An Adaptive Utility Quantification Strategy for Penetration Semantic Knowledge Mining

Yichao Zang[1], Tairan Hu[1(✉)], Rongrong Cao[2], and Junhu Zhu[1]

[1] National Key Laboratory of Mathematical Engineering and Advanced Computing, Zhengzhou 450001, China
aipteamzhouty@aliyun.com

[2] Political College of National Defense University, Shanghai 200433, China

Abstract. Penetration semantic knowledge mining plays an important role in automated penetration testing. The loss of external utility makes it hard to employ high utility itemset mining algorithms to retrieve the knowledge. To overcome this problem, this paper proposes an adaptive utility quantification strategy which could differentiate and quantify external utility of item effectively. Further, incremental high utility pattern tree structure is adopted to maintain utility information for incremental database so as to facilitate the calculation of external utility. The experimental result turns out that high utility itemsets mining algorithms with proposed adaptive utility quantification strategy could mine penetration testing semantic knowledge from raw penetration testing data effectively and efficiently.

Keywords: Penetration semantic knowledge · High utility itemsets mining · Automated penetration testing · Frequent itemsets mining

1 Introduction

With the rapid development of computer network, the security problem becomes much more prominent than ever. Penetration testing shows great advantages in improving security level. It interleaves scanning and vulnerability exploitation actions where scanning action offers information such as operating systems, services etc. based on which security experts choose corresponding vulnerability exploitation program and correct vulnerability exploitation program could motivate further information gathering. Figure 1 shows typical penetration testing scenario where attack path is composed of three steps ①②③, and each step is pair of host information and common vulnerability and exposure exploitation(cve), such as <tomcat 7.0.56, cve-2017-12615>, <windows 7 sp1, cve-2018-0121> etc. The quality or effectiveness of penetration testing depends heavily on security experts' experience and it becomes a hot research topic to automate penetration testing, for which mining penetration semantic knowledge is an essential prerequisite.

G. Xu et al. (Eds.): FCS 2020, CCIS 1286, pp. 654–666, 2020.
https://doi.org/10.1007/978-981-15-9739-8_48

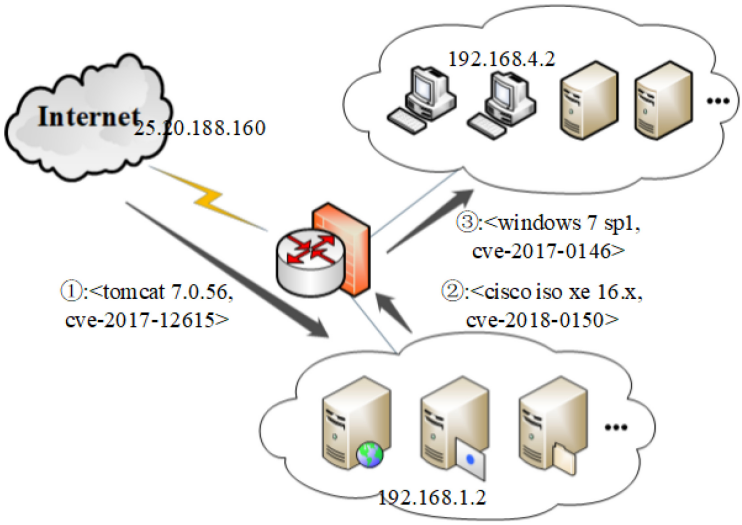

Fig. 1. Typical penetration testing scenario where attack path is composed of three steps ①②③, and each step is pair of host and vulnerability exploitation information.

