TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

NGHIÊM TIẾN ĐẠT - 52000025

KHAI THÁC CÁC TẬP SẢN PHẨM CHIẾM TỶ LỆ CAO

DỰ ÁN CÔNG NGHỆ THÔNG TIN

KỸ THUẬT PHẦN MỀM

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

TỔNG LIÊN ĐOÀN LAO ĐỘNG VIỆT NAM
TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG
KHOA CÔNG NGHỆ THÔNG TIN

NGHIÊM TIẾN ĐẠT - 52000025

KHAI THÁC CÁC TẬP SẢN PHẨM CHIẾM TỶ LỆ CAO

DỰ ÁN CÔNG NGHỆ THÔNG TIN

KỸ THUẬT PHẦN MỀM

Người hướng dẫn
ThS. Doãn Xuân Thanh

THÀNH PHỐ HỒ CHÍ MINH, NĂM 2024

LỜI CẢM ƠN

Đầu tiên, em xin gửi lời cảm ơn chân thành nhất đến thầy Doãn Xuân Thanh. Trong quá trình hoàn thành dự án công nghệ thông tin, em đã nhận được sự giúp đỡ, hướng dẫn rất tận tình từ thầy. Các tài liệu mà thầy cung cấp đã cho em cái nhìn tổng quan về lĩnh vực data mining nói chung và high occupancy itemset mining nói riêng. Ngoài ra, em xin cảm ơn khoa Công nghệ thông tin, trường Đại học Tôn Đức Thắng vì đã tạo điều kiện cho em được thực hiện đề tài "khai thác các tập sản phẩm chiếm tỷ lệ cao". Đây là một đề tài mới mẻ và có tính ứng dụng cao trong thực tiễn.

Mặc dù đã có nền tảng kiến thức cơ bản từ các môn học trước, nhưng sự tiếp thu của mỗi người luôn tồn tại những hạn chế nhất định. Do đó, trong quá trình hoàn thành dự án công nghệ thông tin chắc chắn không thể tránh khỏi những thiếu sót. Em rất mong nhận được những lời góp ý từ quý thầy cô để dự án được hoàn thiện hơn.

Cuối cùng, em xin kính chúc quý thầy cô dồi dào sức khỏe, luôn gặt hái được nhiều thành công trong công việc cũng như trong cuộc sống. Một lần nữa, em xin chân thành cảm ơn!

TP. Hồ Chí Minh, ngày ... tháng … năm 20..

Tác giả

(Ký tên và ghi rõ họ tên)

CÔNG TRÌNH ĐƯỢC HOÀN THÀNH
TẠI TRƯỜNG ĐẠI HỌC TÔN ĐỨC THẮNG

Tôi xin cam đoan đây là công trình nghiên cứu của riêng tôi và được sự hướng dẫn khoa học của ThS. Doãn Xuân Thanh. Các nội dung nghiên cứu, kết quả trong đề tài này là trung thực và chưa công bố dưới bất kỳ hình thức nào trước đây. Những số liệu trong các bảng biểu phục vụ cho việc phân tích, nhận xét, đánh giá được chính tác giả thu thập từ các nguồn khác nhau có ghi rõ trong phần tài liệu tham khảo.

Ngoài ra, trong dự án còn sử dụng một số nhận xét, đánh giá cũng như số liệu của các tác giả khác, cơ quan tổ chức khác đều có trích dẫn và chú thích nguồn gốc.

Nếu phát hiện có bất kỳ sự gian lận nào tôi xin hoàn toàn chịu trách nhiệm về nội dung dự án của mình. Trường Đại học Tôn Đức Thắng không liên quan đến những vi phạm tác quyền, bản quyền do tôi gây ra trong quá trình thực hiện (nếu có).

TP. Hồ Chí Minh, ngày ... tháng ... năm 20..

Tác giả

(Ký tên và ghi rõ họ tên)

KHAI THÁC CÁC TẬP SẢN PHẨM CHIẾM TỶ LỆ CAO

TÓM TẮT

Trong thời đại ngày nay, khai thác dữ liệu (data mining) được ứng rộng rãi vào nhiều lĩnh vực khác nhau của đời sống xã hội. Ví dụ, đối với các siêu thị và cửa hàng tạp hóa, người ta có thể phân tích thói quen tiêu dùng của nhiều nhóm khách hàng khác nhau dựa vào dữ liệu mua hàng của họ. Từ đó, có thể đưa ra quyết định nhập xuất, sắp xếp sản phẩm sao cho tối đa hóa lợi nhuận thu được. Tương tự, nó cũng được ứng dụng trong lĩnh vực tài chính để phân tích xu hướng tăng giảm của giá vàng, ngoại tệ, cổ phiếu, v.v. Có thể nói rằng, sự bùng nổ dữ liệu chính là khởi nguồn của cuộc cách mạng công nghệ thông tin, và data mining là chìa khóa để hiện thực hóa điều đó.

Trong data mining lại chia thành nhiều đề tài khác nhau như khai thác tập phổ biến (frequent itemset mining), khai thác luật kết hợp (association rule mining), khai thác đồ thị (graph mining), v.v. Tất cả đều phục vụ cho những bài toán và mục đích khác nhau. Năm 2017, Deng đã đề xuất một phương pháp mới để khai thác hiệu quả hơn các cơ sở dữ liệu giao dịch (transaction database), gọi là khai thác các tập chiếm tỉ lệ cao (high occupancy itemset mining). Sau đó, năm 2022, Nguyen và các cộng sự đã nghiên cứu và tìm ra thuật toán tối ưu để thực hiện điều đó. Nhận thấy rằng đây là một chủ đề mới và có tính ứng dụng thực tiễn nên em đã quyết định chọn “khai thác các tập sản phẩm chiếm tỷ lệ cao” làm đề tài cho dự án công nghệ thông tin của mình, dưới sự hướng dẫn của Th.S Doãn Xuân Thanh.

HIGH OCCUPANCY ITEMSET MINING

ABSTRACT

Nowadays, data mining is widely applied in many different areas of social life. For example, in supermarkets and grocery stores, one can analyze the comsumption habits of different customer groups based on their purchasing data. From there, they can make decisions about importing and exporting goods, and arranging products to maximize profit. Similarly, it is also applied in the financial sector to analyze the rising and falling trends of gold prices, foreign currencies, stocks, etc. It can be said that the data explosion is the origin of the information techonology revolution, and data mining is the key to doing that.

Data mining is divided into many different topics such as frequent itemset mining, association rule mining, graph mining, etc. They all serve different problems and purposes. In 2017, Deng proposed a new method to more effectively exploit transaction database, called high occupancy itemset mining. Then, in 2022, Nguyen and her colleagues researched and found the optimal algorithm to do that. Realizing that it is a new subject with practical application, I decided to select “high occupancy itemset mining” as the topic for my information techonology project, with the guidance of M.Sc Doan Xuan Thanh.

MỤC LỤC

DANH MỤC HÌNH VẼ

DANH MỤC BẢNG BIỂU

DANH MỤC CÁC CHỮ VIẾT TẮT

ATHOI	auto-thresholding for high occupancy itemset mining
DFHOI	depth first search for high occupancy itemset mining
DFS	Depth First Search
GB	Gigabyte
HEP	high efficient algorithm for mining high occupancy itemsets
HOI	High Occupancy Itemset
MB	Megabyte
RAM	Random Access Memory
TID	Transaction Identification

CHƯƠNG 1. MỞ ĐẦU

Ngày nay, khai thác dữ liệu (data mining) được áp dụng rộng rãi trong nhiều lĩnh vực khác nhau như kinh tế, tài chính, kỹ thuật và khoa học. Trong đó, khai thác tập phổ biến (frequent itemset mining) là một trong những đề tài nghiên cứu giành được nhiều sự quan tâm nhất. Vì có tầm quan trọng lớn trong việc phát hiện các luật kết hợp (association rule) nên rất nhiều thuật toán đã được đề xuất để tối ưu hóa quá trình khai thác tập phổ biến. Một trong những giải thuật ra đời đầu tiên là Apriori, sử dụng cách tiếp cận brute force để phát sinh các tập sản phẩm và tính tần suất xuất hiện (support). Sau đó, thuật toán FP-Growth đã được đề xuất để khắc phục các hạn chế của Apriori như phải quét cơ sở dữ liệu (database) nhiều lần để tìm ra các tập phổ biến bằng cách sử dụng FP-tree (frequent pattern tree). Nhờ đó, việc khai thác các tập phổ biến có thể được hoàn thành chỉ với hai lần quét cơ sở dữ liệu, giúp giảm đáng kể thời gian và chi phí tính toán.

Tuy nhiên, hầu hết các thuật toán khai thác tập sản phẩm hiện nay tập trung vào tần suất xuất hiện, nghĩa là tìm ra các tập có số lần xuất hiện trong các giao dịch (transaction) lớn hơn hoặc bằng một ngưỡng cho trước. Năm 2012, Tang đã phân tích các yêu cầu thực tiễn và đề xuất một phương pháp đo lường mới được gọi là average occupancy – tỷ lệ phần trăm trung bình của tập sản phẩm trong các giao dịch chứa tập đó. Dựa vào support và average occupancy, một tập sản phẩm đủ điều kiện phải thõa mãn yêu cầu sau: đặt ngưỡng support tối thiểu là α và ngưỡng occupancy tối thiểu là β, nếu giá trị support của tập A lớn hơn hoặc bằng α và giá trị average occupancy lớn hơn hoặc bằng β thì tập A đủ điều kiện.

Một lần nữa, rất khó để xác định ngưỡng α và β bởi vì giá trị không phù hợp có thể dẫn đến quá nhiều tập đủ điều kiện hoặc không có tập nào. Do đó, thước đo mới phải có những đặc điểm sau:

1. Nếu giá trị thước đo được đề xuất của tập sản phẩm cao thì tập đó phải có giá trị occupancy và support cao.
2. Thước đo được đề xuất chỉ dùng một ngưỡng duy nhất.

TID	Items
T_1	a, c, d
T_2	a, b, d
T_3	b, c, d, e
T_4	a, d
T_5	c, d, e
T_6	a, b, c, d, e

Bảng 1.1 Cơ sở dữ liệu mẫu

Chắc chắn rằng nếu chỉ sử dụng support hoặc average occupancy được định nghĩa bởi Tang thì không thể đáp ứng được các đặc điểm nêu trên. Giả sử chỉ sử dụng thước đo support, giá trị support của tập $\{c\}$ là 4, chiếm hơn 60% số lượng transaction. Tuy nhiên, giá trị average occupancy của tập đó là $0.28(= 1/4 \times (1/3 + 1/4 + 1/3 + 1/5))$. Ngược lại, nếu chỉ sử dụng thước đo occupancy, giá trị average occupancy của tập $\{a, d\}$ là $0.68(= 1/4 \times (2/3 + 2/3 + 2/2 + 2/5))$ và của tập $\{a, b, c, d, e\}$ là $1(= 1/1 \times 5/5)$. Tuy nhiên, khi so với $\{a, d\}$ thì giá trị support của tập $\{a, b, c, d, e\}$ lại thấp hơn.

Sau nhiều nghiên cứu, vào năm 2017, Deng đã tìm ra một thước đo mới phù hợp là occupancy – tổng các tỷ lệ mà một itemset chiếm tất cả sản phẩm trong các transaction chứa tập đó. Ví dụ, theo định nghĩa trên thì giá trị occupancy của các tập $\{c\}, \{a, d\}, \{a, b, c, d, e\}$ lần lượt là $1.12(= 1/3 + 1/4 + 1/3 + 1/5), 2.73(= 2/3 + 2/3 + 2/2 + 2/5), 1(= 5/5))$. Đây là một phương pháp đo lường thích hợp vì tập $\{a, d\}$ có giá trị occupancy cũng như support cao hơn $\{c\}$ và $\{a, b, c, d, e\}$.

Phần còn lại của bài báo cáo được tổ chức như sau. Chương 2 trình bày về những định nghĩa và công thức liên quan. Thuật toán HEP và ATHOI sẽ được giới thiệu tại chương 3. Chương 4 so sánh hiệu năng của hai thuật toán trên các tập dữ liệu mẫu. Những hạn chế của thuật toán và phương pháp giải quyết được thảo luận ở chương 5. Chương 6 giới thiệu về một ứng dụng web để khai thác các HOI. Cuối cùng, phần kết luận và phương hướng phát triển dự án được thể hiện trong chương 7.

CHƯƠNG 2. KHÁI NIỆM

Trong chương này, chúng ta sẽ tiếp cận một số định nghĩa, định lý và tìm hiểu bản chất của bài toán khai thác các tập chiếm tỷ lệ cao trong cơ sở dữ liệu giao dịch. Để ngắn gọn và dễ hiểu, chúng ta sẽ tập trung vào các ví dụ minh họa. Có thể xem thêm phần chứng minh định lý trong tài liệu tham khảo (Deng, 2017) và (Nguyen, Mai, Pham, Yun, & Vo, 2022).

2.1 Định nghĩa

Gọi $I = \{i_1, i_2, \dots, i_m\}$ là tập các sản phẩm riêng biệt, và cơ sở dữ liệu giao dịch $DB = \{T_1, T_2, \dots, T_n\}$ với mỗi $T_k (1 \leq k \leq n)$ là một transaction chứa tập các sản phẩm có trong I. Một tập sản phẩm khác rỗng bất kì được gọi là một itemset và itemset chứa k sản phẩm được gọi là k-itemset. Cho itemset P, transaction T được xem là chứa P khi và chỉ khi P là tập con của $T (P \subseteq T)$. Từ đó, ta có một số định nghĩa cơ bản được trình bày dưới đây.

Định nghĩa 1. Support của itemset P, kí hiệu $Sup(P)$, là số lượng transaction chứa P. Ví dụ, itemset $\{a\}$ xuất hiện trong bốn transaction T_1, T_2, T_4, T_6 nên $Sup(\{a\}) = 4$. Tương tự, các itemset $\{a, b\}, \{a, c\}, \{a, d\}, \{a, e\}$ có giá trị support là $2, 2, 4, 1$ theo thứ tự.

Định nghĩa 2. Support transaction set của itemset P, kí hiệu $STSet(P)$, là tập các transaction chứa P. Ví dụ, ta có support transaction set của các itemset $\{a\}, \{a, b\}, \{a, c\}, \{a, d\}, \{a, e\}$ là:

$STSet(\{a\}) = \{T_1, T_2, T_4, T_6\}$

$STSet(\{a, b\}) = \{T_2, T_6\}$

$STSet(\{a, c\}) = \{T_1, T_6\}$

$STSet(\{a, d\}) = \{T_1, T_2, T_4, T_6\}$

$STSet(\{a, e\}) = \{T_6\}$

Định nghĩa 3. Occupancy của itemset P, kí hiệu $O(P)$, được tính như sau:

$$O(P) = \sum_{T \in STSet(P)} \frac{|P|}{|T|}$$

Ví dụ, giá trị occupancy của các itemset $\{a\}, \{a,b\}, \{a,c\}, \{a,d\}, \{a,e\}$ là:

$$O(\{a\}) = \sum_{T \in \{T_1,T_2,T_4,T_6\}} \frac{|\{a\}|}{|T|} = \frac{1}{3} + \frac{1}{3} + \frac{1}{2} + \frac{1}{5} \approx 1.37$$

$$O(\{a,b\}) = \sum_{T \in \{T_2,T_6\}} \frac{|\{a,b\}|}{|T|} = \frac{2}{3} + \frac{2}{5} \approx 1.07$$

$$O(\{a,c\}) = \sum_{T \in \{T_1,T_6\}} \frac{|\{a,c\}|}{|T|} = \frac{2}{3} + \frac{2}{5} \approx 1.07$$

$$O(\{a,d\}) = \sum_{T \in \{T_1,T_2,T_4,T_6\}} \frac{|\{a,d\}|}{|T|} = \frac{2}{3} + \frac{2}{3} + \frac{2}{2} + \frac{2}{5} \approx 2.73$$

$$O(\{a,e\}) = \sum_{T \in \{T_6\}} \frac{|\{a,e\}|}{|T|} = \frac{2}{5} = 0.4$$

Định nghĩa 4. Cho ngưỡng occupancy tối thiểu ξ do người dùng đặt ra, itemset P được gọi là high occupancy itemset (HOI) nếu $O(P) \geq \xi$.

Định nghĩa 5. Occupancy-list của itemset P, kí hiệu $OL(P)$, là một danh sách mà mỗi phần tử chứa hai trường gồm:

1. tid là mã định danh của transaction T chứa P.
2. tsize là độ dài của T.

Dựa vào Bảng 1.1, ta có thể xây dựng occupancy-list của các 1-itemset. Ví dụ, itemset $\{a\}$ có trong các transaction T_1, T_2, T_4, T_6 và độ dài các transaction lần lượt là $3, 3, 2, 5$ nên ta có $OL(\{a\}) = \{(T_1, 3), (T_2, 3), (T_4, 2), (T_6, 5)\}$. Ta có thể thực hiện tương tự cho các 1-itemset khác. Bên dưới là occupancy-list của 1-itemset.

$OL(\{a\}) = \{(T_1, 3), (T_2, 3), (T_4, 2), (T_6, 5)\}$

$OL(\{b\}) = \{(T_2, 3), (T_4, 2), (T_6, 5)\}$

$OL(\{c\}) = \{(T_1, 3), (T_3, 4), (T_5, 3), (T_6, 5)\}$

$OL(\{d\}) = \{(T_1, 3), (T_2, 3), (T_3, 4), (T_4, 2), (T_5, 3), (T_6, 5)\}$

$OL(\{e\}) = \{(T_3, 4), (T_5, 3), (T_6, 5)\}$

Sau khi có được occupancy-list của 1-itemset, occupancy-list của 2-itemset có thể được xác định bằng cách lấy phần giao hai occupancy-list của 1-itemset tương ứng mà không cần quét lại cơ sở dữ liệu. Ví dụ, bằng cách giao $OL(\{a\})$ và $OL(\{e\})$, ta được $OL(\{a,e\}) = \{(T_6, 5)\}$. Ta có thể thực hiện tương tự cho các k-itemset. Bên dưới là occupancy-list của 2-itemset chứa sản phẩm a:

$OL(\{a,b\}) = \{(T_2,3),(T_6,5)\}$

$OL(\{a,c\}) = \{(T_1,3),(T_6,5)\}$

$OL(\{a,d\}) = \{(T_1,3),(T_2,3),(T_4,2),(T_6,5)\}$

$OL(\{d,e\}) = \{(T_6,5)\}$

Định nghĩa 6. Giả sử các transaction chứa itemset P có u độ dài khác nhau. Ngoài ra, gọi $n_x(1 \le x \le u)$ là số lượng transaction có cùng độ dài l_x, và $\sum_{i=x}^{u} n_i \times \frac{l_x}{l_i}$ được kí hiệu là UBO_p^x. Giá trị cận trên occupancy (upper-bound occupancy) của P, kí hiệu $UBO(P)$, được tính như sau:

$$UBO(P) = max_{1 \le x \le u} UBO_p^x$$

Ví dụ, giá trị cận trên occupancy của itemset $\{a\}$ được tính như sau:

Ta có: $OL(\{a\}) = \{(T_1,3),(T_2,3),(T_4,2),(T_6,5)\}$

- $x = 1$: $l_1 = 2,\ n_1 = |\{T_4\}| = 1$
- $x = 2$: $l_2 = 3,\ n_2 = |\{T_1, T_2\}| = 2$
- $x = 3$: $l_3 = 5,\ n_3 = |\{T_6\}| = 1$

$$UBO_{\{a\}}^1 = \sum_{i=1}^{3} n_i \times \frac{l_1}{l_i} = 1 \times \frac{2}{2} + 2 \times \frac{2}{3} + 1 \times \frac{2}{5} \approx 2.73$$

$$UBO_{\{a\}}^2 = \sum_{i=2}^{3} n_i \times \frac{l_2}{l_i} = 2 \times \frac{3}{3} + 1 \times \frac{3}{5} \approx 2.6$$

$$UBO_{\{a\}}^3 = \sum_{i=3}^{3} n_i \times \frac{l_3}{l_i} = 1 \times \frac{5}{5} = 1$$

$$\rightarrow UBO(\{a\}) = max_{1 \le x \le 3} UBO_{\{a\}}^x = 2.73$$

2.2 Định lý

Định lý 1. Với itemset P bất kì, ta luôn có $O(P) \leq Sup(P)$. Dựa vào hệ quả của định lý này, ta có thể giảm số lượng itemset cần phải kiểm tra bằng cách loại bỏ các itemset có giá trị support nhỏ hơn ngưỡng occupancy tối thiểu.

