Information Sciences 624 (2023) 529–553

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Efficient mining of top-*k* high utility itemsets through genetic algorithms

José María Luna [a], Rage Uday Kiran [b], Philippe Fournier-Viger [c], Sebastián Ventura [a,*]

[a] *Department of Computer Science and Numerical Analysis, Andalusian Research Institute in Data Science and Computational Intelligence (DaSCI), University of Cordoba, 14071 Córdoba, Spain*
[b] *University of Aizu, Japan*
[c] *Shenzhen University, Shenzhen, China*

A R T I C L E   I N F O

Article history:
Received 22 March 2022
Received in revised form 17 November 2022
Accepted 29 December 2022
Available online 3 January 2023

Keywords:
High utility itemsets
Top-*k* utility mining
Genetic algorithms

A B S T R A C T

Mining high utility itemsets is an emerging and very active research area in data mining. The goal is to mine all itemsets with a utility value, in terms of importance to the user, no less than a predefined threshold value. Setting an appropriate threshold value is not trivial, requiring not only multiple trials but also the know-how in the application field. The advantage of algorithms for mining top-*k* high utility itemsets is they do not require such a utility threshold, but they suffer from very long runtimes and large memory requirements when large input data is considered. We propose a new genetic algorithm for mining top-*k* high utility itemsets, named TKHUIM-GA (Top-K High Utility Itemset Mining through Genetic Algorithms). It guides the search process by considering the utility of each item to produce initial solutions and to combine solutions accordingly, reducing the runtime and memory consumption as a result. A highly efficient data representation is utilized to reduce memory usage and runtime. A key advantage of TKHUIM-GA is that it works on positive, negative, integer and real unit utility values unlike existing approaches. Experiments on popular benchmark datasets demonstrate the high performance of the proposal regarding the state-of-the-art algorithms.

1. Introduction

Data analytic systems are booming thanks to their capacity to discover usable knowledge that is hidden in data. Frequent Itemset Mining (FIM) is an esentail task of data analysis or data mining [22]. It extracts frequently occurring events, patterns or items (a single item or a group of items) from data [1]. High Utility Itemset Mining (HUIM) [6] is an emerging research field that was proposed to overcome a manifest limitation of FIM [22], that is, any pattern is equally important in a database. In HUIM the aim is to extract patterns with a high utility in terms of importance to the user [11]. The utility function might be defined from many different criteria, but it was originally proposed to quantify the profit generated by the sale of an item in retail data [7]. HUIM has been mainly considered as a generalization of the FIM task where each item from the input data is associated with a value that represents its importance or weight in the problem at hand. An additional feature of HUIM is that any item may appear multiple times in one transaction, unlike FIM that simply denotes whether each item appears or not in every transaction.

The first HUIM approaches [6] were published in the early 2000s aiming to produce valuable and more actionable insights. Since then, the task has been considered in many application fields such as market basket analysis [7] or sentiment

* Corresponding author.

https://doi.org/10.1016/j.ins.2022.12.092
0020-0255/© 2022 Elsevier Inc. All rights reserved.

analysis [6], to name but two. Currently, HUIM is a very active research area that includes dozens of efficient algorithms [10,16,20,21] to enumerate all itemsets that yield a utility higher than a user-predefined threshold. Some algorithms [21] include a level-wise candidate generation-and-test method, whereas others [2,32] are based on pattern growth. However, these exact HUIM algorithms may suffer from very long runtimes when working on large search spaces. More precisely, exact approaches tend to slow down as the input data size increases in terms of the number of transactions and, especially, in the sum of different items [6]. As a result, the performance becomes unacceptable for users who cannot wait that long to obtain results. This performance bottleneck of exact HUIM approaches is the reason for the use of bio-inspired optimization approaches, including genetic algorithms [13] and particle swarm optimization [17].

Most of the HUIM algorithms, either exact or bio-inspired, require a minimum utility threshold value to be predefined. To determine the exact value, however, is not trivial, and generally requires a profound background in the application field. Inexpert and many expert users need to try different thresholds by guessing and re-executing the algorithms time and again until results are suitable for them. *Wu et al.* [37] demonstrated that a small change in the threshold value may lead to very few (even zero) or an extremely large set of solutions (a filtering process is then needed), and significantly long execution times. Thus, it may be essential to simply focus on mining the top-k solutions, which is the recent research agenda in HUIM [6]. Users only need to specify the number of desired itemsets, which is more intuitive than setting the threshold and does not depend on database characteristics. The literature includes efficient exact algorithms for mining top-k utility itemsets such as TKU (mining Top-K Utility itemsets) [33] and TKO (mining Top-K utility itemsets in One phase) [33]. Recently, some heuristic-based solutions were proposed such as TKU-CE [30] and TKU-CE+ [31]. However, there are many challenges still to be addressed:

- A challenge for designing efficient top-k HUIM algorithms is that the utility is not monotone nor anti-monotone. Thus, the utility itself cannot be used to reduce the search space. To cope with this issue, upper bounds on the utility [39] have been designed and used in HUIM and top-k HUIM algorithms but these algorithms still have long runtimes and high memory consumption. The use of metaheuristics for mining top-k HUIM may enable us to guide the search process efficiently through promising solutions, without the need for search space pruning.
- Top-k HUIM algorithms and most bio-inspired HUIM algorithms cannot deal with negative and real unit profits. Instead, they work with positive integer unit profits. Thus, proposing a bio-inspired approach for mining top-k utility solutions on different scenarios (positive, negative, real, integer) is an important point in the field.
- HUIM does not follow FIM foundations and the utility of an itemset may be higher or lower than its supersets and subsets. Thus, combining two itemsets to form a new one needs a good strategy that selects the appropriate items to be exchanged. The idea is to add or remove appropriate items to produce better utility values.
- Producing better solutions in HUIM is not trivial, mainly due to the absence of the downward closure property [22]. Thus, any heuristic search process needs to start with a good set of initial solutions. Random initial solutions, on the contrary, may complicate the subsequent searching procedure and slow down the runtime. Here, the use of transaction utility (TU) may be key to producing good initial solutions. Transactions with a high TU are likely to form an initial set of solutions.

In this paper, we address all of the above challenges by proposing a novel genetic algorithm, known as TKHUIM-GA (Top-K High Utility Itemset Mining through Genetic Algorithms), for mining top-k high utility itemsets, where k is the desired number of high utility itemsets to be extracted. The novelty of this paper is summarized as follows: a few parameters bio-inspired algorithm for mining top-k high utility itemsets (the first bio-inspired algorithm for mining top-k utility items); a novel data representation to reduce the runtime and memory requirements; feasibility to work on any type of items (positive, negative, real, integer); a novel initialization based on transaction utility unlike existing approaches that randomly initialize the population; specifically designed genetic operators that auto-tune their probability values based on the convergence of the evolutionary process. We conducted different kinds of experiments on a varied set of eighteen widely known HUIM datasets to evaluate the performance of the proposal together with the effectiveness of the designed strategies. Empirical results show that the results achieved by the proposal are close to that of exact and bio-inspired algorithms for mining high utility itemsets but in a much lower runtime and less memory usage.

The remainder of this paper is organized as follows. Section 2 introduces the background and related work of top-k HUIM. Section 3 presents the proposed algorithm. Experimental results are reported in Section 4. Finally, some conclusions are drawn and future work is discussed in Section 5.

2. Preliminaries and Related Work

This section describes the top-k HUIM problem and presents related studies.

2.1. Top-k High Utility Itemset Mining

A pattern or itemset X is formally defined as a set of elements, events or items $X \subseteq \{I_1, \ldots, I_n\}$ that regularly co-occur in a database Ω and describes valuable features of data. A database Ω is a set of data records or transactions $\{T_1, T_2, \ldots, T_m\} \in \Omega$, and every transaction T_y includes a subset of items $T_y \subseteq \{I_1, \ldots, I_n\}$ and has a unique identifier y $(1 \leqslant y \leqslant m)$. Hence, each item I_x appears in a subset of transactions $\{T_i, T_j, \ldots, T_z\} \in \Omega$, that is, $I_x \in T_y \wedge T_y \in \{T_i, T_j, \ldots, T_z\}$. Conversely, given the finite set of distinct items $I^* = \{I_1, I_2, \ldots, I_n\}$, then $T_y \subseteq I^*$.

HUIM aims to discover patterns in a quantitative transactional database Ω, where the quantities of items in transactions are provided together with the weights or relative importance of each item to the user [6]. Formally, each item $I_x \in T_y$ has a positive number $q(I_x, T_y)$, called its internal utility, that represents the quantity of I_x in T_y. An item that does not appear in a transaction will produce an internal utility value of 0, that is, $q(I_x, T_y) = 0$ when $I_x \notin T_y$. Each item $I_x \in T_y$ also has another positive number $p(I_x)$, called its external utility, that represents the unit profit value or relative importance of I_X and which does not depend on T_y. Let Table 1 be an example database containing five transactions. Each transaction T_y is formed by items and quantities or internal utility values. Thus, the internal utility of item a in T_1 is $q(a, T_1) = 1$, whereas $q(d, T_1) = 0$. Finally, Table 2 represents the external utility value for each item, that is, its relative importance to the user.

The utility of an item I_x in a transaction T_y is calculated as the product of the internal utility of I_x in T_y and the external utility of I_x. Formally, it is denoted as $U(I_x, T_y) = q(I_x, T_y) \times p(I_x)$. Similarly, given an itemset $X \subseteq T_y$, then $U(X, T_y) = \sum_{I_x \in X} U(I_x, T_y)$ if $X \subseteq T_y$. Otherwise, $U(X, T_y) = 0$. Finally, the utility of an itemset X in a dataset Ω is calculated as $U(X, \Omega) = \sum_{T_y \in \Omega, X \subseteq T_y} \sum_{I_x \in X} U(I_x, T_y)$. For example, given the itemset $X = \{a, c\}$ defined in Tables 1 and 2, the utility of each item in T_1 is $U(a, T_1) = 1 \times 5 = 5$ and $U(c, T_1) = 2 \times 2 = 4$. The utility of X in T_1 is $U(X, T_1) = 5 + 4 = 9$. The utility of X in Ω is $U(X, \Omega) = 9 + 9 = 18$.

In HUIM, the transactions play an essential role and it is crucial to define their utility values. The transaction utility (TU) of a transaction T_y is calculated as the utility of the itemset that forms such a transaction. Formally, given $X \equiv T_y$, then $TU(T_y) = U(X, T_y)$. For example, given the transaction T_1 defined in Table 1, then $X \equiv T_y \equiv \{a, b, c\}$ $TU(T_1) = U(X, T_1) = U(a, T_1) + U(b, T_1) + U(c, T_1) = 5 + 30 + 4 = 39$.

Finally, an itemset X is called a high utility itemset (HUI) if and only if the utility value is higher than a predefined threshold value α given by the user, that is, $U(X, \Omega) \geqslant \alpha$. Otherwise, it is denoted as a low utility itemset. HUIM aims to obtain the complete set of HUIs. Taking the sample dataset shown in Tables 1 and 2, the set of HUIs given $\alpha = 30$ is: $X_1 = \{b\}, X_2 = \{a, b\}, X_3 = \{b, c\}$, and $X_4 = \{a, b, c\}$. The utility values for these itemsets are the following: $U(X_1, \Omega) = 40, U(X_2, \Omega) = 55, U(X_3, \Omega) = 48$, and $U(X_4, \Omega) = 39$.

An important challenge of HUIM is that the anti-monotone property cannot be directly used to prune the search space since the utility of an itemset can be equal to, higher or lower than that of its supersets and subsets. Back to the same dataset shown in Tables 1 and 2, the utility of $\{a\}$ is 20 and the utility of its supersets $\{a, c\}$ and $\{a, b\}$ may be lower or higher than 20: $U(\{a, c\}, \Omega) = 18$ and $U(\{a, b\}, \Omega) = 55$. This situation is specially noticeable when the aim is to extract the top-k HUI, giving rise to an even more challenging task [4]. An itemset X is called a top-k HUI in Ω if and only if it is not possible to find more than k itemsets with a utility larger than X. Note that the top-k HUIM may obtain more than k itemsets if there are itemsets with the same utility value and less than k for large values of k.

2.2. Related Work

A recent survey on high utility itemset mining may be found at [9]. Existing HUIM algorithms can be generally categorized into two main groups [33]: two-phase and one-phase algorithms. The first group includes algorithms such as IHUP [2], IIDS [15], UP-Growth [34] that perform a candidate generation step first, and then they calculate the utility of each potential candidate. On the contrary, the second group includes algorithms such as HU-FIMi [14], EFIM [41],d^{2}HUP [19] and HUI-Miner [20], that discover HUI in a single step. It is possible to consider in this last group bio-inspired approaches (e.g. HUIM-GA [13], HUIM-BPSO [18], HUIM-GA-tree [18], HUIM-BPSO-tree [18], HUIF-PSO [27], HUIF-GA [27], HUIF-BA [27], and HUIM-ABC [26]), which were proposed to avoid the performance degradation as the size of the database and the number of distinct items increase. Some additional bio-inspired algorithms have been recently proposed so they are not described in [9]: HUIM-AF [29], HUIM-HC [23], HUIM-SA [23], and HUIM-SPSO [28]. Additionally, some evolutionary approaches were proposed and modeled as a multi-objective problem for the task of mining frequent and high utility itemsets: MOEA-FHUI [38], ISR-MOEA [40], and CP-MOEA [3]. These algorithms do not require any minimum support and utility threshold value and they return the a set of solutions with a good trade-off between utility and support. Additionally, there is

Table 1
Sample quantitative transactional dataset.

TID	Transaction
T_1	$(a, 1), (b, 3), (c, 2)$
T_2	$(b, 1), (c, 2)$
T_3	$(a, 1), (c, 2), (d, 4)$
T_4	$(a, 2), (b, 1), (d, 2)$

Table 2
External utility values.

Items	a	b	c	d
External utility	5	10	2	1

a recent research work [12] that presents the extraction of top-k high utility itemsets using propositional satisfiability and a number of well-known established techniques for SAT-based problem solving.

At this point, it is interesting to analyse characteristics of existing bio-inspired algorithms. Table 3 illustrates a comparison of these algorithms, considering the heuristic type, the data representation, the number of parameters, the number of solutions to return, and if they require a minimum threshold value. All the listed algorithms return an undefined number of solutions, which may turn in an arduous process for the end-user to filter solutions out. Additionally, any of the listed above algorithms require a minimum utility threshold value to be predefined, which is not a trivial issue. As observed by *Wu et al.* [37], even a small change in the threshold value may lead to very few solutions. Such a small change may also produce an extremely large set of itemsets that will require a filtering process. Last but not least, any small variation in the utility threshold may be responsible for long execution times. To avoid the process of determining an appropriate threshold value, recent solutions are focused on mining the top-k HUI solutions [6]. In this task, users only need to specify the number of desired itemsets.

