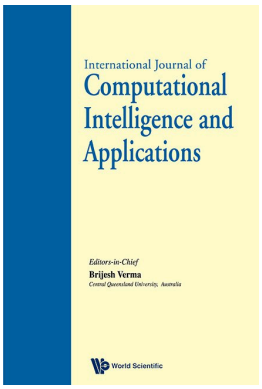
back



International Journal of Computational Intelligence and Applications

ISSN (print): 1469-0268 | ISSN (online): 1757-5885

Supports Open Access

Tools Share Recommend

Submit an article Subscribe

Journal About the Journal

Editorial Board

Editor-in-Chief

Khan M. Iftekharuddin
Professor | Batten Endowed Chair in Machine Learning | Director, Vision Lab
Batten College of Engineering and Technology
Old Dominion University
1105 Engineering Systems Building
Norfolk, VA 23529, USA
Website | Google Scholar | ORCID | Email
Research areas: signal and image processing; neural networks applications; time-frequency analysis; sensors and embedded system design; cybersecurity

Lipo Wang
Associate Professor
School of Electrical and Electronic Engineering
Nanyang Technological University
Block S1, Nanyang Avenue, Singapore 639798
Office Phone: +65 6790 6372
Office: Block S1, Room B1C 98
Website | Google Scholar | ORCID | Email
Research areas: artificial intelligence/machine learning with applications to communications, image/video processing, biomedical engineering, and data mining

Advisory Board, Founding Editor-in-Chief

Brijesh Verma
Professor

back

Science 2 (N34), Room 1.42
170 Kessels Road
Brisbane, Queensland 4111, Australia
Website | Google Scholar | ORCID | Email
Research areas: computational intelligence; pattern recognition

Associate Editors

Agnar Aamodt *(Norwegian University of Science and Technology (NTNU), Norway)*
Professor Emeritus, Professor (retired) in Computer Science and Artificial Intelligence, Faculty of Information Technology and Electrical Engineering
Research areas: artificial intelligence, and in particular, methods for data analysis and experience-based support of human problem solving and learning. In particular: methods for experience capture and reuse of human experiences, and in particular case-based reasoning in combination with generalization-based methods; machine learning and data analysis; knowledge modelling and representation; cognitive science; decision support systems

Hussein Abbass *(UNSW Sydney, Australia)*
Professor, School of Engineering and Information Technology
Research areas: artificial intelligence; swarm intelligence; swarm robotics; neural networks; trust evolutionary game theory; machine learning; evolutionary computation; human-swarm teaming; human-swarm interaction; machine teaching; machine education

James Anderson *(Brown University, USA)*
Professor Emeritus, Department of Cognitive, Linguistic & Psychological Sciences
Research areas: brain-like computing; cognitive computation; cortical computation; neural networks; theoretical neuroscience

Partha Pratim Biswas *(SP Group, Singapore)*
Principal Engineer, Power Quality
Research areas: evolutionary computation; power system optimization; cyber physical system

Salim Bouzerdoum *(University of Wollongong, Australia)*
Senior Professor of Computer Engineering, School of Electrical, Computer & Telecommunications Engineering
Research areas: image processing; artificial intelligence; computer vision; signal processing; neural networks

Antônio de Pádua Braga *(Federal University of Minas Gerais, Brazil)*
Professor, Electronics Engineering Department | Head, Computational Intelligence Laboratory
Research areas: neural networks; machine learning; clustering; data mining.

Lam Thu Bui *(Le Quy Don Technical University, Vietnam*
Associate Professor, Dean, IT Faculty
Research areas: artificial intelligence; computational intelligence: genetic algorithms, neural networksm, and heuristic techniques; multi-objective optimization; robust analysis; scheduling; planning; machine learning; forecasting; decision support systems

André Salles de Carvalho *(University of São Paulo, Brazil)*
Professor, Institute of Mathematics and Statistics

Paramita Chattopadhyay *(Indian Institute of Engineering Science and Technology, India)*
Associate Professor, Department of Electrical Engineering
Research areas: condition monitoring of electrical equipment; machine learning; signal processing; intelligent systems design

back

Aaron Chen *(Victoria University of Wellington, New Zealand)*
Senior Lecturer, School of Engineering and Computer Science
Research areas: distributed computing; systems software

Shyi-Ming Chen *(Asia University, Taiwan)*
Chair Professor, Department of Computer Science and Information Engineering
Research areas: fuzzy systems; intelligent systems; fuzzy decision making; computational intelligence; knowledge-based systems; machine learning; data mining; big data analysis; genetic algorithms; particle swarm optimization techniques; neural networks

Emilio Corchado *(University of Burgos, Spain)*
Associate Professor, Area of Computer Science, Department of Civil Engineering
Research areas: neural networks, with a particular focus on exploratory projection pursuit, maximum likelihood hebbian learning, self-organising maps, multiple classifier systems, and hybrid systems

Garrison Cotrell *(University of California San Diego, USA)*
Professor, Computer Science and Engineering Department
Research areas: applying neural networks and other computational models to problems in cognitive science and artificial intelligence, engineering, and biology

Selvathi Dharmar *(Mepco Schlenk Engineering College, India)*
Senior Professor, Department of Electronics and Communication Engineering
Research areas: biomedical imaging; biomedical signal processing; medical and biomedical image processing; medical electronics

Mike Fairhurst *(University of Kent, UK)*
Emeritus Professor, School of Engineering Research areas: image analysis; computer vision; handwriting analysis; biometric processing; neural architectures; medical image processing

João Gama *(University of Porto, Portugal)*
Professor, Faculty of Economics, Laboratory of Artificial Intelligence and Decision Support
Research areas: machine learning; learning from data streams; ensembles of classifiers; constructive induction; probabilistic reasoning

C. Lee Giles *(Pennsylvania State University, USA)*
Professor, College of Information Sciences and Technology
Research areas: AI cyberinfrastructure; deep learning; web tools; big data; data mining; business and economic models for artificial intelligence, search, and search engines

Nikola Kasabov *(Auckland University of Technology, New Zealand)*
Professor of Knowledge Engineering, Engineering, Computer, and Mathematical Sciences
Research areas: computational intelligence; neuro-computing; bioinformatics; neuroinformatics; speech and image processing; data mining; knowledge representation and knowledge discovery

Maozhen Li *(Brunel University, UK)*
Professor, College of Engineering, Design and Physical Sciences
Research areas: high performance computing; big data analytics; knowledge and data engineering; data mining and machine learning

Jialin Liu *(Southern University of Science and Technology (SUSTech), China)*
Assistant Professor, Department of Computer Science and Engineering
Research areas: AI in playing games, designing games; fair machine learning; optimisation under uncertainty (derivation-free

back

Teresa Ludermir *(Federal University of Pernambuco, Brazil)*
Professor, Center of Informaticsg
Research areas: neural networks; machine learning; artificial intelligence; hybrid intelligent systems

Rammohan Mallipeddi *(Kyungpook National University, South Korea)*
Professor, School of Electronics Engineering
Research areas: evolutionary computation; deep learning; smart home; smart grid; smart farm

Paulo S. G. de Mattos Neto *(Federal University of Pernambuco, Brazil)*
Adjunct Professor, Center of Informatics
Research areas: machine learning; time series modeling; forecasting; artificial neural networks; bioinspired algorithms; hybrid systems

Yi Mei *(Victoria University of Wellington, New Zealand)*
Associate Professor, School of Engineering and Computer Science
Research areas: evolutionary computation for combinatorial optimisation; genetic programming; automatic algorithm design; explainable AI; multi-objective optimisation; transfer/multitask learning and optimisation

Marimuthu Palaniswami *(University of Melbourne, Australia)*
Professor, Department of Electrical and Electronic Engineering
Research areas: internet of things; smart city; image and video processing; computer vision; control and optimization of communication networks; data mining and analytics; learning systems; health care

Fernando Maciano de Paula Neto *(Federal University of Pernambuco, Brazil)*
Adjunct Professor, Center of Informations
Research areas: quantum computing; quantum machine learning; machine learning; theoretical computer science; chaos and dynamical systems; decision support systems in healthcare

Witold Pedrycz *(University of Alberta, Canada)* | Website | Google Scholar
Professor, Faculty of Engineering - Electrical & Computer Engineering Dept
Research areas: computational intelligence; bioinformatics; granular computing

Harold Szu *(The Catholic University of America, USA)* | Website
Research Ordinary Professor, School of Engineering - Department of Biomedical Engineering
Research areas: nanoengineering; biomedical engineering; medicine and bioengineering

Yoshiyasu Takefuji *(Musashino University, Japan)*
Professor, Faculty of Data Science
Research areas: cybersecurity; machine learning; neural computing; energy harvesting; internet of things; automated reasoning; applied AI

Carme Torras *(Spanish Scientific Research Council (CSIC), Spain)* | Website | Google Scholar | ORCID
Professor of Research, Head of Research Line, Perception and Manipulation, Institut de Robòtica i Informàtica Industrial
Research areas: robotics; artificial intelligence; robot learning; robot vision; constraint satisfaction; robot kinematics

Burhan I. Turksen *(University of Toronto, Canada)*
Professor Emeritus, Department of Mechanical & Industrial Engineering
Research areas: industrial engineering intelligent systems; computational intelligence; neuro-fuzzy integration; linguistics to computations; signs and semiotic systems; T-formalism; data mining; data modelling; distributed information systems

Shuo Wang *(University of Birmingham, UK)*
Associate Professor, School of Computer Science

back

Stefan Wermter *(University of Hamburg, Germany)*
Professor, Department of Informatics, Knowledge Technology
Research areas: neural networks; artificial intelligence; hybrid knowledge processing; neurocognitive robotics; multimodal integration; data mining; machine learning

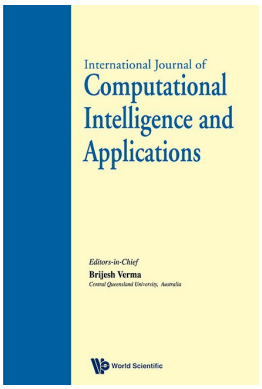
Ning Xiong *(Mälardalen University, Sweden)*
Professor of Artificial Intelligence, Division of Intelligent Future Technologies
Research areas: arious aspects of computational intelligence techniques, incuding machine learning and big data analytics, evolutionary computing, fuzzy systems, uncertainty management, as well as multi-sensor data fusion, for building self-learning and adaptive systems in industrial and medical domains

Simon X. Yang *(University of Guelph, Canada)*
Professor, School of Engineering
Research areas: robotics; intelligent systems; control systems; sensors; multi-sensor fusion; wireless sensor networks; bio-inspired intelligence; neural networks; machine learning; fuzzy systems; evolutionary computation; intelligent agriculture; intelligent communications; intelligent transportation; computational neuroscience

Xin Yao *(University of Birmingham, UK)* | Website | Google Scholar
Professor of Computer Science, School of Computer Science | Director, The Centre of Excellence for Research in Computational Intelligence and Applications (CERCIA)
Research areas: evolutionary computation (evolutionary optimisation, evolutionary learning, evolutionary design); neural network ensembles; multiple classifiers (diversity issue); meta-heuristic algorithms; data mining; global optimisation; simulated annealing; computational complexity of evolutionary algorithms; real-world applications