Định lý 2. Gọi P, P_1, P_2 là ba itemset và occupancy-list của chúng là $OL(P)$, $OL(P_1), OL(P_2)$ theo thứ tự. Nếu $P = P_1 \cup P_2$ thì $OL(P) = OL(P_1) \cap OL(P_2)$. Điều này cho phép ta có thể xây dựng occupancy-list của k-itemset từ occupancy-list của $(k-1)$-itemset mà không cần phải quét cơ sở dữ liệu nhiều lần.

Định lý 3. Với mọi itemset P, ta có $UBO(P) \leq Sup(P)$.

Định lý 4. Với bất kì superset của itemset P, kí hiệu P', ta có $O(P') \leq UBO(P)$. Dựa vào hệ quả của định lý, ta có thể giảm đáng kể phạm vi không gian tìm kiếm vì itemset có giá trị cận trên occupancy nhỏ hơn ngưỡng tối thiểu mà người dùng cung cấp thì các superset của nó không phải là tập chiếm tỷ lệ cao.

2.3 Bài toán

Cho cơ sở dữ liệu giao dịch DB và ngưỡng occupancy tối thiểu ξ do người dùng tự xác định, bài toán khai thác tập chiếm tỷ lệ cao là tìm ra tất cả các high occupancy itemset HOI trong DB có giá trị occupancy lớn hơn hoặc bằng ξ.

$$HOI = \{P \in DB | O(P) \geq \xi\}$$

CHƯƠNG 3. THUẬT TOÁN

3.1 Thuật toán HEP

Thuật toán HEP (high efficient algorithm for mining high occupancy itemsets) do Zhi-Hong Deng đề xuất lần đầu tiên vào năm 2017. Thuật toán sử dụng cách tiếp cận lặp lại theo cấp độ để tìm kiếm tất cả các tập sản phẩm chiếm tỷ lệ cao trong cơ sở dữ liệu, trong đó $(k-1)$-itemset được sử dụng để khai phá k-itemset. Bên cạnh đó, nhằm loại bỏ sớm các tập không đủ điều kiện và thu hẹp không gian tìm kiếm, Deng đã áp dụng giá trị cận trên occupancy. Mã giả của thuật toán HEP (đã điều chỉnh) được trình bày trong Bảng 3.1.

Dòng 1 của thuật toán HEP quét qua toàn bộ cơ sở dữ liệu để xây dựng occupancy-list của các 1-itemset. Từ dòng 3 đến dòng 9, tập các ứng viên 1-itemset C_1 và tập các 1-itemset chiếm tỷ lệ cao HOI_1 được tạo ra. Lưu ý rằng thuật toán HEP sử dụng giá trị support của 1-itemset để lọc ra các tập không đủ điều kiện tại dòng 5. Vì các tập 1-itemset không đủ điều kiện không phải là frequent itemset nên chúng và các superset của chúng không thể là high occupancy itemset theo tính chất bao đóng và hệ quả của Định lý 1. Tiếp theo, từ dòng 9 đến dòng 22, C_{k-1} được sử dụng để phát sinh ra C_k và sau đó C_k được sử dụng để tạo ra HOI_k, với C_k là tập hợp các ứng viên gồm k phần tử và HOI_k là tập hợp các tập sản phẩm chiếm tỷ lệ cao gồm k phần tử. Ở dòng 15, tập k-itemset P được tạo ra bằng cách kết hợp hai tập $(k-1)$-itemset có cùng $k-2$ phần tử. Tương tự, occupancy-list của P được xây dựng bằng cách giao occupancy-list của hai tập $(k-1)$-itemset tương ứng ở dòng 17.

Tuy nhiên, trong quá trình triển khai thuật toán, em phát hiện ra rằng HEP có thể duyệt qua một itemset nhiều lần. Sự trùng lặp này dẫn đến việc lãng phí thời gian và tài nguyên tính toán.Ví dụ, dựa vào cơ sở dữ liệu mẫu trong Bảng 1.1, từ các 1-itemset $\{a\}, \{b\}, \{c\}$ ta có thể phát sinh các 2-itemset như sau:

$P_1 \leftarrow \{a\}, P_2 \leftarrow \{b\}, \{c\}: \{a,b\}, \{a,c\}$

$P_1 \leftarrow \{b\}, P_2 \leftarrow \{a\}, \{c\}: \{b,a\}, \{b,c\}$

$P_1 \leftarrow \{c\}, P_2 \leftarrow \{a\}, \{b\}: \{c,a\}, \{c,b\}$

Bảng 3.1 Thuật toán HEP

Input: transaction database DB and minimum occupancy threshold ξ

Output: high occupancy itemsets $HOIs$

```
1.  Scan DB to obtain occupancy-list of each 1-itemset;
2.  HOIs ← ∅;
3.  C_1 ← ∅; HOI_1 ← ∅;
4.  for each 1-itemset P do
5.      if Sup(P) ≥ ξ then
6.          if UBO(P) ≥ ξ then
7.              C_1 ← C_1 ∪ {P};
8.              if O(P) ≥ ξ then
9.                  HOI_1 ← HOI_1 ∪ {P};
10. for (k = 2; C_{k-1} ≠ ∅; k++) do
11.     C_k ← ∅;
12.     for each itemset P_1 ∈ C_{k-1} do
13.         for each itemset P_2 ∈ C_{k-1} do
14.             if |P_1 ∩ P_2| = k − 2 then
15.                 P = P_1 ∪ P_2;
16.                 if P is checked then continue;
17.                 OL(P) ← OL(P_1) ∪ OL(P_2);
18.                 Scan OL(P) to calculate UBO(P);
19.                 if UBO(P) ≥ ξ then
20.                     C_k ← C_k ∪ {P};
21.     HOI_k ← {P|P ∈ C_k ∧ O(P) ≥ ξ};
22.     HOIs ← HOIs ∪ HOI_k;
23. return HOIs;
```

Có thể thấy rằng các itemset không quan trọng về mặt thứ tự, do đó $\{a.b\} = \{b,a\}, \{a,c\} = \{c,a\}$ và $\{b,c\} = \{c,d\}$. Để giải quyết vấn đề trùng lặp, chúng ta cần phải kiểm tra các itemset đã được duyệt qua hay chưa (dòng 16) trước khi thực hiện các bước tiếp theo.

Nhằm khai thác các HOI thuận lợi hơn, Deng đã triển khai thuật toán HEP bằng cách sử dụng hai kỹ thuật sau đây. Đầu tiên, database DB được sắp xếp theo thứ tự tăng dần độ dài của các transaction và occupancy-list của các 1-itemset được tạo ra bằng cách quét database đã được sắp xếp. Vì các phần tử trong occupancy-list của 1-itemset được sắp xếp theo thứ tự tăng dần độ dài nên các phần tử trong occupancy-list của k-itemset ($k \geq 2$) cũng được sắp xếp theo thứ tự đó. Điều này cho phép tính toán UBO_p^x nhanh chóng trong quá trình quét occupancy-list. Do đó, khi một UBO_p^x không nhỏ hơn ξ, ta có thể chắc chắn rằng $UBO(P)$ lớn hơn hoặc bằng ξ mà không cần quét qua phần còn lại trong occupancy-list, giúp giảm thời gian xác định $UBO(P) \geq \xi$ tại dòng 6 và 19. Thứ hai, tất cả phần tử trong itemset được sắp xếp theo thứ tự support tăng dần, giúp tạo ra k-itemset bằng cách giao $(k-1)$-itemset.

3.2 Thuật toán DFHOI

Định nghĩa 7. Gọi $DB = \{T_1, T_2, \dots, T_n\}$ là cơ sở dữ liệu giao dịch, mảng L chứa index độ dài của các transaction trong database sao cho $L_i = |T_i|$. Mảng L được sử dụng để tiết kiệm bộ nhớ cho việc lưu trữ occupancy-list. Ví dụ, từ Bảng 1.1 ta có $L = \{3,3,4,2,3,5\}$ và occupancy-list của mỗi itemset chỉ cần lưu thông tin support transaction set của itemset đó, nghĩa là $OL(P) = STSet(P)$. Dựa vào đó, ta được occupancy-list của các 1-itemset như sau:

$OL(\{a\}) = \{T_1, T_2, T_4, T_6\}$

$OL(\{b\}) = \{T_2, T_3, T_6\}$

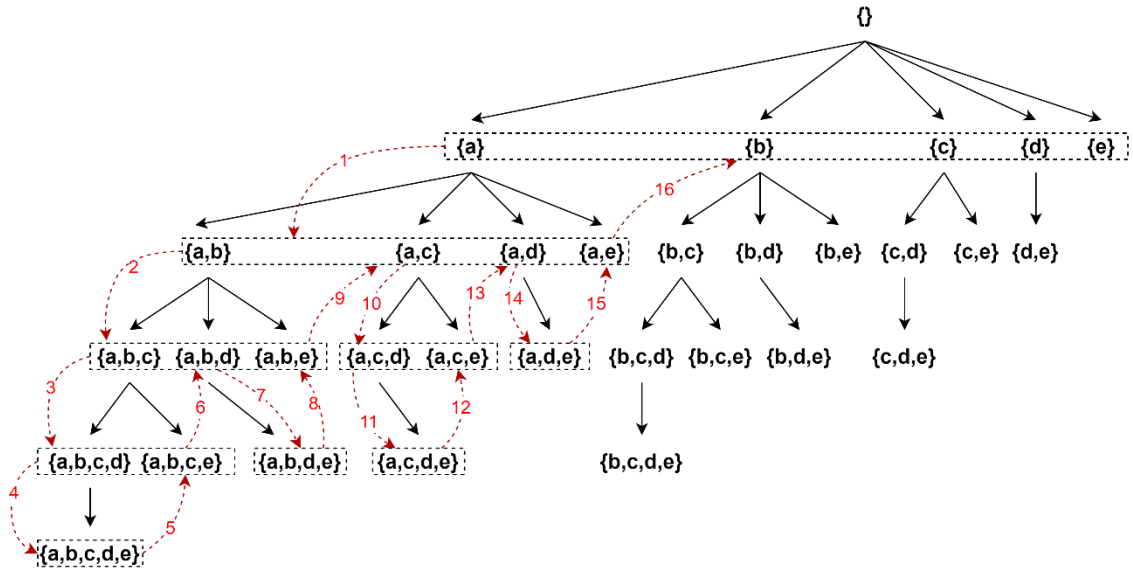
$OL(\{c\}) = \{T_1, T_3, T_5, T_6\}$

$OL(\{d\}) = \{T_1, T_2, T_3, T_4, T_5, T_6\}$

$OL(\{e\}) = \{T_3, T_5, T_6\}$

Giả sử mỗi TID có kích thước 4 byte, sử dụng occupancy-list chiếm $4 \times 8 + 3 \times 8 + 4 \times 8 + 6 \times 8 + 3 \times 8 = 160$ byte, trong khi sử dụng mảng độ dài L và support transaction set chiếm $6 \times 4 + 4 \times 4 + 3 \times 4 + 4 \times 4 + 6 \times 4 + 3 \times 4 = 104$ byte. Trong thực tế, đối với các tập dữ liệu khổng lồ hàng terabyte, dung lượng bộ nhớ được tiết kiệm là đáng kể.

Định lý 5. Nếu tất cả transaction trong database DB có cùng độ dài thì $UBO(P) = Sup(P)$. Điều này cho phép chúng ta không cần lãng phí thời gian và tài nguyên hệ thống để tính giá trị cận trên occupancy của các itemset trong trường hợp đặc biệt này.



Hình 3.1 Khám phá các itemset bằng DFS

Dựa vào các ý tưởng nêu trên, Nguyen và các đồng tác giả khác đã đề xuất một thuật toán hiệu quả để khai thác các tập chiếm tỷ lệ cao vào năm 2022, gọi là DFHOI (depth first search for high occupancy itemset mining). DFHOI sử dụng chiến lược tìm kiếm theo chiều sâu để khám phá các tập ứng viên như Hình 3.1. Sau khi duyệt qua tất cả các itemset bắt đầu bằng item a, thuật toán sẽ lần lượt kiểm tra các itemset bắt đầu bằng item b, c, d và e theo thứ tự.

Đầu tiên, thuật toán DFHOI quét qua cơ sở dữ liệu để tạo ra tập các 1-itemset S và danh sách id của các transaction chứa 1-itemset trong tập S. Thuật toán cũng lưu trữ độ dài của mỗi transaction và xác định rằng liệu tất cả transaction có cùng độ dài hay không tại dòng 1. Sau đó, DFHOI gọi phương thức *Mine_HOI_1Itemset* ở dòng 4 để khai thác các tập chiếm tỷ lệ cao bằng cách duyệt qua từng itemset P trong tập S có giá trị support lớn hơn hoặc bằng ngưỡng tối thiểu ξ. Có hai trường hợp mà phương thức này cần phải xác định:

1. Dòng 11 – 16, nếu transaction trong database có độ dài khác nhau, thuật toán sẽ tính giá trị cận trên occupancy của P. Nếu $UBO(P)$ lớn hơn hoặc bằng ξ thì P là tập chiếm tỷ lệ cao. Itemset P sẽ được thêm vào tập ứng viên cho vòng tiếp theo để khám phá các k-itemset ($k \geq 2$).
2. Dòng 17 – 20, nếu tất cả transaction trong database có cùng độ dài thì P là tập chiếm tỷ lệ cao. Itemset P cũng sẽ được thêm vào tập ứng viên cho vòng kế tiếp để khai thác các k-itemset ($k \geq 2$).

Sau đó, DFHOI gọi phương thức *Mine_Depth_HOIs* để khai thác các k-itemset bằng cách truyền vào tập ứng viên C_1. Tại phương thức này, chúng ta sẽ phát sinh ra các k-itemset bằng cách kết hợp hai $(k-1)$-itemset trong tập C với nhau, với điều kiện itemset P_2 phải nằm sau itemset P_1. Tiếp theo, chúng ta sẽ tìm support transaction set của P bằng cách giao $STSet(P_1)$ và $STSet(P_2)$ tại dòng 26. Tương tự như phương thức *Mine_HOI_1Itemset*, ta cũng kiểm tra giá trị support của itemset P để loại bỏ sớm những tập không đủ điều kiện (dòng 27) và xác định liệu tất cả transaction trong database có cùng độ dài hay không (dòng 28 – 33). Tiếp theo, tại dòng 34, thuật toán sẽ tiến hành gọi đệ quy để tiếp tục khám phá các $(k+1)$-itemset theo chiến lược DFS. Sau khi không còn ứng viên mới nào được phát hiện, ta sẽ tiến hành tính giá trị occupancy của các itemset trong tập C_k và trả về các itemset có giá trị occupancy lớn hơn hoặc bằng ξ. Cuối cùng, thuật toán sẽ lấy kết quả của phương thức này để thêm vào danh sách $HOIs$ tại dòng 7. Mã giả của DFHOI được trình bày trong Bảng 3.2 dưới đây.

Bảng 3.2 Thuật toán DFHOI

Input: transaction database DB and minimum occupancy threshold ξ

Output: high occupancy itemsets $HOIs$

DFHOI()

1. Scan DB to generate a set of 1-itemset S, with an index of length L and tidset of each 1-itemset, and determine if DB has the same length ($sameLength$);
2. $HOIs \leftarrow \emptyset$;
3. $C_1 \leftarrow \emptyset; HOI_1 \leftarrow \emptyset$;
4. $\{C_1, HOI_1\} \leftarrow Mine\_HOI\_1Itemset(S)$;
5. $HOIs \leftarrow HOIs \cup HOI_1$;
6. **if** $C_1 \neq \emptyset$ **then**
7. $\quad HOIs \leftarrow HOIs \cup Mine\_Depth\_HOIs(C_1)$;
8. **return** $HOIs$;

Mine_HOI_1Itemset($\boldsymbol{S}$)

9. **for** each 1-itemset $P \in S$ **do**
10. $\quad$ **if** $Sup(P) \geq \xi$ **then**
11. $\quad\quad$ **if** $sameLength = False$ **then**
12. $\quad\quad\quad$ Scan $STSet(P)$ to calculate $UBO(P)$ based on L;
13. $\quad\quad\quad$ **if** $UBO(P) \geq \xi$ **then**
14. $\quad\quad\quad\quad C_1 \leftarrow C_1 \cup \{P\}$;
15. $\quad\quad\quad\quad$ **if** $O(P) \geq \xi$ **then**
16. $\quad\quad\quad\quad\quad HOI_1 \leftarrow HOI_1 \cup \{P\}$;
17. $\quad\quad$ **else**
18. $\quad\quad\quad C_1 \leftarrow C_1 \cup \{P\}$;
19. $\quad\quad\quad$ **if** $O(P) \geq \xi$ **then**
20. $\quad\quad\quad\quad HOI_1 \leftarrow HOI_1 \cup \{P\}$;
21 **return** $\{C_1, HOI_1\}$;

***Mine_Depth_HOIs*(C)**

22. **for** each itemset $P_1 \in C$ **do**
23. $C_k \leftarrow \emptyset$;
24. **for** each itemset $P_2 \in C$ with P_2 following P_1 **do**
25. $P \leftarrow P_1 \cup P_2$;
26. $STSet(P) \leftarrow STSet(P_1) \cap STSet(P_2)$;
27. **if** $Sup(P) \geq \xi$ **then**
28. **if** $sameLength = False$ **then**
29. Scan $STSet(P)$ to calculate $UBO(P)$ based on L;
30. **if** $UBO(P) \geq \xi$ **then**
31. $C_k \leftarrow C_k \cup \{P\}$;
32. **else**
33. $C_k \leftarrow C_k \cup \{P\}$;
34. **if** $C_k \neq \emptyset$ **then**
35. $Mine\_Depth\_HOIs(C_k)$;
36. $HOIs \leftarrow HOIs \cup \{P | P \in C_k \wedge O(P) \geq \xi\}$;

CHƯƠNG 4. THỰC NGHIỆM

Trong chương này, chúng ta sẽ đánh giá hiệu năng của thuật toán HEP và DFHOI trên các tập dữ liệu mẫu. Cả hai thuật toán đã được triển khai bằng Python và Java, chạy trên máy tính Intel Core-i5 1.00 GHz, 4 Core, 8 Processor, 8GB RAM, hệ điều hành Windows 10. Thuật toán HEP và DFHOI được thử nghiệm trên các tập dữ liệu với nhiều ngưỡng occupancy tối thiểu khác nhau (tỉ lệ phần trăm ứng với kích thước của tập dữ liệu). Sau đó, thời gian thực thi và dung lượng bộ nhớ sử dụng sẽ được ghi lại nhằm mục đích so sánh mức độ hiệu quả của hai thuật toán.