Penetration semantic knowledge is a kind of mapping relationship {*software:version* → *vulnerability*} which means that the specific *version* of *software* may cause the *vulnerability*, based on which we could choose corresponding *vulnerability* exploitation program when faced with specific *version* of *software*. Existed researches extract semantic knowledge from penetration testing data through transforming specific vulnerability database [1] such as metasploit framework. Taking *vulnerability*, numbered cve-2019-0708, for example, the affected platforms include windows 7 sp1, windows server 2003 sp2, windows server 2008 sp2, so there are three individual penetration semantic knowledge, namely {windows 7 sp1 → cve-2019-0708}, {windows server 2003 sp2 → cve-2019-0708}, {windows xp sp3 → cve-2019-0708} meaning that when host owns one of the above operating systems, there is great possibility that it owns vulnerability numbered cve-2019-0708, so that we could use corresponding vulnerability exploitation program to control the host. There are two disadvantages for this kind of approach, one is that the semantic knowledge extracted from specific vulnerability database could not match information gathered through scanners, the other is that the penetration semantic knowledge becomes tedious without considering information gathered through multiple scanners. Table 1 shows an example of application, operating system and vulnerability information gathered by Nmap [2] and Shodan [3] scanner. The mapping relationship {Apache httpd:2.4.29 → cve-2018-7584} is our interested penetration knowledge which owns high utility because vulnerability numbered cve-2018-7584 is caused by Apache httpd:2.4.29. The aim of penetration semantic knowledge mining is to discover all of penetration semantic knowledge which owns high utility from

raw penetration testing data. High utility itemsets mining algorithms (HUIM) seem could solve the problem but fail because the loss of external utility for each item. To solve this problem, we proposed an adaptive utility quantification strategy ARUQ, which could measure importance of item automatically to achieve penetration semantic knowledge mining.

Table 1. Transactions of application, operating system and vulnerability information for each individual host.

IP	Application	Operating system	Vulnerability
171.3.13.3	OpenSSH:7.2 Apache httpd:2.4.29	Linux kernel:2.6.32	cve-2018-7584 cve-2018-10547
11.4.1.63	OpenSSH:7.4 BIND:9.11 Apache httpd:2.4.29	Linux kernel:2.6.32	cve-2018-7584 cve-2018-10547
41.3.5.175	MySQL:5.5.55, Apache httpd:2.4.29	Linux kernel:2.6.32	cve-2018-7584 cve-2018-10547
121.5.7.15	MySQL:5.7	Linux kernel:2.6.32	

The remainder of the paper is organized as follows. Section 2 presents background knowledge of high utility mining and penetration semantic knowledge mining. Section 3 presents our proposed adaptive utility quantification strategy for each item in penetration testing transaction. Section 4 analyses details of penetration testing data and compares performance of high utility itemsets mining algorithms with proposed strategy on these data with frequent itemsets mining algorithms. Section 5 summarizes our study and points out some future research issues.

2 Background

As penetration semantic knowledge mining could be transformed into high utility itemsets mining problem. It is necessary to introduce some preliminaries and background knowledge of high utility itemsets mining first.

2.1 Preliminaries

Definition 1 (Transaction Database). *Given set I, transaction database D is a set where each item satisfies $T_i \subseteq I, T_i \in D$. For example, the penetration testing transaction database is composed of all records shown in Table 1. A positive value $p(s)$ is called external utility for each item $s \in I$, and the number of s in T_i is called internal utility of item s, represented in $q(s, T_i)$.*

Definition 2 (Utility of item in transaction/database). *The utility of item s in transaction T_i is the product of internal utility and external utility, represented in $u(s, T_i) = p(s) * q(s, T_i)$, the utility of item s in database D is the sum of item s in all transactions belonging to D, represented in $u(s, D) = \sum_{i=1}^{|D|} u(s, T_i)$.*

Definition 3 (Utility of set in transaction/database). *The utility of set X in transaction T_i is the sum of all item utility in the transaction, represented in $u(X, T_i) = \sum_{s \in X} p(s) * q(s, T_i)$, the utility of set X is the sum of all items in database, represented in $u(X, T_i) = \sum_{s \in X} \sum_{i=1}^{|D|} u(s, T_i), X \subseteq T_i$.*

Definition 4 (High Utility Itemsets Mining). *Given a user-specified utility threshold ξ, high utility itemsets mining aims to discover all itemsets X from database D which satisfies $u(X, D) \geq \xi$. In Table 1, the high utility itemset is {Apache httpd:2.4.29, cve-2018-7584} because it is the penetration semantic knowledge we want to mine from transaction database.*

Definition 5 (Penetration Semantic Knowledge Mining). *Penetration semantic knowledge is a collection of itemset $\{item_{causal}, item_{effect}\}$ where $item_{causal}$ is precondition item and $item_{effect}$ is result item. Vividly, we could regard $item_{effect}$ as a bucket and each item appeared together with $item_{effect}$ in a transaction is put into the bucket, and the process of penetration semantic knowledge mining is to filter all irrelevant items, the left items with $item_{effect}$ is final interested penetration semantic knowledge. $\{item_{causal}, item_{effect}\}$ denotes that when $item_{causal}$ occurs, $item_{effect}$ occurs with great possibility. We want to find all $item_{causal}$ which could result in $item_{effect}$, and this process could be formalized as a special kind of high utility itemsets mining problem, which owns the following characteristics:*