Jacek M. Zurada *(University of Louisville, Kentucky, USA)*
Professor, Computational Intelligence Laboratory Electrical and Computer Engineering | IEEE Life Fellow
Research areas: neural networks; deep learning; computational intelligence; data mining; image processing; VLSI circuits

back



International Journal of Computational Intelligence and Applications

ISSN (print): 1469-0268 | ISSN (online): 1757-5885

Tools Share Recommend

Submit an article Subscribe

Volume 24, Issue 01 (March 2025)

No Access

Generative Adversarial Network Optimization Algorithm Based on Adaptive Data Augmentation

Yanan Yu, Dunhuang Shi, and Qi Pan

2450021

https://doi.org/10.1142/S1469026824500214

Abstract | **Full text** | **PDF/EPUB**

Preview Abstract

No Access

Student Apartment Access Control System Based on MTCNN-FaceNet Algorithm

Jing Zhang

2450022

https://doi.org/10.1142/S1469026824500226

Abstract | **Full text** | **PDF/EPUB**

Preview Abstract

back

Modified Genetic Algorithm for Efficient High-Utility Itemset Mining

Eduardus Hardika Sandy Atmaja and Kavita Sonawane

2450024

https://doi.org/10.1142/S146902682450024X

Abstract | **Full text** | **PDF/EPUB**

Preview Abstract

No Access

A Bi-Population Competition Adaptive Interior Search Algorithm Based on Reinforcement Learning for Flexible Job Shop Scheduling Problem

Tianhua Jiang and Lu Liu

2450025

https://doi.org/10.1142/S1469026824500251

Abstract | **Full text** | **PDF/EPUB**

Preview Abstract

No Access

Music Generation Using Dual Interactive Wasserstein Fourier Acquisitive Generative Adversarial Network

Tarannum Shaikh and Ashish Jadhav

2450026

https://doi.org/10.1142/S1469026824500263

Abstract | **Full text** | **PDF/EPUB**

Preview Abstract

No Access

Rural Tourist Attractions Recommendation Model Based on Multi-Feature Fusion Graph Neural Networks

Xiangrong Zhang and Xueying Wang

2450027

https://doi.org/10.1142/S1469026824500275

Abstract | **Full text** | **PDF/EPUB**

Preview Abstract

back

A Dual-Stream Fusion Network for Human Energy Expenditure Estimation with Wearable Sensor

Shuo Xiao, Zhiyu Wang, Chaogang Tang, and Zhenzhen Huang

2450028

https://doi.org/10.1142/S1469026824500287

Abstract | Full text | PDF/EPUB

Preview Abstract

No Access

Self-Supervised Image Aesthetic Assessment Based on Transformer

Minrui Jia, Guangao Wang, Zibei Wang, Shuai Yang, Yongzhen Ke, and Kai Wang

2450029

https://doi.org/10.1142/S1469026824500299

Abstract | Full text | PDF/EPUB

Preview Abstract

No Access

Adadelta-CSA: Adadelta-Chameleon Swarm Algorithm for EEG-Based Epileptic Seizure Detection

G. Indu Salini and I. Sowmy

2450030

https://doi.org/10.1142/S1469026824500305

Abstract | Full text | PDF/EPUB

Preview Abstract

No Access

Optimizing the Hybrid Feature Selection in the DNA Microarray for Cancer Diagnosis Using Fuzzy Entropy and the Giza Pyramid Construction Algorithm

Masoumeh Motevalli, Madjid Khalilian, and Azam Bastanfard

2450031

https://doi.org/10.1142/S1469026824500317

Abstract | Full text | PDF/EPUB

Preview Abstract

No Access

back

Meng Li, Ziting Xu, Zhengjie Li, and Yajie Qi

2450032

https://doi.org/10.1142/S1469026824500329

Abstract | **Full text** | **PDF/EPUB**

Preview Abstract

No Access

An Effective Deep Learning-Based Intrusion Detection System for the Healthcare Environment

K. Balaji, S. Satheesh Kumar, D. Vivek, S. Prem Kumar Deepak, K. V. Daya Sagar, and S. Thabassum Khan

2450033

https://doi.org/10.1142/S1469026824500330

Abstract | **Full text** | **PDF/EPUB**

Preview Abstract

No Access

Calendar of Events

2583001

https://doi.org/10.1142/S1469026825830019

First Page | **Full text** | **PDF/EPUB**

Previous

International Journal of Computational Intelligence and Applications
Vol. 24, No. 1 (2025) 2450024 (27 pages)

DOI: 10.1142/S146902682450024X

World Scientific
www.worldscientific.com

Modified Genetic Algorithm for Efficient High-Utility Itemset Mining

Eduardus Hardika Sandy Atmaja *,†,‡ and Kavita Sonawane *,§

*Department of Computer Engineering
St. Francis Institute of Technology
Mumbai 400103, India

†Department of Informatics, Sanata Dharma University
Yogyakarta 55281, Indonesia
‡eduardus@student.sfit.ac.in
§kavitasonawane@sfit.ac.in

Received 24 January 2024
Revised 2 June 2024
Accepted 18 July 2024
Published 30 September 2024

In pattern mining, high-utility itemset mining (HUIM) is useful for discovering high-utility patterns. The study of HUIM using heuristic techniques reflects issues in producing better offspring. It is ineffective in terms of search space organization, population diversity, and utility calculation, which impact runtime and accuracy. It is observed that very few researchers have experimented with genetic algorithm (GA) and are still facing the same issues as mentioned before. To overcome these problems, a novel approach is proposed for HUIM using modified GA and optimized local search (HUIM-MGALS) with six potential contributions. First is linking the utility with the Bitmap dataset to reduce utility access time, leading to effective search space organization. Second, HUIM-MGALS employs a fitness scaling strategy to avoid redundancy. Third, a high-utility itemset (HUI) revision strategy is employed to explore significant HUIs. Modified population diversity maintenance strategy and iterative crossover help to preserve significant HUIs and improve search capability as fourth and fifth contributions. Sixth, the use of multiple mutations refines the wasted individuals to boost accuracy. Extensive experimentation showed that HUIM-MGALS significantly outperforms the presented algorithms, up to 8.6 times faster. It also demonstrates superior HUI discovery capabilities for both sparse and dense datasets. This is supported by the modified population diversity maintenance strategy, which is proved to be the most impactful modification for HUI discovery in HUIM-MGALS.

Keywords: Genetic algorithm; local search; high-utility itemset mining; pattern mining; data mining; HUIM-MGALS.

1. Introduction

The digital era has transformed the way humans store and analyze data. Data are now stored digitally and has grown exponentially, making it difficult to analyze manually. Consequently, researchers use data mining techniques to extract

‡Corresponding author.

interesting patterns from large amounts of data.[1] The patterns exhibit strong relationships and correlations that can aid in decision-making. Association rule mining (ARM)[1] is a prevalent method for extracting patterns and is widely used in market basket analysis to obtain valuable information for marketing strategies. Pattern mining, as the first stage of ARM, plays a significant role in generating itemset combinations that satisfy the user-specified threshold. In the second stage, these combinations are then processed by rule generation to obtain interesting patterns. The frequent itemset mining (FIM) was first introduced[2] as a pattern mining technique to discover itemsets with support greater than the user-predefined threshold. Thereafter, FIM has gained prominence and been implemented in a variety of domains. Because FIM considers frequency, it generates frequent itemsets that reveal high-frequency patterns from the dataset, but these patterns may have low profit. As a result, frequency is insufficient to represent the real problem.

High-utility itemset mining (HUIM) was introduced to address the limitations of FIM. It considers utility such as quantity, profit, or other factors based on the user's preferences. A HUI is an itemset that has utility exceeds the user-predefined threshold. Thus, HUIM generates high-utility patterns that are both frequent and profitable, and may provide better recommendations than FIM. Although HUIM offers more benefits, it suffers from an exponential search space problem due to the utility being neither monotonic nor anti-monotonic.[3,4] The patterns may spread throughout the search space, rendering existing FIM pruning strategy ineffective. To address this problem, several HUIM algorithms have been proposed, including Apriori-based,[4] tree-based,[5,6] and utility list-based algorithms.[7–9] Each of these data structures has its own advantages and disadvantages, as discussed in detail in Ref. 10 This presents new opportunities for the development of more efficient HUIM algorithms.

The previously stated algorithms are deterministic/exact algorithms. These algorithms always obtain the same output because they always follow the same algorithm sequence. Additionally, they guarantee to generate the optimal patterns. However, the search space increases exponentially as the number of distinct items, transaction length, and number of transactions increase. This leads to high runtime, as the algorithm must explore a great number of search spaces. In contrast, the heuristic algorithm provides a nearly optimal solution in a reasonable time frame.[11] Many algorithms have been proposed such as hill climbing and simulated annealing[12] to address the limitations of heuristic algorithm. Evolutionary computation (EC), which is inspired by biological evolution,[13] was also introduced. It uses metaheuristics or stochastics to generate the optimal solution. It is used in various domains, including job shop scheduling problems,[14–16] bin packing problem,[17] and composite structures optimization.[18] In HUIM, EC algorithms have been introduced such as ant colony optimization (ACO),[19,20] genetic algorithms (GAs),[21–23] particle swarm optimization (PSO),[22,24–26] and bat algorithm.[22] However, these algorithms may miss some HUIs because they only explore a subset of the search space, following the heuristic strategy. Additionally, the randomness mechanism in the evolution

procedure may consume high runtime to find HUIs. Therefore, new modifications are required to enhance the existing algorithms.

With the intention to improve the existing heuristic algorithms and focus on the oversight aspects, HUIM-MGALS is proposed which comes with the following novel contributions with respect to the HUIM context:

(1) A Bitmap dataset representation linked with utility helps to reduce fitness calculation time. Additionally, ascending transaction weighted utilization (TWU) order is applied to the promising items to effectively organize the search space.
(2) A fitness scaling strategy prevents repeated itemsets from being chosen frequently to avoid premature convergence. In addition, a HUI revision strategy for repeated HUIs is proposed by prioritizing greater TWU items to explore significant HUIs.
(3) A modified population diversity maintenance strategy involving two latest HUIs ensures that the population contains HUIs in order to obtain the optimum solution in the search process.
(4) An iterative crossover is designed to ensure that at least one of the descendants is better than their parents. This strategy may increase the search capability for the next generation.
(5) A local search function as a novelty is introduced through the application of multiple mutation functions that help to refine the wasted individuals generated by GA search functions in order to obtain a better population.
(6) Extensive experiments illustrate that HUIM-MGALS outperforms existing algorithms in terms of runtime and shows the best ability to discover HUIs.