4.1 Tập dữ liệu

Có 6 tập dữ liệu mẫu được sử dụng để đánh giá hiệu suất của thuật toán. Các tập dữ liệu này được tổng hợp từ FIMI Repository và SPMF. Trong đó, Foodmart, Fruit Hut, Kosarak, Online Retail và Retail là các tập dữ liệu được thu thập từ thực tế, còn T10I4D100K là tập dữ liệu nhân tạo. Bảng 4.1 thể hiện thông tin chi tiết của các tập dữ liệu, bao gồm số lượng giao dịch (#Trans), số lượng sản phẩm (#Items), số lượng item trung bình trong mỗi transaction (AvgLen) và số lượng item lớn nhất trong transaction (MaxLen). Bảng 4.2 cho thấy số lượng high occupancy itemset (#HOI) tương ứng với ngưỡng occupancy tối thiểu (ξ) của các tập dữ liệu.

Bảng 4.1 Thông tin các tập dữ liệu mẫu

Dataset	#Trans	#Items	AvgLen	MaxLen
Foodmart	4,141	1,559	4.42	14
Fruit Hut	181,970	1,265	3.58	36
Kosarak	990,002	41,270	8.1	2,498
Online Retail	541,909	2,603	4.37	8
Retail	88,162	16,470	10.3	76
T10I4D100K	100,000	870	10.1	29

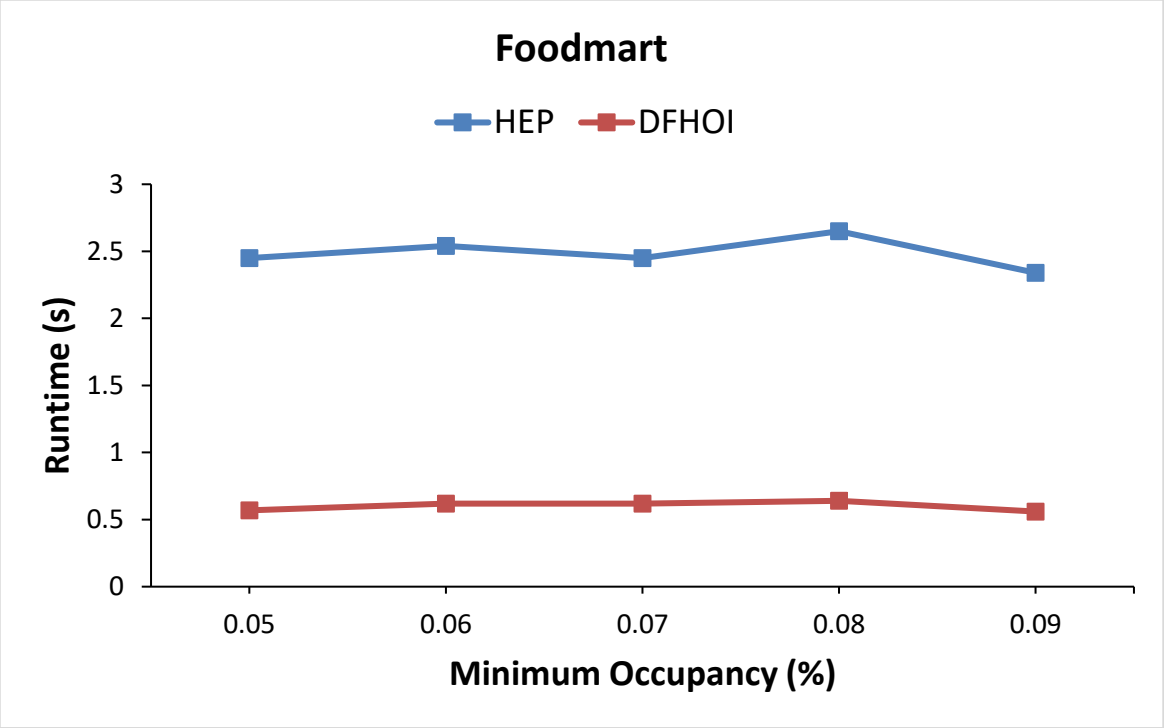
Bảng 4.2 Số lượng HOI của các tập dữ liệu

Foodmart	$\xi(\%)$	0.05	0.06	0.07	0.08	0.09
	#HOI	1,117	838	568	337	185
Fruit Hut	$\xi(\%)$	0.03	0.04	0.05	0.06	0.07
	#HOI	2,476	1,664	1,203	925	730
Kosarak	$\xi(\%)$	0.6	0.7	0.8	0.9	1
	#HOI	71	55	48	36	31
Online Retail	$\xi(\%)$	0.1	0.2	0.3	0.4	0.5
	#HOI	2,004	464	215	112	72
Retail	$\xi(\%)$	0.06	0.07	0.08	0.09	0.1
	#HOI	870	692	561	456	395
T10I4D100K	$\xi(\%)$	0.1	0.2	0.3	0.4	0.5
	#HOI	4,363	259	65	26	10

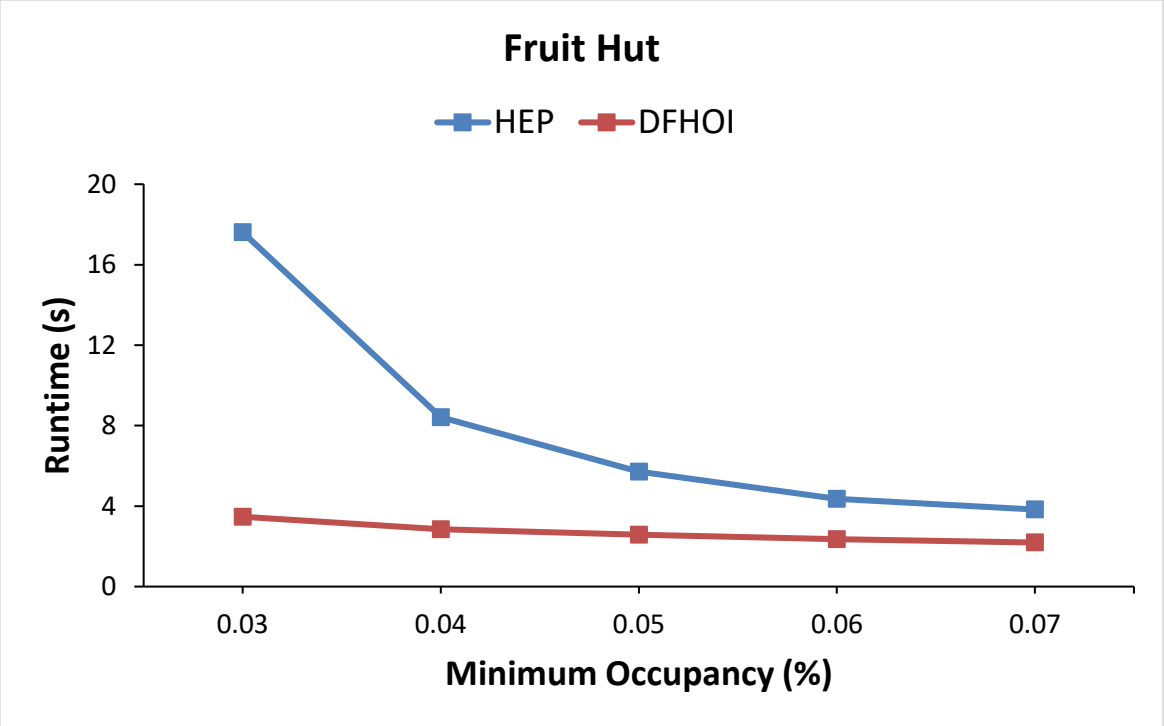
4.2 Thời gian thực thi

Hình 4.1 – Hình 4.6 thể hiện thời gian thực thi (runtime) của HEP và DFHOI viết bằng Python trên sáu tập dữ liệu mẫu. Runtime được tính từ lúc đọc file dữ liệu đầu vào cho đến khi khám phá toàn bộ tập sản phẩm đủ điều kiện. Để đánh giá thời gian chạy, chúng ta sẽ thay đổi ngưỡng tối thiểu cho từng tập dữ liệu. Khi ngưỡng tối thiểu giảm thì số lượng HOI tăng lên đáng kể, và do đó thời gian chạy cũng tăng theo. Ví dụ, đối với tập T10I4D100K, khi ξ thay đổi từ 0.2% thành 0.1%, số lượng HOI tăng từ 256 lên 4,363, và thời gian thực thi cũng tăng từ 12.77s lên 64.54s.

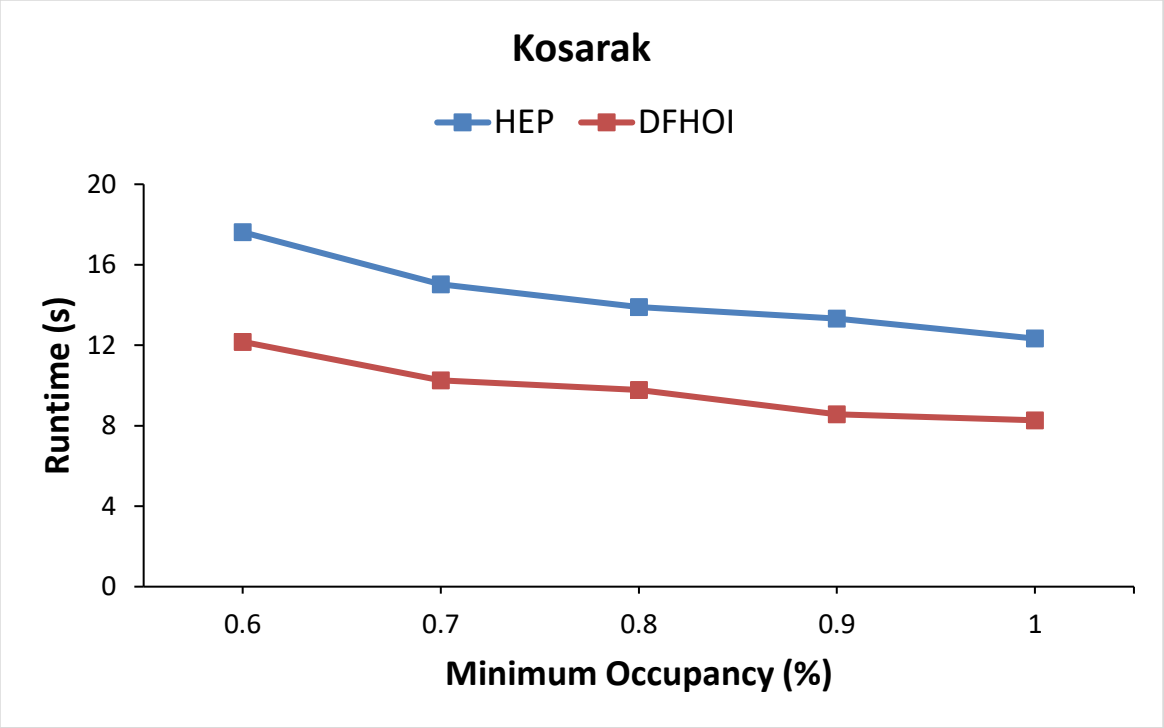
Từ Hình 4.2, chúng ta quan sát được rằng thuật toán DFHOI có khả năng scale tốt hơn thuật toán HEP. Đối với tập Fruit Hut, khi ξ là 0.07%, thời gian chạy của HEP và DFHOI lần lượt là 3.28s và 2.19s. Khi ngưỡng tối thiểu giảm xuống, cả hai thuật toán đều chạy chậm hơn, Tuy nhiên, thời gian chạy của thuật toán DFHOI tăng chậm hơn so với thuật toán HEP. Ví dụ, tại ngưỡng 0.03%, thời gian chạy của DFHOI là 3.47s và HEP là 17.61s. Bằng việc thay đổi ξ từ 0.07% thành 0.03%, thời gian chạy của HEP tăng gấp 8 lần trong khi DFHOI gần như không tăng.



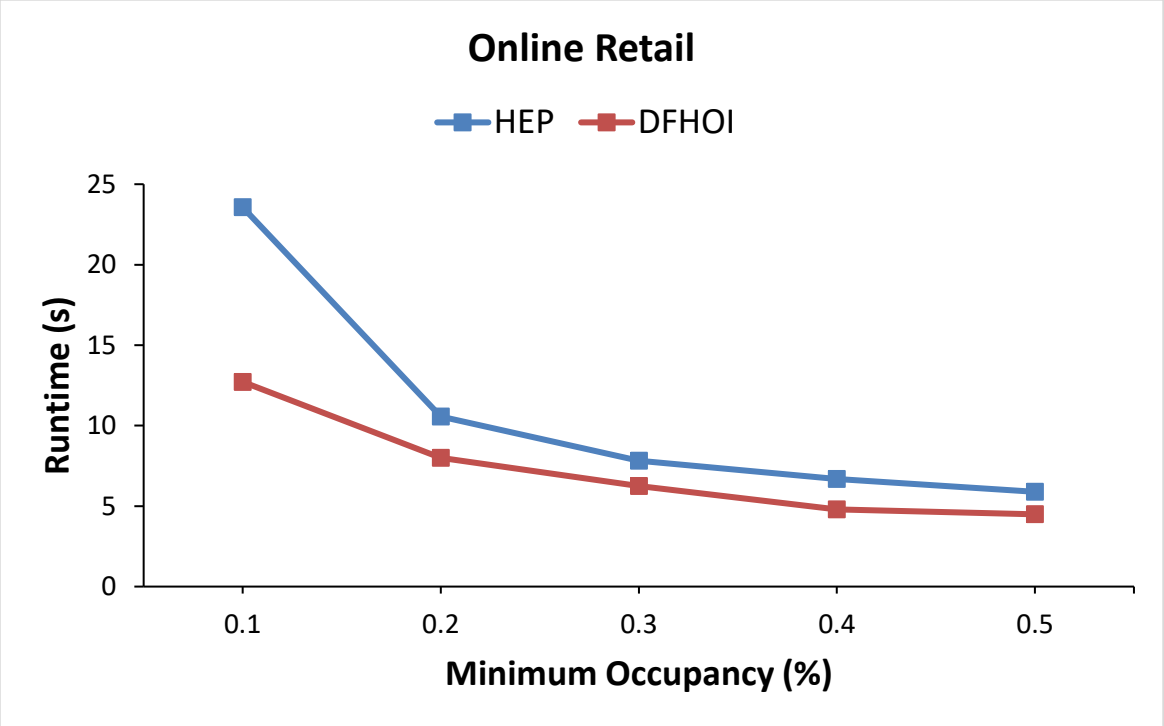
Hình 4.1 Thời gian thực thi trên tập Foodmart



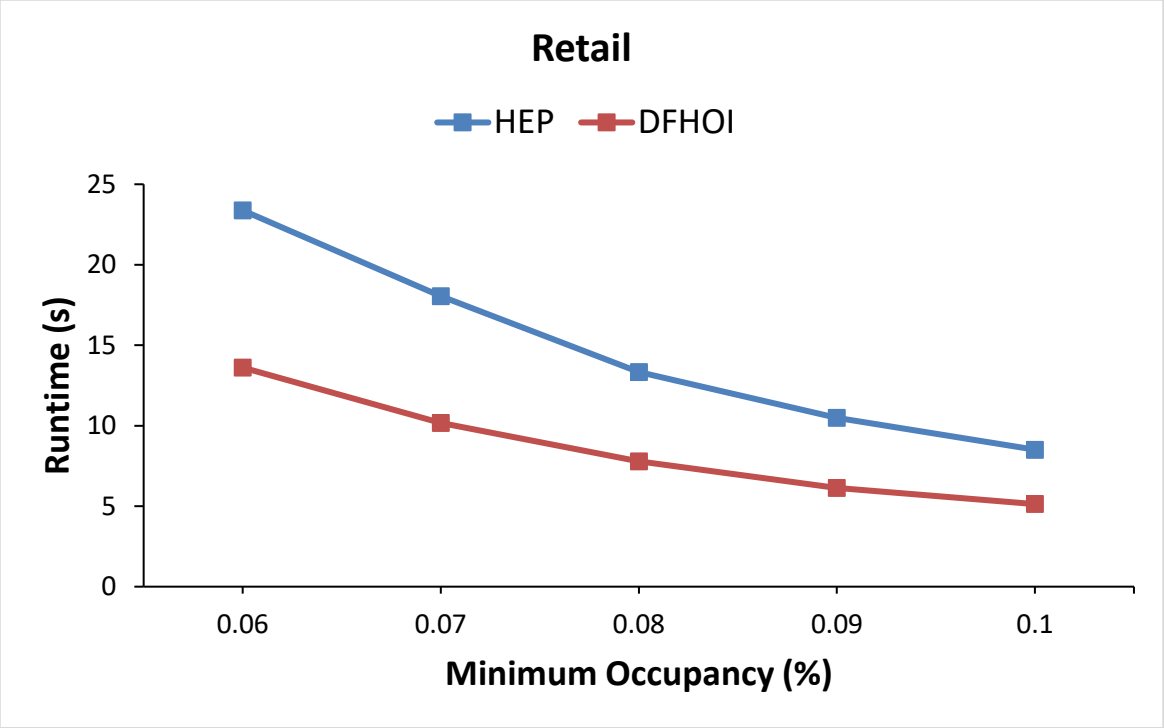
Hình 4.2 Thời gian thực thi trên tập Fruit Hut