- *The external utility for each item is unknown.*
- *The internal utility for each item in each transaction equals 1.*
- *Items appeared frequently for each bucket contribute to high utility.*
- *Items appeared frequently for multiple buckets contribute not to high utility.*
- *The effect item in a transaction must appeared in each individual final high utility itemset.*
- *Every itemset without effect item must not be high utility itemset.*

Penetration semantic knowledge mining aims to discover all causal related itemsets within each individual bucket. The transactions with same effect item are in same bucket, taking Table 1 for example, transactions 1, 2, 3 are in same bucket because all of their effect items include cve-2018-7584. Items Apache httpd:2.4.29 and linux kernel:2.6.32 appeared frequently in the bucket, but Apache httpd:2.4.29 contributes more to discover knowledge {Apache httpd:2.4.29, cve-2018-7584} than linux kernel:2.6.32 because linux kernel:2.6.32 appeared frequently in other buckets, illustrating that linux kernel:2.4.29 is a common item than causal related item in the knowledge. And

the result high utility itemset is {Apache httpd:2.4.29, cve-2018-7584}. Penetration semantic knowledge mining aims to discover all of these causal related high utility itemsets.

2.2 High Utility Itemsets Mining

Enough works have been done to accelerate high utility itemset mining [4,5], which could be divided into three categories, namely candidate based algorithms, without candidate based algorithms and other algorithms. Two-Phase algorithm [6] is a famous and classical candidate based high utility itemset mining algorithm which is composed of two phase, the first phase prunes search space and generates candidates by proposed transaction weight downward closure property while the second phase scans database to filter high utility itemsets from high transaction weight utility itemsets identified in phase I. Amhed et al. proposed IHUP$_{twu}$ (Incremental High Utility Pattern) tree structure [7] to maintain information of incremental databases for exact utility calculation instead of scanning database. Vincent et al. proposed an algorithm named UP-Growth [8] to mine high utility itemsets, which could construct utility pattern tree from database based on DGU (Discarding Global Unpromising items), DGN (Discarding Global Node utility), DLU (Discarding Local Unpromising items), DLN (Discarding Local Node utility) strategies to prune search space. Even though lots of tricks are proposed to prune search space, there are still a lot of candidate itemsets waiting to be tested in phase II which consumes huge memory and time. To overcome these problems, Liu et al. proposed an algorithm called HUI-Miner [9] to mine high utility itemsets without generating candidates. HUI-Miner uses a novel structure, called utility-list, to store both utility information and heuristic information of itemset for pruning search space. Based on utility-list, the high utility itemsets could be mined by joining utility lists instead of scanning database, which shrinks much mining time. Further, Krishnamoorthy et al. proposed an algorithm called HUP-Miner [10] which employs two novel pruning strategies, namely partitioned utility pruning and lookahead utility pruning to prune search space. Peng et al. proposed a modified HUI-Miner(mHUIMiner) [11], which utilizes IHUP tree structure to guide the itemset expansion process to avoid considering itemsets that are nonexistent in the database. Liu et al. proposes a novel algorithm d^{2}HUP [12] which could mine high utility patterns in a single phase without generating candidates, the novelties lies in a lookahead strategy to avoid enumeration and a linear structure CAUL (Chain of Accurate Utility Lists) for scalable representation of utility information. Fournier-Viger et al. proposed a utility list based fast high-utility miner (FHM) [13] algorithm. The algorithm could reduce join action of utility lists effectively after analyzing co-occurrences property of items. Zida et al. proposed a algorithm named EFIM (Efficient high-utility Itemset Mining) [14] to mine high utility itemsets effectively. EFIM outperforms both in terms of execution time and memory through novel database projection and transaction merging techniques. Considering huge memory consumption problem caused by utility-list intersection/join operation, Duong et al. proposed an improved

