leq \epsilon$	▨	▨
s	-	$B.e - A.e > \epsilon$	$B.e - A.e > \epsilon$	$\|B.s - A.s\| \leq \epsilon$	-	-	$\|B.s - A.s\| \leq \epsilon$	▨

Table B.2: Intersections between temporal relations

B.4 Allen's transitivity table

See Table B.3 for details on temporal relations from Allen's transitivity property.

B.5 Complete mining example

This section illustrates the operation of vertTIRP using the example given in Table 4.1. First, vertTIRP builds length-one patterns, as shown in Figure B.1.

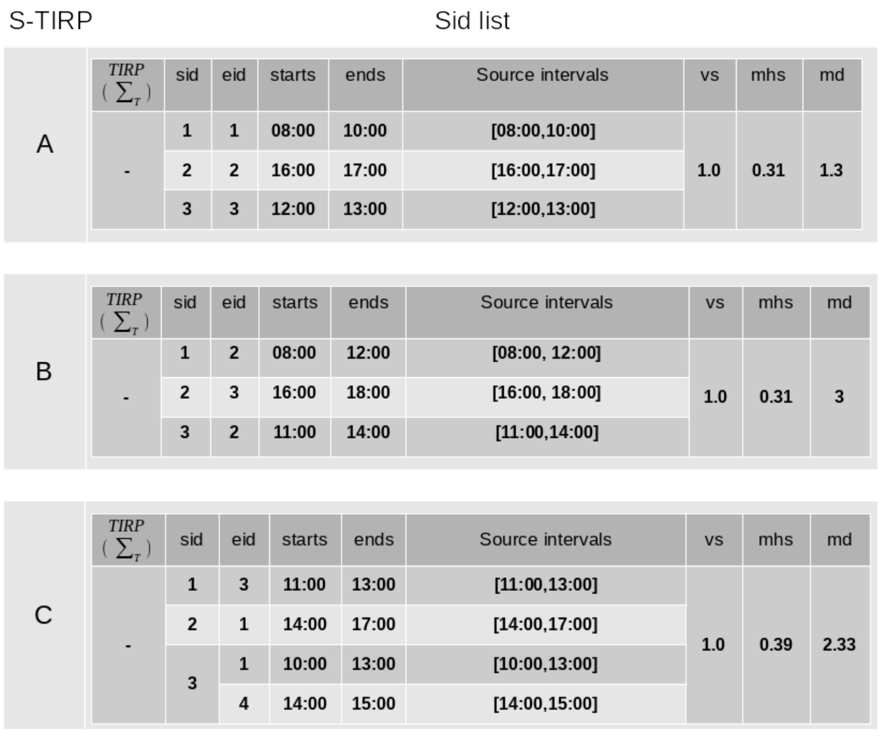
Then, the first extension to be considered is the $A - A$ extension. The extension $A - A$ has zero supporting time interval sequences.

$r_1(I^A, I^B)$ \ $r_2(I^B, I^C)$	b	c	o	m	s	f	e
b "before"	b	b	b	b	b	b	b
c "contains"	b c f m o	c	c f o	c f o	c f o	c	c
o "overlaps"	b	b c f m o	b m o	b	o	b m o	o
m "meets"	b	b	b	b	m	b	m
s "starts"	b	b c f m o	b m o	b	s	b m o	s
f "finished-by"	b	c	o	m	o	f	f
e "equal"	b	c	o	m	s	f	e

Table B.3: Temporal relations from Allen's transitivity property

The next extension to be considered is $A - B$ (see Figure B.2). In this case, there are several time interval sequences supporting it. The algorithm's starting point is the eid 1 of sequence 1 of A, and it extends it with eid 2 of sequence 1 of B. To find the temporal relation between A and B, several relations can be tested using pairing strategies (in this example, b ofmc se). The current data match the definition of starts, where $(B.s - A.s) \geq \epsilon$ and $(B.e - A.e) > \epsilon$: (8 - 8) >= 0 and (12 - 10) > 0. Hence, the pattern $< AB, s >$ is created: the start time of the pattern is the start time of A (shown in red in Figure B.2), since this is the first time interval, while the end time of $< AB, s >$ is the end time of B (shown in black), since this is the latest time of all the time intervals. The *eid* of the new TIRP $< AB, s >$ is the *eid* that appears in B (shown in purple), since the algorithm copies the eid of the pattern that is used to extend the sequence (i.e. the last event that happens), and in this case, it extends A with B. Finally, the vertical and the horizontal supports are set to 0.33 (1/3) and the mean duration to four. Since the support is greater than or equal to the minimum support ($min\_sup = 0.33$), <AB, s> is frequent.