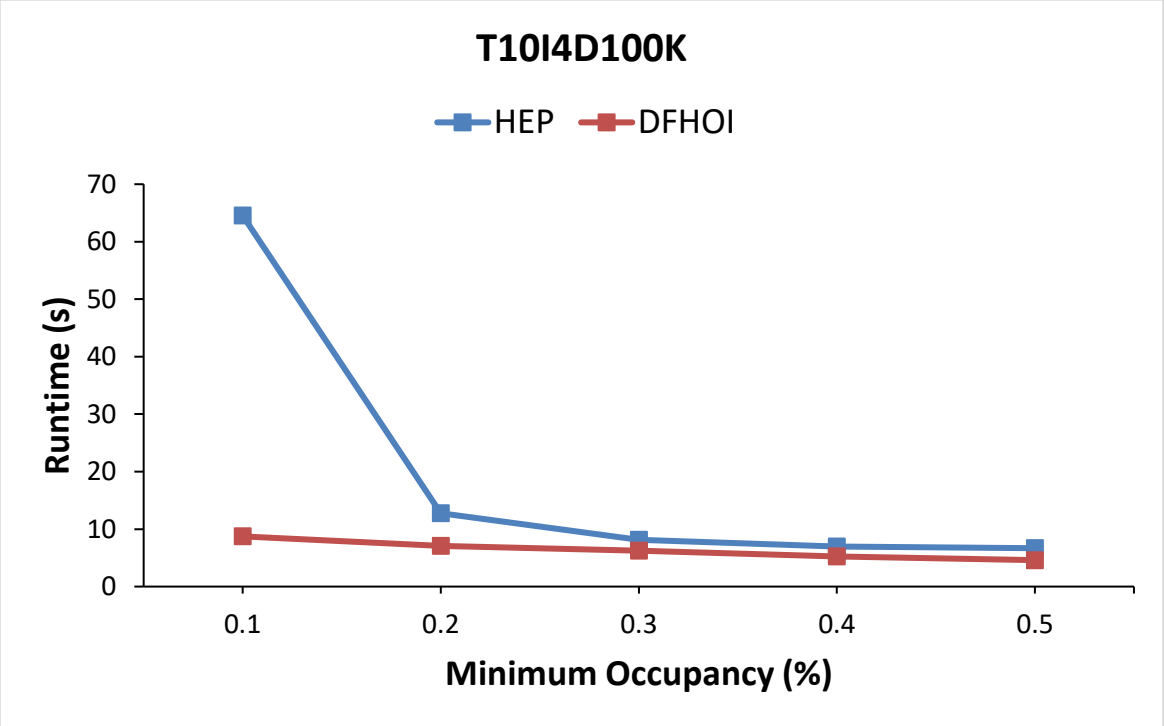
Hình 4.3 Thời gian thực thi trên tập Kosarak



Hình 4.4 Thời gian thực thi trên tập Online Retail



Hình 4.5 Thời gian thực thi trên tập Retail



Hình 4.6 Thời gian thực thi trên tập T10I4D100K

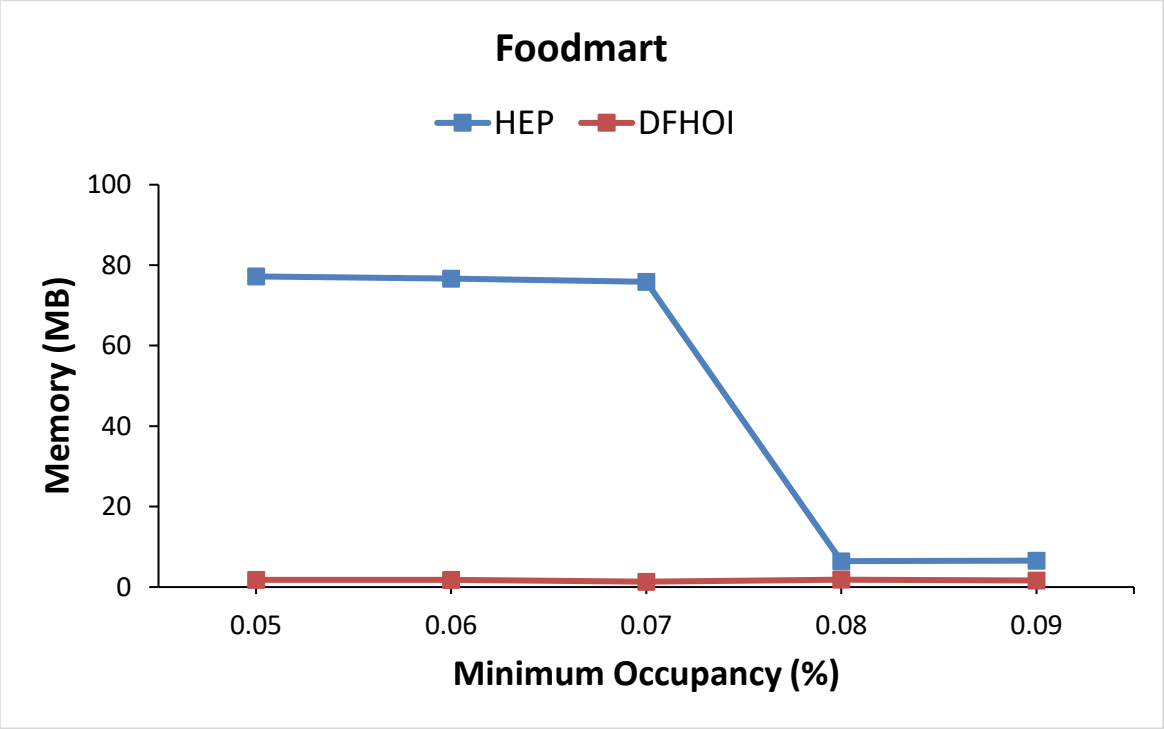
Trên năm trập dữ liệu còn lại, kết quả thực nghiệm cũng chỉ ra rằng tốc độ tăng của DFHOI chậm hơn so với HEP, cho thấy DFHOI có khả năng mở rộng tốt hơn. Điều tương tự cũng được thể hiện trong Bảng 4.3 dưới đây, khi các thuật toán được triển khai bằng Java.

Bảng 4.3 Thời gian thực thi – Java

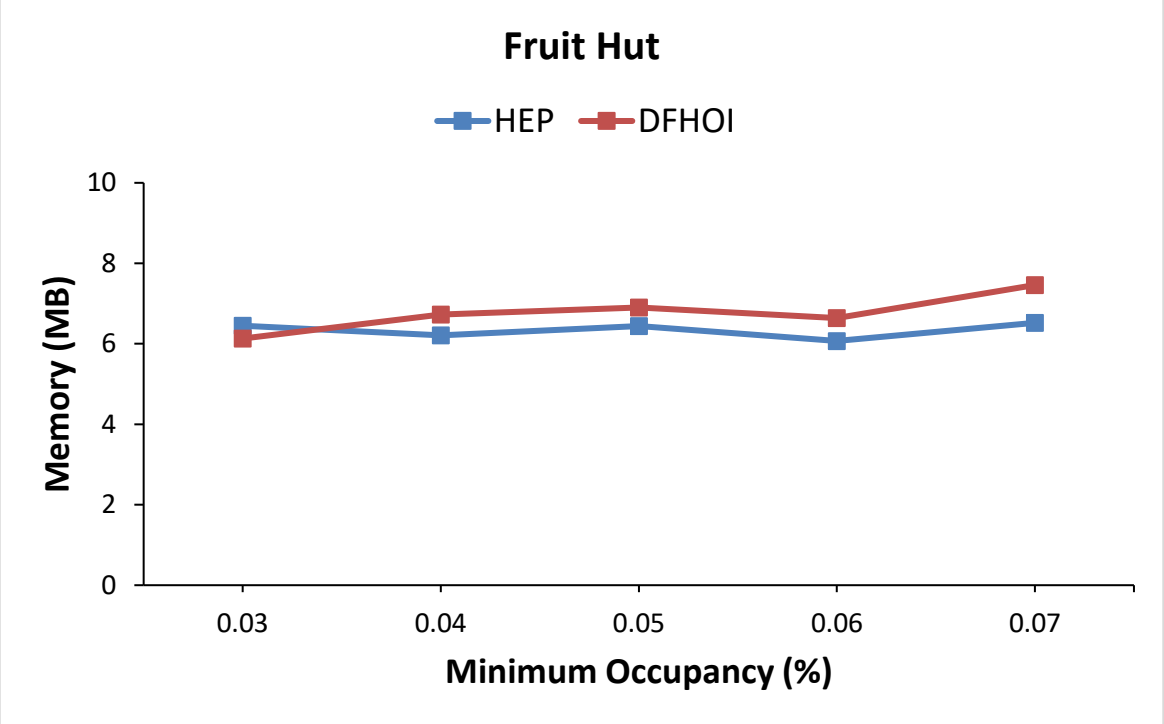
Đơn vị: giây (s)

Foodmart	ξ(%)	0.05	0.06	0.07	0.08	0.09
	HEP	34.42	29.58	29.09	29.61	28.55
	DFHOI	0.93	0.88	0.82	0.84	0.84
Fruit Hut	ξ(%)	0.03	0.04	0.05	0.06	0.07
	HEP	31.8	22.06	15.54	14.07	12.59
	DFHOI	14.58	11.25	9.62	8.6	7.66
Kosarak	ξ(%)	0.6	0.7	0.8	0.9	1
	HEP	91.98	63.31	47.16	37.2	32.11
	DFHOI	49.84	32.71	26.3	21.02	18.91
Online Retail	ξ(%)	0.1	0.2	0.3	0.4	0.5
	HEP	60.23	31.15	19.6	14.78	11.53
	DFHOI	47.18	25.02	16.21	13.51	9.66
Retail	ξ(%)	0.06	0.07	0.08	0.09	0.1
	HEP	189.4	127.47	56.12	42.06	34.33
	DFHOI	90.16	64.17	21.43	17.77	16.06
T10I4D100K	ξ(%)	0.1	0.2	0.3	0.4	0.5
	HEP	60.91	29.37	21.99	17.56	14.57
	DFHOI	26.62	21.9	18.18	14.6	12.28

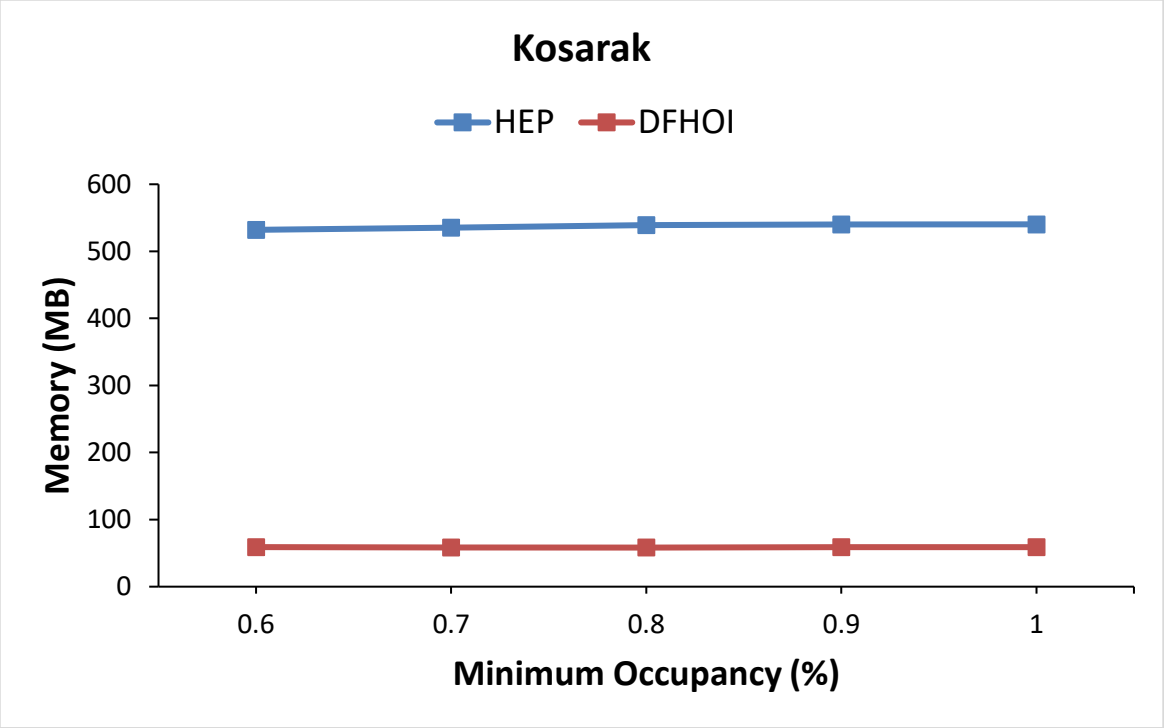
4.3 Bộ nhớ sử dụng



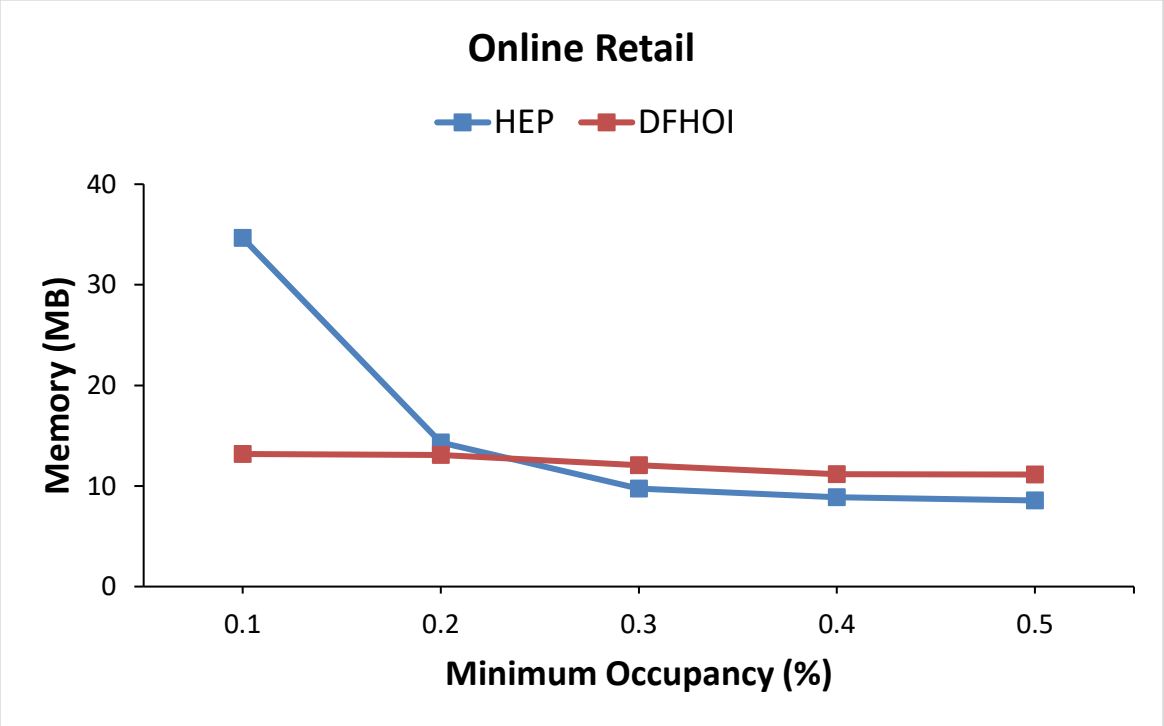
Hình 4.7 Bộ nhớ sử dụng trên tập Foodmart



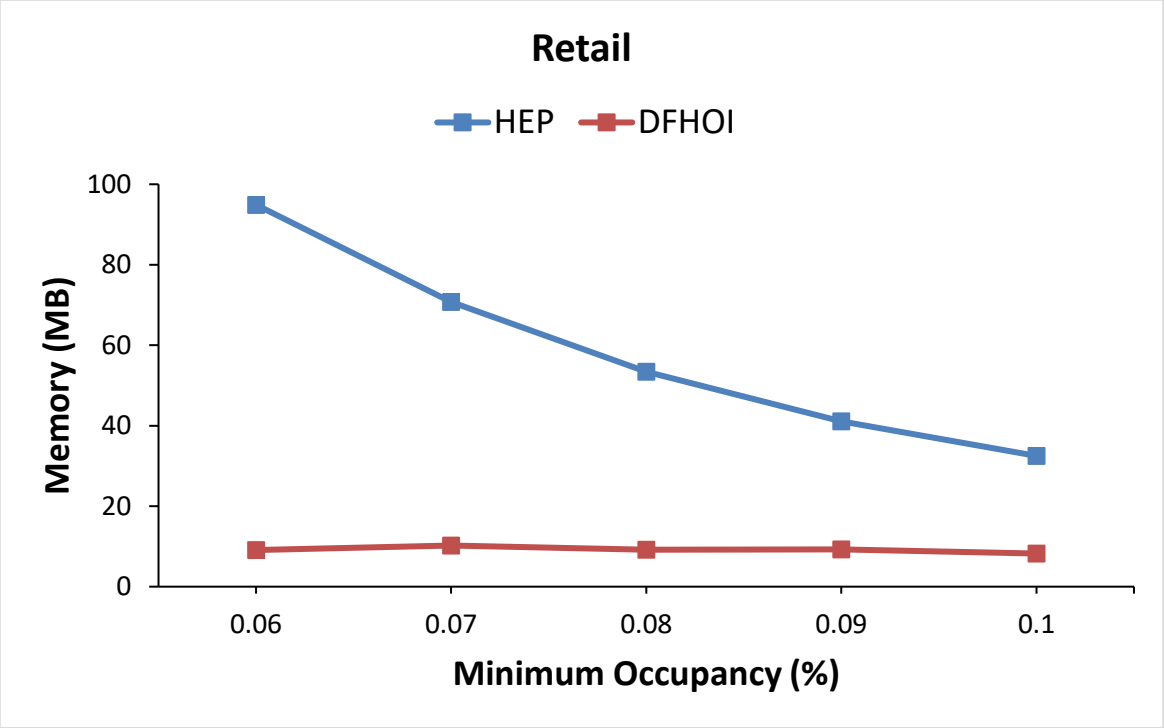
Hình 4.8 Bộ nhớ sử dụng trên tập Fruit Hut



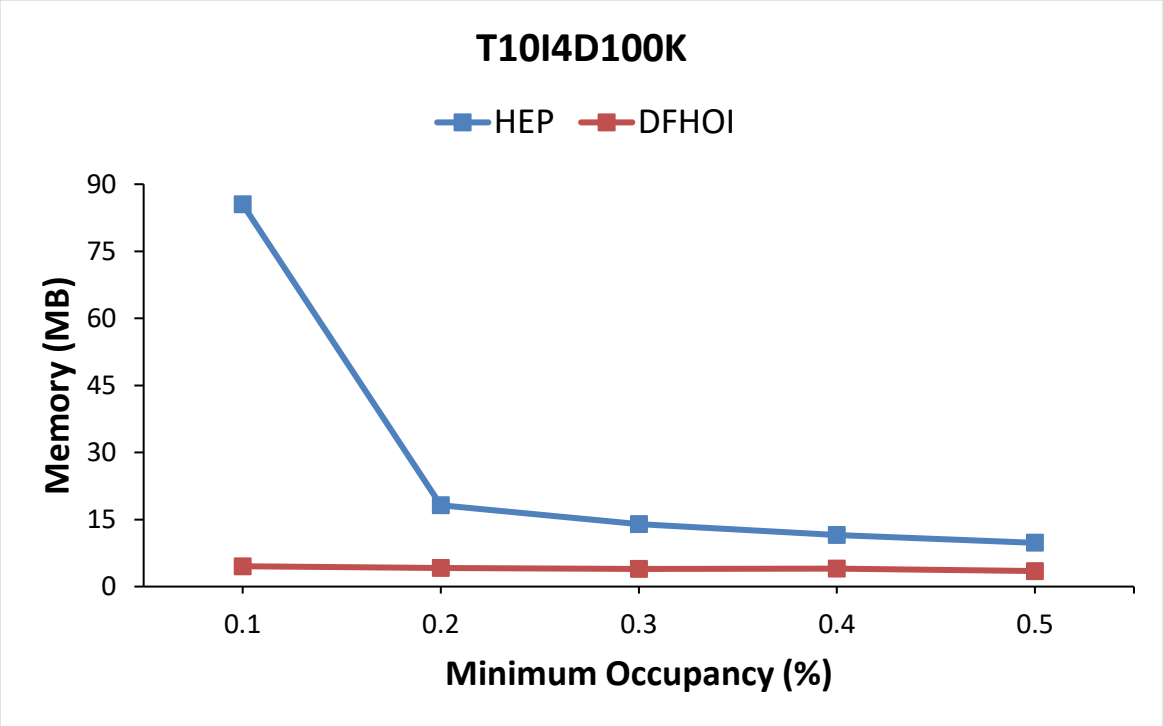
Hình 4.9 Bộ nhớ sử dụng trên tập Kosarak



Hình 4.10 Bộ nhớ sử dụng trên tập Online Retail



Hình 4.11 Bộ nhớ sử dụng trên tập Retail



Hình 4.12 Bộ nhớ sử dụng trên tập T10I4D100K

Hình 4.7 – Hình 4.11 cho thấy dung lượng bộ nhớ sử dụng (memory) của thuật toán HEP và DFHOI viết bằng Python trên các tập dữ liệu mẫu. Kết quả thực nghiệm cho thấy thuật toán DFHOI sử dụng ít bộ nhớ hơn so với HEP trong hầu hết trường hợp. Ví dụ, đối với tập Kosarak, bộ nhớ sử dụng của DFHOI của bằng khoảng 10% HEP. Và từ hình Hình 4.8, ta nhận thấy khi ngưỡng tối thiểu giảm từ 0.07% xuống 0.03%, bộ nhớ sử dụng của HEP tăng 167% trong khi DFHOI gần như không thay đổi. Trên các tập dữ liệu còn lại, tốc độ tăng của thuật toán DFHOI cũng chậm hơn đáng kể so với HEP. Bảng 4.4 cho thấy bộ nhớ sử dụng khi triển khai bằng Java.