Similarly to HUIM, top-k HUIM algorithms can be categorized into two main groups [6]: two-phase and one-phase algorithms. The two-phase approaches first generate top-k HUI candidates, and a second phase is then responsible for mining relevant top-k HUIs. TKU [37] is one of the first proposed top-k HUIM algorithms to work on transactional databases. A few years later, *Ryang et al.* [24] proposed the REPT algorithm, which was proposed as an improvement of TKU [37]. Recently, *Tseng et al.* [33] proposed an improvement of their already described TKU algorithm [37], giving rise to the most efficient algorithm for mining top-k HUI in two phases. On the other hand, one-phase approaches directly extract the top-k HUIs, so no previous candidate generation step is required. In this category, TKO [33] and kHMC [4] are the two most important approaches for mining top-k HUI from transactional databases. The performance of these two efficient algorithms is already analyzed in [6] considering both dense and sparse datasets. It was proved that kHMC works well on sparse data, whereas TKO performs better on dense data. However, in general terms, both algorithms are quite similar, and few differences can be found. Finally, it is important to remark that although the above studies may perform well, they are exact approaches and tend to slow down as the input data size increases. Recently, two heuristic-based approaches (TKU-CE [30] and TKU-CE+ [31]) were proposed for mining top-k high utility itemsets. TKU-CE [30] is based on a cross-entropy method and a bitmap cover representation where each bit represents a different transaction in the database. The proposed procedure iteratively works by updating a probability vector that denotes the probability of selecting each item from the database. Once a stopping criterion is met, the algorithm returns the final probability vector. TKU-CE+ [31] was proposed one year later to improve TKU-CE [30]. Authors considered just the k items with the highest utility values to reduce memory usage. This algorithm also uses just high utility values to generate new itemsets in each iteration, reducing the search space. Finally, in order to add diversity, some randomly generated itemsets are produced. To the best of our knowledge, no bio-inspired approach for mining top-k HUIs has been proposed.

3. The TKHUIM-GA Algorithm

This section describes the data representation of the original database, how each solution is represented in the proposed genetic algorithm, and each of the procedures of the proposed TKHUIM-GA algorithm.

3.1. Data representation

In the proposed approach, the original database is stored in two different data representations (vertical and horizontal) while data are being read, transaction by transaction. The idea behind these data representations is to provide fast data

Table 3
Analysis of well-known heuristic-based algorithms.

Algorithm	Reference	Year	Heuristic	Data representation	#Parameters	#Solutions	Min. threshold
HUIM-GA	[13]	2014	Genetic Algorithm	Bit map	5	Unknown	Yes
HUIM-BPSO	[18]	2016	Particle Swarm Optimization	Transactional dataset	3	Unknown	Yes
HUIM-GA-tree	[18]	2016	Genetic Algorithm	Tree structure	5	Unknown	Yes
HUIM-BPSO-tree	[18]	2016	Particle Swarm Optimization	Tree structure	3	Unknown	Yes
HUIF-PSO	[27]	2018	Particle Swarm Optimization	Bit map	3	Unknown	Yes
HUIF-GA	[27]	2018	Genetic Algorithm	Bit map	3	Unknown	Yes
HUIF-BA	[27]	2018	Bat-inspired	Bit map	3	Unknown	Yes
HUIM-ABC	[26]	2018	Artificial Bee Colony	Bit map	3	Unknown	Yes
MOEA-FHUI	[38]	2018	Multi-Objective Optimization	Bit map	6	Unknown	No
ISR-MOEA	[40]	2019	Multi-Objective Optimization	Encoding vector	5	Unknown	No
CP-MOEA	[3]	2019	Multi-Objective Optimization	Bit map	4	Unknown	No
HUIM-SPSO	[28]	2020	Particle Swarm Optimization	Bit map	3	Unknown	Yes
TKU-CE	[30]	2020	Cross-entropy method	Bit map	3	Top-k	No
HUIM-AF	[29]	2021	Artificial Fish Swarm	Transactional dataset	4	Unknown	Yes
HUIM-HC	[23]	2022	Hill Climbing	Bit map/Encoding Vector	3	Unknown	Yes
HUIM-SA	[23]	2022	Simulated Annealing	Bit map/Encoding Vector	5	Unknown	Yes
TKU-CE+	[31]	2021	Cross-entropy method	Bit map	3	Top-k	No

access, and to avoid useless information being kept as in a binary data representation used by existing heuristic-based HUIM algorithms [27], which needs to keep a 0 value if an element is not in a specific transaction, 1 otherwise.

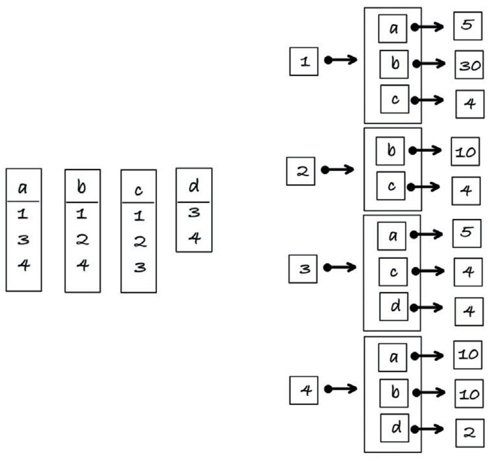
First, vertical data representation creates a list of indices of transactions in which each item appears. The length of each list denotes the support of such an item, so it is quite fast to compute the frequency of each item. Formally, given an item I_x, the list of indices is defined as $L(I_x) = \{i \in \mathbb{N} : I_x \in T_i\}$ being T_i any transaction in the dataset Ω, and $\mathbb{N}$ used to denote the set of all natural numbers. The support or frequency of I_x is denoted as $support(I_x) = |L(I_x)|$. Additionally, the support of an itemset X is defined as $support(X) = |\bigcap_{I_x \in X} L(I_x)|$, that is, the length of the resulting list after intersecting the n lists of indices from X. Additionally, this vertical data representation enables fast access to utilities (product of internal and external utilities) associated with each transaction thanks to the indices of the resulting list. Second, horizontal data representation is based on a hashing function. Given a key k based on the index of a transaction in data, it maps k to the corresponding bucket including information of items belonging to the k-th transaction. Such a bucket is, at the same time, another hashing function in which $k\prime$ is the item that maps to the corresponding final product of internal and external utilities. It is important to highlight that real and negative utility values are also feasible with this representation and it does not require any additional consideration.

Fig. 1 illustrates the data representation utilized by TKHUIM-GA for the sample transactional dataset shown in Table 1. The vertical representation (see Fig. 1(a)) includes four different lists of indices, one per item in the data. Here, it is easy to obtain that the item a appears three times: first, third and fourth transactions. Considering the itemset $\{a, b\}$, the resulting list of indices (intersection of the lists belonging to a and b) is formed by the first and fourth transactions. Hence, the support of that itemset is equal to the length of its list, that is, 2. Similarly, the horizontal data representation (see Fig. 1(b)) is responsible for storing the utilities based on Tables 1 and 2. Computing the utility of the itemset $\{a, b\}$ just needs to index the elements from the resulting list of indices, that is, first and fourth transactions. Index 1 (first transaction) is then indexed by keys a and b, so the utility of this index is equal to the sum of the values returned by these keys, that is, $5 + 30 = 35$. The same process is repeated for index 4, returning $10 + 10 = 20$. The final utility is the sum of all the utilities of all the indices, that is, $35 + 20 = 55$.

3.2. Encoding criterion

Existing heuristic-based approaches [27] use a fixed-length chromosome (vector of values), which is composed of 0s or 1s corresponding to whether an item is absent or present in the individual. The proposed algorithm also uses an encoding vector to represent each individual or solution to the problem. Nevertheless, this proposal does work on fixed-length chromosomes, but each chromosome includes just the items that belong to the represented individual. For extremely large datasets, in terms of the number of different items, this encoding criterion is faster and requires less memory consumption. Additionally, the number of items belonging to that individual is equal to the vector size. As a matter of clarification, $[a, b]$ is a sample individual encoded by considering the sample dataset shown in Tables 1 and 2. The same individual is represented as $[1, 1, 0, 0]$ when considering a fixed-length chromosome as the encoding followed in [27]. Both encoding criteria represent the same individual, but the proposed encoding is simpler since it requires half of the items in this example.

To determine the transactions that satisfy this sample individual, it only has to take each item and point to the corresponding list of indices (vertical data representation). Fig. 2 shows how the previous sample individual points to the specific list of indices for the sample transactional database shown in Tables 1 and 2. An intersection of all these lists of indices pro-



(a) Vertical representation (b) Horizontal representation

Fig. 1. Data representations of the proposed TKHUIM-GA algorithm.

duces the resulting list of indices for that individual (vertical data representation). Note that this encoding works for any utility value (positive, negative, integer and real) because it just considers the items.

3.3. Algorithm

The proposed TKHUIM-GA algorithm comprises three main methods or procedures, which were properly designed to the HUIM task. Descriptions of all these procedures as well as how they are combined to form the whole algorithm can be found below.

Algorithm 1: Producing the initial solutions

```
Algorithm 1 Producing the initial solutions
Require: $\Omega, n, m$                          ▷ Dataset $\Omega$, and $n$ solutions of maximum size $m$
Ensure: $\mathcal{P}$
 1: $\mathcal{P} \leftarrow \emptyset$
 2: for $T_y \in \Omega$ do                       ▷ It analyses every transaction in $\Omega$
 3:     $u \leftarrow TU(T_y)$
 4:     $X \leftarrow$ takes the $m$ items with the highest utility from $T_y$
 5:     if size of $\mathcal{P} < n$ then
 6:         $\mathcal{P} \leftarrow \mathcal{P} \cup X$
 7:     else
 8:         if $u >$ the lowest utility in $\mathcal{P}$ then   ▷ Solutions are stored based on the the transaction utility
 9:             $\mathcal{P} \leftarrow \mathcal{P} \setminus$ itemset with the lowest utility in $\mathcal{P}$
10:             $\mathcal{P} \leftarrow \mathcal{P} \cup X$
11:         end if
12:     end if
13: end for
14: return $\mathcal{P}$                              ▷ Set of $n$ good solutions
```

1. **Initial solutions**. The proposed TKHUIM-GA algorithm does not randomly initialize the initial population as in other bio-inspired approaches [27]. Instead, TKHUIM-GA starts with a good set of solutions to evolve. The main reason behind this idea is to avoid useless generations in which, probably, solutions are not present in the data. In this regard, the proposed approach checks the *TU* or transaction utility value of each transaction while the dataset Ω is being read. To form a population of *n* solutions, those *n* transactions with the best *TU* values are kept. It is important to remind that the *TU* value of a transaction is calculated as the utility of the itemset that forms such a transaction. Thus, the hypothesis behind this

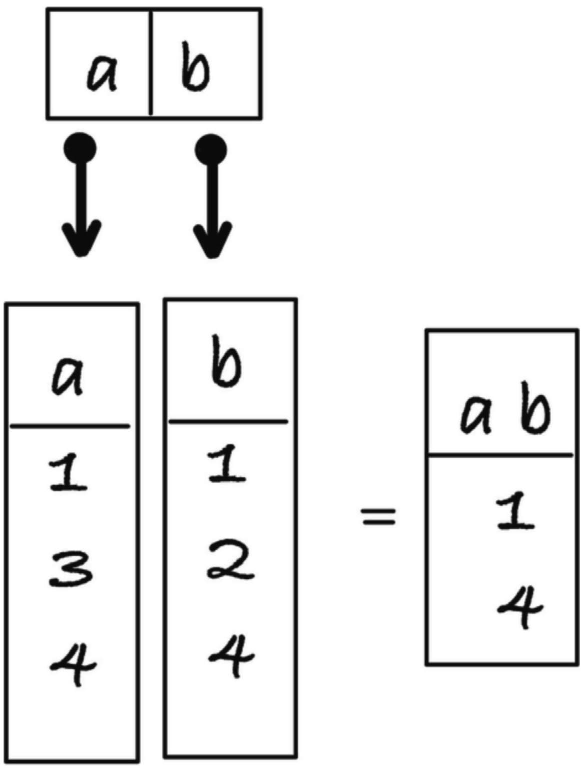


Fig. 2. Encoding criterion of the proposed TKHUIM-GA algorithm, representing an individual that describes the itemset $\{a, b\}$.

procedure is that high *TU* values are probable to produce high utility itemsets. Last but not least, if the user requires solutions with a maximum length m, then the procedure of generating the initial set of solutions takes the best m items from the selected transaction. In other words, with a good transaction based on its *TU* value, those m items with the highest utility are taken from it. Additionally, the value m is a maximum length value, so in those situations where m is higher than the number of items in the selected transaction, then the whole transaction is taken. The only requirement for this procedure is that n should be lower or equal to the number of transactions in Ω. The pseudocode of this procedure is shown in Algorithm 1. Note that this procedure works for any utility value (positive, negative, integer and real) because the *TU* formula is independent of the utility value type. In other words, the sum and product of either positive, negative, integer or real numbers can be equally done.

As a matter of clarification, let us consider the sample transactional dataset shown in Tables 1 and 2. Let us also fix the values $n = 2$ and $m = 2$. In this example, $TU(T_1) = U(a, T_1) + U(b, T_1) + U(c, T_1) = 5 + 30 + 4 = 39$. The m items with the highest utility are taken to form T_1 to form an itemset X, that is, $X = \{a, b\}$. $\mathscr{P}$ is empty at this point, so the itemset X is added to $\mathscr{P}$. Next, $TU(T_2) = U(b, T_2) + U(c, T_2) = 10 + 4 = 14$ and $T_2 \equiv X = \{b, c\}$ this time because the size of T_2 is equal to $m = 2$. Additionally, because the size of $\mathscr{P}$ is smaller than $n = 2, X$ is added to $\mathscr{P}$. At this point the set of solutions is $\mathscr{P} = \{\{a, b\}, \{b, c\}\}$. Next, $TU(T_3) = U(a, T_3) + U(c, T_3) + U(d, T_3) = 5 + 4 + 4 = 13$, and $X = \{a, c\}$. However, adding X to $\mathscr{P}$ requires to remove a worse solution considering the *TU* value. $TU(T_3)$ is not better than $TU(T_1)$ or $TU(T_2)$ so it is discarded. Finally, $TU(T_4) = U(a, T_4) + U(b, T_4) + U(d, T_4) = 10 + 10 + 2 = 22$. The m items with the highest utility are taken from T_4 to form the itemset $X = \{a, b\}$. The lowest *TU* value from $\mathscr{P}$ is $TU(T_2)$, which is lower than $TU(T_4)$. Hence, $TU(T_2)$ is replaced by $TU(T_4)$ to form the new $\mathscr{P}$. As a result, $\mathscr{P}$ includes the most promising transactions according to their *TU* values.

