

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Lê Hoàng Bình Nguyên

PHÁT TRIỂN THUẬT TOÁN KHAI THÁC HIỆU QUẢ
MẪU TRỌNG SỐ HỮU ÍCH PHỔ BIẾN TRÊN LUỒNG DỮ LIỆU

LUẬN ÁN TIẾN SĨ TOÁN TIN

Hà Nội - 2026

ĐẠI HỌC QUỐC GIA HÀ NỘI
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

Lê Hoàng Bình Nguyên

PHÁT TRIỂN THUẬT TOÁN KHAI THÁC HIỆU QUẢ
MẪU TRỌNG SỐ HỮU ÍCH PHỔ BIẾN TRÊN LUỒNG DỮ LIỆU

Chuyên ngành: Cơ sở toán học cho tin học

Mã số: 9460110

LUẬN ÁN TIẾN SĨ TOÁN TIN

NGƯỜI HƯỚNG DẪN KHOA HỌC:

- PGS.TS. Nguyễn Thị Hồng Minh
- PGS.TS. Nguyễn Duy Hàm

Hà Nội - 2026

LỜI CAM ĐOAN

Tôi xin cam đoan luận án này là công trình nghiên cứu của riêng tôi dưới sự hướng dẫn của cán bộ hướng dẫn khoa học. Các số liệu, kết quả đạt được trong luận án là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác.

Tác giả luận án

Lê Hoàng Bình Nguyên

LỜI CẢM ƠN

Luận án này được thực hiện và hoàn thành trong quá trình học tập, nghiên cứu tại Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội. Đây không chỉ là kết quả từ những nỗ lực của bản thân mà còn là thành quả của sự hướng dẫn, hỗ trợ và động viên quý báu từ nhiều cá nhân và tập thể trong suốt thời gian qua. Một hành trình đầy thử thách nhưng cũng rất ý nghĩa đối với tôi.

Trước hết, tôi xin bày tỏ lòng biết ơn sâu sắc đến PGS.TS. Nguyễn Thị Hồng Minh và PGS.TS. Nguyễn Duy Hàm, cán bộ hướng dẫn khoa học của tôi. Cô và Thầy là người luôn tận tình chỉ bảo, định hướng nghiên cứu và dành nhiều thời gian để góp ý, chỉnh sửa trong suốt quá trình thực hiện luận án. Sự nghiêm túc trong học thuật, tinh thần trách nhiệm và sự động viên của Cô và Thầy là nguồn động lực to lớn giúp tôi hoàn thành luận án này.

Tôi xin trân trọng cảm ơn Ban Giám hiệu, Lãnh đạo Khoa Toán – Cơ – Tin học, quý thầy cô, giảng viên Bộ môn Tin học và các cán bộ của Trường Đại học Khoa học Tự nhiên, Đại học Quốc gia Hà Nội đã tạo điều kiện thuận lợi nhất trong suốt quá trình học tập, nghiên cứu và hoàn thành luận án của Nghiên cứu sinh.

Tôi cũng xin gửi lời cảm ơn chân thành đến Ban Lãnh đạo và các đồng nghiệp tại Trường Cao đẳng An ninh mạng ISPACE đã tạo điều kiện về thời gian và công việc để tôi có thể hoàn thành luận án này. Xin cảm ơn các nhà khoa học, nhóm nghiên cứu đã đóng góp ý kiến quý báu và hỗ trợ tôi trong quá trình thực hiện luận án.

Cuối cùng, tôi xin bày tỏ lòng biết ơn sâu sắc đến gia đình, những người luôn ở bên, động viên, chia sẻ và là động lực, chỗ dựa tinh thần vững chắc để tôi vượt qua những khó khăn trong suốt chặng đường học tập, nghiên cứu và hoàn thành luận án.

Hà Nội, ngày tháng năm 2026

Tác giả luận án

Lê Hoàng Bình Nguyên

MỤC LỤC

LỜI CAM ĐOAN

LỜI CẢM ƠN

MỤC LỤC 1

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT 4

DANH MỤC CÁC BẢNG 6

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ 8

MỞ ĐẦU 12

CHƯƠNG 1: TỔNG QUAN VỀ LĨNH VỰC NGHIÊN CỨU 20

1.1 Bài toán khai thác mẫu trên cơ sở dữ liệu định lượng 20

1.1.1 Bài toán khai thác mẫu hữu ích cao 20

1.1.2 Bài toán khai thác mẫu trọng số hữu ích phổ biến 21

1.2 Bài toán khai thác mẫu trên luồng dữ liệu 24

1.3 Các biến thể của bài toán khai thác mẫu 29

1.3.1 Bài toán khai thác mẫu đóng 29

1.3.2 Bài toán khai thác top- k mẫu 33

1.4 Khoảng trống nghiên cứu, thách thức và hướng giải quyết 36

1.5 Kết luận chương 39

CHƯƠNG 2: KHAI THÁC MẪU TRỌNG SỐ HỮU ÍCH PHỔ BIẾN TỪ CƠ SỞ DỮ LIỆU ĐỊNH LƯỢNG ĐỘNG 40

2.1 Giới thiệu bài toán 40

2.2 Một số định nghĩa cơ bản 43

2.3 Thuật toán khai thác mẫu trọng số hữu ích phổ biến từ cơ sở dữ liệu định lượng động 47

2.3.1 Thuật toán dFWUT.....	48
2.3.2 Thuật toán dFWUNL.....	51
2.3.3 Tính đúng đắn và độ phức tạp thuật toán	63
2.3.4 Thực nghiệm và kết quả	66
2.4 Thuật toán khai thác mẫu trọng số hữu ích phổ biến đóng từ cơ sở dữ liệu định lượng động	74
2.4.1 Các chiến lược cắt tĩa	75
2.4.2 Thuật toán FWUCWP	84
2.4.3 Tính đúng đắn và độ phức tạp thuật toán	88
2.4.4 Thực nghiệm và kết quả	89
2.5 Kết luận chương	101
CHƯƠNG 3: KHAI THÁC MẪU TRỌNG SỐ HỮU ÍCH PHỔ BIẾN TRÊN LUỒNG DỮ LIỆU ĐỊNH LƯỢNG ĐỘNG.....	103
3.1 Giới thiệu bài toán.....	103
3.2 Một số định nghĩa	105
3.3 Thuật toán khai thác mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động	107
3.3.1 Thuật toán FWUDS-CT.....	108
3.3.2 Thuật toán FWUDS-DWT.....	116
3.3.3 Tính đúng đắn và độ phức tạp thuật toán	123
3.3.4 Thực nghiệm và kết quả	127
3.4 Khai thác top- k mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động.....	140
3.4.1 Ngưỡng hỗ trợ trọng số hữu ích động	141
3.4.2 Thuật toán TKFWUDS.....	144

3.4.3 Tính đúng đắn và độ phức tạp thuật toán	145
3.4.4 Thực nghiệm và kết quả	147
3.5 Kết luận chương	154
KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN.....	155
DANH MỤC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN.....	158
TÀI LIỆU THAM KHẢO	159

DANH MỤC CÁC KÝ HIỆU VÀ CHỮ VIẾT TẮT

Từ viết tắt	Nội dung/Ý nghĩa
CSDL	Cơ sở dữ liệu (Database)
CTset	Tập định danh giao dịch vòng (Circular Tidset)
dFWUNL	Thuật toán khai thác mẫu trọng số hữu ích phổ biến từ CSDL định lượng động sử dụng cấu trúc WUNList (Dynamic FWUP Mining Using WUNList Structure)
dFWUT	Thuật toán khai thác mẫu trọng số hữu ích phổ biến từ CSDL định lượng động sử dụng cấu trúc Tidset (Dynamic FWUP Mining Using Tidset Structure)
dQDB	Cơ sở dữ liệu định lượng động (Dynamic Quantitative Database)
DSWUN-tree	Cây trọng số hữu ích cho luồng dữ liệu động (Dynamic Streaming Weighted Utility Node Tree)
FCP	Mẫu phổ biến đóng (Frequent Closed Pattern)
FP	Mẫu phổ biến (Frequent Pattern)
FWUCP	Mẫu trọng số hữu ích phổ biến đóng (Frequent Weighted Utility Closed Pattern)
FWUCWP	Thuật toán khai thác mẫu trọng số hữu ích phổ biến đóng từ CSDL định lượng động sử dụng cấu trúc WUNList (FWUCP Mining Using WUNList Structure)
FWUDS-CT	Thuật toán khai thác mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động sử dụng cấu trúc Circular Tidset (FWUP Mining Over Data Streams Using Circular Tidset)
FWUDS-DWT	Thuật toán khai thác mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động sử dụng cấu trúc DSWUN-tree (FWUP Mining Over Data Streams Using DSWUN-tree)
FWUP	Mẫu trọng số hữu ích phổ biến (Frequent Weighted Utility Pattern)
HUP	Mẫu hữu ích cao (High Utility Pattern)

<i>lcTid</i>	Định danh giao dịch cục bộ (Local Transaction Identifier)
<i>minwus</i>	Độ hỗ trợ trọng số hữu ích tối thiểu (Minimum Weighted Utility Support)
MTCP	Chiến lược cắt tỉa kết hợp <i>twu</i> tối đa (Maximal Twu Combination Pruning)
PTP	Bảng lưu trữ <i>twu</i> của cặp mục (Precomputed Twu Pairwise)
<i>Tid</i>	Định danh giao dịch (Transaction Identifier)
TKFWUDS	Thuật toán khai thác top- <i>k</i> mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động (Top- <i>k</i> FWUP Mining Over Dynamic Quantitative Data Streams)
<i>twu</i>	Trọng số hữu ích giao dịch (Transaction Weighted Utility)
WASR	Quan hệ cấu trúc tổ tiên WUNList (WUNList Ancestor Structural Relation)
WUNList	Danh sách nút trọng số hữu ích (Weighted Utility Node List)
WUN-tree	Cây trọng số hữu ích (Weighted Utility Node Tree)
<i>wus</i>	Độ hỗ trợ trọng số hữu ích (Weighted Utility Support)

DANH MỤC CÁC BẢNG

Bảng 1.1. Các nghiên cứu về khai thác mẫu trên CSDL định lượng	23
Bảng 1.2. Các nghiên cứu về khai thác mẫu trên luồng dữ liệu sử dụng mô hình cửa sổ trượt	28
Bảng 1.3. Các nghiên cứu về khai thác mẫu đóng	32
Bảng 1.4. Các nghiên cứu về khai thác top- k mẫu	35
Bảng 2.1. Ví dụ về CSDL định lượng có trọng số tĩnh.....	40
Bảng 2.2. Ví dụ về CSDL định lượng động ($dQDB_E$).....	42
Bảng 2.3. Giá trị twu của tất cả các giao dịch trong $dQDB_E$	45
Bảng 2.4. Tidset và wus của các mục trong $dQDB_E$	50
Bảng 2.5. $dQDB_E$ sau khi áp dụng ngưỡng $minwus = 0.5$	53
Bảng 2.6. Các đặc điểm của CSDL thực nghiệm.....	67
Bảng 2.7. Kết quả khai thác FWUP trên các bộ dữ liệu thực nghiệm	72
Bảng 2.8. Khả năng mở rộng của dFWUT, dFWUNL và WUN-Miner-Mod trên bộ dữ liệu Chainstore với $minwus = 0.02\%$	74
Bảng 2.9. Các FWUP và FWUCP từ $dQDB_E$ với $minwus = 0.5$	75
Bảng 2.10. Enclosure của các mục trong I_1	82
Bảng 2.11. Kết quả khai thác FWUCP trên các bộ dữ liệu thực nghiệm	94
Bảng 2.12. Khả năng mở rộng của NFWCI và FWUCWP trên bộ dữ liệu Chainstore với $minwus = 0.07\%$	95
Bảng 2.13. Tác động của từng chiến lược cắt tĩa đến số lượng các ứng viên được tạo ra trên bộ dữ liệu Pumsb	99
Bảng 2.14. Tác động của từng chiến lược cắt tĩa đến số lượng các ứng viên được tạo ra trên bộ dữ liệu Retail.....	99

Bảng 2.15. Tác động của từng chiến lược cắt tĩa đến số lượng các ứng viên được tạo ra trên bộ dữ liệu Kosarak	100
Bảng 2.16. So sánh số lượng FWUCP và FWUP được tạo ra	101
Bảng 3.1. Tập trọng số của các mục	106
Bảng 3.2. Các tham số thực nghiệm khai thác FWUP trên luồng dữ liệu	128
Bảng 3.3. Tham số thực nghiệm khả năng mở rộng của các thuật toán khai thác FWUP trên luồng dữ liệu theo kích thước cửa sổ và pane.....	128
Bảng 3.4. Kết quả khai thác FWUP trên các luồng dữ liệu thực nghiệm	136
Bảng 3.5. Khả năng mở rộng của các thuật toán khai thác FWUP trên luồng dữ liệu theo kích thước cửa sổ và pane	139
Bảng 3.6. Các tham số thực nghiệm khai thác top- k FWUP trên luồng dữ liệu	148
Bảng 3.7. Các tham số thực nghiệm khả năng mở rộng của các thuật toán khai thác top- k FWUP trên luồng dữ liệu theo kích thước cửa sổ và pane.....	148
Bảng 3.8. Thời gian thực thi và khả năng mở rộng của các thuật toán khai thác top- k FWUP trên các luồng dữ liệu thực nghiệm.....	153

DANH MỤC CÁC HÌNH VẼ, ĐỒ THỊ

Hình 1.1. Các loại mô hình khai thác mẫu trên luồng dữ liệu.	25
Hình 1.2. Sơ đồ mối quan hệ giữa các bài toán được giải quyết trong luận án.	38
Hình 2.1. Thuật toán dFWUT.	49
Hình 2.2. dWUPT-tree từ $dQDB_E$ với $minwus = 0.5$	51
Hình 2.3. Thuật toán xây dựng cấu trúc WUN-tree.	52
Hình 2.4. WUN-tree của $dQDB_E$ với $minwus = 0.5$	53
Hình 2.5. Thuật toán tạo WUNList của mẫu mới.	57
Hình 2.6. Thuật toán dFWUNL.	61
Hình 2.7. I_1 và WUNList của các 1-FWUP từ $dQDB_E$ với $minwus = 0.5$	61
Hình 2.8. Kết quả sau khi kết hợp B với các 1-FWUP.	62
Hình 2.9. Kết quả sau khi kết hợp A với các 1-FWUP.	62
Hình 2.10. Tất cả FWUP được khám phá bởi thuật toán dFWUNL từ $dQDB_E$ với $minwus = 0.5$	63
Hình 2.11. Thời gian chạy của các thuật toán dFWUT, dFWUNL và WUN-Miner-Mod trên các bộ dữ liệu thực nghiệm.	69
Hình 2.12. Bộ nhớ sử dụng của các thuật toán dFWUT, dFWUNL và WUN-Miner-Mod trên các bộ dữ liệu thực nghiệm.	71
Hình 2.13. Khả năng mở rộng về số lượng mục trên bộ dữ liệu Chainstore.	73
Hình 2.14. Khả năng mở rộng về số lượng giao dịch trên bộ dữ liệu Chainstore. ...	73
Hình 2.15. Thuật toán kiểm tra quan hệ cấu trúc tổ tiên giữa hai WUNList.	78
Hình 2.16. Thuật toán cải tiến kiểm tra quan hệ cấu trúc tổ tiên giữa hai WUNList.	79
Hình 2.17. Ma trận tam giác PTP của $dQDB_E$ với $minwus = 0.5$	80

Hình 2.18. Thuật toán xây dựng PTP và Enclosure.	83
Hình 2.19. Thuật toán FWUCWP.	86
Hình 2.20. WUNList và Enclosure của I_1 để bắt đầu khai thác.	86
Hình 2.21. Kết quả sau khi kết hợp lớp tương đương B	87
Hình 2.22. Kết quả sau khi kết hợp lớp tương đương A	87
Hình 2.23. Kết quả sau khi kết hợp lớp tương đương E và C	87
Hình 2.24. Thời gian chạy của thuật toán NFWCI và FWUCWP trên các CSDL thực nghiệm.	91
Hình 2.25. Bộ nhớ sử dụng của thuật toán NFWCI và FWUCWP trên các CSDL thực nghiệm.	93
Hình 2.26. Khả năng mở rộng của thuật toán NFWCI và FWUCWP với số lượng mục khác nhau.	95
Hình 2.27. Khả năng mở rộng của thuật toán NFWCI và FWUCWP với số lượng giao dịch khác nhau.	95
Hình 2.28. Tác động của từng chiến lược cắt tĩa đến hiệu năng thời gian chạy trên bộ dữ liệu Pumsb.	97
Hình 2.29. Tác động của từng chiến lược cắt tĩa đến hiệu năng thời gian chạy trên bộ dữ liệu Retail.	97
Hình 2.30. Tác động của từng chiến lược cắt tĩa đến hiệu năng thời gian chạy trên bộ dữ liệu Kosarak.	98
Hình 3.1. Luồng dữ liệu định lượng động ví dụ (DS_e).	106
Hình 3.2. DS_e với giá trị Tid và $lcTid$	109
Hình 3.3. Thuật toán thêm và xoá pane sử dụng cấu trúc CTset.	111
Hình 3.4. Quá trình cập nhật CTset của các mục khi trượt khung cửa sổ từ W_1 sang W_2 trong DS_e	112

Hình 3.5. Thuật toán giao hai CTset.	114
Hình 3.6. Thuật toán FWUDS-CT.	115
Hình 3.7. Thuật toán xây dựng DSWUN-tree.....	117
Hình 3.8. DSWUN-tree được tạo thành từ W_1 của DS_e	118
Hình 3.9. Thuật toán cập nhật DSWUN-tree.	120
Hình 3.10. Quá trình cập nhật DSWUN-tree từ cửa sổ W_1 sang W_2 của DS_e	121
Hình 3.11. Thuật toán FWUDS-DWT.	122
Hình 3.12. Thời gian cập nhật dữ liệu trên CSDL Retail với <i>minwus</i> khác nhau.	130
Hình 3.13. Thời gian cập nhật dữ liệu trên CSDL Accidents với <i>minwus</i> khác nhau.	130
Hình 3.14. Thời gian cập nhật dữ liệu trên CSDL BMS-POS với <i>minwus</i> khác nhau.	131
Hình 3.15. Thời gian cập nhật dữ liệu trên CSDL Kosarak với <i>minwus</i> khác nhau.	131
Hình 3.16. Thời gian cập nhật dữ liệu trên CSDL Susy với <i>minwus</i> khác nhau.	132
Hình 3.17. Tổng thời gian khai thác FWUP trên các CSDL thực nghiệm với <i>minwus</i> khác nhau.	133
Hình 3.18. Bộ nhớ sử dụng cho quá trình khai thác FWUP trên các bộ dữ liệu thực nghiệm với <i>minwus</i> khác nhau.	135
Hình 3.19. Thời gian cập nhật dữ liệu trên CSDL Kosarak với kích thước cửa sổ và kích thước pane khác nhau.	137
Hình 3.20. Thời gian cập nhật dữ liệu trên CSDL Susy với kích thước cửa sổ và kích thước pane khác nhau.	137
Hình 3.21. Khả năng mở rộng về thời gian khai thác FWUP với kích thước cửa sổ và pane khác nhau trên CSDL Kosarak và Susy.	138
Hình 3.22. Khả năng mở rộng về bộ nhớ sử dụng để khai thác FWUP với kích thước cửa sổ và pane khác nhau trên CSDL Kosarak và Susy.	139

Hình 3.23. Thuật toán TKFWUDS.	145
Hình 3.24. Thời gian chạy của thuật toán TKFWUDS và dFWUNL-mod trên các CSDL thực nghiệm.....	150
Hình 3.25. Khả năng mở rộng về thời gian khai thác của thuật toán TKFWUDS và dFWUNL-mod trên các CSDL thực nghiệm.	152

MỞ ĐẦU

1. Lý do chọn đề tài

Khai thác dữ liệu là quá trình khám phá các mẫu tiềm ẩn và những tri thức có giá trị từ khối lượng dữ liệu lớn. Các công cụ khai thác dữ liệu giúp dự báo những xu hướng trong tương lai và nhận diện những quy luật phục vụ những yêu cầu đặt ra trong hiện tại. Những tri thức này hỗ trợ cho các doanh nghiệp, tổ chức đưa ra các quyết định hợp lý, kịp thời, giảm thiểu rủi ro và nắm bắt các cơ hội mới một cách hiệu quả hơn. Hiện nay, với sự bùng nổ của Big Data và sự tiến bộ của công nghệ lưu trữ, việc phát triển các kỹ thuật khai thác tri thức thông minh ngày càng trở nên cấp thiết trong mọi lĩnh vực của đời sống xã hội.

Một trong những bài toán cơ bản, đóng vai trò then chốt trong khai thác dữ liệu là khai thác mẫu phổ biến (FP). Mục tiêu của bài toán này là xác định các mẫu có tần số xuất hiện thỏa mãn ngưỡng cho trước trong các cơ sở dữ liệu (CSDL) giao dịch quy mô lớn. Bài toán khai thác FP được đề xuất đầu tiên vào năm 1993 bởi Agrawal và đồng sự [1]. Kể từ đó, nhiều thuật toán và phương pháp khác nhau đã được phát triển nhằm nâng cao hiệu quả giải quyết bài toán, tiêu biểu như FP-Growth [26], Eclat [93], PrePost [14].... Mặc dù đạt được nhiều kết quả quan trọng, nhưng thực tiễn triển khai cho thấy mô hình khai thác FP truyền thống vẫn tồn tại những hạn chế lớn. Hạn chế cốt lõi xuất phát từ giả định rằng tất cả các mục đều có mức độ quan trọng như nhau và chỉ được đánh giá dựa trên tần suất xuất hiện. Tuy nhiên, trong thực tế, các mục trong CSDL có sự khác nhau trên nhiều khía cạnh. Ví dụ, trong lĩnh vực thương mại, các mặt hàng có sự khác nhau về giá cả, số lượng và lợi nhuận mang lại. Những sản phẩm có giá trị cao có thể đóng góp phần lớn vào tổng doanh thu mặc dù chúng chỉ xuất hiện trong một số ít giao dịch. Việc chỉ dựa trên tần suất xuất hiện có thể dẫn đến bỏ sót các mẫu quan trọng phục vụ cho quá trình ra quyết định. Do đó, nhiều biến thể của FP đã được nghiên cứu, như mẫu phổ biến có trọng số [6, 39], mẫu hữu ích cao (HUP) [83, 85], mẫu tuần tự [16, 73], nhằm đáp ứng các yêu cầu đa dạng trong các bài toán ứng dụng cụ thể.

Khai thác mẫu trọng số hữu ích phổ biến (FWUP) là một biến thể quan trọng và được tiến hành trên CSDL định lượng. Khác với CSDL nhị phân, CSDL định lượng lưu trữ thêm thông tin về số lượng của từng mục trong mỗi giao dịch, đồng thời mỗi mục còn được gán một giá trị trọng số phản ánh mức độ quan trọng. Sự ràng buộc về tần số xuất hiện, số lượng và trọng số của các mục có ảnh hưởng lớn đến việc trích xuất chính xác các thông tin quan trọng. Khai thác FWUP tập trung vào việc tìm kiếm các mẫu vừa xuất hiện thường xuyên, vừa mang lại nhiều giá trị. Khái niệm mẫu trọng số hữu ích phổ biến đã được Khan và các đồng sự [31] đề xuất nhằm phản ánh sự khác biệt về tầm quan trọng cũng như tần suất của các mẫu thông qua hai thành phần là trọng số và độ hữu ích. Tác giả cũng đề xuất thuật toán Apriori-FWUI để khai thác FWUP từ CSDL định lượng. Sau đó, nhiều thuật toán được đề xuất như MWIT-FWUI [75], MBiS-FWUI [25], WUN-Miner [5] để nâng cao hiệu quả khai thác.

Tuy nhiên, các nghiên cứu về khai thác FWUP hiện tại chỉ tập trung vào CSDL tĩnh với giá trị trọng số của mỗi mục là cố định, không thay đổi theo thời gian. Thực tế cho thấy tầm quan trọng của mỗi mục có thể bị ảnh hưởng bởi nhiều yếu tố khác nhau như thay đổi trong hành vi của người tiêu dùng, tính mùa vụ hoặc chiến lược kinh doanh. Chẳng hạn, các sản phẩm quần áo ấm thường có nhu cầu cao vào đầu mùa đông, hoặc có thể được giảm giá mạnh để giải phóng hàng tồn kho trước khi ra mắt sản phẩm mới. Điều này đòi hỏi trọng số của các mục cần được cập nhật một cách linh hoạt. Việc bỏ qua đặc tính định lượng động này dẫn đến hạn chế trong khả năng áp dụng của các thuật toán hiện có, đồng thời làm giảm chất lượng kết quả khai thác. Kết quả khảo sát tại thời điểm hình thành hướng nghiên cứu của luận án cho thấy chưa có công trình trực tiếp giải quyết bài toán khai thác FWUP từ CSDL định lượng động trong đó trọng số của các mục thay đổi theo thời gian. Do đó, nghiên cứu các phương pháp khai thác FWUP thích ứng với sự thay đổi trọng số là một hướng đi cần thiết và cấp bách.

Bên cạnh đó, sự bùng nổ số lượng mẫu được sinh ra trong quá trình khai thác thường gây áp lực lớn lên tài nguyên hệ thống, làm gia tăng chi phí lưu trữ và khó khăn trong việc phân tích tri thức. Khai thác mẫu phổ biến đóng (FCP) đã được chứng

minh là một giải pháp hiệu quả để giảm không gian mẫu mà vẫn bảo toàn thông tin. Một FCP là mẫu phổ biến mà không có tập cha nào của nó có cùng độ hỗ trợ. Số lượng FCP thường ít hơn đáng kể so với FP và từ FCP, ta có thể tái tạo lại tất cả các FP mà không làm mất thông tin. Nhiều kỹ thuật khai thác mẫu đóng đã được đề xuất như khai thác mẫu phổ biến đóng [2, 35], mẫu phổ biến đóng có trọng số [7, 74], mẫu hữu ích cao đóng [15, 57]. Tuy nhiên, cho đến nay, bài toán khai thác mẫu trọng số hữu ích phổ biến đóng (FWUCP) vẫn chưa được nghiên cứu một cách đầy đủ, đặc biệt trong bối cảnh dữ liệu có trọng số biến đổi theo thời gian.

Hơn nữa, trong các hệ thống hiện đại, dữ liệu thường được sinh ra một cách liên tục với tốc độ cao và khối lượng lớn dưới dạng các luồng dữ liệu. Điều này dẫn đến việc khám phá các mẫu trên luồng dữ liệu là một thách thức lớn bởi vì việc lưu trữ toàn bộ dữ liệu để xử lý là không khả thi. Sử dụng các phương pháp truyền thống để khai thác những dạng mẫu như vậy trên luồng dữ liệu có thể không có tính thực tiễn bởi vì hạn chế về bộ nhớ và khả năng tính toán của hệ thống. Các thuật toán khai thác trên luồng dữ liệu đòi hỏi phải đáp ứng được các yêu cầu nghiêm ngặt như xử lý một lần, cập nhật dữ liệu theo thời gian thực và sử dụng bộ nhớ hiệu quả. Trong bối cảnh đó, việc áp dụng mô hình cửa sổ trượt để khai thác FWUP mới nhất trên luồng dữ liệu định lượng động là một hướng nghiên cứu cấp thiết nhưng đồng thời cũng đặt ra nhiều thách thức, do vấn đề này chưa được quan tâm đầy đủ trong các nghiên cứu hiện có. Đối với luồng dữ liệu định lượng động, bài toán phải xử lý cả nội dung dữ liệu và trọng số của các mục có thể thay đổi theo thời gian. Do đó, cần phát triển các cơ chế biểu diễn và cập nhật dữ liệu phù hợp nhằm hạn chế tối đa việc quét lại toàn bộ cửa sổ, đồng thời bảo đảm hiệu quả và tính chính xác của quá trình khai thác khi dữ liệu liên tục thay đổi.

Ngoài ra, trong môi trường luồng dữ liệu biến động, việc xác định ngưỡng tối thiểu phù hợp để tiến hành khai thác là một vấn đề khó khăn cho người dùng. Một ngưỡng quá cao có thể dẫn đến việc bỏ sót các tri thức quan trọng, trong khi ngưỡng quá thấp lại gây ra hiện tượng bùng nổ số lượng mẫu không kiểm soát. Để giải quyết hạn chế này, bài toán khai thác top- k FWUP trên luồng dữ liệu định lượng động sử

dụng mô hình cửa sổ trượt được xem là một giải pháp hiệu quả. Thay vì dựa vào một ngưỡng xác định trước, phương pháp này tập trung tìm kiếm k mẫu có giá trị cao nhất, giúp người dùng nắm bắt các xu hướng hàng đầu một cách ổn định và trực quan hơn. Đồng thời, việc xác định tham số k là thân thiện và dễ tiếp cận hơn đối với người dùng phổ thông cũng như các hệ thống thông minh.

Xuất phát từ những đòi hỏi cấp thiết về mặt lý thuyết và thực tiễn nêu trên, Nghiên cứu sinh chọn đề tài **“Phát triển thuật toán khai thác hiệu quả mẫu trọng số hữu ích phổ biến trên luồng dữ liệu”** làm đề tài nghiên cứu cho luận án tiến sĩ.

2. Mục tiêu nghiên cứu

Cơ sở dữ liệu định lượng thường xuyên xuất hiện trong nhiều lĩnh vực của đời sống xã hội như kinh doanh, thương mại, tài chính và ngân hàng. Việc trích xuất các thông tin có giá trị từ loại dữ liệu này là một nhu cầu cấp thiết. Tuy nhiên, các phương pháp khai thác FWUP trên CSDL định lượng truyền thống còn nhiều hạn chế khi áp dụng vào thực tiễn, đặc biệt là trọng số các mục không cố định mà có thể thay đổi theo thời gian và dữ liệu được sinh ra liên tục dưới dạng luồng dữ liệu. Do đó, đề tài **“Phát triển thuật toán khai thác hiệu quả mẫu trọng số hữu ích phổ biến trên luồng dữ liệu”** được lựa chọn để giải quyết các thách thức còn tồn tại.

Dựa trên việc khảo sát các công trình nghiên cứu liên quan và những kết quả đã có về bài toán khai thác mẫu nói chung và khai thác FWUP nói riêng, luận án xác định các mục tiêu và nhiệm vụ nghiên cứu chính như sau:

- Nghiên cứu bài toán tổng quát về khai thác mẫu trọng số hữu ích phổ biến từ CSDL định lượng động, với trọng số của các mục có thể thay đổi. Từ đó, phát triển các cấu trúc dữ liệu, phương pháp và thuật toán để giải quyết bài toán này. Đồng thời mở rộng sang bài toán khai thác mẫu trọng số hữu ích phổ biến đóng, là dạng rút gọn nhưng không làm mất thông tin của FWUP.

- Nghiên cứu bài toán khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt, trong đó dữ liệu và trọng số các mục thay đổi theo thời gian. Trên cơ sở đó, phát triển các thuật toán và phương pháp cập nhật, khai thác hiệu

quả, đồng thời mở rộng sang bài toán khai thác top- k FWUP trên luồng dữ liệu định lượng động nhằm giảm sự phụ thuộc của việc lựa chọn ngưỡng tối thiểu.

3. Đối tượng và phạm vi nghiên cứu

- Đối tượng nghiên cứu:

+ Luận án tập trung nghiên cứu các thuật toán khai thác FWUP, bao gồm cả các phương pháp truyền thống và các hướng tiếp cận cải tiến hiện đại.

+ Nghiên cứu các kỹ thuật khai thác mẫu đóng và top- k mẫu trên nhiều loại CSDL khác nhau, bao gồm CSDL nhị phân, CSDL có trọng số và CSDL định lượng.

+ Phân tích và đánh giá các phương pháp khai thác mẫu trên luồng dữ liệu, trong đó tập trung vào mô hình cửa sổ trượt.

- Phạm vi nghiên cứu:

+ Khảo sát các dạng CSDL định lượng, từ dạng tĩnh đến các mô hình động có sự thay đổi về trọng số.

+ Nghiên cứu mô hình luồng dữ liệu định lượng động với đặc trưng là trọng số biến đổi, làm nền tảng cho việc phát triển và đánh giá các thuật toán đề xuất.

+ Sử dụng các bộ CSDL định lượng đã được chuẩn hóa cho hoạt động nghiên cứu khoa học nhằm phục vụ cho quá trình thực nghiệm, đảm bảo tính khách quan và độ tin cậy cho các kết quả trong luận án.

4. Phương pháp luận và phương pháp nghiên cứu

- Phương pháp nghiên cứu lý thuyết:

+ Tổng hợp, phân tích và đánh giá các tài liệu khoa học, bao gồm các bài báo, công trình nghiên cứu đã công bố có nội dung liên quan đến luận án.

+ Nghiên cứu các mô hình dữ liệu, tập trung vào CSDL định lượng động và luồng dữ liệu định lượng động có trọng số thay đổi.

+ Khảo sát và so sánh các phương pháp hiện có nhằm xác định những ưu điểm,

hạn chế và khoảng trống nghiên cứu.

+ Trên cơ sở đó, đề xuất các mô hình, cấu trúc dữ liệu và thuật toán nhằm nâng cao hiệu quả khai thác FWUP.

- Phương pháp nghiên cứu thực nghiệm:

+ Từ cơ sở lý thuyết đã nghiên cứu, tiến hành cài đặt các thuật toán đã đề xuất.

+ Thực nghiệm trên các bộ dữ liệu chuẩn để đánh giá hiệu năng của các thuật toán theo các tiêu chí như thời gian xử lý, mức sử dụng bộ nhớ và khả năng mở rộng.

+ So sánh kết quả thực nghiệm với các phương pháp hiện có để chứng minh tính hiệu quả và ưu việt của các phương pháp đề xuất.

5. Các đóng góp của luận án

Luận án tập trung giải quyết bài toán khai thác mẫu trọng số hữu ích phổ biến trong các môi trường dữ liệu định lượng phức tạp và biến động. Các đóng góp chính của luận án được hình thành theo hai hướng nghiên cứu có tính kế thừa và mở rộng, bao gồm:

- Phát triển các phương pháp khai thác mẫu trọng số hữu ích phổ biến từ CSDL định lượng động. Khác với các nghiên cứu trước đây giả định trọng số là cố định, luận án đã xây dựng cơ chế xử lý sự biến động của trọng số theo thời gian, qua đó phản ánh thực tế đặc trưng của các hệ thống dữ liệu hiện đại. Trên cơ sở đó, các thuật toán dFWUT sử dụng cấu trúc Tidset và dFWUNL sử dụng cấu trúc WUNList được đề xuất nhằm giải quyết hiệu quả bài toán này. Tiếp đó, hướng nghiên cứu được mở rộng sang bài toán khai thác mẫu trọng số hữu ích phổ biến đóng với thuật toán FWUCWP và các chiến lược cắt tỉa nhằm thu hẹp không gian tìm kiếm. Các kết quả này được công bố trong các công trình [I], [II] và [V]; công trình [III] là kết quả mở rộng theo hướng xử lý song song.

- Với các kết quả đạt được với CSDL định lượng động, luận án tiếp tục nghiên cứu bài toán khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt và đề xuất hai thuật toán FWUDS-CT và FWUDS-DWT để giải quyết

bài toán này. Thuật toán FWUDS-CT sử dụng cấu trúc Circular Tidset cải tiến nhằm quản lý hiệu quả định danh giao dịch và tránh hiện tượng tràn số khi xử lý các luồng dữ liệu vô hạn. Thuật toán FWUDS-DWT dựa trên cấu trúc DSWUN-tree và tận dụng sự hiệu quả của cấu trúc WUNList, cho phép cập nhật dữ liệu tức thời và thực hiện khai thác chỉ với một lần quét dữ liệu. Từ đó, luận án phát triển thuật toán TKFWUDS cho bài toán khai thác top- k FWUP trên luồng dữ liệu định lượng động, sử dụng cơ chế nâng ngưỡng kết hợp các chiến lược cắt tia dựa trên khái niệm ngưỡng hỗ trợ trọng số hữu ích động, giúp loại bỏ sự phụ thuộc vào ngưỡng tối thiểu do người dùng thiết lập. Các kết quả thuộc hướng nghiên cứu này được công bố trong các công trình [IV] và [VI].

- Thực nghiệm trên nhiều bộ dữ liệu chuẩn có đặc trưng khác nhau, bao gồm cả dữ liệu thưa và dữ liệu dày đặc, cho thấy các thuật toán đề xuất đạt hiệu quả cao hơn so với các phương pháp hiện có về thời gian xử lý, mức sử dụng bộ nhớ và khả năng mở rộng.

6. Cấu trúc luận án

Ngoài phần Mở đầu, Kết luận, nội dung chính của luận án được tổ chức trong ba chương, cụ thể như sau:

Chương 1: “Tổng quan về lĩnh vực nghiên cứu” trình bày tổng quát về bài toán khai thác mẫu. Chương này tập trung khảo sát và phân tích các công trình nghiên cứu liên quan, qua đó chỉ ra những hạn chế còn tồn tại của các phương pháp hiện có, làm cơ sở cho các đề xuất trong các chương tiếp theo.

Chương 2: “Khai thác mẫu trọng số hữu ích phổ biến từ cơ sở dữ liệu định lượng động” tập trung nghiên cứu bài toán khai thác FWUP trong môi trường dữ liệu động. Chương này trình bày cấu trúc dữ liệu Tidset và WUNList, trên cơ sở đó đề xuất thuật toán dFWUT và dFWUNL để khai thác FWUP với trọng số các mục có thể thay đổi. Đồng thời, bài toán khai thác FWUCP từ CSDL định lượng động cũng được giới thiệu cùng với thuật toán FWUCWP và các chiến lược cắt tia dựa trên cấu trúc WUNList để nâng cao hiệu quả khai thác.

Chương 3: “Khai thác mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động” mở rộng bài toán sang môi trường luồng dữ liệu. Chương này xây dựng mô hình khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt và phát triển các cấu trúc dữ liệu cải tiến kế thừa từ Chương 2 nhằm đáp ứng yêu cầu cập nhật liên tục. Từ đó, các thuật toán FWUDS-CT và FWUDS-DWT được đề xuất để khai thác hiệu quả FWUP. Cùng với đó, thuật toán TKFWUDS được phát triển cho bài toán khai thác top- k FWUP trên luồng dữ liệu, tích hợp cơ chế nâng ngưỡng động, giúp loại bỏ sự phụ thuộc vào ngưỡng tối thiểu truyền thống.

CHƯƠNG 1: TỔNG QUAN VỀ LĨNH VỰC NGHIÊN CỨU

1.1 Bài toán khai thác mẫu trên cơ sở dữ liệu định lượng

CSDL định lượng thường xuất hiện trong các ứng dụng thực tiễn, đặc biệt là trong lĩnh vực bán lẻ và phân tích giao dịch. Đặc trưng của loại dữ liệu này là mỗi giao dịch không chỉ biểu diễn sự xuất hiện của các mục mà còn lưu trữ số lượng tương ứng. Đồng thời, mỗi mục được gán một trọng số phản ánh các khía cạnh như giá trị, lợi nhuận hoặc mức độ quan trọng trong hệ thống. Những đặc điểm này đặt ra yêu cầu cần phát triển các phương pháp khai thác dữ liệu phù hợp nhằm trích xuất hiệu quả các tri thức.

1.1.1 Bài toán khai thác mẫu hữu ích cao

Hướng tiếp cận đầu tiên để khai thác thông tin có giá trị từ CSDL định lượng là khai thác mẫu hữu ích cao. Mục tiêu của bài toán này là tìm kiếm các mẫu có giá trị hữu ích lớn, trong đó một mẫu được xem là HUP nếu giá trị hữu ích của nó không nhỏ hơn ngưỡng hữu ích tối thiểu do người dùng xác định. Bài toán khai thác HUP được Yao và Hamilton [86] giới thiệu vào năm 2004. Thách thức lớn nhất của bài toán khai thác HUP đó là tính chất bao đóng giảm (downward closure property) không còn được đảm bảo. Cụ thể, các tập con hoặc tập cha của một HUP có thể là HUP hoặc không phải là HUP. Điều này dẫn đến không gian tìm kiếm của bài toán là rất lớn. Để giải quyết vấn đề này, Liu và đồng sự [47] đã đề xuất thuật toán Two-Phase dựa trên Apriori, sử dụng khái niệm transaction weighted utilization làm giới hạn trên thoả mãn tính chất bao đóng giảm, qua đó hỗ trợ hiệu quả cho quá trình cắt tỉa. Tiếp theo, dựa trên cảm hứng của thuật toán FP-Growth [26], các thuật toán UP-Growth [71] và UP-Growth+ [68] đã được phát triển với nhiều chiến lược nhằm cải thiện hiệu suất khai thác. Một bước tiến quan trọng khác là thuật toán HUI-Miner [45] do Liu và Qu đề xuất cho phép khai thác HUP chỉ với một lần quét CSDL. HUI-Miner sử dụng cấu trúc mới có tên là utility-list để lưu trữ thông tin cần thiết và hỗ trợ tính toán giá trị hữu ích một cách hiệu quả, từ đó giảm đáng kể chi phí xử lý.

Sau đó, nhiều phương pháp và chiến lược hiệu quả được phát triển để thu hẹp không gian tìm kiếm, tối ưu hoá hiệu năng. Điển hình là thuật toán FHM [20] sử dụng chiến lược cắt tỉa có tên là EUCP để loại bỏ sớm các mẫu không tiềm năng, giảm số lượng các thao tác kết hợp để tạo mẫu mới. Thuật toán EFIM [96] giới thiệu hai giới hạn trên là local utility và sub-tree utility, cùng với hai kỹ thuật là high-utility database projection và high-utility transaction merging để giảm chi phí quét CSDL và nâng cao hiệu suất khai thác. Các nghiên cứu gần đây tập trung vào việc phát triển các cấu trúc dựa trên danh sách hiệu quả và tăng tốc quá trình giao tạo mẫu mới như thuật toán HMiner [33] sử dụng cấu trúc compact utility list và cơ chế virtual hyperlink để lưu trữ thông tin của các mẫu; thuật toán ULB-Miner với cấu trúc utility-list buffer [17]; MEFIM/iMEFIM là cải tiến của thuật toán EFIM sử dụng cấu trúc P-set [56]; thuật toán UBP-Miner sử dụng cấu trúc utility bit partition [83]; và thuật toán FOTH giới thiệu cấu trúc IndexSet cùng phương pháp khai thác mà không cần phép nối [85]. Các phương pháp này đều cho thấy sự hiệu quả về thời gian khai thác và bộ nhớ sử dụng trên các bộ dữ liệu dày và thưa. Bên cạnh đó, khái niệm hữu ích cũng được mở rộng và áp dụng sang nhiều bài toán khác trong khai thác dữ liệu như khai thác mẫu hữu ích cao trung bình [32, 40], khai thác mẫu tuần tự hữu ích cao [16], khai thác mẫu skyline [28] và khai thác mẫu hiệu suất cao [46, 87]..., góp phần mở rộng phạm vi ứng dụng của hướng tiếp cận này.

1.1.2 Bài toán khai thác mẫu trọng số hữu ích phổ biến

Hướng nghiên cứu thứ hai do Khan và đồng sự [31] đề xuất đó là khai thác luật kết hợp trọng số hữu ích trên CSDL định lượng. Bài toán được chia làm hai giai đoạn. Giai đoạn đầu tiên tìm ra tất cả các mẫu trọng số hữu ích phổ biến và giai đoạn thứ hai tạo ra các luật kết hợp từ các mẫu này. Cách tiếp cận này xuất phát từ hạn chế của bài toán khai thác HUP, vốn chỉ tập trung vào giá trị hữu ích mà chưa xét đến tần suất xuất hiện của mẫu. Trong nhiều ứng dụng thực tế, đặc biệt trong lĩnh vực kinh doanh, việc tìm kiếm các sản phẩm vừa mang lại giá trị cao vừa có mức tiêu thụ lớn là hết sức quan trọng để hỗ trợ ra quyết định. Hơn nữa, như phân tích ở trên, không gian tìm kiếm của bài toán khai thác HUP là rất lớn, đòi hỏi nhiều tài nguyên để khai

thác. Do đó, tác giả đã đề xuất hai độ đo mới là trọng số hữu ích giao dịch (*twu*) và độ hỗ trợ trọng số hữu ích (*wus*), kết hợp đồng thời hai yếu tố là tần suất xuất hiện và tầm quan trọng của các mẫu. Điểm đặc biệt của phương pháp này là các độ đo thoả mãn tính chất bao đóng giảm, từ đó cho phép áp dụng hiệu quả các chiến lược cắt tỉa nhằm thu hẹp không gian tìm kiếm. Tác giả cũng đề xuất một thuật toán dựa trên Apriori để giải quyết bài toán. Tuy nhiên, thuật toán này phải quét CSDL nhiều lần và sinh ra số lượng lớn các tập ứng viên.

Sau đó, Vo và đồng sự [75] đề xuất cấu trúc MWIT-tree kết hợp với Tidset để khai thác FWUP chỉ với một lần quét CSDL. Nguyen và đồng sự [25] giới thiệu cấu trúc MBiS, trong đó thông tin được lưu trữ dưới dạng các đoạn con liên tiếp trong Tidset nhằm tối ưu hóa bộ nhớ và tăng hiệu quả khai thác. Tác giả cũng đã giới thiệu một phương pháp hiệu quả để xây dựng các cấu trúc MBiS của các mẫu mới. Tiếp theo, Bui và đồng sự [5] đề xuất cấu trúc WUN-set cùng với thuật toán WUN-Miner. Đây là một phương pháp lai, kết hợp cách tiếp cận dựa trên cây và dựa trên danh sách. Cụ thể, CSDL được thuật toán nén vào cấu trúc cây có tên là WUN-tree, từ đó sinh ra các WUN-set để tiến hành khai thác. Gần đây, Nguyen và đồng sự [54] đã phát triển thuật toán FWUP-SWUNL sử dụng cấu trúc SWUN-list, cùng các định lý tăng tốc quá trình khai thác. Cấu trúc SWUN-list cũng được tạo từ WUN-tree giống như WUN-set. Tuy nhiên, SWUN-list cho khả năng nén dữ liệu tốt hơn, qua đó giảm đáng kể thời gian xử lý và mức sử dụng bộ nhớ. Kết quả thực nghiệm cho thấy FWUP-SWUNL vượt trội hơn các thuật toán đã được đề xuất để khai thác FWUP.

Bên cạnh việc cải thiện hiệu năng, FWUP còn được mở rộng sang nhiều ứng dụng và mô hình dữ liệu khác nhau. Phương pháp này đã được áp dụng trong bài toán phân cụm văn bản [67] và cho kết quả chính xác hơn so với phương pháp sử dụng FP truyền thống. Ngoài ra, Nguyen và đồng sự [55] đã đề xuất các thuật toán MINE_FWUIS và FAST_MINE_FWUIS để khai thác FWUP trên CSDL định lượng phân cấp, trong đó các mục được tổ chức theo cấu trúc phân cấp nhằm biểu diễn mối quan hệ tổng quát hóa giữa các mục.

Các nghiên cứu hiện nay đã đạt được nhiều kết quả quan trọng trong khai thác mẫu từ CSDL định lượng và được mô tả ở Bảng 1.1. Tuy nhiên, hầu hết các phương pháp đều giả định trọng số của các mục là cố định trong suốt quá trình khai thác. Do đó, bài toán khai thác FWUP trên CSDL định lượng động, trong đó trọng số các mục có thể thay đổi, vẫn chưa được nghiên cứu đầy đủ.

Bảng 1.1. Các nghiên cứu về khai thác mẫu trên CSDL định lượng

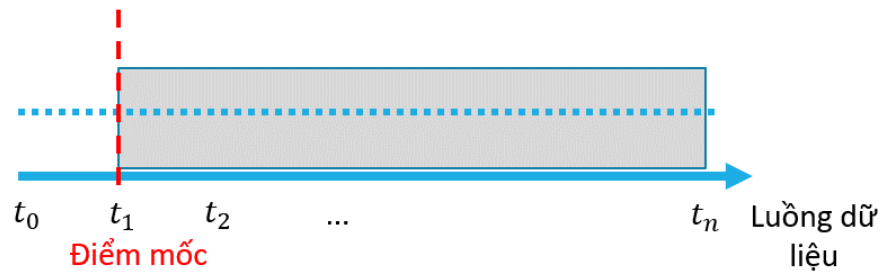
Thuật toán (năm)	Tác giả	Loại mẫu	Mô tả
Two-Phase [47] (2005)	Liu Y. và đồng sự	HUP	Thuật toán dựa trên Apriori và đề xuất khái niệm TWU làm giới hạn trên để cắt tia không gian tìm kiếm, nhưng phải quét CSDL nhiều lần và sinh ra lượng lớn tập ứng viên.
Apriori- FWUI [31] (2008)	Khan M.S. và đồng sự	FWUP	Đề xuất mô hình FWUP kết hợp cả tần suất và trọng số thông qua hai độ đo mới (<i>twu</i> và <i>wus</i>), song thuật toán gặp hạn chế phải quét CSDL nhiều lần do dựa trên Apriori.
UP-Growth [71] (2010)	Tseng V.S. và đồng sự	HUP	Lấy cảm hứng từ FP-Growth giúp khai thác HUP không cần sinh ứng viên đi kèm các chiến lược cắt tia hiệu quả, nhưng phải duy trì cấu trúc cây và có thể đệ quy rất sâu để khai thác.
HUI-Miner [45] (2012)	Liu M. và Qu J.	HUP	Đề xuất cấu trúc utility-list cho phép khai thác HUP chỉ với một lần quét dữ liệu mà không cần sinh ứng viên, tuy nhiên thao tác kết hợp utility-list để tạo mẫu mới còn tốn nhiều chi phí tính toán và bộ nhớ.
MWIT- FWUI [75] (2012)	Vo B. và đồng sự	FWUP	Đề xuất cấu trúc MWIT-tree kết hợp với Tidset giúp khai thác FWUP chỉ bằng một lần quét CSDL, nhưng lại kém hiệu quả trên dữ liệu dày và lớn.
FHM [20] (2014)	Fournier- Viger P. và đồng sự	HUP	Tích hợp chiến lược cắt tia EUCP dựa trên ma trận đồng xuất hiện nhằm giảm thiểu tối đa các phép kết hợp danh sách không tiềm năng, nhưng vẫn phụ thuộc lớn vào đặc trưng dữ liệu và tốn bộ nhớ lưu trữ ma trận ban đầu.

MBiS-FWUI [25] (2015)	Ham N.D và đồng sự	FWUP	Giới thiệu cấu trúc MBiS lưu trữ thông tin dưới dạng các đoạn con liên tiếp trong Tidset nhằm tối ưu bộ nhớ và đẩy nhanh tốc độ xây dựng mẫu mới, nhưng chi phí tính toán để duy trì các phân đoạn này vẫn còn tương đối cao.
WUN-Miner [5] (2016)	Bui H. và đồng sự	FWUP	Nén CSDL vào cấu trúc WUN-tree và sinh ra WUN-set để khai thác, tuy nhiên phải mất hai lần quét dữ liệu để xây dựng và kém hiệu quả trên các bộ dữ liệu rất thưa.
EFIM [96] (2017)	Zida S. và đồng sự	HUP	Đề xuất hai giới hạn trên mới, kết hợp với kỹ thuật chiếu CSDL và gộp giao dịch để giảm chi phí quét dữ liệu, tuy nhiên vẫn phụ thuộc nhiều vào cơ chế đệ quy sâu và bộ nhớ cục bộ.
ULB-Miner [17] (2018)	Duong Q.-H. và đồng sự	HUP	Đề xuất cấu trúc utility-list buffer giúp tối ưu hóa và tái sử dụng bộ nhớ RAM hiệu quả, nhưng tốc độ khai thác còn phụ thuộc vào việc cấu hình kích thước vùng đệm.
MEFIM/ iMEFIM [56] (2019)	Nguyen L.T.T. và đồng sự	HUP	Là cải tiến của EFIM, sử dụng cấu trúc P-set nhằm giảm số lần quét CSDL, tăng tốc độ khai thác.
UBP-Miner [83] (2022)	Wu P. và đồng sự	HUP	Sử dụng cấu trúc Utility Bit Partition và cơ chế BEO nhằm tăng tốc quá trình xây dựng utility-list bằng phép toán bit, tuy nhiên quá trình tiền xử lý tỏ ra phức tạp.
FWUP-SWUNL [54] (2023)	Nguyen H. và đồng sự	FWUP	Phát triển cấu trúc SWUN-list cải tiến từ WUN-tree cho khả năng nén dữ liệu vượt trội và tăng tốc khai thác so với cấu trúc WUN-set.
FOTH [85] (2024)	Yan Y. và đồng sự	HUP	Giới thiệu cấu trúc IndexSet cho phép khai thác trực tiếp HUP mà không cần sử dụng các phép toán nối phức tạp, tuy thuật toán đặc biệt hiệu quả trên dữ liệu dày nhưng hiệu năng chưa được tối ưu trên dữ liệu thưa thớt.

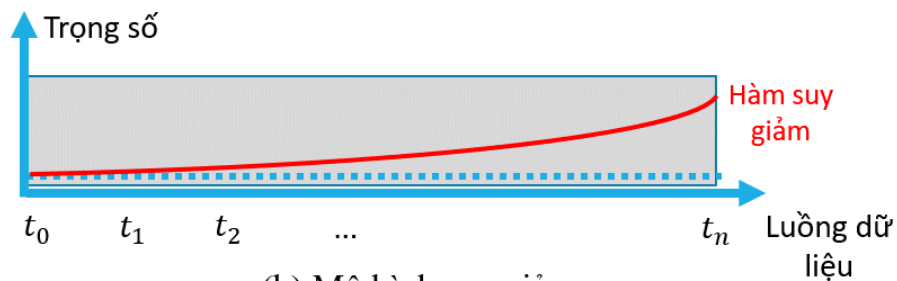
1.2 Bài toán khai thác mẫu trên luồng dữ liệu

Với sự gia tăng nhanh chóng về khối lượng và tốc độ sinh dữ liệu, bên cạnh các phương pháp khai thác trên CSDL tĩnh truyền thống, vấn đề khai thác mẫu trên luồng dữ liệu đang được quan tâm nghiên cứu. Luồng dữ liệu có các đặc trưng điển

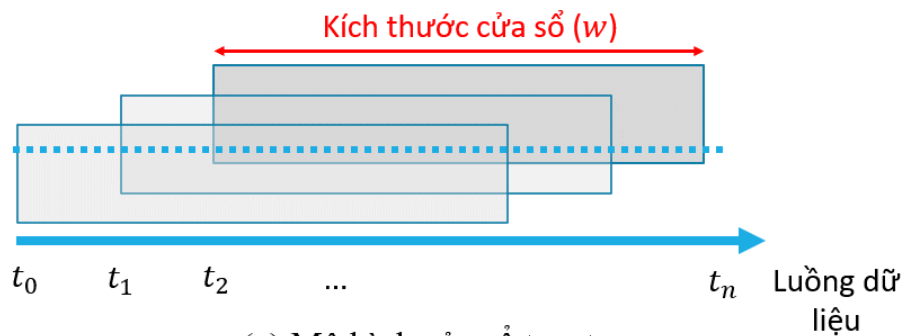
hình như chiều dài vô hạn, tốc độ cao, lưu lượng lớn và liên tục, thứ tự theo thời gian và khả năng biến đổi nhanh chóng. Những đặc điểm này khiến các phương pháp khai thác truyền thống trở nên không phù hợp, bởi việc lưu trữ và xử lý toàn bộ các luồng dữ liệu đòi hỏi tài nguyên lưu trữ và năng lực tính toán rất lớn. Để giải quyết bài toán này, nhiều nghiên cứu đã được đề xuất, trong đó tập trung vào ba mô hình chính: mô hình điểm mốc (landmark model) [66, 97], mô hình suy giảm (damped model) [3, 52, 89] và mô hình cửa sổ trượt (sliding window model) [4, 38, 91]. Mỗi mô hình đều có những ưu điểm và hạn chế riêng, và việc lựa chọn mô hình khai thác phù hợp phụ thuộc vào đặc tính của dữ liệu cũng như yêu cầu của ứng dụng.



(a) Mô hình điểm mốc



(b) Mô hình suy giảm



(c) Mô hình cửa sổ trượt

Hình 1.1. Các loại mô hình khai thác mẫu trên luồng dữ liệu.

Trong mô hình điểm mốc, dữ liệu được xem xét từ một thời điểm khởi đầu xác định, gọi là điểm mốc (landmark), cho đến thời điểm hiện tại (Hình 1.1a). Điểm mốc có thể cố định, thay đổi hoặc được thiết lập lại trong quá trình khai thác. Đặc điểm chính của mô hình này là tất cả các giao dịch trong khoảng thời gian được xét đều có mức độ quan trọng như nhau, nên mô hình điểm mốc phù hợp với các tình huống mà dữ liệu mới và cũ đều có ý nghĩa giống nhau. Thuật toán tiêu biểu có tên là Lossy Counting, được đề xuất bởi Manku và Motwani [49]. Lossy Counting xử lý luồng dữ liệu theo các xô (buckets), duy trì tần suất và loại bỏ các mẫu không tiềm năng dựa trên sai số ước lượng. Yu và các đồng sự [88] sử dụng chặn Chernoff để xấp xỉ hóa tập FP trong cửa sổ landmark. Zhi-Jun cùng đồng sự [95] sau đó sử dụng cấu trúc dàn gọi là cây tập con phổ biến. Cấu trúc này tổ chức các mẫu vào nhiều lớp tương đương của các mẫu xuất hiện trong cùng một giao dịch. Trong từng lớp tương đương, chỉ các FP đại diện cho cả hai biên được giữ lại (các mẫu tối thiểu và tối đa trong mỗi lớp) để tránh việc lưu tất cả mẫu trong bộ nhớ. Thuật toán xấp xỉ MAHUSP [97] giải quyết bài toán khai thác mẫu hữu ích cao tuần tự trên toàn bộ luồng dữ liệu. MAHUSP nổi bật với cơ chế thích nghi bộ nhớ, tự động loại bỏ các mẫu ít triển vọng nhất khỏi cấu trúc MAS-Tree khi bộ nhớ hệ thống đạt ngưỡng giới hạn.

Khác với mô hình điểm mốc, mô hình suy giảm gán trọng số cho các giao dịch dựa trên thứ tự xuất hiện của chúng trên luồng dữ liệu (Hình 1.1b). Trong mô hình này, các giao dịch cũ hơn sẽ ít có ảnh hưởng hơn nên tầm quan trọng và trọng số sẽ giảm dần mỗi khi có dữ liệu mới được thêm vào. Người dùng sẽ xác định một giá trị gọi là hệ số suy giảm (decay factor) nằm trong khoảng từ 0 đến 1 để điều chỉnh mức độ giảm này. Mỗi khi có giao dịch mới được thêm vào, giá trị trọng số của các giao dịch trước đó sẽ bị giảm theo cấp số nhân theo thời gian thông qua hệ số suy giảm đã chọn. Chang và Lee [9] đề xuất thuật toán estDec sử dụng cơ chế cập nhật trọng số liên tục để giảm tầm ảnh hưởng của các giao dịch mà không cần quét lại toàn bộ dữ liệu. Một biến thể khác là estMax [80], được xây dựng dựa trên estDec để khai thác các mẫu phổ biến tối đại trên luồng dữ liệu. Thuật toán TFUHS-Stream [29] kết hợp mô hình suy giảm với cấu trúc hyper-structure để xử lý dữ liệu không chắc chắn.

