Mining Distinct Representations of High-Utility Itemsets Using Particle Swarm Optimization

Simen Carstensen

Master's thesis in Software Engineering at

Department of Computer Science, Electrical Engineering and Mathematical Sciences,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

October 2022

Western Norway
University of
Applied Sciences

Abstract

Closed high-utility itemset mining (CHUIM) and top-k high-utility itemset mining (top-k HUIM) are techniques extensively applied in data analysis to discover patterns or relationships with potentially valuable information. Although these processes are helpful in various decision-making problems, it is challenging to retrieve the solutions in a short amount of time. One way of alleviating this is to find approximate results through heuristic models. However, few heuristics are currently available, and the previous works also suffer performance limitations that impede their overall usability.

This thesis addresses these issues by introducing two distinct algorithms described in respective research papers—CHUI-PSO for CHUIM and TKU-PSO for top-k HUIM. Both algorithms are heuristic techniques based on particle swarm optimization and incorporate several unique strategies aimed at reducing the computational complexity associated with earlier methods. To evaluate the contribution of the models, we compare performance characteristics against the current state-of-the-art approaches. Experimental results indicate that the proposed algorithms outperform previous works in terms of precision and efficiency.

Acknowledgements

First and foremost, I would like to thank my supervisor Jerry Chun-Wei Lin for his professional and passionate guidance throughout this project. I could not have completed this work without you. I also want to express my gratitude to my brother, Carl-Martin, who has provided invaluable feedback. Finally, a big thanks to my friends and family for their encouragement and support.

List of papers

1. An Efficient PSO-based Evolutionary Model for Closed High-Utility Itemset Mining

   - Submitted to: *Applied intelligence - The International Journal of Research on Intelligent Systems for Real Life Complex Problems*.

   - Status: under review.

2. TKU-PSO: An Efficient Particle Swarm Optimization Model for Top-k High-Utility Itemset Mining

   - Submitted to: *IJIMAI - The International Journal of Interactive Multimedia and Artificial Intelligence*.

   - Status: under revision.

Contents

List of Tables

Chapter 1

Motivation

We live in a data-driven society where unfathomable amounts of digital information are formed in every facet of business and life. New data sources are continuously introduced, and data warehouses grow by the minute. This has created opportunities for those who can successfully interpret and transform data into relevant and usable knowledge, and a need for automatic analytical tools has thus emerged.

Data mining algorithms are specialized procedures for finding patterns or relationships within massive datasets. Their ultimate purpose is to extract information regarding potentially useful data characteristics in an easily comprehensible format. This way, raw data can be efficiently utilized to expand domain knowledge and aid decision-making processes. There is a wide variety of applications for these algorithms, such as weather forecasting [1], earthquake prediction [2], crime prevention [3], and medical analysis [4]. However, the work in this thesis is specifically aimed at patterns in transactional data.

High-utility itemset mining (HUIM) [5] is a data mining technique used to discover valuable groups of items in transactional databases and is applied for various analytical purposes. For example, businesses employ HUIM to identify profitable product combinations based on historical customer purchases and use the knowledge to predict future trends and drive further sales. Although HUIM algorithms are useful, they tend to return an overwhelming amount of information, making them unintuitive in practical use. For this reason, *closed high-utility itemset mining* (CHUIM) [6] and *top-k high-utility itemset mining* (top-k HUIM) [7] have been suggested. These procedures aim to reduce the complexity associated with result interpretation by finding a more concise and significant set of patterns, which has been proven successful in several studies [6, 8, 9, 10, 11]. However, finding these itemsets is computationally expensive, and existing methods can require substantial execution times and memory usage due to combinatorial explosion. In order to keep up with rapid data growth, more efficient algorithms are needed.

Chapter 2

Background

This chapter briefly introduces the relevant data mining techniques that inspired CHUIM and top-k HUIM before the traditional particle swarm optimization is described and related works are reviewed.

2.1 Transactional data mining

Mining patterns in transactional data is a concept popularized by Agrawal et al. [12] with the introduction of *frequent itemset mining* (FIM). FIM is a technique used to identify groups of items (itemsets) with at least a *minimum support count*, where the support count describes the number of transactions that contain the itemset. Suppose a transaction is a list of products bought by a customer at a supermarket. FIM may then be applied to identify the products commonly purchased together within a collection of transactions (database). The rationale behind finding these itemsets is that they frequently occur in the data and thus reveal valuable associations. For instance, placing correlated products in the same store aisles or using them in promotional campaigns may induce further sales. This type of data analysis is referred to as *market basket analysis* [13], which retailers use to understand customer purchasing behaviors. However, FIM has also been applied to other real-world problems, such as finding connections between educational resources and students' learning, genes and the risk of cancer, and geolocation and the spread of diseases [14].

Although FIM can be a helpful tool, it assumes the value of a pattern is defined by its frequency, which may not be the case. In market basket analysis, businesses typically want to identify the product combinations that yield the most profit, and these patterns are not necessarily among the frequent purchases. For example, the itemset $\{champagne, beef\}$ may be rarely bought but can contribute overall more revenue than a common itemset such as $\{bread, milk\}$. For this reason, Yao and Hamilton [5] introduced itemset mining based on utilities (HUIM).

Whereas FIM discovers patterns based on support, HUIM finds patterns with utility over a user-specified *minimum utility threshold*, called *high-utility itemsets* (HUIs). HUIM differs from FIM by mining quantitative databases in which

items can occur several times in the same transaction, represented by a quantity. In addition, each item has a weight that expresses its relative importance compared to other items. The utility of an itemset is based on its item weights and quantities and reflects the overall interestingness of the pattern. A key property of this strategy is that the weights can change according to the user's goals. For example, the weights can represent the profit generated by a product, the time spent on a website, or the effectiveness of a medical procedure [15]. This way, HUIs can fit a wider variety of analytical problems than the frequent patterns produced by FIM. However, a significant challenge in HUIM is to select an appropriate minimum utility threshold, as the optimal value depends on data characteristics that are typically unknown to the user. As a result, the algorithms tend to produce massive numbers of HUIs, which requires substantial computational resources and makes the results difficult to interpret.

To address these issues, Tseng et al. [6] proposed CHUIM. CHUIM reduces the number of relevant HUIs without information loss by applying the concept of *closed itemsets*, originally introduced for FIM [16]. A closed itemset is an itemset that does not have any immediate superset with the same support count in the data. In other words, an itemset is not closed if a superset appears in the same number of transactions. The rationale is that non-closed patterns are unnecessary to maintain as a superset holds their exact information. This way, the algorithm stores less redundant data and finds a smaller but significant set of HUIs. Thus, the minimum utility threshold is more forgiving in CHUIM than in HUIM, making it easier to obtain a suitable number of patterns.

Although CHUIM lessens the limitations of HUIM, the minimum utility threshold still needs to be appropriately tuned, which is generally achieved by trial and error. To solve this inconvenience, Wu et al. [7] suggested top-k HUIM to find HUIs without the minimum utility threshold. In top-k HUIM, the minimum utility threshold is replaced with an input parameter k that specifies a desired number of patterns. The algorithm then locates the k HUIs with the largest utilities in the database. This approach is arguably more intuitive than HUIM and CHUIM, as it is easier to select a suitable value for k. However, finding these itemsets are more demanding as the minimum utility threshold enables candidate pruning. Generally, the larger the threshold, the fewer candidates the algorithm has to consider. The initial search space in top-k HUIM is thus equivalent to traditional HUIM with the threshold set to zero. Nonetheless, all types of utility mining are computationally expensive procedures.

2.2 Particle swarm optimization

Evolutionary computation (EC) [17] is a sub-field of heuristic optimization techniques that utilize biological theories to find approximate solutions to large search problems. Generally, EC models stochastically optimize a set of candidate solutions by inheriting features from the most promising solutions discovered. This process repeats for a limited number of iterations, which can enable the model to find optimal solutions without exploring the search space exhaustively. One such method applied in data mining problems is *particle swarm optimization* (PSO) [18, 19]. Many versions of the algorithm exist, but its basic principles are described below.

PSO generates an initial population of randomized candidates called particles. Each particle is associated with a position vector (X) that describes the candidate's location in the solution space, and a velocity vector (V) that determines the movement direction of the coordinates in the position vector. At each iteration t, the velocity and position of a particle i are updated according to the following equations:

$$V_i^j(t+1) = wV_i^j(t) + c_1r_1(pBest_i^j - X_i^j(t)) + c_2r_2(gBest^j - X_i^j(t)) \quad (2.1)$$

$$X_i^j(t+1) = X_i^j(t) + V_i^j(t+1) \quad (2.2)$$

In Eq. (2.1) and Eq. (2.2), j is the current index in X or V; r_1 and r_2 are random numbers between 0 and 1; c_1, c_2, and w are constants; $pBest_i$ are the best previous position of particle i, and $gBest$ are the best previous position of all particles in the population.

$pBest$ and $gBest$ are historic positions maintained by the algorithm. They are determined using an objective function that evaluates particle positions in relation to some quality measure. The individual factor c_1 and social factor c_2 control the amount of influence each particle receives from $pBest$ and $gBest$, and the inertia parameter w decide to which degree each particle should maintain its current velocity. After a particle is updated, the algorithm evaluates the new position and reselects $pBest$ and $gBest$ based on the result from the objective function. The population thus moves towards the optimal value(s) by continuously adjusting particle positions according to the most promising candidates evaluated.

2.3 Related work

The algorithms proposed for utility mining can be categorized as *exact* or *heuristic*. The exact algorithms are deterministic and always find the correct solutions, whereas the heuristic methods cannot give such a guarantee. This section introduces the most relevant exact works for HUIM, CHUIM, and top-k HUIM before reviewing the heuristic alternatives.

2.3.1 Exact algorithms for HUIM

The main challenge of developing efficient HUIM models is to overcome potentially large search spaces. A database with n distinct items contains 2^n-1 HUI candidates, meaning some form of candidate pruning typically must be applied to discover the solutions within a reasonable time frame. In this regard, Liu et al. [20] provided one of the leading studies with the Two-Phase algorithm. They developed a pruning strategy based on *transaction-weighted utilities* (TWU) to avoid assessing each itemset in the database. The TWU is an upper bound on the utility of an itemset. Any candidate with TWU less than the minimum utility threshold is thus unnecessary to evaluate, as it cannot be a HUI. In the first phase of the algorithm, candidates are gradually composed by combining

the items satisfying the TWU constraint. In the second phase, the HUIs are determined by calculating the actual candidate utilities. A limitation of this approach is that the algorithm may produce candidates that do not appear in the input database. For this reason, *pattern-growth* algorithms such as IHUP [21], UP-Growth [22] and MU-Growth [23] have been proposed. These models are based on the two-phase concept but utilize a tree structure during candidate generation. The tree describes valid item combinations based on transaction occurrences and is initialized before the mining process starts.

Although the two-phase algorithms establish boundaries to the search space, they often cannot reduce the number of candidates sufficiently. In addition, the models are subject to computationally expensive database scans during the evaluation phase of the candidates. Newer models thus employ a more efficient one-phase approach without candidate generation, introduced to utility mining with HUI-Miner [24].

HUI-Miner proposed a *utility-list* data structure to store information required for calculating itemset utilities, allowing much faster evaluations than with database scans. The model performs two database scans to construct the initial utility-lists before identifying the HUIs through utility-list *join-operations*. This way, the HUIs are revealed directly without a dedicated phase for candidate generation. Moreover, utility-lists provide a new upper bound through the concept of *remaining utility*, reducing the number of necessary itemset evaluations. Benchmarks showed that the model was up to two orders of magnitude faster than the existing two-phase algorithms. Several improved variants of HUI-Miner has later been proposed, some of which are FHM [25], HUP-Miner [26] and UBP-Miner [27]. These methods apply strategies to reduce the overall cost surrounding utility-list operations, such as *estimated utility co-occurrence structure* (EUCS), *PU-Prune*, *LA-Prune* and *utility bit partition*.

Like in the two-phase paradigm, one-phase algorithms may generate irrelevant candidates. To address this, Zida et al. [28] proposed EFIM, a one-phase, pattern-growth algorithm based on a utility array structure. The model introduced *subtree utility* and *local utility* as new pruning upper bounds and used *fast utility counting* to calculate them in linear time and space. In addition, they applied transaction merging and database projection techniques to reduce the cost of database scans. EFIM demonstrated two to three orders of magnitude faster runtime than existing one-phase and two-phase algorithms while consuming less memory.

2.3.2 Exact algorithms for CHUIM

The field of CHUIM consists of algorithm extensions of previous HUIM and FIM works. The first model introduced was CHUD [6], a two-phase, pattern-growth approach using TWU pruning and a tree structure for candidate generation. The model is similar to IHUP and UP-Growth but is based on a *closed frequent itemset mining* algorithm called DCI-Closed [29]. In contrast to the HUIM methods, each node in the tree contains a *tidset* expressing all transactions a candidate appears. This way, the algorithm can filter non-closed HUIs during the exploration phase of the tree. Although CHUD demonstrated the usefulness of CHUIM, it inherited the same limitations as its two-phase HUIM relatives.

For this reason, Wu et al. [30] adopted the one-phase concept and proposed CHUI-Miner.

CHUI-Miner is a HUI-Miner extension using an *extended utility-list* to accommodate itemset support counts. It performs pruning based on TWU and remaining utility and reveals the solutions through a divide-and-conquer process. Comparisons against CHUD proved that CHUI-Miner was superior regarding execution time and memory usage. Subsequently, Fournier-Viger et al. [8] converted EFIM to CHUIM with EFIM-Closed. Compared to the HUIM model, EFIM-Closed introduced *closure jumping*, *forward closure checking*, and *backward closure checking* to identify relevant closed itemsets. The approach was not evaluated against CHUI-Miner but showed an overall larger improvement against CHUD. The most recent exact CHUIM algorithm is CLS-Miner [9]. It is another utility-list-based approach but applies additional pruning strategies compared to CHUI-Miner. More specifically, *chain-estimated utility co-occurrence pruning*, *lower branch pruning*, and *pruning by coverage*. CLS-Miner is more efficient than CHUI-Miner but cannot compete with EFIM-Closed in dense datasets due to the effectiveness of transaction merging.

2.3.3 Exact algorithms for top-k HUIM

HUIM and CHUIM prune unpromising candidates based on various utility upper bounds compared to the minimum utility threshold. Top-k HUIM does not receive a minimum utility threshold as input and thus faces additional search space challenges. These algorithms rely on threshold-raising strategies to gradually increase the minimum utility threshold during runtime. However, like in CHUIM, the underlying mining logic is adopted from HUIM.

Top-k HUIM was introduced with TKU [7], a two-phase model based on UP-Growth. TKU generates a pattern-growth tree with valid item combinations and applies five threshold-raising strategies to remove candidates with TWU pruning. Ryang and Yun [10] later improved the approach by developing another two-phase model called REPT. REPT is an extension of MU-Growth that uses more effective threshold-raising than TKU during the tree construction process, thus reducing the number of candidates. Although the model is superior to TKU, it requires an input parameter N, which is non-trivial to select.

The other algorithms in top-k HUIM are one-phase approaches. TKO [11] is a HUI-Miner extension that combines novel threshold-raising with the standard utility-list strategies. The model thus avoided the two-phase limitations of TKU and REPT and was shown to outperform both in experiments. Duong et al. [31] then introduced kHMC based on FHM. They applied three threshold-raising strategies together with EUCS and *pruning by coverage* to reduce utility-list join-operations compared to TKO, further reducing runtime and memory consumption. Later, TKEH [32] incorporated the strategies of EFIM and was shown to be up to three orders of magnitude faster than kHMC and TKO. Finally, THUI [33] introduced a *leaf itemset utility structure* to enhance the effectiveness of threshold-raising and pruning over earlier methods, which also provided huge advancements compared to kHMC and TKO.

2.3.4 Heuristic algorithms for HUIM

Whether using HUIM, CHUIM, or top-k HUIM, finding itemsets based on utilities is time-consuming. Although the algorithms mentioned in the previous sections can find the exact solutions, they depend on pruning strategies to reduce the search space. If pruning is ineffective, the algorithms are typically unable to complete within a reasonable time frame, regardless if the approach belongs to the one-phase or two-phase paradigm. For this reason, several heuristic methods have been developed for utility mining. The proposed heuristics are predominantly for traditional HUIM, and they all use TWU pruning as the sole search space reduction.

Particle swarm optimization (PSO) was introduced to HUIM by Lin et al. [19] with HUIM-BPSO. They adopted a binary particle encoding where each index in the position vector determines the presence or absence of a specific item, and the velocity vector expresses item probabilities based on previous evaluations. The algorithm starts by initializing the population using *roulette wheel selection*, which probabilistically chooses items based on their TWU. This means that items with a large TWU are more likely to appear in the initial candidates. During the iterative process, the model modifies each particle using the velocities and a sigmoid update function that ensures binary item values. Each particle is then evaluated through database scans to reveal any potential HUIs. The algorithm was later enhanced with a OR/NOR tree [34] to ensure each particle occurs in at least one transaction, inspired by the pattern-growth models. The tree increased accuracy, but both models struggle with premature convergence to local optima and tend to miss many relevant solutions.

Song and Huang [35] then proposed Bio-HUIF-PSO, which improves the earlier models in several aspects. Bio-HUIF-PSO applies roulette wheel selection on the set of discovered HUIs to determine *gBest* each iteration. The rationale here is that all solutions do not necessarily resemble the best solution. It is thus desirable to occasionally change *gBest* to explore a broader range of particles, which can alleviate local optima when the best solution is found. The authors also suggested particle updates with *bit difference* rather than the traditional velocity logic. Compared to the velocity, bit difference selects a set of items to change based on distinct dissimilarities between particles and *pBest/gBest*. While the velocity may modify all items in a particle, bit difference only considers the most promising changes, which typically promotes faster convergence. In addition, the individual factor, social factor, and inertia parameters are no longer required, reducing the complexity of the model. Finally, Bio-HUIF-PSO introduced a procedure called *promising encoding vector check* (PEV-check) to identify valid item combinations, similar to the OR/NOR tree. The PEV-check performs validation through bit operations on *tidsets* and does not require an initial construction process like the tree approaches. It is thus an overall more efficient procedure.

More recently, Song and Li proposed a set-based PSO for HUIM called HUIM-SPSO [36]. HUIM-SPSO transforms the velocity vector into a discrete set with operational pairs, describing the inclusion or deletion of specific items. The intention is to target the promising items more consistently during particle updates. The algorithm demonstrated superior performance to HUIM-BPSO with

and without the OR/NOR tree. However, Bio-HUIF-PSO is more effective in terms of runtime and accuracy. The main problem of HUIM-SPSO is that the algorithm does not consider the sizes of previous solutions when updating particles. Instead, it generates a random integer to determine the number of items to include, leading to many irrelevant candidates. The issue is slightly alleviated as the algorithms adopt the PEV-check strategy.

The genetic algorithm (GA) is another EC-based method applied in HUIM. Kannimuthu and Premalatha [37] developed HUPE$_{UMU}$-GRAM, which iteratively updates a population of *chromosomes* (candidates) towards the optimal values using *selection*, *mutation*, and *crossover* operations. Although the model proved the feasibility of GA in HUIM, it performs worse than the PSO-based approaches. The standard update operations in GA are computationally demanding and easily lead to local optima. This was later improved with Bio-HUIF-GA [35], which uses the strategies of Bio-HUIF-PSO to increase population diversity, avoid irrelevant candidates, and reduce execution time. Zhang et al. [38] then proposed HUIM-IGA to expand the search space exploration using *neighborhood exploration*, *population diversity maintenance*, *individual repair*, and *elite strategy*. These techniques aim to induce more population diversity to lessen the probability of local optima in the iterative process. Overall, the recent GA-based algorithms provide slightly higher accuracy than the PSO-based algorithms but use more time.

Several other types of EC have also been introduced for HUIM, such as ant colony optimization [39], artificial bee colony algorithm [40], bat algorithm [35], and artificial fish swarm algorithm [41]. There are also heuristic models not based on EC, including hill climbing and simulated annealing [42]. However, Bio-HUIF-PSO remains one of the state-of-the-art approaches, together with Bio-HUIF-GA and HUIM-IGA.

2.3.5 Heuristic algorithms for CHUIM and top-k HUIM

Although heuristics have shown promise in HUIM, few approaches are available for CHUIM and top-k HUIM. In 2022, Pramanik and Goswami [43] developed CHUI-AC, an ant colony optimization (ACO) model for CHUIM. The algorithm transforms the solutions space into a directed graph that is explored using local- and global pheromone rules, inspired by the real-world movements of ants. It showed an advantage in runtime comparisons against CHUI-Miner and CHUD but missed many relevant itemsets.

As for top-k HUIM, TKU-CE+ [44] is currently the only heuristic approach. It is a combinatorial optimization technique based on cross-entropy that updates a set of samples towards the optimal values using a utility probability distribution. The authors also introduced a *critical utility value* that enables TWU pruning before storing the database in memory, thus reducing memory usage and candidate evaluation time. The model was compared to TKU, TKO, and kHMC and demonstrated better runtimes and memory usage, although for a limited range of k.

Table 2.1 provides an overview of the discussed algorithms for CHUIM and top-k HUIM.