2. **Evaluation procedure**. This procedure is responsible for assigning a fitness value F (see Eq. 1) to each individual or solution that represents an itemset X. The fitness function evaluates how close a given solution is to the optimal solution on a dataset Ω. This optimal solution is the best feasible solution to the problem, and the one the genetic algorithm aims to obtain. In the proposed approach, $F(X)$ is formally defined as the utility of X on the dataset Ω, considering positive, negative, integer and real utility values. As it was explained in Section 2.1, the utility value of an itemset X is defined as the sum of the utility values of all its individual items in a dataset Ω. An item that does not appear in a transaction will produce an internal utility value of 0, that is, $q(I_x, T_y) = 0$ when $I_x \notin T_y$. Finally, it is important to remark that the evaluation procedure benefits from the proposed data representation previously described. Additionally, any utility value (positive, negative, integer and real) because the *TU* formula is independent of the utility value type. The sum and product of either positive, negative, integer or real numbers can be equally done.

$$F(X) = U(X, \Omega) \tag{1}$$

As a matter of clarification, let us consider the sample individual (itemset) $X = \{a, d\}$ obtained from in Tables 1 and 2. The fitness value is computed as $F(X) = U(X, T_3) + U(X, T_4)$. As a result, the fitness value is $F(X) = (1 \times 5 + 4 \times 1) + (2 \times 5 + 2 \times 1) = 9 + 12 = 21$.

Algorithm 2: Genetic operators

```
Algorithm 2 Genetic operators
Require: $\mathcal{S}, \alpha, \beta$        ▷ Set $\mathcal{S}$ of individual. $\alpha$ and $\beta$ are probabilities for the operators
Ensure: $\mathcal{P}$
 1: $\mathcal{P} \leftarrow \emptyset$
 2: for all $X_i, X_j \in \mathcal{S}$ do        ▷ Takes itemsets (individuals) two by two in order from $\mathcal{S}$
                                                  ▷ Crossover genetic operator procedure
 3:     if $\alpha >$ random value then
 4:         if $F(X_i) > F(X_j)$ then
 5:             $x \leftarrow$ takes the item $I_x \in X_i$ with the lowest utility value
 6:             $y \leftarrow$ takes the item $I_x \in X_j$ with the highest utility value
 7:         else
 8:             $x \leftarrow$ takes the item $I_x \in X_i$ with the highest utility value
 9:             $y \leftarrow$ takes the item $I_x \in X_j$ with the lowest utility value
10:         end if
11:         $X_i \leftarrow X_i \setminus \{x\}$
12:         $X_i \leftarrow X_i \cup \{y\}$
13:         $X_j \leftarrow X_j \setminus \{y\}$
14:         $X_j \leftarrow X_j \cup \{x\}$
15:     end if
                                                  ▷ Mutation genetic operator procedure
16:     for all $X \in \{X_i, X_j\}$ do
17:         if $\beta >$ random value $\in [0, 1]$ then
18:             if $0.5 >$ random value $\in [0, 1]$ then
19:                 $x \leftarrow$ takes the item $I_x \in X$ with the lowest utility value
20:                 $X \leftarrow X \setminus \{x\}$
21:             else
22:                 $x \leftarrow$ roulette wheel selection based on utilities
23:                 $X \leftarrow X \cup \{x\}$
24:             end if
25:         end if
26:         $\mathcal{P} \leftarrow \mathcal{P} \cup X$
27:     end for
28: end for
29: return $\mathcal{P}$                                      ▷ New population
```

3. **Genetic operators**. TKHUIM-GA includes two genetic operators that were specially designed for this aim. One focuses on an intensification phase where the search focuses on examining neighbors (crossover) and the other focuses on a diversification phase that encourages the search process to examine unvisited regions (mutation). In this proposal, the crossover genetic operator combines the genetic information of two individuals to generate offspring. The mutation operator, on the contrary, maintains genetic diversity by adding or removing elements. Algorithm 2 shows how the genetic operators are applied to a set $\mathscr{S}$ of solutions already selected to be modified. Solutions are taken two by two from $\mathscr{S}$ (see Algorithm 2, line 2) and the crossover operator is applied with a probability α (see Algorithm 2, lines 3 to 15). This operator interchanges the item with the lowest utility value from the best individual according to the fitness value, with the item with the highest utility value from the worst individual according to the fitness value. The result is two individuals (itemsets) X_i and X_j interchanging an item. New individuals will have the same length unless the new item was already in such an individual. As a matter of clarification, let us consider two individuals $X_i = \{a,b\}$ and $X_j = \{c,d\}$. Let us also consider the sample transactional dataset and the external utility values shown in Tables 1 and 2. According to the fitness value described in the evaluation procedure, the fitness values for the two sample individuals X_i and X_j are equal to $F(X_i) = 55$ and $F(X_j) = 8$, respectively. Thus, X_i is better than X_j due to $F(X_i) > F(X_j)$. The crossover operator, therefore, will take the item with the lowest utility from X_i and the one with the highest utility from X_j. The utility values from items within X_i are the following: $U(a,\Omega) = 20, U(b,\Omega) = 50$; whereas the utility values from items within X_j are the following: $U(c,\Omega) = 12, U(d,\Omega) = 6$. Hence, item a is swapped for the item c giving rise to two new solutions $X\prime_i = \{b,c\}$ and $X\prime_j = \{a,d\}$. For sure, it is not possible to guarantee that $F(X\prime_i) > F(X_i)$ since it will depend on X_j, but the idea is to increase the chance to satisfy $F(X\prime_i) > F(X_i)$ by changing its item with the lowest value.
   A mutation operator is then applied to $X\prime_i$ and $X\prime_j$ (see Algorithm 2, lines 16 to 27). Here, $X\prime_i$ and $X\prime_j$ can be new solutions or the existing ones (if the α probability was not satisfied). In both cases, two different mutations can be applied, but just one of them is applied (it is randomly chosen). The first one (see Algorithm 2, lines 18 to 20) removes the item with the lowest utility value from the individual. The second one (see Algorithm 2, lines 21 to 23) adds a new item from data with a certain probability. A roulette wheel selection is applied based on utility for every single item in the data. In this case, it may happen that the new item was already in such an individual, so the final individual remains the same. As a matter of clarification, consider the individual $X = \{a,b,d\}$ with a fitness value $F(X) = 22$. As for the first mutation, that is, the item with the lowest utility value is removed, the resulting individual will be $X\prime = \{a,b\}$ since $U(b,\Omega) = 50 > U(a,\Omega) = 20 > U(d,\Omega) = 6$. As a result, the new fitness value will be $F(X\prime) = 55 > F(X) = 22$. Considering the second mutation, a wheel selection is applied based on the utility value of every single item in the data so the probabilities to choose each item are: $P(a) = 22.72\%, P(b) = 56.82\%, P(c) = 13.63\%, P(d) = 6.82\%$. If item b is randomly chosen to be added to X, the resulting individual $X\prime$ will remain the same $X\prime \equiv X$.

Algorithm 3: Proposed TKHUIM-GA algorithm

```
Algorithm 3 Proposed TKHUIM-GA algorithm
Require: $\Omega, n, m, e$        ▷ Dataset $\Omega$, $n$ individuals of maximum size $m$, and $e$ solutions to be returned
Ensure: $\mathcal{E}$
 1: $\mathcal{P} \leftarrow \emptyset$        ▷ Population set
 2: $\mathcal{E} \leftarrow$ takes $e$ singletons with the highest utility values from $\Omega$
 3: $exit \leftarrow False$
 4: $\mathcal{P} \leftarrow$ creates the initial solutions using $\Omega$, $m$ and $n$        ▷ See Algorithm 1
 5: $\mathcal{P} \leftarrow$ evaluates $\mathcal{P}$
 6: do        ▷ Each different loop is a generation
 7:     $\mathcal{S} \leftarrow$ applies tournament selector to $\mathcal{P}$
 8:     $\mathcal{P} \leftarrow$ applies genetic operators on $\mathcal{S}$, and using $\alpha$ and $\beta$        ▷ See Algorithm 2
 9:     $\mathcal{P} \leftarrow$ evaluates $\mathcal{P}$
10:     $\mathcal{E} \leftarrow$ takes the $e$ best solutions from $\mathcal{P} \cup \mathcal{E}$
11:     $\alpha, \beta \leftarrow$ updates the probabilities        ▷ They are increased/reduced by 0.05
12:     if $\mathcal{E}$ is not improving after a number of iterations then
13:         $exit \leftarrow True$
14:     end if
15: while $exit \neq True$
16: return $\mathcal{E}$        ▷ Set with the best $e$ solutions found so far
```

Finally, it is important to combine all the described above procedures to produce the final TKHUIM-GA algorithm (see Algorithm 3). TKHUIM-GA takes the e singletons with the highest utility values from the dataset Ω, and this set will be the first elite population, that is, the elite population at generation 0. The elite population is responsible for keeping the best solutions found so far, so while reading data, the only information we have is the set of singletons. It is in this process of reading data that the initial population is formed (see Algorithm 3, line 4) as it was described previously. Then, the iterative process is performed during an undetermined number of iterations (generations). Unlike most existing bio-inspired pattern mining algorithms [35], which require a predefined set of generations, TKHUIM-GA iteratively works while the elite population is improving. After several iterations without improving, the algorithm stops and returns the elite population or set of best solutions found over the evolutionary process (see Algorithm 3, lines 12 to 16). In each generation, the algorithm creates new solutions based on the already produced solutions. To this aim, TKHUIM-GA selects a subset of solutions to act as parents through a tournament selector procedure, which is widely used in bio-inspired pattern mining algorithms [35]. This subset of solutions is used by the genetic operators to create a set of new solutions that are then evaluated (see Algorithm 3, lines 7 to 9). Another major characteristic of TKHUIM-GA is that it does not need any fixed probability value for crossover and mutation. Instead, the algorithm automatically updates such values based on whether the elite population is being improved or not (see Algorithm 3, line 11). In the initial generation, there is no information about how the genetic operators should work, so the parameters are initialized to the default value of 0.5. If the algorithm is discovering new and better solutions, then better exploitation is required so the crossover probability is increased ($\alpha \leftarrow \alpha + 0.05$) and the mutation probability is reduced ($\beta \leftarrow \beta - 0.05$). On the contrary, if no better solutions are being found, better exploration is required in the search space. Hence, the mutation probability is increased ($\beta \leftarrow \beta + 0.05$), and the crossover probability is reduced ($\alpha \leftarrow \alpha - 0.05$). These two probabilities maintain a relation of p and $1 - p$, so the sum of both probability values is 1. After 10 generations with the mutation probability to its highest value (high exploration) the algorithm stops.

Last but not least, it is important to know the computational complexity of the proposal. The initial procedure creates a set of solutions from the dataset Ω by considering the utility values of every item in the data. Thus, the complexity of this initial procedure depends on the total number of elements kept in the data or the sum of the lengths (number of different items) of each transaction $d = ||\Omega|| = \sum_{i=1}^{|\Omega|} |T_i|$. Additionally, to evaluate a solution or individual, it requires analysing all the elements in the data, that is, $d = ||\Omega||$, so the complexity of evaluating one individual is also d. The complexity of evaluating n individuals is $n \times d$. As for the crossover operator, it depends on its probability α, m (the size of the individuals), and n (the number of individuals): $\alpha \times m \times n$. Similarly, the complexity of the mutation will depend on its probability β, m, and $n : \beta \times m \times n$. Both complexities can be simplified as $O(m \times n)$. Hence, the complexity of the proposed algorithm is $O(d + n \times d + g \times n \times d + 2 \times m \times n)$. Here, d (data elements) is much larger than m (individual size), so the overall complexity of the proposal can be simplified as $O(g \times n \times d)$. It means that it depends on the number of generations, the number of individuals, and the total number of elements in the data. It is finally important to note that g is not a feasible parameter and it depends on how well the algorithm is performing (it stops after a number of iterations without improvement).

4. Experimental Analysis

This section presents the experimental study aiming to describe the algorithms' behaviour in different scenarios and through different metrics. In this regard, the average utility values, the runtime and the memory consumption are analysed. The results of the experiments carried out in this section are supported by statistical analyses with the aim of determining whether the algorithms behave similarly or whether there are statistical differences among them. The first subsection describes all the algorithms considered in the experimental analysis (16 algorithms) and the datasets (18 different datasets). The second subsection aims to analyse the proposal's parameters, and the impact the parameter setting has on the results. Then, subsection three compares the proposal to existing top-*k* utility mining algorithms, considering both exhaustive and evolutionary approaches. The fifth subsection aims to compare the proposal to well-known bio-inspired algorithms for mining high utility itemsets. Finally, a summary of the knowledge acquired through the experiments is outlined.

4.1. Experimental Setup

All the algorithms used in this comparison are available in the SPMF library [5]. The set of twelve bio-inspired algorithms used in this analysis is the following: HUIM-GA [13], HUIM-BPSO [18], HUIM-GA-tree [18], HUIM-BPSO-tree [18], HUIF-PSO [27], HUIF-GA [27], HUIF-BA [27], HUIM-ABC [26], HUIM-SPSO [28], HUIM-AF [29], HUIM-HC [23], and HUIM-SA [23]. The deterministic TKU [37] and TKO [33] algorithms were also used since they are the baseline algorithms for mining top-*k* high utility itemsets following an exhaustive methodology. Additionally, two heuristic-based algorithms for mining top-*k* high utility patterns were considered: TKU-CE[30] and TKU-CE+[31]. The experiments are carried out on a set of 18 datasets taken from the SPMF library [5]. Table 4 shows the set of datasets and their characteristics. Datasets including positive, negative, integer and real unit profits are considered. Some algorithms cannot run on negative and real unit profits. All the experiments were done in a 64 bits machine, Intel(R) i7 2.8 GHz with 4 cores and 16 GB of RAM. Experiments were repeated 5 times for each dataset, taking the average results.

4.2. Parameter Setting

Before starting the parameter setting, it is important to evaluate the performance of the proposed genetic operators. On this regard, the convergence of the approach is compared to the same genetic operators but considering an equiprobable random policy. Thus, items are not selected based on their utility but randomly. Fig. 3 shows the convergence of the TKHUIM-GA algorithm considering the proposed genetic operators vs random operators on the Mushroom dataset. As it is illustrated, random genetic operators can produce good results in short terms but they produce a premature convergence. On the contrary, the proposed genetic operators produce better results mainly since they do not randomly exploit the search space. A summary of the whole set of results is shown in Table 5. It is important to remark that all the experiments were run

Table 4
Datasets and their characteristics, ordered by ascending order of number of transactions.