TFUHS-Stream sử dụng cấu trúc UH-struct và duy trì chúng trong cấu trúc IS-tree theo phương pháp hướng đến phát hiện dương tính giả. Yun và đồng sự [89] đã phát triển thuật toán MPM để khai thác mẫu hữu ích trung bình cao trên luồng dữ liệu sử dụng mô hình suy giảm. Tác giả đã đề xuất các cấu trúc dữ liệu mới là DAT (cây tiện ích trung bình giảm dần) và TUL (danh sách tiện ích giao dịch), và các chiến lược cắt tia mới dựa trên các giá trị tiện ích trung bình giảm dần được lưu trữ trong các cấu trúc dữ liệu được đề xuất. Mặc dù mô hình suy giảm đã thể hiện được sự khác biệt giữa các giao dịch, tuy nhiên lại đòi hỏi nhiều tài nguyên xử lý vì phải lưu trữ và quản lý toàn bộ lịch sử dữ liệu.

Mô hình thứ ba được sử dụng phổ biến trong khai thác mẫu trên luồng dữ liệu là mô hình cửa sổ trượt. Trong mô hình này, luồng dữ liệu được chia thành nhiều pane dữ liệu có kích thước bằng nhau. Một cửa sổ có kích thước cố định, bao gồm nhiều pane, chỉ lưu trữ và quản lý các dữ liệu mới nhất. Khi có pane dữ liệu mới được thêm vào, pane cũ nhất sẽ bị loại bỏ để duy trì kích thước cửa sổ không đổi (Hình 1.1c). Nhờ đó, quá trình khai thác tập trung vào các dữ liệu mới nhất, phù hợp với nhiều ứng dụng thực tiễn. Nhiều thuật toán đã được đề xuất để khai thác mẫu trên luồng dữ liệu sử dụng mô hình cửa sổ trượt và được mô tả ở Bảng 1.2. Thuật toán MFI-TransSW và MFI-TimeSW [42] sử dụng phương pháp dựa trên cửa sổ trượt để tìm FP trên luồng dữ liệu với kỹ thuật chuỗi bit. Dữ liệu được lưu trữ, cập nhật thông qua các biểu diễn chuỗi bit và sử dụng phương pháp tìm kiếm theo cấp độ để tiến hành khai thác. Tanbeer và đồng sự [65] đã phát triển cấu trúc cây có tên là CPS-tree có khả năng nhanh chóng cập nhật các giao dịch mới nhất của khung cửa sổ và áp dụng thuật toán FP-Growth trên cấu trúc cây này. SHU-Growth [63] cũng là một thuật toán dựa trên FP-growth để khai thác HUP trên luồng dữ liệu sử dụng mô hình cửa sổ trượt. Thuật toán SHU-Growth xây dựng cấu trúc dữ liệu cây toàn cục, kết hợp với nhiều kỹ thuật để ước tính giá trị twu của mẫu. Sau đó, thuật toán SHUPM [90] sử dụng cấu trúc SHUP-List để duy trì thông tin hữu ích cho các pane dữ liệu gần đây trong cửa sổ trượt. SHUPM đề xuất các kỹ thuật cắt tia mới dựa trên việc kết hợp các danh sách để giảm thiểu không gian tìm kiếm mà không cần tạo ra các mẫu ứng viên.

Gần đây, Li và đồng sự [41] đề xuất thuật toán CHUPM-Stream khai thác HUP trên luồng dữ liệu theo mô hình cửa sổ trượt. Thuật toán sử dụng cấu trúc CU-list, kết hợp chiến lược hợp nhất giao dịch trùng lặp để giảm chi phí bộ nhớ, chiến lược duy trì *twu* để hạn chế việc quét lại cửa sổ và kỹ thuật cắt tia LA-prune để thu hẹp không gian tìm kiếm. Kết quả thực nghiệm cho thấy thuật toán cải thiện hiệu quả về thời gian thực thi và bộ nhớ, đặc biệt trên các bộ dữ liệu dày đặc. Mô hình cửa sổ trượt cho phép trích xuất nhanh chóng các thông tin hữu ích từ dữ liệu mới nhất, tuy nhiên việc xác định kích thước cửa sổ và kích thước pane phù hợp là một thách thức cho người sử dụng.

Các thuật toán khai thác mẫu trên luồng dữ liệu đã được nghiên cứu áp dụng cho mẫu phổ biến [8, 84] và các biến thể của nó như mẫu hữu ích cao [27, 52, 90], mẫu có thể xóa [3, 36, 91],... và đạt được thành công nhất định. Nhìn chung, các thuật toán này đều tập trung giải quyết các vấn đề chính sau. Thứ nhất, các thuật toán được thiết kế để hạn chế tối đa việc quét dữ liệu xuống còn một lần duy nhất do khối lượng lớn và tốc độ cao của luồng dữ liệu. Thứ hai, các cấu trúc dữ liệu mới được phát triển có khả năng lưu trữ, quản lý, cập nhật và tiến hành khai thác mẫu một cách hiệu quả khi có dữ liệu mới xuất hiện. Thứ ba, các thuật toán phải đảm bảo được độ tin cậy, tính chính xác và tối ưu tài nguyên sử dụng. Tuy nhiên, cho đến thời điểm hiện tại, vẫn chưa có nghiên cứu nào về khai thác FWUP trên luồng dữ liệu.

Bảng 1.2. Các nghiên cứu về khai thác mẫu trên luồng dữ liệu sử dụng mô hình cửa sổ trượt

Thuật toán (năm)	Tác giả	Loại mẫu	Mô tả
MFI-TransSW [42] (2009)	Li H.-F. và Lee S.- Y.	FP	Sử dụng kỹ thuật chuỗi bit kết hợp phương pháp tìm kiếm theo cấp độ để cập nhật, lưu trữ và tìm FP trong cửa sổ trượt. Tuy nhiên, hiệu năng tính toán của chuỗi bit sẽ bị giảm sút nghiêm trọng nếu kích thước của cửa sổ lớn.
CPS-tree [65] (2009)	Tanbeer S.K. và	FP	Phát triển cấu trúc CPS-tree có khả năng nhanh chóng chèn và loại bỏ các giao dịch để

	đồng sự		cập nhật khung cửa sổ hiện tại, sau đó áp dụng phương pháp FP-Growth để khai thác.
SHU-Growth [63] (2016)	Ryang H. và Yun U.	HUP	Đề xuất cấu trúc SHU-Tree kết hợp kỹ thuật RGE và RLE giúp khai thác đầy đủ HUP mà không bị mất mát dữ liệu. Tuy nhiên, thuật toán vẫn gặp các hạn chế của FP-Growth.
SHUPM [90] (2017)	Yun U. và đồng sự	HUP	Sử dụng cấu trúc SHUP-List để duy trì thông tin các pane gần đây nhất và đề xuất một chiến lược cắt tia để giảm không gian tìm kiếm.
SPPC [36] (2019)	Le T. và đồng sự	Mẫu có thể xóa	Giới thiệu cấu trúc cây mới có tên là SPPC-tree chuyên dụng để cập nhật và khai thác các mẫu có thể xóa trong môi trường luồng dữ liệu trượt, tuy nhiên chi phí bộ nhớ để lưu trữ cây còn lớn.
SWEclat [84] (2020)	Xiao W. và Hu J.	FP	Sử dụng phương pháp Eclat kết hợp mô hình Spark Streaming để xử lý song song dữ liệu theo từng pane trong luồng. Tuy nhiên, phương pháp vẫn có hạn chế về chi phí bộ nhớ khi tidset lớn và hiệu quả phụ thuộc vào khả năng phân phối dữ liệu cũng như cấu hình hệ thống Spark.
HUPM_Stream [27] (2023)	Han M. và đồng sự	HUP	Đề xuất cấu trúc Ext-list và một chiến lược cắt tia IRS để giảm số lượng phép toán kết hợp; tập kết quả được duy trì dựa trên cấu trúc bảng băm HRS để tránh việc tạo lại trong quá trình trượt của cửa sổ, tuy nhiên lại yêu cầu chi phí bộ nhớ lớn.
CHUPM-Stream [41] (2025)	Li G. và đồng sự	HUP	Sử dụng cấu trúc CU-list để khai thác HUP trên luồng dữ liệu, kết hợp DTM để nén giao dịch trùng lặp, TMS để cập nhật <i>twu</i> hiệu quả và LA-prune để giảm không gian tìm kiếm, qua đó cải thiện thời gian thực thi và bộ nhớ.

1.3 Các biến thể của bài toán khai thác mẫu

1.3.1 Bài toán khai thác mẫu đóng

Trong bài toán khai thác mẫu, số lượng mẫu thu được thường rất lớn. Điều này dẫn đến khi áp dụng vào các ứng dụng cho ra tập kết quả lớn, trong đó chứa nhiều

kết quả dư thừa, không cần thiết cho người sử dụng. Xuất phát từ thực tế đó, bài toán khai thác FCP được Pasquier và đồng sự [59] giới thiệu đầu tiên vào năm 1999. Một mẫu phổ biến được gọi là FCP nếu nó không có tập cha nào có cùng độ phổ biến. Số lượng mẫu đóng ít hơn nhiều so với các mẫu thông thường, đồng thời vẫn cho phép tái tạo lại toàn bộ tập mẫu mà không làm mất thông tin. Mẫu đóng được sử dụng trong các bài toán khai thác dữ liệu để đưa ra các kết quả không dư thừa, có ý nghĩa hơn với người sử dụng. Tác giả cũng đề xuất thuật toán A-Close dựa trên phương pháp Apriori. Tuy nhiên, thuật toán này tỏ ra kém hiệu quả vì phải tốn thời gian quét CSDL nhiều lần để tính độ hỗ trợ và sinh ra nhiều ứng viên.

Để khắc phục hạn chế này, các phương pháp không sinh ứng viên dựa trên FP-tree đã được đề xuất, tiêu biểu là thuật toán CLOSET [60] và CLOSET+ [79]. CLOSET nén đường dẫn tiên tố đơn để nhanh chóng xác định FCP, chiếu CSDL dựa trên phân vùng để có thể mở rộng khai thác trong các CSDL lớn. Thuật toán CLOSET+ là phiên bản nâng cấp của CLOSET sử dụng phương pháp hybrid tree projection. CSDL điều kiện được chiếu bằng hai cách: bottom-up physical tree-projection cho các bộ dữ liệu dày và top-down pseudo tree-projection cho các bộ dữ liệu thưa. Cùng với đó, CLOSET+ cũng sử dụng kỹ thuật item skipping để loại bỏ nhanh các mẫu không phù hợp, và subset checking để kiểm tra nhanh các mẫu mới có phải là FCP hay không bằng cách sử dụng bảng băm hai cấp lưu trữ cây tiên tố kết quả. Sau đó, thuật toán FPClose [23] được đề xuất để tối ưu cách tiếp cận dựa trên FP-tree và phương pháp lưu trữ FCP trong cấu trúc CFI-tree để nhanh chóng kiểm tra tính đóng của một FP mới.

Thuật toán CHARM [92], dCHARM [51] chuyển CSDL từ chiều ngang sang chiều dọc, tạo Tidset và Diffset để khai thác FCP. CHARM tìm kiếm FCP thông qua cấu trúc IT-tree và sử dụng kỹ thuật bỏ qua nhiều cấp trong quá trình tìm kiếm. CHARM cũng sử dụng cách tiếp cận bảng băm để lưu trữ và loại bỏ các mẫu không phải là FCP. Cũng dựa trên phương pháp tiếp cận theo chiều dọc, thuật toán DCI_Closed [48] và DCI_Plus [64] được đề xuất với các kỹ thuật khai thác mới. Các thuật toán này không sử dụng kỹ thuật subset checking nên các FCP không cần phải

lưu trữ trong bộ nhớ chính. Thay vào đó, chúng sử dụng phương pháp closure climbing để sinh trực tiếp các FCP. Với việc duyệt theo chiều sâu và thực hiện các thao tác kiểm tra mẫu đóng, tính toán độ hỗ trợ và phát hiện trùng lặp thông qua phép giao các danh sách nên thời gian khai thác và bộ nhớ sử dụng được giảm đáng kể.

Sau đó, Le và Vo đề xuất thuật toán NAFCP [35] sử dụng phương pháp lai để khai thác FCP. NAFCP sử dụng cấu trúc N-list [14] kết hợp với kỹ thuật để kiểm tra nhanh N-list của hai mẫu có phải là tập con hay không. Cấu trúc N-list tỏ ra vượt trội do khả năng nén dữ liệu tốt, đặc biệt là trên các CSDL dày. Gần đây, Aryabarzan và đồng sự đã đề xuất thuật toán NEclatClosed [2], dựa trên hướng tiếp cận của thuật toán Eclat [93] để khai thác FCP. NEclatClosed đưa ra khái niệm Negation Tidset (NegTset), cách tạo NegTset cho 2-pattern, k -pattern và cách tính độ hỗ trợ cho các mẫu thông qua NegTset. Tác giả đề xuất khái niệm partial closure kết hợp với subsumption để cắt tỉa nhanh không gian tìm kiếm. NEclatClosed cũng sử dụng bảng băm để lưu trữ FCP và kiểm tra tính đóng của một mẫu mới. Các kết quả thực nghiệm trên các bộ dữ liệu dày và thưa cho thấy, thuật toán NEclatClosed hiệu quả hơn các thuật toán trước đó cả về thời gian khai thác và bộ nhớ sử dụng.

Bài toán khai thác FCP cũng được mở rộng sang nhiều biến thể khác. Trong khai thác mẫu phổ biến đóng có trọng số, Vo [74] đề xuất hai thuật toán có tên là WIT-FWCI và WIT-FWCI-Diff sử dụng cấu trúc Tidset và Diffset. Sau đó, Bui và đồng sự [7] phát triển thuật toán NFWCI dựa trên cấu trúc WN-list cùng một chiến lược cắt tỉa để loại bỏ các ứng viên không phù hợp. Thuật toán NFWCI thể hiện sự vượt trội hoàn toàn so với thuật toán WIT-FWCI-Diff và thuật toán DiffNFWCI, là phiên bản chỉnh sửa dựa trên cấu trúc DiffNodeset [13].

Đối với khai thác mẫu hữu ích cao đóng, Tseng và đồng sự [69] giới thiệu đầu tiên bài toán này cùng ba thuật toán có tên là AprioriCH, AprioriHC-D và CHUD. Thuật toán AprioriCH và AprioriHC-D được phát triển dựa trên Apriori, còn CHUD lấy cảm hứng từ thuật toán DCI-Closed cùng các chiến lược hiệu quả để giảm số lượng ứng viên được tạo ra. Tác giả cũng giới thiệu phương pháp có tên là DAHU để

khôi phục lại tất cả các HUP mà không cần phải duyệt CSDL ban đầu. Tiếp theo, Wu và đồng sự [82] đề xuất thuật toán CHUI-Miner sử dụng cấu trúc mới có tên là EU-List để lưu trữ các thông tin cần thiết cho quá trình tính toán độ hữu ích và áp dụng phương pháp chia để trị để khai thác đầy đủ HUP đóng mà không cần tạo ứng viên. Fournier-Viger và đồng sự [21] đã phát triển thuật toán EFIM-Closed, là mở rộng của thuật toán EFIM, để khai thác HUP đóng. EFIM-Closed giới thiệu ba chiến lược có tên là closure jumping, forward closure checking và backward closure checking để loại bỏ các mẫu không phải là HUP đóng. Sau đó, Dam và đồng sự [11] đề xuất thuật toán CLS-Miner sử dụng cấu trúc utility-list kết hợp với cấu trúc EUCS cùng các chiến lược cắt tĩa hiệu quả để khai thác HUP đóng. Thuật toán HMiner-Closed [57] lấy cảm hứng từ thuật toán HMiner [33] với các cải tiến về cấu trúc lưu trữ và tối ưu kỹ thuật backward và forward checking giúp giảm đáng kể không gian tìm kiếm. Duong và đồng sự [15] đã đề xuất thuật toán C-HUIM sử dụng chiến lược SPWUB là một giới hạn trên chặt hơn các giới hạn truyền thống. Hai chiến lược cắt tĩa liên quan đến việc kiểm tra mối quan hệ của các mẫu cũng được đề xuất để tăng tốc quá trình khai thác. Gần đây, Mofid và đồng sự [50] đề xuất thuật toán MMC để khai thác HUP đóng, sử dụng cấu trúc ma trận lưu trữ độ hữu ích kết hợp kỹ thuật hợp nhất mục và cắt tĩa không gian tìm kiếm. Thuật toán chỉ yêu cầu một lần quét CSDL, hỗ trợ ngưỡng hữu ích tối thiểu riêng cho từng mục và cải thiện hiệu quả thời gian, bộ nhớ lên đến 3 lần so với các thuật toán được so sánh. Ngoài ra, khai thác mẫu đóng còn được áp dụng vào nhiều môi trường dữ liệu khác nhau như CSDL phân cấp [58], luồng dữ liệu [43, 44]. Mặc dù các dạng mẫu đóng đã được nghiên cứu rộng rãi đối với FP, mẫu có trọng số và HUP (Bảng 1.3), tuy nhiên chưa có công trình nào nghiên cứu bài toán khai thác mẫu trọng số hữu ích phổ biến đóng từ CSDL định lượng động.

Bảng 1.3. Các nghiên cứu về khai thác mẫu đóng

Thuật toán (năm)	Tác giả	Loại mẫu	Mô tả
NAFCP [35] (2015)	Le T. và Vo B.	FCP	Sử dụng cấu trúc N-list với kỹ thuật kiểm tra nhanh quan hệ tập con giữa các mẫu

			giúp nhanh chóng loại bỏ các mẫu không thoả mãn trên CSDL dày, tuy nhiên, hiệu năng trên CSDL thưa chưa thực sự rõ rệt.
EFIM-Closed [21] (2016)	Fournier-Viger P. và đồng sự	HUP đóng	Mở rộng từ EFIM, đề xuất ba kỹ thuật kiểm tra tính đóng mạnh mẽ nhằm loại bỏ sớm các mẫu không đóng.
WIT-FWCI [74] (2017)	Vo B.	FCP có trọng số	Sử dụng cấu trúc lưu trữ chiều dọc dạng Tidset để kiểm tra tính đóng, tuy nhiên, hiệu năng của thuật toán phụ thuộc vào đặc điểm dữ liệu và kém hiệu quả trên các CSDL lớn.
CLS-Miner [11] (2019)	Dam T.-L. và đồng sự	HUP đóng	Sử dụng cấu trúc utility-list kết hợp với các chiến lược cắt tỉa để khai thác mẫu đóng. Hạn chế là chi phí lưu trữ EUCS gây tốn kém tài nguyên trong giai đoạn tiền xử lý dữ liệu.
NEclatClosed [2] (2021)	Aryabarzan N. và Minaei-Bidgoli B.	FCP	Đề xuất cấu trúc NegTset cùng chiến lược partial closure và phương pháp lưu trữ mẫu đóng hiệu quả giúp giảm cả thời gian và bộ nhớ trên các bộ dữ liệu.
NFWCI [7] (2021)	Bui H. và đồng sự	FCP có trọng số	Sử dụng cấu trúc WN-list cùng một chiến lược cắt tỉa để loại bỏ sớm các ứng viên dư thừa, tuy nhiên cần hai lần quét CSDL và kém hiệu quả trên các bộ dữ liệu thưa thớt.
C-HUIM [15] (2022)	Duong H. và đồng sự	HUP đóng	Đề xuất cấu trúc MPUN-list và chiến lược SPWUB để tạo ra giới hạn trên chặt hơn, kết hợp với hai chiến lược kiểm tra mối quan hệ mẫu để tăng tốc quá trình khai thác.
MMC [50] (2024)	Mofid A.H. và đồng sự	HUP đóng	Sử dụng cấu trúc ma trận kết hợp kỹ thuật hợp nhất mục và hỗ trợ ngưỡng hữu ích riêng cho từng mục với chỉ một lần quét dữ liệu, giúp cải thiện thời gian thực thi và bộ nhớ.

1.3.2 Bài toán khai thác top-k mẫu

Các bài toán khai thác mẫu đã được giới thiệu ở trên, mặc dù tỏ ra hiệu quả, nhưng lại đặt ra một thách thức thực tiễn trong việc xác định ngưỡng tối thiểu phù

hợp là không hề đơn giản đối với người dùng. Bởi vì họ thường không biết được cài đặt giá trị ngưỡng như thế nào sẽ đáp ứng tốt nhất cho nhu cầu của mình. Để khắc phục hạn chế này, các nghiên cứu về bài toán khai thác top- k mẫu đã được tập trung phát triển với nhiều phương pháp được tiến hành trên nhiều loại mẫu khác nhau và được mô tả trong Bảng 1.4. Trong khai thác top- k FP, Chuang và đồng sự [10] đã đề xuất thuật toán MTK, dựa trên cách tiếp cận Apriori, với kỹ thuật giới hạn số lượng ứng viên được tạo và kiểm tra trong mỗi lần quét CSDL. Điều này cho phép thuật toán hoạt động trong một giới hạn bộ nhớ do người dùng chỉ định. Tiếp theo, thuật toán CRM [61] được giới thiệu để khai thác top- k FP với các kỹ thuật mẫu đóng nghiêm ngặt và mẫu tổng hợp để giảm số lượng tổ hợp cần xem xét trên các đường đơn trong FP-tree. Sau đó, Iqbal và đồng sự đề xuất thuật toán TKFIM [30] dựa trên các lớp tương đương và lý thuyết tập hợp. TKFIM sử dụng cấu trúc Diffset và áp dụng chiến lược tìm kiếm dựa trên lớp để nhanh chóng xác định các mẫu có độ hỗ trợ cao nhất nằm trong top- k FP. Wan và Han mở rộng giải quyết bài toán khai thác top- k FP trên dữ liệu quy mô lớn, không thể lưu trữ hoàn toàn trong bộ nhớ chính với thuật toán PTF [78]. PTF giới thiệu chiến lược phân hoạch dựa trên tiền tố để loại bỏ nhanh các phân hoạch không thể sinh ra top- k FP. Đối với các phân hoạch cần xử lý, PTF sử dụng khai thác theo chiều dọc với cơ chế lưu trữ lại giữa tid-lists, diff-lists, bit-vectors một cách linh hoạt nhằm tối ưu hiệu năng. Các kết quả thực nghiệm cho thấy, thuật toán PTF hiệu quả hơn so với các phương pháp hiện có.

Đối với khai thác top- k HUP, Wu và đồng sự giới thiệu thuật toán TKU [81], mở rộng cấu trúc UP-Tree, sử dụng phương pháp tiếp cận hai giai đoạn: trước hết sinh các ứng viên tiềm năng và sau đó xác định các mẫu top- k HUP thực sự. Tuy đã áp dụng một số chiến lược nâng ngưỡng và cắt tỉa hiệu quả, nhưng TKU gặp phải hạn chế là phải quét CSDL nhiều lần. Để giải quyết vấn đề này, Tseng và đồng sự đề xuất thuật toán TKO [70], sử dụng cấu trúc utility-list, tích hợp việc nâng ngưỡng và các chiến lược cắt tỉa trong một giai đoạn duy nhất. Sau đó, thuật toán kHMC [18] đưa ra hai cải tiến quan trọng là EUCPT và TEP, giúp giảm đáng kể số lượng ứng viên được sinh ra cũng như không gian tìm kiếm. Một đóng góp quan trọng khác đến từ

Krishnamoorthy với thuật toán THUI [34]. Tác giả đã đề xuất một cấu trúc dữ liệu mới gọi là LIU, thường được triển khai dưới dạng ma trận tam giác hoặc bảng băm, để lưu trữ cô đọng thông tin hữu ích của các mẫu liên kề. THUI cũng đề xuất hai chiến lược nâng ngưỡng mới và hiệu quả là LIU-E và LIU-LB, giúp tăng ngưỡng nhanh chóng, đặc biệt trong giai đoạn đầu, qua đó cải thiện đáng kể hiệu suất khai thác. Gần đây, Vu và đồng sự đã giới thiệu thuật toán FTKHUIM [77] và cho thấy hiệu quả vượt trội nhờ sử dụng chiến lược nâng ngưỡng dựa trên giá trị hữu ích giao dịch để thiết lập một ngưỡng tối thiểu ban đầu cao hơn đáng kể so với các chiến lược chỉ dựa trên từng mục đơn lẻ trước đây. Hơn nữa, FTKHUIM sử dụng một cấu trúc lưu trữ toàn cục để duy trì và tái sử dụng các giá trị hữu ích cao nhất tìm được qua tất cả các bước nâng ngưỡng khác nhau giúp tăng tốc độ nâng ngưỡng tổng thể và hiệu quả cắt tỉa. Kết quả thực nghiệm cho thấy sự kết hợp này giúp FTKHUIM đạt hiệu suất vượt trội so với các thuật toán trước đó trên các bộ dữ.

Ngoài ra, khái niệm khai thác top- k cũng đã được mở rộng sang các loại mẫu và dữ liệu khác, chẳng hạn như CSDL không chắc chắn [12, 37], CSDL phân cấp [72], mẫu tuần tự [94], mẫu chiếm dụng cao [53] góp phần mở rộng phạm vi ứng dụng của hướng tiếp cận này.

Bảng 1.4. Các nghiên cứu về khai thác top- k mẫu

Thuật toán (năm)	Tác giả	Loại mẫu	Mô tả
CRM [61] (2014)	Pyun G. và Yun U.	Top- k FP	Tích hợp kỹ thuật mẫu đóng nghiêm ngặt và mẫu tổng hợp nhằm giảm số lượng tổ hợp cần xem xét trên các đường đơn của cấu trúc cây FP-tree. Tuy nhiên, chi phí tính toán để kiểm tra và duy trì tương đối lớn khi áp dụng trên các tập dữ liệu thưa.
TKO [70] (2016)	Tseng V.S. và đồng sự.	Top- k HUP	Sử dụng cấu trúc utility-list cho phép tích hợp đồng thời việc nâng ngưỡng và cắt tỉa trong một giai đoạn duy nhất. Hạn chế của phương pháp này là chi phí bộ nhớ lớn để lưu trữ các thông tin cần thiết.

THUI [34] (2019)	Krishnamoorthy S.	Top- k HUP	Đề xuất cấu trúc dữ liệu LIU giúp lưu trữ cô đọng thông tin, kết hợp hai chiến lược LIU-E và LIU-LB, giúp tăng nhanh ngưỡng hữu ích tối thiểu. Tuy nhiên quá trình tính toán còn mất nhiều thời gian và bỏ qua việc cập nhật ở giai đoạn đầu.
TKFIM [30] (2021)	Iqbal S. và đồng sự	Top- k FP	Sử dụng lớp tương đương dựa trên cấu trúc đọc Diffset nhằm nhanh chóng tìm kiếm các mẫu phổ biến có độ hỗ trợ cao nhất với chỉ một lần quét CSDL, nhưng vẫn chịu hạn chế bởi chi phí bộ nhớ trên các tập dữ liệu lớn hoặc dày đặc.
FTKHUIM [77] (2023)	Vu V.V. và đồng sự	Top- k HUP	Giới thiệu chiến lược nâng ngưỡng dựa trên giá trị hữu ích giao dịch để tạo ngưỡng ban đầu hiệu quả, kết hợp cấu trúc CUL cùng nhiều chiến lược cắt tỉa giúp thuật toán cho hiệu năng vượt trội, tuy nhiên lại sử dụng nhiều bộ nhớ trên dữ liệu thừa thớt.
PTF [78] (2024)	Wan X. và Han X.	Top- k FP	Khai thác hiệu quả top- k FP trên dữ liệu quy mô lớn nhờ chiến lược phân hoạch dựa trên tiền tố kết hợp cơ chế lưu trữ linh hoạt giữa tid-lists, diff-lists và bit-vectors, nhưng lại có độ phức tạp triển khai cao và chi phí lưu trữ trung gian đáng kể.

1.4 Khoảng trống nghiên cứu, thách thức và hướng giải quyết

Trên cơ sở khảo sát các hướng nghiên cứu về khai thác mẫu từ CSDL định lượng, khai thác mẫu trên luồng dữ liệu cũng như các biến thể như mẫu đóng và top- k mẫu, luận án nhận thấy vẫn còn những khoảng trống nghiên cứu trong khai thác mẫu trọng số hữu ích phổ biến đối với một số bài toán có đặc điểm và mục tiêu khác nhau. Hai hướng nghiên cứu được phát triển theo một lộ trình thống nhất nhằm giải quyết các thách thức ngày càng phức tạp trong khai thác FWUP, đặc biệt là trong môi trường dữ liệu biến động.

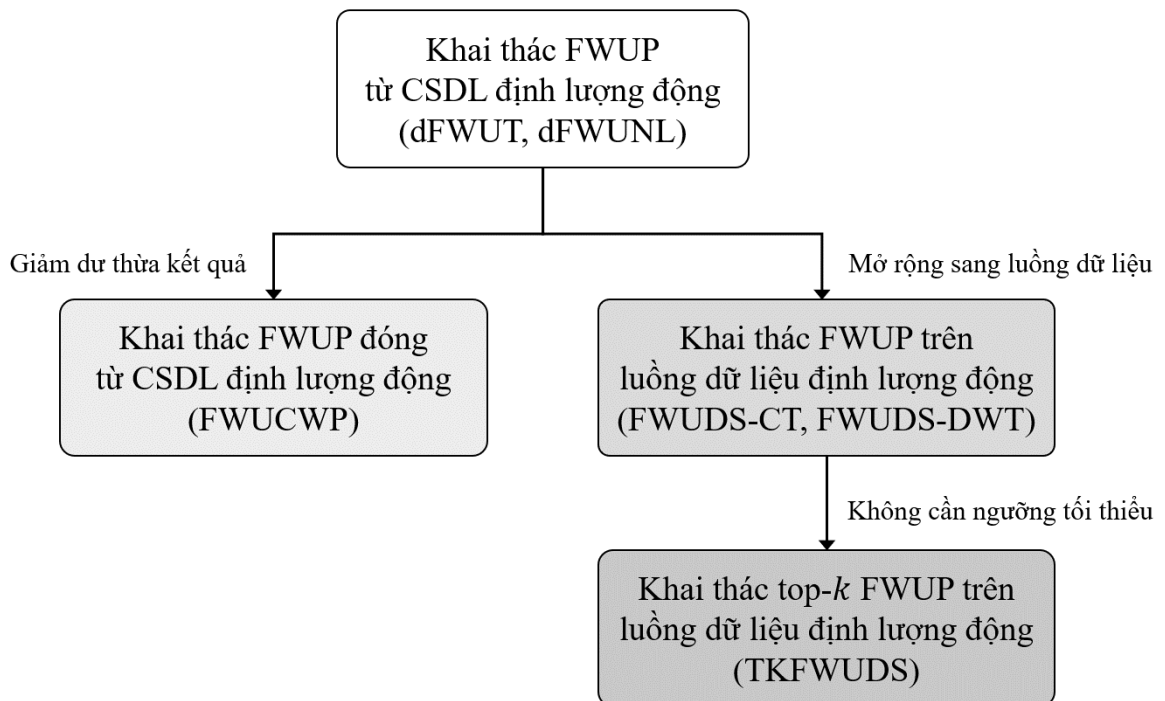
Hướng nghiên cứu đầu tiên là khai thác FWUP trên CSDL định lượng động. Mặc dù các thuật toán hiện hữu đã giải quyết tốt việc kết hợp giữa tần suất và tầm quan trọng, nhưng hầu hết vẫn dựa trên giả định trọng số các mục là cố định. Điều

này bộc lộ hạn chế lớn khi áp dụng vào các ứng dụng thực tế khi giá trị của hàng hóa hoặc tầm quan trọng của các mục dữ liệu luôn biến động theo thời gian. Chính sự thiếu hụt trong việc xử lý trọng số động đã đặt ra yêu cầu cấp thiết về việc xây dựng các phương pháp linh hoạt hơn để phản ánh chính xác giá trị thực của các mẫu dữ liệu. Bên cạnh đó, số lượng FWUP được sinh ra có thể rất lớn, đặt ra nhu cầu khai thác các dạng mẫu cô đọng như mẫu trọng số hữu ích phổ biến đóng. Từ những thách thức này, luận án nghiên cứu các cấu trúc biểu diễn và cơ chế cập nhật phù hợp với trọng số động, phát triển các thuật toán khai thác FWUP và mở rộng sang khai thác FWUCP với các chiến lược cắt tỉa nhằm giảm không gian tìm kiếm.

Hướng nghiên cứu thứ hai tập trung vào khai thác FWUP trên luồng dữ liệu định lượng động, với sự thay đổi cả về dữ liệu và trọng số các mục. Việc khảo sát các kỹ thuật trên luồng dữ liệu đã chỉ ra những thách thức về mặt tốc độ xử lý và giới hạn bộ nhớ. Mô hình cửa sổ trượt được xác định là phương pháp tiếp cận hiệu quả để nắm bắt các tri thức mới nhất một cách liên tục. Tuy nhiên, việc kết hợp giữa khai thác FWUP và luồng dữ liệu vẫn còn là một khoảng trống nghiên cứu lớn, đặc biệt là khi phải đối mặt với yêu cầu xử lý một lần, khả năng tái sử dụng tri thức trước đó và tích hợp yếu tố trọng số động vào mô hình. Ngoài ra, việc lựa chọn ngưỡng tối thiểu cũng trở nên khó khăn khi phân bố và đặc trưng dữ liệu liên tục thay đổi. Để giải quyết các vấn đề này, luận án phát triển các cấu trúc dữ liệu có khả năng tích hợp trọng số thay đổi và cập nhật theo cửa sổ trượt chỉ trong một lần quét, đáp ứng các yêu cầu khắt khe của luồng dữ liệu định lượng động. Đồng thời phát triển cơ chế nâng ngưỡng động và các chiến lược cắt tỉa cho bài toán khai thác top- k FWUP.

Hai hướng nghiên cứu trên là cơ sở trực tiếp cho các phương pháp và thuật toán được trình bày trong Chương 2 và Chương 3. Hình 1.2 trình bày mối quan hệ giữa các bài toán được nghiên cứu trong luận án. Các bài toán này có mối quan hệ kế thừa và phát triển chặt chẽ, trong đó kết quả của bài toán trước là nền tảng để xây dựng các mô hình, cấu trúc dữ liệu và thuật toán cho các bài toán tiếp theo. Cùng với đó, các phương pháp và thuật toán được đề xuất trong luận án được đánh giá thực nghiệm trên các CSDL chuẩn có đặc trưng khác nhau về quy mô và mức độ thưa,

dày. Các tiêu chí đánh giá chủ yếu gồm thời gian thực thi, mức sử dụng bộ nhớ và khả năng mở rộng khi dữ liệu ngày càng lớn. Thời gian thực thi được tính từ thời điểm bắt đầu đến khi kết thúc quá trình xử lý tương ứng với từng nội dung thực nghiệm và mức sử dụng bộ nhớ được ghi nhận trong quá trình thuật toán hoạt động. Đối với các thực nghiệm trên luồng dữ liệu, hiệu quả cập nhật và khai thác được đánh giá qua các lần trượt cửa sổ theo cùng cấu hình. Nhằm hạn chế ảnh hưởng của các yếu tố ngẫu nhiên trong môi trường thực thi, mỗi cấu hình thực nghiệm được chạy lặp lại khoảng 10 lần cho đến khi ổn định kết quả. Các số liệu về thời gian và bộ nhớ được tổng hợp bằng cách lấy giá trị trung bình cộng của các lần thực hiện. Cách tổ chức này được áp dụng thống nhất trong các thực nghiệm của luận án nhằm bảo đảm tính nhất quán và công bằng khi đánh giá, so sánh hiệu quả của các thuật toán. Thông tin chi tiết về CSDL, tham số và thiết lập thực nghiệm được trình bày cụ thể tại các chương tương ứng.



Hình 1.2. Sơ đồ mối quan hệ giữa các bài toán được giải quyết trong luận án.

1.5 Kết luận chương

Chương 1 đã hệ thống hóa các kiến thức nền tảng và khảo sát một cách chuyên sâu các công trình nghiên cứu liên quan đến khai thác mẫu, tạo dựng cơ sở khoa học vững chắc cho các đề xuất cốt lõi của luận án. Những kết quả tổng quan cho thấy, mặc dù lĩnh vực khai thác mẫu đã đạt được nhiều thành tựu đáng kể, nhưng việc kết hợp các yếu tố trọng số động, tính chất đóng, khai thác top- k và luồng dữ liệu trong khai thác FWUP vẫn chưa được nghiên cứu và giải quyết một cách hiệu quả. Cụ thể, chưa có nghiên cứu về bài toán khai thác FWUP trên CSDL định lượng động với trọng số các mục thay đổi, cũng như khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt. Bên cạnh đó, các hướng mở rộng liên quan đến khai thác mẫu trọng số hữu ích phổ biến đóng và khai thác top- k FWUP trong các môi trường này cũng chưa được quan tâm nghiên cứu đầy đủ. Đây chính là động lực và là cơ sở khoa học để luận án tập trung phát triển các cấu trúc dữ liệu, phương pháp và các thuật toán khai thác hiệu quả FWUP cho CSDL định lượng động và luồng dữ liệu định lượng động. Các nội dung này sẽ được trình bày chi tiết trong các chương tiếp theo.

CHƯƠNG 2: KHAI THÁC MẪU TRỌNG SỐ HỮU ÍCH PHỔ BIẾN TỪ CƠ SỞ DỮ LIỆU ĐỊNH LƯỢNG ĐỘNG

2.1 Giới thiệu bài toán

Như đã giới thiệu ở Chương 1, bài toán khai thác mẫu trọng số hữu ích phổ biến là mở rộng của bài toán khai thác mẫu phổ biến. Nhiệm vụ của bài toán là tìm ra tất cả các mẫu có độ hỗ trợ trọng số hữu ích không nhỏ hơn một ngưỡng hỗ trợ trọng số hữu ích tối thiểu. Bài toán này được tiến hành trên CSDL định lượng. Khác với CSDL nhị phân truyền thống chỉ phản ánh sự xuất hiện của các mục, CSDL định lượng còn lưu trữ thêm các thông tin bao gồm số lượng của mỗi mục trong từng giao dịch và trọng số của chúng.

Bảng 2.1. Ví dụ về CSDL định lượng có trọng số tĩnh

A						B	
CSDL giao dịch						Trọng số của các mục	
Giao dịch	A	B	C	D	E	Mục	Trọng số
t_1	1	0	2	3	1	A	0.3
t_2	0	2	1	0	2	B	0.5
t_3	2	0	1	0	1	C	0.2
t_4	1	0	3	2	0	D	0.4
t_5	3	1	1	2	1	E	0.7
t_6	1	2	2	0	3		

Ví dụ Bảng 2.1 mô tả một CSDL định lượng bao gồm sáu giao dịch và năm mục khác nhau. Các mục có thể là các mặt hàng, sản phẩm và được biểu diễn dưới dạng các ký tự A, B, C, D, E . Mỗi giao dịch có một định danh duy nhất trong toàn bộ CSDL và lưu số lượng của từng mục. Ví dụ giao dịch $t_1 = \{1, 0, 2, 3, 1\}$ có nghĩa là các mục A, C, D, E xuất hiện trong giao dịch với số lượng lần lượt là 1, 2, 3, 1, mục B không có trong giao dịch. Cùng với đó, trọng số của từng mục trong CSDL được cho tại Bảng 2.1B. Trọng số này có thể là đơn giá, tầm quan trọng và được đặt cố

định từ ban đầu cho CSDL.

Bài toán khai thác FWUP sử dụng một độ đo mới là độ hỗ trợ trọng số hữu ích (wus). Giá trị wus được tính toán dựa trên giá trị trọng số hữu ích giao dịch (twu) của các giao dịch. Giá trị wus của một mẫu là tổng các giá trị twu của các giao dịch chứa mẫu đó trên tổng giá trị twu của tất cả các giao dịch. Do đó, công thức của wus chịu ảnh hưởng của cả tần suất và trọng số. Đã có nhiều thuật toán được đề xuất để giải quyết bài toán này. Tuy nhiên, các phương pháp trên đều giả định trọng số các mục là cố định. Điều này không phù hợp với thực tế khi giá trị này có thể thay đổi, biến động và bị ảnh hưởng bởi nhiều yếu tố. Chẳng hạn, trong lĩnh vực bán lẻ, giá sản phẩm có thể giảm khi mua số lượng lớn hoặc trong các sự kiện giảm giá, thanh lý, và có thể tăng khi khan hiếm. Ví dụ, áo khoác mùa đông có thể mang lại lợi nhuận cao trong mùa lạnh nhưng giảm đáng kể khi bước sang mùa hè hoặc trong các đợt xả hàng cuối mùa. Tương tự, trong lĩnh vực tài chính, mức độ ưu tiên hoặc giá trị của các sản phẩm đầu tư cũng có thể thay đổi theo diễn biến thị trường. Sự biến động này cho thấy nhu cầu điều chỉnh trọng số của các mục trong CSDL là cần thiết, phù hợp với thực tiễn. Bỏ qua đặc điểm này làm cho các thuật toán khai thác FWUP hiện tại không áp dụng được hoặc cho kết quả không như mong đợi.

Để giải quyết vấn đề này, CSDL định lượng truyền thống được mở rộng để lưu trữ thêm thông tin về sự thay đổi trọng số. Trọng số của các mục sẽ được lưu thành từng lô, mỗi lô trọng số ứng với một tập các giao dịch liên tiếp nhau. Hai lô liên tiếp nhau sẽ khác nhau để thể hiện có sự thay đổi về trọng số của các mục. Bảng 2.2 mô tả một CSDL định lượng có trọng số thay đổi, gọi là CSDL định lượng động. Có hai tập trọng số xuất hiện trong CSDL này là dw_1 và dw_2 . Trong đó, trọng số của các mục trong các giao dịch t_1, t_2, t_3 là dw_1 và trọng số của các mục trong các giao dịch t_4, t_5, t_6 là dw_2 . Hai tập trọng số này là khác nhau cho thấy sự thay đổi về trọng số của các mục. Sự thay đổi này đòi hỏi phải có phương pháp tiếp cận phù hợp để xác định chính xác các FWUP thỏa mãn.

Nội dung Chương 2 trình bày một số định nghĩa về CSDL định lượng động và

bài toán khai thác FWUP từ CSDL định lượng động. Trên cơ sở đó, hai thuật toán có tên là dFWUT sử dụng cấu trúc dữ liệu Tidset và dFWUNL sử dụng một cấu trúc dữ liệu mới là WUNList được đề xuất để giải quyết bài toán này. Cấu trúc WUNList là một mở rộng của cấu trúc N-list, được thiết kế để biểu diễn hiệu quả CSDL định lượng động. Các kết quả thực nghiệm cho thấy sự hiệu quả của thuật toán dFWUNL cả về thời gian khai thác và bộ nhớ sử dụng.

Bảng 2.2. Ví dụ về CSDL định lượng động ($dQDB_E$)

A						
CSDL giao dịch						
Giao dịch	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	Tập trọng số
t_1	1	0	2	3	1	dw_1
t_2	0	2	1	0	2	
t_3	2	0	1	0	1	
t_4	1	0	3	2	0	dw_2
t_5	3	1	1	2	1	
t_6	1	2	2	0	3	

B					
Tập trọng số	<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>
dw_1	0.3	0.5	0.2	0.4	0.7
dw_2	0.4	0.3	0.4	0.2	0.3

Ngoài ra, trong bài toán khai thác FWUP, số lượng mẫu thu được thường rất nhiều khi người dùng đặt ngưỡng nhỏ. Để tối ưu quá trình khai thác và tài nguyên lưu trữ các kết quả, một biểu diễn cô đọng của mẫu FWUP, được gọi là mẫu trọng số hữu ích phổ biến đóng, cũng được nghiên cứu. Mẫu FWUCP giúp giảm số lượng mẫu kết quả nhưng vẫn bảo đảm đầy đủ tri thức. Để khai thác hiệu quả các mẫu phổ biến đóng này trên CSDL định lượng động, luận án phát triển các phương pháp cắt tia không gian tìm kiếm, kiểm tra nhanh tính đóng của một mẫu dựa trên cấu trúc dữ liệu

WUNList đã giới thiệu ở trên. Kết quả thực nghiệm cho thấy các cải tiến có hiệu quả hơn so với thuật toán trước đó, đặc biệt là trên các CSDL thưa và rất thưa.

Các kết quả nghiên cứu trình bày trong chương này đã được công bố trong các công trình [I], [II] và [V]. Trong đó, công trình [I] và [II] trình bày các kết quả về khai thác FWUP trên CSDL định lượng động và công trình [V] phát triển bài toán khai thác mẫu trọng số hữu ích phổ biến đóng trên CSDL định lượng động. Ngoài ra, một phần kết quả của hướng nghiên cứu này được mở rộng theo hướng xử lý song song trong công trình [III].

2.2 Một số định nghĩa cơ bản

Định nghĩa 2.1 (Mẫu). Cho I là một tập hữu hạn có n mục, $I = \{i_1, i_2, \dots, i_n\}$. Một tập $P \subseteq I$ được gọi là mẫu (hoặc tập mục), và được gọi là k -pattern nếu P chứa đúng k mục.

Định nghĩa 2.2 (CSDL định lượng động). Một CSDL định lượng động, ký hiệu là $dQDB$, là một bộ gồm ba thành phần (T, I, DW) . Trong đó $T = \{t_1, t_2, \dots, t_m\}$ là tập các giao dịch định lượng với m là số lượng giao dịch, $I = \{i_1, i_2, \dots, i_n\}$ là tập các mục với n là số lượng mục, và $DW = \{dw_1, dw_2, \dots, dw_q\}$ là các tập trọng số với q là số lượng lô. Mỗi giao dịch t_k có một định danh duy nhất là Tid , có dạng $t_k = \{x_{k1}, x_{k2}, \dots, x_{kn}\}$, $k = 1, 2, \dots, m$, trong đó x_{ki} là số lượng của mục thứ i trong giao dịch t_k , và tương ứng với một lô $dw_j \in DW$. Mỗi lô $dw_j = \{w_{j1}, w_{j2}, \dots, w_{jn}\}$, $j = 1, 2, \dots, q$, là một tập trọng số của các mục trong I .

Thuật ngữ “động” trong ngữ cảnh này khác với cách sử dụng thông thường trong các lĩnh vực như khai thác tăng trưởng hoặc khai thác luồng dữ liệu. Trong bài toán này, tính động đề cập đến sự biến đổi của trọng số gắn với các mục dữ liệu. Cụ thể, trọng số của cùng một mục không được giả định là một giá trị cố định trên toàn bộ CSDL, mà có thể nhận các giá trị khác nhau trong các giao dịch hoặc tại các thời điểm khác nhau. Điều này phản ánh sát hơn các ứng dụng thực tế mà giá bán, lợi nhuận hoặc mức độ quan trọng của các mục thường xuyên thay đổi theo điều kiện

kinh doanh và môi trường hoạt động. Do đó, các công thức tính toán các độ đo liên quan đến trọng số hữu ích được điều chỉnh để xử lý sự không đồng nhất về trọng số của các mục trong quá trình khai thác.

Ví dụ 2.1: Bảng 2.2 mô tả một ví dụ về CSDL định lượng động ($dQDB_E$). CSDL này có tổng cộng sáu giao dịch $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ và năm mục $I = \{A, B, C, D, E\}$ với hai lô trọng số $DW = \{dw_1, dw_2\}$. Trọng số của các mục trong các giao dịch t_1, t_2, t_3 là $dw_1 = \{0.3, 0.5, 0.2, 0.4, 0.7\}$ và trọng số của các mục trong các giao dịch t_4, t_5, t_6 là $dw_2 = \{0.4, 0.3, 0.4, 0.2, 0.3\}$.

Định nghĩa 2.3 (Trọng số hữu ích giao dịch). Giá trị trọng số hữu ích giao dịch, ký hiệu là twu , của một giao dịch t_k với tập trọng số $dw_p \in DW$ được xác định như sau:

$$twu(t_k) = \frac{\sum_{i \in t_k} w_{pi} \times x_{ki}}{|t_k|} \quad (2.1)$$

trong đó, $w_{pi} \in dw_p$ là trọng số của mục i trong giao dịch t_k , x_{ki} là số lượng của mục i trong t_k , và $|t_k|$ là số lượng các mục xuất hiện trong t_k .

Giá trị twu biểu diễn mức trọng số hữu ích trung bình của giao dịch, đồng thời thể hiện ảnh hưởng tương đối của giao dịch đó đối với toàn bộ CSDL. Khác với các mô hình truyền thống sử dụng trọng số cố định cho toàn bộ CSDL, giá trị twu trong nghiên cứu này được tính toán dựa trên tập trọng số tương ứng với từng giao dịch, qua đó phản ánh chính xác đặc tính trọng số thay đổi của dữ liệu. Độ đo này được sử dụng làm thành phần cơ sở để tính độ hỗ trợ trọng số hữu ích của các mẫu trong **Định nghĩa 2.4**.

Ví dụ 2.2: Giao dịch t_2 trong $dQDB_E$ có giá trị twu được tính như sau: $twu(t_2) = (0.5 \times 2 + 0.4 + 0.7 \times 2)/3 \approx 0.87$. Tương tự, giá trị twu của tất cả các giao dịch trong $dQDB_E$ được tính như trong Bảng 2.3.

Định nghĩa 2.4 (Độ hỗ trợ trọng số hữu ích). Gọi $sumtwu = \sum_{t_k \in T} twu(t_k)$ là tổng giá trị twu của các giao dịch trong $dQDB$. Độ hỗ trợ trọng số hữu ích, ký hiệu

là wus , của một mẫu P được xác định như sau:

$$wus(P) = \frac{\sum_{t_k \in T(P)} twu(t_k)}{sumtwu} \quad (2.2)$$

trong đó, $T(P)$ là tập các giao dịch trong CSDL có chứa mẫu P .

Giá trị độ hỗ trợ trọng số hữu ích của một mẫu nằm trong khoảng $[0, 1]$, phản ánh tỷ lệ đóng góp của các giao dịch chứa mẫu đó so với toàn bộ CSDL xét theo giá trị trọng số hữu ích giao dịch. Giá trị này càng lớn cho thấy mẫu xuất hiện trong các giao dịch có mức đóng góp trọng số hữu ích càng cao đối với toàn bộ CSDL. Do đó, wus được sử dụng làm độ đo trung tâm để xác định các mẫu trọng số hữu ích phổ biến trong các phần tiếp theo của luận án.

Bảng 2.3. Giá trị twu của tất cả các giao dịch trong $dQDB_E$

Giao dịch	Tập trọng số	Công thức	twu
t_1	dw_1	$(0.3 \times 1 + 0.2 \times 2 + 0.5 \times 3 + 0.7 \times 1) / 4$	0.65
t_2	dw_1	$(0.5 \times 2 + 0.4 \times 1 + 0.7 \times 2) / 3$	0.87
t_3	dw_1	$(0.3 \times 2 + 0.2 \times 1 + 0.7 \times 1) / 3$	0.50
t_4	dw_2	$(0.4 \times 1 + 0.4 \times 3 + 0.2 \times 2) / 3$	0.67
t_5	dw_2	$(0.4 \times 3 + 0.3 \times 1 + 0.4 \times 1 + 0.2 \times 2 + 0.3 \times 1) / 5$	0.52
t_6	dw_2	$(0.4 \times 1 + 0.3 \times 2 + 0.4 \times 2 + 0.3 \times 3) / 4$	0.68
Tổng giá trị twu ($sumtwu$)			3.89

Ví dụ 2.3: Mục B xuất hiện trong ba giao dịch t_2 , t_5 và t_6 nên giá trị wus của B là: $wus(B) = (twu(t_2) + twu(t_5) + twu(t_6)) / sumtwu = (0.87 + 0.52 + 0.68) / 3.89 \approx 0.53$.

Định nghĩa 2.5 (Mẫu trọng số hữu ích phổ biến). Cho một ngưỡng trọng số hữu ích tối thiểu, ký hiệu là $minwus$, và một mẫu $P \subseteq I$, nếu $wus(P) \geq minwus$ thì P được gọi là mẫu trọng số hữu ích phổ biến.

Định nghĩa 2.6 (Bài toán khai thác mẫu trọng số hữu ích phổ biến). Khai thác mẫu trọng số hữu ích phổ biến từ CSDL định lượng động là bài toán tìm tất cả

các FWUP thoả mãn giá trị $minwus$ cho trước do người dùng chỉ định. Tập đầy đủ các FWUP trong CSDL định lượng động được ký hiệu là $\mathbb{F}$.

Định nghĩa 2.7 (Tidset). Tidset của một mẫu P , ký hiệu là $t(P)$, là tập lưu trữ các giá trị Tid của các giao dịch có chứa P .

Định nghĩa 2.8 (Bao đóng của một mẫu). Bao đóng của một mẫu $P \subseteq I$, ký hiệu là $c(P)$, là mẫu $Q \subseteq I$ dài nhất mà $P \subseteq Q$ và $wus(P) = wus(Q)$. $c(P)$ cũng chính bằng giao của các giao dịch có chứa mẫu P : $c(P) = \bigcap_{t_k \in T(P)} t_k$.

Ví dụ 2.4: Trong $dQDB_E$, ta có BCE là mẫu dài nhất có chứa B mà $wus(BCE) = wus(B)$, nên $c(B) = BCE$. Ngoài ra, $c(B)$ còn được tính bằng $c(B) = t_2 \cap t_5 \cap t_6 = BCE$.

Định nghĩa 2.9 (Mẫu đóng). Cho một mẫu $P \subseteq I$, P được gọi là mẫu đóng khi và chỉ khi $c(P) = P$. Tập tất cả các mẫu đóng trong CSDL định lượng động được ký hiệu là $\mathbb{C}$.

Ví dụ 2.5: Trong $dQDB_E$, mẫu B không phải là mẫu đóng vì $c(B) = BCE \neq B$, còn BCE là mẫu đóng vì $c(BCE) = BCE$.

Định nghĩa 2.10 (Mẫu trọng số hữu ích phổ biến đóng). Cho một mẫu $P \subseteq I$ là một FWUP, P được gọi là mẫu trọng số hữu ích phổ biến đóng khi và chỉ khi không tồn tại một mẫu Q nào mà $P \subset Q$ và $wus(P) = wus(Q)$.

Ví dụ 2.6: Trong $dQBD_E$ với $minwus = 0.5$, mẫu BCE là FWUP vì $wus(BCE) = 0.53 > minwus$. Đồng thời, BCE cũng là mẫu đóng theo Ví dụ 2.5. Do đó, BCE là FWUCP.

Định nghĩa 2.11 (Bài toán khai thác mẫu trọng số hữu ích phổ biến đóng). Khai thác mẫu trọng số hữu ích phổ biến đóng từ CSDL định lượng động là bài toán tìm tất cả các FWUCP thoả mãn ngưỡng $minwus$ cho trước. Tập tất cả các FWUCP trong CSDL định lượng động được ký hiệu là $\mathbb{FC}$. Ta có, $\mathbb{FC} = \mathbb{F} \cap \mathbb{C}$.

2.3 Thuật toán khai thác mẫu trọng số hữu ích phổ biến từ cơ sở dữ liệu định lượng động

Các thuật toán khai thác FWUP hiện có như MBiS-FWUI [25], WUN-Miner [5] đã đạt được nhiều kết quả tích cực trong việc kết hợp đồng thời yếu tố tần suất xuất hiện và tầm quan trọng của các mục dữ liệu. Tuy nhiên, các phương pháp này đều được xây dựng trên giả định rằng trọng số của các mục là cố định trong toàn bộ quá trình khai thác. Giả định này giúp đơn giản hóa việc tính toán các độ đo trọng số hữu ích, nhưng không còn phù hợp với nhiều ứng dụng thực tế khi giá trị hoặc mức độ quan trọng của các mục có thể thay đổi theo thời gian. Do sự khác biệt giữa CSDL định lượng tĩnh truyền thống và *dQDB* nên các độ đo được sử dụng trong khai thác FWUP cũng phải thay đổi tương ứng. Điều này dẫn đến việc các cấu trúc dữ liệu và chiến lược cắt tỉa được thiết kế cho môi trường tĩnh không còn có thể áp dụng trực tiếp. Bên cạnh đó, việc cập nhật trọng số có thể làm thay đổi tập FWUP thu được, đòi hỏi thuật toán phải phản ánh chính xác trạng thái hiện thời của dữ liệu mà vẫn đảm bảo hiệu quả tính toán.

Để giải quyết vấn đề này, luận án đề xuất hai hướng tiếp cận với các cấu trúc dữ liệu cơ sở. Hướng tiếp cận thứ nhất sử dụng cấu trúc Tidset nhằm quản lý trực tiếp các giao dịch chứa mẫu và phát triển cấu trúc dWUPT-tree để tiến hành khai thác, từ đó xây dựng thuật toán dFWUT (tiểu mục 2.3.1). Hướng tiếp cận thứ hai với thuật toán có tên là dFWUNL, sử dụng phương pháp nén dữ liệu vào cấu trúc WUN-tree và tạo cấu trúc WUNList để khai thác FWUP (tiểu mục 2.3.2). Cấu trúc WUNList có khả năng giảm độ dài nhanh chóng sau mỗi lần kết hợp, hỗ trợ tính toán trực tiếp độ hỗ trợ trọng số hữu ích của các mẫu. Cùng với đó, thuật toán dFWUNL còn bao gồm việc tính toán trước các mẫu hai phần tử và nhanh chóng xác định các mẫu có cùng giá trị *wus* mà không cần kết hợp WUNList giúp tăng tốc quá trình khai thác. Hai thuật toán được đề xuất đều hướng tới mục tiêu khai thác đầy đủ tập FWUP trong môi trường trọng số động, đồng thời giảm chi phí tính toán và bộ nhớ sử dụng so với các phương pháp truyền thống.

2.3.1 Thuật toán dFWUT

Thuật toán đầu tiên được đề xuất có tên là dFWUT sử dụng một cấu trúc dữ liệu mới có tên là dWUPT-tree (dynamic Weighted Utility Pattern Tidset tree), là mở rộng của cấu trúc IT-tree [93], để khai thác FWUP từ $dQDB$. Mỗi nút trong dWUPT-tree bao gồm ba phần:

- (1) Mẫu P là một FWUP;
- (2) $t(P)$ là Tidset của mẫu P ;
- (3) $wus(P)$ là giá trị độ hỗ trợ trọng số hữu ích của mẫu P .

dWUPT-tree kế thừa cách tổ chức các mẫu theo lớp tương đương và biểu diễn dọc bằng Tidset của IT-tree. Cụ thể, cấu trúc dWUPT-tree có nhiều cấp, mỗi cấp có nhiều lớp tương đương, mỗi lớp tương đương bao gồm nhiều nút. Các nút trong cùng một lớp tương đương đều có cùng tiền tố và chỉ có mục cuối cùng là khác nhau. Điểm mở rộng của dWUPT-tree là mỗi nút còn lưu giá trị độ hỗ trợ trọng số hữu ích của mẫu để hỗ trợ trực tiếp việc kiểm tra điều kiện FWUP. Hai nút trong cùng một lớp tương đương có thể kết hợp để tạo ra một nút mới ở cấp tiếp theo.