Bảng 4.4 Bộ nhớ sử dụng – Java

Đơn vị: megabyte (MB)

Foodmart	ξ(%)	0.05	0.06	0.07	0.08	0.09
	HEP	361.17	360.77	533.51	347.15	346.26
	DFHOI	96.34	88.71	74.21	75.71	40.71
Fruit Hut	ξ(%)	0.03	0.04	0.05	0.06	0.07
	HEP	337.93	162.38	455.05	198.5	197.08
	DFHOI	326.93	215.23	249.54	304.97	344.66
Kosarak	ξ(%)	0.6	0.7	0.8	0.9	1
	HEP	1,059.47	1,481.44	1,546.14	890.46	1,274.58
	DFHOI	800.65	1,042.36	764.44	1,371.58	1,185.16
Online Retail	ξ(%)	0.1	0.2	0.3	0.4	0.5
	HEP	915.7	1,076.75	982.96	1,045.58	838.08
	DFHOI	367.65	615.07	540.8	736.6	618.08
Retail	ξ(%)	0.06	0.07	0.08	0.09	0.1
	HEP	690.2	839.89	875.08	330.39	498.83
	DFHOI	581.37	166.69	373.82	295.46	286.54
T10I4D100K	ξ(%)	0.1	0.2	0.3	0.4	0.5
	HEP	708.65	362.33	378.44	355.19	269.44
	DFHOI	155.12	251.27	82.1	192.64	129.47

CHƯƠNG 5. VẤN ĐỀ VÀ GIẢI PHÁP

5.1 Nêu vấn đề

Tổng quan, thuật toán HEP do Deng đề xuất đã sử dụng chiến lược phát sinh và kiểm tra các tập ứng viên bằng giá trị cận trên occupancy, sử dụng cấu trúc dữ liệu occupancy-list để tránh quét cơ sở dữ liệu nhiều lần. Điều này đã giúp loại bỏ sớm các tập không đủ điều kiện, giảm đáng kể thời gian và chi phí tính toán so với hướng tiếp cận brute-force của thuật toán baseline truyền thống. Đối với thuật toán baseline, chúng ta sẽ phát sinh ra các tập khác rỗng từ database gồm n sản phẩm, do đó sẽ có tất cả $2^n - 1$ tập cần phải kiểm tra, dẫn đến vùng không gian tìm kiếm vô cùng rộng lớn. Và với mỗi itemset P, thuật toán sẽ quét qua cơ sở dữ liệu và tính giá trị occupancy để xem P có phải là high occupancy itemset hay không. Như vậy, tổng số lần quét database của thuật toán baseline là 2^n. Đối với các tập dữ liệu lớn, phương thức này hoàn toàn không khả thi. Tuy nhiên, trong quá trình triển khai thuật toán HEP, em nhận thấy rằng có sự trùng lặp giữa các itemset được tạo ra. Do các itemset không quan trọng về mặt thứ tự nên việc kiểm tra lại các tập trùng lặp là lãng phí. Cho nên, em đã thêm vào thuật toán HEP bước kiểm tra itemset đã được duyệt hay chưa trước khi thực hiện các thao tác tiếp theo.

Năm 2022, Nguyen và các đồng tác giả khác đã nghiên cứu thuật toán HEP và phát hiện ra rằng để giải quyết vấn đề trùng lặp, có thể sử dụng chiến lược tìm kiếm theo chiều sâu để khai phá các itemset. Điều này giúp giảm đáng kể thời gian kiểm tra và tăng hiệu suất của thuật toán. Bên cạnh đó, họ cũng nhận thấy việc sử dụng cấu trúc occupancy-list chưa tối ưu về mặt lưu trữ. Do đó, Nguyen đã đề xuất sử dụng mảng độ dài L và support transaction set thay thế. Điều này đặc biệt hiệu quả đối với các tập dữ liệu có kích thước khổng lồ. Cuối cùng, họ đã phát hiện rằng đối với các database có cùng độ dài transaction thì giá trị cận trên occupancy của itemset P sẽ bằng với giá trị support của nó. Cho nên, chúng ta không cần lãng phí thời gian tính toán $UBO(P)$ trong trường hợp đặc biệt này. Với tất cả ý tưởng cải tiến nêu trên,

Nguyen và các chuyên gia đã giới thiệu một thuật toán mới hiệu quả hơn để khai thác các tập chiếm tỷ lệ cao trong cơ sở dữ liệu, được gọi là DFHOI.

Tuy nhiên, sau khi thử nghiệm thuật toán HEP và DFHOI trên các tập dữ liệu mẫu thì em nhận thấy rằng người dùng có thể gặp khó khăn trong việc chọn ngưỡng tối thiểu phù hợp. Nếu ξ quá thấp, có rất nhiều itemset cần phải kiểm tra, dẫn đến không gian tìm kiếm tăng đáng kinh ngạc. Hoặc tệ hơn là dung lượng bộ nhớ không đủ dẫn đến máy tính bị treo do phần lớn các thao tác của thuật toán được xử lý thuần túy in-memory. Ngược lại, nếu ξ quá cao thì sẽ không có tập nào đủ điều kiện.

Việc chọn ngưỡng ξ phù hợp phải tiến hành thủ công bằng cách thử nghiệm nhiều giá trị ξ khác nhau cho đến khi số lượng HOI trả về đáp ứng yêu cầu. Điều này đòi hỏi khá nhiều thời gian, công sức và cả kinh nghiệm của người dùng. Để khắc phục hạn chế đó, em đã lên ý tưởng và đề xuất một thuật toán mới có khả năng lấy ngưỡng tối thiểu một cách hoàn toàn tự động, được gọi là ATHOI (auto-thresholding for high occupancy itemset mining). Thuật toán ATHOI thân thiện hơn với người sử dụng, do đó có thể được ứng dụng rộng rãi trong thực tế. Chi tiết về thuật toán ATHOI sẽ được trình bày trong phần sau.

5.2 Thuật toán ATHOI

Về mặt cơ bản, thuật toán ATHOI được phát triển dựa trên DFHOI. Do đó, trong phần này chúng ta chủ yếu đề cập đến những điểm khác biệt của thuật toán ATHOI. Đầu tiên, ATHOI nhận tham số đầu vào là số lượng HOI mà người dùng cần tìm n thay vì ngưỡng occupancy tối thiểu ξ. Điều này cho phép người dùng có thể nhập vào chính xác số lượng itemset mà họ mong muốn mà không phải thử nghiệm với nhiều ngưỡng ξ khác nhau. Thứ hai, thuật toán sẽ khởi tạo ngưỡng ξ mặc định bằng 0 khi bắt đầu (dòng 2) và tiến hành cập nhật tham số này trong quá trình khai phá các itemset. Không giống như DFHOI, trong phương thức *Mine_HOI_1Itemset* và *Mine_Depth_HOIs*, ta sẽ tính giá trị occupancy để tìm các tập đủ điều kiện trước khi tính giá trị cận trên occupancy để tìm các ứng viên. Lý do cho việc này là sau khi

thêm một itemset mới vào tập $HOIs$, chúng ta có thể cần cập nhật lại giá trị của ξ. Tại đây, có hai trường hợp có thể xảy ra như sau:

1. Dòng 12 – 13 và dòng 28 – 29, khi số lượng itemset nhỏ hơn n thì thuật toán sẽ thêm itemset P vào tập $HOIs$.
2. Dòng 14 – 16 và dòng 30 – 32, khi số lượng itemset lớn hơn hoặc bằng n thì thuật toán sẽ tìm ra itemset có giá trị occupancy nhỏ nhất trong $HOIs$ và xóa nó khỏi danh sách. Tiếp theo, tiến hành cập nhật ngưỡng ξ bằng cách lấy giá trị occupancy nhỏ nhất của các itemset trong $HOIs$.

Dựa vào cơ chế trên, thuật toán ATHOI có thể tự động điều chỉnh ngưỡng occupancy tối thiểu sao cho phù hợp với số lượng itemset mà người dùng cần khai thác. Ta có thể thấy rằng thuật toán ATHOI hiệu quả hơn nhiều so với cách tiếp cận brute-force, tính occupancy của toàn bộ itemset và sắp xếp theo thứ tự giảm dần để tìm ra top n itemset có giá trị occupancy cao nhất. Phương pháp này vô cùng tốn kém về mặt thời gian và chi phí tính toán nên không khả thi đối với các tập dữ liệu lớn. Tuy nhiên, so với DFHOI thì thuật toán ATHOI có hiệu suất thấp hơn do hai nguyên nhân chủ yếu sau. Thứ nhất, thao tác tìm kiếm và xóa itemset có giá trị occupancy thấp nhất trong danh sách $HOIs$ gồm n phần tử tốn $\Theta(n)$ thời gian. Thứ hai, đối với các tập dữ liệu đặc biệt có độ dài các transaction bằng nhau, ATHOI không thể tận dụng được Định lý 5 vì ngưỡng ξ thay đổi liên tục. Nhìn chung, thuật toán ATHOI chỉ phù hợp khi người dùng muốn tìm ra top n itemset với giá trị n không quá lớn. Mã giả của thuật toán được trình bày trong Bảng 5.1 dưới đây.

Bảng 5.1 Thuật toán ATHOI

Input: transaction database DB and the number of high occupancy itemsets n **Output:** n high occupancy itemsets $HOIs$
ATHOI() 1. Scan DB to generate a set of 1-itemset S, with an index of length L and tidset of each 1-itemset ; 2. $HOIs \leftarrow \emptyset; \xi \leftarrow 0;$

3. $C_1 \leftarrow \emptyset; HOI_1 \leftarrow \emptyset;$
4. $\{C_1, HOI_1\} \leftarrow Mine\_HOI\_1Itemset(S);$
5. $HOIs \leftarrow HOIs \cup HOI_1;$
6. **if** $C_1 \neq \emptyset$ **then**
7. $\quad HOIs \leftarrow HOIs \cup Mine\_Depth\_HOIs(C_1);$
8. **return** $HOIs;$

***Mine_HOI_1Itemset*(S)**

9. **for** each 1-itemset $P \in S$ **do**
10. $\quad$ **if** $Sup(P) > \xi$ **then**
11. $\quad\quad$ **if** $O(P) > \xi$ **then**
12. $\quad\quad\quad$ **if** $len(HOI_1) < n$ **then**
13. $\quad\quad\quad\quad HOI_1 \leftarrow HOI_1 \cup \{P\};$
14. $\quad\quad\quad$ **else**
15. $\quad\quad\quad\quad$ Find itemset having the smallest occupancy value and remove it from HOI_1;
16. $\quad\quad\quad\quad \xi \leftarrow minOcp(HOI_1);$
17. $\quad\quad$ Scan $STSet(P)$ to calculate $UBO(P)$ based on L;
18. $\quad\quad$ **if** $UBO(P) > \xi$ **then**
19. $\quad\quad\quad C_1 \leftarrow C_1 \cup \{P\};$
20. **return** $\{C_1, HOI_1\};$

***Mine_Depth_HOIs*(C)**

21. **for** each itemset $P_1 \in C$ **do**
22. $\quad C_k \leftarrow \emptyset;$
23. $\quad$ **for** each itemset $P_2 \in C$ with P_2 following P_1 **do**
24. $\quad\quad P \leftarrow P_1 \cup P_2;$
25. $\quad\quad STSet(P) \leftarrow STSet(P_1) \cap STSet(P_2);$

```
26.         if Sup(P) > ξ then
27.             if O(P) > ξ then
28.                 if len(HOIs) < n then
29.                     HOIs ← HOIs ∪ {P};
30.                 else
31.                     Find itemset having the smallest occupancy value and
                        remove it from HOIs;
32.                     ξ ← minOcp(HOIs);
33.             Scan STSet(P) to calculate UBO(P) based on L;
34.             if UBO(P) > ξ then
35.                 C_k ← C_k ∪ {P};
36.     if C_k ≠ ∅ then
37.         Mine_Depth_HOIs(C_k);
```

5.3 Mô phỏng

Chúng ta sẽ mô phỏng cách thức hoạt động của chiến lược tìm kiếm theo chiều sâu và lấy ngưỡng tự động trong thuật toán ATHOI để khai thác các tập sản phẩm chiếm tỷ lệ cao từ cơ sở dữ liệu giao dịch trong Bảng 1.1. Đầu tiên, giả sử chọn số lượng high occupancy itemset là 10, ta có:

- C_1: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}$
- HOI_1: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}$
- $\xi = 0$

Từ kết quả trên, thuật toán ATHOI thực hiện gọi đệ quy phương thức *Mine_Depth_HOIs* như sau:

$Mine\_Depth\_HOIs(\{a\}, \{b\}, \{c\}, \{d\}, \{e\})$, kết quả:

- C: $\{ab\}, \{ac\}, \{ad\}, \{ae\}$
- $HOIs$: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{ab\}, \{ac\}, \{ad\}, \{ae\}$
- $\xi = 0$
- $Mine\_Depth\_HOIs(\{ab\}, \{ac\}, \{ad\}, \{ae\})$, kết quả:

- C: $\{abc\}, \{abd\}, \{abe\}$
- $HOIs$: $\{a\}, \{b\}, \{c\}, \{d\}, \{e\}, \{ab\}, \{ac\}, \{ad\}, \{abc\}, \{abd\}$
- $\xi = 0.6$
- $Mine\_Depth\_HOIs(\{abc\}, \{abd\}, \{abe\})$, kết quả:
  - C: $\{abcd\}, \{abce\}, \{abde\}$
  - $HOIs$: $\{a\}, \{c\}, \{d\}, \{ab\}, \{ac\}, \{ad\}, \{abd\}, \{abcd\}, \{abce\}, \{abde\}$
  - $\xi = 0.8$
  - $Mine\_Depth\_HOIs(\{abcd\}, \{abce\}, \{abde\})$, kết quả:
    - C: $\{abcde\}$
    - $HOIs$: $\{a\}, \{c\}, \{d\}, \{ab\}, \{ac\}, \{ad\}, \{abd\}, \{abcd\}, \{abde\}, \{abcde\}$
    - $\xi = 0.8$
- C: $\{acd\}, \{ace\}$
- $HOIs$: $\{a\}, \{c\}, \{d\}, \{ab\}, \{ac\}, \{ad\}, \{abd\}, \{abde\}, \{abcde\}, \{acd\}$
- $\xi = 0.8$
- $Mine\_Depth\_HOIs(\{acd\}, \{ace\})$, kết quả:
  - C: $\{\}$
  - $HOIs$: $\{a\}, \{c\}, \{d\}, \{ab\}, \{ac\}, \{ad\}, \{abd\}, \{abde\}, \{abcde\}, \{acd\}$
  - $\xi = 0.8$
- C: $\{ade\}$
- $HOIs$: $\{a\}, \{c\}, \{d\}, \{ab\}, \{ac\}, \{ad\}, \{abd\}, \{abde\}, \{abcde\}, \{acd\}$
- $\xi = 0.8$

- $C = \{bc\}, \{bd\}, \{be\}$
- $HOIs$: $\{a\}, \{c\}, \{d\}, \{ab\}, \{ac\}, \{ad\}, \{abd\}, \{abcde\}, \{acd\}, \{bd\}$

- $\xi = 1.07$
- $Mine\_Depth\_HOIs(\{bc\}, \{bd\}, \{be\})$, kết quả:
    - $C: \{bcd\}, \{bce\}$
    - $HOIs: \{a\}, \{c\}, \{d\}, \{ac\}, \{ad\}, \{abd\}, \{acd\}, \{bd\}, \{bcd\}, \{bce\}$
    - $\xi = 1.07$
    - $Mine\_Depth\_HOIs(\{bcd\}, \{bce\})$, kết quả:
        - $C: \{bcde\}$
        - $HOIs: \{a\}, \{c\}, \{d\}, \{ad\}, \{abd\}, \{acd\}, \{bd\}, \{bcd\}, \{bce\}, \{bcde\}$
        - $\xi = 1.12$
    - $C: \{bde\}$
    - $HOIs: \{a\}, \{d\}, \{ad\}, \{abd\}, \{acd\}, \{bd\}, \{bcd\}, \{bce\}, \{bcde\}, \{bde\}$
    - $\xi = 1.35$
- $C: \{cd\}, \{ce\}$
- $HOIs: \{a\}, \{d\}, \{ad\}, \{abd\}, \{acd\}, \{bd\}, \{bcde\}, \{bde\}, \{cd\}, \{ce\}$
- $\xi = 1.35$
- $Mine\_Depth\_HOIs(\{cd\}, \{ce\})$, kết quả:
    - $C: \{cde\}$
    - $HOIs: \{a\}, \{d\}, \{ad\}, \{abd\}, \{acd\}, \{bd\}, \{bcde\}, \{cd\}, \{ce\}, \{cde\}$
    - $\xi = 1.36$
- $C: \{de\}$
- $HOIs: \{d\}, \{ad\}, \{abd\}, \{acd\}, \{bd\}, \{bcde\}, \{cd\}, \{ce\}, \{cde\}, \{de\}$
- $\xi = 1.57$

Thông tin chi tiết về giá trị support, occupancy và upper-bound occupancy của các k-itemset được trình bày trong Bảng 5.2.