utility-list structure called utility-list buffer to reduce the memory consumption and speed up the join operation. This structure is integrated into a novel algorithm named ULB-Miner [15]. Rather than pruning search space by monotonic properties, there are also some works integrating evolutionary computation algorithms into mining high utility itemsets. Kannimuthu and Premalatha firstly adopted the genetic algorithm into high utility itemsets mining, proposed two algorithms HUPEumu-GRAM and HUPEwumu-GRAMs with/without specified minimum utility threshold separately [16]. Lin et al. adopted particle swarm optimization to mine high utility itemsets and proposed an algorithm called HUIM-BPSO$_{sig}$ [17], the algorithm encodes particles as binary variables and takes utility function as fitness function to achieve evolutionary optimization. Wu et al. adopted ant colony optimization into high utility itemsets mining and proposed HUIM-ACS algorithm [18], which could map completed solution space into routing graph to mine high utility itemsets as well as to avoid generating unreasonable solutions.

3 Methodology

To Achieve penetration semantic knowledge mining, we proposed an adaptive utility quantification strategy for individual item whose external utility in bucket m is calculated as follows:

$$p_m(i) = \alpha \frac{N_m}{N} \tag{1}$$

where N_m is the number of transactions containing item i in bucket m, N is the number of transactions containing item i in whole database and α is coefficient to differentiate external utility for item in the bucket. This formula conveys the idea that the external utility of item gets higher when it appeared frequently in the bucket and less in whole database, satisfying characteristic 3 and 4 in Sect. 2.1. After quantifying external utility for each item, all of the classical high utility itemsets mining algorithms could be adopted to mine penetration semantic knowledge which owns high utility.

Even though it seems easy to implement the calculation of external utility, it has to rescan whole database again to update utility for each item when new transactions appear, which consumes huge time. To facilitate high utility mining for incremental database, the IHUP$_{aruq}$ tree structure is proposed, whose construction method is similar to IHUP$_{twu}$ [19]. IHUP$_{aruq}$ is composed of three parts, namely global header table (GHT), local header table (LHT) and local IHUP$_{aruq}$ tree. Element in GHT is composed of three fields, including item_name, count, link, where item_name is the name of item, count is the number of item appeared in whole database and link is pointer to link corresponding item in each local table sequentially. Element in LHT is also composed of three fields, item_name, count, and link, where item_name is the name of item, count is the number of item appeared in the bucket and link is pointer to link corresponding item in IHUP$_{aruq}$ tree sequentially. IHUP$_{aruq}$ tree is constructed for each bucket in adaptive utility descending order. The details of constructing IHUP$_{aruq}$ tree is shown in Algorithm 1 described as follows:

Step1: Scan database to calculate the adaptive external utility for each item in bucket k. Create local header table for items of the bucket in adaptive utility descending order and reorganize transactions of each bucket in utility descending order. Set the count field of each element in LHT_k to the number of item in the bucket. Finally add each item to GHT.
Step2: Create local tree for each individual bucket, the item_name of root node is denoted as effect item. Insert each reorganized transaction into local IHUP_{aruq} tree with prefix share strategy. And increase the number of shared prefix node by 1 or create new branches to maximal share prefix path with increasing node in maximal shared prefix path by 1 and setting the count of items in new branches to 1.
Step3: For each item in GHT, link the pointer field to each LHT with same item sequentially and set the count field in GHT to the sum of count field in each linked LHT, finally reorganize GHT in count descending order to finish the creation of IHUP_{aruq} tree. Based on constructed IHUP_{aruq} tree for each bucket, the high utility itemsets could be retrieved by depth first search method to discover all of those items whose external utility is higher than user specified threshold.

Further, in order to facilitate high utility mining process for incremental database, we could adjust IHUP_{aruq} tree structure instead of creating new one as follows:

Algorithm 1. IHUP_{aruq} tree construction algorithm

Input: original database *db*
Output: GHT, LHTs, IHUP_{aruq}

for *transaction* $\in$ *db* **do**
  $m \leftarrow$ check_bucket(*transaction*)
  buffer_m.append(*transaction*)
  for *item* in *transaction* **do**
    $\text{LHT}_m(item)$ += 1
    $\text{GHT}(item)$ += 1
for k = 1, ... n **do**
  for *item* in buffer_k **do**
    $p(item) = \text{LHT}_k(item)/\text{GHT}_k(item)$
  $\text{r-buffer}_k \leftarrow \text{reorganize}(\text{buffer}_k, p)$
  $\text{IHUP}^k_{aruq} \leftarrow create\_tree(\text{r-buffer}_k)$
return LHTs, GHT, IHUP_{aruq}