Quá trình xây dựng dWUPT-tree được thực hiện đồng thời với quá trình khai thác FWUP trong thuật toán dFWUT và được mô tả ở Hình 2.1. Đầu tiên, thuật toán quét $dQDB$ một lần để tính twu của các giao dịch, xác định Tidset và tính wus cho từng mục. Các mục có giá trị wus thỏa $minwus$ được giữ lại để tạo các nút ở cấp đầu tiên của cây và hình thành lớp tương đương ban đầu C_1 . Các mục không thỏa ngưỡng được loại bỏ. Từ một lớp tương đương tại cấp hiện tại, dFWUT lần lượt kết hợp hai mẫu có cùng tiền tố để tạo mẫu mới như sau.

Cho Px và Py là hai $(k-1)$ -pattern có cùng tiền tố P và là hai nút trong dWUPT-tree với Tidset tương ứng là $t(Px)$ và $t(Py)$. Tidset của mẫu Pxy được xác định bằng phép giao:

$$t(Pxy) = t(Px) \cap t(Py) \quad (2.3)$$

Độ hỗ trợ trọng số hữu ích của mẫu Pxy được tính dựa trên $t(Pxy)$.

$$wus(Pxy) = \frac{\sum_{t_k \in t(Pxy)} twu(t_k)}{sumtwu} \quad (2.4)$$

Nếu $wus(Pxy) \geq minwus$ thì Pxy là FWUP và được thêm vào dWUPT-tree để tiếp tục mở rộng. Quá trình này được thực hiện đệ quy cho đến khi không còn mẫu mới thỏa điều kiện nào được tạo thành. Như vậy, thuật toán dFWUT chỉ cần quét CSDL một lần để chuyển đổi CSDL từ chiều ngang sang chiều dọc, tạo cấu trúc Tidset của các mục và sử dụng Tidset để tính toán và tìm FWUP.

Thuật toán 2.1. Thuật toán dFWUT
Input: CSDL định lượng động $dQDB$ và ngưỡng $minwus$.
Output: Tập tất cả các mẫu trọng số hữu ích phổ biến $FWUPs$.
1. Quét $dQDB$ để tính twu của mỗi giao dịch, giá trị wus và Tidset của mỗi mục trong I 2. Lọc các mục có $wus \geq minwus$ và thêm vào C_1 3. $FWUPs \leftarrow C_1$ 4. $dWUPT-tree \leftarrow \emptyset$ 5. Tid_Mine($[C_1]$) 6. return $FWUPs$
Procedure Tid_Mine ($[C]$) 1. $dWUPT-tree \leftarrow [C]$ 2. foreach $c_i \in [C]$ do 3. $[C_{next}] \leftarrow \emptyset$ 4. foreach $c_j \in [C]$, với $j > i$ do 5. $P \leftarrow c_i \cup c_j$ 6. $t(P) \leftarrow t(c_i) \cap t(c_j)$ 7. if $wus(P) \geq minwus$ then 8. $FWUPs \leftarrow P$ 9. $[C_{next}] \leftarrow \{P, t(P), wus(P)\}$ 10. Tid_Mine($[C_{next}]$)

Hình 2.1. Thuật toán dFWUT.

Ví dụ 2.7: Xét $dQDB_E$ trong Bảng 2.2 với $minwus = 0.5$, quá trình khai thác của thuật toán dFWUT như sau. Đầu tiên, thuật toán quét $dQDB_E$ để tính twu của các giao dịch (Bảng 2.3), Tidset và wus của các mục (Bảng 2.4), và $sumtwu = 3.89$. Ví dụ, mục A xuất hiện trong các giao dịch t_1, t_3, t_4, t_5, t_6 nên $t(A) = \{1, 3, 4, 5, 6\}$, và $wus(A) = [twu(t_1) + twu(t_3) + twu(t_4) + twu(t_5) + twu(t_6)]/sumtwu = (0.65 + 0.5 + 0.67 + 0.52 + 0.68)/3.89 \approx 0.78$. Tiếp theo, tập các 1-pattern C_1 được tạo ra bằng cách thêm các nút chứa các mục có wus không nhỏ hơn $minwus$. Vì $wus(B) = 0.43 < minwus = 0.5$ nên mục B bị loại. Do đó, ta có các mục A, C, D và E được thêm vào C_1 . Sau đó, thuật toán khởi tạo dWUPT-tree và gọi thủ tục Tid_Mine để tiến hành khai thác.

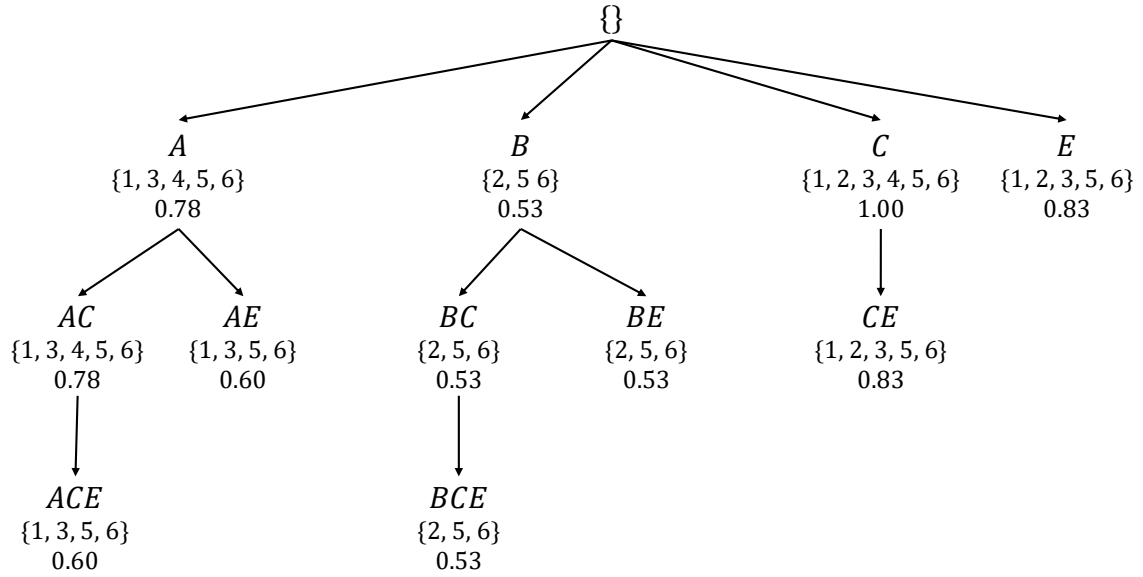
Bảng 2.4. Tidset và wus của các mục trong $dQDB_E$

Mục	Tidset	wus
A	$\{1, 3, 4, 5, 6\}$	0.78
B	$\{2, 5, 6\}$	0.53
C	$\{1, 2, 3, 4, 5, 6\}$	1.00
D	$\{1, 4, 5\}$	0.47
E	$\{1, 2, 3, 5, 6\}$	0.83

Thuật toán tiến hành kết hợp A với các nút B, C và E để tạo ra AB, AC và AE tương ứng. Ta có $t(AB) = t(A) \cap t(B) = \{1, 3, 4, 5, 6\} \cap \{2, 5, 6\} = \{5, 6\}$ và $wus(AD) = (0.52 + 0.68)/3.89 \approx 0.31 < minwus$, nên AB không thoả mãn. Tiếp theo, $t(AC) = t(A) \cap t(C) = \{1, 3, 4, 5, 6\} \cap \{1, 2, 3, 4, 5, 6\} = \{1, 3, 4, 5, 6\}$ và $wus(AC) = (0.65 + 0.5 + 0.67 + 0.52 + 0.68)/3.89 \approx 0.73 > minwus$, do đó AC được thêm vào dWUPT-tree. Cuối cùng, $t(AE) = t(A) \cap t(E) = \{1, 3, 4, 5, 6\} \cap \{1, 2, 3, 5, 6\} = \{1, 3, 5, 6\}$ và $wus(AE) = (0.65 + 0.5 + 0.52 + 0.68)/3.89 \approx 0.60 > minwus$, do đó AE được thêm vào dWUPT-tree.

Tiếp theo, thuật toán dFWUT gọi đệ quy để tạo ra tất cả các lớp tương đương con. Xét lớp tương đương $[A]$, thuật toán kết hợp AC với AE để tạo ra ACE . Ta có

$t(ACE) = t(AC) \cap t(AE) = \{1, 3, 4, 5, 6\} \cap \{1, 3, 5, 6\} = \{1, 3, 5, 6\}$, $wus(ACD) = (0.65 + 0.67 + 0.52 + 0.68)/3.89 \approx 0.60 > minwus$, do đó ACE được thêm vào dWUPT-tree. Vì không có lớp tương đương nào được tạo từ $[AC]$, thuật toán quay lại và tiếp tục kết hợp các nút có cùng lớp tương đương tương tự như trên để tạo các nút mới. Kết quả cuối cùng được hiện thị ở Hình 2.2.



Hình 2.2. dWUPT-tree từ $dQDB_E$ với $minwus = 0.5$.

2.3.2 Thuật toán dFWUNL

2.3.2.1 Cấu trúc WUNList

Năm 2012, Deng và đồng sự [14] đề xuất một cấu trúc hiệu quả có tên là N-list với nhiều ưu điểm vượt trội để khai thác các mẫu phổ biến. Lấy cảm hứng từ cấu trúc này, luận án phát triển cấu trúc có tên là WUNList để khai thác FWUP từ $dQDB$. WUNList được tạo thành ban đầu từ cấu trúc WUN-tree và được định nghĩa như sau.

Định nghĩa 2.12 (WUN-tree). WUN-tree là một cấu trúc cây tiền tổ đa nhánh với một nút gốc và một tập hợp các nút con. Mỗi nút lưu trữ một bộ năm giá trị ($name, childlist, pre, post, weight$). Trong đó, $name$ là tên mục trọng số hữu ích phổ biến đại diện cho nút, $childlist$ là tập hợp các nút con của nó, pre là thứ hạng tiền thứ tự của nút, $post$ là thứ hạng hậu thứ tự của nút và $weight$ là tổng giá trị twu

của các giao dịch đi qua nút này.

Thuật toán xây dựng WUN-tree được mô tả trong Hình 2.3.

Thuật toán 2.2. Xây dựng WUN-tree
Input: CSDL định lượng động $dQDB$ và ngưỡng $minwus$.
Output: WUN-tree.
Procedure Build_WUN-tree($dQDB, minwus$) 1. Quét $dQDB$ để tính wus cho mỗi mục 2. Thêm các mục có $wus \geq minwus$ vào I_1 và sắp xếp I_1 giảm dần theo wus 3. Tạo nút gốc R của WUN-tree, với tên là “Null” 4. foreach giao dịch $t \in dQDB$ do 5. $ft = \{i \mid i \in t \wedge i \in I_1\}$ 6. Sắp xếp các mục trong ft giảm dần theo I_1 7. Insert_WUN-tree(ft, R) 8. Duyệt WUN-tree để tạo pre và $post$ cho các nút 9. return R
Procedure Insert_WUN-tree(ft, R) 1. i là mục đầu tiên của ft 2. if R có nút con N với $N.name = i$ then 3. $N.weight += twu(ft)$ 4. else 5. Tạo nút con N mới của R 6. $N.name \leftarrow i; N.weight \leftarrow twu(ft);$ 7. if $ft \neq \emptyset$ then 8. Insert_WUN-tree($ft/\{i\}, N$)

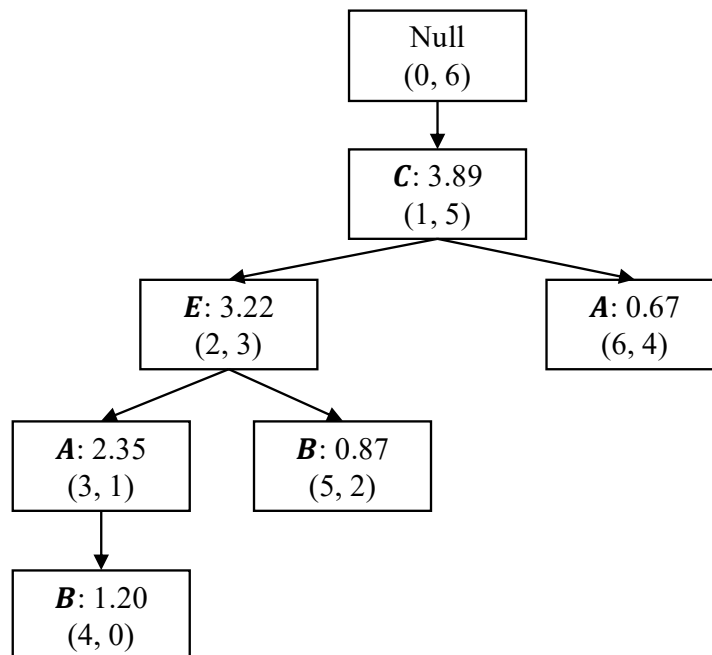
Hình 2.3. Thuật toán xây dựng cấu trúc WUN-tree.

Ví dụ 2.8: Xét $dQDB_E$ trong Bảng 2.2 với $minwus = 0.5$, WUN-tree được xây dựng theo các bước sau. Đầu tiên, ta quét CSDL để tạo I_1 là tập hợp các 1-FWUP và sắp xếp chúng theo thứ tự giảm dần dựa trên giá trị wus (Bảng 2.5B). Tiếp theo,

các mục trong mỗi giao dịch được sắp xếp theo thứ tự I_1 và mục D không có trong I_1 bị loại bỏ (Bảng 2.5A). Thuật toán tiến hành thêm lần lượt các giao dịch đã sắp xếp vào WUN-tree. Sau đó, ta duyệt WUN-tree để thêm các giá trị pre và $post$ cho mỗi nút. Kết quả ta thu được WUN-tree của $dQDB_E$ được hiển thị trong Hình 2.4.

Bảng 2.5. $dQDB_E$ sau khi áp dụng ngưỡng $minwus = 0.5$

A							B	
CSDL đã sắp xếp							I_1	
Giao dịch	C	E	A	B	D	twu	Mục	wus
t_1	2	1	1	0	3	0.65	C	1
t_2	1	2	0	2	0	0.87	E	0.83
t_3	1	1	2	0	0	0.50	A	0.78
t_4	3	0	1	0	2	0.67	B	0.53
t_5	1	1	3	1	2	0.52	D	0.47
t_6	2	3	1	2	0	0.68		
$sumtwu$						3.89		



Hình 2.4. WUN-tree của $dQDB_E$ với $minwus = 0.5$.

Định nghĩa 2.13 (WUN-code). WUN-code của một nút N trong WUN-tree có dạng $(N.pre, N.post, N.weight)$.

Từ WUN-code, ta có thể xác định mối quan hệ tổ tiên giữa các nút như sau.

Định lý 2.1 [24]. Xét hai nút N_1 và N_2 thuộc WUN-tree với WUN-code lần lượt là $(pre_1, post_1, weight_1)$ và $(pre_2, post_2, weight_2)$. N_1 là tổ tiên của N_2 khi và chỉ khi $pre_1 < pre_2$ và $post_1 > post_2$.

Ví dụ 2.9: Với WUN-tree hiển thị trong Hình 2.4, hai nút C và E có WUN-code lần lượt là $(1, 5, 3.89)$ và $(2, 3, 3.22)$. Nút C là tổ tiên của nút E vì $C.pre = 1 < E.pre = 2$ và $C.post = 5 > E.post = 3$.

Định nghĩa 2.14 (WUNList của một mục). WUNList của một mục x , ký hiệu là $wunl(x)$, là dãy tất cả các WUN-code $\{(pre_1, post_1, weight_1), (pre_2, post_2, weight_2), \dots, (pre_n, post_n, weight_n)\}$ của nút đại diện cho mục x trong WUN-tree. WUNList được sắp xếp tăng dần theo giá trị pre ($pre_1 < pre_2 < \dots < pre_n$).

Ví dụ 2.10: WUNList của các mục từ WUN-tree ở Hình 2.4 lần lượt là: $wunl(C) = \{(1, 5, 3.89)\}$, $wunl(E) = \{(2, 3, 3.22)\}$, $wunl(A) = \{(3, 1, 2.35), (6, 4, 0.67)\}$, và $wunl(B) = \{(4, 0, 1.20), (5, 2, 0.87)\}$.

Định nghĩa 2.15 (WUNList của một k -pattern). Cho Px và Py là hai $(k-1)$ -pattern có cùng tiền tố P , và x đứng sau y theo thứ tự trong I_1 (ký hiệu là $y < x$). $wunl(Px)$ và $wunl(Py)$ lần lượt là WUNList của Px và Py . $wunl(Pxy)$ là WUNList của k -pattern Pxy được tạo theo hai bước sau:

(1) Với mỗi WUN-code $C_u \in wunl(Px)$ và $C_v \in wunl(Py)$, nếu C_v là tổ tiên của C_u , ta thêm $(C_v.pre, C_v.post, C_u.weight)$ vào $wunl(Pxy)$.

(2) Duyệt tất cả WUN-code trong $wunl(Pxy)$ và hợp nhất các WUN-code có cùng giá trị pre vào một WUN-code với giá trị trọng số mới bằng tổng trọng số của các nút đang xét.

Ví dụ 2.11: Từ Ví dụ 2.10, ta có $wunl(C) = \{(1, 5, 3.89)\}$, $wunl(E) =$

$\{(2, 3, 3.22)\}$, và $wunl(A) = \{(3, 1, 2.35), (6, 4, 0.67)\}$. Theo Bảng 2.5, mục C đứng trước mục E , và mục E đứng trước mục A trong I_1 . $wunl(AC)$ được tính như sau: $(1, 5, 3.89)$ là tổ tiên của $(3, 1, 2.35)$, do đó, $(1, 5, 2.35)$ được thêm vào $wunl(AC)$. Tiếp theo, $(1, 5, 3.89)$ là tổ tiên của $(6, 4, 0.67)$ nên $(1, 5, 0.67)$ được thêm vào $wunl(AC)$. Kết thúc bước 1, ta thu được $wunl(AC) = \{(1, 5, 2.35), (1, 5, 0.67)\}$. Thuật toán tiến hành hợp nhất các WUN-code có cùng giá trị pre trong $wunl(AC)$, và kết quả cuối cùng là $wunl(AC) = \{(1, 5, 3.02)\}$. Tương tự, ta tính được $wunl(AE) = \{(2, 3, 2.35)\}$ và $wunl(AEC) = \{(1, 5, 2.35)\}$.

Định lý 2.2. Cho $P = x_1x_2 \dots x_n$ là một mẫu với x_i đứng sau x_{i+1} theo I_1 , có WUNList là $wunl(P) = \{(pre_1, post_1, weight_1), (pre_2, post_2, weight_2), \dots, (pre_m, post_m, weight_m)\}$. Độ hỗ trợ trọng số hữu ích, $wus(P)$, của mẫu P được tính như sau:

$$wus(P) = \frac{\sum_{i=1}^m weight_i}{sumtwu} \quad (2.5)$$

Chứng minh. Với $(pre_i, post_i, weight_i)$ là một WUN-code trong WUNList của mẫu P , theo **Định nghĩa 2.14** và **Định nghĩa 2.15**, ta dễ dàng suy ra rằng luôn có một nút đại diện cho mục x_n trên WUN-tree với cùng giá trị $(pre_i, post_i)$ của WUN-code. (a)

Ngược lại, cũng theo **Định nghĩa 2.15**, giá trị $weight_i$ của WUN-code thuộc WUNList của mẫu P là tổng trọng số của các WUN-code có cùng giá trị pre_i , với tất cả đều là hậu duệ của x_n . Dựa trên quá trình xây dựng WUN-tree, $weight_i$ cũng là tổng giá trị twu của các giao dịch có chứa mẫu P và đi qua cây con có gốc là nút đại diện cho mục x_n . (b)

Gọi T_i là tập các giao dịch có chứa mẫu P và đi qua cây con có gốc là nút đại diện cho mục x_n , khi đó, ta có $weight_i = \sum_{t_k \in T_i} twu(t_k)$. (c)

Gọi $T(P)$ là tập các giao dịch có chứa P . Từ (a), ta có $T(P) = \bigcup_{i=1}^m T_i$. (d)

Theo Phương trình 3.2, kết hợp với (c) và (d), ta có:

$$wus(P) = \frac{\sum_{t_k \in T(P)} twu(t_k)}{sumtwu} = \frac{\sum_{i=1}^m \sum_{t_k \in T_i} twu(t_k)}{sumtwu} = \frac{\sum_{i=1}^m weight_i}{sumtwu}.$$

Định lý đã được chứng minh.

Ví dụ 2.12: WUNList của mẫu B là $wunl(B) = \{(4, 0, 1.2), (5, 2, 0.87)\}$ và $sumtwu = 3.89$. Theo **Định lý 2.2**, ta có $wus(B) = (1.2 + 0.87)/3.89 \approx 0.53$.

Định lý 2.2 cho thấy độ hỗ trợ trọng số hữu ích của một mẫu có thể được tính trực tiếp từ WUNList mà không cần truy xuất lại WUN-tree hoặc quét lại CSDL. Do đó, để khai thác các mẫu có độ dài lớn hơn, vấn đề đặt ra là việc xây dựng hiệu quả WUNList của mẫu mới từ các WUNList đã có. Theo **Định nghĩa 2.15**, WUNList của một k -pattern có thể được tạo bằng cách kết hợp các WUNList của hai mẫu thành phần. Quá trình này dựa trên việc xác định các WUN-code thỏa mãn quan hệ tổ tiên trên WUN-tree thông qua các giá trị *pre* và *post*. Thuật toán tạo WUNList của mẫu mới được trình bày trong Hình 2.5. Thuật toán bắt đầu bằng việc khởi tạo WUNList kết quả và các biến hỗ trợ (dòng 1-3). Hai WUNList đầu vào được duyệt đồng thời theo thứ tự tăng dần của giá trị *pre*. Khi một WUN-code trong $wunl_2$ là tổ tiên của một WUN-code trong $wunl_1$, tức là thỏa mãn điều kiện $pre_2 < pre_1$ và $post_2 > post_1$, một WUN-code mới sẽ được tạo hoặc cập nhật trong WUNList kết quả (dòng 5-10). Đồng thời, giá trị trọng số hữu ích tương ứng được cộng dồn vào biến SC . Trong quá trình duyệt, các biến CR_1 và CR_2 được cập nhật để phản ánh phần độ hỗ trợ trọng số hữu ích tối đa còn có thể được tạo ra từ hai WUNList đầu vào. Nếu tổng giá trị đã tích lũy và giá trị tối đa còn có thể đạt được nhỏ hơn ngưỡng yêu cầu, thuật toán dừng sớm và trả về *NULL* (dòng 14). Chiến lược này giúp nhanh chóng kết thúc việc tạo WUNList mới của các mẫu không thỏa mãn.

Như vậy, cấu trúc WUN-tree nén các tiền tố chung của các giao dịch nên số lượng nút được lưu trữ thường nhỏ hơn tổng số mục xuất hiện trong CSDL, đặc biệt đối với các bộ dữ liệu dày. Được xây dựng từ WUN-tree, cấu trúc WUNList kế thừa khả năng nén dữ liệu này, đồng thời cung cấp cơ chế truy xuất trực tiếp các thông tin cần thiết để tính độ hỗ trợ trọng số hữu ích của mẫu mà không cần phải duyệt lại cây

hiều lần trong quá trình khai thác. Với cấu tạo này, độ dài WUNList thường nhỏ hơn đáng kể so với cấu trúc Tidset. Một đặc điểm quan trọng của WUNList là khả năng xây dựng trực tiếp WUNList của mẫu mới thông qua phép kết hợp các WUNList hiện có mà không cần quét lại CSDL. Trong quá trình này, chỉ các phần tử thỏa mãn điều kiện mới được giữ lại, nên kích thước WUNList sẽ giảm dần sau mỗi lần kết hợp, giúp giảm chi phí tính toán khi mở rộng mẫu.

Thuật toán 2.3. Tạo WUNList mới
Input: WUNList của FWUP P_x $wunl_1$ và WUNList của FWUP P_y $wunl_2$
Output: WUNList của mẫu P_{xy}
Procedure Construct_WUNList ($wunl_1, wunl_2$) 1. $wunl_3 \leftarrow \emptyset, SC \leftarrow 0$ 2. $i \leftarrow 0, j \leftarrow 0$ 3. $CR_1 = wus(P_x), CR_2 = wus(P_y)$ 4. while $i < wunl_1 $ and $j < wunl_2 $ do 5. if $wunl_2[j].pre < wunl_1[i].pre$ then 6. if $wunl_2[j].post > wunl_1[i].post$ then 7. if $ wunl_3 > 0$ and $wunl_3[wunl_3 - 1].pre = wunl_2[j].pre$ then 8. $wunl_3[wunl_3 - 1].weight += wunl_1[i].weight$ 9. else 10. $wunl_3 \leftarrow (wunl_2[j].pre, wunl_2[j].post, wunl_1[i].weight)$ 11. $SC += wunl_1[i].weight, CR_1 -= wunl_1[i].weight; i++$ 12. else $CR_2 -= wunl_2[j].weight, j++$ 13. else $CR_1 -= wunl_1[i].weight, i++$ 14. if $SC + \min(CR_1, CR_2) < (minwus \times sumtwu)$ then return NULL 15. return $wunl_3$

Hình 2.5. Thuật toán tạo WUNList của mẫu mới.

Tuy nhiên, việc tạo và kết hợp các WUNList mới vẫn là thao tác tốn thời gian, nên cần có chiến lược để loại bỏ sớm những trường hợp không cần thiết phải xây dựng WUNList mới. Trên cơ sở đó, định lý sau được đề xuất nhằm giảm số lượng

phép kết hợp WUNList phải thực hiện trong quá trình khai thác.

Định lý 2.3. Cho một mẫu P và một mục i ($i \notin P$), $wus(P) = wus(P \cup \{i\})$, khi đó, với mọi mẫu X thoả mãn $(X \cap P) = \emptyset \wedge i \notin X$, thì $wus(X \cup P) = wus(X \cup P \cup \{i\})$.

Chứng minh. Gọi T_1 là tập các giao dịch có chứa mẫu P và T_2 là tập các giao dịch có chứa cả P và $\{i\}$, ta có, $T_2 \subseteq T_1$, nên

$$\sum_{t_i \in T_1} twu(t_i) \geq \sum_{t_j \in T_2} twu(t_j).$$

Dựa vào **Định nghĩa 2.4**, ta có $wus(P) \geq wus(P \cup \{i\})$, đẳng thức xảy ra khi và chỉ khi $T_1 = T_2$. Do đó, nếu $wus(P) = wus(P \cup \{i\})$ thì giao dịch có chứa P cũng phải chứa mục i .

Gọi T_3 là tập các giao dịch có chứa $X \cup P$. T_3 chứa P nên T_3 cũng phải chứa mục i , ta có:

$$\sum_{t_i \in T_3(X \cup P)} twu(t_i) = \sum_{t_j \in T_3(X \cup P \cup \{i\})} twu(t_j).$$

Do đó, $wus(X \cup P) = wus(X \cup P \cup \{i\})$. Định lý được chứng minh.

Ví dụ 2.13: Từ Ví dụ 2.11, ta có $wunl(AC) = \{(1, 5, 3.02)\}$, $wunl(AE) = \{(2, 3, 2.35)\}$ và $sumtwu = 3.89$. Vì $wus(A) = wus(AC) = 3.02/3.89$, bằng cách áp dụng **Định lý 2.3**, ta có thể xác định $wus(AEC) = wus(AE) = 2.35/3.89$ mà không cần phải tính WUNList của mẫu AEC .

Định lý 2.3 cho phép ta có thể nhanh chóng tính wus của các mẫu mà không cần phải tạo WUNList của các mẫu đó, giúp tăng tốc quá trình khai thác và giảm không gian tìm kiếm.

2.3.2.2 Mô tả thuật toán

Thuật toán dFWUNL sử dụng cây liệt kê tập hợp (set-enumerations-tree) [62] kết hợp với các định nghĩa và định lý được trình bày ở trên để khai thác FWUP từ

$dQDB$. Trong quá trình duyệt cây liệt kê tập hợp, các nhánh sẽ có chung tiền tố. Do đó, ta có thể xác định k -FWUP được tạo thành từ hai $(k-1)$ -FWUP. Với đặc tính này, để tiết kiệm bộ nhớ, thuật toán không cần lưu lại toàn bộ mẫu mà chỉ lưu trữ danh sách các mục cuối cùng và danh sách tiền tố chung. Ví dụ, với lớp mẫu $\{CEA, CED\}$, thuật toán chỉ lưu tiền tố CE và danh sách $\{A, D\}$ thay vì lưu trữ toàn bộ mẫu như trên. Thuật toán dFWUNL được mô tả trong Hình 2.6.

Trong giai đoạn đầu tiên của thuật toán dFWUNL, thủ tục *Build_WUN-tree* được gọi để tạo I_1 và WUN-tree (dòng 1), và các mục trong I_1 được thêm vào tập kết quả *FWUPs* (dòng 2). Tiếp theo, WUNList của các mục trong I_1 được tạo ra bằng cách duyệt WUN-tree (dòng 3). Một mảng $F2$ được sử dụng để lưu trữ trọng số hữu ích của 2-pattern được tính toán bởi thuật toán ở các dòng 4-8. Sau đó, tập I_1 được sử dụng để tạo 2-FWUP bằng cách kiểm tra mảng $F2$ (dòng 12), so sánh các giá trị wus của 2-pattern với $minwus$ và với giá trị của mục hiện tại để thêm vào tập S hoặc để cập nhật L (dòng 13-18). *Prefix* và S được sử dụng để tạo FWUP mà không cần tính toán WUNList bằng thủ tục *Find_FWUP_sameWUS* (dòng 19). Mỗi tập con của S được tạo ra và kết hợp với *Prefix* để tạo thành FWUP (dòng 38-40). Tiếp theo, thủ tục *Find_FWUP* tính toán và sử dụng $Prefix_{next}$, L_{next} và S_{next} để tìm FWUP trong lớp tiếp theo bằng cách tìm kiếm đệ quy (dòng 22, 24, 31, 33-35). Hàm *Construct_WUNList* được sử dụng để tạo ứng viên C trong lớp $(k+1)$ từ hai mẫu tương ứng trong lớp k (L_k). Nếu ứng viên đó là FWUP thì C được kiểm tra để xác định được thêm vào S_{next} hay L_{next} (dòng 27-32). Sau đó, thủ tục *Find_FWUP_sameWUS* được gọi để tạo FWUP từ S_{next} và ta gọi đệ quy *Find_FWUP* để tìm các FWUP tiếp theo (dòng 33-34).

Thuật toán 2.4. Thuật toán dFWUNL
Input: CSDL định lượng động $dQDB$ và ngưỡng $minwus$
Output: Tập tất cả các mẫu trọng số hữu ích phổ biến <i>FWUPs</i>
1. Gọi Build_WUN-tree($dQDB$, $minwus$) để tạo I_1 và WUN-tree 2. $FWUPs \leftarrow I_1$

```

3. Duyệt WUN-tree để tạo WUNList của các mục trong  $I_1$ 
4. Khởi tạo  $F2[|I_1|][|I_1|]$ 
5. Duyệt WUN-tree theo phép duyệt tiên thứ tự pre-order
6.   foreach nút  $N$  lưu mục  $i_x$  do
7.     foreach nút tổ tiên  $N_a$  của  $N$  lưu mục  $i_y$  do
8.        $F2[i_x][i_y] += N.weight$ 
9. for  $x = |I_1| - 1$  downto 1 do
10.   $Prefix_2 \leftarrow \{i_x\}, L_2 \leftarrow \emptyset, S_2 \leftarrow \emptyset$ 
11.  for  $y = 0$  to  $x - 1$  do
12.     $wus(i_{xy}) \leftarrow F2[i_x][i_y]/sumtwu$ 
13.    if  $wus(i_{xy}) = wus(i_x)$  then  $S_2 \leftarrow i_y$ 
14.    else if  $wus(i_{xy}) \geq minwus$  then
15.       $i_{xy} \leftarrow i_y$ 
16.       $wunl(i_{xy}) \leftarrow \text{Construct\_WUNList}(wunl(i_x), wunl(i_y))$ 
17.       $L_2 \leftarrow i_{xy}$ 
18.       $FWUPs \leftarrow i_x i_y$ 
19.    if  $|S_2| > 0$  then  $\text{Find\_FWUP\_sameWUS}(Prefix_2, S_2)$ 
20.    if  $|L_2| > 0$  then  $\text{Find\_FWUP}(Prefix_2, L_2, S_2)$ 
21. return  $FWUPs$ 

```

Procedure Find_FWUP ($Prefix_k, L_k, S_k$)

```

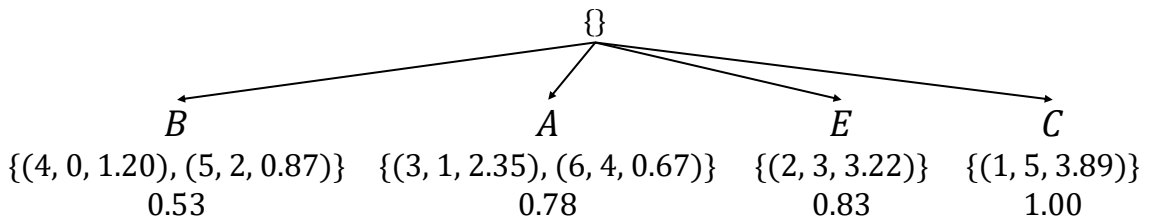
22.  $Prefix_{next} \leftarrow Prefix_k$ 
23. for  $i = |L_k| - 1$  downto 1 do
24.   $Prefix_{next} \leftarrow L_k[i], L_{next} \leftarrow \emptyset, S_{next} \leftarrow S_k$ 
25.  for  $j = 0$  to  $i - 1$  do
26.     $C \leftarrow L_k[j]$ 
27.     $wunl(C) \leftarrow \text{Construct\_WUNList}(wunl(L_k[i]), wunl(L_k[j]))$ 
28.    if  $wunl(C) \neq NULL$  then
29.      if  $wus(C) = wus(L_k[i])$  then  $S_{next} \leftarrow L_k[j]$ 
30.    else

```

31.	$L_{next} \leftarrow C$
32.	$FWUPs \leftarrow \{Prefix_k \cup L_k[i] \cup L_k[j]\}$
33.	if $ S_{next} > 0$ then Find_FWUP_sameWUS ($Prefix_{next}, S_{next}$)
34.	if $ L_{next} > 0$ then Find_FWUP ($Prefix_{next}, L_{next}, S_{next}$)
35.	Xoá mục cuối cùng của $Prefix_{next}$
36.	$Prefix_{next} \leftarrow L_k[0]$
37.	if $ S_k > 0$ then Find_FWUP_sameWUS ($Prefix_{next}, S_k$)
Procedure Find_FWUP_sameWUS ($Prefix, S$)	
38.	Sinh tập $Child$ lưu tất cả tập con của S
39.	foreach $s_i \in Child$ do
40.	$FWUPs \leftarrow \{Prefix \cup s_i\}$

Hình 2.6. Thuật toán dFWUNL.

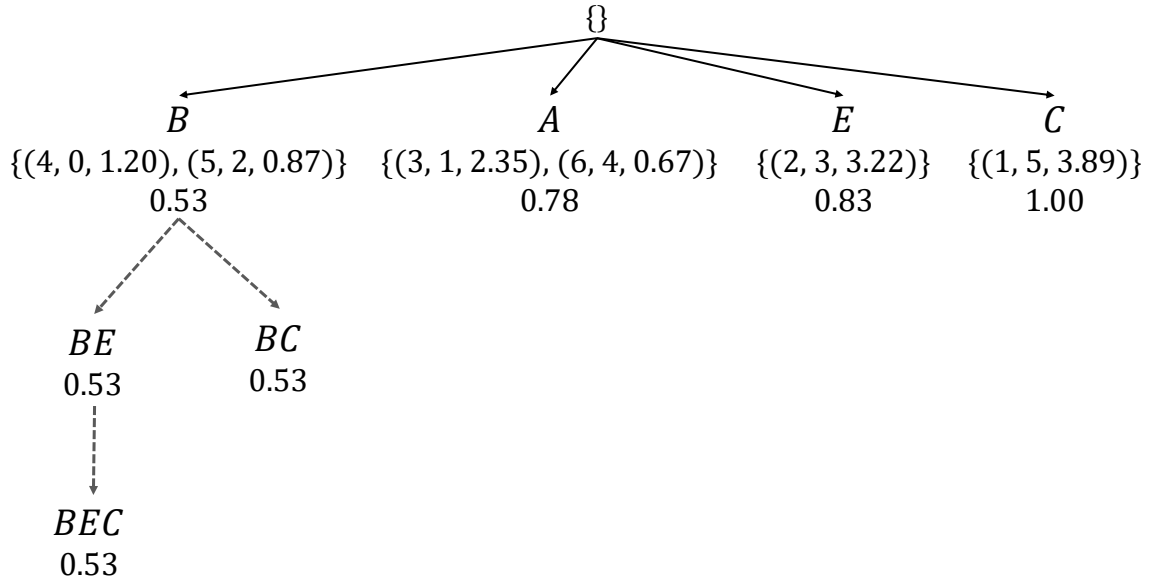
Ví dụ 2.14: Với $dQDB_E$ trong Bảng 2.2 và $minwus = 0.5$, thuật toán dFWUNL đầu tiên gọi thủ tục *Build_WUN-tree* để xây dựng WUN-tree (Hình 2.4); 1-FWUPs $I_1 = \{C, E, A, D\}$ và $sumtwu = 3.89$. I_1 được thêm vào tập kết quả: $FWUPs = \{C, E, A, B\}$. Tiếp theo, WUNList của các mục trong I_1 được tạo từ WUN-tree (Hình 2.7). Quá trình khai thác được tiến hành như sau.



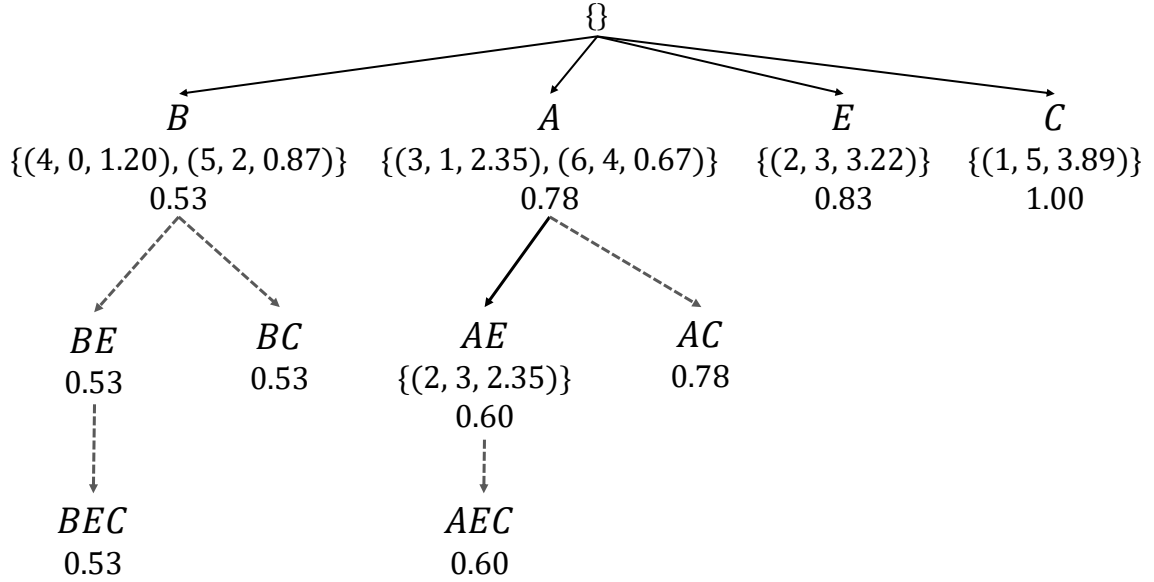
Hình 2.7. I_1 và WUNList của các 1-FWUP từ $dQDB_E$ với $minwus = 0.5$.

Xét nhánh B : $Prefix_2 = \{B\}$, $S_2 = \emptyset$, $L_2 = \emptyset$, ta có: $wus(BA) = F2(BA)/3.89 = 1.20/3.89 < 0.5$ (không thỏa mãn); $wus(BE) = F2(BE)/3.89 = 2.07/3.89 = wus(B)$ nên E được thêm vào S_2 ; $wus(BC) = F2(BC)/3.89 = 2.07/3.89 = wus(B)$ nên C được thêm vào S_2 ; Vì $S_2 = \{E, C\} \neq \emptyset$, ta gọi thủ tục *Find_FWUP_sameWUS*($Prefix_2 = \{B\}, S_2 = \{E, C\}$) để cập nhật $FWUPs = \{C, E, A, B, BE, BC, BEC\}$. Kết quả của bước này được hiển thị ở Hình 2.8. Các nút có

mũi tên nét đứt hướng tới thể hiện được tạo từ thủ tục *Find_FWUP_sameWUS* mà không cần tạo WUNList.



Hình 2.8. Kết quả sau khi kết hợp *B* với các 1-FWUP.



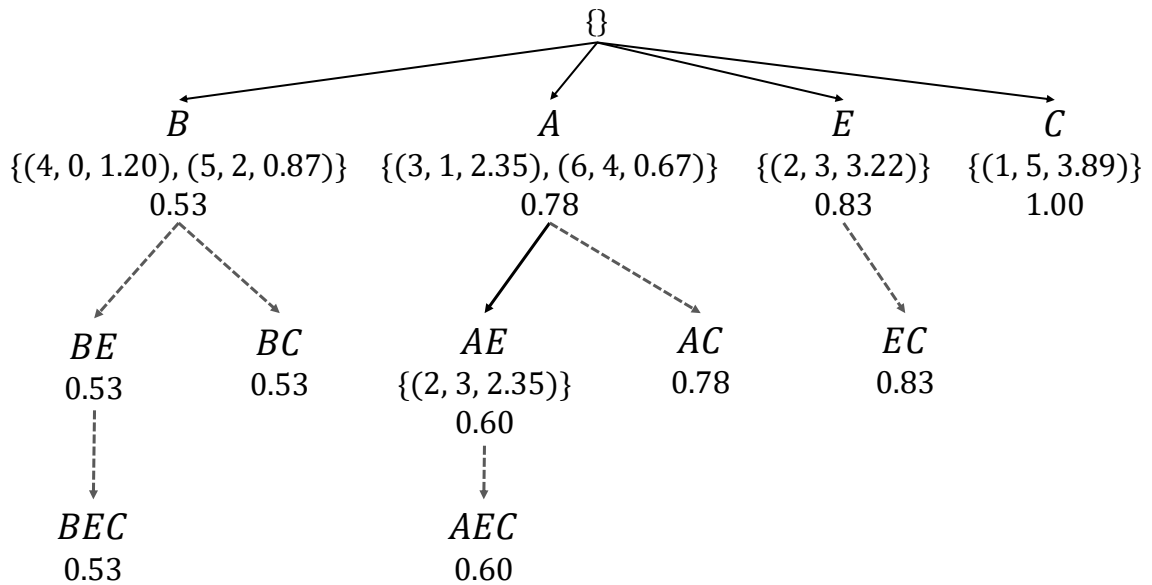
Hình 2.9. Kết quả sau khi kết hợp *A* với các 1-FWUP.

Xét nhánh *A*: $Prefix_2 = \{A\}$, $S_2 = \emptyset$, $L_2 = \emptyset$, ta có $wus(AE) = F2(AE)/3.89 = 2.35/3.89 > 0.5$ và $wus(AE) \neq wus(A)$ nên *E* được thêm vào L_2 , $wunl(AE) = \{(2, 3, 2.35)\}$, và thêm *AE* vào *FWUPs*. $wus(AC) = F2(AC)/3.89 =$

$3.02/3.89 = wus(A)$ nên C được thêm vào S_2 ; Vì $S_2 = \{D\} \neq \emptyset$, ta gọi thủ tục $Find_FWUP_sameWUS(Prefix_2 = \{A\}, S_2 = \{C\})$ để cập nhật $FWUPs = \{C, E, A, B, BE, BC, BEC, AE, AC\}$. Mặt khác, $L_2 = \{E\} \neq \emptyset$, nên ta gọi đệ quy $Find_FWUP(Prefix_2 = \{A\}, L_2 = \{E\}, S_2 = \{C\})$. Vì L_2 chỉ có một phần tử nên thủ tục $Find_FWUP_sameWUS(\{AE\}, \{C\})$ được gọi để sinh các FWUP và cập nhật $FWUPs = \{C, E, A, B, BE, BC, BEC, AE, AC, AEC\}$ (Hình 2.9).

Xét nhánh E : $Prefix_2 = \{E\}$, $S_2 = \emptyset$, $L_2 = \emptyset$, ta có $wus(EC) = F2(EC)/3.89 = 3.22/3.89 = wus(E)$ nên C được thêm vào S_2 . Vì $S_2 = \{C\} \neq \emptyset$, nên ta gọi $Find_FWUP_sameWUS(Prefix_2 = \{E\}, S_2 = \{C\})$ để cập nhật $FWUPs = \{C, E, A, B, BE, BC, BEC, AE, AC, AEC, EC\}$.

Thuật toán kết thúc, ta thu được 11 FWUP đó là: $FWUPs = \{C, E, A, B, BE, BC, BEC, AE, AC, AEC, EC\}$. Cây liệt kê tập hợp của quá trình khai thác được mô tả ở Hình 2.10.



Hình 2.10. Tất cả FWUP được khám phá bởi thuật toán dFWUNL từ $dQDB_E$ với $minwus = 0.5$.

2.3.3 Tính đúng đắn và độ phức tạp thuật toán

Để đánh giá khả năng khai thác FWUP trên CSDL định lượng động, phần này

tiến hành phân tích các khía cạnh lý thuyết của hai thuật toán dFWUT và dFWUNL, bao gồm tính đúng đắn, tính đầy đủ, độ phức tạp tính toán cũng như đặc điểm hiệu năng của từng phương pháp. Mặc dù hai thuật toán sử dụng các cấu trúc dữ liệu và chiến lược khai thác khác nhau, nhưng cả hai đều được xây dựng dựa trên tính chất phản đơn điệu nhằm giảm không gian tìm kiếm và nâng cao hiệu quả khai thác.

Đầu tiên, tính đúng đắn của hai thuật toán được đảm bảo dựa trên mối quan hệ phản đơn điệu của giá trị wus . Cụ thể, với hai mẫu $X \subset Y$, luôn tồn tại quan hệ $wus(Y) \leq wus(X)$. Điều này cho thấy nếu một mẫu X không thỏa điều kiện $wus(X) \geq minwus$ thì mọi siêu mẫu của X cũng không thể là FWUP. Do đó, việc loại bỏ các mẫu không đạt ngưỡng trong quá trình khai thác là hoàn toàn an toàn và không làm mất đi bất kỳ mẫu FWUP nào. Dựa trên nguyên lý này, thuật toán dFWUT thực hiện cắt tía không gian tìm kiếm thông qua việc kiểm tra giá trị wus của các mẫu được sinh ra từ phép kết hợp Tidset. Việc xây dựng Tidset bằng phép giao đảm bảo rằng tập giao dịch chứa mẫu X được xác định chính xác, từ đó giá trị $wus(X)$ được tính đúng theo định nghĩa. Trong khi đó, thuật toán dFWUNL sử dụng cấu trúc WUN-tree và WUNList để nén toàn bộ CSDL mà không làm mất mát thông tin. Với mỗi mẫu X , WUNList của X lưu trữ đầy đủ thông tin về quan hệ tổ tiên giữa các nút trên cây và các giao dịch chứa mẫu thông qua các WUN-code, đồng thời tính toán chính xác giá trị $wus(X)$. Trong cả hai trường hợp trên, chỉ các mẫu có giá trị wus không nhỏ hơn ngưỡng $minwus$ mới tiếp tục được khai thác, do đó mọi mẫu được sinh ra đều là FWUP hợp lệ.

Bên cạnh tính đúng đắn, hai thuật toán cũng đảm bảo tính đầy đủ trong quá trình khai thác. Cả dFWUT và dFWUNL đều bắt đầu từ tập các mục đơn thỏa ngưỡng $minwus$ và mở rộng dần để tạo ra các mẫu có kích thước lớn hơn. Quá trình mở rộng được thực hiện có hệ thống dựa trên thứ tự của các mục nhằm đảm bảo mọi tổ hợp hợp lệ đều được duyệt qua. Do hai thuật toán duyệt toàn bộ không gian mở rộng hợp lệ và không loại bỏ bất kỳ mẫu nào trước khi kiểm tra điều kiện wus , nên mọi FWUP đều sẽ được phát hiện trong quá trình khai thác. Đồng thời, cơ chế mở rộng theo thứ

tự cũng giúp mỗi mẫu chỉ được sinh đúng một lần, tránh hiện tượng trùng lặp trong không gian tìm kiếm. Đặc biệt, thuật toán dFWUNL còn tích hợp chiến lược tối ưu hóa dựa trên mẫu có cùng giá trị wus bằng thủ tục $Find_FWUP_sameWUS$, giúp giảm bớt việc mở rộng và tính toán WUNList dư thừa.

Về độ phức tạp tính toán, giả sử CSDL gồm n mục và m giao dịch với độ dài tối đa của mỗi giao dịch là l_{max} . Thuật toán dFWUT thực hiện hai giai đoạn chính là giai đoạn khởi tạo và giai đoạn khai thác đệ quy. Trong giai đoạn khởi tạo, $dQDB$ được quét để tính toán các giá trị twu , wus và Tidset của từng mục, độ phức tạp của giai đoạn này là $O(m \cdot l_{max})$. Sau khi lọc tập 1-FWUP C_1 ban đầu, thuật toán tiến hành đệ quy khai thác thông qua xây dựng cây dWUPT-tree và phép giao Tidset. Vì thuật toán phải duyệt hết các FWUP nên số lượng mẫu có thể tạo ra trong trường hợp xấu nhất lên tới $2^{|C_1|} - 1$. Với mỗi phép kết hợp, thuật toán tiến hành giao Tidset để tạo mẫu mới với chi phí là $O(ts_{max})$, trong đó ts_{max} là độ dài tối đa của Tidset. Do đó độ phức tạp thời gian của thuật toán dFWUT là $O(m \cdot l_{max} + ts_{max} \cdot 2^{|C_1|})$, hay $O(2^{|C_1|})$ mang bản chất lũy thừa theo số lượng mục. Khi CSDL có mật độ thấp, kích thước Tidset thường nhỏ, giúp giảm đáng kể chi phí giao dữ liệu, cải thiện hiệu năng khai thác. Ngược lại, đối với các CSDL dày đặc hoặc có số lượng giao dịch lớn, kích thước Tidset tăng lên đáng kể, dẫn đến chi phí bộ nhớ và thời gian xử lý tăng theo.

Khác với dFWUT, thuật toán dFWUNL sử dụng cấu trúc WUN-tree kết hợp với WUNList nhằm giảm chi phí tạo mẫu mới và tối ưu hóa quá trình tính toán. Ở giai đoạn đầu tiên, quy trình xây dựng cây WUN-tree đòi hỏi phải quét toàn bộ CSDL, trong đó chi phí tính toán chủ đạo phát sinh từ việc sắp xếp các mục trong mỗi giao dịch và chèn chúng vào cây, dẫn đến độ phức tạp thời gian là $O(m \cdot l_{max} \log l_{max})$. Trong giai đoạn khai thác, cần trên lý thuyết của không gian tìm kiếm vẫn được xác định bởi tổng số tập con có thể sinh ra từ I_1 là tập các mục là FWUP, tức là $2^{|I_1|}$. Tại mỗi nút của cây tìm kiếm này, thuật toán thực hiện các phép toán để tạo WUNList mới với độ phức tạp là $O(wunl_{max})$, trong đó $wunl_{max}$ là độ dài tối đa của một WUNList. Do đó, độ phức tạp của giai đoạn này là $O(wunl_{max} \cdot 2^{|I_1|})$. Vì hàm mũ là

rất lớn nên độ phức tạp của thuật toán dFWUNL trong trường hợp xấu nhất cũng là $O(2^{|I_1|})$. Cần lưu ý rằng độ phức tạp $O(2^{|I_1|})$ là một đặc trưng cố hữu của các bài toán khai thác mẫu chính xác. Tuy nhiên, nhờ cơ chế nén dữ liệu vượt trội nên kích thước WUNList thường nhỏ hơn đáng kể so với Tidset, đặc biệt trên các CSDL có mật độ cao. Điều này giúp dFWUNL giảm số lượng phép tính dư thừa và cải thiện đáng kể hiệu năng thực thi so với phương pháp dựa trên Tidset. Hơn nữa, nhờ ma trận $F2$ lưu trữ trọng số của các cặp mục, dFWUNL có khả năng cắt tia sớm các nhánh không tiềm năng ở mức 2-pattern với chi phí chỉ là $O(1)$.

Các phân tích về mặt lý thuyết cho thấy thuật toán dFWUT ưu tiên tính đơn giản và khả năng triển khai trực tiếp dựa trên Tidset, còn dFWUNL hướng đến tối ưu hiệu năng thông qua cấu trúc dữ liệu lai và chiến lược khai thác giảm sinh ứng viên dư thừa. Thuật toán dFWUT phù hợp hơn với các CSDL thưa và quy mô vừa, trong khi dFWUNL cho thấy ưu thế rõ rệt trên các CSDL lớn và dày đặc nhờ khả năng nén dữ liệu và mở rộng không gian tìm kiếm hiệu quả hơn.

2.3.4 Thực nghiệm và kết quả

2.3.4.1 Thiết lập thực nghiệm

Phần này tiến hành các thực nghiệm để so sánh hai thuật toán được đề xuất, dFWUT và dFWUNL, và một phiên bản sửa đổi của thuật toán WUN-Miner [5] (gọi là WUN-Miner-Mod) để khai thác FWUP từ CSDL định lượng động về thời gian chạy và bộ nhớ sử dụng. WUN-Miner được lựa chọn làm thuật toán đối sánh do đây là phương pháp hiệu quả để khai thác FWUP trên CSDL định lượng truyền thống. Để WUN-Miner có thể hoạt động trên CSDL định lượng động, bảo đảm khả năng so sánh trong cùng điều kiện, hai sửa đổi quan trọng đã được thêm vào như sau: (1) Chỉnh sửa quy trình đọc dữ liệu từ CSDL truyền thống sang CSDL định lượng động; (2) Sử dụng định nghĩa mới để tính toán giá trị trọng số hữu ích giao dịch dành cho trọng số động được trình bày trong phần trước. Tất cả các chỉnh sửa trong thuật toán WUN-Miner-Mod đảm bảo giữ nguyên cấu trúc dữ liệu, quy trình khai thác và không làm thay đổi độ phức tạp của thuật toán gốc. Các thực nghiệm được thực hiện trên

cùng một máy tính chạy hệ điều hành Windows 10 với bộ xử lý Intel Core i3-2310M 2.10 GHz và RAM 8 GB. Tất cả các thuật toán thực nghiệm đều được triển khai bằng ngôn ngữ lập trình Visual C# 2017 (.NET framework 4.5).

2.3.4.2 Cơ sở dữ liệu thực nghiệm

Bảng 2.6. Các đặc điểm của CSDL thực nghiệm

CSDL	Số lượng mục	Số lượng giao dịch	Độ dài trung bình giao dịch	Mật độ (%)	Số lượng giao dịch mỗi lô
Chess	76	3,196	37	49.33	200
Pumsb	2,113	49,046	74	3.50	3,000
Connect	129	67,557	43	33.33	6,000
Retail	16,470	88,162	10.3	0.06	10,000
Accidents	468	340,183	33.8	7.22	5,000
BMS-POS	1,657	515,597	6.5	0.39	50,000
Kosarak	41,270	990,002	8.1	0.02	100,000
Chainstore	46,086	1,112,949	7	0.02	80,000
Susy	190	5,000,000	19	10	200,000
CI10K	10,000	703,365	3	0.03	80,000
CI20K	20,000	911,503	5	0.03	80,000
CI30K	30,000	976,722	6	0.02	80,000
CI40K	40,000	1,085,267	6	0.02	80,000
CI45K	45,000	1,108,718	7	0.02	80,000
CT300K	46,086	300,000	7	0.02	80,000
CT500K	46,086	500,000	7	0.02	80,000
CT700K	46,086	700,000	7	0.02	80,000
CT900K	46,086	900,000	7	0.02	80,000
CT1100K	46,086	1,100,000	7	0.02	80,000

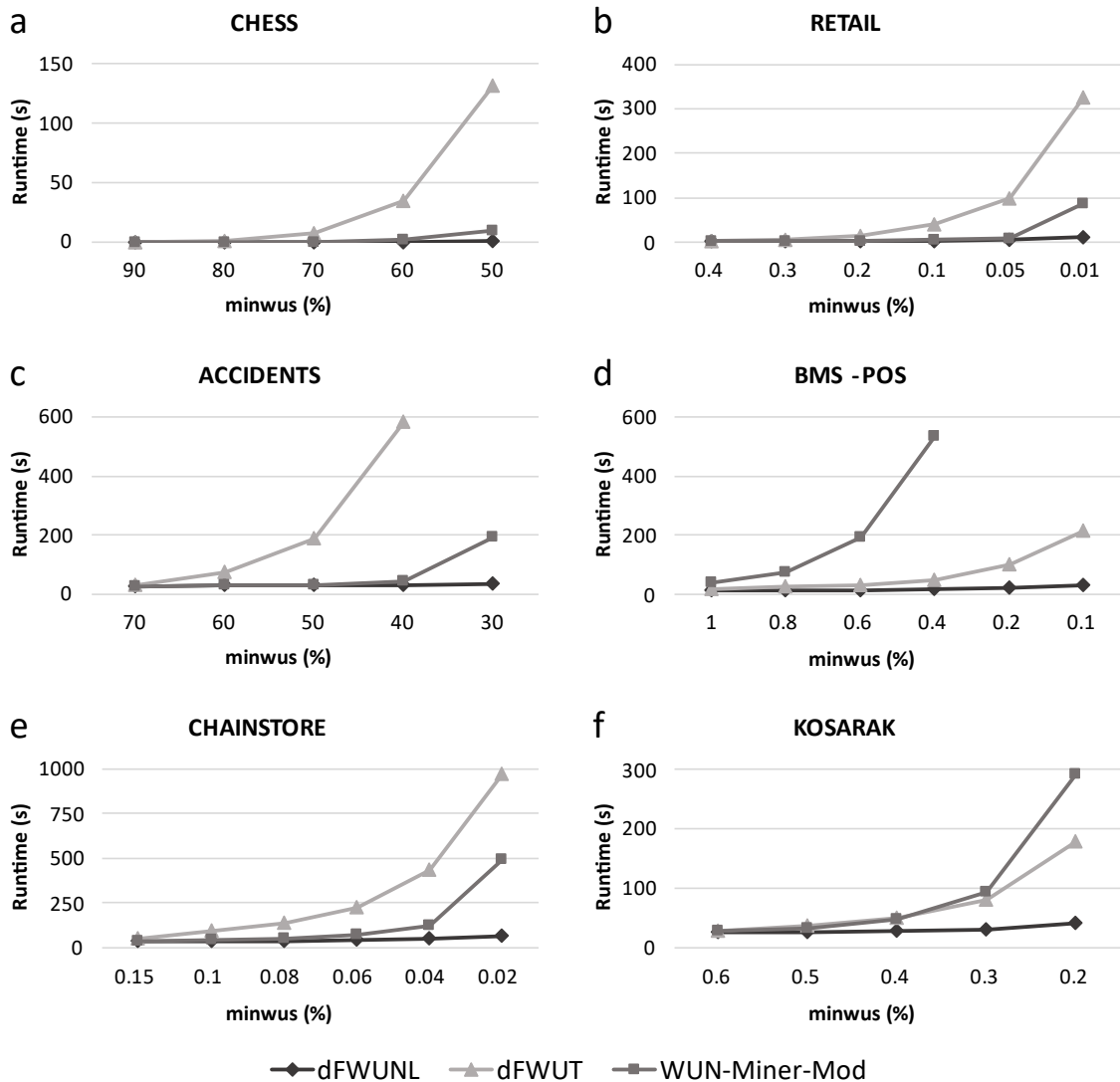
Bảng 2.6 hiển thị thông tin của các CSDL thực nghiệm phổ biến trong cộng đồng nghiên cứu khai thác dữ liệu. Các bộ dữ liệu Chess, Pumsb, Connect, Retail,

Accident, BMS-POS và Kosarak được tải xuống từ FIMI [22], còn Chainstore và Susy được lấy từ thư viện mã nguồn mở SPMF [19]. Các CSDL chuẩn được sử dụng trong thực nghiệm ban đầu không chứa thông tin về trọng số động. Vì vậy, để có được CSDL định lượng động và không mất tính tổng quát, mỗi CSDL thực nghiệm được chia thành nhiều lô. Mỗi lô có số lượng giao dịch bằng nhau, ngoại trừ lô cuối cùng chứa các giao dịch còn lại. Mỗi lô giao dịch tương ứng với một tập trọng số được tạo ngẫu nhiên theo phân phối đều trong khoảng [1,10]. Khi chuyển sang lô tiếp theo, trọng số được sinh lại, qua đó mô phỏng sự thay đổi mức độ quan trọng của các mục theo thời gian. Cùng một tập trọng số được sử dụng cho tất cả các thuật toán trong mỗi lần thực nghiệm nhằm bảo đảm tính nhất quán khi so sánh. Ví dụ, CSDL Chess có 3196 giao dịch được chia thành 16 lô. 15 lô đầu tiên đều có 200 giao dịch mỗi lô và lô cuối cùng là 196 giao dịch còn lại. Thông tin này được hiển thị ở cột cuối cùng trong Bảng 2.6. Ngoài ra, bộ dữ liệu Chainstore được trích xuất để đánh giá khả năng mở rộng của các thuật toán. Các bộ dữ liệu CI10K, CI20K, CI30K, CI40K và CI45K bao gồm các mục có chỉ số không quá 10000, 20000, 30000, 40000 và 45000. Trong khi đó, các bộ dữ liệu CT300K, CT500K, CT700K, CT900K và CT1100K đại diện cho 300000, 500000, 700000, 900000 và 1100000 giao dịch đầu tiên trong bộ dữ liệu Chainstore động.