Name	#Transactions	#Items	Avg. item/trans	#Elements	Utility type
Chess	3,196	75	37.00	118,252	Positive (integer)
ChessNeg	3,196	75	37.00	118,252	Negative (integer)
Foodmart	4,141	1,559	4.42	18,304	Positive (integer)
Mushroom	8,416	119	23.00	193,568	Positive (integer)
MushroomNeg	8,416	119	23.00	193,568	Negative (integer)
Liquor	9,284	2,626	2.70	25,067	Positive (real)
Ecommerce	17,535	3,803	15.40	270,039	Positive (integer)
Pumsb	49,046	2,113	74.00	3, 629,404	Positive(integer)
BMS	59,602	3,340	4.62	275,362	Positive (integer)
Connect	67,557	129	43.00	2,904,951	Positive (integer)
Retail	88,162	16,470	10.30	908,069	Positive (integer)
RetailNeg	88,162	16,470	10.30	908,069	Negative (integer)
Fruithut	181,970	1,265	3.58	651,453	Positive (integer)
Accidents	340,183	468	33.80	11,498,186	Positive (integer)
AccidentsNeg	340,183	468	33.80	11,498,186	Negative (integer)
Kosarak	990,002	41,270	8.10	8,019,017	Positive (integer)
KosarakNeg	990,002	41,270	8.10	8,019,017	Negative (integer)
ChainStore	1,112,949	46,086	7.23	8,046,622	Positive (integer)

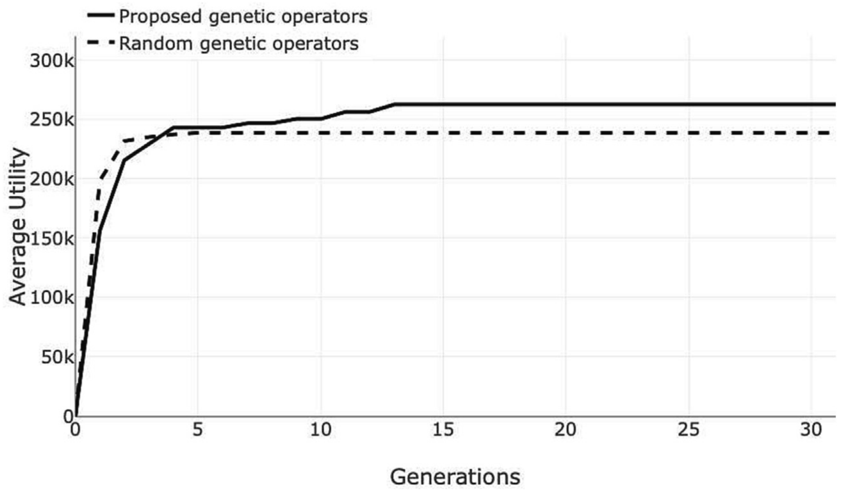


Fig. 3. Analysis of convergence of the proposed genetic operators on the Mushroom dataset.

considering the same parameters. The best 15 solutions found along the evolutionary process are returned, and the average utility of this set of solutions is analysed. Table 5 shows the average results and standard deviation after five different runs per algorithm and dataset. This analysis is important to be done since they are not exhaustive approaches. The results show that both proposals work well and produce similar results. The Wilcoxon signed-rank test [36], a non-parametric test which compares two related samples, reveals that there are significant differences among them (p-value = 0.021824) at a significance level of 95%, the proposed genetic operators performing statistically better. This analysis reveals that the use of the proposed genetic operators is worth it, so they should be used.

The next analysis before the parameter setting is carried out aims to demonstrate that the proposal as a whole is worth it. On this regard, it is compared to a random search approach to proof that the the proposal guides the searching process to better solutions (see Table 6). To perform this analysis, we consider the same population size (25 individuals) and the same number of solutions to be returned (the best 15 solutions found along the process). Both processes end if no improvement of the solutions is reached after 15 generations. Additionally, both approaches consider a maximum number of attributes of five in the solutions, a parameter that can vary according to the user's preferences. Unlike TKHUIM-GA, the random search approach just creates 25 new solutions in each iteration, not considering any genetic operator. Table 6 shows the average results after five different runs per algorithm and dataset. Results are compared through a non-parametric test (Wilcoxon signed-rank test [36]) to determine whether the null-hypothesis that the two algorithms equally perform is accepted. A p-value of $7.63e-6$ is obtained, rejecting the null-hypothesis with a significant level of 99%. Thus, it is statistically demonstrated that TKHUIM-GA is good at guiding the search process for new and better solutions.

Table 5
Analysis of the convergence of the proposed genetic operators. Bold-typeface denotes the best values. Results are the average of 5 different runs.

Dataset	Random operators Avg. Utility	Proposed operators Avg. Utility
Chess	383,007.4	**419,734.9**
ChessNeg	**121,443.6**	114,959.6
Foodmart	**21,882.6**	**21,882.6**
Mushroom	244,932.2	**276,387.4**
MushroomNeg	247,552.4	**315,927.6**
Liquor	**399,959.5**	**399,959.5**
Ecommerce	**3,742,181.8**	**3,742,181.8**
Pumsb	3,200,592.5	**5,476,778.7**
BMS	**2,649,495.8**	**2,649,495.8**
Connect	7,516,177.3	**9,475,213.3**
Retail	**134,577.2**	**134,577.2**
RetailNeg	**153,174.2**	**153,174.2**
Fruithut	**5,239,169.9**	**5,239,169.9**
Accidents	15,350,678.3	**17,504,950.1**
AccidentsNeg	9,820,243.9	**16,917,881.0**
Kosarak	**2,944,726.3**	**2,944,726.3**
KosarakNeg	**2,151,699.4**	2,104,056.9
ChainStore	2,944,726.3	**17,530,566.6**

Table 6
Analysis of the performance of TKHUIM-GA compared to a random search. Bold-typeface denotes the best values. Results are the average of 5 different runs.

Dataset	Random search Avg. Utility	TKHUIM-GA Avg. Utility
Chess	217,364.4	**419,734.9**
ChessNeg	82,879.6	**114,959.6**
Foodmart	19,254.0	**21,882.6**
Mushroom	147,556.9	**276,387.4**
MushroomNeg	135,534.4	**315,927.6**
Liquor	106,927.2	**399,959.5**
Ecommerce	1,505,081.9	**3,742,181.8**
Pumsb	852,271.6	**5,476,778.7**
BMS	1,970,658.8	**2,649,495.8**
Connect	3,493,123.9	**9,475,213.3**
Retail	22,935.9	**134,577.2**
RetailNeg	7,829.9	**153,174.2**
Fruithut	2,536,922.7	**5,239,169.9**
Accidents	6,113,733.3	**17,504,950.1**
AccidentsNeg	3,903,578.3	**16,917,881.0**
Kosarak	81,709.8	**2,944,726.3**
KosarakNeg	29,809.1	**2,104,056.9**
ChainStore	821,220.4	**17,530,566.6**

Let us perform the parameter setting of the proposed algorithm, aiming to demonstrate whether the parameter values have a huge impact on the results and which combination of values is the most appropriate. First, as it was described in Section 3.3, the overall complexity of the proposal is $O(g \times n \times d)$, where g is the number of generation, n the number of individuals, and d is the total number of elements in the data. Additionally, it is important to remark that g is not an input variable and it depends on the data distribution (there is not a fixed number of generations and the algorithm stops once the results are not improving. With all of this in mind, it is interesting to analyse the algorithm's performance for different n values and on different datasets (different d values). Results are analysed in terms of the obtained average utility and runtime. In this parameter setting, we consider four different population sizes and the results are shown in Tables 7 and 8. A hypothesis testing by means of non-parametric statistical tests has been conducted with the aim of determining whether there exist significant differences in the overall performance when different population sizes are considered. The Friedman's test [8] has been used to analyze the general differences, whereas the Shaffer's post hoc test [25] has been employed to perform all pairwise comparisons. Starting with the runtime (see Table 7), the Friedman's test detected that there were general statistical differences in the four versions at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $2.6e - 11$. Then, the Shaffer's post hoc test was performed to detect where these significant differences were located. The results for this post hoc test, at a significance level of $\alpha = 0.01$, are summarized through the critical difference

Table 7
Analysis of the runtime in seconds of TKHUIM-GA for different population sizes. Bold-typeface denotes the best values. Results are the average of 5 different runs.

	Population size			
Dataset	25	50	75	100
Chess	**0.7**	1.4	2.4	2.6
ChessNeg	**0.4**	0.7	0.9	1.1
Foodmart	**0.1**	**0.1**	0.2	0.3
Mushroom	**0.6**	1.1	1.7	1.9
MushroomNeg	**0.6**	1.1	1.4	1.2
Liquor	**1.0**	1.3	1.7	2.2
Ecommerce	**0.5**	0.6	0.9	1.2
Pumsb	**15.5**	49.0	93.5	99.4
BMS	**0.5**	1.0	1.1	1.4
Connect	**24.8**	75.3	101.4	136.5
Retail	**2.8**	4.2	5.1	6.9
RetailNeg	**3.4**	4.4	5.4	6.9
Fruithut	**1.8**	2.6	2.8	3.5
Accidents	**45.3**	68.7	98.4	140.4
AccidentsNeg	**31.6**	56.9	75.2	103.8
Kosarak	**23.6**	27.6	33.2	43.2
KosarakNeg	**23.1**	32.9	36.8	44.9
ChainStore	**25.5**	37.5	43.1	47.7

Table 8
Analysis of the average utility obtained by TKHUIM-GA for different population sizes. Bold-typeface denotes the best values. Results are the average of 5 different runs.

	Population size			
Dataset	25	50	75	100
Chess	419,734.9	501,987.0	512,471.3	**544,427.1**
ChessNeg	114,959.6	124,418.6	137,574.0	**137,667.2**
Foodmart	**21,882.7**	**21,882.7**	**21,882.7**	**21,882.7**
Mushroom	276,387.4	453,919.5	**467,093.5**	460,447.0
MushroomNeg	315,927.6	314,813.2	339,644.2	**347,837.0**
Liquor	**399,959.5**	**399,959.5**	**399,959.5**	**399,959.5**
Ecommerce	**3,742,181.8**	**3,742,181.8**	**3,742,181.8**	**3,742,181.8**
Pumsb	5,476,778.7	6,703,944.3	6,695,817.2	**6,854,227.7**
BMS	**2,649,495.8**	**2,649,495.8**	**2,649,495.8**	**2,649,495.8**
Connect	9,475,213.3	10,665,171.5	10,913,429.7	**10,928,971.4**
Retail	**134,577.2**	**134,577.2**	**134,577.2**	**134,577.2**
RetailNeg	**153,174.3**	**153,174.3**	**153,174.3**	**153,174.3**
Fruithut	**5,239,169.9**	**5,239,169.9**	**5,239,169.9**	**5,239,169.9**
Accidents	17,504,950.1	19,769,532.4	19,926,647.5	**22,682,004.1**
AccidentsNeg	16,917,881.0	16,983,144.1	15,118,627.7	**18,238,338.4**
Kosarak	**2,944,726.3**	**2,944,726.3**	**2,944,726.3**	**2,944,726.3**
KosarakNeg	**2,104,056.9**	**2,104,056.9**	**2,104,056.9**	**2,104,056.9**
ChainStore	**17,530,566.6**	**17,530,566.6**	**17,530,566.6**	**17,530,566.6**

diagram shown in Fig. 4, illustrating that a population size of 25 significantly outperformed the runtime required by 75 and 100 of population size. However, no significant difference can be obtained between 25 and 50. The runtime increases with the population size, as it was expected according to the complexity of the algorithm. This increment is not linear and depends on the dataset. It is important to remind that the algorithm's complexity is based on both d (data size) and n (population size). Hence, low average items per transaction (see Table 1) will have a low impact on the runtime. It explains why the runtime on the Fruithut dataset increases in a low rate with the population size (the runtime is the double for 100 individuals than for 25 individuals), whereas the runtime on the Pumsb dataset increases in a high rate (the runtime by 100 individuals is 6 times the one required by 25 individuals).

Following with the average utility (see Table 8), the Friedman's test detected that there were no general statistical differences in the four versions at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than 0.0503. Statistical differences were found at a significance level of $\alpha = 0.1$. Then, the Shaffer's post hoc test was performed to detect where these significant differences were located. The results for this post hoc test, at a significance level of $\alpha = 0.1$, are summarized through the critical difference diagram shown in Fig. 5, illustrating that the best results were obtained with a population size of 100 but no statistical difference can be reached with 75 and 50. Similarly, population sizes of 25, 50 and 75 equally perform according to the Shaffer's post hoc test.

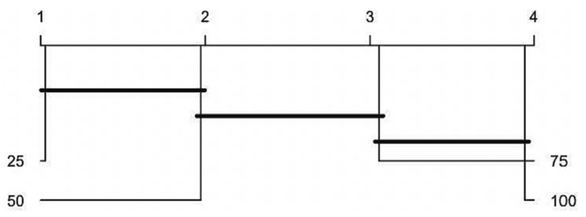


Fig. 4. Critical difference diagram showing a statistical comparison of the runtime with different population sizes conducted using the Shaffer's test.

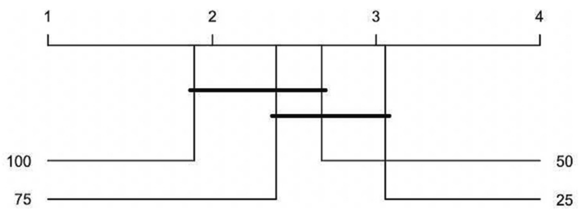


Fig. 5. Critical difference diagram showing a statistical comparison of the average utility with different population sizes conducted using the Shaffer's test.

To conclude this analysis, and taking into account the statistical results provided by the Friedman's and Shaffer's tests, a population size of 50 can be considered as a good one. In terms of runtime, this population size is the second best option and no statistical difference is obtained with regard to the best option (population size of 25). Similarly, a population size of 50 is the third best option in terms of average utility values but no statistical difference is obtained with regard to the first and second options (population sizes of 100 and 75). In fact, no difference for the average utility values were obtained among the four versions for $\alpha < 0.1$.

4.3. Comparison to Existing Top-k Utility Mining Algorithms

TKHUIM-GA is an algorithm that returns the best k solutions found along an evolutionary process. It is therefore required to analyse how good the obtained solutions are versus solutions obtained by exhaustive approaches and hueristic-based algorithms. It is important to highlight that due to TKHUIM-GA is an evolutionary approach, it cannot obtain better solutions than those returned by exhaustive approaches, but these solutions should be as good as possible. In this analysis, we have considered TKU [37] and TKO [33], which have been demonstrated to be the reference algorithms for mining top-k utility itemsets together with kHMC [4] with no significant differences between TKO and kHMC. As for heuristic-based approaches, we have considered the two existing algorithms: TKU-CE[30] and TKU-CE+[31]. A comparison in terms of runtime and memory requirements is also performed. Because exhaustive search approaches do not enable a maximum length (number of attributes) to be pre-defined, TKHUIM-GA has considered the maximum feasible value to carry out a fair comparison. Last but not least, exhaustive approaches might not terminate due to memory requirements (we are using 16 GB of memory) or after a limit of 600 s. Additionally, TKO [33] does not work with negative unit profits so the number of datasets considered in this analysis varies with regard to that of previous analyses. Many authors [4,33,37] have considered a reduced set of datasets or a small percentage of transactions for large datasets to avoid that problem, so we have followed a similar methodology.