Table 2.1: Summary of algorithms for CHUIM and top-k HUIM

Algorithm	Type	Base-algorithm	Year
Closed High-Utility Itemset Mining			
CHUD [6]	Exact (Two-phase)	DCI_Closed [29]	2011
CHUI-Miner [30]	Exact (One-phase)	HUI-Miner [24]	2015
EFIM-Closed [8]	Exact (One-phase)	EFIM [28]	2016
CLS-Miner [9]	Exact (One-phase)	FHM [25]	2018
CHUI-AC [39]	Heuristic	ACO [45]	2022
Top-k High-Utility Itemset Mining			
TKU [7]	Exact (Two-phase)	UP-Growth [22]	2012
REPT [10]	Exact (Two-phase)	MU-Growth [23]	2015
TKO [11]	Exact (One-phase)	HUI-Miner [24]	2015
kHMC [31]	Exact (One-phase)	FHM [25]	2016
TKEH [32]	Exact (One-phase)	EFIM [28]	2019
THUI [33]	Exact (One-phase)	HUI-Miner [24]	2019
TKU-CE+ [44]	Heuristic	Cross-entropy [46]	2021

2.4 Research gap

Heuristics are a vital research topic in data mining as they alleviate the computational burden associated with analyzing massive datasets, which can be crucial for swift decision-making. However, most heuristic studies target traditional HUIM, and the field of CHUIM and top-k HUIM are still largely unexplored, although they provide obvious advantages. The existing works also share two commonalities that limit their usability:

- They tend to use excessive time as they evaluate all candidates generated. Candidate evaluations are computationally expensive as the algorithm must scan the database to determine the itemset's utility. The execution time is thus closely related to the database size and number of evaluations performed. In order to compete with the superior pruning strategies of modern non-heuristic methods, there is a need for techniques that can assess candidate quality without using database scans.
- They tend to provide inadequate accuracy in large search spaces. As the search space grows, it is increasingly challenging to generate suitable initial candidates. If they share few similarities with the actual solutions, the model typically falls into local optima before creating appropriate candidates. Thus, there is a need for models with better pruning- and population initialization strategies.

2.5 Research question

The main goal of this master thesis is to investigate whether the current methods for CHUIM and top-k HUIM can be improved. More specifically, can we develop a PSO-based algorithm that addresses the performance limitations of existing exact- and heuristic approaches?

2.6 Research method

Given that the performance of pattern mining algorithms is distinctly measurable, a quantitative research method is appropriate to evaluate the research question. We assessed the performance of the developed models by comparing metrics such as execution time, accuracy, and memory usage against existing heuristic and non-heuristic approaches. The results were collected on 8 datasets in total, provided by the SPMF data mining library [47]. The datasets have been extensively applied in earlier pattern mining studies and consist of real-world and synthetic data prepared for utility mining. On each dataset, we tested up to 6 different input parameters (minimum utility threshold and k) to observe the model characteristics over a range of scenarios. In addition, the algorithms performed up to 10 runs on the same input parameters to ensure reliable measurements through average and median values.

Chapter 3

Results

The work in this thesis consists of two research papers related to heuristic discovery of CHUIs and top-k HUIs. This chapter presents the contributions and findings of the papers.

3.1 An Efficient PSO-based Evolutionary Model for Closed High-Utility Itemset Mining

This paper introduces CHUI-PSO, a PSO model for CHUIM that employs three new strategies to target the limitations of existing heuristic works, described in section 2.4. In order to reduce the number of necessary candidate evaluations, the algorithm uses two techniques to find approximate itemset utilities in linear time, which we call *maximum-* and *average estimates*. The estimates provide upper bounds on the utility of itemsets and enable the model to bypass database scans for certain unpromising candidates. In addition, a structure called *explored-set* is proposed to maintain all candidates generated during runtime. The purpose of the set is to identify whether a newly created particle has been previously evaluated. This way, the algorithm can avoid reassessing redundant particles, further reducing the number of database scans. Moreover, the explored-set is employed during the update procedure to alter explored particles and induce population diversity. Finally, the paper suggests *extended TWU pruning* (ETP) to improve the model's capabilities in large search spaces. ETP is a recursive extension of the traditional TWU pruning [20] that can identify additional unnecessary candidates before the mining process starts.

In the paper, the performance of CHUI-PSO is assessed through comparisons with EFIM-Closed [8], CLS-Miner [9] and the heuristic CHUI-AC [43]. The results showed that the new model was the fastest overall. In total, CHUI-PSO used just over 4 minutes to finish the test on all datasets, while EFIM-Closed, CLS-Miner, and CHUI-AC took 31 minutes, 8 hours, and 41 hours, respectively. In addition, experiments without ETP demonstrated that the new pruning could significantly reduce the runtime in large search spaces. In terms of accuracy, CHUI-PSO discovered on average 98.8% of the correct itemsets compared to 46.8% with the heuristic CHUI-AC. Moreover, the results showed that the pro-

posed model found more solutions with ETP than without. We also compared the convergence rate of the heuristic models, which revealed that CHUI-PSO always converged in fewer iterations than CHUI-AC while also discovering additional patterns. Finally, candidate comparisons between ETP and the traditional TWU pruning displayed that the new method removed more unpromising items from each database, thus providing further search space reduction.

3.2 TKU-PSO: An Efficient Particle Swarm Optimization Model for Top-k High-Utility Itemset Mining

This paper focuses on the field of top-k HUIM and presents another PSO-Based model. TKU-PSO is built upon the same framework as CHUI-PSO and adopts *average estimates* and the *explored-set* to reduce the number of necessary particle evaluations. In addition, the algorithm introduces a new population initialization strategy that differs from earlier works by generating the first candidates deterministically rather than stochastically. The process selects the initial population based on known itemset utilities to increase the likelihood of suitable candidates in huge search spaces. The paper also proposes a threshold-raising concept called *minimum solution fitness*, which the model utilizes to remove unpromising items during particle updates, thus reducing the number of possible candidates.

In the experiments, TKU-PSO was compared to an improved version of TKO (TKO+) [11], THUI [33], and the heuristic TKU-CE+ [44]. The runtime results demonstrated that TKU-PSO always outperformed the other algorithms. Most notably, THUI and TKO+ could not complete tests on one of the datasets due to excessive execution times (over 14 hours), while our model finished in less than 10 seconds. In the other datasets, TKU-PSO was up to 63, 282, and 390 times faster than THUI, TKU-CE+, and TKO+, respectively. The accuracy comparisons revealed that TKU-PSO discovered overall 99.8% of the correct top-k HUIs while TKU-CE+ found 16.5%. In addition, we tested our model with the traditional population initialization strategy, which confirmed that the new method significantly increased accuracy in scenarios with large search spaces. Finally, we measured the memory usage of the algorithms and showed that our model was the most efficient in 4 of 6 datasets.

Chapter 4

Conclusion

The goal of this master thesis was to determine whether the existing methods for CHUIM and top-k HUIM could be improved with new models. To answer this question, we identified limiting factors in previous works and developed two distinct PSO-based algorithms. The model contributions were then measured by comparing performance characteristics with state-of-the-art approaches in the literature.

Both papers show that the new models can find itemsets with significantly higher precision and efficiency than existing heuristics in CHUIM and top-k HUIM. In addition, the results indicate that the proposed strategies are fundamental to the overall performance of the developed algorithms. Based on this, we can conclude that the work in this thesis improves the current heuristics. Nonetheless, we acknowledge that there is always room for additional testing, and our results may not be indicative of performance in all possible scenarios. However, given the time required to collect reliable data, our experiments include a diverse group of datasets with input parameters reflecting realistic real-world use.

Concerning exact algorithms, our results show they are generally effective at search space pruning, particularly in CHUIM. However, this is not always the case, and CHUI-PSO and TKU-PSO can provide tremendous reductions to runtimes while discovering most or all solutions. Despite this, it is difficult to gauge whether the proposed models directly improve the exact algorithms. The overall benefit of using heuristics depends on the speedup compared to the loss in precision. Therefore, more extensive testing is needed to determine if the proposed models are reliable alternatives, especially on larger datasets, which are the primary application of heuristics. Nonetheless, we have demonstrated that it is possible to find the correct patterns more efficiently than the current state-of-the-art models.

Chapter 5

Further work

Although the developed algorithms show promising results, there are still several opportunities for improving performance in utility mining. Currently, the limiting factor to the speed of heuristics is the time required for candidate evaluations through database scans. Further work should thus focus on strategies that can reduce this cost, either by omitting certain evaluations as proposed in our papers or by developing a database projection that can be more efficiently scanned. Another possible research direction is to investigate whether the utility of discovered itemsets can be utilized to retrieve the utility of new candidates.

Regarding accuracy, we have shown that pruning and population initialization can significantly influence the algorithm's ability to find the correct solutions. Further development of these procedures can thus be promising topics to explore. Specifically, work can be conducted to determine the feasibility of using estimates for search space reduction and better candidate generation.

Bibliography

[1] Folorunsho Olaiya and Adesesan Barnabas Adeyemo. "Application of data mining techniques in weather prediction and climate change studies." *International Journal of Information Engineering and Electronic Business* 4.1 (2012), pp. 51–59.

[2] Qianlong Wang, Yifan Guo, Lixing Yu, and Pan Li. "Earthquake prediction based on spatio-temporal data mining: an LSTM network approach." *IEEE Transactions on Emerging Topics in Computing* 8.1 (2017), pp. 148–158.

[3] Hsinchun Chen, Wingyan Chung, Jennifer Jie Xu, Gang Wang, Yi Qin, and Michael Chau. "Crime data mining: a general framework and some examples." *Computer* 37.4 (2004), pp. 50–56.

[4] Riccardo Bellazzi and Blaz Zupan. "Predictive data mining in clinical medicine: current issues and guidelines." *International journal of medical informatics* 77.2 (2008), pp. 81–97.

[5] Hong Yao and Howard J. Hamilton. "Mining itemset utilities from transaction databases." *Data & Knowledge Engineering* 59.3 (2006), pp. 603–626.

[6] Cheng Wei Wu, Philippe Fournier-Viger, Philip S. Yu, and Vincent S. Tseng. "Efficient Mining of a Concise and Lossless Representation of High Utility Itemsets." In: *2011 IEEE 11th International Conference on Data Mining*. 2011, pp. 824–833.

[7] Cheng Wei Wu, Bai-En Shie, Vincent S. Tseng, and Philip S. Yu. "Mining Top-K High Utility Itemsets." In: *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2012, pp. 78–86.

[8] Philippe Fournier-Viger, Souleymane Zida, Jerry Chun-Wei Lin, Cheng-Wei Wu, and Vincent S. Tseng. "EFIM-Closed: Fast and Memory Efficient Discovery of Closed High-Utility Itemsets." In: *Machine Learning and Data Mining in Pattern Recognition*. 2016, pp. 199–213.

[9] Thu-Lan Dam, Kenli Li, Philippe Fournier-Viger, and Quang-Huy Duong. "CLS-Miner: efficient and effective closed high-utility itemset mining." *Frontiers of Computer Science* 13 (2018), pp. 357–381.

[10] Heungmo Ryang and Unil Yun. "Top-k high utility pattern mining with effective threshold raising strategies." *Knowledge-Based Systems* 76 (2015), pp. 109–126.

[11] Vincent Tseng, Cheng-Wei Wu, Philippe Fournier-Viger, and Philip Yu. "Efficient Algorithms for Mining Top-K High Utility Itemsets." *IEEE Transactions on Knowledge and Data Engineering* 28 (2015), pp. 54–67.

[12] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. "Mining Association Rules between Sets of Items in Large Databases." *SIGMOD Record* 22.2 (1993), pp. 207–216.

[13] Manpreet Kaur and Shivani Kang. "Market Basket Analysis: Identify the Changing Trends of Market Data Using Association Rule Mining." *Procedia Computer Science* 85 (2016), pp. 78–85.

[14] José Maria Luna, Philippe Fournier-Viger, and Sebastián Ventura. "Frequent itemset mining: A 25 years review." *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery* 9.6 (2019).

[15] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Tin Truong-Chi, and Roger Nkambou. "A Survey of High Utility Itemset Mining." In: *High-Utility Pattern Mining: Theory, Algorithms and Applications*. 2019, pp. 1–45.

[16] Nicolas Pasquier, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. "Discovering Frequent Closed Itemsets for Association Rules." In: *Proceedings of International Conference on Database Theory*. 1999, pp. 398–416.

[17] David B. Fogel. "What is evolutionary computation?" *IEEE spectrum* 37.2 (2000), pp. 26–32.

[18] James Kennedy and Russell Eberhart. "Particle swarm optimization." In: *Proceedings of International Conference on Neural Networks*. Vol. 4. 1995, pp. 1942–1948.

[19] Jerry Chun-Wei Lin, Lu Yang, Philippe Fournier-Viger, Jimmy Ming-Thai Wu, Tzung-Pei Hong, Leon Shyue-Liang Wang, and Justin Zhan. "Mining high-utility itemsets based on particle swarm optimization." *Engineering Applications of Artificial Intelligence* 55 (2016), pp. 320–330.

[20] Ying Liu, Wei-keng Liao, and Alok Choudhary. "A Fast High Utility Itemsets Mining Algorithm." In: *Proceedings of the 1st International Workshop on Utility-Based Data Mining*. 2005, pp. 90–99.

[21] Chowdhury Farhan Ahmed, Syed Khairuzzaman Tanbeer, Byeong-Soo Jeong, and Young-Koo Lee. "Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases." *IEEE Transactions on Knowledge and Data Engineering* 21.12 (2009), pp. 1708–1721.

[22] Vincent Tseng, Cheng-Wei Wu, Bai-En Shie, and Philip Yu. "UP-Growth: An efficient algorithm for high utility itemset mining." In: *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 2010, pp. 253–262.

[23] Unil Yun, Heungmo Ryang, and Keun Ho Ryu. "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates." *Expert Systems with Applications* 41.8 (2014), pp. 3861–3878.

[24] Mengchi Liu and Junfeng Qu. "Mining High Utility Itemsets without Candidate Generation." In: *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*. 2012, pp. 55–64.

[25] Philippe Fournier-Viger, Cheng-Wei Wu, Souleymane Zida, and Vincent S. Tseng. "FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning." In: *Foundations of Intelligent Systems*. 2014, pp. 83–92.

[26] Srikumar Krishnamoorthy. "Pruning strategies for mining high utility itemsets." *Expert Systems with Applications* 42.5 (2015), pp. 2371–2381.

[27] Peng Wu, Xinzheng Niu, Philippe Fournier-Viger, Cheng Huang, and Bing Wang. "UBP-Miner: An efficient bit based high utility itemset mining algorithm." *Knowledge-Based Systems* 248 (2022), p. 108865.

[28] Souleymane Zida, Philippe Fournier-Viger, Jerry Chun-Wei Lin, Cheng-Wei Wu, and Vincent Tseng. "EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining." In: *Advances in Artificial Intelligence and Soft Computing*. 2015, pp. 530–546.

[29] Claudio Lucchese, Salvatore Orlando, and Raffaele Perego. "Fast and memory efficient mining of frequent closed itemsets." *IEEE Transactions on Knowledge and Data Engineering* 18.1 (2006), pp. 21–36.

[30] Cheng-Wei Wu, Philippe Fournier-Viger, Jia-Yuan Gu, and Vincent S. Tseng. "Mining closed+ high utility itemsets without candidate generation." In: *2015 Conference on Technologies and Applications of Artificial Intelligence*. 2015, pp. 187–194.

[31] Quang-Huy Duong, Bo Liao, Philippe Fournier-Viger, and Thu-Lan Dam. "An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies." *Knowledge-Based Systems* 104 (2016), pp. 106–122.

[32] Kuldeep Singh, Shashank Sheshar Singh, Ajay Kumar, and Bhaskar Biswas. "TKEH: An Efficient Algorithm for Mining Top-k High Utility Itemsets." *Applied Intelligence* 49.3 (2019), pp. 1078–1097.

[33] Srikumar Krishnamoorthy. "Mining top-k high utility itemsets with effective threshold raising strategies." *Expert Systems with Applications* 117 (2019), pp. 148–165.

[34] Jerry Chun-Wei Lin, Lu Yang, Philippe Fournier-Viger, Tzung-Pei Hong, and Miroslav Voznak. "A Binary PSO Approach to Mine High-Utility Itemsets." *Soft Comput.* 21.17 (2017), pp. 5103–5121.

[35] Wei Song and Chaomin Huang. "Mining High Utility Itemsets Using Bio-Inspired Algorithms: A Diverse Optimal Value Framework." *IEEE Access* 6 (2018), pp. 19568–19582.

[36] Wei Song and Junya Li. "Discovering High Utility Itemsets Using Set-Based Particle Swarm Optimization." In: *Advanced Data Mining and Applications*. 2020, pp. 38–53.

[37] Kannimuthu Subramanian and Kandasamy Premalatha. "Discovery of High Utility Itemsets Using Genetic Algorithm with Ranked Mutation." *Applied Artificial Intelligence* 28.4 (2014), pp. 337–359.

[38] Qiang Zhang, Wei Fang, Jun Sun, and Quan Wang. "Improved genetic algorithm for high-utility itemset mining." *IEEE Access* 7 (2019), pp. 176799–176813.

[39] Jimmy Ming-Tai Wu, Justin Zhijun Zhan, and Jerry Chun-Wei Lin. "An ACO-based approach to mine high-utility itemsets." *Knowl. Based Syst.* 116 (2017), pp. 102–113.

[40] Wei Song and Chaomin Huang. "Discovering High Utility Itemsets Based on the Artificial Bee Colony Algorithm." In: *Advances in Knowledge Discovery and Data Mining*. 2018, pp. 3–14.

[41] Wei Song, Junya Li, and Chaomin Huang. "Artificial Fish Swarm Algorithm for Mining High Utility Itemsets." In: *Advances in Swarm Intelligence*. 2021, pp. 407–419.

[42] Muhammad Saqib Nawaz, Philippe Fournier-Viger, Unil Yun, Youxi Wu, and Wei Song. "Mining High Utility Itemsets with Hill Climbing and

Simulated Annealing." *ACM Transactions on Management Information Systems* 13.1 (2021), pp. 1–22.

[43] Subhadip Pramanik and Adrijit Goswami. "Discovery of closed high utility itemsets using a fast nature-inspired ant colony algorithm." *Applied Intelligence* 52 (2022), pp. 1–17.

[44] Wei Song, Chuanlong Zheng, Chaomin Huang, and Lu Liu. "Heuristically mining the top-k high-utility itemsets with cross-entropy optimization." *Applied Intelligence* (2021), pp. 1–16.

[45] Marco Dorigo, Mauro Birattari, and Thomas Stutzle. "Ant colony optimization." *IEEE computational intelligence magazine* 1.4 (2006), pp. 28–39.

[46] Pieter-Tjerk De Boer, Dirk P. Kroese, Shie Mannor, and Reuven Y. Rubinstein. "A tutorial on the cross-entropy method." *Annals of operations research* 134.1 (2005), pp. 19–67.

[47] Philippe Fournier-Viger, Jerry Chun-Wei Lin, Antonio Gomariz, Ted Gueniche, Azadeh Soltani, Zhihong Deng, and Hoang Thanh Lam. "The SPMF Open-Source Data Mining Library Version 2." In: *Machine Learning and Knowledge Discovery in Databases - European Conference*. Vol. 9853. 2016, pp. 36–40.

Appendix A

Source code

CHUI-PSO: `https://github.com/Simencar/CHUI-PSO`

TKU-PSO: `https://github.com/Simencar/TKU-PSO`

Appendix B

An Efficient PSO-based Evolutionary Model for Closed High-Utility Itemset Mining

An Efficient PSO-based Evolutionary Model for Closed High-Utility Itemset Mining

Simen Carstensen[1] and Jerry Chun-Wei Lin[2*†]

[1*]University of Bergen, Bergen, Norway.
[2]Western Norway University of Applied Sciences, Bergen, Norway.

*Corresponding author(s). E-mail(s): jerrylin@ieee.org;
Contributing authors: Simencarstensen@gmail.com;
†Corresponding author.

Abstract

High-utility itemset mining (HUIM) is a widely adopted data mining technique for discovering valuable patterns in transactional databases. Although HUIM can provide useful knowledge in various types of data, it can be challenging to interpret the results when many patterns are found. To alleviate this, closed high-utility itemset mining (CHUIM) has been suggested, which provides users with a more concise and meaningful set of solutions. However, CHUIM is a computationally demanding task, and current approaches can require prolonged runtimes. This paper aims to solve this problem and proposes a meta-heuristic model based on particle swarm optimization (PSO) to discover CHUIs, called CHUI-PSO. Moreover, the algorithm incorporates several new strategies to reduce the computational cost associated with similar existing techniques. First, we introduce Extended TWU pruning (ETP), which aims to decrease the number of possible candidates to improve the discovery of solutions in large search spaces. Second, we propose two utility upper bounds to estimate itemset utilities and bypass expensive candidate evaluations. Finally, to increase population diversity and prevent redundant computations, we utilize a structure called ExploredSet to maintain all evaluated particles. Experimental results show that CHUI-PSO outperforms the state-of-the-art algorithms regarding execution time, accuracy, and convergence.

Keywords: evolutionary computation, closed high-utility itemset, data mining, optimization, pso

1 Introduction

Knowledge discovery in databases (KDD) is a process for extracting new and potentially helpful information in the form of patterns or relationships hidden in large amounts of data. As database exploration can require enormous resources, data analysts rely on various algorithms to efficiently extract the desired type of knowledge. In the last decades, several data mining techniques have been developed to find a variety of patterns in all kinds of data, including frequent itemset mining (FIM) and association rule mining (ARM) [1]. FIM aims to obtain a set of item combinations (itemsets) that occur no less than a user-specified minimum support threshold. ARM is a similar approach but requires an additional minimum confidence threshold to identify relationships with a certain conditional probability. FIM and ARM are commonly applied to find the most frequent customer purchases in retail transactional data. However, both methods assume that the importance of each distinct item is identical. As a result, the provided solutions can occur with high frequency but may only contribute a small portion to some other significant measure, such as profit. Therefore, these patterns can be of little interest from a business perspective.