As there are no more eids in time interval sequence 1 of B, it continues with the


S-TIRP
Sid list

S-TIRP	TIRP ($\sum_T$)	sid	eid	starts	ends	Source intervals	vs	mhs	md
A	-	1	1	08:00	10:00	[08:00,10:00]	1.0	0.31	1.3
		2	2	16:00	17:00	[16:00,17:00]			
		3	3	12:00	13:00	[12:00,13:00]			

S-TIRP	TIRP ($\sum_T$)	sid	eid	starts	ends	Source intervals	vs	mhs	md
B	-	1	2	08:00	12:00	[08:00, 12:00]	1.0	0.31	3
		2	3	16:00	18:00	[16:00, 18:00]			
		3	2	11:00	14:00	[11:00,14:00]			

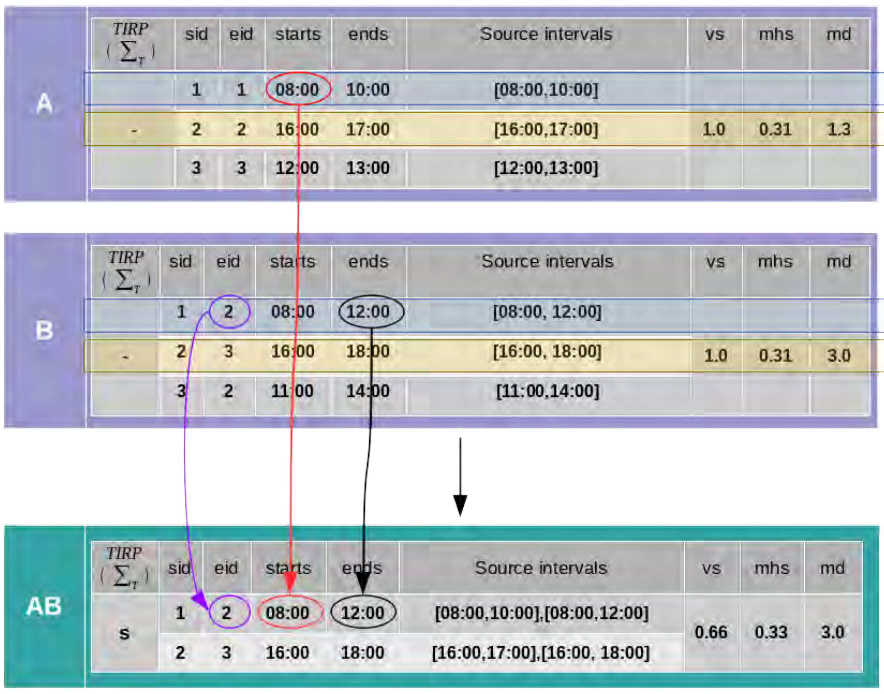
S-TIRP	TIRP ($\sum_T$)	sid	eid	starts	ends	Source intervals	vs	mhs	md
C	-	1	3	11:00	13:00	[11:00,13:00]	1.0	0.39	2.33
		2	1	14:00	17:00	[14:00,17:00]			
		3	1	10:00	13:00	[10:00,13:00]			
			4	14:00	15:00	[14:00,15:00]			

Figure B.1: Length-one S-TIRPs for the example in Table 4.1

first *eid* of the next sequence of A, with id 2. The first eid in sequence 2 of B has a value of three, i.e. greater than two. Moreover, the start relation is met between eid 2 and eid 3. Since <AB,s> already exists, it updates the information related to the pattern by increasing the vertical support to 0.66 (2/3=0.66), the mean horizontal to 0.33 ((1/3+1/3)/2=0.33) and the duration to 3 ((4+2)/2=3).

It continues with the next sequence, i.e. sequence 3 of A. The first eid is eid 3, so it searchs in B's eids for an id value greater than or equal to 3; however, such an eid does not exist, so it breaks and continues with the next sequence. As there are no more sequences, the process stops.

Let us now consider the extension $AB - C$ (represented in Figure B.3). The algorithm starts with *eid* 2 of sequence 1 of AB, and finds *eid* 3 of sequence 1 of C as a possible candidate. The algorithm now has to find the temporal relations for BC and AC. First, between B and C, the pairing strategy gives the overlaps relation. Then, the relation between A and C is computed by means of the transitivity table (see Table 4.7). From the relations r1(A,B)=s and r2(B,C)=o, it finds that the



TIRP (Σ_T)	sid	eid	starts	ends	Source intervals	vs	mhs	md
	1	1	08:00	10:00	[08:00,10:00]			
-	2	2	16:00	17:00	[16:00,17:00]	1.0	0.31	1.3
	3	3	12:00	13:00	[12:00,13:00]			