Supposing the incremental database for bucket m is denoted as db'_m, and we could scan db'_m to count the number of each item i as N'_i, also we could resort to LHT and GHT for item i appeared in whole database and bucket m represented as N and N_i respectively. So the new external utility for item i in bucket m could be updated as follows:

$$p'_m(i) = \alpha \frac{N_i + N'_i}{N + N'_i} \tag{2}$$

After updating new external utility for each item in bucket m, we could got new utility descending order, based on which we could compare with the old one to find those pairs that need to be exchanged through bubble sorting algorithm. Then we could update information in LHT_m, GHT and adjust those exchange required paths in IHUP_{aruq} tree whose pairs need to be exchanged to keep in new utility descending order. Finally, after updating IHUP_{aruq} tree, we could discover all itemsets whose utility is higher than user specified threshold through depth first search method, and these itemsets are final penetration semantic knowledge which owns high utility. The detail of IHUP_{aruq} update algorithm is shown in Algorithm 2.

Algorithm 2. IHUP_{aruq} update algorithm

Input: incremental database db_k^+, LHT_k, GHT, IHUP_{aruq}^k
Output: output u-GHT, u-LHT_k, u-IHUP_{aruq}^k

for *item* in db_k^+ **do**
  $N_{item} \leftarrow \text{count}(db_k^+, item)$
  $p'(item) \leftarrow (\text{LHT}(item) + N_{item})/(\text{GHT}(item) + N_{item})$
  $\text{GHT}(item) \leftarrow \text{GHT}(item) + \text{count}(item)$
  $\text{LHT}_k(item) \leftarrow \text{LHT}_k(item) + \text{count}(item)$
$pairs \leftarrow \text{exchange}(p', p)$
r-$\text{db}_k \leftarrow \text{reorganize}(\text{db}_k, p')$
for *path* in IHUP_{aruq}^k **do**
  for $< n_1, n_2 >$ in *pairs* **do**
    $path \leftarrow \text{adjust}(path, < n_1, n_2 >)$
$\text{IHUP}_{aruq}^k \leftarrow update\_tree(\text{r-db}_k)$
return LHT_k, GHT, IHUP_{aruq}^k

4 Experiment

In this section, we verified the effectiveness of our proposed adaptive utility quantification strategy by comparing high utility itemsets mining algorithms with three classical frequent itemsets mining algorithms on four datasets. The experiment aim is to illustrate that our proposed strategy could quantify item utility effectively to make high utility itemsets mining algorithms available in mining penetration semantic knowledge.

4.1 Metric and Datasets

Experiment datasets are gathered through penetration testing on four common services, including Apache, IIS, MySQL and nginx. The experiment compares

performance of four high utility itemsets mining algorithms, Two-Phase, FHM, EFIM and HUI-Miner with proposed adaptive utility quantification strategy. Also, there are three frequent itemsets mining algorithms, Apriori [19], LCM-Freq [20] and PrePost+ [21] are implemented to compare performance. The threshold of algorithms ranges from 0.1 to 0.9. The metric adopted in the experiment is true positive ratio (TPR) and false positive ratio (FPR) whose calculation formula is shown as follows:

$$TPR = \frac{|D \cap P|}{|P|} \times 100\%, \quad FPR = \frac{|D \cap N|}{|N|} \times 100\% \tag{3}$$

where P is set of correct penetration semantic knowledge, N is set of wrong knowledge and D is set of discovered knowledge. ROC (Receiver Operating Characteristic curve) is adopted to integrate TPR and FPR metric to describe the performance of algorithm under different algorithm parameter.

