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)

Received 16 September 2022 | Accepted 11 November 2022 | Early Access 15 January 2024

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 times 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 utilizing 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. Our results show that TKU-PSO is faster than state-of-the-art competitors in all datasets tested. Most notably, existing algorithms could not complete certain experiments due to excessive runtimes, whereas our model discovered the correct solutions within seconds. Moreover, TKU-PSO achieved an overall accuracy of 99.8% compared to 16.5% with the current heuristic approach, while memory usage was the smallest in ⅔ of all tests.

Keywords

Data Mining, Evolutionary Computation, Fitness Estimation, Particle Swarm Optimization, Threshold-Raising Strategy, Top-k High-Utility Itemset.

DOI: 10.9781/ijimai.2024.01.002

I. Introduction

Data mining is a popular field of research focused on extracting interesting patterns from massive datasets. These patterns are highly beneficial as they can help reveal and comprehend hidden 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 count, where the support describes the number of transactions that contain the itemset. 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, the usefulness of FIM is limited as it assumes frequency always defines itemset importance. Concerning customer purchases, businesses are typically interested in the patterns that contribute the most profit, and these itemsets are not necessarily among the common purchases. For this reason, data mining based on utilities has been proposed.

* Corresponding author.
E-mail addresses: simencarstensen@gmail.com (S. Carstensen), jerrylin@ieee.org (J. C.-W. Lin).

High-utility itemset mining (HUIM) [2] is an extension of FIM for discovering valuable patterns within data. The value of an itemset is quantified by a utility, and HUIM algorithms aim to reveal all itemsets with utility over a user-specified minimum utility threshold (HUIs). A key property of this strategy is that the utility can characterize different quality measures of itemsets, e.g., profit, cost, time, or even frequency. This way, HUIs can fit a wider variety of analytical problems than the frequent patterns produced by FIM. The most common application of HUIM is to identify consumer behaviors through market basket analysis [3]. However, recent studies have also shown its usefulness in problems such as emerging topic detection [4], travel pattern analysis [5], and cardiovascular disease detection [6].

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 model's runtime. Top-*k* HUIM [7] is an approach aimed at solving this by retrieving HUIs without using 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 utilities in the database. These models are more intuitive as it is easier to set *k* appropriately than the minimum utility threshold. However, top-*k* HUIM is computationally demanding compared to traditional

Please cite this article in press as:
S. Carstensen, J. Chun-Wei Lin. TKU-PSO: An Efficient Particle Swarm Optimization Model for Top-k High-Utility Itemset Mining, International Journal of Interactive Multimedia and Artificial Intelligence, (2024), http://dx.doi.org/10.9781/ijimai.2024.01.002

HUIM as the minimum utility threshold is applied to prune the search space. 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) [8] is a collection of meta-heuristic models utilizing biological principles to explore search spaces efficiently. The purpose of EC is to obtain a set of approximate solutions by analyzing problems for a limited number of iterations. One such method applied to various utility mining and search problems is particle swarm optimization (PSO) [9]–[12]. Like other EC models, PSO iteratively optimizes a problem by evolving a set of candidate solutions regarding a given quality measure. New candidates are continuously created by inheriting traits from the best solutions in previous generations, which allows the algorithm to find optimal values without exploring the search space exhaustively.

This paper proposes a 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 by maintaining previously explored candidates. Finally, to reduce the algorithm's required search space, unpromising items are pruned with a threshold-raising concept called minimum solution fitness.
- 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 gives an overview of the exhaustive algorithms proposed for HUIM and top-*k* HUIM before reviewing the heuristic alternatives.