To avoid the limitations of FIM/ARM and increase the usefulness of the patterns, the concept of high-utility itemset mining (HUIM) has been proposed [2]. HUIM differs from previous approaches by using quantitative databases in which each item can occur more than once in a transaction. In addition, each item is associated with a weight representing its relative importance or value. In HUIM, the item quantities and weights are combined to obtain a utility value of the itemsets in the database. An itemset is considered a high-utility itemset (HUI) if its utility is no less than a minimum utility threshold defined by the user. In the context of retail transactional data, the utility value typically represents the total revenue generated by the itemset. This way, HUIM can discover the most profitable patterns, unlike the frequent patterns found by FIM and ARM.

Although there has been extensive research on HUIM [4–13, 18–22], the usefulness of the algorithms is often limited by the fact that they generate vast amounts of HUIs. Consequently, it can be challenging to interpret and analyze the results efficiently. Moreover, mining large sets of patterns may degrade algorithm performance as runtime and memory requirements grow. To address these concerns, closed high utility itemset mining (CHUIM) has been proposed [3]. CHUIM strongly resembles HUIM but aims to reduce the total number of HUIs by using a closed itemset property. This property states that an itemset is closed if it has no immediate superset extension with the same support value. As any such superset has higher utility than the subset, it is desirable to neglect the subset as it contains redundant information. This way, CHUIM can produce a smaller but more significant set of HUIs, alleviating the challenges described above.

However, HUIM and CHUIM are both considered challenging tasks as the computational complexity can grow exponentially on the input. Therefore, it may require immense computational resources to retrieve all patterns when the database is large. This has motivated the adoption of various heuristic models for utility mining, in particular evolutionary computation (EC). EC-based algorithms are optimization techniques that utilize biological evolutionary principles to find approximate solutions to large search problems. Genetic algorithm (GA), particle swarm optimization (PSO), ant colony optimization (ACO), and bat algorithm (BA) are some examples of EC models that have been extensively applied in various utility mining problems [4, 18–23, 25]. However, most existing studies target HUIM, and the potential of EC in CHUIM is still largely unexplored. There is also a general need for more efficient EC models, as the existing approaches tend to waste resources by evaluating many redundant and unpromising candidates. In addition, they are prone to premature converge to local optima, which affects the algorithm's ability to discover all available solutions.

This paper addresses all the above challenges by presenting a meta-heuristic EC-based PSO model for discovering CHUIs, called CHUI-PSO. As far as we know, there is no existing research on using PSO for CHUIM. The main contributions of this work can be summarized as follows:

- The CHUI-PSO algorithm for CHUIM is proposed. Apart from adopting the bio-inspired framework of Song and Huang [4], three main strategies are developed to improve the overall performance of utility mining algorithms. First, to reduce the required search space of the algorithm, we propose a technique called Extended TWU Pruning (ETP). Second, to avoid expensive candidate evaluation of low-utility items, we utilize Fitness Estimates to find approximate itemsets utilities. Finally, we suggest a structure called ExploredSet to prevent redundant evaluations and improve population diversity during particle updates.
- Extensive experiments are performed on a mixture of real and synthetic databases to evaluate the performance of the proposed model against existing heuristic and non-heuristic approaches. In addition, ETP is assessed by testing CHUI-PSO with the traditional TWU model and the new method.

2 Related work

This section briefly reviews work related to high-utility itemset mining (HUIM), closed high-utility itemset mining (CHUIM), and evolutionary computation (EC) in three sections, respectively.

2.1 High-utility itemset mining (HUIM)

Over the years, many algorithms have been developed to solve the problem of HUIM. In 2004, Liu et al. proposed a two-phase model [5] using a transaction-weighted downward closure property (TWU model) to overcome

the combinatorial explosion which occurred in earlier algorithms. If a database contains n distinct items, there are a total of $2^n - 1$ distinct itemsets. It can therefore be an incredibly challenging task to examine all candidates when n is large. The TWU model calculates upper bounds on the utility of itemsets to identify unpromising items, which are removed from the database to reduce the required search space. In the first phase of the algorithm, all itemsets that satisfy the TWU constraint are generated, while the second phase calculates the exact itemset utilities to determine the actual HUIs. IHUP [6] and UP-Growth [7] are both models based on this two-phase concept but employ tree-based structures to improve the overall performance. However, the two-phase approach with candidate generation is generally considered inefficient because too many low-utility candidates are explored during the mining process.

For this reason, HUI-Miner [8] developed a database projection technique called utility-list, which allows further search space pruning, faster utility calculations, and discovery of HUIs in one phase without candidate generation. As a result, the algorithm significantly outperformed the previous approaches. FHM [9] later improved on this idea by using estimated utility co-occurrence pruning to reduce the number of necessary join operations on the utility-lists of unpromising itemsets. This decreased execution times as the join operations can be costly to perform. EFIM [10] then introduced subtree utility and local utility as new utility upper bounds and used fast utility counting to compute them in linear time. In addition, the algorithm reduced the overall cost of database scans with high-utility transaction merging and high-utility database projection. EFIM is often considered the fastest algorithm for mining the complete set of HUIs. However, superior alternatives exist for sparse databases [11], as transaction merging is most effective in dense scenarios with many similar transactions. Nonetheless, HUI-PR, [12] has recently reduced the overall memory consumption and execution time of the EFIM approach through additional pruning strategies. Finally, in 2022 UBP-Miner [13] integrated a bit-partition utility list structure to speed up the utility-list construction process and was shown to outperform HUI-Miner.

2.2 Closed high-utility itemset mining (CHUIM)

CHUIM was originally proposed by Tseng et al. [3] to address the overwhelming number of patterns returned by HUIM algorithms. They developed three separate models for CHUIM, namely AprioriHC, AprioriHC-D, and CHUD. CHUD is the most efficient, but it is a two-phase model that relies on the TWU model for search space pruning. It was therefore outperformed when CHUI-Miner [14] adopted the one-phase concept from more recent HUIM algorithms. The model uses an extended utility-list for tighter utility upper bounds and faster itemset evaluations, while the CHUIs are discovered using a divide-and-conquer approach without producing candidates. Fournier-Viger et al. then released an extension of EFIM for CHUIM, named EFIM-Closed [15]. Compared to the HUIM model, EFIM-Closed provides additional pruning strategies

with closure jumping, forward closure checking, and backward closure checking to identify the relevant closed itemsets quickly during mining. Finally, Dam et al. proposed improvements to the utility list approach of CHUI-Miner with the CLS-Miner algorithm [16]. They introduced chain-estimated utility co-occurrence pruning, lower branch pruning, and pruning by coverage to remove unpromising candidates without fully constructing their utility-lists. An efficient pre-check method was also proposed to perform subsumption checks and closure computations. As far as we know, EFIM-Closed and CLS-Miner are currently the most advanced approaches for non-heuristic discovery of CHUIs. However, the effectiveness of each model varies depending on the database characteristics. EFIM-Closed generally prefers dense databases, while CLS-Miner performs best on sparse. Both algorithms may be unable to find the solutions in a reasonable amount of time if the search space is large, regardless of the database type.

2.3 Evolutionary computation (EC)

Evolutionary Computation (EC) attempts to overcome the computational limitations of non-heuristic models by utilizing past experiences to explore the search space efficiently. Generally, an initial set of candidate solutions is produced and updated by inheriting traits from the most promising solutions. This process repeats for a limited number of generations to achieve a good tradeoff between runtime and accuracy. One such EC algorithm is particle swarm optimization (PSO). The traditional PSO [17] maintains a population of particles representing potential solutions. Each particle is assigned a fitness value, which indicates the quality of the solution, and a velocity vector that determines how the particle should evolve. The velocity is calculated based on the distance to the particle's personal fittest offspring (pBest) and the all-time fittest particle in the entire population (gBest). At each iteration of the algorithm, the population is updated based on the velocities, and pBest and gBest are redetermined. As a result, the particles tend to naturally evolve toward fitter solutions as features of pBest and gBest are favored.

Lin et al. have proposed two PSO-based models for HUIM, namely HUIM-BPSO$_{sig}$ [18] and HUIM-BPSO [19]. HUIM-BPSO differs from HUIM-BPSOsig by using a OR/NOR tree structure during particle creation. The OR/NOR tree ensures all generated itemsets appear in the database, and evaluations of irrelevant solutions are thus avoided. Song and Huang explored this idea further with a bio-inspired framework for PSO, genetic algorithm (GA), and bat algorithm (BA) [4]. The framework takes advantage of bitset database representation and a promising encoding vector check (PEV-check) to verify valid item combinations. It also replaces the velocity mechanism with a more efficient update procedure that uses bit difference. Once the population is updated, gBest is selected by applying roulette wheel selection on the set of discovered HUIs instead of selecting the fittest particle. As a result, gBest changes more frequently, the diversity in the population increases, and the

algorithm is better able to avoid local optima. Many similar HUIM EC models have been developed, such as [20–22, 25], but there are few alternatives for CHUIM. In 2021, Pramanik and Goswami proposed CHUI-AC [23], an ant colony optimization (ACO) method for discovering CHUIs. The model maps the search space to a directed graph that is traversed using local and global pheromone rules, inspired by the real-world movements of ants. The algorithm showed promising runtime results compared to CHUI-Miner but missed a large portion of the overall patterns. This is a common problem with EC algorithms, as they focus on building new patterns close to the currently best solutions in the search space. However, certain solutions may closely resemble low-fitness patterns and are thus missed.

3 Preliminary and problem statement

Let the set $I = \{i_1, i_2, \ldots, i_m\}$ contain m distinct items, where i_k is a unique item such that $1 \leq k \leq m$. A transactional database $D = \{T_1, T_2, \ldots, T_n\}$ is a set of n transactions, where each transaction $T_q \subseteq I$ and q is a unique transaction identifier (TID) such that $1 \leq q \leq n$. Moreover, each item $i_k \subseteq D$ is associated with a profit value, denoted $p(i_k, D)$, and a purchase quantity for each transaction, denoted $q(i_k, T_q)$. The set $X \subseteq I$ is called an itemset and is included in transaction T_q if $X \subseteq T_q$.

The database shown in Table 1 is used as a running example in this paper. It contains 5 transactions and 7 distinct items named from A to G, with the corresponding purchase quantities inside the parentheses. The associated profit value of each item is shown in Table 2.

Table 1 A quantitative transactional database

TID	Transaction (item:quantity)	Transaction utility (tu)
T_1	(B:5), (C:1), (D:2), (E:1), (F:1)	25
T_2	(B:4), (C:3), (D:2), (E:1)	20
T_3	(A:1), (C:1), (E:1)	8
T_4	(A:2), (C:6), (E:2), (G:5)	25
T_5	(B:2), (C:2), (E:1), (G:2)	11

Table 2 Profit table

Item	A	B	C	D	E	F	G
Unit profit	4	2	1	3	3	5	1

Definition 1 The local utility of an item i_k in a transaction T_q is denoted $u(i_k, T_q)$ and is defined as:

$$u(i_k, T_q) = q(i_k, T_q) \times p(i_k, D) \tag{1}$$

Example 1 The local utility of item B in transaction T_1 is calculated as $5 \times 2 = 10$.

Definition 2 The local utility of an itemset X in a transaction T_q is denoted $u(X, T_q)$ and is defined as:

$$u(X, T_q) = \sum_{i_k \in X, X \in T_q} u(i_k, T_q) \tag{2}$$

Example 2 The local utility of itemset (DF) in transaction T_1 is calculated as $2 \times 3 + 1 \times 5 = 11$.

Definition 3 The utility of an itemset X in a database D is denoted $u(X)$ and is defined as:

$$u(X) = \sum_{X \in T_q, T_q \in D} u(X, T_q) \tag{3}$$

Example 3 The utility of itemset (EG) is calculated as $2 \times 3 + 5 \times 1 + 1 \times 3 + 2 \times 1 = 16$.

Definition 4 The TID-set of an itemset X in a database D is denoted $TID(X)$ and is defined as:

$$TID(X) = \{q \mid q \geq 1, q \leq n, X \in T_q, T_q \in D\} \tag{4}$$

Example 4 The TID-set of itemset (BC) is {1,2,5}, since (BC) occurs in T_1, T_2 and T_5.

Definition 5 The support count of an itemset X is denoted $sup(X)$ and is defined as:

$$sup(X) = |TID(X)| \tag{5}$$

Example 5 The support of itemset (BC) is calculated as $|\{1,2,5\}| = 3$.

Definition 6 The transaction utility of a transaction T_q is denoted $tu(T_q)$ and is defined as:

$$tu(T_q) = \sum_{i_k \in T_q} u(i_k, T_q) \tag{6}$$

Example 6 The transaction utility of T_3 is calculated as: $1 \times 4 + 1 \times 1 + 1 \times 3 = 8$

Definition 7 The total utility of a database D is denoted $TU(D)$ and is defined as:

$$TU(D) = \sum_{T_q \in D} tu(T_q) \tag{7}$$

Example 7 The total utility of the database in Table 1 is calculated as 25 + 20 + 8 + 25 + 11 = 89.

Definition 8 The transaction weighted utility (TWU) of an itemset X is denoted $TWU(X)$ and is defined as:

$$TWU(X) = \sum_{q \in TID(X)} tu(T_q) \tag{8}$$

Example 8 The TWU of itemset (A) is calculated as 8 + 25 = 33.

Definition 9 An itemset X is a high transaction weighted utilization itemset (HTWUI) if $TWU(X) \geq \delta$; otherwise, X is a low transaction weighted utilization itemset (LTWUI). Note that δ is the minimum utility threshold, which is set by the user. Furthermore, a HTWUI/LTWUI with k items is denoted k-HTWUI/k-LTWUI.

Example 9 If the minimum utility threshold is set to 35, then itemset (A) is a 1-LTWUI because $TWU(A) = 33$, while itemset (B) is a 1-HTWUI as $TWU(B) = 56$.

Definition 10 An itemset X is a high-utility itemset (HUI) if $u(X) \geq \delta$, where δ is the minimum utility threshold.

Example 10 If the minimum utility threshold is 35, then itemset (BDE) is a HUI as $u(BDE) = 36$.

Definition 11 A high-utility itemset X is a closed high-utility itemset (CHUI) if there exists no itemset Y, where $X \subset Y$ and $sup(X) = sup(Y)$.

Example 11 If the minimum utility threshold is 35, the HUI (BDE) is not a CHUI as it is a subset of itemset $(BCDE)$ and $sup(BDE) = 2$, which is equal to $sup(BCDE) = 2$. However, itemset $(BCDE)$ is a CHUI.

Problem statement: Based on the above definitions and given a minimum utility threshold, the problem of closed high-utility itemset mining is to discover all closed high-utility itemsets within a quantitative transactional database.

4 Proposed EC based framework for CHUIM

The proposed algorithm is a heuristic EC-based model for discovering CHUIs. It applies a pruning strategy to reduce the search space before a population of particles is generated based on the remaining candidates. The population

is then iteratively updated and evaluated to reveal the CHUIs until a maximum number of iterations is reached. In this section, we first describe the adopted strategy of bitset database representation and promising coding vector [4] before the main developed strategies (1) Extended TWU pruning, (2) Fitness estimates, and (3) Particle updates with ExploredSet are introduced. Finally, all parts are put together, and the pseudo-code of the complete model is explained.

4.1 Promising encoding vector

Let the original database D consist of n transactions and m items. The bitset representation of the database $B(D)$ is then a $n \times m$ matrix of Boolean values. The entry in $B(D)$ corresponding to transaction T_q $(1 \leq q \leq n)$ and item i_k $(1 \leq k \leq m)$ is denoted as (q, k) and is the q-th row and k-th column of $B(D)$. Each entry is then defined as:

$$B(q,k) = \begin{cases} 1, & \text{if } i_k \in T_q \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

That is, the entry (q, k) of $B(D)$ is set to 1 if transaction T_q contains item i_k, otherwise it is set to 0.

The bitset cover of an item i_k corresponds to the k-th column vector of $B(D)$, and is denoted as $Bit(i_k)$. This extends to itemsets in the following way: The bitset cover of itemset X is computed as the bitwise-AND$_{i \in X}(Bit(i))$, which is the intersection between all bitset covers of the items in X. Similarly, the bitset cover of two itemsets X and Y is calculated as $Bit(X) \cap Bit(Y)$. For example, the bitset cover of items B and D in Table 1 are {1,1,0,0,1} and {1,1,0,0,0}, respectively. The bitset cover of itemset (BD) is then calculated as {1,1,0,0,1} $\cap$ {1,1,0,0,0} = {1,1,0,0,0}. Thus, the itemset (BD) occurs in transactions T_1 and T_2 as positions 1 and 2 of $Bit(BD)$ are set to 1.

In the designed model, each generated itemset is represented by a particle. The particle maintains a bit vector that keeps track of the items present in the particle, called encoding vector. If the j-th position of an encoding vector is set to 1, then item j is included in the particle itemset. Otherwise, item j is not included. The size of the encoding vector corresponds to the number of 1-HTWUI in the database. For example, if we assume that each item in Table 1 is a 1-HTWUI, then the encoding vector size is 7 and the itemset (BDE) is represented as {0,1,0,1,1,0,0}. Naturally, evolutionary algorithms with candidate generation can produce itemsets that do not appear in any transaction. To avoid these itemsets and improve the model's effectiveness, the concept of promising encoding vector is applied.

Definition 12 Let P represent the encoding vector of an itemset X. If the bitset cover $Bit(X)$ only contains 0s, then the encoding vector P is called an unpromising encoding vector $(UPEV)$; otherwise P is a promising encoding vector (PEV).

Example 12 The itemset (AB) in Table 1 is a $UPEV$ as $Bit(AB)$= {0,0,0,0,0}, while the itemset (AC) is a PEV as $Bit(AC)$= {0,0,1,1,0}

The procedure to determine whether an encoding vector is a PEV is called PEV-check. The PEV-check is applied to ensure each candidate is present in at least one transaction of the input database. When a UPEV is detected, the procedure removes items from the encoding vector until it is a PEV. This way, the model avoids irrelevant itemsets, which improves the ability to discover the correct solutions. Further detail on the PEV-check is described by Song and Huang [4].

4.2 Extended TWU pruning (ETP)

The traditional TWU model pruning is based on the transaction-weighted downward closure property [5]. This property states that if an item's TWU (Def. 8) is less than the minimum utility threshold, then no superset of this item can be a HUI. Any such item is considered unpromising (1-LTWUI) as it cannot be part of a solution, and it is thus removed from the database to reduce the algorithm's search space. The TWU model performs two database scans to complete this process. The first scan calculates the TWU of each item, while the second scan removes the 1-LTWUIs from the database. The remaining items after the second scan are the 1-HTWUIs.

We propose an extension to this process to further reduce the number of possible candidates, called Extended TWU Pruning (ETP). ETP uses the same two database scans as the TWU model to identify and remove 1-LTWUIs. However, if any items are pruned during the second scan, ETP restarts the entire process and performs both scans again. The rationale for this strategy is that certain transaction utilities (Def. 6) decrease by removing items from the database. As the TWU values are a sum of transaction utilities, additional items may be classified as 1-LTWUI if their TWUs are redetermined after the second scan. Altogether, ETP is a recursive transformation of the traditional TWU model that can significantly impact the effectiveness of pruning. Section 5 illustrates an example of the procedure.

4.3 Fitness estimates

Definition 13 The fitness of a particle p_i is defined as:

$$fitness(p_i) = u(X), \tag{10}$$

where X is the itemset in the encoding vector of p_i.

At each iteration of the algorithm, the fitness function is applied to newly generated particles to determine the quality of the solutions. Calculating the fitness requires a database scan and can be computationally expensive, especially if the database is large. We propose two types of utility estimates to alleviate this, described below.

4.3.1 Maximum estimates

Definition 14 The maximum utility of an item i in a database D is denoted $mu(i)$ and is defined as:

$$mu(i) = max\{u(i, T_q)\}_{\forall T_q \in D} \tag{11}$$

Example 13 The maximum utility of item (B) in Table 1 is calculated as $max\{10, 8, 4\} = 10$

Definition 15 The maximum estimate on the utility of an itemset X is denoted $maxEst(X)$ and is defined as:

$$maxEst(X) = sup(X) \times \sum_{i_k \in X} mu(i_k) \tag{12}$$

Example 14 The maximum estimate of itemset (B) in Table 1 is calculated as $3 \times 10 = 30$.

As the maximum utility of an item cannot be less than any of its local utilities, it follows that the maximum estimate is a definite upper bound on the utility of an itemset ($maxEst(X) \geq u(X)$). The developed model utilizes maximum estimates to determine whether a newly generated particle should be evaluated. If the estimate is less than the minimum utility threshold and the fitness of the corresponding $pBest$, then the particle can be safely ignored as it cannot improve $pBest/gBest$ or be a CHUI. For instance, the maximum estimate of itemset (B) in Table 1 is 30. If the minimum utility threshold and fitness of $pBest$ are at least 31, then the algorithm bypasses the evaluation of particles containing itemset (B). The purpose of this strategy is to assess the particle quality before performing the fitness calculation. This way, the algorithm can save time by omitting costly database scans for unpromising candidates, as it is unnecessary to identify their actual utility. Note that the maximum utilities are stored to avoid calculating them repeatedly for each estimate. In addition, the model retrieves the support count during the particle's PEV-check at no additional computation cost. As a result, each estimate is calculated in linear time on the size of the itemset.