Bảng 5.2 Thông tin các itemset

Itemset	$\boldsymbol{Sup(P)}$	$\boldsymbol{UBO(P)}$	$\boldsymbol{O(P)}$
$\{a\}$	4	2.73	1.37
$\{b\}$	3	2.35	0.78
$\{c\}$	4	3.35	1.12
$\{d\}$	6	4.35	1.95
$\{e\}$	3	2.35	0.78
$\{ab\}$	2	1.6	1.07
$\{ac\}$	2	1.6	1.07
$\{ad\}$	4	2.73	2.73
$\{ae\}$	1	2.6	0.4
$\{bc\}$	2	1.8	0.9
$\{bd\}$	3	2.35	1.57
$\{be\}$	2	1.8	0.9
$\{cd\}$	4	2.35	2.23
$\{ce\}$	3	2.35	1.57
$\{de\}$	3	2.35	1.57
$\{abc\}$	1	1	0.6
$\{abd\}$	2	1.6	1.6
$\{abe\}$	1	1	0.6
$\{acd\}$	2	1.6	1.6
$\{ace\}$	1	1	0.6
$\{ade\}$	1	1	0.6
$\{bcd\}$	2	1.8	1.35
$\{bce\}$	2	1.8	1.35
$\{bde\}$	2	1.8	1.35
$\{cde\}$	3	2.35	2.35
$\{abcd\}$	1	1	0.8

Itemset	$\boldsymbol{Sup(P)}$	$\boldsymbol{UBO(P)}$	$\boldsymbol{O(P)}$
$\{abce\}$	1	1	0.8
$\{abde\}$	1	1	0.8
$\{bcde\}$	2	1.8	1.8
$\{abcde\}$	1	1	1

5.4 Thực nghiệm

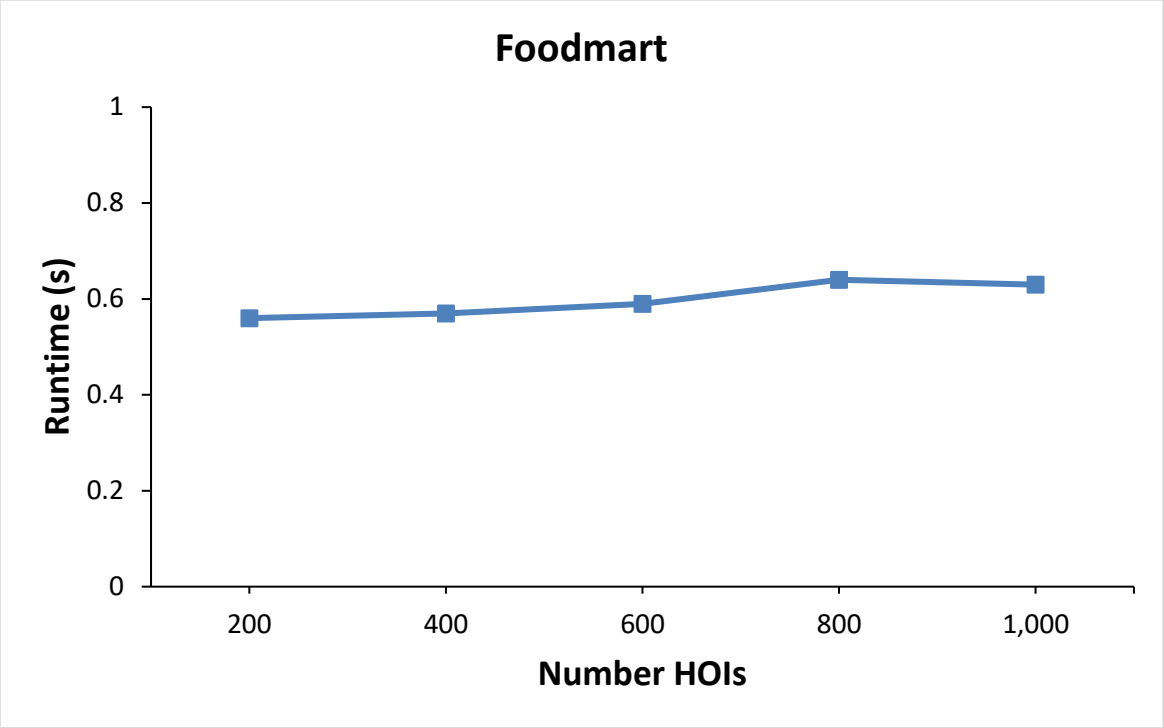
Trong phần này, chúng ta sẽ đánh giá hiệu suất của thuật toán ATHOI bằng cách chạy thử nghiệm trên sáu tập dữ liệu mẫu trong Bảng 4.1. Các thông số về thời gian thực thi và dung lượng bộ nhớ sử dụng sẽ được ghi lại nhằm mục đích phân tích và so sánh. Thuật toán được triển khai bằng ngôn ngữ Python với môi trường và phương thức thực nghiệm tương tự như trong chương 4.

5.4.1 Thời gian thực thi

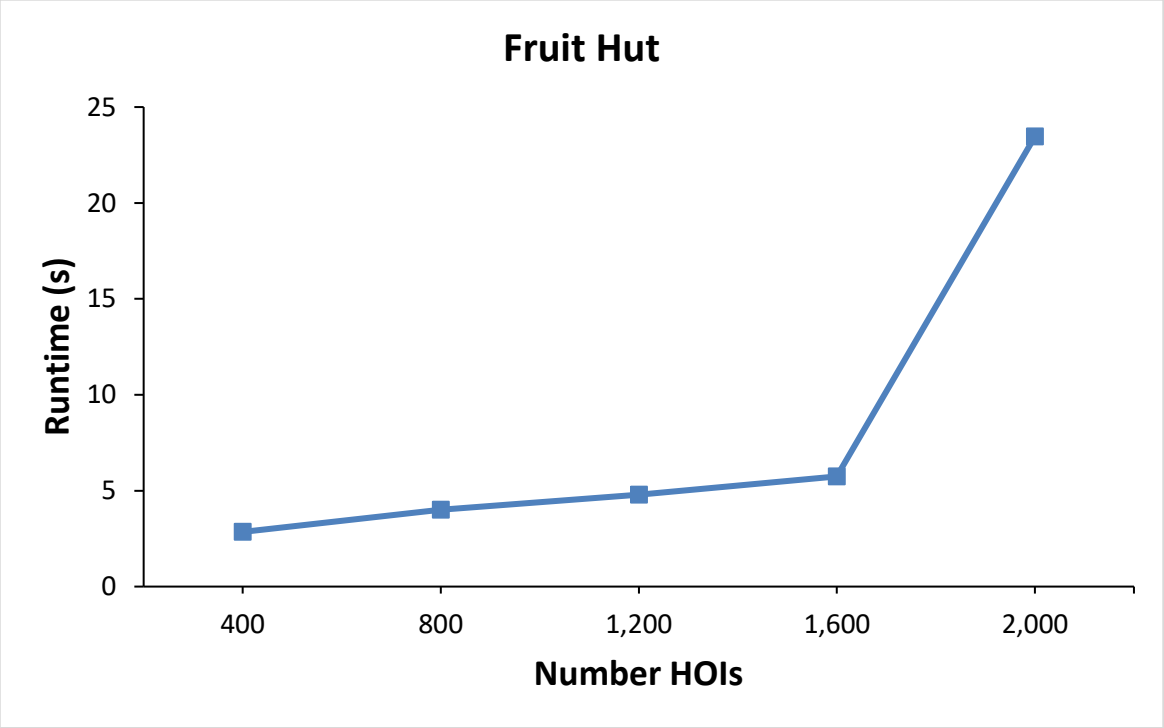
Hình 5.1 – Hình 5.6 thể hiện thời gian thực thi (runtime) của thuật toán ATHOI trên các tập dữ liệu. Ta có thể thấy rằng khi số lượng itemset cần khai thác n tăng lên thì thời gian chạy cũng tăng lên. Ví dụ, đối với tập Fruit Hut, khi n tăng từ 400 lên 2,000 thì runtime cũng tăng từ 2.85s lên 23.46s. Tuy nhiên, tốc độ tăng trưởng này không đáng ngại vì trong thực tế, người dùng chỉ muốn tìm $10 - 100$ tập sản phẩm chiếm tỷ lệ cao nhất trong cơ sở dữ liệu.

5.4.2 Bộ nhớ sử dụng

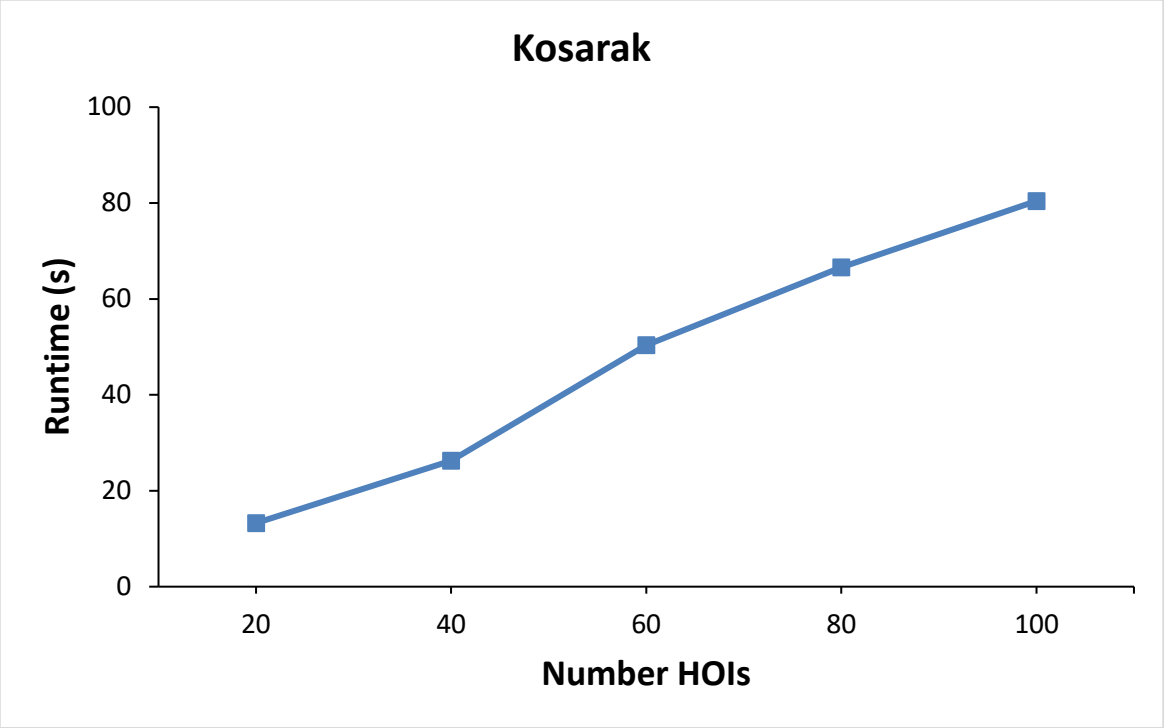
Hình 5.7 – Hình 5.12 cho thấy dung lượng nhớ nhớ sử dụng (memory) khi chạy thuật toán ATHOI trên sáu tập dữ liệu mẫu. Nhìn chung, khi số lượng itemset cần khai thác tăng lên thì dung lượng bộ nhớ sử dụng cũng tăng nhưng không đáng kể. Ví dụ, đối với tập Retail, khi n tăng gấp 5 lần $(200 - 1{,}000)$ thì bộ nhớ sử dụng chỉ tăng khoảng 69% (7.60MB – 12.9MB). Tương tự cho các tập dữ liệu còn lại.



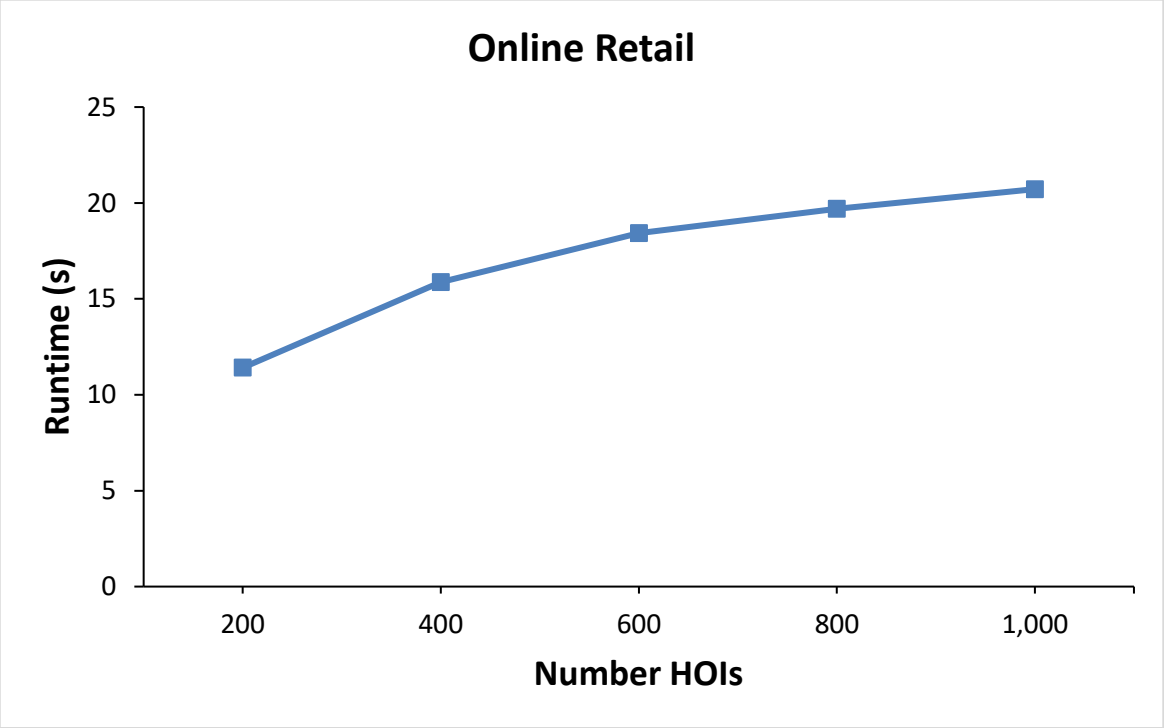
Hình 5.1 Thời gian thực thi trên tập Foodmart



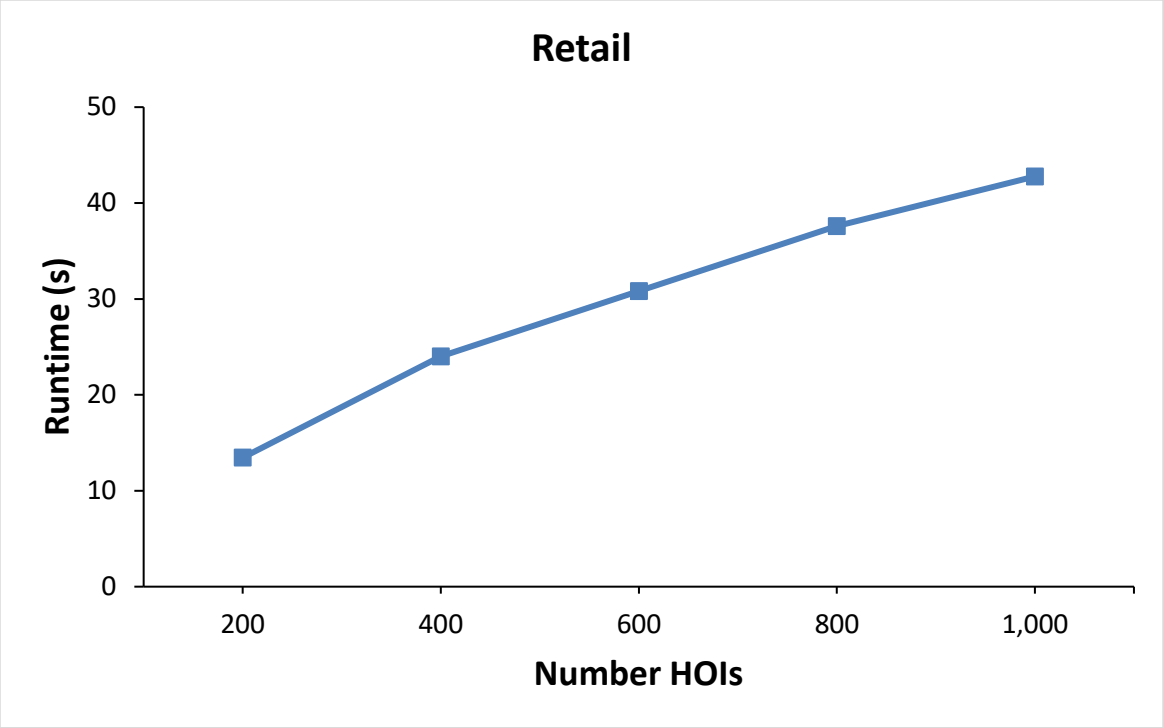
Hình 5.2 Thời gian thực thi trên tập Fruit Hut



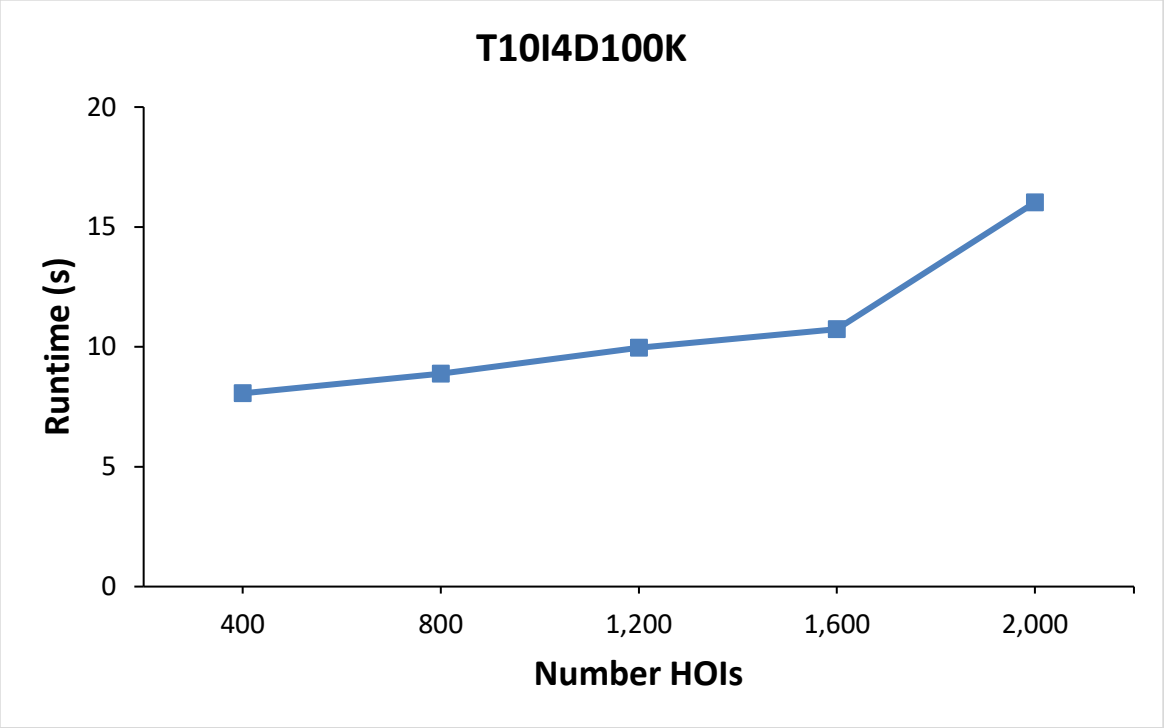
Hình 5.3 Thời gian thực thi trên tập Kosarak