2.3.4.3 So sánh về thời gian chạy

Hình 2.11 hiển thị thời gian chạy của thuật toán dFWUT, dFWUNL và WUN-Miner-Mod trên các CSDL thực nghiệm với các giá trị *minwus* khác nhau. Lưu ý, có một số hình loại bỏ một số giá trị thời gian chạy của thuật toán WUN-Miner-Mod và dFWUT ở một số ngưỡng vì chúng quá lớn. Các giá trị này làm cho việc so sánh sự khác biệt về thời gian chạy giữa các thuật toán trở nên khó khăn. Kết quả cho thấy thuật toán dFWUNL nhanh hơn thuật toán dFWUT và WUN-Miner-Mod trên tất cả các bộ dữ liệu. Với ngưỡng *minwus* lớn, sự khác biệt về thời gian chạy giữa ba thuật toán là không nhiều. Ví dụ, trên CSDL Kosarak, với *minwus* = 0.6%, thời gian khai thác của các thuật toán dFWUT, dFWUNL và WUN-Miner-Mod lần lượt là 28.3 giây, 24.6 giây và 26.5 giây. Trên bộ dữ liệu Accidents, với *minwus* = 70%,

dFWUT mất 31.2 giây trong khi dFWUNL mất 27.8 giây và WUN-Miner-Mod mất 27.9 giây. Đặc biệt, khi *minwus* càng nhỏ, thuật toán dFWUNL càng cho thấy sự vượt trội so với thuật toán dFWUT và WUN-Miner-Mod. Ví dụ, trên bộ dữ liệu Kosarak, với *minwus* = 0.2%, thời gian khai thác của thuật toán dFWUNL là 40.6 giây, trong khi thời gian khai thác của thuật toán dFWUT là 177 giây và thuật toán WUN-Miner-mod là 292.2 giây. Tương tự, trên CSDL Accidents, với *minwus* = 40%, thuật toán dFWUNL mất 31.4 giây, dFWUT mất 584.5 giây và WUN-Miner-Mod mất 44.8 giây.



Hình 2.11. Thời gian chạy của các thuật toán dFWUT, dFWUNL và WUN-Miner-Mod trên các bộ dữ liệu thực nghiệm.

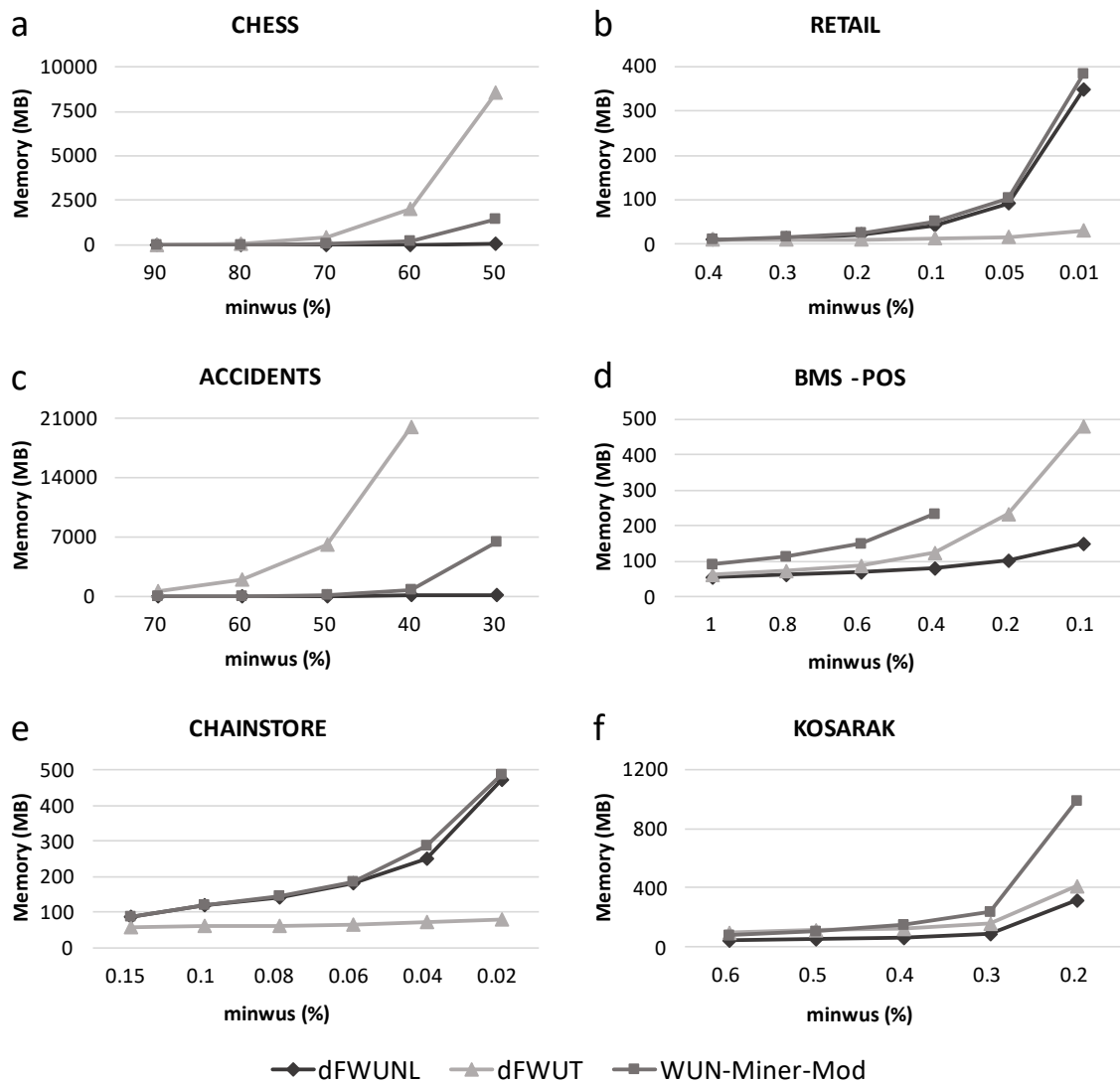
Các kết quả này được giải thích như sau. Thuật toán dFWUNL nhanh hơn WUN-Miner-Mod vì cấu trúc WUNList hiệu quả hơn WUN-set và dFWUNL có một số cải tiến để cải thiện hiệu suất thời gian khai thác. Điểm nổi bật của thuật toán dFWUT là chỉ quét CSDL một lần để tạo Tidset và sử dụng Tidset để khai thác, trong khi thuật toán dFWUNL cần quét CSDL hai lần để tạo cấu trúc WUN-tree và WUNList. Tuy nhiên, cấu trúc WUNList có khả năng nén dữ liệu tốt, giúp quá trình tìm kiếm FWUP nhanh hơn so với phương pháp Tidset. Ngoài ra, thuật toán dFWUNL cũng áp dụng các cải tiến như tính toán trước hỗ trợ tiện ích có trọng số của các 2-pattern, nhanh chóng tìm ra các mẫu có cùng độ hỗ trợ trọng số hữu ích mà không cần phải tính toán WUNList. Do đó, thuật toán dFWUNL vượt trội đáng kể so với thuật toán dFWUT để khai thác FWUP từ *dQDB* về mặt thời gian xử lý.

2.3.4.4 So sánh về bộ nhớ sử dụng

Hình 2.12 hiển thị mức sử dụng bộ nhớ của thuật toán dFWUNL và thuật toán dFWUT so với WUN-Miner-Mod trên các CSDL thực nghiệm. Từ kết quả ta rút ra hai kết luận. Kết luận đầu tiên là thuật toán dFWUNL tốt hơn một chút về mặt bộ nhớ so với WUN-Miner-Mod. Kết luận thứ hai là thuật toán dFWUNL sử dụng ít bộ nhớ hơn đáng kể so với thuật toán dFWUT trên hầu hết các bộ dữ liệu thực nghiệm như Accidents, Chess, BMS-POS và Kosarak. Tuy nhiên, trên một số CSDL rất thưa thớt như Retail và Chainstore, thuật toán dFWUNL lại sử dụng nhiều bộ nhớ hơn dFWUT.

Cấu trúc WUNList phức tạp hơn rất nhiều so với cấu trúc Tidset vì nó lưu trữ nhiều thông tin, nhưng WUN-tree cung cấp khả năng nén dữ liệu vượt trội, đặc biệt là trên các CSDL dày đặc. Do đó, độ dài của WUNList nhỏ hơn nhiều so với Tidset, cũng như so với WUN-set, vì vậy thuật toán dFWUNL sử dụng bộ nhớ hiệu quả hơn. Ví dụ, trên CSDL Accidents, với $minwus = 60\%$, mức sử dụng bộ nhớ của thuật toán dFWUT là 1872 MB, thuật toán WUN-Miner-Mod là 16.2 MB, trong khi thuật toán dFWUNL là 5.1 MB. Đối với các bộ dữ liệu thưa thớt, mức nén dữ liệu của WUN-tree bị giảm đi, dẫn đến cấu trúc WUNList sử dụng nhiều bộ nhớ hơn. Ví dụ, trên bộ dữ liệu Chainstore, với $minwus = 0.06\%$, thuật toán dFWUNL sử dụng

179.5 MB, thuật toán dFWUT sử dụng 65.5 MB và thuật toán WUN-Miner-Mod sử dụng 183.6 MB. Từ đó cho thấy, thuật toán dFWUNL sử dụng bộ nhớ tối ưu hơn trên hầu hết các CSDL thực nghiệm và chỉ kém hơn thuật toán dFWUT trên một số bộ dữ liệu thừa thớt. Để bổ sung số liệu định lượng cho các kết quả so sánh, Bảng 2.7 tổng hợp thời gian thực thi, mức sử dụng bộ nhớ và số lượng FWUP thu được của các thuật toán trên từng bộ dữ liệu tại các ngưỡng *minwus* đại diện.



Hình 2.12. Bộ nhớ sử dụng của các thuật toán dFWUT, dFWUNL và WUN-Miner-Mod trên các bộ dữ liệu thực nghiệm.

Bảng 2.7. Kết quả khai thác FWUP trên các bộ dữ liệu thực nghiệm

CSDL (<i>minwus</i>)	dFWUNL			dFWUT			WUN-Miner-Mod		
	Thời gian (s)	Bộ nhớ (MB)	FWUP	Thời gian (s)	Bộ nhớ (MB)	FWUP	Thời gian (s)	Bộ nhớ (MB)	FWUP
Chess (60%)	0.6	13	248164	34.8	2009	248164	1.8	184.1	248164
Retail (0.01%)	13.3	349	7715	326.0	31.7	7715	88.5	383.9	7715
Accidents (60%)	28.5	5.1	2082	73	1872	2082	28.7	16.2	2082
BMS-POS (0.4%)	16.6	79.7	6650	49.2	124.1	6650	535.1	235.3	6650
Kosarak (0.2%)	40.6	318.9	35217	177.0	410.6	35217	292.2	985.8	35217
Chainstore (0.04%)	47.5	251.2	5545	430.2	70.5	5545	123.7	285.4	5545

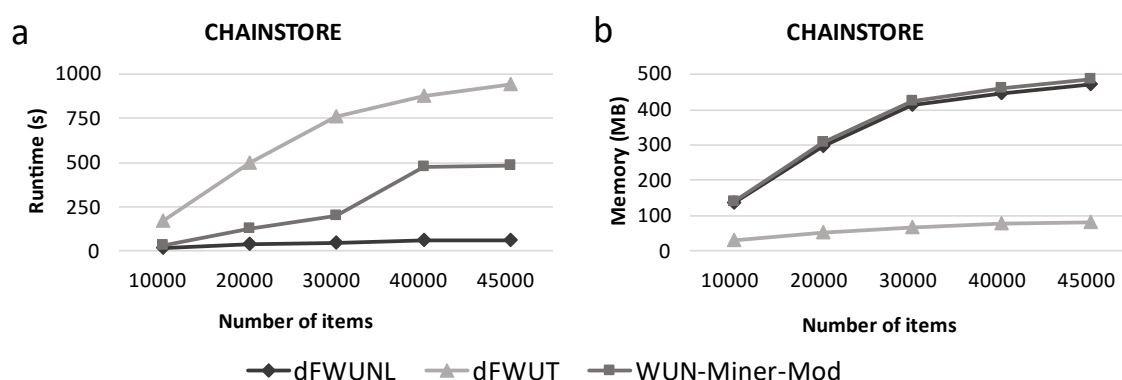
2.3.4.5 So sánh về khả năng mở rộng

Hình 2.13 hiển thị khả năng mở rộng về thời gian chạy và bộ nhớ của các thuật toán dFWUNL, dFWUT và WUN-Miner-Mod khi chạy trên các phiên bản tùy chỉnh của CSDL Chainstore theo số lượng mục (các bộ dữ liệu CI10K, CI20K, CI30K, CI40K và CI45K) với ngưỡng *minwus* = 0.02%. Các kết quả trong chỉ ra rằng thuật toán dFWUNL vượt trội so với dFWUT và WUN-Miner-Mod về khả năng mở rộng thời gian chạy. Cụ thể, khi số lượng mục tăng từ 10,000 lên 45,000, thời gian chạy của thuật toán dFWUNL tăng từ 19.5 giây lên 61.4 giây, trong khi thời gian chạy của WUN-Miner-Mod tăng từ 35.5 giây lên 481 giây, và dFWUT tăng từ 173.3 giây lên 943.5 giây. Về mặt bộ nhớ, mặc dù dFWUNL sử dụng nhiều bộ nhớ hơn so với dFWUT, nhưng tốc độ tăng bộ nhớ vẫn ở mức chấp nhận được.

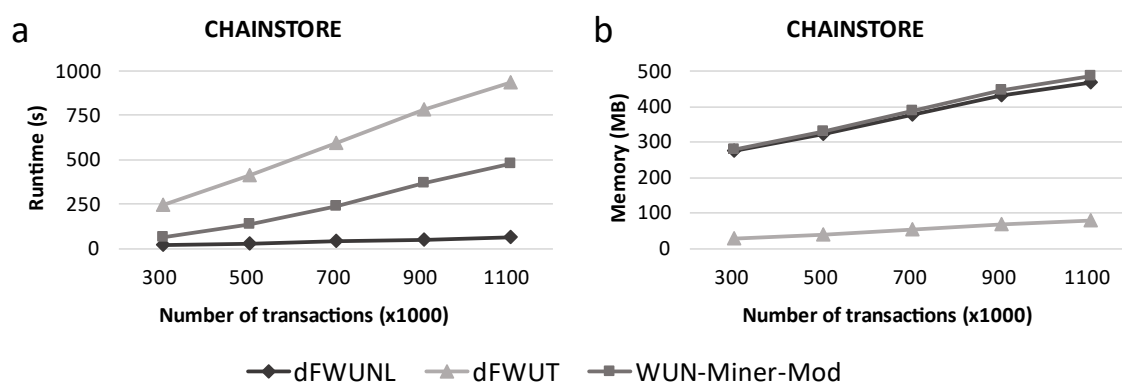
Hình 2.14 mô tả khả năng mở rộng của các thuật toán dFWUNL, dFWUT và WUN-Miner-Mod khi chạy trên các phiên bản tùy chỉnh của CSDL Chainstore theo số

lượng giao dịch (các bộ dữ liệu CT300K, CT500K, CT700K, CT900K và CT1100K) với ngưỡng $minws = 0.02\%$. Các kết quả cho thấy thuật toán dFWUNL hiệu quả hơn so với dFWUT và WUN-Miner-Mod về khả năng mở rộng thời gian chạy, trong khi tốc độ tăng bộ nhớ vẫn ở mức ổn định. Cụ thể, khi số lượng giao dịch tăng từ 300,000 lên 1,100,000, thời gian chạy của dFWUNL tăng từ 17.9 giây lên 61.2 giây, trong khi thời gian chạy của WUN-Miner-Mod tăng từ 64.3 giây lên 479.3 giây, và dFWUT tăng từ 244.8 giây lên 939 giây. Các số liệu cụ thể về kết quả thực nghiệm đánh giá khả năng mở rộng của các thuật toán được mô tả ở

Bảng 2.8.



Hình 2.13. Khả năng mở rộng về số lượng mục trên bộ dữ liệu Chainstore.



Hình 2.14. Khả năng mở rộng về số lượng giao dịch trên bộ dữ liệu Chainstore.

Bảng 2.8. Khả năng mở rộng của dFWUT, dFWUNL và WUN-Miner-Mod trên bộ dữ liệu Chainstore với $minwus = 0.02\%$

Tiêu chí mở rộng		dFWUNL			dFWUT			WUN-Miner-Mod		
		Thời gian (s)	Bộ nhớ (MB)	FWUP	Thời gian (s)	Bộ nhớ (MB)	FWUP	Thời gian (s)	Bộ nhớ (MB)	FWUP
Số lượng mục	10000	19.5	137.7	5787	173.3	31.3	5787	35.5	139.6	5787
	20000	38.2	297.2	10451	501	54.3	10451	128.3	305.7	10451
	30000	50.3	411.8	12393	758	65.3	12393	197	422.3	12393
	40000	57.9	445	13852	876.4	77.1	13852	472.7	460.2	13852
	45000	61.4	469.9	14134	943.5	80.3	14134	481	485.8	14134
Số lượng giao dịch (x1000)	300	17.9	273.6	14846	244.8	25.9	14846	64.3	278.5	14846
	500	28	321.4	14202	409	38.9	14202	134.4	328.7	14202
	700	39.3	378.4	14464	595.3	53.1	14464	239.5	389	14464
	900	51	431.3	14495	783	67.2	14495	370.8	444.7	14495
	1100	61.2	469.5	14159	939	79.8	14159	479.3	485.2	14159

2.4 Thuật toán khai thác mẫu trọng số hữu ích phổ biến đóng từ cơ sở dữ liệu định lượng động

Mặc dù các thuật toán dFWUT và dFWUNL đã giải quyết hiệu quả bài toán khai thác mẫu trọng số hữu ích phổ biến từ CSDL định lượng động, nhưng thực tế cho thấy số lượng mẫu sinh ra thường rất lớn, đặc biệt là với các bộ dữ liệu dày đặc hoặc khi ngưỡng khai thác thấp. Điều này không chỉ gây áp lực lên tài nguyên lưu trữ mà còn làm giảm hiệu quả của các bước phân tích dữ liệu sau đó, do sự xuất hiện của nhiều mẫu dư thừa. Vì vậy, hướng nghiên cứu khai thác mẫu trọng số hữu ích phổ

biến đóng trở nên thiết yếu. Các mẫu đóng đóng vai trò là một dạng biểu diễn thu gọn, cho phép giảm đáng kể số lượng mẫu đầu ra nhưng vẫn đảm bảo bảo toàn đầy đủ thông tin về giá trị độ hỗ trợ trọng số hữu ích của toàn bộ tập mẫu.

Ví dụ 2.15: Bảng 2.9 hiển thị tất cả các FWUP và FWUCP cùng giá trị wus được khai thác từ CSDL định lượng động $dQDB_E$ ở Bảng 2.2 với $minwus = 0.5$. Ta thấy, có tất cả 11 FWUP thoả mãn ngưỡng, trong khi số lượng FWUCP thu được chỉ là 5 mẫu.

Bảng 2.9. Các FWUP và FWUCP từ $dQDB_E$ với $minwus = 0.5$

<i>wus</i>	FWUP	FWUCP
1.00	<i>C</i>	<i>C</i>
0.83	<i>E, CE</i>	<i>CE</i>
0.78	<i>A, AC</i>	<i>AC</i>
0.60	<i>AE, ACE</i>	<i>ACE</i>
0.53	<i>B, BC, BE, BCE</i>	<i>BCE</i>

Bài toán khai thác mẫu trọng số hữu ích phổ biến đóng từ CSDL định lượng động là việc tìm tất cả các FWUCP có giá trị hỗ trợ trọng số hữu ích không nhỏ hơn ngưỡng $minwus$ do người dùng chỉ định. Để giải quyết bài toán này, luận án đề xuất thuật toán có tên là FWUCWP khai thác đầy đủ tập mẫu trọng số hữu ích phổ biến đóng. Thuật toán này được xây dựng dựa trên sự kế thừa trực tiếp và phát huy tối đa tính ưu việt của cấu trúc dữ liệu WUNList đã được trình bày và chứng minh hiệu năng ở các phần trước. Tuy nhiên, để xử lý hiệu quả tính chất đóng của mẫu và thu hẹp không gian tìm kiếm một cách tối ưu, việc chỉ dựa vào cấu trúc dữ liệu là chưa đủ. Thuật toán FWUCWP còn được trang bị thêm các cơ chế kiểm soát chặt chẽ nhằm loại bỏ sớm các nhánh không hứa hẹn thông qua các chiến lược cắt tỉa chuyên biệt, sẽ được chi tiết hóa trong các tiểu mục dưới đây.

2.4.1 Các chiến lược cắt tĩa

2.4.1.1 Quan hệ cấu trúc tổ tiên WUNList

Hướng tiếp cận ngây thơ đầu tiên để khai thác FWUCP đó là tìm tất cả các FWUP, sau đó loại bỏ các mẫu không phải là mẫu đúng, ta sẽ thu được FWUCP. Phương pháp này tỏ ra không hiệu quả do phải sinh ra tất cả FWUP. Bằng cách sử dụng cấu trúc WUNList, ta có nhanh chóng loại bỏ các mẫu không thoả mãn trong quá trình khai thác. Ý tưởng của chiến lược này dựa trên khái niệm bao hàm cấu trúc, được xác định thông qua quan hệ tổ tiên trong WUNList. Nếu WUNList của mẫu X có quan hệ cấu trúc tổ tiên với WUNList của mẫu Y ($X \subset Y$), thì điều đó ngụ ý rằng mỗi khi X xuất hiện thì Y cũng xuất hiện trong cùng các giao dịch. Do đó, X không thể là một FWUCP, bởi vì nó có một tập cha Y có cùng độ hỗ trợ trọng số hữu ích. Chiến lược cắt tĩa dựa trên quan hệ cấu trúc tổ tiên WUNList được trình bày cụ thể như sau.

Định nghĩa 2.16 (Quan hệ cấu trúc tổ tiên WUNList - WASR). Cho Px và Py ($y < x$) là hai FWUP có chung tiền tố P . Px và Py có WUNList lần lượt là $wunl(Px)$ và $wunl(Py)$. $wunl(Px)$ được gọi là có quan hệ cấu trúc tổ tiên với $wunl(Py)$, ký hiệu là $wunl(Px) \sqsubset_{anc} wunl(Py)$, khi và chỉ khi $\forall C_u \in wunl(Px)$, $\exists C_v \in wunl(Py)$ sao cho C_v là tổ tiên của C_u .

Ví dụ 2.16: Hai mẫu B và C trong $dQDB_E$ có WUNList lần lượt là $wunl(B) = \{(4, 0, 1.2), (5, 2, 0.87)\}$, $wunl(C) = \{(1, 5, 3.89)\}$ và B đứng sau C trong I_1 . Vì WUN-code $(1, 5, 3.89) \in wunl(C)$ đều là tổ tiên của hai WUN-code của $wunl(B)$ nên $wunl(B)$ quan hệ cấu trúc tổ tiên với $wunl(C)$.

Định lý 2.4. Cho Px và Py ($y < x$) là hai FWUP có chung tiền tố P . Px và Py có WUNList lần lượt là $wunl(Px)$ và $wunl(Py)$.

a. Nếu $wunl(Px) \sqsubset_{anc} wunl(Py)$, thì Px không phải là mẫu trọng số hữu ích phổ biến đúng.

b. Nếu $wunl(Px) \sqsubset_{anc} wunl(Py)$ và $wus(Px) = wus(Py)$, thì Px và Py

đều không phải là mẫu trọng số hữu ích phổ biến đúng.

Chứng minh:

a. Xét mẫu Pxy được tạo thành từ mẫu Px và Py . Theo **Định nghĩa 2.15**, WUNList của mẫu Pxy , ký hiệu là $wunl(Pxy)$, được tính bằng cách xác định các cặp (C_u, C_v) sao cho $C_u \in wunl(Px)$, $C_v \in wunl(Py)$, và C_v là tổ tiên của C_u . Với mỗi cặp hợp lệ, một WUN-code mới được thêm vào $wunl(Pxy)$ với trọng số là $C_u.weight$.

Nếu $wunl(Px) \sqsubseteq_{anc} wunl(Py)$, theo **Định nghĩa 2.16**, với mỗi nút $C_u \in wunl(Py)$, tất yếu tồn tại một nút tổ tiên $C_v \in wunl(Py)$. Điều này suy ra rằng phép kết hợp giữ lại toàn bộ trọng số các nút từ $wunl(Px)$ vào $wunl(Pxy)$. Do đó, tập các giá trị trọng số của Pxy là giống với của Px . Suy ra $\sum_{C_k \in wunl(Pxy)} C_k.weight = \sum_{C_k \in wunl(Px)} C_k.weight$. Chia cả hai vế cho $sumtwu$, ta được:

$$wus(Pxy) = \frac{\sum_{C_k \in wunl(Pxy)} C_k.weight}{sumtwu} = \frac{\sum_{C_u \in wunl(Px)} C_u.weight}{sumtwu} = wus(Px)$$

Vì Pxy là tập cha của Px và cả hai có cùng độ hỗ trợ trọng số hữu ích, theo **Định nghĩa 2.10**, Px không phải là một FWUCP.

b. Chứng minh tương tự như (a), ta có $wus(Px) = wus(Pxy)$. Nếu $wus(Px) = wus(Py)$, thì $wus(Pxy) = wus(Px) = wus(Py)$. Vì Px và Py đều là tập con của Pxy , nên Px và Py không phải là FWUCP.

Ví dụ 2.17: Từ Ví dụ 2.16 ta có $wunl(B) = \{(4, 0, 1.2), (5, 2, 0.87)\}$, $wunl(C) = \{(1, 5, 3.89)\}$, nên $wunl(BC) = \{(1, 5, 2.07)\}$. Vì mẫu BC là tập cha của mẫu B và $wus(B) = wus(BC) = 2.07/3.89$ nên B không phải là FWUCP. Như vậy, để xác định mẫu B không phải là FWUCP ta cần phải tính WUNList của mẫu BC . Thay vào đó, dựa vào **Định lý 2.4**, ta hoàn toàn có thể loại bỏ nhanh mẫu B bằng cách chỉ cần kiểm tra $wunl(B) \sqsubseteq_{anc} wunl(C)$.

Khái niệm quan hệ cấu trúc tổ tiên WUNList được sử dụng để xem xét mối quan hệ giữa hai FWUP thông qua WUNList. Từ đó, ta có thể xác định và loại bỏ các

mẫu không phải là FWUCP. Hàm kiểm tra này được thuật toán FWUCWP sử dụng và mô tả trong Hình 2.15. Hàm *WUNList_Check* kiểm tra $wunl_1$ có quan hệ cấu trúc tổ tiên với $wunl_2$ hay không bằng cách duyệt lần lượt các WUN-code để xác định quan hệ tổ tiên - hậu duệ (dòng 1-6). Nếu tất cả WUN-code của $wunl_1$ đều được duyệt hết thì $wunl_1 \sqsubset_{anc} wunl_2$ và trả về *true*, ngược lại, thuật toán trả về *false* (dòng 7-9).

Thuật toán 2.5. Kiểm tra quan hệ cấu trúc tổ tiên WUNList
Input: $wunl_1$ và $wunl_2$ lần lượt là WUNList của hai mẫu Px và Py
Output: Trả về <i>true</i> nếu $wunl_1 \sqsubset_{anc} wunl_2$; ngược lại, trả về <i>false</i>
Procedure WUNList_Check($wunl_1, wunl_2$) 1. $i \leftarrow 0, j \leftarrow 0$ 2. while $i < wunl_1 $ and $j < wunl_2 $ do 3. if $wunl_2[j].pre < wunl_1[i].pre$ and $wunl_2[j].post > wunl_1[i].post$ then 4. $i++$ 5. else 6. $j++$ 7. if $i = wunl_1 $ then 8. return true 9. return false

Hình 2.15. Thuật toán kiểm tra quan hệ cấu trúc tổ tiên giữa hai WUNList.

Theo **Định nghĩa 2.16**, quan hệ WASR không tồn tại giữa hai WUNList khi có ít nhất một WUN-code trong $wunl(Px)$ không có WUN-code tổ tiên tương ứng trong $wunl(Py)$. Dựa trên nhận xét này, định lý sau trình bày một cơ chế hiệu quả để phát hiện nhanh trường hợp không thoả mãn mà không cần phải duyệt hết tất cả các WUN-code để kiểm tra như hàm *WUNList_Check*.

Định lý 2.5. Cho Px và Py ($y < x$) là hai FWUP có chung tiền tổ P . Px và Py có WUNList lần lượt là $wunl(Px)$ với độ dài là m và $wunl(Py)$ với độ dài là n .

Giả sử ta đã xét WUN-code $C_v \in wunl(Py)$ ($1 \leq v < n$) là tổ tiên của $C_u \in wunl(Px)$ ($1 \leq u < m$). Tiếp tục xét hai WUN-code $C_{u+1} \in wunl(Px)$ và $C_{v+1} \in wunl(Py)$, nếu $C_{u+1}.pre < C_{v+1}.pre$ thì $wunl(Px)$ và $wunl(Py)$ không có quan hệ cấu trúc tổ tiên.

Chứng minh: Ta có, WUNList của các mẫu được sắp xếp tăng dần theo giá trị pre . Như vậy, nếu $C_{u+1}.pre < C_{v+1}.pre$ thì $C_{u+1}.pre < C_{v+1}.pre < C_{v+2}.pre \dots < C_n.pre$ nên $C_{u+1} \in wunl(Px)$ không có tổ tiên nào trong $wunl(Py)$, suy ra $wunl(Px)$ không có quan hệ cấu trúc tổ tiên với $wunl(Py)$.

Cải tiến này giúp tăng tốc quá trình kiểm tra quan hệ cấu trúc tổ tiên WUNList và được mô tả ở Hình 2.16.

Thuật toán 2.6. Cải tiến kiểm tra quan hệ cấu trúc tổ tiên WUNList
Input: $wunl_1$ và $wunl_2$ lần lượt là WUNList của hai mẫu Px và Py
Output: Trả về <i>true</i> nếu $wunl_1 \sqsubset_{anc} wunl_2$; ngược lại, trả về <i>false</i>
Procedure Improved_WUNList_Check($wunl_1, wunl_2$) 1. $i \leftarrow 0, j \leftarrow 0$ 2. while $i < wunl_1 $ and $j < wunl_2 $ do 3. if $wunl_2[j].pre < wunl_1[i].pre$ then 4. if $wunl_2[j].post > wunl_1[i].post$ then 5. $i++$ 6. Else 7. $j++$ 8. else //kết thúc sớm 9. return false 10. if $i = wunl_1 $ then 11. return true 12. return false

Hình 2.16. Thuật toán cải tiến kiểm tra quan hệ cấu trúc tổ tiên giữa hai WUNList.

2.4.1.2 Cắt tia kết hợp *twu* tối đa

Chiến lược cắt tia thứ hai, có tên là Cắt tia kết hợp *twu* tối đa (Maximal Twu Combination Pruning - MTCP), được lấy cảm hứng từ phương pháp EUCP [20] và được thiết kế chuyên biệt để khai thác khả năng nén dữ liệu vượt trội của cấu trúc WUN-tree. Chiến lược này cung cấp một cơ chế kiểm tra sớm dựa trên cận trên của độ hỗ trợ trọng số hữu ích. Trước khi thực hiện phép kết hợp tốn kém giữa hai mẫu, MTCP đánh giá xem giá trị *wus* tối đa có thể đạt được khi kết hợp hai mẫu đó có còn thỏa mãn ngưỡng *minwus* hay không. Nếu cận trên này không đạt ngưỡng yêu cầu, mẫu mới kết hợp đó có thể được loại bỏ ngay lập tức một cách an toàn. Nhờ vậy, MTCP loại bỏ hiệu quả các mẫu không thỏa mãn mà không cần thực hiện quá trình tạo WUNList. Chiến lược này dựa trên tính chất phản đơn điệu và một cấu trúc mới gọi là Precomputed Twu Pairwise (*PTP*). Cấu trúc *PTP* lưu trữ các giá trị cận trên *twu* cho từng cặp mục, từ đó cho phép đưa ra các quyết định cắt tia một cách nhanh chóng và hiệu quả.

Định nghĩa 2.17 (Precomputed Twu Pairwise). Cấu trúc Precomputed Twu Pairwise, ký hiệu là *PTP*, là một ma trận lưu trữ trước giá trị trọng số hữu ích giao dịch của các cặp mục thuộc tập I_1 . Mỗi phần tử của ma trận được xác định như sau:

$$PTP[x, y] = twu(xy), \quad \forall x, y \in I_1, y < x \quad (2.6)$$

Ví dụ 2.18: Hình 2.17 mô tả *PTP* của $dQDB_E$ với *minwus* = 0.5 và $I_1 = \{C, E, A, B\}$. *PTP* được biểu diễn dưới dạng một ma trận tam giác, trong đó mỗi phần tử lưu trữ giá trị *twu* của các mẫu gồm hai mục tương ứng với hàng và cột của ma trận. Ta có, $PTP[E, C] = 3.22$ có nghĩa là giá trị *twu* của mẫu *EC* bằng 3.22.

Mục	<i>C</i>	<i>E</i>	<i>A</i>	<i>B</i>
<i>C</i>				
<i>E</i>	3.22			
<i>A</i>	3.02	2.35		

B	2.07	2.07	1.20	
----------	------	------	------	--

Hình 2.17. Ma trận tam giác PTP của $dQDB_E$ với $minwus = 0.5$.

Phương pháp cơ bản đầu tiên để tạo PTP đó là sinh ra tất cả tập con hai phần tử của từng giao dịch định lượng động và cộng giá trị twu tương ứng vào PTP . Tuy nhiên, dựa vào cấu trúc WUN-tree, ta có thể nhanh chóng xây dựng PTP như sau. Với mỗi nút N (giả sử có $N.name$ là x) trên WUN-tree, ta duyệt lên các tổ tiên của nó cho đến nút gốc. Với mỗi nút tổ tiên Q (giả sử có $Q.name$ là y), ta tăng $PTP[x, y]$ lên $N.weight$. Vì CSDL đã được nén vào WUN-tree nên quá trình xây dựng PTP từ WUN-tree sẽ mất ít thao tác hơn so với phương pháp truyền thống. Sau khi xây dựng xong PTP , ta có thể áp dụng chiến lược MTCP thông qua định lý sau.

Định lý 2.6. Cho hai FWUP Px và Py ($y < x$) có chung tiền tố P . Nếu $PTP[x, y]/sumtwu < minwus$ thì mẫu Pxy không phải là FWUP.

Chứng minh. Ta có $PTP[x, y]$ lưu giá trị twu của mẫu xy . Nếu $wus(xy) = PTP[x, y]/sumtwu < minwus$ thì mẫu xy không phải là FWUP. Theo tính chất phản đơn điệu, nếu mẫu xy không phải là FWUP thì các tập cha của nó cũng không phải là FWUP. Do đó, mẫu Pxy không phải FWUP.

Chiến lược MTCP được áp dụng trước khi tiến hành kết hợp hai FWUP giúp hạn chế các thao tác giao không có ý nghĩa, loại bỏ nhanh các ứng viên chắc chắn không phải là FWUP và từ đó cũng không phải là FWUCP.

2.4.1.3 Cắt tỉa dựa trên khái niệm Enclosure

Phần này trình bày định nghĩa về khái niệm Enclosure, phương pháp tính nhanh Enclosure của các mục và cách sử dụng khái niệm này để cắt tỉa không gian tìm kiếm, tăng tốc quá trình khai thác FWUCP. Khái niệm Enclosure mô tả các mục luôn đồng xuất hiện một cách nghiêm ngặt với một mục khác trong các giao dịch. Cụ thể, nếu mục x luôn xuất hiện trong mọi giao dịch có chứa mục y , thì mọi mẫu đóng chứa x bắt buộc cũng phải chứa y . Ta có thể khai thác tính chất này để mở rộng mẫu trực tiếp mà không cần thực hiện tìm kiếm. Khái niệm mới này được lấy cảm hứng

từ khái niệm Subsume index [76] và được định nghĩa như sau.

Định nghĩa 2.18 (Enclosure). Enclosure của một mục $x \in I_1$, ký hiệu là $Enc(x)$, được tính bằng:

$$Enc(x) = \{y \in I_1 \mid t(x) \subseteq t(y), y < x\}, \quad (2.7)$$

trong đó $t(x)$ và $t(y)$ là lần lượt là Tidset của mục x và y .

Định lý 2.7. Cho một mục $x \in I_1$ có WUNList là $wunl(x)$, ta có:

$$Enc(x) = \{y \in I_1 \mid wunl(x) \sqsubset_{anc} wunl(y), y < x\} \quad (2.8)$$

Chứng minh. Theo **Định nghĩa 2.16**, nếu $wunl(x) \sqsubset_{anc} wunl(y)$ thì tất cả các WUN-code trong $wunl(x)$ đều có tổ tiên trong $wunl(y)$. Điều này có nghĩa là tất cả các giao dịch có chứa mục x cũng chứa mục y . Vì theo cấu tạo của WUN-tree, nếu tồn tại một giao dịch có chứa mục x mà không chứa mục y thì sẽ xuất hiện một nhánh có nút đại diện cho mục x trong cây mà không có tổ tiên là mục y , trái với **Định nghĩa 2.16**. Từ đây, ta có $t(x) \subset t(y)$, suy ra $y \in Enc(x)$.

Như vậy, ta có thể tạo Enclosure của từng mục thông qua việc kiểm tra Tidset, hoặc dựa vào WUNList theo **Định nghĩa 2.18** và **Định lý 2.7**. Tuy nhiên, có một phương pháp khác giúp tạo Enclosure nhanh hơn thông qua định lý sau.

Định lý 2.8. Cho hai mục x và y ($x, y \in I_1, y < x$). Nếu $wus(x) = wus(xy)$ thì $y \in Enc(x)$.

Chứng minh. Ta có $wus(x) = \frac{\sum_{tid \in t(x)} twu(tid)}{sumtwu}$, và tương tự $wus(xy) = \frac{\sum_{tid \in t(xy)} twu(tid)}{sumtwu}$.

Vì $t(xy) = t(x) \cap t(y) \subseteq t(x)$ nên $wus(x) \geq wus(xy)$. Dấu “=” xảy ra khi và chỉ khi $t(x) \cap t(y) = t(x)$ hay $t(x) \subseteq t(y)$. Theo **Định nghĩa 2.18**, suy ra $y \in Enc(x)$.

Bảng 2.10. Enclosure của các mục trong I_1

Mục	C	E	A	B
Enclosure	$\{\}$	$\{C\}$	$\{C\}$	$\{C, E\}$

Ví dụ 2.19: Theo PTP ở Hình 2.17 ta có $wus(E) = wus(EC) = 3.22/3.89$ nên $C \in Enc(E)$. Tương tự, Enclosure của các mục trong I_1 hiển thị ở Bảng 2.10.

Định lý 2.8 giúp ta xác định được mối quan hệ của hai mục thông qua giá trị wus khi kết hợp hai mục này với nhau. Giá trị wus này có thể được tính nhanh bằng cấu trúc PTP đã trình bày trong phần trước. Do đó, ta có thể nhanh chóng xây dựng Enclosure của các mục thông qua PTP . Hơn nữa, việc tạo Enclosure còn có thể được thực hiện đồng thời với quá trình tạo PTP . Thuật toán tạo PTP và Enclosure được mô tả ở Hình 2.18.

Thuật toán 2.7. Xây dựng PTP và Enclosure
Input: WUN-tree R
Output: Cấu trúc PTP và Enclosure của các 1-FWUP
Procedure Construct_PTP_Enclosure(R) 1. Khởi tạo PTP và Enclosure của các 1-FWUP 2. foreach nút N của WUN-tree (duyệt pre-order) do 3. $x \leftarrow N.name$ 4. foreach nút P là tổ tiên của nút N do 5. $y \leftarrow P.name$ 6. $PTP[x][y] += N.weight$ 7. if $PTP[x][y]/sumtwu = wus(x)$ then 8. thêm y vào $x.Enc$

Hình 2.18. Thuật toán xây dựng PTP và Enclosure.

Định lý 2.9. Cho Px và Py ($y < x$) là hai FWUP có chung tiền tố P .

- Nếu $y \in Enc(x)$, thì Px không phải là mẫu trọng số hữu ích phổ biến đóng.
- Nếu $y \in Enc(x)$ và $wus(Px) = wus(Py)$, thì Px và Py không phải là mẫu trọng số hữu ích phổ biến đóng.

Chứng minh:

a. Nếu $y \in Enc(x)$ thì $t(x) \subseteq t(y)$ nên $t(x) \cap t(y) = t(x)$. Ta có, $t(Pxy) = t(P) \cap t(x) \cap t(y) = t(P) \cap t(x) = t(Px)$ nên $wus(Px) = wus(Pxy)$. Mà Px là tập con của Pxy nên Px không phải là FWUCP.

b. Chứng minh tương tự (a), ta có nếu $y \in Enc(x)$ thì $wus(Px) = wus(Pxy)$. Nếu $wus(Px) = wus(Py)$ thì $wus(Pxy) = wus(Px) = wus(Py)$. Vì Px và Py là tập con của Pxy nên Px và Py đều không phải là FWUCP.

Như vậy, theo định nghĩa về mẫu đóng, nếu mục $y \in Enc(x)$ thì tất cả các mẫu đóng có chứa mục x cũng chứa y . Chiến lược này được áp dụng vào quá trình kiểm tra và tạo các ứng viên kết hợp với chiến lược WASR.

2.4.2 Thuật toán FWUCWP

Thuật toán FWUCWP sử dụng các chiến lược cắt tỉa đã trình bày ở trên kết hợp với phương pháp duyệt không gian tìm kiếm dựa trên cây liệt kê tập hợp để khai thác FWUCP. Hình 2.19 mô tả thuật toán FWUCWP. Đầu tiên, dòng 1-3, thuật toán tiến hành quét CSDL để tạo WUN-tree, PTP , Enclosure và WUNList của các 1-FWUP, đồng thời khởi tạo danh sách lưu trữ các FWUCP. Tiếp theo, thuật toán gọi thủ tục *Mine_FWUCPs* để tiến hành khai thác tất cả FWUCP thỏa mãn ngưỡng $minwus$ (dòng 5). Thủ tục *Mine-FWUCPs* có ba tham số *Prefix*, *S*, *C* lần lượt là tiền tố chung, danh sách các mục thuộc mẫu đóng của lớp hiện tại, và danh sách các mục cuối cùng của các ứng viên. Trong thủ tục này, ta sử dụng các biến $Prefix_{next}$, S_{next} , và C_{next} để lưu trữ thông tin phục vụ cho lần đệ quy ở lớp tiếp theo (dòng 7, 9). Lần lượt hai mục C_i và C_j trong C được kiểm tra. Nếu $PTP[C_i, C_j]/sumtwu < minwus$ thì ta chuyển sang duyệt mục tiếp theo vì việc kết hợp hai mục này không tạo thành FWUP theo MTCP (dòng 11). Ngược lại, dòng 12-17, nếu C_j thuộc Enclosure của C_i hoặc WUNList của C_i có quan hệ cấu trúc tổ tiên với WUNList của C_j thì C_j được thêm vào S_{next} vì tập đóng có chứa C_i cũng chứa C_j . Đồng thời, nếu C_i và C_j có cùng giá trị wus thì ta xoá C_j khỏi danh sách theo **Định lý 2.4** và **Định lý**

2.9. Nếu các chiến lược cắt tia trên không thoả mãn, ứng viên mới sẽ được tạo thành và được thêm vào C_{next} nếu là FWUP (dòng 18-22). Sau đó, nếu mẫu của lớp hiện tại là FWUCP thì được thêm vào kết quả và ta gọi đệ quy để khai thác ở lớp tiếp theo (dòng 23-26). Thủ tục này dừng lại khi không có mẫu mới nào được tạo thành và tất cả FWUCP được cung cấp cho người dùng.

Thuật toán 2.8. Thuật toán FWUCWP
Input: CSDL định lượng động $dQDB$ và một ngưỡng $minwus$;
Output: Tập tất cả các mẫu trọng số hữu ích phổ biến đóng $FWUCPs$
1. $R \leftarrow \mathbf{Build_WUN-tree}(dQDB, minwus)$ 2. Construct-PTP-Enclosure (R) 3. Duyệt WUN-tree R để tạo WUNList cho các mục trong I_1 4. $FWUCPs \leftarrow \emptyset$ 5. Mine_FWUCPs ($\emptyset, \emptyset, I_1$) 6. return $FWUCPs$
Procedure Mine_FWUCPs ($Prefix, S, C$) 7. $Prefix_{next} \leftarrow Prefix$ 8. for $i \leftarrow C - 1$ downto 0 do 9. $Prefix_{next} \leftarrow Prefix \cup C_i, S_{next} \leftarrow S, C_{next} \leftarrow \emptyset$ 10. for $j \leftarrow i - 1$ downto 0 do 11. if $PTP[C_i][C_j]/sumtwu < minwus$ then continue 12. if $C_j \in C_i.Enc$ or Improved_WUNList_Check ($C_i.wunl, C_j.wunl$) then 13. $S_{next} \leftarrow S_{next} \cup C_j$ 14. if $C_i.wus = C_j.wus$ then 15. Xoá C_j 16. $i--$ 17. Continue 18. $Cand \leftarrow C_j$ 19. $Cand.wunl \leftarrow \mathbf{Construct_WUNList}(C_i.wunl, C_j.wunl)$

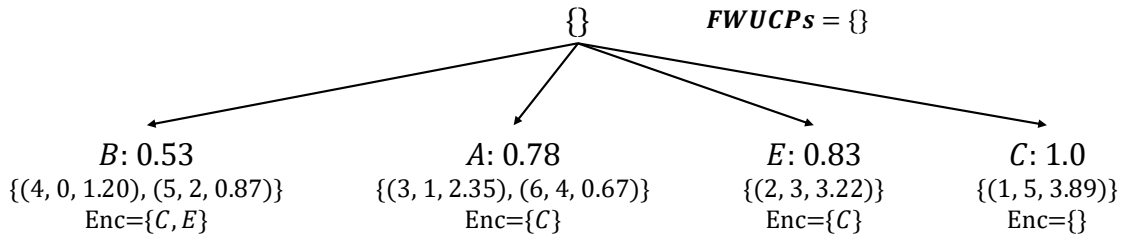
```

20.   if  $Cand.wunl \neq NULL$  then
21.        $Cand.Enc \leftarrow C_i.Enc \cup C_j.Enc$ 
22.        $C_{next} \leftarrow Cand \cup C_{next}$ 
23.    $P \leftarrow Prefix_{next} \cup S_{next}, P.wus \leftarrow C_i.wus$ 
24.   if  $P$  là mẫu đóng trong  $FWUCPs$  then
25.        $FWUCPs \leftarrow FWUCPs \cup P$ 
26.   Mine_FWUCPs( $Prefix_{next}, S_{next}, C_{next}$ )
27.   Xoá mục cuối cùng của  $Prefix_{next}$ 

```

Hình 2.19. Thuật toán FWUCWP.

Ví dụ 2.20: Cách thức hoạt động của thuật toán FWUCWP trên CSDL ví dụ $dQDB_E$ ở Bảng 2.2 với $minwus = 0.5$ như sau. Đầu tiên, thuật toán FWUCWP tiến hành xây dựng WUN-tree (Hình 2.4), $I_1 = \{C, E, A, B\}$, PTP (Hình 2.17) và Enclosure của các mục trong I_1 (Bảng 2.10). Tiếp theo, thuật toán tạo WUNList của các mục trong I_1 , khởi tạo $FWUCPs$ để lưu trữ các FWUCP, và bắt đầu quá trình khai thác dựa trên cây liệt kê tập hợp (Hình 2.20) với $Prefix = \emptyset, S = \emptyset, C = I_1$.

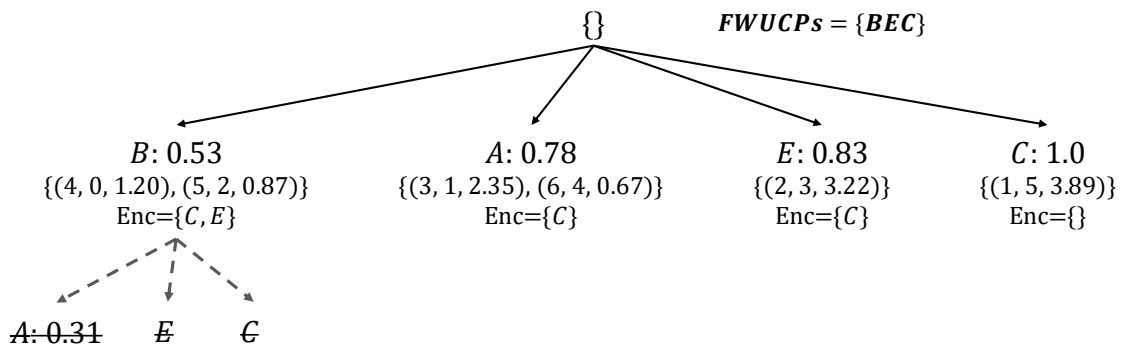


Hình 2.20. WUNList và Enclosure của I_1 để bắt đầu khai thác.

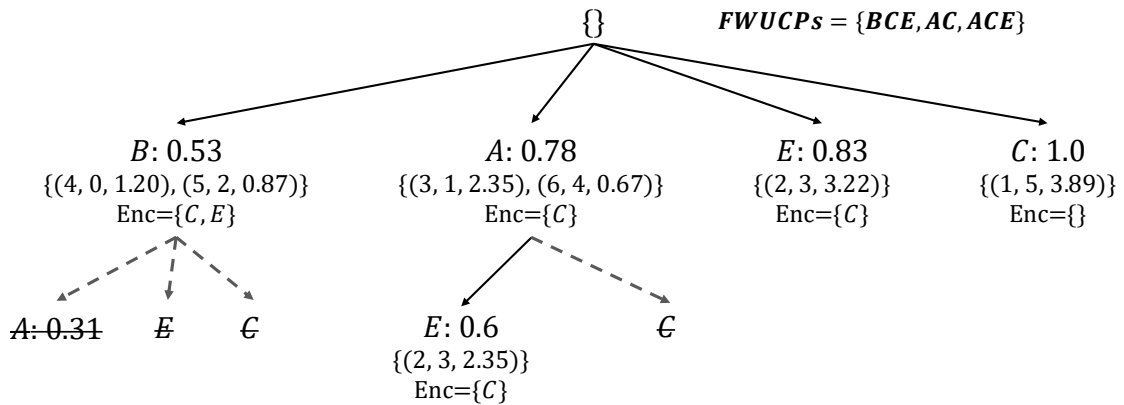
Xét nhánh B , ta có $Prefix_{next} = \{B\}, S_{next} = \emptyset, C_{next} = \emptyset$. Lần lượt kết hợp mẫu B với các mẫu C, E , và A . Vì mẫu C và E đều thuộc Enclosure của B nên hai mẫu này được thêm vào S_{next} . $PTP[B, A] = 1.20/3.89 < minwus$ nên mẫu BA bị loại vì không phải là FWUP. Kết thúc lớp tương đương của B , ta thêm mẫu BCE vào $FWUCPs$. Kết quả được hiển thị ở Hình 2.21, mũi tên nét đứt thể hiện quá trình cắt tỉa các ứng viên không phù hợp.

Xét nhánh A , ta có $Prefix_{next} = \{A\}, S_{next} = \emptyset, C_{next} = \emptyset$. Vì mẫu C thuộc

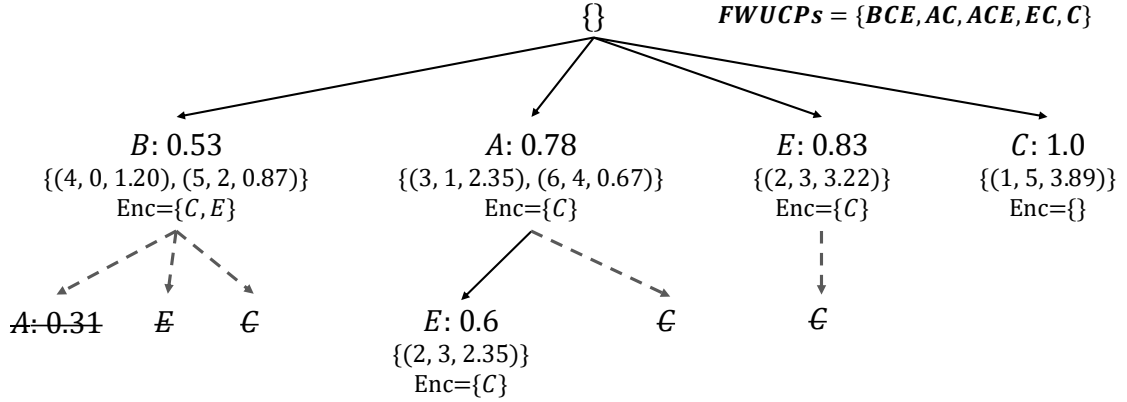
enclosure của mẫu A nên ta cập nhật $S_{next} = \{C\}$. $PTP[A, E] = 2.35/3.89 > minwus$ nên ứng viên AE được tạo thành với $wunl(AE) = \{(2, 3, 2.35)\}$, $Enc(AE) = \{C\}$, và thêm E vào C_{next} . Kết thúc lớp tương đương của A , ta thêm mẫu AC vào $FWUCPs$. Tiếp tục duyệt lớp tương đương của AE với thủ tục $Mine_FWUCPs(Prefix = \{A\}, S = \{C\}, C = \{E\})$. Lúc này, ta có $Prefix_{next} = \{A, E\}$, $S_{next} = \{C\}$, $C_{next} = \emptyset$. Vì không có ứng viên nào để kết hợp nên ta kết thúc lớp tương đương này, đồng thời mẫu ACE được thêm vào $FWUCPs$ (Hình 2.22).



Hình 2.21. Kết quả sau khi kết hợp lớp tương đương B.



Hình 2.22. Kết quả sau khi kết hợp lớp tương đương A.



Hình 2.23. Kết quả sau khi kết hợp lớp tương đương E và C .

Tiến hành tương tự với nhánh E và nhánh C (Hình 2.23), ta tìm được thêm hai mẫu EC và C là FWUCP và thêm hai mẫu này vào $FWUCPs$.

Thuật toán kết thúc, ta thu được tập các mẫu trọng số hữu ích phổ biến đóng là $FWUCPs = \{BCE, AC, ACE, EC, C\}$.

2.4.3 Tính đúng đắn và độ phức tạp thuật toán

Thuật toán FWUCWP được phát triển dựa trên nền tảng cấu trúc WUN-tree và WUNList đã được chứng minh tính hiệu quả. Cùng với đó, thuật toán FWUCWP đảm bảo cả tính đầy đủ và tính đúng đắn thông qua chiến lược duyệt không gian tìm kiếm trên cây liệt kê tập hợp được tăng cường bởi các kỹ thuật cắt tỉa dựa trên nguyên lý phản đơn điệu. Cụ thể, chiến lược cắt tỉa MTCP tại dòng 11 của thủ tục *Mine-FWUCPs* đảm bảo rằng nếu ước lượng chặn trên wus của một cặp mục không thỏa ngưỡng $minwus$, thì mọi mẫu mở rộng chứa cặp mục đó chắc chắn không thể là FWUP, nên việc loại bỏ nhánh là an toàn. Thêm vào đó, việc xử lý các quan hệ Enclosure và WUNList cho phép thuật toán nhóm các mục luôn đồng xuất hiện thành một đơn vị xử lý thống nhất. Cơ chế này không làm mất bất kỳ mẫu nào vì các mục này không thể được tách thành các mẫu đóng riêng biệt mà chúng bắt buộc phải luôn xuất hiện cùng nhau trong tất cả các FWUCP. Do đó, không gian tìm kiếm của các mẫu đóng được bảo toàn đầy đủ. Đồng thời, tính đúng đắn của tập kết quả còn được kiểm soát chặt chẽ qua quy trình xác minh hai lớp, tập trung vào tính chất đóng của mẫu. Lớp thứ nhất được thực thi qua việc kiểm tra Enclosure và hàm *Improved-*

WUNList-Check tại dòng 12-17 của thủ tục khai thác để phát hiện các quan hệ siêu tập cục bộ. Lớp thứ hai thực hiện kiểm tra bao đóng toàn cục bằng cách đối chiếu từng ứng viên với tập FWUCP đã tìm được. Điều này đảm bảo mọi mẫu kết quả đều là mẫu đóng, không dư thừa và thoả mãn ngưỡng *minwus*.