2. Related Work

Various HUIM algorithms have been introduced since Yao *et al.*[27] presented the fundamental principles of HUIM. To address the combinatorial problem, Liu *et al.*[4] introduced a new upper bound known as the TWU model to considerably prune the search space early. Due to its efficiency, this model is used by the majority of HUIM algorithms. The pattern growth algorithms[5,6] were also introduced to address the limitations of Apriori-based algorithms. They proposed more compact tree structures to effectively store the dataset. Several strategies were also proposed to improve the mining efficiency. Subsequently, utility list-based algorithms with compact list structures were proposed.[7–9] In Ref. 7, the utility list-based algorithm was first introduced. Further, it was enhanced by Duong *et al.*[8] by integrating the buffer concept to improve memory utilization and mining efficiency. In Ref. 9, it was also improved by employing cluster computing framework with several effective load balancing mechanisms to boost runtime for small and large problems. Several HUIM variants were also proposed, including correlated HUIM,[28] closed HUIM,[29] and Top-K HUIM.[30] These variants make the results more related to real problems.

GA is an EC-based algorithm inspired by the structure of natural selection and natural genetics.[31] It can solve combinatorial problems by discovering the search space using the survival of the fittest concept and randomized information exchange. The basic GA algorithm consists of initialization, evaluation, reproduction, crossover, and mutation.[11] Some researchers have proposed new crossover operators,[32,33] as summarized by Katoch *et al.*[13] These crossover operators increase the discovery rates of solutions because they generate a greater variety of offspring. Elitism is another technique that can be used in GA.[22,23] It preserves high-quality solutions, enhancing the ability to obtain the global optimal solution. Some researchers have proposed population diversity maintenance strategies[22,23] to keep the population diverse, so that it can generate high-quality solutions even when the population contains some low-quality individuals.[26] Due to the limitations of GA's local search,[16] some researchers have combined GA with local search mechanisms.[15–18] This can be used to refine the wasted population generated by standard genetic functions. GA can be adapted to HUIM because it is capable of solving combinatorial problems.

Kannimuthu and Premalatha[21] introduced the first GA-based HUIM algorithms, one with and one without the minimum utility threshold. The GA with the minimum utility threshold works similarly to the fundamental HUIM, considering itemsets with utility higher than the threshold as HUIs. The GA without the minimum utility threshold adopts the Top-K HUIM concept, choosing HUIs by sorting the itemsets in descending order of utility and retrieving the Top-K itemsets. The negative utility values are also excluded from item/itemsets to ensure accurate utility calculations. Their algorithms rely on several basic genetic functions, such as roulette wheel selection, single-point and two-point crossover, and ranked mutation. However, these functions may limit their ability to find all possible HUIs.

Song and Huang[22] designed bio-inspired algorithm based on HUIM frameworks (Bio-HUIFs). They proposed Bio-HUIF with GA (Bio-HUIF-GA) to improve the standard roadmap of GA. It efficiently stores the dataset using Bitmap dataset representation, where transactions are transformed into a two-dimensional Boolean matrix. Columns represent items, rows represent transactions, and Boolean values indicate the presence of items in the corresponding transactions. This Bitmap dataset avoids expensive searching for itemset combinations because Bitwise-AND operation can easily retrieve transactions containing a combined itemset. To ensure candidate itemsets exist in the dataset, a promising encoding vector (PEV) check is employed. It performs Bitwise-AND operation to verify if an item combination exists in the Bitmap dataset. This also removes irrelevant items, keeping only those present in the transactions. For genetic functions, Bio-HUIF-GA utilizes roulette wheel selection, bit difference crossover, and one-point mutation. To improve HUI discovery, it incorporates a population diversity maintenance strategy. This strategy uses roulette wheel selection to choose two HUIs from the solution list, which then replace two randomly selected chromosomes in the current population. Despite having the best ability to discover HUIs using the proposed approach, it has the slowest runtime among Bio-HUIFs.

Zhang *et al.*[23] introduced HUIM based on an improved GA (HUIM-IGA). They modified the PEV check strategy[22] to an individual repair strategy. This strategy sorts items based on descending TWU order, prioritizing items likely to appear in HUIs, which presumably improves efficiency. For genetic functions, HUIM-IGA utilizes roulette wheel selection, uniform crossover, one-point mutation, and elite strategy. They also modified the population diversity maintenance strategy to preserve population diversity, allowing the algorithm to explore more HUIs. This strategy selects two consecutive HUIs from the solution list to replace two randomly selected chromosomes in the current population. Additionally, they added a neighborhood exploration strategy applied to repeated HUIs to improve HUI discovery. This strategy employs neighborhood mutation by flipping bits 1 and 0 in two random positions of the chromosome. Finally, these strategies enable HUIM-IGA to outperform the other presented algorithms.

PSO is another EC-based algorithm. It has been used by some researchers to solve HUIM problems, such as Lin *et al.*[24] who proposed HUIM based on binary PSO with sigmoid function (HUIM-BPSOsig). Unlike GA, PSO has only few parameters, such as particle and velocity. HUIM-BPSOsig employs the TWU model to construct the initial particle size for search space reduction during the evolution process. It also uses the sigmoid function to update the particle process, resulting in more efficient HUI discovery. Lin *et al.*[25] later improved it by proposing an OR/NOR-tree data structure to prevent invalid itemsets. This tree maintains information about the possible extension of itemsets, thus irrelevant combinations can be detected and pruned without scanning the database multiple times. This approach leads to improved HUIs discovery and efficiency. However, it still generates fewer HUIs in some datasets.

Song and Huang[22] proposed Bio-HUIF with PSO (Bio-HUIF-PSO). Like other Bio-HUIF algorithms, it utilizes the Bitmap dataset representation and the PEV check strategy. Specifically, they modified the standard velocity formula using bit difference approach. During particle updating, the new velocity indicates the number of bits that must be changed to achieve the optimal values for HUI discovery. Experimental results demonstrate that BIO-HUIF-PSO has efficient runtime but generates fewer HUIs. Gunawan *et al.*[26] recently proposed HUIM without a minimum utility threshold based on binary PSO (HUIM-BPSO-nomut). Unlike other PSO-based algorithms, it does not require a minimum utility threshold. The initial particle is defined using transactions with the k-highest utility, where k is the population size. This strategy aims to leverage the most promising particles for better solution generation. During particle updating, a combination of the v_max method and a piece-wise linear function is employed to update velocity, enhancing the algorithm's ability to discover HUIs. According to their experiments, HUIM-BPSO-nomut discovers more HUIs with greater total utility. However, it may obtain lower total utility in non-complex datasets.

Artificial fish swarm algorithm is another optimization algorithm inspired by the collective behavior of fish. Song *et al.*[34] used this approach to model the HUIM

problem. The proposed algorithm, HUIM-AF, discovers HUIs by recording only the current position. This leads to the discovery of a large number of diverse results with the help of follow, swarm, and prey operations. The roulette wheel selection is used to initialize the position vector (PV). Then, the bit difference operation is used to update PV during mining. Finally, if the updated PV has a utility higher than the threshold, it is added to the collection of HUIs. Based on their experiments, HUIM-AF discovers more HUIs, However, the runtime is not consistent, with some datasets exhibiting high runtime.

3. Preliminaries

Consider $I = \{i_1, i_2, i_3, \ldots, i_j\}$ is a collection of items from a dataset D, where D contains transactions $\{T_1, T_2, T_3, \ldots, T_k\}$. A transaction T_k represents an itemset, where an itemset X is a collection of items and $X \subseteq I$. Each item i in T_k has a quantity q and a profit p. Tables 1–4 present an instance of D, an instance of profit, TWU values, and the properties of HUIM, respectively.

4. Proposed Algorithm (HUIM-MGALS)

Based on a thorough literature survey, specifically in Sec. 2, it is observed that a few concepts are still not being focused on when applying GA for HUIM.[21–23] This research work tries to cover these aspects by proposing modifications to various phases of GA, along with adding a new phase (Algorithm 6). The proposed HUIM-MGALS with all novel contributions and modifications is presented in Algorithm 1 as complete. The preprocessing steps are described in lines 1–6. The first population is

Table 1. Instance of D.

T_k	Transaction
T_1	(b:3) (f:2)
T_2	(a:1) (b:4) (c:1) (e:4)
T_3	(c:2) (e:1) (f:2)
T_4	(b:3) (e:4) (f:2)
T_5	(b:3) (c:2) (d:2) (e:3) (f:1)

Table 2. Instance of profit.

Item	a	b	c	d	e	f
Profit	5	1	4	3	2	4

Table 3. Items with their TWUs.

Item	a	b	c	d	e	f
TWU	21	78	66	27	85	75

Table 4. Properties of HUIM.

No.	Definition	Equation	Eq. Nos.
1	Utility of i in T_k	$u(i_j, T_k) = q(i_j, T_k) \times p(i_j)$	(1)
2	Utility of X in T_k	$u(X, T_k) = \sum_{i \in X \wedge X \subseteq T_k} u(i, T_k)$	(2)
3	Utility of X in D	$u(X) = \sum_{X \subseteq T_k \wedge T_k \in D} u(X, T_k)$	(3)
4	Utility of a transaction	$\text{tu}(T_k) = \sum_{i \in T_k} u(i, T_k)$	(4)
5	Total utility of D	$\text{TU}(D) = \sum_{T_k \in D} \text{tu}(T_k)$	(5)
6	TWU of X	$\text{TWU}(X) = \sum_{X \subseteq T_k \wedge T_k \in D} \text{tu}(T_k)$	(6)
7	Minimum utility threshold	$\sigma = \text{TU}(D) \times \delta$, where δ is minimum utility threshold in percent	(7)
8	HUI	$u(X) \geq \sigma$ Itemset X is a HUI, if X has utility equal to or greater than σ	(8)
9	Promising items	$\text{TWU}(i) \geq \sigma$ Each item $i \in I$ is a promising item if i has TWU equal to or greater than σ. If $\sigma = 24$ then b, c, d, e, and f are promising items. Table 3 shows the TWU values of I.	(9)

initialized in line 7 using Algorithm 2. In lines 8–20, the genetic functions are applied iteratively until the maximum fitness evaluation number is reached. The iteration begins by maintaining population diversity using Algorithm 3 (line 9). Lines 10–14 describe the crossover function using Algorithm 4. Both parents ($c1$ and $c2$) are chosen randomly using roulette wheel selection with scaling factor (lines 11 and 12). The new population is then mutated using Algorithm 5 in line 15. The elite strategy is applied in lines 16–18 to preserve high-quality solutions for the next generations by selecting the top-N individuals from both parents and offspring.[23] The last step is executing the local search function to refine the wasted chromosome using Algorithm 6 (line 19). The specific details of these steps are explained in the next subsections.

4.1. *Preprocessing*

Lines 1–6 in Algorithm 1 describe the preprocessing stage, where the transactional dataset D (line 1) and distinct items I (line 2) are retrieved from a file. I is then filtered using Eq. (1) to obtain promising items I^* (line 3) by calculating TWU values using Eqs. (4) and (6). The promising items are obtained using the TWU model.[4] The minimum utility threshold is also calculated using Eqs. (5) and (7). We incorporate two new strategies into this stage: (i) sorting the promising items in ascending TWU order and (ii) new representation for the Bitmap dataset. We consider sorting the promising items in ascending TWU order (line 4), which has not been applied to other GA-based algorithms. This sorting order may reduce the search space and improve efficiency during exploration.[7,35] The ascending TWU order is the most

Algorithm 1. HUIM-MGALS.