First, we analyse the returned solutions in terms of their average utility considering the k values 5, 10 and 15 (see Tables 9–11). Focusing first on the best 5 itemsets (see Table 9) TKHUIM-GA behaves really well and it returns solutions in all the datasets and, in some cases, the same results as the exhaustive search approaches (TKU [37] and TKO [33]). The rest of algorithms have some problems related to the runtime limit (more than 600 s) and, in one case, related to the heap space (memory limit surpassed). Additionally, some algorithms cannot work on real unit profits (Liquor dataset). Finally, it is important to remind that TKHUIM-GA is not an exhaustive approach and it cannot overcome the solutions found by TKU [37] and TKO [33]. Results are compared through non-parametric statistical tests. The Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $8.3e-4$. Then, the Shaffer's post hoc test was performed to detect where these significant differences were located. The results for this post hoc test, at a significance level of $\alpha = 0.01$, are summarized through the critical difference diagram shown in Fig. 6, illustrating that TKHUIM-GA is good at guiding the search process for good solutions. No statistical differences were obtained with regard to TKU and TKO (exhaustive search approaches) and the heuristic-based approaches TKU-CE and TKU-CE+. The same analysis is done considering the best 10 itemsets (see Table 10). The Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $3.2e-4$. Then, the Shaffer's post hoc (see Fig. 7), at a significance level of

Table 9
Average utility values of the top-5 itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	558,189.4	**865,295.6**	**865,295.6**	680,916.0	680,916.0
Foodmart	**24,050.0**	**24,050.0**	**24,050.0**	18,351.6	18,351.6
Mushroom	160,626.0	**566,848.4**	**566,848.4**	547,347.0	451,952.2
Liquor	**635,370.7**	-	-	-	-
Ecommerce	**5,923,470.0**	HS	**5,923,470.0**	3,171,046.6	4,375,408.8
Pumsb 1%	76,049.2	TL	**141,576.4**	123,987.6	105,942.2
Pumsb 2%	125,441.2	TL	**273,942.4**	235,967.6	200,750.2
Pumsb 3%	205,384.6	TL	**395,494.8**	**395.494.8**	337,533.4
Pumsb 4%	259,559.2	TL	**504,655.6**	**504,655.6**	431,367.6
BMS 10%	682,198.6	**717,005.0**	**717,005.0**	TL	697,556.0
BMS 12%	793,666.8	TL	**835,526.0**	TL	816,375.0
BMS 14%	865,295.6	TL	**902,599.6**	TL	865,295.6
Connect	14,557,757.2	TL	**16,905,460.8**	**16,905,460.8**	**16,905,460.8**
Retail	282,034.0	**345,063.4**	**345,063.4**	TL	**345,063.4**
Fruithut	**6,867,533.2**	**6,867,533.2**	**6,867,533.2**	6,758,734.6	**6,867,533.2**
Accidents	9,150,290.6	TL	**30,540,178.4**	TL	TL
Kosarak	7,028,428.8	TL	**11,814,436.0**	TL	TL
ChainStore	**32,378,504.2**	TL	HS	HS	13,810,344.6

Table 10
Average utility values of the top-10 itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	497,733.9	TL	**677,009.4**	**677,009.4**	**677,009.4**
Foodmart	**22,787.5**	**22,787.5**	**22,787.5**	16,019.0	16,634.1
Mushroom	437,675.4	**561,967.5**	**561,967.5**	537,418.4	435,274.9
Liquor	**480,728.1**	-	-	-	-
Ecommerce	4,431,360.5	HS	**4,730,748.6**	2,221,557.2	2,998,100.8
Pumsb 1%	47,733.4	TL	**140,953.9**	123,390.4	99,285.0
Pumsb 2%	169,673.0	TL	**272,966.1**	**272,966.1**	200,172.5
Pumsb 3%	211,928.6	TL	**394,283.7**	**394,283.7**	309,601.4
Pumsb 4%	245,793.8	TL	**503,202.2**	431,399.0	431,491.4
BMS 10%	435,313.4	TL	**484,025.5**	TL	457,017.4
BMS 12%	507,197.7	TL	**570,545.7**	TL	546,431.1
BMS 14%	554,910.4	TL	TL	TL	**561,183.4**
Connect	12,057,210.5	TL	**16,875,146.9**	**16,875,146.9**	16,874,116.0
Retail	179,535.7	**287,154.7**	**287,154.7**	TL	60,438.3
Fruithut	**5,996,516.9**	**5,996,516.9**	**5,996,516.9**	5,353,426.4	5,962,580.8
Accidents	7,126,967.6	TL	**30,143,764.2**	TL	**30,143,764.2**
Kosarak	4,051,755.0	TL	**8,560,873.1**	HS	TL
ChainStore	**21,811,480.1**	TL	HS	HS	10,514,814.6

Table 11
Average utility values of the top-15 itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	501,987.0	HS	**673,957.9**	**673,957.9**	**673,957.9**
Foodmart	**21,882.7**	**21,882.7**	**21,882.7**	15,584.8	18,583.3
Mushroom	453,919.5	**557,208.9**	**557,208.9**	517,224.3	439,932.9
Liquor	**399,959.5**	-	-	-	-
Ecommerce	**3,742,181.8**	HS	HS	1,944,421.9	2,383,675.7
Pumsb 1%	69,537.1	TL	**140,660.1**	123,223.2	99,006.8
Pumsb 2%	106,997.5	TL	**271,726.7**	271,708.0	202,018.7
Pumsb 3%	164,714.9	TL	**392,430.1**	392,416.9	298,115.3
Pumsb 4%	240,682.8	TL	**501,363.5**	501,362.2	430,207.0
BMS 10%	341,108.8	TL	**391,076.1**	TL	382,669.9
BMS 12%	399,054.1	TL	**461,309.1**	TL	440,298.3
BMS 14%	440,002.7	TL	TL	TL	**442,985.3**
Connect	10,665,171.5	TL	**16,843,077.0**	TL	16,840,818.8
Retail	134,577.2	**248,834.1**	**248,834.1**	TL	59,149.6
Fruithut	**5,239,169.9**	**5,239,169.9**	**5,239,169.9**	TL	5,045,155.5
Accidents	19,769,532.4	TL	**29,802,392.5**	TL	TL
Kosarak	2,944,726.3	TL	**7,049,468.5**	HS	TL
ChainStore	**17,530,566.6**	HS	HS	HS	TL

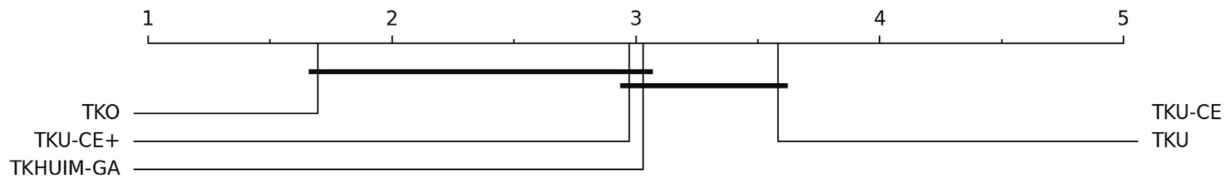


Fig. 6. Critical difference diagram showing a statistical comparison of the average utility obtained by different approaches on $k = 5$ conducted using the Shaffer's test.

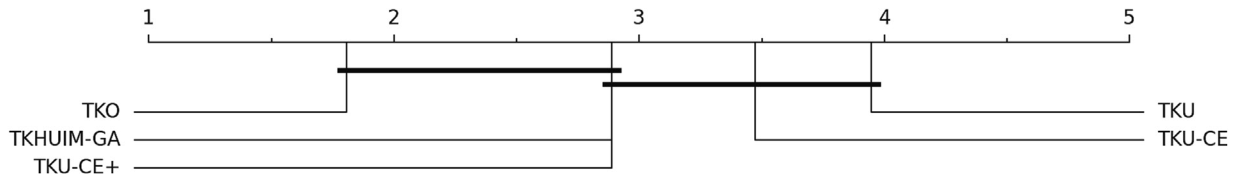


Fig. 7. Critical difference diagram showing a statistical comparison of the average utility obtained by different approaches on $k = 10$ conducted using the Shaffer's test.

$\alpha = 0.01$, demonstrated that TKHUIM-GA behaves really well, with no significant differences with TKU-CE + and TKO. TKHUIM-GA behaves better than using top-k 5 (see Fig. 6). Finally, we perform the analysis on the best 15 itemsets (see Table 11). The Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $5.8e - 4$, and the subsequent Shaffer's post hoc (see Fig. 8) also demonstrated that TKHUIM-GA, TKU-CE + and TKO are the best algorithms with no differences at a significance level of $\alpha = 0.01$. To sum up, the results demonstrated that TKO, TKU-CE + and the proposed approach produce similar results on different scenarios (different k values). TKU-CE + behaves worse and worse with the increment of value k. The proposed TKHUIM-GA algorithm extracts patterns in all the datasets unlike the rest of algorithms, which fails in some datasets due to incompatibility of the values, memory requirements or time limits.

Focusing on the runtime, the same analysis was performed for different k values (see Tables 12–14). The results for most of the datasets and k values demonstrate that TKHUIM-GA is the fastest algorithm, returning solutions in few seconds. Results are compared through non-parametric statistical tests. Considering first the runtime required for extracting the top 5 itemsets (see Table 12), the Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $8.8e - 8$. Then, the Shaffer's post hoc test was performed to detect where these significant differences were located. The results for this post hoc test, at a significance level of $\alpha = 0.01$, are summarized through the critical difference diagram shown in Fig. 9, illustrating that KHUIM-GA is the fastest approach and it statistically outperforms TKU, TKU-CE and TKU-CE+. Considering the top 10 itemsets (see Table 13), the Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $1.6e - 7$. Then, the Shaffer's post hoc test was performed with a significance level of $\alpha = 0.01$, and the results are summarized through the critical difference diagram shown in Fig. 10, illustrating that KHUIM-GA is the fastest approach and it statistically outperforms again TKU, TKU-CE and TKU-CE+. The TKU algorithm appears as the slower algorithm. Finally, an analysis on top 15 itemsets (see Table 14) was performed. The Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $2.0e - 8$. Then, the Shaffer's post hoc test was performed with a significance level of $\alpha = 0.01$, and the results are summarized through the critical difference diagram shown in Fig. 11, illustrating that KHUIM-GA is the fastest approach but no statistical differences were found with regard to TKO. However, the differences betwen these two algorithms are larger and larger with the increment of the values k. TKU is again the slowest algorithm.

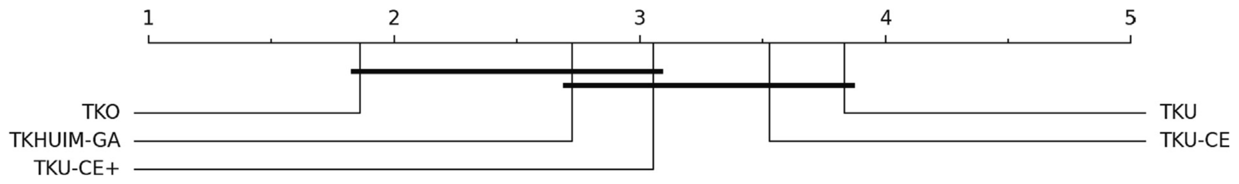


Fig. 8. Critical difference diagram showing a statistical comparison of the average utility obtained by different approaches on $k = 15$ conducted using the Shaffer's test.

Table 12
Runtime in seconds for mining top-5 utility itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	2.3	423.6	**1.6**	13.8	8.3
Foodmart	**0.1**	0.4	0.4	1.6	1.8
Mushroom	**0.5**	19.3	0.7	15.5	42.3
Liquor	**1.5**	-	-	-	-
Ecommerce	**0.6**	HS	5.3	27.5	6.2
Pumsb 1%	**0.3**	TL	10.8	10.1	3.6
Pumsb 2%	**0.4**	TL	24.9	25.8	6.3
Pumsb 3%	**0.7**	TL	56.9	38.6	10.9
Pumsb 4%	**0.8**	TL	86.9	51.9	8.1
BMS 10%	0.2	0.3	**0.1**	TL	13.2
BMS 12%	0.2	TL	**0.1**	TL	11.2
BMS 14%	**0.3**	TL	0.9	TL	461.3
Connect	129.0	TL	**80.4**	512.2	276.7
Retail	6.1	2.5	**0.5**	TL	25.9
Fruithut	3.6	3.1	**1.1**	145.2	18.7
Accidents	**27.9**	TL	62.7	TL	TL
Kosarak	38.8	TL	**4.6**	TL	TL
ChainStore	**42.9**	TL	HS	HS	265.9

Table 13
Runtime in seconds for mining top-10 utility itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	**1.8**	TL	**1.8**	18.4	8.9
Foodmart	**0.2**	0.3	0.4	1.7	7.1
Mushroom	1.4	38.7	**0.7**	26.1	96.3
Liquor	**1.7**	-	-	-	-
Ecommerce	**0.8**	HS	5.1	24.9	3.4
Pumsb 1%	**0.3**	TL	14.9	10.3	95.6
Pumsb 2%	**0.7**	TL	32.7	26.0	161.2
Pumsb 3%	**0.8**	TL	64.6	37.2	12.6
Pumsb 4%	**0.9**	TL	92.7	56.9	7.4
BMS 10%	**0.1**	TL	0.2	TL	18.7
BMS 12%	**0.2**	TL	**0.2**	TL	17.3
BMS 14%	**0.2**	TL	TL	TL	410.9
Connect	**73.3**	TL	101.9	542.2	268.1
Retail	6.1	3.6	**0.8**	TL	141.8
Fruithut	3.5	3.2	**1.3**	170.8	16.2
Accidents	**31.4**	TL	71.5	TL	504.4
Kosarak	33.6	TL	**4.7**	HS	TL
ChainStore	**44.5**	TL	HS	HS	190.6

Table 14
Runtime in seconds for mining top-15 utility itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	**1.4**	HS	2.0	17.2	9.4
Foodmart	**0.1**	0.3	0.5	1.6	64.3
Mushroom	1.1	42.7	**0.8**	34.4	58.5
Liquor	**1.3**	-	-	-	-
Ecommerce	**0.6**	HS	HS	31.2	6.8
Pumsb 1%	**0.2**	TL	14.5	13.2	143.8
Pumsb 2%	**0.3**	TL	35.7	35.1	178.2
Pumsb 3%	**0.5**	TL	83.1	69.4	10.6
Pumsb 4%	**0.7**	TL	112.6	83.0	8.9
BMS 10%	**0.1**	TL	0.2	TL	30.7
BMS 12%	**0.1**	TL	0.2	TL	34.1
BMS 14%	**0.2**	TL	TL	TL	468.9
Connect	**75.3**	TL	117.8	TL	275.2
Retail	4.2	4.3	**0.7**	TL	485.9
Fruithut	2.8	3.9	**1.9**	TL	28.3
Accidents	**68.7**	TL	73.7	TL	TL
Kosarak	27.6	TL	**5.4**	HS	TL
ChainStore	**37.5**	HS	HS	HS	TL

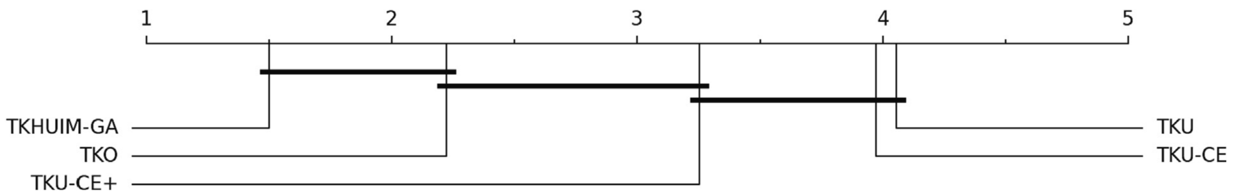


Fig. 9. Critical difference diagram showing a statistical comparison of the average runtime required by different approaches on $k = 5$ conducted using the Shaffer's test.