A. High-Utility Itemset Mining

A vital challenge in HUIM is to deal with potentially huge search spaces. A database with n distinct items contains 2^n-1 HUI candidates, which means naive approaches quickly struggle due to combinatorial explosion. In this respect, Liu et al. [13] provided one of the main breakthroughs in HUIM with the Two-Phase algorithm. They introduced a technique to reduce the number of candidates based on transaction-weighted utilities (TWU). If the TWU of an itemset is less than the minimum utility threshold, then no superset extension of the itemset can be a HUI. This concept is employed during the first phase of the algorithm to only generate candidates that satisfy the TWU constraint. The second phase then identifies the actual HUIs by determining the utility of each candidate. Several other algorithms based on the two-phase approach have later been suggested, such as IHUP [14], UP-Growth [15], and MU-Growth [16]. They apply different tree structures during candidate generation to avoid creating itemsets that do not appear in the input database, thus reducing the number of necessary evaluations.

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. In order to alleviate this, Liu and Qu [17] proposed HUI-Miner, a one-phase approach without candidate generation. They developed a utility-list data structure to hold itemset information instead of the database. The model performs two database scans to construct an initial set of utility-lists before the HUIs are identified directly through utility-list join-operations. This way, the algorithm bypasses the candidate generation phase, which requires each candidate to be cached, potentially leading to memory limitations. Moreover, utility-lists enable more efficient evaluations than database scans while providing further search space pruning through the concept of remaining utility. The approach has later been improved with algorithms that reduce the computational cost associated with join-operations, some of which are FHM [18], HUP-Miner [19], and UBP-Miner [20].

There have also been introduced one-phase approaches that avoid irrelevant itemsets, similar to the tree-based, two-phase algorithms. The d^{2}HUP [21] algorithm enumerates itemsets as prefix extensions by using a hyper-structure database projection, which was shown to be generally more efficient than the earlier utility-list-based methods. Later, EFIM [22] reduced the cost of database scans with transaction merging and database projection techniques. The model utilizes a utility-array structure to hold item information, allowing linear time utility calculations. In addition, EFIM introduced subtree- and local utility upper bounds for further search space reduction.

B. Top-K High-Utility Itemset Mining

HUIM algorithms perform search space pruning by comparing different utility upper bounds to the user-specified minimum utility threshold. In top-*k* HUIM, the minimum utility threshold is initialized to zero to overcome the difficulty of selecting an appropriate value. These algorithms thus face additional search space challenges and rely on threshold-raising strategies to gradually prune unpromising candidates. However, the mining- and pruning logic are generally adopted from earlier HUIM works.

Wu et al. [7] 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 with TWU pruning. 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 [23]. REPT builds upon the same two-phase concept but applies more effective threshold-raising 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 inherits the same limitations as their HUIM relatives. Later methods thus adopt the superior one-phase strategy. TKO [24] is a HUI-Miner extension that combines novel threshold-raising with the utility-list structure. The model reveals HUIs without producing candidates and performs pruning based on TWU and remaining utility, which alleviates the computational burden associated with the earlier two-phase algorithms.

Duong et al. [25] then introduced kHMC, which also employs the utility-list strategy. In addition, kHMC applies three threshold-raising techniques to reduce candidates and uses estimated utility co-occurrence pruning and pruning by coverage to limit the number of necessary join-operations on utility-lists. The model was compared to TKO and REPT and showed overall better efficiency.

TKEH [26] is an extension of EFIM 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 the utility-array structure to facilitate linear time utility calculations. The model performs particularly well in dense databases since transaction merging is effective in scenarios with many similar transactions.

To improve the discovery of extremely long patterns, Liu et al. [27] 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 [28] 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. Experiments showed the model to be one to three orders of magnitude faster than kHMC and TKO, especially on dense datasets.

Finally, top-k HUIM extensions for specialized data environments have also been suggested. E.g., PTM [29] proposed a prefix-based partitioning strategy to accommodate massive datasets, TKN [30] has been introduced for mining data with negative or positive item utilities, and TKUS [31] has been proposed for finding patterns in sequential data. However, such extensions are outside the scope of this paper.

C. Heuristic HUIM and Top-K 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 belongs to 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+ [32] 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 TWU model from traditional HUIM [13]. In addition, they used a sample refinement strategy and smoothing mutation to increase sample diversity and mining performance. The algorithm demonstrated competitive runtimes and memory usage compared to TKU, TKO, and kHMC, although for a limited range of k. 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 new velocity is acquired, the particle is updated and evaluated, and *pBest* and *gBest* are redetermined. This way, the population evolves towards the optimal solution(s) by modifying particles according to the most promising candidates evaluated.