Input:

- max_ev, maximum fitness evaluation
- pop_size, population size
- σ, minimum utility threshold

Output:

- HUIs, high-utility itemsets

1 $D = \{T_1, T_2, T_3, \ldots, T_k\}$
2 $I = \{i_1, i_2, i_3, \ldots, i_j\}$
3 $I^* = \{i \mid i \in I, \sum_{i \subseteq T_k \wedge T_k \in D} \mathrm{tu}(T_k) \geq \sigma\}$ ▷ Eq. (9)
4 $I^* = \{i_1 \prec i_2 \prec i_3 \prec \cdots \prec i_j\}$ //sorting
5 $D^* = \{T \mid T \in D, \forall\, i \in T \wedge \forall\, i \in I^*\}$
6 $B_{k,j} = \left[\begin{cases} 1 & \text{if } i_j \in T_k \\ 0 & \text{otherwise} \end{cases}, \begin{cases} u(i_j, T_k) & \text{if } i_j \in T_k \\ 0 & \text{otherwise} \end{cases} \right]$ ▷ Eq. (10)
7 population = initialize() ▷ Alg. 2
8 **while** ev_num<max_ev **do**
9 population = maintainDiversity() ▷ Alg. 3
10 **while** newPop_size<pop_size **do**
11 $c1 = \mathrm{rand}\left(\left\{c \mid Pc = \frac{c.f \times c.s}{\sum_{l=1}^{\text{pop\_size}} \text{population}_l.f \times \text{population}_l.s}\right\}\right)$
12 $c2 = \mathrm{rand}\left(\left\{c \mid Pc = \frac{c.f \times c.s}{\sum_{l=1}^{\text{pop\_size}} \text{population}_l.f \times \text{population}_l.s}\right\}\right)$
13 newPopulation $\ll$ crossover($c1, c2$) ▷ Alg. 4
14 **end while**
15 newPopulation = mutation() ▷ Alg. 5
16 newPopulation $\ll$ population //insert population
17 newPopulation = $\{c_1 \succ c_2 \succ c_3 \succ \cdots \succ c_l\}$ //sorting
18 population = $\{c \mid c_{1,2,3,\ldots,\text{pop\_size}}, c \in \text{newPopulation}\}$
19 population = localSearch() ▷ Alg. 6
20 **end while**

efficient compared to the descending TWU order[23] and the lexicographical order[22] because it requires fewer dataset scans to calculate the utility since lower TWU items appear in the dataset less frequently. This mechanism avoids unnecessary search space with low-utility itemsets and focuses on HUIs. It is also useful for the individual repair strategy.

Based on the promising items, the dataset is revised by removing low-utility items (line 5). This revised dataset D^* is used to construct Bitmap dataset representation $B(D^*)$ to optimize the mining process (line 6). The Bitmap contains $k \times j$ matrix where k is the size of D^* and j is the size of I^*. Each value in $B(D^*)$ corresponds to transaction T_k and item i_j. It has two values that are defined by Eq. (10). The value

Algorithm 2. Population initialization.

Input:

- $B_{k,j}$, Bitmap dataset
- TWU, TWU values of promising items
- σ, minimum utility threshold

Output:

- population, collection of N individuals

while pop_size<N **do**
  $m = \text{rand}(|I^*|)$ //random value between 0 and $|I^*|$
//new chromosome c to be created with its fitness f scaling factor s, and length l
  $c.c = \left\{ i \,|\, i_{1,2,\ldots,m}, i = \text{rand}\left(\left\{ i \,|\, Pi = \frac{\text{TWU}(i)}{\sum_{j=1}^{|I^*|} \text{TWU}(I_j^*)} \right\} \right) \right\}$ ▷ Eq. (11)
  $c = \text{repair}(c)$
  $c.f = \sum_{(c.c \subseteq T_k \wedge T_k \in D^*)} u(c.c, T_k)$ ▷ Eq. (3)
  population $\ll c$
  if $c.f \geq \sigma$ **do** ▷ Eq. (8)
HUIs $\ll \begin{cases} c & \text{if } c \notin \text{HUIs} \\ \text{reduce}(c.s) = c.s - \frac{1}{\text{pop\_size}} & \text{otherwise} \end{cases}$
  end if
end while

Algorithm 3. Maintain population diversity.

Input:

- HUIs, HUI list
- population, collection of N individuals

Output:

- population, replaced population

$c1 = \text{rand}(\text{pop\_size})$
$c2 = \text{rand}(\text{pop\_size})$
if HUIs_size = 1 **do**
  population[$c1$] = HUIs[0]
  population[$c2$] = HUIs[0]
else if HUIs_size > 1 **do**
  population[$c1$] = HUIs[HUIs_size − 2]
  population[$c2$] = HUIs[HUIs_size − 1]
end if

Algorithm 4. Iterative uniform crossover.

Input:

- parent1, a parent chromosome
- parent2, a parent chromosome

Output:

- population2, collection of N new individuals

```
1  c1 = parent1
2  c2 = parent2
3  do
4     while x < c.l do
5        if rand() > 0.5 do
6           swap(c1.c[x], c2.c[x])
7     end if
8     end while
9     c1 = repair(c1)
10    c2 = repair(c2)
11    c1.f = Σ_{(c1.c⊆T_k ∧ T_k∈D*)} u(c1.c, T_k)        ▷ Eq. (3)
12    c2.f = Σ_{(c2.c⊆T_k ∧ T_k∈D*)} u(c2.c, T_k)        ▷ Eq. (3)
13    if c1.f ≥ σ do                                    ▷ Eq. (8)
14 HUIs ≪ { c1                                  if c1 ∉ HUIs
          { reduce(c1.s) = c1.s − 1/pop_size    otherwise
15    end if
16    if c2.f ≥ σ do                                    ▷ Eq. (8)
17 HUIs ≪ { c2                                  if c2 ∉ HUIs
          { reduce(c2.s) = c2.s − 1/pop_size    otherwise
18    end if
19    if c1.f ≥ parent1.f || c2.f ≥ parent2.f do
20       population2 ≪ c1
21       population2 ≪ c2
22       break
23    end if
24 while c1.f < parent1.f && c2.f < parent2.f
```

$B_{k,j}[1]$ is 1 if item i_j exists in transaction T_k, otherwise it is 0. Unlike existing work, we added the second value to save runtime to calculate utility from D^*. If item i_j exists in transaction T_k then $B_{k,j}[2]$ contains the utility of item i_j in transaction T_k (Eq. (1)). This data structure eliminates the need for searching during utility calculation. The Bitmap dataset representation of D^* is shown in Table 5.

$$B_{k,j} = \left[\begin{cases} 1 & \text{if } i_j \in T_k, \\ 0 & \text{otherwise,} \end{cases} \begin{cases} u(i_j, T_k) & \text{if } i_j \in T_k, \\ 0 & \text{otherwise.} \end{cases} \right] \tag{10}$$

Algorithm 5. Mutation.

Input:

- population2, collection of N individuals

Output:

- population2, collection of N mutated individuals

while $x <$ population2_size **do**
  $c =$ population2$[x]$
  $n =$ rand$(|I^*|)$
  flip$(c.c[n])$
  $c =$ repair(c)
  if $c \in$ HUIs **do**
    $n =$ rand$(|I^*|)$
    $c.c \ll i_n \begin{cases} 0, & \text{if } i_n = 1 \text{ then } i_{c.l-a} = 1 \\ 1, & \text{if } i_n = 0 \text{ then } i_{0+a} = 0 \\ & \text{where } a = 0, 1, \ldots, c.l \end{cases}$
    $c =$ repair(c)
  end if
  $c.f = \sum_{(c.c \subseteq T_k \wedge T_k \in D^*)} u(c.c, T_k)$ ▷ Eq. (3)
  if $c.f \geq \sigma$ **do** ▷ Eq. (8)
  $\text{HUIs} \ll \begin{cases} c & \text{if } c \notin \text{HUIs} \\ \text{reduce}(c.s) = c.s - \frac{1}{\text{pop\_size}} & \text{otherwise} \end{cases}$
  end if
end while

4.2. *Initializing population*

The initialization stage incudes several steps to generate a population that contains N number of individuals. Each individual is represented as a chromosome by an encoding vector, with 1 indicating that an item appears in a chromosome and 0 indicating that an item does not appear in a chromosome. The chromosome has length $|I^*|$ and each value in the chromosome represents a sequential item in I^*. Thus, a chromosome is an itemset. Figure 1 depicts a chromosome for itemset $\{d, f, b\}$. During the generation or modification of a chromosome, there is a possibility that the chromosome representing an itemset may not exist in the dataset. This wastes runtime and makes the search space larger. Therefore, an individual repair strategy[22,23] can be applied to modify the chromosome to form an itemset that exists in the dataset by checking pairs of items consecutively. This strategy is similar to exploring search space in exact algorithms. Since the promising items are sorted in ascending TWU order, as in exact algorithms, the individual repair strategy may form an effective combination that leads to efficient mining.

Algorithm 6. Local search (optimized).

Input:

- population, collection of N individuals

Output:

- population, collection of N refined individuals

1 **while** $x <$ population_size **do**
2 $c =$ population$[x]$
3 **if** $c.f \geq \sigma$ **do**
4 **continue**
5 **else**
6 best $= c$
7 **while true do**
8 $f(y) =$ rand$(f(y))$
where $f(y) \in F\{f_1(), f_2(), \ldots, f_n()\}$
9 $f(y) \ll c.c$
10 $c =$ repair(c)
11 $cf = \sum_{(c.c \subseteq T_k \wedge T_k \in D^*)} u(c.c, T_k)$ ▷ Eq. (3)
12 **if** $c.f \geq \sigma$ **do** ▷ Eq. (8)
13 HUIs $\ll \begin{cases} c & \text{if } c \notin \text{HUIs} \\ \text{reduce}(c.s) = c.s - \frac{1}{\text{pop\_size}} & \text{otherwise} \end{cases}$
14 **end if**
15 **if** $c.f \geq$ best.f **do**
16 best $= c$
17 **else**
18 population$[x]$ = best
19 **break**
20 **end if**
21 **end while**
22 **end if**
23 **end while**

The process of population initialization is depicted in Algorithm 2. The algorithm iterates the initialization process until the population has N individuals (line 1). The number of bits 1 in a chromosome is generated randomly and assigned to m (line 2). Then, each index that represents an item i is set to 1 based on roulette wheel selection using Eq. (11) (line 4). In line 5, the chromosome is repaired using individual repair strategy. In line 6, Eq. (3) which requires Eq. (2) is used to calculate the fitness value. Since the item/itemset and its utility are available in $B_{k,j}$, the runtime to search and calculate the fitness is reduced. The generated chromosome is then added to the population, increasing the population size by 1 (line 7). In line 8, Eq. (8) is used to

Table 5. Bitmap dataset representation of D^*.