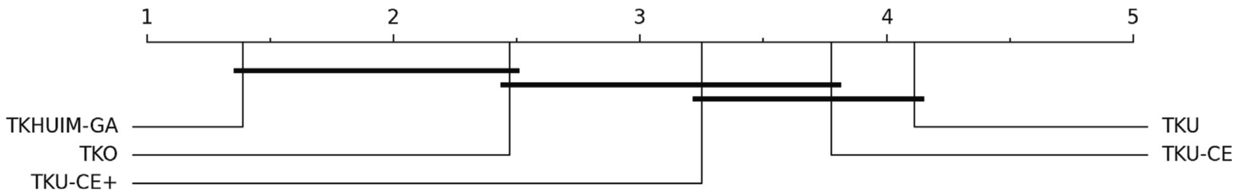


Fig. 10. Critical difference diagram showing a statistical comparison of the average runtime required by different approaches on $k = 10$ conducted using the Shaffer's test.

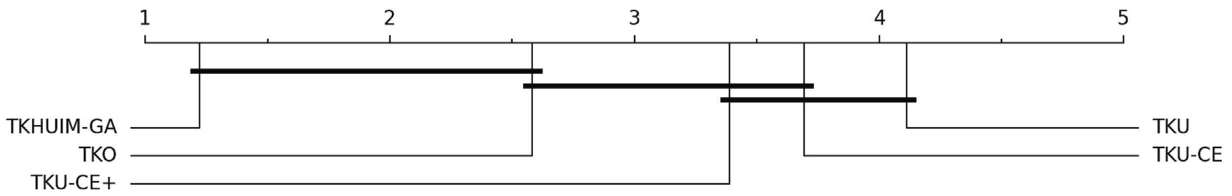


Fig. 11. Critical difference diagram showing a statistical comparison of the average runtime required by different approaches on $k = 15$ conducted using the Shaffer's test.

The next analysis is related to memory usage (see Tables 15–17). Huge differences are obtained, and the proposed TKHUIM-GA algorithm returns solutions requiring a low quantity of memory where some approaches cannot run due to it requires more than 16 GB of memory (heap space). As for the extraction of top 5 itemsets (see Table 15), the Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $8.7e - 9$. Then, the Shaffer's post hoc test was performed considering a significance level of $\alpha = 0.01$ and the results are summarized through the critical difference diagram in Fig. 12. It is obtained that TKHUIM-GA outperforms all the algorithms under analysis except for TKO. These two algorithms do not present statistical differences. The same results were obtained for the top 10 itemsets (see Table 16). The Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $1.5e - 9$ for $k = 10$. Then, the Shaffer's post hoc test was performed considering a significance level of $\alpha = 0.01$ and the results are summarized through the critical difference diagram in Fig. 13. TKHUIM-GA is the best algorithm with no statistical difference versus TKO. Finally, considering the top k 15 itemsets (see Table 17), the Friedman's test detected that there were general statistical differences in the four algorithms at a significance level of $\alpha = 0.01$, rejecting the null hypothesis with a p-value smaller than $3.7e - 9$. This time, the Shaffer's post hoc test revealed statistical differences with $\alpha = 0.01$. As it is summarized through the critical difference diagram in Fig. 14, TKHUIM-GA outperforms all the algorithms under analysis.

Last but not least, we analyse the solutions returned by TKHUIM-GA in terms of whether they are within the best solutions found by TKO. In this regard, we analyse (see Table 18) whether the best solution was found, and whether any of the solutions returned by TKHUIM-GA are within the actual set of 5, 10 and 15 best solutions found by TKO. As it is shown, TKHUIM-GA returns any of the top 5, 10 and 15 solutions in all the datasets except for Accidents. The best solution is also found in most of the datasets. There are four datasets (denoted by "Cannot be checked") that cannot be run on exhaustive search approaches (memory or runtime problems) and, therefore, the solutions returned by TKHUIM-GA cannot be analysed.

4.4. Comparison to Bio-inspired Approaches

The aim of this analysis is to compare the proposed TKHUIM-GA algorithm to existing bio-inspired approaches. Since existing bio-inspired algorithms need a predefined utility threshold as any exhaustive approach for mining high utility item-

Table 15
Memory usage in MB for mining top-5 utility itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	**39.7**	515.2	153.3	156.3	126.4
Foodmart	**21.4**	39.5	84.5	543.8	967.9
Mushroom	**56.8**	151.9	152.7	411.5	461.2
Liquor	**106.3**	-	-	-	-
Ecommerce	**56.9**	HS	226.9	806.5	611.8
Pumsb 1%	**26.2**	TL	353.6	540.8	378.2
Pumsb 2%	**33.6**	TL	403.7	557.3	247.9
Pumsb 3%	**39.4**	TL	460.1	170.9	163.2
Pumsb 4%	**45.6**	TL	445.1	164.9	441.5
BMS 10%	21.5	33.9	**17.8**	TL	703.0
BMS 12%	**22.5**	TL	22.9	TL	195.5
BMS 14%	**23.3**	TL	40.4	TL	1,036.0
Connect	**576.0**	TL	773.6	1,495.1	1,552.9
Retail	**214.1**	2,443.3	939.6	TL	2,287.1
Fruithut	**175.8**	1,417.9	892.5	2,401.7	2,179.2
Accidents	1,935.4	TL	**1,564.5**	TL	TL
Kosarak	1,566.7	TL	**977.1**	TL	TL
ChainStore	1,751.1	TL	HS	HS	**765.7**

Table 16
Memory usage in MB for mining top-10 utility itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	**40.4**	TL	187.5	182.9	136.7
Foodmart	**21.4**	35.1	93.1	359.8	645.0
Mushroom	**57.1**	953.1	152.6	643.2	562.6
Liquor	**105.7**	-	-	-	-
Ecommerce	**58.1**	HS	320.2	122.0	304.2
Pumsb 1%	**25.5**	TL	353.8	701.8	643.3
Pumsb 2%	**33.0**	TL	404.9	643.5	449.9
Pumsb 3%	**40.9**	TL	429.1	641.5	267.4
Pumsb 4%	**46.2**	TL	429.4	TL	440.3
BMS 10%	21.3	TL	**18.7**	TL	721.2
BMS 12%	**22.7**	TL	23.2	TL	999.3
BMS 14%	**23.4**	TL	TL	TL	1,062.5
Connect	**571.3**	TL	891.9	1,536.8	1,687.7
Retail	**215.3**	2,368.7	1,081.7	TL	1,535.2
Fruithut	**177.2**	2,220.9	862.7	1,921.9	1,554.8
Accidents	1,873.9	TL	1,642.6	TL	**1,514.6**
Kosarak	1,576.9	TL	**955.2**	HS	TL
ChainStore	1,751.1	TL	HS	HS	**1,257.8**

Table 17
Memory usage in MB for mining top-15 utility itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s). HS stand for heap space (16 GB memory). A dash means that the algorithm cannot handle with the type of utility values.

Dataset	TKHUIM-GA	TKU	TKO	TKU-CE	TKU-CE+
Chess	**40.6**	HS	169.4	209.8	134.8
Foodmart	**21.4**	35.9	102.5	463.6	1,199.6
Mushroom	**57.7**	1,301.8	152.8	430.6	406.8
Liquor	**107.1**	-	-	-	-
Ecommerce	**57.1**	HS	HS	244.1	620.9
Pumsb 1%	**25.7**	TL	365.9	268.6	903.3
Pumsb 2%	**31.8**	TL	406.6	188.5	296.8
Pumsb 3%	**42.8**	TL	450.6	348.1	305.0
Pumsb 4%	**47.7**	TL	457.9	333.3	410.5
BMS 10%	23.6	TL	**19.8**	TL	501.2
BMS 12%	**23.6**	TL	25.5	TL	600.1
BMS 14%	**26.4**	TL	TL	TL	622.3
Connect	**570.5**	TL	910.8	TL	883.8
Retail	**217.4**	1,999.7	1,073.8	TL	1,695.5
Fruithut	**178.9**	1,649.3	1,026.7	TL	1,506.9
Accidents	**1,894.4**	TL	1,975.4	TL	TL
Kosarak	1,576.9	TL	**866.1**	HS	TL
ChainStore	**1,730.6**	HS	HS	HS	TL

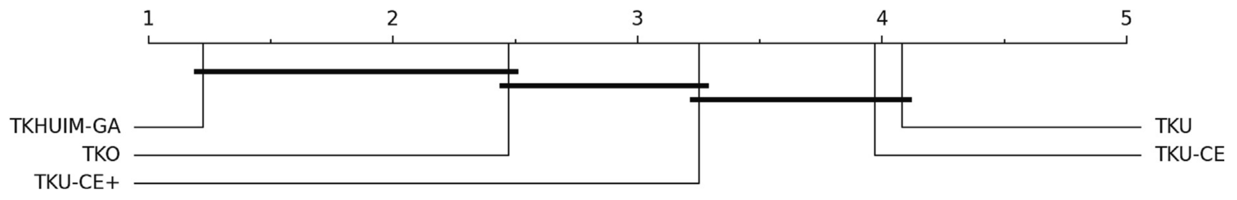


Fig. 12. Critical difference diagram showing a statistical comparison of the memory required by different approaches on $k = 5$ conducted using the Shaffer's test.

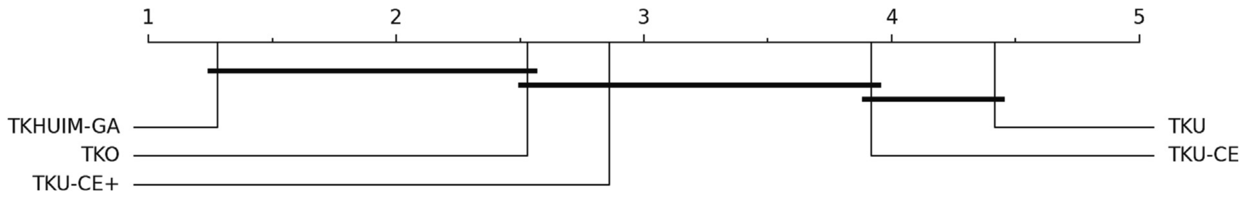


Fig. 13. Critical difference diagram showing a statistical comparison of the memory required by different approaches on $k = 10$ conducted using the Shaffer's test.

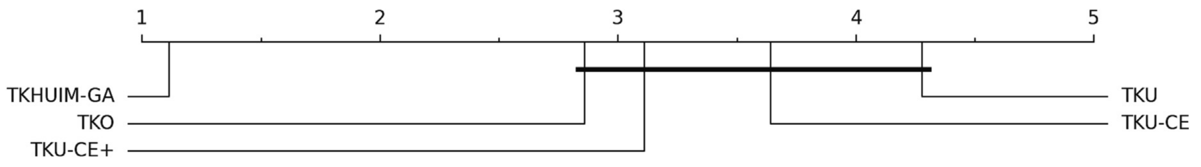


Fig. 14. Critical difference diagram showing a statistical comparison of the memory required by different approaches on $k = 15$ conducted using the Shaffer's test.

Table 18
Analysis of whether any solution returned by TKHUIM-GA is in the actual top-k.

Dataset	Best solution	Top-5	Top-10	Top-15
Chess	Yes	Yes	Yes	Yes
Foodmart	Yes	Yes	Yes	Yes
Mushroom	Yes	Yes	Yes	Yes
Liquor	Cannot be checked			
Ecommerce	Cannot be checked			
Pumsb 1%	No	Yes	Yes	Yes
Pumsb 2%	No	Yes	Yes	Yes
Pumsb 3%	Yes	Yes	Yes	Yes
Pumsb 4%	No	Yes	Yes	Yes
BMS 10%	Yes	Yes	Yes	Yes
BMS 12%	Yes	Yes	Yes	Yes
BMS 14%	Cannot be checked			
Connect	Yes	Yes	Yes	Yes
Retail	No	Yes	Yes	Yes
Fruithut	Yes	Yes	Yes	Yes
Accidents	No	No	No	Yes
Kosarak	No	Yes	Yes	Yes
ChainStore	Cannot be checked			

sets does, the comparison is not fair in terms of solutions. Additionally, existing bio-inspired algorithms do not return the same number of solutions and this fact has a profound impact on the runtime. Hence, in this study we statistically analyse the runtime (see Table 19) and and memory usage (see Table 20) when different utility threshold values are used. We also show the number of solutions found by each algorithm (see Table 21). It is also important to remark that some datasets cannot be run by existing bio-inspired algorithms (e.g. Liquor dataset was not considered since these approaches cannot deal with real utility values) so they were not included in this analysis. Additionally, most of the existing algorithms cannot obtain solutions in less than 600 s (time limit denoted as TL), which is a clear advantage of TKHUIM-GA.

Table 19
Runtime in seconds for mining high utility itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s).