4.3.2 Average estimates

The maximum estimates can often greatly exaggerate the utility of itemsets. To further reduce the number of necessary fitness evaluations, we employ a strategy called average estimates.

Definition 16 The average utility of an item i in a database D is denoted $au(i)$ and is defined as:

$$au(i) = \left\lceil \frac{\sum_{T_q \in D} u(i, T_q)}{sup(i)} \right\rceil \tag{13}$$

Example 15 The average utility of item B in Table 1 is calculated as $\lceil \frac{10+8+4}{3} \rceil = 8$.

Definition 17 The average estimate on the utility of an itemset X is denoted $avgEst(X)$ and is defined as:

$$avgEst(X) = sup(X) \times \sum_{i_k \in X} au(i_k) + \sigma, \tag{14}$$

where the deviation σ is calculated as:

$$\sigma = \left\lceil \frac{\sum_{i_k \in 1\text{-}HTWUI} mu(i_k) - au(i_k)}{|1\text{-}HTWUI|} \right\rceil \tag{15}$$

Example 16 Assuming all items in Table 1 are 1-HTWUI, the deviation is calculated as $\left\lceil \frac{(8-6)+(10-8)+(6-3)+(6-6)+(6-4)+(5-5)+(5-4)}{7} \right\rceil = 2$. Thus, the average estimate of itemset (B) is calculated as $3 \times (8 + 2) = 30$.

Contrary to maximum estimates, average estimates cannot provide a definite upper bound on the utility of an itemset. A solution can thus be missed if the estimate is completely trusted and the fitness evaluation is mistakenly skipped. In order to account for this uncertainty, the mean deviation between all maximum- and average utilities is applied to increase the overall estimates. This may seem counter-intuitive as the difference between maximum- and average estimates diminish. However, the deviation can be adjusted during runtime. Each time the algorithm calculates the fitness of a particle, it compares the estimate to the actual fitness. If the estimate is less than the fitness, it is considered an underestimate; otherwise, it is an overestimate. When the sample size is sufficiently large, the algorithm measures the ratio of underestimates to overestimates and updates the deviation according to the following formula:

$$\sigma = \begin{cases} \sigma - 1, & \text{if } \sigma > 0 \text{ and } \frac{u}{o} < 0.01 \\ \sigma, & \text{otherwise,} \end{cases} \tag{16}$$

where the number of over- and underestimates are denoted as o and u, respectively.

Example 17 Assuming $u = 0$ and $o = 1{,}000$. The deviation of Table 1 is updated as 2 - 1 = 1, and the average estimate of itemset (B) becomes $3 \times (8 + 1) = 27$. For comparison, the actual utility of itemset (B) is 24.

This way, average estimates can adapt to the dataset and become more effective than maximum estimates as the algorithm progresses. Although this works well, some CHUIs can still be missed by incorrectly omitting their corresponding particles' fitness calculation. In order to promote model accuracy rather than speed, average estimates are only used when the measured deviation is less than 0.01% of the given minimum utility threshold. The type of

estimate is determined before the mining process starts and does not change during runtime.

4.4 Particle update with ExploredSet

At each iteration of the algorithm, the population is updated with the influence of the previous best particles. Two types of historic particles are referenced—the personal fittest offspring of each particle ($pBest$) and the all-time fittest particle of the entire population ($gBest$). The update procedure starts by evolving the particles towards $pBest$ and $gBest$ by using the concept of bit difference [4], defined below.

Definition 18 The bit difference of two particles p_i and p_j, denoted $BitDiff(p_i, p_j)$, is defined as the bitwise-XOR operation on the particle's encoding vectors.

Example 18 Let p_i = {1,1,1,0,0} and p_j = {1,0,1,0,1}. The bit difference of p_i and p_j is then {0,1,0,0,1}.

In other words, bit difference creates a bit vector of non-identical bits between two encoding vectors. If the vector's k-th position is set to 1, then item k is present in one of the particles while absent in the other. During the update procedure, bit difference is used to compare a particle with $pBest$ and $gBest$, and the non-identical bits determine the items that can change in the particle. Consequently, the similarity between the particle and $pBest/gBest$ increases.

However, if $pBest$ and $gBest$ remain unchanged over many iterations, the algorithm tends to create many identical candidates and converge to local optima. We alleviate this by performing a random modification to each particle after the update towards $pBest$ and $gBest$ is complete. The strategy utilizes a hash set containing all candidates the model has generated during runtime, called ExploredSet. A particle p is explored if $p \in$ ExploredSet; otherwise p is unexplored. If an explored particle is created after updating towards $pBest$ and $gBest$, the algorithm either includes or excludes a random item in the particle. This way, the population diversity increases while we avoid modifying unexplored particles to redundant candidates previously evaluated. As a result, more unique candidates are generated, and the algorithm is better able to evade local optima. Formally, the total number of bits b_i to change during the update of a particle p_i is expressed as:

$$b_i = b_{i1} + b_{i2} + b_{i3}, \tag{17}$$

where b_{i1}, b_{i2}, and b_{i3} are calculated as:

$$b_{i1} = \left\lfloor r_1 \times \sum BitDiff(p_i, pBest_i) \right\rfloor \tag{18}$$

$$b_{i2} = \left\lfloor r_2 \times \sum BitDiff(p_i, gBest) \right\rfloor \tag{19}$$

$$b_{i3} = \begin{cases} 1, & \text{if } p_i \in ExploredSet \\ 0, & \text{otherwise} \end{cases} \tag{20}$$

where r_1 and r_2 are random numbers between [0,1]. Note that during update, b_{i3} is calculated after $b_{i1} + b_{i2}$ changes are made to the particle. An example of the update procedure is given in Section 5.

4.5 Proposed CHUI-PSO

CHUI-PSO takes as input a transactional database, a minimum utility threshold, a population size, and a number of iterations. In Algorithm 1, the model starts by calling the initialization procedure of Algorithm 2, where the database is pruned, and the initial population of particles is generated and evaluated (line 1). As this is the first population, *pBest* and *ExploredSet* are copies of the current population, while the fittest particle is selected as *gBest* (lines 2 and 3). To determine the appropriate estimate type for the dataset, the deviation σ is calculated (line 4). Average estimates are only used if σ is less than 0.01% of the given minimum utility threshold. Subsequently, the main loop of the model is initiated in which the population is updated and evaluated *iter* times (lines 5-25). The process begins by updating a particle before the PEV-check is applied to ensure it appears in one or more transactions (lines 7 and 8). The new particle is then compared to the set of explored particles to avoid further evaluation of redundant candidates (line 9). If the particle is unexplored, its fitness is estimated using either average- or maximum estimates, depending on the size of the initial deviation (lines 10 and 11). The exact fitness is only determined if the estimate is greater than the minimum utility threshold and fitness of *pBest* (lines 12 and 13). This way, costly database scans are avoided for particles that cannot improve *pBest*/*gBest* or be a CHUI. If the fitness is calculated, Algorithm 3 verifies the closure of the particle, and any CHUI is appended to the solution set (lines 14-16). *pBest* and *gBest* are then updated accordingly (lines 17 and 18) before the particle is marked as explored (line 20). After the entire population is updated and evaluated, *gBest* is re-selected to one of the discovered CHUIs, using either roulette wheel selection as described in [4] or random selection (line 23). The algorithm introduces random selection if no new CHUIs have been discovered in 50 iterations to induce more population diversity during convergence. Before the next iteration begins, the deviation is updated according to the number of over- and under-estimates (line 24). This step is only relevant if average estimates are used and will not change the estimate type for the subsequent iterations. Finally, when all iterations are complete, the algorithm returns the set of discovered CHUIs and terminates (line 26). An overview of the complete model is shown in the flowchart in Fig. 1.

Algorithm 2 describes the population initialization procedure in which *pop_size* particles are constructed. First, the original database is scanned with

Algorithm 1 Proposed CHUI-PSO Algorithm

Input: D: a transactional database, δ: the minimum utility threshold, pop_size: the population size, $iter$: the number of iterations.

Output: $CHUIS$: a set of closed high-utility itemsets.

1: $Pop, CHUIS \leftarrow$ init(D, pop_size, δ);
2: $pBest$, $ExploredSet \leftarrow Pop$;
3: $gBest \leftarrow$ fittest particle in Pop;
4: calculate deviation σ using Eq. (15);
5: **for** $i = 1$ to $iter$ **do**
6: **for** $k = 1$ to pop_size **do**
7: change b_i items in Pop_k using Eq. (17);
8: PEV-check Pop_k;
9: **if** $Pop_k \notin ExploredSet$ **then**
10: $X \leftarrow$ the itemset in Pop_k;
11: $est \leftarrow$ estimate the utility of X using Eq. (12) **or** (14);
12: **if** $est \geq \delta$ **and** $est \geq fitness(pBest_k)$ **then**
13: $fit \leftarrow$ calculate fitness of Pop_k using Eq. (10);
14: **if** $fit \geq \delta$ **and** closed(Pop_k) **then**
15: $CHUIS \leftarrow CHUIS \cup Pop_k$
16: **end if**
17: $pBest_k \leftarrow$ fittest of $pBest_k$ and Pop_k;
18: $gBest \leftarrow$ fittest of $gBest$ and Pop_k;
19: **end if**
20: $ExploredSet \leftarrow ExploredSet \cup Pop_k$;
21: **end if**
22: **end for**
23: update $gBest$ with roulette wheel selection **or** random selection;
24: update deviation σ using Eq. (16);
25: **end for**
26: return $CHUIS$;

ETP to reduce the search space (line 1). The remaining items in D are the 1-HTWUI and determine the encoding vector size of each particle. The population and solution set are then initialized to empty (line 2) before the particles are iteratively created in the main loop of the procedure (lines 3-12). At each iteration, a random number r between 1 and the longest transaction length in D is generated (line 4). The particle is then created with r bits set to 1 in its encoding vector, which means it contains an r-itemset. The items are selected using roulette wheel selection, where a large TWU value increases the probability of selection. The PEV-check is then applied to avoid irrelevant itemsets, and the fitness of the particle is calculated (lines 6 and 7). The fitness and algorithm 3 are then used to determine any CHUIs before the particle is inserted into the population (lines 8-11). After each particle is created, the population and potential CHUIs are returned, and the procedure terminates (line 13).

Algorithm 2 Population initialization, *init*()

Input: D: a transactional database, pop_size: the population size, δ: the minimum utility threshold.

Output: Pop: the first population, $CHUIS$: closed high-utility itemsets.

1: $D \leftarrow$ prune D with ETP
2: $Pop, CHUIS \leftarrow \emptyset$;
3: **for** $i = 1$ to pop_size **do**
4: generate random number r;
5: create p_i with r bits set to 1;
6: PEV-check p_i;
7: $fit \leftarrow$ calculate fitness of p_i using Eq. (10);
8: **if** $fit \geq \delta$ **and** closed(p_i) **then**
9: $CHUIS \leftarrow CHUIS \cup p_i$;
10: **end if**
11: $Pop_i \leftarrow p_i$;
12: **end for**
13: Return $Pop, CHUIS$;

The closure check in Algorithm 3 describes the process which determines the closure of a particle. The loop (lines 2-6) checks whether any 1-HTWUI that does not appear in itemset X of the particle can be appended to X and attain the same support value. This is done by comparing the bitset cover of X and $X \cup i_k$ (line 3). If the two vectors are equal, then the superset occurs in the same transactions, and the particle is not closed by definition 11 (line 4). The closure is verified if no such superset exists, and the procedure returns True (line 7).

Algorithm 3 Closure check, closed()

Input: P: the particle

Output: `True`: P is closed, `False`: P is not closed

1: $X \leftarrow$ the itemset in P;
2: **for each** $i_k \in$ 1-HTWUI , $i_k \notin X$ **do**
3: **if** $Bit(X \cup i_k) == Bit(X)$ **then**
4: Return `False`;
5: **end if**
6: **end for**
7: Return `True`;

5 An illustrated example

In this section, the database in Table 1 is used as an example to illustrate the flow of the developed algorithm. The minimum utility threshold and population size are set to 35 and 5, respectively. Initially, the database is scanned with

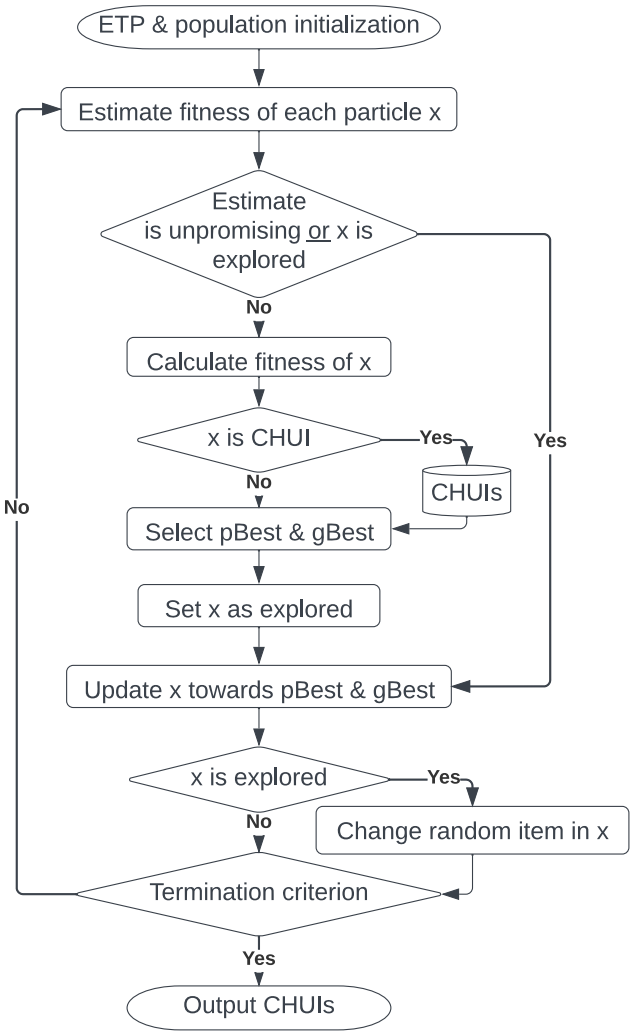

Fig. 1 Flowchart of CHUI-PSO

ETP to remove each 1-LTWUI. The process goes as follows: First, the TWU of each item is calculated by searching the database once {A:33, B:56, C:89, D:45, E:89, F:25, G:36}. Thus, items A and F are 1-LTWUI and are removed from the transactions in which they occur. As a result, the transaction utilities change and are updated to {T_1:20, T_2:20, T_3:4, T_4:17, T_5:11}, and the TWU of the remaining items are recalculated to {B:51, C:72, D:40, E:72, G:28}. This time, item G is also classified as 1-LTWUI and is deleted from the database. The transaction utilities are then updated {T_1:20, T_2:20, T_3:4, T_4:12, T_5:9}, and the TWUs are recalculated {B:49, C:65, D:40, E:65}. None of these items are 1-LTWUIs, and the final set of 1-HTWUIs are thus {B, C, D, E}.

The maximum utilities of the 1-HTWUI are {B:10, C:6, D:6, E:6}, the average utilities are {B:8, C:3, D:6, E:4}, and the deviation is 2. Consequently, maximum estimates are used throughout this example. The initial particles of the population are then initialized, with the size of the encoding vector set to the number of discovered 1-HTWUIs, i.e., 4. Assume the algorithm creates the particles shown in Table 3.

Table 3 The initial particles in the population

Particle	B	C	D	E
P_1	0	0	0	1
P_2	0	1	0	1
P_3	1	0	0	1
P_4	1	0	1	1
P_5	0	1	1	1

As this is the first population, the fitness of each particle is calculated $\{P_1$:18, P_2:31, P_3:31, P_4:36, P_5:16$\}$, and $pBest$ is a copy of the population. Based on the fitness values, particle P_4 is the fittest and is selected as $gBest$. The itemset (BDE) of P_4 is also a HUI. However, it is not a CHUI, as the support of (BDE) is equal to the support of the superset $(BCDE)$. Each particle is then appended to the set of explored solutions before the update procedure begins. To illustrate, let us examine the update of particle P_1 with the random numbers r_1 and r_2 generated to 0.9 and 0.5, respectively. As $BitDiff(P_1, pBest_1) = \{0, 0, 0, 0\}$, b_{11} is $\lfloor 0.9 \times 0 \rfloor = 0$. Therefore, no bits in P_1 are changed. $BitDiff(P_1, gBest) = \{1, 0, 1, 0\}$ and $b_{12} = \lfloor 0.5 \times 2 \rfloor = 1$. This means that one of the non-identical bits between P_1 and $gBest$ must be flipped in P_1, either the bit representing item B or D. Assuming item B is randomly selected, P_1 becomes $\{1, 0, 0, 1\}$. This encoding vector is already explored as it was evaluated by P_3 in the last population. As a result, b_{13} is calculated to 1 and an additional random bit in P_1 is flipped. However, all bits in P_1 are now eligible to change. Assuming the bit representing item C is selected, P_1 becomes $\{1, 1, 0, 1\}$. The update is then complete as this encoding vector is a PEV. Similarly, the other particles are updated: P_2:$\{0, 1, 1, 1\}$, P_3:$\{1, 0, 1, 1\}$, P_4:$\{0, 0, 1, 1\}$ and P_5:$\{0, 0, 1, 0\}$.

Following the update procedure, the maximum estimate of each particle is calculated $\{P_1$:66, P_2:36, P_3:44, P_4:24, P_5:12$\}$. As a result, the fitness evaluation of P_4 and P_5 are neglected as their estimates are below the minimum utility threshold and the fitness of $pBest_4/pBest_5$. In addition, P_3 is ignored since it was explored in the last population. The fitness of P_1 and P_2 is calculated to 37 and 22, respectively. Consequently, $pBest_1$ and $gBest$ are updated to P_1, while $pBest_2$ remains unchanged. The itemset (BCE) of P_1 is also a CHUI and is appended to the solution set. Finally, each particle in the current population is marked as explored, and the next iteration begins. When the algorithm terminates, the discovered CHUIs with their utilities are $\{(BCE)$:37, $(BCDE)$:40$\}$.

6 Experimental evaluation

This section compares the performance of the proposed CHUI-PSO against three existing models: the exact algorithms CLS-Miner and EFIM-Closed, and the heuristic EC-based CHUI-AC. In addition, to evaluate the impact of the ETP strategy, a version of CHUI-PSO using the traditional TWU model is

tested, called CHUI-PSO(np). The authors provided the source code of CLS-Miner and CHUI-AC, while we downloaded EFIM-Closed from the SPMF data mining library [24]. The code for CHUI-PSO is available at GitHub[1]. All the compared algorithms are written in Java and were executed using JDK 17.0.1, with the Java heap size set to 2 GB. The experiments were performed on a 64-bit Windows 10 computer with a Ryzen 5 5600x CPU, a Radeon RX 5700 GPU, and 16 GB 3200 MHz CL 16 RAM. We evaluated the performance of each algorithm using six datasets downloaded from SPMF. The datasets can be classified as dense or sparse based on the number of items and average transaction length. Dense databases generally contain fewer items but longer transactions compared to sparse. We selected a mixture of both types to illustrate the performance in various scenarios. Table 4 shows the datasets and their characteristics.

Table 4 Database characteristics

Dataset	# items	# transactions	Avg. trans. length	Type
Accidents	468	340,183	33.8	Dense
Chainstore	46,086	1,112,949	7.23	Sparse
Chess	75	3,196	37	Dense
Connect	129	67,557	43	Dense
Retail	16,470	88,162	10.3	Sparse
Kosarak	41,270	990,002	8.1	Sparse

In the experiments, the number of iterations and population size is set to 10,000 and 20 for all the heuristic algorithms. CHUI-AC has defined a termination criterion that stops the execution prematurely if it discovers no CHUIs during an iteration. This logic is disabled for a clearer comparison. All other algorithm parameters are set as suggested by the authors. Note that in each figure, the minimum utility threshold is expressed as a percentage of the total utility in the database (Def. 7).

6.1 Runtime

First, we evaluate the runtimes of the algorithms on five different minimum utility thresholds on each dataset. Fig. 2 shows the results.