Lin et al. introduced two PSO models with HUIM-BPSO+ [10] and HUIM-BPSO– [33]. 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 [34] 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. [35] 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 [36] 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 [37], 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 [34], 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. [38] 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 [39], bat algorithm [34], and artificial fish swarm algorithm [40]. There are also heuristic HUIM techniques not based on EC, such as hill climbing and simulated annealing [41].

Altogether, the PSO-based approaches have shown the most promise for heuristic discovery of HUIs. The GA models can provide slightly higher accuracy but will generally use more time as their update procedures require additional computations. The other EC approaches tend to struggle with local optima in the iterative stage and thus miss a large portion of the available solutions. Based on this, the PSO algorithm is an opportune candidate for a heuristic top-k HUIM model.

Table I gives an overview of the current top-k HUIM algorithms and their main characteristics.

TABLE I. Overview of Top-*K* HUIM Algorithms

Algorithm	Type	Base-algorithm	Year
TKU [7]	Exact (two-phase)	Up-Growth [15]	2012
REPT [23]	Exact (two-phase)	MU-Growth [16]	2015
TKO [24]	Exact (one-phase)	HUI-Miner [17]	2015
kHMC [25]	Exact (one-phase)	FHM [18]	2016
TONUP [27]	Exact (one-phase)	d2HUP [21]	2018
TKEH [26]	Exact (one-phase)	EFIM [22]	2019
THUI [28]	Exact (one-phase)	HUI-Miner [17]	2019
TKU-CE+ [32]	Heuristic	Cross-entropy [42]	2021

D. Limitations of Prior Works

Heuristics are a vital research topic in data mining as they alleviate the computational burden associated with analyzing massive datasets. However, as the last section shows, 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 calculate the utility. The total number of evaluations thus significantly affects the algorithm's overall runtime. Some studies 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 decline.

Another concern with current heuristics is their accuracy in large search spaces. As the search space grows, it is increasingly difficult to generate suitable initial candidates. If they share few similarities with the best solutions, the algorithm tends to fall into local optima before generating any appropriate candidates.

The goal of this paper is thus to devise a heuristic top-k HUIM model that also mitigates these limitations of previous works.

III. Preliminaries and Problem Statement

Let the set $I = \{i_1, i_2, ..., i_m\}$ contain m distinct items, where i_k is a unique item such that $1 \le k \le m$. A transactional database $D = \{T_1, T_2, ..., T_n\}$ is a set of n transactions, where each transaction $T_q \subseteq I$ and q is a unique transaction identifier (TID) such that $1 \le q \le n$. 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 II 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 III shows the associated profit value of each item.

TABLE II. 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 III. 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 calculated by (1).

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

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 calculated by (2).

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

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 calculated by (3).

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

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 calculated by (4).

$$TID(X) = \{q \mid q \ge 1, q \le n, X \subseteq T_q, T_q \in D\} \tag{4}$$

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 calculated by (5).

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

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 $t_u(T_q)$ and is calculated by (6).

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

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 calculated by (7).

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

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 $TWU(X) \ge \delta$; 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 since $TWU\ (B) = 16$, 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 $u(X) \ge \delta$.

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 smallest 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 calculated by (8).

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

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 II 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 proposed model removes 1-LTWUIs by setting the minimum utility threshold to the critical utility value (CUV) [32]. 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 largest utilities in the database. Previous algorithms initialize the first candidates to random itemset sizes between 1 and the number of 1-HTWUIs, which means they will generate huge itemsets in databases with many 1-HTWUIs. As a result, the model likely converges to local optima as the best solutions generally are much smaller than the number of 1-HTWUIs. Initialization with 1-itemsets can thus provide particles more similar to the relevant solutions and simplify the evolutionary process. In addition, the algorithm's performance becomes more consistent as the first population is selected deterministically rather than stochastically.