	d	c	f	b	e
T_1	$[0,0]$	$[0,0]$	$[1,8]$	$[1,3]$	$[0,0]$
T_2	$[0,0]$	$[1,4]$	$[0,0]$	$[1,4]$	$[1,8]$
T_3	$[0,0]$	$[1,8]$	$[1,8]$	$[0,0]$	$[1,2]$
T_4	$[0,0]$	$[0,0]$	$[1,8]$	$[1,3]$	$[1,8]$
T_5	$[1,6]$	$[1,8]$	$[1,4]$	$[1,3]$	$[1,6]$

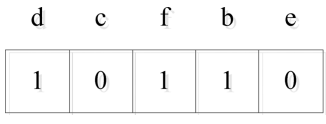


Fig. 1. An instance of a chromosome.

compare the fitness value of the chromosome to the minimum utility threshold. If it is equal to or greater than the threshold, the HUI list is updated based on two situations (line 9): first, the generated chromosome is added to the HUI list if it does not exist in the HUI list; second, we introduce scaling factor strategy. Since the roulette wheel selection is used, some extraordinary chromosomes may be generated from the start of the algorithm because item/itemsets with greater utility have a higher possibility of being chosen. These chromosomes may dominate the population, leading to premature convergence.[31] In contrast, lower-quality chromosomes may generate high-quality solutions by providing new genetic material into the population, thereby improving variation.[26] Thus, we propose a scaling factor. It works by reducing the probability of selecting extraordinary chromosomes (HUIs) in the selection process by $\frac{1}{\text{pop\_size}}$. We use this equation because for repeated extraordinary chromosomes, they have already been generated once as HUIs, so reducing their selection probability by one chance gives lower-quality chromosomes a greater chance of being selected for mating. This strategy is reasonable because selecting extraordinary chromosomes too frequently for mating may lead to discover a limited variation of combinations.

$$Pi = \frac{\text{TWU}(i)}{\sum_{j=1}^{|I^*|} \text{TWU}(I_j^*)}. \tag{11}$$

4.3. *Maintaining population diversity*

The generated population from the previous iteration focuses on the best individuals, which may limit the ability to explore HUIs. The population diversity maintenance strategy is essential for keeping the population diverse and allowing the algorithm to explore more HUIs. In this work, we suggest selecting two of the latest HUIs to replace two random individuals from the population, instead of selecting the HUIs randomly[22] or consecutively.[23] This may help to broaden the search space because the latest HUIs may have greater fitness and may also have different characteristics

from the current population, leading to the exploration of more HUIs. Algorithm 3 depicts the process of maintaining population diversity. In lines 1–2, $c1$ and $c2$ are initialized as random chromosomes from the population. Line 3 indicates the first case, if the HUI list contains only one HUI, then the chromosomes corresponding to $c1$ and $c2$ are replaced by HUI at index 0 (lines 4–5). Line 6 indicates the second case, if the HUI list contains more than one HUI, then the chromosomes corresponding to $c1$ and $c2$ are replaced by the last two HUIs (lines 7–8).

4.4. *Crossover*

We employ the well-known uniform crossover to generate new offspring. It promotes unbiased exploration and the generation of potential offspring for recombination.[13] However, the reproduction process, in which genes from two parents join, may generate offspring that is not necessarily superior to its parents. This reduces the capability to find HUIs. Thus, the iterative uniform crossover is proposed (Algorithm 4). Lines 1–2 define the offspring based on two selected parents. Lines 3–24 repeat the crossover to obtain at least one of the offspring has greater fitness than its parents, because better offspring may generate better solutions. Lines 4–8 perform the uniform crossover. The chromosomes of the offspring are repaired in lines 9 and 10. The fitness of the offspring is calculated in lines 11 and 12. Lines 13–18 compare the offspring's fitness to the minimum utility threshold. If it is equal to or greater than the threshold, the HUI list is updated. Line 19 checks whether any of the offspring has greater fitness than their parents, if so, the offspring is added to the new population (lines 20–21) and the loop breaks (line 22).

4.5. *Mutation*

The mutation operator is required to avoid local optima caused by other genetic operators. It works by exploring new states using some efficient strategies. In this work, the single point mutation is used by flipping one random gene from the chromosome. The mutation algorithm is depicted in Algorithm 5. The mutation process is repeated for all offspring in lines 1–15. Lines 2–4 define the mutation. Line 5 is responsible for repairing the mutated chromosome. If the mutated chromosome exists in the HUI list, it is revised in lines 6–10. We propose a new revision strategy for repeated HUIs that prioritizes greater TWU items to find HUIs faster. The revision process starts in line 7 by generating a random number between 0 and $|I^*|$ to define the flipping point. Line 8 revises the chromosome based on two conditions: first, if it is bit 1, it is flipped to 0. Since one item is removed, we add one on the right side to prioritize the high TWU item by flipping the first bit 0 from the right side; second, if it is bit 0, it is flipped to 1. Since one item is added, we remove one low TWU item from the left side by flipping the first bit 1 from the left side. This strategy prioritizes the items on the right side because they have higher utility than items on the left side since they are sorted in ascending TWU order. The fitness value is then calculated in line 11. Lines 12–14 compare the fitness value to the minimum utility threshold. If it is equal to or greater than the threshold, the HUI list is updated.

4.6. *Proposed local search strategy*

Local search can be implemented in GA to enhance the search capability in order to find a better solution. We designed a local search function at the end of each iteration to refine the wasted chromosomes generated by GA search functions. The local search function is based on two mutation operators, namely swap and inverse, as suggested by Viana *et al.*[15] and Asadzadeh.[16] We also included single point mutation. The swap operation can be applied to the bits at two randomly selected positions b and c of the chromosome. The inverse operation can be applied to every bit between two randomly selected positions b and c. The flip operation can be applied to a bit at one randomly selected position b.

Algorithm 6 describes the local search function. Lines 1–23 repeat the local search for all individuals. The current chromosome is defined in line 2. If the fitness value is greater than the minimum utility threshold, it is skipped during the local search process (lines 3–4). Note that we only process low-fitness chromosomes. The best chromosome is defined as the current chromosome in line 6. Lines 7–21 describe the heart of the local search function. In line 8, one mutation operator is randomly chosen. The current chromosome is mutated using the selected operator (line 9) and is then revised in line 10. The fitness value is then calculated in line 11. Lines 12–14 compare the fitness value to the minimum utility threshold. If it is equal to or greater than the threshold, the HUI list is updated. The fitness values of the current chromosome and best chromosome are compared. If it is greater, the best chromosome is replaced with the current chromosome, and the local search iteration continues (lines 15–16). Otherwise, the population is updated with the best chromosome and the loop breaks to avoid unnecessary exploration (lines 17–19).

4.7. *Time complexity analysis*

The time complexity of one HUIM-MGALS iteration, considering population size M and chromosome length l, can be broken down into the following main components:

(1) Population diversity maintenance strategy has a complexity of $O(1)$.
(2) Roulette wheel selection has a complexity of $O(\log M)$, since it is based on binary search.
(3) Iterative uniform crossover has a complexity of $O(M \times l \times i)$, where i is the number of iterations for iterative crossover.
(4) Fitness scaling strategy has a complexity of $O(1)$.
(5) Individual repair strategy has a complexity of $O(M \times l)$.
(6) One-point mutation has a complexity of $O(M)$.
(7) HUI revision strategy has a complexity of $O(1)$.
(8) Elite strategy has a complexity of $O((2M)\log(2M))$, since it is based on merge sort.
(9) Local search has a complexity of $O(M \times j)$, where j is the number of iterations for each chromosome.

Table 6. Datasets details.

Dataset	Avg. Length	$\|I\|$	$\|D\|$
Connect	43	129	67,557
Accident_10%	34	468	34,019
Mushroom	23	119	8124
Chess	37	75	3196
Foodmart	4	1559	4141
Real retail	3	3114	24,511

This complexity is similar to that of HUIM-IGA,[23] which is comparable to the general complexity of GA. According to line 8 of Algorithm 1, the main iteration is limited by the maximum number of evaluations. Therefore, additional iterations i and j from iterative uniform crossover and local search may not significantly degrade the overall runtime. This is because these inner loops also perform fitness evaluations, which contribute to the total evaluation number and potentially reduce the number of main iterations. However, even within this limited number of iterations, the proposed strategies have improved the search's ability to generate HUIs.

5. Experimentation Setup

The proposed HUIM-MGALS and several existing EC-based algorithms, including HUIM-IGA,[23] Bio-HUIF-GA,[22] Bio-HUIF-PSO,[22] and HUIM-BPSO[25] were executed on a common set of datasets. These algorithms were written in Java and were executed on a laptop with a 64-bit Windows 10 OS, an AMD Ryzen 7-5800H CPU clocked 3.2 GHz, and 32 GB of RAM.

Datasets: The experimentation used five standard datasets including connect, accident, mushroom, chess, and foodmart.[36] A real transaction dataset from Malang Indonesia was also used, namely real retail.[37] The minimum utility thresholds varied for each dataset. Table 6 describes the dataset's characteristics.

Validation parameters: The performance of all algorithms was evaluated in terms of runtime, convergence speed, accuracy, and the number of generated HUIs. Table 7 presents the parameter settings. The values of the fitness evaluation number and the population size were set the same as those used in existing algorithms in order to make the comparisons fair on a uniform setup.

Table 7. Parameters.

Parameter	Value
Fitness evaluation number	60,000
Population size	20
Number of independent executions	5
Mutation	One-point
Crossover	Uniform (iterative)
Selector	Roulette wheel (modified)

6. Results and Discussions

6.1. *Comparison of runtime*

We calculated the average runtimes of several independent executions to evaluate the efficiency of HUIM-MGALS. Since heuristic algorithms commonly generate different results for every execution, the average value is calculated to represent the overall performance. Figure 2 illustrates the runtime comparisons for four dense datasets.

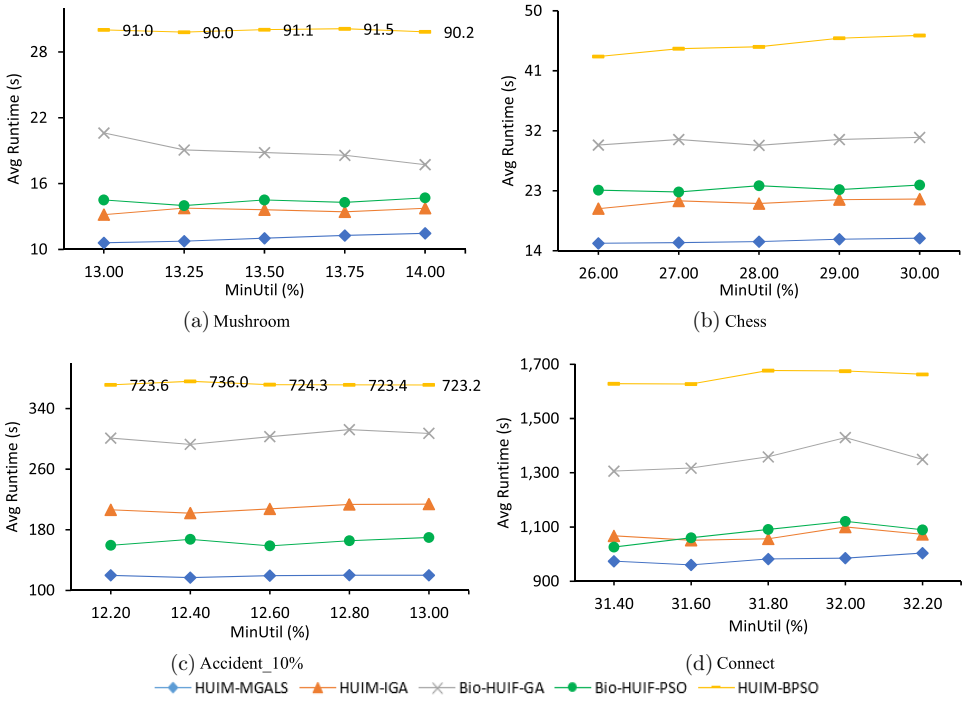


Fig. 2. Runtime for four dense datasets.