The experiments in Fig. 2 demonstrate that the proposed CHUI-PSO is significantly faster than the heuristic CHUI-AC on every dataset tested. On Accidents and Kosarak, CHUI-AC finished in roughly 3 hours on all minimum utility thresholds, while our model used about 10 seconds. The smallest difference was observed in Chess, where CHUI-AC was at best 73 times slower than CHUI-PSO. The proposed method also took less time than CLS-Miner and EFIM-Closed on Accidents, Kosarak, and Retail, while it was the fastest for certain minimum utility thresholds on Connect and Chainstore. Chess is the only dataset where our algorithm could not achieve the best runtime at any minimum utility threshold; however, the difference to EFIM-Closed was

[1] https://github.com/Simencar/CHUI-PSO

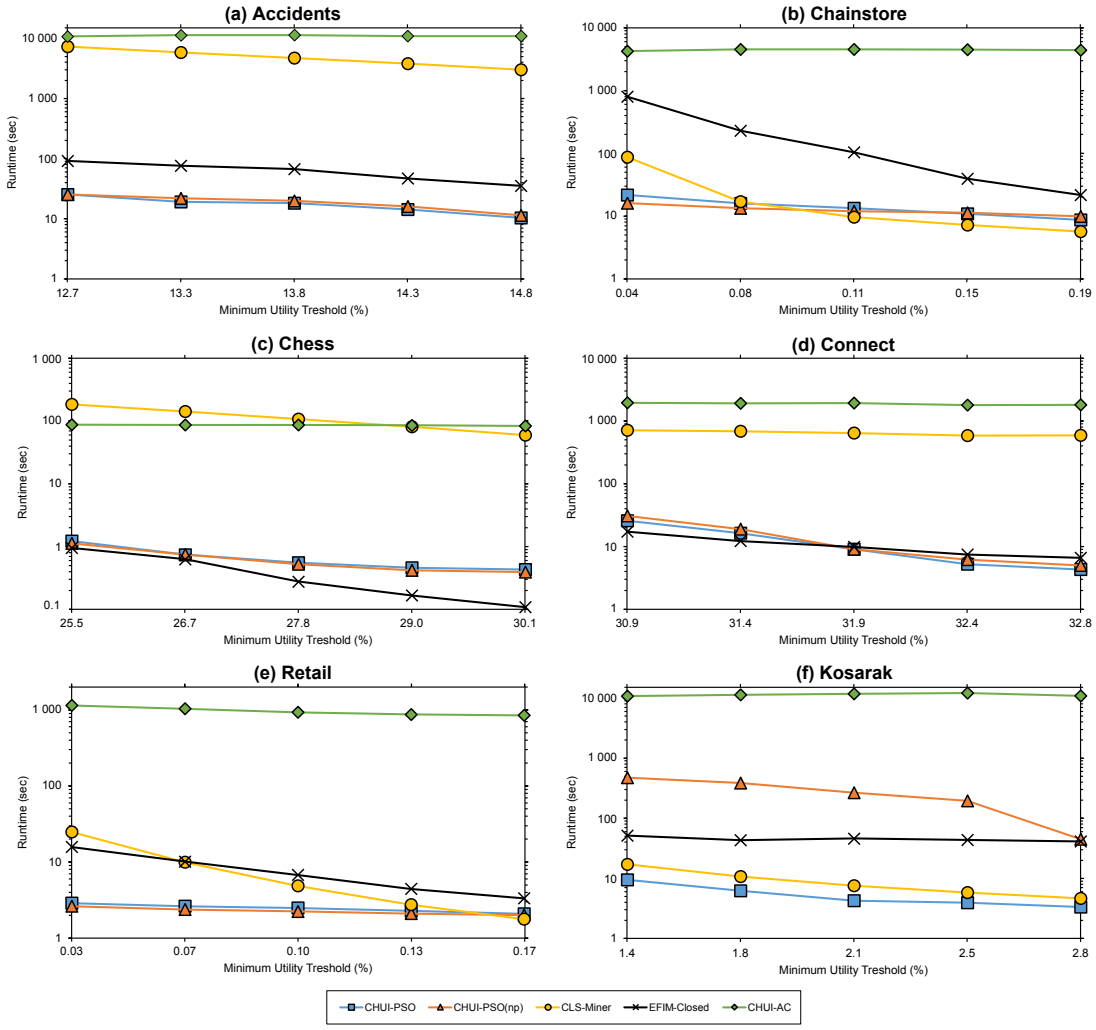

Fig. 2 Execution times of the compared algorithms.

at most 0.3 seconds. This dataset is particularly dense, which naturally leads to many transaction similarities, suiting EFIM's transaction merging strategy. It should also be noted that our model always completes the specified number of iterations, even if it discovers all available patterns before the termination criterion. This primarily affects the runtime on the largest minimum utility threshold values, as the model typically uncovers the CHUIs in a few iterations. Nonetheless, CHUI-PSO provided comparable execution times to EFIM-Closed in Chess and Connect, while the other datasets showed a clear performance advantage for our model. Compared to CLS-Miner, the proposed algorithm demonstrated much better results in the dense datasets and was at most 292 times faster. CHUI-PSO was also generally faster in sparse scenarios, but a slight advantage for CLS-miner can be seen on the higher minimum utility thresholds on Chainstore.

Overall, the proposed method was the fastest of the compared algorithms. CHUI-PSO used a total of 4 minutes and 16 seconds to complete all the tests shown in Fig. 2, while EFIM-Closed, CLS-Miner, and CHUI-AC took 31 minutes, 8 hours, and 41 hours, respectively. The efficiency of CLS-Miner and

EFIM-Closed strongly depended on the database type, while our model performed consistently well in dense and sparse scenarios. The main contributors to the speed of CHUI-PSO are the strategies for explored particles and fitness estimates, which allows the model to omit many costly and unnecessary fitness calculations. Moreover, the results show that ETP can drastically improve the algorithm's execution time. The runtime variability between CHUI-PSO and CHUI-PSO(np) was mostly minor, but the Kosarak results reveal a substantial difference caused by ETP pruning more items. As the number of candidates decreases, the size of the database becomes smaller. As a result, the algorithm can perform faster database scans to determine the fitness of particles. In addition, the probability of generating an explored particle grows quicker when there are fewer candidates, and CHUI-PSO tends to skip more particle evaluations than CHUI-PSO(np).

Nonetheless, CHUI-PSO and CHUI-PSO(np) generally performed similarly in terms of runtime. ETP executes more database scans than the TWU model to conduct the pruning, which requires additional time. Thus, the difference in speed is determined by the overall search space reduction compared to the pruning duration.

6.2 Number of discovered CHUIs

This section compares the number of discovered CHUIs between CHUI-PSO, PSO-CHUI(np), and CHUI-AC. EFIM-Closed and CLS-Miner are exact algorithms that find all available patterns. Therefore, we only included results for CLS-Miner as a reference to the maximum number of CHUIs. The results are shown in Fig. 3.

Fig. 3 displays that the proposed CHUI-PSO outperformed CHUI-AC in every experiment regarding the number of discovered CHUIs after 10,000 iterations. The developed algorithm found all available patterns on Accidents, Connect, and Kosarak, while it missed a small portion of the solutions on the other datasets. The worst result for CHUI-PSO is seen in Retail, where the accuracy was 94% for the lowest minimum utility threshold. In comparison, CHUI-AC could never discover all the solutions and performed the poorest on Connect, where it identified as low as 9% of the CHUIs. Altogether, the average accuracy of CHUI-PSO and CHUI-AC was 98.8% and 48.6%, respectively, demonstrating that our model can find significantly more itemsets while providing satisfactory results in both dense and sparse scenarios.

Several factors influence the accuracy of the PSO model. The changes to the particle update phase enable the algorithm to explore more of the search space while preventing wasted evaluations of redundant candidates. In addition, random selection for $gBest$ can induce population diversity in scenarios where $gBest$ rarely changes, reducing the chance of convergence to local optima. The results also suggest that ETP can increase the model's performance. CHUI-PSO found more CHUIs than CHUI-PSO(np) on Chainstore, Chess, Retail, and Kosarak. Most notably, CHUI-PSO(np) was only able to discover CHUIs

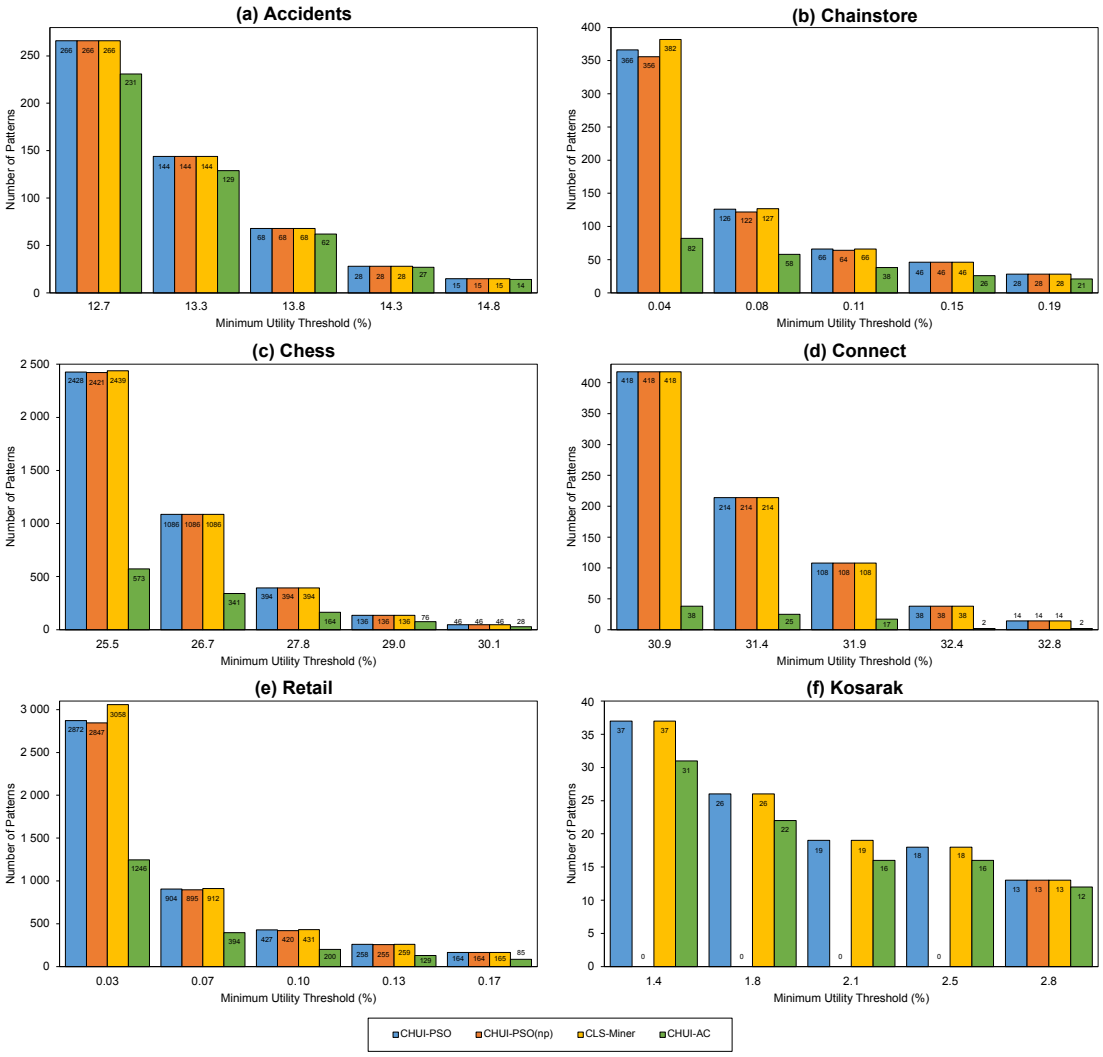

Fig. 3 Number of discovered CHUIs of the compared algorithms.

on Kosarak when the minimum utility threshold was set to 2.8%, while CHUI-PSO revealed all available patterns at all thresholds. This indicates that an additional reduction of the solution space can improve the ability to uncover CHUIs.

6.3 Number of candidates

This section compares the number of candidates between CHUI-PSO with the proposed ETP strategy and CHUI-PSO(np) with the traditional TWU model. The candidates are measured as the number of 1-HTWUIs discovered during the pruning phase. Therefore, few candidates are preferred since it reflects the algorithm's search space. We performed the tests on the same datasets and minimum utility thresholds as in the previous experiments. The results are shown in Fig. 4.

Fig. 4 reveals that the designed ETP strategy of CHUI-PSO consistently removed more unpromising candidates compared to the TWU model of CHUI-PSO(np). The most noteworthy differences are observed in Chainstore, Retail,

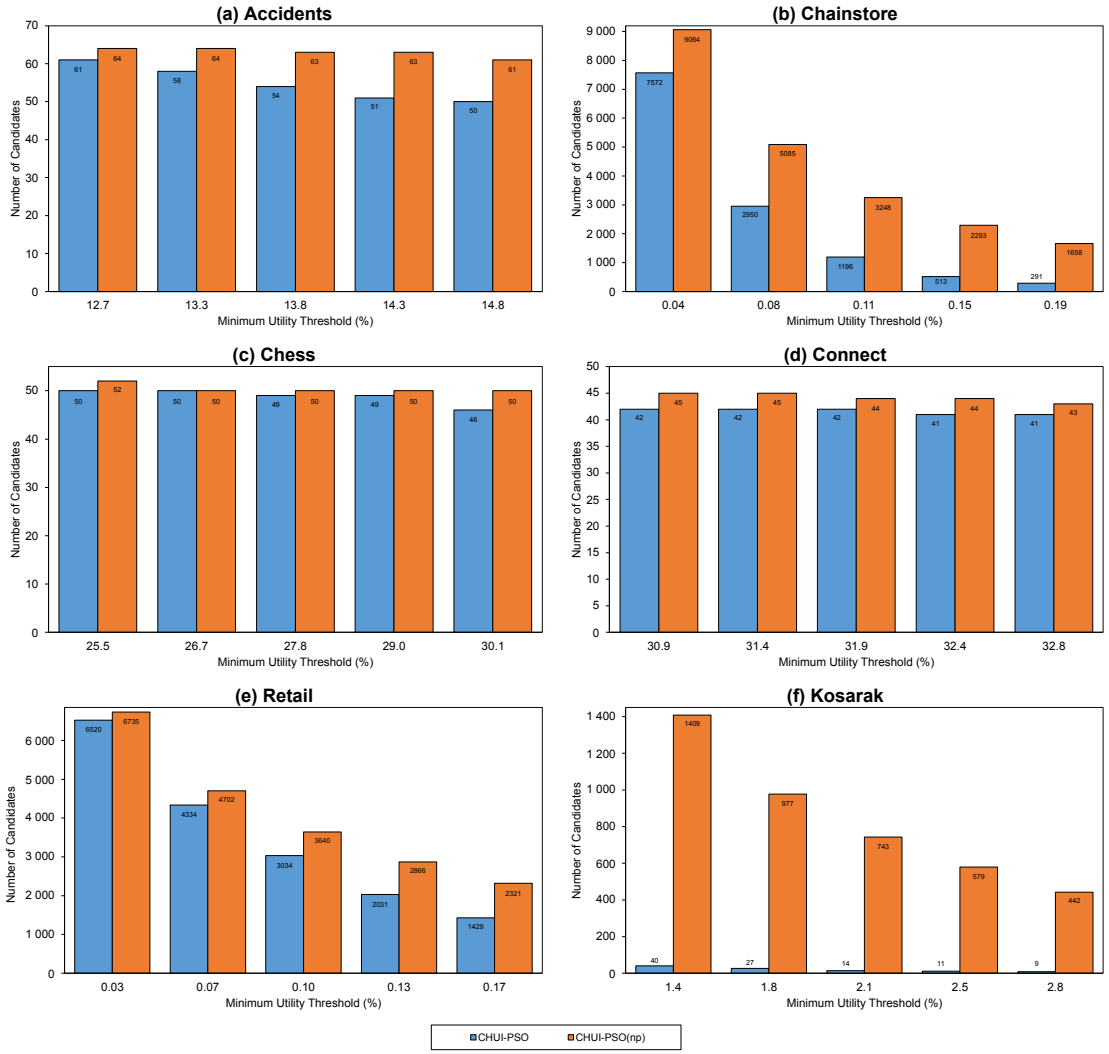

Fig. 4 Number of candidates in CHUI-PSO and CHUI-PSO(np)

and Kosarak, implying that ETP is generally the most effective when the database is sparse. However, the new pruning strategy also provided additional search space reductions for the dense datasets, especially Accidents. The results on Kosarak also reveal the reason for the large disparity in runtime and accuracy between CHUI-PSO and CHUI-PSO(np). When the minimum utility threshold is 1.4%, CHUI-PSO and CHUI-PSO(np) discovered 40 and 1,409 1-HTWUI, respectively. As a result, CHUI-PSO(np) considers an additional 1,369 items during candidate generation that can never be part of a CHUI, reducing the probability of creating the correct solutions. In addition, the items must be maintained in the database, increasing the time required for fitness calculations.

Although the developed strategy can prune more candidates than the traditional TWU model, CHUI-PSO(np) provides comparable accuracy to CHUI-PSO in most datasets, as shown in Fig. 2. This outcome is due to the PSO algorithm naturally learning to disregard unpromising items, which likely are the additional items pruned with ETP. Therefore, ETP may have a larger impact on models not based on evolutionary principles.

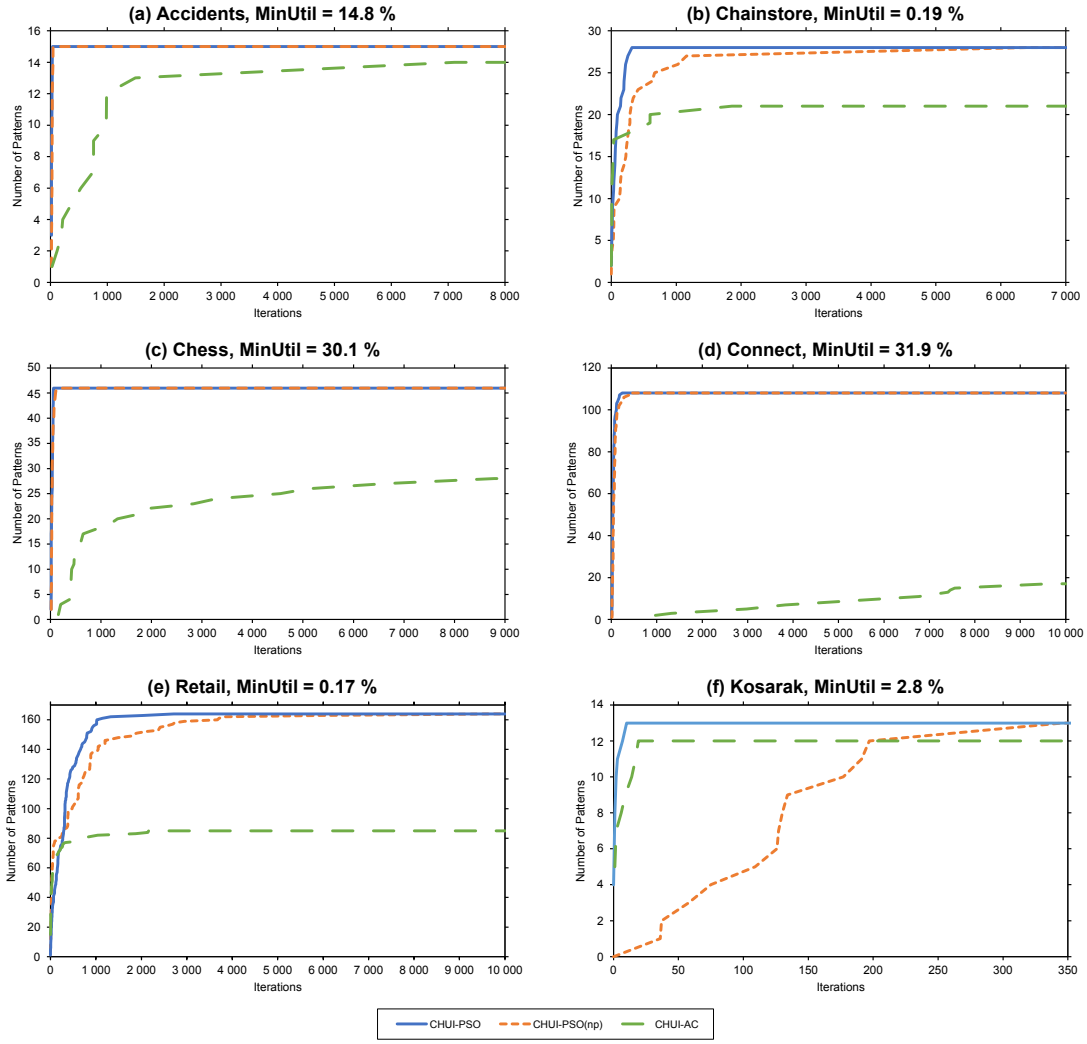

Fig. 5 Convergence rate of the compared EC based algorithms.

6.4 Convergence

Finally, we compare the convergence rate of CHUI-PSO, CHUI-PSO(np), and CHUI-AC with a fixed minimum utility threshold on each dataset. CHUI-Miner and EFIM-Closed are excluded as they are not iterative approaches. Note that all algorithms completed 10,000 iterations, but the scale of the x-axis (iterations) varies depending on the iteration number the last CHUI was discovered. Fig. 5 shows the results.

According to Fig. 5, CHUI-PSO achieved the fastest convergence on all six datasets. The results for CHUI-AC display that the algorithm often fell into local optima, causing it to miss many relevant solutions. In addition, it used more iterations than CHUI-PSO to reach the state of convergence. Overall, the largest differences between CHUI-PSO and CHUI-AC are observed on the dense datasets, where our model used at most 238 iterations to find all available CHUIs. In comparison, CHUI-AC discovered new patterns after 7000 iterations.

Comparing the PSO versions, CHUI-PSO and CHUI-PSO(np) performed almost identically on the dense datasets, but an advantage for CHUI-PSO

is seen on the sparse. As shown in Fig. 4, the designed ETP strategy is most effective in Chainstore, Retail, and Kosarak. Thus, CHUI-PSO avoids more unpromising candidates on these databases, increasing the probability of obtaining the solutions in fewer iterations. Altogether, the convergence rate of the proposed CHUI-PSO allowed the model to comfortably discover the CHUIs before reaching the termination criterion in these experiments. This suggests that the iterations can be reduced to improve the algorithm's runtime while maintaining accuracy.