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 [34].

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 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). Thereafter, 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).

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: $pi \leftarrow$ 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;

C. Fitness Evaluation Strategies

The model evaluates the quality of each particle in the population with a fitness function.

Definition 12. The fitness of a particle p_i is denoted $fit(p_i)$ and is defined in (9).

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

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

Calculating the utility of an itemset is 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 runtime, especially if the algorithm converges. As it is unnecessary to assess these solutions repeatedly, the proposed 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 approximate the fitness, which we call fitness estimation.

Definition 13. The maximum utility of an item i in a database D is denoted $mu(i)$ and is calculated by (10).

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

Example 11. The maximum utility of item D in Table II 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 calculated by (11).

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

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

Definition 15. The estimated utility of an itemset X is denoted $Est(X)$ and is calculated by (12).

$$Est(X) = sup(X) \times \left(\sum_{i_k \in X} a\, u(i_k) + \sigma \right) \tag{12}$$

where the deviation σ is defined by (13).

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

Example 13. Assuming all items in Table II are 1-HTWUI, the deviation is calculated as $\left\lceil \frac{(3-3)+(3-3)+(8-5)+(10-9)+(6-6)+(9-5)}{6} \right\rceil = 2$. 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 the 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 actual 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 (14).

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

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 I is updated as $^2/_2 = 1$, 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 fitness 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 [34]. It is defined as follows:

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

Example 15. Let $p_1 = \{0, 1, 1, 0\}$ and $p_2 = \{1, 0, 1, 0\}$, then *BitDiff* $(p_1, p_2) = \{1, 1, 0, 0\}$.

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 by (15).

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

where b_{i1}, b_{i2}, and b_{i3} are defined in (16), (17), and (18), respectively.

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

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

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

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 and thus improves the algorithm's ability to generate the actual solutions.

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: $p_i \leftarrow$ 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* $(p_i, gBest_i)$;
8: **for each** *item* $\in I$ **do**
9: $p_i \leftarrow$ 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;

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 repeats 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).

E. TKU-PSO

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 particles 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 -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).

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

V. An Illustrated Example

This section demonstrates the process of the designed model on the database in Table II. 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-HTWUIs, each particle is initialized to the 1-itemsets with the greatest utilities. Table IV shows the initial population.

TABLE IV. 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 are 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. After the algorithm terminates, the discovered top-k HUIs are (*DE*) and (*D*).

VI. Experimental Results

This section evaluates the performance of the designed TKU-PSO against THUI, TKO, and TKU-CE+. The authors of TKO provided a significantly improved version of the basic TKO algorithm. We call