TIRP (Σ_T)	sid	eid	starts	ends	Source intervals	vs	mhs	md
	1	2	08:00	12:00	[08:00, 12:00]			
-	2	3	16:00	18:00	[16:00, 18:00]	1.0	0.31	3.0
	3	2	11:00	14:00	[11:00,14:00]			

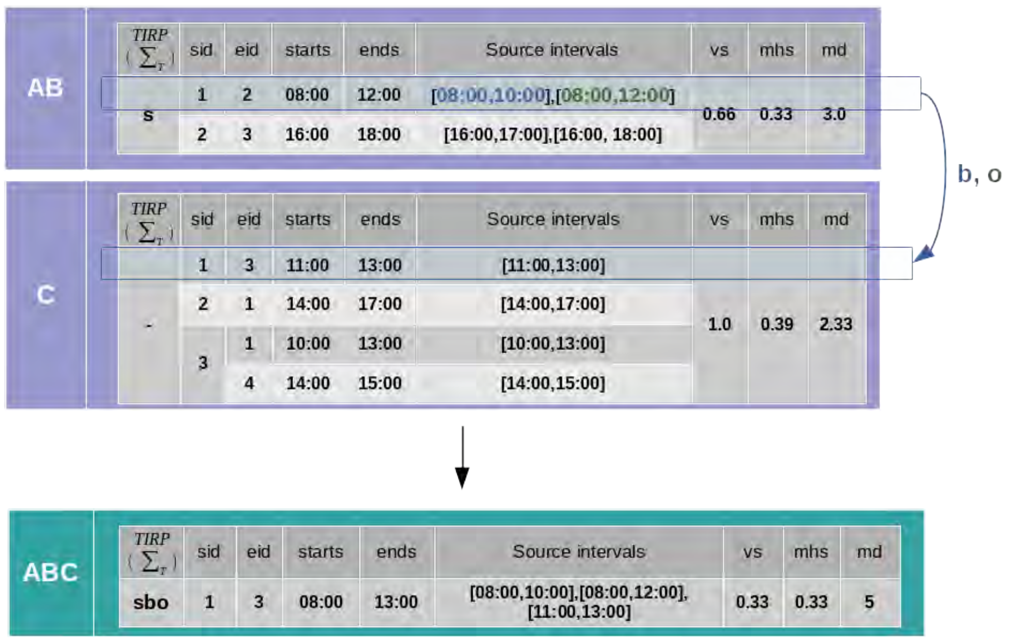
TIRP (Σ_T)	sid	eid	starts	ends	Source intervals	vs	mhs	md
s	1	2	08:00	12:00	[08:00,10:00],[08:00,12:00]	0.66	0.33	3.0
	2	3	16:00	18:00	[16:00,17:00],[16:00, 18:00]			

Figure B.2: Illustration of the extension $A - B$.

possibilities for the relation r3(A,C) are the b,o,m relations. First, it tests the most frequent temporal relation according to the pairing strategy, i.e. the before relation *b*, which turns out to be the correct one. Thus it creates the pattern <ABC, *sbo*>, and sets the vertical and the horizontal supports to 0.33 and the mean duration to five. Since the support is greater than the minimum support (0.33), ABC is found to be frequent.

As there are no more eids in sequence 1 in C, and there are no more eids in sequence 1 in AB, it continues with the first *eid* 3 of AB of the next sequence, i.e. sequence 2. As there are no *eids* in sequence 2 in C greater than 3, and since sequence 2 is the last sequence in AB, no more support is found for ABC.

All of the patterns found by vertTIRP for the simple example given above are shown in Figure 4.5.



AB

TIRP (Σ_r)	sid	eid	starts	ends	Source intervals	vs	mhs	md
s	1	2	08:00	12:00	[08:00,10:00],[08:00,12:00]	0.66	0.33	3.0
	2	3	16:00	18:00	[16:00,17:00],[16:00, 18:00]			

C

TIRP (Σ_r)	sid	eid	starts	ends	Source intervals	vs	mhs	md
-	1	3	11:00	13:00	[11:00,13:00]	1.0	0.39	2.33
	2	1	14:00	17:00	[14:00,17:00]			
	3	1	10:00	13:00	[10:00,13:00]			
		4	14:00	15:00	[14:00,15:00]			

ABC

TIRP (Σ_r)	sid	eid	starts	ends	Source intervals	vs	mhs	md
sbo	1	3	08:00	13:00	[08:00,10:00],[08:00,12:00], [11:00,13:00]	0.33	0.33	5

Figure B.3: Illustration of the extension $AB - C$.