7 Conclusion

This paper proposed CHUI-AC, a heuristic particle swarm optimization (PSO) model for closed high-utility itemset mining (CHUIM). To our knowledge, this is the first work on PSO in CHUIM. We also introduced several new strategies to address the performance limitations of similar heuristics in utility mining. In the proposed approach, unnecessary and costly fitness calculations are avoided by estimating itemset utilities and utilizing a hash set containing explored particles. In addition, the explored particles are employed during particle updates to increase population diversity while alleviating the number of redundant candidates. The algorithm also uses a new pruning strategy (ETP) to improve its mining capabilities through search space reduction.

The experimental results demonstrated that CHUI-PSO was overall faster than EFIM-Closed, CLS-Miner, and the heuristic CHUI-AC. CHUI-PSO took roughly 4 minutes to complete 30 tests in total, while EFIM-Closed, CLS-Miner, and CHUI-AC used 31 minutes, 8 hours, and 41 hours, respectively. The developed model provided an average accuracy of 98.8% compared to 48.6% with CHUI-AC, while its convergence rate was also much quicker. In addition, search space comparisons proved that ETP could significantly reduce the number of candidates compared to the traditional TWU model.

There are several opportunities for future work based on this paper. We plan to investigate the impact of the proposed framework on other EC-based utility mining algorithms, such as genetic algorithm, bat algorithm, ant colony optimization, or artificial fish swarm algorithm. We will also explore the possibility of developing a more efficient and effective pruning strategy by improving the overall runtime of ETP and utilizing the new utility upper bounds defined by the fitness estimates.

References

[1] R. Agrawal, T. Imieliński, and A. Swami, Mining association rules between sets of items in large databases, ACM SIGMOD Record, vol. 22, no. 2, pp. 207–216, 1993.

[2] H. Yao and H. J. Hamilton, Mining itemset utilities from transaction databases, Data & Knowledge Engineering, vol. 59, no. 3, pp. 603–626, 2006.

[3] V. S. Tseng, C. W. Wu, P. Fournier-Viger, and P. S. Yu, Efficient Algorithms for Mining the Concise and Lossless Representation of High Utility Itemsets, IEEE Transactions on Knowledge and Data Engineering, vol. 27, no. 3, pp. 726–739, 2015.

[4] W. Song and C. Huang, Mining High Utility Itemsets Using Bio-Inspired Algorithms: A Diverse Optimal Value Framework, IEEE Access, vol. 6, pp. 19568–19582, 2018.

[5] Y. Liu, W. Liao, and A. Choudhary, A Two-Phase Algorithm for Fast Discovery of High Utility Itemsets, Advances in Knowledge Discovery and Data Mining, vol. 3518, pp. 689–695, 2005.

[6] C. F. Ahmed, S. K. Tanbeer, B. S. Jeong, and Y. K. Lee, Efficient Tree Structures for High Utility Pattern Mining in Incremental Databases, IEEE Transactions on Knowledge and Data Engineering, vol. 21, no. 12, pp. 1708-1721, 2009.

[7] V. Tseng, C. W. Wu, B. E. Shie, and P. Yu, UP-Growth: An Efficient Algorithm for High utility Itemset Mining, ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 253–262, 2010.

[8] M. Liu and J. Qu, Mining High Utility Itemsets without Candidate Generation, ACM International Conference on Information and Knowledge Management, pp. 55-64, 2012.

[9] P. Fournier-Viger, C. W. Wu, S. Zida, and V. S. Tseng, FHM: Faster High-Utility Itemset Mining Using Estimated Utility Co-occurrence Pruning, Lecture Notes in Computer Science, vol. 8502, pp. 83–92, 2014.

[10] S. Zida, P. Fournier-Viger, J. C. W. Lin, C. W. Wu, and V. S. Tseng, EFIM: A Highly Efficient Algorithm for High-Utility Itemset Mining, Lecture Notes in Computer Science, vol. 9413, pp. 530–546, 2015.

[11] J. Liu, K. Wang, and M. Fung, Direct Discovery of High Utility Itemsets without Candidate Generation, IEEE 12th International Conference on Data Mining, pp. 984–989, 2012.

[12] J. M. T. Wu, J. C. W. Lin, and A. Tamrakar, High-Utility Itemset Mining with Effective Pruning Strategies, ACM Transactions on Knowledge Discovery from Data, vol. 13, no. 6, pp. 1–22, 2019.

[13] P. Wu, X. Niu, P. Fournier-Viger, C. Huang, and B. Wang, UBP-Miner: An Efficient Bit based High Utility Itemset Mining Algorithm, Knowledge-Based Systems, vol. 248, no. 4, p. 108865, 2022.

[14] C. W. Wu, P. FournierViger, J. Y. Gu, and V. S. Tseng, Mining Closed+ High Utility Itemsets without Candidate Generation, in Conference on Technologies and Applications of Artificial Intelligence, pp. 187–194, 2015.

[15] P. Fournier-Viger, S. Zida, J. C. W. Lin, C. W. Wu, and V. S. Tseng, EFIM-Closed: Fast and Memory Efficient Discovery of Closed High-Utility Itemsets, Machine Learning and Data Mining in Pattern Recognition, vol. 9729, pp. 199–213, 2016.

[16] T. L. Dam, K. Li, P. Fournier-Viger, and Q. H. Duong, CLS-Miner: Efficient and Effective Closed High-Utility Itemset Mining, Frontiers of Computer Science, vol. 13, no. 2, pp. 357–381, 2019.

[17] J. Kennedy and R. Eberhart, Particle Swarm Optimization, The International Conference on Neural Networks, vol. 4, pp. 1942–1948, 1995.

[18] J. C. W. Lin, L. Yang, P. Fournier-Viger, J. M. T. Wu, T. P. Honf, L. S. L. Wang, and J. Zhan, Mining High-Utility Itemsets based on Particle Swarm Optimization, Engineering Applications of Artificial Intelligence, vol. 55, pp. 320–330, 2016.

[19] J. C. W. Lin, L. Yang, P. Fournier-Viger, T. P. Hong, and M. Voznak, A Binary PSO Approach to Mine High-Utility Itemsets, Soft Computing, vol. 21, no. 17, pp. 5103–5121, 2016.

[20] W. Song and J. Li, Discovering High Utility Itemsets Using Set-Based Particle Swarm Optimization, Advanced Data Mining and Applications, vol. 12447, pp. 38–53, 2020.

[21] W. Fang, Q. Zhang, H. Lu, and J. C. W. Lin, High-Utility Itemsets Mining based on Binary Particle Swarm Optimization with Multiple Adjustment Strategies, Applied Soft Computing, vol. 124, p. 109073, 2022.

[22] W. Song and J. Nan, Mining High Utility Itemsets Using Ant Colony Optimization, Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery, vol. 88, pp. 98–107, 2021.

[23] S. Pramanik and A. Goswami, Discovery of Closed High Utility Itemsets Using a Fast Nature-Inspired Ant Colony Algorithm, Applied Intelligence, vol. 52, no. 8, pp. 8839–8855, 2021.

[24] P. Fournier-Viger, J. C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam, The SPMF OpenSource Data Mining Library Version 2, Joint European Conference on Machine Learning and Knowledge Discovery in Databases, pp. 36–40, 2016.

[25] S. Kannimuthu and K. Premalatha, Discovery of High Utility Itemsets Using Genetic Algorithm with Ranked Mutation, Applied Artificial Intelligence, vol. 28, no. 4, pp. 337–359, 2014.

Appendix C

TKU-PSO: An Efficient Particle Swarm Optimization Model for Top-k High-Utility Itemset Mining

TKU-PSO: An Efficient Particle Swarm Optimization Model for Top-k High-Utility Itemset Mining

Simen Carstensen[1], Jerry Chun-Wei Lin[2]

[1]University of Bergen, Bergen, (Norway)
[2]Western Norway University of Applied Sciences, Bergen, (Norway)

ABSTRACT

Top-k high-utility itemset mining (top-k HUIM) is a data mining procedure used to identify the most valuable patterns within transactional data. Although many algorithms are proposed for this purpose, they require substantial execution time when the search space is vast. For this reason, several meta-heuristic models have been applied in similar utility mining problems, particularly evolutionary computation (EC). These algorithms are beneficial as they can find optimal solutions without exploring the search space exhaustively. However, there are currently no evolutionary heuristics available for top-k HUIM. This paper addresses this issue by proposing an EC-based particle swarm optimization model for top-k HUIM, which we call TKU-PSO. In addition, we have developed several strategies to relieve the computational complexity throughout the algorithm. First, redundant and unnecessary candidate evaluations are avoided by maintaining explored solutions and estimating itemset utilities. Second, unpromising items are pruned during execution based on a threshold-raising concept we call minimum solution fitness. Finally, the traditional population initialization approach is revised to improve the model's ability to find optimal solutions in huge search spaces. Experimental results show that TKU-PSO is superior to the state-of-the-art algorithms regarding runtime, accuracy, and memory usage.

KEYWORDS

Data mining, Evolutionary computation, Particle swarm optimization, Fitness estimation,Top-k high-utility itemset, Threshold-raising strategy

I. Introduction

Data mining is a popular research field focused on extracting interesting patterns from massive datasets. These patterns are beneficial as they can help us reveal and comprehend relationships within data. Several distinctive data mining approaches exist, each specialized in locating a specific type of pattern.

Frequent itemset mining (FIM) [1] is a subfield within data mining for finding item combinations (itemsets) that occur no less than a minimum support threshold, which the user defines. In other words, FIM returns the most prevalent patterns in the data. There is a wide variety of applications for FIM, such as finding co-occurring words in a text or products often bought together in a store. However, concerning customer purchases, a business is typically interested in the patterns that contribute the most profit, and these itemsets are not necessarily the most frequent.

High-utility itemset mining (HUIM) [2] is an extension of FIM for discovering the most valuable patterns within data. The value of an itemset is quantified by a utility, representing anything the user characterizes as important. HUIM algorithms aim to discover all itemsets with utility over a user-specified minimum utility threshold. In transactional data, the utility value typically constitutes the total revenue generated by the itemset, allowing HUIM to find profitable itemsets rather than the frequent patterns produced by FIM.

Although there has been extensive research on HUIM, the algorithms tend to be unintuitive in practice. The required minimum utility threshold is challenging to set properly without knowing specific data characteristics. Typically, the user has to test multiple threshold values to find a reasonable number of patterns, which may not be feasible depending on the runtime of the model. Top-k HUIM [3] is an approach aimed at solving this by retrieving HUIs without setting a minimum utility threshold. Instead, the user provides an input parameter k, which represents a desired number of HUIs, and the algorithm's objective is to discover the k HUIs with the largest utility in the database. These models are more intuitive because it is easier to set k appropriately than the minimum utility threshold. However, top-k HUIM is computationally demanding compared to traditional HUIM because the minimum utility threshold is used to perform search space pruning. Generally, the larger the minimum utility threshold is, the fewer candidates the algorithm has to consider. Therefore, the initial search space in top-k HUIM is equivalent to HUIM with the minimum utility threshold set to zero.

Evolutionary computation (EC) [4] is a collection of meta-heuristic models utilizing biological principles to explore search spaces efficiently. The purpose of EC is to find approximate solutions to a problem within a limited number of iterations. One such method applied in various utility mining problems is particle swarm optimization (PSO) [5]. Like other EC models, PSO iteratively optimizes

* Corresponding authors:
E-mail address: simencarstensen@gmail.com (Simen Carstensen), jerrylin@ieee.org (Jerry Chun-Wei Lin).

a problem by evolving a set of candidate solutions with regard to a given measure of quality. New candidates are continuously created by inheriting traits from the best solutions in previous generations. This allows the algorithm to find optimal values without exploring the search space exhaustively.

This paper proposes a meta-heuristic model based on PSO to find the top-k HUIs, called TKU-PSO. To our knowledge, this is the first work on EC in top-k HUIM. The main contributions of the paper are listed below:

- We formulate the problem of top-k HUIM from the perspective of evolutionary computation and particle swarm optimization, in which candidate quality is evaluated based on a utility fitness function.
- We introduce several new strategies to improve the general performance of heuristics in utility mining. First, to enhance the model's ability to find optimal solutions in large search spaces, the best 1-itemsets are utilized for better population initialization. Second, redundant and unnecessary particle evaluations are avoided through fitness estimation and maintaining previously explored solutions. Finally, unpromising candidates are pruned with a threshold-raising concept called minimum solution fitness, which can reduce the search space considerably and allow faster convergence.
- We conduct a series of experiments on real- and synthetic data to evaluate the performance of the designed model against existing top-k HUIM methods. The results show that TKU-PSO outperforms the current state-of-the-art approaches in all tested datasets.

The remainder of this paper is organized as follows: Section II reviews related works. Section III presents the preliminaries and problem statement. Section IV introduces the proposed strategies and algorithm. Section V illustrates the model with an example. Section VI discusses the results of the conducted experiments. Section VII gives a conclusion of the presented work.

II. Related work

This section reviews work related to heuristic and non-heuristic models in top-k HUIM and traditional HUIM.

A. Non-heuristic top-k HUIM

Several approaches have been proposed to overcome the difficulty of setting an appropriate minimum utility threshold in traditional HUIM. Wu et al. [3] were the first to introduce top-k HUIM with the TKU algorithm. TKU is a two-phase model that relies on five threshold-raising strategies to reduce the number of candidates. The first phase of the algorithm maps potential top-k HUIs (PKHUIs) to a tree-based structure (UP-Tree) by scanning the input data twice. The second phase then determines the actual top-k HUIs by traversing the tree and evaluating the utility of the PKHUIs. To improve the performance of TKU, Ryang and Yun developed REPT [6]. REPT builds upon the same two-phase concept but applies more effective pruning strategies and thus generates fewer PKHUIs. Although the algorithm is superior to TKU, it requires an additional input parameter N, which can be challenging to select.

Due to the two-phase paradigm, TKO and REPT are subject to computationally expensive database scans during the evaluation phase of the PKHUIs. For this reason, recent methods use a more efficient one-phase strategy. TKO [7] employs a utility list data structure to hold itemset information instead of the database. The model scans the database twice to construct the utility list of 1-itemsets before it reveals the top-k HUIs without producing any candidates. As the utility list contains the details needed to evaluate itemset utilities, the model avoids the complexity associated with the second phase in the two-phase approach.

Duong et al. [8] introduced kHMC, which also uses the utility list strategy. In addition, kHMC applies three threshold-raising methods to reduce candidates and uses estimated utility co-occurrence pruning to limit the number of necessary join operations on utility lists. The model was compared to TKO and REPT and was overall more efficient.

TKEH [9] is another one-phase approach that utilizes transaction merging and database projection techniques to reduce the cost of database scans. It employs three threshold-raising strategies and two pruning strategies to evade unpromising candidates. Moreover, the utility list is exchanged with a utility array structure to calculate itemset utilities in linear time. The model performs particularly well in dense databases as transaction merging is most effective in datasets with similar transactions.

To improve the discovery of extremely long patterns, Liu et al. [10] developed TONUP. TONUP is a utility list-based, opportunistic pattern growth approach that uses five strategies for maintaining shortlisted patterns. The model grows the patterns as prefix extensions, shortlists patterns with the top k utilities, and prunes the search space with novel utility upper bounds. Experiments proved the model to be significantly faster than TKU and TKO, as well as several traditional HUIM algorithms tuned with an optimal minimum utility threshold.

THUI [11] is an approach that applies a leaf itemset utility structure to maintain itemset information and a novel utility lower bound estimation method to improve the effectiveness of threshold-raising and pruning. The results showed the model to be one to three orders of magnitude faster than kHMC and TKO, especially on dense datasets.

More recently, PTM [12] was proposed to deal with top-k HUIM on massive data using a partitioning strategy, and Ashraf et al. [13] introduced TKN for mining on negative or positive item utilities.

B. Heuristic top-k HUIM and traditional HUIM

Although the algorithms mentioned in the previous section can discover the exact top-k HUIs, they cannot efficiently deal with huge search spaces, regardless if the approach is in the one-phase or two-phase paradigm. For this reason, several heuristic algorithms have been proposed to tackle the problem of HUIM, particularly evolutionary computation (EC). These methods can find optimal solutions to large search problems without exploring the entire search space, which can be crucial for swift decision-making.

Currently, TKU-CE+ [14] is the only heuristic model available for top-k HUIM. However, it does not belong to the EC domain. It is an iterative approach based on cross-entropy that generates random samples and updates parameters to produce better samples in subsequent iterations. The authors also proposed a pruning strategy based on a critical utility value (CUV). During the initialization process, the model calculates 1-itemsets utilities and sets CUV to the k-th largest utility. Unpromising candidates are then pruned based on the transaction-weighted downward closure property (TWU model) introduced in the Two-Phase [15] algorithm for

HUIM. In addition, they used a sample refinement strategy and smoothing mutation to increase mining performance and sample diversity. As there are no other heuristics for top-k HUIM, the rest of this section outlines the most relevant works introduced for traditional HUIM. All of these approaches utilize the basic TWU model for search space pruning.

Particle swarm optimization (PSO) is an evolutionary-based procedure extensively applied in HUIM. PSO maintains a population of particles that represent potential solutions. Each particle is assigned a fitness value and a velocity vector. The fitness determines the quality of the solution, while the velocity decides how the particle evolves. At each iteration of the algorithm, the velocity is updated based on two historical particles—the personal fittest offspring of the particle (pBest) and the all-time fittest particle in the entire population (gBest). After the velocity is acquired, the particle is updated and evaluated, and pBest and gBest are redetermined. This way, the population continuously evolves towards fitter solutions. Lin et al. introduced two PSO models with HUIM-BPSO$^+$ [16] and HUIM-BPSO$^-$ [17]. The difference between the approaches is that HUIM-BPSO$^+$ uses an OR-NOR tree to produce valid item combinations and thus avoids evaluating irrelevant solutions. Song and Huang [18] used a similar approach in Bio-HUIF-PSO where a promising encoding vector check (PEV-check) is applied to prune the candidates that do not appear in any transaction. In addition, they improved population diversity by using roulette wheel selection to update gBest among the discovered HUIs. The velocity function was also replaced with a more effective bit difference strategy. More recently, Fang et al. [19] introduced HUIM-IBPSO, which uses several adjustment strategies to escape local optima and improve the overall convergence and accuracy.

The genetic algorithm (GA) is also a biologically inspired technique in which a population of chromosomes evolves towards the optimal values using selection, crossover, and mutation operations. Kannimuthu and Premalatha [20] introduced two GA models for HUIM. Their distinction is whether a minimum utility threshold is required or not. However, both methods struggle with premature convergence to local optima. To improve this, Zhang et al. introduced HUIM-IGA [21], which employs neighborhood exploration, population diversity maintenance, individual repair, and elite strategy for better search space exploration. Another GA model was proposed with Bio-HUIF-GA [18], which uses the strategies of Bio-HUIF-PSO to avoid irrelevant candidates and boost performance.

Several other types of EC have also been proposed for HUIM. Wu et al. [22] used ant colony optimization to map the search space to a routing graph and explored it using pheromone rules. Song et al. have developed approaches with artificial bee colony algorithm [23], bat algorithm [18], and artificial fish swarm algorithm [24]. There are also heuristic HUIM techniques not based on EC, such as hill climbing and simulated annealing [25].

As shown above, there is an abundance of heuristics available for HUIM but only one method for top-k HUIM. We also argue that all these previous works suffer the same fault—they spend too much time evaluating unpromising or redundant solutions. Fitness evaluation of a candidate can be extremely costly as the algorithm must scan the database to find the utility. The number of evaluations is thus essential to the performance of the model. Some try to solve this with various termination criteria. However, due to the random nature of stochastic optimization, convergence is unpredictable and challenging to measure, and the model's accuracy will typically suffer.

III. Preliminaries and problem statement

Let the set $I = \{i_1, i_2, \ldots, i_m\}$ contain m distinct items, where i_k is a unique item such that $1 \leq k \leq m$. A transactional database $D = \{T_1, T_2, \ldots, T_n\}$ is a set of n transactions, where each transaction $T_q \subseteq I$ and q is a unique transaction identifier (TID) such that $1 \leq q \leq n$. Moreover, each item $i_k \subseteq D$ is associated with a profit value, denoted $p(i_k, D)$, and a purchase quantity for each transaction, denoted $q(i_k, T_q)$. The set $X \subseteq I$ is called an itemset and is included in transaction T_q if $X \subseteq T_q$. In addition, an itemset with p items is called a p-itemset.

The database shown in Table 1 is used as a running example in this paper. It contains six transactions and six distinct items named from A to F, with the corresponding purchase quantities inside the parentheses. Table 2 shows the associated profit value of each item.

Table 1: A quantitative transactional database

TID	Trans (item : quantity)	tu
T_1	(*D*:2), (*E*:3)	16
T_2	(*A*:1), (*D*:2), (*E*:2)	17
T_3	(*A*:1), (*B*:2), (*F*:1)	6
T_4	(*C*:4), (*E*:3)	14
T_5	(*B*:3), (*C*:1), (*D*:1)	10
T_6	(*F*:9)	9

Table 2: Profit table

Item	*A*	*B*	*C*	*D*	*E*	*F*
Unit profit	3	1	2	5	2	1

Definition 1. The utility of an item i_k in a transaction T_q is denoted $u(i_k, T_q)$ and is defined as:

$$u(i_k, T_q) = q(i_k, T_q) \times p(i_k, D) \tag{1}$$

Example 1. The utility of item D in transaction T_1 is calculated as $2 \times 5 = 10$.