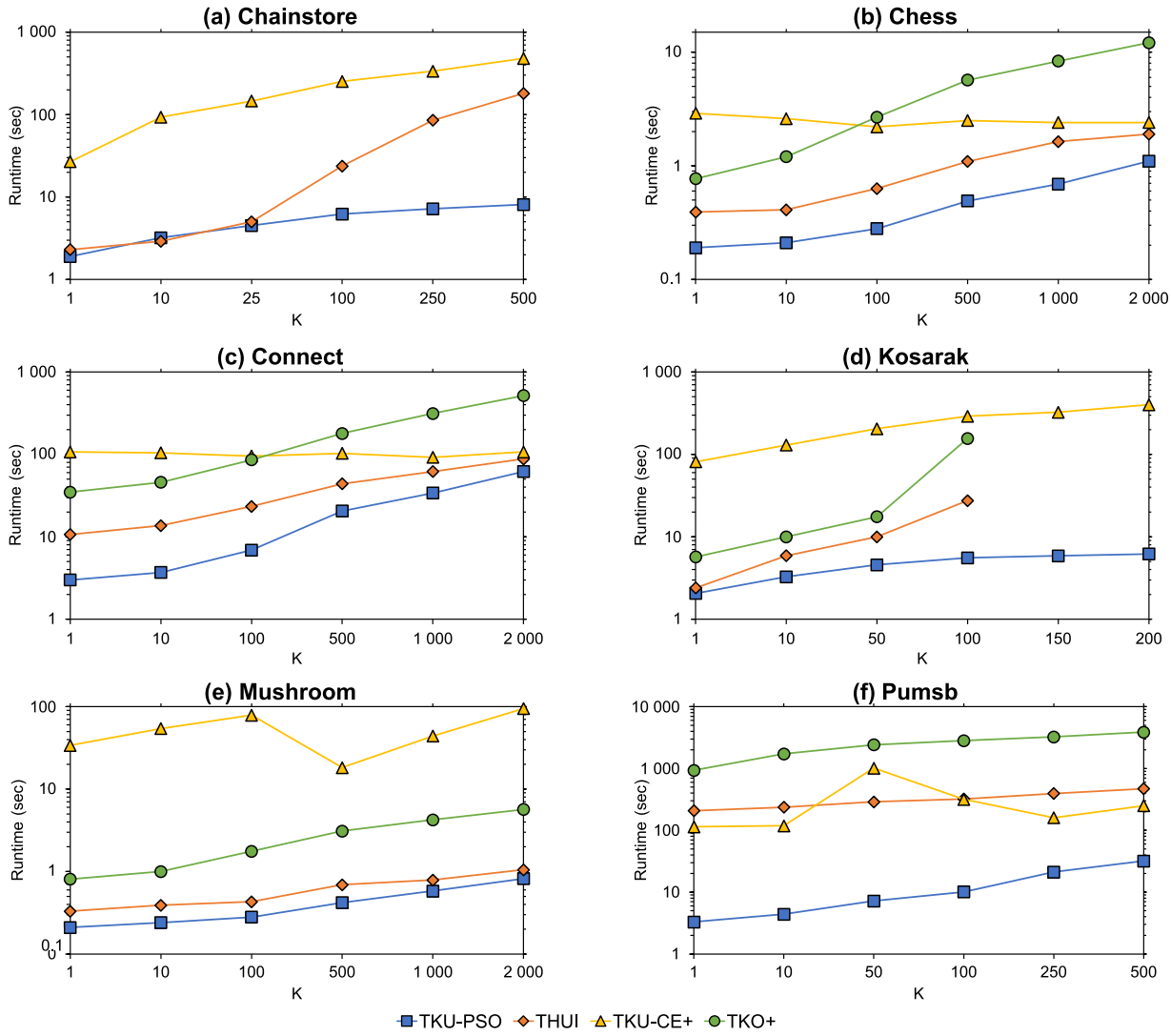

Fig. 1. The runtimes of the compared algorithms.

this version TKO+ throughout the experiments. The source code of THUI was sent to us by the author while we downloaded TKU-CE+ from the SPMF data mining library [43]. The source code for 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 V 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 V. 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

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

In all the tests, the proposed model is set to 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 the specified 10,000 iterations. For these reasons, the tested input parameters are fair.

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) displays 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 up to 59 times 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 [22] 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 HUIs.

Table VI shows that the proposed TKU-PSO found 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.

TABLE VI. 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 %

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 outperforms 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 databases 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.