Về mặt hiệu năng, độ phức tạp tính toán của thuật toán FWUCWP được xác định bởi tổng chi phí của ba giai đoạn xử lý tuần tự: khởi tạo cấu trúc cây, tiền xử lý dữ liệu và khai thác mẫu đệ quy. Gọi m là số lượng giao dịch, l_{max} là độ dài giao dịch lớn nhất và $|I_1|$ là số lượng 1-FWUP trong CSDL đầu vào. Giai đoạn đầu tiên xây dựng WUN-tree có độ phức tạp thời gian là $O(m \cdot l_{max} \log l_{max})$ và bộ nhớ là $O(m \cdot l)$. Tiếp theo, việc tạo cấu trúc *PTP* và danh sách Enclosure được thực hiện bằng cách duyệt WUN-tree. Tại mỗi nút, thuật toán phải truy xuất ngược lên các tổ tiên để cập nhật giá trị trọng số hữu ích, do đó với kích thước cây tối đa tiệm cận $m \cdot l$ và chiều cao cây bị chặn bởi $|I_1|$, chi phí thời gian cho bước tiền xử lý này là $O(m \cdot l \cdot |I_1|)$ và cần không gian lưu trữ là $O(|I_1|^2)$. Tuy nhiên, trọng tâm tính toán nằm ở thủ tục *Mine-FWUCPs* khi thuật toán duyệt không gian tìm kiếm thông qua cây liệt kê tập hợp. Mặc dù các kỹ thuật cắt tỉa có thể giúp thu hẹp đáng kể số lượng trường hợp cần xét trong thực tế, nhưng cận trên vẫn là $2^{|I_1|}$, tức là tổng số tập con có thể sinh ra từ $|I_1|$. Tại mỗi nút của cây tìm kiếm này, thuật toán thực hiện các phép toán kết hợp WUNList với độ phức tạp là $O(wunl_{max})$, trong đó $wunl_{max}$ là độ dài tối đa của một WUNList. Khi có một ứng viên tiềm năng được tạo ra, quá trình kiểm tra bao đóng sẽ được thực hiện với chi phí là $O(F_w \cdot |I_1|)$, trong đó F_w là số lượng FWUCP trung bình của các nhóm mẫu đóng có cùng *wus*. Trong trường hợp xấu nhất, khi các mẫu tìm được đều là FWUCP thì bộ nhớ cần thiết để lưu trữ kết quả có thể lên đến $O(2^{|I_1|} \cdot |I_1|)$. Do đó, độ phức tạp thời gian tổng thể của thuật toán FWUCWP là $O(m \cdot l_{max} \log l_{max} + m \cdot l \cdot |I_1| + 2^{|I_1|} \cdot (wunl_{max} + F_w \cdot |I_1|))$ và bộ nhớ là $O(m \cdot l_{max} + |I_1|^2 + 2^{|I_1|} \cdot |I_1|)$. Do hàm mũ thường rất lớn nên độ phức tạp của thuật toán FWUCWP về thời gian là $O(2^{|I_1|})$ và về bộ nhớ là $O(2^{|I_1|})$. Tuy nhiên, trong thực tế triển khai, số lượng các mẫu FWUCP thoả mãn điều kiện thường chiếm

tỷ lệ rất nhỏ so với không gian lý thuyết, phụ thuộc lớn vào đặc điểm phân bố dữ liệu. Chính trong bối cảnh này, các chiến lược cắt tia không gian tìm kiếm mạnh mẽ kết hợp với hiệu quả của cấu trúc dữ liệu WUNList đóng vai trò quan trọng trong việc loại bỏ sớm các mẫu không phù hợp, giúp thuật toán đạt được tốc độ khai thác thực tế nhanh hơn đáng kể so với các phân tích lý thuyết phân tích.

2.4.4 Thực nghiệm và kết quả

2.4.4.1 Thiết lập thực nghiệm

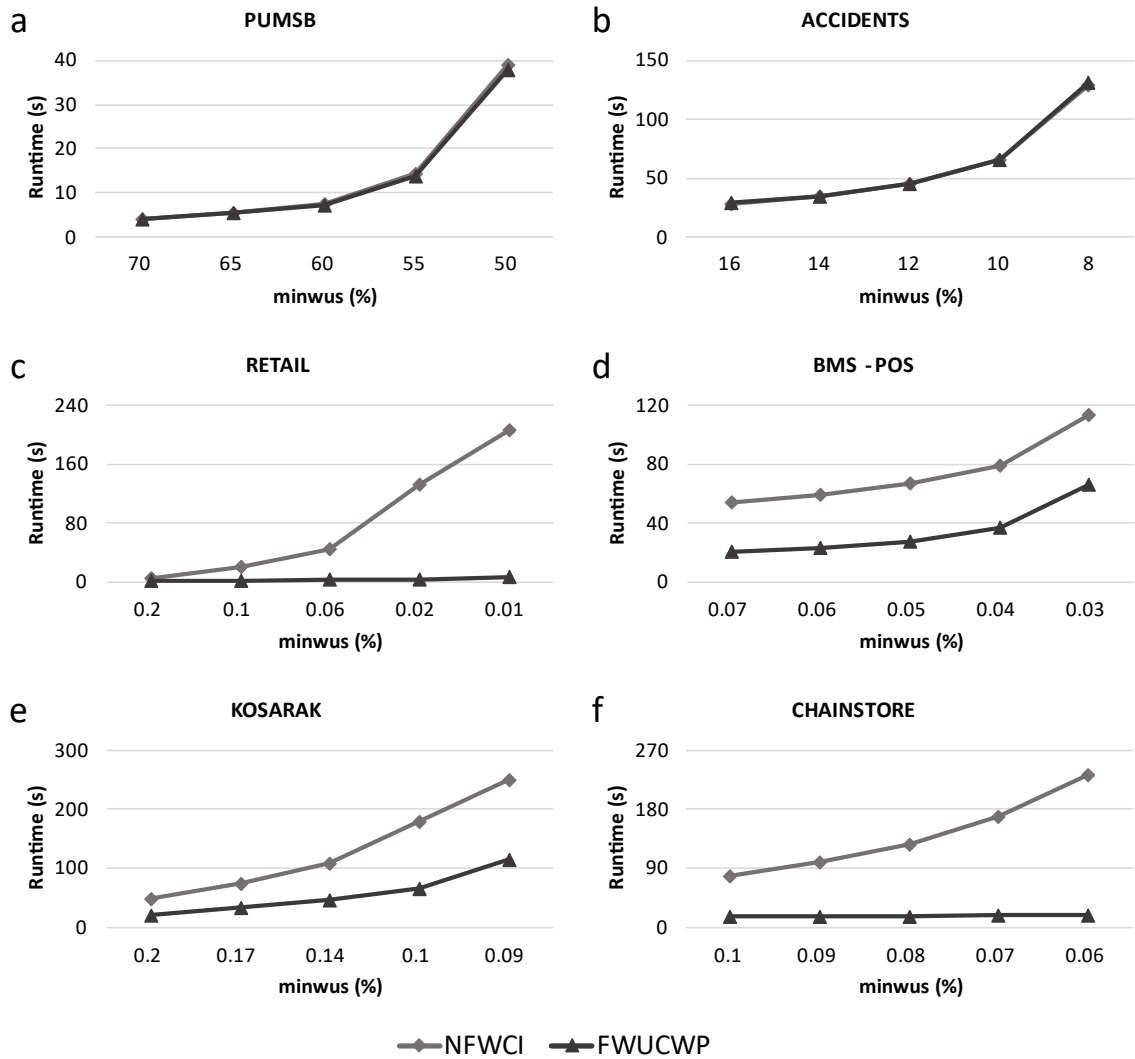
Trong phần này, luận án trình bày kết quả thực nghiệm đánh giá hiệu suất của thuật toán FWUCWP. Vì hiện tại chưa có thuật toán nào áp dụng cho bài toán khai thác mẫu trọng số hữu ích phổ biến đóng từ CSDL định lượng động, nên luận án tiến hành so sánh thuật toán đề xuất với thuật toán NFWCI [7]. Đây là thuật toán khai thác mẫu trọng số phổ biến đóng trên CSDL trọng số tốt nhất hiện nay. Thuật toán NFWCI được chỉnh sửa lại để hoạt động tốt trên CSDL định lượng động và đảm bảo không ảnh hưởng đến độ phức tạp.

Các thuật toán được lập trình bằng ngôn ngữ C# sử dụng môi trường phát triển tích hợp Microsoft Visual Studio 2022 trên nền tảng .NET Framework 4.7.2. Quá trình thực nghiệm được thực hiện trên một máy tính trang bị CPU Intel Core i5-9400F 2.90 GHz, RAM 32GB và hệ điều hành Windows 11. Các bộ dữ liệu thực nghiệm bao gồm Pumsb, Retail, Accidents, BMS-POS, Kosarak, Chainstore, CI10K - CI45K và CT300K - CT1100K, đã được mô tả trong Bảng 2.6.

Vì các bộ dữ liệu thực nghiệm có kích thước và đặc điểm khác nhau nên các ngưỡng *minwus* được lựa chọn khác nhau cho từng bộ dữ liệu. Điều này đảm bảo cho việc đánh giá hiệu suất được thể hiện rõ ràng giữa các thuật toán. Thuật toán FWUCWP và NFWCI đều tìm được đầy đủ và chính xác tất cả các FWUCP nên việc phân tích hiệu suất của các thuật toán này dựa trên số liệu về thời gian chạy và bộ nhớ sử dụng. Trong đó, thời gian chạy của thuật toán được tính từ lúc bắt đầu cho đến khi thuật toán dừng lại, và bộ nhớ sử dụng là bộ nhớ tối đa sử dụng trong suốt quá trình thuật toán hoạt động.

2.4.4.2 So sánh về thời gian chạy

Hình 2.24 hiển thị thời gian khai thác của hai thuật toán trên các bộ dữ liệu thực nghiệm khi thay đổi ngưỡng *minwus*. Các biểu đồ đều cho thấy khi ngưỡng giảm, thời gian khai thác của các thuật toán tăng lên, do số lượng các mẫu FWUCP tạo ra nhiều hơn. Kết quả trên các bộ dữ liệu dày được hiển thị ở Hình 2.24a và Hình 2.24b. Ta thấy cả hai thuật toán có thời gian khai thác tương đương nhau. Trên bộ dữ liệu Pumsb với ngưỡng $\text{minwus} = 55\%$, thuật toán NFWCI mất 14.3 giây, còn thuật toán FWUCWP mất 13.8 giây. Trên bộ dữ liệu Accidents với ngưỡng $\text{minwus} = 10\%$, thời gian khai thác của thuật toán NFWCI và FWUCWP lần lượt là 65.6 giây và 66.3 giây. Tuy nhiên, với các bộ dữ liệu thưa và rất thưa (Hình 2.24c-f), thuật toán FWUCWP hiệu quả hơn so với thuật toán NFWCI trên tất cả các ngưỡng *minwus*. Hơn nữa, khi ngưỡng càng nhỏ thì sự chênh lệch càng thể hiện rõ rệt. Ví dụ, trên bộ dữ liệu Retail ở Hình 2.24c, thuật toán FWUCWP nhanh hơn thuật toán NFWCI gần 30 lần, và nhanh hơn gần 12 lần trên bộ dữ liệu Chainstore (Hình 2.24f).



Hình 2.24. Thời gian chạy của thuật toán NFWCI và FWUCWP trên các CSDL thực nghiệm.

Cả hai thuật toán đều được xây dựng dựa trên N-list nên đều tận dụng được sức mạnh của cấu trúc dữ liệu này. Sự khác biệt về hiệu năng giữa chúng chủ yếu xuất phát từ mức độ hiệu quả của các chiến lược cắt tỉa so với đặc tính của dữ liệu. Khi CSDL càng dày thì khả năng nén dữ liệu càng hiệu quả nên độ dài của cấu trúc dữ liệu của các mẫu càng ngắn. Hơn nữa, các ứng viên mới được tạo thành có cấu trúc dựa trên N-list sẽ càng nhỏ lại sau mỗi lần kết hợp, dẫn đến tốc độ tạo ứng viên là rất nhanh. Mặc dù thuật toán FWUCWP sử dụng các chiến lược cắt tỉa nhưng lại mất thời gian để tạo và duy trì các chiến lược này (*PTP* và *Enclosure*). Do đó, trong

các bộ dữ liệu dày, thời gian khai thác của hai thuật toán không có sự chênh lệch.

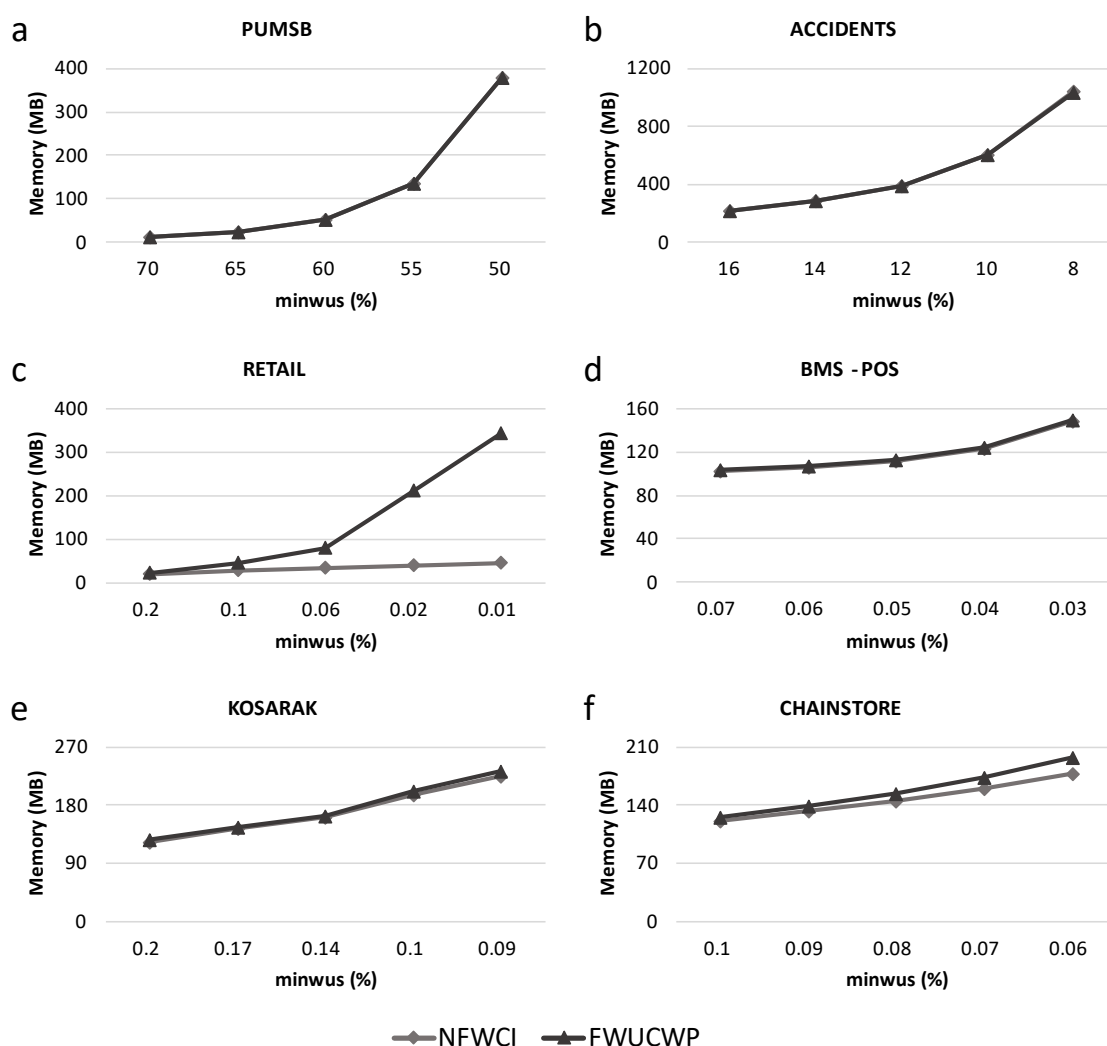
Ngược lại, với các bộ dữ liệu thưa và rất thưa, lúc này các chiến lược cắt tia của thuật toán FWUCWP phát huy hiệu quả của mình. Các bộ dữ liệu thưa thường chứa nhiều mục khác nhau, nhưng chúng thường ít xuất hiện cùng nhau. Điều này tạo ra một không gian tìm kiếm lớn với nhiều mẫu ứng viên có độ hỗ trợ trọng số hữu ích thấp. Trong các trường hợp này, thuật toán NFWCI mất nhiều thời gian trong việc tạo mẫu mới do khả năng nén dữ liệu kém đi, đặc biệt khi ngưỡng *minwus* càng nhỏ, càng có nhiều mẫu được tạo thành. Trong khi đó, chiến lược MTCP của thuật toán FWUCWP tỏ ra rất hiệu quả vì nó ước lượng cận trên độ hỗ trợ trọng số hữu ích của một mẫu mới. Bằng cách kiểm tra cấu trúc *PTP*, FWUCWP có thể loại bỏ sớm một số lượng lớn các phần mở rộng không hợp lệ trong cây tìm kiếm mà không cần thực hiện phép tạo WUNList mới tốn kém về tính toán. Do đó, FWUCWP đạt được thời gian khai thác tốt hơn đáng kể so với thuật toán NFWCI.

2.4.4.3 So sánh về bộ nhớ sử dụng

Hình 2.25 hiển thị bộ nhớ sử dụng trên các bộ dữ liệu thực nghiệm của hai thuật toán. Các kết quả đều cho thấy khi ngưỡng giảm dần thì mức độ sử dụng bộ nhớ của các thuật toán đều tăng lên. Trong đó, sự chênh lệch về bộ nhớ sử dụng là không nhiều. Ví dụ, trên bộ dữ liệu Pumsb trong Hình 2.25a với ngưỡng là 60%, thuật toán NFWCI sử dụng 52 MB bộ nhớ, còn thuật toán FWUCWP là 51.9 MB. Trên bộ dữ liệu BMS-POS ở Hình 2.25d với ngưỡng là 0.04%, bộ nhớ sử dụng của hai thuật toán NFWCI và FWUCWP lần lượt là 122.6 MB và 124.13 MB. Kết quả tương tự cũng diễn ra trên bộ dữ liệu còn lại.

Tuy nhiên, kết quả của bộ dữ liệu Retail ở Hình 2.25c cho thấy mức sử dụng bộ nhớ của thuật toán FWUCWP tăng khá đáng kể, trong khi mức sử dụng bộ nhớ của thuật toán NFWCI là ít hơn khi ngưỡng giảm. Điều này là do thuật toán FWUCWP phải tạo cấu trúc *PTP* để áp dụng chiến lược MTCP. Chiến lược tuy rất hiệu quả về thời gian chạy nhưng lại tốn bộ nhớ vì phải phụ thuộc số lượng 1-FWUP. Khi ngưỡng càng nhỏ thì số lượng các 1-FWUP càng nhiều dẫn đến bộ nhớ sử dụng

càng tăng. Hơn nữa, thuật toán FWUCWP còn tốn thêm bộ nhớ để duy trì Enclosure của các mẫu nên bộ nhớ sử dụng của thuật toán FWUCWP nhiều hơn thuật toán NFWCI trên các bộ dữ liệu thực nghiệm. Bảng 2.11 tổng hợp các kết quả thực nghiệm khai thác FWUCP trên từng bộ dữ liệu tại các ngưỡng *minwus* đại diện, nhằm cung cấp số liệu cụ thể để đối chiếu và đánh giá hiệu quả của các thuật toán được so sánh.



Hình 2.25. Bộ nhớ sử dụng của thuật toán NFWCI và FWUCWP trên các CSDL thực nghiệm.

Bảng 2.11. Kết quả khai thác FWUCP trên các bộ dữ liệu thực nghiệm

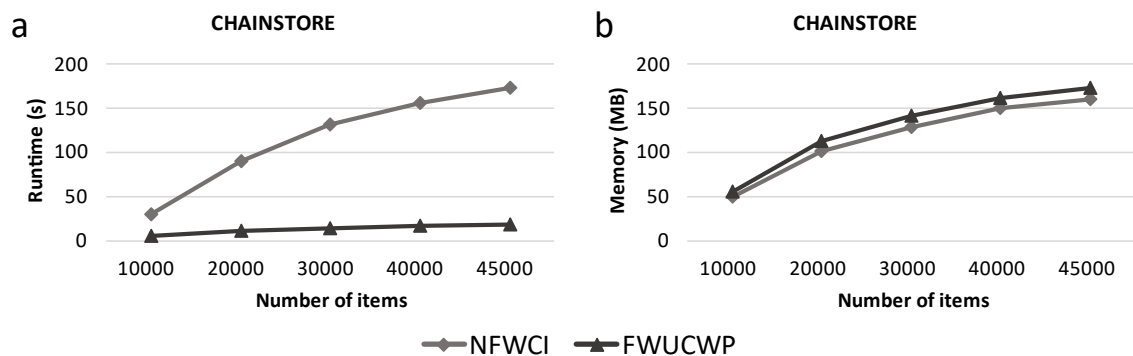
CSDL (<i>minwus</i>)	NFWCI	FWUCWP
------------------------	-------	--------

	Thời gian (s)	Bộ nhớ (MB)	FWUCP	Thời gian (s)	Bộ nhớ (MB)	FWUCP
Pumsb (50%)	39	378.4	7291181	38	378.3	7291181
Accidents (12%)	45	388.4	5598157	45.3	387.7	5598157
Retail (0.01%)	205.5	43.7	182242	6.8	341.9	182242
BMS-POS (0.03%)	113.7	147.7	1743701	66	149.6	1743701
Kosarak (0.09%)	251.4	225.4	1051717	114.6	232.6	1051717
Chainstore (0.06%)	234	178.1	3106	18.9	198.3	3106

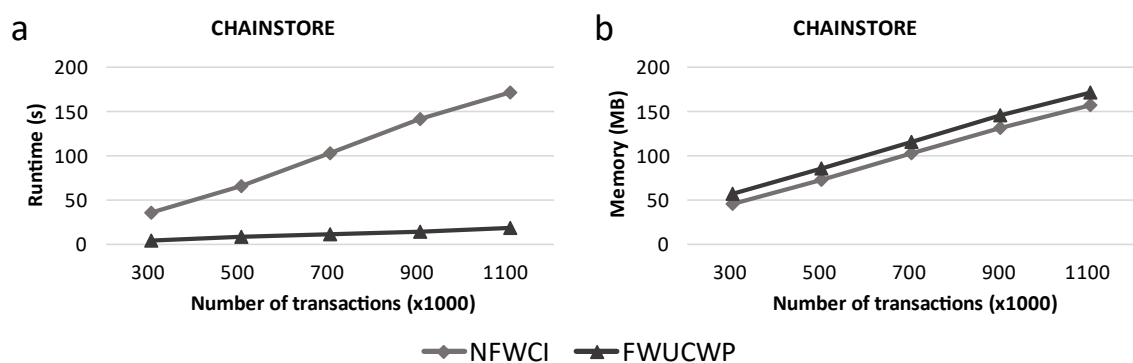
2.4.4.4 So sánh về khả năng mở rộng

Phần này đánh giá khả năng mở rộng của hai thuật toán trên các bộ dữ liệu được trích xuất từ bộ dữ liệu Chainstore. Các số liệu kết quả chi tiết được tổng hợp trong Bảng 2.12. Hình 2.26 hiển thị kết quả về khả năng mở rộng khi số mục tăng từ 10000 đến 45000, với giá trị ngưỡng cố định là 0.07%. Kết quả cho thấy khi số lượng mục tăng lên thì thời gian chạy và bộ nhớ sử dụng của hai thuật toán đều tăng lên. Số lượng mục tăng lên làm cho dữ liệu trên các giao dịch nhiều hơn dẫn đến cấu trúc dữ liệu của các thuật toán cần nhiều thời gian và bộ nhớ hơn để tạo và lưu trữ thông tin. Ta có thể thấy được rằng thuật toán FWUCWP hoạt động nhanh hơn thuật toán NFWCI, nhưng lại tiêu thụ nhiều bộ nhớ hơn.

Hình 2.27 hiển thị kết quả kiểm tra khả năng mở rộng được thực hiện bằng cách tăng số lượng giao dịch từ 300000 đến 1100000, với giá trị ngưỡng *minwus* là 0.07%. Với số lượng giao dịch là 300000, thuật toán NFWCI mất 36.5 giây và thuật toán FWUCWP mất 4.7 giây, còn khi số lượng giao dịch là 1100000, thuật toán NFWCI mất 171.8 giây, còn thuật toán FWUCWP chỉ mất 18.2 giây. Từ kết quả trên, ta thấy thời gian chạy của thuật toán NFWCI tăng nhanh khi số lượng giao dịch tăng lên, trong khi mức tăng của thuật toán FWUCWP là ổn định hơn. Do đó, thuật toán FWUCWP có khả năng mở rộng tốt hơn thuật toán NFWCI.



Hình 2.26. Khả năng mở rộng của thuật toán NFWCI và FWUCWP với số lượng mục khác nhau.



Hình 2.27. Khả năng mở rộng của thuật toán NFWCI và FWUCWP với số lượng giao dịch khác nhau.

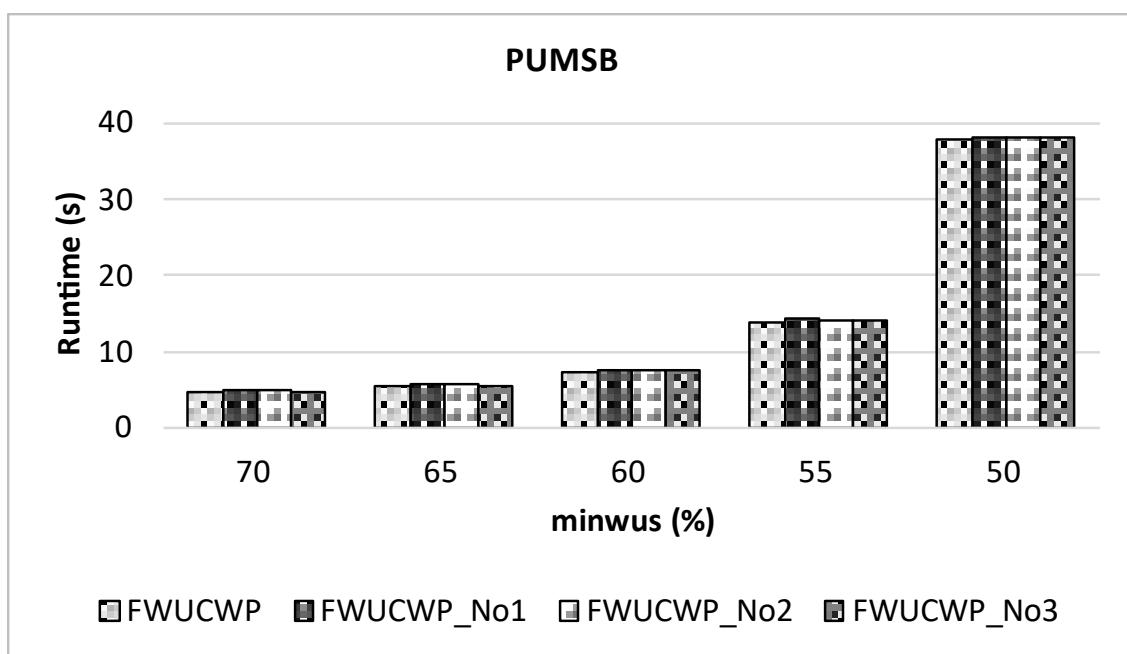
Bảng 2.12. Khả năng mở rộng của NFWCI và FWUCWP trên bộ dữ liệu Chainstore với $minwus = 0.07\%$

Tiêu chí mở rộng		NFWCI			FWUCWP		
		Thời gian (s)	Bộ nhớ (MB)	FWUCP	Thời gian (s)	Bộ nhớ (MB)	FWUCP
Số lượng mục	10000	30.2	49.84	1362	6	55.6	1362
	20000	89.4	102.08	2061	11.6	112.35	2061
	30000	131	129.1	2307	14.7	142.31	2307

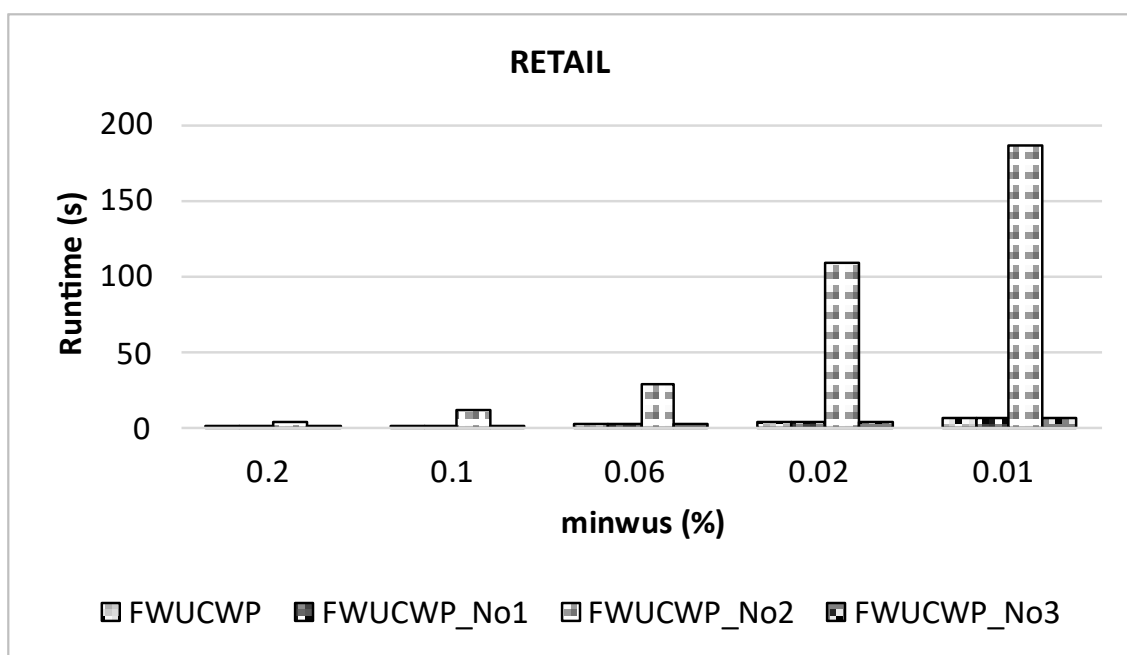
	40000	156.4	149.87	2401	17.2	162.16	2401
	45000	173.8	159.91	2463	18.4	173.12	2463
Số lượng giao dịch (x1000)	300	36.5	45.39	2504	4.7	57.75	2504
	500	66.6	72.88	2443	7.9	85.5	2443
	700	103.2	102.86	2470	11.3	115.9	2470
	900	141.3	132.44	2490	15	145.76	2490
	1100	171.8	158.13	2458	18.2	171.26	2458

2.4.4.5 Phân tích hiệu quả của chiến lược cắt tĩa

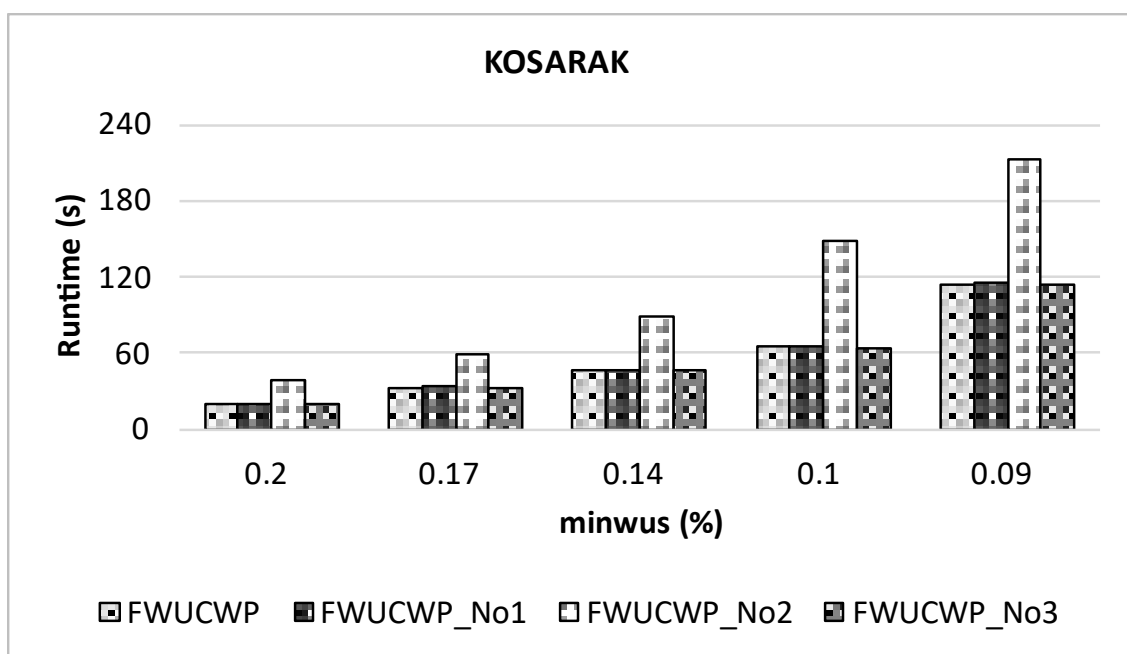
Để đánh giá định lượng và chính xác sự đóng góp của từng chiến lược cắt tĩa vào thuật toán đề xuất, một nghiên cứu loại trừ được tiến hành trong phần này. Thuật toán hoàn chỉnh FWUCWP được so sánh với ba biến thể của nó là FWUCWP_No1 (vô hiệu hóa chiến lược 1 Quan hệ cấu trúc tổ tiên WUNList), FWUCWP_No2 (vô hiệu hóa chiến lược 2 Cắt tĩa kết hợp *twu* tối đa), và FWUCWP_No3 (vô hiệu hóa chiến lược 3 Cắt tĩa dựa trên khái niệm Enclosure). Quá trình thực nghiệm được tiến hành trên ba bộ dữ liệu có đặc tính khác nhau là Pumsb, Retail và Kosarak. Hình 2.28-Hình 2.30 và Bảng 2.13-Bảng 2.15 mô tả hiệu suất của các thuật toán, được đánh giá dựa trên hai thước đo chính là thời gian chạy và tổng số ứng viên được sinh ra. Kết quả thực nghiệm cho thấy sự khác biệt rõ rệt về mức độ hiệu quả của các chiến lược cắt tĩa giữa các bộ dữ liệu dày đặc và thưa thớt. Dữ liệu chỉ ra rằng khi ngưỡng giảm, cả thời gian khai thác lẫn số lượng ứng viên được sinh ra đều tăng lên. Trên các bộ dữ liệu thưa như Retail và Kosarak, chiến lược 2 tỏ ra cực kỳ hiệu quả và đóng vai trò then chốt. Việc vô hiệu hóa chiến lược 2 trong thuật toán FWUCWP_No2 đã dẫn đến sự suy giảm hiệu suất đáng kể. Ví dụ, trên bộ dữ liệu Retail với $minwus = 0.01\%$, việc thiếu đi chiến lược 2 khiến số lượng ứng viên tăng vọt từ khoảng 01 triệu lên hơn 40 triệu ứng viên, kéo theo thời gian chạy tăng mạnh từ 6.8 giây lên 186 giây, tức cao hơn 27 lần. Tuy nhiên, trên bộ dữ liệu dày Pumsb, sự khác biệt về số lượng ứng viên và thời gian chạy giữa FWUCWP_No2 và FWUCWP là không đáng kể.



Hình 2.28. Tác động của từng chiến lược cắt tỉa đến hiệu năng thời gian chạy trên bộ dữ liệu Pumsb.



Hình 2.29. Tác động của từng chiến lược cắt tỉa đến hiệu năng thời gian chạy trên bộ dữ liệu Retail.



Hình 2.30. Tác động của từng chiến lược cắt tỉa đến hiệu năng thời gian chạy trên bộ dữ liệu Kosarak.

Trong khi đó, chiến lược 1 đóng vai trò tổng quát hơn. Là chiến lược cuối cùng và toàn diện nhất, WSR có nhiệm vụ xử lý các ứng viên phức tạp đã vượt qua hai chiến lược trước đó. Việc vô hiệu hóa chiến lược 1 trong thuật toán FWUCWP_No1 luôn dẫn đến sự gia tăng cả về số lượng ứng viên và thời gian chạy trên tất cả các bộ dữ liệu, mặc dù mức độ ảnh hưởng là khác nhau. Trên bộ dữ liệu Kosarak, tác động của WSR là đặc biệt rõ rệt, trong khi trên Pumsb và Retail, ảnh hưởng này là ít hơn. Điều này cho thấy chiến lược 1 hoạt động như một bước kiểm tra cuối cùng nhằm đảm bảo tính chính xác và góp phần cải thiện hiệu năng tổng thể.

Bên cạnh đó, chiến lược 3 được xác định là thành phần có hiệu quả thấp nhất trong cả ba chiến lược. Dữ liệu từ FWUCWP_No3 cho thấy việc loại bỏ Enclosure chỉ gây ra những thay đổi nhỏ về hiệu năng trên tất cả các bộ dữ liệu. Về mặt lý thuyết, chiến lược 3 được thiết kế như một lớp bộ nhớ đệm một phần cho chiến lược 1. Tuy nhiên, thực nghiệm cho thấy giả thuyết này không mang lại lợi ích đáng kể trong thực tế. Với các bộ dữ liệu thưa, có rất ít Enclosure của các mục được tạo ra. Trong khi trên bộ dữ liệu dày với nhiều Enclosure hơn, chiến lược 3 tuy giúp giảm

được việc kiểm tra của chiến lược 1 nhưng quá trình này tiêu tốn rất ít thời gian nhờ sự hiệu quả của cấu trúc WUNList. Điều này cũng giải thích vì sao WSR kém hiệu quả hơn trên các bộ dữ liệu này, do lớp Enclosure đã hỗ trợ sẵn cho quá trình kiểm tra. Ngoài ra, việc duy trì cấu trúc Enclosure còn phát sinh chi phí thời gian.

Mặc dù mức độ hiệu quả của từng chiến lược riêng lẻ phụ thuộc vào đặc tính của bộ dữ liệu, nhưng thuật toán FWUCWP hoàn chỉnh vẫn luôn đạt được hiệu năng tốt trong tất cả các kịch bản thực nghiệm. Điều này chứng minh tính hiệu quả của việc kết hợp nhiều chiến lược cắt tĩa trong phương pháp được đề xuất.

Bảng 2.13. Tác động của từng chiến lược cắt tĩa đến số lượng các ứng viên được tạo ra trên bộ dữ liệu Pumsb

<i>minwus (%)</i>	FWUCWP	FWUCWP_No1	FWUCWP_No2	FWUCWP_No3
70	271,838	271,875	271,993	271,838
65	558,815	558,905	558,927	558,815
60	1,245,802	1,246,045	1,245,964	1,245,802
55	3,290,031	3,290,614	3,290,203	3,290,031
50	8,783,515	8,784,903	8,783,819	8,783,515

Bảng 2.14. Tác động của từng chiến lược cắt tĩa đến số lượng các ứng viên được tạo ra trên bộ dữ liệu Retail

<i>minwus (%)</i>	FWUCWP	FWUCWP_No1	FWUCWP_No2	FWUCWP_No3
0.2	2,427	2,427	463,537	2,427
0.1	8,396	8,399	2,287,425	8,396
0.06	20,174	20,199	5,990,878	20,174
0.02	179,358	180,090	22,456,491	179,358
0.01	998,088	1,009,388	40,117,888	998,088

Bảng 2.15. Tác động của từng chiến lược cắt tĩa đến số lượng các ứng viên được tạo ra trên bộ dữ liệu Kosarak

<i>minwus</i> (%)	FWUCWP	FWUCWP_No1	FWUCWP_No2	FWUCWP_No3
0.2	38,411	41,403	208,652	38,411
0.17	124,811	156,920	398,789	124,811
0.14	249,830	315,569	643,795	249,830
0.1	722,933	841,845	1,487,196	722,933
0.09	1,772,553	1,971,740	2,704,968	1,772,553

2.4.4.6 Phân tích tính cô đọng của FWUCP

Để đánh giá lợi ích của việc khai thác các mẫu đóng, phần này so sánh số lượng mẫu được phát hiện giữa thuật toán FWUCWP (khai thác các FWUCP) và thuật toán dFWUNL (khai thác toàn bộ các FWUP). Các thực nghiệm được tiến hành trên nhiều tập dữ liệu với các ngưỡng *minwus* khác nhau. Kết quả được tổng hợp trong Bảng 2.16. Ta thấy số lượng FWUCP luôn ít hơn hoặc bằng số lượng FWUP, qua đó khẳng định tính cô đọng của biểu diễn mẫu đóng. Trên các tập dữ liệu dày đặc, mức độ giảm là đặc biệt rõ rệt. Chẳng hạn, trên tập dữ liệu Pumsb với ngưỡng *minwus* bằng 50%, thuật toán dFWUNL sinh ra hơn 171 triệu FWUP, trong khi FWUCWP chỉ tạo ra 7.2 triệu FWUCP, tương ứng với mức giảm 95.76%. Sự nén này loại bỏ khoảng 164 triệu mẫu con dư thừa có cùng giá trị *wus* với các mẫu cha của chúng. Tương tự, trên tập dữ liệu Accidents, biểu diễn đóng đã loại bỏ hơn 2 triệu mẫu dư thừa.

Trên các tập dữ liệu thưa, chẳng hạn như Kosarak và Retail, mức giảm lần lượt là 29.55% và 21.46%. Những kết quả này cho thấy ngay cả trong các phân bố thưa, một tỷ lệ đáng kể các FWUP vẫn là dư thừa. Một trường hợp đặc biệt xảy ra với tập dữ liệu Chainstore, có số lượng FWUCP bằng với số lượng FWUP (mức giảm 0%). Điều này cho thấy tất cả các FWUP đều là mẫu đóng trong cấu hình được đánh giá, tức là không tồn tại mẫu cha nào có cùng giá trị *wus*. Trong những trường hợp như

vậy, tập mẫu đóng bảo toàn hoàn toàn tập FWUP mà không đạt được sự nén nào.

Bảng 2.16. So sánh số lượng FWUCP và FWUP được tạo ra

CSDL	<i>minwus</i> (%)	FWUCPs	FWUPs	Giảm (%)
Pumsb	50	7,291,181	171,796,484	95.76
Accidents	8	19,101,342	21,457,710	10.98
Retail	0.01	182,242	232,036	21.46
BMS-POS	0.03	1,743,701	1,917,179	9.05
Kosarak	0.09	1,051,717	1,492,912	29.55
Chainstore	0.06	3,106	3,106	0

Các thực nghiệm này chứng minh rằng việc khai thác toàn bộ các FWUP thường tạo ra một tập kết quả quá lớn do tính dư thừa. Biểu diễn FWUCP cô đọng đầu ra này thành một tập nhỏ hơn rất nhiều nhưng vẫn bảo toàn toàn bộ tri thức, đồng thời cải thiện đáng kể hiệu quả lưu trữ và khả năng mở rộng của hệ thống.

2.5 Kết luận chương

Chương 2 đã tập trung giải quyết bài toán khai thác mẫu trọng số hữu ích phổ biến và mẫu trọng số hữu ích phổ biến đóng từ CSDL định lượng động với trọng số của các mục không cố định mà biến động theo thời gian. Đây là một thách thức lớn hơn so với các mô hình khai thác truyền thống nhưng phản ánh sát thực tế của các hệ thống dữ liệu hiện đại. Để giải quyết vấn đề này, luận án đã đề xuất hai thuật toán nền tảng là dFWUT và thuật toán dFWUNL. Thuật toán dFWUT thể hiện ưu điểm nổi bật về tính đơn giản trong cài đặt và chi phí thấp trên các CSDL thưa, với chiều dài của các Tidset tương đối ngắn. Tuy nhiên, trên các dữ liệu dày đặc, hiệu năng tính toán của dFWUT bị suy giảm đáng kể do chi phí bùng nổ của phép giao tập hợp. Trong khi đó, thuật toán dFWUNL nổi bật với việc sử dụng cấu trúc WUNList cải tiến giúp tối ưu hóa quá trình nén dữ liệu và tốc độ tính toán. Hiệu quả của dFWUNL càng được củng cố thông qua các chiến lược tối ưu hóa như tính toán trước mẫu hai

phần tử và xác định nhanh giá trị wus , giúp thuật toán đạt hiệu suất cao và duy trì sự ổn định cả về thời gian thực thi lẫn mức sử dụng bộ nhớ ngay cả trong trường hợp ngưỡng khai thác nhỏ.

Trên cơ sở kết quả khai thác FWUP, chương này tiếp tục mở rộng sang bài toán khai thác mẫu trọng số hữu ích phổ biến đóng từ CSDL định lượng động để giảm thiểu sự bùng nổ số lượng mẫu mà vẫn bảo toàn được thông tin quan trọng và phát triển thuật toán FWUCWP. Bằng việc tận dụng cấu trúc WUNList và tích hợp ba chiến lược cắt tĩa bao gồm cắt tĩa dựa trên quan hệ cấu trúc tổ tiên WUNList, cắt tĩa kết hợp twu tối đa và cắt tĩa dựa trên khái niệm Enclosure, FWUCWP đã chứng minh được khả năng thu hẹp không gian tìm kiếm một cách hiệu quả. Các kết quả thực nghiệm trên nhiều loại dữ liệu khác nhau cho thấy FWUCWP đạt hiệu quả tốt hơn so với phương pháp đối sánh, đặc biệt trên các CSDL thưa. Tuy nhiên, việc xác định mẫu đóng và áp dụng các điều kiện cắt tĩa bổ sung cũng tạo thêm chi phí xử lý. Điều này dẫn đến tính hiệu quả vẫn còn phụ thuộc vào đặc trưng dữ liệu và mức độ rút gọn của tập mẫu đóng.

Những đóng góp về mặt thuật toán và cấu trúc dữ liệu trong chương này không chỉ giải quyết hiệu quả bài toán trong CSDL định lượng động mà còn tạo dựng một nền tảng lý thuyết và kỹ thuật vững chắc. Đây chính là tiền đề quan trọng để luận án tiếp tục mở rộng nghiên cứu, ứng dụng các cấu trúc dữ liệu này vào bài toán khai thác mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động, là một thách thức lớn hơn về mặt thời gian và tài nguyên sẽ được trình bày chi tiết trong Chương 3.

CHƯƠNG 3: KHAI THÁC MẪU TRỌNG SỐ HỮU ÍCH PHỔ BIẾN TRÊN LUỒNG DỮ LIỆU ĐỊNH LƯỢNG ĐỘNG

3.1 Giới thiệu bài toán

Tiếp nối các nghiên cứu tại Chương 2 về khai thác mẫu trọng số hữu ích phổ biến từ CSDL định lượng động, Chương 3 mở rộng phạm vi nghiên cứu sang một bối cảnh phức tạp và thực tế hơn đó là luồng dữ liệu định lượng động. Trong kỷ nguyên số, dữ liệu không chỉ dừng lại ở các tập tin lưu trữ tĩnh mà liên tục được sinh ra từ các hệ thống giao dịch trực tuyến, mạng cảm biến, và các nền tảng thương mại điện tử, mạng xã hội, dưới dạng các luồng dữ liệu với tốc độ cao và khối lượng vô hạn. Ví dụ, các nền tảng thương mại điện tử liên tục ghi nhận các đơn hàng mới, các hệ thống thanh toán điện tử phát sinh hàng nghìn giao dịch mỗi giây, trong khi các hệ thống giám sát mạng liên tục thu thập các sự kiện và nhật ký hoạt động. Việc khai thác các FWUP từ luồng dữ liệu này không chỉ đơn thuần là bài toán thống kê mà còn là yêu cầu cấp thiết để các tổ chức, doanh nghiệp có thể theo dõi và đưa ra các quyết định chiến lược một cách nhanh chóng.

Khác với các mô hình truyền thống, trong luồng dữ liệu định lượng động, cả số lượng mục trong giao dịch và trọng số của chúng đều biến động liên tục theo thời gian. Khai thác FWUP cho phép xác định những mẫu không chỉ xuất hiện thường xuyên mà còn có giá trị cao, giúp phản ánh chính xác hơn hiệu quả kinh tế hoặc tầm quan trọng của các mẫu dữ liệu tại thời điểm hiện tại. Sự chuyển dịch từ CSDL định lượng động sang luồng dữ liệu đặt ra những thách thức mới và yêu cầu khắt khe. Các hệ thống thực tế đòi hỏi việc cập nhật dữ liệu phải diễn ra trong thời gian thực, dữ liệu chỉ được duyệt qua một lần và tài nguyên bộ nhớ phải được kiểm soát chặt chẽ do không thể lưu trữ toàn bộ lịch sử luồng dữ liệu. Hơn nữa, trong phân tích luồng dữ liệu, người dùng thường quan tâm nhiều hơn đến các thông tin mới nhất. Mô hình cửa sổ trượt đã trở thành một lựa chọn ưu tiên nhờ khả năng cập nhật dữ liệu liên tục bằng cách thêm các giao dịch mới và loại bỏ các giao dịch cũ, từ đó phản ánh sát thực tế các xu hướng tiêu dùng hoặc hành vi hệ thống tại thời điểm hiện tại. Tuy nhiên,

các thuật toán đã đề xuất trong Chương 2 chỉ được thiết kế cho dữ liệu cố định giao dịch, không có cơ chế tự động cập nhật và loại bỏ dữ liệu lỗi thời theo cửa sổ trượt, dẫn đến việc không thể đáp ứng các đặc trưng của luồng dữ liệu. Điều này đặt ra yêu cầu mở rộng bài toán khai thác FWUP sang môi trường luồng dữ liệu.

Một thách thức lớn khác trong khai thác FWUP trên luồng dữ liệu là việc thiết lập ngưỡng hỗ trợ trọng số hữu ích tối thiểu. Do đặc tính biến động liên tục, việc xác định ngưỡng *minwus* cố định là một vấn đề khó khăn đối với người dùng phổ thông. Nếu ngưỡng quá nhỏ có thể dẫn đến việc sinh ra quá nhiều mẫu hoặc nếu ngưỡng quá lớn thì có quá ít mẫu thoả mãn khi dữ liệu thay đổi đột ngột. Mặt khác, trong một số hệ thống thông minh thường chỉ quan tâm đến các mẫu quan trọng nhất, giúp nhanh chóng đưa ra các gợi ý, dự báo xu hướng hoặc phát hiện sớm các hành vi bất thường. Để giải quyết hạn chế này, bài toán khai thác top- k FWUP trên luồng dữ liệu định lượng động cũng được đề xuất. Thay vì dựa vào một ngưỡng định trước, bài toán top- k cho phép người dùng xác định số lượng mẫu có độ hỗ trợ trọng số hữu ích cao nhất cần quan tâm. Việc chỉ định số k là đơn giản và dễ tiếp cận hơn nhiều so với *minwus*.

Nội dung Chương 3 trình bày bài toán khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt với trọng số các mục thay đổi. Từ đó, luận án phát triển thuật toán FWUDS-CT sử dụng cấu trúc Circular Tidset để quản lý hiệu quả sự biến động của các giao dịch trong mô hình cửa sổ trượt. Song song với đó, thuật toán FWUDS-DWT cũng được giới thiệu với việc sử dụng cấu trúc DSWUN-tree và WUNList được kế thừa và phát triển từ các nghiên cứu ở Chương 2. Kết quả thực nghiệm trên nhiều tập dữ liệu khác nhau cho thấy thuật toán FWUDS-DWT đạt được sự vượt trội đáng kể về khả năng cập nhật dữ liệu mới, thời gian khai thác nhanh chóng cùng khả năng mở rộng linh hoạt. Bên cạnh việc khai thác tập mẫu đầy đủ, chương này còn phát triển thuật toán TKFWUDS chuyên biệt cho bài toán khai thác top- k FWUP, tận dụng tối đa sức mạnh của cấu trúc DSWUN-tree kết hợp cùng các chiến lược cắt tỉa mạnh mẽ dựa trên ngưỡng hỗ trợ trọng số hữu ích động. Các kết quả nghiên cứu trình bày trong chương này được công bố trong các công trình [IV] và [VI]. Trong đó, công trình [IV] giải quyết bài toán khai thác top- k FWUP trên

luồng dữ liệu định lượng động và công trình [VI] phát triển các thuật toán khai thác FWUP trên luồng dữ liệu định lượng động theo mô hình cửa sổ trượt. Những đóng góp này không chỉ mang lại hiệu quả về mặt tính toán mà còn cung cấp một giải pháp toàn diện cho việc phân tích tri thức trên các luồng dữ liệu phức tạp.

3.2 Một số định nghĩa

Định nghĩa 3.1 (Luồng dữ liệu định lượng động). Một luồng dữ liệu định lượng động, ký hiệu là DS , là danh sách các giao dịch định lượng động được sắp xếp theo thứ tự thời gian, $DS = [t_1, t_2, \dots, t_m]$. DS không giới hạn số lượng các giao dịch nên các giao dịch $t_{m+1}, t_{m+2}, \dots$ có thể được thêm vào sau giao dịch t_m .

Định nghĩa 3.2 (Mô hình cửa sổ trượt). Một cửa sổ trượt trên luồng dữ liệu được xác định bởi hai giá trị là kích thước cửa sổ và kích thước pane. Trong đó một pane chứa số lượng giao dịch cố định và một cửa sổ chứa nhiều pane hiện tại. Ta ký hiệu $|P|$ là số lượng giao dịch có trong một pane, và $|W|$ là số lượng giao dịch trong cửa sổ, với w là kích thước cửa sổ thì $|W| = |P| * w$. Một DS được chia thành nhiều pane, $DS = [P_1, P_2, \dots, P_l]$. Cập nhật cửa sổ là quá trình xóa pane cũ nhất và thêm dữ liệu từ pane mới vào cửa sổ. Quá trình này được tiến hành cho đến khi không còn dữ liệu được thêm vào.

Thuật ngữ “động” trong Chương 3 mang ý nghĩa rộng hơn so với Chương 2. Nếu ở Chương 2, tính động chỉ xuất phát từ sự thay đổi trọng số của các mục trong CSDL định lượng, thì trong môi trường luồng dữ liệu, bài toán đồng thời phải xử lý hai dạng biến động. Thứ nhất, trọng số của các mục có thể thay đổi theo thời gian, làm ảnh hưởng trực tiếp đến các độ đo trọng số hữu ích của mẫu. Thứ hai, tập giao dịch trong cửa sổ khai thác cũng liên tục thay đổi do cơ chế cửa sổ trượt, khi các giao dịch mới được bổ sung và các giao dịch cũ bị loại bỏ. Sự kết hợp của hai yếu tố này làm cho giá trị của các mẫu không chỉ phụ thuộc vào sự xuất hiện của chúng trong luồng dữ liệu mà còn phụ thuộc vào sự biến động của trọng số theo thời gian. Do đó, các thuật toán khai thác FWUP trên luồng dữ liệu định lượng động phải đồng thời giải quyết bài toán cập nhật dữ liệu theo thời gian thực và thích ứng với sự thay đổi

của trọng số, trong khi vẫn đảm bảo hiệu quả về thời gian xử lý và bộ nhớ sử dụng.

Ví dụ 3.1: Hình 3.1 hiển thị mô hình cửa sổ trượt của một luồng dữ liệu định lượng động DS_e với kích thước cửa sổ và kích thước pane lần lượt là 2 và 2. DS_e có tất cả sáu giao dịch $\{t_1, t_2, t_3, t_4, t_5, t_6\}$ và năm mục $\{A, B, C, D, E\}$ được chia thành hai cửa sổ $\{W_1, W_2\}$. Mỗi cửa sổ chứa hai pane và mỗi pane có hai giao dịch. Danh sách tập các trọng số của các mục hiển thị ở Bảng 3.1. Trọng số của các mục trong giao dịch t_1, t_2, t_3 là $dw_1 = \{0.2, 0.4, 0.2, 0.4, 0.5\}$, của giao dịch t_4, t_5, t_6 là $dw_2 = \{0.5, 0.3, 0.4, 0.6, 0.6\}$.



Hình 3.1. Luồng dữ liệu định lượng động ví dụ (DS_e).

Bảng 3.1. Tập trọng số của các mục

Tập trọng số	A	B	C	D	E
dw_1	0.2	0.4	0.2	0.4	0.5
dw_2	0.5	0.3	0.4	0.6	0.6

Trong môi trường luồng dữ liệu, các độ đo cần được tính toán trên cửa sổ hiện hành thay vì trên toàn bộ dữ liệu. Do đó, các định nghĩa được mở rộng như sau.

Định nghĩa 3.3 (Độ hỗ trợ trọng số hữu ích trong một cửa sổ). Gọi $sumtwu(W_i) = \sum_{t_j \in W_i} twu(t_j)$ là tổng giá trị twu của các giao dịch trong cửa sổ W_i . Độ hỗ trợ trọng số hữu ích của mẫu X trong cửa sổ W_i được xác định như sau:

$$wus(X, W_i) = \frac{\sum_{t_j \in T(X, W_i)} twu(t_j)}{sumtwu(W_i)} \quad (3.1)$$

trong đó, $T(X, W_i)$ là tập các giao dịch trong cửa sổ W_i có chứa mẫu X , $twu(t_j)$ là

giá trị trọng số hữu ích giao dịch của giao dịch t_j .

Ví dụ 3.2: Giá trị wus của mẫu AC thuộc cửa sổ W_1 trong DS_e là $wus(AC, W_1) = [twu(t_1) + twu(t_3) + twu(t_4)]/sumtwu(W_1) = (0.48 + 0.37 + 0.97)/2.52 \approx 0.72$.

Định nghĩa 3.4 (Mẫu trọng số hữu ích phổ biến trong một cửa sổ). Cho một ngưỡng trọng số hữu ích tối thiểu $minwus$ và một mẫu $X \subseteq I$ trong cửa sổ W_i , nếu $wus(X, W_i) \geq minwus$ thì X được gọi là mẫu trọng số hữu ích phổ biến trong cửa sổ W_i .

Định nghĩa 3.5 (Bài toán khai thác mẫu trọng số hữu ích phổ biến trên luồng dữ liệu). Bài toán khai thác mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động với trọng số các mục có thể thay đổi, sử dụng mô hình cửa sổ trượt, là tìm tất cả các FWUP có độ hỗ trợ trọng số hữu ích lớn hơn hoặc bằng ngưỡng $minwus$ do người dùng chỉ định trong mỗi cửa sổ dữ liệu.

Định nghĩa 3.6 (Top- k mẫu trọng số hữu ích phổ biến trong một cửa sổ). Top- k mẫu trọng số hữu ích phổ biến trong cửa sổ W_i của luồng dữ liệu, ký hiệu là $TK(W_i)$, là tập hợp đúng k FWUP có độ hỗ trợ trọng số hữu ích lớn nhất trong W_i .