4.2 Result and Analysis

The experiment result is shown in Fig. 2, which describes ROC performance of algorithms on four datasets, among which (a)(c)(e)(g) are intact picture of algorithms and (b)(d)(f)(h) are local detail picture (enlargement of black box) for observing performance of algorithms on datasets. From Fig. 3, we could see that the high utility itemsets mining algorithms, Two Phase, FHM, EFIM and HUI_Miner with proposed ARUQ strategy outperformed comparative ones, the area under curve (AUC) is larger than others, proving the effectiveness of our proposed strategy. Detaily, Fig. 2(a) shows that the high utility itemsets mining algorithms with ARUQ strategy shares similar performance on Apache dataset and the highest ratio reaches 85% while Apriori, LCMFreq and PrePost+ algorithms reaches 22.5% at most which is less than those high utility itemsets mining algorithms with our proposed ARUQ strategy. Also, we could conclude from the performance on the other three datasets that Two Phase algorithm with ARUQ strategy achieves best performance and the true positive rate of all frequent itemsets mining algorithms are less than 40% on the former three dataset. Further, we could see from (b)(d)(f)(h) that apriori algorithm shows good performance than other frequent itemsets mining algorithms in mining penetration semantic knowledge, but still far less than the comparative high utility itemsets mining algorithms with ARUQ strategy. To sum up, we could conclude from the experiment result that high utilities itemsets mining algorithms with proposed ARUQ strategy could mine penetration semantic knowledge from raw dataset effectively.

To better understand the performance of algorithms in mining penetration testing knowledge, we compare the CPU and memory consumption performance of algorithms on Apache dataset, and curves in each subfigure shows the details of algorithms in mining penetration semantic knowledge from Apache dataset. "Bottom points" in each curve is used to differentiate algorithm parameter. From Fig. 3 we could see that high utility itemsets mining algorithms consumes