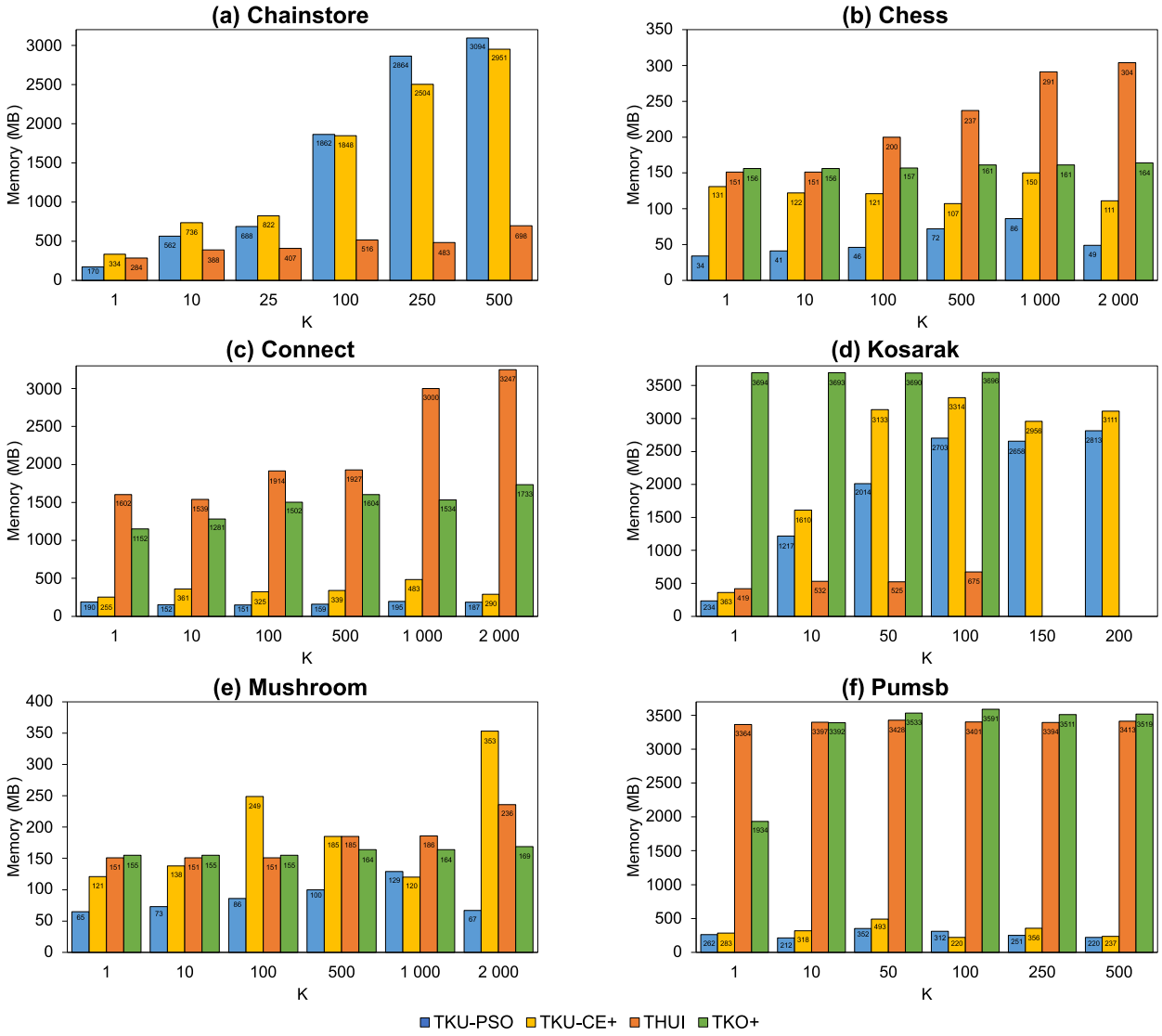

Fig. 2. The memory usage of the compared algorithms.

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 through fitness estimation and by utilizing explored candidates. Second, we introduced the concept of minimum solution fitness, which is employed 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 datasets regarding execution time and accuracy. Most notably, THUI and TKO+ could not complete certain tests due to excessive runtimes, while TKU-PSO used less than 10 seconds to discover the correct solutions. In the other experiments, TKU-PSO was up to 63, 282, and 390 times faster than THUI, TKU-CE+, and TKO+, respectively. In addition, our 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.

Although the proposed algorithm displays promising results, there are several opportunities for improving mining performance. Currently, the limiting factor to the speed of heuristics is the time

required for candidate evaluations. Future work should thus focus on strategies that can reduce this cost, e.g., by omitting certain evaluations as proposed in this paper or designing a database projection that can be more efficiently scanned. Another possibility is to introduce parallel execution in the iterative stage of the algorithm such that it can perform concurrent evaluations. Regarding accuracy, we have demonstrated that population initialization can significantly impact the algorithm's ability to discover the correct solutions. Investigating whether this process can be further enhanced is thus an important topic to explore. Finally, the developed framework can also be adopted by other evolutionary techniques and extended for different utility mining problems.