Dataset	TKHUIM-GA	Minimum Threshold	HUIM-GA	HUIM-BPSO	HUIM-GA-tree	HUIM-BPSO-tree	HUIF-PSO	HUIF-GA	HUIF-BA	HUIM-ABC	HUIM-SPSO	HUIM-AF	HUIM-HC	HUIM-SA
Chess	**1.4**	4×10^5	150.1	251.4	29.5	31.6	125.6	390.9	189.3	TL	96.6	253.9	4.8	14.1
		4.5×10^5	150.4	249.8	37.0	33.1	254.3	410.2	167.9	TL	89.2	239.7	4.6	14.9
		5×10^5	145.3	251.1	32.5	35.8	247.4	412.9	183.3	TL	92.2	262.5	5.1	14.2
ChessNeg	**0.7**	4×10^4	354.7	354.7	TL	44.4	165.6	302.2	269.5	TL	90.7	256.3	33.0	15.4
		4.5×10^4	188.9	219.5	57.2	35.5	154.1	299.1	229.8	TL	89.0	226.5	4.0	15.4
		5×10^4	TL	283.1	TL	40.2	106.9	286.5	219.8	TL	74.4	232.9	12.8	12.4
Foodmart	**0.1**	5×10^2	TL	TL	7.9	108.6	13.3	31.5	43.5	TL	TL	TL	5.7	24.6
		2×10^3	TL	TL	23.7	102.5	12.0	28.5	30.3	TL	TL	TL	6.1	17.5
		5×10^3	TL	TL	48.3	103.1	11.1	20.6	22.2	TL	TL	TL	6.2	18.6
Mushroom	**1.1**	1×10^1	596.8	TL	215.5	166.4	163.0	135.3	599.5	TL	153.5	TL	255.3	TL
		5×10^1	TL	TL	90.2	158.8	155.6	238.7	TL	TL	133.8	TL	265.7	TL
		1×10^2	TL	TL	182.8	136.8	241.8	511.3	TL	TL	143.5	TL	269.5	TL
MushroomNeg	**1.7**	1×10^1	TL	TL	116.3	163.9	192.9	TL	532.5	TL	149.5	TL	231.3	TL
		5×10^1	TL	TL	146.4	147.3	192.7	TL	556.1	TL	161.7	TL	254.5	TL
		1×10^2	TL	TL	192.9	147.1	203.3	TL	560.8	TL	148.6	TL	255.2	TL
Ecommerce	**0.6**	5×10^5	TL	TL	108.9	TL	19.3	165.6	44.6	TL	TL	TL	15.4	36.7
		7×10^5	TL	TL	TL	TL	39.7	218.6	34.7	TL	TL	TL	14.6	40.7
		1×10^6	TL	TL	TL	TL	33.7	195.3	28.7	TL	TL	TL	13.9	40.6
Pumsb 1%	**0.2**	1×10^1	TL	550.2	32.1	86.1	23.5	TL	69.7	TL	TL	TL	49.0	TL
		5×10^1	42.9	531.9	33.9	64.9	21.1	TL	TL	TL	TL	TL	48.3	TL
		1×10^2	TL	479.4	32.5	78.8	22.8	TL	TL	TL	TL	TL	104.1	TL
BMS 10%	**0.1**	6×10^4	TL	597.6	6.2	39.4	2.7	4.3	8.6	TL	483.3	195.7	3.7	10.3
		7×10^4	TL	598.4	66.8	42.2	4.9	3.9	8.1	TL	484.8	188.7	3.2	10.1
		8×10^4	TL	TL	39.1	39.8	2.3	3.9	11.1	TL	464.6	180.6	3.4	10.6
Connect	**75.3**	6.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		7×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	254.9	TL
		7.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
Retail	**4.2**	5×10^1	TL	TL	TL	TL	571.6	TL	TL	TL	TL	TL	198.0	242.2
		1×10^2	TL	TL	TL	TL	598.3	TL	TL	TL	TL	TL	206.5	299.4
		5×10^2	TL	TL	TL	TL	331.5	280.6	488.6	TL	TL	TL	273.7	197.8
Fruithut	**2.8**	5×10^2	TL	TL	TL	TL	339.0	TL	TL	TL	TL	TL	182.3	TL
		7×10^2	TL	TL	TL	TL	327.5	TL	TL	TL	TL	TL	185.6	TL
		1×10^3	TL	TL	TL	TL	310.7	TL	TL	TL	TL	TL	191.9	TL
Accidents	**68.7**	6×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	233.3	TL
		6.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	206.5	TL
		7×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	576.2	TL
Kosarak	**27.6**	1×10^2	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		5×10^3	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		1×10^6	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
ChainStore	**37.5**	1×10^2	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		5×10^3	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		1×10^6	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL

Table 20
Memory usage in MB for mining high utility itemsets. Bold-typeface denotes the best values. TL stands for time limit (more than 600 s).

Dataset	TKHUIM-GA	Minimum Threshold	HUIM-GA	HUIM-BPSO	HUIM-GA-tree	HUIM-BPSO-tree	HUIF-PSO	HUIF-GA	HUIF-BA	HUIM-ABC	HUIM-SPSO	HUIM-AF	HUIM-HC	HUIM-SA
Chess	**40.6**	4×10^5	864.0	901.1	884.9	880.7	841.7	394.6	452.4	TL	351.2	768.2	258.2	691.2
		4.5×10^5	883.7	919.2	833.8	882.9	954.2	424.1	572.2	TL	888.9	821.9	268.1	662.3
		5×10^5	881.2	889.5	873.6	858.3	910.1	436.8	945.4	TL	707.1	788.5	246.9	732.3
ChessNeg	**54.1**	4×10^4	268.2	204.5	TL	171.9	122.1	144.1	464.5	TL	309.6	124.1	345.7	391.9
		4.5×10^4	171.3	211.5	190.9	292.2	241.6	130.7	325.8	TL	355.7	436.9	344.0	246.0
		5×10^4	TL	293.9	TL	180.8	287.8	333.8	683.3	TL	438.6	133.0	386.3	169.3
Foodmart	**21.4**	5×10^2	TL	TL	543.2	1,318.5	1,413.5	910.9	1,002.3	TL	TL	TL	760.4	1,089.6
		2×10^3	TL	TL	163.9	148.4	256.2	397.1	296.5	TL	TL	TL	804.2	1,075.4
		5×10^3	TL	TL	128.2	266.9	190.1	71.8	165.7	TL	TL	TL	382.9	413.7
Mushroom	**57.7**	1×10^1	410.4	TL	1,035.5	1,300.1	623.6	1,111.4	1,040.4	TL	621.7	TL	939.0	TL
		5×10^1	TL	TL	1,114.5	748.8	989.2	1,061.6	TL	TL	1,330.6	TL	966.5	TL
		1×10^2	TL	TL	532.3	1,007.2	634.5	607.5	TL	TL	646.7	TL	1,244.3	TL
MushroomNeg	**73.2**	1×10^1	TL	TL	1,140.8	175.8	1,056.2	TL	1,037.5	TL	955.9	TL	412.4	TL
		5×10^1	TL	TL	990.6	175.7	895.2	TL	956.4	TL	842.1	TL	901.4	TL
		1×10^2	TL	TL	936.4	762.9	631.2	TL	537.3	TL	955.3	TL	570.1	TL
Ecommerce	**57.1**	5×10^5	TL	TL	497.4	TL	1,234.4	1,323.1	43.5	TL	TL	TL	200.2	164.1
		7×10^5	TL	TL	TL	TL	128.5	1,452.5	835.3	TL	TL	TL	139.7	181.1
		1×10^6	TL	TL	TL	TL	133.7	885.2	964.1	TL	TL	TL	225.2	170.0
Pumsb 1%	**25.7**	1×10^1	TL	210.7	1,306.1	297.5	364.4	TL	362.2	TL	TL	TL	2,004.6	TL
		5×10^1	297.3	. 1,204.1	1,088.9	1,107.9	1,114.3	TL	TL	TL	TL	TL	585.8	TL
		1×10^2	TL	236.3	1,613.1	318.3	458.9	TL	TL	TL	TL	TL	2,022.1	TL
BMS 10%	**23.6**	6×10^4	TL	985.4	814.9	605.9	1,086.8	1,066.4	1,086.9	TL	1,247.9	465.4	558.7	421.2
		7×10^4	TL	660.3	1,021.3	1,035.3	873.1	888.1	992.9	TL	959.1	1,208.3	548.8	331.4
		8×10^4	TL	TL	1,139.6	848.4	1,391.4	1,373.0	822.5	TL	247.4	1,221.8	569.8	317
Connect	**570.5**	6.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		7×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	1,221.5	TL
		7.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
Retail	**217.4**	5×10^1	TL	TL	TL	TL	390.0	TL	TL	TL	TL	TL	635.7	632.4
		1×10^2	TL	TL	TL	TL	702.4	TL	TL	TL	TL	TL	2,741.4	2,236.8
		5×10^2	TL	TL	TL	TL	548.9	558.3	486.2	TL	TL	TL	534.8	446.4
Fruithut	**178.9**	5×10^2	TL	TL	TL	TL	1,195.4	TL	TL	TL	TL	TL	2,243.3	TL
		7×10^2	TL	TL	TL	TL	2,753.3	TL	TL	TL	TL	TL	2,276.2	TL
		1×10^3	TL	TL	TL	TL	2,115.9	TL	TL	TL	TL	TL	2,233.0	TL
Accidents	1,894.4	6×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	**853.7**	TL
		6.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	**849.5**	TL
		7×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	**709.0**	TL
Kosarak	**1,576.9**	1×10^2	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		5×10^3	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		1×10^6	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
ChainStore	**1,730.6**	1×10^2	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		5×10^3	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		1×10^6	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL

Table 21
Number of solutions found when mining high utility itemsets. TL stands for time limit (more than 600 s).

Dataset	TKHUIM-GA	Minimum Threshold	HUIM-GA	HUIM-BPSO	HUIM-GA-tree	HUIM-BPSO-tree	HUIF-PSO	HUIF-GA	HUIF-BA	HUIM-ABC	HUIM-SPSO	HUIM-AF	HUIM-HC	HUIM-SA
Chess	15	4×10^5	120	3,145	77	4,282	24,178	5,155	26,921	TL	14,573	8,812	0	0
		4.5×10^5	56	2,872	10	2,677	18,543	4,287	16,384	TL	8,428	4,231	0	0
		5×10^5	46	2,432	50	1,625	17,982	2,565	26,874	TL	4,836	2,255	0	0
ChessNeg	15	4×10^4	151	1,343	TL	6,101	40,063	5,395	56,006	TL	14,131	22,761	17,085	28,755
		4.5×10^4	84	0	46	5,265	36,928	5,000	47,929	TL	13,844	19,599	0	0
		5×10^4	TL	1,640	TL	4,677	31,603	5,126	41,129	TL	11,031	16,128	6,117	0
Foodmart	15	5×10^1	TL	TL	30	2,267	4,858	5,781	5,911	TL	TL	TL	3,252	8,954
		2×10^3	TL	TL	17	1,192	2,319	2,959	2,550	TL	TL	TL	946	2,848
		5×10^3	TL	TL	3	170	1,093	1,103	1,098	TL	TL	TL	94	361
Mushroom	15	1×10^1	136	TL	67	29,777	40,671	8,690	65,449	TL	14,571	TL	77,335	TL
		5×10^1	TL	TL	119	29,444	42,188	7,877	TL	TL	5,194	TL	77,691	TL
		1×10^2	TL	TL	85	29,789	43,649	9,992	TL	TL	12,715	TL	77,983	TL
MushroomNeg	15	1×10^1	TL	TL	93	21,088	39,936	TL	66,529	TL	5,983	TL	71,070	TL
		5×10^1	TL	TL	63	20,760	39,400	TL	66,791	TL	6,638	TL	69,872	TL
		1×10^2	TL	TL	74	20,673	39,329	TL	65,080	TL	4,921	TL	71,713	TL
Ecommerce	15	5×10^5	TL	TL	1	TL	0	1,040	573	TL	TL	TL	130	1,185
		7×10^5	TL	TL	TL	TL	299	619	404	TL	TL	TL	48	464
		1×10^6	TL	TL	TL	TL	166	245	190	TL	TL	TL	28	139
Pumsb 1%	15	1×10^1	TL	0	35	40,019	15,450	TL	15,600	TL	TL	TL	24,571	TL
		5×10^1	47	0	56	40,020	15,236	TL	TL	TL	TL	TL	25,376	TL
		1×10^2	TL	0	49	40,020	15,473	TL	TL	TL	TL	TL	41,745	TL
BMS 10%	15	6×10^4	TL	0	0	177	0	0	0	TL	32	0	35,908	279
		7×10^4	TL	0	0	149	165	0	0	TL	41	0	172	179
		8×10^4	TL	TL	0	114	0	0	125	TL	44	0	121	131
Connect	15	6.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		7×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	13,900	TL
		7.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
Retail	15	5×10^1	TL	TL	TL	TL	361	TL	TL	TL	TL	TL	544	1,699
		1×10^2	TL	TL	TL	TL	385	TL	TL	TL	TL	TL	528	1,751
		5×10^2	TL	TL	TL	TL	0	0	0	TL	TL	TL	180	0
Fruithut	15	5×10^2	TL	TL	TL	TL	42,526	TL	TL	TL	TL	TL	20,858	TL
		7×10^2	TL	TL	TL	TL	40,551	TL	TL	TL	TL	TL	22,102	TL
		1×10^3	TL	TL	TL	TL	36,940	TL	TL	TL	TL	TL	21,044	TL
Accidents	15	6×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	0	TL
		6.5×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	0	TL
		7×10^5	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	11,105	TL
Kosarak	15	1×10^2	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		5×10^3	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		1×10^6	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
ChainStore	15	1×10^2	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		5×10^3	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL
		1×10^6	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL	TL

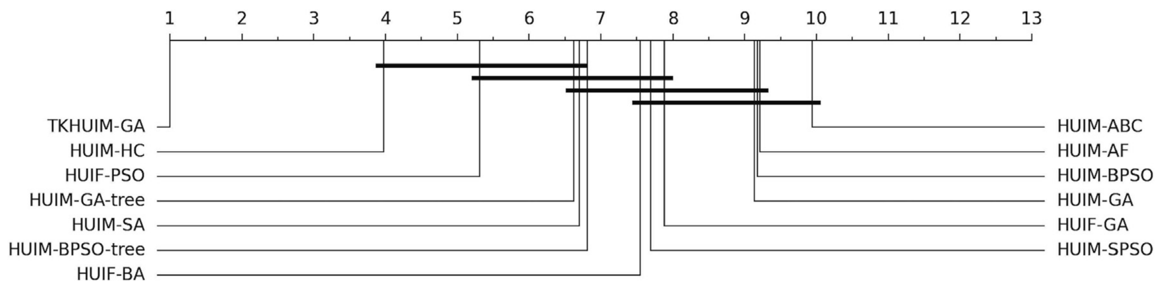


Fig. 15. Critical difference diagram showing a statistical comparison of the runtime conducted using the Shaffer's test.

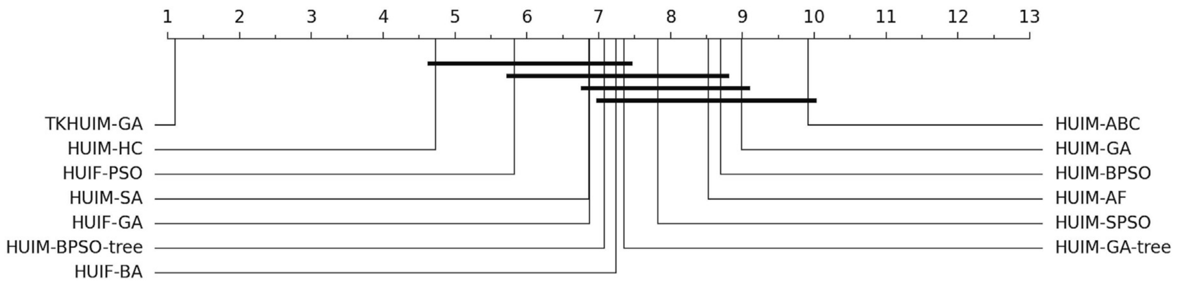


Fig. 16. Critical difference diagram showing a statistical comparison of the required memory conducted using the Shaffer's test.