Definition 2. The utility of an itemset X in a transaction T_q is denoted $u(X, T_q)$ and is defined as:

$$u(X, T_q) = \sum_{i_k \in X, X \in T_q} u(i_k, T_q) \tag{2}$$

Example 2. The utility of itemset (BC) in transaction T_5 is calculated as $3 \times 1 + 1 \times 2 = 5$.

Definition 3. The utility of an itemset X in a database D is denoted $u(X)$ and is defined as:

$$u(X) = \sum_{X \in T_q, T_q \in D} u(X, T_q) \tag{3}$$

Example 3. The utility of itemset (DE) is calculated as $2 \times 5 + 3 \times 2 + 2 \times 5 + 2 \times 2 = 30$.

Definition 4. The TID-set of an itemset X in a database D is denoted $TID(X)$ and is defined as:

$$TID(X) = \{q \mid q \geq 1, q \leq n, X \in T_q, T_q \in D\} \tag{4}$$

Example 4. The TID-set of itemset (D) is {1,2,5}, as (D) occurs in T_1, T_2 and T_5.

Definition 5. The support count of an itemset X is denoted $sup(X)$ and is defined as:

$$sup(X) = |TID(X)| \tag{5}$$

Example 5. The support of itemset (D) is calculated as $|\{1,2,5\}| = 3$.

Definition 6. The transaction utility of a transaction T_q is denoted $tu(T_q)$ and is defined as:

$$tu(T_q) = \sum_{i_k \in T_q} u(i_k, T_q) \tag{6}$$

Example 6. The transaction utility of T_5 is calculated as: $3 \times 1 + 1 \times 2 + 1 \times 5 = 10$

Definition 7. The transaction weighted utility (TWU) of an itemset X is denoted $TWU(X)$ and is defined as:

$$TWU(X) = \sum_{q \in TID(X)} tu(T_q) \tag{7}$$

Example 7. The TWU of itemset (E) is calculated as 16 + 17 + 14 = 47.

Definition 8. Given a minimum utility threshold δ, an itemset X is a high transaction-weighted utilization itemset (HTWUI) if $TWU(X) \geq \delta$; otherwise, X is a low transaction-weighted utilization itemset (LTWUI). In addition, a HTWUI/LTWUI with p items is denoted p-HTWUI/p-LTWUI.

Example 8. If the minimum utility threshold is set to 20, then itemset (B) is a 1-LTWUI because $TWU(B) = 16$, while itemset (A) is a 1-HTWUI as $TWU(A) = 23$.

Definition 9. Given an minimum utility threshold δ, an itemset X is a high-utility itemset (HUI) if $u(X) \geq \delta$.

Example 9. If the minimum utility threshold is 20, then itemset (D) is a HUI as $u(D) = 25$.

Definition 10. An itemset X is a top-k HUI in a database D if its utility is among the k largest in D.

Example 10. If k is 3, then the set of top-k HUIs is {$(DE$:30), $(D$:25), $(ADE$:17)}.

Problem statement: Given a desired number of HUIs (k) and a database D, the problem of top-k HUIM is to determine the k HUIs with the largest utilities in D.

IV. Proposed algorithm for top-k huim

The proposed TKU-PSO is an iterative approach that prunes the search space before a population of particles is generated based on the remaining candidates. The top-k HUIs are discovered by evaluating and updating the population for a desired number of iterations. We will explain the model in five parts, where the first four describe the main developed strategies, and the last section introduces the complete model.

A. Minimum solution fitness

To maintain the discovered top-k HUIs, we employ a set with the maximum capacity of k (the desired number of HUIs), where each solution is sorted in descending order of utility. In other words, the solution with the lowest utility is always at the tail of the set. For simplicity throughout the paper, we call the utility of the tail-itemset the minimum solution fitness. It is defined as follows:

Definition 11. The minimum solution fitness is denoted $MSF(H)$ and is defined as:

$$MSF(H) = \begin{cases} H_k, & \text{if } |H| = k \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where H is the set of current top-k HUIs sorted in descending order of utility, and k is the desired number of HUIs

The minimum solution fitness is zero until the top-k set reaches its capacity, and the model only stores a new solution if its utility exceeds the current value. Once the set is full, new solutions replace the tail-itemset. This way, the minimum solution fitness is a dynamic threshold that grows as the algorithm progresses. The following sections explain how the model utilizes the minimum solution fitness to avoid fitness evaluations and prune candidates.

B. Population initialization strategy

The designed model represents each particle with a bit vector, called an encoding vector. The encoding vector length corresponds to the number of 1-HTWUI in the database, and each bit describes a specific item. If position i of an encoding vector is 1, then item i is included in the particle; otherwise, item i is not included. For example, assuming all items in Table 1 are 1-HTWUI, the encoding vector of itemset (ABF) is {1, 1, 0, 0, 0, 1}.

As there is no minimum utility threshold in top-k HUIM, all items are initially 1-HTWUI. However, the model removes 1-LTWUIs by setting the minimum utility threshold to the critical utility value (CUV) [14]. CUV is found by calculating all 1-itemset utilities and sorting them in descending order of utility. We utilize this to initialize the first particles to the 1-itemsets with the highest utility. This way, the model's performance becomes more consistent as the first population is identical in each run. In addition, the best solutions are often relatively small itemsets compared to the number of 1-HTWUIs. Previous models initialize the first candidates to random sizes between 1 and the number of 1-HTWUIs, which means they will generate huge itemsets in databases with many 1-HTWUIs, and the model likely falls in a local optimum.

However, if the population size is larger than the number of 1-HTWUIs, not all particles can be initialized to a unique 1-itemset. In this scenario, we generate the leftover particles with roulette wheel selection. Moreover, any particle generated with roulette wheel selection is PEV-checked. The PEV-check ensures the particle appears in at least one transaction, and the algorithm avoids evaluating irrelevant solutions. The implementation details of roulette wheel selection and PEV-check are described by Song and Huang [18].

Algorithm 1 shows the population initialization procedure. First, the database is scanned once to calculate the utility and TWU of each 1-itemset (line 1). The minimum utility threshold is then set to the k-th largest utility, and each 1-LTWUI is pruned from the database (line

Algorithm 1 Population initialization, $init()$

Input: D: a transactional database, pop_size: the population size, k: the number of desired HUIs
Output: Pop: the first population, $pBest$: initial offspring, H: the current top-k HUIs

1: calculate utility and TWU of each item in D;
2: remove items with TWU less than kth largest utility;
3: $I \leftarrow$ each 1-HTWUI, in descending order of utility;
4: $Pop, pBest, H \leftarrow \emptyset$;
5: **for** $i = 1$ to pop_size **do**
6: **if** $|I| > 0$ **then**
7: $p_i \leftarrow$ generate to the first item in I;
8: remove the first item in I;
9: **else**
10: $p_i \leftarrow$ generate with roulette wheel selection;
11: $p_i \leftarrow$ PEV-check p_i;
12: **end if**
13: fit $\leftarrow$ calculate fitness of p_i using Eq. (9);
14: **if** fit $> MSF(H)$ **then**
15: insert p_i into H;
16: **end if**
17: $Pop_i, pBest_i \leftarrow p_i$;
18: **end for**
19: **if** $pop\_size < k$ **and** $|I| > 0$ **then**
20: fill H with the remaining 1-itemsets in I;
21: **end if**
22: Return $Pop, pBest, H$;

2). The 1-HTWUIs are then sorted in descending order of utility before the population, pBest, and solutions are initialized to empty (lines 3 and 4). After that, the main loop of the procedure starts, where pop_size particles are generated (lines 5-18). At each iteration, it is checked whether the set of 1-HTWUI is empty (line 6). If not, the first 1-HTWUI in I is popped, and the particle is initialized to the 1-itemset representing this 1-HTWUI. (lines 7 and 8). Otherwise, the particle is generated with roulette wheel selection and PEV-checked (lines 9-12). Next, the created particle is evaluated by calculating its fitness (line 13). If the fitness is larger than the minimum solution fitness, the particle is put in the set of top-k HUIs as described in Section A (lines 14-16). Finally, the particle is placed in the population and its corresponding pBest before the next iteration starts (line 17). After the entire population is created, the set of top-k HUIs is filled with the remaining 1-itemsets until it is full, or there are no more 1-itemsets (lines 19-21). This step is performed to increase the minimum solution fitness quickly. Finally, the population, pBest, and current top-k HUIs are returned, and the procedure terminates (line 22).

C. Fitness evaluation strategies

The model evaluates the quality of each particle with the following fitness function:

Definition 12. The fitness of a particle p_i is defined as:

$$fit(p_i) = u(X), \tag{9}$$

where X is the itemset in the encoding vector of p_i

Calculating the utility of an itemset is often a costly operation in heuristic utility mining algorithms. The time complexity is approximately $O(s \times a)$, where s is the support of the itemset, and a is the average transaction length in the database. Therefore, it is desirable to skip the evaluation of certain unpromising candidates to improve the execution time of the model. First, many redundant particles are created during the algorithm's runtime, especially if it converges. As it is unnecessary to assess these solutions repeatedly, the model maintains each created particle in a hash set. If the set contains a specific particle, the solution is redundant, and the algorithm does not perform the fitness evaluation. By doing this, the model quickly terminates when it converges as it will primarily create explored solutions.

To further reduce the number of evaluations, we employ a strategy to estimate the fitness, described below.

Definition 13. The maximum utility of an item i in a database D is denoted $mu(i)$ and is defined as:

$$mu(i) = max\{u(i, T_q)\}_{\forall T_q \in D} \tag{10}$$

Example 11. The maximum utility of item D in Table 1 is calculated as $max\{10, 10, 5\} = 10$

Definition 14. The average utility of an item i in a database D is denoted $au(i)$ and is defined as:

$$au(i) = \left\lceil \frac{\sum_{T_q \in D} u(i, T_q)}{sup(i)} \right\rceil \tag{11}$$

Example 12. The average utility of item D in Table 1 is calculated as $\lceil \frac{10+10+5}{3} \rceil = 9$.

Definition 15. The estimated utility of an itemset X is denoted $Est(X)$ and is defined as:

$$Est(X) = sup(X) \times \sum_{i_k \in X} au(i_k) + \sigma, \tag{12}$$

where the deviation σ is calculated as:

$$\sigma = \left\lceil \frac{\sum_{i_k \in 1\text{-}HTWUI} mu(i_k) - au(i_k)}{|1\text{-}HTWUI|} \right\rceil \tag{13}$$

Example 13. Assuming all items in Table 1 are 1-HTWUI, the deviation is calculated as $\lceil \frac{(3-3)+(3-3)+(8-5)+(10-9)+(6-6)+(9-5)}{6} \rceil = 2$. Thus, the estimated utility of itemset (D) is calculated as $3 \times (9 + 2) = 33$.

The model uses the estimated utility to determine whether evaluating a particular particle is worthwhile. It does this by comparing the estimate to the fitness of pBest and the minimum solution fitness. If the estimate is less than both values, the particle will likely not improve the population or be a top-k HUI, and evaluation is thus skipped. Based on Example 13, the model ignores the evaluation of itemset (D) if the fitness of pBest and the minimum solution fitness is at least 33.

The purpose of the deviation is to avoid underestimates. An underestimate occurs when an estimate is less than the particle's fitness. Otherwise, the estimate is an overestimate. The model keeps track of the number of over- and underestimates during runtime and occasionally updates the deviation according to the following formula:

$$\sigma = \begin{cases} \frac{\sigma}{2}, & \text{if } \sigma > 1 \text{ and } \frac{u}{o} < 0.01 \\ \sigma, & \text{otherwise,} \end{cases} \tag{14}$$

where the number of over- and underestimates are denoted as o and u, respectively.

Example 14. Assuming $u = 0$ and $o = 100$. The deviation of Table 1 is updated as $\frac{2}{2} = 1$, and the estimated utility of itemset (D) is calculated as $3 \times (9 + 1) = 30$.

This way, the model adapts to the data and produces more accurate estimates as the deviation is progressively tuned. Each estimate is calculated in linear time on the size of the itemset, which is negligible compared to the complexity of finding the actual utility. The algorithm can thus save significant time when generating many low-fitness particles.

D. Particle update strategy

The designed model updates each particle towards pBest and gBest using the concept of bit difference [18]. It is defined as follows:

Definition 16. The bit difference of two particles p_i and p_j, denoted $BitDiff(p_i, p_j)$, is defined as the bitwise-XOR operation on the encoding vectors of the particles.

Example 15. Let $p_i = \{0, 1, 1, 0\}$ and $p_j = \{1, 0, 1, 0\}$, then $BitDiff(p_i, p_j) = \{1, 1, 0, 0\}$.

In other words, bit difference creates a bit vector of non-identical bits between two particles. The update procedure uses bit difference to compare a particle to pBest and gBest, and the bits set to 1 in the vector represent the items that can change in the particle. However, if the population only evolves based on the previously best solutions, the model typically falls in a local optimum due to insufficient diversity. We increase the amount of exploration by performing a random modification to the particle after the update towards pBest and gBest is complete. The model only executes this step if the current particle is a redundant solution. Thus, we avoid randomly altering new solutions to previously explored ones. The total number of bits b_i to change in a particle p_i is determined as follows:

$$b_i = b_{i1} + b_{i2} + b_{i3}, \tag{15}$$

where b_{i1}, b_{i2}, and b_{i3} are calculated as:

$$b_{i1} = \left\lfloor r_1 \times \sum BitDiff(p_i, pBest_i) \right\rfloor \tag{16}$$

$$b_{i2} = \left\lfloor r_2 \times \sum BitDiff(p_i, gBest) \right\rfloor \tag{17}$$

$$b_{i3} = \begin{cases} 1, & \text{if } p_i \in E \\ 0, & \text{otherwise} \end{cases} \tag{18}$$

where r_1 and r_2 are random numbers between [0,1], and E is the hash set of explored particles. Note that b_{i3} is determined after $b_{i1} + b_{i2}$ changes are made to the particle.

The update procedure selects b_i items and flips their corresponding bit in the particle's encoding vector. However, some 1-HTWUIs can have a TWU value less than the minimum solution fitness as it grows during runtime. An itemset containing any such 1-HTWUI cannot be part of a top-k HUI. Therefore, the algorithm always performs the bit clear operation on these items in the particle. Doing this lowers the number of potential candidates, improving the algorithm's ability to generate the actual solutions.

Algorithm 2 shows the particle update procedure. First, b_{i1} of the different items between p_i and $pBest_i$ are randomly selected and put into the set I (lines 1 and 2). Each item in I is flipped or cleared in the particle, depending on the item's TWU value and the current minimum solution fitness (lines 3-5). Next, the above process is repeated for p_i and $gBest$ (lines 6-10), before b_{i3} is calculated by identifying whether the current particle is redundant (line 11). If it is redundant, one additional random item is flipped or cleared in the particle (line 13). Finally, the updated particle is PEV-checked and returned (lines 14 and 15).

Algorithm 2 Particle update, $update()$

Input: p_i: the particle
Output: p_i': the updated particle

1: $b \leftarrow$ calculate b_{i1} using Eq. (16);
2: $I \leftarrow b$ random items set to 1 in $BitDiff(p_i, pBest_i)$;
3: **for each** $item \in I$ **do**
4: $\quad p_i \leftarrow$ flip or clear $item$ in p_i;
5: **end for**
6: $b \leftarrow$ calculate b_{i2} using Eq. (17);
7: $I \leftarrow b$ random items set to 1 in $BitDiff(p_i, gBest)$;
8: **for each** $item \in I$ **do**
9: $\quad p_i \leftarrow$ flip or clear $item$ in p_i;
10: **end for**
11: $b \leftarrow$ calculate b_{i3} using Eq. (18);
12: $item \leftarrow b$ random 1-HTWUI;
13: $p_i \leftarrow$ flip or clear $item$ in p_i;
14: $p_i \leftarrow$ PEV-check p_i;
15: return p_i';

E. TKU-PSO

Algorithm 3 Proposed TKU-PSO Algorithm

Input: D: a transactional database, k: the desired number of HUIs, pop_size: the population size, $iter$: the number of iterations.
Output: H: set of top-k HUIs

1: $Pop, pBest, H \leftarrow$ init(D, pop_size, k);
2: $gBest \leftarrow$ the fittest particle in Pop;
3: $E \leftarrow Pop$;
4: $\sigma \leftarrow$ calc. using Eq. (13);
5: **for** i = 1 to $iter$ **do**
6: $\quad$ **for** j = 1 to pop_size **do**
7: $\quad\quad Pop_j \leftarrow$ update(Pop_j);
8: $\quad\quad$ **if** $Pop_j \notin E$ **then**
9: $\quad\quad\quad X \leftarrow$ the itemset in Pop_j;
10: $\quad\quad\quad est \leftarrow$ estimate the utility of X using Eq. (12);
11: $\quad\quad\quad$ **if** $est > MSF(H)$ **or** $est > fit(pBest_j)$ **then**
12: $\quad\quad\quad\quad fit \leftarrow$ calc. fitness of Pop_j using Eq. (9);
13: $\quad\quad\quad\quad$ **if** $fit > MSF(H)$ **then**
14: $\quad\quad\quad\quad\quad$ insert Pop_j into H;
15: $\quad\quad\quad\quad$ **end if**
16: $\quad\quad\quad\quad pBest_j \leftarrow$ fittest of Pop_j and $pBest_j$;
17: $\quad\quad\quad\quad gBest \leftarrow$ fittest of Pop_j and $gBest$;
18: $\quad\quad\quad$ **end if**
19: $\quad\quad\quad E \leftarrow E \cup Pop_j$;
20: $\quad\quad$ **end if**
21: $\quad$ **end for**
22: $\quad gBest \leftarrow$ update with roulette wheel selection;
23: $\quad \sigma \leftarrow$ update using Eq. (14);
24: **end for**
25: return H;

Algorithm 3 shows the designed TKU-PSO in its entirety. The model takes as input a transactional database, the number of desired HUIs, the population size, and the number of iterations. First, the population, $pBest$, and the set of top-k HUIs are initialized by calling the initialization procedure of Algorithm 1 (line 1). Next, $gBest$ is set to the fittest particle, and the set of explored solutions is filled

with the current population (lines 2 and 3). The deviation of the maximum- and average utilities are then calculated (line 4) before the main loop of the procedure starts, where the population is iteratively updated and evaluated (lines 5-24). At each iteration, the particles are updated using Algorithm 2 (line 7). If a new particle is redundant, it is not evaluated further, and the procedure continues with the next particle in the population (line 8). Otherwise, the particle's fitness is estimated to determine if evaluation should proceed (lines 9 and 10). The particle's exact fitness is only found if the estimate is greater than the fitness of $pBest$ or the current minimum solution fitness (lines 11 and 12). If the fitness is greater than the minimum solution fitness, the particle is a new top-k HUI and is inserted into the solution set as described in Section A (lines 13-15). Then, $pBest$ and $gBest$ are updated accordingly (lines 16 and 17), and the particle is marked as explored (line 19). When the entire population is updated and evaluated, $gBest$ is reselected to one of the current top-k HUIs using roulette wheel selection (line 22). This step is not performed if $gBest$ was updated naturally during the current iteration. The deviation is then updated according to the number of over- and underestimates before the next iteration starts (line 23). Finally, when all iterations are complete, the set of top-k HUIs is returned, and the algorithm terminates (line 25).

V. An illustrated example

This section demonstrates the process of the designed model on the database in Table 1. The population size and k (the number of desired HUIs) are 3 and 2, respectively.

First, we find the TWU {A:23, B:16, C:24, D:43, E:47, F:15} and utility {A:6, B:5, C:10, D:25, E:16, F:10} of each 1-itemset. The minimum utility threshold is then set to the k-th largest utility, which is 16. Based on this, item F is pruned from the database since its TWU is less than the minimum utility threshold. The set of 1-HTWUIs is thus {A,B,C,D,E}. As the population size is less than the number of 1-HTWUI, each particle is initialized to the fittest 1-itemsets. Table 3 shows the initial population.

Table 3: The initial particles in the population

Particle	A	B	C	D	E
P_1	0	0	0	1	0
P_2	0	0	0	0	1
P_3	0	0	1	0	0

The fittest particles are placed in the set of top-k HUIs {D:25, E:16}, and the minimum solution fitness changes to the tail-itemset's utility (16). Next, pBest is initialized as a copy of the population and gBest is set to P_1. Before the update procedure starts, each current particle is marked as explored.

The update of P_2 with $r_1 = 0.7$ and $r_2 = 0.5$ goes as follows: First, $BitDiff(P_2, pBest_2)$ is calculated to {0,0,0,0,0} and $b_{21} = \lfloor 0.7 \times 0 \rfloor$, which is 0. Therefore, no items change in P_2. Next, $BitDiff(P_2, gBest)$ is calculated to {0,0,0,1,1} and $b_{22} = \lfloor 0.5 \times 2 \rfloor$, which is 1. As a result, one non-identical bit between P_2 and gBest must change, either the bit representing item D or E. Assuming item D is selected, its bit is flipped because the TWU of D (43) is larger than the minimum solution fitness (16), and P_2 becomes {0,0,0,1,1}. P_2 is not a redundant solution, and b_{23} is thus 0. The update is then complete as this encoding vector is a PEV.