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: https://doi.org/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] P. Fournier-Viger, J. Chun-Wei Lin, T. Truong-Chi, R. Nkambou, *A Survey of High Utility Itemset Mining*, pp. 1–45. Cham: Springer International Publishing, 2019.

[4] H.-J. Choi, C. H. Park, "Emerging topic detection in twitter stream based on high utility pattern mining," *Expert Systems with Applications*, vol. 115, pp. 27–36, 2019, doi: https://doi.org/10.1016/j.eswa.2018.07.051.

[5] H. Q. Vu, G. Li, R. Law, "Discovering highly profitable travel patterns by high-utility pattern mining," *Tourism Management*, vol. 77, p. 104008, 2020, doi: https://doi.org/10.1016/j.tourman.2019.104008.

[6] M. Iqbal, M. N. Setiawan, M. I. Irawan, K. M. N. K. Khalif, N. Muhammad, M. K. B. M. Aziz, "Cardiovascular disease detection from high utility rare rule mining," *Artificial Intelligence in Medicine*, vol. 131, p. 102347, 2022, doi: https://doi.org/10.1016/j.artmed.2022.102347.

[7] 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*, 2012, p. 78–86.

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

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

[10] 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.

[11] H. Rezk, J. Arfaoui, M. R. Gomaa, "Optimal parameter estimation of solar pv panel based on hybrid particle swarm and grey wolf optimization algorithms," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 6, no. 6, pp. 145–155, 2021, doi: https://doi.org/10.9781/ijimai.2020.12.001.

[12] K. K. Verma, B. M. Singh, "Deep multi- model fusion for human activity recognition using evolutionary algorithms," *International Journal of Interactive Multimedia and Artificial Intelligence*, vol. 7, no. 2, pp. 44–58, 2021, doi: https://doi.org/10.9781/ijimai.2021.08.008.

[13] Y. Liu, W.-k. Liao, A. Choudhary, "A two-phase algorithm for fast discovery of high utility itemsets," in *Advances in Knowledge Discovery and Data Mining*, 2005, pp. 689–695.

[14] C. F. Ahmed, S. K. Tanbeer, B.-S. Jeong, 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, doi: https://doi.org/10.1109/TKDE.2009.46.

[15] V. S. Tseng, C.-W. Wu, B.-E. Shie, P. S. Yu, "Up-growth: An efficient algorithm for high utility itemset mining," in *Proceedings of the 16th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2010, p. 253–262.

[16] U. Yun, H. Ryang, K. H. Ryu, "High utility itemset mining with techniques for reducing overestimated utilities and pruning candidates," *Expert Systems with Applications*, vol. 41, no. 8, pp. 3861–3878, 2014, doi: https://doi.org/10.1016/j.eswa.2013.11.038.

[17] M. Liu, J. Qu, "Mining high utility itemsets without candidate generation," in *Proceedings of the 21st ACM International Conference on Information and Knowledge Management*, 2012, p. 55–64.

[18] P. Fournier-Viger, C.-W. Wu, S. Zida, V. S. Tseng, "Fhm: Faster high-utility itemset mining using estimated utility co-occurrence pruning," in *Foundations of Intelligent Systems*, 2014, pp. 83–92.

[19] S. Krishnamoorthy, "Pruning strategies for mining high utility itemsets," *Expert Systems with Applications*, vol. 42, no. 5, pp. 2371–2381, 2015, doi: https://doi.org/10.1016/j.eswa.2014.11.001.

[20] P. Wu, X. Niu, P. Fournier-Viger, C. Huang, B. Wang, "Ubp-miner: An efficient bit based high utility itemset mining algorithm," *Knowledge-Based Systems*, vol. 248, p. 108865, 2022, doi: https://doi.org/10.1016/j.knosys.2022.108865.