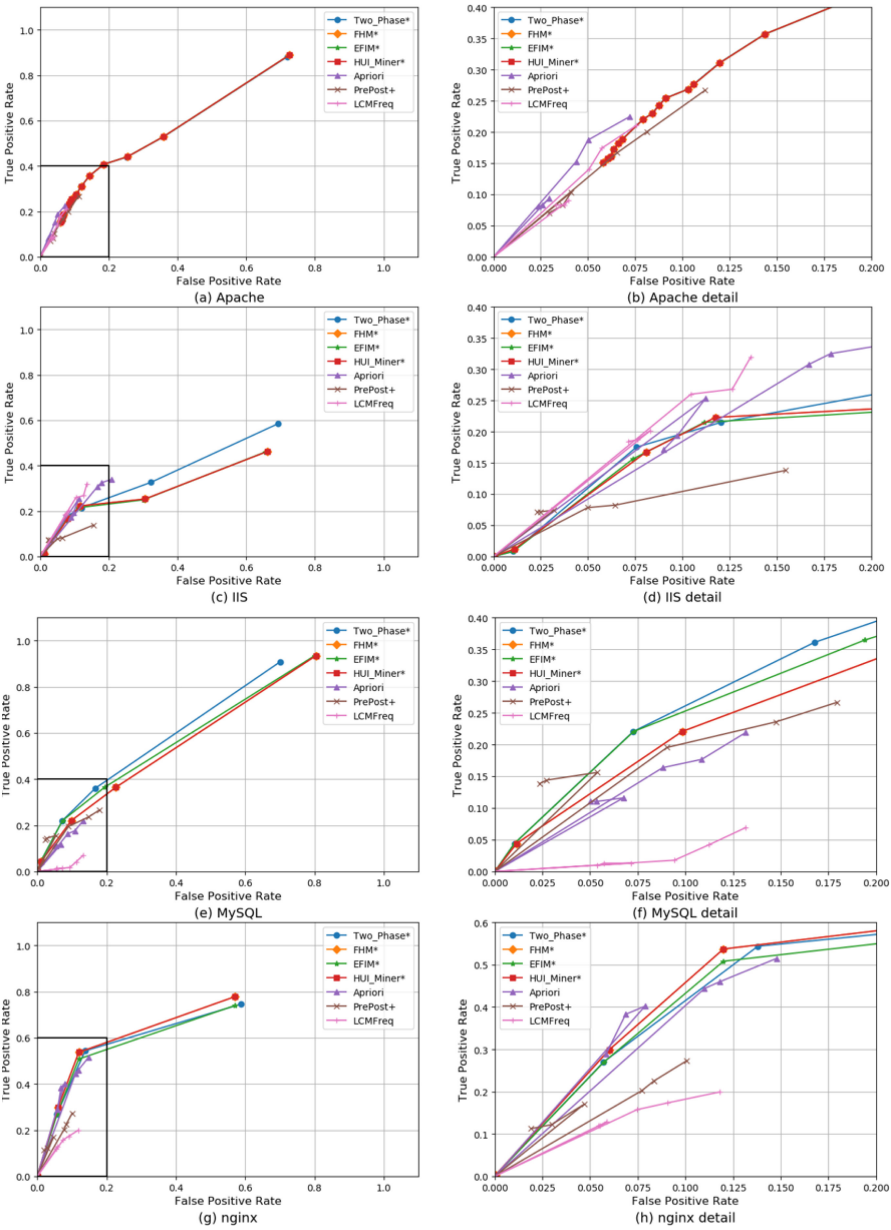

Fig. 2. The receiver operating characteristic curve and locally enlarged receiver operating characteristic curve of algorithms on each experiment dataset.

much CPU time than frequent itemsets mining algorithms in mining penetration semantic knowledge. The average CPU consumption ratio for high utility itemsets mining algorithms with proposed ARUQ strategy is 15000% (overfrequency) while those frequent itemsets mining algorithms are all below 10%. This phenomenon demonstrates that high utility itemsets mining algorithms with proposed ARUQ strategy is much more computation intensive. In contrast, we also could find from Fig. 3 that those high utility itemsets mining algorithms consume far less memory than comparative Apriori, LCMFreq and PrePost+ algorithm because there is no generated candidates for knowledge mining in high utility itemsets mining algorithms, so that they consume similar less memory. In conclusion, the experiment results tell us that high utility itemsets mining algorithms with our proposed adaptive utility quantification strategy outperform frequent itemsets mining algorithm in both accuracy and memory consumption performance.

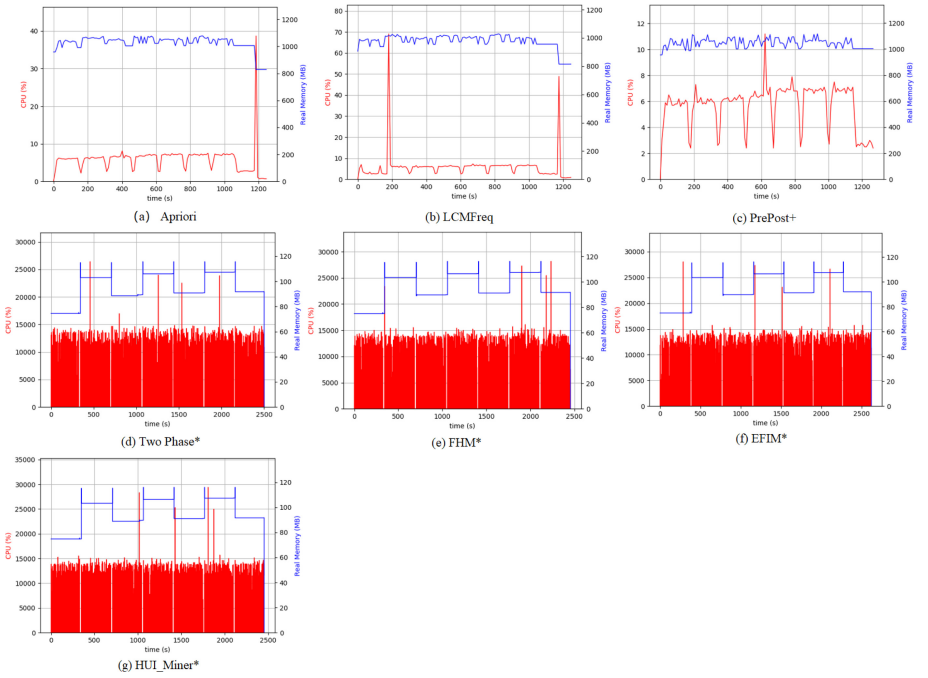

Fig. 3. The comparison of memory and CPU consumption for each data mining algorithm under Apache dataset.

5 Conclusion

In this paper, we have proposed a novel adaptive utility quantification strategy for penetration semantic knowledge mining and construct IHUP$_{aruq}$ tree

structure to maintain utility information. Adaptive utility quantification strategy could quantify external utility for each item effectively but avoid rescanning database which saves a lot of time. Experimental results show that high utility itemsets mining algorithms with adaptive utility quantification strategy achieve significant performance improvement over these algorithms in both accuracy and memory consumption.