Định nghĩa 3.7 (Bài toán khai thác top- k mẫu trọng số hữu ích phổ biến trên luồng dữ liệu). Cho một luồng dữ liệu định lượng động và một số nguyên dương k , bài toán khai thác top- k mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt là việc tìm ra chính xác k FWUP có độ hỗ trợ trọng số hữu ích lớn nhất trong mỗi cửa sổ dữ liệu.

Ví dụ 3.3: Với ngưỡng $k = 3$, top-3 FWUP trong cửa sổ W_1 của DS_e ở Hình 3.1 là: $TK_3(W_1) = \{C: 1, D: 0.85, CD: 0.85\}$.

3.3 Thuật toán khai thác mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động

Các thuật toán được đề xuất trong Chương 2 đã giải quyết hiệu quả bài toán khai thác FWUP trên CSDL định lượng động. Tuy nhiên, các phương pháp này đều

tiến hành trên dữ liệu được lưu trữ đầy đủ và có thể quét nhiều lần khi cần thiết. Do đó, chúng không phù hợp với môi trường luồng dữ liệu, nơi dữ liệu được sinh liên tục với tốc độ cao và không thể lưu trữ toàn bộ trong bộ nhớ. Điều này đặt ra yêu cầu phải xây dựng các cấu trúc dữ liệu mới có khả năng cập nhật nhanh khi cửa sổ trượt dịch chuyển, duy trì tính hiệu quả cho việc tính toán độ hỗ trợ trọng số hữu ích khi cả trọng số và giao dịch cùng thay đổi, hạn chế tối đa số lần quét dữ liệu và sử dụng bộ nhớ hiệu quả. Để giải quyết các vấn đề trên, luận án đề xuất hai phương pháp khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt. Hướng tiếp cận thứ nhất là thuật toán FWUDS-CT sử dụng cấu trúc Circular Tidset nhằm quản lý hiệu quả định danh giao dịch trong môi trường luồng dữ liệu vô hạn và hỗ trợ cập nhật nhanh khi cửa sổ thay đổi. Hướng tiếp cận thứ hai là thuật toán FWUDS-DWT dựa trên cấu trúc DSWUN-tree kết hợp với cấu trúc WUNList để tận dụng khả năng nén dữ liệu và tính toán trực tiếp độ hỗ trợ trọng số hữu ích của các mẫu. Cấu trúc DSWUN-tree còn có khả năng tái sử dụng dữ liệu, cho phép việc thêm và xóa nhanh chóng với chỉ một lần quét CSDL. Hai thuật toán được thiết kế theo các nguyên lý khác nhau nhằm đánh giá hiệu quả của hai hướng tiếp cận trong môi trường luồng dữ liệu định lượng động.

3.3.1 Thuật toán FWUDS-CT

Thuật toán được đề xuất đầu tiên để khai thác FWUP trên luồng dữ liệu định lượng động có tên là FWUDS-CT, sử dụng cấu trúc dữ liệu Circular Tidset. Cấu trúc Tidset mặc dù được thiết kế cho CSDL tĩnh, cố định số lượng giao dịch, nhưng ta hoàn toàn có thể điều chỉnh để nó hoạt động được trên luồng dữ liệu bởi vì đây là thuật toán quét dữ liệu một lần. Tuy nhiên, một trong những hạn chế của cấu trúc dữ liệu Tidset đó là giá trị *Tid* sẽ tăng liên tục, mà trong luồng dữ liệu, số lượng giao dịch là không bị giới hạn. Điều này dẫn đến giá trị *Tid* sẽ có thể rất lớn và việc xử lý các số lớn là phức tạp, đòi hỏi nhiều tài nguyên. Do đó, luận án phát triển một cấu trúc dữ liệu mới, có tên là Circular Tidset, ký hiệu là CTset, là cải tiến của cấu trúc Tidset, để giải quyết hạn chế này, phù hợp với luồng dữ liệu.

3.3.1.1 Cấu trúc dữ liệu Circular Tidset

Định nghĩa 3.8 (Định danh giao dịch cục bộ). Định danh giao dịch cục bộ, ký hiệu là $lcTid$, là giá trị định danh của giao dịch trong cửa sổ hiện tại. Ban đầu, $lcTid$ có giá trị bằng 0. Nếu có một giao dịch mới được thêm vào thì $lcTid$ của giao dịch đó được tính như sau:

$$lcTid = \begin{cases} lcTid + 1, & \text{if } lcTid < |W| \\ 1, & \text{if } lcTid = |W| \end{cases} \quad (3.2)$$

trong đó, $|W|$ là kích thước của cửa sổ tính theo số lượng giao dịch.

Ví dụ 3.4: Hình 3.2 mô tả DS_e sau khi đã thêm các giá trị $lcTid$. Cửa sổ W_1 có 4 giao dịch là t_1, t_2, t_3 và t_4 với giá trị $lcTid$ lần lượt là 1, 2, 3 và 4. Cửa sổ W_2 có 4 giao dịch là t_3, t_4, t_5 và t_6 với giá trị $lcTid$ lần lượt là 3, 4, 1 và 2. Dễ dàng thấy rằng, giá trị $lcTid$ tối đa không vượt quá $|W|$.

		Giao dịch	Tid	lcTid	A	B	C	D	E	Tập trọng số
W_2 $(P_2 - P_3)$	W_1 $(P_1 - P_2)$	t_1	1	1	1	0	2	2	1	dw_1
		t_2	2	2	0	2	1	2	2	
		t_3	3	3	2	0	1	0	1	
		t_4	4	4	1	0	3	2	0	
	P_3	t_5	5	1	0	1	1	1	0	dw_2
		t_6	6	2	2	0	2	3	1	

Hình 3.2. DS_e với giá trị Tid và $lcTid$.

Định nghĩa 3.9 (Circular Tidset). Circular Tidset, ký hiệu là $CTset$, của một mẫu X trong cửa sổ W_i là danh sách lưu trữ các $lcTid$ của các giao dịch có chứa X trong cửa sổ W_i .

$$CTset(X, W_i) = \{lcTid(t) | t \in W_i, X \subseteq t\} \quad (3.3)$$

Giá trị độ hỗ trợ trọng số hữu ích của mẫu X trong cửa sổ W_i được tính dựa trên $CTset(X, W_i)$ như sau.

$$wus(X, W_i) = \frac{\sum_{j \in CTset(X, W_i)} twu(t^{(j)})}{sumtwu(W_i)}, \text{ với } t^{(j)} \in W_i \text{ và } lcTid(t^{(j)}) = j. \quad (3.4)$$

Ví dụ 3.5: Theo Hình 3.2, trong cửa sổ W_1 , mẫu A xuất hiện trong ba giao dịch là t_1 , t_3 , và t_4 nên $CTset(A, W_1) = \{1, 3, 4\}$. Tương tự, trong cửa sổ W_2 , mẫu A xuất hiện trong ba giao dịch là t_3 , t_4 và t_6 nên $CTset(A, W_2) = \{3, 4, 2\}$. Ta có $wus(A, W_2) = (twu(t_3) + twu(t_4) + twu(t_6))/sumtwu(W_2) = (0.37 + 0.97 + 1.05)/2.82 \approx 0.85$.

3.3.1.2 Duy trì CTset trên luồng dữ liệu

Sau khi hoàn thành việc khởi tạo cửa sổ đầu tiên, quá trình trượt khung cửa sổ sẽ được tiến hành. Lúc này, dữ liệu trong cửa sổ sẽ thay đổi, các giao dịch trong pane cũ nhất bị xoá đi và các giao dịch mới được thêm vào. CTset của các mục cần được cập nhật để đảm bảo lưu trữ dữ liệu trong cửa sổ mới nhất hiện tại. Đối với thao tác thêm một pane mới vào cửa sổ, $lcTid$ của giao dịch mới sẽ được tính theo **Định nghĩa 3.8**. Tiếp theo, $lcTid$ sẽ được thêm vào cuối CTset của các mục xuất hiện trong giao dịch đó. Như vậy, thứ tự thông tin các giao dịch mới nhất trong CTset sẽ nằm ở cuối danh sách, còn đầu danh sách sẽ là các giao dịch cũ. Thao tác xoá một pane cũ nhất ra khỏi cửa sổ là phức tạp hơn và được thực hiện trước thao tác thêm. Ta có thể xác định được các $lcTid$ của pane cũ nhất này để tiến hành xoá như sau.

Định lý 3.1. Giả sử $lcTid$ của giao dịch mới nhất trong cửa sổ hiện tại có giá trị là n (cửa sổ đã đầy), thì pane cũ nhất trong cửa sổ này sẽ chứa các giao dịch có $lcTid$ nằm trong $[n + 1, n + |P|]$ nếu $n < |W|$ và nằm trong $[1, |P|]$ nếu $n = |W|$.

Chứng minh. Dễ thấy, với $n = |W|$, $lcTid$ của các giao dịch trong cửa sổ này lần lượt là $\{1, 2, \dots, n - 1, n\}$, nên pane cũ nhất sẽ bao gồm các giao dịch có $lcTid$ là $[1, 2, \dots, |P| - 1, |P|]$. Tương tự, với $n < |W|$, cửa sổ hiện tại có các $lcTid$ lần lượt là $\{n + 1, n + 2, \dots, |W|, 1, 2, \dots, |n|\}$, nên pane cũ nhất bao gồm các $lcTid$ là $\{n + 1, n + 2, \dots, n + |P|\}$.

Như vậy, ta sẽ tiến hành duyệt CTset của tất cả các mục xuất hiện trong cửa sổ từ đầu đến cuối danh sách. Với mỗi mục, các $lcTid$ thoả mãn **Định lý 3.1** thuộc pane cũ nhất sẽ bị xoá khỏi CTset. Ngược lại ta sẽ kết thúc và chuyển sang mục tiếp

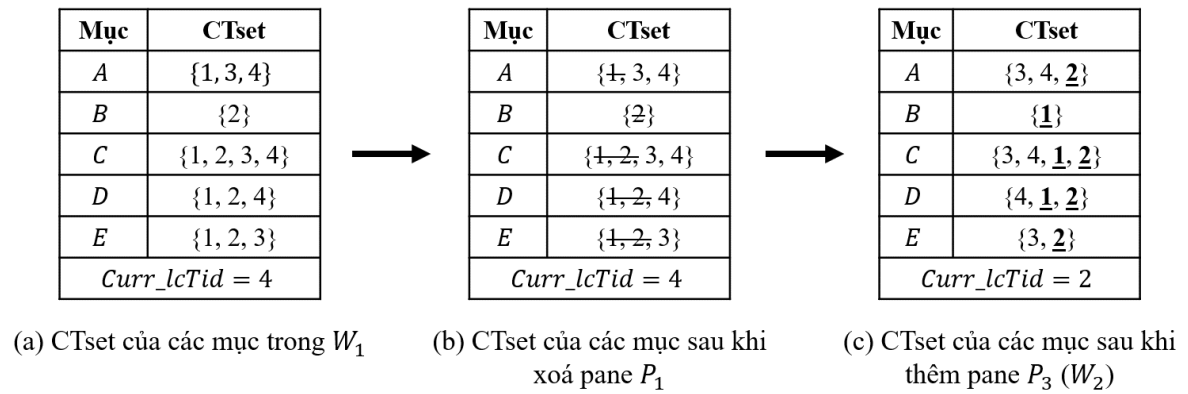
theo vì các giao dịch phía sau chắc chắn không thuộc pane cũ nhất. Với việc phải thêm và xoá ở hai đầu danh sách, cấu trúc dữ liệu như danh sách liên kết được sử dụng để lưu trữ CTset. Thuật toán của hai thao tác này được mô tả ở Hình 3.3.

Thuật toán 3.1. Thêm và xoá pane sử dụng CTset
Procedure InsertNewPane ($P, Curr_lcTid, CTS, w$)
Input: Pane mới nhất P , giá trị $lcTid$ hiện tại $Curr_lcTid$, tập CTset của các mục CTS và kích thước cửa sổ w . Output: CTS được cập nhật cho khung cửa sổ mới.
1. foreach giao dịch t trong P do 2. $Curr_lcTid \leftarrow Curr_lcTid + 1$ 3. if $lcTid > w * P $ then 4. $Curr_lcTid \leftarrow 1$ 5. foreach mục i trong t do 6. Thêm $Curr_lcTid$ vào cuối $CTS[i]$
Procedure RemoveOldestPane ($Curr_lcTid, CTS, w, p$)
Input: Giá trị $lcTid$ hiện tại $Curr_lcTid$, tập CTset của các mục CTS , kích thước cửa sổ w và kích thước pane p . Output: CTS được cập nhật cho khung cửa sổ mới.
1. $rmFrom \leftarrow 1; rmTo \leftarrow p$ 2. if $Curr_lcTid < w * p$ then 3. $rmFrom \leftarrow Curr_lcTid + 1$ 4. $rmTo \leftarrow Curr_lcTid + p$ 5. foreach CTset CT_i trong CTS do 6. foreach giá trị $lcTid$ trong CT_i do 7. if $lcTid \geq rmFrom$ and $lcTid \leq rmTo$ then 8. Xoá $lcTid$ khỏi CT_i 9. else break

Hình 3.3. Thuật toán thêm và xoá pane sử dụng cấu trúc CTset.

Ví dụ 3.6: Cửa sổ W_1 trong DS_e bao gồm hai pane với bốn giao dịch ($|W| =$

4). CTset của các mục trong W_1 được xác định như Hình 3.4a, và giá trị $lcTid$ mới nhất trong W_1 là $Curr_lcTid = 4$. Quá trình trượt khung cửa sổ từ W_1 sang W_2 , CTset của các mục được cập nhật như sau. Đầu tiên, ta tiến hành xóa pane cũ nhất P_1 trong W_1 . Vì $Curr_lcTid = |W|$ nên pane cũ nhất bao gồm các $lcTid$ có giá trị là 1 và 2 sẽ bị xóa trong CTset của các mục (Hình 3.4b). Tiếp theo, pane mới P_3 gồm hai giao dịch t_5, t_6 với giá trị $lcTid$ lần lượt là 1 và 2 được thêm vào. Giao dịch t_5 có các mục là B, C và D nên giá trị $lcTid(t_5) = 1$ được thêm vào cuối CTset của các mục này. Tương tự, $lcTid(t_6) = 2$ được thêm vào CTset của các mục là A, C, D và E . Kết thúc quá trình, ta thu được CTset của các mục trong W_2 (Hình 3.4c), và $Curr_lcTid = 2$.



Hình 3.4. Quá trình cập nhật CTset của các mục khi trượt khung cửa sổ từ W_1 sang W_2 trong DS_e .

3.3.1.3 Phương pháp tuyến tính để giao các CTset

CTset lưu trữ thông tin giao dịch của các mẫu trong cửa sổ hiện tại. Tương tự cấu trúc Tidset, ta có thể xác định được CTset của một mẫu mới bằng cách giao CTset của hai mẫu có cùng tiền tố với nhau. Với cách tiếp cận ngây thơ thì độ phức tạp của hàm giao này là $O(mn)$, trong đó, m và n lần lượt là độ dài của hai CTset. Cấu trúc Tidset truyền thống đảm bảo tính đơn điệu tăng dần nên ta dễ dàng tiến hành giao hai Tidset với độ phức tạp là $O(m + n)$. CTset không có tính chất này, tuy nhiên, ta vẫn có thể đạt được độ phức tạp tuyến tính cho hàm giao CTset như sau.

Định lý 3.2. Cho t_x và t_y là hai giao dịch khác nhau trong cùng cửa sổ hiện tại với $lcTid$ lần lượt là $lcTid_x$ và $lcTid_y$, và giá trị $lcTid$ của giao dịch mới nhất

trong cửa sổ này là $Curr_lcTid$. Giao dịch t_y là mới hơn giao dịch t_x khi (1) $lcTid_x < lcTid_y \leq Curr_lcTid$ hoặc (2) $Curr_lcTid < lcTid_x < lcTid_y$ hoặc (3) $lcTid_x > Curr_lcTid$ và $lcTid_y \leq Curr_lcTid$.

Chứng minh. Với $Curr_lcTid = |W|$, giá trị $lcTid$ của các giao dịch trong cửa sổ theo thứ tự thời gian là $\{1, 2, \dots, Curr_lcTid - 1, Curr_lcTid\}$. Dễ thấy giao dịch t_y mới hơn giao dịch t_x khi $lcTid_x < lcTid_y \leq Curr_lcTid$.

Với $Curr_lcTid < |W|$, giá trị $lcTid$ của các giao dịch trong cửa sổ theo thứ tự thời gian là $\{Curr_lcTid + 1, Curr_lcTid + 2, \dots, |W|, 1, 2, \dots, Curr_lcTid\}$. Trong trường hợp này, cửa sổ có hai chu kỳ là $\{Curr_lcTid + 1, Curr_lcTid + 2, \dots, |W|\}$ thuộc chu kỳ cũ và $\{1, 2, \dots, Curr_lcTid\}$ thuộc chu kỳ mới. Nếu giao dịch t_x và t_y đều thuộc cùng một chu kỳ thì giao dịch t_y mới hơn giao dịch t_x khi $lcTid_x < lcTid_y \leq Curr_lcTid$ (cùng chu kỳ mới), hoặc $Curr_lcTid < lcTid_x < lcTid_y$ (cùng chu kỳ cũ).

Trường hợp hai giao dịch ở hai chu kỳ khác nhau thì giao dịch t_y mới hơn giao dịch t_x khi giao dịch t_y nằm trong chu kỳ mới, còn giao dịch t_x nằm trong chu kỳ cũ, hay $lcTid_x > Curr_lcTid$ và $lcTid_y \leq Curr_lcTid$.

Dựa trên **Định lý 3.2**, quá trình tạo CTset của mẫu mới được thực hiện hiệu quả với độ phức tạp tuyến tính và được mô tả trong Hình 3.5. Đầu vào của hàm $CTset_Intersection$ lần lượt là CTset CT_1 của mẫu Px , CTset CT_2 của mẫu Py , $Curr_lcTid$ là giá trị $lcTid$ hiện tại của giao dịch mới nhất, w và p là giá trị kích thước cửa sổ và kích thước pane. Đầu tiên, khởi tạo CTset CT_3 của mẫu Pxy là rỗng, ptr_1 và ptr_2 lần lượt lưu phần tử đầu tiên của CT_1 và CT_2 (dòng 1-2). Lần lượt so sánh hai giá trị $lcTid$ của ptr_1 và ptr_2 theo **Định lý 3.2**. Ở đây, để dễ dàng cho việc so sánh, kỹ thuật tăng $lcTid$ lên $|W| = w * p$ đơn vị nếu giá trị này nhỏ hơn hoặc bằng $Curr_lcTid$ được sử dụng. Điều này thể hiện là $lcTid$ này thuộc về giao dịch mới thêm vào để phân biệt với các giao dịch cũ hơn nhưng có $lcTid$ lớn hơn $Curr_lcTid$ (dòng 4-6). Tiếp theo, so sánh hai giá trị mới được cập nhật là $lcTid_1$ và

$lcTid_2$ theo các trường hợp (dòng 7-11):

1) Nếu $lcTid_1 = lcTid_2$, ta thêm giá trị $lcTid$ của ptr_1 vào CT_3 và chuyển qua phần tử tiếp theo của cả CT_1 và CT_2 .

2) Nếu $lcTid_1 < lcTid_2$, ta xét phần tử tiếp theo của CT_1 .

3) If $lcTid_1 > lcTid_2$, ta xét phần tử tiếp theo của CT_2 .

Quá trình này được thực hiện liên tục cho đến khi một trong hai CTset được duyệt hết. Kết quả cuối cùng ta thu được CTset của mẫu P_{xy} với độ phức tạp là $O(m + n)$.

Thuật toán 3.2. Giao hai CTset
Input: CTset của mẫu P_x CT_1 , CTset của mẫu P_y CT_2 , Giá trị $lcTid$ hiện tại $Curr_lcTid$, kích thước cửa sổ w và kích thước pane p
Output: CTset của mẫu P_{xy} CT_3
Procedure CTset_Intersection ($CT_1, CT_2, Curr_lcTid, w, p$) 1. $CT_3 \leftarrow \emptyset$ 2. $ptr_1 \leftarrow CT_1.first; ptr_2 \leftarrow CT_2.first$ 3. while $ptr_1 \neq NULL$ and $ptr_2 \neq NULL$ do 4. $lcTid_1 \leftarrow ptr_1.lcTid; lcTid_2 \leftarrow ptr_2.lcTid$ 5. if $lcTid_1 \leq Curr_lcTid$ then $lcTid_1 += w * p$ 6. if $lcTid_2 \leq Curr_lcTid$ then $lcTid_2 += w * p$ 7. if $lcTid_1 = lcTid_2$ then 8. Thêm $ptr_1.lcTid$ vào cuối CT_3 9. $ptr_1 \leftarrow ptr_1.next; ptr_2 \leftarrow ptr_2.next$ 10. else if $lcTid_1 < lcTid_2$ then $ptr_1 \leftarrow ptr_1.next$ 11. else $ptr_2 \leftarrow ptr_2.next$ 12. return CT_3

Hình 3.5. Thuật toán giao hai CTset.

3.3.1.4 Mô tả thuật toán

Thuật toán 3.3. Thuật toán FWUDS-CT
Input: Pane mới nhất P, giá trị $lcTid$ hiện tại $Curr_lcTid$, tập CTset của các mục CTS, kích thước cửa sổ w, kích thước pane p và ngưỡng $minwus$ Output: Tập tất cả các mẫu trọng số hữu ích phổ biến $FWUPs$ trong cửa sổ mới
1. RemoveOldestPane($Curr_lcTid, CTS, w, p$) 2. InsertNewPane($P, Curr_lcTid, CTS, w$) 3. Loại các mục có wus không nhỏ hơn $minwus$ và lưu vào C_1 4. $FWUPs \leftarrow C_1$ 5. MineFWUPs_CTset($C_1, Curr_lcTid, w, p$) 6. return $FWUPs$
Procedure MineFWUPs_CTset($C, Curr_lcTid, w, p$) 7. foreach $c_i \in C$ do 8. $C_{next} \leftarrow \emptyset$ 9. foreach $c_j \in C$ ($j > i$) do 10. $X \leftarrow c_i \cup c_j$ 11. $CTset(X) \leftarrow \mathbf{CTset_Intersection}(CTset(c_i), CTset(c_j), Curr_lcTid, w, p)$ 12. if $wus(X) \geq minwus$ then 13. $FWUPs \leftarrow X$ 14. $C_{next} \leftarrow \{X, CTset(X), wus(X)\}$ 15. if $C_{next} > 1$ then 16. MineFWUPs_CTset($C_{next}, Curr_lcTid, w, p$)

Hình 3.6. Thuật toán FWUDS-CT.

Thuật toán FWUDS-CT để khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt được mô tả ở Hình 3.6. Khi một pane dữ liệu mới được thêm vào, thuật toán sẽ trả về tất cả các FWUP trong cửa sổ mới thỏa mãn

ngưỡng *minwus* cho người dùng. Đầu tiên, thuật toán cập nhật các CTset trong cửa sổ mới bằng cách xóa pane cũ nhất bằng thủ tục *RemoveOldestPane* (dòng 1) và thêm pane mới bằng thủ tục *InsertNewPane* (dòng 2). Sau khi cập nhật dữ liệu thành công, thuật toán lọc các 1-FWUP, là các mục có giá trị *wus* lớn hơn hoặc bằng *minwus*, lưu trữ các 1-FWUP này trong C_1 và gọi thủ tục *MiningFWUPs_CTset* để khai thác các FWUP (dòng 3-6).

Thủ tục *MiningFWUPs_CTset* được gọi đệ quy bằng cách sử dụng cấu trúc dữ liệu CTset để khai thác các FWUP. CTset của ứng viên k -pattern X được tạo thành từ các $(k-1)$ -FWUP có cùng lớp tương đương được lưu trữ trong C (dòng 10, 11). Nếu *wus* của mẫu X không nhỏ hơn *minwus* thì X là một FWUP và được thêm vào C_{next} để phục vụ cho quá trình tạo ứng viên tiếp theo (dòng 12-14). Quá trình này tiếp tục cho đến khi không còn mẫu mới nào được sinh ra. Cuối cùng, ta thu được tất cả các FWUP từ dữ liệu cửa sổ mới nhất và thuật toán được lặp lại bất cứ khi nào một pane mới được thêm vào.

3.3.2 Thuật toán FWUDS-DWT

3.3.2.1 Cấu trúc dữ liệu DSWUN-tree

Chương 2 của luận án đã đề xuất cấu trúc WUNList để khai thác FWUP trên CSDL định lượng động và cho kết quả tốt hơn cả về thời gian lẫn bộ nhớ sử dụng. WUNList được tạo ra từ cấu trúc WUN-tree, là phần mở rộng của cấu trúc N-list [14]. WUN-tree là cấu trúc cây nén CSDL sau khi đã loại bỏ các mục không phổ biến. Nhược điểm của WUN-tree là chỉ áp dụng với CSDL cố định số lượng giao dịch, và mất đến hai lần duyệt CSDL để tạo cây. Việc xây dựng WUN-tree tốn nhiều thời gian, đặc biệt là khi kích thước dữ liệu tăng lên. Do đó, cấu trúc này không phù hợp để áp dụng vào mô hình cửa sổ trượt trên luồng dữ liệu. Để khắc phục vấn đề này, luận án đề xuất một cấu trúc cây mới, có tên là DSWUN-tree, để khai thác hiệu quả FWUP trên luồng dữ liệu định lượng động.

Định nghĩa 3.10 (DSWUN-tree). DSWUN-tree là cấu trúc cây bao gồm một nút gốc và tập các nút con. Mỗi nút của DSWUN-tree bao gồm 6 trường (*name*,

weight, *pre*, *post*, *children*, *parent*). Trong đó, *name* là tên của mục đại diện cho nút, *weight* là tổng *twu* của các giao dịch đi qua nút này, *pre* là thứ tự của nút khi duyệt tiền thứ tự, *post* là thứ tự của nút khi duyệt hậu thứ tự, *children* là danh sách các nút con, và *parent* là nút cha của nút hiện tại.

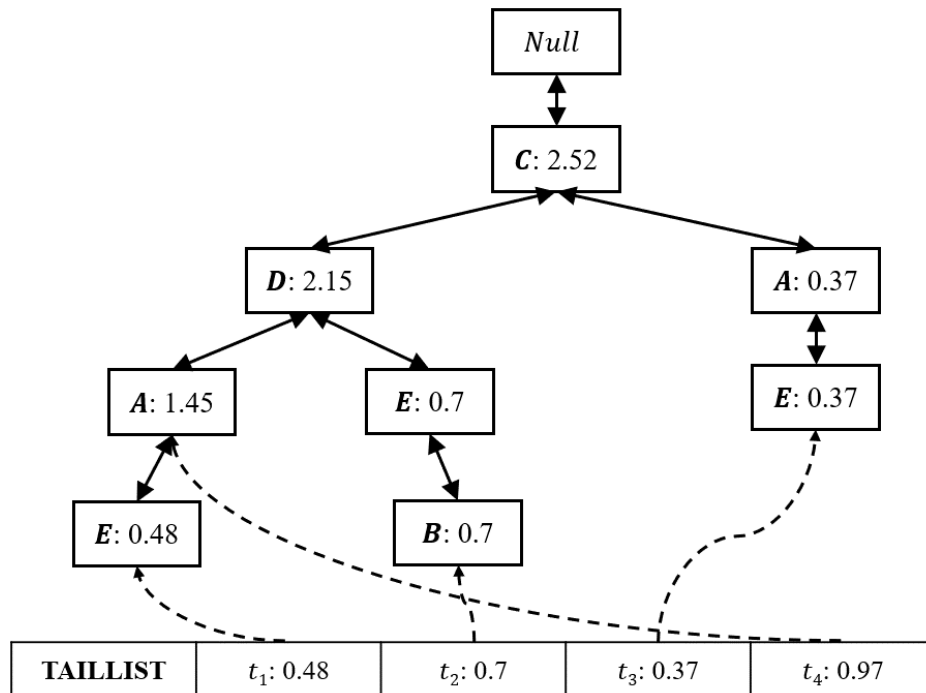
Thuật toán 3.4. Xây dựng DSWUN-tree
Input: Cửa sổ đầu tiên IW Output: DSWUN-tree
Procedure Construct_DSWUN-tree(T, R) 1. Tạo nút gốc của DSWUN-tree, $R = 'Null'$ 2. foreach giao dịch t trong IW do 3. Sắp xếp t giảm dần theo thứ tự wus của các mục trong IW 4. Insert_DSWUN-tree (t, R) 5. return R
Procedure Insert_DSWUN-tree(t, R) 1. i_1 là mục đầu tiên trong t 2. if tồn tại nút N trong $R.child$ có $N.name = i_1.name$ then 3. $N.weight += twu(T)$ 4. else 5. Tạo nút con mới N của R 6. $N.name \leftarrow i_1.name; N.weight \leftarrow twu(t); N.parent \leftarrow R$ 7. if $ t = 1$ then Thêm N vào cuối $TAILLIST$ 8. else Insert_DSWUN-tree ($t/\{i_1\}, N$)

Hình 3.7. Thuật toán xây dựng DSWUN-tree.

Khác với WUN-tree, cấu trúc được đề xuất DSWUN-tree lưu trữ thêm nút cha để phục vụ cho quá trình cập nhật cây khi dữ liệu thay đổi. Cùng với đó, DSWUN-tree lưu trữ toàn bộ CSDL, không loại bỏ các mục không thoả mãn. Các mục được sắp xếp theo thứ tự *wus* giảm dần của cửa sổ đầu tiên và các giao dịch được sắp xếp lại theo cùng thứ tự trước khi chèn vào DSWUN-tree. Sau khi được xác lập, thứ tự này được giữ cố định trong toàn bộ quá trình xử lý luồng và được xem là thứ tự cấu

trúc của DSWUN-tree. Mặc dù giá trị wus của các mục có thể thay đổi khi cửa sổ trượt, các giao dịch thuộc các pane tiếp theo vẫn được sắp xếp theo cùng thứ tự trước khi cập nhật vào cây. Điểm vượt trội của cấu trúc DSWUN-tree đó là chỉ mất một lần quét CSDL để cập nhật cây, sau đó có thể duy trì và khai thác nhiều lần với nhiều ngưỡng khác nhau. Tuy nhiên, DSWUN-tree tốn nhiều bộ nhớ lưu trữ hơn do phải lưu trữ nhiều thông tin hơn. Ngoài ra, DSWUN-tree còn sử dụng thêm một danh sách liên kết, có tên là TAILLIST, để lưu trữ các nút cuối cùng của từng giao dịch khi được thêm vào cây. Danh sách này được sử dụng để nhanh chóng cập nhật lại cây khi dữ liệu thay đổi. Thuật toán xây dựng DSWUN-tree được mô tả trong Hình 3.7.

Ví dụ 3.7: Hình 3.8 minh họa DSWUN-tree được tạo từ bốn giao dịch của cửa sổ W_1 trong DS_e , ký hiệu là $DSWUN-tree(W_1)$. Các giao dịch được sắp xếp lại dựa trên các giá trị wus và được thêm vào cây. Cụ thể, giao dịch t_1 được sắp lại thành $\{C, D, A, E\}$, t_2 thành $\{C, D, E, B\}$, t_3 thành $\{C, A, E\}$, và t_4 thành $\{C, D, A\}$. Đối với mỗi giao dịch được thêm vào DSWUN-tree, nút cuối cùng của giao dịch này được thêm vào TAILLIST. Kết quả là TAILLIST có bốn nút đuôi tương ứng với bốn giao dịch trong W_1 .



Hình 3.8. DSWUN-tree được tạo thành từ W_1 của DS_e .

3.3.2.2 Cập nhật DSWUN-tree

Trong giai đoạn trượt khung cửa sổ, khi một pane mới được thêm vào thì pane cũ nhất sẽ bị xóa khỏi DSWUN-tree vì nó không còn trong cửa sổ hiện tại. Sử dụng TAILLIST và thông tin nút cha được lưu trữ tại mỗi nút, ta có thể dễ dàng trích xuất các giao dịch trong pane cũ nhất và xóa chúng khỏi DSWUN-tree. Hình 3.9 mô tả thuật toán cập nhật DSWUN-tree. Đầu tiên, mỗi giao dịch trong pane mới nhất được thêm vào DSWUN-tree bằng cách sử dụng thủ tục *Insert_DSWUN-tree* (dòng 1, 2). Sau đó, thuật toán lặp $|P|$ lần, tương ứng với $|P|$ giao dịch trong pane cũ nhất. Với mỗi lần lặp, thuật toán xem xét nút đầu tiên trong TAILLIST tương ứng với giao dịch t_i . Tiếp theo, ta giảm trọng số của nút hiện tại theo giá trị twu của giao dịch này. Nếu trọng số của nút này giảm xuống bằng không, nó sẽ bị xóa khỏi cây. Sau đó, ta di chuyển lên nút cha của nó. Quá trình này lặp lại cho đến khi đi đến nút gốc của DSWUN-tree (dòng 4-7). Cuối cùng, ở dòng 8, thuật toán tiến hành xóa nút đầu tiên khỏi TAILLIST và bắt đầu lần lặp tiếp theo. Kết thúc quá trình cập nhật, ta thu được DSWUN-tree của cửa sổ mới.

Định lý 3.3. Giả sử DSWUN-tree trước khi cập nhật tuân theo thứ tự toàn cục đã được xác lập tại cửa sổ đầu tiên. Sau khi thêm một pane mới hoặc loại bỏ pane cũ theo quy trình cập nhật của thuật toán, DSWUN-tree vẫn duy trì cùng thứ tự này trên mọi đường đi từ gốc đến nút lá.

Chứng minh. Đối với thao tác thêm pane mới, trước khi chèn vào DSWUN-tree, các mục trong mỗi giao dịch của pane được sắp xếp theo thứ tự toàn cục đã xác lập. Quá trình chèn lần lượt cập nhật hoặc tạo mới các nút theo chính thứ tự này. Do đó mọi đường đi mới được tạo ra đều tuân theo cùng thứ tự và các đường đi đã tồn tại không bị thay đổi.

Đối với thao tác loại bỏ pane cũ, thuật toán sử dụng TAILLIST để xác định các nút thuộc các giao dịch cũ nhất và lần lượt cập nhật hoặc loại bỏ các nút tương ứng. Quá trình này không thực hiện phép hoán đổi hoặc sắp xếp lại các nút còn lại. Do đó thứ tự giữa các mục trên các đường đi còn tồn tại là không thay đổi.

Vì cả hai thao tác đều không làm thay đổi thứ tự toàn cục của các mục, DSWUN-tree sau mỗi lần cập nhật vẫn duy trì thứ tự đã được xác lập ban đầu.

Từ **Định lý 3.3**, có thể thấy việc cập nhật DSWUN-tree luôn duy trì một thứ tự toàn cục nhất quán giữa các giao dịch. Đây là cơ sở bảo đảm tính đúng đắn của quá trình xây dựng và kết hợp WUNList, bởi quan hệ tổ tiên - hậu duệ giữa các nút được xác định theo cùng một thứ tự, không phụ thuộc vào việc thứ tự này có còn tương ứng với giá trị wus giảm dần tại cửa sổ hiện tại hay không.

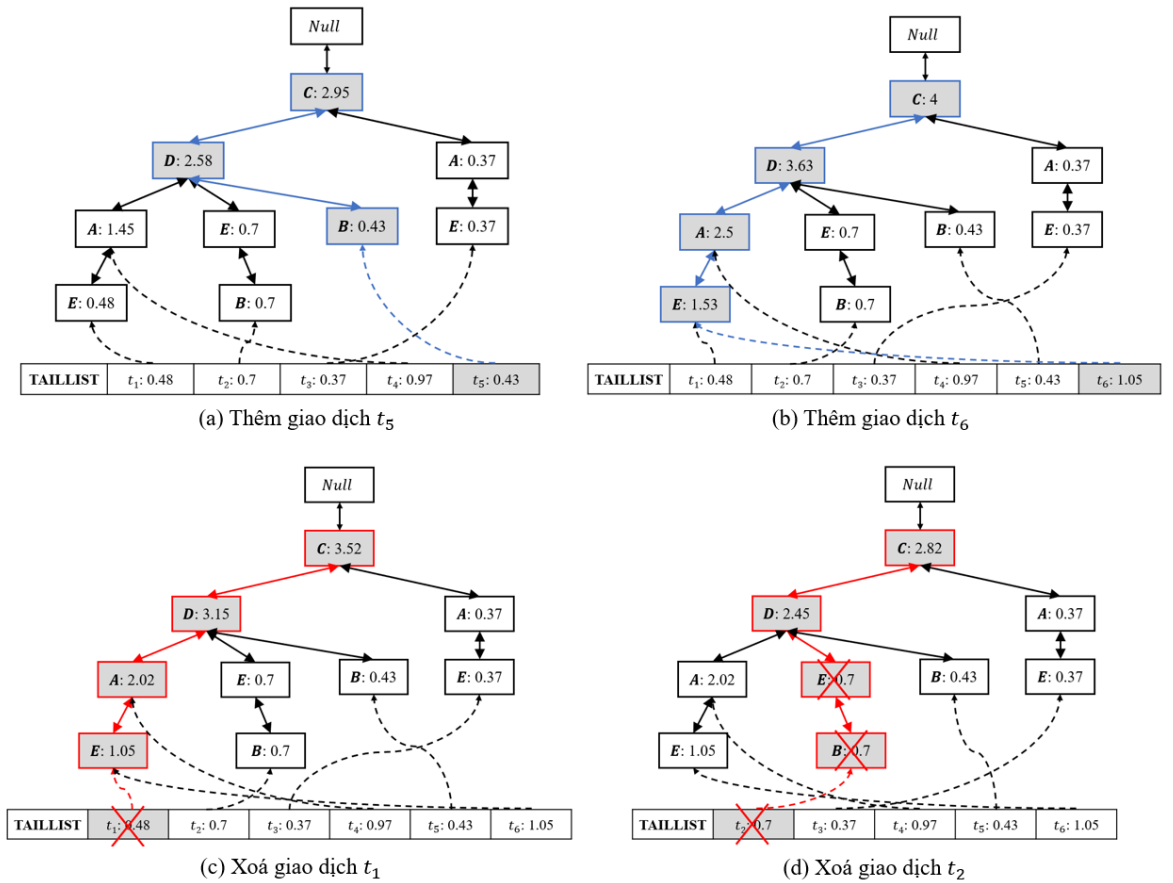
Thuật toán 3.5. Cập nhật DSWUN-tree
Input: DSWUN-tree R và pane mới nhất P Output: DSWUN-tree đã được cập nhật
Procedure Update_DSWUN-tree(R, P) 1. foreach giao dịch t trong P do 2. Insert_DSWUN-tree(t, R) 3. for $i \leftarrow 1$ to $ P $ do 4. L là phần tử đầu tiên trong $TAILLIST$ 5. foreach nút $N \neq \text{root}$ từ nhánh L đến root của R do 6. $N.weight -= twu(L.tid)$ 7. if $N.weight = 0$ then Xoá N 8. Xoá L khỏi $TAILLIST$ 9. return R

Hình 3.9. Thuật toán cập nhật DSWUN-tree.

Ví dụ 3.8: Xét quá trình trượt khung cửa sổ từ cửa sổ W_1 đến W_2 trong DS_e , trong đó P_3 là pane mới nhất và P_1 là pane cũ nhất. Quá trình cập nhật DSWUN-tree được mô tả trong Hình 3.10. Đầu tiên, pane P_3 bao gồm hai giao dịch t_5 và t_6 lần lượt được thêm vào DSWUN-tree như Hình 3.10a và Hình 3.10b. Sau đó, các giao dịch t_1 và t_2 của pane P_1 không có trong W_2 bị xóa khỏi DSWUN-tree. Với giao dịch t_1 , nút đuôi trong TAILLIST là $(E: 1.53)$, ta giảm trọng số của nút này đi $twu(t_1) = 0.48$ và nút được cập nhật thành $(E: 1.05)$. Thuật toán di chuyển lên nút cha của nó

(A : 2.02) và giảm trọng số của nó đi 0.48. Ta tiếp tục giảm trọng số 0.48 trên các nút tổ tiên của nó là (D : 3.63) và (C : 4). DSWUN-tree thu được sau khi xóa giao dịch t_1 được mô tả trong Hình 3.10c.

Quá trình xóa giao dịch t_2 được thực hiện tương tự và được hiển thị trong Hình 3.10d. Ta giảm trọng số của nút đuôi (B : 0.7) với $twu(t_2) = 0.7$. Giá trị trọng số còn lại là 0, do đó nút này bị xóa khỏi DSWUN-tree. Tiếp theo, nút cha (E : 0.7) cũng bị xóa khỏi DSWUN-tree vì trọng số còn lại của nó bằng không sau khi giảm đi 0.7. Cuối cùng, ta duyệt hai nút tổ tiên (D : 3.15) và (C : 3.52) và cập nhật chúng thành (D : 2.45) và (E : 2.82). Quá trình hoàn tất và ta thu được DSWUN-tree đã cập nhật với dữ liệu mới trong cửa sổ W_2 .



Hình 3.10. Quá trình cập nhật DSWUN-tree từ cửa sổ W_1 sang W_2 của DS_e .

3.3.2.3 Mô tả thuật toán

Với việc sử dụng các kỹ thuật được mô tả ở trên, thuật toán được đề xuất

FWUDS-DWT, tìm kiếm hiệu quả các FWUP trên luồng dữ liệu định lượng động dựa trên mô hình cửa sổ trượt, được mô tả trong Hình 3.11.

Thuật toán 3.6. Thuật toán FWUDS-DWT
Input: Pane mới nhất P , DSWUN-tree của cửa sổ hiện tại R và ngưỡng $minwus$ Output: Tập tất cả các mẫu trọng số hữu ích phổ biến của cửa sổ mới $FWUPs$
1. $R \leftarrow \text{Update_DSWUN-tree}(P, R)$ 2. Lưu các mục thoả mãn $minwus$ theo thứ tự vào I_1 3. Tạo WUNList cho các mục trong I_1 4. $FWUPs \leftarrow I_1$ 5. MineFWUPs_WUNList (I_1) 6. return $FWUPs$
Procedure MineFWUPs_WUNList (C) 7. for $i \leftarrow C - 1$ downto 1 do 8. $C_{next} \leftarrow \emptyset$; 9. for $j \leftarrow i - 1$ downto 0 do 10. $X \leftarrow C[i] \cup C[j]$ 11. $wunl(X) \leftarrow \text{Construct_WUNList}(wunl(C[i]), wunl(C[j]))$ 12. if $wunl(X) \neq NULL$ then 13. $FWUPs \leftarrow X$ 14. $C_{next} \leftarrow \{X, wunl(X), wus(X)\}$ 15. if $ C_{next} > 1$ then 16. MineFWUPs_WUNList (C_{next})

Hình 3.11. Thuật toán FWUDS-DWT.

Đầu tiên, ở dòng 1, thuật toán gọi thủ tục *Update_DSWUN-tree* để cập nhật dữ liệu trong cửa sổ mới. Pane cũ nhất trong DSWUN-tree bị xóa và pane mới nhất được thêm vào DSWUN-tree. Tiếp theo, thuật toán tiến hành tìm các 1-FWUP và lưu trữ chúng trong I_1 (dòng 2). Ta duyệt DSWUN-tree để tạo các giá trị *pre* và *post* cho mỗi nút và tạo WUNList cho các mẫu này trong I_1 (dòng 3). Sau đó, thuật toán

gọi thủ tục *MineFWUPs_WUNList* để khai thác tất cả các FWUP thỏa mãn ngưỡng *minwus* trong cửa sổ hiện tại (dòng 4, 5). *MineFWUPs_WUNList* sử dụng phương pháp tìm kiếm theo chiều sâu, kết hợp với cấu trúc dữ liệu WUNList, để nhanh chóng xác định các FWUP tiếp theo. Quá trình này dừng lại khi không có mẫu mới nào được tạo thành và các FWUP được cung cấp cho người dùng. Sau đó, thuật toán tiếp tục quá trình trượt khung cửa sổ tiếp theo.

3.3.3 Tính đúng đắn và độ phức tạp thuật toán

Hai thuật toán FWUDS-CT và FWUDS-DWT đều được đề xuất nhằm giải quyết bài toán khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt. Mặc dù áp dụng các cấu trúc dữ liệu khác nhau để hỗ trợ cập nhật cửa sổ, cả hai thuật toán đều kế thừa cơ chế tính toán giá trị độ hỗ trợ trọng số hữu ích, chiến lược khai thác dựa trên tính phản đơn điệu và quá trình mở rộng mẫu đã được chứng minh ở Chương 2. Do đó, các phân tích lý thuyết trong phần này tập trung vào tính đúng đắn, sự ảnh hưởng và tính hiệu quả của các cơ chế cập nhật dữ liệu trên luồng dữ liệu. Đối với thuật toán FWUDS-CT, cải tiến quan trọng nhất là việc sử dụng cấu trúc Circular Tidset để thay thế Tidset truyền thống. CTset khắc phục hạn chế kéo dài vô hạn của Tidset bằng cách ánh xạ các định danh giao dịch vào một miền giá trị hữu hạn tương ứng với kích thước cửa sổ và thực hiện xoay vòng các định danh khi cửa sổ trượt. Mặc dù giá trị định danh được tái sử dụng, mỗi định danh tại mọi thời điểm luôn đại diện duy nhất cho một giao dịch đang tồn tại trong cửa sổ hiện hành. Khi một pane cũ rời khỏi cửa sổ, các định danh tương ứng được giải phóng; khi pane mới được thêm vào, các định danh giao dịch cục bộ này được gán lại cho các giao dịch mới. Vì vậy, CTset luôn phản ánh chính xác tập giao dịch thuộc cửa sổ hiện tại và không làm thay đổi kết quả của các phép giao. Do đó, các giá trị *wus* được tính toán chính xác trong các khung cửa sổ, bảo đảm tính đúng đắn của thuật toán.

Đối với thuật toán FWUDS-DWT, điểm cải tiến nằm ở việc xây dựng cấu trúc DSWUN-tree để biểu diễn trực tiếp toàn bộ cửa sổ dữ liệu chỉ với một lần quét. DSWUN-tree kế thừa cơ chế nén dữ liệu của WUN-tree, đồng thời bổ sung khả năng

cập nhật động thông qua danh sách liên kết TAILLIST. Mỗi giao dịch sau khi được chèn vào cây đều có một nút cuối cùng tương ứng được lưu trong TAILLIST. Khi cửa sổ dịch chuyển, thuật toán chỉ cần truy xuất trực tiếp các nút cuối này để lần ngược lên đường đi của giao dịch và cập nhật trọng số hoặc loại bỏ các nút không còn xuất hiện. Nhờ đó, mọi thay đổi trên cửa sổ đều được phản ánh đầy đủ trên DSWUN-tree mà không cần xây dựng lại toàn bộ cấu trúc dữ liệu. Sau mỗi lần cập nhật, DSWUN-tree luôn biểu diễn chính xác tập giao dịch của cửa sổ hiện tại. Hơn nữa, việc duy trì thứ tự cố định của các mục trong DSWUN-tree cũng bảo đảm tính nhất quán của quá trình xây dựng WUNList. Khi điều kiện này được duy trì, quan hệ tổ tiên - hậu duệ giữa các nút được xác định thống nhất, từ đó các WUNList được xây dựng và kết hợp theo đúng quan hệ cấu trúc của cây và chứa đầy đủ thông tin cần thiết để sinh tất cả các mẫu hợp lệ, không bỏ sót FWUP trong quá trình khai thác. Tuy nhiên, khi giá trị wus của các mục thay đổi đáng kể qua các cửa sổ, thứ tự cố định được xác lập ban đầu có thể không còn tương ứng với thứ tự của cửa sổ hiện tại. Điều này có thể làm giảm khả năng chia sẻ tiền tố giữa các giao dịch, dẫn đến kích thước DSWUN-tree và chiều dài WUNList tăng lên. Tuy vẫn bảo đảm tính đúng đắn của kết quả khai thác nhưng vấn đề này có thể làm tăng thời gian xử lý và mức sử dụng bộ nhớ.

Về hiệu năng, độ phức tạp thời gian của thuật toán FWUDS-CT bao gồm hai phần chính là chi phí duy trì cửa sổ và chi phí khai thác FWUP. Khi một pane có kích thước p mới đến, quá trình cập nhật cửa sổ tiến hành với thủ tục *InsertNewPane* và *RemoveOldestPane* (nếu cửa sổ đã đầy). Thủ tục *InsertNewPane* lặp qua p giao dịch, mỗi giao dịch có tối đa l_{max} mục, với l_{max} biểu thị độ dài giao dịch lớn nhất trong cửa sổ hiện tại. Mỗi mục yêu cầu một thao tác chèn $lcTid$ với chi phí hằng số $O(1)$, dẫn đến tổng thời gian là $O(p \cdot l_{max})$. Đối với thủ tục *RemoveOldestPane*, thuật toán duyệt qua n CTset, trong đó n là số lượng mục phân biệt trong cửa sổ. Tổng số thao tác kiểm tra và xóa $lcTid$ trên tất cả n danh sách được giới hạn bởi $O(p \cdot l_{max})$ nên chi phí cho thủ tục này là $O(n + p \cdot l_{max})$. Như vậy, tổng thời gian cập nhật cửa sổ cho mỗi pane mới là $O(n + p \cdot l_{max})$. Đối với chi phí khai thác, quá trình này được

bắt đầu bằng lời gọi đệ quy *MineFWUPs_CTsets*. Thao tác cốt lõi bao gồm việc sinh các mẫu ứng viên X và tính toán CTset tương ứng thông qua thủ tục *CTsets_Intersection*. Phép giao này được thực hiện tuyến tính trên hai CTset có kích thước tối đa ct_{max} , nên có độ phức tạp là $O(ct_{max})$. Gọi I_1 là tập các mục 1-FWUP thoả mãn ngưỡng *minwus*, trong trường hợp xấu nhất, thuật toán phải khám phá toàn bộ không gian tìm kiếm, tức là tổng số mẫu là $2^{|I_1|} - 1$. Do đó, độ phức tạp thời gian cho giai đoạn khai thác là $O(ct_{max} \cdot 2^{|I_1|})$.

Đối với thuật toán FWUDS-DWT, giai đoạn khởi tạo DSWUN-tree cần xử lý $m = w \cdot p$ giao dịch trong cửa sổ đầu tiên, trong đó w là kích thước cửa sổ. Với mỗi giao dịch, tất cả các mục đều được sắp xếp lại và thêm vào cây. Do đó, tổng chi phí một lần để khởi tạo cây là $O(m \cdot l_{max} \cdot \log l_{max})$. Đối với mỗi pane mới đến, tổng chi phí thời gian bao gồm giai đoạn cập nhật cây và khai thác. Giai đoạn cập nhật được thực hiện bởi thủ tục *Update_DSWUN-tree*, bao gồm hai thao tác là chèn p giao dịch mới và xóa p giao dịch cũ. Việc chèn p giao dịch mới đòi hỏi p lần gọi tới thủ tục *InsertTree*, mất $O(p \cdot l_{max} \cdot \log l_{max})$. Tương tự, thao tác xóa ta cũng phải lặp p lần, trong đó mỗi lần duyệt từ một nút lá trong TAILLIST lên đến nút gốc, với chi phí là $O(l_{max})$ cho mỗi lần duyệt. Do đó, tổng thời gian cập nhật cây trên mỗi pane là $O(p \cdot l_{max} \cdot \log l_{max})$. Giai đoạn khai thác bắt đầu bằng việc tạo WUNList cho tập I_1 mất $O(|I_1| \cdot wunl_{max})$, với $wunl_{max}$ là độ dài tối đa của một WUNList. Sau đó, thủ tục đệ quy *MineFWUPs_WUNList* được gọi. Thuật toán FWUDS-DWT sử dụng cây liệt kê tập hợp để khai thác FWUP. Trong trường hợp xấu nhất, độ phức tạp là $O(2^{|I_1|})$ tương ứng với không gian tìm kiếm là $2^{|I_1|} - 1$ nút trong cây. Với mỗi nút, thuật toán tiến hành kết hợp hai WUNList để tạo WUNList cho mẫu mới với độ phức tạp là $O(wunl_{max})$. Vì vậy, tổng độ phức tạp thời gian cho quá trình khai thác trên mỗi cửa sổ là $O(wunl_{max} \cdot 2^{|I_1|})$.

Dựa trên các phân tích về lý thuyết, ta thấy được cả hai thuật toán đề xuất đều hiệu quả trong việc cập nhật dữ liệu khi trượt khung cửa sổ. Nguyên nhân chính nằm ở cả hai đều áp dụng cơ chế cập nhật gia tăng theo từng pane, thay vì phải xử lý lại

toàn bộ giao dịch trong mỗi cửa sổ. Về độ phức tạp thời gian khai thác, cả hai thuật toán về lý thuyết đều có chi phí theo hàm mũ trong trường hợp xấu nhất. Đây cũng là đặc điểm chung của các thuật toán khai thác mẫu. Tuy nhiên, trên thực tế, thuật toán FWUDS-DWT thường cho thời gian khai thác thấp hơn. Điều này là do FWUDS-DWT sử dụng cấu trúc WUNList với khả năng nén dữ liệu và có xu hướng rút gọn dần sau mỗi lần kết hợp, giúp quá trình tạo mẫu diễn ra nhanh hơn so với cấu trúc CTset. Ngược lại, về độ phức tạp không gian, cả hai thuật toán đều phải lưu trữ toàn bộ dữ liệu của cửa sổ hiện tại nên sẽ cần $O(m \cdot l_{max})$ bộ nhớ. Tuy nhiên, cấu trúc DSWUN-tree lại phức tạp hơn rất nhiều so với CTset và còn bao gồm thêm cấu trúc TAILLIST với $O(m)$ bộ nhớ, nên trong thực tế DSWUN-tree cần nhiều bộ nhớ hơn, đặc biệt là trên các bộ dữ liệu thưa thớt. Mặt khác, cả hai thuật toán đều sử dụng phương pháp duyệt sâu để tiến hành khai thác FWUP, nên độ phức tạp không gian tương ứng cho giai đoạn này của thuật toán FWUDS-CT là $O(ct_{max} \cdot |I_1|^2)$ và thuật toán FWUDS-DWT là $O(wunl_{max} \cdot |I_1|^2)$. Kết quả là trên các bộ dữ liệu lớn và dày đặc, các mục xuất hiện nhiều trên các giao dịch, dẫn đến thuật toán FWUDS-CT sẽ cần rất nhiều bộ nhớ lưu trữ để khai thác.

Như vậy, thuật toán FWUDS-DWT đánh đổi việc sử dụng thêm bộ nhớ để lưu trữ cấu trúc nén DSWUN-tree và danh sách WUNList ngắn gọn, cho phép thời gian khai thác nhanh hơn. Trong khi đó, thuật toán FWUDS-CT tập trung vào cấu trúc CTset cực kỳ gọn nhẹ, đơn giản, nhưng chi phí tính toán khai thác cao hơn khi các CTset trở nên dài hơn hoặc số lượng mẫu ứng viên lớn. Sự đánh đổi này trực tiếp quyết định đến việc ứng dụng, giới hạn và khả năng mở rộng của chúng. Cụ thể, thuật toán FWUDS-DWT là lựa chọn phù hợp hơn cho các hệ thống có tài nguyên bộ nhớ dồi dào và yêu cầu khai thác với tần suất cao, vì khả năng nén dữ liệu giúp giảm đáng kể chi phí các phép kết hợp WUNList, từ đó cải thiện hiệu suất thực thi. Ngược lại, thuật toán FWUDS-CT là giải pháp lý tưởng cho ứng dụng mà dữ liệu đầu vào chủ yếu là dữ liệu thưa, nhờ cấu trúc gọn nhẹ và mức tiêu thụ bộ nhớ thấp. Về giới hạn và khả năng mở rộng, nút thắt cổ chai của hai thuật toán có bản chất khác nhau. Đối với FWUDS-DWT, hạn chế cốt lõi nằm ở bộ nhớ, do yêu cầu duy trì cấu trúc

DSWUN-tree cho toàn bộ cửa sổ hiện tại khiến bộ nhớ tăng lên, từ đó hạn chế khả năng mở rộng đối với các cửa sổ rất lớn trên dữ liệu thưa. Trong khi đó, hạn chế của FWUDS-CT nằm ở thời gian xử lý, khi hiệu suất khai thác suy giảm đáng kể nếu ct_{max} tăng hoặc ngưỡng *minwus* được đặt quá thấp trên các bộ dữ liệu dày đặc.

3.3.4 Thực nghiệm và kết quả

3.3.4.1 Thiết lập thực nghiệm

Phần này trình bày kết quả thực nghiệm so sánh hai thuật toán được thiết kế để khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt. Hai thuật toán đề xuất, FWUDS-CT và FWUDS-DWT, được so sánh với phiên bản đã sửa đổi của thuật toán dFWUNL (gọi là dFWUNL-stream). Các sửa đổi của thuật toán dFWUNL giúp hoạt động được trên luồng dữ liệu và vẫn đảm bảo hiệu suất cũng như độ phức tạp giống với phiên bản gốc. Các thuật toán được triển khai bằng ngôn ngữ lập trình C# 2022 (.NET Framework 4.7.2). Các thực nghiệm đều được tiến hành trên một máy tính được trang bị CPU Intel Core i5-9400F 2.90 GHz, RAM 32 GB và chạy trên hệ điều hành Windows 11. Các bộ dữ liệu thực nghiệm bao gồm Retail, Accidents, BMS-POS, Kosarak, Susy. Thông tin về số lượng giao dịch, số lượng mục, đặc tính và tập các trọng số động của từng bộ dữ liệu đã được trình bày trong Bảng 2.6. Để mô phỏng môi trường luồng dữ liệu, các giao dịch trong mỗi bộ dữ liệu được xử lý tuần tự và tổ chức thành các pane theo thiết lập của từng thực nghiệm.

Vì các bộ dữ liệu thực nghiệm có kích thước và đặc điểm khác nhau nên kích thước cửa sổ, kích thước pane và ngưỡng *minwus* được lựa chọn khác nhau cho từng bộ dữ liệu. Điều này đảm bảo cho việc đánh giá hiệu suất được thể hiện rõ ràng giữa các thuật toán. Quá trình thực nghiệm trên từng bộ dữ liệu được tiến hành như sau. Đầu tiên, các thuật toán quét lần lượt các pane trong cửa sổ đầu tiên, xây dựng cấu trúc dữ liệu lưu trữ chúng và tiến hành khai thác FWUP trong khung cửa sổ này. Tiếp theo, quá trình trượt khung cửa sổ diễn ra, pane cũ nhất bị xóa đi và pane mới được thêm vào. Sau đó, các FWUP trong khung cửa sổ mới cập nhật sẽ được khai thác. Tất cả các thực nghiệm được thực hiện bằng cách trượt cửa sổ 50 lần. Trong toàn bộ 50

lần trượt cửa sổ, FWUDS-CT, FWUDS-DWT và dFWUNL-stream đều thu được cùng số FWUP tại từng cửa sổ tương ứng. Kết quả này xác nhận rằng các thuật toán tạo ra tập kết quả nhất quán trên cùng dữ liệu và cùng tham số. Do cả ba thuật toán đều tìm được đầy đủ và chính xác tất cả các FWUP nên hiệu suất của chúng được đánh giá dựa trên thời gian chạy, bộ nhớ sử dụng và khả năng mở rộng. Cụ thể, bốn khía cạnh chính được đánh giá bao gồm: (1) thời gian cập nhật dữ liệu trong quá trình trượt cửa sổ, phản ánh hiệu quả của từng cấu trúc dữ liệu; (2) tổng thời gian thực thi, cho biết chi phí thời gian tính toán tổng thể; (3) mức tiêu thụ bộ nhớ, thể hiện hiệu quả không gian của các mô hình được đề xuất; và (4) khả năng mở rộng, đánh giá hiệu suất của từng thuật toán khi khối lượng dữ liệu tăng lên.