Appendix C

TA4L's appendix

C.1 Implementation of TA4L

The implementation of TA4L is available from the Bitbucket repository at "git clone https://invited2bynatalia@bitbucket.org/natalia_mordvanyuk/td4l.git". The following username and password can be used to log in:

username: invited2bynatalia

email: invited2bynatalia@gmail.com

password: 2wGW4hxgMVUtGkF..,

C.2 Auxiliar functions of TA4L

This section provides auxiliary functions used in the main TA4L algorithm. The first one is the FDC function.

The inputs of the FDC algorithm include time-series TS, the maximum gap max_gap allowed between two consecutive time intervals, the minimum duration of time intervals $duration$, and $LSTIS_{seq}$, that is the LSTIS for the sequence seq. The algorithm returns $LSTIS_{seq}$ updated with all new time intervals.

The first step of the algorithm consists of obtaining the first two TIs (the previous TI and the actual TI). If the previous and the actual TI have the same symbol, the end of the previous TI will be updated with the end of the actual TI unless the gap between them is greater than max_gap. The algorithm introduces the previous TI to the LSTIS in each iteration. The current TI is inserted when there are no more elements in the series.

```
Algorithm 9: FDC
   input: TS: multivariate time series
          max_gap: the maximum gap constraint
          duration: duration of the interval
          $LSTIS_{seq}$: list of LSTIs
1  output: $LSTIS_{seq}$: list of LSTIs for sequence 'seq'
2  i = 1
3  <$TI_{prev}$, i> = updateTI($TS$, i, duration, NULL, max_gap)
4  <$TI_{curr}$, i> = updateTI($TS$, i, duration, $TI_{prev}$,max_gap)
5  while i < len($TS$) do
6      while ($TI_{curr}$ != NULL) and ($TI_{prev}.sym$ == $TI_{curr}.sym$) do
7          <$TI_{curr}$, i> = updateTI($TS$, i, duration, $TI_{prev}$,max_gap)
8      $LSTIS_{seq}$ = insert($TI_{prev}$, $LSTIS_{seq}$)
9      if (i >= len($TS$)) and $TI_{curr}$ != NULL then
           /* last element                                            */
10         if ($TI_{prev}$ != $TI_{curr}$) then
11             $LSTIS_{seq}$ = insert($TI_{curr}$,$LSTIS_{seq}$)
12     $TI_{prev}$ = $TI_{curr}$
13 return $LSTIS_{seq}$
```

The inputs of the SCS algorithm include time-series TS, the maximum gap *max_gap* allowed between two consecutive time intervals and the $LSTIS_{seq}$, that is the LSTIS for the sequence *seq*. The algorithm returns $LSTIS_{seq}$ updated with all new time intervals.

We can see that in the loop, in each iteration, a new TI is being added within LSTISs. To create the TI, you need the symbol, the start time, and the end time of the TI. The symbol and the initial time are set at the beginning of the loop. If the current symbol is equal to the previous symbol and the maximum gap is not exceeded, the counter is incremented to determine the final time. If any of these conditions are not met, the final time is set.

When we reach the end of the time series, it must be decided whether to change the final time of the last TI or whether to create a new TI and insert it into LSTISs. This decision is made in the lines 17 - 23.

Algorithm 10: SCS

```
input: TS: multivariate time series
       max_gap: the maximum gap constraint
       $LSTIS_{seq}$: list of LSTIs
output: $LSTIS_{seq}$ : list of LSTIs for sequence 'seq'
i = 1
while i < (len($TS$)-1) do
    $sym_{cur}$ = getValue(TS, i)
    $sym_{next}$ = getValue(TS, i+1)
    $start_{TI}$ = getTimestamp(TS, i)
    gap = getTimestamp(TS, i+1) - getTimestamp(TS, i)
    while (i < (len($TS$)-1)) and ($sym_{cur}$ == $sym_{next}$) and gap<max_gap do
        i = i+1
        $sym_{cur}$ = getValue(TS, i)
        $sym_{next}$ = getValue(TS, i+1)
        gap = getTimestamp(TS, i+1) - getTimestamp(TS, i)
    $end_{TI}$ = getTimestamp(TS, i)
    $TI_{curr}$ = TI($sym_{cur}$, $start_{TI}$,$end_{TI}$)
    $LSTIS_{seq}$ = insert($TI_{prev}$,$LSTIS_{seq}$)
if i > 1 and i < (len($TS$) then
    last_time = getTimestamp(TS, i)
    gap = getTimestamp(TS,i)-getTimestamp(TS,i-1)
    if (getValue(TS,i) == getValue(TS,i-1) and gap <= max_gap then
        $LSTIS_{seq}$ = changeEnd($TI_{curr}$, last_time, $LSTIS_{seq}$)
    else
        $TI_{curr}$ = TI($sym_{cur}$, last_time, last_time)
        $LSTIS_{seq}$ = insert($TI_{prev}$,$LSTIS_{seq}$)
return $LSTIS_{seq}$
```