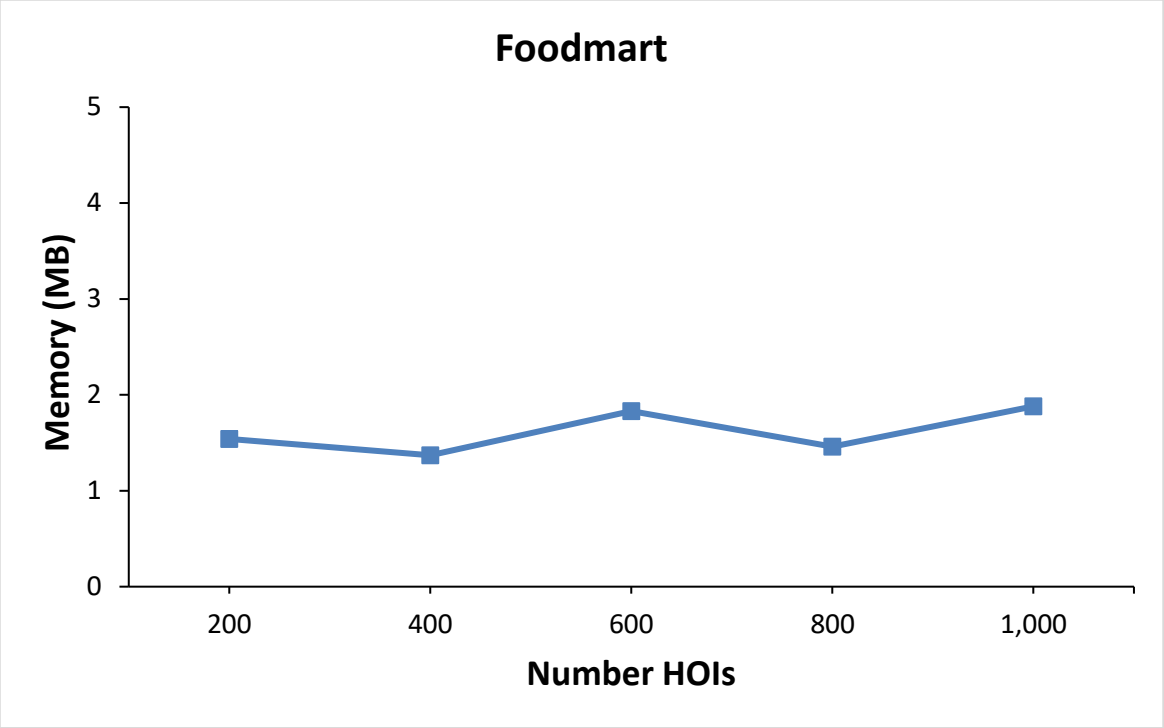
Hình 5.4 Thời gian thực thi trên tập Online Retail



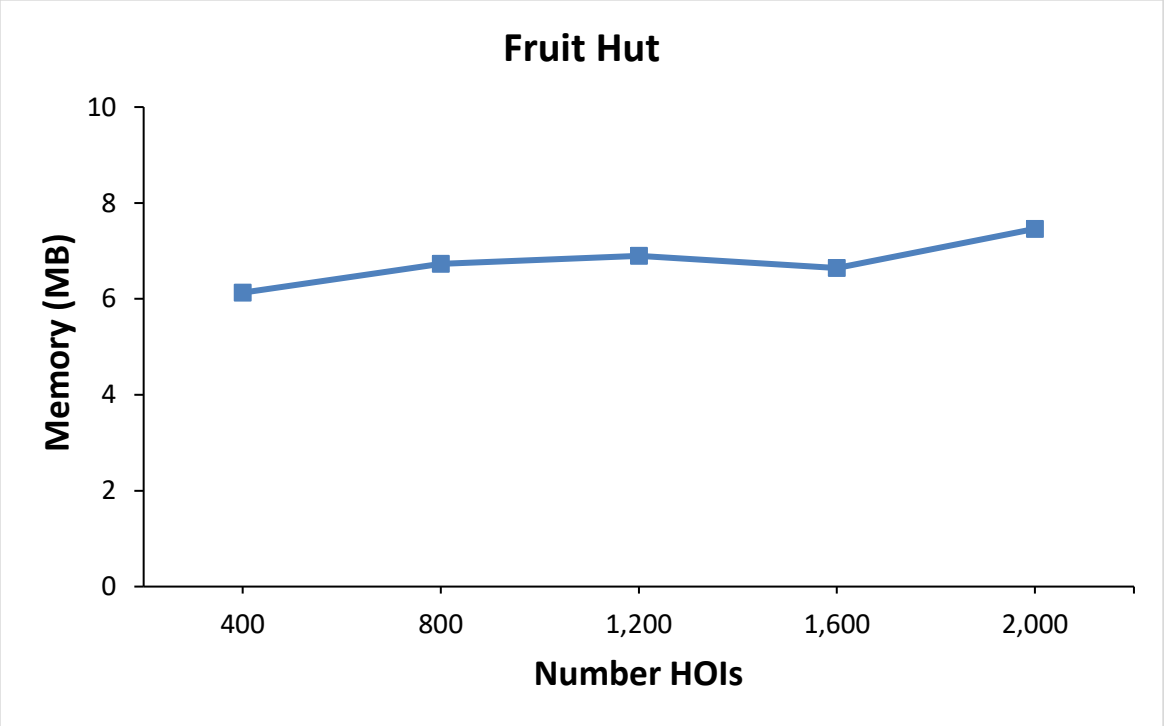
Hình 5.5 Thời gian thực thi trên tập Retail



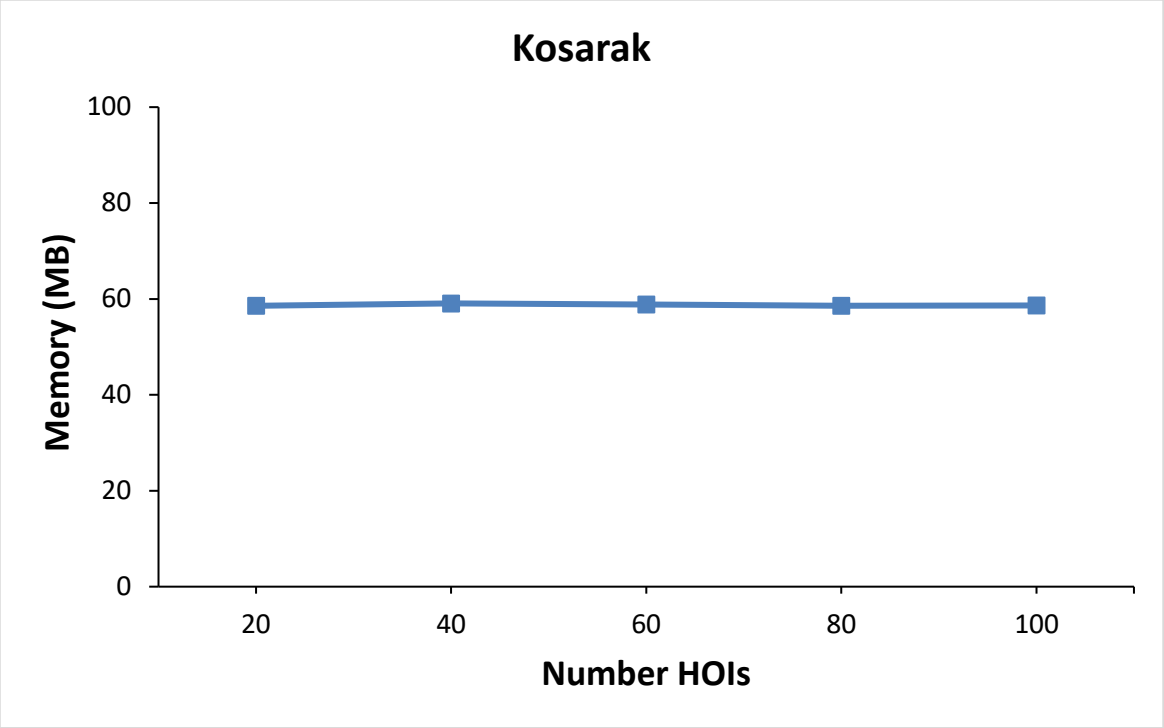
Hình 5.6 Thời gian thực thi trên tập T10I4D100K



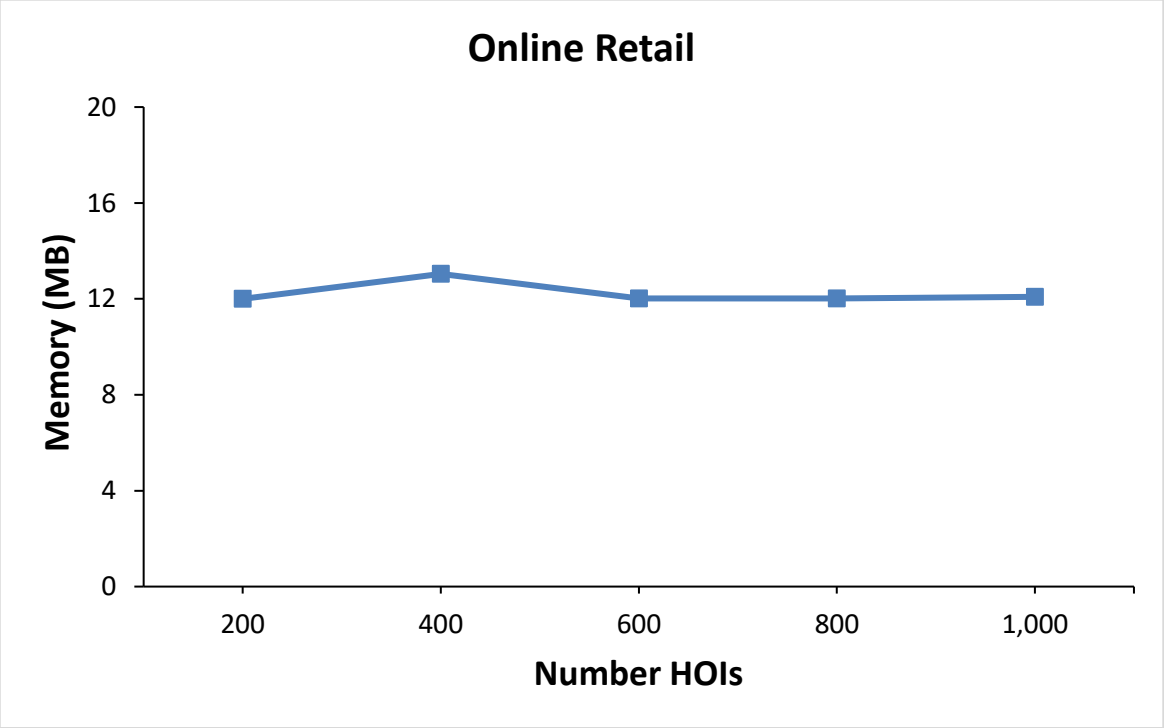
Hình 5.7 Bộ nhớ sử dụng trên tập Foodmart



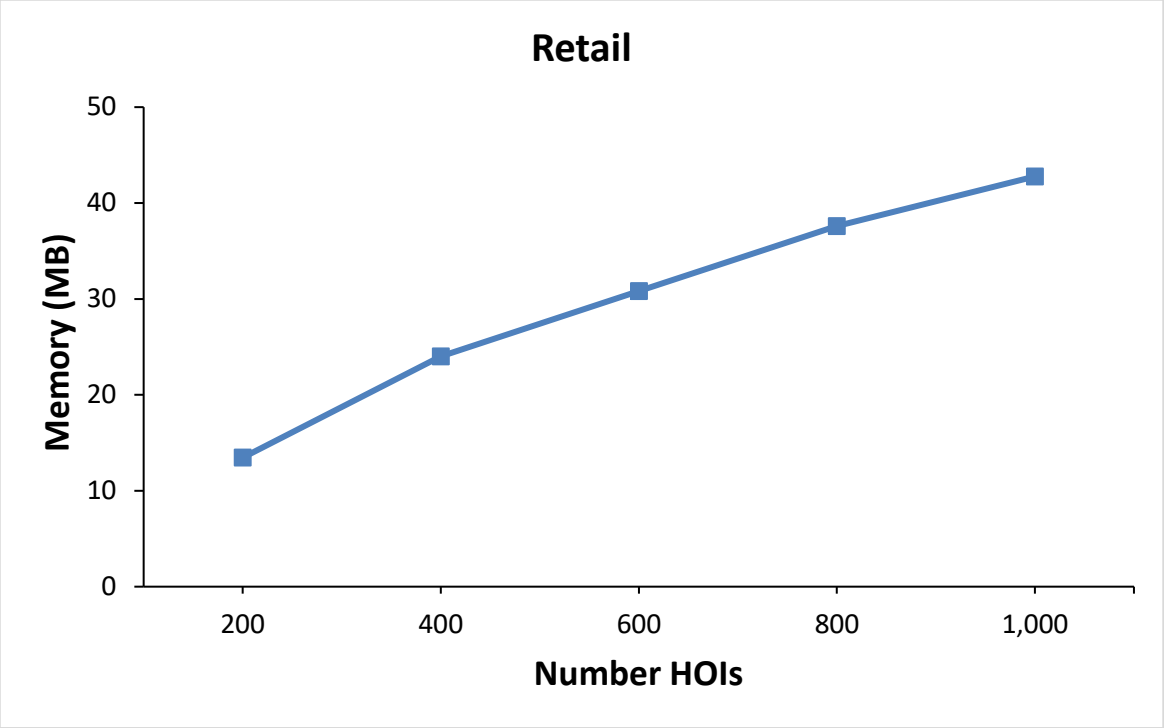
Hình 5.8 Bộ nhớ sử dụng trên tập Fruit Hut



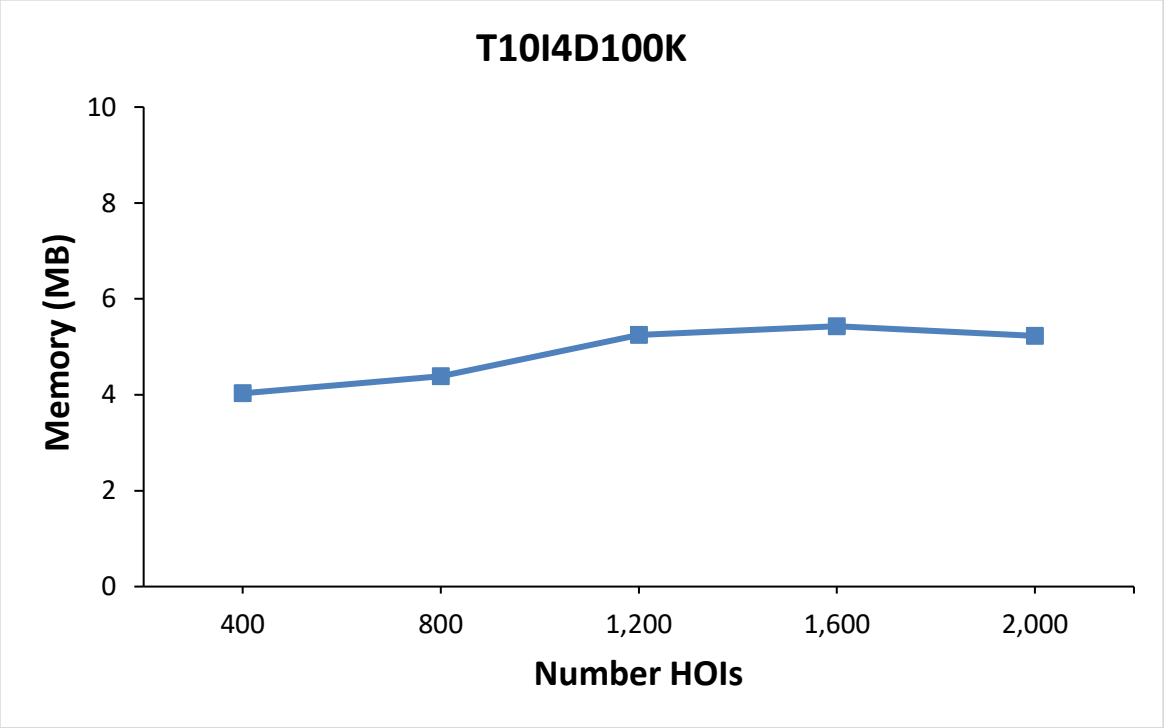
Hình 5.9 Bộ nhớ sử dụng trên tập Kosarak