Bảng 3.2. Các tham số thực nghiệm khai thác FWUP trên luồng dữ liệu

CSDL	Kích thước cửa sổ	Kích thước pane	Các giá trị <i>minwus</i>
Retail	88000P	1	0.3%, 0.2%, 0.1%, 0.05%, 0.03%
Accidents	340000P	1	60%, 50%, 40%, 30%, 20%
BMS-POS	515000P	1	1%, 0.7%, 0.4%, 0.2%, 0.1%
Kosarak	980000P	1	0.5%, 0.4%, 0.3%, 0.25%, 0.2%
Susy	4000000P	1	13%, 11%, 10%, 9.5%, 9%

Bảng 3.3. Tham số thực nghiệm khả năng mở rộng của các thuật toán khai thác FWUP trên luồng dữ liệu theo kích thước cửa sổ và pane

CSDL	<i>minwus</i>	Các cấu hình (Kích thước cửa sổ - Kích thước pane)
Kosarak	0.2%	(700P – 600), (750P – 800), (800P – 1000)
Susy	10%	(800P – 3000), (900P – 3500), (1000P – 4000)

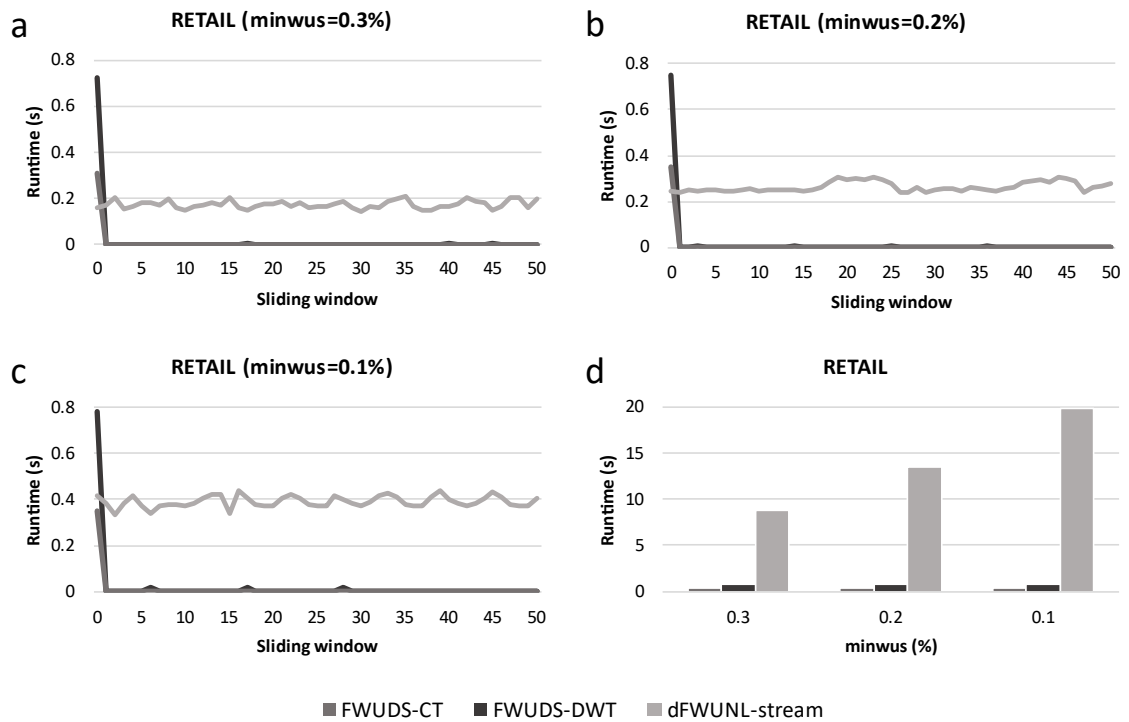
Đối với thực nghiệm đánh giá thời gian cập nhật dữ liệu và thời gian khai thác FWUP, các thuật toán được thực hiện với ba giá trị *minwus* khác nhau trên mỗi bộ dữ liệu với kích thước cửa sổ và pane được cố định. Đối với thực nghiệm đánh giá

khả năng mở rộng, hai bộ dữ liệu lớn Kosarak và Susy được sử dụng, trong đó kích thước cửa sổ và kích thước pane được tăng dần qua ba mức khác nhau trong khi giữ nguyên giá trị *minwus*. Các tham số cụ thể của từng thực nghiệm được trình bày trong Bảng 3.2 và Bảng 3.3. Thời gian chạy được đo bằng tổng thời gian từ khi bắt đầu thuật toán đến khi hoàn thành. Trong khi đó, mức tiêu thụ bộ nhớ được xác định dựa trên lượng bộ nhớ cao nhất mà thuật toán sử dụng trong quá trình thực thi.

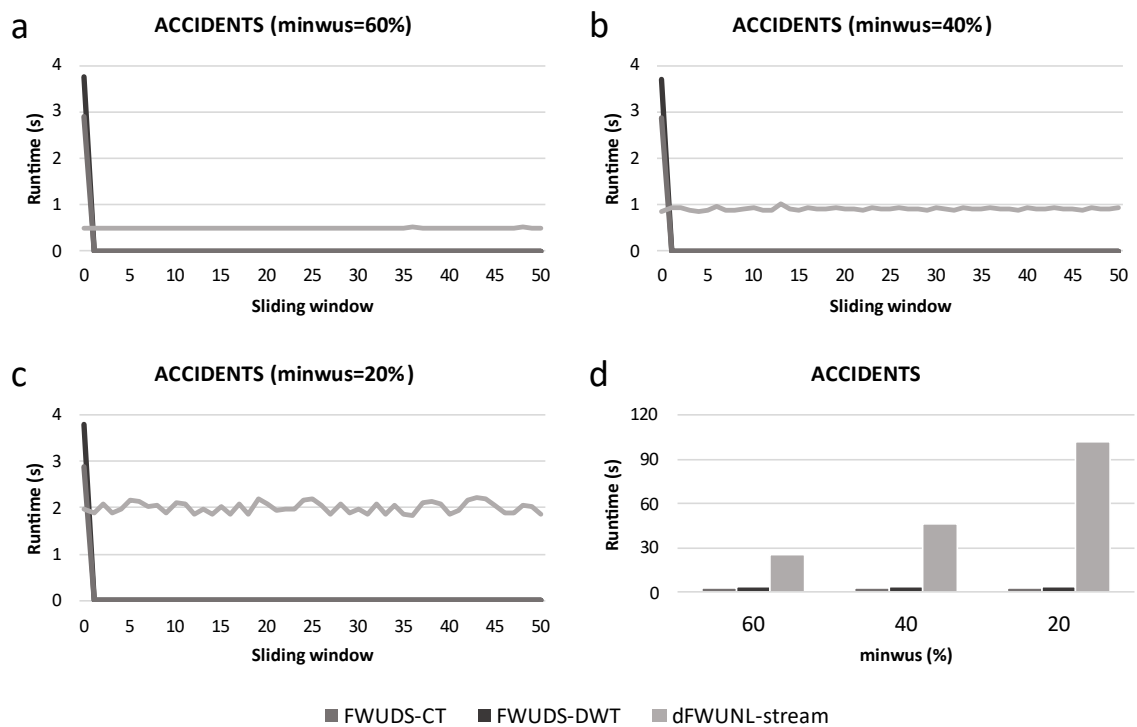
3.3.4.2 So sánh về thời gian cập nhật dữ liệu

Đầu tiên, các thực nghiệm được tiến hành để đánh giá khả năng của các thuật toán trong việc xây dựng cấu trúc và cập nhật dữ liệu trong quá trình trượt khung cửa sổ. Đây là một nhiệm vụ quan trọng ảnh hưởng đáng kể đến hiệu quả của các thuật toán, đặc biệt là khi xét đến tốc độ và tính liên tục của luồng dữ liệu. Hình 3.12 - Hình 3.16 hiển thị kết quả trên các bộ dữ liệu thực nghiệm. Các thuật toán được thực hiện trên ba ngưỡng *minwus* khác nhau cho mỗi bộ dữ liệu, với kích thước pane cố định là 1. Tổng thời gian cho quá trình cập nhật dữ liệu sau đó được tổng hợp lại.

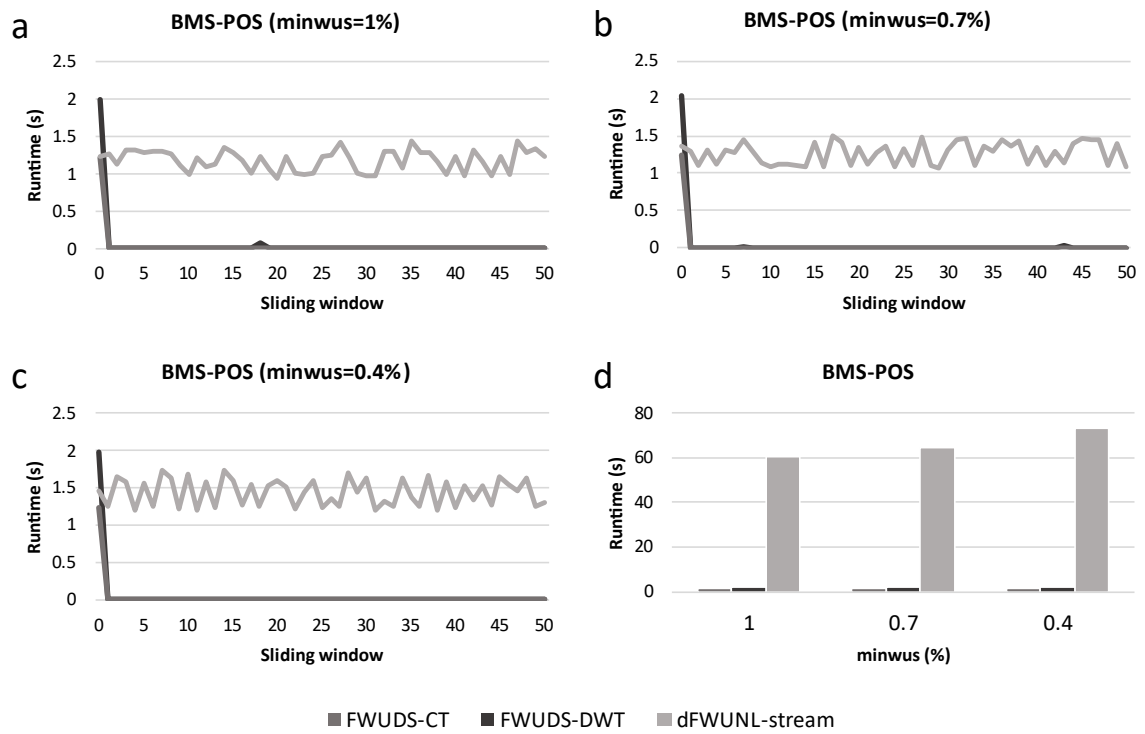
Hình 3.12 cho thấy kết quả trên tập dữ liệu Retail với kích thước cửa sổ là 88000P (88000 pane) và ngưỡng *minwus* lần lượt là 0.3%, 0.2% và 0.1%. Biểu đồ trong Hình 3.12d cho thấy các thuật toán FWUDS-CT và FWUDS-DWT mất rất ít thời gian để xây dựng và cập nhật dữ liệu so với thuật toán dFWUNL-stream, trong đó thuật toán FWUDS-CT có thời gian chạy ngắn nhất. Hơn nữa, khi ngưỡng *minwus* giảm, thời gian chạy của thuật toán dFWUNL-stream tăng lên, trong khi hai thuật toán được đề xuất duy trì thời gian chạy gần như ổn định. Điều này là do thuật toán dFWUNL-stream cần liên tục quét lại CSDL để tạo WUN-tree cho mỗi cửa sổ mới. Ngược lại, hai phương pháp được đề xuất chỉ mất thời gian ban đầu để xây dựng cấu trúc CTset và DSWUN-tree cho cửa sổ đầu tiên, và hầu như không cần thêm thời gian cho quá trình trượt cửa sổ sau đó. Dữ liệu từ cửa sổ cũ được sử dụng lại và quá trình cập nhật chỉ cần xóa pane cũ nhất, thêm pane mới nhất. Do đó, sau 50 lần trượt khung cửa sổ, tổng thời gian cập nhật dữ liệu của các thuật toán FWUDS-CT và FWUDS-DWT vượt trội hơn so với thuật toán dFWUNL-stream.



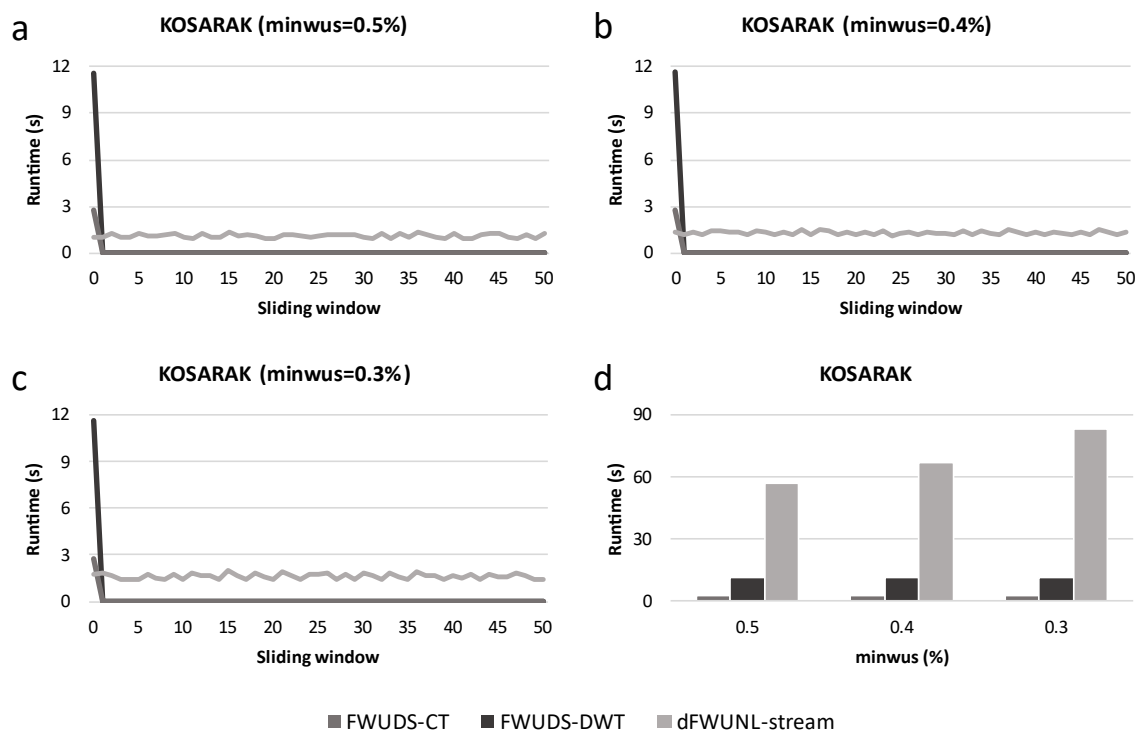
Hình 3.12. Thời gian cập nhật dữ liệu trên CSDL Retail với *minwus* khác nhau.



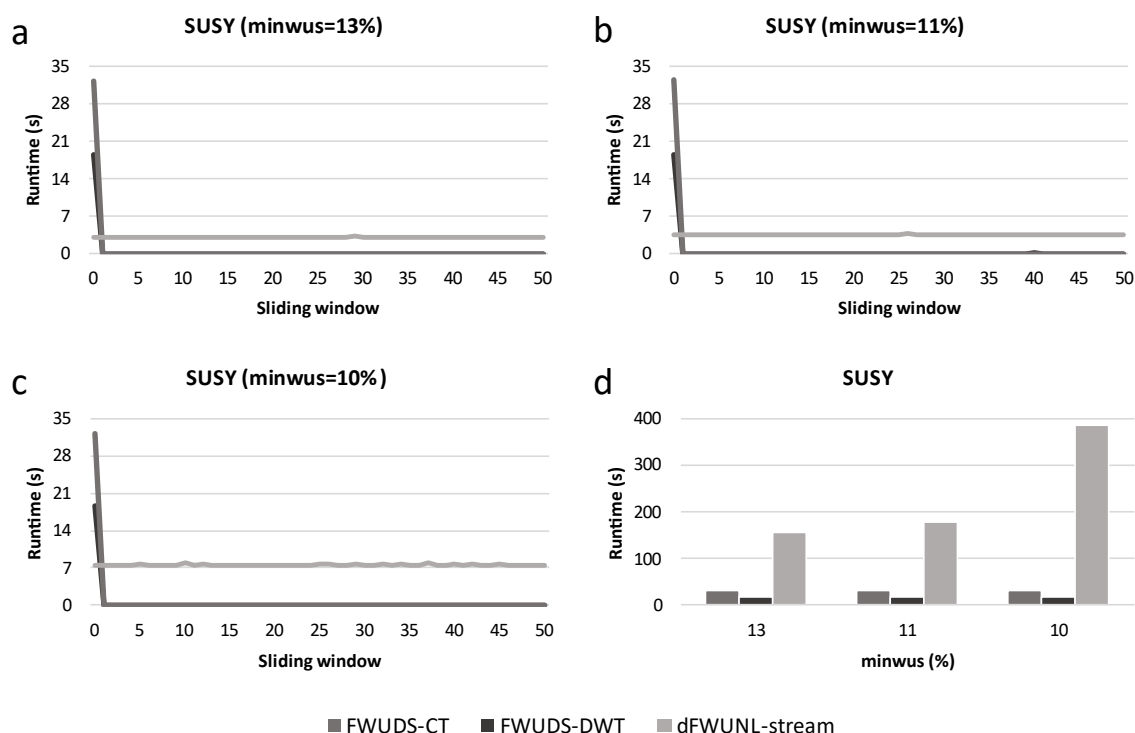
Hình 3.13. Thời gian cập nhật dữ liệu trên CSDL Accidents với *minwus* khác nhau.



Hình 3.14. Thời gian cập nhật dữ liệu trên CSDL BMS-POS với *minwus* khác nhau.



Hình 3.15. Thời gian cập nhật dữ liệu trên CSDL Kosarak với *minwus* khác nhau.

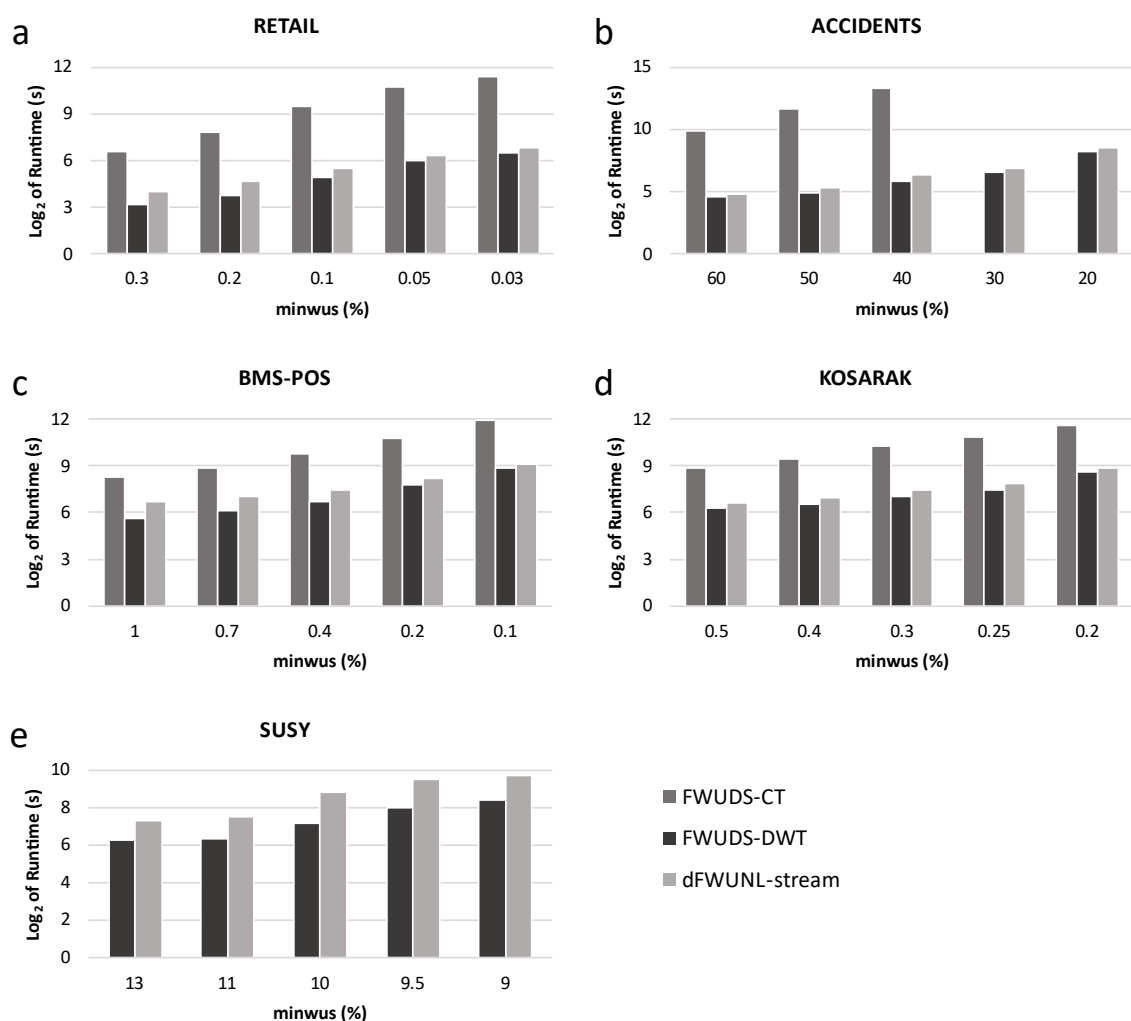


Hình 3.16. Thời gian cập nhật dữ liệu trên CSDL Susy với *minwus* khác nhau.

Kết quả tương tự cũng diễn ra đối với các tập dữ liệu Accidents, BMS-POS, Kosarak và Susy, với kích thước cửa sổ lần lượt là 340000P, 515000P, 980000P và 4000000P. Đáng chú ý, khi kích thước cửa sổ tăng lên, sự khác biệt về tổng thời gian cần thiết để xây dựng và cập nhật dữ liệu giữa các thuật toán càng trở nên rõ rệt hơn.

3.3.4.3 So sánh về thời gian khai thác FWUP

Hình 3.17 hiển thị tổng thời gian khai thác FWUP của các thuật toán FWUDS-CT, FWUDS-DWT và dFWUNL-stream trên các bộ dữ liệu có ngưỡng *minwus* khác nhau. Thời gian khai thác được đo từ khi bắt đầu thuật toán cho đến khi kết thúc. Kích thước cửa sổ và kích thước pane là cố định, và có giá trị được thiết lập như phần trên. Do sự khác biệt đáng kể về thời gian chạy giữa các thuật toán, thang logarit ($\log_2$) cho thời gian chạy được sử dụng để dễ dàng so sánh và trình bày trên biểu đồ. Kết quả cho thấy xu hướng khi ngưỡng *minwus* giảm thì thời gian khai thác của các thuật toán đều tăng lên. Điều này là do khi ngưỡng giảm thì số lượng FWUP tăng, khiến các thuật toán mất nhiều thời gian hơn để tìm tất cả các mẫu thỏa mãn.



Hình 3.17. Tổng thời gian khai thác FWUP trên các CSDL thực nghiệm với *minwus* khác nhau.

Mặc dù thuật toán FWUDS-CT hiệu quả trong việc xây dựng và cập nhật dữ liệu, nhưng lại mất nhiều thời gian hơn để khai thác FWUP. Hơn nữa, với ngưỡng rất nhỏ hoặc tập dữ liệu rất lớn, chẳng hạn như Susy, thuật toán FWUDS-CT sẽ không hoạt động được do tràn bộ nhớ. Ngược lại, thuật toán FWUDS-DWT có thời gian chạy ngắn nhất trong ba thuật toán thực nghiệm. Sự hiệu quả này là do khả năng cập nhật dữ liệu hiệu quả bằng cấu trúc DSWUN-tree, kết hợp với việc tối ưu hóa quá trình khai thác thông qua cấu trúc WUNList. Ví dụ, trên tập dữ liệu Retail có ngưỡng *minwus* là 0.2%, thuật toán FWUDS-CT mất 223 giây, thuật toán dFWUNL-stream mất 25 giây, trong khi thuật toán FWUDS-DWT chỉ mất 13.5 giây. Trên tập dữ liệu

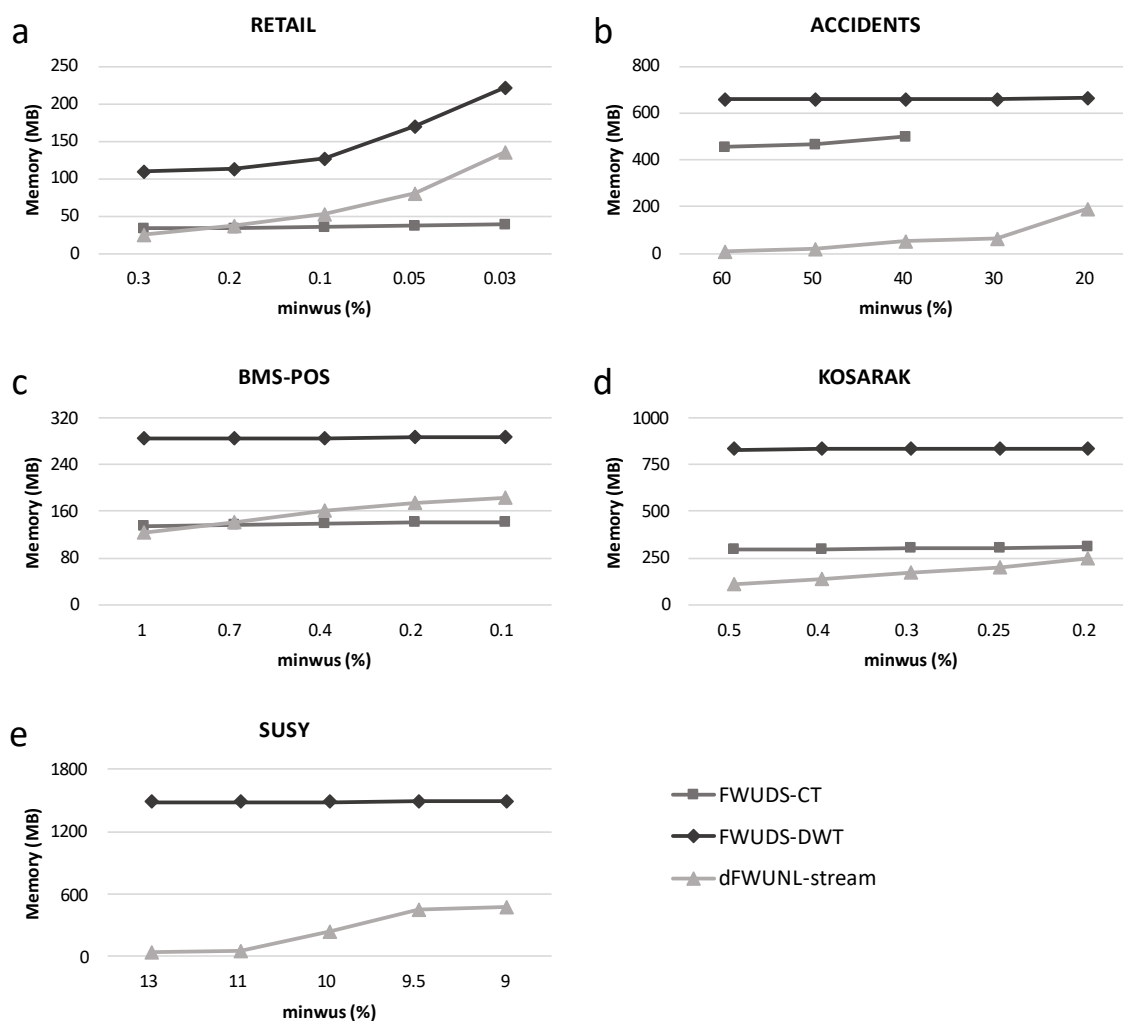
BMS-POS với ngưỡng 0.4%, tổng thời gian khai thác của các thuật toán FWUDS-CT, FWUDS-DWT và dFWUNL-stream sau 50 lần trượt khung cửa sổ lần lượt là 845 giây, 105.8 giây và 174.9 giây.

3.3.4.4 So sánh về bộ nhớ sử dụng

Để có được đánh giá một cách toàn diện, luận án tiến hành phân tích có hệ thống mức sử dụng bộ nhớ của các thuật toán đề xuất so với thuật toán cơ sở. Phân tích này đặc biệt quan trọng vì hiệu quả sử dụng bộ nhớ thường là một ràng buộc then chốt trong xử lý luồng dữ liệu. Mức sử dụng bộ nhớ được ghi nhận cho tất cả các thuật toán trên năm bộ dữ liệu khi thay đổi ngưỡng *minwus* và được hiển thị trong Hình 3.18. Các giá trị kích thước cửa sổ và kích thước pane được cài đặt giống như phần so sánh hiệu quả về thời gian chạy.

Kết quả thực nghiệm cho thấy những khác biệt rõ rệt và những đánh đổi đặc trưng trong thiết kế của từng thuật toán. Nhìn chung, khi ngưỡng *minwus* giảm thì có nhiều FWUP thỏa mãn được tìm thấy, dẫn đến bộ nhớ sử dụng của các thuật toán đều tăng lên. Thuật toán FWUDS-DWT duy trì mức sử dụng bộ nhớ cao nhất nhưng ổn định nhất trên tất cả các bộ dữ liệu. Cụ thể, mức tiêu thụ bộ nhớ của thuật toán này gần như không thay đổi trên một số bộ dữ liệu, khoảng 1500 MB trên Susy và 835 MB trên Kosarak, bất kể giá trị của *minwus*. Điều này là do thuật toán FWUDS-DWT phải lưu trữ toàn bộ cấu trúc DSWUN-tree trong bộ nhớ chính để biểu diễn tất cả các giao dịch trong khung cửa sổ hiện tại. Hơn nữa, cấu trúc dữ liệu DSWUN-tree khá phức tạp, cần phải lưu trữ nhiều thông tin dẫn đến cần nhiều không gian bộ nhớ. Mặc dù cách tiếp cận này dẫn đến tăng chi phí bộ nhớ, song độ ổn định giúp FWUDS-DWT đạt được hiệu quả thời gian chạy và hỗ trợ cập nhật dữ liệu mới một cách nhanh chóng. Ngược lại, thuật toán FWUDS-CT thể hiện hiệu quả bộ nhớ tốt hơn trên các bộ dữ liệu thưa nhưng gặp hạn chế với bộ dữ liệu dày. Trên các bộ dữ liệu Retail và BMS-POS, đây là thuật toán tiết kiệm bộ nhớ nhất. FWUDS-CT chỉ sử dụng khoảng 38.1 MB trên Retail với ngưỡng *minwus* = 0.03%, trong khi FWUDS-DWT cần 222 MB và dFWUNL-stream là 134.9 MB. Hiệu quả này đến từ cấu trúc dữ liệu

CTset đơn giản, giúp giảm thiểu phân bổ bộ nhớ trong các ngữ cảnh dữ liệu thưa thớt. Tuy nhiên, khi áp dụng trên các tập dữ liệu dày, kích thước CTset của các mục tăng nhanh, gây ra hiện tượng bùng nổ bộ nhớ. Do đó, FWUDS-CT chạy quá lâu trên Accidents khi ngưỡng nhỏ và không thể chạy trên bộ dữ liệu lớn Susy vì tràn bộ nhớ.



Hình 3.18. Bộ nhớ sử dụng cho quá trình khai thác FWUP trên các bộ dữ liệu thực nghiệm với *minwus* khác nhau.

Trong khi đó, thuật toán cơ sở dFWUNL-stream thể hiện sự hiệu quả về bộ nhớ sử dụng trên các bộ dữ liệu. Mức hiệu quả này có được vì thuật toán dFWUNL-stream chỉ xây dựng WUN-tree cho các mục thỏa ngưỡng *minwus* và loại bỏ cấu trúc này sau khi sinh ra danh sách WUNList trong mỗi cửa sổ độc lập. Còn hai thuật toán đề xuất đều phải lưu trữ lại toàn bộ dữ liệu của cửa sổ để phục vụ việc cập nhật

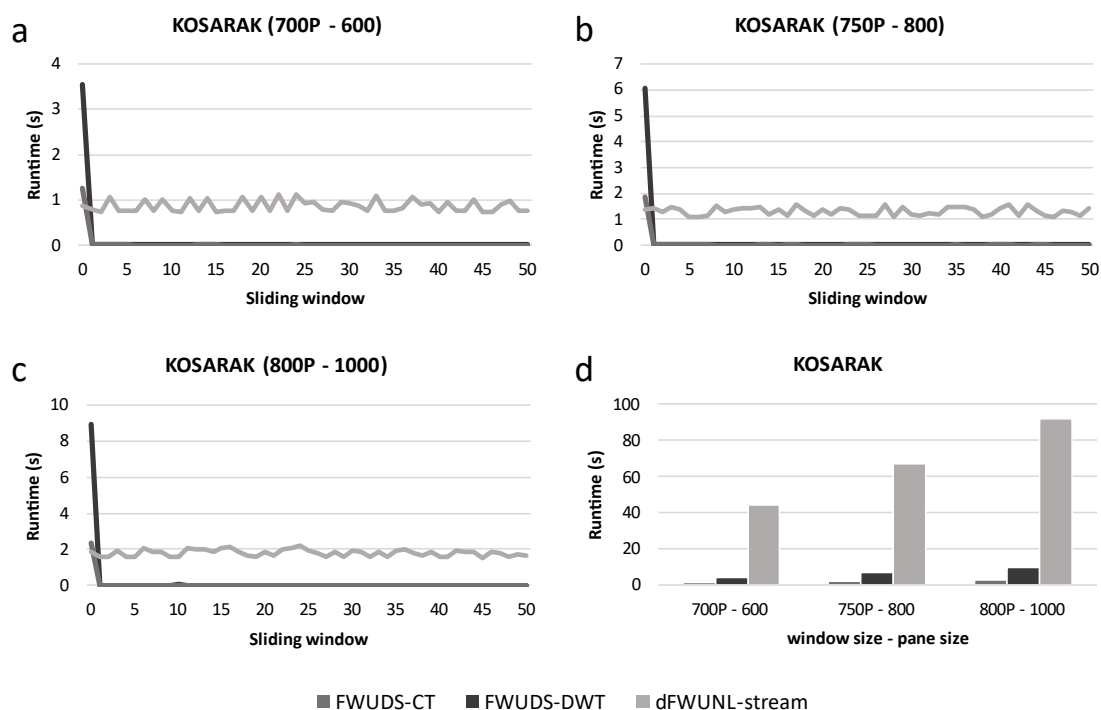
nhANH chóng. Ví dụ, trên bộ dữ liệu Kosarak với ngưỡng *minwus* là 0.3%, thuật toán FWUDS-CT sử dụng 300.9 MB, thuật toán FWUDS-DWT sử dụng 833.5 MB, còn thuật toán dFWUNL-stream sử dụng 173.3 MB. Bảng 3.4 tổng hợp kết quả khai thác FWUP của các thuật toán trên từng luồng dữ liệu tại các ngưỡng *minwus* đại diện, qua đó làm rõ sự khác biệt về hiệu quả thực thi và mức sử dụng bộ nhớ.

Bảng 3.4. Kết quả khai thác FWUP trên các luồng dữ liệu thực nghiệm

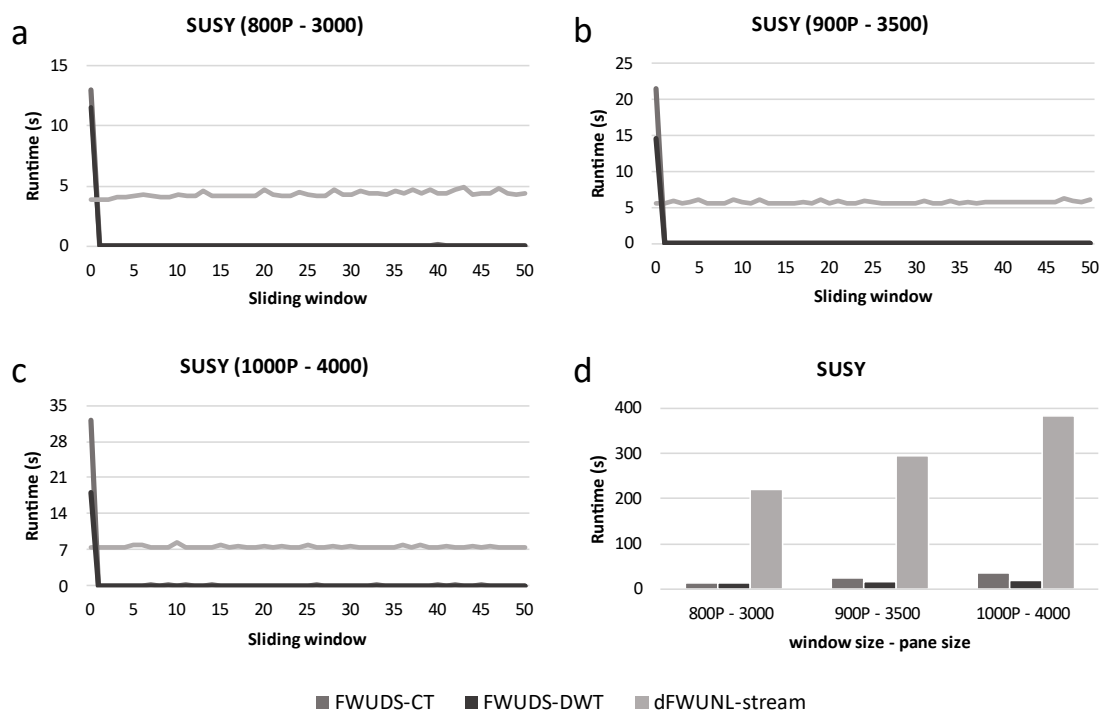
CSDL (<i>minwus</i>)	FWUDS-CT		FWUDS-DWT		dFWUNL-stream	
	Thời gian (s)	Bộ nhớ (MB)	Thời gian (s)	Bộ nhớ (MB)	Thời gian (s)	Bộ nhớ (MB)
Retail (0.03%)	2756.5	38.1	90.6	222	111.5	134.9
Accidents (50%)	3158.1	468.6	30.1	659.4	39.3	17.9
BMS-POS (0.2%)	1768	140.2	218.4	286.5	299.7	174.7
Kosarak (0.25%)	1828.4	303.9	173.9	835.2	225.4	201.9
Susy (10%)	x	x	143.2	1500.8	450.3	235.5

3.3.4.5 So sánh về khả năng mở rộng

Phần này đánh giá khả năng mở rộng của các thuật toán trên hai tập dữ liệu lớn là Kosarak và Susy. Đánh giá được thực hiện bằng cách kiểm tra tính hiệu quả của các thuật toán khi kích thước cửa sổ và kích thước pane tăng dần. Hình 3.19 hiển thị kết quả khả năng mở rộng cho quá trình xây dựng và cập nhật dữ liệu trên bộ dữ liệu Kosarak, với ngưỡng *minwus* cố định ở 0.2%. Kích thước cửa sổ được đặt là 700P, 750P và 800P, tương ứng với kích thước pane lần lượt là 600, 800 và 1000. Kết quả cho thấy khi kích thước cửa sổ và kích thước pane tăng lên, thời gian xây dựng và cập nhật dữ liệu của các thuật toán cũng tăng lên. Theo biểu đồ, thời gian cập nhật cho các phương pháp dựa trên cấu trúc CTset và DSWUN-tree tăng chậm, trong khi thời gian cập nhật của thuật toán dFWUNL-stream tăng nhanh khi kích thước lớn dần.



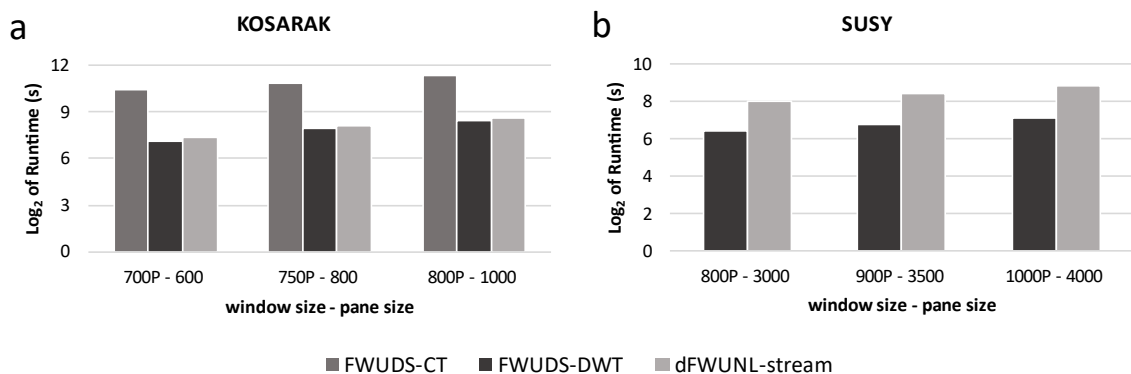
Hình 3.19. Thời gian cập nhật dữ liệu trên CSDL Kosarak với kích thước cửa sổ và kích thước pane khác nhau.



Hình 3.20. Thời gian cập nhật dữ liệu trên CSDL Susy với kích thước cửa sổ và kích thước pane khác nhau.

Hình 3.20 cho thấy thời gian cập nhật dữ liệu trên tập dữ liệu Susy với ngưỡng *minwus* là 10% sau 50 lần trượt khung cửa sổ. Kích thước cửa sổ và kích thước pane được đặt lần lượt là 800P – 3000, 900P – 3500 và 1000P – 4000. Tương tự như tập dữ liệu Kosarak, kết quả trên tập dữ liệu Susy cũng chứng minh tính ổn định của các thuật toán FWUDS-CT và FWUDS-DWT trong quá trình cập nhật dữ liệu, ngay cả khi kích thước cửa sổ và kích thước pane tăng lên. Trong Hình 3.20d, với kích thước cửa sổ là 800P và kích thước pane là 3000, thuật toán FWUDS-CT mất tổng thời gian cập nhật là 14 giây, FWUDS-DWT mất 12.1 giây, trong khi thuật toán dFWUNL-stream mất 220.6 giây. Khi kích thước cửa sổ tăng lên 900P và kích thước pane là 3500 giao dịch, thuật toán FWUDS-CT mất 23.6 giây, FWUDS-DWT mất 15.3 giây còn dFWUNL-stream mất 294.8 giây.

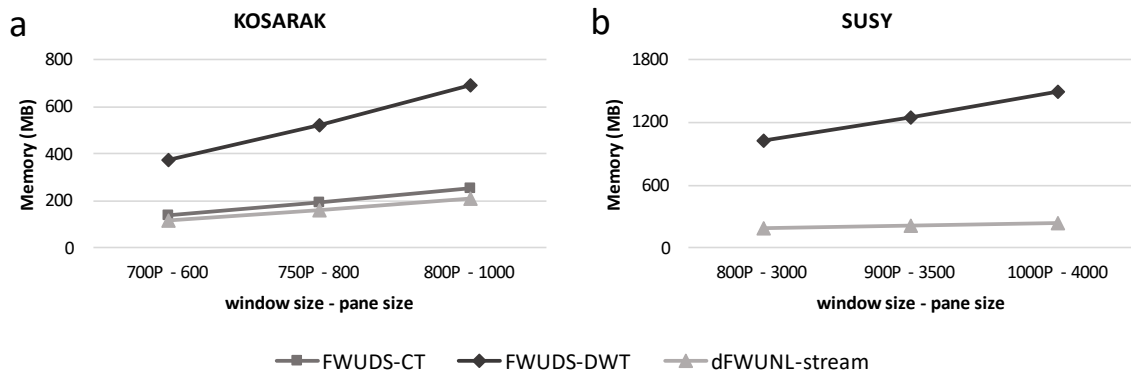
Hình 3.21 hiển thị kết quả khả năng mở rộng theo tổng thời gian khai thác của các thuật toán trên cả hai bộ dữ liệu. Khi kích thước cửa sổ và pane tăng lên, số lượng giao dịch trong mỗi cửa sổ cũng tăng lên, khiến các thuật toán mất nhiều thời gian hơn để xác định tất cả các FWUP. Thuật toán FWUDS-DWT thể hiện sự gia tăng ổn định hơn về thời gian và mất ít thời gian hơn so với hai thuật toán còn lại.



Hình 3.21. Khả năng mở rộng về thời gian khai thác FWUP với kích thước cửa sổ và pane khác nhau trên CSDL Kosarak và Susy.

Đối với khả năng mở rộng về bộ nhớ của các thuật toán trong cùng điều kiện thực nghiệm, các kết quả được hiển thị ở Hình 3.22. Trên bộ dữ liệu thừa Kosarak, mức tiêu thụ bộ nhớ của cả ba thuật toán, FWUDS-CT, dFWUNL-stream, và

FWUDS-DWT, tăng gần tuyến tính theo kích thước cửa sổ. Điều này hoàn toàn phù hợp với lý thuyết, bởi khi lưu trữ nhiều giao dịch hơn trong bộ nhớ, các cấu trúc dữ liệu tương ứng cũng sẽ cần nhiều không gian lưu trữ hơn. Trong ba thuật toán, FWUDS-DWT duy trì mức sử dụng bộ nhớ tuyệt đối cao nhất, do thiết kế lưu trữ toàn bộ cấu trúc DSWUN-tree biểu diễn tất cả giao dịch trong cửa sổ hiện tại.



Hình 3.22. Khả năng mở rộng về bộ nhớ sử dụng để khai thác FWUP với kích thước cửa sổ và pane khác nhau trên CSDL Kosarak và Susy.

Bảng 3.5. Khả năng mở rộng của các thuật toán khai thác FWUP trên luồng dữ liệu theo kích thước cửa sổ và pane

CSDL (minwus)	Kích thước cửa sổ	Kích thước pane	FWUDS-CT		FWUDS-DWT		dFWUNL- stream	
			Thời gian (s)	Bộ nhớ (MB)	Thời gian (s)	Bộ nhớ (MB)	Thời gian (s)	Bộ nhớ (MB)
Kosarak (0.2%)	700P	600	1372.1	139.2	139.9	375.4	165.6	118.4
	750P	800	1858.2	193.2	245.2	523.3	282.3	159.3
	800P	1000	2588.9	254.6	341.8	691.2	393.5	207.8
Chainstore (10%)	800P	3000	x	x	86.9	1024.3	263.4	184
	900P	3500	x	x	111.6	1253.6	347.4	210
	1000P	4000	x	x	138.2	1500.9	450	235.6

Kết quả trên bộ dữ liệu dày đặc Susy mang lại góc nhìn sâu sắc hơn về tác động của mật độ dữ liệu đối với khả năng mở rộng bộ nhớ. Như đã quan sát trước đó, FWUDS-CT không thể thực thi trong điều kiện này do tràn bộ nhớ. FWUDS-DWT có mức tiêu thụ bộ nhớ tăng từ 1024.3 MB lên 1500.9 MB khi kích thước cửa sổ mở rộng. Đáng chú ý, thuật toán cơ sở dFWUNL-stream cho thấy khả năng mở rộng bộ nhớ tốt nhất trong kịch bản này, với mức sử dụng chỉ tăng nhẹ từ 184 MB lên 235.6 MB. Kết quả này có thể giải thích bởi ngưỡng *minwus* cao được áp dụng cho bộ Susy, giúp giới hạn số lượng các mục phổ biến và giữ cho cấu trúc WUN-tree nhỏ gọn ngay cả khi dữ liệu đầu vào tăng đáng kể. Nhìn chung, mức tăng bộ nhớ của thuật toán FWUDS-DWT là chấp nhận được khi kích thước cửa sổ và kích thước pane tăng lên để đạt được hiệu quả về mặt thời gian khai thác. Chi tiết số liệu kết quả thực nghiệm về khả năng mở rộng được trình bày trong Bảng 3.5.

3.4 Khai thác top- k mẫu trọng số hữu ích phổ biến trên luồng dữ liệu định lượng động

Thuật toán FWUDS-CT và FWUDS-DWT đã chứng minh được tính hiệu quả trong việc khai thác FWUP trên luồng dữ liệu với một ngưỡng *minwus* định trước. Tuy nhiên, trong các hệ thống khai thác dữ liệu thực tế, việc thiết lập ngưỡng này trên các luồng dữ liệu biến động liên tục là một thách thức lớn đối với người sử dụng phổ thông. Nếu ngưỡng được đặt quá cao thì sẽ có nhiều mẫu quan trọng có thể bị bỏ sót; ngược lại, nếu ngưỡng quá thấp, số lượng mẫu thu được sẽ tăng đột biến và gây khó khăn cho việc phân tích. Trong bối cảnh đó, cách tiếp cận top- k cho phép người dùng chỉ cần xác định số lượng mẫu mong muốn thay vì phải lựa chọn ngưỡng khai thác. Ví dụ, nhà quản trị có thể yêu cầu hệ thống trả về 20 hoặc 50 nhóm sản phẩm có giá trị nhất tại thời điểm hiện tại. Điều này giúp quá trình khai thác trở nên trực quan hơn, đồng thời phù hợp với các môi trường dữ liệu động và thay đổi liên tục.

Nhằm tối ưu hóa quá trình trích xuất tri thức và cung cấp những thông tin mang tính chọn lọc cao nhất mà không phụ thuộc vào các thông số đầu vào khó xác định, phần này của luận án đề xuất thuật toán TKFWUDS để giải quyết bài toán khai

thác top- k FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt. Thuật toán TKFWUDS được thiết kế để tận dụng tối đa khả năng lưu trữ linh hoạt và cập nhật dữ liệu nhanh chóng của cấu trúc DSWUN-tree, kết hợp cùng các chiến lược cắt tỉa mạnh mẽ nhằm thu hẹp không gian tìm kiếm của cấu trúc WUNList. Để làm nền tảng cho việc xây dựng các chiến lược cắt tỉa và so sánh thứ tự ưu tiên giữa các mẫu trong cửa sổ trượt, trước hết cần thiết lập một thước đo định lượng về một ngưỡng mới. Do đó, các tiêu mục dưới đây sẽ lần lượt trình bày định nghĩa về độ hỗ trợ trọng số hữu ích động, các chiến lược tăng tốc độ cập nhật và cắt tỉa dựa trên ngưỡng động trong quá trình thực thi. Chi tiết của thuật toán TKFWUDS được mô tả và đánh giá tính hiệu quả thông qua các kết quả thực nghiệm.

3.4.1 Ngưỡng hỗ trợ trọng số hữu ích động

Phương pháp truyền thống đầu tiên để khai thác top- k FWUP đó là sử dụng phương pháp vét cạn. Ta tìm tất cả các mẫu trọng số hữu ích có thể có trong cửa sổ hiện tại. Sau đó sắp xếp các mẫu này giảm dần theo độ hỗ trợ trọng số hữu ích của chúng và lấy k FWUP đầu tiên. Phương pháp vét cạn rõ ràng không hiệu quả cả về thời gian và bộ nhớ sử dụng vì phải sinh ra tất cả các mẫu. Hơn nữa, khác với bài toán khai thác mẫu truyền thống, khai thác top- k FWUP không có ngưỡng hỗ trợ trọng số hữu ích tối thiểu. Mặc dù dễ tiếp cận hơn với người dùng phổ thông nhưng việc khai thác top- k FWUP sẽ trở nên khó khăn hơn. Bởi vì ngưỡng *minwus* đóng vai trò quan trọng trong việc loại bỏ các mẫu không phổ biến trong quá trình khai thác. Thay vào đó, luận án đề xuất một ngưỡng khác có tên là ngưỡng hỗ trợ trọng số hữu ích động và được định nghĩa như sau.

Định nghĩa 3.11 (Ngưỡng hỗ trợ trọng số hữu ích động). Ngưỡng hỗ trợ trọng số hữu ích động của khung cửa sổ W_i , ký hiệu là $dWus(W_i)$, là giá trị độ hỗ trợ trọng số hữu ích của mẫu lớn thứ k tính đến thời điểm khai thác hiện tại.

Định nghĩa 3.12 (Ngưỡng hỗ trợ trọng số hữu ích tối ưu). Ngưỡng hỗ trợ trọng số hữu ích tối ưu của khung cửa sổ W_i , ký hiệu là $OpWus(W_i)$, được xác định như sau:

$$OpWus(W_i) = \min\{wus(X, W_i) | X \in TK(W_i)\} \quad (3.5)$$

Trong luận án, bài toán khai thác top- k FWUP được xác định là tìm đúng k mẫu có giá trị wus lớn nhất trong mỗi cửa sổ dữ liệu, trong đó wus là tiêu chí duy nhất để xếp hạng các mẫu. Trong trường hợp có nhiều mẫu cùng giá trị wus tại vị trí thứ k , các mẫu này được xem là đồng hạng theo tiêu chí của bài toán. Do không đặt thêm tiêu chí ưu tiên giữa các mẫu đồng hạng, thuật toán ưu tiên giữ các mẫu được phát hiện trước. Cụ thể, khi tập lưu trữ top- k FWUP đã chứa đủ k mẫu, một mẫu được phát hiện sau có wus bằng với $dWus$ thì không thay thế mẫu đồng hạng đang được duy trì trong tập kết quả. Quy tắc xử lý đồng hạng không làm thay đổi giá trị $opWus$, do tất cả các mẫu tại vị trí thứ k đều có cùng giá trị wus , đồng thời tập kết quả vẫn gồm đúng k mẫu theo định nghĩa của bài toán. Từ ngưỡng hỗ trợ trọng số hữu ích động và quy tắc xử lý này, các chiến lược cắt tia được phát triển nhằm nâng cao hiệu quả quá trình khai thác như sau.

Chiến lược 1. Tăng ngưỡng hỗ trợ trọng số hữu ích động. Chiến lược này sử dụng một danh sách Lk để duy trì top- k FWUP hiện tại trong suốt quá trình khai thác. Ban đầu, $dWus$ được gán bằng 0 và Lk không chứa phần tử nào. Quá trình sinh ứng viên tìm thấy một mẫu X , nếu $wus(X, W_i) > dWus$ thì X được thêm vào Lk . Nếu Lk có nhiều hơn k mẫu thì một mẫu có độ hỗ trợ trọng số hữu ích nhỏ nhất sẽ bị xóa khỏi Lk , bảo đảm Lk có tối đa k phần tử. Ngưỡng $dWus$ sẽ được nâng lên bằng với giá trị của mẫu có độ hỗ trợ nhỏ nhất trong Lk sau khi đã cập nhật. Sau khi kết thúc quá trình khai thác, ta thu được top- k FWUP của cửa sổ W_i và ngưỡng $dWus(W_i)$ chính bằng giá trị $opWus(W_i)$.

Chiến lược 2. Loại bỏ nhanh các mẫu không thỏa mãn. Chiến lược này được áp dụng trong quá trình sinh ứng viên bằng việc sử dụng ngưỡng động $dWus$ ở Chiến lược 1. Trong quá trình sinh ứng viên, các mẫu có độ hỗ trợ trọng số hữu ích không lớn hơn $dWus$ sẽ bị loại bỏ và không có ứng viên mới nào được tạo thành từ các mẫu này. Cụ thể, xét hai mẫu Px và Py có chung tiền tố P trong cửa sổ W_i . Hai mẫu này khi kết hợp với nhau sẽ tạo thành mẫu Pxy . Nếu $wus(Px, W_i) \leq dWus(W_i)$

hoặc $wus(Py, W_i) \leq dWus(W_i)$ thì theo tính chất phản đơn điệu, giá trị wus của mẫu Pxy không vượt quá tập con của nó là Px và Py , dẫn đến $wus(Pxy, W_i) \leq dWus(W_i)$. Do đó, Pxy không nằm trong top- k FWUP của khung cửa sổ W_i và được loại bỏ ngay lập tức.

Chiến lược 3. Cắt tỉa tăng cường dựa trên cặp mục. Chiến lược này được thực hiện ngay sau Chiến lược 2 nhằm cung cấp thêm một bộ lọc loại bỏ nhanh các mẫu không nằm trong top- k FWUP. Tương tự như Chiến lược cắt tỉa kết hợp twu tối đa trong thuật toán FWUCWP ở Chương 2, một ma trận tam giác, ký hiệu là $PTP(W_i)$, được sử dụng để tính nhanh giá trị wus của 2-pattern trong cửa sổ W_i dựa trên DSWUN-tree. Với hai mẫu Px và Py thoả mãn Chiến lược 2, tức là $wus(Px, W_i) > dWus(W_i)$ và $wus(Py, W_i) > dWus(W_i)$, nếu $PTP(W_i)[x, y] / sumtwu(W_i) \leq dWus(W_i)$ thì mẫu Pxy không nằm trong top- k FWUP. Do $PTP(W_i)[x, y] / sumtwu(W_i)$ chính là wus của 2-pattern xy trong cửa sổ W_i , nên nếu $wus(xy) \leq dWus(W_i)$ thì $wus(Pxy) \leq dWus(W_i)$ và mẫu Pxy bị loại bỏ. Như vậy, Chiến lược 2 và 3 giúp nhanh chóng xác định các mẫu không nằm trong top- k FWUP mà không cần phải tiến hành phép tạo WUNList các mẫu, từ đó, giảm không gian tìm kiếm, tăng tốc quá trình khai thác.

Chiến lược 4. Sử dụng hàng đợi ưu tiên lưu trữ top- k FWUP. Hàng đợi ưu tiên, ký hiệu là PQ , có tối đa k phần tử để duy trì top- k FWUP của cửa sổ hiện tại, thay thế cho danh sách Lk ở Chiến lược 1. PQ cung cấp các phương thức phù hợp và tốc độ cao với việc liên tục cập nhật và tìm giá trị nhỏ nhất trong top- k FWUP. Theo Chiến lược 1, ban đầu PQ bằng rỗng. Trong quá trình khai thác, các mẫu thoả mãn sẽ được thêm vào PQ , và các mẫu trong PQ nếu không còn thoả mãn sẽ bị xoá bỏ (dựa vào $dWus$), đồng thời ngưỡng động $dWus$ cũng được cập nhật bằng PQ . Thao tác tìm giá trị độ hỗ trợ trọng số hữu ích nhỏ nhất trong PQ , tăng ngưỡng động $dWus$ có độ phức tạp là $O(1)$; các thao tác thêm và xoá một phần tử trong PQ có độ phức tạp tối đa là $O(\log k)$. Do đó, sử dụng PQ là hiệu quả để lưu trữ top- k FWUP so với cấu trúc danh sách truyền thống.