References

1. Sarraute, C.: Automated attack planning. arXiv preprint arXiv:1307.7808 (2013)
2. Lyon, G.: Nmap security scanner. Nmap.org [En linea] (2014). http://nmap.org/. Accessed 20 Apr 2015
3. Genge, B., Enachescu, C.: ShoVAT: Shodan-based vulnerability assessment tool for Internet-facing services. Secur. Commun. Netw. **9**(15), 2696–2714 (2016)
4. Zhang, C., et al.: An empirical evaluation of high utility itemset mining algorithms. Expert Syst. Appl. **101**, 91–115 (2018)
5. Gan, W., et al.: A survey of incremental high utility itemset mining. Wiley Interdisc. Rev. Data Min. Knowl. Discov. **8**(2), e1242 (2018)
6. Liu, Y., Liao, W., Choudhary, A.: A two-phase algorithm for fast discovery of high utility itemsets. In: Ho, T.B., Cheung, D., Liu, H. (eds.) PAKDD 2005. LNCS (LNAI), vol. 3518, pp. 689–695. Springer, Heidelberg (2005). https://doi.org/10.1007/11430919_79
7. Ahmed, C.F., et al.: Efficient tree structures for high utility pattern mining in incremental databases. IEEE Trans. Knowl. Data Eng. **21**(12), 1708–1721 (2009)
8. Tseng, V.S., et al.: UP-Growth: an efficient algorithm for high utility itemset mining. In: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2010)
9. Liu, M., Qu, J.: Mining high utility itemsets without candidate generation. In: Proceedings of the 21st ACM International Conference on Information and Knowledge Management (2012)
10. Krishnamoorthy, S.: Pruning strategies for mining high utility itemsets. Expert Syst. Appl. **42**(5), 2371–2381 (2015)
11. Peng, A.Y., Koh, Y.S., Riddle, P.: mHUIMiner: a fast high utility itemset mining algorithm for sparse datasets. In: Kim, J., Shim, K., Cao, L., Lee, J.-G., Lin, X., Moon, Y.-S. (eds.) PAKDD 2017. LNCS (LNAI), vol. 10235, pp. 196–207. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57529-2_16
12. Liu, J., Wang, K., Fung, B.C.M.: Mining high utility patterns in one phase without generating candidates. IEEE Trans. Knowl. Data Eng. **28**(5), 1245–1257 (2015)
13. Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Andreasen, T., Christiansen, H., Cubero, J.-C., Raś, Z.W. (eds.) ISMIS 2014. LNCS (LNAI), vol. 8502, pp. 83–92. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08326-1_9
14. Zida, S., Fournier-Viger, P., Lin, J.C.W., Wu, C.W., Tseng, V.S.: EFIM: a highly efficient algorithm for high-utility itemset mining. In: Sidorov, G., Galicia-Haro, S. (eds.) MICAI 2015. LNCS, vol. 9413, pp. 530–546. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-27060-9_44
15. Duong, Q.-H., et al.: Efficient high utility itemset mining using buffered utility-lists. Appl. Intell. **48**(7), 1859–1877 (2018). https://doi.org/10.1007/s10489-017-1057-2

16. Kannimuthu, S., Premalatha, K.: Discovery of high utility itemsets using genetic algorithm. Int. J. Eng. Technol. (IJET) **5**(6), 4866–4880 (2013)
17. Lin, J.C.-W., et al.: Mining high-utility itemsets based on particle swarm optimization. Eng. Appl. Artif. Intell. **55**, 320–330 (2016)
18. Wu, J.M.-T., Zhan, J., Lin, J.C.-W.: An ACO-based approach to mine high-utility itemsets. Knowl. Based Syst. **116**, 102–113 (2017)
19. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of the 20th International Conference on Very Large Data Bases, VLDB, vol. 1215, pp. 487–499 (1994)
20. Uno, T., Asai, T., Uchida, Y., et al.: LCM: an efficient algorithm for enumerating frequent closed item sets. In: Workshop on Frequent Itemset Mining Implementations, FIMI 2003. DBLP (2003)
21. Deng, Z.H., Lv, S.L.: PrePost+: an efficient N-lists-based algorithm for mining frequent itemsets via Children–Parent Equivalence pruning. Expert Syst. Appl. **42**(13), 5424–5432 (2015)