Suppose the updated population is $\{P_1 : \{0,0,0,0,1\}$, $P_2 : \{0,0,0,1,1\}$, $P_3 : \{0,1,1,0,0\}\}$. Consequently, P_1 is not evaluated because it was explored in the last population. The maximum utilities of the 1-HTWUI are {A:3, B:3, C:8, D:10, E:6}, the average utilities are {A:3, B:3, C:5, D:9, E:6}, and the deviation is 1. As a result, the estimated fitness of P_2 and P_3 is 34 and 10, respectively. As the estimate of P_3 does not exceed the minimum solution fitness (16) or the fitness of $pBest_3$ (10), its fitness evaluation is skipped. The fitness of P_2 is 30, which is greater than the minimum solution fitness. The top-k HUIs is thus updated to {DE:30, D:25}, and the new minimum solution fitness is 25. In addition, $pBest_2$ and $gBest$ change to P_2. At last, the population is put in the set of explored particles, and the next iteration begins.

VI. Experimental results

This section evaluates the performance of the designed TKU-PSO against THUI, TKO, and TKU-CE+. The authors of TKO provided us with a significantly improved version of the basic TKO algorithm. We call this version TKO+ throughout the experiments. The author also gave us the source code of THUI while we downloaded TKU-CE+ from the SPMF data mining library [26]. The source code of TKU-PSO is available at GitHub[1] . All the compared algorithms are written in Java and were executed with a heap size of 2 GB on JDK 17.0.1. We performed the experiments on a 64-bit Windows 10 computer with a Ryzen 5 5600x CPU and 16 GB of 3200 MHz CL 16 RAM. Table 4 shows the characteristics of the datasets used in the comparisons. They are a mixture of real and synthetic data downloaded from SPMF. We have categorized each database as dense or sparse based on the ratio of the average transaction length to the number of distinct items in the database. Generally, sparse databases have more diverse transactions.

Table 4: Database characteristics

Dataset	#Items	#Trans	Avg.Trans.Len.	Type
Chainstore	46,086	1,112,949	7.23	Sparse
Chess	75	3,196	37	Dense
Connect	129	67,557	43	Dense
Kosarak	41,270	990,002	8.1	Sparse
Mushroom	119	8,416	23	Dense
Pumsb	2,113	49,046	74	Sparse

In all the tests, the proposed model uses 10,000 iterations with a population size of 20. The iterations and sample size in TKU-CE+ are 2,000, and the quantile parameter is 0.2, as suggested by the authors. We used a lower iteration number for TKU-CE+ because it is unclear how the sample size compares to the population size of TKU-PSO. Only a proportion of the total samples are updated each iteration. In addition, TKU-CE+ uses a termination criterion that stops the execution prematurely if it determines it has converged, and the algorithm rarely completes all iterations. Our model always performs 10,000 iterations. For these reasons, the tested input parameters are fair.

[1] https://github.com/Simencar/TKU-PSO

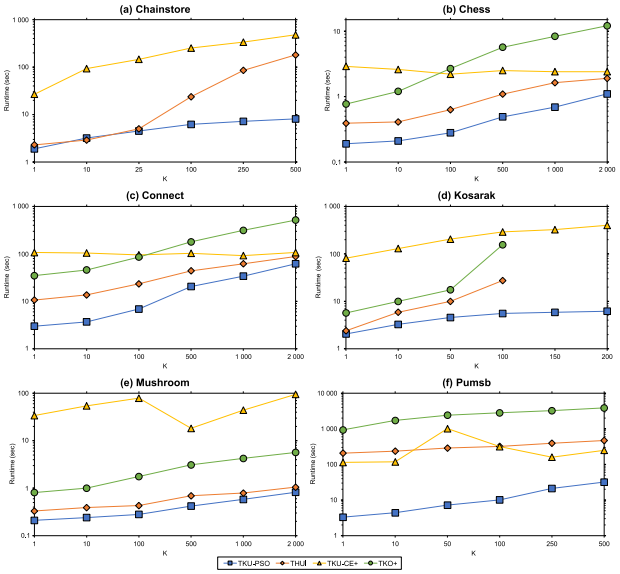

Fig. 1. The runtimes of the compared algorithms

A. Runtime

First, we compare the runtimes of the algorithms on the six datasets with various values of k. Fig. 1 shows the results.

Fig. 1(a) shows the comparison for Chainstore, where TKU-PSO and THUI used a similar amount of time for small values of k, but our model was up to 22 times faster as k increased. The heuristic TKU-CE+ was significantly slower than TKU-PSO in Chainstore. It also terminated in less than 20 iterations on all tests. TKO+ is not included in Fig. 1(a) as it ran out of memory.

The results in the dense databases Chess and Connect are almost identical to each other, Fig. 1(b-c). Our model was the fastest for all values of k, followed by THUI. Then, TKO+ was quicker than TKU-CE+ when k was less than 100, while they swapped places for higher numbers of HUIs.

Fig. 1(d) demonstrates a clear advantage of the heuristic models in Kosarak. When k was 150 and 200, THUI and TKO+ could not finish due to the search space size. We ran THUI for over 14 hours without getting a result, while TKO+ was stopped after 3 hours. Although TKU-CE+ could complete the tests on Kosarak, it repeatedly terminated after the first iteration and was still up to 64 times slower than TKU-PSO. In addition, our model outperformed THUI and TKO+ for smaller values of k.

The Mushroom dataset also shows that TKU-PSO was the most efficient model, closely followed by THUI, Fig. 1(e). TKU-CE+ was at worst 282 times slower than TKU-PSO, while the runtime also fluctuated due to the unpredictability of the termination criterion.

Finally, Fig. 1(e) shows that TKU-PSO was much faster than the other approaches in Pumsb. THUI, TKU-CE+, and TKO+ were up to 63, 141, and 390 times slower, respectively. This was the only dataset where TKU-CE+ could finish quicker than THUI, but the runtime was inconsistent, like on Mushroom.

Overall, our model achieved the best results in terms of runtime. TKU-CE+ is slower in all tests while also performing fewer iterations. THUI is generally the closest to our model, but it cannot deal with colossal search spaces, as seen on Kosarak. Kosarak has many candidates with similar utility, and the threshold-raising pruning of THUI thus becomes ineffective. The main contributions to the speed of TKU-PSO are the strategies for redundant particles and fitness estimation, which reduces the number of necessary particle evaluations. The dynamic minimum solution fitness can also improve the runtime of the model. During particle update, we avoid 1-HTWUIs with TWU less than the minimum solution fitness. Thus, the algorithm converges quicker to the point where it creates primarily redundant solutions, which are not evaluated.

B. Accuracy

The heuristic models cannot guarantee the discovery of the correct patterns before termination. Therefore, some of the found itemsets may not correspond with the actual top-k HUIs in the database. This section compares the percentage of correct top-k HUIs between TKU-PSO and TKU-CE+. In addition, we test the proposed model without the new population initialization strategy. This model is called TKU-PSO- and uses the traditional roulette wheel selection approach. We obtained the accuracy by comparing the results of the heuristic algorithms with the output of THUI. On Kosarak, the exact patterns were retrieved with the threshold-based EFIM [27] as THUI and TKO+ could not finish for large k. The accuracy was measured with the following formula:

$$Accuracy = \frac{c}{k} \times 100, \tag{19}$$

where c is the number of correct top-k HUIs discovered by the heuristic algorithm, and k is the desired number of top-k HUIs.

Table 5: The accuracy of TKU-PSO, TKU-PSO- and TKU-CE+ compared

Chainstore						
k	1	10	25	100	250	500
TKU-PSO	100 %	100 %	100 %	99 %	98 %	96 %
TKU-CE+	100 %	50 %	24 %	0 %	0 %	0 %
TKU-PSO-	100 %	100 %	100 %	98 %	93.6 %	88.8 %
Chess						
k	1	10	100	500	1,000	2,000
TKU-PSO	100 %	100 %	100 %	100 %	100 %	99.9 %
TKU-CE+	100 %	100 %	90 %	51.6 %	34 %	22.5 %
TKU-PSO-	100 %	100 %	100 %	100 %	100 %	99.9 %
Connect						
k	1	10	100	500	1,000	2,000
TKU-PSO	100 %	100 %	100 %	100 %	100 %	99.9 %
TKU-CE+	100 %	100 %	80 %	39.8 %	30 %	20.1 %
TKU-PSO-	100 %	100 %	100 %	100 %	100 %	99.9 %
Kosarak						
k	1	10	50	100	150	200
TKU-PSO	100 %	100 %	100 %	100 %	100 %	100 %
TKU-CE+	100 %	0 %	0 %	0 %	0 %	0 %
TKU-PSO-	100 %	0 %	0 %	0 %	0 %	0 %
Mushroom						
k	1	10	100	500	1,000	2,000
TKU-PSO	100 %	100 %	100 %	100 %	100 %	100 %
TKU-CE+	0 %	0 %	0 %	0 %	0.01 %	0.01 %
TKU-PSO-	100 %	100 %	100 %	100 %	100 %	100 %
Pumsb						
k	1	10	50	100	250	500
TKU-PSO	100 %	100 %	100 %	100 %	100 %	100 %
TKU-CE+	0 %	0 %	0 %	0 %	0 %	0 %
TKU-PSO-	100 %	100 %	0 %	0 %	0 %	0 %

Table 5 shows that the proposed TKU-PSO found

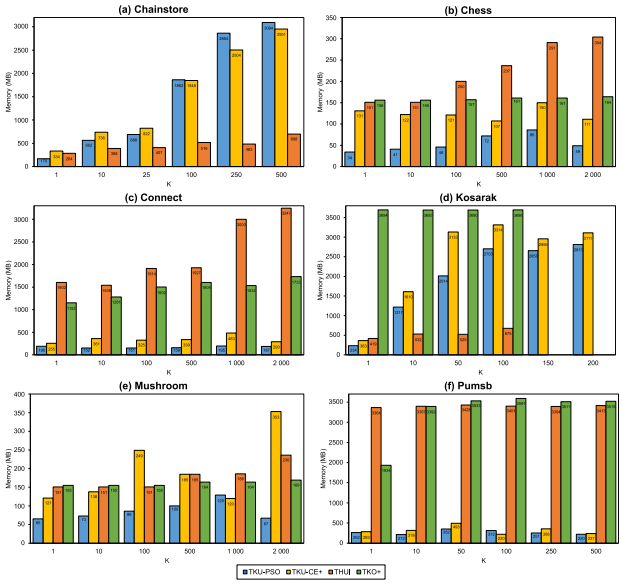

Fig. 2. The memory usage of the compared algorithms

significantly more correct top-k HUIs than TKU-CE+. In Kosarak, Mushroom, and Pumsb, the accuracy of our model was always 100%, while TKU-CE+ missed nearly all relevant patterns. In Chess and Connect, TKU-CE+ found the actual top-k HUIs for k up to 10, but the accuracy gradually fell to 22.5% and 20.1% as k increased. In contrast, TKU-PSO returned one incorrect itemset when k was 2,000 and maintained 100% accuracy in the other tests. In Chainstore, the proposed model performed slightly worse than in the other databases but still provided an accuracy of 96% or more. TKU-CE+ found the correct HUI at the smallest k but missed all relevant itemsets for k above 25.

Altogether, TKU-PSO and TKU-CE+ discovered 13,113 and 2,165 correct top-k HUIs, respectively, corresponding to an overall accuracy of 99.8% and 16.5%. In other words, our model outperformed TKU-CE+ by a wide margin in these experiments. TKU-PSO can consistently find the relevant itemsets even if the search space is huge. The Kosarak results demonstrate this as the correct solutions were returned within 10 seconds, while the non-heuristic algorithms were unable to finish in any reasonable amount of time, Fig. 1(d). The main contributor to this is the proposed population initialization strategy. TKU-PSO has better accuracy than TKU-PSO- in all the sparse databases. These datasets have massive numbers of 1-HTWUIs when k is large, but their best itemsets are relatively small in comparison. Therefore, it is advantageous to avoid initialization with roulette wheel selection as it will create too big particles and lead the model to a local optimum. TKU-PSO- discovered most of the correct solutions on Chainstore due to the PEV-check reducing the particle sizes. Nonetheless, the new population initialization strategy always provided higher or identical accuracy.

C. Memory

Finally, we compare the maximum memory usage of each algorithm on the same datasets and k as in the previous experiments. THUI and TKO+ are missing from some graphs for the reasons stated in Section A. The memory was measured using the native Java Runtime class.

According to the results in Fig. 2, TKU-PSO used the least memory on Chess, Connect, Mushroom and Pumsb, while THUI was most efficient on Chainstore and Kosarak. This is primarily caused by the database size and the algorithm's strategy for holding item information. The heuristic models store the pruned database on the heap while THUI and TKO+ construct utility list variations. Generally, the utility list approach is more efficient when the database is sparse and large, as seen on the highest k in Fig. 2(a)(d). However, our model used less memory for the smallest k in Chainstore and Kosarak because pruning reduced the database size considerably. As k increases, pruning is less effective, and memory requirements grow. TKO+ does not perform the initial pruning used by the other models. For this reason, it ran out of memory in Chainstore and performed the worst in Kosarak.

Comparing the heuristic models, TKU-PSO used overall less memory than TKU-CE+ in Fig. 2(b-f). On Chainstore, our model generates a high number of unique candidates due to the size of the search space. The memory usage then increases as the algorithm stores all explored particles. This does not happen to the same extent on the similar-sized Kosarak as the model converges early, and overall fewer candidates are examined.

Altogether, TKU-PSO was the most memory-efficient algorithm. The utility list of THUI could use less memory in extremely sparse databases but was outperformed in other scenarios.

VII. Conclusion

This paper proposed TKU-PSO, a heuristic model based on particle swarm optimization for discovering top-k high-utility itemsets. TKU-PSO introduces several efficient strategies that are fundamental to the model's performance. First, we effectively reduced the number of particle evaluations by estimating itemset utilities and maintaining explored candidates. Second, we introduced the concept of minimum solution fitness, which is utilized in several stages of the algorithm to prune unpromising candidates. Finally, we revised the traditional population initialization and thus improved the model's ability to find optimal solutions in large search spaces. The experimental results show that our approach is superior in all tested databases regarding runtime and accuracy. More specifically, TKU-PSO is up to 63, 282, and 390 times faster than THUI, TKU-CE+, and TKO+. The model achieved an overall accuracy of 99.8% compared to 16.5% with TKU-CE+, and memory usage was the smallest on 4 of 6 datasets. There are several opportunities for future work based on this paper. The developed framework can be adopted by other heuristic and evolutionary approaches such as genetic algorithm, ant colony optimization, bat algorithm, or artificial fish swarm algorithm. The proposed model can also be modified for utility mining variations such as top-k quantitative- or sequential HUIM.

References

[1] R. Agrawal, T. Imieliński, A. Swami, "Mining association rules between sets of items in large databases," *ACM SIGMOD Record*, vol. 22, pp. 207–216, 1993, doi: 10.1145/170036.170072.

[2] H. Yao, H. J. Hamilton, "Mining itemset utilities from transaction databases," *Data & Knowledge*

Engineering, vol. 59, pp. 603–626, 2006, doi: https://doi.org/10.1016/j.datak.2005.10.004.

[3] C. W. Wu, B.-E. 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, New York, NY, USA, 2012, p. 78–86, Association for Computing Machinery.

[4] D. Fogel, "What is evolutionary computation?," *IEEE Spectrum*, vol. 37, no. 2, pp. 26–32, 2000, doi: 10.1109/6.819926.

[5] J. Kennedy, R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948.

[6] H. Ryang, U. Yun, "Top-k high utility pattern mining with effective threshold raising strategies," *Knowledge-Based Systems*, vol. 76, pp. 109–126, 2015, doi: 10.1016/j.knosys.2014.12.010.

[7] V. S. Tseng, C.-W. Wu, P. Fournier-Viger, P. S. Yu, "Efficient algorithms for mining top-k high utility itemsets," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 1, pp. 54–67, 2016, doi: 10.1109/TKDE.2015.2458860.

[8] Q.-H. Duong, B. Liao, P. Fournier-Viger, T.-L. Dam, "An efficient algorithm for mining the top-k high utility itemsets, using novel threshold raising and pruning strategies," *Knowledge-Based Systems*, vol. 104, pp. 106–122, 2016, doi: 10.1016/j.knosys.2016.04.016.

[9] K. Singh, S. S. Singh, A. Kumar, B. Biswas, "TKEH: an efficient algorithm for mining top-k high utility itemsets," *Appl. Intell.*, vol. 49, no. 3, pp. 1078–1097, 2019, doi: 10.1007/s10489-018-1316-x.

[10] J. Liu, X. Zhang, B. C. Fung, J. Li, F. Iqbal, "Opportunistic mining of top-n high utility patterns," *Information Sciences*, vol. 441, pp. 171–186, 2018, doi: 10.1016/j.ins.2018.02.035.

[11] S. Krishnamoorthy, "Mining top-k high utility itemsets with effective threshold raising strategies," *Expert Systems with Applications*, vol. 117, pp. 148–165, 2019, doi: https://doi.org/10.1016/j.eswa.2018.09.051.

[12] X. Han, X. Liu, J. Li, H. Gao, "Efficient top-k high utility itemset mining on massive data," *Information Sciences*, vol. 557, pp. 382–406, 2021, doi: 10.1016/j.ins.2020.08.028.

[13] M. Ashraf, T. Abdelkader, S. Rady, T. F. Gharib, "TKN: an efficient approach for discovering top-k high utility itemsets with positive or negative profits," *Information Sciences*, vol. 587, pp. 654–678, 2022, doi: https://doi.org/10.1016/j.ins.2021.12.024.

[14] W. Song, C. Zheng, C. Huang, L. Liu, "Heuristically mining the top-k high-utility itemsets with cross-entropy optimization," *Applied Intelligence*, pp. 1–16, 2021, doi: 10.1007/s10489-021-02576-z.

[15] Y. Liu, W.-k. Liao, A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Adv Knowl Discov Data Min. Hanoi*, vol. 3518, 2005, pp. 689–695.

[16] J. C.-W. Lin, L. Yang, P. Fournier-Viger, J. M.-T. Wu, T.-P. Hong, L. S.-L. Wang, J. Zhan, "Mining high-utility itemsets based on particle swarm optimization," *Engineering Applications of Artificial Intelligence*, vol. 55, pp. 320–330, 2016, doi: https://doi.org/10.1016/j.engappai.2016.07.006.

[17] J. C.-W. Lin, L. Yang, P. Fournier-Viger, T.-P. Hong, M. Voznak, "A binary pso approach to mine high-utility itemsets," *Soft Comput.*, vol. 21, no. 17, p. 5103–5121, 2017, doi: 10.1007/s00500-016-2106-1.

[18] W. Song, C. Huang, "Mining high utility itemsets using bio-inspired algorithms: A diverse optimal value framework," *IEEE Access*, vol. 6, pp. 19568–19582, 2018, doi: 10.1109/ACCESS.2018.2819162.

[19] W. Fang, Q. Zhang, H. Lu, J. C.-W. Lin, "High-utility itemsets mining based on binary particle swarm optimization with multiple adjustment strategies," *Applied Soft Computing*, vol. 124, p. 109073, 2022, doi: 10.1016/j.asoc.2022.109073.

[20] S. Kannimuthu, K. Premalatha, "Discovery of high utility itemsets using genetic algorithm with ranked mutation," *Applied Artificial Intelligence*, vol. 28, no. 4, pp. 337–359, 2014, doi: 10.1080/08839514.2014.891839.

[21] Q. Zhang, W. Fang, J. Sun, Q. Wang, "Improved genetic algorithm for high-utility itemset mining," *IEEE Access*, vol. 7, pp. 176799–176813, 2019, doi: 10.1109/ACCESS.2019.2958150.

[22] J. M.-T. Wu, J. Zhan, J. C.-W. Lin, "An aco-based approach to mine high-utility itemsets," *Knowledge-Based Systems*, vol. 116, pp. 102–113, 2017, doi: 10.1016/j.knosys.2016.10.027.

[23] W. Song, C. Huang, "Discovering high utility itemsets based on the artificial bee colony algorithm," in *Advances in Knowledge Discovery and Data Mining - 22nd Pacific-Asia Conference, PAKDD , Proceedings, Part III*, vol. 10939 of *Lecture Notes in Computer Science*, 2018, pp. 3–14, Springer.

[24] W. Song, J. Li, C. Huang, "Artificial fish swarm algorithm for mining high utility itemsets," in *Advances in Swarm Intelligence - 12th International Conference, ICSI , Proceedings, Part II*, vol. 12690 of *Lecture Notes in Computer Science*, 2021, pp. 407–419, Springer.

[25] M. S. Nawaz, P. Fournier-Viger, U. Yun, Y. Wu, W. Song, "Mining high utility itemsets with hill climbing and simulated annealing," *ACM Trans. Manage. Inf. Syst.*, vol. 13, no. 1, 2021, doi: 10.1145/3462636.

[26] P. Fournier-Viger, C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, H. T. Lam, "The SPMF open-source data mining library version 2," *Proc. 19th European Conference on Principles of Data Mining and Knowledge Discovery (PKDD)*, pp. 36–40, 2016.

[27] S. Zida, P. Fournier Viger, C.-W. Lin, C.-W. Wu, V. Tseng, "EFIM: a fast and memory efficient algorithm for high-utility itemset mining," *Knowledge and Information Systems*, vol. 51, pp. 595–625, 05 2017, doi: 10.1007/s10115-016-0986-0.

Simen Carstensen
Received his B.Sc. degree in computer science at the University of Bergen (UiB), Norway, 2020. Currently, he is pursuing his M.Sc. degree in software development at UiB. Research interests include data mining, meta-heuristics, and machine learning.