3.4.2 Thuật toán TKFWUDS

Thuật toán TKFWUDS khai thác top- k FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt được trình bày trong Hình 3.23. Thuật toán TKFWUDS hoạt động như sau. Đầu tiên, dòng 1, thuật toán tiến hành cập nhật dữ liệu trong cửa sổ mới bằng thủ tục *Update_DSWUN-tree*. Dữ liệu trong pane mới được thêm vào và pane cũ nhất bị xoá khỏi DSWUN-tree. Tiếp theo, giá trị $dWus$ được gán bằng 0, PQ và C được khởi tạo bằng rỗng (C được sử dụng để lưu trữ các ứng viên trong quá trình khai thác) (dòng 2). Sau đó, ta thêm k mục có giá trị wus lớn nhất vào PQ và C , đồng thời cập nhật lại $dWus$ (dòng 3-5). TKFWUDS quét DSWUN-tree để tạo WUNList của các mục trong C và bảng PTP . Dòng 7-13, ta gọi thủ tục *GenTKCandidate* để tiến hành tìm kiếm các mẫu nằm trong top- k FWUP của cửa sổ hiện tại. Thủ tục *GenTKCandidate* sử dụng Chiến lược 1 và 4 (dòng 29), Chiến lược 2 (dòng 18, 21 và 25) và Chiến lược 3 (dòng 22) để loại bỏ các mẫu không thoả mãn. Các mẫu mới được tạo thành sẽ được kiểm tra với $dWus$ để xác định có nằm trong top- k FWUP hiện tại hay không (Dòng 23-31). Thủ tục kết thúc khi không còn ứng viên mới nào được sinh ra và top- k FWUP được cung cấp cho người dùng. Thuật toán tiếp tục quá trình trượt khung cửa sổ cho pane dữ liệu tiếp theo.

Thuật toán 3.7. Thuật toán TKFWUDS
Input: Pane mới nhất P , DSWUN-tree của cửa sổ hiện tại R và một số nguyên dương k ; Output: Top- k FWUP trong cửa sổ mới PQ .
1. $R \leftarrow \text{Update_DSWUN-tree}(R, P)$ 2. Khởi tạo $dWus \leftarrow 0, PQ \leftarrow \emptyset$ và $C \leftarrow \emptyset$ 3. k mục có wus lớn nhất được thêm vào PQ và C 4. if $PQ.Len() = k$ then //Chiến lược 1 và 4 5. $dWus \leftarrow PQ.Top()$ 6. Duyệt DSWUN-tree để tạo WUNList và PTP của các mục trong C 7. while $C \neq \emptyset$ do

8. $C_{new} \leftarrow \text{GenTKCandidate}(C)$ 9. $C \leftarrow \emptyset$ 10. for $i \leftarrow 0$ to $ C_{new} - 1$ do 11. $Z \leftarrow C_{new}[i]$ 12. if $Z.wus > dWus$ then //Chiến lược 2 13. $C \leftarrow Z$ 14. return PQ
Procedure GenTKCandidate (C_{curr}) 15. $C_{next} \leftarrow \emptyset$; 16. for $j \leftarrow 1$ to $ C_{curr} - 1$ do 17. $Q \leftarrow C_{curr}[j], x \leftarrow C_{curr}[j].last()$ 18. if $Q.wus > dWus$ then //Chiến lược 2 19. for $i \leftarrow 0$ to $j - 1$ do 20. $P \leftarrow C_{current}[i], y \leftarrow C_{curr}[i].last()$ 21. if $P.wus > dWus$ then //Chiến lược 2 22. if $PTP[u][v]/sumtwu > dWus$ then //Chiến lược 3 23. if P và Q cùng lớp tương đương then 24. $X.wunl \leftarrow \text{Construct_WUNList}(P.wunl, Q.wunl)$ 25. if $X.wus > dWus$ then //Chiến lược 2 26. $X.name \leftarrow Q.name \cup P.name$ 27. $C_{next} \leftarrow X$ 28. $PQ.Push(X)$ 29. if $PQ.Len() > k$ then //Chiến lược 1 và 4 30. $PQ.Pop()$ 31. $dWus \leftarrow PQ.Top()$ 32. return C_{next}

Hình 3.23. Thuật toán TKFWUDS.

3.4.3 Tính đúng đắn và độ phức tạp thuật toán

Thuật toán TKFWUDS được xây dựng trên nền tảng cấu trúc DSWUN-tree

và WUNList đã được trình bày ở phần trước, do đó nó kế thừa trực tiếp tính đúng đắn của quá trình tính toán giá trị wus , cơ chế cập nhật cửa sổ trượt và phép kết hợp WUNList trong quá trình sinh mẫu mới. Tính đúng đắn của TKFWUDS còn phụ thuộc vào cơ chế nâng ngưỡng động $dWus$, tính đầy đủ của quá trình sinh mẫu và tính an toàn của các chiến lược cắt tĩa được sử dụng trong quá trình khai thác. Trong thuật toán, giá trị $dWus$ luôn biểu diễn ngưỡng hỗ trợ trọng số hữu ích nhỏ nhất trong tập top- k FWUP hiện tại. Giá trị $dWus$ được khởi tạo bằng 0 và được nâng dần trong quá trình khai thác. Theo quy tắc xử lý đồng hạng đã xác định, mọi mẫu có giá trị wus nhỏ hơn hoặc bằng $dWus$ đều không thể xuất hiện trong tập kết quả cuối cùng. Do đó, việc nâng ngưỡng động không làm mất đi bất kỳ mẫu kết quả hợp lệ nào. Đồng thời, do giá trị wus thỏa tính chất phản đơn điệu nên Chiến lược 2 loại các mẫu có $wus \leq dWus$ và các mở rộng của chúng, còn Chiến lược 3 tiếp tục loại các phép kết hợp mà giá trị wus của 2-pattern tương ứng, được xác định bởi PTP , không vượt quá $dWus$. Tính an toàn của các điều kiện này đã được phân tích và chứng minh trong từng chiến lược, qua đó bảo đảm loại bỏ các mẫu không có khả năng xuất hiện trong tập kết quả, giúp thu hẹp dần không gian tìm kiếm theo quá trình khai thác.

Thuật toán cũng đảm bảo tính đầy đủ trong quá trình khai thác với việc bắt đầu từ tập các 1-pattern và mở rộng dần để tạo thành các mẫu có kích thước lớn hơn thông qua các lớp tương đương. Các ứng viên mới của mỗi lớp sau khi tạo thành sẽ được đánh giá theo ngưỡng $dWus$ để dùng cho bước mở rộng tiếp theo. Do đó, thuật toán chỉ kết hợp các mẫu nằm trong top- k FWUP hiện tại để tìm kiếm các mẫu mới và cơ chế mở rộng giúp tránh hiện tượng sinh ứng viên trùng lặp, không bỏ sót kết quả. Ngoài ra, TKFWUDS còn sử dụng hàng đợi ưu tiên PQ để lưu trữ top- k FWUP. PQ không làm thay đổi không gian tìm kiếm hay cơ chế sinh mẫu mà chỉ hỗ trợ quá trình cập nhật nhanh ngưỡng động $dWus$. Giá trị nhỏ nhất trong PQ luôn được sử dụng làm ngưỡng cắt tĩa hiện tại, nên tập top- k luôn được duy trì chính xác trong suốt quá trình khai thác. Việc sử dụng hàng đợi ưu tiên không ảnh hưởng đến tính đầy đủ của thuật toán vì mọi mẫu thỏa điều kiện vẫn được xem xét trước khi quyết định cập nhật hoặc loại bỏ khỏi PQ . Do đó, kết thúc quá trình khai thác của từng cửa sổ, tập

kết quả chứa đúng k mẫu có giá trị wus lớn nhất và giá trị cuối cùng của $dWus$ chính là $opWus$ của cửa sổ đó.

Về độ phức tạp tính toán, TKFWUDS sử dụng cấu trúc DSWUN-tree với chi phí khởi tạo cây là $O(m \cdot l_{max} \cdot \log l_{max})$ và chi phí cập nhật khi trượt khung cửa sổ là $O(p \cdot l_{max} \cdot \log l_{max})$, trong đó m là số lượng giao dịch, l_{max} là độ dài giao dịch lớn nhất trong một cửa sổ và p là kích thước pane. Tiếp theo, thuật toán gọi thủ tục *GenTKCandidate* để tiến hành khai thác. Khác với phương pháp khai thác FWUP truyền thống, không gian tìm kiếm của TKFWUDS được giới hạn mạnh bởi cơ chế top- k và các chiến lược cắt tia động. Cụ thể, tại mỗi mức khai thác, theo cơ chế duy trì top- k , số lượng mẫu ứng viên được giữ để tiếp tục mở rộng không vượt quá k . Vì vậy, số phép kết hợp tối đa giữa các mẫu trong một mức được giới hạn bởi $k(k-1)/2$. Với $wunl_{max}$ là chiều dài tối đa của WUNList, khi đó, độ phức tạp thời gian của giai đoạn khai thác là $O(l_{max} \cdot k^2 \cdot (wunl_{max} + \log k))$.

Trong thực tế, số lượng phép kết hợp thường ít do tác động đồng thời của tính phân đơn điệu và cơ chế nâng ngưỡng động. Khi quá trình khai thác diễn ra, giá trị $dWus$ tăng dần, kéo theo số lượng mẫu đủ điều kiện mở rộng giảm nhanh theo độ sâu của không gian tìm kiếm. Đồng thời, Chiến lược 2 và Chiến lược 3 giúp loại bỏ phần lớn các ứng viên trước khi thực hiện phép kết hợp WUNList, là thao tác chiếm chi phí lớn của thuật toán. Về độ phức tạp bộ nhớ, thuật toán cần lưu trữ DSWUN-tree, WUNList, ma trận PTP và hàng đợi ưu tiên PQ . Trong đó, PQ chỉ lưu tối đa k phần tử nên chi phí bộ nhớ bổ sung cho cơ chế top- k là không đáng kể. Mặc dù việc sử dụng thêm ma trận PTP làm tăng chi phí lưu trữ nhưng cấu trúc này giúp giảm đáng kể số lượng phép kiểm tra, từ đó cải thiện hiệu quả tổng thể của thuật toán trên các luồng dữ liệu có quy mô lớn và thưa thớt.

3.4.4 Thực nghiệm và kết quả

3.4.4.1 Thiết lập thực nghiệm

Phần này tiến hành các thực nghiệm đánh giá hiệu suất thuật toán TKFWUDS để khai thác top- k FWUP trên luồng dữ liệu định lượng động sử dụng mô hình của

sổ trượt. Tất cả các thuật toán đều được triển khai bằng ngôn ngữ lập trình C# 2022 (.NET Framework 4.7.2). Các thực nghiệm được tiến hành trên máy tính với cấu hình CPU Intel Core i3-2310M 2.10 GHz, 8 GB RAM, và chạy hệ điều hành Windows 10. Bảy bộ dữ liệu thực nghiệm bao gồm Chess, Connect, Accidents, Retail, BMS-POS, Kosarak và Susy, đã được mô tả trong Bảng 2.6. Các thực nghiệm trong phần này sử dụng cùng dữ liệu và cùng bộ trọng số động đã được tạo và sử dụng trong các thực nghiệm trước của luận án, qua đó bảo đảm các thuật toán được đánh giá trên cùng đầu vào. Các tham số kích thước cửa sổ, kích thước pane, giá trị k trong từng thực nghiệm cụ thể được mô tả trong Bảng 3.6 và Bảng 3.7.

Bảng 3.6. Các tham số thực nghiệm khai thác top- k FWUP trên luồng dữ liệu

CSDL	Kích thước cửa sổ	Kích thước pane	Các giá trị k
Chess	3000P	1	50, 100, 200, 300, 400
Connect	67000P	1	50, 100, 150, 200, 250
Accidents	340000P	1	50, 100, 150, 200, 250
Retail	88000P	1	500, 1000, 1500, 2000, 2500
BMS-POS	500000P	1	1000, 1200, 1400, 1600, 1800
Kosarak	900000P	1	200, 400, 600, 800, 1000
Susy	4000000P	1	30, 50, 70, 90, 110

Bảng 3.7. Các tham số thực nghiệm khả năng mở rộng của các thuật toán khai thác top- k FWUP trên luồng dữ liệu theo kích thước cửa sổ và pane

CSDL	k	Các cấu hình (Kích thước cửa sổ - Kích thước pane)
Accidents	100	(300P – 500), (350P – 600), (400P – 700)
BMS-POS	1400	(1100P – 100), (1300P – 200), (1500P – 300)

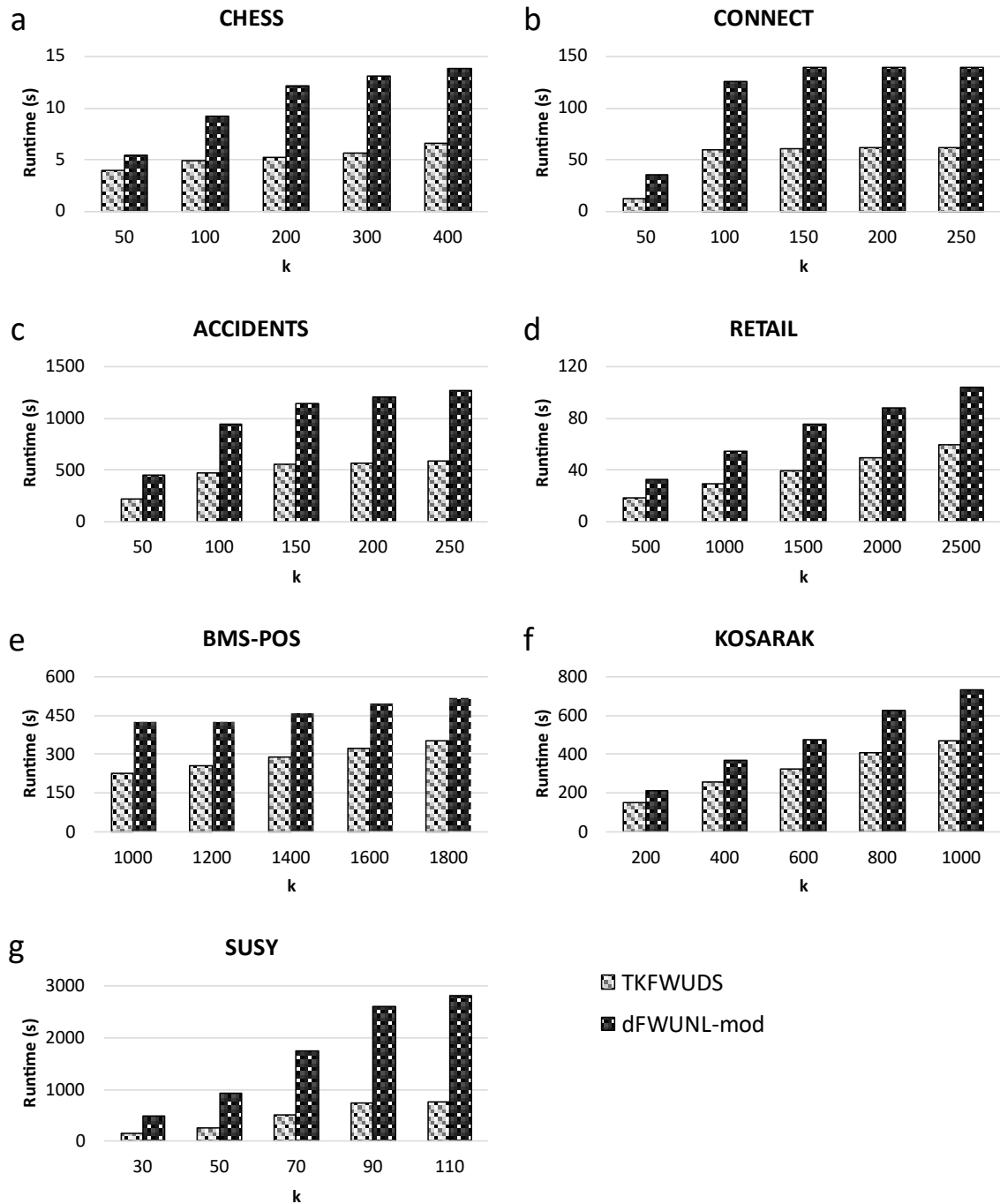
Kosarak	800	(700P – 600), (750P – 800), (800P – 1000)
Susy	70	(800P – 3000), (900P – 3500), (1000P – 4000)

Quy trình thực nghiệm đối với mỗi bộ dữ liệu bao gồm các bước sau. Đầu tiên, các thuật toán sẽ quét và xử lý dữ liệu ở cửa sổ đầu tiên, tạo ra cấu trúc dữ liệu để lưu trữ và khai thác top- k FWUP trong cửa sổ này. Sau đó, cơ chế trượt cửa sổ sẽ được thực hiện với việc thêm pane dữ liệu mới vào. Tiếp theo, các thuật toán tiến hành cập nhật dữ liệu và khai thác top- k FWUP trong cửa sổ mới. Toàn bộ quá trình trượt cửa sổ được thực hiện tổng cộng 50 lần cho mỗi lần thử nghiệm trên từng bộ dữ liệu. Do hiện chưa có thuật toán nào áp dụng cho bài toán khai thác top- k FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt, nên thuật toán dFWUNL, là thuật toán tốt nhất khai thác FWUP trên CSDL định lượng động, được chỉnh sửa làm thuật toán cơ sở và đặt tên là dFWUNL-mod. Thuật toán dFWUNL-mod giữ nguyên cấu trúc WUN-tree, WUNList và cơ chế sinh mẫu của dFWUNL. Vì dFWUNL không được thiết kế cho luồng dữ liệu, nên thuật toán này được chạy trên mỗi cửa sổ riêng biệt. Đồng thời, để đảm bảo công bằng trong việc tìm top- k FWUP, Chiến lược 1 và Chiến lược 2 được áp dụng cho thuật toán dFWUNL-mod thay thế cho việc sử dụng *minwus*. Các thay đổi này không làm thay đổi cách tính giá trị *wus* hay phép sinh WUNList và cho phép cả hai thuật toán được đánh giá trong điều kiện khai thác top- k như nhau. Kết quả trên các bộ dữ liệu thực nghiệm đều cho thấy trong mỗi cửa sổ, TKFWUDS và dFWUNL-mod đều thu được k FWUP có cùng các giá trị *wus* tương ứng và cùng giá trị *opWus*. Trường hợp có các mẫu đồng hạng tại vị trí thứ k , thành phần cụ thể của tập kết quả có thể phụ thuộc vào thứ tự phát hiện theo quy tắc đồng hạng đã xác định, nhưng không làm thay đổi các mẫu kết quả còn lại và *opWus*.

3.4.4.2 So sánh về thời gian khai thác

Hình 3.24 hiển thị tổng thời gian khai thác top- k FWUP trên các bộ dữ liệu với các ngưỡng k khác nhau và kích thước pane được cố định là 1. Kích thước cửa sổ cho các bộ dữ liệu Chess, Connect, Accidents, Retail, BMS-POS, Kosarak và Susy được đặt lần lượt là 3000P, 67000P, 340000P, 88000P, 500000P, 900000P và

4000000P. Thời gian chạy được tính từ lúc bắt đầu cho đến khi thuật toán kết thúc và trả về kết quả của các cửa sổ.



Hình 3.24. Thời gian chạy của thuật toán TKFWUDS và dFWUNL-mod trên các CSDL thực nghiệm.

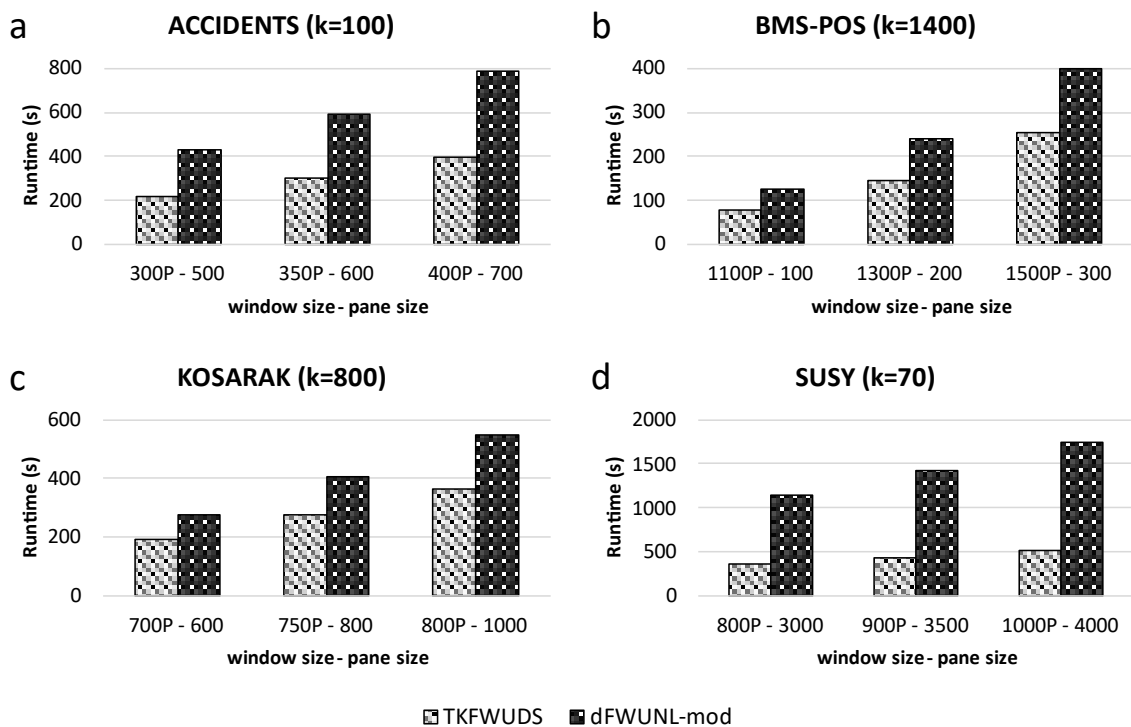
Ta thấy rằng khi ngưỡng k càng tăng thì thời gian khai thác của các thuật toán cũng tăng. Xu hướng này được thể hiện rõ trong các biểu đồ. Trong mọi trường hợp, thuật toán TKFWUDS đều vượt trội hơn thuật toán dFWUNL-mod. Ví dụ, trên bộ dữ liệu Chess, thuật toán TKFWUDS và dFWUNL-mod có thời gian chạy lần lượt là 4.9 giây và 9.2 giây với ngưỡng k bằng 100, và mất 6.6 giây và 13.9 giây khi ngưỡng k bằng 400. Tương tự, trên bộ dữ liệu Kosarak với ngưỡng k bằng 200, thuật toán TKFWUDS mất 147.8 giây, còn thuật toán dFWUNL-mod mất 213.1 giây; và khi ngưỡng k bằng 800, thuật toán TKFWUDS mất 407.8 giây, trong khi thuật toán dFWUNL-mod mất 625.3 giây. Sự gia tăng thời gian khai thác là do khi ngưỡng k lớn, số lượng mẫu tạo ra nhiều hơn và các thuật toán cần nhiều thời gian hơn để xử lý. Thuật toán dFWUNL-mod sử dụng cấu trúc WUN-tree được thiết kế cho CSDL tĩnh, yêu cầu hai lần quét toàn bộ dữ liệu cho mỗi cửa sổ, trong khi, DSWUN-tree có thể tái sử dụng dữ liệu từ các cửa sổ trước đó. Quá trình cập nhật của nó chỉ đơn giản là thêm dữ liệu mới và xóa dữ liệu cũ nhất mà không cần quét lại CSDL. Hơn nữa, thuật toán TKFWUDS còn kết hợp với các chiến lược dựa trên một ngưỡng hỗ trợ trọng số hữu ích động giúp loại bỏ nhanh các mẫu không thoả mãn, tăng tốc quá trình khai thác. Điều này làm cho thuật toán TKFWUDS nhanh hơn thuật toán dFWUNL-mod trên tất cả các bộ dữ liệu dày và thưa.

3.4.4.3 So sánh về khả năng mở rộng

Các thực nghiệm đánh giá khả năng mở rộng của các thuật toán được tiến hành bằng cách tăng dần kích thước cửa sổ và kích thước pane. Khả năng mở rộng đề cập đến khả năng của các thuật toán xử lý khối lượng dữ liệu ngày càng tăng trong khi vẫn duy trì hiệu suất chấp nhận được. Bốn bộ dữ liệu lớn được lựa chọn để kiểm tra khả năng mở rộng của các thuật toán là Accidents, BMS-POS, Kosarak và Susy. Hình 3.25 hiển thị thời gian khai thác của thuật toán TKFWUDS và dFWUNL-mod với các kích thước cửa sổ và kích thước pane khác nhau. Kết quả cho thấy TKFWUDS có thời gian chạy thấp hơn đáng kể so với dFWUNL-mod trên tất cả các cấu hình thực nghiệm. Hơn nữa, khi kích thước cửa sổ và pane tăng lên, thời gian chạy của cả hai thuật toán đều tăng, nhưng sự chênh lệch hiệu năng giữa TKFWUDS và

dFWUNL-mod ngày càng lớn. Sự gia tăng về thời gian này là do khối lượng dữ liệu cần xử lý trong cửa sổ trượt lớn hơn. Ví dụ, bộ dữ liệu BMS-POS trong Hình 3.25b, với ngưỡng $k = 1400$, kích thước cửa sổ là 1100P và kích thước pane là 100 giao dịch, thuật toán TKFWUDS mất 77.4 giây, còn thuật toán dFWUNL-mod mất 220.6 giây. Khi kích thước cửa sổ tăng lên thành 1500P và kích thước pane là 3500 giao dịch, TKFWUDS chỉ mất 255.4 giây, trong khi dFWUNL-stream mất 399.8 giây. Như vậy, thuật toán TKFWUDS có khả năng mở rộng tốt hơn, đảm bảo hiệu suất khi kích thước dữ liệu trong luồng dữ liệu thay đổi.

Bảng 3.8 tổng hợp thời gian thực thi của TKFWUDS và dFWUNL-mod tại các giá trị k đại diện, đồng thời hiển thị các số liệu về khả năng mở rộng theo các cấu hình kích thước cửa sổ và pane được tiến hành trên các luồng dữ liệu thực nghiệm.



Hình 3.25. Khả năng mở rộng về thời gian khai thác của thuật toán TKFWUDS và dFWUNL-mod trên các CSDL thực nghiệm.

Bảng 3.8. Thời gian thực thi và khả năng mở rộng của các thuật toán khai thác top- k FWUP trên các luồng dữ liệu thực nghiệm

CSDL	k	Kích thước cửa sổ	Kích thước pane	TKFWUDS (s)	dFWUNL-mod (s)
A. So sánh thời gian khai thác					
Chess	400	3000P	1	6.6	13.9
Connect	150	67000P	1	62.2	139.4
Accidents	250	340000P	1	591	1267.6
Retail	2500	88000P	1	59.7	104.4
BMS-POS	1800	500000P	1	350.2	518.8
Kosarak	1000	900000P	1	466.1	729
Susy	110	4000000P	1	762.1	2809.9
B. So sánh khả năng mở rộng					
Accidents	100	300P	500	218.5	428.3
		350P	600	300.7	593
		400P	700	397.8	788.6
BMS-POS	1400	1100P	100	77.4	125.5
		1300P	200	145.6	240.4
		1500P	300	255.4	399.8
Kosarak	800	700P	600	193.1	277.9
		750P	800	276.6	404.7
		800P	1000	364.5	549.6
Susy	70	800P	3000	361.1	1136.4
		900P	3500	433.5	1429
		1000P	4000	508.1	1748.7

3.5 Kết luận chương

Chương 3 tập trung nghiên cứu về bài toán khai thác FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt. Trên cơ sở kế thừa và phát triển các nền tảng từ Chương 2, luận án đã đề xuất hai thuật toán là FWUDS-CT và FWUDS-DWT để quản lý sự thay đổi động của trọng số mục và sự gia nhập liên tục của các giao dịch trong luồng dữ liệu. Cả hai thuật toán đều được xây dựng với khả năng tận dụng tri thức trước đó mà không cần quét lại toàn bộ cửa sổ. Thuật toán FWUDS-CT nổi bật với cấu trúc dữ liệu Circular Tidset cải tiến, cho phép xoay vòng các định danh giao dịch để tránh việc xử lý số lớn khi luồng dữ liệu kéo dài vô hạn. Trong khi đó, thuật toán FWUDS-DWT lại tập trung vào việc nén dữ liệu vào cấu trúc DSWUN-tree, là cải tiến của cấu trúc WUN-tree, và tạo ra danh sách WUNList cho các mẫu, giúp quá trình cập nhật và khai thác diễn ra nhanh chóng chỉ với một lần quét dữ liệu. Các kết quả thực nghiệm đã khẳng định rằng, mặc dù tiêu tốn nhiều bộ nhớ hơn, thuật toán FWUDS-DWT đạt thời gian thực thi thấp hơn và khả năng mở rộng tốt hơn so với các phương pháp tiếp cận khác, đặc biệt là trong việc xử lý các pane dữ liệu mới và loại bỏ dữ liệu cũ trong cửa sổ trượt.

Không chỉ dừng lại ở việc khai thác tập mẫu đầy đủ, chương này còn giải quyết thách thức của bài toán khai thác top- k FWUP trên luồng dữ liệu thông qua thuật toán TKFWUDS. TKFWUDS tận dụng sự hiệu quả của cấu trúc DSWUN-tree để cập nhật dữ liệu mới, kết hợp cùng các chiến lược cắt tỉa và kỹ thuật nâng ngưỡng dựa trên khái niệm ngưỡng hỗ trợ trọng số hữu ích động. Sự kết hợp này đã giúp giảm thiểu đáng kể chi phí tính toán và không gian tìm kiếm mẫu ứng viên. Các đánh giá thực nghiệm cho thấy tính hiệu quả của thuật toán TKFWUDS về cả thời gian khai thác lẫn khả năng mở rộng so với hướng tiếp cận truyền thống. Các cấu trúc dữ liệu và chiến lược cắt tỉa được đề xuất trong Chương 3 không chỉ giải quyết các yêu cầu khắt khe về mặt tài nguyên mà còn cung cấp một bộ công cụ mạnh mẽ để khai thác các tri thức quan trọng nhất từ luồng dữ liệu. Đây là cơ sở để tiếp tục nghiên cứu, đánh giá khả năng ứng dụng của các thuật toán trong những môi trường luồng dữ liệu thực tế với đặc trưng biến động và yêu cầu xử lý đa dạng hơn.

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

1. Kết luận

Luận án đã tập trung nghiên cứu, phát triển các cấu trúc dữ liệu và đề xuất các thuật toán để giải quyết hiệu quả một số bài toán khai thác mẫu trọng số hữu ích phổ biến trong môi trường dữ liệu định lượng động.

Đối với khai thác FWUP trên CSDL định lượng động, luận án đã xây dựng hệ thống định nghĩa và mô hình hóa bài toán này để giải quyết hạn chế của các mô hình truyền thống vốn mặc định trọng số các mục là cố định. Luận án đã đề xuất hai thuật toán nền tảng là dFWUT sử dụng cấu trúc Tidset và thuật toán dFWUNL sử dụng cấu trúc WUNList. Là mở rộng của cấu trúc N-list, WUNList thể hiện sự hiệu quả nhờ khả năng nén dữ liệu, cơ chế giao các WUNList và khả năng tự giảm kích thước sau mỗi lần giao mẫu. Hơn nữa, thuật toán dFWUNL còn kết hợp các phương pháp tối ưu hóa như tính toán trước mẫu có hai phần tử và cơ chế xác định nhanh giá trị độ hỗ trợ trọng số hữu ích của các mẫu. Kết quả thực nghiệm trên các bộ dữ liệu dày và thưa cho thấy thuật toán dFWUNL mang lại hiệu suất vượt trội so với dFWUT và hướng tiếp cận truyền thống về cả thời gian thực thi lẫn bộ nhớ sử dụng.

Bên cạnh việc khai thác tập FWUP đầy đủ, luận án cũng đã đề xuất thuật toán FWUCWP để giải quyết bài toán khai thác mẫu trọng số hữu ích phổ biến đóng từ CSDL định lượng động. Thuật toán FWUCWP được đề xuất không chỉ kế thừa sức mạnh của cấu trúc WUNList mà còn tích hợp một hệ thống các chiến lược cắt tỉa chuyên sâu bao gồm cắt tỉa dựa trên quan hệ cấu trúc tổ tiên WUNList, cắt tỉa kết hợp *twu* tối đa và chiến lược cắt tỉa dựa trên khái niệm Enclosure. Những cải tiến này giúp thu hẹp nhanh chóng không gian tìm kiếm và cải thiện hiệu năng rõ rệt, đặc biệt là trên các tập dữ liệu thưa thớt với tốc độ xử lý tốt hơn so với thuật toán tốt nhất hiện nay, đồng thời vẫn bảo toàn trọn vẹn tri thức cần thiết cho việc ra quyết định.

Đối với bài toán khai thác trên luồng dữ liệu định lượng động, luận án đã mở rộng thành công mô hình khai thác sang bối cảnh dữ liệu đến liên tục, đòi hỏi xử lý một lần với thời gian và tài nguyên bộ nhớ hạn chế. Thông qua việc sử dụng mô hình

cửa sổ trượt, luận án đã đề xuất thuật toán FWUDS-CT sử dụng cấu trúc Circular Tidset, giải quyết hiệu quả vấn đề tràn số định danh giao dịch khi luồng dữ liệu kéo dài. Song song với đó, luận án cũng phát triển thuật toán FWUDS-DWT với cấu trúc DSWUN-tree cho phép cập nhật các dữ liệu mới và loại bỏ dữ liệu lỗi thời trong cửa sổ trượt chỉ với một lần quét dữ liệu. Với sự kết hợp giữa cấu trúc DSWUN-tree và WUNList đã giúp thuật toán FWUDS-DWT đạt được tốc độ khai thác FWUP và khả năng cập nhật dữ liệu hiệu quả, đáp ứng yêu cầu khắt khe của luồng dữ liệu.

Đóng góp cuối cùng của luận án là phát triển thuật toán TKFWUDS để khai thác top- k FWUP trên luồng dữ liệu định lượng động sử dụng mô hình cửa sổ trượt. Luận án đã đề xuất khái niệm ngưỡng hỗ trợ trọng số hữu ích động trên mỗi cửa sổ và kỹ thuật nâng ngưỡng tự động. Các chiến lược cắt tĩa dựa trên ngưỡng động này cũng đc áp dụng giúp trích xuất nhanh chóng và chính xác k FWUP giá trị nhất mà không cần người dùng thiết lập ngưỡng hỗ trợ trọng số hữu ích tối thiểu, là một rào cản lớn trong khai thác luồng dữ liệu thực tế.

Các kết quả nghiên cứu nêu trên đã được công bố trên các tạp chí và hội thảo trong nước và quốc tế uy tín. Những kết quả này không chỉ đóng góp về mặt học thuật trong lĩnh vực khai thác dữ liệu mà còn cung cấp bộ giải pháp thực tiễn mạnh mẽ cho các hệ thống hỗ trợ ra quyết định thông minh dựa trên dữ liệu định lượng và luồng dữ liệu lớn.

2. Hướng phát triển

Mặc dù đã đạt được những kết quả khả quan, luận án vẫn mở ra nhiều hướng nghiên cứu tiềm năng nhằm đáp ứng sự phức tạp ngày càng tăng của dữ liệu lớn. Trong tương lai, một trong những ưu tiên hàng đầu là việc tối ưu hóa hiệu năng tính toán thông qua các kiến trúc lập trình song song như bộ xử lý đa lõi, CUDA hoặc các hệ thống tính toán phân tán như Spark, Hadoop. Việc áp dụng các kỹ thuật này không chỉ giúp xử lý các CSDL ở quy mô lớn hơn mà còn cho phép thuật toán thực thi trong các môi trường đòi hỏi độ trễ thấp. Bên cạnh việc tiếp tục cải thiện hiệu năng, một hướng nghiên cứu trong tương lai là mở rộng phạm vi đánh giá các phương pháp đề

xuất trên những luồng dữ liệu thực tế với các đặc trưng và quy mô khác nhau. Việc thực nghiệm trong các điều kiện tốc độ dữ liệu đến không đồng đều, khối lượng dữ liệu lớn và yêu cầu khác nhau về độ trễ xử lý sẽ giúp đánh giá toàn diện hơn khả năng đáp ứng và tính khả thi của các thuật toán trong các hệ thống xử lý luồng dữ liệu thực tế. Cùng với đó, nghiên cứu sẽ tập trung vào việc phát triển các cơ chế thích ứng cho luồng dữ liệu, cho phép hệ thống tự động điều chỉnh kích thước pane dữ liệu hoặc tích hợp các chiến lược giảm tải để duy trì sự ổn định khi luồng dữ liệu xuất hiện các biến động bất thường hoặc bùng nổ về lưu lượng.

Một hướng đi quan trọng khác là mở rộng mô hình khai thác lên các loại hình dữ liệu không chắc chắn, nơi sự tồn tại của các mục hoặc trọng số của chúng mang tính xác suất. Điều này đòi hỏi việc tái cấu trúc các định nghĩa và chiến lược cắt tĩa để đảm bảo tính chính xác của tri thức trích xuất được. Cuối cùng, việc đa dạng hóa các mô hình cửa sổ khai thác như cửa sổ dựa trên thời gian thực tế hoặc cửa sổ mốc, đồng thời tích hợp các cơ chế khai thác tăng trưởng để tối ưu hóa việc tái sử dụng tri thức cũ cũng là một hướng đi tiềm năng. Những nghiên cứu này không chỉ hoàn thiện bức tranh về khai thác FWUP mà còn góp phần xây dựng các hệ thống thông minh, linh hoạt và mạnh mẽ hơn trong tương lai.

DANH MỤC CÔNG TRÌNH KHOA HỌC CỦA TÁC GIẢ LIÊN QUAN ĐẾN LUẬN ÁN

- [I] **Le N.**, Bui H., Nguyen M., Nguyen H., and Le T. (2022), “Mining frequent weighted utility patterns in dynamic quantitative databases”, *2022 RIVF International Conference on Computing and Communication Technologies (RIVF)*, pp. 536–541.
- [II] Nguyen H., **Le N.**, Bui H., and Le T. (2023), “Mining frequent weighted utility patterns with dynamic weighted items from quantitative databases”, *Applied Intelligence* 53(16), pp. 19629–19646.
- [III] **Lê Hoàng Bình Nguyên**, Nguyễn Duy Hàm, Nguyễn Thị Hồng Minh (2023), “Thuật toán song song khai thác nhanh các mẫu trọng số hữu ích phổ biến từ cơ sở dữ liệu định lượng động”, *Chuyên san Các công trình nghiên cứu, phát triển và ứng dụng Công nghệ Thông tin và Truyền thông* 2023(2).
- [IV] **Le N.**, Nguyen H., Bui H., and Nguyen M. (2025), “The Problem Of Mining Top-K Frequent Pattern Over Dynamic Quantitative Data Stream”, *The 18th National Conference on Fundamental and Applied Information Technology Research (FAIR'2025)*, pp. 615-621.
- [V] **Le N.**, Nguyen H., Nguyen M., Bui H., Yun U., and Vo B. (2026), “Efficiently mining frequent weighted utility closed patterns with pruning strategies from dynamic quantitative databases”, *The Journal of Supercomputing* 82(3), pp. 129.
- [VI] **Le N.**, Nguyen H., Nguyen M., Bui H., Yun U., and Vo B. (2026), “Fast and scalable sliding-window-based algorithms for mining frequent weighted utility patterns over dynamic quantitative data streams”, *Information Sciences* 748, pp. 123488.

TÀI LIỆU THAM KHẢO

1. Agrawal R., Imieliński T., Swami A. (1993), "Mining association rules between sets of items in large databases", *ACM sigmod record*, ACM, pp. 207–216.
2. Aryabarzan N., Minaei-Bidgoli B. (2021), "NEclatClosed: A vertical algorithm for mining frequent closed itemsets", *Expert Systems with Applications* 174, pp. 114738.
3. Baek Y., Yun U., Kim H., Nam H., Lee G., Yoon E., Vo B., Lin J.C.-W. (2020), "Erasable pattern mining based on tree structures with damped window over data streams", *Engineering Applications of Artificial Intelligence* 94, pp. 103735.
4. Bui H., Nguyen-Hoang T.-A., Vo B., Nguyen H., Le T. (2021), "A Sliding Window-Based Approach for Mining Frequent Weighted Patterns Over Data Streams", *IEEE Access* 9, pp. 56318–56329.
5. Bui H., Vo B., Nguyen H. (2016), "WUN-miner: A new method for mining frequent weighted utility itemsets", *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, IEEE, pp. 001365–001370.
6. Bui H., Vo B., Nguyen H., Nguyen-Hoang T.-A., Hong T.-P. (2018), "A weighted N-list-based method for mining frequent weighted itemsets", *Expert Systems with Applications* 96, pp. 388–405.
7. Bui H., Vo B., Nguyen-Hoang T.-A., Yun U. (2021), "Mining frequent weighted closed itemsets using the WN-list structure and an early pruning strategy", *Applied Intelligence* 51(3), pp. 1439–1459.
8. Calders T., Dexters N., Gillis J.J.M., Goethals B. (2014), "Mining frequent itemsets in a stream", *Information Systems* 39, pp. 233–255.
9. Chang J.H., Lee W.S. (2006), "Finding recently frequent itemsets adaptively over online transactional data streams", *Information Systems* 31(8), pp. 849–869.
10. Chuang K.-T., Huang J.-L., Chen M.-S. (2008), "Mining top-k frequent patterns in the presence of the memory constraint", *The VLDB Journal* 17(5), pp. 1321–1344.
11. Dam T.-L., Li K., Fournier-Viger P., Duong Q.-H. (2019), "CLS-Miner: efficient and effective closed high-utility itemset mining", *Frontiers of Computer Science* 13(2), pp. 357–381.
12. Davashi R. (2023), "ITUFP: A fast method for interactive mining of Top-K frequent patterns from uncertain data", *Expert Systems with Applications* 214, pp. 119156.
13. Deng Z.-H. (2016), "DiffNodesets: An efficient structure for fast mining

- frequent itemsets", *Applied Soft Computing* 41, pp. 214–223.
14. Deng Z., Wang Z., Jiang J. (2012), "A new algorithm for fast mining frequent itemsets using N-lists", *Science China Information Sciences* 55(9), pp. 2008–2030.
 15. Duong H., Hoang T., Tran T., Truong T., Le B., Fournier-Viger P. (2022), "Efficient algorithms for mining closed and maximal high utility itemsets", *Knowledge-Based Systems* 257, pp. 109921.
 16. Duong H., Truong T., Le B., Fournier-Viger P. (2024), "Mining Interesting Sequential Patterns using a Novel Balanced Utility Measure", *Knowledge-Based Systems* 294, pp. 111796.
 17. Duong Q.-H., Fournier-Viger P., Ramampiaro H., Nørvåg K., Dam T.-L. (2018), "Efficient high utility itemset mining using buffered utility-lists", *Applied Intelligence* 48(7), pp. 1859–1877.
 18. Duong Q.-H., Liao B., Fournier-Viger P., Dam T.-L. (2016), "An efficient algorithm for mining the top- k high utility itemsets, using novel threshold raising and pruning strategies", *Knowledge-Based Systems* 104, pp. 106–122.
 19. Fournier-Viger P., Gomariz A., Gueniche T., Soltani A., Wu C.-W., Tseng V.S. (2014), "SPMF: a Java open-source pattern mining library", *The Journal of Machine Learning Research* 15(1), pp. 3389–3393.
 20. Fournier-Viger P., Wu C.-W., Zida S., Tseng V.S. (2014), *FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning*, Springer International Publishing, Cham.
 21. Fournier-Viger P., Zida S., Lin J.C.-W., Wu C.-W., Tseng V.S. (2016), *EFIM-Closed: Fast and Memory Efficient Discovery of Closed High-Utility Itemsets*, Springer International Publishing, Cham.
 22. Goethals B., Zaki M.J. (2003), "FIMI'03: Workshop on frequent itemset mining implementations", *Third IEEE International Conference on Data Mining Workshop on Frequent Itemset Mining Implementations*, pp. 1–13.
 23. Grahne G., Zhu J. (2005), "Fast algorithms for frequent itemset mining using FP-trees", *IEEE Transactions on Knowledge and Data Engineering* 17(10), pp. 1347–1362.
 24. Grust T. (2002), "Accelerating XPath location steps", *SIGMOD/PODS02: International Conference on Management of Data and Symposium on Principles Database and Systems*, ACM, Madison Wisconsin, pp. 109–120.
 25. Ham N.D., Bay V.D., Minh N.T.H., Hong T.-P. (2015), "MBiS: an efficient method for mining frequent weighted utility itemsets from quantitative databases", *Journal of Computer Science and Cybernetics* 31(1), pp. 17–30.
 26. Han J., Pei J., Yin Y. (2000), "Mining frequent patterns without candidate

- generation", *ACM sigmod record*, ACM, pp. 1–12.
27. Han M., Li M., Chen Z., Wu H., Zhang X. (2023), "High utility pattern mining algorithm over data streams using ext-list.", *Applied Intelligence* 53(22), pp. 27072–27095.
 28. He J., Han X., Wan X., Wang J. (2024), "Efficient Skyline Frequent-Utility Itemset Mining Algorithm on Massive Data", *IEEE Transactions on Knowledge and Data Engineering* 36(7), pp. 3009–3023.
 29. HewaNadungodage C., Xia Y., Lee J.J., Tu Y. (2013), "Hyper-structure mining of frequent patterns in uncertain data streams", *Knowledge and Information Systems* 37(1), pp. 219–244.
 30. Iqbal S., Shahid A., Roman M., Khan Z., Al-Otaibi S., Yu L. (2021), "TKFIM: Top-K frequent itemset mining technique based on equivalence classes", *PeerJ Computer Science* 7, pp. e385.
 31. Khan M.S., Muyeba M., Coenen F. (2008), "A weighted utility framework for mining association rules", *2008 Second UKSIM European Symposium on Computer Modeling and Simulation*, IEEE, pp. 87–92.
 32. Kim H., Kim H., Cho M., Vo B., Chun-Wei Lin J., Fujita H., Yun U. (2024), "Efficient approach of high average utility pattern mining with indexed list-based structure in dynamic environments", *Information Sciences* 657, pp. 119924.
 33. Krishnamoorthy S. (2017), "HMiner: Efficiently mining high utility itemsets", *Expert Systems with Applications* 90, pp. 168–183.
 34. Krishnamoorthy S. (2019), "Mining top-k high utility itemsets with effective threshold raising strategies", *Expert Systems with Applications* 117, pp. 148–165.
 35. Le T., Vo B. (2015), "An N-list-based algorithm for mining frequent closed patterns", *Expert Systems with Applications* 42(19), pp. 6648–6657.
 36. Le T., Vo B., Fournier-Viger P., Lee M.Y., Baik S.W. (2019), "SPPC: a new tree structure for mining erasable patterns in data streams", *Applied Intelligence* 49(2), pp. 478–495.
 37. Le T., Vo B., Huynh V.-N., Nguyen N.T., Baik S.W. (2020), "Mining top-k frequent patterns from uncertain databases", *Applied Intelligence* 50(5), pp. 1487–1497.
 38. Lee C., Ryu T., Kim H., Kim H., Vo B., Lin J.C.-W., Yun U. (2022), "Efficient approach of sliding window-based high average-utility pattern mining with list structures", *Knowledge-Based Systems* 256, pp. 109702.
 39. Lee G., Yun U., Ryu K.H. (2017), "Mining frequent weighted itemsets without storing transaction ids and generating candidates", *International Journal of*

Uncertainty, Fuzziness and Knowledge-Based Systems 25(01), pp. 111–144.

40. Li G., Shang T., Zhang Y. (2023), "Efficient mining high average-utility itemsets with effective pruning strategies and novel list structure", *Applied Intelligence* 53, pp. 6099–6118.
41. Li G., Wang J., Fang W., Xiang J., Shang T. (2025), "Efficient high utility itemsets mining over data streams with compact utility list structure", *Applied Intelligence* 55(12), pp. 872.
42. Li H.-F., Lee S.-Y. (2009), "Mining frequent itemsets over data streams using efficient window sliding techniques", *Expert Systems with Applications* 36(2), pp. 1466–1477.
43. Li M., Han M., Chen Z., Wu H., Zhang X. (2023), "FCHM-stream: fast closed high utility itemsets mining over data streams", *Knowledge and Information Systems* 65(6), pp. 2509–2539.
44. Liu J., Ye Z., Yang X., Wang X., Shen L., Jiang X. (2022), "Efficient strategies for incremental mining of frequent closed itemsets over data streams", *Expert Systems with Applications* 191, pp. 116220.
45. Liu M., Qu J. (2012), "Mining high utility itemsets without candidate generation", *Proceedings of the 21st ACM international conference on Information and knowledge management*, pp. 55–64.
46. Liu X., Chen G., Wu F., Zuo W., Wu Y. (2026), "OPHEP-Miner: One-phase high-efficiency pattern mining utilizing tree structures", *Engineering Applications of Artificial Intelligence* 164, pp. 113288.
47. Liu Y., Liao W., Choudhary A. (2005), "A two-phase algorithm for fast discovery of high utility itemsets", *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, Springer, pp. 689–695.
48. Lucchese C., Orlando S., Perego R. (2006), "Fast and memory efficient mining of frequent closed itemsets", *IEEE Transactions on Knowledge and Data Engineering* 18(1), pp. 21–36.
49. Manku G.S., Motwani R. (2002), "Approximate frequency counts over data streams", *VLDB '02: Proceedings of the 28th International Conference on Very Large Databases*, Elsevier, pp. 346–357.
50. Mofid A.H., Daneshpour N., Torabi Z. (2024), "MMC: efficient and effective closed high-utility itemset mining", *The Journal of Supercomputing* 80(13), pp. 18900–18918.
51. Mohammed J. Zaki, Karam Gouda (2003), "Fast vertical mining using diffsets", *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, 956788, pp. 326–335.
52. Nam H., Yun U., Yoon E., Lin J.C.-W. (2020), "Efficient approach of recent

- high utility stream pattern mining with indexed list structure and pruning strategy considering arrival times of transactions", *Information Sciences* 529, pp. 1–27.
53. Ngo T.-K., Nguyen H.-S., Pedrycz W., Vo B. (2026), "Efficient algorithms for mining top-k high occupancy itemsets", *Information Sciences* 741, pp. 123226.
 54. Nguyen H., Le N., Bui H., Le T. (2023), "A new approach for efficiently mining frequent weighted utility patterns", *Applied Intelligence* 53(1), pp. 121–140.
 55. Nguyen H., Le T., Nguyen M., Fournier-Viger P., Tseng V.S., Vo B. (2022), "Mining frequent weighted utility itemsets in hierarchical quantitative databases", *Knowledge-Based Systems* 237, pp. 107709.
 56. Nguyen L.T., Nguyen P., Nguyen T.D., Vo B., Fournier-Viger P., Tseng V.S. (2019), "Mining high-utility itemsets in dynamic profit databases", *Knowledge-Based Systems* 175, pp. 130–144.
 57. Nguyen L.T.T., Vu V.V., Lam M.T.H., Duong T.T.M., Manh L.T., Nguyen T.T.T., Vo B., Fujita H. (2019), "An efficient method for mining high utility closed itemsets", *Information Sciences* 495, pp. 78–99.
 58. Nguyen T.D.D., Tung N.T., Nguyen L.T.T., Pham T.T., Vo B. (2024), "MLC-miner: Efficiently discovering multi-level closed high utility patterns from quantitative hierarchical transaction databases", *Expert Systems with Applications* 254, pp. 124383.
 59. Pasquier N., Bastide Y., Taouil R., Lakhal L. (1999), "Discovering frequent closed itemsets for association rules", *International Conference on Database Theory*, Springer, pp. 398–416.
 60. Pei J., Han J., Mao R. (2000), "CLOSET: An efficient algorithm for mining frequent closed itemsets", *ACM SIGMOD workshop on research issues in data mining and knowledge discovery*, pp. 21–30.
 61. Pyun G., Yun U. (2014), "Mining top-k frequent patterns with combination reducing techniques", *Applied Intelligence* 41(1), pp. 76–98.
 62. Ron Rymon (1992), *Search through systematic set enumeration*, Cambridge, MA.
 63. Ryang H., Yun U. (2016), "High utility pattern mining over data streams with sliding window technique", *Expert Systems with Applications* 57, pp. 214–231.
 64. Sahoo J., Das A.K., Goswami A. (2015), "An effective association rule mining scheme using a new generic basis", *Knowledge and Information Systems* 43(1), pp. 127–156.
 65. Tanbeer S.K., Ahmed C.F., Jeong B.-S., Lee Y.-K. (2009), "Sliding window-based frequent pattern mining over data streams", *Information Sciences* 179(22), pp. 3843–3865.

66. Trabold D., Horváth T. (2017), *Mining Strongly Closed Itemsets from Data Streams*, Springer International Publishing, Cham.
67. Tran T., Vo B., Le T.T.N., Nguyen N.T. (2017), "Text clustering using frequent weighted utility itemsets", *Cybernetics and Systems* 48(3), pp. 193–209.
68. Tseng V.S., Shie B.-E., Wu C.-W., Philip S.Y. (2012), "Efficient algorithms for mining high utility itemsets from transactional databases", *IEEE transactions on knowledge and data engineering* 25(8), pp. 1772–1786.
69. Tseng V.S., Wu C.-W., Fournier-Viger P., Yu P.S. (2015), "Efficient Algorithms for Mining the Concise and Lossless Representation of High Utility Itemsets", *IEEE Transactions on Knowledge and Data Engineering* 27(3), pp. 726–739.
70. Tseng V.S., Wu C.-W., Fournier-Viger P., Yu P.S. (2016), "Efficient Algorithms for Mining Top-K High Utility Itemsets", *IEEE Transactions on Knowledge and Data Engineering* 28(1), pp. 54–67.
71. Tseng V.S., Wu C.-W., Shie B.-E., Yu P.S. (2010), "UP-Growth: an efficient algorithm for high utility itemset mining", *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, Association for Computing Machinery, pp. 253–262.
72. Tung N.T., Nguyen T.D.D., Vu D.-L., Nguyen L.T.T. (2026), "An efficiency method for mining top-k multi-level high utility itemsets through multi-core parallelism", *Cluster Computing* 29(8), pp. 466.
73. Van T., Vo B., Le B. (2018), "Mining sequential patterns with itemset constraints", *Knowledge and Information Systems* 57(2), pp. 311–330.
74. Vo B. (2017), "An Efficient Method for Mining Frequent Weighted Closed Itemsets from Weighted Item Transaction Databases", *Journal of Information Science and Engineering* 33(1), pp. 199–216.
75. Vo B., Le B., Jung J.J. (2012), "A tree-based approach for mining frequent weighted utility itemsets", *International Conference on Computational Collective Intelligence*, Springer, pp. 114–123.
76. Vo B., Le T., Coenen F., Hong T.-P. (2016), "Mining frequent itemsets using the N-list and subsume concepts", *International Journal of Machine Learning and Cybernetics* 7(2), pp. 253–265.
77. Vu V.V., Lam M.T.H., Duong T.T.M., Manh L.T., Nguyen T.T.T., Nguyen L.V., Yun U., Snasel V., Vo B. (2023), "FTKHUIM: A Fast and Efficient Method for Mining Top-K High-Utility Itemsets", *IEEE Access* 11, pp. 104789–104805.
78. Wan X., Han X. (2024), "Efficient Top-k Frequent Itemset Mining on Massive Data", *Data Science and Engineering* 9(2), pp. 177–203.

79. Wang J., Han J., Pei J. (2003), "CLOSET+: searching for the best strategies for mining frequent closed itemsets", *KDD03: The Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, Washington, D.C., pp. 236–245.
80. Woo H.J., Lee W.S. (2008), "EstMax: Tracing maximal frequent item sets instantly over online transactional data streams", *IEEE transactions on knowledge and data engineering* 21(10), pp. 1418–1431.
81. Wu C.W., Shie B.-E., Tseng V.S., Yu P.S. (2012), "Mining top-K high utility itemsets", *KDD '12: The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ACM, Beijing China, pp. 78–86.
82. Wu C.-W., Fournier-Viger P., Gu J.-Y., Tseng V.S. (2015), "Mining closed+ high utility itemsets without candidate generation", *2015 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*, IEEE, Tainan, Taiwan, pp. 187–194.
83. Wu P., Niu X., Fournier-Viger P., Huang C., Wang B. (2022), "UBP-Miner: An efficient bit based high utility itemset mining algorithm", *Knowledge-Based Systems* 248, pp. 108865.
84. Xiao W., Hu J. (2020), "SWEclat: a frequent itemset mining algorithm over streaming data using Spark Streaming", *The Journal of Supercomputing* 76(10), pp. 7619–7634.
85. Yan Y., Niu X., Zhang Z., Fournier-Viger P., Ye L., Min F. (2024), "Efficient high utility itemset mining without the join operation", *Information Sciences* 681, pp. 121218.
86. Yao H., Hamilton H.J., Butz C.J. (2004), "A Foundational Approach to Mining Itemset Utilities from Databases", *Proceedings of the 2004 SIAM International Conference on Data Mining*, Society for Industrial and Applied Mathematics, pp. 482–486.
87. Yildirim I. (2025), "An efficient algorithm for fast discovery of high-efficiency patterns", *Knowledge-Based Systems* 313, pp. 113157.
88. Yu J.X., Chong Z., Lu H., Zhang Z., Zhou A. (2006), "A false negative approach to mining frequent itemsets from high speed transactional data streams", *Information Sciences* 176(14), pp. 1986–2015.
89. Yun U., Kim D., Yoon E., Fujita H. (2018), "Damped window based high average utility pattern mining over data streams", *Knowledge-Based Systems* 144, pp. 188–205.
90. Yun U., Lee G., Yoon E. (2017), "Efficient High Utility Pattern Mining for Establishing Manufacturing Plans With Sliding Window Control", *IEEE Transactions on Industrial Electronics* 64(9), pp. 7239–7249.
91. Yun U., Lee G., Yoon E. (2019), "Advanced approach of sliding window based

- erasable pattern mining with list structure of industrial fields", *Information Sciences* 494, pp. 37–59.
92. Zaki M.J., Hsiao C.-J. (2002), "CHARM: An efficient algorithm for closed itemset mining", *Proceedings of the 2002 SIAM international conference on data mining*, SIAM, pp. 457–473.
 93. Zaki M.J. (2000), "Scalable algorithms for association mining", *IEEE Transactions on Knowledge and Data Engineering* 12(3), pp. 372–390.
 94. Zhang C., Du Z., Gan W., Yu P.S. (2021), "TKUS: Mining top-k high utility sequential patterns", *Information Sciences* 570, pp. 342–359.
 95. Zhi-Jun X., Hong C., Li C. (2006), "An efficient algorithm for frequent itemset mining on data streams", *Industrial Conference on Data Mining*, Springer, pp. 474–491.
 96. Zida S., Fournier-Viger P., Lin J.C.-W., Wu C.-W., Tseng V.S. (2017), "EFIM: a fast and memory efficient algorithm for high-utility itemset mining", *Knowledge and Information Systems* 51(2), pp. 595–625.
 97. Zihayat M., Chen Y., An A. (2017), "Memory-adaptive high utility sequential pattern mining over data streams", *Machine Learning* 106(6), pp. 799–836.