Hình 5.10 Bộ nhớ sử dụng trên tập Online Retail



Hình 5.11 Bộ nhớ sử dụng trên tập Retail



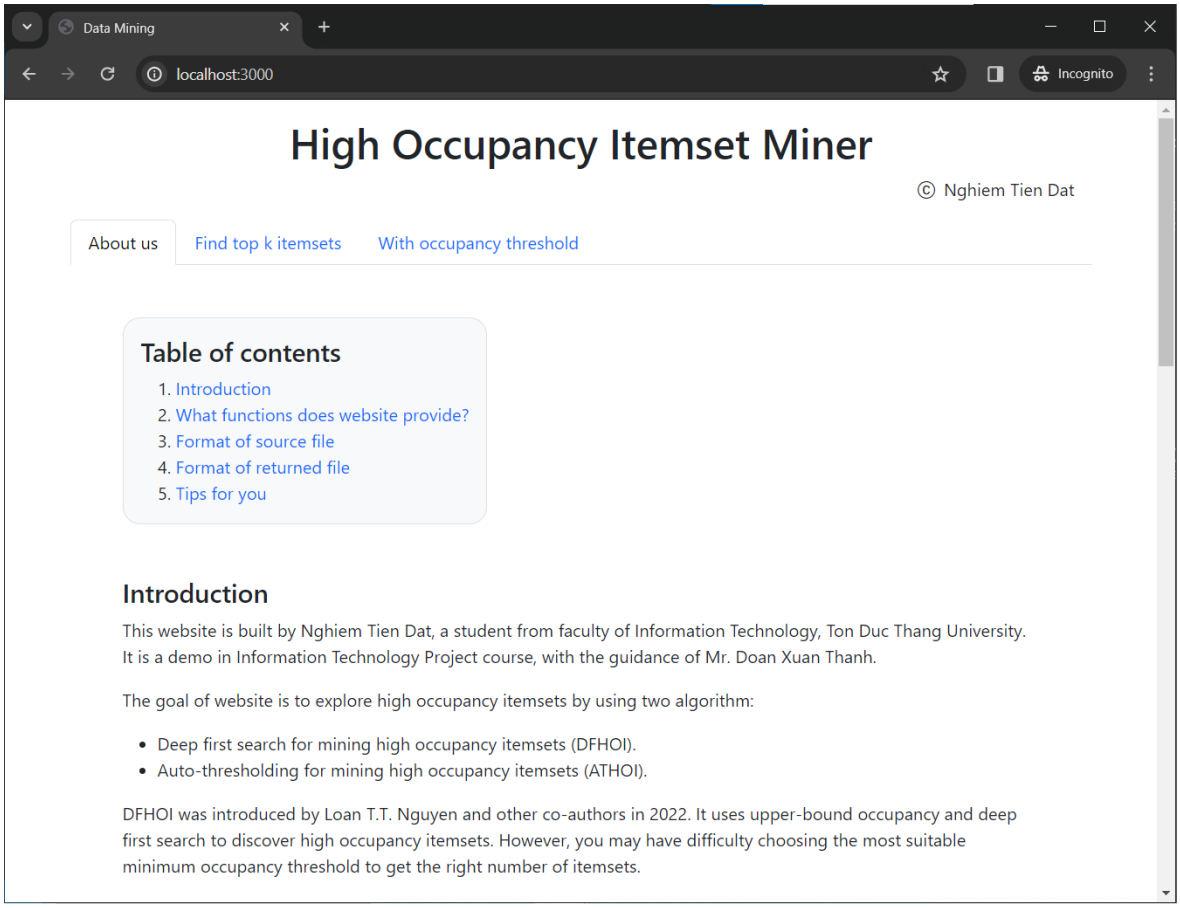
Hình 5.12 Bộ nhớ sử dụng trên tập T10I4D100K

CHƯƠNG 6. ỨNG DỤNG

6.1 Giới thiệu

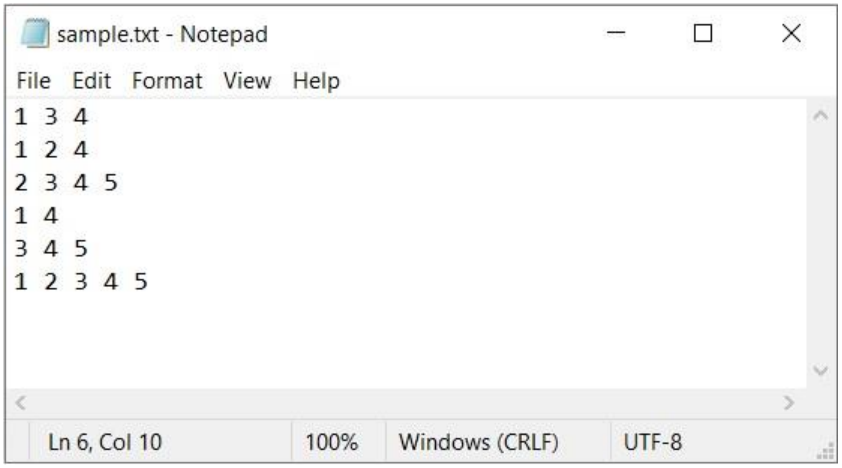
Nhằm triển khai các thuật toán khai thác tập chiếm tỷ lệ cao trong cơ sở dữ liệu, em đã xây dựng một ứng dụng đơn giản trên nền tảng web với tên gọi là High Occupancy Itemset Miner. Về công nghệ, phần back-end của website được viết bằng Nodejs và Python, trong khi phần front-end sử dụng HTML, Bootstrap và JavaScript. Về tính năng, trang web cung cấp cho người dùng hai chức năng cơ bản là:

1. Tìm top *k* itemset có giá trị occupancy cao nhất trong tập dữ liệu. Tính năng này được xây dựng dựa trên thuật toán ATHOI.
2. Tìm tất cả itemset có giá trị occupancy lớn hơn hoặc bằng ngưỡng tối thiểu mà người dùng cho trước. Tính năng này được xây dựng dựa trên thuật toán DFHOI.

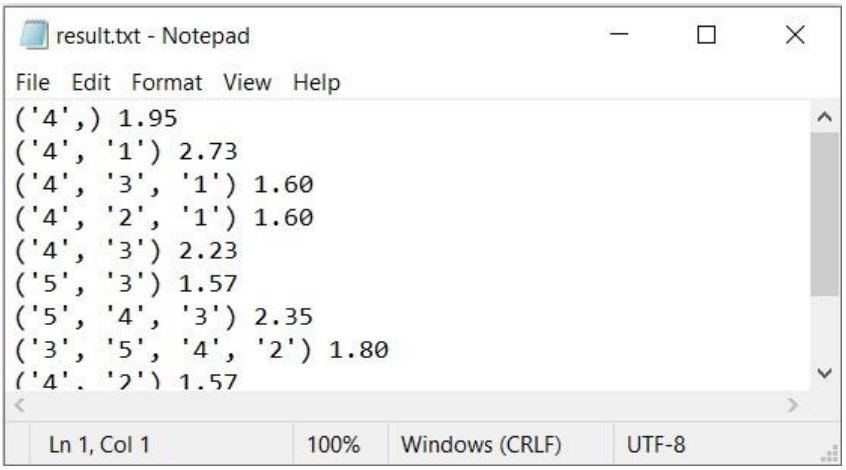


Hình 6.1 Ứng dụng High Occupancy Itemset Miner

Một số lưu ý khi sử dụng High Occupancy Itemset Miner như sau. Đầu tiên, file đầu vào (source file) là một file text chứa dữ liệu giao dịch. Trong đó, mỗi dòng đại diện cho một transaction và mỗi item trong transaction cách nhau bằng kí tự khoảng trắng. Thứ hai, file trả về (returned file) cũng là một file text chứa các tập chiếm tỷ lệ cao khai thác được từ dữ liệu đầu vào. Trong đó, mỗi dòng là một itemset và giá trị occupancy tương ứng của nó, cách nhau bởi kí tự khoảng trắng. Các item trong tập sản phẩm nằm trong dấu ngoặc đơn và được ngăn cách bởi dấu phẩy. Chúng ta có thể xem format của source file và returned file trong Hình 6.2 và Hình 6.2 theo thứ tự. Source file trong Hình 6.2 dựa theo cơ sở dữ liệu trong Bảng 1.1 với các item a, b, c, d, e lần lượt là 1, 2, 3, 4, 5. Và Hình 6.3 thể hiện kết quả trả về khi chạy thuật toán ATHOI với $k = 10$.



Hình 6.2 Format của source file



Hình 6.3 Format của returned file

Tiếp theo, để tiết kiệm thời gian và tài nguyên hệ thống, em quyết định đặt ra một số hạn chế như sau. Kích thước file tối đa có thể tải lên server là 5MB. Nếu thời gian thực thi vượt quá 5 phút thì tiến trình đó sẽ bị chấm dứt. Returned file được lưu trữ trên server trong thời gian tối đa là 5 phút, sau đó file sẽ bị xóa và người dùng không thể tải xuống được nữa. Đồng thời, website cũng khuyến kích người dùng chọn giá trị k hoặc ngưỡng occupancy tối thiểu phù hợp. Chi tiết về cách sử dụng các tính năng của trang web được trình bày trong phần tiếp theo.

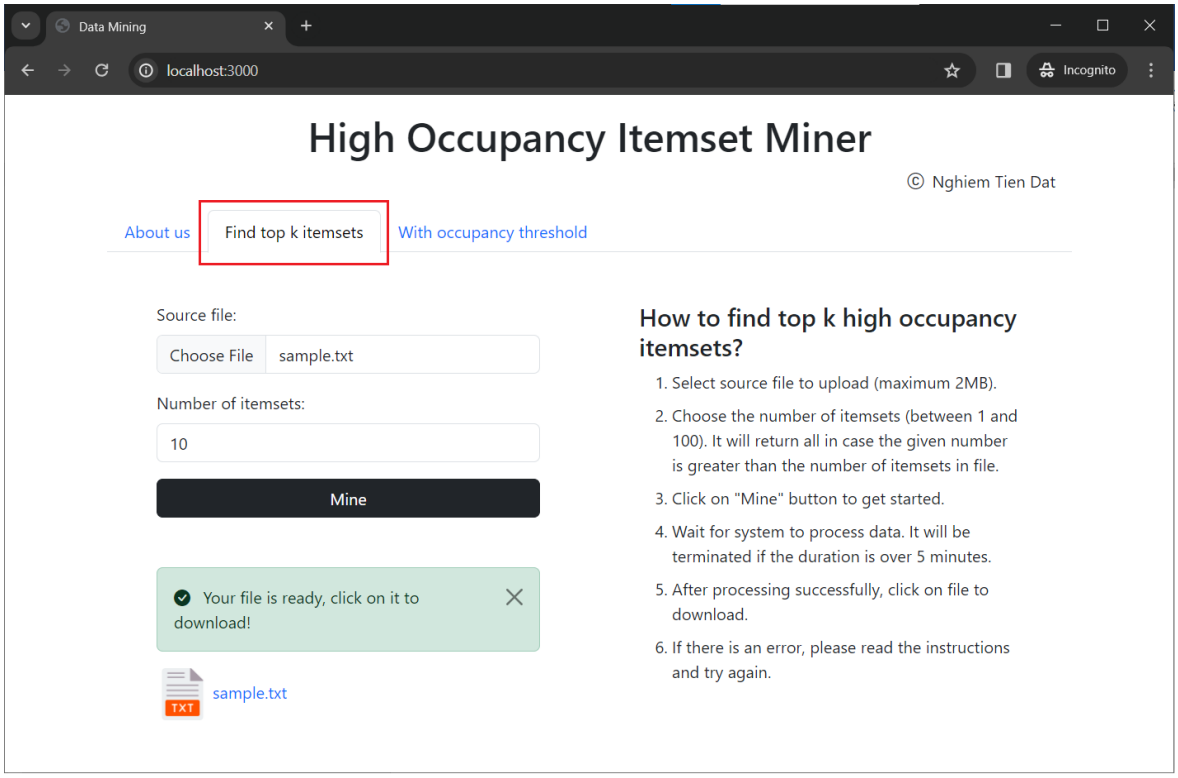
6.2 Tìm top k HOI

Để tìm top k itemset có giá trị occupancy cao nhất, nhấp chọn tag "find top k itemset" trên navigation bar (Hình 6.4) và thực hiện theo các bước sau:

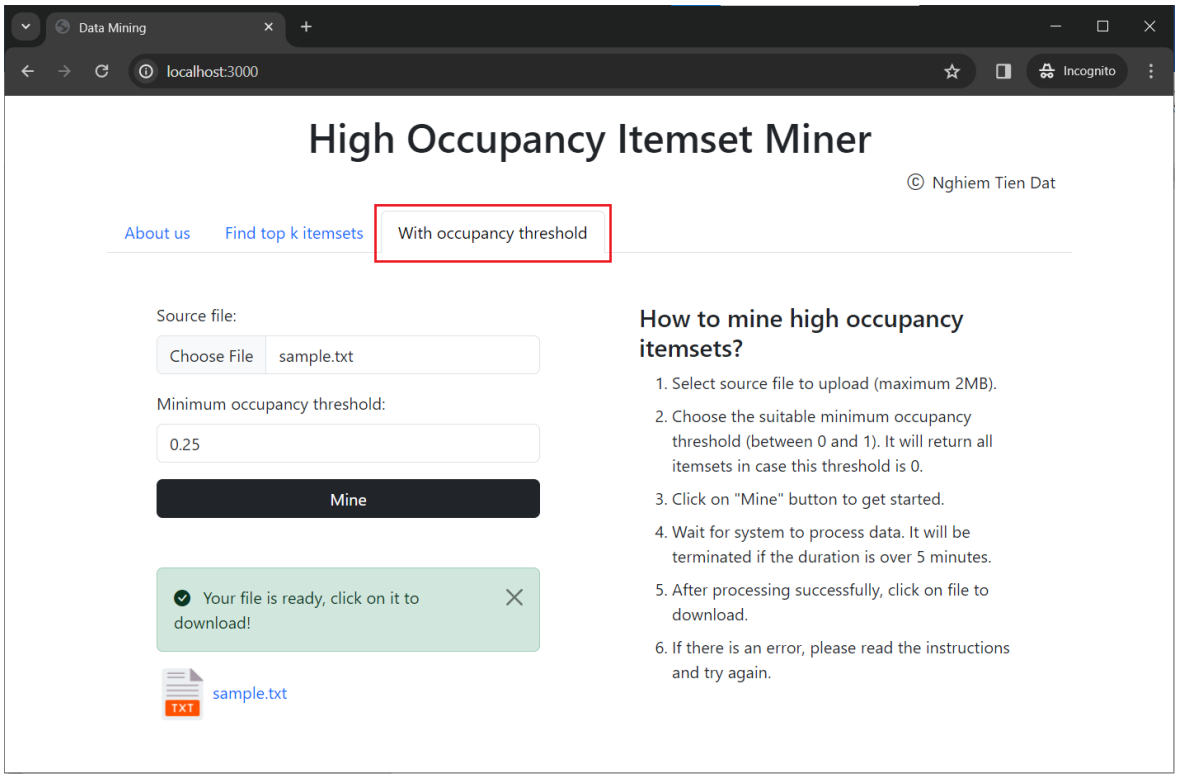
1. Chọn source file để tải lên (tối đa 5MB).
2. Chọn số lượng itemset k cần khai thác (từ 1 đến 100). Lưu ý rằng nếu số lượng itemset trong tập dữ liệu đầu vào nhỏ hơn k thì thuật toán sẽ trả về toàn bộ itemset.
3. Nhấn vào nút "Mine" để bắt đầu.
4. Chờ hệ thống xử lý dữ liệu. Nếu thời gian thực thi vượt quá 5 phút thì tiến trình sẽ bị chấm dứt.
5. Sau khi chạy thành công, nhấn vào file trả về để tải xuống.
6. Nếu có lỗi xảy ra, vui lòng đọc hướng dẫn và thử lại.

6.3 Tìm HOI sử dụng ngưỡng tối thiểu

Để tìm tất cả các itemset có giá trị occupancy lớn hơn hoặc bằng ngưỡng tối thiểu cho trước, nhấp chọn tag "With occupancy threshold" (Hình 6.5) và thực hiện tương tự. Tuy nhiên, thay vì chọn số lượng itemset thì chúng ta sẽ chọn ngưỡng occupancy tối thiểu ξ phù hợp (từ 0 đến 1). Lưu ý rằng thuật toán sẽ trả về toàn bộ itemset trong trường hợp $\xi = 0\%$.



Hình 6.4 Tìm top k itemset



Hình 6.5 Sử dụng ngưỡng tối thiểu

CHƯƠNG 7. KẾT LUẬN

7.1 Nội dung đã thực hiện

Trong môn dự án công nghệ thông tin, em đã tìm hiểu tổng quan về vấn đề khai thác các tập sản phẩm chiếm tỷ lệ cao thuộc lĩnh vực data mining. Đồng thời, em cũng triển khai các thuật toán thường được sử dụng như HEP và DFHOI bằng ngôn ngữ lập trình Python và Java. Thuật toán HEP của Deng đã sử dụng cấu trúc occupancy-list để tránh quét database nhiều lần và upper-bound occupancy để loại bỏ sớm các tập ứng viên không đủ điều kiện. Tuy nhiên, các itemset có thể bị trùng lặp nên cần phải thêm bước kiểm tra xem itemset đã được duyệt qua hay chưa. Thuật toán DFHOI của Nguyen và các đồng tác giả khác đã khắc phục được vấn đề này bằng cách sử dụng chiến lược tìm kiếm theo chiều sâu. Đồng thời, họ cũng đề xuất sử dụng mảng độ dài và support transaction set thay cho occupancy-list để tối ưu hóa bộ nhớ lưu trữ. Ngoài ra, đối với các database có cùng độ dài, giá trị cận trên occupancy sẽ bằng với giá trị support của itemset. Em đã thử nghiệm cả hai thuật toán trên sáu tập dữ liệu mẫu. Kết quả chỉ ra rằng DFHOI hiệu quả hơn HEP cả về thời gian xử lý lẫn dung lượng bộ nhớ sử dụng.

Tuy nhiên, hạn chế của HEP và DFHOI là không thân thiện với người sử dụng, nghĩa là người dùng sẽ gặp nhiều khó khăn trong việc xác định ngưỡng occupancy tối thiểu phù hợp. Nếu ngưỡng này quá thấp, phạm vi không gian tìm kiếm tăng đáng kể dẫn đến tốn kém thời gian và tài nguyên tính toán. Ngược lại, khi ngưỡng này quá cao, không một itemset nào đáp ứng được yêu cầu. Do đó, em đã đề xuất một thuật toán mới dựa trên DFHOI, được gọi là ATHOI. Thuật toán này cho phép người dùng nhập số lượng itemset cần khai thác và ngưỡng tối thiểu sẽ được tính toán hoàn toàn tự động. Em cũng đã thử nghiệm thuật toán trên nhiều tập dữ liệu khác nhau và nhận thấy rằng ATHOI đặc biệt phù hợp khi người dùng muốn khai thác một số lượng nhỏ high occupancy itemset. Đồng thời, em đã xây dựng một ứng dụng web đơn giản bằng Nodejs gọi là High Occupancy Itemset Miner với các tính năng dựa theo thuật toán DFHOI và ATHOI.

7.2 Hướng phát triển dự án

Trong tương lai, em sẽ tiếp tục cải tiến thuật toán ATHOI. Có thể thấy rằng, thời gian để tìm kiếm itemset có giá trị occupancy nhỏ nhất và cập nhật lại ngưỡng tối thiểu sẽ tăng lên đáng kể khi số lượng itemset cần khai thác càng lớn. Bên cạnh đó, các thuật toán khai thác tập chiếm tỷ lệ cao hiện tại đang xem mỗi item trong cơ sở dữ liệu có tầm quan trọng ngang nhau. Nhưng trong thực tế, có những sản phẩm đắt tiền và mang lại nhiều lợi nhuận hơn nên sẽ có mức độ quan trọng cao hơn. Lấy ví dụ đơn giản, lợi nhuận thu được khi bán 100 chai nước suối thấp hơn 1 chiếc TV. Do đó, chúng ta có thể thêm trọng số cho mỗi item để tìm ra danh sách các itemset chiếm lợi nhuận cao nhất trong cơ sở dữ liệu (high utility occupancy itemset mining).

TÀI LIỆU THAM KHẢO

Deng, Z. H. (2017, February 18). Mining high occupancy itemsets. *Elsevier*, 222-229. Retrieved February 25, 2024, from https://www.sciencedirect.com

Nguyen, T. T., Mai, T., Pham, G. H., Yun, U., & Vo, B. (2022, February 20). An efficient method for mining high occupancy itemsets based on equivalence class and early pruning. *Elsevier*, 1-21. Retrieved February 25, 2024, from https://www.sciencedirect.com