Observation 1: From Fig. 2, HUIM-MGALS has the best runtime. Overall observations claim that HUIM-MGALS is up to 8.6 times faster than the other algorithms for four dense datasets.

Under various minimum utility thresholds, HUIM-MGALS always achieves the best runtime outperforming the other EC-based algorithms. It outperforms HUIM-IGA, Bio-HUIF-GA, Bio-HUIF-PSO, and HUIM-BPSO for the mushroom dataset up to 1.3, 1.9, 1.4, and 8.6 times faster, respectively. It is faster up to 1.4, 2.0, 1.5, and 2.9 times, respectively, for the chess dataset. It is up to 1.8, 2.6, 1.4, and 6.3 times faster, respectively, for the accident_10% dataset. It is faster up to 1.1, 1.5, 1.1, and 1.7 times, respectively, for the connect dataset. HUIM-BPSO has the slowest runtime compared to the other algorithms because it only utilizes the standard search

function of PSO, which has a limited ability to find HUIs. Although Bio-HUIF-GA employs some novel strategies to improve the search ability thus it outperforms HUIM-BPSO, it is still not faster than the other three algorithms. Bio-HUIF-PSO and HUIM-IGA perform similarly for the mushroom and connect datasets. Bio-HUIF-PSO performs better for the accident_10% dataset and HUIM-IGA performs better for the chess dataset. However, these two algorithms are still slower than HUIM-MGALS because some new efficient strategies have been applied to HUIM-MGALS to improve the search ability and reduce some unnecessary computation.

Furthermore, compared to HUIM-AF[34] (as presented in their paper), HUIM-MGALS runs faster for all datasets. Based on the reported results, in the Chess dataset with 28% and 29% thresholds, HUIM-MGALS has less than 16s runtime, whereas HUIM-AF takes more than 200 s. Similar trends are observed in the Mushroom dataset (14% threshold), Accident_10% dataset (12.6–13% thresholds), and Connect dataset (31.8%, 32%, and 32.2% thresholds). In these cases, HUIM-MGALS has runtimes less than 12, 121, and 1005 s, respectively, while HUIM-AF takes over 200, 1500, and 11,000 s, respectively.

6.2. *Comparison of convergence speed*

The convergence speed of all algorithms is compared, as demonstrated in Fig. 3. For the given fitness evaluation number, we investigate how many HUIs are

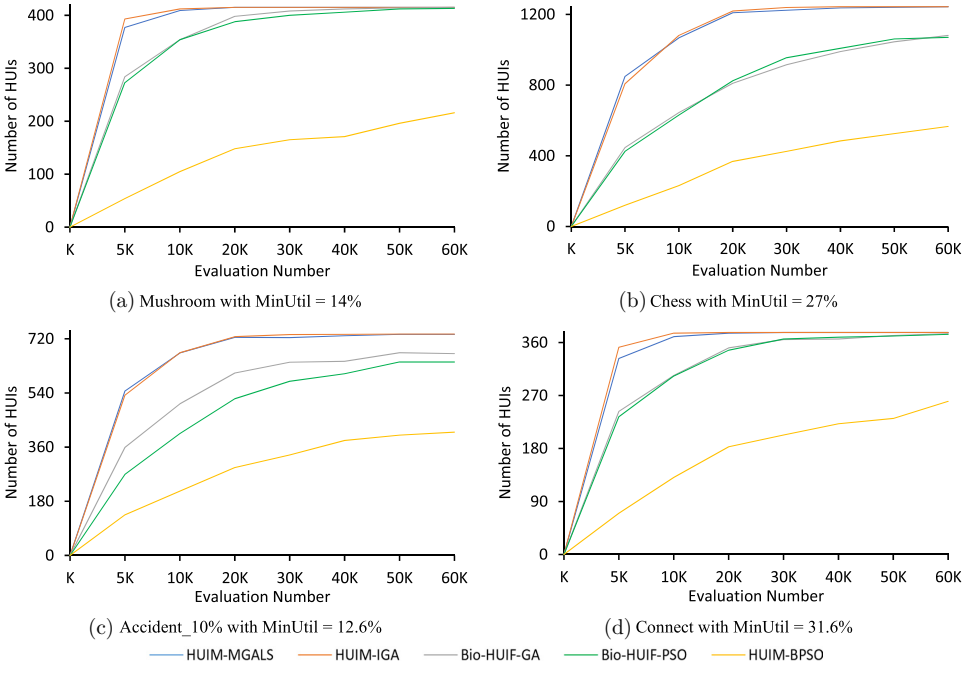


Fig. 3. Convergence speed for four dense datasets.

generated. The fitness evaluation number varied between 5 K and 60 K, with 5 K representing the first stage and 60 K representing the later stage of the algorithm's execution. We expect the algorithm to generate complete HUIs within a limited number of evaluations. The maximum number of HUIs (representing 100% accuracy) for the mushroom, chess, accident_10%, and Connect datasets in Fig. 3 are 415, 1246, 736, and 377, respectively. By varying the fitness evaluation number and recording the generated HUIs stage by stage, we can illustrate how fast (as early as possible) the algorithm converges (reaches 100%/Maximum accuracy).

Observations 2: Figure 3 demonstrates that HUIM-MGALS and HUIM-IGA are comparable. HUIM-MGALS also outperforms the other three algorithms for four dense datasets.

For all datasets, HUIM-MGALS and HUIM-IGA are equally better. Below 10 K fitness evaluation number, the accuracy decreases due to the limited number of generations. For the chess and accident_10% datasets, HUIM-MGALS generates more HUIs at the earliest stage (5 K fitness evaluation number), which are 849 and 547, respectively. However, for the mushroom and connect datasets, HUIM-IGA generates more HUIs at the earliest stage (5 K evaluation number), which are 393 and 352, respectively. Both HUIM-MGALS and HUIM-IGA converge faster than the other three algorithms and the results are also consistent for all datasets. Bio-HUIF-PSO and Bio-HUIF-GA are comparable, with the exception of the accident_10% dataset, where Bio-HUIF-GA has better performance. However, they are not better than HUIM-MGALS. Bio-HUIF-PSO and Bio-HUIF-GA cannot generate complete HUIs for all datasets, thus they are ineffective in finding HUIs. It is because they have a few strategies to improve effectiveness. HUIM-BPSO gives the worst performance due to the standard search function, which makes it difficult to explore HUIs.

6.3. *Comparison of accuracy*

Generating the maximum number of HUIs determines the strength and accuracy of heuristic HUIM algorithms. In this paper, we use PLB[6] as an exact algorithm as a benchmark to evaluate the accuracy of HUIM-MGALS and also to compare the performances of the other algorithms on similar lines. The comparison of the best and average performances of all algorithms is presented in Table 8. The best results reflect the ability/strength of the algorithm to find HUIs. Average performances are indicators of the sustainability of this strength with varying parameters such as minimum utility threshold and dataset.

Table 8 shows that for the mushroom dataset, HUIM-MGALS outperforms the other algorithms. HUIM-IGA can generate complete HUIs with MinUtil = 13.75% or higher, whereas Bio-HUIF-GA can generate complete HUIs with MinUtil = 14%. Bio-HUIF-PSO and HUIM-BPSO are unable to generate the complete HUIs under the given thresholds. For the chest dataset, HUIM-MGALS (99.69%) and HUIM-IGA (99.86%) are equally performing better. Bio-HUIF-GA (89.23%) and HUIF-PSO (90.09%) are performing similarly, but they are not better than HUIM-MGALS.

Table 8. Accuracy (%) for four dense datasets.

		Algorithm									
		HUIM-MGALS		HUIM-IGA		Bio-HUIF-GA		Bio-HUIF-PSO		HUIM-BPSO	
Dataset	MinUtil (%)	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best	Avg.	Best
Mushroom	13.00	99.86	100	99.58	99.83	85.02	87.59	89.90	91.23	32.22	33.94
	13.25	99.87	100	99.62	99.79	89.39	90.29	93.24	93.63	34.22	35.70
	13.50	99.95	100	99.74	99.74	94.23	95.32	95.32	96.75	35.90	37.27
	13.75	100	100	100	100	98.12	98.43	97.98	98.61	41.53	43.90
	14.00	100	100	100	100	99.76	100	99.08	99.52	46.17	47.95
Avg.		99.94	100	99.79	99.87	93.30	94.33	95.10	95.95	38.01	39.75
Chess	26.00	98.56	98.61	99.01	99.44	57.45	59.51	65.52	66.99	28.45	29.32
	27.00	99.73	99.84	99.78	99.84	84.25	87.24	84.43	85.71	43.61	45.18
	28.00	100	100	100	100	98.74	99.39	97.28	97.77	59.84	62.68
	29.00	100	100	100	100	100	100	100	100	71.93	73.86
	30.00	100	100	100	100	100	100	100	100	85.59	88.24
Avg.		99.66	99.69	99.76	99.86	88.09	89.23	89.45	90.09	57.88	59.86
Accident_10%	12.20	98.83	99.29	99.86	99.91	86.55	88.11	78.14	79.15	48.11	48.98
	12.40	99.09	99.78	99.78	100	89.42	90.53	82.68	85.09	50.66	52.77
	12.60	99.54	99.86	99.89	100	91.90	93.07	87.74	89.54	54.86	57.07
	12.80	99.59	100	99.93	100	94.65	97.46	91.20	92.89	58.07	60.91
	13.00	99.71	100	100	100	95.45	97.31	93.26	94.63	62.23	63.84
Avg.		99.35	99.79	99.89	99.98	91.60	93.30	86.60	88.26	54.79	56.72
Connect	31.40	99.90	100	100	100	95.49	96.01	97.02	97.75	56.85	58.58
	31.60	99.79	100	100	100	98.94	99.47	98.89	100	62.28	64.19
	31.80	100	100	100	100	99.69	100	99.92	100	62.53	65.90
	32.00	100	100	100	100	100	100	100	100	73.22	76.02
	32.20	100	100	100	100	100	100	100	100	81.22	82.65
Avg.		99.94	100	100	100	98.83	99.10	99.17	99.55	67.22	69.47

For the accident_10% dataset, HUIM-IGA outperforms the other algorithms at MinUtil = 12.4013%. It generates the complete set of HUIs. Whereas, HUIM-MGALS achieves the same performance for threshold ranges between 12.8% and 13%. The other three algorithms, on the other hand, do not provide the desired performance, as they are not able to retrieve the HUIs completely. For the connect dataset, for all thresholds, HUIM-MGALS and HUIM-IGA are equally performing best among all. Bio-HUIF-PSO (99.55%) outperforms Bio-HUIF-GA (99.1%), whereas HUIM-BPSO gives the worst performance among all. In general, HUIM-BPSO gives the worst performance as it generates only 50–70% of HUIs.