[21] J. Liu, K. Wang, B. C. Fung, "Mining high utility patterns in one phase without generating candidates," *IEEE Transactions on Knowledge and Data Engineering*, vol. 28, no. 5, pp. 1245–1257, 2016, doi: https://doi.org/10.1109/TKDE.2015.2510012.

[22] S. Zida, P. Fournier Viger, J. 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, 2017, doi: https://doi.org/10.1007/s10115-016-0986-0.

[23] 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: https://doi.org/10.1016/j.knosys.2014.12.010.

[24] 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: https://doi.org/10.1109/TKDE.2015.2458860.

[25] 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: https://doi.org/10.1016/j.knosys.2016.04.016.

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

[27] 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: https://doi.org/10.1016/j.ins.2018.02.035.

[28] 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.

[29] 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: https://doi.org/10.1016/j.ins.2020.08.028.

[30] 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.

[31] C. Zhang, Z. Du, W. Gan, P. S. Yu, "Tkus: Mining top-k high utility sequential patterns," *Information Sciences*, vol. 570, pp. 342–359, 2021, doi: https://doi.org/10.1016/j.ins.2021.04.035.

[32] 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: https://doi.org/10.1007/s10489-021-02576- z.

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

[34] 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: https://doi.org/10.1109/ACCESS.2018.2819162.

[35] 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: https://doi.org/10.1016/j.asoc.2022.109073.

[36] 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: https://doi.org/10.1080/08839514.2014.891839.

[37] 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: https://doi.org/10.1109/ACCESS.2019.2958150.

[38] 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: https://doi.org/10.1016/j.knosys.2016.10.027.

[39] 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*, vol. 10939, 2018, pp. 3–14.

[40] W. Song, J. Li, C. Huang, "Artificial fish swarm algorithm for mining high utility itemsets," in *Advances in Swarm Intelligence - 12th International Conference*, vol. 12690, 2021, pp. 407–419.

[41] M. S. Nawaz, P. Fournier-Viger, U. Yun, Y. Wu, W. Song, "Mining high utility itemsets with hill climbing and simulated annealing," *ACM Transactions on Management Information Systems*, vol. 13, no. 1, 2021, doi: https://doi.org/10.1145/3462636.

[42] R. Rubinstein, "The cross-entropy method for combinatorial and continuous optimization," *Methodology and computing in applied probability*, vol. 1, no. 2, pp. 127–190, 1999, doi: https://doi.org/10.1023/A:1010091220143.

[43] P. Fournier-Viger, J. C.-W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, H. T. 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.

Simen Carstensen

Simen recieved his B.Sc. in Computer Science (2020) from the University of Bergen (UiB), and a subsequent M.Sc in Software Engineering (2022) from Western Norway University of Applied Sciences (HVL). With a passion for optimizing computer systems through algorithms, his expertise is primarily centered around data mining, meta-heuristics, and machine learning. Presently, Simen applies his skills as a developer at Dataloy Systems, focusing on building efficient and streamlined solutions for the shipping industry.

Jerry Chun-Wei Lin

Jerry Chun-Wei Lin received his Ph.D. from the Department of Computer Science and Information Engineering, National Cheng Kung University, Tainan, Taiwan in 2010. He is currently a full Professor with the Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway. He has published more than 300 research articles in refereed journals (IEEE TKDE, IEEE TCYB, ACM TKDD, ACM TDS, ACM TMIS) and international conferences (IEEE ICDE, IEEE ICDM, PKDD, PAKDD). His research interests include data mining, soft computing, artificial intelligence and machine learning, and privacy preserving and security technologies. He is also the project co-leader of well-known SPMF: An Open-Source Data Mining Library, which is a toolkit offering multiple types of data mining algorithms. He also serves as the Editor-in-Chief of the International Journal of Data Science and Pattern Recognition. He is the IET Fellow, senior member for both IEEE and ACM.