Starting with the runtime (see Table 19), the Friedman's test rejected the null hypothesis that all the 13 algorithms equally behave at a significance level of $\alpha = 0.01$. Then, the Shaffer's post hoc test was performed to detect where these significant differences were located. The results for this post hoc test, at a significance level of $\alpha = 0.05$, are summarized through the critical difference diagram shown in Fig. 15, illustrating that the proposed TKHUIM-GA algorithm is the fastest, existing significant differences with the rest of algorithms. HUIM-HC is the second best algorithm, not presenting significant differences with regard to HUIF-PSO, HUIM-GA-tree and HUIM-SA. It is also important to remark that fixing an accurate minimum utility is not easy and it requires many trials and a long time as demonstrated by *Wu et al.* [37]. Here, we have considered three different utility thresholds and, for some of them, existing bio-inspired algorithms required more than 600 s (TL), specially when using large datasets (number of transactions and items). To extend these results (not included in Tables 19–21 due to the time limit), let us consider HUIM-BPSO-tree on ecommerce and a utility threshold of 500,000. This algorithm required 942.5 s, 1,158.2 MB of memory and returned 130 solutions. Additionally, the algorithm HUIM-SA on accidents data and a utility threshold of 700,000 required 2,023.9 s (far from the time limit), 1,128.4 MB of memory and returned 35,908 solutions. Two additional examples can be taken from the Mushroom negative dataset and a utility threshold of 10: HUIF-GA requires 699.9 s, a memory usage of 983.6 MB and returns 5,303 patterns; whereas HUIM-BPSO needs 1,228.4 s, 105.1 MB of memory and does not return any pattern. Finally, let us focus on large datasets in which most of the algorithms achieved the time limit (see Connect and Fruithut datasets in Table 19). The HUIM-HC algorithm, which is one of the fastest algorithms, required 913.3 s for a threshold of 6,500,000, 1,170.5 MB and returned 77,544 when it was run on Connect. As for the Fruithut dataset, considering a utility threshold of 500, the HUIM-SA algorithm returned 64,198 solutions in 840.6 s and using a memory of 1,085.9 MB. The above are some examples of runs that were included as TL in Tables 19–21 due to they overpass the time limit.

Continuing with memory usage (see Table 20), the Friedman's test rejected the null hypothesis that all the 13 algorithms equally behave at a significance level of $\alpha = 0.01$. Then, the Shaffer's post hoc test was performed to detect where these significant differences were located. The results for this post hoc test, at a significance level of $\alpha = 0.05$, are summarized through the critical difference diagram shown in Fig. 16, illustrating that the proposed TKHUIM-GA algorithm is the one that requires less memory requirements, existing significant differences with the rest of algorithms. HUIM-HC is the second best algorithm, not presenting significant differences with regard to HUIF-PSO, HUIM-SA, HUIF-GA, HUIM-BPSO-tree, HUIF-BA and HUIM-GA-tree.

4.5. Summary of the Experiments

The experimental stage has demonstrated the usefulness of the proposed TKHUIM-GA algorithm for mining top-*k* high utility patterns in different scenarios. It is able to extract solutions considering positive and negative utilities, which is not possible when using exhaustive search algorithms. Additionally, the proposal is able to work with integers and real numbers, whereas existing bio-inspired algorithms only work on integer utility values.

The proposed TKHUIM-GA algorithm obtains really good solutions, close to those obtained by exhaustive search approaches as was demonstrated. In fact, the proposal is able to extract the best solution in most of the datasets and always obtain a solution from the top-15. This is done in a much lower runtime and requires lower memory usage. In some datasets, exhaustive search approaches such as TKO returned heap space problems (more than 16 GB of memory).

Compared to bio-inspired approaches, TKHUIM-GA was able to run on any dataset and did not need any utility threshold. At this point, it is important to remind that this is the first bio-inspired algorithm for mining top-*k* utility patterns (TKU-CE and TKU-CE + are heuristic-based approaches). Additionally, the statistical analysis revealed that TKHUIM-GA was statistically better in terms of runtime and memory usage. Last but not least, it is important to remark on the usefulness of mining the best *k* patterns without needing any threshold value. Putting the right threshold value is not easy and depends on the problem (dataset) to be analysed. In fact, choosing a valid value requires different trials and the subsequent waste of time.

5. Conclusions

In this paper, we have proposed the TKHUIM-GA algorithm for mining top-*k* utility itemsets. This is a genetic algorithm that does not follow a random process, either to produce initial solutions or to combine solutions. The proposal includes a really efficient data representation that is key to improving the runtime and memory usage. TKHUIM-GA considers the utility of each item to guide the search process accordingly. Some major advantages of TKHUIM-GA are the following: it does not require large runtimes; it works on any scenario, dealing with positive, negative, integer and real unit profits; it is able to work with extremely large datasets where other exhaustive and bio-inspired approaches fail. Experimental results show that the proposed algorithm not only outperforms other state-of-the-art bio-inspired and heuristic-based HUIM algorithms, but it is also extremely efficient compared to top-*k* utility exhaustive algorithms. The main limitation of the proposed algorithm is the uncertainty that the best solutions are found since it works heuristically. In real scenarios such as medicine, when the exact solutions are required, the use of an evolutionary algorithm may be unsuitable. However, the experimental analysis has shown that very good solutions are found in a reduced time where exhaustive search approaches cannot run. As for future work, we plan to explore the use of this methodology in periodic patterns as well as high-utility itemset mining considering periodicity.

Data availability

Data will be made available on request.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

Grant PID2020-115832 GB-I00 funded by MICIN/AEI/10.13039/501100011033. This project was also supported by project UCO-FEDER 18 REF.1263116 MOD.A.

References

[1] C.C. Aggarwal, J. Han, Frequent Pattern Mining. Springer Publishing Company, 2014. ISBN: 978-3-319-07821-2.
[2] C.F. Ahmed, S.K. Tanbeer, B. Jeong, Y. Lee, Efficient tree structures for high utility pattern mining in incremental databases, IEEE Transaction on Data and Knowledge Engineering 21 (12) (2009) 1708–1721.
[3] H. Cao, S. Yang, Q. Wang, Q. Wang, L. Zhang, A closed itemset property based multi-objective evolutionary approach for mining frequent and high utility itemsets, in: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2019, Wellington, New Zealand, June 10–13, 2019, IEEE, 2019, pp. 3356–3363.
[4] Q. Duong, B. Liao, P. Fournier-Viger, T. Dam, An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies, Knowledge Based Systems 104 (2016) 106–122.
[5] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu, V.S. Tseng, SPMF: a java open-source pattern mining library, Journal of Machine Learning Research 15 (1) (2014) 3389–3393.
[6] P. Fournier-Viger, J.C.-W. Lin, R. Nkambou, B. Vo, V.S. Tseng, High-Utility Pattern Mining: Theory, Algorithms and Applications, 1st edition., Springer Publishing Company, 2019.
[7] P. Fournier-Viger, C. Wu, S. Zida, V.S. Tseng, FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning, in: Proceedings of the 21st International Symposium on Foundations of Intelligent Systems, ISMIS 2014, Roskilde, Denmark, June 25–27, 2014. Proceedings, volume 8502, Springer, 2014, pp. 83–92.
[8] M. Friedman, A comparison of alternative tests of significance for the problem of m rankings, The Annals of Mathematical Statistics 11 (1) (1940) 86–92.
[9] W. Gan, J.C. Lin, P. Fournier-Viger, H. Chao, T. Hong, H. Fujita, A survey of incremental high-utility itemset mining, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 8 (2) (2018).
[10] W. Gan, J.C. Lin, P. Fournier-Viger, H. Chao, P.S. Yu, HUOPM: high-utility occupancy pattern mining, IEEE Transactions on Cybernetics 50 (3) (2020) 1195–1208.
[11] W. Gan, J.C. Lin, J. Zhang, P. Fournier-Viger, H. Chao, P.S. Yu, Fast utility mining on sequence data, IEEE Transaction on Cybernetics 51 (2) (2021) 487–500.

[12] A. Hidouri, S. Jabbour, B. Raddaoui, M. Chebbah, B.B. Yaghlane, A declarative framework for mining top-k high utility itemsets, in: Proceedings of the 23rd International Conference on Big Data Analytics and Knowledge Discovery, DaWaK 2021, Virtual Event, September 27–30, 2021, volume 12925, Springer, 2021, pp. 250–256.
[13] S. Kannimuthu, K. Premalatha, Discovery of high utility itemsets using genetic algorithm with ranked mutation, Applied Artificial Intelligence 28 (4) (2014) 337–359.
[14] R.U. Kiran, T.Y. Reddy, P. Fournier-Viger, M. Toyoda, P.K. Reddy, M. Kitsuregawa, Efficiently finding high utility-frequent itemsets using cutoff and suffix utility, in: Proceedings of the 23rd Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD 2019, Macau, China, April 14–17, 2019, Proceedings, Part II, volume 11440, Springer, 2019, pp. 191–203.
[15] Y. Li, J. Yeh, C. Chang, Isolated items discarding strategy for discovering high utility itemsets, Data & Knowledge Engineering 64 (1) (2008) 198–217.
[16] C. Lin, T. Hong, W. Lu, An effective tree structure for mining high utility itemsets, Expert Systems with Applications 38 (6) (2011) 7419–7424.
[17] J.C. Lin, L. Yang, P. Fournier-Viger, T. Hong, M. Voznák, A binary PSO approach to mine high-utility itemsets, Soft Computing 21 (17) (2017) 5103–5121.
[18] J.C. Lin, L. Yang, P. Fournier-Viger, J.M. Wu, T. Hong, S.L. Wang, J. Zhan, Mining high-utility itemsets based on particle swarm optimization, Engineering Applications of Artificial Intelligence 55 (2016) 320–330.
[19] J. Liu, K. Wang, B.C.M. Fung, Direct discovery of high utility itemsets without candidate generation, in: Proceedings of the 12th IEEE International Conference on Data Mining, ICDM 2012, IEEE Computer Society, 2012, pp. 984–989.
[20] M. Liu, J. Qu, Mining high utility itemsets without candidate generation, in: Proceedings of the 21st ACM International Conference on Information and Knowledge Management, ACM, 2012, pp. 55–64.
[21] Y. Liu, W. Liao, A.N. Choudhary, A two-phase algorithm for fast discovery of high utility itemsets, in: Proceedings of the 9th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD 2005, Hanoi, Vietnam, May 18–20, 2005, Proceedings, volume 3518, Springer, 2005, pp. 689–695.
[22] J.M. Luna, P. Fournier-Viger, S. Ventura, Frequent itemset mining: A 25 years review, Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery 9 (6) (2019).
[23] M.S. Nawaz, P. Fournier-Viger, U. Yun, Y. Wu, W. Song, Mining high utility itemsets with hill climbing and simulated annealing. ACM Transactions on, Management Information Systems 13 (1) (2022) 4:1–4:22.
[24] H. Ryang, U. Yun, Top-k high utility pattern mining with effective threshold raising strategies, Knowledge and Based Systems 76 (2015) 109–126.
[25] J.P. Shaffer, Modified sequentially rejective multiple test procedures, Journal of the American Statistical Association 81 (395) (1986) 826–831.
[26] W. Song, C. Huang, Discovering high utility itemsets based on the artificial bee colony algorithm, in: Proceedings of the 22nd Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining, PAKDD 2018, Melbourne, VIC, Australia, June 3–6, 2018, Proceedings, Part III, volume 10939, Springer, 2018, pp. 3–14.
[27] W. Song, C. Huang, Mining high utility itemsets using bio-inspired algorithms: A diverse optimal value framework, IEEE Access 6 (2018) 19568–19582.
[28] W. Song, J. Li, Discovering high utility itemsets using set-based particle swarm optimization, in: Proceedings of the 16th International Conference on Advanced Data Mining and Applications, ADMA 2020, Foshan, China, November 12–14, 2020, Proceedings, volume 12447, Springer, 2020, pp. 38–53.
[29] W. Song, J. Li, C. Huang, Artificial fish swarm algorithm for mining high utility itemsets, in: Proceedings of the 12th International Conference on Advances in Swarm Intelligence, ICSI 2021, Qingdao, China, July 17–21, 2021, Proceedings, Part II, volume 12690, Springer, 2021, pp. 407–419.
[30] W. Song, L. Liu, C. Huang, TKU-CE: cross-entropy method for mining top-k high utility itemsets, in: Proceedings of the 33rd International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems, IEA/AIE 2020, Kitakyushu, Japan, September 22–25, 2020, 2020, pp. 846–857.
[31] W. Song, C. Zheng, C. Huang, L. Liu, Heuristically mining the top-k high-utility itemsets with cross-entropy optimization, Applied Intelligence, Published online 29 (2021), July 2021.
[32] V.S. Tseng, B. Shie, C. Wu, P.S. Yu, Efficient algorithms for mining high utility itemsets from transactional databases, IEEE Transaction on Data and Knowledge Engineering 25 (8) (2013) 1772–1786.
[33] V.S. Tseng, C. Wu, P. Fournier-Viger, P.S. Yu, Efficient algorithms for mining top-k high utility itemsets, IEEE Transactions on Knowledge and Data Engineering 28 (1) (2016) 54–67.
[34] V.S. Tseng, C. Wu, B. Shie, P.S. Yu, Up-growth: an efficient algorithm for high utility itemset mining, in: Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Washington, DC, USA, July 25–28, 2010, ACM, 2010, pp. 253–262.
[35] S. Ventura, J.M. Luna, Pattern Mining with Evolutionary Algorithms 978-3-319-33858-3, Springer, ISBN, 2016.
[36] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (6) (1945) 80–83.
[37] C. Wu, B. Shie, V.S. Tseng, P.S. Yu, Mining top-k high utility itemsets, in: Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD ’12, Beijing, China, August 12–16, 2012, ACM, 2012, pp. 78–86.
[38] L. Zhang, G. Fu, F. Cheng, J. Qiu, Y. Su, A multi-objective evolutionary approach for mining frequent and high utility itemsets, Applied Soft Computing 62 (2018) 974–986.
[39] L. Zhang, P. Luo, E. Chen, M. Wang, Revisiting bound estimation of pattern measures: A generic framework, Information Sciences 339 (2016) 254–273.
[40] L. Zhang, S. Yang, X. Wu, F. Cheng, Y. Xie, Z. Lin, An indexed set representation based multi-objective evolutionary approach for mining diversified top-k high utility patterns, Engineering Applications of Artificial Intelligence 77 (2019) 9–20.
[41] S. Zida, P. Fournier-Viger, J.C. Lin, C. Wu, V.S. Tseng, EFIM: a fast and memory efficient algorithm for high-utility itemset mining, Knowledge and Information Systems 51 (2) (2017) 595–625.