For the average performance, HUIM-MGALS has the best performance producing 99.94% of HUIs. HUIM-IGA (99.79%) performance is in second place, followed by Bio-HUIF-PSO (95.1%), Bio-HUIF-GA (93.3%), and HUIM-BPSO (38.01%). These three algorithms are not able to retrieve the HUIs completely. For the chess dataset, HUIM-MGALS (99.66%), and HUIM-IGA (99.76%) are equally performing better. Bio-HUIF-PSO and Bio-HUIF-GA are performing similarly, but they are only able to retrieve less than 90% of HUIs. HUIM-BPSO performance is in the last place with only generating 57.88% of HUIs. For the accident_10% dataset, HUIM-MGALS (99.35%), and HUIM-IGA (99.89%) are equally performing better followed by Bio-HUIF-GA (91.6%), Bio-HUIF-PSO (86.6%), and HUIM-BPSO (54.79%). For the connect dataset, HUIM-IGA outperforms the other algorithms, which can generate the complete HUIs for all thresholds. HUIM-IGA performs equally with HUIM-MGALS, which can generate only 0.06% fewer HUIs on average. Bio-HUIF-PSO (99.17%) outperforms Bio-HUIF-GA (98.83%), whereas HUIM-BPSO (67.22%) gives the worst performance among all. In conclusion, HUIM-MGALS is comparable to HUIM-IGA. It generates only 0.48% fewer HUIs on average. Furthermore, in most cases, it outperforms the other three algorithms.

Observations 3: Table 8 shows that HUIM-MGALS has the best accuracy for the mushroom and connect datasets for the best cases. For average cases, HUIM-MGALS has the best accuracy for the mushroom dataset.

Furthermore, compared to HUIM-AF[34] (as presented in their paper), HUIM-MGALS achieves higher accuracy by generating more HUIs for average results for all datasets. In the Chess dataset with 28% and 29% thresholds, HUIM-MGALS achieves 100% accuracy (generating 493 and 176 HUIs, respectively), whereas HUIM-AF generates less than 300 and 150 HUIs, respectively. Similarly, in the Mushroom dataset with 14% threshold, HUIM-MGALS achieves 100% accuracy (generating 415 HUIs), whereas HUIM-AF generates less than 400 HUIs. In the Accident_10% dataset with 12.6–13% thresholds, HUIM-MGALS achieves 99.54%, 99.59%, and 99.71% accuracy (generating 732, 588, and 482 HUIs, respectively), whereas HUIM-AF generates less than 140, 120, and 100 HUIs, respectively. Lastly, in the Connect dataset with 31.8%, 32%, and 32.2% thresholds, HUIM-MGALS achieves 100% accuracy (generating 261, 171, and 98 HUIs, respectively), whereas HUIM-AF generates less than 250, 150, and 90 HUIs, respectively.

6.4. *Comparison of number of generated HUIs*

HUIM-MGALS and HUIM-IGA are competitive in terms of convergence speed and accuracy, both demonstrating near-optimal performance with 98–100% accuracy on four dense datasets. Consequently, to prove the superiority of our proposed techniques in HUIM-MGALS for HUI discovery over those in HUIM-IGA, we conducted further tests on sparse datasets. We selected the Foodmart and Real Retail datasets with low thresholds, where the high sparsity of missing entries challenges the effectiveness of heuristic algorithms in identifying HUIs. The results are presented in Table 9. With an average of 4148 HUIs discovered, HUIM-MGALS has the best performance on the Foodmart dataset. While it identifies 547 more HUIs on average than its closest competitor, HUIM-IGA. Bio-HUIF-GA, Bio-HUIF-PSO, and HUIM-BPSO fall further behind, generating 1401, 1513, and 1475 fewer HUIs on average, respectively. For the Real Retail dataset, HUIM-MGALS demonstrates a remarkable performance compared to the others. At the 0.9% threshold, where the maximum number of HUIs is 17, HUIM-MGALS successfully identifies all of them, while the others are not. This distinction becomes even more pronounced at 0.8% threshold and below, where the other algorithms get stuck in local optima, hindering their ability to accurately generate HUIs. In contrast, HUIM-MGALS consistently generates a significantly higher number of HUIs, clearly demonstrating its superior HUI discovery capabilities.

These two results clearly demonstrate the superior effectiveness of our proposed techniques in HUIM for HUI discovery, particularly in challenging scenarios like mining sparse transactions with low thresholds. We can infer that introducing a population diversity maintenance strategy, a fitness scaling factor strategy, and a HUI revision strategy play a crucial role in achieving these superior results. These strategies can preserve potential individuals for mating and prevent local optimum

Table 9. Average number of HUIs for the foodmart and real retail datasets.

		Algorithm				
Dataset	MinUtil (%)	HUIM-MGALS	HUIM-IGA	Bio-HUIF-GA	Bio-HUIF-PSO	HUIM-BPSO
Foodmart	0.002	4302	3700	2796	2736	2823
	0.004	4249	3674	2807	2734	2824
	0.006	4190	3668	2761	2712	2751
	0.008	4128	3570	2748	2582	2588
	0.01	3869	3392	2619	2410	2380
Avg.		4148	3601	2746	2635	2673
Real Retail	0.5	21,117	28	28	28	10
	0.6	21,785	22	22	22	10
	0.7	21,537	21	21	21	10
	0.8	4686	16	16	16	10
	0.9	17	15	15	15	9
Avg.		13,828	20	20	20	10

from happening. Additionally, HUIM-MGALS employs local search and iterative crossover to explore more possible solutions in a limited fitness evaluation number.

Observations 4: Table 9 shows that HUIM-MGALS has the best performance by generating the highest number of HUIs among the others.

6.5. *Impact of the proposed strategies*

To evaluate the impact of the proposed strategies in HUIM-MGALS on accuracy and convergence speed, we conducted an experiment on the Real Retail dataset with 0.5% threshold. We selected the Real Retail dataset because it is a challenging dataset with the greatest number of HUIs, as discussed in Sec. 6.4. Table 10 presents the average number of generated HUIs. The first row shows the result achieved by the complete HUIM-MGALS, while the remaining rows present the results when running HUIM-MGALS without each individual strategy. Generally, a higher number of HUIs indicates both higher accuracy faster convergence speed.

HUIM-MGALS generates 21,117 HUIs with complete strategies. However, without iterative crossover, it generates 17,548 HUIs, losing 3569 HUIs due to limited exploration in the crossover step. Similarly, without the modified population diversity maintenance strategy, HUIM-MGALS generates only 746 HUIs. In most executions, this lack of diversity leads HUIM-MGALS to get stuck in local optima. The HUI revision strategy also plays a crucial role. Without it, HUIM-MGALS generates only 3484 HUIs, demonstrating the importance of this strategy in exploring repeated HUIs for potentially better solutions. Optimized local search further improves the results. Without it, HUIM-MGALS generates 8774 HUIs, demonstrating the effectiveness of local search in refining and exploring low-utility chromosomes. Finally, HUIM-MGALS without fitness scaling strategy generates 15,490 HUIs, losing 5627 HUIs due to unbalanced selection of high and low utility chromosomes.

Table 10. Average number of HUIs for the real retail dataset with MinUtil = 0.5%.

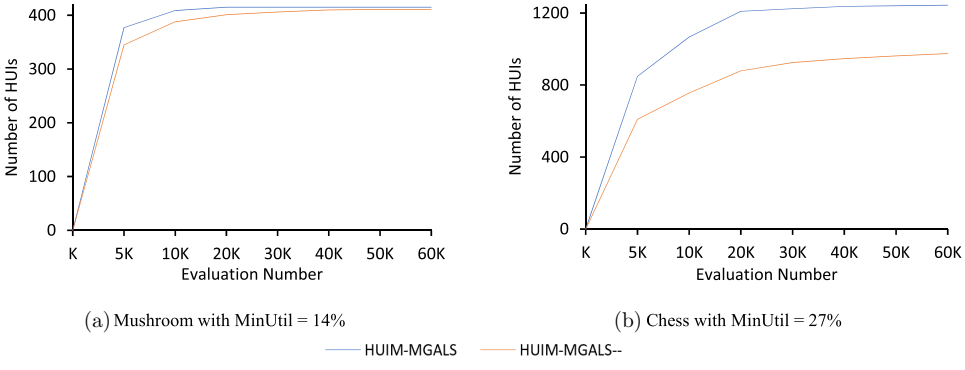
Strategic variations	Algorithm	Average number of HUIs
With Modifications	HUIM-MGALS (complete-all variations)	21,117
Individual impact of each step if proposed modification not applied	1. HUIM-MGALS without iterative crossover	17,548
	2. HUIM-MGALS without modified population diversity maintenance	746
	3. HUIM-MGALS without HUI revision	3484
	4. HUIM-MGALS without optimized local search	8774
	5. HUIM-MGALS without fitness scaling	15,490

Observations 5: Table 10 shows that the modified population diversity maintenance strategy is the most impactful modification. This is because without this strategy, HUIM-MGALS generates only 746 HUIs on average.

The most impactful modification is the modified population diversity maintenance step. As we can see clearly the generated HUIs without this modification are only 746. However, if we want to improve the accuracy, it will take a long time to reach its maximum, which may not be even equivalent to the best result (21,117). This clearly states that modified population diversity maintenance has a potential impact on accuracy as well as convergence speed.

6.6. *Effect of population diversity on convergence speed*

To observe the effect of population diversity on convergence speed, we conducted experiments with HUIM-MGAL and HUIM-MGALS–(without population diversity maintenance strategy) on the Mushroom and Chess datasets. Figure 4 presents the convergence speed for both algorithms.



(a) Mushroom with MinUtil = 14%

(b) Chess with MinUtil = 27%

Fig. 4. Convergence speed for two dense datasets.

Observations 6: Figure 4 demonstrates that HUIM-MGALS achieves faster convergence speed. This highlights the importance of population diversity in generating HUIs more effectively.

For the mushroom dataset, HUIM-MGALS consistently outperforms HUIM-MGALS–throughout the evolutionary stages, particularly in the middle stages where population diversity naturally decreases. HUIM-MGAL converges by the 20 K fitness evaluation, while HUIM-MGALS–fails to converge until the 60 K fitness evaluation. Similarly, HUIM-MGAL exhibits significantly faster convergence on the Chess dataset compared to HUIM-MGALS–. Without diversity maintenance, HUIM-MGALS–misses many HUIs due to its limited exploration of the search space. These results demonstrate that the proposed strategy promotes the generation of more HUIs, especially in the middle and later stages of evolution, by maintaining higher population diversity.

7. Conclusion

This research experimentation has brought novelty by proposing the use of Modified GA for two significant reasons. First, as per the literature survey, GA being the popular algorithm with better performance in many other contexts, but its use has not been encouraged much in HUIM. Second, even though it has been used by a few existing HUIM experiments, many aspects have been oversight, which have been discussed as issues in Secs. 1 and 2. In order to focus on those aspects with the intention to provide an optimal solution in a reasonable time frame, we proposed and implemented a modified GA with six major modifications.

The extensive experiments demonstrate that HUIM-MGALS performs best among the presented algorithms in terms of runtime, up to 8.6 times faster in four dense datasets. It also stands out with its superior HUI discovery capabilities, generating the highest number of HUIs compared to other algorithms while also effectively avoiding local optima, particularly in challenging scenarios like mining sparse datasets with low thresholds. We also discovered that the modified population diversity maintenance step has the most impactful modification in generating HUIs compared to other modifications. This is evident because HUIM-MGALS generates the fewest HUIs without this modification. The convergence speed of HUIM-MGALS is comparable to HUIM-IGA,[23] but it generates only four fewer HUIs on average. HUIM-MGALS and HUIM-IGA are equally better in terms of accuracy, with only a 0.48% difference in accuracy, on average. HUIM-MGALS also performs better than the other three algorithms, namely Bio-HUIF-GA,[22] Bio-HUIF-PSO,[22] and HUIM-BPSO,[25] in terms of convergence speed and accuracy. Further comparison with HUIM-AF[34] shows that HUIM-MGALS has faster runtime and higher accuracy for all four dense datasets.

Using the proposed strategies and intended contributions, the HUIM-MGALS experimentations with respect to varying thresholds and datasets achieve the desired and consistent results for all evaluation parameters. Overall study has proven the effectiveness of six potential modifications at various phases of GA compared with existing contributions. So, this can be recommended as one of the effective heuristic strategies for HUIM, titled HUIM-MGALS. Future research may consider other EC-based algorithms such as ACO, PSO, and Bat to utilize the proposed strategies. Some efficient strategies can also be designed to increase the ability to find HUIs. Applying parallel frameworks i.e., MPI and Apache Spark might be proven powerful to boost the runtime for large datasets.

ORCID

Eduardus Hardika Sandy Atmaja https://orcid.org/0000-0003-1739-0116
Kavita Sonawane https://orcid.org/0000-0003-0865-6760

References

1. J. Han, M. Kamber and J. Pei, *Data mining: Concepts and Techniques*, 3rd edn. (Morgan Kaufmann, Waltham, 2012).
2. R. Agrawal and R. Srikant, Fast algorithms for mining association rules, in *Proc. 20th Int. Conf. Very Large Data Bases* (Morgan Kaufmann, San Francisco, 1994), pp. 487–499.
3. P. Fournier-Viger, J. C. W. Lin, T. Truong-Chi and R. Nkambou, Studies in big data, in *High-Utility Pattern Mining: A Survey of High Utility Itemset Mining*, eds. P. Fournier-Viger, J. C. W. Lin, R. Nkambou, B. Vo and V. S. Tseng (Springer, Cham, 2019), pp. 1–45.
4. Y. Liu, W. Liao and A. Choudhary, Lecture notes in computer science, in *Advances in Knowledge Discovery and Data Mining: A Two-phase Algorithm for Fast Discovery of High Utility Itemsets*, eds. T. B. Ho, D. Cheung and H. Liu (Springer, Berlin, 2005), pp. 689–695.
5. V. S. Tseng, B. E. Shie, C. W. Wu and P. S. Yu, Efficient algorithms for mining high utility itemsets from transactional databases, *IEEE Trans. Knowl. Data Eng.* **25**(8) (2013) 1772–1786.
6. E. H. S. Atmaja and K. Sonawane, Lecture notes in networks and systems, in *ICT Analysis and Applications: Parallel Algorithm to Efficiently Mine High Utility Itemset*, eds. S. Fong, N. Dey and A. Joshi (Springer, Singapore, 2022), pp. 167–178.
7. M. Liu and J. Qu, Mining high utility itemsets without candidate generation, in *Proc. 21st ACM Int. Conf. Information and Knowledge Management* (Association for Computing Machinery, New York, 2012), pp. 55–64.
8. Q. H. Duong, P. Fourfnier-Viger, H. Ramampiaro, K. Norvag and T. L. Dam, Efficient high utility itemset mining using buffered utility-lists, *Appl. Intell.* **48** (2018) 1859–1877.
9. E. H. S. Atmaja and K. Sonawane, Optimization of search space division with enhanced shared memory-based utility list buffer miner with a parallel framework (SM-PLB) for effective high utility itemset mining, *Int. J. Inf. Technol.* **15**(3) (2023) 1597–1609.
10. E. H. S. Atmaja and K. Sonawane, Lecture notes in networks and systems, in *Pattern Recognition and Data Analysis with Applications: A Review of High Utility Itemset Mining for Transactional Database*, eds. D. Gupta, R. S. Goswami, S. Banerjee, M. Tanveer and R. B. Pachori (Springer, Singapore, 2022), pp. 15–27.
11. A. Telikani, A. H. Gandomi and A. Shahbahrami, A survey of evolutionary computation for association rule mining, *Inf. Sci.* **524** (2020) 318–352.
12. M. S. Nawaz, P. Fournier-Viger, U. Yun, Y. Wu and W. Song, Mining high utility itemsets with hill climbing and simulated annealing, *ACM Trans. Manag. Inf. Syst.* **13**(1) (2022) 1–22.
13. S. Katoch, S. S. Chauhan and V. Kumar, A review on genetic algorithm: Past, present, and future, *Multimedia Tools Appl.* **80** (2021) 8091–8126.
14. M. S. Umam, M. Mustafid and S. Suryono, A hybrid genetic algorithm and tabu search for minimizing makespan in flow shop scheduling problem, *J. King Saud Univ. — Comput. Inf. Sci.* **34**(9) (2022) 7459–7467.
15. M. S. Viana, O. M. Junior and R. C. Contreras, A modified genetic algorithm with local search strategies and multi-crossover operator for job shop scheduling problem, *Sensors* **20**(18) (2020) 5440.
16. L. Asadzadeh, A local search genetic algorithm for the job shop scheduling problem with intelligent agents, *Comput. Ind. Eng.* **85** (2015) 376–383.
17. F. Souza and D. Grimes, Combining local search and genetic algorithm for two-dimensional guillotine bin packing problems with partial sequence constraint, in *Proc. 28th Irish Conf. Artificial Intelligence and Cognitive Science* (Technological University Dublin, Dublin, 2020), pp. 97–108.

18. L. Gharsalli and Y. Guerin, A hybrid genetic algorithm with local search approach for composite structures optimization, in *Proc. 8th European Conf. Aeronautics and Space Sciences* (EUCASS, Madrid, 2019).
19. J. M. T. Wu, J. Zhan and J. C. W. Lin, An ACO-based approach to mine high-utility itemsets, *Knowl.-based Syst.* **2017** (2017) 102–113.
20. W. Song and J. Nan, Lecture notes on data engineering and communications technologies, in *Advances in Natural Computation, Fuzzy Systems and Knowledge Discovery: Mining High Utility Itemsets Using Ant Colony Optimization*, eds. H. Meng, T. Lei, M. Li, K. Li, N. Xiong and L. Wang (Springer, Cham, 2021), pp. 98–107.
21. S. Kannimuthu and K. Premalatha, Discovery of high utility itemsets using genetic algorithm with ranked mutation, *Appl. Artif. Intell.* **28**(4) (2014) 337–359.
22. W. Song and C. Huang, Mining high utility itemsets using bio-inspired algorithms: A diverse optimal value framework, *IEEE Access* **6** (2018) 19568–19582.
23. Q. Zhang, W. Fang, J. Sun and Q. Wang, Improved genetic algorithm for high-utility itemset mining, *IEEE Access* **7** (2019) 176799–176813.
24. J. C. W. Lin, L. Yang, P. Fournier-Viger, J. M. T. Wu, T. P. Hong, T. S. L. Wang and J. Zhan, Mining high-utility itemsets based on particle swarm optimization, *Eng. Appl. Artif. Intell.* **55** (2016) 320–330.
25. 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 Comput.* **21** (2017) 5103–5121.
26. R. Gunawan, E. Winarko and R. Pulungan, BPSO-based method for high-utility itemset mining without minimum utility threshold, *Knowl.-Based Syst.* **190** (2020) 105164.
27. H. Yao, H. J. Hamilton and C. J. Butz, A foundational approach to mining itemset utilities from databases, in *Proc. 4th SIAM Int. Conf. Data Mining* (DBLP, Florida, 2004), pp. 482–486.
28. B. Vo, L. V. Nguyen, V. V. Vu, M. T. H. Lam, T. T. M. Duong, L. T. Manh, T. T. T. Nguyen, L. T. T. Nguyen and T. P. Hong, Mining correlated high utility itemsets in one phase, *IEEE Access* **8** (2020) 90465–90477.
29. T. Wei, B. Wang, Y. Zhang, K. Hu, Y. Yao and H. Liu, FCHUIM: Efficient frequent and closed high utility itemsets mining, *IEEE Access* **8** (2020) 109928–109939.
30. C. H. Lin, C. W. Wu, J. T. Huang and V. S. Tseng, Lecture notes in computer science, in *Advances in Knowledge Discovery and Data Mining: Parallel Mining of Top-k high Utility Itemsets in Spark in-memory Computing Architecture*, eds. Q. Yang, Z. H. Zhou, Z. Gong, M. L. Zhang and S. J. Huang (Springer, Cham, 2019), pp. 253–265.
31. D. E. Goldberg, *Genetic Aalgorithms in Search, Optimization and Machine Learning* (Addison-Wesley, USA, 1989).
32. A. J. P. Delima, A. M. Sison and R. P. Medina, A modified genetic algorithm with a new crossover mating scheme, *Indonesian J. Electr. Eng. Inf.* **7**(2) (2019) 165–181.
33. S. Rimcharoen and N. Leelathakul, Ring-based crossovers in genetic algorithms: Characteristic decomposition and their generalization, *IEEE Access* **9** (2021) 137902–137922.
34. W. Song, J. Li and C. Huang, Lecture notes in computer science, in *Advances in Swarm Intelligence: Artificial Fish Swarm Algorithm for Mining High Utility Itemsets*, eds. Y. Tan and Y. Shi (Springer, Cham, 2021), pp. 407–419.
35. G. Liu, H. Lu, W. Lou, Y. Xu and J. X. Yu, Efficient mining of frequent patterns using ascending frequency ordered prefix-tree, *Data Mining Knowl. Discov.* **9** (2004) 249–274.
36. P. Fournier-Viger, C. W. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng and H. T. Lam, Lecture notes in computer science, in *Machine Learning and Knowledge Discovery in Databases: The SPMF Open-Source Data Mining Library Version 2*, eds. B. Berendt, B. Bringmann, E. Fromont, G. Garriga, P. Miettinen, N. Tatti and V. Tresp (Springer, Cham, 2016), pp. 36–40.
37. Real retail dataset, https://tinyurl.com/realretail (Accessed March 2023).